From the Institute for Neuro- and Bioinformatics
of the University of Lübeck
Director: Prof. Dr. rer. nat. Thomas Martinetz

Efficient Bio-Inspired Sensing

Dissertation
for Fulfillment of
Requirements
for the Doctoral Degree
of the University of Lübeck

from the Department of Computer Sciences

Submitted by
Irina Burciu
from Piteşti, Romania
Lübeck, January 2018

First referee: Prof. Dr.-Ing. Erhardt Barth
Second referee: Prof. Dr. rer. nat. habil. Heinz Handels

Date of oral examination: June 1st, 2018

Approved for printing. Lübeck, June 6th, 2018

Abstract

In this work we address the problem of how to efficiently sample the visual world in order to solve a particular pattern-recognition task. In this sense, we propose five novel sensing methods, namely, Visual Manifold Sensing, Foveated Manifold Sensing, Hybrid Foveated Manifold Sensing, Hierarchical Manifold Sensing with foveation and adaptive partitioning of the dataset, and Sensing Forest.

Assuming that natural images lie on nonlinear low-dimensional manifolds, these sensing methods learn manifolds of increasing but low dimensions. The sensing strategies are adaptive because sensing operations are selected depending on both the environment and the particular scene that is sensed. Inspired by the sensing strategies of biological systems, the proposed methods sample directly the features of natural images. The proposed sensing strategies are developed in the context of how one can shrink the processing resources with minimal loss, rather than develop classifiers with a high complexity trained with huge amounts of data.

The efficiency of the proposed methods is evaluated in terms of the resulting recognition rate that can be achieved with the acquired samples, i.e., sensing values. We evaluate the performance of our methods on the benchmarks: UMIST for face-, COIL and ALOI for everyday object-, and MNIST for digit-recognition. We show that only by using a simple nearest neighbor classifier, with only a few sensing values, high recognition rates are achieved. Moreover, we show that our approach is more effective than Compressive Sensing, and than the traditional Random Forest method. Furthermore, the prototypes of our Sensing Forest can be regarded as an hierarchical sensing basis, and are similar to the features learned with highly complex deep learning models.

Kurzfassung

Im Rahmen dieser Arbeit behandeln wir die Fragestellung, wie Abtastalgorithmen gestaltet werden können, um eine konkrete Aufgabe zur Mustererkennung erfolgreich und effizient maschinell bearbeiten zu können. Dazu präsentieren wir fünf neue Abtastmethoden, namentlich Visual Manifold Sensing, Foveated Manifold Sensing, Hybrid Foveated Manifold Sensing, Hierarchical Manifold Sensing mit Foveation und adaptiver Partitionierung der Daten, sowie Sensing Forest.

Unter der Annahme, dass natürliche Bilder auf nichtlinearen und niedrigdimensionalen Mannigfaltigkeiten liegen, lernen die hiermit vorgestellten Methoden Mannigfaltigkeiten mit zunehmenden aber niedrigen Dimensionen. Die Abtaststrategien agieren adaptiv derart, dass jeder Abtastschritt direkt von der aktuellen Umgebung und der konkreten Szene abhängt. Inspiriert durch biologische Abtaststrategien nehmen die hier vorgestellten Verfahren die direkt zur Erkennung notwendigen Merkmale natürlicher Bilder auf. Um beschränkte Verarbeitungsressourcen besser adressieren zu können, liegt ein weiterer Fokus dieser Arbeit auf der Vereinfachung der Abtastalgorithmen, anstatt Klassifikatoren hoher Komplexität zu entwickeln, welche mit riesigen Datenmengen vorab trainiert werden müssen.

Die Leistungsfähigkeit unserer Methoden evaluieren wir anhand der Erkennungsrate in Relation zu den notwendigen Abtastschritten in mehreren Benchmarks: UMIST für Gesichtserkennung, COIL und ALOI für Alltagsgegenstände sowie MNIST für die Erkennung von handgeschriebenen Ziffern. Wir zeigen, dass selbst mit einem einfachen Nearest-Neighbor-Klassifizierer und nur sehr wenigen Abtastwerten immer noch hohe Erkennungsraten erreicht werden können. Darüber hinaus zeigen wir, dass unser Ansatz effizienter als Compressive Sensing und der herkömmliche Ansatz von Random Forest ist. Zudem können die in unserem Sensing Forest mittels k-means Clusterbildung und Linearvektorquantisierung gelernten Prototypen als eine hierarchische Abtastbasis betrachtet werden und ähneln den Merkmalen, welche mit typischerweise sehr komplexen Deep-Learning-Modellen gelernt werden.

Contents

1 Introduction

1.1 Motivation

Our eye movements can gaze at different objects, or persons from the world around and somehow continue to acquire information in order to fulfill a task [1]. Moreover, this is done within a fraction of a second [2]. For example finding a specific object, or finding a person in a group of people, can be done instantaneously.

From a philosophical point of view, considering our daily actions, we are not only continuously sensing the world around us but also the world inside us, seen as the thoughts, perceptions, and emotional states that we have in our mind. On the other side, what we sense influences our thoughts and our actions. Thoughts are created based on how our mind characterizes what we sense. These characterizations do not simply rely on a current perception but are highly influenced by prior experiences. Following the ideas of Hermann von Helmholtz, Gregory argues that in order to build up our perceptions, we use our knowledge and the past experiences [3]. For example children analyse a new item for a long time. Adults only need a blink, because they already know the item. Thus, if we have more experience, e.g., we have seen that object or a similar one before, then we will be able to characterize more accurately what we are sensing, and so it will be easier to find the object that we have already seen, or find a similar object. The authors of the ”Active Vision” book [4] argue that the eye samples what is interesting in a scene but what is interesting can change at any moment, influenced and guided by the thoughts and action plans of the observer. Even Gibson, who believed in the passive vision idea (what you see is what you get), appreciated, in a far-sighted way, that a major function of vision is to direct action: *What causes the eyes to move in one direction rather than another, and to stop at one part of the array instead of another? The answer can only be that interesting structures in the array, and interesting bits of structure, particularly motions, draw the foveas towards them* [5].

Closely related to those theories of perception are the function and characteristics of the two visual streams of the visual system: the Gibsonian dorsal stream or the ”What” pathway (concerned with perception for action with almost no use of stored information) and the ventral stream or the ”Where” pathway (concerned with perception for recognition and uses stored information for processing fine details) [6].

Human sensing involves eye movements and attention [7]. For example when we look at the face of a person, we are scanning that person’s face and we are immediately able to recognize that person. We are doing this so many times every day, but do we ever wonder how are we actually capable of doing this? Looking only at the scan paths drawn by the eye movements of different persons while watching the face of other people, we can see that our attention is concentrated mostly on some areas of the shown faces, e.g., the eyes, the nose, and the mouth. Examples of scan paths corresponding to several observers are shown in Figure 1.1. Consequently, instead of perceiving and processing the whole image, we need only some information in order to recognize a person. The sensing strategy of biological systems can be shortly described as extracting features during perception, rather than sampling the whole scene and afterwards extracting features.

Given the limited information processing resources [8], an agent must use efficient sensing strategies in order to solve, for example, a recognition task. Our work is inspired by the concepts of Active Vision [4] and *the world as an outside memory* [9]: the outside world is considered as a sort of external memory store which can be accessed instantaneously by directing someone’s attention to some location. The outside memory can always be accessed for solving a particular problem, i.e., there is no need to store images of the world in the brain.

This thesis is written in the context of the project ”Learning efficient Sensing for Active Vision (Esensing)” which aims at developing efficient sensing strategies for agents which are acting in the real world. The sensing optimization together with the representation of the world is done by using and extending methods from the field of Compressive Sensing (CS) [10, 11] and Sparse Coding [12]. Therefore, CS can be more efficient by using hierarchical sensing schemes, and the sensing strategies based on machine learning algorithms can be optimized for particular datasets, e.g., objects, and for a specific kind of task, e.g., for a specific kind of object. Accordingly, the agents will independently adapt their sensing strategies and the world representation to a particular environment and to a particular task.

1.2 Contributions

In this work we address the problem of how to efficiently sample (note that we use sampling as a synonym of sensing) the visual world for a particular pattern-recognition task. Traditional sampling or sensing approaches assess the performance of the sampling strategy based on the quality of the reconstructed signal. Here, the performance criterion is the recognition rate that can be achieved

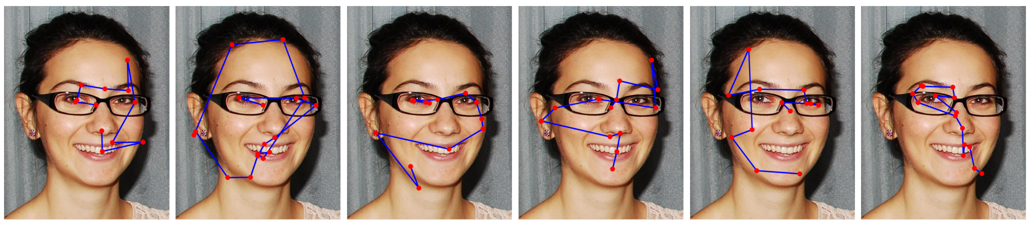


Figure 1.1: Scan paths (in blue) drawn by the eye movements of different persons while looking at a picture (with the author's face). The fixations (in red) show the areas of the face on which the attention is focused. Note that in average, the attention is concentrated on the eyes, the nose and the mouth.

with the samples. The proposed strategy is to compress the input to a classifier while sensing such as to maximize both compression and recognition rates. Given the current trend towards high-complexity classifiers trained with huge amounts of data, it is reasonable to also consider, as an opposite trend, how one can shrink the processing resources with minimal loss. Part of this strategy is not to sample a whole image and then extract features but to sample the features directly, i.e., to sample only the information needed for classification.

We present in this thesis five novel sensing methods, which involve an hierarchical partitioning of the dataset, opposed to the classical sensing methods. This implies that the world is sensed differently depending on the task at hand. The methods presented in this thesis illustrate the benefits of efficient sensing strategies for Active Vision. Very few sensing actions are sufficient to solve a particular recognition task and the interaction of a sensing agent with the environment can proceed from first obtaining a gist of the scene, based on very few sensing values, and then acquiring more refined samples until the task at hand can be solved. The sensing schemes are adaptive because sensing operations are not conducted in a random fashion but are selected depending on both the environment and the particular scene that is sensed.

Our approach is inspired by Compressive Sensing (CS) [13] in the sense that each acquired sensing value is a weighted sum over the whole visual scene (in classical pixel-wise sensing, the sensing matrix has entries equal to one on the diagonal and is zero elsewhere). Moreover, our approach extends CS by introducing a two-fold adaptivity: (i) the sensing algorithm adapts to a particular dataset, and (ii) every new sensing value depends on the previously acquired sensing values. Therefore, in the methods that we here describe, the sensing is done adaptively and not randomly as with CS.

We propose five methods for sampling natural images, namely, Visual Manifold Sensing (VMS) [14], Foveated Manifold Sensing (FMS) [15], Hybrid Foveated Manifold Sensing (HyFMS) [15], Hierarchical Manifold Sensing (HMS) with foveation and adaptive partitioning of the dataset [16, 17], and Sensing Forest (SF) [18]. These are based on learning manifolds of increasing but low dimensions assuming that natural images lie on nonlinear low-dimensional manifolds. FMS is an extension of VMS and is inspired by the sampling strategy of biological systems. Accordingly, FMS includes a foveation part and senses the most salient areas of an object. The hybrid version of FMS, HyFMS, which is introduced as an even more efficient extension of FMS, is again inspired by biological vision. HyFMS acquires a global gist [19] of a scene and then proceeds to a more refined sampling. In case of foveation, the sensing matrix is also sparse in addition to being learned, i.e., adapted to the specific dataset. HMS includes an adequate partitioning of the dataset which is learned before the actual sensing. This is achieved by combining two approaches: Principal Component Analysis (PCA) [20] for dimensionality reduction and k-means clustering [21, 22]. The last proposed method, called Sensing Forest (SF), follows the ideas of HMS but uses k-means for learning the low-dimensional representation of the data [23]. SF is a prototype-based Random Forest (RF) [24, 25] with prototypes learned with k-means clustering, and refined by using Learning Vector Quantization (LVQ) [26]. An overview of the five proposed methods is shown in Table 1.1.

	Bio-inspired sensing methods				
	Online learning			Offline learning	
	VMS	FMS	HyFMS	HMS	SF
learns low-dimensional hierarchical representation (algorithm used)	✓ (LLE)	✓ (LLE)	✓ (LLE)	✓ (PCA & k-means)	✓ (k-means & LVQ)
adapts to particular datasets and to particular tasks	✓	✓	✓	✓	✓
adapts dataset based on acquired sensing values	✓	✓	✓	✓	✓
learns adaptive, hierarchical sensing basis for a particular task	✓	✓	✓	✓	✓
prototype-based Random Forest	✗	✗	✗	✗	✓
foveation	✗	✓	✓	✓	✗
sparse sensing matrix	✗	✓	✓	✓	✗
gist-like sensing	✓	✗	✓	✗	✓

Table 1.1: Overview of our proposed bio-inspired sensing methods: Visual Manifold Sensing (VMS), Foveated Manifold Sensing (FMS), Hybrid Foveated Manifold Sensing (HyFMS), Hierarchical Manifold Sensing (HMS) with foveation and adaptive partitioning of the dataset, and Sensing Forest (SF). The features of the five methods are shown on the left side of the table. While sensing a novel scene using VMS, FMS, or HyFMS, the dataset is continuously adapted and the corresponding embedding is learned such that online learning is performed. The corresponding embeddings are learned using Locally Linear Embedding (LLE). On the other side, HMS and SF are offline learning sensing methods. Thus, they learn an adequate hierarchical partitioning of the dataset before the actual sensing. HMS learns a tree corresponding to the dataset by using Principal Component Analysis (PCA) and k-means, and SF learns a forest consisting of several trees by using k-means and Learning Vector Quantization (LVQ).

1.3 Outline

In the following we describe the structure of this work. Chapter 2 presents the core methods of this work. First, the Efficient Coding field is described and representative methods are presented: Compressive Sensing (CS), which developed based on the Sparse Coding concept, the gist concept of a scene which comes as an alternative view to the traditional view on visual processing, and the geometric invariants of the structure tensor which can be used for determining key-points in a scene. Further on, we introduce the manifold concept and we describe how manifolds are related to visual perception. In this sense, we present Locally Linear Embedding (LLE), a manifold learning algorithm, and we compare it to other manifold learning algorithms. Another relevant method for this work is the traditional Random Forest (RF) method for learning decision trees. Lastly, Chapter 2 describes Learning Vector Quantization (LVQ), a prototype-based classification approach.

Chapter 3 presents the five methods that we proposed for efficient sensing. Visual Manifold Sensing (VMS) is the first method that we propose and it is followed by Foveated Manifold Sensing (FMS). We describe in detail how FMS involves foveation by using a foveated dataset and we present the FMS algorithm. Afterwards, HyFMS, the extended version of FMS, is presented. Further on, Hierarchical Manifold Sensing (HMS) with foveation and adaptive partitioning of the dataset is introduced, and we explain how the hierarchical partitioning of the dataset and how the hierarchical sensing of unknown scenes are done. The last method that we propose is the Sensing Forest (SF) and we present the corresponding algorithms for learning the Sensing Forest and for sensing novel scenes.

Chapter 4 presents the results we obtained for the proposed methods. We describe for each method the experiments that were done in order to measure the performance of the methods. We also show how the learned sensing basis functions adapt while sensing and we compare the performance of our methods to the performance of other state-of-the-art methods.

In the last chapter the conclusions of this work are presented, together with future development ideas for the efficient bio-inspired sensing approach.

2 Methods

In this chapter we introduce the core methods used in this work. The Efficient Coding field is first described and afterwards representative methods are presented: Compressive Sensing (CS), the gist concept, and the structure tensor with the corresponding geometric invariants. Further on, the manifold learning approach is introduced and a manifold learning algorithm is presented, i.e., Locally Linear Embedding (LLE). The Random Forest (RF) approach based on learning decision trees is also described. Finally, the Learning Vector Quantization (LVQ) method for learning prototypes is introduced.

2.1 Efficient Coding

The efficient coding hypothesis was introduced in 1961 by Horace Barlow [27] as a theoretical model, which gives insights about the neural representations of the sensory coding activity in the brain. The theory itself started with ideas based on the insight that statistics of sensory stimuli received from the outside world are essential for perception and cognition [28, 29]. Barlow pointed out that the role of visual coding was to remove the statistical redundancy from the visual representation. The efficient coding hypothesis states that the visual input is encoded as efficient as possible, i.e., with minimal neural activity. There are also several works, which sustain the idea that neurons behave like efficient information channels [30]. Accordingly, it was shown that there are cells which minimize the area to which their output overlaps with the output of their neighbors, e.g., retinal ganglion cells [31], and simple cells in the striate cortex [32].

Later on, the efficient coding hypothesis continued to develop such that it led to the field of Sparse Coding introduced by Olshausen and Field [12]. Sparse Coding is based on the fact that natural images can be sparsely encoded [33, 34]. Field argues that natural images can be efficiently encoded using wavelet transforms [35] because their response histograms show high kurtosis, i.e., they are sparse [33]. In this context, the goal of efficient coding is to find a set of basis functions which span the image space. Such a representation leads to statistically independent coefficient values which permits the extraction of intrinsic structure in sensory signals [33].

Recently, the field of Compressive Sensing (CS) developed based on the Sparse Coding concept [10, 11]. It was shown that if the signal can be sparsely encoded, then by using an appropriate random sensing matrix, one can accurately represent the signal by acquiring a reduced number of samples [10]. Thus, not all the information about the world around is necessary in order to describe it, only a clever subsampling is needed. This idea brought many interesting applications to the field of CS, from the Rice single pixel camera [36], to seismic applications [37], and Magnetic Resonance Imaging [38].

In the following we will describe the sensing problem in the CS field in more detail. In the context of efficient coding we will also introduce the gist concept which is a recent view on how an observer performs scene recognition, i.e., from global to local, rather than the traditional view, i.e., from local to global. Furthermore, we present a geometrical method used for measuring the local information content in an image based on the structure tensor and the intrinsic dimension (iD) of the image. The invariants of the structure tensor are known to be good predictors of human eye movements for static scenes [39], and they are used to model image saliency [40].

2.1.1 Compressive Sensing

Compressive Sensing (CS) has attracted attention in several scientific fields. CS predicts that high-dimensional signals, which allow a sparse representation by a suitable basis, can be recovered from relatively few measurements. It also states that less measurements are required than expected from the well-known Shannon/Nyquist sampling theorem [41]. Shannon's theorem states that the sampling rate must be at least twice the highest signal frequency in the signal. Rather than sampling first at a high rate and afterwards compressing the sampled data, CS comes with solutions for directly sensing the data in a compressed form, at a lower sampling rate [42]. In the following paragraphs we will define the sensing problem and give some exemplary applications of CS.

The sensing problem

Given a data vector $x \in \mathbb{R}^D$ (depending on the context, a digital image or signal) we assume the existence of a basis (such as the wavelet basis) $\Psi \in \mathbb{R}^{D\times D}$ where the coefficient vector $x_s \in \mathbb{R}^D$ is a sparse vector [35, 43], i.e., it has many entries equal to zero, and represents the data vector x in the basis Ψ. As we already mentioned at the beginning of this chapter, natural images are known to be sparse [33, 34]. Natural images contain specific statistical regularities that set them apart from random images, and they are related to the response properties of neurons at early stages of the visual system [12]. There are also theoretical studies which suggest that the primary visual cortex

(V1) uses a sparse code to efficiently represent natural scenes [44].

In order to compress the original data vector x, CS uses a random projection matrix $R \in \mathbb{R}^{d \times D}$ such that: $y = Rx \in \mathbb{R}^d$, with $d << D$. The number of sparse components in x_s results in a small d or leads to a small or a zero error $||x - \Psi x_s(Rx)||^2$. That means, if x is sparse or compressible, then x is well approximated by x_s and the error between them is small. Thus, one can basically discard a large number of coefficients without a big loss. The rows of R can be seen as measurements y_i, $i = 1, \ldots, d$ obtained by correlating the original data vector x with the sensing waveforms r_i:

$$y_i = \langle x, r_i \rangle,\ i = 1, \ldots, d. \tag{2.1.1}$$

In order to recover the original signal by using only the collected data from the observed subset, the l_1-norm minimization is used [13]. Under the sparsity assumption, one can reconstruct an approximation to x from y by solving the following l_1-norm minimization problem:

$$\min_{R\Psi^{-1}f=y} ||f||_{l_1},\ \Psi x = f,\ ||f||_{l_1} = \sum_i |f_i|. \tag{2.1.2}$$

There are several algorithms for l_1 minimization, as well as other methods for recovering sparse solutions [45].

Figure 2.1 shows an example of CS subsampling and image reconstruction. The pixel values in the original image in Figure 2.1 (a) range from 0 to 255. The wavelet coefficients in Figure 2.1 (b) show that only few of the wavelet transform coefficients are needed to represent the image. In Figure 2.1 the reconstruction obtained by keeping the 25.000 largest values from the wavelet expansion is shown [46, 13]. This example shows that the perceptual loss is hardly noticeable between the original image and its approximation obtained by discarding 97.5% of the coefficients.

The relation between sparsity and sampling is very strong: CS is based on a nonlinear sampling theorem which shows that a signal x with D samples and only K nonzero components can be exactly recovered from $d \geq cK \log(D/K)$ measurements, where c is a small constant. Thus, the number of measurements required for the exact reconstruction is much smaller than the number of signal samples and this is possible due to the sparsity level of x [13].

In the following we present some of the applications of CS that highlight the importance and the novelty of this topic nowadays.

Applications

CS is used in a mobile phone camera sensor [47]. A custom CMOS chip with a 256×256 pixel image sensor was designed to show the importance of CS. The chip contains associated electronics

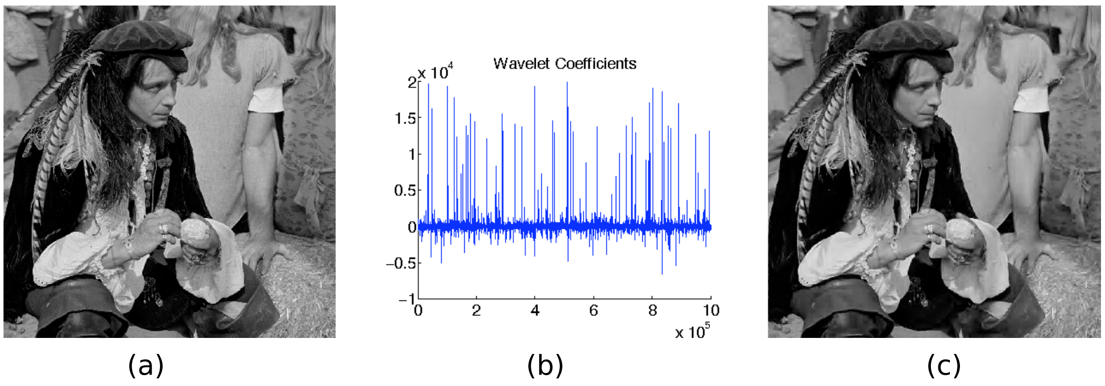


Figure 2.1: CS subsampling and image reconstruction [46, 13]. (a) Original image. (b) Wavelet transform coefficients of the original image. The image is highly compressible, i.e., only a few coefficients are needed for capturing the energy of the signal. (c) The reconstruction of the image obtained by discarding 97.5% of the image coefficients. The difference between the original image and the reconstructed image is nearly indistinguishable.

that can sum random combinations of analog pixel values. The digital output of this chip is already in compressed form. The approach allows a reduction in image acquisition energy per image with a factor of 15 but with considerable costs for the complex decompression algorithms. The chip is the first one being capable to capture many different random combinations simultaneously, with the disadvantage that it needs to take multiple images for each compressed frame.

CS is used in the single-pixel cameras from Rice University [36]. The single-pixel digital camera directly acquires d random linear measurements without taking first all the D pixels. Figure 2.2 shows how the single-pixel camera works [48]: the incident light field corresponding to the desired image is reflected off a digital micromirror device (DMD) which consists of an array of D small mirrors. Afterwards, the reflected light is collected by a second lens and focused onto a single photodiode (the single pixel). Each mirror can be independently oriented towards the photodiode (this corresponds to a 1) or away from the photodiode (this corresponds to a 0). In order to take measurements, a random number generator (RNG) sets the orientation of the mirror in a pseudorandom $1/0$ pattern to create the vector measurement. Then, the voltage at the photodiode equals y_j (the inner product between the measurement vector and the desired image). The process is repeated for d times to obtain all the entries of y.

CS is used to shorten Magnetic Resonance Imaging (MRI) scanning sessions. MRI is an essential medical imaging tool with an inherently slow data acquisition process. Applying CS to MRI offers potentially significant scan time reductions, with benefits for patients and health care economics [49]. MRI obeys the two most important requirements for successful application of CS: (i) medical imagery is naturally compressible by sparse coding in an appropriate transform domain, e.g., by wavelet transform, (ii) MRI scanners naturally acquire encoded samples, rather than direct

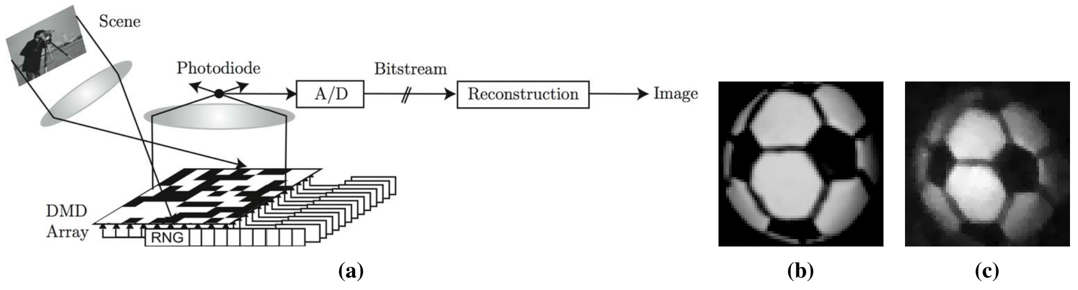


Figure 2.2: (a) Single-pixel camera. The digital micromirror device (DMD) consists of an array of D small mirrors. The grid of micromirrors reflects some part of the incoming light beam toward the sensor, i.e., a single photodiode, and other parts of the image, i.e., the black squares, are sent away. Each measurement done by the photodiode is a random combination of many pixels. (b) Conventional digital camera image. (c) 64×64 image recovered from $d = 1600$ random measurements taken by the single-pixel camera in (a). Note that 1600 random measurements are enough to create an image which is comparable to a 4096-pixel camera as shown in (b) [48].

pixel samples, e.g., in spatial-frequency encoding.

CS is also applied for face recognition [50]. Thus, the underlying sparsity in the problem is exploited in order to improve the robustness, speed, and accuracy with which classification is performed.

Furthermore, CS is utilized to reduce the hardware complexity of scalable robotic tactile skins and the transmission of the data with a fast and accurate reconstruction of the full resolution signal [51]. Such a tactile skin, i.e., a tactile system which covers areas of the body of a robot, brings improvements for the awareness and manipulation abilities of a robot. In the approach presented in [51], the tactile data is first compressed and afterwards sent to the processing unit. Subsequently, the tactile data is reconstructed from the compressed data when it is needed. Moreover, the measured compressed signals are further used for tactile object classification. The results show that the system is capable to compress the signal to a forth of the original dimension. Additionally, it produces a qualitative reconstruction of the signal and it achieves high classification rates, similar to the classification on the original sensor signals.

2.1.2 Gist and hierarchical processing

The gist of a scene refers to the fast capability of an observer to understand and acquire a multitude of perceptual and semantic information of a real world scene [19]. Therefore, an observer performs first a fast, global recognition of a scene, known as gist, and afterwards continues with refined sampling. For example, people can recognize if a scene is a street, a kitchen, a mountain, or while rapidly changing the television channels they can grasp the meaning of the channel (news, movie, sports, etc.). The gist of a scene contains low level features (color), intermediate image characteristics (surface, volume), and goes as far as to high level information (objects) [52]. This idea comes in contrast to the traditional view on scene recognition, which is seen as proceeding from local to global, i.e., starting with local measurements (edges, surfaces) which are afterwards integrated into a complex percept [53].

2.1.3 Structure Tensor

We consider a gray scale image modeled by a function $I : \mathbb{R}^2 \to \mathbb{R}$. For an open region Ω, for all $(x, y) \in \Omega$: (i) $I(x, y) =$ constant, or (ii) $I(x, y) = g(ax + by)$, $\forall g,\ a,\ b$, or (iii) I varies along all the directions. It was shown in [54] that I is considered to locally have intrinsic dimension (iD) 0 (i0D), 1 (i1D) or 2 (i2D). The iD refers to the relation between the degrees of freedom of a signal domain and the actual degrees of freedom used by a signal. Thus, the signals with i0D are constant within a local window, signals with i1D can be approximated by a function of only one variable (e.g., straight lines, edges) and signals of i2D are for example corners, curved lines, junctions and curved edges. The iD is important for image coding because i0D and i1D regions are predominant in natural images [34]. The properties of the image regions selected by the saccadic eye movements during experiments were analyzed in terms of higher order statistics [39]. It was shown that image regions with a statistically less redundant structure, as the ones given by the signals with i2D, contain all the necessary information of a static scene. Therefore, signals with i2D are considered to be more salient. Later on, it was also shown that natural images are fully determined by the i2D regions such that curved image regions contain all the information in an image [55].

The mathematical representation of the concept was presented in [56] as follows. Given a region Ω, for a linear subspace $E \subset \mathbb{R}^2$:

$$I(x + v) = I(x) \text{ for all } x,\ v \text{ such that } x,\ x + v \in \Omega,\ v \in E. \tag{2.1.3}$$

The iD of I is then $2 - dim(E)$, and $n - dim(E)$ in the case of n-dimensional signals. The iD can be estimated using different methods but we will here present a differential method, i.e., the structure tensor [57]. The subspace E is estimated as the subspace spanned by the set of unity vectors which minimize the energy functional:

$$\mathcal{E}(v) = \int_\Omega \left|\frac{\partial I}{\partial v}^2\right| d\Omega = v^T \mathbf{J} v, \tag{2.1.4}$$

given the equivalence in Ω of Equation 2.1.3 and the constraint:

$$\frac{\partial I}{\partial v} = 0, \text{for all } v \in E. \tag{2.1.5}$$

J is given by:

$$\mathbf{J} = \int_{\Omega} \nabla I \otimes \nabla I \mathrm{d}\Omega = \int_{\Omega} \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \mathrm{d}\Omega = w * \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}, \tag{2.1.6}$$

where ∇I is the gradient of image I, $\otimes$ denotes the tensor product, and I_x, I_y are notations for the partial derivatives, i.e., $\frac{\partial I}{\partial x}$, and $\frac{\partial I}{\partial y}$, respectively for the second order derivatives, I_x^2, I_y^2, and $I_x I_y$. The integral over Ω can be implemented as smoothing with a Gaussian function, $*$ denotes convolution with the kernel w. The derivatives are computed by first convolving the image with a Gaussian kernel and then with the derivative kernels $(-1,\ 0,\ 1)$ in the different directions.

E is the eigenspace associated with the smallest eigenvalues of **J**, and the iD of I is given by the rank of **J** which can be obtained from the eigenvalues of **J**. It has been shown that instead of the eigenvalue analysis one can estimate v by using the minors of **J** [58], thus by computing the symmetric invariants of **J**, i.e., K and S. For 2-dimensional functions, in our case image-intensity function $I(x, y)$, the invariants are given by:

$$H = \text{trace}(\mathbf{J}) = I_x^2 + I_y^2 \tag{2.1.7}$$

$$K = \det(\mathbf{J}) = I_x^2 \cdot I_y^2 - (I_x I_y)^2. \tag{2.1.8}$$

The geometric invariant K is different from zero only when the image intensity varies in all directions, therefore K is used for key-point detection.
The structure tensor **J** can also be interpreted as the auto-correlation between local patches which are shifted relative to each other as shown in [59]. Given two images, I_0 and I_1, the weighted summed square difference between two patches of the two images, is given by:

$$E_{WSSD}(\boldsymbol{u}) = \sum_i w(x_i)[I_1(x_i + \boldsymbol{u}) - I_0(x_i)]^2, \tag{2.1.9}$$

where $\boldsymbol{u} = (u, v)$ is the displacement vector, $w(x)$ is a spatially varying weighting function or window, and the sum is done over all the pixels in the patch. For feature detection it is not known which patches or image regions the feature will be matched against. This is why it can only be studied how stable the chosen metric is with respect to small variations in position $\Delta\boldsymbol{u}$. Thus, image patches are compared against itself. This is known as auto-correlation function or surface:

$$E_{AC}(\boldsymbol{u}) = \sum_i w(x_i)[I_0(x_i + \Delta\boldsymbol{u}) - I_0(x_i)]^2 \tag{2.1.10}$$

The auto-correlation surface can be approximated by using a Taylor Series expansion of the image

$I_0(x_i + \Delta\boldsymbol{u}) \approx I_0(x_i) + \nabla I_0(x_i) \cdot \Delta\boldsymbol{u}$ as follows:

$$\begin{aligned} E_{AC}(\boldsymbol{u}) &= \sum_i w(x_i)[I_0(x_i + \Delta\boldsymbol{u}) - I_0(x_i)]^2 && (2.1.11) \\ &\approx \sum_i w(x_i)[I_0(x_i) + \nabla I_0(x_i) \cdot \Delta\boldsymbol{u} - I_0(x_i)]^2 && (2.1.12) \\ &= \sum_i w(x_i)[I_0(x_i) \cdot \Delta\boldsymbol{u}]^2 && (2.1.13) \\ &= \Delta\boldsymbol{u}^T \mathbf{J} \Delta\boldsymbol{u}, && (2.1.14) \end{aligned}$$

where ∇I_0 is the image gradient at x_i:

$$\nabla I_0(x_i) = \left(\frac{\partial I_0}{\partial x}, \frac{\partial I_0}{\partial y} \right)(x_i) \tag{2.1.15}$$

The gradient can be computed by convolving the image with horizontal and vertical derivatives of a Gaussian. The auto-correlation matrix **J** can be seen as a tensor image as in Equation 2.1.6, where the outer products of the gradients are convolved with a weighting function w for a pixel estimation of the local shape of the auto-correlation function.

2.2 Manifold learning

One of the many interesting challenges of scientists who are trying to build vision machines that achieve a comparable visual object recognition performance as humans, is how our brain is capable to constantly perceive the world when the world is in a continuous flux [60]. In this context, we will introduce in this chapter the manifold concept. In the following we will describe how manifolds are related to visual perception and how they are used in order to reduce the dimensionality of data. Moreover, we will present a manifold learning algorithm, Locally Linear Embedding (LLE).

2.2.1 An introduction to manifolds

Bernhard Riemann presented in his habilitation lecture “Über die Hypothesen welche der Geometrie zu Grunde liegen” for the first time, his ideas on differential geometry concentrated on the concept of a differentiable manifold (Mannigfaltigkeit) [61]. There, Riemann extended Gauss’ theory of the intrinsic geometry of surfaces to higher-dimensions. The aspects of the geometry of surfaces are intrinsic in the sense that they depend only on the chosen metric, opposed to the geometrical aspects which depend on the way in which the surface is embedded in space.

Mathematically, a manifold is a topological space that is locally Euclidean, i.e., around every point, there is a neighborhood that is topologically the same as the open unit ball in $\mathbb{R}^n$ [62]. A manifold is differentiable if there is a unique tangent space at each point on it. The manifold is Riemannian if one can perform the inner product in the tangent space [63, 64]. For example, we consider the ancient belief that the Earth was flat as contrasted with the modern evidence that it is round. The difference comes essentially from the fact that on the small scales that we see, the Earth does indeed look flat. Locally, at each point on the surface of the Earth, there is a 3D coordinate system (two coordinates for location and one coordinate for the altitude). Globally, it is a 2D sphere in a 3D space. In general, any object that is locally flat is a manifold and thus can be locally treated as an Euclidean space.

In order to characterize the variability of images and of other perceptual stimuli, a mathematical approach is considered [65]. In this sense, an image can be regarded as a collection of numbers, each specifying light intensity at an image pixel. Moreover, an image can be identified with a point in an abstract image space [60]. For example, a set of images $\mathcal{M}$ with differently oriented faces are considered. As the faces are rotated, the images vary smoothly and thus the set defines a continuous curve in the image space. Moreover, the set is a curve generated by the variation of the angle of rotation, i.e., a single degree of freedom. Therefore, $\mathcal{M}$ is intrinsically one dimensional, even though $\mathcal{M}$ is embedded in the image space with a higher dimension given by the number of pixels in an image. $\mathcal{M}$ is considered to be a manifold embedded in the image space [60]. In order to recognize faces, the brain has to assimilate all images from the same manifold, but distinguish between images from different manifolds. An example is shown in Figure 2.3: as the faces rotate, they trace out nonlinear curves embedded in the image space. The manifolds sketch from the paper of Seung and Lee [60] was adapted for images with two different persons from the UMIST database [66].

For the moment it is not known how the brain is capable of representing image manifolds. One hypothesis states that they are stored in the brain as manifolds of stable neural-activity patterns [67]. Population activity is typically described by a collection of neural firing rates and so it can be represented by a point in an abstract space with the dimensionality equal to the number of neurons. It was found that the firing rate of each neuron in a population can be written as a smooth function of a small number of variables, such as the angular position of the eye or the direction of the head. This led to the idea that the population activity is constrained to lie on a low-dimensional manifold [60]. Because the possible images of an object lie on a manifold, it has been hypothesized that a visual memory is stored as a manifold of stable states, or a continuous attractor [67], [68].

Manifolds offer a powerful framework for dimensionality reduction. The key idea of reducing the dimensionality of the data is to find the most concise low-dimensional representation that is embedded in a higher dimensional space [62]. The dimensionality of a manifold is defined by the number of coordinate axis used in the local Euclidean approximations. There are several manifold learning algorithms which identify overlapping patches of the given dataset, which can be locally

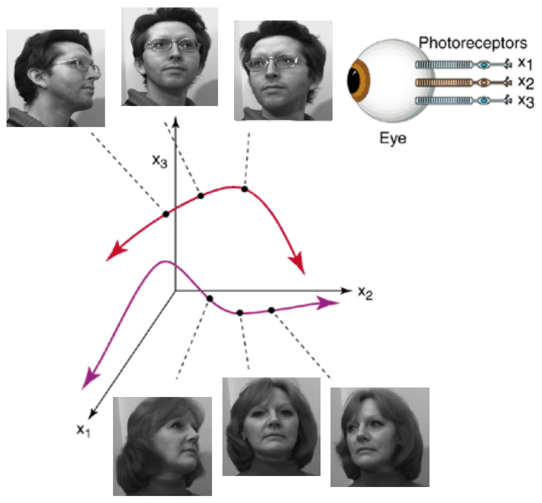


Figure 2.3: Manifolds in visual perception with images from the UMIST database [66]. The retinal image is a collection of signals received from the photoreceptor cells. If these numbers are considered as coordinates in an abstract image space, then an image is represented by a point. Only three dimensions of the image space are used but actually the dimensionality is equal to the number of the photoreceptor cells. In case there are, for example, changes in scale, illumination, etc., then the images would lie on low-dimensional manifolds, instead of on the shown one-dimensional curves [60].

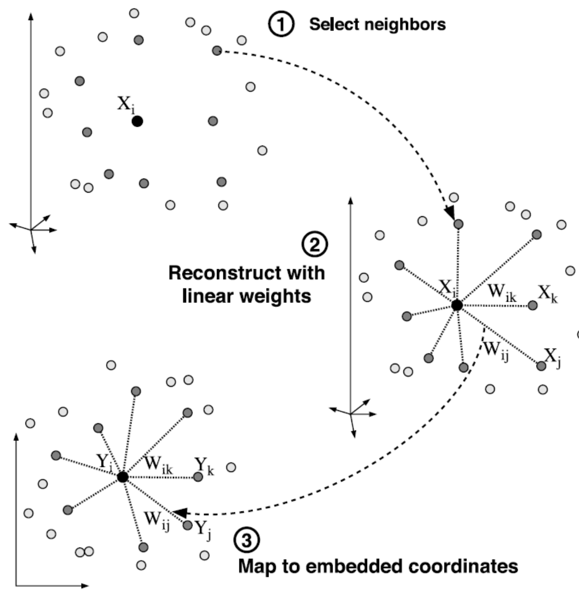
described by Euclidean models. In the following we present one of the manifold learning algorithms, Locally Linear Embedding (LLE).

2.2.2 Locally Linear Embedding

Locally Linear Embedding (LLE) was introduced by Roweis and Saul [65] as a manifold learning algorithm. As the name suggests, LLE is based on the idea that a manifold can be approximated locally by a linear structure. LLE is an unsupervised algorithm that computes low-dimensional, neighborhood-preserving embedding of high dimensional inputs.

By exploiting the local symmetries of linear reconstructions, LLE is able to learn the global structure of nonlinear manifolds, such as those generated by images with objects, faces or documents of text. The coherent structure in the world leads to strong correlations between inputs (like the neighboring pixels in images), generating observations that lie on or close to a low-dimensional manifold.

We suppose the data consists of p real-valued vectors $\vec{x}_i$, each of dimensionality D, sampled from an underlying manifold. In the following we present the steps of LLE, which are also graphically sketched in Figure 2.4: (1) selecting neighbors for each data point $\vec{x}_i$, (2) computing the weights w_{ij} that best linearly reconstruct $\vec{x}_i$ from its neighbors, and (3) computing the low-dimensional representation $\mathbf{Y}$ with the vectors $\vec{y}_i$ by using the weights w_{ij} from the previous step.



1 Select neighbors
X_i
2 Reconstruct with linear weights
X_i
W_ik
X_k
W_ij
X_j
Y_i
W_ik
Y_k
W_ij
Y_j
3 Map to embedded coordinates

Figure 2.4: Schematic illustration of the LLE algorithm as shown in [65]. (1) Select neighbors for each data point $\vec{x}_i$. (2) Each data point $\vec{x}_i$ in the input space is reconstructed from its neighbors $\vec{x}_k$ with a set of weights w_{ik}. (3) The same weights w_{ik} are used to reconstruct the projected data point $\vec{y}_i$ from the projected neighbors $\vec{y}_k$.

Considering that there is sufficient data in order to obtain a well-sampled manifold, we expect that each data point and its neighbors lie on or close to a locally linear patch of the manifold. The local geometry of these patches is described by linear coefficients which reconstruct each data point from the neighbors of the respective data point. The reconstruction errors are measured by the cost function:

$$E(\mathbf{W}) = \sum_i \left(\vec{x}_i - \sum_j w_{ij}\vec{x}_j\right)^2, \qquad (2.2.1)$$

which sums up the squared distances between all the data points and their reconstructions. The weights w_{ij} summarize the contribution of the j-th data point to the i-th reconstruction.

In order to compute the weights w_{ij} we minimize the cost function subject to two constraints: each data point $\vec{x}_i$ is reconstructed only from its neighbors such that $w_{ij} = 0$ if $\vec{x}_j$ does not belong to the set of neighbors of $\vec{x}_i$, and the rows of the weight matrix sum to 1, $\sum_j w_{ij} = 1$. For any particular data point, the weights are invariant to translation, rotation, and scale of data point and local neighborhood. By symmetry, the reconstruction weights characterize intrinsic geometric properties of each neighborhood. The invariance to translations is specifically enforced by the sum to one constraint on the rows of the weight matrix. The optimal weights w_{ij} subject to these constraints are found by solving a least-square problem [65, 69]. We assume that the data lies on or close to a smooth manifold of dimension $d << D$. Then there exists a linear mapping (translation, rotation, scaling) that maps high dimensional coordinates of neighborhood to global internal coordinates on the manifold. This means that weights are still valid in a new representation.

Afterwards, each high-dimensional data point $\vec{x}_i$ is mapped to a low-dimensional vector $\vec{y}_i$ which represents the global internal coordinates on the manifold. Thus, d ($d < D$) coordinates from $\vec{y}_i$ are found by minimizing the cost function:

$$\Phi(\mathbf{Y}) = \sum_i \left(\vec{y}_i - \sum_j w_{ij}\vec{y}_j\right)^2 \tag{2.2.2}$$

Compared to the previous cost function from Equation (2.2.1), the cost function in Equation (2.2.2) has fixed weights w_{ij} while optimizing the coordinates $\vec{y}_i$. $\Phi(\mathbf{Y})$ can be minimized by solving a sparse $p \times p$ eigenvalue problem [65, 69]. The embedding cost defines in the last cost function, a quadratic form:

$$\Phi(\mathbf{Y}) = \sum_{ij} m_{ij}(\vec{y}_i \cdot \vec{y}_j) = \mathbf{Y}^T\mathbf{M}\mathbf{Y}, \tag{2.2.3}$$

which involves inner products of the embedding vectors and the $p \times p$ matrix $\mathbf{M}$ with elements m_{ij} given by:

$$m_{ij} = \delta_{ij} - w_{ij} - w_{ji} + \sum_k w_{ki}w_{kj}, \tag{2.2.4}$$

where δ_{ij} is 1 if $i = j$ and 0 otherwise. Thus, $\mathbf{M}$ can be stored and used as the sparse symmetric matrix [69]:

$$\mathbf{M} = (\mathbf{I} - \mathbf{W})^T(\mathbf{I} - \mathbf{W}). \tag{2.2.5}$$

The cost function $\Phi(\mathbf{Y})$ is minimized subject to two constraints in order to make the problem well-posed. First, the coordinates $\vec{y}_i$ can be translated by a constant displacement without affecting the cost function $\Phi(\mathbf{Y})$. Thus, this degree of freedom is removed by centering the coordinates on the origin:

$$\sum_i \vec{y}_i = \vec{0}. \tag{2.2.6}$$

Secondly, in order to avoid degenerate solutions ($\vec{y}_i = \vec{0}$), the embedding vectors are constrained to have unit covariance:

$$\frac{1}{p}\sum_i \vec{y}_i\vec{y}_i^T = \mathbf{I}, \tag{2.2.7}$$

where **I** is the $d \times d$ identity matrix. The embedding is encoded in the eigenvectors which correspond to the d largest eigenvalues of matrix **M** in Equation 2.2.5 [65].

It was shown for LLE that it is equivalent to kernel PCA [70] with the kernel:

$$\lambda_{max}\mathbf{I} - \mathbf{M}, \tag{2.2.8}$$

where λ_{max} is the largest eigenvalue of **M**. LLE is thus a kernel PCA with the kernel learned for a particular dataset [71].

The pseudocode of the LLE algorithm is presented in Algorithm 1 as described in [65]. The data points are reconstructed from their K nearest neighbors, as measured by the Euclidean distance. The algorithm involves a single pass through the three steps in Figure 2.4 such that the global minimum of the reconstruction and embedding costs from the Equations (2.2.1) and (2.2.2) are found.

The *LLE* function takes as input the original data **X** which consists of $\vec{x}_i$ vectors, $i = 1, \ldots, p$ of dimension D, the number of nearest neighbors K considered for each vector $\vec{x}_i$, and the dimension $d < D$ of the manifold. First, the neighbors are chosen as shown in Algorithm 2, then the optimal weights of the matrix **W** and the embedded coordinates matrix **Y** are computed respectively in Algorithm 3, and 4. The corresponding notations are shown in Table 2.1.

X	$D \times p$ matrix consisting of p input data vectors in D dimensions
Y	$d \times p$ matrix consisting of $d < D$ dimensional embedding coordinates for **X**
x_i	i-th column of **X**, i.e., the i-th data point of **X**
y_i	the i-th column of **Y**, i.e., the embedding coordinates of the i-th point in **Y**
K	number of nearest neighbors considered for each x_i
W	weight matrix which best linearly reconstructs **X** from its neighbors
d	dimension of the manifold, $d < D$
I	the identity matrix
1	a column vector of all ones
Z	all the columns of **X** corresponding to the neighbors of x_i but not x_i itself

Table 2.1: Notations used for the LLE Algorithm 1, and for the corresponding functions presented in Algorithms 2, 3, and 4.

Algorithm 1 LLE manifold learning algorithm

```
1: function LLE($\mathbf{X}$, $K$, $d$)
2:     $neigh(\mathbf{X}) \leftarrow$ LLE-findNeighbors($\mathbf{X}$, $K$)
3:     $\mathbf{W} \leftarrow$ LLE-recWeights($neigh(\mathbf{X})$)
4:     $\mathbf{Y} \leftarrow$ LLE-embedding($\mathbf{W}$, $d$)
5: return $\mathbf{Y}$
6: end function
```

Algorithm 2 LLE - find neighbors in **X** space

```
1: function LLE-findNeighbors($\mathbf{X}$, $K$)
2:     for $i = 1 : p$ do
3:         compute distance from $x_i$ to every other point $x_j$
4:         find the $K$ smallest distances
5:         $neigh(x_i) \leftarrow$ the corresponding points as neighbors of $x_i$
6:     end for
7: return $neigh(\mathbf{X})$
8: end function
```

Algorithm 3 LLE - solve for reconstruction weights **W**

```
1: function LLE-recWeights($neigh(\mathbf{X})$)
2:     for $i = 1 : p$ do
3:         $Z \leftarrow neigh(x_i)$
4:         substract $x_i$ from every column of $Z$
5:         $C = Z^T Z$
6:         solve linear system $C \cdot w = 1$ for $w$
7:         if $j$ is not a neighbor of $i$ then
8:             $w_{ij} = 0$
9:         end if
10:        $\mathbf{W} \leftarrow w/$ sum$(w)$
11:    end for
12: return $\mathbf{W}$
13: end function
```

Algorithm 4 LLE - compute embedding coordinates **Y** using weights **W**

```
1: function LLE-embedding($\mathbf{W}$, $d$)
2:     $\mathbf{M} \leftarrow (\mathbf{I} - \mathbf{W})^T(\mathbf{I} - \mathbf{W})$
3:     $\mathbf{Y} \leftarrow$ bottom $d$ eigenvectors of $\mathbf{M}$
4: return $\mathbf{Y}$
5: end function
```

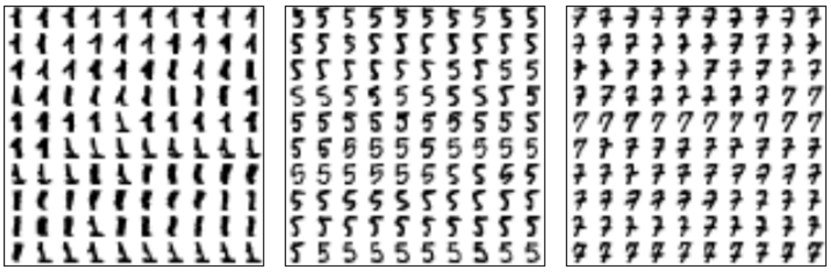
Compared to other methods (e.g., Multidimensional scaling (MDS) [72], Isomap [73]), LLE does not need to estimate pairwise distances between widely separated data points. Instead, LLE recovers the global nonlinear structure from locally linear fits. In comparison to Isomap [73] which uses a traversal graph, LLE puts the small linear neighborhoods together by finding a set of weights that perform local linear interpolations which are able of approximating the data [69]. The advantage of LLE is that even though it requires solving an $p \times p$ eigenvalue problem just like Isomap does, the matrix for LLE has a higher degree of sparsity (because many weights are zero) in comparison with the matrix for Isomap, which has zero values only on the diagonal. PCA [20] and MDS map faraway data points to nearby points in the plane and they are not able to discover the underlying structure of the manifold. In Figure 2.5 different embeddings are shown for a dataset which consists of selected digits from MNIST [74] corresponding to digits $1,\ 5$ and 7. For each digit around 180 images were selected. The results were obtained by using the *sklearn.ensemble.randomTreesEmbedding* class implemented in Python [75].

LLE was applied to various domains, e.g., it was used on images of lips in order to find meaningful attributes, such as the pose and facial expression for a set of human face images [76]. It was also used in vision as an application for the construction of two dimensional parameterizations of different digit shapes [76]. The LLE algorithm was generalized to use other metric distances besides Euclidean and it was applied on the MNIST handwritten digit dataset [74]. LLE was also used in other domains than computer vision, for example it was applied to a sample of 500 galaxy spectra and it was found that the position of a galaxy within the embedding space is directly correlated with the mean age of the galaxy and its spectral type [77].

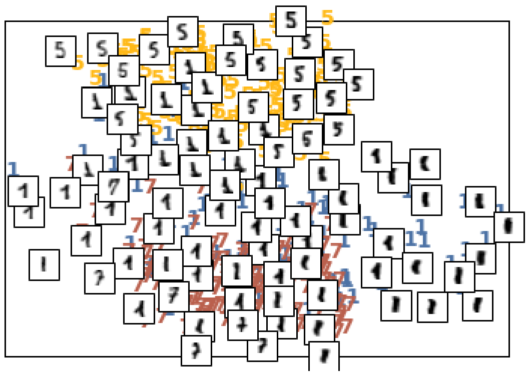
2.3 Random Forest

The Random Forest (RF) approach is based on learning decision trees for either classification or regression. In 1984, in the book of Breiman et al. "Classification and Regression trees" [78], the basics of decision trees and their use for classification and regression were described. By then, the decision trees were used individually and it was later on shown that boosting algorithms with iterative weighting of the training data, can actually linearly combine "weak" learners into a "strong" classifier with higher accuracy and generalization [79]. Following this idea, a random decision forest is an ensemble of randomly trained decision trees [80]. For example, for digit recognition, the tree training was done by using random feature selection and different split functions [81]. Afterwards, it was shown that random decision trees generalize better than boosting algorithms on digit and shape recognition data [82, 83]. Further on, Breiman came with the idea to bring randomness in the forest by randomly choosing a specific amount from the original training dataset [24, 25]. The RF, consisting of a specific number of trees, chooses in the end the class with the most votes.

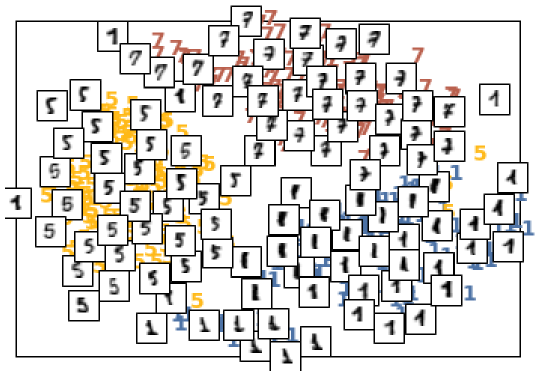
RFs are now widely used because they perform better or equivalent to other techniques [84]. They are also known for the success of the Microsoft Kinect for XBox 360 [85] which is capable to



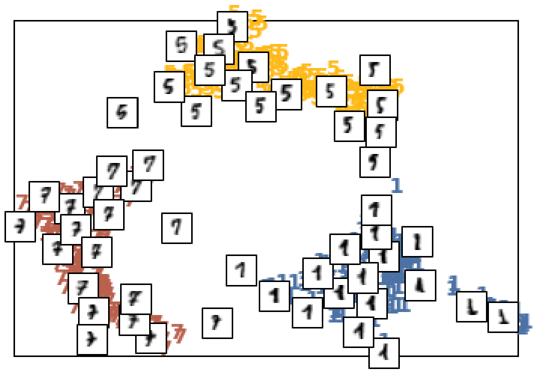
(a) Selected 64-dimensional digits (digits used: 1, 5, and 7) from MNIST dataset [74]



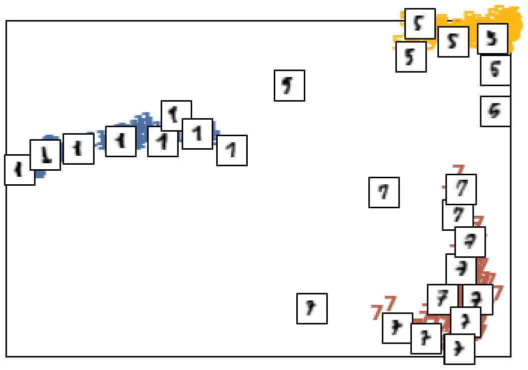
(b) Projection of the digits using PCA



(c) Projection of the digits using MDS



(d) Projection of the digits using Isomap ($K = 16$)



(e) Projection of the digits using LLE ($K = 16$)

Figure 2.5: (a) Dataset consists of selected digits (digits used: 1, 5, and 7) from MNIST [74]. Embeddings obtained with (b) PCA, (c) MDS, (d) Isomap, and (e) LLE are shown. Note that with PCA the 3 digits are mixed and it is therefore not possible to separate them. Digits 1 and 7 are quite similar to each other and are easily confused while using PCA, and are better separated but are still too close to each other while using MDS. With MDS, the distances in the two-dimensional representation preserve well the distances in the original 64-dimensional space. By applying Isomap the 3 digits are better separated in comparison to PCA and MDS. By using LLE local clusters are formed. When compared to PCA, MDS or Isomap, LLE shows a higher degree of separability of the three digits, even for the digits which are easily confused, 1 and 7.

accurately predict the 3D positions of body joints from a single depth image, without using temporal information [86, 87].

A forest contains a specific number of trees, $nrMaxTrees$: $\{T_1, T_2, \ldots, T_{nrMaxTrees}\}$ which are independently trained. This can be efficiently done in parallel. After training, each new data point x_{test} is sent through the learned trees. In the end, the corresponding prediction of x_{test}, i.e., the class label output, is given by a single forest prediction, rather than single trees prediction. In the following a more detailed explanation of the process will be presented.

2.3.1 Training the Random Forest

The training step is done offline and it is meant to optimize the parameters corresponding to the split function of each node of the tree. Randomness is used only during the training phase such that the computed trees are different from each other. Therefore, the generalization is improved. As mentioned earlier, two of the most frequently used ways of inducing randomness for the trees in the RF are: (i) random training dataset sampling, e.g., bagging [25], and (ii) node optimization with randomly chosen features [81].

The split functions are weak learners. At each node j of a tree there is a binary split function:

$$h(x, \theta_j) \in \{0, 1\}, \tag{2.3.1}$$

where 0 represents false and 1 true. Depending on the split function, a data point x is sent to the left or to the right branch of the respective tree rooting in j. The split function $\theta = (\Phi, \Psi, \tau)$ has several parameters: Φ is the filter function which selects only some features from x, Ψ gives the geometric primitive which is used for separating the data, and τ consists of the binary tests thresholds. For example, the most common weak learner is the axis-aligned hyperplane [88], but also general-oriented hyperplane is used [80]. In order to obtain the optimal parameters θ_j^* of the split node j, the information gain is maximized:

$$\theta_j^* = \operatorname*{argmax}_{\theta_j} IG_j, \tag{2.3.2}$$

where $IG_j = IG(D^j, D_L^j, D_R^j, \theta_j)$ depends on the training data points from before the split, D^j, and after the split: D_L^j, D_R^j, and on the split function, θ_j. The information gain obtained by splitting the data is given by:

$$IG_j = H(D^j) - \sum_{i \in \{L,R\}} \frac{|D_i^j|}{|D^j|} H(D_i^j), \tag{2.3.3}$$

where i indexes the left and right child of the node j, and $H(D) = -\sum_{c \in \mathcal{C}} p(c)log(p(c))$ is the Shannon entropy.

We consider the example shown in Figure 2.6. For a given training dataset $\mathcal{D}$ with the corresponding labels (in this case there are 3 classes: red, green and blue), a subset D_{11} of data points is randomly chosen from $\mathcal{D}$. D_{21}^{11} represents the subset of training data points which reach the left child of the root, D_{11}, respectively the right child, D_{22}^{11}. At each node j the objective function θ_j is optimized, i.e., the split which maximizes the information gain IG_j is chosen. The terminal nodes (leaves), contain the corresponding classes probability distributions. This can be seen in the considered example in Figure 2.6 for trees with two levels.

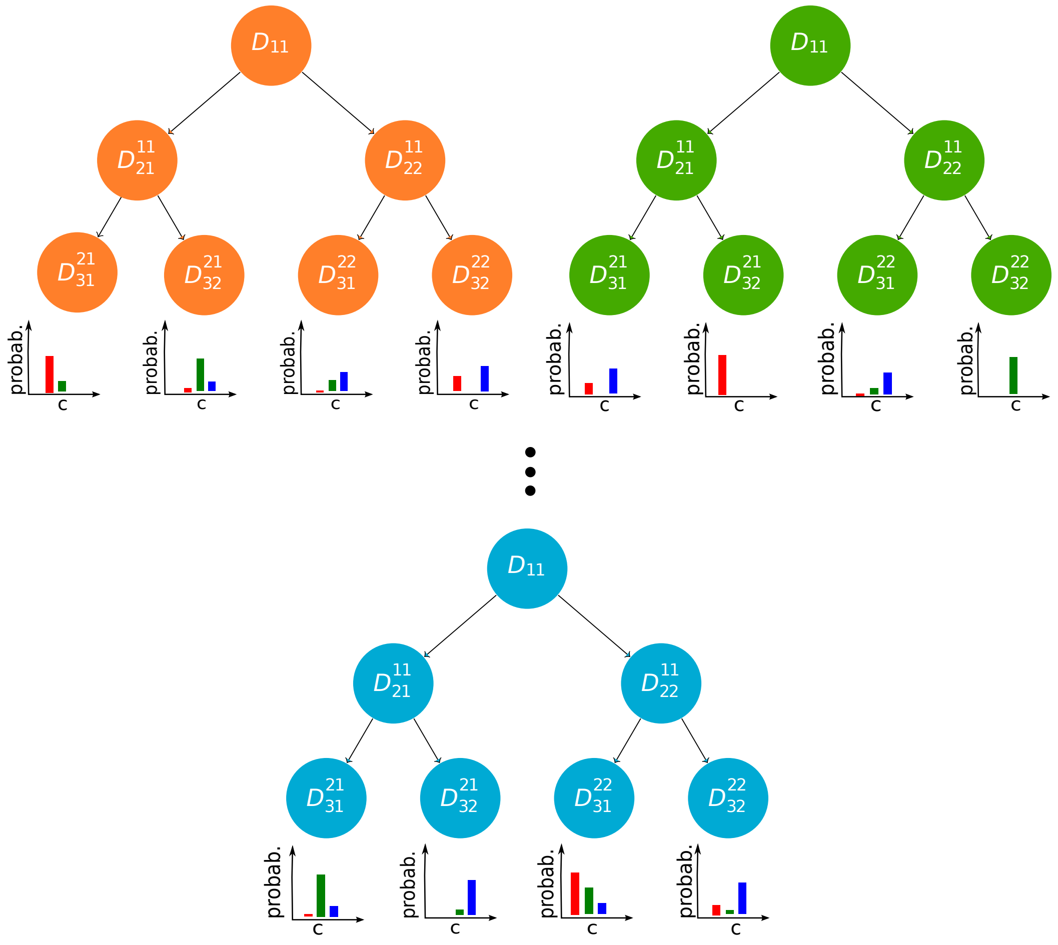


Figure 2.6: A RF example where several trees are learned. Each tree contains in the root D_{11} randomly chosen data from the whole dataset $\mathcal{D}$. Different colors were chosen for the trees to emphasize that the trees are different, even though the data in the root has the same notation, i.e., D_{11}. At each level the data is split using a split function which maximizes the information gain in the respective node. In the leaves the class probability distributions are stored and will be further used for classifying a new data point as shown in Figure 2.7.

2.3.2 Testing the Random Forest

The testing step is done online such that an unseen data point, x_{test}, traverses the learned trees until it reaches the leaves. It starts with the root and depending on the corresponding split function, x_{test} is then sent to the right or left child of each of the learned trees. The terminal node, the leaf, contains a classifier which predicts an output for x_{test}. For classification, the leaves store the classes distributions corresponding to the subset of the training data which managed to reach that specific leaf. That means, in each leaf of the t-th tree of the forest, $t = 1, \dots, nrMaxTrees$, a probability distribution of classes c is stored: $p_t(c|x_{\text{test}})$. In order to make a decision for x_{test}, the outputs from the leaves of several trees are combined such that the class with most of the votes gives the winner class. The most common approach for a RF prediction is by averaging the class probabilities over all the trees up to a maximum number of trees $NrMaxTrees$ [25]:

$$p(c|x_{\text{test}}) = \frac{1}{nrMaxTrees} \sum_{t=1}^{nrMaxTrees} p_t(c|x_{\text{test}}). \tag{2.3.4}$$

The class of x_{test} is given by the class with the most votes: max$(p(c|x_{\text{test}}))$.

The process is graphically shown in the example from Figure 2.7. An unknown data point, x_{test}, is pushed down the learned trees shown in Figure 2.6. At each level the split function $h(x_{\text{test}}, \theta_j^*)$ is applied for the test data point in order to decide on which branch to continue until x_{test} reached the leaf of each tree. The decision is then made by averaging the stored $p_t(c|x_{\text{test}})$ corresponding to each learned tree t.

2.4 Learning Vector Quantization

Learning Vector Quantization (LVQ) is a prototype based classification approach introduced by Kohonen [26]. LVQ uses adaptive prototype schemes based on attraction and repulsion during the learning process. LVQ models have low complexity and computational costs [89] and therefore they are widely applied in industrial applications, e.g., intelligent sensor systems [90]. LVQ is related to Vector Quantization (VQ) [91, 92] and to Self Organizing Maps (SOM) [93] with the distinction that LVQ is based on supervised learning, whereas the other ones are unsupervised learning methods.

In the following the basic standard schemes of LVQ are presented as described in [26] and with the mathematical details from [90]. Kohonen argues in [26] that LVQ is based on the Bayes theory of decisions [94] and VQ. In order to have such a classifier, the estimation of the class densities could be obtained by using unsupervised VQ with class-related data densities.

We suppose the training data $\mathcal{D} \in R^n$, with each $v \in \mathcal{D}$ and the corresponding class label $c(v) \in \mathcal{C} = \{1, \dots, C\}$. We also assume k prototypes $W = \{w_i \in R^n, i = 1, \dots, k\}$ and at least

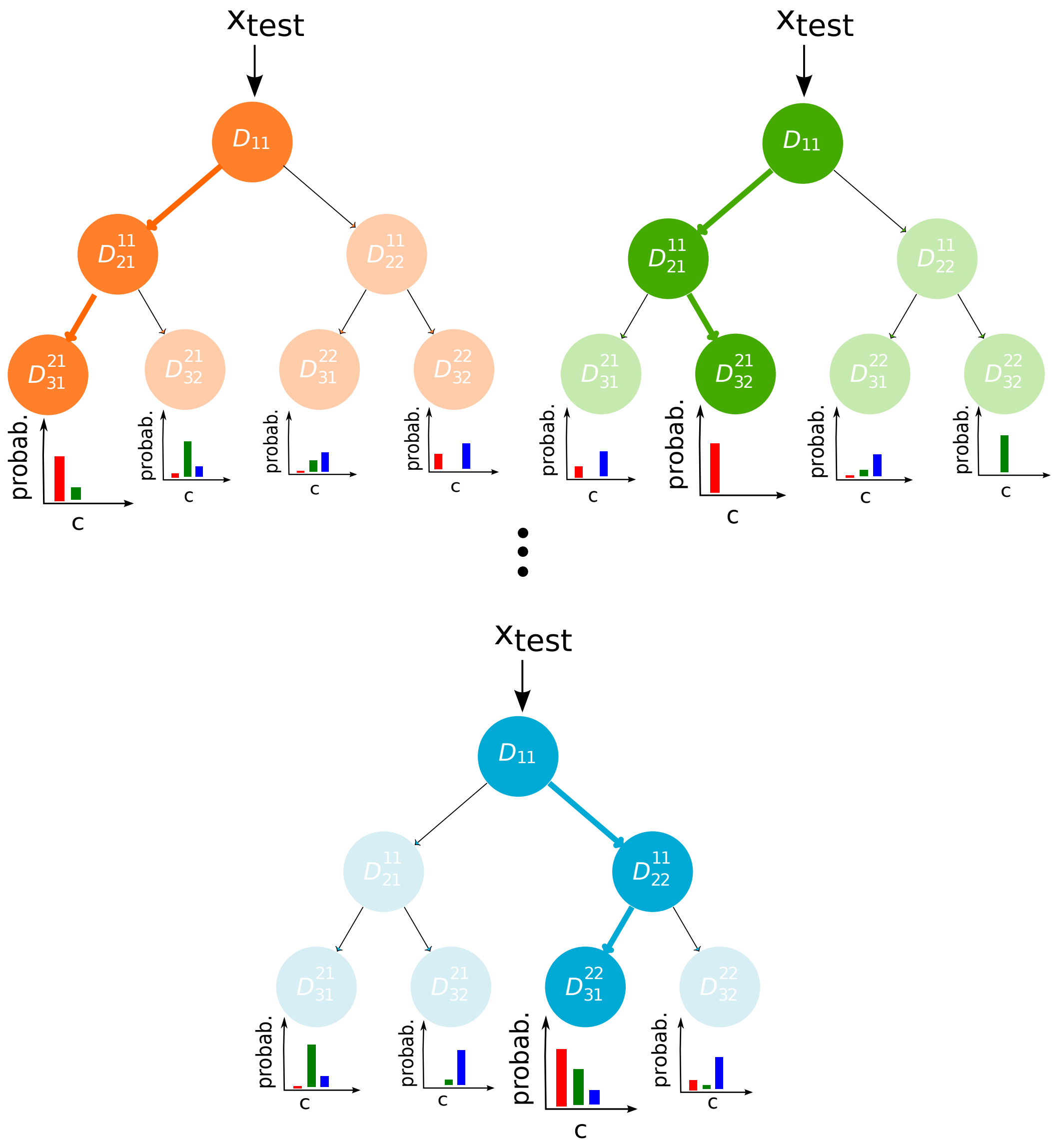


Figure 2.7: The trees learned using RF in the example from Figure 2.6 are used for classifying a novel data point x_{test}. The test data point traverses each tree deciding at each level either the right or left branch. The decision is made based on the learned split functions corresponding to each node of a tree. The process is repeated until x_{test} reaches the leaves which have stored the class probabilities. The class of x_{test} is given by the class with the most votes.

one prototype is assigned to each class, $c(w_i) \in \mathcal{C}$. Given a class probability model with classes $\mathcal{C} \in R^n$, the a priori probability of class $c \in \mathcal{C}$ is P_c, and $P(x|c)$ is the conditional probability that a vector $x \in R^n$ is generated by class c. $P(x) = \sum_c P(x|c)$ is the overall model density function.

For given training data $v \in \mathcal{D}$, the conditional probability $P(v|c) = 1$ if $c(v) = c$ and it is zero otherwise. From the Bayes theory [94] the model discriminant function is given by:

$$\delta_c(x) = P(x|c) \cdot P_c, \tag{2.4.1}$$

and the optimum decision is given by:

$$\delta_{c^*}(x) = \underset{c \in \mathcal{C}}{\operatorname{argmax}}\, \delta_c(x). \tag{2.4.2}$$

The Bayes class region B_c of class c contains the vectors $x \in R^n$ with $c^* = c$, that means the function for determining the class of the Bayes model is greater than zero for a normalization constant $\beta > 0$:

$$b_c(x) = \begin{cases} \frac{\delta_c(x) - \delta_{h^*}(x)}{\beta} & \text{if } x \in B_c \\ 0 & \text{if } x \notin B_c \end{cases} \tag{2.4.3}$$

with $h^* = \underset{h \in \mathcal{C} \setminus \{c\}}{\operatorname{argmax}}\, \delta_h(x)$, i.e., the incorrect class with respect to the Bayes model class region B_c. For $\beta = \int b(x)dx$, $b(x) = \sum_{c \in \mathcal{C}} b_c(x)$ is a class model density function which can be considered as a Bayes decision based class probability density which vanishes at the Bayesian class borders.

The winner takes it all rule is applied in order to get the best representation of $\mathcal{D}$ for the k prototypes W which have no label representation. The winner takes it all rule is based on a nearest prototype principle:

$$s(v) = \underset{i=1,\dots,k}{\operatorname{argmin}}\, d(v, w_i), \tag{2.4.4}$$

with d a general dissimilarity measure, usually chosen as the squared Euclidean distance, and $w_{s(v)}$ is the overall winner prototype. The expected quantization error depends on the overall model density $P(x)$ evaluated for data vector v and is given by:

$$E = \int_{\mathcal{D}} d(v, w_s(v)) P(v) dv \tag{2.4.5}$$

The set $R_i = \{v \in \mathcal{D} | s(v) = i\}$ represents the receptive field of the prototype w_i and it is also known as the Voronoi cell [95]. By replacing the overall density $P(v)$ in Equation 2.4.5 by the class model density function b for the data vector v, the quantization error becomes:

$$E_{\text{class-VQ}} = \int_{\mathcal{D}} d(v, w_s(v)) b(v) dv. \tag{2.4.6}$$

Therefore, $b(v)$ depends on $P(v|c)$ and not on the model probabilities $P(x|c)$. Further on, we assume $s^* = s^*(v)$ is the index such that v belongs to the Bayes model class region $v \in B_{s^*(v)}$. The quantization error shown in Equation 2.4.5, and respectively in Equation 2.4.6, can be optimized by stochastic gradient descent leading to an averaged update rule [90]:

$$\nabla_{w_i} E = -2 \int \delta_{i,s(v)}(v - w_i)P(v)dv \tag{2.4.7}$$

It follows that if $s^* = c(v)$ then the prototype $w_{s(v)}$ is moved towards the center of $B_{s^*(v)}$. Otherwise, if $s^* \neq c(v)$ then $w_{s(v)}$ is moved away from $B_{s^*(v)}$.

This leads to the basic learning rule introduced by Kohonen in [26] and known as LVQ1. The goal is to estimate the Bayes regions B_c. This is achieved by prototypes with assigned class labels. The prototype learning for LVQ1 is done by using the winner takes all scheme from Equation 2.4.4. Thus, the LVQ1 prototype adaptation scheme is given by:

$$\Delta w_i = \alpha S(v)(v - w_i), \tag{2.4.8}$$

where $\alpha \in (0, 1)$ and the shift (attraction or repulsion) is defined as:

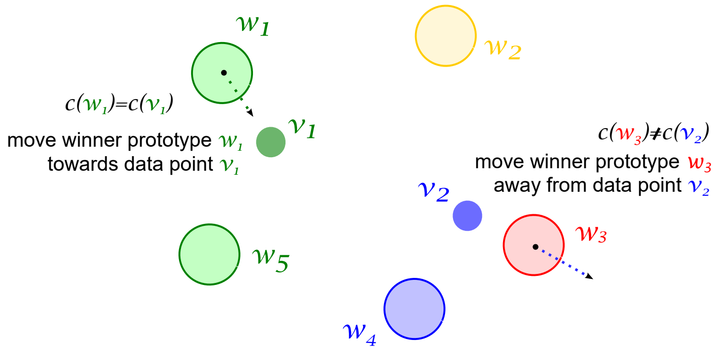
$$S(v) = \begin{cases} 1 & \text{if } s(v) = i \wedge c(w_i) = c(v), \\ -1 & \text{if } s(v) = i \wedge c(w_i) \neq c(v), \\ 0 & \text{else} . \end{cases}$$

The LVQ1 scheme is graphically sketched in the example from Figure 2.8. We consider 5 prototypes $w_i,\ i = 1, \ldots, 5$ corresponding to 4 different classes represented with different colors: class green has two prototypes $c(w_1) = c(w5)$, and the other classes (red, violet and yellow) have one representative prototype. Two new data points $v_1,\ v_2 \in \mathbb{R}^n$ are presented one at a time, the winner is identified (the closest prototype), and the winner prototype w_1 is moved closer to v_1 because they have the same class. On the other side w_3 is moved away from v_2 as they belong to different classes.

Improvements of LVQ1 were proposed by using the second and the third type of LVQ. LVQ2 uses a symmetric window around the midplane between two codebook vectors (prototypes), and a correction of the prototypes is done only if the training data v falls into the defined window.

LVQ3 updates also the second winner, i.e., the two closest codebook vectors are simultaneously updated, e.g., w_1 and w_2. Therefore, the updating rules for LVQ3 for a given data vector v are:

1. if v belongs to the same class of either w_1 or w_2, then the prototype with the same class as v will be moved towards v and away from v for the second winner; for example, if w_1 has the



w_1
w_2
$c(w_1)=c(v_1)$
move winner prototype w_1 towards data point v_1
v_1
$c(w_3)\neq c(v_2)$
move winner prototype w_3 away from data point v_2
v_2
w_3
w_5
w_4

Figure 2.8: LVQ1 prototype learning adaptation scheme example. The winner prototype w_1 is moved towards data point v_1, whereas the winner prototype w_3 is moved away from data point v_2 as they belong to different classes. Both prototypes are moved with learning rate α.

same class as $v \in \mathcal{D}$ then the codebook is updated using the learning rate α as follows:

$$\begin{cases} \Delta w_1 = \alpha(v - w_1) \\ \Delta w_2 = -\alpha(v - w_2) \end{cases} \tag{2.4.9}$$

2. if both w_1 and w_2 belong to the same class as v, then the codebook is updated using the learning rate α and $\varepsilon \in (0, 1)$ as follows:

$$\begin{cases} \Delta w_1 = \varepsilon\alpha(v - w_1) \\ \Delta w_2 = \varepsilon\alpha(v - w_2). \end{cases} \tag{2.4.10}$$

The LVQ3 scheme is graphically sketched in Figure 2.9 for the same example shown in 2.8. In this case, the two nearest prototypes to v_1, respectively to v_2, are moved. The two winner prototypes of v_1 are w_1 and w_5 which belong to the same class as v_1. Therefore, both w_1 and w_5 are moved towards $v1$ with learning rates α and ε. For v_2, w_3 is moved further away as their classes differ. The second winner prototype, w_4 with the same class as v_2, $c(w_4) = c(v_2)$, is moved towards v_2 with learning rate α.

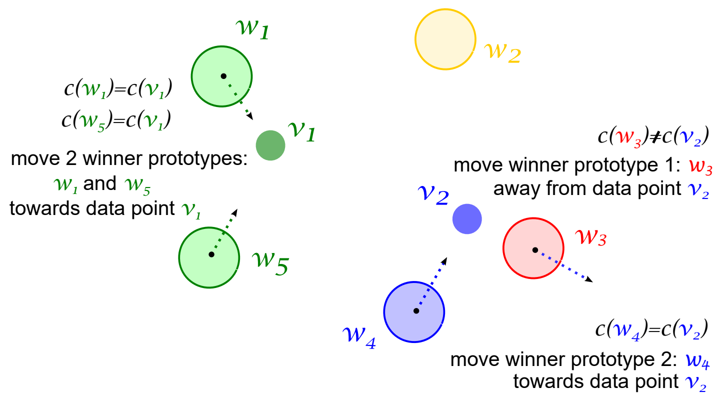

w_1
w_2
$c(w_1)=c(v_1)$
$c(w_5)=c(v_1)$
v_1
move 2 winner prototypes: w_1 and w_5 towards data point v_1
w_5
$c(w_3)\neq c(v_2)$
move winner prototype 1: w_3 away from data point v_2
v_2
w_3
w_4
$c(w_4)=c(v_2)$
move winner prototype 2: w_4 towards data point v_2

Figure 2.9: LVQ3 prototype learning adaptation scheme example: the two winner prototypes, w_1 and w_5, having the same class as data point v_1, are moved towards v_1 with learning rates α and ε. Whereas for data point v_2, the first winner prototype w_3 is moved away from v_2 with learning rate α because they have different classes. The second winner prototype w_4 is moved towards v_2 with learning rate α as they belong to the same class.

3 Algorithms for efficient sensing

This chapter presents the methods that we developed for efficient sensing. In the order of development, the following methods will be introduced: Visual Manifold Sensing (VMS), Foveated Manifold Sensing (FMS), with the corresponding extended version Hybrid Foveated Manifold Sensing (HyFMS), Hierarchical Manifold Sensing (HMS) with foveation and adaptive partitioning of the dataset, and Sensing Forest (SF).

3.1 Visual Manifold Sensing

Visual Manifold Sensing (VMS) is based on a geometric approach to the problem of efficient sensing. For a particular type of environment, i.e., images I^i in a dataset $\mathcal{D} = \left\{I^1, \ldots, I^p\right\}$, size$(\mathcal{D}) = p \times D$, the VMS approach consists of the following steps illustrated in Figure 3.1:

1. Learning step: for a given dataset a manifold of low dimension N, $N < D$ is learned by using either a linear subspace method such as the Principal Component Analysis (PCA) [20] or nonlinear manifold learning algorithms, such as Locally Linear Embedding (LLE) (introduced in Chapter 2.2.2).

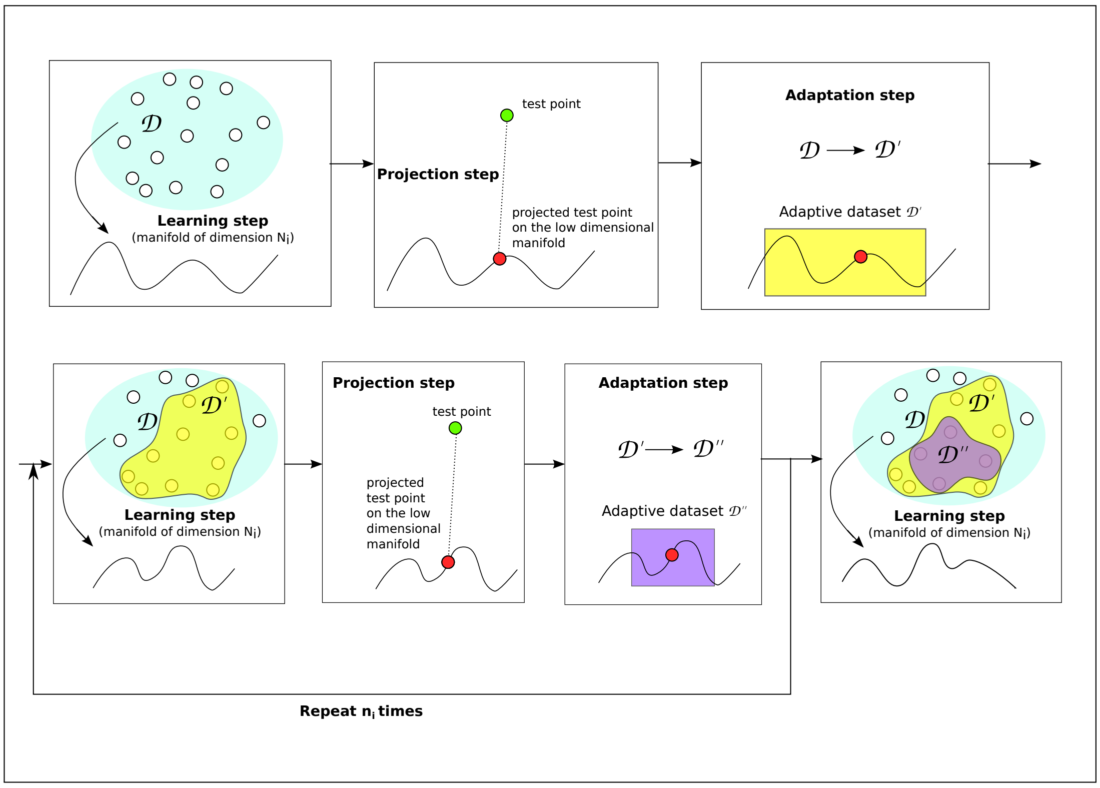
2. Projection step: any new data point, i.e., a test point outside $\mathcal{D}$, is first projected on the learned manifold by using the principle components ($\mathcal{D}_{\text{PCA}}$) or the pseudoinverse-matrix that minimizes the mean projection error on $\mathcal{D}$.

3. Adaptation step: the adaptive dataset $\mathcal{D}_{\text{VMS}}$ is a subset of the original dataset (defined as a neighborhood of the location obtained by the projection step) embedded in the same dimension N:

$$\mathcal{D}_{\text{VMS}} : \{\mathcal{D} \rightarrow \mathcal{D}' \rightarrow \mathcal{D}'' \rightarrow \ldots \rightarrow \mathcal{D}^k\}, \tag{3.1.1}$$

with $k = N_{\max}$, the maximum dimension in which we learn the manifolds. The adaptive dataset $\mathcal{D}_{\text{PCA}}$ is obtained in a similar way as $\mathcal{D}_{\text{VMS}}$ with the difference that the low dimensional representation is obtained by using principal components:

$$\mathcal{D}_{\text{PCA}} : \{\mathcal{D} \rightarrow \mathcal{D}'_{pc} \rightarrow \mathcal{D}''_{pc} \rightarrow \ldots \rightarrow \mathcal{D}^{k}_{pc}\}. \tag{3.1.2}$$



Learning step
(manifold of dimension N_i)
test point
Projection step
projected test point on the low dimensional manifold
Adaptation step
D → D'
Adaptive dataset D'
D'
D''
D' → D''
Adaptive dataset D''
projected test point on the low dimensional manifold
Repeat n_i times

Figure 3.1: The main steps of VMS are: the learning step followed by projection and adaptation. **Learning step**: a low N_i-dimensional representation is learned at the current iteration i for the dataset $\mathcal{D}$. **Projection step**: an unknown scene (test point) is projected on the previously low-dimensional learned representation. **Adaptation step**: the dataset $\mathcal{D}$ is adapted to $\mathcal{D}'$ by considering a neighborhood of the projection of the test point on the learned manifold. The three steps are repeated n_i times and the dataset is adapted correspondingly: $\mathcal{D} \rightarrow \mathcal{D}' \rightarrow \mathcal{D}''$.

The process of embedding the adaptive dataset $\mathcal{D}_{\text{VMS}}$ in a manifold of dimension N_i corresponding to iteration i, is repeated n_i times with different sizes of $\mathcal{D}_{\text{VMS}}$, as it can be seen in the second row of Figure 3.1. Thus, n_i denotes how often we iterate the embedding in a particular dimension N_i, before moving to a manifold with higher dimension N_g, $N_g > N_i$, and until we reach a predefined maximal dimension $N_{\max}$. The goal of doing iterations in the same dimension is to obtain a better local linear approximation of the manifold and thus a better projection. The pseudoinverse-matrix is needed for the projection of the test points because the test images are unknown, and thus we cannot use LLE to find the low-dimensional coordinates of the test points.

The VMS method is based on the implementation of an iterative LLE algorithm which is presented in Algorithm 5. The manifold learning algorithm LLE was introduced in Chapter 2.2.2 in Algorithm 1. The corresponding notations for Algorithm 5 are explained in Table 3.1. The *VMS* function takes as input the dataset $\mathcal{D}_{\text{VMS}}$, the number of nearest neighbors used by the LLE algorithm, the dimension N_i of the learned representation at iteration i, the maximum dimension N_{max} and the test image. The algorithm returns the adapted dataset $\mathcal{D}_{\text{VMS}}$ and the projected testing data on $\mathcal{D}_{\text{VMS}}$ corresponding to dimension N_i of the learned manifolds. In line 3 of Algorithm 5 a low dimensional manifold of dimension N_i is learned for the dataset $\mathcal{D}_{\text{VMS}}$ using the function *LLE*$(\mathcal{D}_{\text{VMS}},\, r,\, N_i)$ defined in Chapter 2.2.2, Algorithm 1, with r neighbors. Afterwards, the pseudoinverse-matrix that minimizes the mean projection error on $\mathcal{D}_{\text{VMS}}$ is learned and it is used in the next step to project the test data X_{test} on the learned manifold.

The adapted dataset $\mathcal{D}_{\text{VMS}}$ has a decreasing number q of data points. This is graphically shown in Figure 3.1 by: $\mathcal{D} \rightarrow \mathcal{D}' \rightarrow \mathcal{D}''$. VMS defines for the adaptive dataset, subsets of the neighborhood of size q and uses the following heuristics for q:

$$q = (n_i - i + 2) \cdot r \tag{3.1.3}$$

Thus, the number of data points in $\mathcal{D}_{\text{VMS}}$ depends on: (i) the number r of neighbors that we select for the VMS algorithm, (ii) the current iteration i, and (iii) the total number of iterations n_i corresponding to iteration i. Note that, at each iteration $i,\ i > 1$, the number of points for $\mathcal{D}_{\text{VMS}}$ is reduced. At line 8 of the algorithm, the dimension of the manifold is further increased and the process is repeated until a maximum dimension, N_{max} is reached.

While sensing an unknown scene, VMS continuously adapts $\mathcal{D}_{\text{VMS}}$ and the corresponding em-

Algorithm 5 Visual Manifold Sensing (VMS)

```
 1: function VMS($\mathcal{D}_{\text{VMS}}$, $r$, $N_i$, $N_{\text{max}}$, $X_{\text{test}}$)
 2:     while $N_i \leqslant N_{max}$ do
 3:         $Y \leftarrow$ LLE(($\mathcal{D}_{\text{VMS}}, r, N_i$))          ▷ Learning manifold
 4:         $A^+ \leftarrow Y \cdot \mathcal{D}_{\text{VMS}}^{-1}$          ▷ Learning pseudo-inverse matrix
 5:         $Y_{\text{test}} \leftarrow A^+ \cdot X_{\text{test}}$          ▷ Projecting test data
 6:         compute all $d(y^l_{\text{test}}, y^j)$          ▷ Computing distances
 7:         adapt $\mathcal{D}_{\text{VMS}}$          ▷ Adapting dataset
 8:         $N_i \leftarrow N_i + 1$          ▷ Increasing dimension of manifold
 9:     end while
10: return $\mathcal{D}_{\text{VMS}}$, $Y_{\text{test}}$
11: end function
```

Notation	Description
$\mathcal{D}_{\text{VMS}}$	adaptive dataset with p data points (images) of size D
r	number of nearest neighbors for each data point
N_i	dimension of the manifold corresponding to iteration i
$N_{\max}$	maximum dimension of the manifold
X_{test}	test data
i	current iteration
Y	low dimensional embedding coordinates of the data points in $\mathcal{D}_{\text{VMS}}$
y^j	j-th element of Y, i.e., coordinates of a particular point in $\mathcal{D}_{\text{VMS}}$
A^+	pseudo-inverse matrix
Y_{test}	projected test data on the low dimensional manifold
y^l_{test}	l-th element of Y_{test}, i.e., coordinates of a particular point in X_{test}
$d(y^l_{\text{test}}, y^j)$	Euclidean distance between y^l_{test} and y^j

Table 3.1: Notations for VMS Algorithm 5.

bedding is learned. Thus, VMS is based on an online learning, such that for each novel scene, a new representation is learned. In the case of a real-time sensing scenario, one would need to learn a hierarchical representation of the data such that the corresponding manifolds of $\mathcal{D}_{\text{VMS}}$ are learned prior to the sensing of the novel scenes. Therefore, an offline learning would be necessary. In this case, while sensing an unknown scene, the already learned manifolds corresponding to $\mathcal{D}_{\text{VMS}}$ can be accessed directly and a decision can be made based on the chosen classifier.

By using PCA for learning linear low-dimensional representations and adapting the dataset as shown in Equation 3.1.2, we obtain an iterative PCA as shown in Algorithm 6. The Algorithm follows the same steps as VMS Algorithm 5 with the difference that any new data point is simply projected on the N_i principal components of $\mathcal{D}_k$, which correspond to the N_i largest eigenvalues of $\mathcal{D}_{\text{PCA}}$. The function *Iter-PCA* in Algorithm 6 takes as input the dataset $\mathcal{D}_{\text{PCA}}$, the number of principal components N_i chosen at a specific iteration i, the maximum number of principal components $N_{\max}$ and the data used for testing. In line 3 of the algorithm the function $\text{princomp}(\mathcal{D})$ performs principal component analysis on $\mathcal{D}$ and returns the principal component coefficients stored in matrix U of size $(D \times D)$. Y represents the data from the adaptive data set $\mathcal{D}_{\text{PCA}}$ projected on the N_i principal components of $\mathcal{D}_{\text{PCA}}$ that correspond to the N_i largest eigenvalues of $\mathcal{D}_{\text{PCA}}$, y^j is the j-th element of Y. Y_{test} consists of the test data projected on the N_i principal components of $\mathcal{D}_{\text{PCA}}$; y^l_{test} is the l-th element of Y_{test}.

Algorithm 6 Iterative PCA

```
 1: function Iter-PCA($\mathcal{D}_{\text{PCA}}$, $N_i$, $N_{\text{max}}$, $X_{\text{test}}$)
 2:     while $N_i \leqslant N_{max}$ do
 3:         $U \leftarrow \text{princomp}(\mathcal{D}_{\text{PCA}})$            ▷ Learning Karhunen-Loeve matrix
 4:         $Y \leftarrow U(1:N_i,:) \cdot \mathcal{D}_{\text{PCA}}^{\text{T}}$
 5:         $Y_{\text{test}} \leftarrow U(1:N_i,:) \cdot X_{\text{test}}$           ▷ Projecting test data
 6:         compute all $d(y_{\text{test}}^l, y^j)$                  ▷ Computing distances
 7:         adapt $\mathcal{D}_{\text{PCA}}$                              ▷ Adapting dataset
 8:         $N_i \leftarrow N_i + 1$                        ▷ Increasing dimension of manifold
 9:     end while
10: return $\mathcal{D}_{\text{PCA}}$, $Y_{\text{test}}$
11: end function
```

3.2 Foveated Manifold Sensing

Foveated Manifold Sensing (FMS) follows the geometrical approach of VMS but operates on a different dataset which we call the foveated dataset. Thus, the new dataset is created from the original dataset such that it contains only the regions of interest extracted by using a saliency model based on the geometric invariants of the structure tensor of the images in the dataset $\mathcal{D}$. The saliency model based on the structure tensor is described in Chapter 2.1.3. By using the foveated dataset, the FMS approach resembles biological visual sensing strategies, which efficiently sense only the information required for a particular task. FMS is based, as VMS, on online learning, which was discussed in Chapter 3.1.

In this section we describe how the foveated dataset is created, we present the FMS algorithm, and we propose a hybrid version of FMS, that is HyFMS. HyFMS is also inspired by biological vision because it acquires first the gist of a scene and afterwards continues with a more refined sampling.

3.2.1 Foveation

Given the original dataset $\mathcal{D}$, we involve foveation in order to create a foveated dataset $\mathcal{D}_{foveated}$. This is done by considering only the pixels which are salient on average over the original dataset $\mathcal{D}$. Although these pixels do not necessarily form a compact region of interest (ROI) we will denote the collection of salient pixels as ROI. The ROI is extracted by using a saliency model based on the geometric invariants of the structure tensor of the images in the dataset $\mathcal{D}$. The structure tensor was already described in Chapter 2.1.3. The foveated dataset $\mathcal{D}_{foveated}$ is computed in the following three steps:

1. Structure tensor J and invariant K as saliency measures

For each image I^i in $\mathcal{D}$, $i = 1, \dots, p$ (p data points), the geometric invariant K^i is computed based on the structure tensor **J** . The structure tensor **J** and the invariant K (the determinant of the matrix that contains the locally averaged products of first-order partial derivatives I_x and I_y) were presented in Chapter 2.1.3 - Equations 2.1.6 and 2.1.7.

2. Choosing and optimizing the saliency threshold

For each image I_i we normalize K^i to the range $[0, 1]$, we choose a threshold θ and define an average saliency template R as:

$$R = \begin{cases} 0, & \sum_i K^i(x,y) < \theta \\ 1, & \sum_i K^i(x,y) \geq \theta \end{cases} \tag{3.2.1}$$

Thus, the threshold θ and the average saliency in the dataset determine the areas of interest used in the foveated approach. When evaluating the recognition performance, the recognition rate is computed for different values of the threshold θ (resulting in differently sized regions of interest) and an optimal threshold is determined (based on the training set).

3. Regions of interest

We now blend the images in the dataset by using the average saliency template R to obtain the regions of interest T^i for every image I^i: $T^i = R \cdot I^i$. The resulting foveated dataset has the following structure:

$$\begin{gathered} \mathcal{D}_{\text{foveated}} : \{\mathcal{D}_f \to \mathcal{D}'_f \to \dots \to \mathcal{D}^k_f\}, \\ \begin{cases} \mathcal{D}_f = \{T^1, \dots, T^p\}, \text{size}(\mathcal{D}_f) = p \\ \mathcal{D}'_f = \left\{T^1, \dots, T^{p'}\right\}, \text{size}(\mathcal{D}'_f) = p',\ p' < p \\ \mathcal{D}''_f = \left\{T^1, \dots, T^{p''}\right\}, \text{size}(\mathcal{D}''_f) = p'',\ p'' < p' < p \\ \vdots \\ \mathcal{D}^k_f = \left\{T^1, \dots, T^{p^k}\right\}, \text{size}(\mathcal{D}^k_f) = p^k,\ p^k < \dots < p' < p \end{cases} \end{gathered} \tag{3.2.2}$$

with $k = N_{\text{max}}$, the maximum dimension in which we learn the manifolds.

Algorithm 7 sketches the steps of creating the foveated dataset. The notations used for the algorithm are included in Table 3.2. Algorithm 7 computes for the given dataset $\mathcal{D}$, the corresponding foveated dataset $\mathcal{D}_{foveated}$ of the same size p and dimension D; the only difference being, that nonsalient pixels are set to zero in every image. For each image I^i in the given dataset $\mathcal{D}$ the geometric invariant K^i is computed as shown in line 4 of the algorithm. K is defined as the determinant of the structure tensor **J** (a matrix with the locally averaged products of first order partial derivatives of image I^i - in line 3). Each invariant image K^i is normalized to the range $[0, 1]$; the normalized K^i are then summed over all images and the resulting average saliency map is then transformed into a binary saliency map R based on a threshold θ.

Notation	Description
$\mathcal{D}$	$\mathcal{D} = \left\{I^1, \ldots, I^p\right\}$ contains p data points of dimension D
I^i	image i with coordinates (x, y)
J	structure tensor
$*$, w	convolution with kernel w
I_x, I_y	first order partial derivatives of I^i
K	geometric invariant of **J**
R	average saliency template
$\circ$	element-wise product of matrices
T^i	region of interest for image I^i

Table 3.2: Notations for Algorithm 7.

Algorithm 7 Creation of foveated dataset

```
1:  function fov-ds($\mathcal{D}$)
2:      for all $I^i$ in $\mathcal{D}$ do
3:          $\mathbf{J} \leftarrow w * \left[I_x^2 \; I_x I_y; I_x I_y \; I_y^2\right]$
4:          $K^i \leftarrow \det(\mathbf{J})$
5:          $K^i_{norm} \leftarrow \text{normalize}(K^i)$
6:      end for
7:      if $\sum_i \left(K^i_{norm}(x, y)\right) \geq \theta$ then
8:          $R(x, y) \leftarrow 1$
9:      else
10:         $R(x, y) \leftarrow 0$
11:     end if
12:     for all $I^i$ in $\mathcal{D}$ do
13:         $T^i \leftarrow R \circ I^i$
14:         $\mathcal{D}_{foveated}(i, :) \leftarrow T^i(:)$
15:     end for
16:     return $\mathcal{D}_{foveated}$
17: end function
```

3.2.2 Foveated Manifold Sensing algorithm

The FMS algorithm is obtained by applying the VMS Algorithm 5 on a different dataset, $\mathcal{D}_{\text{foveated}}$, instead of $\mathcal{D}_{\text{VMS}}$: $VMS(\mathcal{D}_{\text{foveated}}, r, N_i, N_{\max}, X_{\text{test}})$. The VMS algorithm is described in Algorithm 8 and the corresponding notations are shown in Table 3.3.

Notation	Description
$\mathcal{D}_{\text{foveated}}$	foveated dataset, $\mathcal{D}_{\text{foveated}} : \{\mathcal{D}_f \to \mathcal{D}'_f \to \ldots \to \mathcal{D}^k_f\}$
r	parameter for LLE in VMS Algorithm 5 number of nearest neighbors for each data point
N_i	parameter for LLE in VMS Algorithm 5 dimension of the manifold corresponding to iteration i
$N_{\max}$	parameter of VMS Algorithm 5: maximum dimension of the manifold
X_{test}	test data
Y_{test}	projected test data on the low dimensional manifold

Table 3.3: Notations for FMS Algorithm 8.

Algorithm 8 Foveated Manifold Sensing (FMS)

```
1: function FMS($\mathcal{D}$, $r$, $N_i$, $N_{\max}$, $X_{\text{test}}$)
2:     $\mathcal{D}_{foveated} \leftarrow$ fov-ds($\mathcal{D}$)
3:     $[\mathcal{D}_{\text{foveated}}, Y_{\text{test}}] \leftarrow$ VMS($\mathcal{D}_{\text{foveated}}, r, N_i, N_{\max}$)
4: return $\mathcal{D}_{\text{foveated}}, Y_{\text{test}}$
5: end function
```

The algorithm uses LLE for learning manifolds of foveated dataset $\mathcal{D}_{\text{foveated}}$. The unknown scene X_{test} is first projected on the learned manifold. Afterwards, the dataset $\mathcal{D}_{\text{foveated}}$ is adapted as described in Algorithm 5. We repeat the process until we reach the maximum given dimension $N_{\max}$. The *FMS* function returns the adapted foveated dataset, and the projected test data on the low dimensional manifold.

3.2.3 Hybrid Foveated Manifold Sensing algorithm

The hybrid version of FMS, denoted HyFMS, is a sensing strategy which starts with VMS and then continues with FMS. The HyFMS follows the more recent view on how a human observer performs a visual scene recognition by extracting the gist of a scene. Therefore, the HyFMS extracts first the global information of a scene by performing VMS, and continues with more refined and localized underlying sampling by applying FMS. The gist of a scene concept was presented in Chapter 2.1.2.

This hybrid algorithm is described in Algorithm 9. We apply the FMS Algorithm 8 with the same parameters r, N, N_{max}, X_{test} and with the hybrid dataset defined as:

$$\mathcal{D}_{\text{hybrid}} : \{\mathcal{D} \to \mathcal{D}'_f \to \ldots \to \mathcal{D}^k_f\},$$

where $\mathcal{D}$ is the original dataset and $\mathcal{D}'_f, \ldots, \mathcal{D}^k_f$ are defined in the FMS section 3.2.

Algorithm 9 Hybrid Foveated Manifold Sensing (HyFMS)

```
1: function HyFMS($\mathcal{D}_{\text{hybrid}}$, $r$, $N$, $N_{\text{max}}$, $X_{\text{test}}$)
2:     [$\mathcal{D}_{\text{hybrid}}$, $Y_{\text{test}}$] ← FMS($\mathcal{D}_{\text{hybrid}}$, $r$, $N$, $N_{\text{max}}$, $X_{\text{test}}$)
3: return $\mathcal{D}_{\text{hybrid}}$, $Y_{\text{test}}$
4: end function
```

3.3 Hierarchical Manifold Sensing with foveation and adaptive partitioning of the dataset

VMS and FMS strongly depend on the choice of the following parameters: (i) the number of neighbors used for LLE algorithm, (ii) the decreasing size of the adaptive dataset, and (iii) the dimensions of the manifolds at each iteration of the algorithm. Moreover, both approaches operate on the entire dataset while sensing. As discussed in Chapter 3.1 and 3.2, VMS and FMS are both based on an online learning algorithm because the manifolds corresponding to the adaptive subsets are learned during the actual sensing of novel scenes. Thus, for each novel scene, all the three steps of the algorithm (learning, projection and adaptation) are performed. The online learning approach is not suitable for a real-time sensing scenario. In this case, one would need to perform an offline learning such that the manifolds (corresponding to the possible subsets of data) are learned before doing the actual sensing. Thus, the two steps (learning and adaptation) should be done offline. Consequently, the learned low-dimensional representations can be accessed directly while a novel scene is sensed and a decision can be taken according to the chosen classifier. Therefore, Hierarchical Manifold Sensing is proposed as an extension of VMS and FMS. HMS includes an adequate hierarchical partitioning of the data, learned prior to sensing.

As VMS and FMS, HMS also learns low-dimensional manifolds. For simplicity, a linear method, PCA [20], is used for learning the low-dimensional representations of the foveated dataset. The hierarchical partitioning of the dataset is done by clustering the data in the low-dimensional manifolds using the k-means algorithm [21, 22]. In the following we will describe how the hierarchical partitioning of the dataset is done, and how a new data point is sensed by using the learned hierarchical representations.

3.3.1 Hierarchical partitioning of the dataset

We create a tree with L levels which contains the hierarchical partitioning of the dataset. The partitioning is done in the following way: (i) we learn a manifold of dimension N_L (corresponding to level L of the tree) by using PCA, and (ii) we cluster the N_L-dimensional representation of the data into k clusters using the k-means algorithm. In the approach presented here, N_L increases with 1 at each level of the tree. The structure of the tree is presented in Figure 3.2. The root of the tree is defined as $\mathcal{D}_{11}$ and for $L > 1$, the nodes of the tree are denoted as $\mathcal{D}_{Lk_c}^{(L-1)k_f}$, with $k_c = \overline{1,k}$ being the current cluster and k_f the index of the father node.

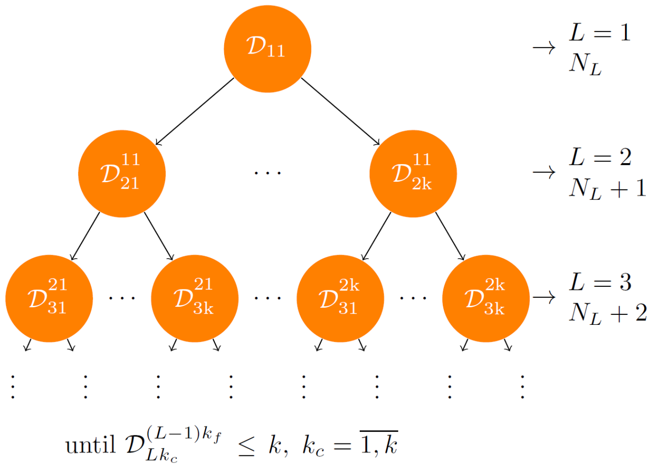


Figure 3.2: Hierarchical partitioning of the dataset $\mathcal{D}$. The root is defined as $\mathcal{D}_{11}$. L is the level of the tree, k is the number of clusters, k_c is the current cluster and k_f is the index of the father node. Each node $\mathcal{D}_{Lk_c}^{(L-1)k_f}$ of the tree is split into k clusters and contains the learned manifold in dimension N_L, $N_L + 1$, etc., of the corresponding cluster.

Considering the representation of the tree shown in Figure 3.2, we initialize the nodes by applying the *Partitioning* function presented in Algorithm 10. The notations used in Algorithm 10 are presented in Table 3.4. The *partitioning* function expects as input the current cluster, $currCluster$, the number k of clusters, and N_L, the number of principal components. The function computes for each node of the tree: the matrix U which contains the feature vectors of dimensionality N_L (line 3 of the algorithm), the projected data points of the current cluster on the matrix U (line 4 of the algorithm), the index which indicates to which of the k clusters the images in $currCluster$ were clustered to by using the k-means algorithm (line 5 in the algorithm). We use the $index$ to create the subset for the k-th child: we select with the *keep* function in line 7 of Algorithm 10 only the images which belong to cluster j. Recursively we compute U, Y, $index$ and $child$ (line 8 of the algorithm). If the current cluster does not have more data points than the number of clusters, the cluster will be empty and the algorithm stops.

Notation	Description
$currCluster$	current cluster
L	current level of the tree
k	number of clusters for k-means
N_L	number of principal components for each L
U	matrix containing the feature vectors, size$(U) = (D \times N_L)$
Y	projected data points of $currCluster$ on U, size$(Y) = (N_L \times p)$
$index$	$index = \overline{1, k}$, each I_i in $currCluster$ belongs to cluster $index$
$currCluster.child$	contains the children of the current cluster node
$currCluster.hp$	contains U, Y and $index$ for each node of the tree

Table 3.4: Notations for Algorithm 10.

Algorithm 10 Hierarchical partitioning of the dataset

```
1:  function Partitioning($currCluster$, $k$, $N_L$)
2:      if size($currCluster$) $> k$ then
3:          $U \leftarrow$ PCA( $currCluster$, $N_L$)
4:          $Y \leftarrow U' \cdot currCluster'$
5:          $index \leftarrow$ kMeans( $Y$, $k$)
6:          for $j = 1 : k$ do
7:              $currCluster.child(j) \leftarrow$ Keep($currCluster, index, j$)
8:          Partitioning($currCluster.child(j)$, $k$, $N_L + 1$)
9:          end for
10:         $currCluster.hp \leftarrow \{U, Y, index\}$
11:     else
12:         $currCluster.hp \leftarrow \varnothing$
13:     end if
14: return $currCluster$
15: end function
```

3.3.2 Hierarchical sensing of unknown scenes

The HMS approach uses the hierarchical partitioning of the dataset presented before to solve the sensing problem in an efficient way. Therefore, an unknown scene, i.e., test point x_{test} outside $\mathcal{D}$, that we wish to sense, is successively projected on the learned tree. We project x_{test} on the cluster which contains the nearest neighbor of the projected test point on the learned embedding. Algorithm 11 presents this procedure. The notations used in Algorithm 11 are included in Table 3.5. The test point is first projected on the low-dimensional representation of the $rootCluster$ in line 5 of the algorithm. In line 6 the function *find1NN* searches for the nearest neighbor $nearNeigh$ of the projected point on the learned manifold. The *classify* function checks if the test point was correctly classified (if the nearest neighbor and the test point belongs to the same class). In line 8 we check to which cluster the nearest neighbor belongs to and we define it as $clusterNN$. In line 9 of the algorithm, the $currCluster$ is updated and it is given by the child which corresponds to cluster $clusterNN$.

The algorithm continues by projecting the test point on the next level L of the tree as long as the current cluster is not empty.

Notation	Description
$currCluster$	current cluster which contains $currCluster.child$ and $currCluster.hp$
x_{test}	test data point, size$(x_{\text{test}}) = 1 \times D$
y_{test}	projected x_{test} on the low-dimensional representation
$nearNeigh$	nearest neighbor of y_{test} on the low-dimensional representation
$recogHMS$	is 1 if x_{test} and $nearNeigh$ have the same class and 0 otherwise
$clusterNN$	indicates to which cluster the $nearNeigh$ belongs to

Table 3.5: Notations for Algorithm 11.

Algorithm 11 Sensing novel scenes using the hierarchical partitioning of the dataset

```
1: **function** projectTestData($rootCluster$, $x_{\text{test}}$)
2:     $currCluster \leftarrow rootCluster$
3:     **while** $currCluster \neq \varnothing$ **do**
4:         $[U, Y, index] \leftarrow currCluster.hp$
5:         $y_{\text{test}} \leftarrow U' \cdot {x_{\text{test}}}'$
6:         $nearNeigh \leftarrow$ find1NN($y_{\text{test}}, Y$)
7:         $recogHMS \leftarrow$Classify($nearNeigh, x_{\text{test}}$)
8:         $clusterNN \leftarrow index(nearNeigh)$
9:         $currCluster \leftarrow currCluster.child(clusterNN)$
10:    **end while**
11: **end function**
```

3.4 Sensing Forest

The Sensing Forest (SF) is a prototype-based Random Forest (RF) with prototypes learned with k-means clustering [21, 22], which are afterwards finally tuned using Learning Vector Quantization (LVQ). The SF learns, as HMS does, low-dimensional representations of the data in a hierarchical manner. For the SF, the low-dimensional representation of the data is learned by using k-means [23]. The k-centroids computed with the k-means algorithm are used as feature vectors, which adapt while traversing the hierarchical representation. RF and LVQ were already introduced in Chapter 2.3, respectively in Chapter 2.4.

In the following we describe the approach for learning the hierarchical partitioning of the dataset for each tree of the SF. Those steps represent the offline part of the Sensing Forest (SF). After we have learned the hierarchical representation of the given dataset, we can project on it an unknown scene, i.e., a test point outside the dataset $\mathcal{D}$ that we wish to sense.

3.4.1 Learning the Sensing Forest

We first learn a tree with L levels which performs a hierarchical partitioning of the dataset. The partitioning is computed as follows: (i) we learn a low-dimensional representation of the data of dimension k for each level L of the tree by using k centroids computed with the k-means algorithm, (ii) we randomly choose N_L features, i.e., N_L centroids, from the k-dimensional learned representation, $N_L < k$, and (iii) we split the dataset $\mathcal{D}$ based on the learned representation (of size $D \times N_L$) such that the information gain is maximized. In the approach presented here $N_L = 2$ and so binary trees are computed.

In order to find better prototypes, thus a refined codebook for our approach, we use LVQ algorithms, i.e., the basic schemes LVQ1 and LVQ3 presented in Chapter 2.4. The LVQ1 approach moves only the winner prototype, whereas LVQ3 updates, at each step, both the winner and the runner up prototype. We here use as initial codebook for LVQ1 the k centroids computed with k-means. In the following we describe the proposed algorithm in detail.

First the nodes of the tree are initialized by applying the *PARTITIONING* function presented in Algorithm 12. In this regard the function in Algorithm 13 (learning the initial prototypes with the k-Means algorithm) and the function in Algorithm 14 (fine tuning the initial prototypes with LVQ) are used. The notations used throughout the algorithms are explained in Table 3.6.

The *Partitioning* function expects as input $currCluster$, the current cluster, the corresponding labels, $labelsCurrCl$, and N_L, the number of feature vectors for each level L. First, a feature representation is learned using the *kMEANSIG* function described in Algorithm 13. k-means clustering is applied for the current cluster and k centroids are learned. k is equal to the number of classes from the respective $currCluster$ at each level L. The learned features are stored in matrix

C_{kms} with k components of dimension D and have the corresponding $labelsC_{kms}$ labels (line 2 in Algorithm 13). Afterwards, N_L random features are chosen from C_{kms} with $N_L < k$ and stored in matrix U_{kms} (line 3 in Algorithm 13). Next, the nearest neighbor of the projected $currCluster$ on the learned representation is searched. The corresponding index is stored in $index_{kms}$ (line 4 in Algorithm 13) and the information gain IG_{kms} for the split is computed.

Algorithm 12 subsequently applies the $LVQIG$ function described in Algorithm 14. The function uses the *LVQ* function in line 3 of Algorithm 14 to learn prototypes. *LVQ* takes as input the current cluster with corresponding labels, the codebook (centroids and labels learned with k-Means) and the learning rate $\alpha \in (0, 1]$. *LVQ* uses $codebook$ for the initial prototypes and adapts them (into C_i) by using schemes of attraction and repulsion with different learning rates α_i. In this work, α_i takes values from 0 to 1 in steps of 0.02. In a similar way as *kMEANSIG*, N_L prototypes are randomly chosen from C_i into U_i (line 4). For each α_i the nearest prototype is searched for $currCluster$ after projection (line 5) and the corresponding information gain IG_i is computed (line 6). Finally, the prototypes U_{lvq} (of size $D \times N_L$, randomly chosen from C_{lvq} of size $D \times k$) which maximize the information gain are chosen. *LVQIG* returns C_{lvq}, U_{lvq}, the corresponding cluster to which $currCluster$ belongs to, $index_{lvq}$, and the corresponding information gain IG_{lvq}. This process of tuning the prototypes learned with k-means by using LVQ is graphically shown in the example from Figure 3.3. Three different classes are represented with three different colors. Each class has an initial prototype, corresponding to the 3 centroids computed with k-means, $k = 3$: C^1_{kms}, C^2_{kms}, and C^3_{kms}. By using this initial codebook, the 3 prototypes are finely tuned by applying LVQ. Thus, the codebook consists of: C^1_{lvq}, C^2_{lvq}, and C^3_{lvq}. As mentioned before, the prototypes which maximize the information gain are then chosen.

Afterwards, Algorithm 12 chooses the prototypes (U_{kms} or U_{lvq}) that maximize the information gain (line 7) and computes in P the class probabilities distributions for the corresponding split. The returned $index$ (line 7) is used to create the subset for the N_L-th child: we use the *KEEP* function in line 10 of Algorithm 12 to keep the images that correspond to cluster j. We compute recursively the hierarchical partitioning $currCluster.hp$ with U, $index$, P and $currCluster.child$. If the current cluster does not have enough data points (is smaller than N_L) then the cluster will be empty.

Several trees are created using Algorithm 12 and the dataset $\mathcal{D}$ is computed for each tree by random sampling with replacement. The Sensing Forest algorithm is sketched in Figure 3.4. In Figure 3.5 an example of a forest is shown with the leaves carrying the class probability distributions.

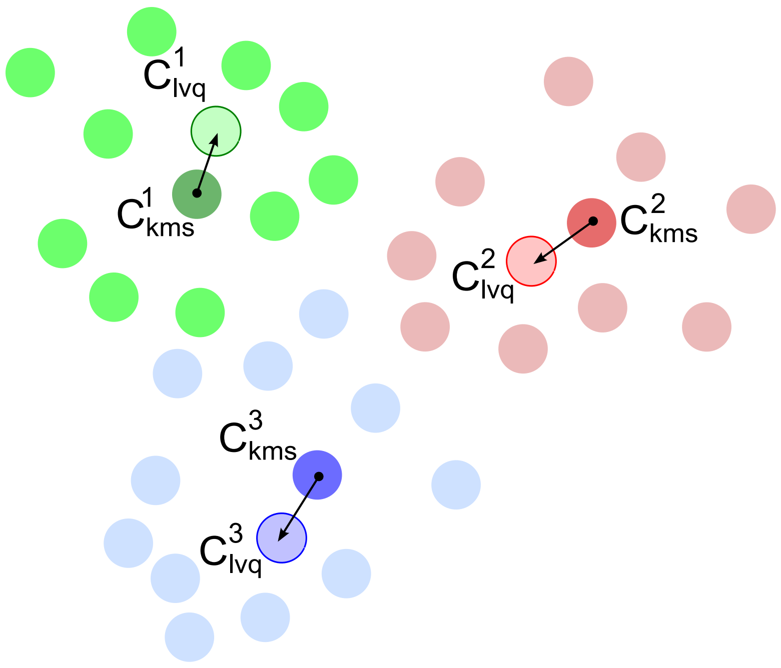
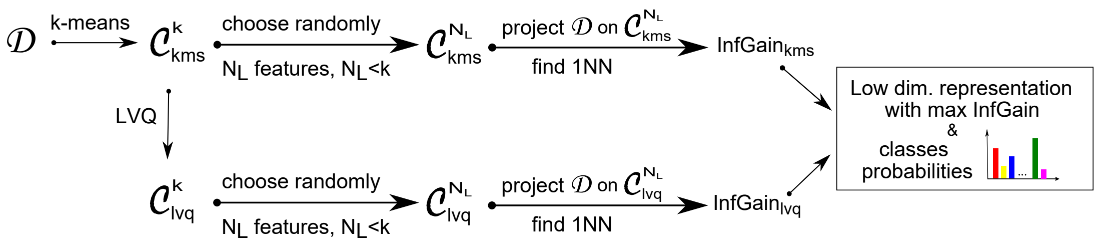


Figure 3.3: Example of a codebook learned with k-means and finely tuned with LVQ. Three different classes are represented with three different colors and a prototype for each class. The initial prototypes are given by the 3 centroids computed with k-means: C^1_{kms}, C^2_{kms}, and C^3_{kms}. The initial codebook is further adapted by using LVQ: C^1_{lvq}, C^2_{lvq}, and C^3_{lvq}. The prototypes which maximize the split of the data are further used for representation as explained in Algorithm 14.



$\mathcal{D}$
k-means
$\mathcal{C}^k_{kms}$
choose randomly
N_L features, N_L<k
$\mathcal{C}^{N_L}_{kms}$
project $\mathcal{D}$ on $\mathcal{C}^{N_L}_{kms}$
find 1NN
$InfGain_{kms}$
LVQ
$\mathcal{C}^k_{lvq}$
choose randomly
N_L features, N_L<k
$\mathcal{C}^{N_L}_{lvq}$
project $\mathcal{D}$ on $\mathcal{C}^{N_L}_{lvq}$
find 1NN
$InfGain_{lvq}$
Low dim. representation with max InfGain & classes probabilities

Figure 3.4: The scheme applied at each node of a tree for learning the SF. Given a dataset $\mathcal{D}$, a feature representation is learned by using the centroids obtained with k-means, $C_{kms} = \mathbf{C}^k_{kms}$. k is given by the number of classes from the current node. N_L features are randomly chosen from the k-dimensional representation into $U_{kms} = \mathbf{C}^{N_L}_{kms}$. For binary trees, as it is the case here, $N_L = 2$. The data $\mathcal{D}$ corresponding to the current node of the tree is projected on the N_L-dimensional representation and the nearest centroid is considered. The information gain corresponding to the split with N_L centroids as prototypes is computed. Simultaneously a LVQ fine tuning of the initial prototypes $\mathbf{C}^k_{kms}$ obtained with k-means is done. $\mathbf{C}_{lvq} = \mathbf{C}^k_{lvq}$ is obtained. Afterwards the same steps are applied and the information gain $InfGain_{lvq}$ corresponding to the LVQ split is computed. For the considered node the split which maximizes the information gain is considered and the distributions of classes are stored. This scheme is applied as we go deeper into the tree until reaching the leaves.

Notation	Description
$currCluster$	current cluster
$labelsCurrCluster$	corresponding labels for $currCluster$
L	current level of the tree
k	number of clusters for k-means
N_L	number of feature vectors at each L
$kMEANSIG$	computes k-Means with information gain maximization and chooses randomly $N_L < k$ centroids
$LVQIG$	computes LVQ with maximum information gain
$codebook$	centroids and corresponding labels (from $kMEANSIG$) used for initializing the prototypes in LVQ
C_{kms}, $labelsC_{kms}$, C_{lvq}	k prototypes (with labels) learned with k-means/LVQ $size(C_{kms}) = size(C_{lvq}) = (D \times k)$
U_{kms}, U_{lvq}, U	matrix containing the feature vectors learned with k-Means/LVQ, $size(U_{kms}) = size(U_{lvq}) = (D \times N_L)$
$index_{kms}$, $index_{lvq}$, $index$	$index = \overline{1, N_L}$, each I_i in $currCluster$ belongs to cluster $index$; the same for $index_{kms}$ and $index_{lvq}$
α_i	learning rate used for *LVQ* algorithm
$P, ClassesProbabilities$	class probability distribution
IG_{kms}, IG_{lvq}	gained information by splitting data with k-means/LVQ
$currCluster.child$	contains the children of the current cluster node
$currCluster.hp$	contains the hierarchical partitioning and stores: U, $index$, and P for each node of the tree
x_{test}	test data point, $size(x_{\text{test}}) = 1 \times D$
$nearNeigh$	nearest neighbor (from the learned prototypes) of the projected data on the learned representation
$ClassProbabLeaf_{x_{\text{test}}}$	corresponding probability class distribution for x_{test} in the leaf of the learned tree on which it was projected

Table 3.6: Notations for Algorithms 12, 13, 14, and 15.

Algorithm 12 Learning the Sensing Forest (SF)

```
1: function Partitioning($currCluster$, $labelsCurrCl, N_L$)
2:     if $size(currCluster) > N_L$ then
3:         $k \leftarrow$ NrOfClasses(currCluster)
4:         $[C_{kms}, labelsC_{kms}, U_{kms}, index_{kms}, IG_{kms}] \leftarrow$
               $\leftarrow$ kMeansIG($currCluster$, $labelsCurrCl$, $k$, $N_L$)
5:         $codebook \leftarrow [C_{kms}, labelsC_{kms}]$
6:         $[C_{lvq}, U_{lvq}, index_{lvq}, IG_{lvq}] \leftarrow$
               $\leftarrow$ lvqIG($currCluster$, $labelsCurrCl$, $k$, $N_L$, $codebook$)
7:         $[U, index] \leftarrow$ Max($IG_{kms}, IG_{lvq}$)
8:         $P \leftarrow$ ClassesProbabilities($labelsCurrCl$, $index$)
9:         for $j = 1 : N_L$ do
10:            $currCluster.child(j) \leftarrow$ Keep($currCluster, index, j$)
11:            Partitioning($currCluster.child(j)$, $labelsCurrCl$, $N_L$)
12:        end for
13:        $currCluster.hp \leftarrow \{U,\ index,\ P\}$
14:    else
15:        $currCluster.hp \leftarrow \varnothing$
16:    end if
17: return $currCluster$
18: end function
```

Algorithm 13 Learning feature vectors using k-means algorithm

```
1: function kMeansIG($currCluster$, $labelsCurrCl$, $k$, $N_L$)
2:     $[C_{kms}, labelsC_{kms}] \leftarrow$ kMeans($currCluster$, $labelsCurrCl$, $k$)
3:     $U_{kms} \leftarrow$ Random($C_{kms}, N_L$)
4:     $index_{kms} \leftarrow$ Find1NN($U_{kms}^T \cdot U_{kms}, currCluster \cdot U_{kms}$)
5:     $IG_{kms} \leftarrow$ InfoGain($index_{kms}$)
6: return $C_{kms}, labelsC_{kms}, U_{kms}, index_{kms}, IG_{kms}$
7: end function
```

3.4.2 Sensing novel scenes

The SF uses the hierarchical partitioning of the dataset, learned as described in the previous subsection, for sensing. Therefore, an unknown scene, i.e., a test point x_{test} outside $\mathcal{D}$, which we wish to sense and classify, is successively projected on each of the learned trees. Algorithm 15 presents the steps of the procedure. The corresponding notations are also described in Table 3.6.

The test point is first projected on the $rootCluster$ in line 2 of the algorithm. x_{test} is projected repetitively on the cluster which contains the nearest prototype of the projected test point on the learned representation. The function in Algorithm 15 returns the class distribution probabilities

Algorithm 14 Learning prototypes for the Sensing Forest (SF) using LVQ

```
 1: function lvqIG($currCluster$, $labelsCurrCl$, $k$, $N_L$, $codebook$)
 2:     for $each\, \alpha_i$ do
 3:         $C_i \leftarrow$ LVQ($currCluster, labelsCurrCl, codebook, \alpha_i$ )
 4:         $U_i \leftarrow$ Random($C_i, N_L$)
 5:         $nearNeigh_i \leftarrow$ Find1NN($U_i^T \cdot U_i, currCluster \cdot U_i$)
 6:         $IG_i \leftarrow$ InfoGain($nearNeigh_i$)
 7:     end for
 8:     $[IG_{lvq}, idxIG] \leftarrow$ Max($IG$)
 9:     $index_{lvq} \leftarrow nearNeigh_{idxIG}$
10:     $C_{lvq} \leftarrow C_{idxIG}$
11:     $U_{lvq} \leftarrow U_{idxIG}$
12: return $C_{lvq}, U_{lvq}, index_{lvq}, IG_{lvq}$
13: end function
```

Algorithm 15 Sensing novel scenes using the learned trees

```
 1: function ProjectTestData($rootCluster$, $x_{\text{test}}$)
 2:     $currCluster \leftarrow rootCluster$
 3:     while $currCluster \neq \varnothing$ do
 4:         $[U,\, index,\, P] \leftarrow currCluster.hp$
 5:         $nearNeigh \leftarrow$ Find1NN($U^T \cdot U, x_{\text{test}} \cdot U$)
 6:         $clusterNN \leftarrow index(nearNeigh)$
 7:         $currCluster \leftarrow currCluster.child(clusterNN)$
 8:     end while
 9:     $ClassProbabLeaf_{x_{\text{test}}} = P$
10: return $ClassProbabLeaf_{x_{\text{test}}}$
11: end function
```

corresponding to the leaf where x_{test} was projected, $ClassProbabLeaf_{x_{\text{test}}}$. The final prediction for x_{test} is given by summing up $ClassProbabLeaf_{x_{\text{test}}}$ over several trees and taking the class with maximum votes:

$$max\left(\sum_{treeNr=1}^{maxTreeNr} ClassProbabLeaf_{x_{\text{test}}}^{treeNr}\right).$$

Class distribution probabilities could be stored at different levels such that decisions could be made earlier or later in the sensing process. In Figure 3.6 a novel data point x_{test} is sensed by using the learned SF example showed in Figure 3.5.

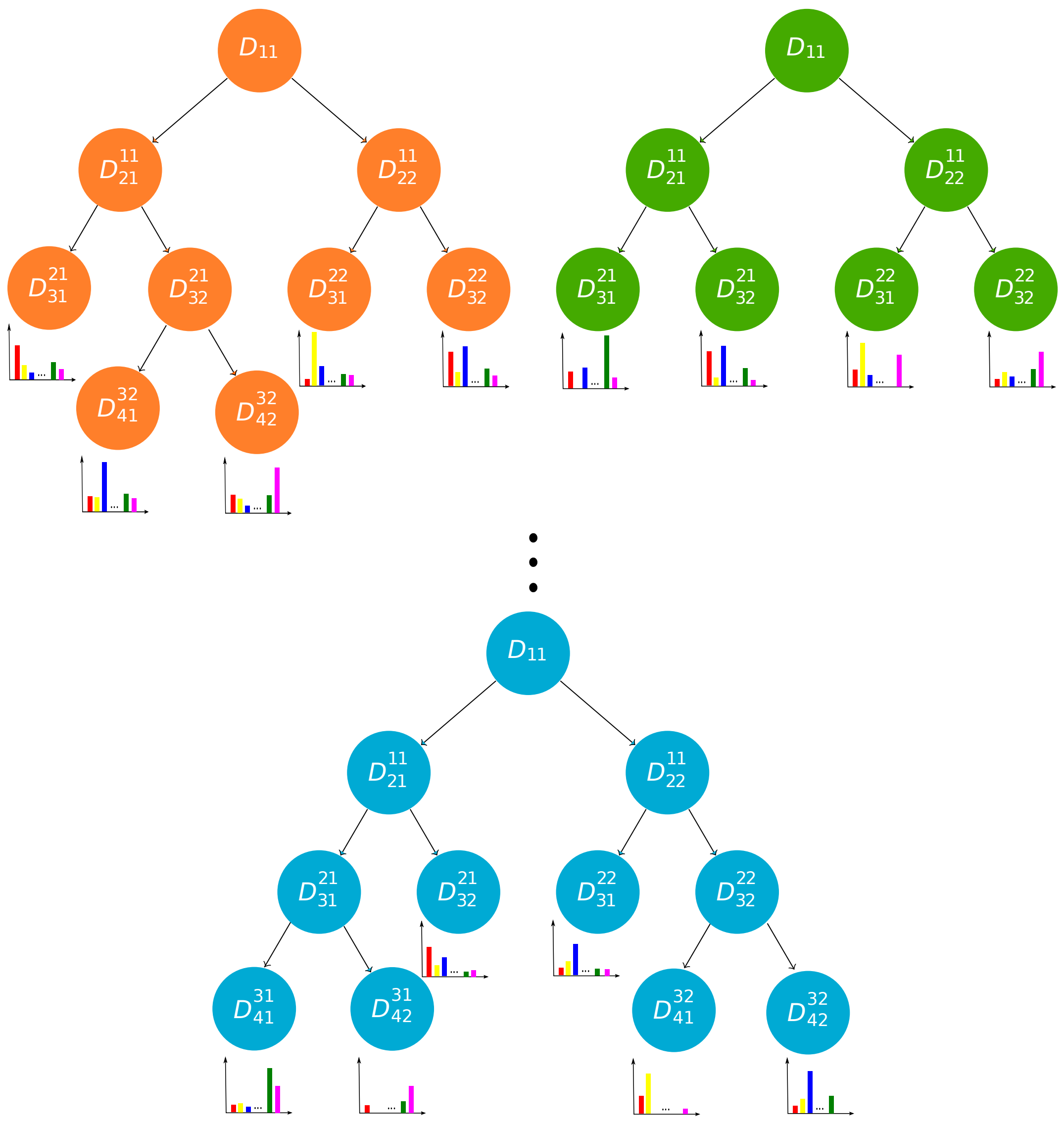


Figure 3.5: Sensing Forest (SF) example which consists of several trees carrying in the leaves the class probability distributions. Each tree contains in the root randomly chosen data from the whole dataset $\mathcal{D}$ in D_{11}. Different colors were chosen for the trees to emphasize that the trees are different, even though the data in the root has the same notation, i.e., D_{11}. At each level the data is split using the prototypes which maximize the information, either by using k-means, or LVQ algorithms. The split is done as long as there are enough data points in the subclusters. In this example the maximum level reached is $L = 4$. In the leaves the class probability distributions are stored and will be further used for classifying a new data point as shown in Figure 3.6.

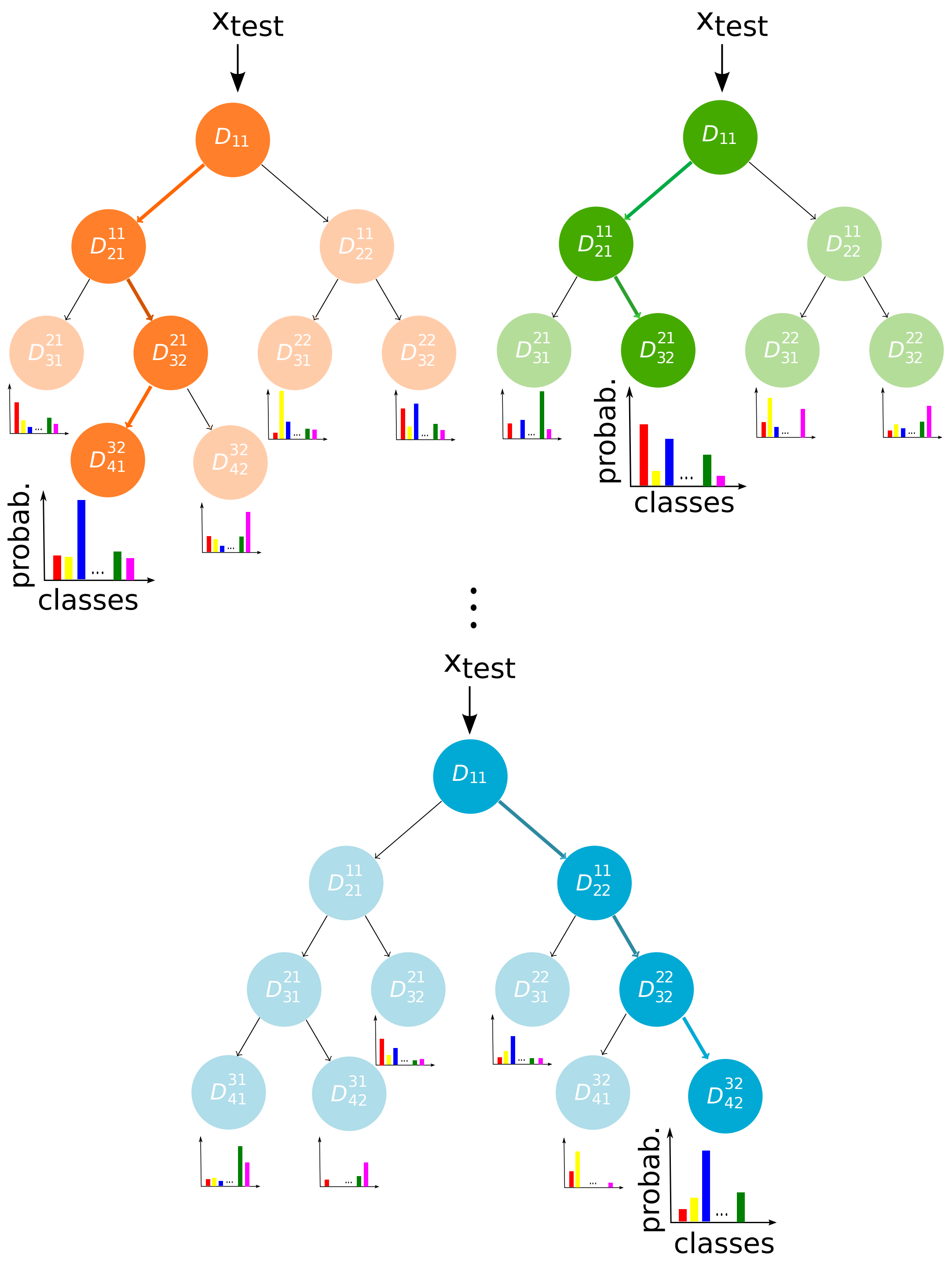


Figure 3.6: The trees learned using the Sensing Forest (SF) in the example from Figure 3.5 are used for sensing a novel data point x_{test}. The test data point is traversing each tree deciding at each level either to go on the right or left branch. The decision is made based on the nearest prototype of the test data. The process is repeated until x_{test} reaches the leaves which have stored the class probabilities. The class of x_{test} is given by the class with the most votes.

4
Results for efficient sensing

In this chapter we present the results obtained for the efficient sensing methods which we introduced in the previous chapter: Visual Manifold Sensing (VMS), Foveated Manifold Sensing (FMS) with the corresponding extended version Hybrid Foveated Manifold Sensing (HyFMS), Hierarchical Manifold Sensing (HMS) with foveation and adaptive partitioning of the dataset, and Sensing Forest (SF).

4.1 Visual Manifold Sensing

The Visual Manifold Sensing (VMS) approach was described in Chapter 3.1. The corresponding algorithm was presented in Algorithm 5.

VMS is based on an online learning algorithm such that while sensing a novel scene, the data is continuously adapted and the corresponding manifold is learned (either by using LLE or PCA). The adaptation of the dataset is done based on the nearest neighbors of the projected new scene on the learned low-dimensional representation.

4.1.1 Experiments

We worked under the assumption that the goal of sensing is to acquire information for a particular task and not for representing the world. We therefore applied the VMS method for both face recognition and object recognition. We evaluated the performance of VMS on two benchmarks: one for face-recognition, UMIST [66], and one for the recognition of everyday objects, ALOI (Amsterdam Library of Object Images) [96]. The UMIST database with faces contains twenty different persons in different poses and a total of 1000 images of size 256×256 pixel. We worked with images resized at 128×128 pixels. We considered from the ALOI database a subset of images of the first twenty objects. These are everyday objects images with different rotation angles resulting in a total of 1400 images of size 192×144 pixel.

We measured the performance of VMS by computing: (i) the Signal to Noise Ratio (SNR) between the test images and the corresponding nearest images in dataset $\mathcal{D}$ and (ii) the recognition rate. Regarding the SNR, note that the selection of the nearest image is based on the distances in the low-dimensional embedding but the actual SNR is computed in the original image space as a measure for how much information is retained in the low-dimensional embedding. The recognition rate is based on a classifier that assigns each low-dimensional representation y^l_{test} (l-th element of Y_{test}, the projected test data on the low-dimensional manifold) to the class of the nearest y^j (j-th element of the low-dimensional representation Y of the data points in $\mathcal{D}$) in the low-dimensional dataset.

We compared VMS with: (i) Principal Component Analysis (Non-iterative PCA): new data points are projected on the considered principal components of dataset $\mathcal{D}$, and (ii) Iterative PCA presented in Chapter 3.1 Algorithm 6.

VMS does not have any formal criterion for how to proceed from lower- to higher-dimensional manifolds, and for how to choose the number of neighbors r for the LLE algorithm and the size q of the adaptive dataset. In order to evaluate VMS, we consider a vector of N_i components which corresponds to the number of dimensions of the learned manifolds. Each component of the vector tells us how many n_i iterations we have to compute for each dimension N_i. For simplicity, we call this vector representation a combination. Figure 4.1 shows a particular case of a combination with five components. For each iteration we compute a different number of iterations, n_i, $i = 1, \ldots, 5$.

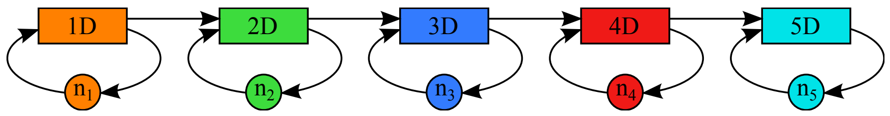


Figure 4.1: Graphical representation of the different combinations used to evaluate VMS: we iterate n_1 times in the dimension $D = 1$, then n_2 times in the dimension $D = 2$, and so on up to $D = 5$.

In our simulations all the combinations have five components (dimensions) and for each dimension we compute maximum five iterations, $N_{\text{max}} = 5$. We define n as the total number of iterations, $\sum n_i$, with n_i the number of iterations in dimension N_i, $N_i = 1 : N_{\text{max}}$. The total number of measurements is $M = \sum N_i \cdot n_i$.

As an example, we consider the combination 0 0 1 3 5 presented in Table 4.1. Thus, the following steps are computed: first we compute one iteration in the 3rd dimension, we consider the result obtained after the first step and we go on to the 4th dimension and we compute one iteration. We proceed in the same way with the result obtained after the second step and we compute another iteration in the same dimension, the 4th dimension. We go on and compute another iteration in the same dimension, the 4th dimension. With the result obtained before we move to the 5th dimension and step by step we compute $n_5 = 5$ iterations.

N_i (dimensions)	5
$N_{\max}$ (maximum iterations)	5
n_i (number of iterations in dimension N_i)	$n_1 = 0, n_2 = 0, n_3 = 1, n_4 = 3, n_5 = 5$
n (total number of iterations)	$0 + 0 + 1 + 3 + 5 = 9$
M (number of measurements)	$1 \cdot 0 + 2 \cdot 0 + 3 \cdot 1 + 4 \cdot 3 + 5 \cdot 5 = 40$

Table 4.1: Example for the combination 0 0 1 3 5

To explore the potential of VMS we applied the VMS algorithm on different combinations using different values for r starting from 30 to 80 neighbors with steps of 10. The decreasing size of the transferred neighborhoods, q, is an internal parameter of the iterative VMS algorithm. It is defined by Equation (3.1.3) in such a way that the size of the adaptive data set will always be larger than the number of neighbors used in the LLE algorithm.

4.1.2 Learned sensing basis functions

VMS projects an unknown scene on the learned manifold by using the pseudoinverse matrix and defines a neighborhood on the manifold, which is taken as the new dataset. Each projection is a scalar product between the reflectivity values in the scene and the weights that are learned in the different dimensions. These scalar products represent the sensing values used.

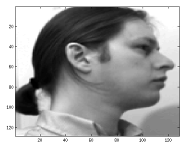
Figure 4.2 shows how the pseudoinverse matrix evolved after each iteration computed with the VMS method for the image test from Figure 4.2 (a) from UMIST database. The pseudoinverse matrices correspond to learning low-dimensional manifolds according to the combination 02123 by using 40 neighbors for the LLE algorithm.

The shown images represent the basis in which VMS is actually sensing. As we continue sensing, the templates evolute from rather random to more specific templates as it can be seen in Figure 4.2 (b).

4.1.3 Performance of Visual Manifold Sensing vs. Principal Component Analysis and iterative Principal Component Analysis

We evaluated VMS by computing (i) the Signal to Noise Ratio (SNR) between the test image and the image which is the nearest neighbor on the learned manifold, and (ii) the recognition rate (test images assigned to the class of the nearest neighbor) on the two mentioned databases, UMIST and ALOI. We compared VMS with the classical PCA and with the iterative PCA described in Chapter 3.1 in Algorithm 6. We also mention the number of sensing values (measurements) used in order to solve the recognition task.

(a) Test image from UMIST database [66]



(b) VMS sensing basis functions

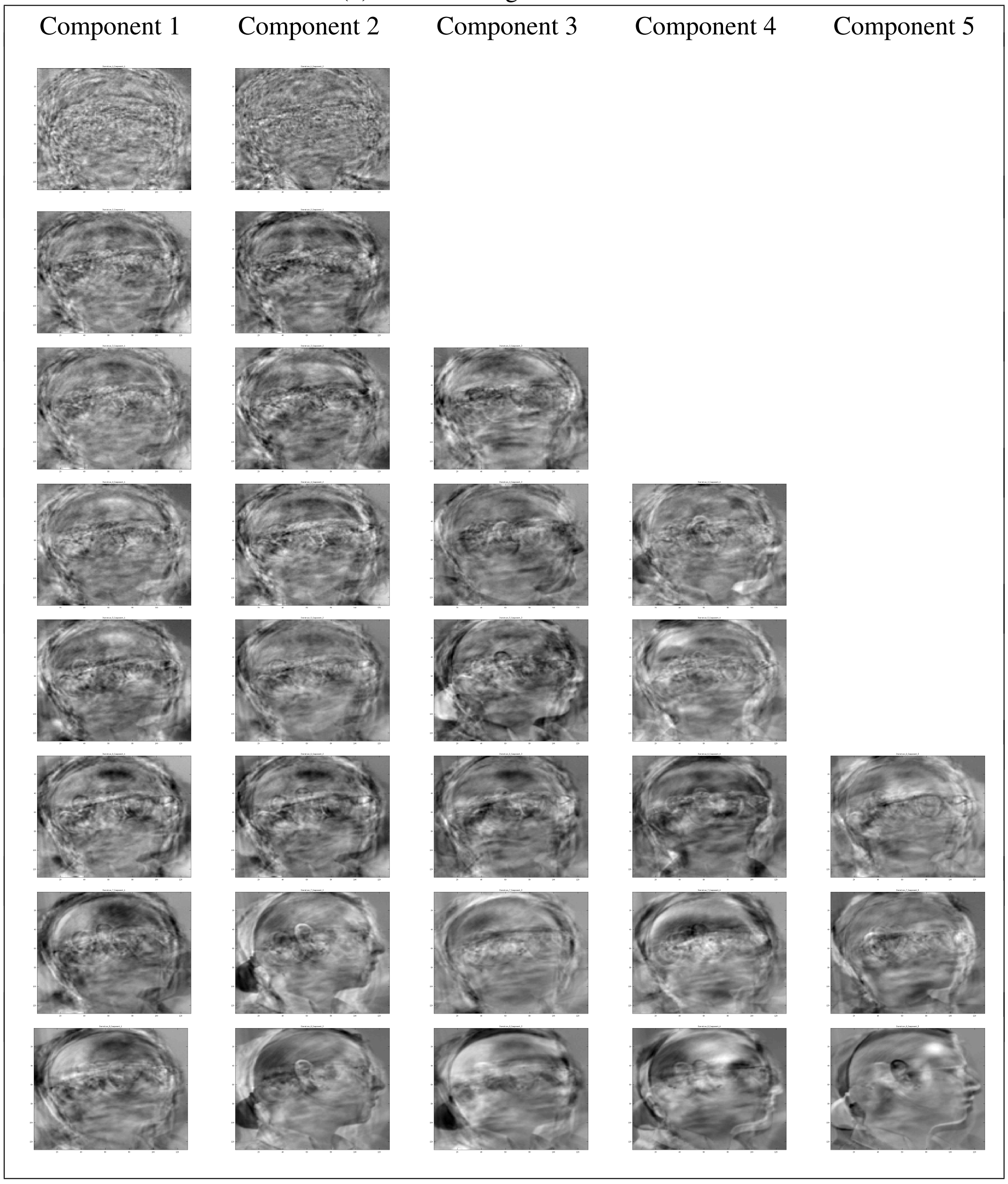


Figure 4.2: (a) Image test UMIST [66]. (b) VMS sensing basis functions (sbf) corresponding to the test image (a). These images represent the basis in which VMS is actually sensing, i.e., the components of the learned pseudoinverse matrices with VMS. The shown basis is obtained for the combination 02123 with 40 neighbors. Each row shows the components (in descending order of the singular values) of the pseudoinverse matrix for each iteration. Note the evolution from rather random to more specific templates (projection axes).

We present first the results we obtained on the UMIST database. By using VMS with only 30 measurements, i.e., a compression ratio greater than 2000, we obtained an average SNR over 20 test images (one per class) of 22.70 dB (the best possible average SNR for the database, when the correct nearest neighbor was identified in all cases, was 22.73 dB). On the same data, PCA with 30 components yielded an average SNR of 22.60 dB. The recognition rate for VMS and PCA with 30 measurements was 100%, and for Iterative PCA 85%. We obtained this result for the combination 0 2 1 2 3 with 40 neighbors. For the combination 0 0 1 0 0, i.e., with only 3 measurements, the recognition rate was 75%. One could argue that human vision employs a gist of a scene as explained in Chapter 2.1.2. In this context, it seems particularly striking that acceptable face recognition is possible with only three measurements, i.e., with only three samples of the visual world.

We also evaluated VMS for object recognition on the ALOI database. With only 38 measurements, i.e., a compression ratio greater than 700, we obtained an average SNR of 15.39 dB (in this case the best possible SNR was 15.49 dB, PCA with 38 components yielded 15.15 dB) and a 100% recognition rate.

The results we obtained by applying VMS on the two different benchmarks for object and face recognition are summarized in Table 4.2. r is the number of neighbors used by the LLE algorithm used in VMS Algorithm 5. M represents the number of sensing values. The compression ratio is defined by the ratio between the original image size and the number of sensing values used for recognition. For each benchmark we mentioned the combination for which we got the best result and we also compared the average SNR obtained for VMS with the values obtained for PCA and with the best possible SNR for the respective benchmark.

Benchmark	Combination	r	Average SNR (dB)	Recognition rate	M	Compression ratio
UMIST (256×256)	0 2 1 2 3	40	VMS: 22.70 dB PCA: 22.60 dB Best possible: 22.73 dB	100 %	30	> 2000
ALOI (192×144)	0 0 3 1 5	80	VMS: 15.39 dB PCA: 15.15 dB Best possible: 15.49 dB	100 %	38	> 700

Table 4.2: Best results obtained by applying VMS for UMIST [66] and ALOI [96] benchmarks.

A detailed comparison between VMS, Iterative PCA, and Non-iterative PCA for face recognition is shown in Figure 4.3, and respectively, for object recognition in Figure 4.4. As it can be seen in both figures, the VMS method yielded better results than both, PCA and Iterative PCA.

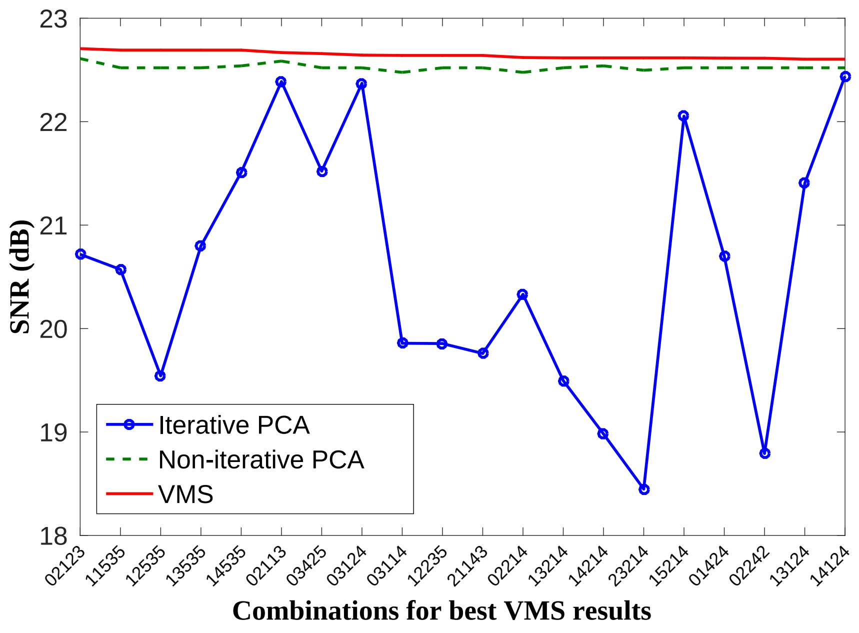


Figure 4.3: VMS vs. PCA and iterative PCA for UMIST database [66].

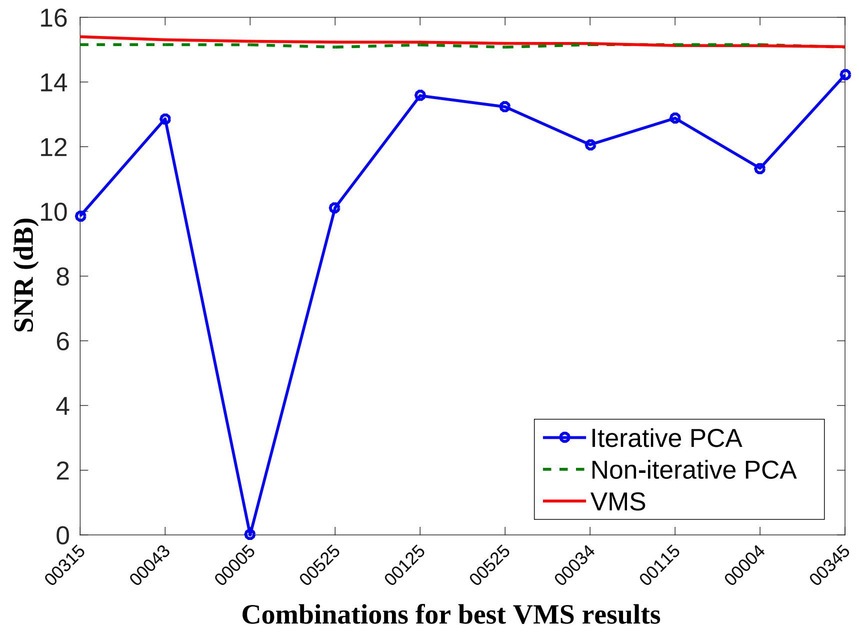


Figure 4.4: VMS vs. PCA and iterative PCA for ALOI database [96].

The performance of VMS was evaluated on two benchmarks: for face recognition (UMIST [66]) and for everyday objects recognition (ALOI [96]). Thus, the information gathered during sensing was quantified by the recognition performance that it enabled. Perfect recognition was possible for UMIST with 30 measurements, and for ALOI with 38 sensing values. Moreover, the distance to the nearest neighbor in the dataset, a measure that corresponds to a reconstruction error, was also evaluated. Compared to PCA and iterative PCA, VMS yielded better results.

4.2 Foveated Manifold Sensing

The Foveated Manifold Sensing (FMS) approach and its extended version, Hybrid Foveated Manifold Sensing (HyFMS), were described in Chapter 3.2 and in Chapter 3.2.3. The corresponding algorithms were presented in Algorithm 8, respectively in Algorithm 9.

FMS is an extension of the Visual Manifold Sensing (VMS) method such that FMS involves foveation. Therefore, FMS operates only on the foveated dataset which is created as presented in Algorithm 7, rather than on the initial dataset (as VMS does). HyFMS performs first on the initial dataset, and afterwards continues learning manifolds on the foveated dataset. Therefore, HyFMS starts with VMS and continues with FMS.

4.2.1 Experiments

We evaluated FMS and HyFMS on the same databases as VMS: UMIST database [66] and Amsterdam Library of Object Images (ALOI) [96]. We evaluated both methods in terms of the recognition rate while using as few sensing actions as possible. Furthermore, we compared FMS and HyFMS with VMS.

4.2.2 Foveation

The FMS and HyFMS methods involve foveation. Thus, a foveated dataset is created as presented in Algorithm 7. In the following we illustrate the steps used for creating the foveated dataset for HyFMS for the example test image shown in Figure 4.5.

Figure 4.5: Example image from the UMIST database [66].

Geometric invariants K^i

Figure 4.6 shows the geometric invariant K^i (in this case four images are shown, $i = 1, \ldots, 4$) for different images from the UMIST dataset. The selected images correspond to the nearest neighbor of the projected test image from Figure 4.5 as we proceed with HyFMS to sense the test image. Thus, Figure 4.6 (a) depicts the K^i of the nearest neighbor on the learned manifold after only 1 sensing action. With 3 sensing actions, i.e., after the second iteration, Figure 4.6 (b) is the new nearest neighbor and so on.

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

Figure 4.6: Geometric invariants K^i of the nearest neighbors of the projected test image in Figure 4.5 after (a) 1 sensing action, (b) after 3, (c) after 6 and (d) after 10 sensing actions.

Regions of interest

Figure 4.7 shows the regions of interest T^i that correspond to the same images as in Figure 4.6. In this case the regions of interest retain only 13% of the pixels in an image. Note that the regions of interest change as we acquire more sensing values. In Figure 4.7 (a) we can see that the regions of interest are focused mainly on the contour of the head and after the 10-th sensing action the region of interest moves also on the face.

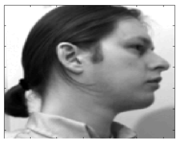
(a) **(b)** **(c)** **(d)**

Figure 4.7: Regions of interest T^i corresponding to the images shown in Figure 4.6.

4.2.3 Learned sensing basis functions

FMS projects an unknown scene on the learned manifold by using the pseudoinverse matrix and defines a neighborhood on the manifold, which is taken as the new dataset. Each projection is a scalar product between the reflectivity values in the scene and the weights that are learned in the different dimensions. These scalar products represent the sensing values used. Figure 4.8 (b) shows how the pseudoinverse matrix evolved after each adaptation step of FMS for the test image in Figure 4.8 (a). Similarly, Figure 4.9 (b) shows how the pseudoinverse matrix evolved after each adaptation step of HyFMS for the same test image. The FMS sensing basis functions concentrate only on the contour of the head, while in the case of HyFMS they are also defined on the face of the test image.

(a) Test image from UMIST database [66]



(b) FMS sensing basis functions

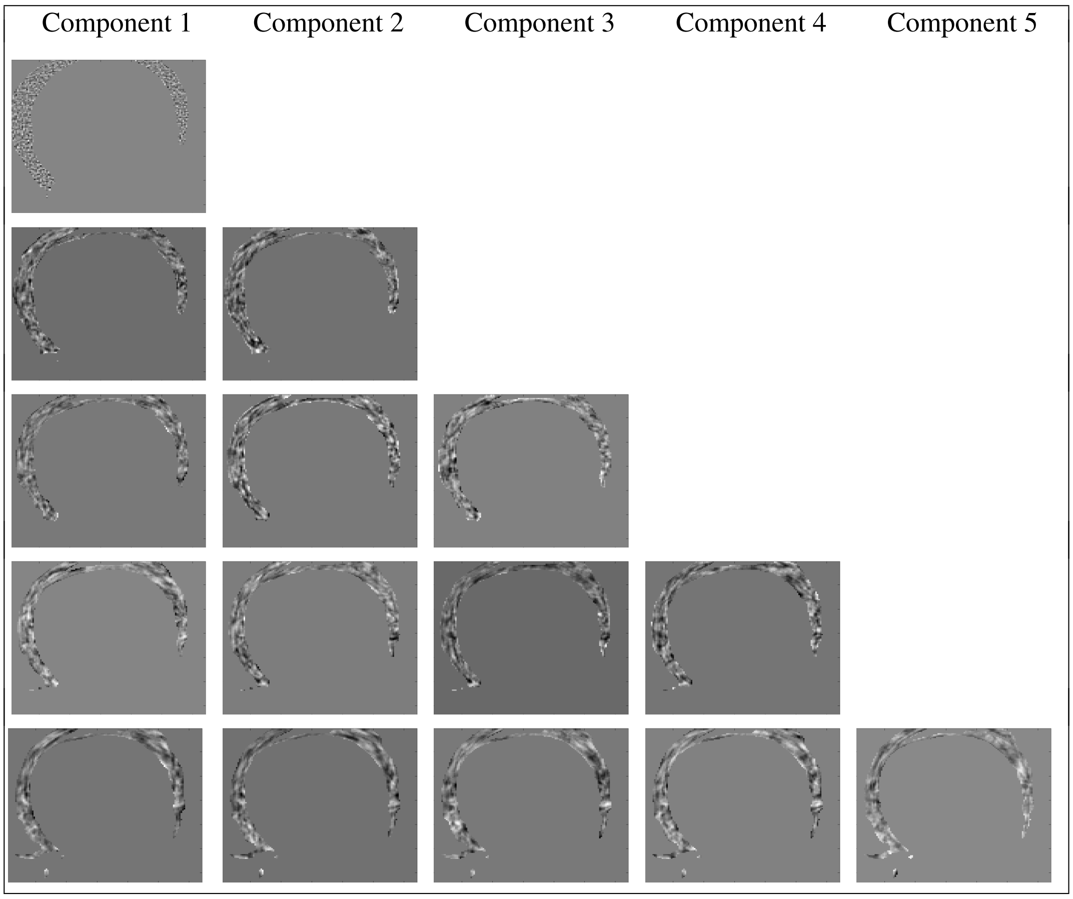
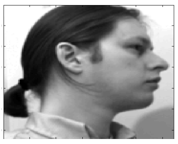


Figure 4.8: (a) Selected test image. (b) FMS sensing basis functions corresponding to the shown test image. These images represent the basis in which FMS is actually sensing, i.e., the components of the learned pseudoinverse matrices with FMS. The shown basis is obtained for an initial dimension of the learned manifold equal to one, which is afterwards increased in steps of one. The process is repeated five times, such that the maximum dimension of the manifold is five. Each row shows the components (in descending order of the singular values) of the pseudoinverse matrix for each iteration. The templates have a region of interest of 13%. Note that the templates concentrate on the contour of the head.

(a) Test image from UMIST database [66]



(b) HyFMS sensing basis functions

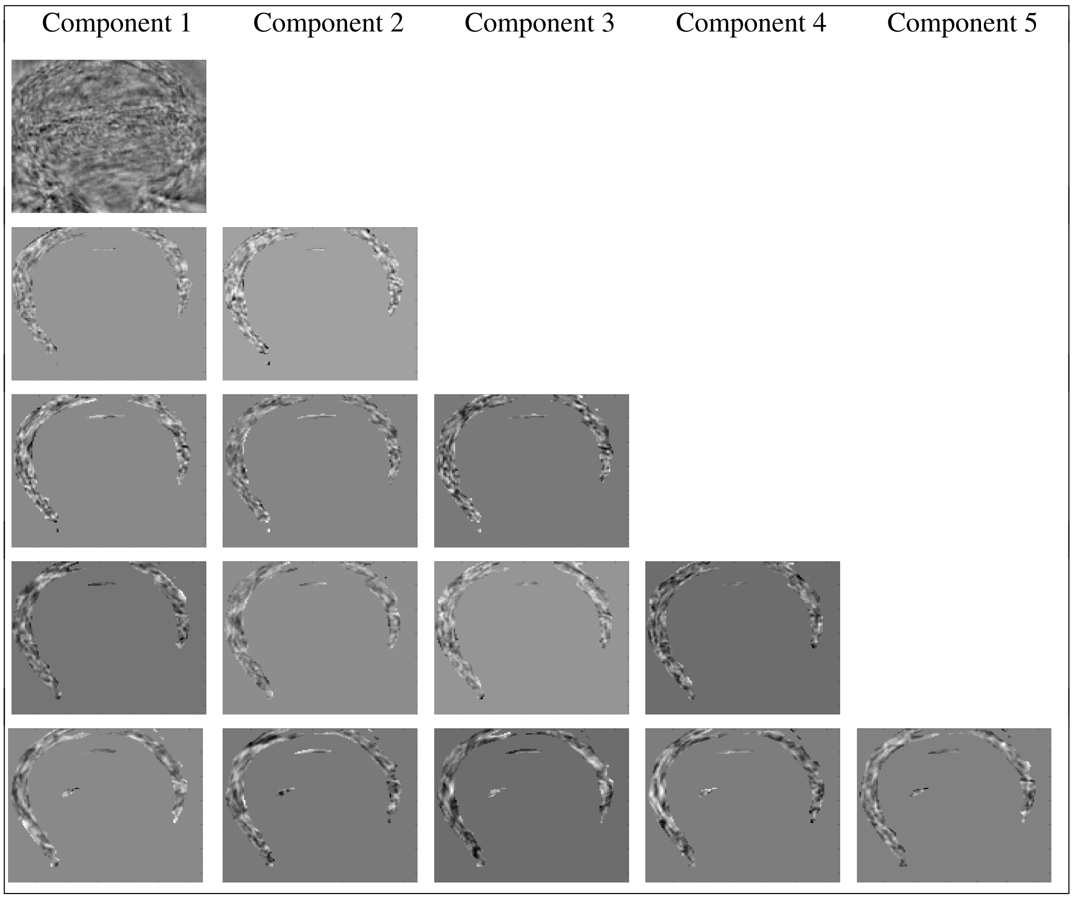


Figure 4.9: (a) Selected test image. (b) HyFMS sensing basis functions corresponding to the shown test image. These images represent the basis in which HyFMS is actually sensing, i.e., the components of the learned pseudoinverse matrices with HyFMS. The shown basis is obtained for an initial dimension of the learned manifold equal to one and the dimension increases in steps of one. The process is repeated five times, such that the maximum dimension of the manifold is five. Each row shows the components (in descending order of the singular values) of the pseudoinverse matrix for each iteration. The first template is the whole image and afterwards the templates have a region of interest of 13%. Note that initially the templates concentrate on the contour of the head and as we continue sensing, the templates are defined also on the face of the test image.

FMS and HyFMS sense only the most salient areas of an object compared to VMS which uses global sensing basis functions as shown in Figure 4.2.

4.2.4 Performance of Foveated Manifold Sensing vs. Visual Manifold Sensing vs. Hybrid Foveated Manifold Sensing

We evaluated all three online-learning methods, VMS, FMS, and HyFMS by computing the recognition rates (test images assigned to the class of the nearest neighbor). The goal is to use as few sensing actions as possible and still obtain the highest possible recognition rate. For each dataset we evaluated the performance by leaving one sample out. The results for the two datasets are presented in Table 4.3. The compression ratio is defined by the ratio between the original image size and the number of sensing values used for recognition. For simplicity, instead of evaluating the performance for different combinations, as in VMS, we here learn a low-dimensional representation in the first dimension, and afterwards we increase at each step the dimension with one, until we reach the maximum dimension $N_{\text{max}} = 5$.

Benchmark	Method	Region of interest	Person/Object recognition rate	Sensing values	Compression ratio
UMIST	VMS	100 %	100 %	15	> 3000
	FMS	13 %	75 %	15	> 3000
	HyFMS	13 %	100 %	10	> 5000
ALOI	VMS	100 %	100 %	15	> 1800
	FMS	37 %	65 %	15	> 1800
	HyFMS	37 %	100 %	10	> 2700

Table 4.3: Results (average recognition rates) obtained by applying VMS, FMS, and HyFMS for UMIST [66] and ALOI [96] benchmarks.

As shown in Table 4.3, with the VMS method we obtained on both the UMIST face recognition benchmark and the ALOI object recognition benchmark a recognition rate of 100% with 15 sensing values (compression ratios > 3000 and > 1800 respectively). Compared to VMS and FMS, HyFMS requires fewer sensing actions: with HyFMS a recognition rate of 100% is obtained with only 10 sensing values (compression ratio > 5000) for UMIST. The threshold θ for the region of interest (ROI) was chosen such as to obtain an average size of the regions of interest of 13%. On the ALOI database we again obtained a 100% recognition rate with only 10 sensing values. However, here the resulting optimal size of the region of interest was larger, namely 37%. The optimal size of the region of interest was obtained by evaluating the results of HyFMS for different sizes of the region of interest (from 1% to 100% in steps of 1) and then choosing the (smallest) size that would give maximum performance over the training set.

The evaluations on FMS and HyFMS have shown that a few sensing actions are sufficient in order to solve a face- or an object-recognition task. We also showed that a sensing strategy which starts with VMS and then continues with FMS is even more efficient. Therefore, the interaction of a sensing agent with the environment can proceed from first obtaining a gist of the scene, based on very few sensing values, and then acquiring more refined samples until the task at hand can be solved. Compared to Compressive Sensing (CS) which uses a random sensing matrix, FMS and HyFMS employ a matrix which is adapted to the dataset, and furthermore, is also sparse since it only senses the most salient areas of the face of a person, or an object. The sparse sensing matrices distinguish FMS and HyFMS also from VMS, which uses adapted but global sensing basis functions. HyFMS requires even fewer sensing actions compared to VMS.

4.3 Hierarchical Manifold Sensing with foveation and adaptive partitioning of the dataset

The Hierarchical Manifold Sensing (HMS) approach was described in Chapter 3.3. The corresponding algorithm for the hierarchical partitioning of the data was presented in Algorithm 10. Sensing unknown data is done by following Algorithm 11.

HMS learns an adequate hierarchical partitioning of the data by using PCA and k-means clustering. Learning is therefore done offline and the adapted dataset with the corresponding learned embedding (at each level of the hierarchical partitioning) can be directly accessed while sensing a novel scene. Thus, HMS, by using offline learning, overcomes the limitation of the previous approaches, VMS, FMS, and HyFMS which are based on an online learning algorithm, i.e., while sensing a novel scene, the dataset is continuously adapted and the corresponding embedding is learned.

4.3.1 Experiments

We evaluated HMS on Amsterdam Library of Object Images (ALOI) [96] and on Columbia Object Image Library (COIL-20) [97] database. The COIL-20 database contains 1440 gray scale images of 20 objects with 72 images for each object. The images have 128×128 pixels and were taken at object-pose intervals of 5 degrees. In our experiments we used from the ALOI database 50 classes at a quarter resolution 192×144. We also evaluated HMS with and without foveation on the MNIST [74] benchmark which consists of handwritten digits from 0 to 9. There are 60000 images for training and 10000 for testing.

In order to evaluate the presented HMS method with foveation and adapted data partitioning we divided the datasets into training and test data. We computed the recognition rates for the test data by assigning to each test image the corresponding class of the nearest neighbor. For each dataset we chose randomly one image per class and we tested them against the other images that belong to the training dataset.

The goal of HMS, as of all sensing strategies proposed here, is to use as few sensing actions as possible and still obtain the highest possible recognition rate. Therefore, we searched for the minimum number of sensing actions that HMS needs to perform in order to achieve the highest possible recognition rate for all the tested images.

4.3.2 Learned sensing basis functions

Each projection is a scalar product between the reflectivity values in the scene and the weights that are learned in the different dimensions by using the hierarchical partitioning of the data. These scalar products represent the sensing values used. Figure 4.10 shows a selection of 7 out of 28 HMS sensing basis functions with a ROI of 8% (first row), 16% (second row) and without foveation (third row). The shown results were obtained on the ALOI database with 20 classes. Note that the basis functions for the two different ROIs are specific to the test image shown right as each new basis function depends on the previously acquired sensing values. Thus, the basis functions evolve, as we continue sensing, from rather generic to more specific templates. It can also be seen that the ROIs adapt accordingly. In comparison, the third row of Figure 4.10 shows the corresponding selection of basis function for the case that the hierarchical partitioning was computed for the original dataset and not for the foveated dataset as shown before. Without foveation, the basis functions do also adapt during the hierarchical partitioning but the adaption is less specific.

Figure 4.11 shows a selection of 5 sensing basis functions for MNIST database obtained with a hierarchical partitioning with $k = 2$ and $N_1 = 9$. In the first column sample test images with the corresponding number and class are shown and on each line the corresponding sensing basis functions for a different number of sensing values, i.e., for $L = 1$, $L = 3$, $L = 7$, $L = 8$, and for $L = 9$. Note the evolution from rather generic to more specific templates.

4.3.3 Performance of Hierarchical Manifold Sensing with and without foveation vs. Random Projections

We explored the benefits of the presented approach by comparing HMS, with and without foveation, with the classical Compressive Sensing (CS) method, which uses Random Projections, i.e., a random Gaussian matrix with rows that have unit length and a smaller number of components. In order to do this, we considered the simplest configuration for the hierarchical partitioning of the data: $k = 2$ clusters and the dimension N_L for the first level of the tree equal to 1. We computed the recognition rate for differently sized regions of interest, e.g., 16% for a foveated dataset and up to 100% for the original dataset. For both HMS and CS we used the first nearest neighbor classifier.

Figure 4.12 shows, for the database ALOI with 20 classes, how the recognition rate depends on

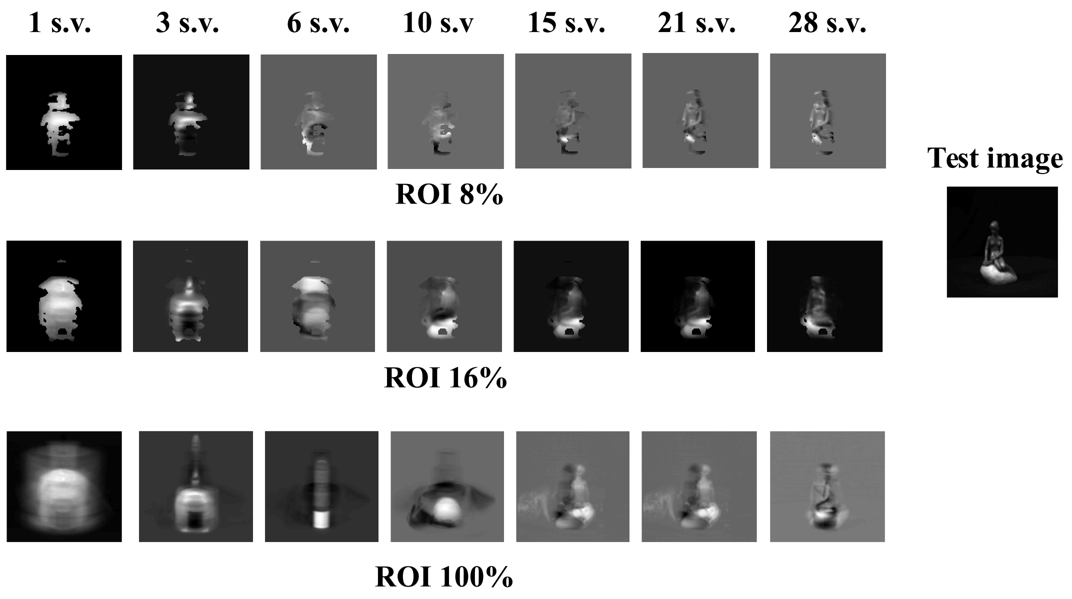


Figure 4.10: Selected HMS sensing basis functions (7 out of 28 sensing values) with a ROI of 8% (1st row), 16% (2nd row) for the foveated dataset, without foveation (ROI of 100% - 3rd row) and the corresponding test image from ALOI [96] with 20 classes. For the hierarchical partitioning we used $k = 2$ clusters and $N_L = 1$ for $L = 1$. Note that with foveation, as we continue sensing, the basis functions evolve from rather generic to more specific templates which resemble the test image.

the region of interest, i.e., the number of salient pixels. The curves are plotted for three different numbers (1, 3, and 6) of sensing values, i.e., for $L = 1$, $L = 2$, and $L = 3$ respectively. For $L = 1$, HMS senses with a sensing matrix of dimension $(N_1 \times D)$, where $N_1 = 1$, i.e., it takes only one sensing value. For $L = 2$, HMS senses with a sensing matrix of $(2 \times D)$ which adds up to $1+2 = 3$ sensing values, and so on. Note that for $L = 1$, where only one sensing value is available, foveation deteriorates the result. However, already with 3 and 6 sensing values, the recognition performance does not increase with the number of pixels that are considered, i.e., performance is equally high with a ROI of only 5% of the image.

In Figures 4.13 and 4.14 we present representative results with foveation (ROI $= 16\%$) and without foveation (ROI $= 100\%$) for different benchmarks: COIL with 20 classes and ALOI with 20 and 40 classes. We compare the results of HMS with the results obtained by using a Random Projections matrix for sensing with the corresponding number of sensing values. For all methods we computed 100 runs and we present in Figures 4.13 and 4.14 the average of the recognition rate over these runs. As it can be seen in Figure 4.14 (a)-(b), on the ALOI database with 20 and 40 classes and COIL-20 database (Figure 4.13) we are able to reach a recognition rate of 100% with a region of interest of only 16% and 6 sensing values, which corresponds to a compression ratio greater than 4500 in the case of ALOI and greater than 2500 for COIL. The compression ratio is defined

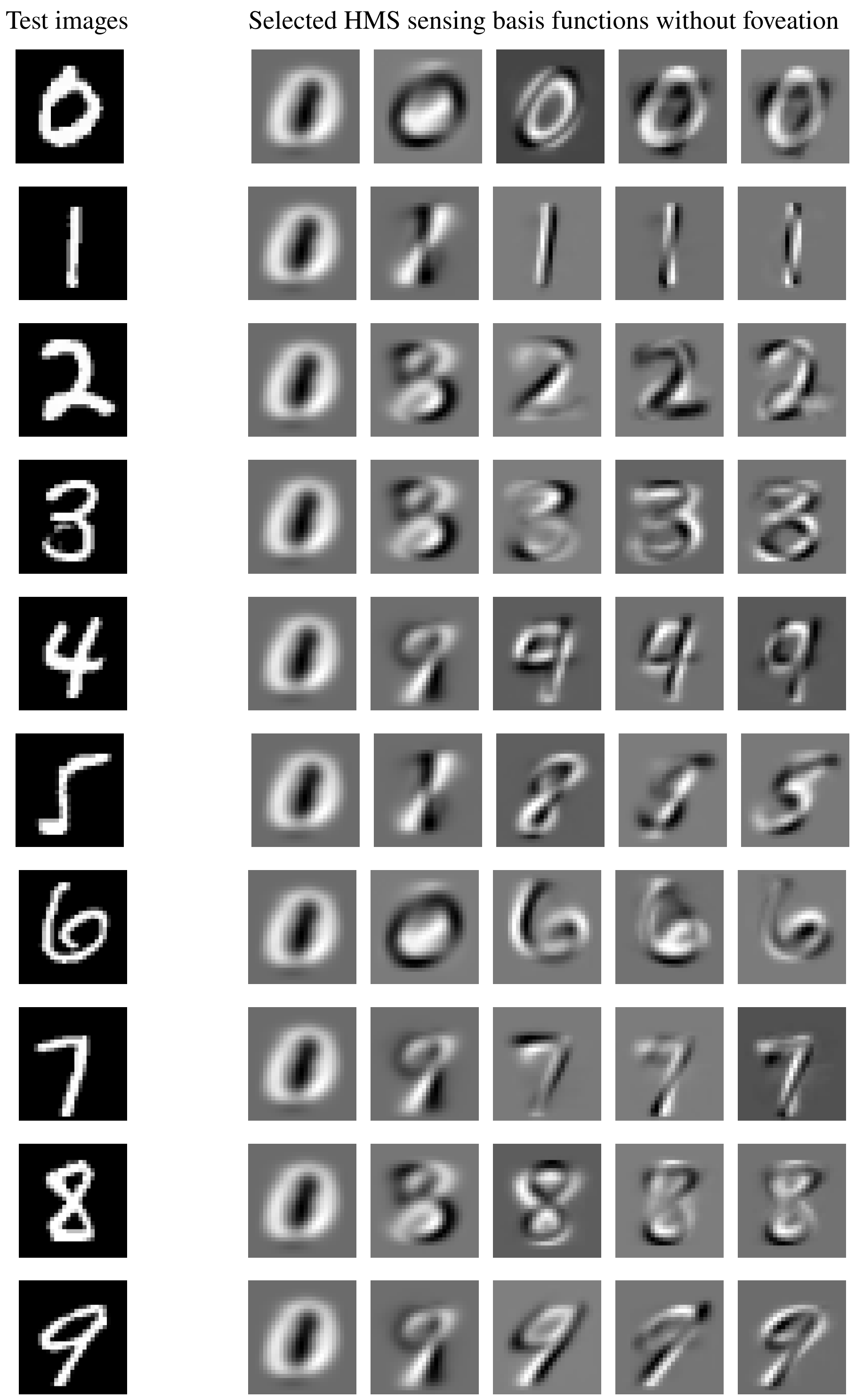


Figure 4.11: Selected HMS sensing basis functions without foveation (2nd to 5th column - for $L = 1, 3, 7, 8, 9$) and the corresponding sample test image (1st column) from MNIST dataset. For the hierarchical partitioning we used $k = 2$ clusters and $N_L = 9$ for $L = 1$.

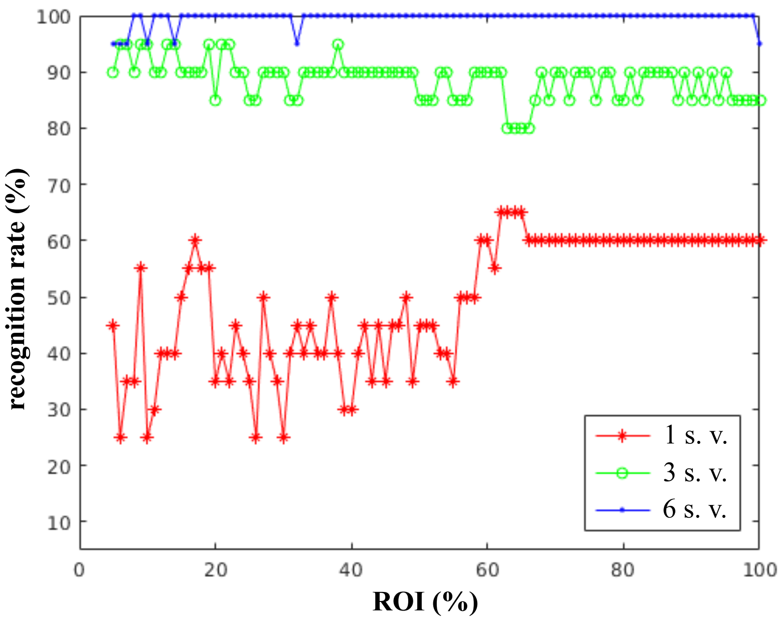


Figure 4.12: HMS results for the ALOI 20 database. Recognition rates are shown for different regions of interest with 1, 3, and 6 sensing values respectively. The hierarchical partitioning is done with $k = 2$ and $N_L = 1$ for $L = 1$.

by the ratio between the original size of the images of the respective dataset (number of pixels) and the number of sensing values used for recognition. Note that the recognition performance of HMS is higher than the recognition rate obtained with the CS Random Projections approach on the considered databases.

We conclude that for small databases, as COIL and ALOI with 20 and 40 classes, it is enough to use the HMS algorithm with a simple configuration: a number of 2 clusters and $N_L = 1$ for $L = 1$. As the database contains more training images, e.g., ALOI with 50 classes, it is worth to study the evolution of HMS for different hierarchical trees focusing on the different number of principal components of the first level of the tree, N_1, and for a different number of clusters k. We show in Figure 4.15 the results obtained with HMS for $N_1 = 1$, 2, and 3 in the case of $k = 2$ (a), and $k = 3$ (b). As expected, HMS performs better with a higher number of principal components for the first level of the hierarchical partitioning and with a proper k considering the number of data points in the training dataset.

We also evaluated the algorithm on the highly competitive MNIST [74] benchmark. We first considered the simple configuration for partitioning the data with $N_L = 1$ for $L = 1$ and only $k = 2$ clusters. Although the overall performance of a sensing and recognition scheme with, for example, $L = 12$ is limited to a recognition rate of 93.14%, it is interesting to note that of the

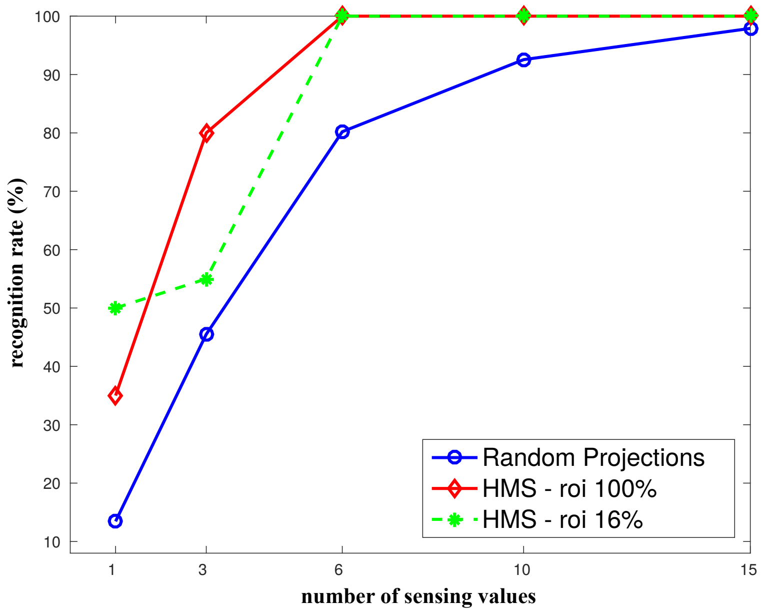
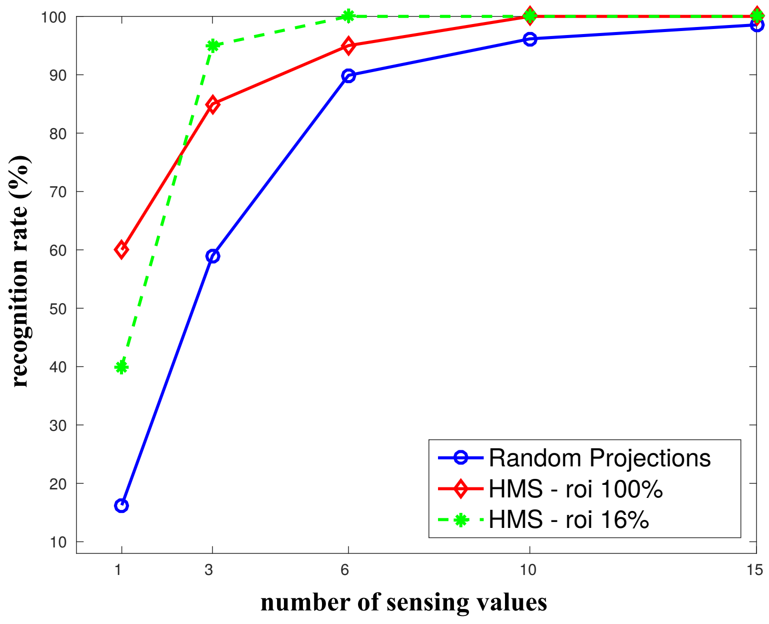


Figure 4.13: Representative results of HMS with and without foveation vs. Random Projections for the COIL-20 database.

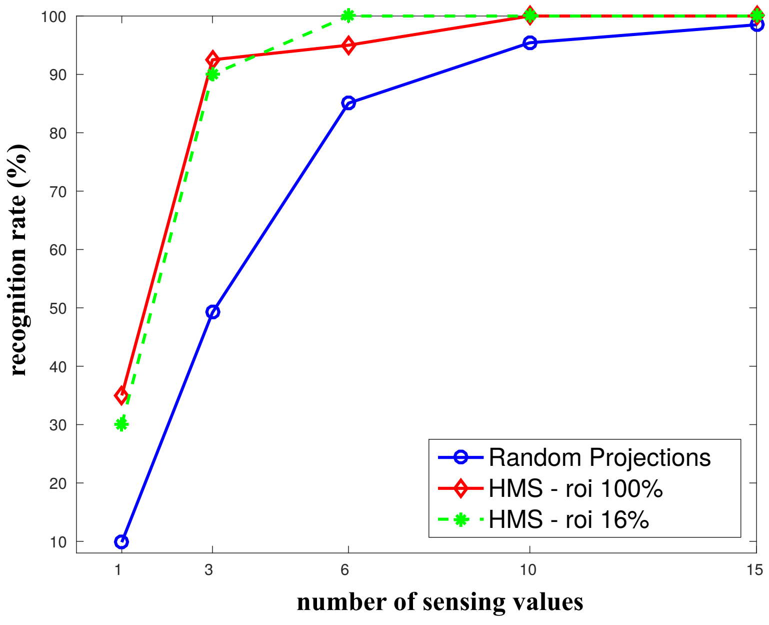
10000 test images 2491 are already correctly recognized with only 1 sensing action ($L = 1$). Of the remaining test images, 2436 are correctly classified with $L = 2$, 2689 of the remaining with $L = 3$, and 1290 of the the remaining with $L = 4$. If this scheme is continued up to $L = 12$ and $L = 13$, a total of 98.50% and 98.51%, respectively, of the test images are correctly classified. The difference between 98.50% and 93.14% at $L = 12$ is due to the fact that a few images are obviously misclassified with more sensing values although they would have been correctly classified with fewer.

We explored the performance of HMS on the MNIST dataset for different hierarchical partitionings of the training dataset, i.e., with different values of k and N_1. As shown before, for the previously considered databases, the recognition rate grows with N_1. We show in Figure 4.16 the performance of HMS for different values of N_1 in the case of (a) $k = 2$ and (b) $k = 3$ clusters. The curves are plotted for different numbers of sensing values, i.e., for $L = 1$, $L = 2$, $\ldots$ and $L = 9$ in Figure 4.16 (a), and in Figure 4.16 (b) for $L = 1$, $L = 2$, $\ldots$ and $L = 6$. As it can be seen in Figure 4.16 (a) for $k = 2$ and $N_1 = 20$ we reach a recognition rate of 96.69% for $L = 3$, i.e., with 63 sensing values and for $L = 4$, we reach a higher recognition rate of 96.82%.

If we accumulate the sensing values over the different levels of the sensing tree the recognition rate improves as shown in Figure 4.16. With $k = 2$ and $N_1 = 20$ HMS reaches a recognition rate

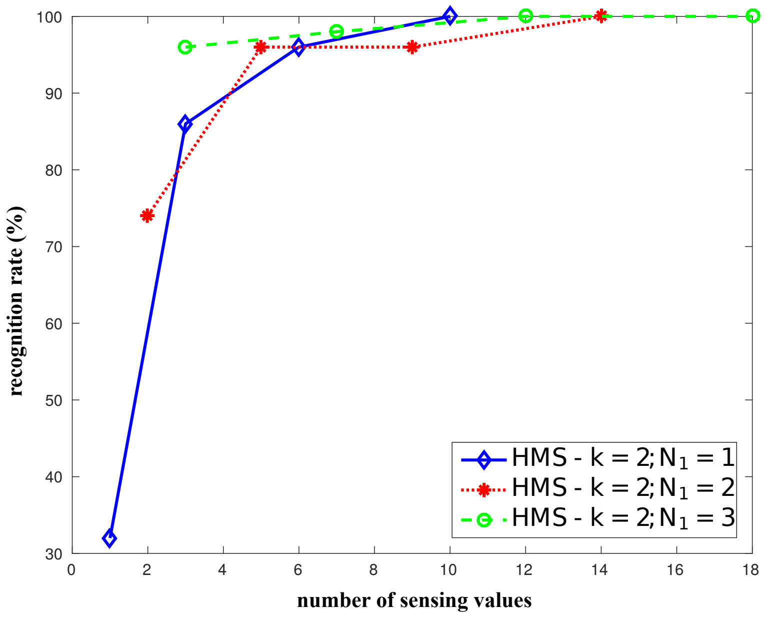


(a) ALOI 20 classes

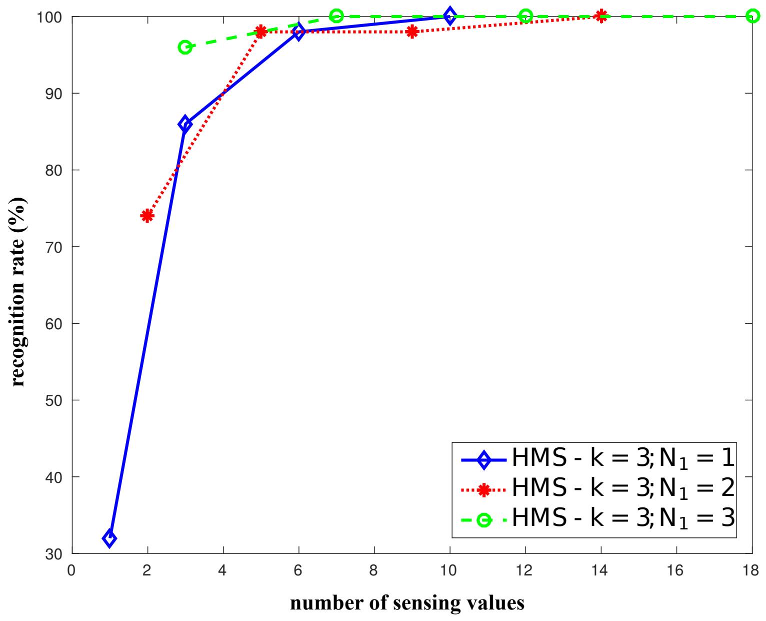


(b) ALOI 40 classes

Figure 4.14: Representative results of HMS with and without foveation vs. Random Projections for the ALOI database with (a) 20, and (b) 40 classes.



(a) $k = 2$



(b) $k = 3$

Figure 4.15: Results of HMS for different hierarchical partitionings of the training data; results obtained for ALOI with 50 classes for (a) 2 clusters, respectively (b) 3 clusters.

of 96.88% with 63 sensing values and 96.93% with 86 sensing values.

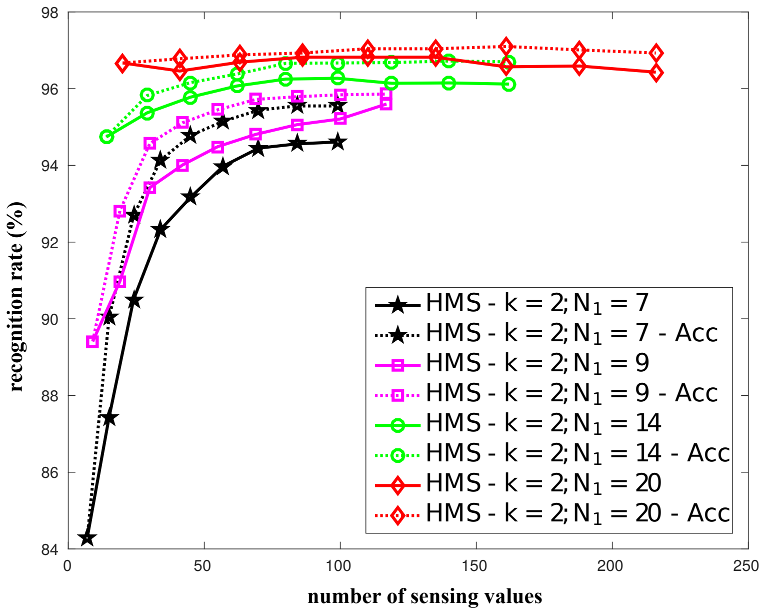
In Figure 4.17 we show the results for HMS compared to CS. The parameters for HMS are $N_1 = 20$ and $k = 2$ and we computed only 10 runs. Note that HMS outperforms CS. The region of interest of 23% seems not to contain enough salient pixels in order to reach a higher performance than HMS without foveation as in the case of the other tested databases.

4.3.4 Performance of Hierarchical Manifold Sensing vs. Visual Manifold Sensing and Foveated Manifold Sensing

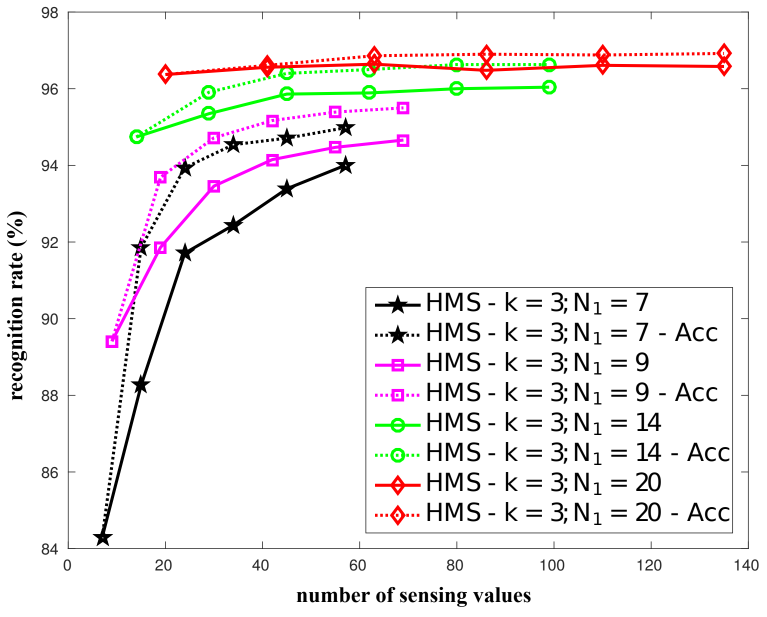
We showed in Table 4.3 that for the ALOI database with 20 classes, VMS needs 15 sensing values in order to reach 100% recognition rate. Whereas FMS reaches only 65% recognition rate with 15 sensing values. By using HMS with foveation we showed that for a 100% recognition rate only 6 sensing values are needed and 10 sensing values are needed for HMS without foveation. Moreover, VMS and FMS strongly depend on the number of neighbors selected for the Locally Linear Embedding (LLE) algorithm used to learn the manifolds, on the decreasing size of the adaptive dataset, and on the dimension of the manifolds at each iteration of the algorithm. Furthermore, for HMS the partitioning of the dataset is done prior to sensing, and not during the actual sensing as with VMS, FMS, and HyFMS.

4.3.5 Performance of Hierarchical Manifold Sensing vs. state of the art

In 2014 Dornaika et al. [98] developed a semi-supervised feature extraction with an out of sample extension algorithm which they applied on a subset of COIL-20 (18 images from 72 available for each object) database. They randomly selected 50% of the data as training dataset and the rest as test dataset. From the training dataset they randomly labeled 1, 2 and 3 samples per class and the rest of the data was used as unlabeled data. The data is first preprocessed: Principal Component Analysis (PCA) is computed in order to preserve 98% energy of the dataset. The work [98] provides a comparison between methods which are based on label propagation and on graph-based semi-supervised embedding. They report the best average classification results on ten random splits for their method for 3 label samples and for unlabeled (80.4%) and test data (77.4%). They also show that when one labeled sample per class is used, their method reaches 61% recognition rate with around 19 feature dimension. In order to compare HMS with the approach proposed in [98] we divided the COIL dataset with 72 objects per class into a training and test in a similar way as described before. Although the training dataset consists only of 720 images, HMS performs better than the semi-supervised feature extraction algorithm in [98]. Thus, for a hierarchical partitioning of the training data with $k = 2$ and $N_1 = 1$, HMS reaches an average recognition rate (over ten random splits of the data) of 94.98% with 15 sensing values. If the partitioning is done with the same number of clusters but $N_1 = 2$, a higher recognition rate of 95.98% at $L = 5$ is reached with



(a) $k = 2$



(b) $k = 3$

Figure 4.16: Results of HMS on MNIST for a hierarchical partitioning of the training data with $N_1 = 7, 9, 14, 20$ and (a) $k = 2$, (b) $k = 3$, and, for using accumulated sensing values (Acc) over the different levels of the sensing tree.

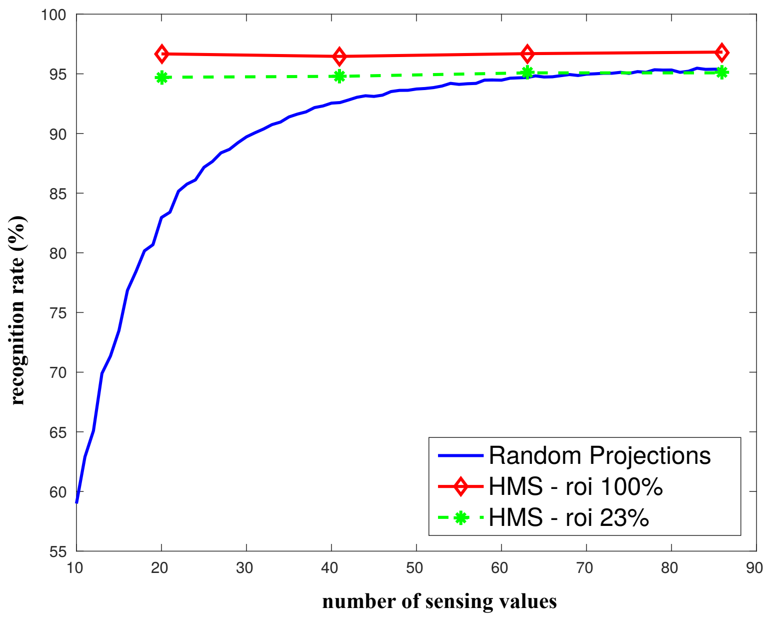


Figure 4.17: Results on MNIST for HMS vs. Random Projections.

20 sensing values.

A recent paper [99] presents an out-of-sample generalization algorithm for supervised manifold learning for classification which is evaluated on COIL-20 dataset. They use 71 images for each of the 20 objects in COIL which they normalize, convert to grayscale and downsample to a resolution of 32×32 pixels. The algorithm embeds the images in a 25-dimensional space. They obtain the minimum average misclassification rate over 5 runs approximately 2%. We compared HMS with the approach in [99] and we obtained an average misclassification rate with 10 sensing values of 4.2% and 100% recognition rate with 15 sensing values. For the hierarchical partitioning we used $k = 2$ and $N_1 = 1$. Thus, HMS reaches 100% recognition rate with even fewer sensing values than in [99].

State-of-the-art performance on COIL-20 database has been achieved in 2017 by a Compact Feature Representation algorithm (CRF-ELM) by using Extreme Learning Machine (ELM) under a shallow network framework [100]. The approach reaches a testing error of 3.91% by using a two layers network, eight filters, the size of patches is 7×7 and the max pooling size is 2×2.

The algorithm presented in [101] is evaluated on a subset of ALOI with 50 randomly chosen classes corresponding to different objects. The authors divide the dataset into training (contains odd views of each object) and test dataset (consists of even views). The recognition rate for each class is measured so we averaged all the rates and obtained a recognition rate of 92.88%. We evaluated

HMS for a similar training and test dataset and we obtained an average recognition rate of 99.5% with 21 sensing actions for a hierarchical partitioning with $k = 2$ and $N_1 = 1$. The recognition rate grows depending on N_1 such that with 15 sensing actions HMS reaches 99.5% ($N_1 = 2$), 99.72% ($N_1 = 3$) and maximum performance 100% ($N_1 = 7$).

On the MNIST database, a baseline method [74] which uses as input to a second degree polynomial classifier the 40 dimensional feature vector of the training data obtained with PCA has an error rate of 3.3% compared to our 3.32% with 41 sensing values and 3.12% with 63 sensing values as shown in Figure 4.16 (a).

State-of-the-art performance on MNIST has been achieved by a Recurrent Convolutional Neural Network (RCNN) approach which introduces recurrent connections into each convolutional layer [102]. The approach reaches a testing error of 0.31% and uses $670,000$ parameters. Our goal is to explore the HMS algorithm in terms of the best recognition rate reached with as few sensing actions as possible, rather than increasing complexity for maximum performance.

We have shown that on benchmarks such as COIL and ALOI, perfect recognition with Hierarchical Manifold Sensing (HMS) is possible with only about 10 sensing values. On harder benchmarks such as MNIST, state-of-the-art performance could not be reached, but we showed that a large number of test images could be recognized with very few sensing values only. Such performance resembles human performance, since humans can recognize without effort a multitude of objects based on just the gist of a scene and require scrutiny for less familiar objects and more difficult recognition tasks. As Foveated Manifold Sensing (FMS) and Hybrid Foveated Manifold Sensing (HyFMS), HMS involves foveation and we have shown that gist-like sensing and recognition requires the whole image, whereas more refined sensing can be reduced to only few salient locations without deteriorating recognition performance.

4.4 Sensing Forest

The Sensing Forest (SF) approach was described in Chapter 3.4. The corresponding algorithm for the hierarchical partitioning of the data was presented in Algorithm 12. Sensing novel scenes is done by using Algorithm 15.

The SF extends the Hierarchical Manifold Sensing (HMS) method from learning a hierarchical representation of the data (corresponding to a tree), to learning a forest (several trees) following the classical Random Forest (RF). Therefore, by learning several trees with bagging randomness, overfitting is reduced. HMS offers a high number of possibilities for choosing the parameters (dimension of the embedding, number of clusters) by using PCA for learning embeddings and k-means for clustering. Therefore, the SF uses directly k-means for learning the low-dimensional representation of the data, and uses only two clusters for splitting the data such that all the learned trees are binary. Moreover, the codebook with prototypes learned with k-means can be further improved by using

Learning Vector Quantization (LVQ) algorithms. Similarly to the classical RF, the features used for the SF are randomly chosen: from the features learned with k-means and LVQ, only a subset are randomly chosen and used afterwards for splitting the data in the nodes of the tree. Consequently, the SF is a prototype-based RF with prototypes learned with k-means and LVQ.

4.4.1 Experiments

We evaluated the SF on the Columbia Object Image Library (COIL-100) [97] and MNIST [74] database. COIL-100 contains color images of 100 objects with 72 images/object of size 128×128 pixels. We worked with gray-level images. Images of the objects were taken at pose intervals of 5 degrees.

In order to evaluate the presented SF method we divided the datasets into training and testing. We used 80% of the whole dataset for training and the rest for testing. For each tree created with Algorithm 12 we randomly chose 80% of the whole training data with replacement. Afterwards, we sensed the testing dataset using Algorithm 15 and we computed the corresponding recognition rates by assigning to each test data the class with the maximum votes. The goal of the SF is to use as few sensing values as possible with the highest possible recognition rate. We compared the performance of our SF with the performance of a state-of-the-art Random Forest (RF).

4.4.2 Learned sensing basis functions

Figure 4.18 shows a selection of six sensing basis functions for MNIST and COIL-100 learned with the SF. The sensing basis functions were obtained for the sample test images shown in the first row of the figure. The number of the sensing values increases from the second row to the last row and the templates evolve from rather generic to more specific ones. Sensing basis functions from two different trees are shown in the columns for each test image. The sensing basis functions are actually the components of the codebook which are learned at each level of each tree, i.e., the centroids learned with either k-means, or the finely tuned prototypes learned with Learning Vector Quantization (LVQ). The best codebook which leads to the best split of the data is chosen such that the information gain is maximized.

Note that the sensing basis functions learned with the SF on MNIST look similar to the feature maps learned with Convolutional Neural Networks (CNNs). Examples of feature maps learned with different CNN approaches are shown in [103], [104].

4.4.3 Sensing Forest vs. Random Forest

We explored the benefits of the presented approach by comparing it with Random Forest (RF) in the classification tasks. We computed the recognition rate for both algorithms. We also determined

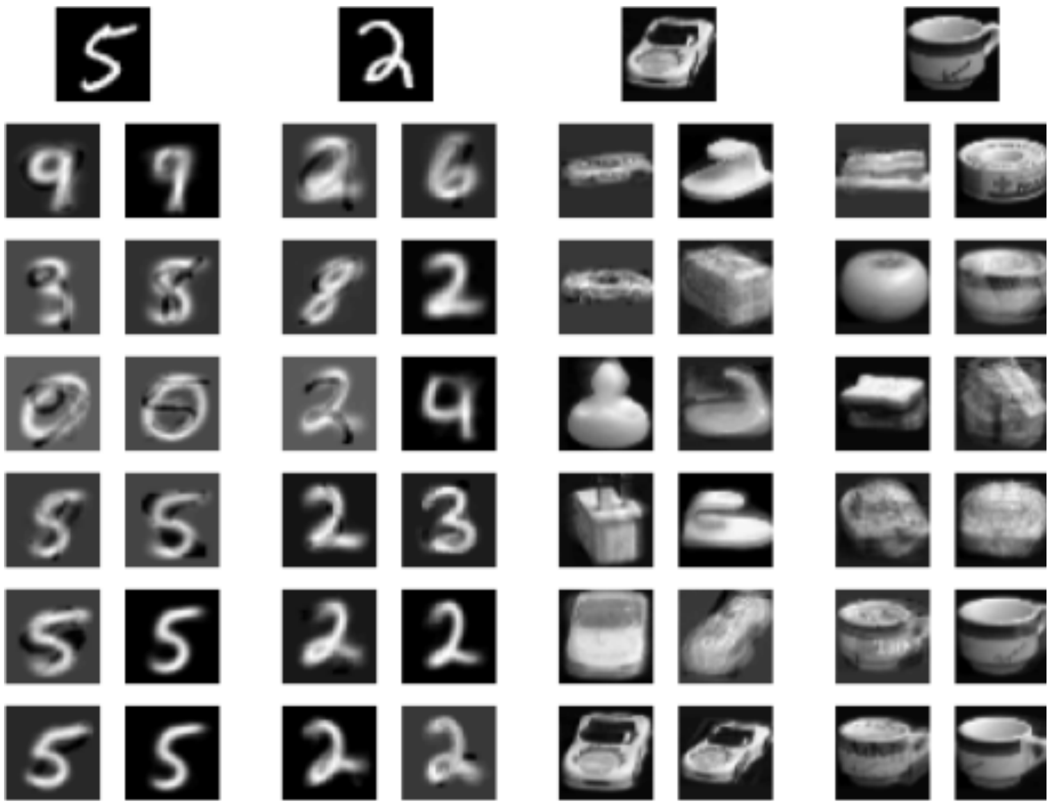
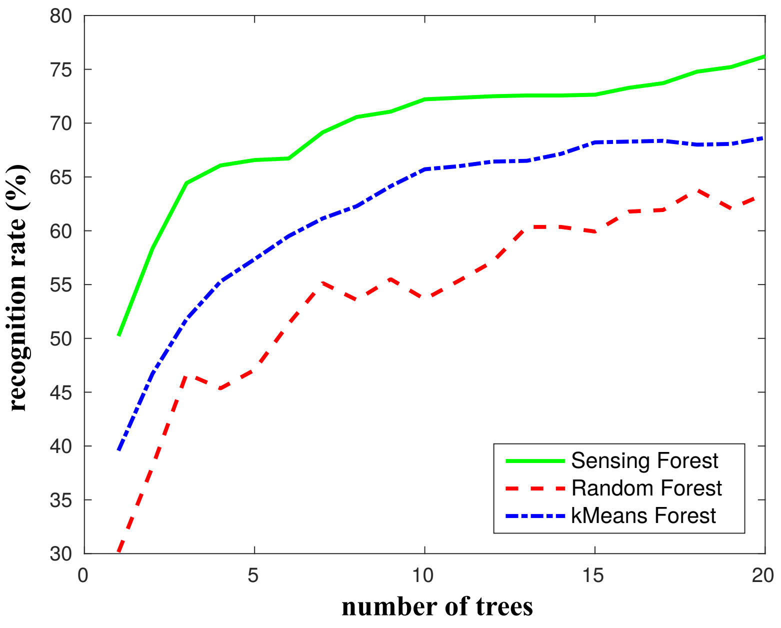


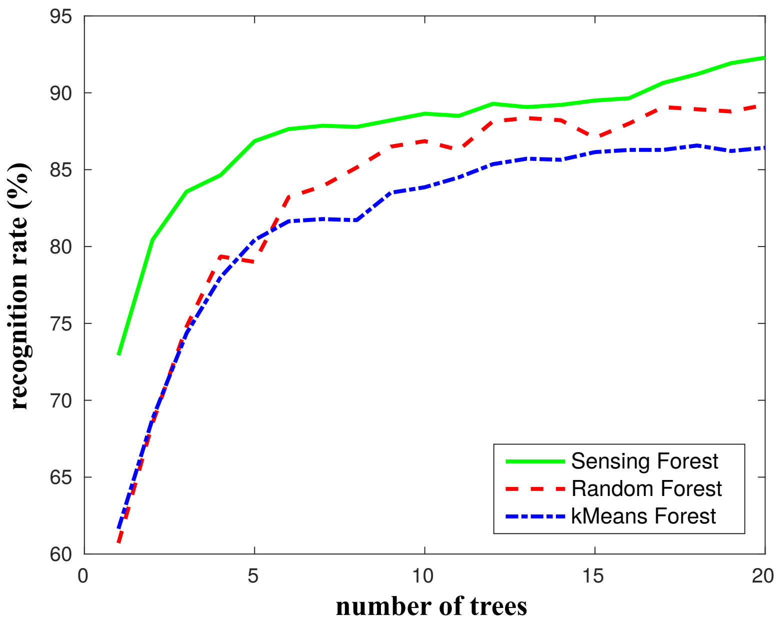
Figure 4.18: Selected sensing basis functions learned with the SF (second to 6th row) and the corresponding sample test image (first row) for MNIST and COIL-100 dataset. The number of the sensing values increases from the second row to the last row and the templates evolve from rather generic to more specific. Sensing basis functions from two different trees are shown on the two columns for each test image.

the corresponding compression ratios which are defined by the ratio between the original size of the images of the respective dataset (number of pixels) and the number of sensing values used for recognition. Figures 4.19 (COIL-100) and 4.20 (MNIST) show how the classification rate evolves after L levels as we add more trees to both approaches. For both datasets, COIL-100 and MNIST, the SF approach has a higher classification rate. The sensing values are represented by the scalar product between the reflectivity values in the scene and the weights that are learned in the different dimensions by using the tree partitioning of the data.

We can see in Figure 4.19 (b) and 4.20 that with one tree with 15 levels, which corresponds to using 30 sensing values (2 sensing values are used for each level) and a compression ratio greater than 546, the recognition rate is around 73% (for COIL-100) and a compression ratio greater than 26 with a recognition rate of 89.85% (for MNIST). Random Forest (RF) with only one tree reaches a recognition rate of 60.71% (for COIL-100) and 80% (for MNIST). The classification rate continues to increase as we add more trees and reaches 92.28% with 20 trees (for COIL-100) and 96% (for



(a) COIL-100, $L = 6$



(b) COIL-100, $L = 15$

Figure 4.19: SF recognition rate as a function of forest size (number of trees) compared to kMeans Forest and Random Forest (RF). The results are shown for COIL-100 for trees learned with maximum (a) $L = 6$ levels and (b) $L = 15$ levels.

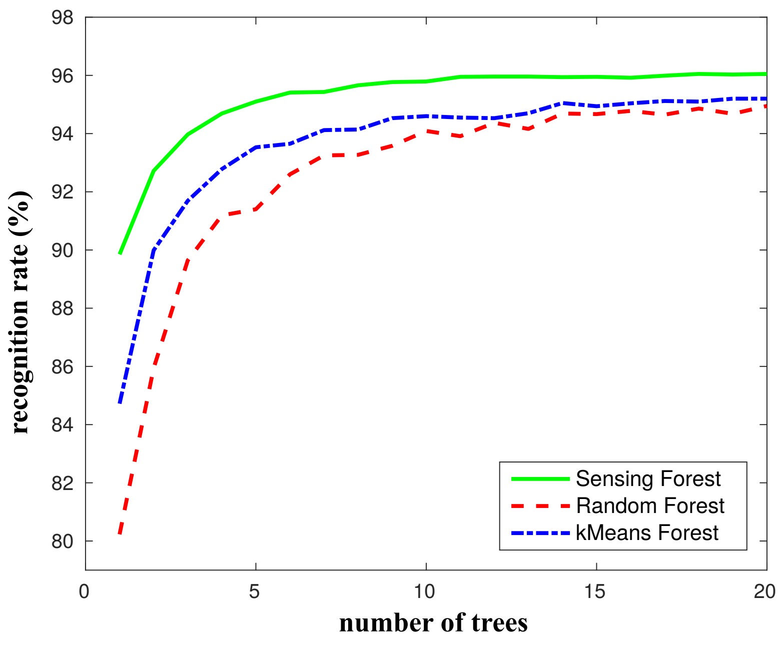


Figure 4.20: SF recognition rate as a function of forest size (number of trees) compared to kMeans Forest and Random Forest (RF). The results are obtained for MNIST with maximum 15 levels, $L = 15$.

MNIST) with 12 trees.

Figures 4.19 and 4.20 include also results obtained without the Learning Vector Quantization (LVQ) adaptation of the initial centroids, thus by only learning the features of the trees with the k-means algorithm. These results are labeled in Figures 4.19 and 4.20 as kMeans Forest. As it can be seen in Figure 4.20 for MNIST, kMeans Forest performs better than RF. For the COIL-100 dataset, kMeans Forest with 15 levels performs similar to RF only when a small number of trees is used. This behavior can be seen in Figure 4.19 (b). Figure 4.19 (a) shows that kMeans Forest has also a higher recognition rate in comparison to RF but for trees which are not so deep, in this example, $L = 6$. The SF approach has a higher classification rate compared to both kMeans Forest and Random Forest.

Figure 4.21 and Figure 4.22 show the performance of the SF approach in comparison to RF for different number of levels and different number of trees. With less sensing values the difference between the two approaches is even higher. This sustains our goal of exploring SF in terms of the best recognition rate reached with as few sensing values as possible.

By using the Sensing Forest (SF), we have shown how the performance depends on the number of samples, i.e., the depth of the trees and the size of the forest. The SF provides a novel way of learning how to sense the visual world given a particular task. We have here simulated the sensing

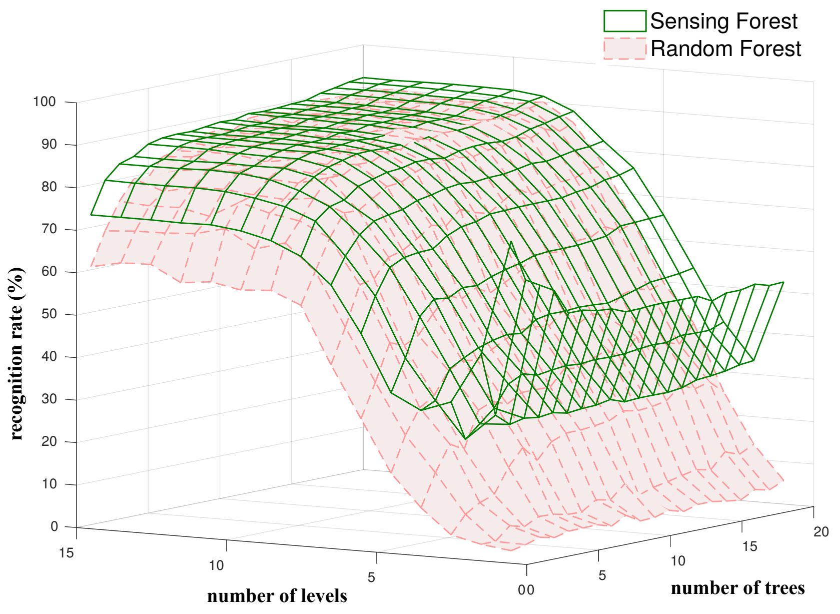


Figure 4.21: COIL-100: SF recognition rate as a function of depth (number of levels in a tree) and forest size (number of trees).

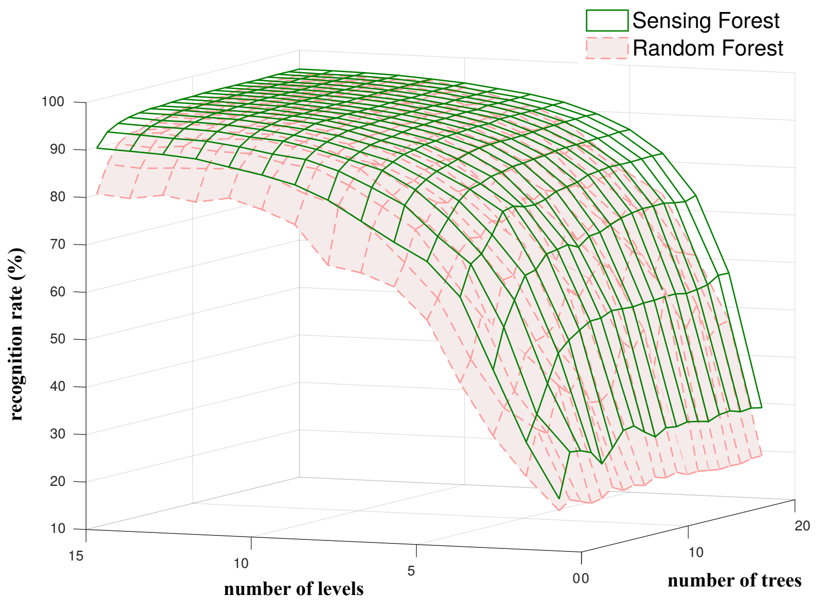


Figure 4.22: MNIST: SF recognition rate as a function of depth (number of levels in a tree) and forest size (number of trees).

by re-sampling digital images. We showed that the recognition performance of the kMeans Forest (which uses k-means centroids as prototypes) can be further increased by using Learning Vector Quantization (LVQ) to obtain better prototypes. The prototypes can be regarded as an adaptive, hierarchical sensing basis learned for a particular task. Overall we have shown that the SF has a higher recognition rate in comparison to a state-of-the-art Random Forest (RF).

5 Conclusions

Human sensing involves eye movements and attention. While looking around, our eye movements can acquire a great amount of information in order to fulfill a particular task as they are moving attention from one object to another. Based not only on current perceptions but also on previous experiences, objects can be recognized (or identified) instantaneously. Thus, we can characterize more accurately what we are sensing if, for example, we are looking for an object that we have already seen before, or at least we saw a similar object. In order to fulfill a task, and based on previous experiences, we are continuously acquiring information concentrating on the relevant information. The data is continuously adapted based on the already acquired information. For example, eye movement experiments with people looking at images with faces show scan paths which concentrate mostly on some areas of the face (eyes, nose, mouth). Thus, the sensing strategy of biological systems is based on directly extracting features while sensing a scene, rather than sampling the whole scene and afterwards extracting features.

In this context, we addressed in this thesis the problem of how to efficiently sample the visual world for a particular pattern-recognition task. Therefore, we presented five novel methods: Visual Manifold Sensing (VMS), Foveated Manifold Sensing (FMS), with the corresponding extended version Hybrid Foveated Manifold Sensing (HyFMS), Hierarchical Manifold Sensing (HMS), and Sensing Forest (SF). These methods are based on learning manifolds of low but increasing dimensions assuming that natural images lie on nonlinear low-dimensional manifolds. Following this geometrical approach, VMS learns manifolds using Locally Linear Embedding (LLE). FMS is an extension of VMS and involves foveation such that a foveated dataset is used while following the three steps of the VMS algorithm: learning, adaptation and sensing novel scenes. HyFMS acquires first a global gist of the scene by using the initial dataset, as VMS does, and afterwards continues sensing by using the foveated dataset, as FMS does. HMS learns a hierarchical partitioning by using Principal Component Analysis (PCA) and k-means clustering. The SF method is a prototype-based Random Forest (RF) with prototypes learned with k-means and Learning Vector Quantization (LVQ).

The proposed methods were inspired by Compressive Sensing (CS), in the sense that each acquired sensing value is a weighted sum over the whole visual scene. Furthermore, our approach extends Compressive Sensing (CS) by using hierarchical sensing schemes, and the sensing strategies based on machine learning algorithms can be optimized for particular datasets and for a specific kind of task. The benefits of the proposed methods are twofold. First, the sampling strategy is learned for a particular dataset, a procedure that reduces the number of required samples (sensing actions). Secondly, by data adaptation, every new sample depends on previously acquired samples. The sensing scheme is adaptive because sensing actions are not performed in a random fashion as with CS but are selected depending on both the environment and the particular scene that is sensed.

We showed that the proposed algorithms aim at efficient sensing and we have evaluated the efficiency in terms of the resulting recognition performance. We assumed the availability of a dataset of images that represent the type of objects and scenes that need to be sensed and recognized. Based on this data, the goal is to learn a sensing strategy such that recognition is possible with few sensing values. In this thesis we addressed a problem which is not typically addressed in current computer vision challenges but is becoming increasingly relevant as computer vision systems are becoming more pervasive. While currently the focus is on maximizing recognition performance, an equally challenging problem consists of finding the simplest solution for a given problem. Simplicity can be defined in different ways and in this work we adopt the approach of using a minimum number of sensing values and a simple classifier. This reduces both the required bandwidth of the sensor and the required processing power.

We evaluated the performance of the proposed sensing methods on benchmarks with natural images for face-, everyday object- and handwritten digit-recognition. The information gathered during sensing was quantified by the recognition performance that it enabled. In other words, the acquired samples were mainly assessed by how much they contributed to a particular task and not by how accurate they represented the world. We showed that we only need a few sensing actions and a simple classifier in order to solve such tasks.

While sensing a novel scene using either VMS, FMS, or HyFMS, the dataset is continuously adapted and the corresponding embedding is learned. Therefore, both methods operate on the entire dataset while sensing and they perform online learning. In a real-time sensing scenario we would need to learn the manifolds of the adaptive dataset before proceeding to the actual sensing. Thus, while sensing a novel scene, the already learned low-dimensional manifolds can be directly accessed. In order to overcome this problem, we provided HMS as an extension of VMS and FMS. HMS includes an adequate hierarchical partitioning of the dataset learned before the actual sensing. The underlying assumption is that sensing, and pattern recognition in general, rely on life-long learning processes that learn on a multitude of data subsets with different granularities, and that during sensing we can jump to the appropriate subsets based on the information gained from the already acquired sensing values.

VMS and FMS strongly depend on the choice of the corresponding parameters: the number of neighbors used for the manifold learning algorithm LLE; they depend also on the decreasing size of the adaptive dataset, and on the dimensions of the manifolds at each iteration of the algorithm. Therefore, we used for HMS the linear method PCA for learning the low-dimensional representations of the dataset where we only need to set the dimension of the representation. The hierarchical representation in the case of HMS is done by clustering the data in the low-dimensional representation using the k-means algorithm (where the number of clusters used has to be set). Still, HMS offered a rather high number of choices for the parameters. For example, it is not obvious in which dimension to start sensing, how to increase the dimension of the embedding manifolds and how many clusters to use for clustering the data. Therefore, we extended HMS to the Sensing Forest (SF). The SF method uses directly k-means for learning the low-dimensional representation of the data. The centroids computed with k-means are used as feature vectors which adapt while traversing the learned trees. The learned feature vectors can be improved by using Learning Vector Quantization (LVQ). In comparison to HMS, which learns only one hierarchical representation for the data, a tree, Sensing Forest (SF) was inspired by the Random Forest (RF) approach and learns several trees. As with RF, in order to avoid overfitting, SF learns several trees by using bagging randomness (training data is randomly chosen with replacement) and the features used for splitting the data are also randomly chosen.

FMS, HyFMS and HMS were inspired by the sampling strategy of biological systems and thus, they involve foveation. Consequently, they do not acquire global sensing values as CS, but they only sense the most salient areas of an object. However, we do not use the saliency of the actual image but only the average saliency of all images in the dataset. In addition to not being random, the sensing matrix employed by FMS, HyFMS, and HMS is also sparse since it only senses the most salient areas of an object. This high degree of sparseness in the sensing matrix distinguishes FMS, HyFMS and HMS from both VMS and CS. Moreover, the hybrid sensing scheme HyFMS that starts with VMS and continues with FMS, is again inspired by biological visual processing such that it first acquires a gist of the scene (global sensing) and then proceeds to more refined foveated sensing (local sensing). A possible weakness of the FMS approach is that the average saliency of all images are not useful if there is a large variance in object position and appearance. Indeed, one has to assume that such a foveated sensing scheme can only be applied once there is an expectation for, e.g., a face being at a certain position. In case of larger variations in appearance, position, and scale, the sensing scheme will still work well if these variations are covered by the dataset. There is a limitation of our approach in the sense that if we train for many different objects the sensing will be less efficient. This scenario would require a preliminary stage of object detection and rough object segmentation.

We showed that although we used a simple nearest-neighbor classifier for all the proposed methods, perfect recognition is possible on COIL-20 and ALOI with maximum 50 classes, with only 6 sensing values and on UMIST with 10 sensing values. On benchmarks such as MNIST, state-of-the-art performance could not be reached, but we showed that with HMS a large number of test images could be recognized with very few sensing values only. Such performance resembles human performance, since humans can effortlessly recognize a multitude of objects based on just the gist of a scene and require scrutiny for less familiar objects and more difficult recognition tasks. Our approaches are focused on finding the simplest solution for a recognition task by using a minimum number of sensing values and a simple classifier, rather than increasing the complexity for maximum performance. The Sensing Forest (SF) can be seen as a method in which the features used for recognition are sensed directly based on a sensing strategy learned in a semi-supervised manner. By using SF we have shown how performance depends on the number of samples, i.e., the depth of the trees and the size of the forest. We have shown that the way we learn the trees for sensing is a good way to learn a SF for recognition since it outperforms traditional Random Forest (RF) algorithms. We showed that recognition performance can be further increased by using LVQ to obtain better prototypes. Furthermore, the prototypes can be regarded as an adaptive, hierarchical basis learned for a particular task. The hierarchical basis learned with SF resembles the features learned by using approaches based on Convolutional Neural Networks (CNNs). The difference being the simplicity of the SF model, compared to the complexity of the CNNs.

A further application of our approach could be to compare the methods which involve foveation, FMS, HyFMS, and HMS, with visual sensing by recording eye movements on various datasets and using the eye movements instead of the structure tensor in order to define the regions of interest (the average saliency of the images in the dataset). Foveation can be also used for the SF method to increase the degree of sparsity in the sensing matrix. The performance of SF can be increased by using more complex LVQ algorithms.

Overall our novel methods are capable to enable agents acting in the real world, to independently adapt their representations and their sensing strategies to a particular environment and to a particular task. We do believe that bio-inspired sensors will be build that do not record an image but just the few visual features needed for the particular task they are used for. In this context, we have shown that our sensing strategies achieve promising results by using only a few sensing actions, i.e., only the features needed for classification. Our sensing strategies show the great potential of bio-inspired methods that can enable machines to sense the world like humans do.

Bibliography

[1] I. Biederman, “Recognition-by-components: A theory of human image understanding,” *Psychological Review*, vol. 94, pp. 115–147, 1987.

[2] S. Thorpe, D. Fize, and C. Marlot, “Speed of processing in the human visual system,” *Nature*, vol. 381, no. 6582, pp. 520–2, 1996. [Online]. Available: http://dx.doi.org/10.1038/381520a0

[3] R. L. Gregory, *The intelligent eye*. Weidenfeld and Nicolson, 1970.

[4] J. M. Findlay and I. D. Gilchrist, *Active Vision: The Psychology of Looking and Seeing*. Oxford University Press, 2003.

[5] J. J. Gibson, *The senses considered as perceptual systems*. Houghton Mifflin, 1966.

[6] M. A. Goodale and D. A. Westwood, “An evolving view of duplex vision: separate but interacting cortical pathways for perception and action,” *Current Opinion in Neurobiology*, vol. 14, no. 2, pp. 203 – 211, 2004.

[7] A. L. Yarbus, *Eye movements and vision*. Springer, 1967.

[8] S. Poslad, *Ubiquitous Computing: Smart Devices, Environments and Interactions*. Wiley, 2009.

[9] J. K. O’Regan, “Solving the “real” mysteries of visual perception: The world as an outside memory,” *Canadian Journal of Psychology*, vol. 46, no. 3, pp. 461–488, 1992.

[10] D. L. Donoho, “Compressed Sensing,” *IEEE Transactions on Information Theory*, vol. 52, no. 4, pp. 1289–1306, 2006.

[11] J. R. E. Candès and T. Tao, “Robust uncertainty principles: Exact signal reconstruction from highly incomplete Fourier information,” *IEEE Trans. Inform. Theory*, vol. 52, pp. 489–509, 2006.

[12] B. A. Olshausen and D. J. Field, “Natural image statistics and efficient coding,” *Network: Computation in Neural Systems*, vol. 7, no. 2, pp. 333–339, 1996.

[13] E. J. Candès and M. Wakin, “An introduction to Compressive Sampling,” *IEEE Signal Processing Magazine*, vol. 25, no. 2, pp. 21–30, 2008.

[14] I. Burciu, A. Ion-Margineanu, T. Martinetz, and E. Barth, “Visual Manifold Sensing,” in *Human Vision and Electronic Imaging XIX*, ser. Proc. of SPIE Electronic Imaging, B. E. Rogowitz, T. N. Pappas, and H. de Ridder, Eds., vol. 9014, 2014, pp. 90 141B–1–90 141B–8.

[15] I. Burciu, T. Martinetz, and E. Barth, “Foveated Manifold Sensing for object recognition,” in *IEEE International Black Sea Conference on Communications and Networking*, 2015, pp. 196–200.

[16] I. Burciu, T. Martinetz, and E. Barth, “Hierarchical Manifold Sensing with foveation and adaptive partitioning of the dataset,” in *Human Vision and Electronic Imaging 2016*, vol. 2016, no. 16, 2016, pp. 1–10.

[17] I. Burciu, T. Martinetz, and E. Barth, “Hierarchical Manifold Sensing with Foveation and Adaptive Partitioning of the Dataset,” *Journal of Imaging Science and Technology*, vol. 60, no. 2, pp. 020 402–1–020 402–10, 2016.

[18] I. Burciu, T. Martinetz, and E. Barth, “Sensing Forest for Pattern Recognition,” in *Advanced Concepts for Intelligent Vision Systems: 18th International Conference, ACIVS 2017, Proceedings*, ser. Lecture Notes in Computer Science, J. Blanc-Talon, R. Penne, W. Philips, D. Popescu, and P. Scheunders, Eds. Springer International Publishing, 2017, vol. 10617, pp. 126–137.

[19] A. Oliva and A. Torralba, “Modelling the shape of the scene: A holistic representation of the spatial envelope,” *International Journal of Computer Vision*, vol. 42, pp. 145–175, 2001.

[20] I. T. Jollife, *Principal Component Analysis*, 2nd ed. Springer, 2002.

[21] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, ser. Springer Series in Statistics. Springer New York Inc., 2001.

[22] D. Arthur and S. Vassilvitskii, “K-means++: the advantages of careful seeding,” in *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2007.

[23] A. Coates and A. Y. Ng, “Learning feature representations with k-means,” in *Neural Networks: Tricks of the Trade: Second Edition*, G. Montavon, G. B. Orr, and K.-R. Müller, Eds. Springer Berlin Heidelberg, 2012, pp. 561–580. [Online]. Available: https://doi.org/10.1007/978-3-642-35289-8_30

[24] L. Breiman, “Random forests,” C. Berkley, Tech. Rep. TR567, 1999.

[25] L. Breiman, “Random forests,” *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[26] T. Kohonen, “Learning vector quantization,” in *Self-Organizing Maps*. Springer Berlin Heidelberg, 1995, pp. 175–189.

[27] H. Barlow, *Possible principles underlying the transformations of sensory messages*. MIT Press, 1961, pp. 217–234.

[28] E. Mach, *The analysis of sensations and the relation of the physical to the psychical*. The Open Court Publishing Company, 1886.

[29] F. Attneave, "Some informational aspects of visual perception," *Psychological Review*, vol. 61, no. 3, pp. 183–193, 1954. [Online]. Available: http://view.ncbi.nlm.nih.gov/pubmed/13167245

[30] J. Stone, *Vision and Brain: How We Perceive the World*, ser. Vision and Brain. MIT Press, 2012.

[31] S. Nirenberg, S. M. Carcieri, A. L. Jacobs, and P. E. Latham, "Retinal ganglion cells act largely as independent encoders," *Nature*, vol. 411, no. 6838, pp. 698–701, 2001. [Online]. Available: http://dx.doi.org/10.1038/35079612

[32] A. S. Ecker, P. Berens, G. A. Keliris, M. Bethge, N. K. Logothetis, and A. S. Tolias, "Decorrelated neuronal firing in cortical microcircuits," *Science*, vol. 327, no. 5965, pp. 584–587, 2010. [Online]. Available: http://science.sciencemag.org/content/327/5965/584

[33] D. J. Field, "What is the goal of sensory coding?" *Neural Computation*, vol. 6, no. 4, pp. 559–601, 1994. [Online]. Available: http://dx.doi.org/10.1162/neco.1994.6.4.559

[34] C. Zetzsche, E. Barth, and B. Wegmann, "The importance of intrinsically two-dimensional image features in biological vision and picture coding," in *Digital Images and Human Vision*, A. B. Watson, Ed. MIT Press, 1993, pp. 109–38. [Online]. Available: http://www.inb.uni-luebeck.de/~/papers/ZeBaWe93a.html

[35] S. Mallat, *A Wavelet Tour of Signal Processing, The Sparse Way*. Academic, 2008.

[36] M. F. Duarte, M. A. Davenport, D. Takhar, J. N. Laska, T. Sun, K. F. Kelly, and R. G. Baraniuk, "Single-pixel Imaging via Compressive Sampling," *IEEE Signal Processing Magazine*, vol. 25, no. 2, pp. 83–91, 2008.

[37] F. J. Herrmann, M. P. Friedlander, and O. Yilmaz, "Fighting the curse of dimensionality: Compressive sensing in exploration seismology," *IEEE Signal Processing Magazine*, vol. 29, no. 3, pp. 88–100, 2012.

[38] Y. Zhang, Z. Dong, P. Phillips, S. Wang, G. Ji, and J. Yang, "Exponential Wavelet Iterative Shrinkage Thresholding Algorithm for Compressed Sensing Magnetic Resonance Imaging," *Information Sciences*, vol. 322, no. C, pp. 115–132, 2015.

[39] C. Zetzsche, K. Schill, H. Deubel, G. Krieger, E. Umkehrer, and S. Beinlich, "Investigation of a sensorimotor system for saccadic scene analysis: an integrated approach," in *From animals to animats 5: Proceedings of the Fifth International Conference on Simulation of Adaptive Behavior*, R. P. et al., Ed., vol. 5. MIT Press, Cambridge, 1998, pp. 120–126.

[40] E. Vig, M. Dorr, and E. Barth, "Efficient visual coding and the predictability of eye movements on natural movies," *Spatial Vision*, vol. 22, no. 5, pp. 397–408, 2009.

[41] R. Marks, *Introduction to Shannon Sampling and Interpolation Theory*. Springer-Verlag, 1991.

[42] Y. C. E. M. A. Davenport, M.F. Duarte and G. Kutyniok, "Introduction to Compressed Sensing," in *Compressed Sensing: Theory and Applications*, Y. C. Eldar and G. Kutyniok, Eds. Cambridge University Press, 2012, pp. 1–65.

[43] J. L. Starck, F. Murtagh, and J. M. Fadili, *Sparse Image and Signal Processing: Wavelets, Curvelets, Morphological Diversity*. Cambridge University, 2010.

[44] W. E. Vinje and J. L. Gallant, "Sparse coding and decorrelation in primary visual cortex during natural vision," *Science*, vol. 287, no. 5456, pp. 1273–1276, 2000.

[45] S. Foucart and H. Rauhut, *A mathematical introduction to Compressive Sensing*. Springer, 2013.

[46] E. Candès and J. Romberg, "Sparsity and incoherence in Compressive Sampling," *Inverse Problems*, vol. 23, no. 3, pp. 969–985, 2007.

[47] D. Schneider, "New camera chip captures only what it needs," *IEEE Spectrum*, 2013.

[48] R. Baraniuk, "Compressive Sensing," *IEEE Signal Processing Magazine*, vol. 24, no. 4, pp. 118–121, 2007.

[49] J. M. S. M. Lustig, D. L. Donoho and J. M. Pauly, "Compressed Sensing MRI," *IEEE Signal Processing Magazine*, vol. 25, no. 2, pp. 72–82, 2008.

[50] P. Nagesh and L. Baoxin, "A Compressive Sensing approach for expression-invariant face recognition," *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1518–1525, 2009.

[51] B. Hollis, S. Patterson, and J. Trinkle, "Compressed Sensing for tactile skins," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 150–157.

[52] A. Oliva, “Gist of the scene,” in *Neurobiology of Attention*, 2005, vol. 696, pp. 251–256.

[53] H. Barrow, “Recovering intrinsic scene characteristics from images,” *Computer Vision Systems*, pp. 3–26, 1978.

[54] C. Zetzsche and E. Barth, “Fundamental limits of linear filters in the visual processing of two-dimensional signals,” *Vision Research*, vol. 30, no. 7, pp. 1111–1117, 1990.

[55] C. Mota and E. Barth, “On the uniqueness of curvature features,” in *Dynamische Perzeption*, ser. Proceedings in Artificial Intelligence, G. Baratoff and H. Neumann, Eds., vol. 9. Infix Verlag, 2000, pp. 175–178. [Online]. Available: http://www.inb.uni-luebeck.de/publications/pdfs/ulm2000.html

[56] T. Aach, I. Stuke, C. Mota, and E. Barth, “Estimation of Multiple Local Orientations in Image Signals,” in *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP) 2004*, vol. 3, 2004, pp. 553–556.

[57] B. Jahne, P. Geissler, and H. Haussecker, Eds., *Handbook of Computer Vision and Applications with Cdrom*, 1st ed. Morgan Kaufmann Publishers Inc., 1999.

[58] E. Barth, “The minors of the structure tensor,” in *Mustererkennung 2000*, G. Sommer, Ed. Springer, 2000, pp. 221–228. [Online]. Available: http://www.inb.uni-luebeck.de/publications/pdfs/dagm2000.html

[59] R. Szeliski, *Computer Vision: Algorithms and Applications*, 1st ed. Springer-Verlag New York, Inc., 2010.

[60] S. Seung, and D. Lee, “The manifold ways of perception,” *Science*, vol. 290, no. 5500, pp. 2268–2269, 2000.

[61] B. Riemann, “Über die Hypothesen, welche der Geometrie zu Grunde liegen,” in *Abhandlungen der Königlichen Gesellschaft der Wissenschaften zu Göttingen*, 1867, vol. 13.

[62] L. W. Tu, *An Introduction to Manifolds*. Springer, 2011.

[63] H. Weyl and G. R. MacLane, *The concept of a Riemann surface*, ser. Dover Books on Mathematics. Dover, 2009.

[64] W. M. Boothby, *An introduction to differentiable manifolds and Riemannian geometry*, ser. Pure and Applied Mathematics. Academic Press, 1986.

[65] S. Roweis and L. Saul, “Nonlinear dimensionality reduction by locally linear embedding,” *Science*, vol. 290, no. 5500, pp. 2323–2326, 2000.

[66] The University of Sheffield, Image Engineering Laboratory, "The Sheffield/UMIST face database." [Online]. Available: http://www.sheffield.ac.uk/eee/research/iel/research/face

[67] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *PNAS*, vol. 79, no. 8, p. 2554–2558, 1982.

[68] H. S. Seung, "Learning continuous attractors in recurrent networks," *Advances in Neural Information Processing Systems*, vol. 10, pp. 654–660, 1998.

[69] L. K. Saul and S. T. Roweis, "An introduction to Locally Linear Embedding ," *Research Report, AT&T Labs*, 2000.

[70] B. Schölkopf, A. Smola, and K.-R. Müller, "Nonlinear Component Analysis as a Kernel Eigenvalue Problem," *Neural Computation*, vol. 10, no. 5, pp. 1299–1319, 1998.

[71] J. Ham, D. Lee, S. Mika, and B. Schölkopf, "A kernel view of the dimensionality reduction of manifolds," in *Proceedings of the 21st International Conference on Machine Learning*. ACM, 2004, pp. 369–376.

[72] I. Borg and P. J. F. Groenen, *Modern multidimensional scaling: theory and applications*. Springer, 1997.

[73] J. B. Tenenbaum, V. de Silva, and J. C. Langford, "A global geometric framework for nonlinear dimensionality reduction," *Science*, vol. 290, pp. 2319–2323, 2000.

[74] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[75] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux, "API design for machine learning software: experiences from the scikit-learn project," in *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, 2013, pp. 108–122.

[76] L. K. Saul and S. T. Roweis, "Think globally, fit locally: Unsupervised learning of nonlinear manifolds," *Journal of Machine Learning Research*, vol. 4, pp. 119–155, 2003.

[77] R. C. Nichol, S. Chong, and L. Wasserman, "Computational astrostatistics: Fast and efficient tools for analysing huge astronomical data sources," *Statistical Challenges in Modern Astronomy III*, 2001. [Online]. Available: http://arxiv.org/abs/astro-ph/0110230

[78] L. Breiman, J. Friedman, R. Olshen, and C. Stone, *Classification and Regression Trees*. Wadsworth and Brooks, 1984.

[79] R. E. Schapire, "The strength of weak learnability," *Machine Learning*, vol. 5, no. 2, pp. 197–227, 1990. [Online]. Available: https://doi.org/10.1007/BF00116037

[80] A. Criminisi, J. Shotton, and E. Konukoglu, "Decision forests: A unified framework for classification, regression, density estimation, manifold learning and semi-supervised learning," *Found. Trends. Comput. Graph. Vis.*, vol. 7, no. 2-3, pp. 81–227, 2012.

[81] T. K. Ho, "Random decision forests," in *Third International Conference on Document Analysis and Recognition, ICDAR 1995*, vol. I, 1995, pp. 278–282.

[82] Y. Amit and D. Geman, "Shape quantization and recognition with randomized trees," *Neural Comput.*, vol. 9, no. 7, pp. 1545–1588, 1997.

[83] T. K. Ho, "The random subspace method for constructing decision forests," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 20, no. 8, pp. 832–844, 1998.

[84] R. Caruana, N. Karampatziakis, and A. Yessenalina, "An empirical evaluation of supervised learning in high dimensions," in *Proceedings of the 25th International Conference on Machine Learning*, ser. ICML '08. ACM, 2008, pp. 96–103.

[85] Microsoft Corp. Redmond WA., Tech. Rep.

[86] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake, "Real-time human pose recognition in parts from single depth images," in *Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition*, ser. CVPR '11. IEEE Computer Society, 2011, pp. 1297–1304.

[87] J. Shotton, T. Sharp, A. Kipman, A. Fitzgibbon, M. Finocchio, A. Blake, M. Cook, and R. Moore, "Real-time human pose recognition in parts from single depth images," *Commun. ACM*, vol. 56, no. 1, pp. 116–124, 2013.

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

[89] M. Biehl, B. Hammer, and T. Villmann, "Prototype-based models in machine learning," *Wiley Interdisciplinary Reviews: Cognitive Science*, vol. 7, no. 2, pp. 92–111, 2016. [Online]. Available: http://dx.doi.org/10.1002/wcs.1378

[90] T. Villmann, A. Bohnsack, and M. Kaden, "Can learning vector quantization be an alternative to SVM and deep learning? - Recent trends and advanced variants of learning vector quantization for classification learning," *Journal of Artificial Intelligence and Soft Computing Research*, vol. 7, no. 1, pp. 65–81, 2017.

[91] R. Gray, "Vector quantization," *IEEE ASSP Magazine*, vol. 1, no. 2, pp. 4–29, 1984.

[92] K. L. Oehler and R. M. Gray, "Combining image compression and classification using vector quantization," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 17, no. 5, pp. 461–473, 1995.

[93] T. Kohonen, *The basic SOM*. Springer Berlin Heidelberg, 1995.

[94] J. O. Berger, "Bayesian Analysis," in *Statistical Decision Theory and Bayesian Analysis*. Springer New York, 1985, pp. 118–307.

[95] G. Voronoi, "Nouvelles applications des paramètres continus à la théorie des formes quadratiques," *Journal für die Reine und Angewandte Mathematik*, vol. 133, no. 133, pp. 97—178, 1908.

[96] J. M. Geusebroek, G. J. Burghouts, and A. W. M. Smeulders, "The amsterdam library of object images," *International Journal of Computer Vision*, vol. 61, no. 1, pp. 103–112, 2005.

[97] S. A. Nene, S. K. Nayar, and H. Murase, "Columbia object image library (coil-100)," 1996. [Online]. Available: http://www1.cs.columbia.edu/CAVE/software/softlib/coil-100.php

[98] F. Dornaika, Y. E. Traboulsi, B. Cases, and A. Assoum, "Image classification via semi-supervised feature extraction with out-of-sample extension," in *Advances in Visual Computing - 10th International Symposium (ISVC)*, 2014, pp. 182–192.

[99] E. Vural and C. Guillemot, "Out-of-sample generalizations for supervised manifold learning for classification," *IEEE Transactions on Image Processing*, vol. 25, no. 3, pp. 1410–1424, 2016.

[100] D. Cui, G. Zhang, W. Han, L. Lekamalage Chamara Kasun, K. Hu, and G.-B. Huang, "Compact Feature Representation for Image Classification Using ELMs," in *The IEEE International Conference on Computer Vision (ICCV)*, 2017.

[101] V. Pawar and S. Talbar, "Machine learning approach for object recognition," *International Journal of Modeling and Optimization*, vol. 2, no. 5, 2012.

[102] M. Liang and X. Hu, "Recurrent convolutional neural network for object recognition," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[103] "Visualizing CNN feature maps and filters on tensorboard - MNIST visualization example," accessed: 2017-11-23. [Online]. Available: https://github.com/jireh-father/tensorflow-cnn-visualization

[104] Y. Wang, Z. Xie, K. Xu, Y. Dou, and Y. Lei, "An efficient and effective convolutional auto-encoder extreme learning machine network for 3d feature learning," *Neurocomputing*, vol. 174, no. Part B, pp. 988 – 998, 2016.

Acronyms

CS	Compressive Sensing
PCA	Principal Component Analysis
MDS	Multidimensional scaling
LLE	Locally Linear Embedding
LVQ	Learning Vector Quantization
SOM	Self Organizing Maps
VQ	Vector Quantization
iD	intrinsic dimension
VMS	Visual Manifold Sensing
FMS	Foveated Manifold Sensing
HyFMS	Hybrid Foveated Manifold Sensing
HMS	Hierarchical Manifold Sensing
RF	Random Forest
SF	Sensing Forest

List of Figures

List of Tables

List of Algorithms