EVALUATION OF 2D LOCAL IMAGE DESCRIPTORS AND FEATURE ENCODING METHODS FOR DEPTH IMAGE BASED OBJECT CLASS RECOGNITION

by

Güney Kayım

B.S., Mechatronics Engineering, Bahçeşehir University, 2009

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Electrical and Electronics Engineering
Boğaziçi University

2014

ACKNOWLEDGEMENTS

First of all, I would like to thank my unofficial supervisor, Ceyhun Burak Akgül. Without his infinite support, encouragement, guidance and patience I would surely be less of a researcher. He has been educating me since the day we met in 2010. We work on this thesis even when we were on a business trip in Paris. During my thesis study, he was not just a supervisor to me he was also an older brother to me. There are no words that can describe how grateful I am to him.

I would also like to express how much I am thankful to my thesis supervisor, Bülent Sankur. The comments that he made wisely on my thesis work at our progress meetings, has profound effect on this thesis to finalize.

As a Software R&D Specialist in Provus A MasterCard Company, I would like to show my gratitude to my manager Koray Geçici for his endless support and concern. I would also like to thank him for letting me work on my thesis for a long time and not show up at the company.

I should also mention my younger brother Halil İbrahim Kayım. During my thesis study, he brought me coffee all the time to keep me awake, he prepared the meals, took care of other house work and let me use his super computer. Life would be so hard if he were not there for me.

Finally, I would like to thank my family and friends who always supported me and were there for me.

ABSTRACT

EVALUATION OF 2D LOCAL IMAGE DESCRIPTORS AND FEATURE ENCODING METHODS FOR DEPTH IMAGE BASED OBJECT CLASS RECOGNITION

In this thesis, we have investigated the 3D object class recognition problem. We used an approach that solves this problem with the use of depth images obtained from 3D object models. In the approach we used, 3D object class recognition system is composed of two stages; training and testing. In both stages, first, keypoints are detected from the images, and then 2D local image descriptors are built around these keypoints. This is continued by encoding local descriptors into a single descriptor. Just before this step, in training stage, a codebook is learned, and it is used for encoding local descriptors in both stages. Another extra step in training stage is, after the descriptors are encoded, for each class a binary classifier is trained. Then, these classifiers are used in testing stage. We have evaluated different keypoint detection methods, 2D local image descriptors and encoding methods. Then, we experimentally show their superiorities and weaknesses over each other. Our experiments clearly show the best performing keypoint detection method, local image description method and feature encoding method in the depth image domain, which are densely sampled SIFT descriptors and Fisher Vector encoding. Using different experimental setups yields similar results, thus the validity of the methods that are selected as best is proven.

ÖZET

DERİNLİK İMGESİ TABANLI NESNE SINIFI TANIMA İÇİN İKİ BOYUTLU YEREL İMGE TANIMLAYICILARININ VE ÖZNİTELİK KODLAMA YÖNTEMLERİNİN DEĞERLENDİRİLMESİ

Bu çalışmada 3 boyutlu nesne sınıfı tanıma problemi incelenmiştir. Problemin çözümü için, 3 boyutlu nesne modellerinden derinlik imgeleri elde edilmiştir ve bu imgelere dayalı bir yaklaşım kullanılmıştır. Kullanılan yaklaşımda 3 boyutlu nesne sınıfı tanıma sistemi eğitim ve test aşamalarından oluşmaktadır. Her iki aşamada da ilk olarak imgelerden nirengi noktaları çıkarılır, daha sonra bu noktaların etrafında iki boyutlu yerel imge betimleyicileri oluşturulur. Ardından yerel imge betimleyicileri kodlanarak tek bir betimleyiciye dönüştürülür. Kodlama adımından önce, eğitim aşamsında bir kod defteri oluşturulur ve her iki aşamada da kodlama bu kod defteriyle yapılır. Ayrıca eğitim aşamasında her sınıf için birer adet ikili sınıflandırıcı eğitilir. Bu eğitilen sınıflandırıcılar test aşamasında kullanılır. Çalışma kapsamında farklı nirengi noktası belirleme yöntemleri, yerel imge betimleyicileri ve kodlama yöntemleri değerlendirilmiştir. Kullanılan yöntemlerin birbirlerine karşı olan üstünlükleri ve zayıflıkları deneysel olarak gösterilmiştir. Yapılan deneyler sonrası, derinlik imgesi alanında en iyi sonuç veren nirengi noktası belirleme, yerel imge betimleyicisi ve kodlama yöntemini açıkça görülebilmektedir. Bunlar yoğun örnekleme, SIFT betimleyicileri ve Fisher Vektör kodlaması olarak berlirlenmiştir. Farklı deneysel düzeneklerden alınan benzer sonuçlar, en iyi sonuç veren olarak seçilen metotların doğruluğunu ispatlamaktadır.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

$\mathbf{c}_i$	Word in a dictionary
$D(x, y, \sigma)$	Difference of Gaussian smoothed images
$G(x, y, \sigma)$	Variable-scale Gaussian function
$\mathbf{H}$	Hessian matrix
$I(x, y)$	Image
$I_\Sigma(x, y)$	Integral image
$L(x, y, \sigma)$	Gaussian smoothed image
$p(x \mid \lambda)$	Multivariate Gaussian distribution
$\mathbf{q}_i$	Assignments of features to dictionary words
$\mathbf{x}_i$	Single feature
$\mathbf{X}$	Feature set
λ	GMM parameter set
μ_i	Mean value of a Gaussian distribution
π_i	Prior probability of a Gaussian distribution
σ	Standard deviation of a Gaussian filter kernel
Σ_i	Covariance matrix of a Gaussian distribution

LIST OF ACRONYMS/ABBREVIATIONS

2D	Two Dimensional
3D	Three Dimensional
AGAST	Adaptive and Generic Accelerated Segment Test
AVC	Adaptive Views Clustering
BoW	Bag of Words
BRIEF	Binary Robust Independent Elementary Features
BRISK	Binary Robust Invariant Scalable Keypoints
CMVD	Compact Multi-View Descriptor
DCT	Discrete Cosine Transform
DoG	Difference of Gaussians
ED	Elevation Descriptor
EM	Expectation Maximization
ESB	Purdue Engineering Shape Benchmark
FAST	Features from Accelerated Segment Test
FREAK	Fast Retina Keypoint
FV	Fisher Vector
GLOH	Gradient Location and Orientation Histogram
GMM	Gaussian Mixture Models
KD-Tree	K-Dimensional Tree
KNN	K-Nearest Neighbor
LFD	Light Field Descriptor
MCVT	Matlab Computer Vision Toolbox
MSB	McGill 3D Shape Benchmark
NN	Neural Networks
ORB	Oriented FAST and Rotated BRIEF
PCA	Principal Component Analysis
PSB	Princeton Shape Benchmark
RBF	Radial Basis Function

SIFT	Scale Invariant Feature Transform
SSR	Signed Square Rooting
SURF	Speeded Up Robust Features
SVM	Support Vector Machine
ToF	Time of Flight
VLAD	Vector of Locally Aggregated Descriptors

1. INTRODUCTION

Object recognition is a high level computer vision problem, and can be investigated in terms of two physical domains: (i) The 2D domain, where data are 2D images and (ii) the 3D domain, where data are 3D point clouds or meshes. A 2D image is captured through a light camera and contains intensity values of the objects in the scene. 3D point clouds or meshes, on the other hand, consist of sampled points or 2D planes locally approximating a 2D surface embedded in the 3D domain. As such, the 3D modality also contain the depth information (as defined by the physical distance from the sensing modality) that is not included in 2D images. The 3D modality may also be augmented by texture information.

An object has many characteristic features: color, texture and shape are among the fundamental ones. In 2D domain, all of these can be taken into account, however, due to the limit caused by the appearance, it is not sufficient to represent an object completely from a single image. In contrast, in 3D domain, mostly, shapes of the objects are used that is provided by the depth information. Moreover, the quality of the objects, e.g., number of points in the cloud or number of meshes, is the only factor that limits the representation. In addition, if it is available, color and texture information can also be used in 3D domain.

The problem of object recognition generally can be cast to two closely related tasks: object instance recognition and object class recognition. Object instance recognition aims at identifying specific (previously seen) object instances. That is, the objects to be identified are already stored in the database of objects serving as templates. On the other hand, in object class recognition, the focus is on recognizing object instances of predefined object categories. This is usually a more challenging task because: (i) objects to be recognized have not seen before, (ii) visual differences across categories might be very small (such as chair and office chair), and (iii) visual differences within a category may be very high (such as different tree types). For example, in Figure 1.1 and 1.2 visual differences within “mug” and “guitar” classes are

illustrated respectively. In object class recognition, if the object to be recognized is an instance of the mug class, then the aim is to assign the class label "mug"; but in object instance recognition, the aim is to identify which specific mug it is.

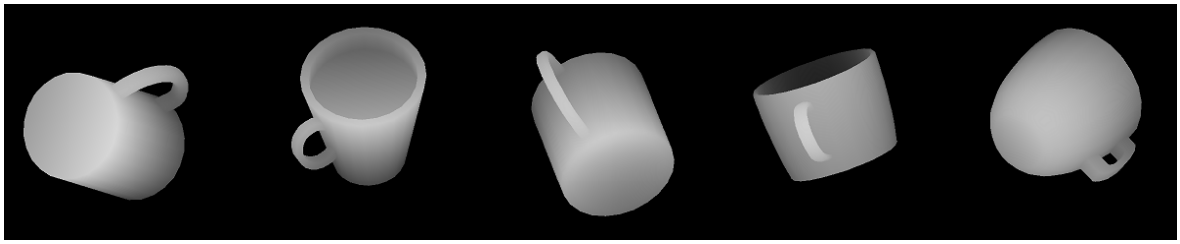


Figure 1.1. Arbitrary views of different objects from the mug class.

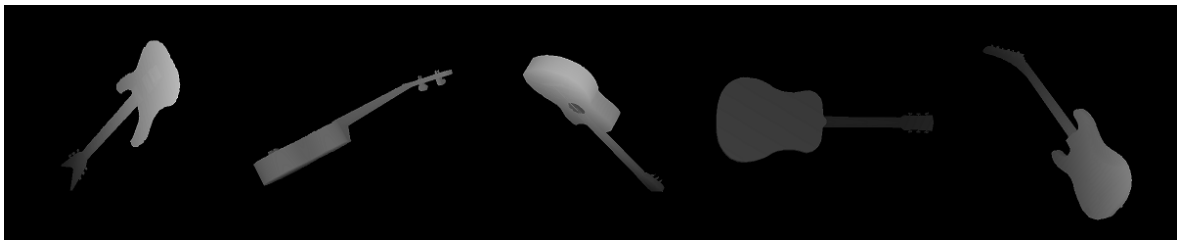


Figure 1.2. Arbitrary views of different objects from the guitar class.

The 3D object recognition problem can be tackled in two approaches; (i) The 3D model-based approach and (ii) The 2D view-based approach. 3D model-based approaches require complete or partial 3D models [1, 2], the associated algorithms run in the 3D domain. Their advantage is they contain a richer set of information, in other words, with appropriately developed methods all the information needed for recognition can be extracted. In view-based approaches 3D models are not directly used, but appearances from different viewpoints generated from the models are used. In other words, the 3D data can be projected onto a 2D plane and the resulting view images can be used for recognition. One advantage of view-based approaches is that all known 2D image analysis methods can be used as is for processing 3D objects.

In view-based approaches, each 3D model is represented by multiple 2D views (2D depth images), which can be used in three different ways to accomplish the recognition task. In the first, only one global descriptor can be extracted for each view and recog-

nition is performed by matching these descriptors, e.g., the Discrete Cosine Transform (DCT) [3] coefficients from the whole image. In this approach, one-to-one matching is performed. In the second one, local descriptors are extracted at the specified patches of the views. Then, views are matched by matching their local descriptors; in other words, for recognition, many-to-many matching is performed. Third approach uses the power of local descriptors and employ one-to-one matching scheme. In order to do this, local features of each view are encoded into single descriptors. Then, these descriptors are used in matching process.

The desideratum of successful object recognition is to provide the highest recognition accuracy with the lowest computation time in order to enable real-life applications such as robot guidance and scene understanding. The accuracy challenge should be addressed by designing object description schemes that are invariant to geometric transformations and robust against adverse effects such as sensor noise and irrelevant visual details while maintaining the discriminating aspects across a large corpus of object categories. The time challenge should be addressed by developing computational algorithms that can efficiently extract these descriptors.

A shallow pipeline for the particular way of performing 3D object recognition, that we employed, is as follows: (i) Obtain depth images from 3D models, (ii) extract keypoints from depth images, (iii) build descriptors around the keypoints, (iv) generate a visual codebook from descriptors, (v) encode features using the visual codebook, (vi) train classifiers using encoded descriptors, (vii) classify test objects using their encoded descriptors. More detailed version of this pipeline will be given in the following section.

In this thesis, we have evaluated several well-known keypoint detection methods [4–6], 2D local image descriptors [4, 6–10] and feature encoding methods [11–15] for object class recognition applied to depth images of 3D objects. We have worked with objects that do not have background clutter and there was no occlusion. Furthermore, the depth images that we generate, does not have any noise, which is not the case for real applications. Also, we mostly canalized on the accuracy challenge.

The contributions of this thesis can be given as follows:

- It presents performance evaluation of several well-known 2D methods applied to depth images for 3D object class recognition. To our knowledge this kind of study has not been done before, so it can be deemed as a guide that shows the effects of the methods, in this problem.
- Our approach uses depth images, and recognition is done with single depth image, hence, using a depth image sensor such as Microsoft Kinect [16] is possible to build a 3D object class recognition system. However, to build such system, further research must be done with noisy depth images or real data that has been captured with a 3D sensor.

1.1. 3D Object Class Recognition

The 3D object class recognition problem is defined as follows: "Given a depth image of a 3D object under an arbitrary view, how can we recognize the category of that object?". In Figure 1.3, several depth images from different objects and categories are illustrated.

The 3D object class recognition system that we have put into practice runs on depth images of 3D objects. We will, first give information about depth images, and then continue with presenting steps of 3D object recognition pipeline.

Depth image, also referred as range image or depth map, is an image where pixel values correspond to distances between the camera center line and the object surface. Depth images are usually produced by depth or 3D sensors. One of the several depth imaging techniques is stereo triangulation. In order to measure the depth using a stereo camera system, it is necessary to find corresponding points in different images. Dealing with the correspondence problem is one of the main disadvantages of this type of technique. For example, if image points in the scene lie inside a region with homogeneous intensity, solving the correspondence problem is difficult if not impossible. In Time-of-Flight (ToF) technique, depth image is obtained by measuring the time-

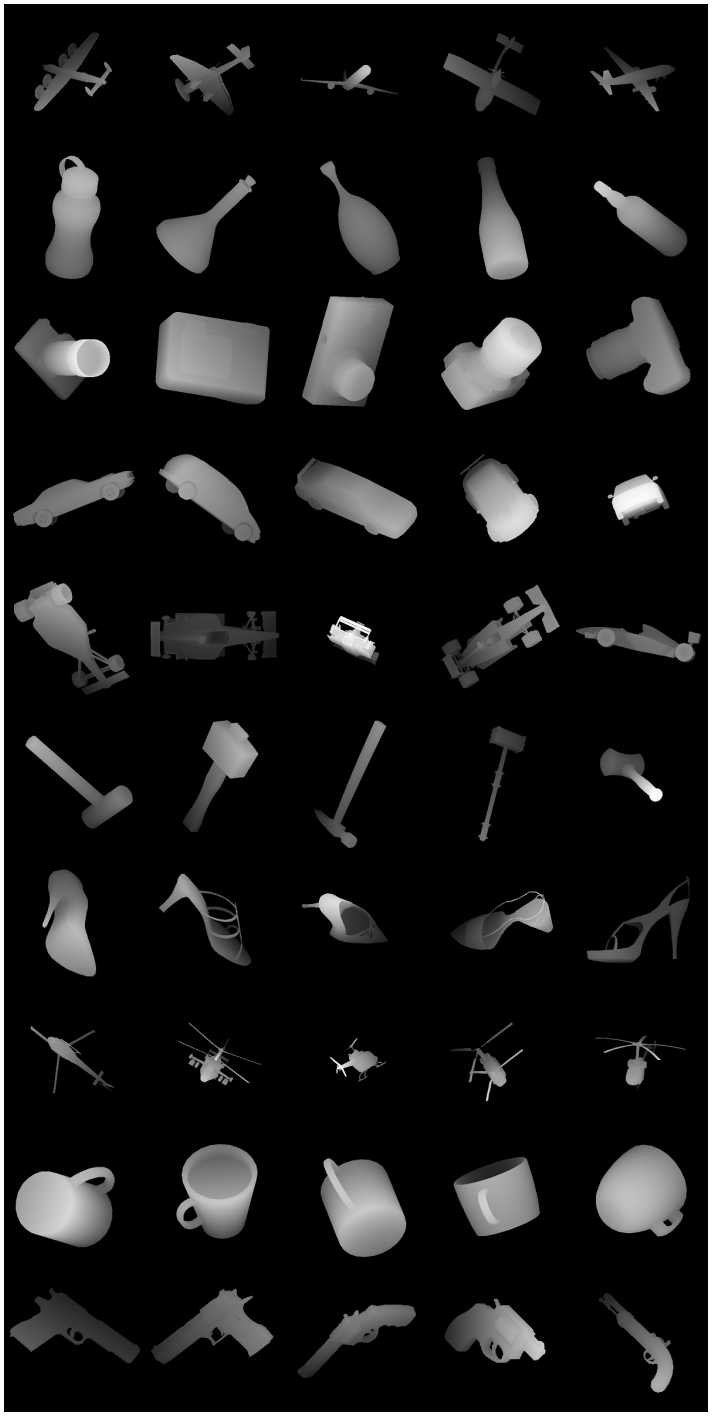


Figure 1.3. Depth image samples from different objects and categories. In each row different category is shown. In each column different object from corresponding category is shown. Shown categories are airplane, bottle, camera, car, formula car, hammer, heels, helicopter, mug and pistol respectively. These are samples from 3D-Net database. Background is colored as black which has a value of 0.

of- flight of a light pulse between the camera and the object. With the known time measurements and the speed of light, distances are obtained. One other depth imaging technique is using structured light, that is, by illuminating the scene with a specially designed light pattern, depth can be determined. For instance, the Microsoft Kinect device [16] uses this technique with infrared light pattern and an infrared camera, which is just a bit more sensitive to near-infrared range.

Instead of acquiring depth images with cameras or sensors, another approach to obtain depth images is to generate them. From a 3D object model, depth images can be generated by orthogonally projecting them to image plane from selected viewpoints and setting the distance from the 3D object point to the projection center as the depth value. Example projections of a 3D object to image plane from different view points can be seen in Figure 1.4.

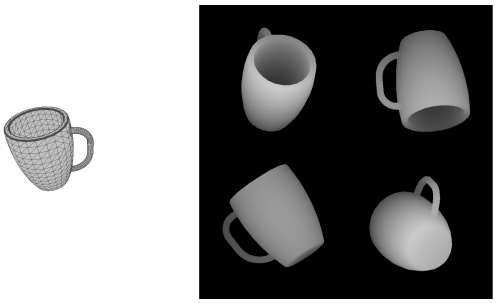


Figure 1.4. Projection of 3D model to 2D plane. Left: 3D model; Right: Projected depth images from four different viewpoints.

3D object class recognition system consists of two stages. In the first stage, system is trained using given training objects and in the second stage given test objects are identified using the trained system. These two stages have their own pipelines as illustrated in Figures 1.5 and 1.6 and they have common steps.

Pipeline of the training stage has the following steps:

- Normalize objects: In this step, all given 3D models are scale normalized. We also apply translation normalization to ensure that whole object lies in the 2D

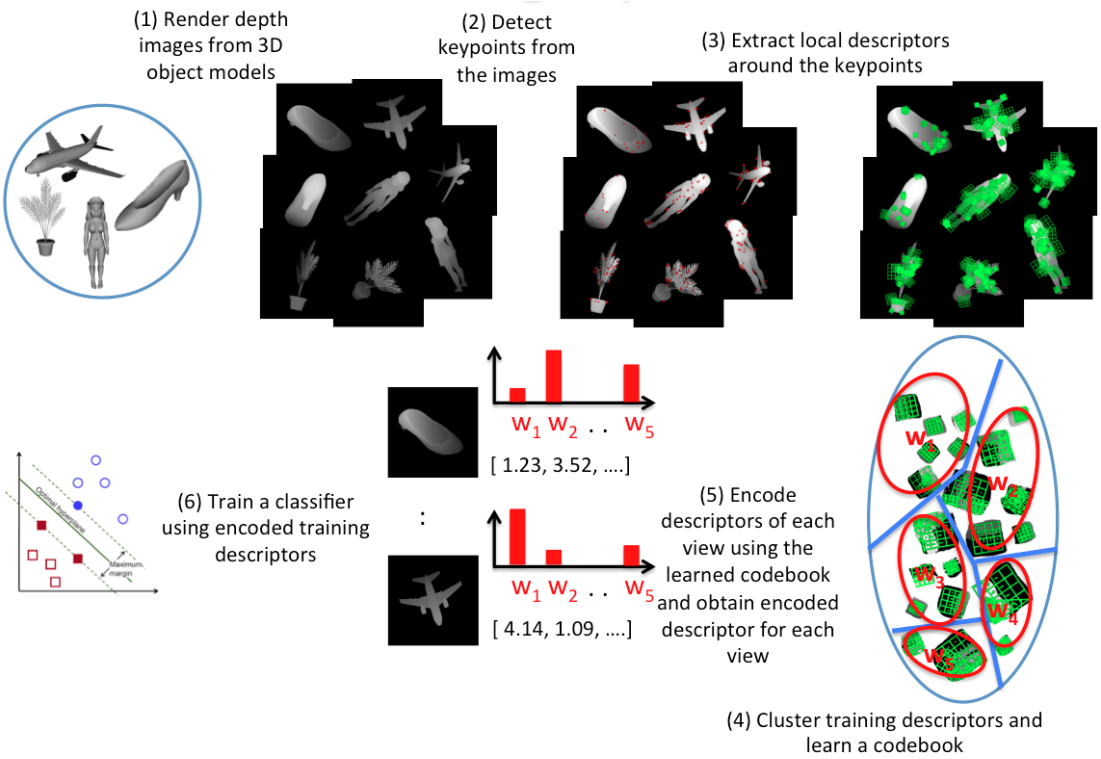


Figure 1.5. Training pipeline of a 3D object class recognition system.

image when it is rendered. Details of normalization are explained in Appendix A.

- Multi-view rendering: In this step we generate depth images from each 3D model from different viewpoints, which are distributed uniformly on a view sphere that surrounds the model. Details of multi-view rendering are explained in Appendix A.
- Extract local features from views: This step involves specifying points on the images (views) and then extracting features around those points. Points can be specified either by detecting salient ones or by sampling them. Hence, each view is represented by a set of local features that are built around the points. Keypoint detectors and local image descriptors are discussed in detail in Chapters 2 & 3 respectively.
- Learn dictionary from features: In this step, a visual dictionary is learned from extracted training features. The dictionary consists of visual words, which are in fact clusters of the given feature set. Details of methods for learning a dictionary

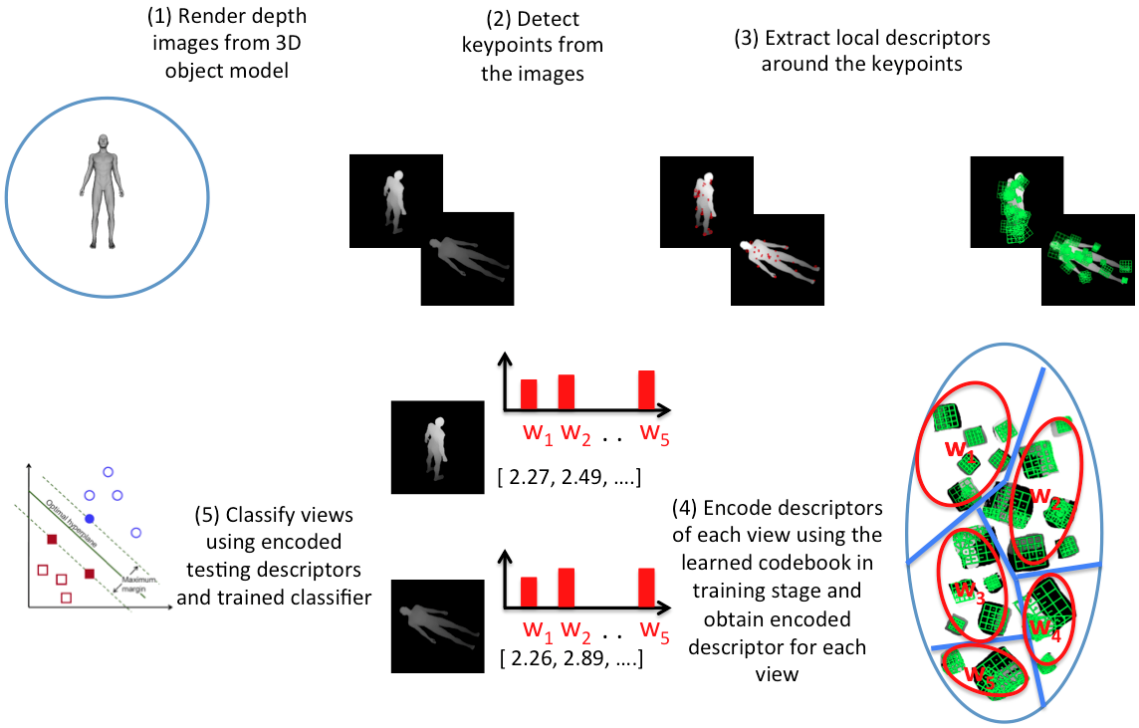


Figure 1.6. Testing pipeline of a 3D object class recognition system.

can be found in Section 4.1.

- Encode features of views into a descriptor: Features from each view are encoded into a descriptor using the learned visual dictionary. This is done by using a relation between the features that are obtained form the views and the words of the dictionary. Each encoding method uses a different relation, such as counts of words that the features are assigned to or distances between the features and words. After the encoding, each view is represented by a global descriptor instead of several local features. Details of different encoding schemes can be found in Chapter 4.
- Train classifiers from encoded descriptors: This is the final step of training stage. In this step, classifiers are trained so that any test object can be queried. There are several classification approaches that can be employed [17–20], where K Nearest Neighbor (KNN), Neural Networks (NN) and Support Vector Machines (SVM) are examples of most popular ones. In this work, we have chosen to use SVMs for their proven discriminatory power. For each object class, we have trained a

binary classifier, which distinguish between one of the classes and the rest (one-versus-all). Details of the SVMs can be found in Chapter 5.

Pipeline of the testing stage has the following steps:

- Normalize objects, Multi-view rendering & Extract local features from views: These steps are same as in the training stage.
- Encode features of views into a descriptor: This step is also same as in the training stage. It should be pointed out that, local features that are extracted from testing views did not play any role in learning the dictionary.
- Classify objects: Testing objects are classified using their views, where each view of an object is handled separately. In other words, each view of an object, result an independent classification decision. Though, another approach that uses multiple views for classification is also employed. Classification performances can be found in Chapter 6.

1.2. Related Work

Descriptors that are used for 3D object recognition can be classified into two groups as 3D shape descriptors, which are examples of 3D model-based approaches, and 2D image descriptors for view-based approaches.

One of the first studies about 3D shape descriptors is proposed by Ankerst *et al.* [21] in 1999. They have used 3D Shape Histograms. Later, in 2001, Osada *et al.* [22] proposed a method that uses shape function and it simply measures the distance between two random points on the 3D objects surface. Histogram of these distances of random points creates the descriptor. In 2002, Zaharia *et al.* [23] proposed a new shape descriptor which is based on 3D Hough transform. The studies [24–26] of Akgül *et al.* results better in performance than these previous works. They developed probabilistic generative description of 3D objects local shape. It describes 3D objects with multivariate probability density functions of chosen shape features. These methods need complete 3D models. Wohlkinger *et al.* improved the study [22] and

used it for partial 3D models [27]. They kept improving by increasing the used shape functions; they used area and angle differences in addition to distance. Wohlkinger *et al.* compared their studies [27–30] with other partial 3D shape descriptors [31–33]. Castellani *et al.* [34] and Zaharescu *et al.* [35] proposed methods for keypoint detection in 3D domain by using Difference-of-Gaussians. Chen *et al.* [36] proposed a method for measuring patch quality point-wise. Zhong *et al.* [37] and Mian *et al.* [38] proposed methods for measuring patch quality region-wise. However, last three methods work on fixed scale.

Another approach is to extract 2D image descriptors from the views of the 3D models. Chen *et al.* [39] introduced Light Field Descriptor (LFD). LFDs are computed using Zernike moments and Fourier descriptors on views of 3D model. Viewpoints are selected using vertices of a dodecahedron. Ansary *et al.* [40] proposed Adaptive Views Clustering (AVC). In AVC initial view count is 320 and by using adaptive view clustering with Bayesian information criteria, high discriminative views are selected. Then, a probabilistic method is employed for matching. Daras *et al.* [41] proposed Compact Multi-View Descriptor (CMVD). In CMDV, 2D views are taken from uniformly distributed viewpoints. For description extraction, the methods Polar-Fourier transform, Zernike moments and Krawtchouk Moments are employed. Shih *et al.* [42] proposed Elevation Descriptor (ED). In ED, 6 views are taken from the faces of the objects bounding box. The views contain the altitude information.

All approaches mentioned above, need 3D models for both training and testing stages. The view-based approach, which is explained in detail in the previous section, need 3D models only in the training stage. Testing can be done using only single view of a model. This approach need keypoint detectors, 2D local image descriptors and feature encoding methods. In order to extract local descriptors, first, interest points should be detected, since descriptors are built around them. The very first interest point detector is proposed by Harris and Stephens [43], the intuition in their work was to consider corners as keypoints. Later, it is improved and made multi-scale by Mikolajczyk and Schmid [44]. Another way of detection keypoints is to select local extrema of the responses of certain filters. Lowe [4] used Difference-of-Gaussians for

filtering the image in SIFT. Bay *et al.* [5] used a Fast Hessian detector in SURF. Rosten and Drummond proposed FAST [45] for corner detection. Later, it is improved by Mair *et al.* [46] and named AGAST. Leutenegger *et al.* used a multi-scale AGAST as keypoint detector in BRISK [6]. For convenience we will call it BRISK keypoint detector. In this thesis we have used SIFT, SURF and BRISK keypoint detectors to evaluate descriptors.

After the interest points are detected, local image descriptors are computed at the image patches around the points. One of the most popular descriptor is SIFT [4]. It is build from a grid of histograms of oriented gradients and it is 128 dimensional. It has been ranked as the reference keypoint descriptor for past 10 years because of its high descriptive power and robustness to the illumination changes. Moreover, it became a pioneer for many descriptors [5,7–9], which we call SIFT-like descriptors. One derivation for SIFT is PCA-SIFT [7], which reduces the dimension of the descriptors from 128 to any size using principal component analysis. In PCA-SIFT dimension is reduced, which provides lower matching time with the trade of higher descriptor building time. Mikolajczyk and Schmid proposed GLOH [8], which is an extension of SIFT with more descriptive power. The drawback of this descriptor is that it is slower than SIFT. Another SIFT-like descriptor is SURF [5], which has similar matching performances with SIFT. Prominent side of SURF is that it is a lot faster than SIFT. It also relies on histograms of gradients. It uses Haar-wavelet responses and builds 64 or 128 dimensional descriptors. Its dimension of 128 is not practical for large-scale and real-time applications. The dimension can be reduced using PCA or hashing functions [47], however these steps are also time-consuming. These aspects motivated research toward binary descriptors that we describe next.

Instead of building a descriptor and then reducing its size, Calonder *et al.* [48] proposed a short descriptor called BRIEF. BRIEF does not need dimensionality reduction and it is a binary descriptor. Binary descriptors have independent bits. The most important benefit of using binary descriptors is that they can use simple Hamming distance instead of using Euclidean distance. In building BRIEF descriptor, first, image is smoothed by a Gaussian to be less sensitive to noise. Then, the descriptor is built by

comparing intensity values of pre-selected 512 pairs of pixels. Selection of pairs is done randomly according to a Gaussian distribution built around the keypoint. Although BRIEF has a clear advantage on SIFT-like descriptors in the sense that computing speed and distance calculation speed, it is not rotation invariant. Rublee *et al.* [49] improved BRIEF by adding rotation invariance and proposed Oriented FAST and Rotated BRIEF (ORB). Then, Leutenegger *et al.* [6] attuned to this trend and proposed BRISK which is also a rotation invariant binary descriptor. Contrary to BRIEF they used a specific sampling pattern for selecting pixels for intensity comparison. Each point in this pattern contributes to many comparisons. The comparisons are divided into two subsets by their distance. Long-distance subset is used to estimate the direction of the keypoint, which brings rotation invariance. Short-distance subset is used to build the descriptor itself. Alahi *et al.* [10] proposed another binary descriptor FREAK. Starting point of FREAK was imitating human visual system, so, they used a retinal sampling pattern. Another difference of FREAK is that, it is a cascaded descriptor such that first 128 bits contain coarse information. Next 128 bits contain finer details and this goes on for all 512 bits. Each 128 bit of the descriptor is obtained by different subsample of point pairs. In this thesis we have selected SIFT and PCA-SIFT from SIFT-like descriptors and BRISK and FREAK from binary descriptors for evaluation.

Matching using local features is more feasible when they are coded in to one global descriptor. For this, several different encoding methods are proposed [11–15]. Common point in all encoding methods is that, they all need a dictionary. They differ from each other by the technique they use the dictionary. For dictionary creation, mostly used methods are k-means clustering and its variants and Gaussian Mixture Model (GMM) clustering. The most well-known approach is the classical Bag of Words (BoW) with hard assignments [11]. In the BoW approach, first, a dictionary is learned from extracted features. Then, each descriptor in the image is assigned to words in the dictionary. This is the so-called hard assignment approach. After assigning all the descriptors, resulting histogram is the encoded feature. In Kernel codebook encoding [12], descriptors are assigned to words in soft manner, so not only the closest word also other closer words takes role in encoding step. In both of these techniques, resulting encoded descriptor has the same size with dictionary. In Fisher Vector encoding [13],

GMM clustering is used and descriptor is built by concatenating first and second order differences between descriptors and words. Thus, the encoded descriptor size is equal to $2 \times d \times K$ where d and K are descriptor and dictionary sizes respectively. Then, Fisher Vector encoding is improved [50] by adding l_2 and power normalization steps and showed its superiority over BoW. Super Vector encoding [14] is similar to the Fisher Vector encoding. It has two variants one with hard assignments and another with soft assignments. It has a dimension of $K \times (d + 1)$. In VLAD encoding [15], differences between each descriptor and the words that it assigned to are accumulated. Then, these accumulated vectors are concatenated and l_2 normalized. The size of the resulting feature is $K \times d$. VLAD has also been improved by adding signed square root normalization [51], same as in power normalization done in Fisher Vector encoding. In addition, intra-normalization is applied by l_2 normalizing each accumulated cluster before concatenation. In this thesis, we have selected classical BoW approach with hard assignments, improved Fisher Vector encoding and improved VLAD encoding for evaluation.

The organization of this thesis is as follows: We will give detailed information about selected keypoint detectors, 2D local image descriptors and encoding methods in Chapters 2, 3 and 4 respectively. In Chapter 5, we will give information about classifier we used for recognition. Evaluation results and comments of keypoint detection, image description and encoding methods will be given in Chapter 6. Finally in Chapter 7, we will conclude.

2. KEYPOINT DETECTORS

In our evaluation pool, we have considered SIFT, SURF and BRISK methods for keypoint detection since they are proven to be robust keypoint detectors against rotation and scale, and also yield high repeatability [4–6]. In addition to these visual saliency-based methods, we have also implemented the dense sampling approach, where there is no detection process employed. The points are selected from the image by regularly sampling the image grid.

2.1. SIFT Keypoint Detector

SIFT keypoint detection is composed of 2 stages. The first stage is scale-space extrema detection. Simply, this well-known method searches every location of the image with different scales and identifies the candidate interest points. Second stage is keypoint localization. At each candidate keypoint, a stability check is performed and the ones that are not stable enough are eliminated. Points that are along the edges or the ones with that are not the real extrema are deemed to be unstable.

In order to detect extrema of the scale-space, we first need to define the scale-space, which is defined as a collection of function:

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y) \tag{2.1}$$

indexed by the scale variable σ, where $G(x, y, \sigma)$ is variable-scale Gaussian function, $I(x, y)$ is an image and $*$ is the convolution operator in x and y. The Gaussian function is defined as:

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-\left(x^2+y^2\right)/2\sigma^2} \tag{2.2}$$

To detect stable keypoint locations in scale-space, Lowe proposed [52] to use scale-space

extrema in the Difference-of-Gaussian (DoG) function

$$D(x, y, \sigma) = (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y) \tag{2.3}$$

$$= L(x, y, k\sigma) - L(x, y, \sigma) \tag{2.4}$$

where k is a constant multiplicative factor to separate nearby scales. Figure 2.1 visualizes the construction of $D(x, y, \sigma)$.

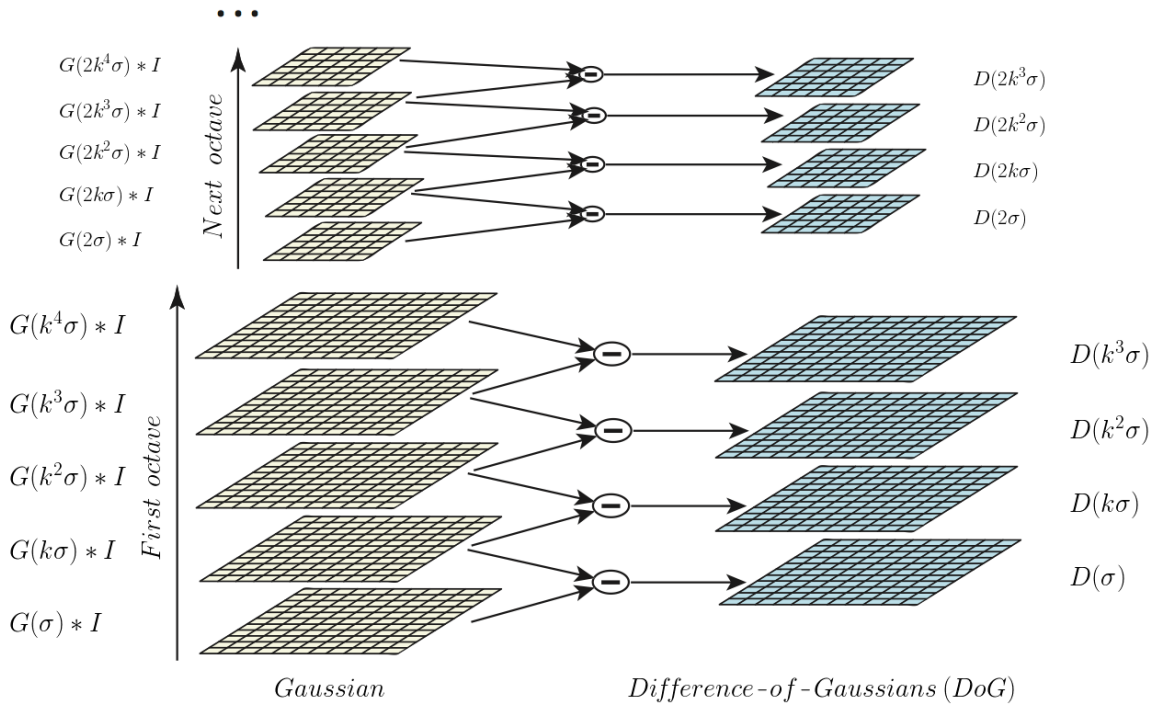


Figure 2.1. Gaussian scale-space and Difference-of-Gaussians [4].

After $D(x, y, \sigma)$ is constructed, local extrema (maxima and minima) can be detected. For this, each point in the image is compared to its 26 neighbor points (8 from current image and 9 per images one scale above and one scale below), as illustrated in Figure 2.2. If the intensity of the point is larger or lower than all other 26 neighbor points, then that point is selected as maximum or minimum.

Once the possible keypoints are detected, the points with low contrast and the ones located along the edges are eliminated. If the extrema are very close to each

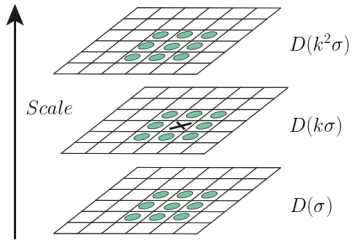


Figure 2.2. Extrema detection in Difference-of-Gaussians [4].

other, at least one of them is possibly a false extremum. Elimination of these points is accomplished by comparing the absolute value of the DoG scale-space at keypoints with a threshold. If the point's value is below the threshold, it is discarded. DoG scale-space value of points are calculated using Taylor expansion of the scale-space function, $D(x, y, \sigma)$, proposed by Brown [53]:

$$D(\mathbf{x}) = D + \frac{\partial D'}{\partial \mathbf{x}}\mathbf{x} + \frac{1}{2}\mathbf{x}'\frac{\partial^2 D}{\partial \mathbf{x}^2}\mathbf{x} \tag{2.5}$$

where D and its derivatives are evaluated at $\mathbf{x} = (x, y, \sigma)$. The location of extremum, $\hat{\mathbf{x}}$, is determined by taking the derivative of this function with respect to $\mathbf{x}$ and setting it to zero, giving:

$$\hat{\mathbf{x}} = -\frac{\partial^2 D}{\partial \mathbf{x}^2}^{-1}\frac{\partial D}{\partial \mathbf{x}} \tag{2.6}$$

Unstable extrema with low contrast can be eliminated by the function value at the extremum:

$$D(\hat{\mathbf{x}}) = D + \frac{1}{2}\frac{\partial D'}{\partial \mathbf{x}}\hat{\mathbf{x}} \tag{2.7}$$

which can be obtained by substituting Equation 2.6 into 2.5 . In addition, points at the edges need to be eliminated if they are poorly determined. To identify poorly defined extrema, their principal curvatures are examined. If an extrema is poorly determined

in the DoG function, than one of its principal curvatures has a large value, which is across the edge, and the other one has a small value, which is perpendicular to the edge. So if a point has a strong response only in one direction, it is discarded. Since eigenvalues of the Hessian matrix, $\boldsymbol{H}$, give the principal curvature, they can be used for elimination. $\boldsymbol{H}$ is given as:

$$\boldsymbol{H} = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix} \tag{2.8}$$

where D_{xx}, D_{xy} and D_{yy} are derivatives and estimated by taking differences of neighboring points. The eigenvalues of $\boldsymbol{H}$ is proportional to the principal curvatures of D. Since our concern is the ratio of the principal axes, thus the eigenvalues, the following test can be used to check if a point has a strong response only in one direction:

$$\frac{tr(\boldsymbol{H})^2}{det(\boldsymbol{H})} < \frac{(r+1)^2}{r}$$

where r is the ratio between the largest magnitude eigenvalue (α) and smallest magnitude eigenvalue (β), such that $\alpha = r\beta$. For instance if $r = 10$, then the points that have a ratio between principal ratios greater than 12.1 will be discarded.

2.2. SURF Keypoint Detector

SURF is the abbreviation for 'Speeded-Up Robust Features' and finds similar features to SIFT but with an approximation, thus it is much faster. In this section, first, information about integral images [54] will be given, since they are used in the process of extracting keypoints. Then we will explain the Hessian matrix based interest point detection and its proposed approximation. Then, we will show how SURF finds scale invariant keypoints without building an image pyramid.

Integral images allow fast computation of convolution filters with rectangular support. The entry of an integral image $I_\Sigma(x, y)$ represents the sum of all pixels in the

input image I within a rectangular region formed by the origin $(0,0)$ and (x,y),

$$I_{\Sigma}(x,y) = \sum_{i=0}^{i\leq x}\sum_{j=0}^{j\leq y} I(i,j) \tag{2.9}$$

After integral image is calculated, it only takes four additions to calculate the sum of the intensities over any upright, rectangular area, independent of the size. This is important in SURF keypoint detection since big sized are used.

For detection of keypoints, the Hessian matrix, $\boldsymbol{H}$, is computed for each point. Then points that have higher value for $det(\boldsymbol{H})$ are selected. So, for a point (x,y) in image I, Hessian matrix $\boldsymbol{H}(x,y,\sigma)$ at scale σ is defined as follows:

$$\boldsymbol{H}(x,y,\sigma) = \begin{bmatrix} L_{xx}(x,y,\sigma) & L_{xy}(x,y,\sigma) \\ L_{xy}(x,y,\sigma) & L_{yy}(x,y,\sigma) \end{bmatrix} \tag{2.10}$$

where $L_{xx}(x,y,\sigma)$ is the convolution of the second order Gaussian derivative masks $\frac{\partial^2}{\partial x^2}g(x,y,\sigma)$ and with the image I in point (x,y), and similarly for $L_{xy}(x,y,\sigma)$ and $L_{yy}(x,y,\sigma)$. Then, instead of using second order Gaussian derivative masks directly, Bay *et al.* [5] have used approximations to the these masks, as in Figure 2.3.

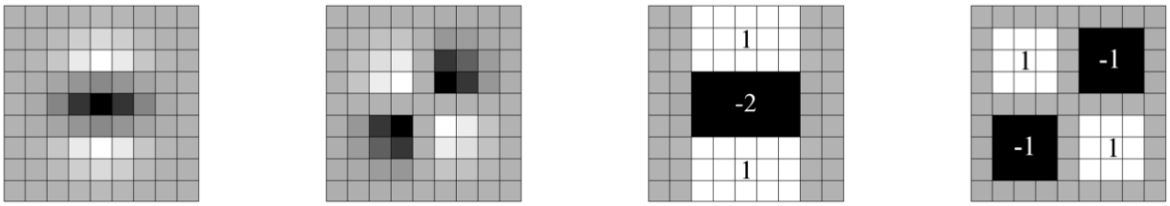


Figure 2.3. Left to right: Second order Gaussian partial derivatives in y-direction, xy-direction and their approximates [5].

Now, approximated Hessian matrix, $\boldsymbol{H}_{approx}$ can be written as:

$$\boldsymbol{H}_{approx}(x, y, \sigma) = \begin{bmatrix} \hat{L}_{xx}(x, y, \sigma) & \hat{L}_{xy}(x, y, \sigma) \\ \hat{L}_{xy}(x, y, \sigma) & \hat{L}_{yy}(x, y, \sigma) \end{bmatrix} \tag{2.11}$$

where $\hat{L}_{xx}(x, y, \sigma)$, $\hat{L}_{xy}(x, y, \sigma)$ and $\hat{L}_{yy}(x, y, \sigma)$ terms result from convolution of the approximate second order Gaussian derivative masks and with the image I in point (x, y). It should be noted that the integral images together with the binary masks in Figure 2.3, enable very rapid convolution. Thus, $det(\boldsymbol{H}_{approx})$ in Equation 2.11, can be easily calculated with the following:

$$det(\boldsymbol{H}_{approx}) = \hat{L}_{xx}\hat{L}_{yy} - (w\hat{L}_{xy})^2 \tag{2.12}$$

where w is the relative weight of filter responses, which can be taken as 0.9 as the authors suggested.

In order to detect keypoints in different scales, a scale-space should be built. Since convolution of integral images with binary filters with rectangular supports takes constant time independent from the filter size, increasing the size of the approximate filter is sufficient for building the scale-space, contrary to building an image pyramid as in SIFT. The use of filters with rectangular supports have low repeatability between the variations of the same image with different zoom levels. This is because it preserves high-frequency components, which might get lost in zoomed images. Thus, this can limit scale-invariance. However, authors report that it is not noticeable in the experiments. Figure 2.4 illustrates building scale-space both by building an image pyramid as in SIFT and by enlarging the approximate filter mask.

After the image is filtered with the specified filters, by applying non-maxima suppression in $3 \times 3 \times 3$ neighborhood (current scale, scale below and scale above) keypoints are detected in the image.

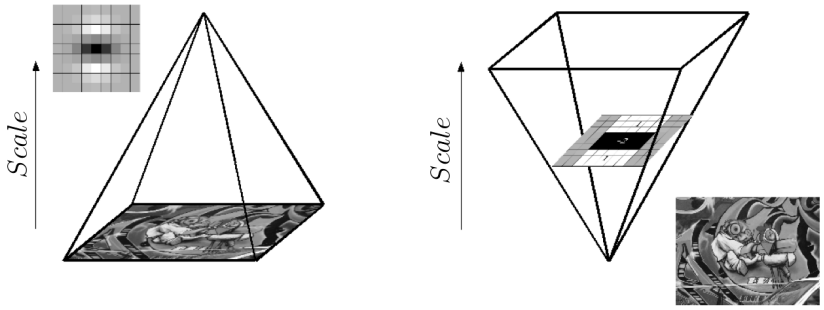


Figure 2.4. Left: Scale-space with the use of second order Gaussian derivative mask and downsampling image; Right: Scale-space with the use of approximate second order Gaussian derivative mask and enlarging the filter [5].

2.3. BRISK Keypoint Detector

BRISK keypoints are detected with the use of the so-called FAST [45] criterion. An interesting aspect of this technique is that it does not only find keypoints at the given scales but it searches deeper and finds true scales of the keypoints.

FAST criterion is a simple test for the given pixel p by examining a circle of 16 pixels (a Bresenham circle of radius 3 [55]) around it. For a point p to pass the test and selected as salient, intensities of at least 9 consecutive pixels on an arc of a circle centred on the pixel, should be all above or all below the intensity of point p by some threshold th. The concept is illustrated in Figure 2.5.

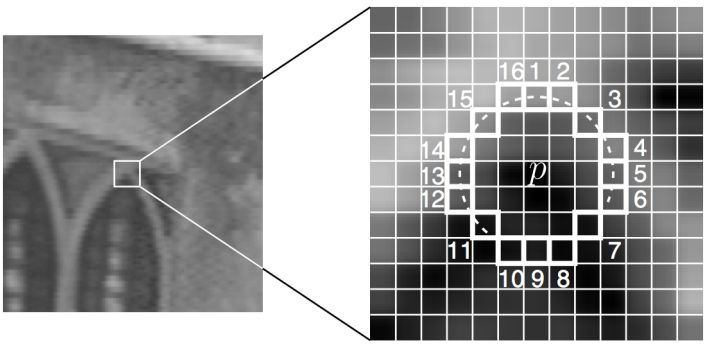


Figure 2.5. Application of FAST criterion on a point [45].

This test can be done faster by just testing the pixels 1, 5, 9 and 13 before testing the others since at least two of them should be above or below the intensity of point p by threshold th. For a salient point, FAST score is equal to maximum threshold that still makes that point pass the test.

In BRISK, image pyramid for scale-space representation is build using n octaves c_i and n intra-octaves d_i for $i = \{0, 1, ..., n-1\}$. Octaves are generated by downsampling previous octave by a factor of 2 and the original image is equivalent to first octave c_0. Intra-octaves are located in between octaves c_i and $c_{(i+1)}$. First intra-octave d_0 is obtained by downsampling the first octave c_0 by a factor of 1.5. Next intra-octaves are generated by downsampling the previous intra-octave by a factor of 2. There is also one more octave $c_{(i-1)}$ and intra-octave $d_{(i-1)}$ built which are virtual, and those will be used during non-maxima suppression to detect keypoints in first octave. Figure 2.6 shows the image pyramid with octaves and intra-octaves.

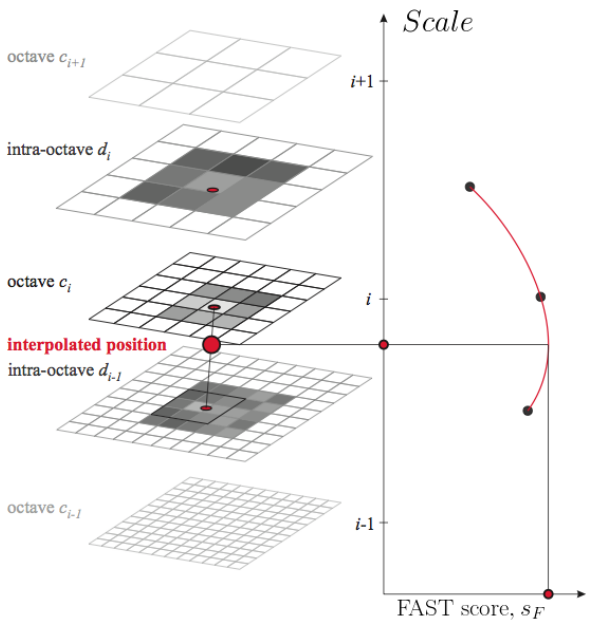


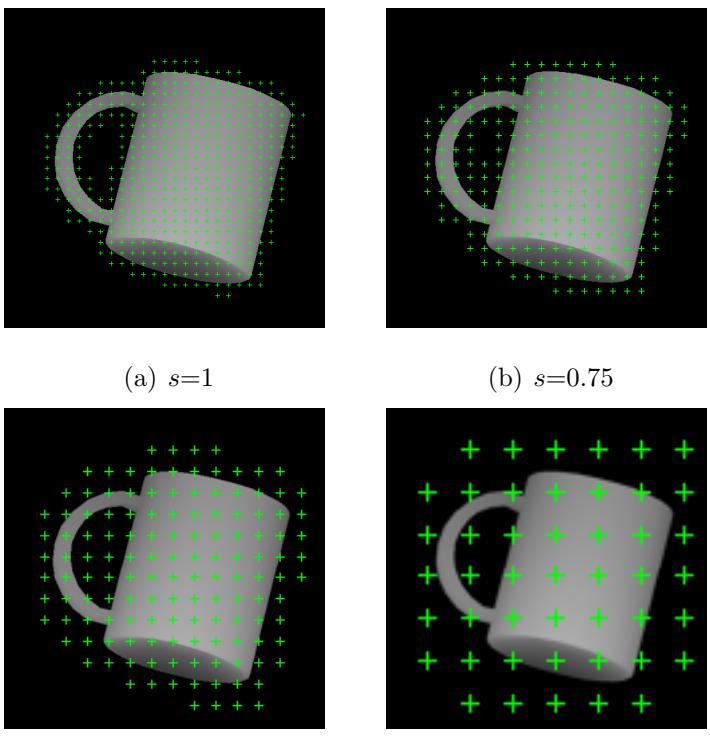
Figure 2.6. BRISK scale-space keypoint detection [6].

After the image pyramid is built, keypoints for each octave and intra-octave are found by FAST criterion. Then these points are selected using non-maxima suppression in scale-space. This has two steps: First, in each layer, points that the have maximum

FAST score around their 8 neighbors are selected. Then, the points that have higher FAST score than a layer below and a layer above are selected. Moreover by fitting a 1D parabola to the FAST scores, along the scale axis, true scale is estimated. Then, by re-interpolating the image coordinates, exact position of the point can be found. This is illustrated in Figure 2.6.

2.4. Dense Points

For dense points, a regular grid is placed on the image and the grid points are taken as keypoints. Standard detectors find keypoints in different scales, similarly dense points should be sampled at different scales. Figure 2.7 is an illustration of sampled dense points on an image with different scales. Although dense points are just sampled points, we will refer to it as dense keypoints in the sequel for convenience.



(a) s=1 (b) s=0.75

(c) s=0.5 (d) s=0.25

Figure 2.7. Dense keypoints. (a) is the original image. (b), (c) and (d) are the downsampled images with 0.75, 0.5 and 0.25 respectively. In these images grid sampling is done by 16 pixel steps.

3. 2D LOCAL IMAGE DESCRIPTORS

In our evaluation pool, we have chosen SIFT and PCA-SIFT from SIFT-like descriptor family and BRISK and FREAK from binary descriptor family.

3.1. SIFT Descriptor

SIFT descriptor extraction consists of 2 stages:

(i) Orientation assignment: Based on local image gradient directions and magnitudes, each keypoint gets at least one orientation assignment as described below.
(ii) Building the descriptor: Building the SIFT descriptor is also based on local image gradient directions and magnitudes. Basically, quantized histogram of local gradient directions gives the descriptors.

For orientation assignment to keypoints, first, Gaussian smoothed image, L, is selected using the scales of the keypoints as described in Section 2.1. This provides scale-invariance in orientation assignment step. Then, for image $L(x, y)$ the gradient magnitude $m(x, y)$ and orientation $\theta(x, y)$ are computed:

$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2} \tag{3.1}$$

$$\theta(x,y) = \tan^{-1}\left((L(x,y+1) - L(x,y-1))/(L(x+1,y) - L(x-1,y))\right) \tag{3.2}$$

For each keypoint, orientation histogram with 36 bins (choice of author) that covers the 360 degree range is formed from computed gradient orientations. Each sample added to the histogram is weighted by its gradient magnitude and by a Gaussian-weighted circular window σ for the Gaussian is 1.5 times of the scale of the keypoint.

After the histogram is formed, peaks in the histogram, which correspond to dominant direction of local gradients are selected. The highest peak and any other peak

within 80% neighborhood of the highest peak are selected. This yields multiple keypoints with different orientations at the same location.

The image gradient magnitudes and locations are taken around the keypoint location, specifically, in a 16×16 block. The coordinates of the descriptor and the gradient orientations in this patch are rotated relative to the pre-determined keypoint orientation to satisfy the rotation invariance. Furthermore, gradient magnitudes are weighted by a Gaussian function with σ equal to half of the width of the descriptor, as illustrated by the circle in Figure 3.1. These gradient orientations are accumulated into 8 binned orientation histograms over the 4×4 subregions, where the values of the histogram bins are calculated by summing the magnitudes of the gradients in that subregion. The histogram bin values are represented as the rose diagrams in the right image in Figure 3.1.

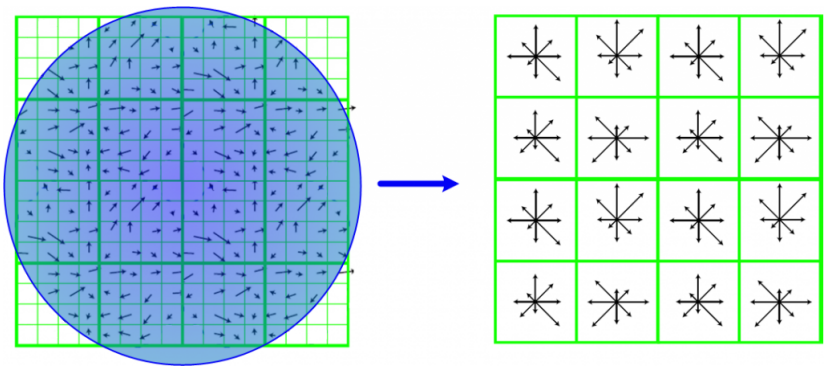


Figure 3.1. SIFT descriptor calculation [56].

Subregion histograms are concatenated into $4 \times 4 \times 8 = 128$ sized descriptor, and the resulting vector is unit normalized. Finally values of this vector are clipped at 0.2 to reduce the effect of large gradients, and then unit normalized again.

3.2. PCA-SIFT

In PCA-SIFT, the idea is to reduce the dimension of the SIFT descriptor via Principal Component Analysis (PCA). PCA is a standard technique for dimensionality reduction [57], by linearly projecting descriptor vectors to a lower dimensional feature

space. Details of PCA can be found in Appendix B.

PCA-SIFT has been applied in the literature with two different negihborhood sizes. In the first one [7], a 41×41 block is used and descriptor is built using horizontal and vertical gradients, which results in a $39 \times 39 \times 2 = 3042$ sized vector. Then, this vector is reduced to size 20 via PCA. In the second PCA approach, dimensionality reduction is applied to 128 sized SIFT features, which uses 16×16 blocks as described in Section 3.1.

3.3. BRISK

BRISK is a binary descriptor that can be created for a keypoint just by concatenating the results of pair-wise intensity comparison tests. The idea of applying intensity comparison test is proposed by Calonder *et al.* [48]. A comparison test can be defined as:

$$\begin{cases} 1 & \text{if } \mathbf{p}_i < \mathbf{p}_j \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

where $\mathbf{p}_i$ and $\mathbf{p}_j$ are intensities of sampled points around the keypoint. Point pairs that are used in comparison tests are sampled around the keypoint using the sampling pattern shown in Figure 3.2.

When sampling a point $\mathbf{p}_i$ according to the patter in Figure 3.2, a small patch around it is taken and Gaussian smoothing with standard deviation of σ_i is applied to it. The red circles in Figure 3.2 are illustrations of the sizes of the standard deviations used for each point. BRISK sampling pattern has $N = 60$ points, which yields $N(N-1)/2 = 1770$ sampling point pairs, denoted by $\mathcal{A}$:

$$\mathcal{A} = \{(\mathbf{p}_i, \mathbf{p}_j) \in \mathbb{R}^2 \times \mathbb{R}^2 \mid i < N \wedge j < i \wedge i, j \in \mathbb{N}\} \tag{3.4}$$

Over the set $\mathcal{A}$, two subsets are defined; short-distance pairing set $\mathcal{S}$ and long-distance

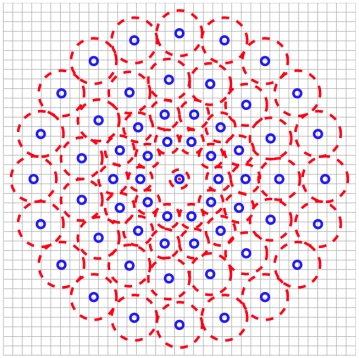


Figure 3.2. BRISK sampling pattern. The small blue circles are sampling locations. Bigger red circles are drawn at radius σ and corresponds to the smoothing factor of corresponding point [6].

pairing set $\mathcal{L}$:

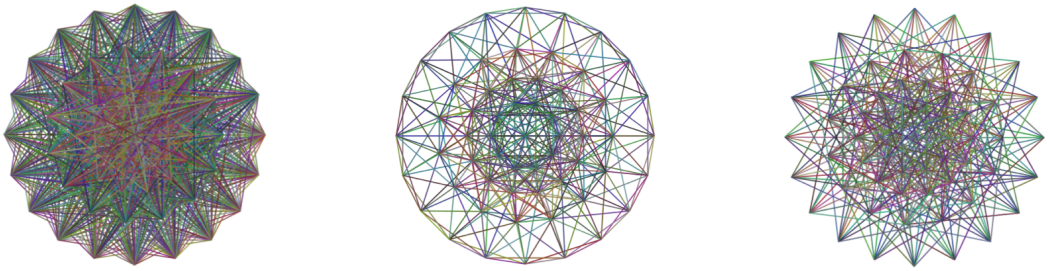
$$\mathcal{S} = \{(\mathbf{p}_i, \mathbf{p}_j) \in \mathcal{A} \mid ||\mathbf{p}_j - \mathbf{p}_i|| < \delta_{max}\} \subseteq \mathcal{A} \tag{3.5}$$

$$\mathcal{L} = \{(\mathbf{p}_i, \mathbf{p}_j) \in \mathcal{A} \mid ||\mathbf{p}_j - \mathbf{p}_i|| > \delta_{min}\} \subseteq \mathcal{A} \tag{3.6}$$

The thresholds $\delta_{max} = 9.75s$ and $\delta_{min} = 13.67s$ are given by the authors, where s is the scale of the keypoint. These values yield $L = 870$ long-distance pairings which are used for orientation estimation, 512 short-distance pairings which are used for building descriptor. The remaining 388 pairings are not used. These pairings at the pattern are illustrated in Figure 3.3.

Using the point pairs in $\mathcal{L}$, pattern direction, in other words orientation, is calculated as following:

$$\mathbf{g} = \begin{pmatrix} g_x \\ g_y \end{pmatrix} = \frac{1}{L} \sum_{(\mathbf{p}_i, \mathbf{p}_j) \in \mathcal{L}} \mathbf{g}(\mathbf{p}_i, \mathbf{p}_j) \tag{3.7}$$



(a) Long point pairs (b) Short point pairs (c) Unused point pairs

Figure 3.3. BRISK point pairs.

where $\mathbf{g}(\mathbf{p}_i, \mathbf{p}_j)$ is the local gradient:

$$\mathbf{g}(\mathbf{p}_i, \mathbf{p}_j) = (\mathbf{p}_j - \mathbf{p}_i)\frac{I(\mathbf{p}_j, \sigma_j) - I(\mathbf{p}_i, \sigma_i)}{||\mathbf{p}_j - \mathbf{p}_i||^2} \tag{3.8}$$

where $I(\mathbf{p}_i, \sigma_i)$ and $I(\mathbf{p}_j, \sigma_j)$ are smoothed intensity values of the points $\mathbf{p}_i$ and $\mathbf{p}_j$ respectively.

After the orientation is calculated, sampling pattern is rotated by $\alpha = arctan2(g_y, g_x)$ around the keypoint. Then, the binary string is created by applying intensity comparison tests to the point pairs in set $\mathcal{S}$:

$$b = \begin{cases} 1 & \text{if } I(\mathbf{p}_j^\alpha, \sigma_j) > I(\mathbf{p}_i^\alpha, \sigma_i) \\ 0 & \text{otherwise} \end{cases} \tag{3.9}$$

where $I(\mathbf{p}_j^\alpha, \sigma_j)$ and $I(\mathbf{p}_i^\alpha, \sigma_i) \in \mathcal{S}$ are smoothed intensities of rotated point pairs. By concatenating b for all bits, 512 bit string descriptor is created.

3.4. FREAK

FREAK descriptor is inspired by human visual system [10], in fact by retinal architecture to be more precise. Therefore, FREAK sampling pattern (Figure 3.4) is designed based on human retinal organization, which has more receptors near the foveal center, and their density exponentially drops towards to periphery. Similarly FREAK

sampling pattern has more points near the center and less points at larger distances. In addition, points that are near the center have smaller receptive fields, and that are farther away have larger receptive fields. Receptive field corresponds to the Gaussian smoothing kernel used at each point.

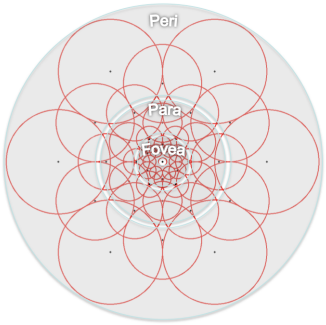


Figure 3.4. FREAK sampling pattern [10].

The FREAK descriptor is built by applying intensity comparison tests to the point pairs, as in BRISK (Equation 3.3). Hence, these two methods differ from each other in the way that they select point pairs. Since there are 43 points on the sampling pattern of FREAK, the maximum number of pairs can be $43 \times 42/2 = 903$. In FREAK, a point pair selection method is applied and 512 point pairs are selected such that these pairs are the most uncorrelated pairs with the highest variance according to [49]. This yields a feature with high descriptiveness. Steps of the point pair selection method are as follows:

- Create a matrix from nearly fifty thousand keypoints. Each row of the matrix corresponds to a 903 dimensional binary descriptor.
- Compute the mean of each column. A mean value of 0.5 leads to highest variance in a binary distribution since the components are independent. According to [49], in order to obtain a feature with high descriptiveness, high variance is desired. Therefore 0.5 mean is desired.
- Reorder the columns with respect to the variance in descending order.
- Keep the column with the highest variance (or the one which has the mean closest to 0.5). Iteratively search and select the columns that have low correlation with

the previously selected columns.

- Each column stands for a point pair. Selecting them with high variance and low correlation leads to building a highly descriptive feature.

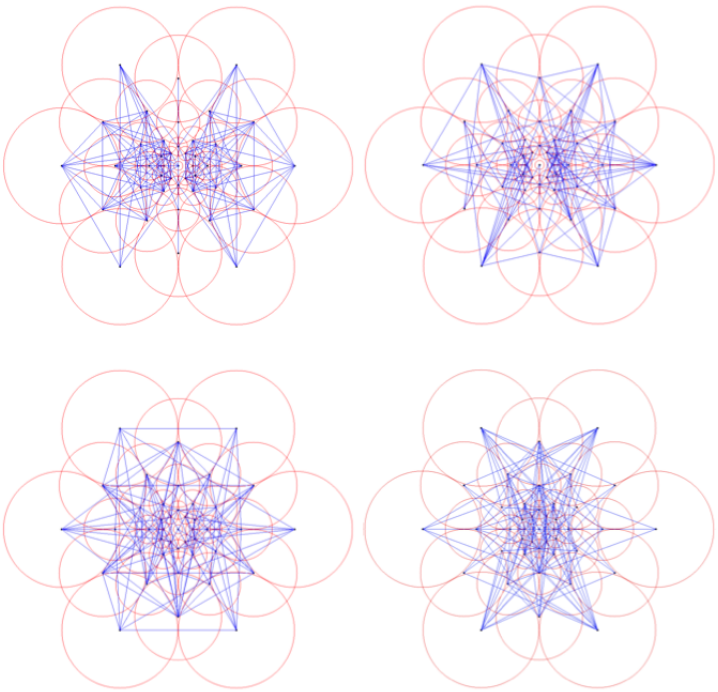


Figure 3.5. Selected point pairs in FREAK sampling pattern for descriptor generation. Top left is the first point 128 pairs, top right is the second 128 point pairs, bottom left is third point pairs 128 and bottom right is last 128 point pairs [10].

Amazingly and also as somewhat to be expected, this automatic point pair selection approach yields point pair selection with a structure. This structure mimics the behavior of the human eye. The selected 512 point pairs are in 4 groups with 128 point pairs per group (each group can be seen in Figure 3.5). The pairs in the first group mostly use points that are farther from center whereas the last group mostly uses the points that are closer to the center. This is similar to the human eye behavior since we first use the outer receptive fields of our eye to detect initial location estimates of the objects, and then use more densely distributed inner receptive fields for detecting exact locations of objects.

For orientation detection, different point pairs are used and these pairs are selected from the structure in Figure 3.6. These pairs are selected from symmetric points with respect to the center. The sum of estimated local gradients over these pairs gives the orientation as in BRISK (Equation 3.7).

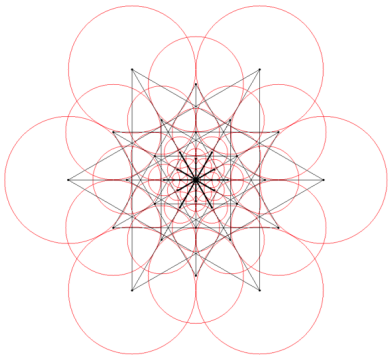


Figure 3.6. Selected point pairs in FREAK sampling pattern for orientation calculation [10].

4. FEATURE ENCODING METHODS

In this chapter, we will provide the details of computing a single descriptor from a given set of features, which is known as feature encoding in the literature. Feature encoding involves two steps: learning a dictionary and encoding features using learned dictionary.

4.1. Learning the Shape Dictionary

Dictionary generation is simply clustering the given data into partitions such that the overlap between these partitions is low and partitions together represent the information content of the whole data as much as possible. We have investigated two different dictionary generation methods: K-means and Gaussian Mixture Models (GMM). K-means is a well-known algorithm without any prior knowledge of the data model, hence, a non-parametric approach. In contrast, GMM is a parametric probability density function represented as a weighted sum of Gaussian distributions. Each Gaussian stands for a "word in the dictionary" and the mean value of each of the Gaussians stand as a visual word center.

4.1.1. K-means Clustering Algorithm

Given N features $\{\mathbf{x}_1, ..., \mathbf{x}_n\}$, the goal of the K-means is to find K centers $\{\mathbf{c}_1, ..., \mathbf{c}_k\}$ and assignments of descriptors $\{q_1, ..., q_n\} \in \{1, ..., K\}$ to the centers such that the error (energy), E, between the cluster centers and assignments is minimized. E can be formulated as:

$$E(\mathbf{c}_1, ..., \mathbf{c}_k, q_1, ..., q_n) = \sum_{i=1}^{N} ||\mathbf{x}_i - \mathbf{c}_{q_i}|| \tag{4.1}$$

where $||.||$ is l_2 norm. First algorithm for minimizing E was proposed by Lloyd [58]. In this early version, initial centers are selected randomly and iteratively updated by

assigning all descriptors to these clusters in each iteration until E converges or maximum iteration limit is reached. This algorithm gives satisfactory result, unfortunately for large databases, high dimensional features, large dictionary size or any combination of these three, this algorithm runs very slow.

To overcome this issue, Elkan [59] proposed another approach for minimizing E, which avoids most of the calculations when assigning descriptors to the cluster centers, with the use of triangle inequality:

$$||\mathbf{p}_i - \mathbf{p}_k|| \leq ||\mathbf{p}_i - \mathbf{p}_j|| + ||\mathbf{p}_j - \mathbf{p}_k|| \tag{4.2}$$

where $||.||$ is the distance metric and can be selected as l_2 norm and $\mathbf{p}_{i,j,k}$ are descriptors in the data or cluster centers. The idea behind this algorithm is that, when a center is updated, if its location does not change too much, then descriptor-to-center assignment computations for that center can be avoided. To determine if computations are needed or not, upper and lower bounds for each updated center is calculated using triangle inequality. With Elkan's calculation avoiding approach, algorithm becomes faster than the original one. Yet it is not fast enough to query the dictionary for large set of descriptors. For querying the dictionary, another approach can be used.

For querying the dictionary, a KD-tree [60] based search algorithm can be employed. A KD-tree builds a structure that can efficiently index high dimensional vector spaces. This indexing provides faster searching of the data.

KD-tree algorithm recursively partitions the d dimensional feature space into hyper-rectangles for a given set of descriptors $\{\mathbf{x}_1, ..., \mathbf{x}_k\}$. In our case, this descriptor set is the dictionary that has been created. Data partitions are organized into binary trees with the root node corresponds to whole feature space. Each partition is refined by dividing it into two halves by thresholding a specified dimension. The splitting dimension is selected such that it has the largest variance of that partition and the threshold value is selected as mean or median value of that dimension. Lowermost partition contains only one descriptor from the given set. After this tree is built, each

query descriptor is matched to one of the leaves by using the pre-determined dimension and threshold values.

4.1.2. Gaussian Mixture Models (GMM)

A GMM is a mixture of K multivariate Gaussian distributions,

$$p(x \mid \lambda) = \sum_{i=1}^{K} \pi_i p(x \mid \mu_i, \Sigma_i) \tag{4.3}$$

where x is the given data, which is d dimensional, π_i stands for prior probabilities (or weights) of each Gaussian component, and $p(x \mid \mu_i, \Sigma_i)$ represents Gaussian distribution,

$$p(x \mid \mu_i, \Sigma_i) = \frac{1}{\sqrt{(2\pi)^d \det \Sigma_k}} \exp\left[-\frac{1}{2}(x-\mu_i)' \Sigma_i^{-1} (x-\mu_i)\right] \tag{4.4}$$

with mean vector μ and covariance matrix Σ. Sum of the weights are equal to 1, $\sum_{i=1}^{K} \pi_i = 1$. The complete Gaussian mixture model is parameterized by mean vector, covariance matrices and mixture weights. To represent these parameters, the following notation is used:

$$\lambda = \{\pi_i, \mu_i, \Sigma_i\} \text{ for } i = 1, ..., K \tag{4.5}$$

Now, learning the dictionary is equal to learning the parameters of the GMM $p(x \mid \lambda)$, which is usually done by maximizing the log-likelihood of the data. For this problem Expectation Maximization (EM) algorithm [61] is employed. We have used its Matlab implementation in *VLFeat Toolbox* [62].

4.2. Feature Encoding Methods

After feature quantization, one needs to obtain a descriptor vector from the totality of a quantizer feature vectorsin the image. We have experimented with three methods, namely, Bag-of-Words (BoW), VLAD and Fisher Vector (FV). BoW simply creates frequency histograms of the given feature set. In VLAD, differences between features in the given set and words in the dictionary is accumulated, in other words distances of features to the words are encoded. Finally in FV, mean and covariance deviation vectors are stacked to create the encoded feature.

4.2.1. Bag-of-Words (BoW)

As mentioned above, BoW creates frequency histograms of the given feature set. In other words, by using assignment indexes, a frequency histogram is generated for given descriptor set. Then, histogram is unit normalized. This histogram has the same size of the dictionary is generated. Figure 4.1 illustrates example BoW encoding.

4.2.2. VLAD

For a given descriptor set $\mathbf{x} = \{\mathbf{x}_n, n = 1, ..., N\}$, each desciptor is assigned to its nearest word $NN(\mathbf{x}_n) = \arg\min_{\mathbf{c}_i} ||\mathbf{x}_n - \mathbf{c}_i||$. Afterwards, for each visual word, differences of the visual word and descriptors that are assigned to that visual word are accumulated to create v_i:

$$\mathbf{v}_i = \sum_{\mathbf{x}_t : NN(\mathbf{x}_t) = \mathbf{c}_i} \mathbf{x}_t - \mathbf{c}_i \tag{4.6}$$

This characterizes the distribution of the vectors with respect to the center. Then, signed square rooting (SSR) function [15], i.e. $sign(z)\sqrt{|\ z\ |}$, is applied to all the $\mathbf{v}_i$'s, and each $\mathbf{v}_i$ is l_2 normalized, $\mathbf{v}_i = \mathbf{v}_i / ||\mathbf{v}_i||_2$. The resulting $\mathbf{v}_i$'s are concatenated to form the final VLAD descriptor.

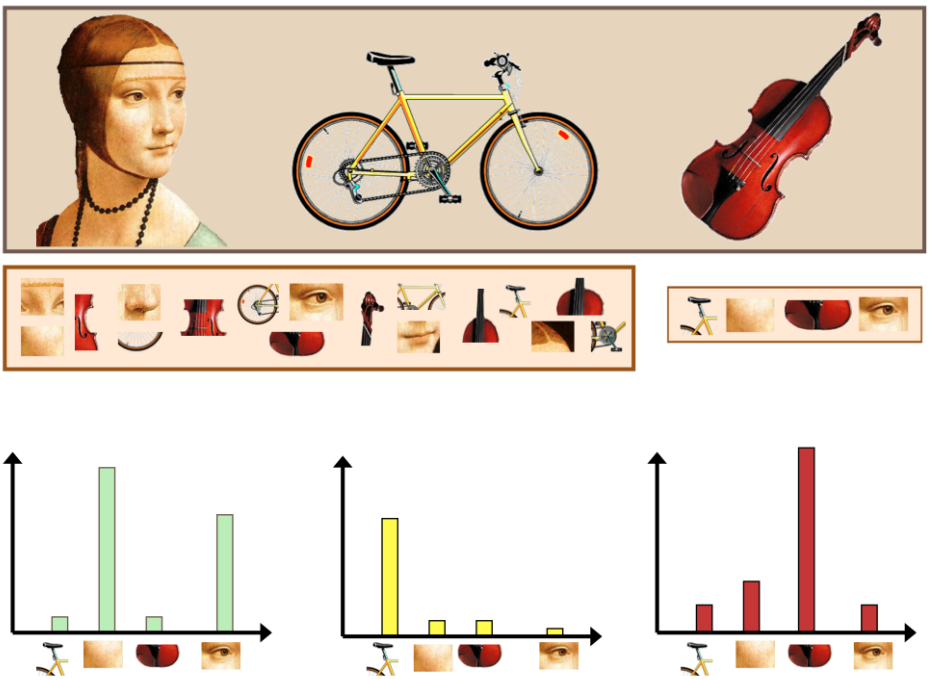


Figure 4.1. Example BoW encoding [63]. Top region of the image shows different images. In the middle left there is extracted descriptor from those images and in the middle right there is dictionary learned from those descriptors. In the bottom, frequency histograms of each image can be seen.

4.2.3. Fisher Vector Encoding

For a given set of descriptors $\mathbf{x} = \{\mathbf{x}_1, ..., \mathbf{x}_n\}$, FV encoding is constructed by first order and second order differences between the descriptors and dictionary words, as formulated in Equations , thus dictionary should be learned by GMM clustering to enable second order difference calculation. For $\mathbf{x}$, let $\mathbf{q}_{ik}$ be the soft assignments of N descriptors to K Gaussian components:

$$\mathbf{q}_{ik} = p(k \mid \mathbf{x}_i, \lambda) = \frac{\pi_k p_k(\mathbf{x}_i \mid \lambda)}{\sum_{j=1}^{K} \pi_j p_j(\mathbf{x}_i \mid \lambda)} \tag{4.7}$$

where $p(\mathbf{x} \mid \lambda)$ is Gaussian distribution (Equation 4.4) and $\{i = 1, ..., N\}, \{k = 1, ..., K\}$. Then, define the vectors:

$$u_{ik} = \frac{1}{N\sqrt{\pi_k}} \sum_{i=1}^{N} q_{ik} \frac{x_i - \mu_k}{\sigma_k} \tag{4.8}$$

$$v_{ik} = \frac{1}{N\sqrt{2\pi_k}} \sum_{i=1}^{N} q_{ik} \left[(\frac{x_i - \mu_k}{\sigma_k})^2 - 1 \right] \tag{4.9}$$

FV of $\mathbf{x}$ is concatenation of $\mathbf{u}_k$ and $\mathbf{v}_k$ vectors for all K components. After this, SSR function is applied to vector as in VLAD, and then, vector is l_2 normalized.

5. CLASSIFICATION

For classification, we have employed Support Vector Machines (SVM). For experimental cases involving classifier fusion, we have used the basic majority voting scheme.

5.1. Support Vector Machines (SVM)

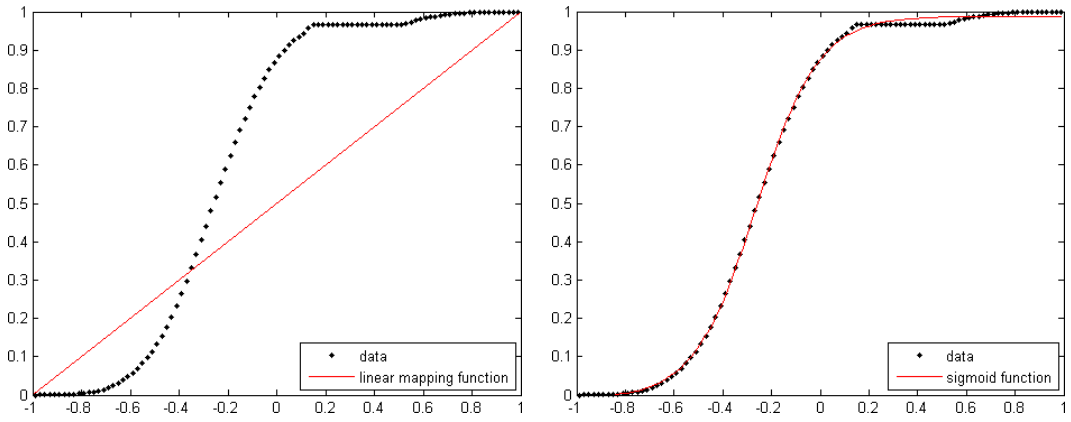
We have selected SVMs due to its known state-of-the-art performance in classification problems. As SVM involves regularization, it is known to be robust against over-fitting provided a rich and varied training set is provided. SVMs deliver a unique solution, as the training problem is defined as convex optimization. This is an advantage compared to NN, which have multiple solutions associated with local minima and for this reason may not be robust over different samples. Also, with the use of the kernel trick, SVMs gain flexibility in the choice of the form of the decision boundary that separates the data. We have used linear SVM, though SVM with RBF kernel could be employed. Lastly, SVM is an approximation to a bound on the test error rate, and there is a substantial body of theory behind it that suggests it should be a good idea. Details about the theory of SVM can be found in Appendix C.

We have used SVM with one-versus-all approach, and trained a classifier for each class. In this approach, classifiers distinguish between one of the classes and the rest.

5.1.1. Mapping Distances to Probabilities

SVM classification results a distance value for each query. This value is obtained by the distance between the new query and the optimal hyperplane. Thus, it depends on the training data because the optimal hyperplane is found by using the training data. Since we are using n binary classifiers, we cannot use these distance values directly. This is because each classifier is trained with the data which has different labels for each binary classification, hence, distances of samples to the optimal hyperplane would

be different. To overcome this issue, distances should be mapped to probabilities and mapping should be done correctly. A linear map would result incorrect probabilities for distances, thus, a sigmoid function is used for mapping as in [64]. Both mapping approaches are illustrated in Figure 5.1.



(a) linear mapping (b) sigmoid fitting

Figure 5.1. Score to probability mapping. x-axis is cumulative histogram of SVM distances, y-axis is probabilities. (a) is erroneous linear mapping. (b) is mapping via sigmoid function.

5.2. Majority Voting

Different description methods yield different assignments with different probabilities to query images. For instance using SIFT keypoints with BRISK descriptors and applying BoW encoding gives different assignments than using Dense keypoints with SIFT descriptors and applying FV encoding. In the case that accuracy of different methods is not sufficient, applying decision level majority voting to the assignments of these methods could increase the accuracy. Three different majority voting schemes are employed

In the first strategy, for each query image, the number of votes that each class got over different methods is counted and the class that got the most assignments is selected. If there are more than one class that got the most assignments, then the one with the highest average probability is selected. The downside of this strategy is that, if a class gets most of the assignments but with low probabilities, that class is still selected.

In the second strategy, the average probability of each class is calculated, and the one with the highest average probability is selected. This strategy overcomes the downside of the first strategy.

In the third and the last strategy, class that has the highest probability over all assignments is selected. For convenience, we will respectively use the terms maximum assignment pooling, average probability pooling and maximum probability pooling to these three approaches in the sequel.

6. EVALUATIONS

In this section we will present experimental results in detail. First, we will give information about used databases and experimental setup. Then, we will give information about implementation details. Next, we will start presenting experimental results with comments and observations.

6.1. 3D Model Databases

There are plenty of publicly available 3D model databases and each of them has models from different types of categories. Princeton Shape Benchmark (PSB) [65] contains a total 1814 models from a diverse range of 161 classes such as, airplane, human, monster truck, sword, tree, etc. This database has its own splits as train and tests sets, each having 907 models and the number of models for classes varies between 4 and 100. McGill 3D Shape Benchmark (MSB) [66] has two sets of databases, one for articulated models and one for non-articulated models. There are 10 different articulated models such as, ant, crab, octopus, snake, etc. with a total 246 models and in non-articulated set there are 9 models with a total 202 models. Airplane, chair, cup, table are some example classes of non-articulated set. Purdue Engineering Shape Benchmark (ESB) [67] has 45 categories with a total 865 models. These models are mechanical parts such as bearing, gear, handle, and housing. 3D-Net Database [30] has also two sets of databases. First one has 10 classes with a total of 351 models, second one has 60 classes with a total 1373 models. For our tests, we have chosen the 60-class version of the 3D-Net database and the PSB database since they contain higher number of models with diverse categories than others.

6.2. Experimental setup

We have prepared different setups to use the two above mentioned databases. First, we split the databases into two, based on the richness of each category in terms of the number of models within. A class is defined as rich if it contains at least 20

models, and defined as poor if the number of models is less than 20. We have used rich sets for evaluation. The resulting setups are as follows:

- **3D-Net-Rich:** Contains 1094 models from 27 classes. Model count for each class vary between 20 and 83.
- **PSB-Rich:** Contains 739 models from 21 classes. Model count for each class vary between 20 and 100.
- **3D-Net-Rich + PSB-Rich:** Same classes of 3D-Net-Rich and PSB-Rich are merged, so number of classes is 41 and contains a total 1833 models. Model count for each class vary between 20 and 152.

For the 3D-Net-Rich setup, we randomly split the models into train and test sets. For the PSB-Rich setup, we used the given splits. Initially, we have generated 200 2D views for each model. For the training set, we have chosen 100 of them for each model; for the test set, we have chosen 50 of them for each model, where each test view is handled as an independent query. This means that we are training the system with 100 views of each model and trying to identify a query model by a single view. These two sets are independent such that the training and test sets contain views of different models, that is, no model supplies views for both training and test sets.

We have 4 different point sets (3 keypoint sets and 1 sampled point set), 4 different local image descriptors and 3 different feature encoding methods. Their combination produces 48 different methods. We have tested the 3D-Net-Rich setup with all of these methods. For the PSB-Rich and the combination of the 3D-Net-Rich + PSB-Rich setup, we have selected a subset of these 48 methods, which mostly consists of the best performing ones on 3D-Net-Rich setup.

6.3. Implementation Details

We have used the Matlab implementations of the methods from *Matlab Computer Vision Toolbox (MCVT)* and *VLFeat Toolbox* [62]. BRISK and SURF keypoints and BRISK and FREAK descriptors are used from *MCVT* and the rest of the methods are

used from *VLFeat Toolbox*. Parameters that are used for the methods are as follows:

- BRISK Keypoints:
  - (i) $MinContrast = 0.7$
  - (ii) $MinQuality = 0.7$
  - (iii) $NumOctaves = 4$
- SURF Keypoints:
  - (i) $MetricThreshold = 2500.0$
  - (ii) $NumOctaves = 4$
  - (iii) $NumScaleLevels = 3$
  - (iv) $InitialScale = 1.6$
- SIFT Keypoints & Descriptors:
  - (i) $FirstOctave = 0$
  - (ii) $Octaves = 4$
  - (iii) $Levels = 3$
  - (iv) $k = \sqrt{2}$
- Dense Keypoints:
  - (i) $Octaves = 4$
  - (ii) $Step = 4$
- K-Means:
  - (i) $Initialization =$ Random
  - (ii) $Distance = L_2$
  - (iii) $Algorithm =$ Lloyd
  - (iv) $NumTrees = 2$
  - (v) $MaxNumComparisons = 100$
- GMM:
  - (i) $Initialization =$ K-Means
  - (ii) $CovarianceBound = maximum\ variance\ value\ of\ descriptors \times 0.0001$
- SVM:
  - (i) $BiasMultiplier = 1$
  - (ii) $Epsilon = 0.001$
  - (iii) $MaxNumIterations = 100 \times train\ sample\ count$

For rest of the parameters default implemented ones are used.

6.4. Experimental Results

In this section, we will present and discuss experimental results on the described setups. We will start by presenting the quality of learned dictionaries and the effect of different dictionary sizes over the performance. After that, we will show the effect of the PCA dimension for the SIFT descriptor. After specifying the parameters in the light of the presented information, we will show recognition performances of each setup and explain effects of keypoint extraction methods, local feature description methods and feature encoding methods on the performance. Next, we will give information about recognition performances and confusions between categories. Then, we will show the change in performance by increasing the used number of views for making a classification decision. Finally, we will show how performance is affected when the classification results of multiple methods are combined.

Accuracy is calculated as follows:

$$\frac{\text{number of true positive} + \text{number of true negative}}{\text{number of all positive} + \text{number of all negative}}$$

6.4.1. Quality of the Dictionary

In order to get discriminative encodings, one needs a well-learned dictionary. One way to understand the quality of a learned dictionary is to analyze its convergence behavior. Figure 6.1 illustrates the convergence of K-means algorithm. As the number of iterations increase, the reconstruction error converges. This error stands for the total distance between the cluster centers (words) and the assigned descriptor vectors.

Similarly, for the GMM algorithm, we can look at Figure 6.2. As noted in Section 4.1.2, the parameters of GMM is estimated by maximizing the log-likelihood of the data. We can see from the figure that, as the number of iteration increases log-likelihood is

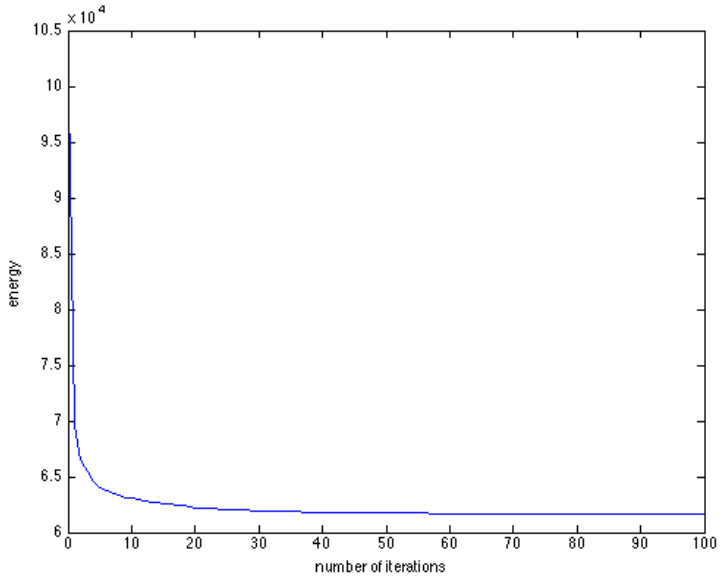


Figure 6.1. This plot shows the convergence of K-means algorithm. As the number of iterations increase energy is minimized and converged.

converges to its local maximum value.

6.4.2. Effect of the Dictionary Size

The size of the dictionary is another important issue for getting discriminative encodings. We have experimented with 4 different dictionary sizes with 3 different encodings and in each of them, accuracy increased as the dictionary size is increased. In Tables 6.1, 6.2 and 6.3, we can see the effect of dictionary sizes for all the three encoding methods BoW, VLAD, and FV methods respectively. Although each table shows us that larger dictionaries yield higher accuracies, increasing the size more and more would yield accuracy to converge and maybe decrease at some point. If we look at the increment in accuracies on these tables, the amount of increment is getting marginal as the dictionary gets larger. This observation supports the previous assumption.

Although this study does not evaluate methods based on their execution time or memory consumption, we would like to point out that, using larger dictionaries is

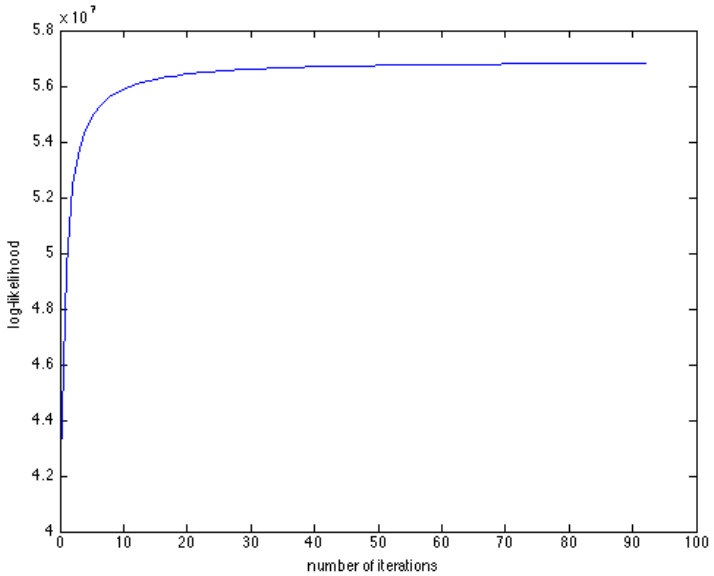


Figure 6.2. This plot shows the convergence of GMM algorithm. As the number of iterations increase log-likelihood of the data is maximized and converged.

Table 6.1. Effect of dictionary size on BoW encoding. These accuracies are obtained dense SIFT with BoW encoding on 3D-Net-Rich setup.

Dictionary size	**Accuracy (%)**
512	59.76
1024	65.06
2048	68.58
4096	70.66

Table 6.2. Effect of dictionary size on VLAD encoding. These accuracies are obtained using dense SIFT with VLAD encoding on 3D-Net-Rich database.

Dictionary size	**Accuracy (%)**
32	67.79
64	69.73
128	71.56
256	72.85

Table 6.3. Effect of dictionary size on FV encoding. These accuracies are obtained using dense SIFT with FV encoding on 3D-Net-Rich database, which also is the best performing method.

Dictionary size	**Accuracy (%)**
16	69.75
32	72.80
64	75.15
128	76.71

not practically convenient. In BoW, learning a dictionary with bigger size will take too much time. VLAD and FV uses smaller dictionary sizes, so dictionary learning stage will not be affected too much, however, they will lead to very high dimensional encoded descriptors. For dictionary with 256 words and 128 dimensional features, VLAD encoding result $128 \times 256 = 32768$ sized encoded descriptor and FV encoding result $128 \times 256 \times 2 = 65536$ sized encoded descriptor. These sizes are not convenient for both classification and memory consumption. For our evaluations, we have fixed dictionary sizes as 4096, 256, and 128 for BoW, VLAD, and FV encoding respectively.

6.4.3. Effect of PCA Dimension

From the Table 6.4 we can see that, reducing the dimension of SIFT to 96 or 64 from 128 does not incur to significant performance drops. In fact, 64 principal components yield slightly better performance than 96 components. For our evaluations, we have selected the PCA dimension as 64.

Table 6.4. Effect of PCA dimension. These accuracies are obtained using dense PCA-SIFT with FV encoding on 3D-Net-Rich database, which also is the best performing method.

Dimension	**Accuracy (%)**
32	73.56
64	76.40
96	76.03
128	76.71

6.4.4. Evaluation Pool Results

First, we have experimented all 48 method combinations on 3D-Net-Rich setup and the results can be seen in Table 6.5.

From the Table 6.5, it can be seen that method combinations that use dense key-

Table 6.5. Accuracy of different method combinations using 3D-Net-Rich setup.

Accuracy(%)			
Keypoints / Descriptors	**Encodings**		
	BoW	**VLAD**	**FV**
BRISK / SIFT	37.52	47.98	52.20
SIFT / SIFT	59.80	68.67	70.58
SURF / SIFT	42.48	51.96	54.86
Dense / SIFT	70.66	72.85	76.71
BRISK / PCA-SIFT	39.00	45.51	52.19
SIFT / PCA-SIFT	60.29	66.01	68.76
SURF / PCA-SIFT	43.95	49.37	54.61
Dense / PCA-SIFT	70.67	71.45	76.03
BRISK / BRISK	42.10	48.19	50.29
SIFT / BRISK	49.38	55.07	59.30
SURF / BRISK	50.45	57.24	59.74
Dense / BRISK	67.48	71.41	70.37
BRISK / FREAK	37.34	41.98	46.28
SIFT / FREAK	41.78	48.55	52.37
SURF / FREAK	42.06	47.49	50.10
Dense / FREAK	62.27	65.44	63.50

points outperforms others. This shows us that, every location of the image is important and extracting descriptors from the points that covers the image more densely yields better performance. The average number of keypoints detected using BRISK keypoint detector, SURF keypoint detector and SIFT keypoint detector are around 25, 40 and 50 respectively. Since their number varies from each other our evaluation is not decisive enough in the sense that if one is better than the other. However, we can say that as the keypoints get denser, encodings get stronger. If we fix the encoding method, we can see that using denser keypoints, yield higher accuracy for all description methods, except for BRISK. Although the number of the SIFT keypoints are higher than the SURF keypoints, latter yields higher accuracy when BRISK description is employed. Yet, this does not indicate SURF keypoints power over others.

If we fix the keypoints and encoding methods in Table 6.5, we can see that SIFT-like descriptors are one step forward than binary descriptors. At this point, we would like to remind that SIFT vectors are 128 dimensional and other three are 64 dimensional. In most of the keypoint-encoding combinations, SIFT descriptors yield higher accuracies and the runner up is PCA-SIFT. From this observation, we can say that the superiority of the SIFT descriptor is not because of its larger dimension as dimensionally reduced SIFT also beats binary descriptors. There are several cases where BRISK has higher accuracy than SIFT-like descriptors. If the keypoints are denser, BRISK does not have any superiority over SIFT or PCA-SIFT, but if the keypoints are sparser BRISK beats both. On the other hand, the other binary descriptor, FREAK, did not show any superiority on any of the descriptors. Other than these, unexpectedly, PCA-SIFT beats SIFT when features are encoded using BoW. It is unexpected in the sense that PCA-SIFT contains less information than SIFT. The interesting thing is that, PCA-SIFT beats SIFT only with BoW encoding. In the light of this information, we can say that BoW cannot encode features strongly enough since it uses hard assignments, but PCA-SIFT is more compact version of SIFT and it implicitly encodes some information of SIFT that BoW cannot encode, so it can beat SIFT with BoW encoding.

Except for Dense-FREAK and Dense-BRISK for all keypoint-description combi-

nations in Table 6.5, FV beats the VLAD and VLAD beats BoW. This is the expected behavior of the encoding methods, since BoW encodes features by calculating frequency of hard assignments. Thus, it loses information that can be coded using second order statistics. VLAD also uses hard assignments but not the frequency information; it encodes features by calculating distances to the assignments. It can be told that VLAD put another brick on BoW. Lastly in FV, soft assignments are used with second order statistics, which yields higher performance than VLAD and BoW. In Dense-FREAK and Dense-BRISK-VLAD, superiority over FV is most probably caused by over-fitting.

Other than keypoints, descriptors and encodings, the experimental setup also plays important role in achieved performances. The number of categories can affect performance or if there are similar categories they can also confused with each other. In Figure 6.3, the confusion matrix is illustrated. Airplane, biplane and fighter jet classes are confused with each other and since they are all aircraft this error is reasonable. Similarly armchair and chair, axe and hammer, car and convertible, heels and shoe, calculator and keyboard classes are confused, and again these errors are reasonable since they are all coming from similar categories they look like each other, an example is illustrated in Figure 6.4. Contrary to these, there also confused classes that do not look like each other, such as stapler and calculator, bowl and mug. Although these are not coming from similar categories they have some geometrical similarities. For example, stapler mostly consists of flat regions, thus, it can be confused with a flat object, or bowl has round surfaces, thus, it can be confused with objects that have round surfaces.

For experiments using PSB-Rich and 3D-Net-Rich + PSB-Rich setups, we did not evaluated all method combinations, we just selected best performing combinations. Their performance results can be seen in Tables 6.6 and 6.7.

All observations we made for 3D-Net-Rich setup are also valid for other setups. SIFT-like descriptors beat binary descriptors in each setup and FV outperforms other encoding methods. In Table 6.6 and 6.7, we can see that FV encoding yields higher performance also with BRISK and FREAK descriptors, which was not the case in

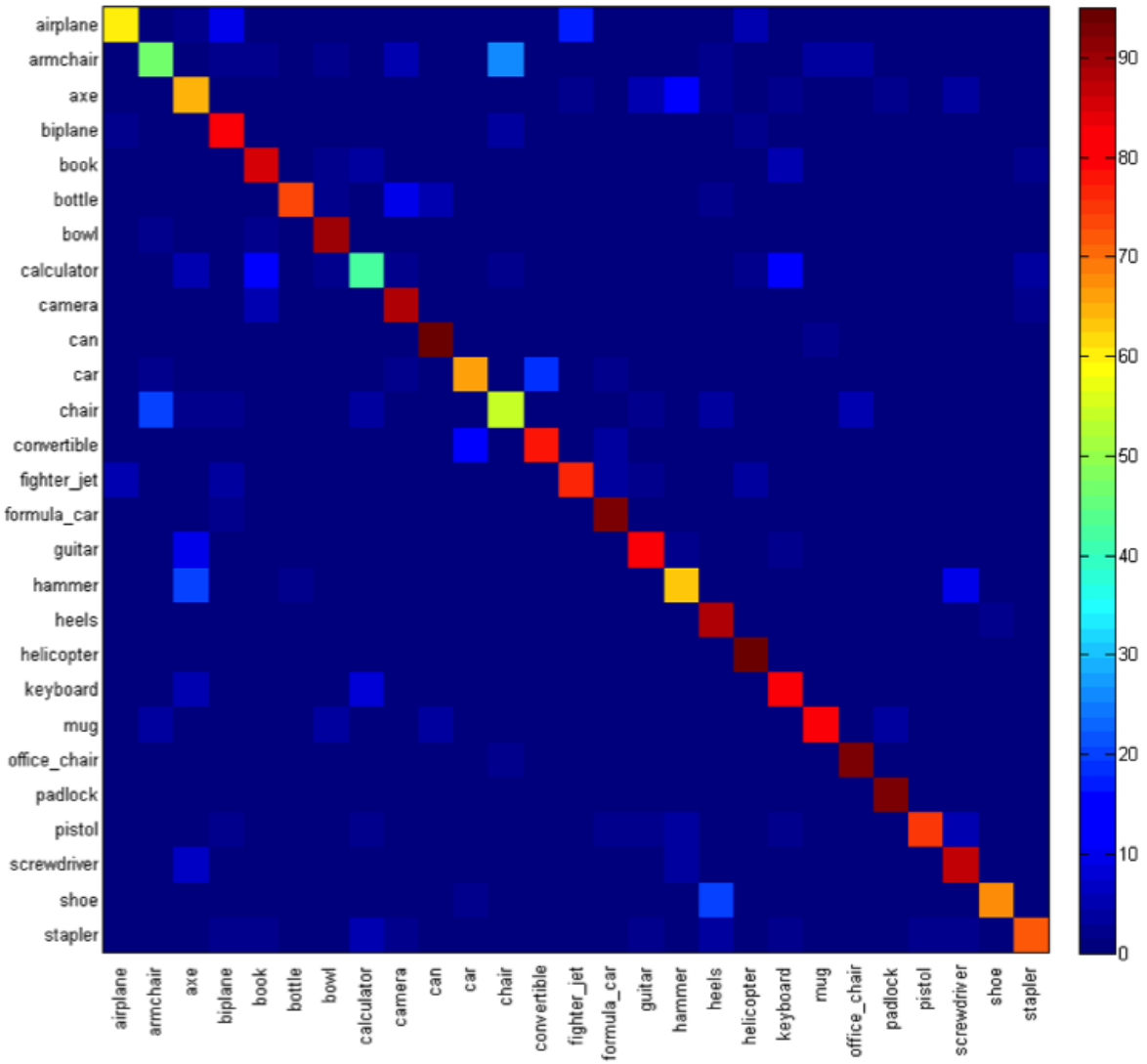
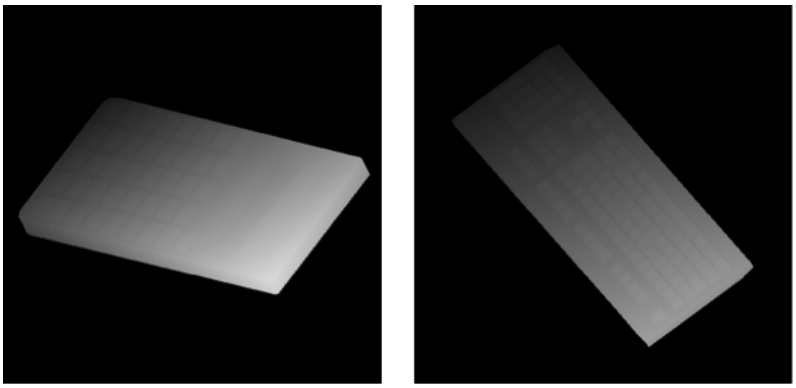


Figure 6.3. Confusion matrix of classes in 3D-Net-Rich setup with Dense-SIFT-FV. Diagonal shows the performance of each class and the others show the false recognized classes.



(a) calculator (b) keyboard

Figure 6.4. Instances of objects in “calculator” and “keyboard” classes.

Table 6.6. Accuracy of different method combinations using PSB-Rich setup.

Accuracy(%)			
Keypoints / Descriptors	**Encodings**		
	BoW	**VLAD**	**FV**
Dense / SIFT	78.74	80.73	82.88
Dense / PCA-SIFT	78.74	80.75	82.53
Dense / BRISK	76.24	76.55	76.48
Dense / FREAK	72.74	71.64	73.96

Table 6.7. Accuracy of different method combinations using 3D-Net-Rich + PSB-Rich setup.

Accuracy(%)			
Keypoints / Descriptors	**Encodings**		
	BoW	**VLAD**	**FV**
Dense / SIFT	69.45	71.58	75.59
Dense / PCA-SIFT	68.82	71.61	75.28
Dense / BRISK	64.35	66.59	67.41
Dense / FREAK	59.59	59.42	61.54

3D-Net-Rich setup. This observation supports the assumption that, there is an over fitting to data when using binary descriptors and FV encoding in 3D-Net Rich setup. One other observation that can be made is as the class count decreases performance increases or vice-versa.

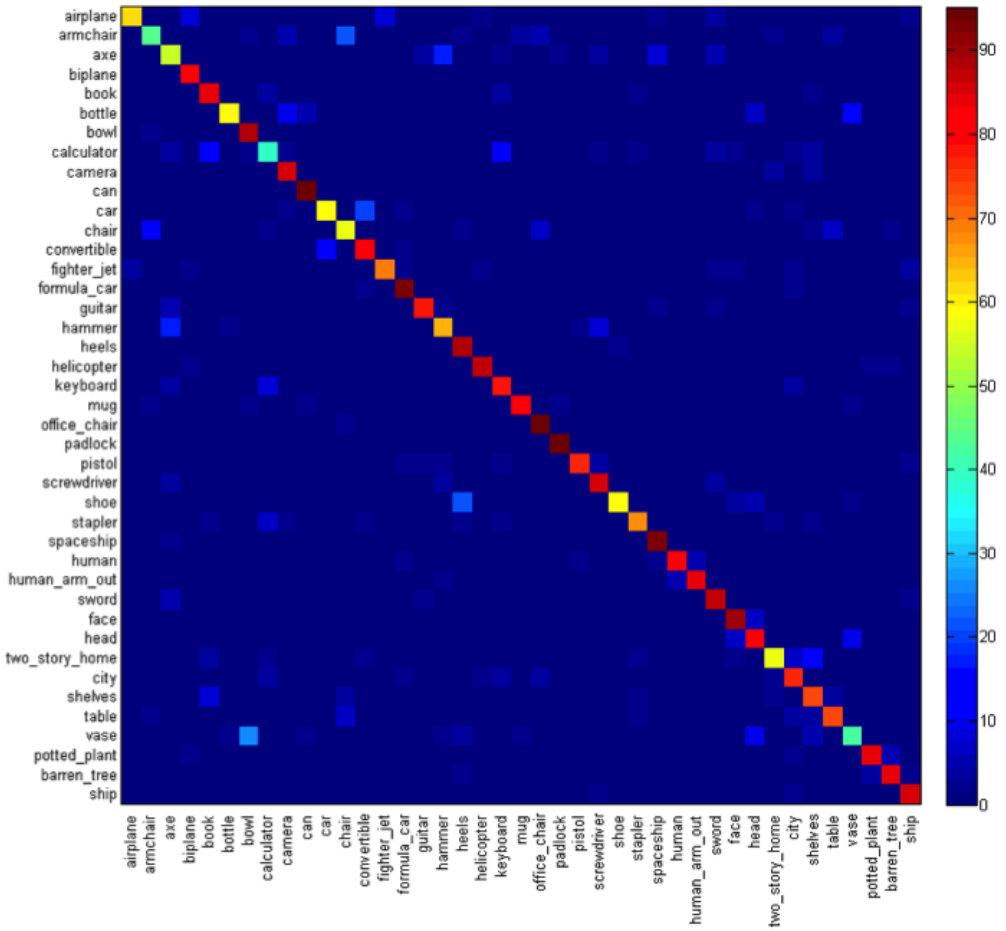
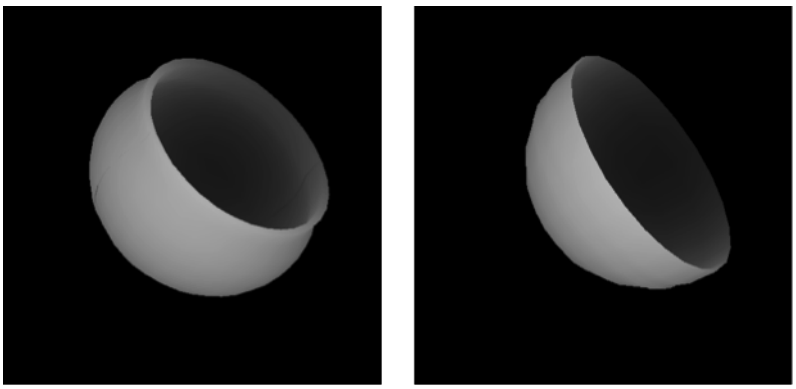


Figure 6.5. Confusion matrix of classes in 3D-Net-Rich + PSB-Rich setup with Dense-SIFT-FV. Diagonal shows the performance of each class and the others show the false recognized classes.

If we look at Figure 6.5, we can see that confused classes in 3D-Net-Rich + PSB-Rich setup are again in classes with high inter-class similarities. Figure 6.6 illustrates the most confused classes in this setup.

From observations attained from the results, we can say that, superiority of the



(a) vase (b) bowl

Figure 6.6. Instances of objects in "vase" and "bowl" classes.

methods against each other are valid in the 3D object class recognition problem with the use of depth images. We have made similar observations with 3 different setups.

6.4.5. Effect of Using Multiple Views for Classification Decision

In Section 6.2, we have stated that, classification is done using a single view. Now, we are bringing up a variation to this. In this variation, we use multiple views to make a decision. The motivation behind this idea is, if you are trying to identify an object, observing it from just a single view might not be enough, instead viewing it few more times from different perspectives possibly provide more accurate identification. For this variation one needs to fuse multiple classification results. For fusion we have employed 3 different pooling mechanisms, which we have give information in Section 5.2.

We have 50 different views of each object for testing, which means for a single decision the maximum number of views that could be used is 50. We have chosen various numbers of views and as illustrated in Figure 6.7 and 6.8, performance increases as the number of views increase, though, increment gets marginal. Moreover it is converged around at 20 views. This behavior is not valid for maximum probability pooling mechanism. In that pooling approach, if there is an incorrect decision for any of the views with high probability then the final decision becomes incorrect. This unwanted behavior occurs after around 10 views and keep decreasing the performance as the number

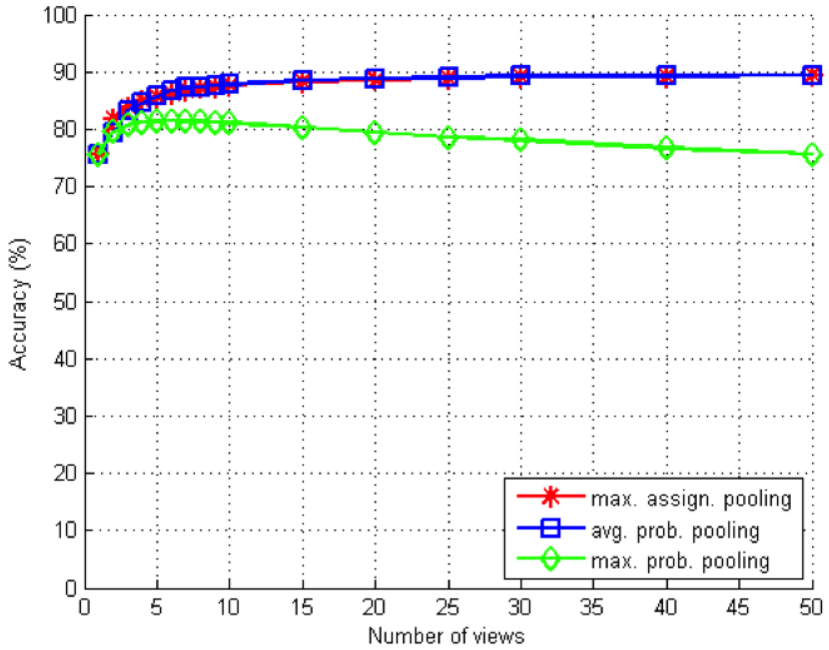


Figure 6.7. This figure shows the performance change when using multiple views for classification decision. Selected setup is 3D-Net-Rich + PSB-Rich and the method is the best performing one, Dense-SIFT-FV.

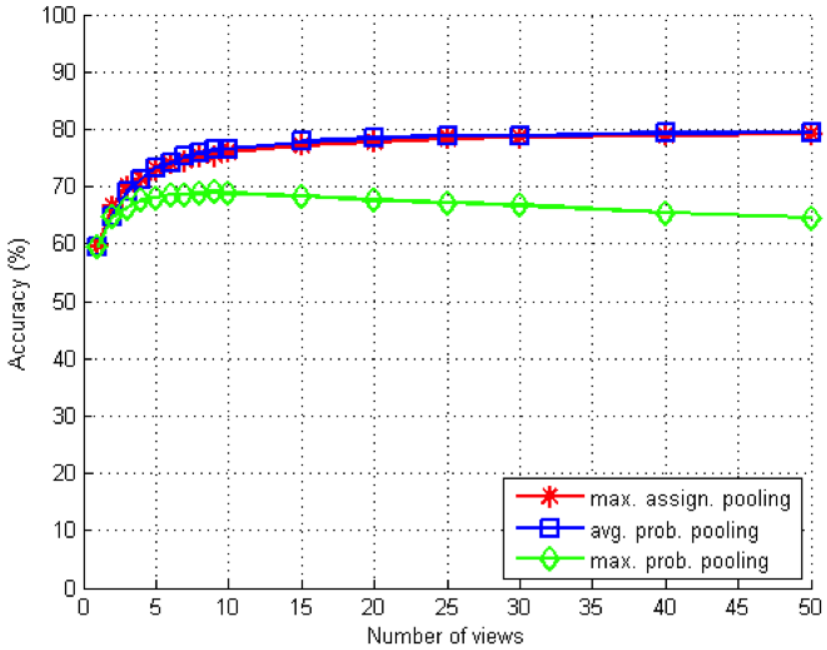


Figure 6.8. This figure shows the performance change when using multiple views for classification decision. Selected setup is 3D-Net-Rich + PSB-Rich and the method is the worst performing one, Dense-FREAK-VLAD.

of views increase. We have tested this with for both best performing (Dense-SIFT-FV) and worst performing (Dense-FREAK-VLAD) methods in the Table 6.7, and in both maximum assignment pooling and average probability pooling increases the performance, just the latter one is slightly better. If we look at the resulting confusion matrix of this variation in Figure 6.9 it is clear that that confusions in the classes are less as compared to confusion matrix of single view performances in Figure 6.5. Table 6.8 shows the performance increments of classes that have less than 60% accuracy when single view is used.

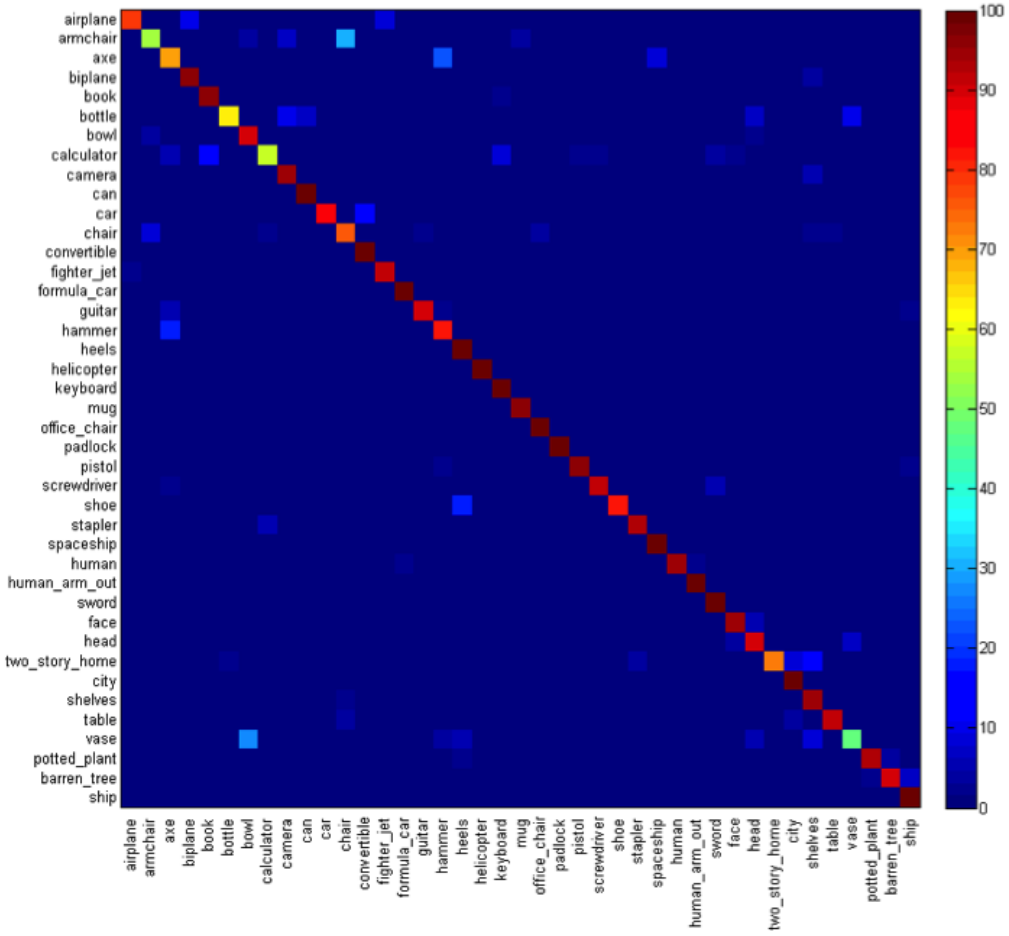


Figure 6.9. Confusion matrix of classes in 3D-Net-Rich + PSB-Rich setup with Dense-SIFT-FV encoding. Classification decision is done by using 20 views. Diagonal shows the performance of each class and the others show the false recognized classes.

Table 6.8. Performance increment achieved via multiview classification.

Class	**Single view accuracy (%)**	**20 view accuracy (%)**
armchair	44.57	53.86
axe	54.46	69.08
bottle	58.00	62.80
calculator	40.06	56.67
car	58.41	84.91
chair	56.51	74.03
shoe	58.20	80.20
two_story_home	57.40	72.60

6.4.6. Effect of Using Multiple Methods for Classification Decision

We have defined a new variation to our testing scheme in the previous section, now we are defining another variation. In this variation, we did not change the number of views but fused the classification results of multiple methods. From the results we can see that not any single method is classifying all views correctly. While a method gives incorrect result for a view another method might give a correct result.

We have 12 different method combinations (best performing ones out of 48) and we have tested the system with each of them independently. However, fusing decisions of multiple methods might lead better performances. For decision fusion of multiple methods, initially we have selected the best performing method (Dense-SIFT-FV) and added each of the remaining methods one by one to specify the method that provides highest increment. Then, we have selected the best performing method and the method with highest contribution and apply the same operation. We have repeated this until we got a convergence. Figure 6.10 illustrates the increment in performance for 3 different pooling mechanisms. As in fusing multiple views, average probability pooling is slightly better than maximum assignment pooling and maximum probability pooling does not lead any better performance after fusing 3 methods. Each pooling mechanism obtained

similar methods as highest contributors, which are shown in Table 6.9. If we look at the order of the contributors, worst performing method (Dense-FREAK-VLAD) in Table 6.7 is the fifth contributor. Although there are no remarkable performance increments, this shows that even worst performing method has advantages against other methods.

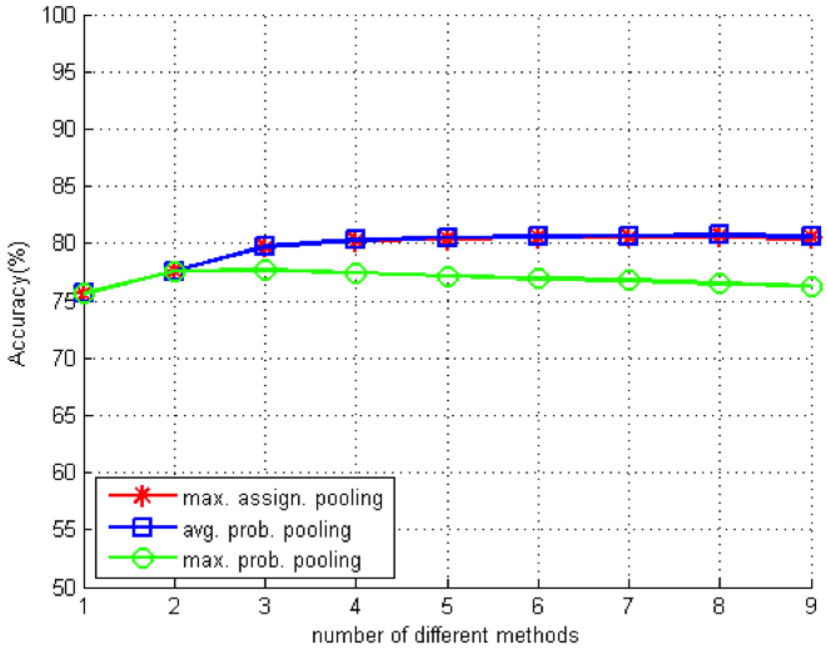


Figure 6.10. Classification performance increment with multiple methods.

Table 6.9. Highest contributing methods for each pooling mechanism. Each column is for different pooling mechanism. First row is best performing method. Next rows are highest contributing methods to previous ones.

	Max. assign. pooling accuracy (%)	**Avg. prob. pooling accuracy (%)**	**Max. prob. pooling accuracy (%)**
1	Dense-SIFT-FV	Dense-SIFT-FV	Dense-SIFT-FV
	75.59	75.59	75.59
2	Dense-PCA-SIFT-FV	Dense-PCA-SIFT-FV	Dense-PCA-SIFT-FV
	77.63	77.63	77.63
3	Dense-BRISK-FV	Dense-BRISK-FV	Dense-BRISK-FV
	79.70	79.70	77.70
4	Dense-SIFT-VLAD	Dense-SIFT-VLAD	Dense-SIFT-VLAD
	80.26	80.27	77.45
5	Dense-FREAK-VLAD	Dense-FREAK-VLAD	Dense-PCA-SIFT-BoW
	80.40	80.51	77.17
6	Dense-PCA-SIFT-VLAD	Dense-PCA-SIFT-VLAD	Dense-PCA-SIFT-VLAD
	80.49	80.62	76.96
7	Dense-BRISK-VLAD	Dense-BRISK-VLAD	Dense-BRISK-BoW
	80.58	80.69	76.77
8	Dense-SIFT-BoW	Dense-SIFT-BoW	Dense-SIFT-BoW
	80.61	80.74	76.51
9	Dense-FREAK-FV	Dense-FREAK-VLAD	Dense-BRISK-VLAD
	80.47	80.62	76.29

7. CONCLUSION & FUTURE WORK

In this thesis, we have experimentally analyzed a well-known 2D object recognition framework for 3D object class recognition from depth images of 3D object models. This framework is instantiated with four different keypoint detection methods: BRISK, SIFT and SURF keypoint detectors and dense sampling with the use of a regular grid; four different local image descriptors: SIFT, PCA-SIFT, BRISK and FREAK descriptors; and three different feature encoding methods: BoW, VLAD and FV encodings. For classification, we employed SVMs and three different pooling mechanisms: maximum assignment, average probability and maximum probability pooling. We have experimentally evaluated all of the combinations of these methods and based on our observations, we have deduced the superiorities or weaknesses of these methods with respect to each other. We obtained the highest accuracy with densely sampled SIFT descriptors encoded by FV method. We have evaluated the methods on three different setups prepared using two different 3D model databases. In each of the three setups, the highest accuracy has been achieved with the same methods. Additionally, based on our experiments, we can say that, sufficiently large dictionaries yield higher performance. We say 'sufficiently large' because we show that using oversized dictionaries would not provide substantial increase in accuracy.

We first evaluated the keypoints. Our tests showed that, using dense keypoints is more effective than trying to detect keypoints, because no matter how salient the keypoints are, in every image description - feature encoding combination dense keypoints performed better than keypoint detection methods.

Secondly, we evaluated the image descriptors. It can be conveniently said that, SIFT is the most powerful descriptor among the others. This observation is interesting because BRISK descriptor is developed years later than SIFT and it is claimed and to be competitively performing with SIFT. Similarly, the FREAK descriptor is developed later than both SIFT and BRISK, and also it is claimed to be better performing than both. In our tests, this is not observed, moreover FREAK performed worst and BRISK

could not get close to the performance of SIFT. So, we do not hesitate to say that, binary descriptors are not powerful in the depth image domain against SIFT even when SIFT is dimensionally reduced via PCA. We need to add that, if the used keypoints are sparse, BRISK beats SIFT.

The evaluation of encoding methods experimentally showed us that the information gathered from local image description methods can be revealed more and more with different encoding methods. In other words, no matter how powerful the image description method is, they need to be encoded properly to show their full power. In most cases, FV encoding is better than VLAD and VLAD is better than BoW. This is actually expected, because BoW only counts the assigned words, VLAD uses the differences between words and descriptors, and FV uses both the differences and variances of the words.

When we investigate the classes with low accuracy, we see that they are usually confused with geometrically similar object classes, which is reasonable.

After evaluating the specified methods, we go further for improving and tried two different classification variations, which uses multiple decisions for making a classification decision, where decisions are obtained from either different views or different methods. When we use multiple views we have achieved almost 15% increment in the performance, however when we fused decisions of multiple methods we only got 5% increment.

We also investigated the effect of different pooling mechanisms when combining multiple decisions. It is experimentally shown that average probability pooling is the best performing scheme. Average probability pooling enables the selection of the correct class more accurately by treating assignment probabilities with higher precedency than assignment counts.

This study can be extended in several ways:

- For depth image description, in addition to currently used methods, curvature fields and surface normal can be used.
- Different local image descriptors and feature encoding methods can be used.
- Classification can be done using SVM with RBF kernel.
- We have experimentally shown that, denser keypoints yield higher performance; to emphasize this argument different stride parameters can be used for dense sampling.
- We have experimentally shown that, different majority voting schemes in decision fusion step, affects performance differently. With application of other schemes, such as max-max pooling, its limit can be examined.
- We have worked on a clean data, since we render depth images ourselves. We can add noise to depth images when generating them, or more we can directly use real depth data with noise.
- In addition to depth information, color or texture of the objects can also be used for recognition, if available.

APPENDIX A: PREPROCESSING STEPS

A.1. Model Normalization

Although local features that are used are invariant to rotation, scale and translation effects, it is still beneficial to normalize 3D models, just in case if the methods are not robust enough against the rotation and scale. Normalization steps and their justifications are as follows:

- Place a bounding box around the 3D model by using the minimum and maximum coordinates.
- Normalize the model using longest diagonal of the bounding box.
- Place the model into a unit sphere such that the geometric center of the model and the unit sphere coincide.

First two steps satisfy the scale normalization and at the same time it guarantees that whole object lies in the rendered image. Translation normalization is guaranteed by the third step. Although the database models all lie in the same center point, the third step is applied just in case if there are any improper models. For rotation invariance, no explicit method is employed. Robustness against rotation has been satisfied with the view generation. Each object is represented by multiple depth images that are generated from different viewpoints and each depth image contains the information of the object from different rotations.

A.2. Multi-view Rendering

View generation is used as a way to provide robustness against rotation. To this end, viewpoints need to be selected such that they are uniformly distributed over the unit sphere. An icosahedron is used, which has 20 faces, to guide selection of viewpoints. 10 random points are selected in each face of the icosahedron, and then unit normalized to get final sampling points. Unit normalization is done to get viewpoint on the surface

of the sphere. This results in 200 sampling points, which shown experimentally that it is an adequate approximation for rotation invariance. This operation is repeated for each object randomly.

After the 3D viewpoints are obtained the models are orthogonally projected on the image plane. In this step, object coordinates are converted into image coordinates. Depth values of the coordinates are normalized such that maximum value of each object is 255, which corresponds to the closest point. Minimum value, which corresponds to the farthest point, changes for each view.

3D models are in triangular mesh format. This means that, 3D data has some points (vertices) and some metadata to connect points as triangles (faces). However, in the projection step, depth values of all points in the triangles are needed. They are obtained using barrycentric interpolation. Details of this method are explained next.

A.3. Barrycentic Interpolation

Barrycentic interpolation is used for obtaining the values inside a triangle with given vertices values. Let's define the barrycentric interpolation with an example. Suppose we have a triangle defined by vertices x_1, x_2, and x_3 with known values v_1, v_2 and v_3. To find the value, v_p, of point p, which is inside the defined triangle as illustrated in Figure A.1, values of vertices should be multiplied with some coefficients. In barrycentric interpolation these coefficients are obtained by the area of small triangles inside the original triangle.

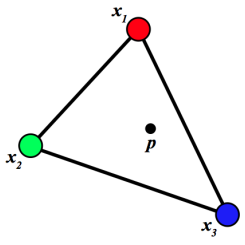


Figure A.1. Vertices of a triangle and a point inside it, such that values of x_1, x_2 and x_3 is known and value of p need to be obtained [68].

Coefficients are calculated as follows:

$$\lambda_i = A_i/A, \;\; i = \{1, 2, 3\} \tag{A.1}$$

where A_i is the area of small triangle inside the original triangle illustrated in Figure A.2, $A = \sum_i A_i$ and $\sum_i \lambda_i = 1$.

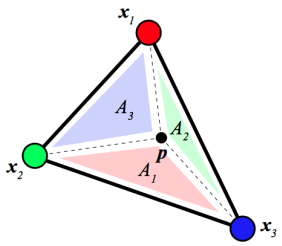


Figure A.2. Areas of small triangles inside the original triangle [68].

Once the coefficients, λ_i, are calculated, value at point p is calculated as follows:

$$v_p = \sum_i v_i \lambda_i, \;\; i = \{1, 2, 3\} \tag{A.2}$$

Different points in the triangle results different coefficients for vertices and as a point gets closer to a vertice, coefficient of that vertice increases since its corresponding small triangle area gets larger, as illustrated in Figure A.3.

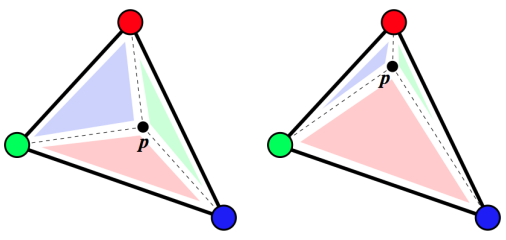


Figure A.3. Different points inside a triangle [68].

APPENDIX B: PRINCIPAL COMPONENT ANALYSIS

PCA is mathematically defined as an orthogonal linear transformation, that project the data to a new coordinate system such that the data with the greatest variance lie on the first coordinate (called the first principal component), the data with second greatest variance on the second coordinate, and so on. The goal is to transform given data matrix $\mathbf{X}_{n\times d}$ to $\mathbf{Y}_{n\times l}$, where n is the number of observations, d is the dimension of each observation vector in $\mathbf{X}$ and l is the dimension of vectors in $\mathbf{Y}$. $\mathbf{Y}$ is dimensionally reduced version of $\mathbf{X}$, thus $l < d$. To obtain $\mathbf{Y}$, we start with calculating empirical mean of $\mathbf{X}$:

$$\mathbf{u} = \frac{1}{n}\sum_{i} \mathbf{x}_i, \;\; i = 1, .., n \tag{B.1}$$

where $\mathbf{x}_i$ are the observation vectors of $\mathbf{X}$ and $\mathbf{u}$ is a d dimensional row vector. Then, for centring the data, $\mathbf{u}$ is subtracted from each observation vector:

$$\mathbf{B} = \mathbf{X} - \mathbf{hu} \tag{B.2}$$

where $\mathbf{h}$ is n dimensional column vector with each value is equal to 1 and $\mathbf{B}$ is centered version of data matrix $\mathbf{X}$. Then, we calculate the $d \times d$ covariance matrix:

$$\mathbf{C} = \frac{1}{n}\mathbf{B}'\mathbf{B} \tag{B.3}$$

The eigenvectors of the covariance matrix corresponds to principal components of the data $\mathbf{X}$, and the eigenvalues of the covariance matrix specify the order of the principal components, that is, eigenvector with the greatest eigenvalue is the first principal component, and so on. The eigenvectors, $\mathbf{V}$, diagonalizes the covariance

matrix:

$$\mathbf{V}^{-1}\mathbf{C}\mathbf{V} = \mathbf{D} \tag{B.4}$$

where diagonals of $d \times d$ matrix $\mathbf{D}$ are the eigenvalues of $\mathbf{C}$ and $\mathbf{V}$ is also $d \times d$ sized. Each column $\mathbf{V}$ is an eigenvector. The eigenvalues represent the distribution of datas energy among the eigenvectors. Side by side concatenation of the eigenvectors with the l greatest eigenvalues produces the transformation matrix, $\mathbf{T}$, which is $d \times l$ matrix. Finally, dimensionally reduced, $n \times l$ sized matrix $\mathbf{Y}$ can be obtained by:

$$\mathbf{Y} = \mathbf{X}\mathbf{T} \tag{B.5}$$

APPENDIX C: SUPPORT VECTOR MACHINES

SVM is a discriminative classifier, in formal it is a separating hyperplane. For a given data with class labels, SVM obtains the optimal hyperplane that separates the classes and that hyperplane is used to classify new samples. The term 'optimal' is important, since deciding optimal hyperplane out of several of them is a problem. This problem can be clarified better with an example. If we are given a 2D linearly separable data with n samples $\mathbf{x} = \{\mathbf{x}_i, ..., \mathbf{x}_n\}$ and with labels $\{y_i, ..., y_n\} \in \{1, -1\}$, there can be multiple lines (hyperplane is a line in 2D) that separates this data, as can be seen in Figure C.1.

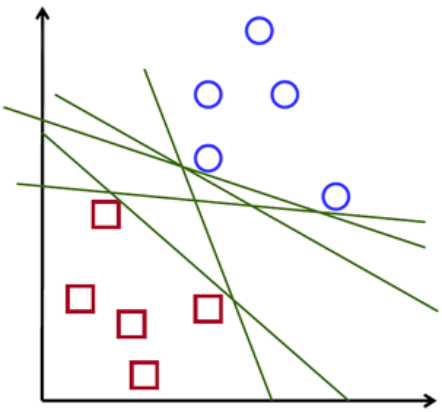


Figure C.1. Possible separating lines of 2D data [69]. There are multiple lines that separate the data. Blue circles and red squares belong to the classes *1* and *-1* respectively.

The optimal hyperplane should be as far away from the data of both classes as possible. Distances of each classes closest points to the hyperplane is called margin, m, then we can say maximizing m yields the optimal hyperplane as in Figure C.2.

Now, finding the optimal hyperplane is a maximization problem. A hyperplane is denoted as:

$$\mathbf{w}'\mathbf{x} + b = 0 \tag{C.1}$$

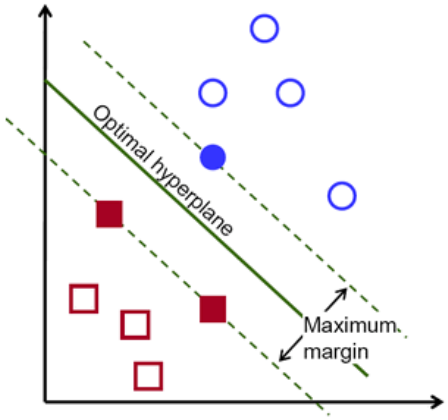


Figure C.2. Optimal hyperplane of 2D data [69]. It is obtained by maximizing the margin. Filled data points are called support vectors.

where $\mathbf{w}$ is weight vector, which is normal to the hyperplane and b is bias. $\mathbf{x}$ are support vectors of the training data, which are closest data samples to the hyperplane. Support vectors are shown in Figure C.2 as filled blue circles and red squares. The dashed lines that intersects support vectors of each class in Figure C.2 respectively denoted as:

$$\mathbf{w}'\mathbf{x}_1 + b = 1 \tag{C.2}$$

$$\mathbf{w}'\mathbf{x}_{-1} + b = -1 \tag{C.3}$$

where $\mathbf{x}_1$ and $\mathbf{x}_{-1}$ are support vectors of classes *1* and *-1* respectively. By subtracting Equation C.3 from C.2 the margin is found as:

$$m = \frac{2}{||\mathbf{w}||} \tag{C.4}$$

Maximizing m is equal to minimizing $||\mathbf{w}||$, but norm minimization is difficult to solve problem. Fortunately, it is possible to alter the equation by substituting $||\mathbf{w}||$ with $\frac{1}{2}||\mathbf{w}||^2$ ($\frac{1}{2}$ is being used for mathematical convenience).

Equations C.2 and C.3 formulates the support vectors of each class. For whole data these equations can be written as:

$$\mathbf{w}'\mathbf{x}_i + b \geq 1, \text{ for } y_i = \textit{1} \tag{C.5}$$

$$\mathbf{w}'\mathbf{x}_i + b \leq -1, \text{ for } y_i = \textit{-1} \tag{C.6}$$

which can be written in more compact way, $y_i(\mathbf{w}'\mathbf{x}_i + b) \geq 1,\ \forall i$. Then, the optimal hyperplane can be found by minimizing:

$$\frac{1}{2}||\mathbf{w}||^2 \tag{C.7}$$

subject to:

$$y_i(\mathbf{w}'\mathbf{x}_i + b) \geq 1 \tag{C.8}$$

In the case where data is not linearly separable, there will be error terms and error should also be minimized to find the optimal hyperplane. In Figure C.3, an example of linearly non-separable data is illustrated. The errors, ξ_i, of erroneous data samples are distances of these data samples to their correct place. Hence, formulation of minimization problem altered. Now, the optimal hyperplane can be found by minimizing:

$$\frac{1}{2}||\mathbf{w}||^2 + C\ \xi_i \tag{C.9}$$

subject to:

$$y_i(\mathbf{w}'\mathbf{x}_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0,\ \forall i \tag{C.10}$$

where C is trade-off parameter between margin and error. This minimization problem can be solved by Quadratic Programming [70].

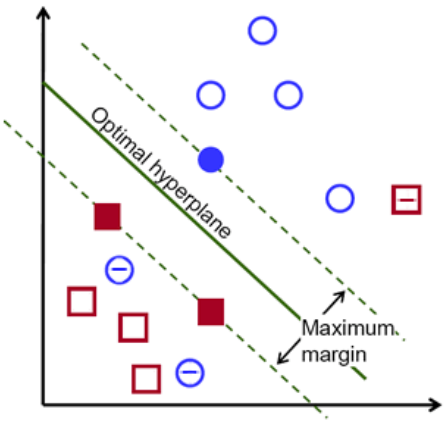


Figure C.3. Linearly non-separable data [69]. Data points that has an minus in them are erroneous points, The error, ξ_i, of an erroneous point is its distance to its correct place.

REFERENCES

1. Bimbo, A. D. and P. Pala, "Content-based Retrieval of 3D Models", *ACM Transactions on Multimedia Computing, Communications and Applications*, Vol. 2, No. 1, pp. 20–43, 2006.

2. Tangelder, J. W. and R. C. Veltkamp, "A Survey of Content Based 3D Shape Retrieval Methods", *Multimedia Tools and Applications*, Vol. 39, No. 3, pp. 441–471, 2008.

3. Ahmed, N., T. Natarajan, and K. Rao, "Discrete Cosine Transform", *Computers, IEEE Transactions on*, Vol. C-23, No. 1, pp. 90–93, 1974.

4. Lowe, D. G., "Distinctive Image Features from Scale-Invariant Keypoints", *International Journal of Computer Vision*, Vol. 60, No. 2, pp. 91–110, 2004.

5. Bay, H., A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-Up Robust Features (SURF)", *Computer Vision and Image Understanding*, Vol. 110, No. 3, pp. 346–359, 2008.

6. Leutenegger, S., M. Chli, and R. Siegwart, "BRISK: Binary Robust invariant scalable keypoints", *Computer Vision, IEEE International Conference on*, pp. 2548–2555, 2011.

7. Ke, Y. and R. Sukthankar, "PCA-SIFT: A More Distinctive Representation for Local Image Descriptors", *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, CVPR'04, pp. 506–513, IEEE Computer Society, Washington, DC, USA, 2004.

8. Mikolajczyk, K. and C. Schmid, "A Performance Evaluation of Local Descriptors", *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, Vol. 27, No. 10, pp. 1615–1630, 2005.

9. Ambai, M. and Y. Yoshida, "CARD: Compact and Real-time Descriptors", *Computer Vision, IEEE International Conference on*, pp. 97–104, 2011.

10. Alahi, A., R. Ortiz, and P. Vandergheynst, "FREAK: Fast Retina Keypoint", *Computer Vision and Pattern Recognition, IEEE Conference on*, pp. 510–517, 2012.

11. Csurka, G., C. R. Dance, L. Fan, J. Willamowski, and C. Bray, "Visual Categorization with Bags of Keypoints", *In Workshop on Statistical Learning in Computer Vision*, pp. 1–22, 2004.

12. Gemert, J. C., J.-M. Geusebroek, C. J. Veenman, and A. W. Smeulders, "Kernel Codebooks for Scene Categorization", *Proceedings of the 10th European Conference on Computer Vision: Part III*, ECCV '08, pp. 696–709, Springer-Verlag, Berlin, Heidelberg, 2008.

13. Perronnin, F. and C. Dance, "Fisher Kernels on Visual Vocabularies for Image Categorization", *Computer Vision and Pattern Recognition, IEEE Conference on*, pp. 1–8, 2007.

14. Zhou, X., K. Yu, T. Zhang, and T. S. Huang, "Image Classification Using Supervector Coding of Local Image Descriptors", *Proceedings of the 11th European Conference on Computer Vision: Part V*, ECCV '10, pp. 141–154, Springer-Verlag, Berlin, Heidelberg, 2010.

15. Jegou, H., M. Douze, C. Schmid, and P. Perez, "Aggregating Local Descriptors into a Compact Image Representation", *Computer Vision and Pattern Recognition, IEEE Conference on*, pp. 3304–3311, 2010.

16. Microsoft, "Kinect for Windows", 2010, `http://www.microsoft.com/en-us/kinectforwindows/`, [Accessed August 2014].

17. Chang, C.-C. and C.-J. Lin, "LIBSVM: A Library for Support Vector Machines",

ACM Transactions on Intelligent Systems and Technology, Vol. 2, No. 3, pp. 27:1–27:27, May 2011.

18. Qiao, Y.-L., J.-S. Pan, and S. he Sun, "Improved K Nearest Neighbor Classification Algorithm", *Circuits and Systems, IEEE Asia-Pacific Conference on*, Vol. 2, pp. 1101–1104, 2004.

19. Criminisi, A., J. Shotton, and E. Konukoglu, "Decision Forests: A Unified Framework for Classification, Regression, Density Estimation, Manifold Learning and Semi-Supervised Learning", *Foundations and Trends in Computer Graphics and Vision*, Vol. 7, No. 2-3, pp. 81–227, 2012.

20. Bishop, C. M., *Neural Networks for Pattern Recognition*, Oxford University Press, Inc., New York, NY, USA, 1995.

21. Ankerst, M., G. Kastenmüller, H.-P. Kriegel, and T. Seidl, "3D Shape Histograms for Similarity Search and Classification in Spatial Databases", *Advances in Spatial Databases*, Vol. 1651 of *Lecture Notes in Computer Science*, pp. 207–226, Springer, Berlin, Heidelberg, 1999.

22. Osada, R., T. Funkhouser, B. Chazelle, and D. Dobkin, "Matching 3D models with shape distributions", *Proceedings International Conference on Shape Modeling and Applications*, pp. 154–166, 2001.

23. Zaharia, T. and F. Preteux, "Shape-Based Retrieval of 3D Mesh Models", *IEEE International Conference on Multimedia and Expo*, Vol. 1, pp. 437–440, 2002.

24. Akgül, C. B., B. Sankur, F. Schmitt, and Y. Yemez, "Multivariate Density-Based 3D Shape Descriptors", *IEEE International Conference on Shape Modeling and Applications*, pp. 3–12, 2007.

25. Akgül, C. B., B. Sankur, Y. Yemez, and F. Schmitt, "Density-Based 3D Shape Descriptors", *EURASIP Journal on Advances in Signal Processing*, Vol. 2007, No. 1,

pp. 1–17, 2007.

26. Akgül, C. B., B. Sankur, Y. Yemez, and F. Schmitt, "3D Model Retrieval Using Probability Density-Based Shape Descriptors.", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 31, No. 6, pp. 1117–1133, 2009.

27. Wohlkinger, W. and M. Vincze, "Ensemble of Shape Functions for 3D Object Classification", *IEEE International Conference on Robotics and Biomimetics*, pp. 2987– 2992, 2011.

28. Wohlkinger, W. and M. Vincze, "Shape-Based Depth Image to 3D Model Matching and Classification with Inter-view Similarity", *IEEE RSJ International Conference on Intelligent Robots and Systems*, pp. 4865–4870, 2011.

29. Wohlkinger, W. and M. Vincze, "Shape Distributions on Voxel Surfaces for 3D Object Classification from Depth Images", *IEEE International Conference on Signal and Image Processing Applications*, pp. 115–120, 2011.

30. Wohlkinger, W., A. Aldoma, R. B. Rusu, and M. Vincze, "3DNet: Large-Scale Object Class Recognition from CAD Models", *IEEE International Conference on Robotics and Automation*, pp. 5384–5391, 2012.

31. Rusu, R. B. and S. Cousins, "3D is Here: Point Cloud Library (PCL)", *International Conference on Robotics and Automation*, Vol. 29, No. 2, pp. 1–4, 2011.

32. Aldoma, A., M. Vincze, N. Blodow, D. Gossow, S. Gedikli, R. B. Rusu, and G. Bradski, "CAD-Model Recognition and 6DOF Pose Estimation using 3D Cues", *IEEE International Conference on Computer Vision Workshops*, pp. 585–592, IEEE, 2011.

33. Tombari, F., S. Salti, and L. D. Stefano, "Unique Signatures of Histograms for Local Surface Description", *European Conference on Computer Vision*, Vol. 6313, pp. 356–369, 2010.

34. Castellani, U., M. Cristani, S. Fantoni, and V. Murino, "Sparse Points Matching by Combining 3D Mesh Saliency with Statistical Descriptors", *Computer Graphics Forum*, Vol. 27, No. 2, pp. 643–652, 2008.

35. Zaharescu, A., E. Boyer, and R. Horaud, "Keypoints and Local Descriptors of Scalar Functions on 2D Manifolds", *International Journal of Computer Vision*, Vol. 100, No. 1, pp. 78–98, 2012.

36. Chen, H. C. H. and B. B. B. Bhanu, "3D Free-Form Object Recognition in Range Images using Local Surface Patches", *International Conference on Pattern Recognition*, Vol. 3, pp. 1252–1262, 2004.

37. Zhong, Y., "Intrinsic Shape Signatures: A Shape Descriptor for 3D Object Recognition", *International Conference on Computer Vision Workshops*, pp. 689–696, 2009.

38. Mian, A., M. Bennamoun, and R. Owens, "On the Repeatability and Quality of Keypoints for Local Feature-based 3D Object Retrieval from Cluttered Scenes", *International Journal of Computer Vision*, Vol. 89, No. 2-3, pp. 348–361, 2009.

39. Chen, D.-Y., X.-P. Tian, Y.-T. Shen, and M. Ouhyoung, "On Visual Similarity Based 3D Model Retrieval", *Computer Graphics Forum*, Vol. 22, No. 3, pp. 223–232, 2003.

40. Ansary, T., M. Daoudi, and J.-P. Vandeborre, "A Bayesian 3-D Search Engine Using Adaptive Views Clustering", *Multimedia, IEEE Transactions on*, Vol. 9, No. 1, pp. 78–88, 2007.

41. Daras, P. and A. Axenopoulos, "A 3D Shape Retrieval Framework Supporting Multimodal Queries", *International Journal of Computer Vision*, Vol. 89, No. 2-3, pp. 229–247, 2010.

42. Shih, J.-L., C.-H. Lee, and J. T. Wang, "A New 3D Model Retrieval Approach

Based on the Elevation Descriptor", *Pattern Recognition*, Vol. 40, No. 1, pp. 283–295, 2007.

43. Harris, C. and M. Stephens, "A Combined Corner and Edge Detector", *Alvey Vision Conference*, pp. 147–151, 1988.

44. Mikolajczyk, K. and C. Schmid, "Indexing Based on Scale Invariant Interest Points", *Computer Vision, IEEE International Conference on*, Vol. 1, pp. 525–531 vol.1, 2001.

45. Rosten, E. and T. Drummond, "Machine Learning for High-Speed Corner Detection", *European Conference on Computer Vision*, Vol. 1, pp. 430–443, 2006.

46. Mair, E., G. D. Hager, D. Burschka, M. Suppa, and G. Hirzinger, "Adaptive and Generic Corner Detection Based on the Accelerated Segment Test", *European Conference on Computer Vision: Part II*, ECCV'10, pp. 183–196, Springer-Verlag, Berlin, Heidelberg, 2010.

47. Philbin, J., O. Chum, M. Isard, J. Sivic, and A. Zisserman, "Object Retrieval with Large Vocabularies and Fast Spatial Matching", *Computer Vision and Pattern Recognition, IEEE Conference on*, pp. 1–8, 2007.

48. Calonder, M., V. Lepetit, C. Strecha, and P. Fua, "BRIEF: Binary Robust Independent Elementary Features", *European Conference on Computer Vision: Part IV*, ECCV '10, pp. 778–792, Springer-Verlag, Berlin, Heidelberg, 2010.

49. Rublee, E., V. Rabaud, K. Konolige, and G. Bradski, "ORB: An Efficient Alternative to SIFT or SURF", *International Conference on Computer Vision*, ICCV '11, pp. 2564–2571, IEEE Computer Society, Washington, DC, USA, 2011.

50. Perronnin, F., J. Sánchez, and T. Mensink, "Improving the Fisher Kernel for Large-scale Image Classification", *European Conference on Computer Vision: Part IV*, ECCV '10, pp. 143–156, Springer-Verlag, Berlin, Heidelberg, 2010.

51. Delhumeau, J., P.-H. Gosselin, H. Jégou, and P. Pérez, "Revisiting the VLAD Image Representation", *ACM International Conference on Multimedia*, MM '13, pp. 653–656, ACM, New York, NY, USA, 2013.

52. Lowe, D., "Object Recognition From Local Scale-Invariant Features", *Computer Vision, IEEE International Conference on*, Vol. 2, pp. 1150–1157 vol.2, 1999.

53. Brown, M. and D. Lowe, "Invariant Features from Interest Point Groups", *In British Machine Vision Conference*, pp. 656–665, 2002.

54. Simard, P., L. Bottou, P. Haffner, and Y. LeCun, "Boxlets: A Fast Convolution Algorithm for Signal Processing and Neural Networks", *Advances in Neural Information Processing Systems*, pp. 571–577, MIT Press, 1999.

55. Wikipedia, "Midpoint Circle Algorithm", 2008, `http://en.wikipedia.org/wiki/Midpoint_circle_algorithm`, [Accessed August 2014].

56. CSDN, "SIFT Principals and Source Code Analysis", 2006, `http://blog.csdn.net/zd0303/article/details/8365680`, [Accessed August 2014].

57. Jolliffe, I., *Principal Component Analysis*, John Wiley & Sons, Ltd, 2005.

58. Lloyd, S., "Least Squares Quantization in PCM", *Information Theory, IEEE Transactions on*, Vol. 28, No. 2, pp. 129–137, 1982.

59. Elkan, C., "Using the Triangle Inequality to Accelerate K-Means", *International Conference on Machine Learning*, pp. 147–153, 2003.

60. Beis, J. S. and D. G. Lowe, "Shape Indexing Using Approximate Nearest-Neighbour Search in High-Dimensional Spaces", *Conference on Computer Vision and Pattern Recognition*, CVPR '97, pp. 1000–1006, IEEE Computer Society, Washington, DC, USA, 1997.

61. Dempster, A. P., N. M. Laird, and D. B. Rubin, "Maximum Likelihood from

Incomplete Data via the EM Algorithm", *Journal of the Royal Sstatistical Society, Series B*, Vol. 39, No. 1, pp. 1–38, 1977.

62. VLFeat, "The VLFeat Open Source Library", 2012, `http://www.vlfeat.org`, [Accessed August 2014].

63. Levi, G., "Bag of Words Models for Visual Categorization", 2013, `http://gilscvblog.wordpress.com/2013/08/23/bag-of-words-models-for-visual-categorization/`, [Accessed August 2014].

64. Zadrozny, B. and C. Elkan, "Transforming Classifier Scores into Accurate Multiclass Probability Estimates", *ACM International Conference on Knowledge Discovery and Data Mining*, KDD '02, pp. 694–699, ACM, New York, NY, USA, 2002.

65. Shilane, P., P. Min, M. Kazhdan, and T. Funkhouser, "The Princeton Shape Benchmark", *Shape Modeling International*, 2004.

66. Siddiqi, K., J. Zhang, D. Macrini, A. Shokoufandeh, S. Bouix, and S. Dickinson, "Retrieving Articulated 3-D Models Using Medial Surfaces", *Machine Vision Applications.*, Vol. 19, No. 4, pp. 261–275, 2008.

67. Jayanti, S., Y. Kalyanaraman, N. Iyer, and K. Ramani, "Developing an engineering shape benchmark for CAD models", *Computer-Aided Design*, Vol. 38, No. 9, pp. 939 – 953, 2006.

68. Davis, J., "Introduction to Computer Graphics", 2010, `http://classes.soe.ucsc.edu/cmps160/Fall10/resources/`, [Accessed August 2014].

69. OpenCV, "Introduction to Support Vector Machines", 2010, `http://docs.opencv.org/doc/tutorials/ml/introduction_to_svm/introduction_to_svm.html`, [Accessed August 2014].

70. Nocedal, J. and S. Wright, *Numerical Optimization*, Springer Series in Operations Research and Financial Engineering, Springer New York, 2006.