Towards an Integrated Model of Feedforward-Feedback Processing in the Visual Cortex

by

Ivaylo P. Riskov

Submitted to the Department of
Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

May 2008

Author .....
Department of Electrical Engineering and Computer Science
May 23, 2008

Certified by .....
Tomaso Poggio
Eugene McDermott Professor
Thesis Supervisor

Certified by .....
Thomas Serre
Postdoctoral Associate
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Professor of Electrical Engineering
Chairman, Department Committee on Graduate Theses

Towards an Integrated Model of Feedforward-Feedback Processing in the Visual Cortex

by

Ivaylo P. Riskov

Submitted to the Department of Electrical Engineering and Computer Science
on May 23, 2008, in partial fulfillment of the
requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

The goal of this work is to explore a potential improvement on a visual recognition system. The system is a biologically-plausible computational model of the feedforward part of the ventral stream in the visual cortex and successfully models human performance on visual recognition tasks for the first 50–100 milliseconds since the presentation of the visual stimulus.

We make the first steps to a possible augmentation of the system that will account for both feedforward and feedback processes in the ventral stream. We explore the plausibility of Bayesian network models for feedback. Our results show that although the resulting system underperforms the original, it has a better rate of improvement as more and more training examples are added to it.

Thesis Supervisor: Tomaso Poggio
Title: Eugene McDermott Professor

Thesis Supervisor: Thomas Serre
Title: Postdoctoral Associate

Acknowledgments

I would like to thank my advisors Thomas Serre and Tomaso Poggio for their support and guidance in this work. Without their help I would not have made it this far. When I started, I knew almost nothing about the neurobiological aspect of the project, but here I am writing a thesis about it.

Thanks to Sharat Chikkerur for his tips on making the Bayesian network better and for assembling and providing the dataset.

Special thanks to my wife Maria for her moral support and for helping me with some of the figures.

Last but not least, thanks to everyone else at CBCL for their help and for providing a good working atmosphere.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The goal of this Master of Engineering thesis work is to explore a few approaches towards plausible computational modeling of the feedforward-feedback processes that take place in the ventral stream of the visual cortex in primates.

1.1 Brief Introduction to the Neocortex

The visual cortex is the part of the neocortex responsible for perceiving and processing information received from the sight sensors, the eyes. Locating and recognizing objects, detecting motion, focusing attention, perceiving three dimensional objects, and eye control are some of its primary functions. The visual cortex consists of interconnected visual areas, each responsible for a particular function.

In another logical division, we can differentiate information pathways that go through certain visual areas and ultimately accomplish one of the primary functions mentioned above. Two such pathways are the *ventral* and the *dorsal* streams. [21] The ventral stream is the processing path that follows $V1$ (visual area 1), $V2$, $V4$, and IT (inferotemporal cortex) areas in this order. It is responsible for recognizing objects and forms, as well as building and storing their high level representations. The dorsal stream, on the

other hand, starts from the $V1$ area and goes through $V2$ and MT (middle temporal) in order. It is associated with motion processing.

1.2 The Role of Feedback in Neural Activities

It is widely accepted that these pathways are not simply cascades of neural activity propagating in one direction, for example starting from $V1$ and going sequentially to $V2$ and then to $V4$, but rather feedback systems in which upstream areas can provide feedback information to downstream areas. So $V4$ will provide feedback input back to $V2$ and $V2$, in turn, will provide feedback input back to $V1$. In fact, it is speculated [9] that there are ten times as many feedback connections as there are feedforward connections.

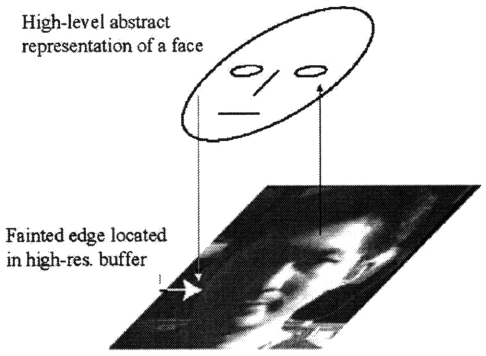

Figure 1-1: *The "occluded face" example of feedback activity.* Adapted from Lee and Mumford, [19].

Let us consider an example for why feedback connections may play an important role in the visual system. Figure 1-1 shows a person's face with one part lit and the other part occluded by a shadow. There is a sharp contrast between the two parts. Without feedback connections, the brain would have problems discerning the two parts and identifying them as one face and, much like a digital camera, only capture and pay attention to the lit part of the face potentially not detecting it as a face at all. In a feedforward-feedback system, a higher level representation of the lit part will be formed

as a result of the feedforward sweep of neural activity. At this point, however, higher layers will be able to put higher prior on the presence of a face on the image. Given this prior and the fact that faces are oval and have vertical symmetry, the feedback sweep will prompt lower layers to focus more on the shadowed part of the image and make out the complete oval and symmetric contour of the face, thus overcoming the problem with the sharp contrast.

In general, the role of feedback is to fill in missing information. As the example on Figure 1-2 shows, the three corners of a perceived square are enough to build up a belief that there is a square in the figure. In this case, feedback provides higher prior on the whole given the available partial information. In [19], Lee and Mumford show that monkeys see squares in similar settings.

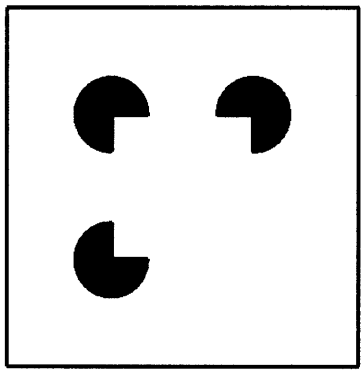

Figure 1-2: *An example where the visual cortex fills in missing information.* Not only the edges of the perceived square are missing, but so is one of the corners. Yet, we can still "see" a square.

1.3 Thesis Statement

In this thesis work we approach the problem of modeling feedback in visual object recognition tasks. In particular, we evaluate two object recognition systems, one is a biologically plausible model of the feedforward part of the ventral stream [29]; and the

other is a speculative model [22, 19] of the feedback paths based on Bayesian networks.

We propose a method to augment the first system using ideas from the second to test the hypothesis that the new system will account for both feedforward and feedback processes. We describe in detail the two systems, our approach, and the obtained results.

1.4 Contributions

- An augmentation of a biologically plausible model of the feedforward part of the ventral stream;
- The use of real-world images with the Bayesian network model;
- An implementation and evaluation of three approaches for generating codebooks of visual features.

1.5 Overview of the Contents

In the second chapter, we present the two computational systems we base our work on. This chapter also defines most of the terminology used throughout the paper.

The third chapter presents our experiments on a simple combination of the two systems. We call this combination a hybrid model. We identify two problems that fork off our efforts in two separate directions – the susceptibility of the model to good dictionary of visual prototypes and the numerical instabilities and other parameter estimation problems arising from the use of large-scale Bayesian networks with lots of parameters.

The following chapter examines in detail the first of these problems, the problem of finding a good dictionary of visual features. We present three different solutions and evaluate their relative performance.

The final chapter presents evidence and an in-depth analysis of the experiments carried on the augmented model on a large database of input images. Additionally, we

describe several improvements on our initial model.

We conclude with an evaluation of the model's capability of exploiting feedforward-feedback connections, it's general applicability, and whether or not the results we obtained merit further research in this particular approach. We also present some open problems for which we either did not find acceptable solutions or that represent extensions to the project that can further testify to the validity of the model.

Chapter 2

Background

2.1 Computer Vision and Machine Learning

Computer vision is a broad field of scientific disciplines devoted to making machines interpret and make sense of visual input. Biological computer vision is a branch with the explicit goal of developing computer systems that emulate the processes in the visual cortex. This approach proves to be invaluable for testing various hypotheses related to the workings of the cortex.

In general, computer vision research utilizes numerous techniques from artificial intelligence and machine learning that provide the mathematical and computation framework in which visual systems are developed and quantitatively analyzed.

Supervised learning is the most common machine learning approach used in computer vision. In its essence it dictates that the parameters of the model are learned based on a set of inputs for which the desired output is known and is provided to the model (hence the word supervised). This set of inputs is colloquially known as the *training set* (also *training data*). The model with learned parameters can then be used to predict the output for any other input.

Different models may provide different predictions for the same input, so comparing

how successful they are is important. This is usually done by evaluating a model on a *testing set* of input data. The desired output for the testing set is known, but not provided to the model. Instead the model's predicted output is compared to the known desired output and an error/success measure is computed. A common example of a measure is the fraction of all predictions that match the desired output. Another measure applicable for numerical outputs is the L_2 distance.

2.2 Object Recognition

Computer vision is a large field and includes a broad range of problems. Some of these problems are directly inspired by the tasks the visual cortex performs. Object localization, object recognition, scene segmentation, and attention are all examples of such tasks.

This work considers the problem of object recognition. Its simplest formulation is: given an input stimulus/image, determine if a class of objects is depicted or appears on the image. This is the binary formulation of the problem that we consider. The multiclass formulation prompts for recognition of a set of classes of objects and can either be reduced to a set of binary recognition problems or approached directly with a multi-class model. From now on, when we refer to object recognition, we understand binary object recognition.

In reference to the above description of supervised learning, we can point out that in binary recognition the output is usually binary ("yes" or "no") or single-dimensional scalar representing the level of confidence that can be thresholded to produce a binary answer. Similarly, training and testing data is usually labeled as positive (the class of object under consideration appears on the image) or negative.

2.3 Challenges in Object Recognition

According to Riesenhuber and Poggio [26], the biggest challenge in object recognition and especially in modeling the neurophysical aspect of it is the level of versatility and variability of the visual cortex. It is truly amazing that a single object can appear in various orientations, in different positions, under different lightning conditions, with partial occlusions and we still manage to recognize it in an instant. To achieve such versatility in recognition, a highly-abstract and invariant description of the input is necessary.

Coming up with highly-abstract, yet descriptive enough, representations is possibly the primary challenge for computer vision. Moreover, such representations need to be position-, orientation-, and scale-tolerant.

A different set of challenges arises from technological point of view. Unlike cortical pathways, computational systems run on serial or small-scale parallel machines. Thus, technical feasibility and resource usage are major factors that need also be considered. Making the right trade-off is an important challenge that will come up several times during this thesis work.

2.4 Two Object Recognition Systems

Our work combines the ideas of two existing object recognition systems. The first system is a biologically inspired model of the ventral stream designed and developed in the Center for Biological and Computational Learning (CBCL) at MIT. The second system is implemented by Dean in [6] and in essence represents a Bayesian network with a special topology. The following section contains detailed description of each system and an introduction to some of the computational tools and terms that we use in our work.

2.4.1 Standard Model of the Visual Cortex

Relation to Physiology

In [29] Serre *et al.* describe a neurophysically-inspired computer model of the ventral stream. This system, which we colloquially dub "the standard model", attempts to go further than the mainstream account of only the $V1$ and $V2$ areas of the visual cortex. In fact, the system successfully models processing stages along the entire ventral stream starting from the $V1$ and $V2$ areas going all the way to the IT area. Empirical results show that output from the highest processing stages of the system match readouts from the IT cortical area in monkeys.

It has been shown empirically that the standard model matches the performance of human subjects on rapid object recognition tasks. Evidence suggests that the system models reasonably accurately the first 50-100 milliseconds of neural activity after a visual stimulus is presented.

Detailed Description

The standard model consists of a stack of alternating simple and complex processing layers. Each layer is composed of cells/units of the same type. Simple layers evaluate fixed mathematical functions, whereas complex layers perform a pooling operation over a range of simple units. The set of simple units over which each complex cell pools is called a *receptive field.*

In a simplified model, there are 4 layers, 2 of each type. They are the $S1$, $C1$, $S2$, and $C2$ layers in order from the lowest to the highest in the hierarchy. The visual stimulus is a two-dimensional intensity or gray-valued map (we discard the entire color information) that enters the $S1$ layer and its abstract representation is the output of the $C2$ layer.

The $S1$ layer corresponds in function to the simple cells in the primary visual cortex, the $V1$ cortical area. It has been shown in [15] that the $S1$ layer ($V1$ cells respectively)

evaluates a set of spatial Gabor filters of the form:

$$f_G(x, y|\sigma, \lambda, \gamma, \phi, \chi) = e^{-\frac{x_0^2+\gamma^2 y_0^2}{2\sigma^2}} \cos\left(\frac{2\pi}{\lambda}x_0 + \chi\right)$$

$$x_0 = x\cos\phi + y\sin\phi \qquad y_0 = -x\sin\phi + y\cos\phi$$

As evident from the formula, the Gabor filter is an exponential with a particular scale(σ) superimposed on a sine wave a given wavelength(λ) and orientation (ϕ). The $S1$ units are the responses of the visual stimulus for a set of Gabor filters with varying orientation, wavelength, and scale. In effect, Gabor filters respond to edges and bars of different orientation and scale.

The $C1$ processing layer collects the output of the $S1$ units and groups them by *scale bands* which are pairs of neighboring scales. $C1$ cells compute a max operator across the pool of all $S1$ units in the group.This operator is computed across all spatial positions and orientations resulting in one $C1$ unit (we also refer to this as a $C1$ map) for each scale band. Note that the $C1$ unit contains the entire orientation information as a third, non-spatial dimension. This is in addition to the 2 spatial dimensions that result from computing the max operator across all positions. Figure 2-1 shows an example of a $C1$ map in all four orientations for a given scale.

Finally, $C1$ maps are downsampled in order to increase the computational speed of the system. After the max operator is computed for all locations, only each x-th location is considered in the downsampled image. x is called the *sampling step*. Downsampling, however, can have undesirable effects if performed naïvely because the sampled location may fall at a local minimum. Instead, the value of the maximum response in a window around the sampled location is taken.

The $S2$ stage computes a radial basis function (RBF) of the form $S2_i(x, y) = e^{-\beta||C1(x,y)-X_i||^2}$ between the $C1$ maps and a set, X, of precomputed prototypes. The entire computation is done across positions in a sliding window manner: a window of

Figure 2-1: *Example of $C1$ maps in 4 orientations.* To the left is the original image. The right part of the figure shows the $C1$ maps in all four orientations: horizontal (top-left), vertical (top-right), slanted (bottom left and right).

size $N \times N$, the size of the prototypes, is moved across the $C1$ map and $C1(x, y)$ is the part of the map beneath the window. This is equivalent to having units of the same type at each location, see Masquelier *et al.* [20]. The RBF function treats $C1(x, y)$ and X_i as linear vectors of dimensionality $N^2 r$, where r is the number of orientations. The resulting $S2$ map contains two spatial dimensions resulting from the sliding window, but no orientation dimension because all orientation information was absorbed in the process.

The prototypes play the role of a *codebook* of features against which the $C1$ units are compared for similarity. This is also known as a *dictionary of visual features*, see Serre [28]. In a sense, the codebook defines the building blocks to which we can compose or decompose the input and the radial basis function provides a distance metric as to how

similar a given part of the image is to a particular building block.

In the standard model, the codebook is built by random sampling the $C1$ maps.

Finally, the $C2$ stage applies a max operator on $S2$ maps across all positions and scales to produce a set of values, one for each prototype, that together make the $C2$ map. Therefore, each value in the map is the maximum RBF response from the stimulus to each of the corresponding prototypes. Because of the relation between RBF and L_2 distance, $C2$ maps can be considered as a scale- and position-invariant closeness metric between the input and a set of visual building blocks, the prototypes.

The final stage is the classification stage. During supervised learning we produce one $C2$ map for each training image and we associate it with its class (positive or negative). The result is then fed to a linear support vector machine (SVM) [3] classifier that builds the final model. The classification model can then be used to evaluate the performance on the testing set.

Feedforward Processing

As mentioned earlier, the standard model is biologically inspired. $S1$ and $C1$ stages correspond to computations that take place in the $V1$ area, while $S2$ and $C2$ units resemble the $V4/IT$ area in functionality. Note, however, that the entire computational process is serial and the system lacks any feedback mechanisms or pathways. In the previous chapter we discussed that this is not how the cortex works. Yet, the model is empirically observed to matches human behavior during the first 50-100 milliseconds before feedback is activated [29]. After that, humans start to outperform it. This is likely due to the feedback paths the standard model does not account for.

It is reasonable to assume that introduction of feedback mechanisms to the model will improve the accuracy of the model allowing it to match that of humans for exposures longer than 100 milliseconds. This is what motivates this thesis work. In particular, we focus our attention on modeling feedback using Bayesian networks. The second system

that we examine is one of the pioneers of (non-trivial) Bayesian networks in object recognition.

2.4.2 Pyramidal Bayesian Network Approach

In 1992, Mumford [22] proposed a hierarchical Bayesian model to address the problem of modeling feedback mechanisms in the cortex. He suggests the use of conditional probability dependencies between computational structures responsible for neighboring visual areas to allow higher layers to modify the prior distributions of lower layers, much like in the "occluded face" example from Chapter 1. Figure 2-2 illustrates the idea. The suggestion by Lee and Mumford[19] to use Bayesian networks as graphical models follows naturally from the inherent ability of Bayesian networks to encode conditional probability information.

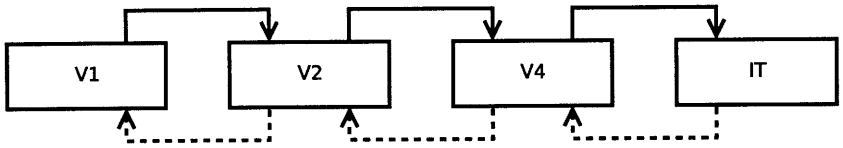

Figure 2-2: *An illustration of the feedback paths in the visual cortex.* The feedback paths that provide the object priors are the dotted lines. Adapted from Dean, [6].

However, Lee and Mumford did not provide a working implementation of their idea. In 2005, Dean [6] first proposed a working model to follow the suggestions outlined by Lee and Mumford. In particular, his approach uses a Pyramidal Bayesian network model (PBN) to account for feedback.

Pyramidal Bayesian networks play the role of the final classification stage, which, in the standard model, is performed by the linear SVM classifier. Although Bayesian networks as classifiers can be used as a simple drop-in replacement for linear SVMs, we take the approach one step further by introducing the classifier at the $C1$ stage and also provide a method of turning the non-probabilistic nature of the standard model

into beliefs that can be used with the PBN. This allows us to draw parallel between the layered structure of the Bayesian network and the pathways in the visual areas of the ventral stream.

Bayesian Networks

A Bayesian network [14, 25] is a graph in which nodes represent random variables and (directed) edges encode conditional dependencies between the variables. The entire network implicitly stores the joint probability distribution of all variables as conditional distributions between dependent variables. In the example network on Figure 2-3, the total probability distribution can be computed as $P(x, y, z, t, v) = P(x).P(y|x).P(z).P(t|x, z).P(v|t)$. If in the example we assume all variables to be independent then explicitly describing the total probability distribution will require $2^5 = 32$ parameters. If we instead use the Bayesian network on Figure 2-3, we only need $2 + 2^2 + 2 + 2^3 + 2 = 18$ parameters to describe the distribution. Effectively, Bayesian networks exploit conditional dependencies between the variables to provide a way of storing the joint probability distribution using far less memory than if we were to store it explicitly.

The most common operation in Bayesian networks is to marginalize the distribution given the values of some variables. This is called *inference* and is implemented by algorithms called *inference engines.* Examples of inference engines are the Junction Tree engine[17] for exact inference, the loopy belief propagation[24] and the Gibbs sampling[12] engine for approximate inference.

Learning engines, on the other hand, are algorithms for supervised learning of the probability distributions for a set of training data points.

Computer libraries implementing Bayesian networks provide implementation of various inference and learning engines. Following Dean's implementation, we use the Junction Tree Inference engine and the popular Expectation Maximization learning engine [8].

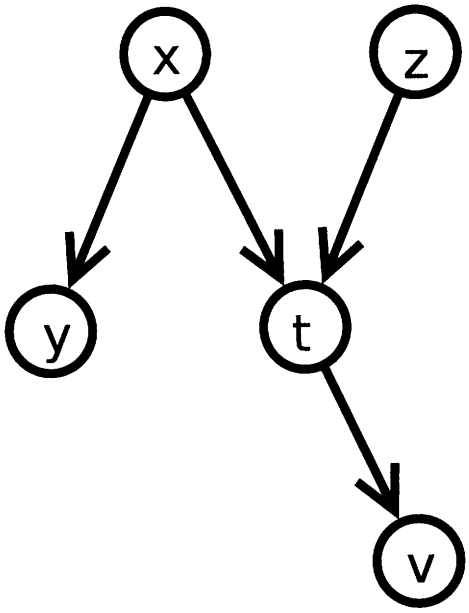

Figure 2-3: *A simple example of a Bayesian network with five nodes.* Here nodes represent random variables and edges represent conditional probabilities.

Pyramidal Bayesian Networks (PBN)

In [6, 7], Dean uses Bayesian networks of special pyramidal topology. PBNs are hierarchical graphs in which each layer (or level) of the hierarchy is a rectangular mesh of nodes. Each node in the mesh has children nodes in the layer immediately below. Conversely, all nodes (except the one at the top) have parents in the layer above. For each node, its children form a receptive field. The fact that units and their receptive fields are of pyramidal shape is where the model takes its name. The receptive fields of the nodes on any given layer (except the last) form the mesh of the next layer. The uppermost level contains only one node without any parents, the root, while the layer at the bottom is the one that receives the input. Figure 2-4 shows an example of a PBN where all nodes, but those on the bottom, has a receptive field of size 3×3.

We allow for the receptive fields of two neighboring nodes on one level to overlap. Figure 2-5 shows an example of such network in which the receptive fields of the nodes

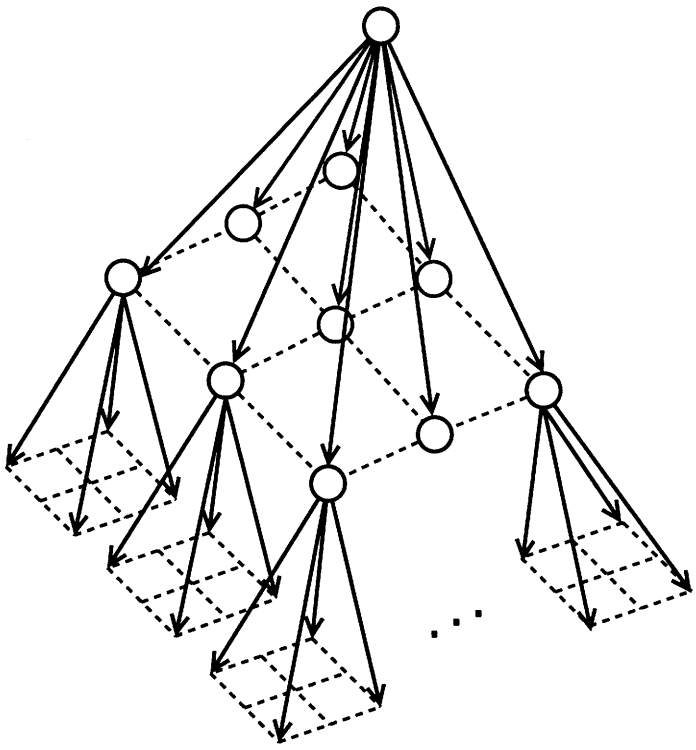

Figure 2-4: *A pyramidal Bayesian network with 3 layers and receptive fields of size* 3×3. Note that nodes in each layer of the network are arranged in a rectangular grid.

in the middle layer overlap by 1 unit.

PBN Inputs

In this model, all random variables in the network are discrete. In particular, the variable for the root node is binary as it encodes the probability of the input being positive or negative. The input itself is also over a discrete domain.

Because the input layer is a rectangular mesh, we can maintain the notion of a 2-dimensional visual stimulus.

In the original implementation, Dean proposes the use of Gaussian mixtures [5] to filter the visual stimulus and produce a rectangular matrix of discrete values suitable for use as input to a PBN. The Gaussian mixture filter works as follows:

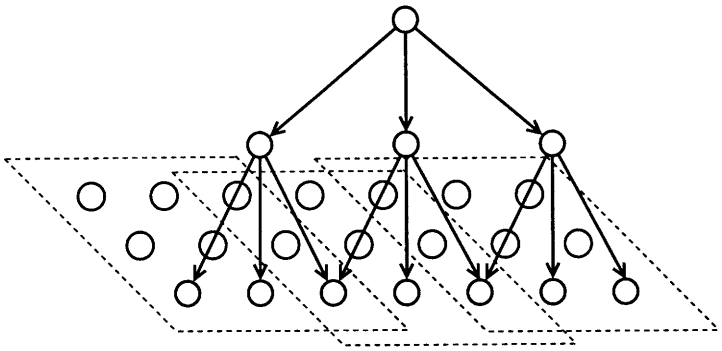

Figure 2-5: *An example of a pyramidal Bayesian network where receptive fields overlap by one location.*

1. Consider each stimulus from the training dataset as a gray-valued image (the usual computer representation of an image).

2. For each image, take crops of size $W \times W$. Consider each crop a vector in $\mathbb{R}^{W^2}$.

3. Learn the parameters of a Gaussian Mixture Model (GMM) for a set of M Gaussians. For more details see [5].

4. For each training image and for each location, consider the patch of size $W \times W$ centered at that location. The W^2-dimensional vector x can be expressed as a linear combination of the M multivariate Gaussian functions, $x = \sum_{i=1}^{M} \alpha_i G_i$. Consider the index of the Gaussian with the highest weight, $J = \arg\max \alpha_i$.

5. Finding the index of the "closest" Gaussian for each training image for each location gives us a map or matrix with the same size as the image and with values between 1 and M. This map is then used as an input to the PBN.

Essentially Gaussian mixtures learn a codebook of visual features in which each Gaussian represents one feature. This is very similar to how the standard model produces $S2$ maps using stored prototypes of visual features.

Lateral Connections

Another distinction of the PBN is the option of having lateral connections at some of the levels of the network. Lateral connections are additional conditional dependencies imposed on neighboring nodes in the mesh. They are applicable only to hidden layers (those other than the root node and the input layer). During learning, lateral connections help in the learning of spatial dependencies. During testing, they effectively force those dependencies on the input. We study the role of lateral connection in the model, but overall we expect that their presence favors inputs that are smooth, which real images are.

Figure 2-6 shows a mesh grid with lateral connections. A consideration when adding lateral connections is to avoid introducing directed cycles in the network graph became some inference engines, the Junction Tree engine in particular, cannot work with such graphs.

Dealing with Computational Complexity

In addition to introducing the idea of a pyramidal Bayesian network, Dean also provided an implementation that exploits the structure to speed up the learning process. To illustrate why this is important, consider an input of size 27×27 to a PBN with 4 levels where each node has a receptive field of size 3×3. (Figure 2-4) This results in a total of 820 variables. If we further assume each variable has domain size of roughly 30, the total number of parameters that have to be learned is over 100000. Lateral connection will increase this further. Learning such a network could take months.

Instead, Dean's code decomposes the Bayesian network into subnetworks, with each subnetwork roughly consisting of a node and its receptive field. The algorithm then proceeds to evaluate the parameters of the network by evaluating entire subnetworks starting from the bottom of the hierarchy. For more details on this approach, refer to the *hierarchical expectation refinement* algorithm in [6, 7].

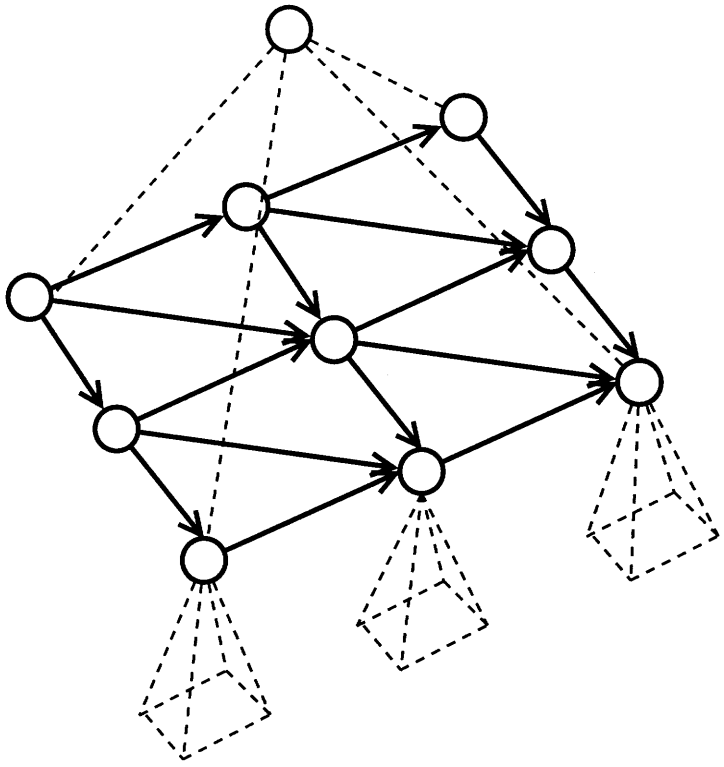

Figure 2-6: *Lateral connection in a pyramidal Bayesian network.* In order to keep the graph associated with the network acyclic, edges can only go in a fixed direction.

The bottom line is that the approach greatly speeds up the process of learning the network parameters when compared to learning the full network directly. It is important to note that this approach is exclusively for optimization and is orthogonal to the entire object recognition problem that PBNs try to solve. In fact, results in [7] suggest a trade-off between speed of computations and performance.

On the other hand, this approach can be parallelized on a large-scale and subsequent implementations of the PBN model do indeed provide support for multiprocessing.

Relation to the Standard Model

Unlike the standard model, the PBN approach has no direct mapping to cortical mechanisms. The PBN classifier cannot be used as a simple replacement of the SVM at the $C2$

stage because $C2$ maps are not probabilistic in nature. However, the layered approach combined with the natural ability of Bayesian networks to encode feedback information make PBNs good candidates for a framework in which to model feedforward-feedback connections. What this framework needs is a way to recast the standard model. We propose a few ways to modify the standard model such that it encodes belief and is, therefore, compatible with the probabilistic nature of the PBN model.

2.5 The Pedestrian and Car Datasets

For evaluating our models, we consider two datasets. The first one is the car database gathered at MIT [29, 2]. This database contains approximately 800 positive images of cars taken from photos of cars from actual street scenes. The cars appear under various illumination conditions, orientations, and occlusions. Some positive examples from the dataset are shown on Figure 2-7. Each image is composed of 128×128, 8-bit, gray-valued pixels. All positive samples have been resized so that the cars are roughly the same scale.

The dataset does not offer negative examples explicitly. To produce those, we took large photos of non-car scenes and from each image extracted numerous random crops of size 128×128. In total, we took about 8000 random samples.

To further separate the data into training and testing, we chose approximately $1/3$ of the samples uniformly at random from the positive and negative pools respectively. We use this part for testing, and other $2/3$ for training. This dataset is used exclusively in Chapter 3.

For Chapters 4 and 5, we use the pedestrian dataset constructed from images taken from both the street scene [29] and the LabelMe [27] databases. The pedestrian examples were manually extracted and the negative examples were randomly cropped from those images. Care was taken during extraction that none of the negative examples overlaps

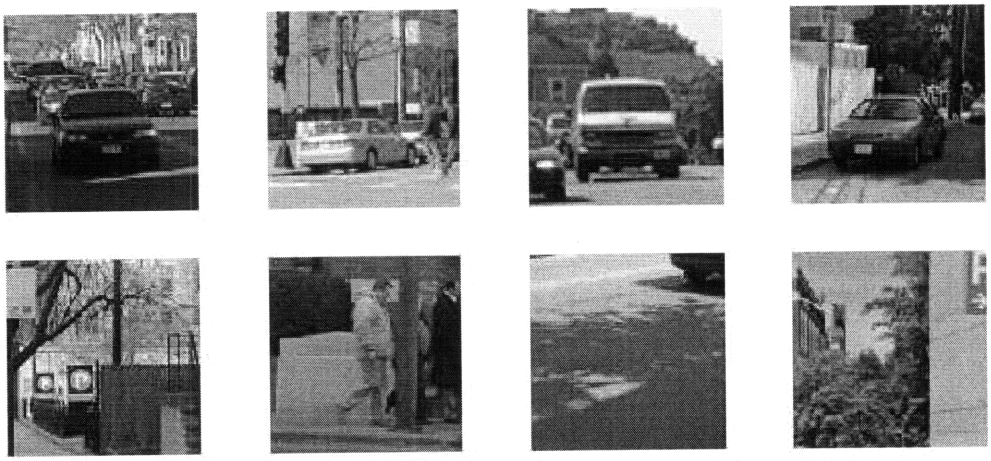

Figure 2-7: *Samples from the car database.* The top row contains positive samples and the bottom row contains negative samples.

with a positive example. In total about 2600 positive and 13500 negative examples were extracted. All examples were subsequently split into training and testing with 2000 positive and 12000 negative training examples. The split was random, but fixed.

Images in the pedestrian dataset are 8-bit, gray-valued images of size 64×128. Examples of images are shown on Figure 2-8.

Figure 2-8: *Samples from the pedestrian database.* All of these samples are positive. The negatives look similar to the negatives on the car database on Figure 2-7.

2.6 Performance Measures

2.6.1 Accuracy

Throughout this work, we evaluate performance in two ways. The first one is the fraction of correctly recognized testing examples, the accuracy. Although a straightforward metric, it is not always descriptive because of the following problems:

- A large discrepancy in the number of positive and negative examples or a bug may lead to a case where all examples are predicted as the same class, yet the accuracy is high.

- Most classifiers including the SVM and the Bayesian network provide more information than just a binary output. Such classifiers usually evaluate a measure of confidence and threshold it to determine the class. Not only is this information discarded in the accuracy measure, but the accuracy is also affected by the choice of the threshold.

2.6.2 Receiver Operator Curve (ROC)

The two problems above are alleviated to some extend by the use of *Receiver Operator Curves* or ROC curves. In essence, these curves represent the fraction of *false positives* (negative examples erroneously recognized as positive) and *true positives* (correctly recognized positive examples) for all possible thresholds. Intuitively, as we increase the threshold we will correctly recognize more examples as positive, but we may also get negative examples with high weights. A sample ROC curve is shown on figure 2-9.

A perfect classifier will always add true positives as the threshold increases up to the point when there are no more positive examples. After that it will always add negative examples. A "classifier" that makes random guesses will add an equal fraction of true positives and false positives as the threshold is increased. Such classifiers look like a

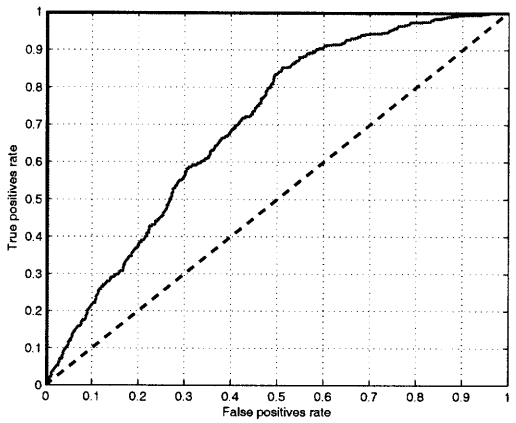

Figure 2-9: *Three examples of ROC curves.* The dashed curve is typical for a fair-coin-toss "classifier". The curve on top is the one for a perfect classifier, whereas the one in the middle shows a middleground classifier. Note also that the area under the ROC curve is also correlated with performance, though we do not use it as a measure in this work.

straight line. Therefore, the shape of the curve is indicative of the overall performance of the classifier.

The point for which the fraction of *false negatives* (positive examples recognized as negative) is equal to the fraction of false positives is called the *equal-error rate*(EER). Sometimes instead of plain accuracy, we provide the accuracy at the equal error rate as a measure. Figure 2-10 shows an example of how to read ROC curves.

2.7 Computational Tools

Both of the two systems described in this chapter are written in the MATLAB programming language and use third party libraries to implement SVM classification and Bayesian network inference, also written in MATLAB. Our implementation is based on the standard model codebase and uses the PBN code as a library. It inherits the third-party dependencies of the systems. In particular, we use *Kevin Murphy's Bayesian*

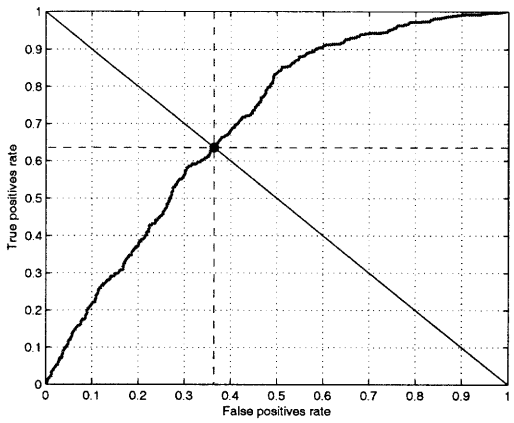

Figure 2-10: *Example of an equal-error rate in ROC curves.* The point indicates the point at equal error rate. The horizontal and vertical dashed lines indicate the true positive and false positive rates respectively.

network toolbox [23] and the open source *OSU SVM* toolbox for MATLAB.

Additionally, we experimented with *Intel's open source probabilistic network library (OpenPNL)* [1], the C++ analogue of Kevin Murphy's toolbox.

All experiments were done on the CBCL's computing cluster and the computing cluster at the Computer Science and Artificial Intelligence Laboratory (CSAIL) at MIT. We have empirically determined that we can regard the work nodes of the two clusters as being of equivalent computational power, especially given that the scale of the running time of our experiments is on the order of hours. Thus, we do not differentiate between work nodes for all quotations on running time. Indeed, running time is of secondary priority for us.

Chapter 3

Hybrid Model

Henceforth, the work we describe is our own attempt in providing a Hybrid model that combines the two approaches from Chapter 2. In particular, the Hybrid model is based on the pyramidal Bayesian network from the PBN model, but replaces the Gaussian Mixture Model (GMM) filtering stage to use $C1$ maps instead of gray-level inputs. This chapter is intended as a preliminary evaluation of the combination and to identify potential problems. The following chapters consider the consequent replacement of GMM with other input filters and modifications to how the Pyramidal Bayesian network is used.

3.1 Initial Evaluation

We start by evaluating the two models out-of-the-box. The goal is to quantify their performance in a similar setting such that we can make comparisons and draw conclusions for their relative performance.

We compare the two models on the car dataset. We also evaluate the standard model on the pedestrian database.

3.1.1 Evaluating the Standard Model

Choosing a Regime of Parameters

In the last chapter we described in detail the workings of the standard model. Here we briefly comment on the parameters of the model.

- *Scale bands.* The parameters of the Gabor filters at the $S1$ stage include the number of orientations, wavelengths, and scales. The number of scales is of particular importance because it determines how susceptible the model is to variations in scale. Because the PBN model does provide scale-invariance, we choose only one band in our experiments.

- *Number of Orientations* As evident from their response, Gabor filters are sensitive to edges at a given orientation. However, images usually have edges of varying orientations at different locations, therefore, it is important for the model to tolerate this variation in orientation. In the standard model $C1$ units carry response information from each of a set of orientations. The number of orientations is a trade-off decision between model performance and computational resource requirements. For this experiment, we use a set of four orientations: vertical, horizontal, and the two slanted orientations at ± 45 degrees.

- *Prototype Size.* The model relies on a precomputed set of visual prototypes called *a codebook.* These prototypes can be of different size. A prototype of size 16×16 pixels will represent a visual feature of significantly higher level than a prototype of size 8×8. For example, a prototype of size 16×16 could describe a significantly large part of a car, whereas a prototype of size 8×8, or $1/4$ of that area, can only match small parts such as a mirror, a tire, etc. A comprehensive model should consider both, but for compatibility with the PBN model, which does not provide a way to recombine prototypes of different size, we only use prototypes of size 8×8.

- *Number of Prototypes.* This parameter is important as it determines the variety of visual features in the codebook. Because prototypes are randomly extracted from $C1$ maps of positive images, having more increases the chance of coming up upon discriminative prototypes. There is no straightforward mapping between number of prototypes in the standard model and number of Gaussians in the Gaussian Mixture Model because the latter are learned and the former are random. Thus, we use a default value of 50 prototypes, but we also provide experiments that vary this parameter.

Empirical Results

Initially, we tried to tap the full potential of the model by testing it on a parameter set similar to the provided examples:

- A total of 14 scales starting from 7×7 to 33×33 in increments of 2. The scales are pairwise grouped in 7 scale bands.
- 4 orientations: at $0, 90$, and ± 45 degrees.
- 4 prototype sizes: 4×4, 8×8, 12×12, 16×16
- 250 prototypes for each prototype size.

The resulting classifier has an accuracy of 92.75% on the car dataset and 91.2% on the pedestrian dataset.

The following experiments each try to see how different scale bands and prototype sizes contributed to the result by examining the performance of the model when restricted to those particular scale bands or patch sizes. The results are shown on Tables 3.1 and 3.2. Overall, there does not seem to be a significant difference in using any particular combination of scaleband and patch size. In order to be compatible with the PBN model,

Band	Gabor filter size (scales)	Accuracy(car)	Accuracy(pedestrian)
Band 1	7×7, 9×9	92.21%	82.67%
Band 2	11×11, 13×13	92.35%	84.57%
Band 3	15×15, 17×17	91.57%	85.59%
Band 4	19×19, 21×21	93.81%	85.40%
Band 5	23×23, 25×25	93.31%	84.52%
Band 6	27×27, 29×29	92.92%	84.57%
Band 7	31×31, 33×33	93.10%	85.03%

Table 3.1: *Comparison of individual scale bands on the standard model.* Here we use only patches of size 8×8. There are a total of 7 scale bands starting from the smallest to the largest. Recall that each scale band is assembled from two neighboring scales at the $C1$ level.

Patch size	Accuracy(car)
4×4	93.95%
8×8	92.92%
12×12	91.93%
16×16	92.64%

Table 3.2: *Comparison of individual patch sizes on the standard model, car dataset.* We only use scale band 1.

we will use scale band 1 and patch size of 8×8. (Recall that the PBN model cannot pool across different patch sizes or scale bands)

For the final set of experiments, we try to establish a relation between the number of prototypes and the performance.

3.1.2 Evaluating the PBN Model

Parameter Setup

The source code distribution of the PBN model provides an example setup for the MNIST database of handwritten digits[18]. We will not go into the details of that dataset since it is not relevant to our research, but we base our parameter setup on the MNIST example.

- A Gaussian Mixture with $n = 16$ is trained on patches of input images of size 8×8.

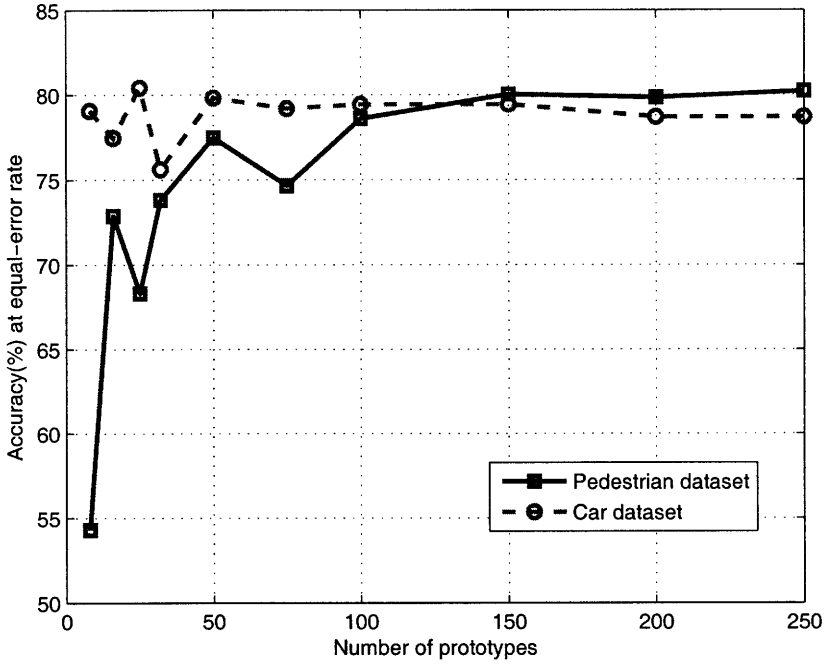

Figure 3-1: *Recognition accuracy of the standard model (band 1) at the EER on Pedestrian and Car datasets.* This shows the EER performance on $C2$ maps. Note that because of the random extraction, the proper procedure requires providing error measures which we omit here.

- The patches are treated as 64-dimensional vectors. The Gaussian functions in the mixture are themselves 64-variate.

- All input examples are filtered by an arg max filter that finds the index of the "closest" Gaussian to a given patch. (Here closeness is defined as the maximum response of the multivariate function evaluated on the patch). The filter is applied at every 18-th location yielding a 7×7 map over the integers $1 \ldots 16$. This serves as the input to the Bayesian network classifier.

- The Bayesian network consists of three layers. The top layer is the root node which is a Bernoulli random variable representing the probability of each of the

two classes. The root node has a receptive field of 3×3. Each of the child nodes is again a random variable over the domain $1 \ldots 10$. Their receptive fields are also of size 3×3, but they overlap by one node, thus the bottom, input layer is of size 7×7 instead of 9×9. A diagram of the network is shown on Figure 2-4.

- Finally, the GMM learns n multivariate Gaussians which are described by a set of mean vectors and covariance matrices, one pair per Gaussian. Since the covariance matrix is larger than the mean vector by a power of 2, learning it takes significant effort and is likely to be incomplete for the small number of samples. To solve this, GMM offers two modes of learning. In the first mode, the entire matrix is learned, but in the second mode, the matrix is assumed to be diagonal (the dimensions of the Gaussians have independent variances) and only the diagonal is learned. We did not notice any significant difference in results. The results shown are for the case where the entire matrix is learned.

Although these parameters were directly adapted from the ones used with the MNIST dataset, that dataset is of size 28×28. The car dataset is of size 128×128 and it is not appropriate to downsample the input map to 7×7 input because the sampling step of $128/7 \approx 18$ is twice as large as the size of the prototype, 8. A Bayesian network trained this way will learn relations between neighboring visual features that are 18 pixels apart. Obviously, this problem does not exist with the MNIST database, because $28/7 = 4$.

One solution is to downscale in advance the entire dataset to 28×28. This introduces another discrepancy, however. A patch of size 8×8 will capture visual feature of significantly higher level than a patch of the same size used in the standard model, because the standard model does not downscale the dataset. Effectively, this will make the two results incomparable.

A better solution is to downscale the dataset such that the sampling step is managable, but upscale the patch size for the standard model. For example, if we rescale the input to 64×64 yielding a sampling step of 9 with a patch size of 4, this will correspond

to a prototype of size 8 for the standard model. A variation on this approach is to combine downsampling with cropping the central part of the image. Cropping presents the trade-off of reducing the sampling step but at the cost of discarding data from the input proportional to the square of the reduction ratio.

We show results for different experiments and variations in the parameter space. The goal is to present a comprehensive evaluation of what may work and what may not.

Results

Input size	Patch size	Number of Gaussians	Accuracy
128×128	8×8	16	30% (!)
64×64	8×8	16	30% (!)
64×64	4×4	16	67.13%
64×64	4×4	32	68.38%
64×64	4×4	50	68.87%
64×64	4×4	100	69.12%
64×64	4×4	200	67.87%

Table 3.3: *Empirical results from the PBN system on the car dataset.* The presence of (!) next to the performance measure indicates a problem with the setup. Below we provide explanation of these problems.

Table 3.3 presents the results from the evaluation of the PBN system on the car dataset. We have marked some of the experiments that produce likely erroneous results. On close inspection, we notice the primary problem of GMM: the large dimensionality (64, in the case of 8×8 patches) leads to numerical instabilities. Appendix A explains in more detail why this occurs. At this point, however, we observe that patches of size 4×4 do not exhibit this problem, while patches of size 8×8 do.

3.2 Description of the Hybrid Model

The Hybrid model we propose combines the best aspects from the two paradigm models: the $C1$ maps from the standard model and the hierarchical structure of the Bayesian

networks in the PBN model.

In this chapter, the only change we made to Dean's model is that we train the GMM on crops of $C1$ maps instead of gray-valued patches.

We now proceed to examine the changes necessitated by this decision. Recall from the previous chapter, that $C1$ maps are the peak responses of Gabor filters for each combination of scale band, orientation, and spatial position. As we elaborated earlier, we only use one scale band as we completely disregard the problem of scale by assuming that recognition objects appear at the same scale throughout the input images.

As for the orientations, we use the default number from the standard model of 4 orientations. Because GMM uses crops of $C1$ maps the dimensionality of each crop is increased by a factor of 4 to 64 for patches of size 4, and 256 for patches of size 8. This is already in the range where we observe numerical instabilities. (See Table 3.3 and Appendix A)

Instead, we propose a solution that reduces the dimensionality at the expense of losing some information. The method is called principle component analysis (PCA) and involves projecting the d-dimensional input space to a k-dimensional subspace in a way that selects the k basis vectors to be the directions with the greatest variance in value. In other words, the information from the other $d - k$ dimensions is discarded. The success of this method depends on whether the $d - k$ discarded dimensions contain potentially useful information or have low variance.

Results from PCA

The PCA method finds a covariance matrix that describes the variance of data along each dimension. The eigenvalues and eigenvectors of this matrix are the directions with the greatest variance. Figure 3-2 shows the "amount" of variance explained by including each subsequent dimensions starting with the one with the greatest variance.

There is little sense in trying different values for k. Reducing the number of di-

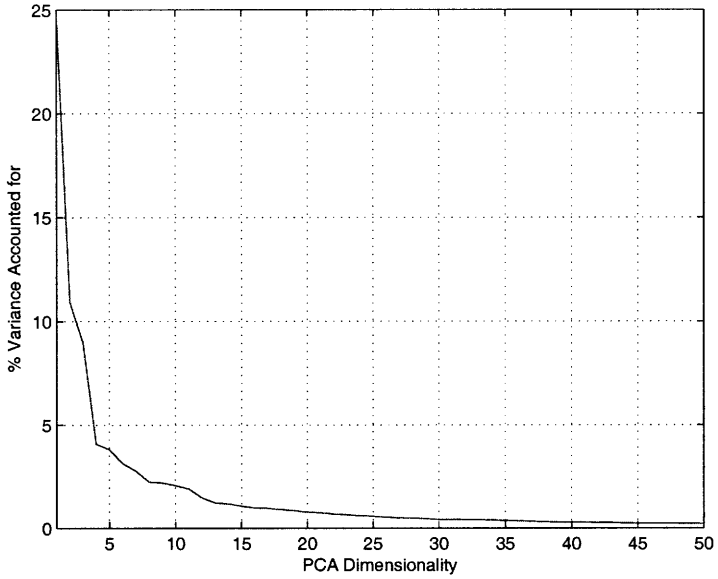

Figure 3-2: *The amount of variance that each subsequent dimension from PCA explains.*

mensions will only deteriorate the performance. The best choice for k is the highest value that does not exhibit the problem which necessitated PCA, the numerical overflow/underflow due to the exponential dependence on the number of dimensions. Our choice of value is 16, because it is already known to work and because adding more dimensions does not increase the explained variance significantly (see Figure 3-2).

If we compare results from Tables 3.3 and 3.4, we see a slight improvement by using $C1$ maps in the GMM. The results, however, are still significantly below the standard model.

Input size	Patch size	Number of Gaussians	Accuracy at the EER
64×64	$8 \times 8 \times 4$	8	77.34%
64×64	$8 \times 8 \times 4$	16	77.22%
64×64	$8 \times 8 \times 4$	32	72.99%
64×64	$8 \times 8 \times 4$	50	65.59%
64×64	$8 \times 8 \times 4$	100	40.9% (!)
64×64	$8 \times 8 \times 4$	200	37.5% (!)
64×64	$4 \times 4 \times 4$	16	70.81%

Table 3.4: *Empirical results from the Hybrid system using PCA.* Evaluated on the car dataset. Results marked with (!) indicate problems.

3.3 Identifying the Problems

3.3.1 The Need for Better Codebooks

We already mentioned the numerical instabilities that occur during GMM filtering. (see Appendix A for more technical explanation) Instead of working around the problem, we make the decision to abandon Gaussian Mixture Models altogether. With that, we also acknowledge the need for a codebook replacement.

To justify our decision, we mention that in computer vision other approaches such as AdaBoost selection or random sampling are usually preferred to GMM. The next chapter examines these and other approaches for generating codebooks.

3.3.2 Problems with Learning the PBN

Switching from the MNIST to the car dataset does not only change the size of the input, but also the number of samples. While the MNIST database contains upwards of 50000 training examples, the car dataset only has 800. It is clear that a model trained in this way will not receive enough diversity in the input to exercise all of its parameters and will only remember the few samples it has seen during training. The model will only rule a positive class for inputs that are very close if not matching one of the examples seen during training. This is referred to as overfitting and is considered a serious problem.

The evidence for this is the near zero elements in the condition probability tables for the nodes in the network. Having priors that are practically indistinguishable from zero makes it impossible for the model to influence this prior in any way.

Lack of training examples is only one source for this problem. The domain size of the input layer is what expands the size of the probability tables. We already saw evidence in Table 3.4 of poor results when the domain size is increased.

Luckily, there are many opportunities that may lead to solutions. In particular, the vast space of settings for the network layout of the PBN model is appealing. With these opportunities comes a set of challenges in actually finding ideas that work and produce solutions. Even if all ideas fail, switching to a dataset with lots of examples is another alternative. Chapter 5 is devoted to exploring these ideas.

Chapter 4

Finding Good Codebooks

In this chapter we examine three algorithms for generating codebooks and filtering the input stimuli. We also propose a scheme to quantitatively evaluate each method independently of the Bayesian network.

4.1 Overview

Ultimately, the goal of this processing stage is to provide a function that transforms $C1$ maps into matrices over the $[n]$ domain ($[x] = \{1, \ldots, x\}$). We call such function *a filter*. For example, GMM is the defining example of a filter in our particular problem. For the purposes of the Bayesian network, we consider the outputs of this function to be observations of a random variable over a discrete distribution, $[n]$.

The use of a library of visual features (the codebook) introduces a type of random variables which domain is over the prototypes in the codebook. The probability mass function of such variables denote how close the patch it represents is to each prototype in the codebook. The PBN input layer (filtered with a GMM) is composed of exactly this type of random variables. The observed value of this variable is the "closest" Gaussian to the patch at each location.

Providing a working definition for *codebook* and *filter* is possible once we adopt this framework. A *codebook* is the set of visual prototypes/features that defines the domain of the random variables in the input layer of the Bayesian network. Conversely, in the context of a codebook, a *filter* is a function that assigns a value (an observation) to these variables given a codebook and a patch.

To better illustrate the parallel, we will now show how to recast the $S2$ units in this framework. $S2$ units are spatial maps of the responses of the patches at each location for each prototype. Consider the index of the prototype with the maximum RBF response at each spatial location, Figure 4-1. It is not difficult to observe that the resulting map fits in our framework. The codebook is the set of prototypes which are randomly sampled from $C1$ maps of positive training examples. The filtering method is the application of the RBF function at each location and the subsequent selection of the prototype with the maximum response, the arg max operator.

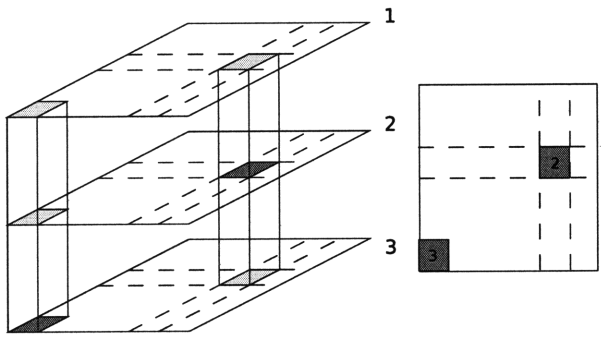

Figure 4-1: *An example showing how to transform $S2$ maps into index $S2$ input.* To the right are the $S2$ maps, one map per codebook prototype. All $S2$ maps are RBF responses. To the right is the index $S2$ map where each location represents an index of the codebook with maximum response.

It is important to mention that the $S2$ map computed this way can now serve as input to a Bayesian network, but is no longer applicable for use with the SVM classifier. The reason is that its values no longer define a metric space. Indeed, there is no way to compare two values that represent the indices of two prototypes. The prototypes

themselves are in no special relation. To emphasize the difference, we call this type of maps *index maps.*

4.2 Input Filters

We present two filtering algorithms. The first one is a straightforward transformation of $C1$ maps into index input without the use of codebooks. The second algorithm is the example from above: the application of arg max on the $S2$ maps over the prototypes in the codebook.

Given the parallel with $C1$ and $S2$ units, we refer to these as the index $C1$ and $S2$ maps, respectively.

4.2.1 Index $C1$ Maps

This filter does not require codebooks because the random variables are not over codebooks, but over the domain of possible orientations at the $C1$ stage. Since the $C1$ map is a spatial map of Gabor filter responses for each orientation, we can choose the observed value of this variable to be the orientation with the maximum response. Effectively, the random variable encodes the probability of an edge of a given orientation at each location.

This is one of the simplest filters and has the added benefit that the domain is of small size. For our tests we used 4, 8, and 16 orientations. All orientations are encoded as directional angles of the form $k\pi/n$, where n is the number of orientations and $k \in [n]$ is the k-th orientation. Note that directions x and $x + \pi$ represent the same orientation.

4.2.2 Index $S2$ Maps

Recall the computations at the $S2$ stage. Given an input stimulus and a prototype, the *windowed patch distance* is the squared L_2 distance at each location between the

prototype and an image crop at each location. It is the result of sliding a window with the size of the prototype over the input map and only considering the patch below the window.

The RBF function is of the form $f(x,y) = e^{-||p-I(x,y)||_2^2}$ and can be computed directly from the windowed patch distance, which is $g(x,y) = ||p - I(x,y)||_2^2$. Usually, explicit computation of the RBF is not necessary because substituting $g(x,y)$ for $f(x,y)$ only changes the pooling operator from (arg) min to (arg) max. Therefore, we can argue equally well about maximum response to an RBF or minimum L_2 distance.

Often the filter output has to be downsampled to fit the layout of the input layer of the Bayesian network. This downsampling uses the same procedure as the downsampling of $C1$ maps.

4.3 Evaluating Codebooks

The primary goal of this chapter is to evaluate codebook algorithms independently from the Bayesian network. We do this using a linear SVM classifier on the $S2$ units. As mentioned earlier, SVM classifiers cannot work with index maps so we compute separately both the index (Bayesian net compatible) and the distance (SVM-compatible) maps. We always use the appropriate one for each experiment.

For this chapter we show experiments done exclusively on the pedestrian dataset.

4.4 Codebook Algorithms

Here we present three approaches of generating codebooks. All of them work on $C1$ maps. Although they can just as easily work on graylevel inputs, we explained why adhering to the neurobiological plausibility of the project requires the use of $C1$ units.

Throughout this section we assume that n denotes the number of visual features in

the codebook and p denotes both the width and height of the feature. We assume that visual features are square, of dimensionality $f \cdot p^2$, where f is the number of orientations in the input $C1$ map. Obviously, prototypes in the resulting codebook need to retain the orientation information kept as an extra non-spatial dimension in $C1$ maps. Hence, the prototypes are themselves $C1$ maps. This is what makes possible the computation of the RBF response: the patch underneath the window and the prototype must be of the same shape.

4.4.1 Random Sampling

Random sampling is the most intuitive approach. Given a set of $C1$ maps, one map per input stimulus, the algorithm picks n patches of size $p \times p$ such that all patches from all images have an equal probability of being selected. Only positive training examples are used in the process. Random sampling is effectively what the standard model uses to select the visual prototypes.

The benefit of this approach is that it is very fast. Additionally, the visual features in the codebook are guaranteed to be "proper" in the sense that their values were not generated, but represent parts of real images.

On the downside, selecting patches at random reduces the quality of the codebook. Indeed, the chance of picking out a feature that is well-represented for positive images and uncommon in negative images is very small. This problem is alleviated to some extend for large values of n.

Figure 4-2 shows the accuracy of the algorithm for different values of n. Because of the random nature, results vary from run to run. We collected the results from 5 runs and took the average accuracy. The standard deviation is shown as error bars.

Because of the similarity between randomly-sampled codebooks evaluated on the SVM classifier and the standard model a comparison between the two is possible. Essentially, the difference in the two approaches is that the standard model uses $C2$ maps

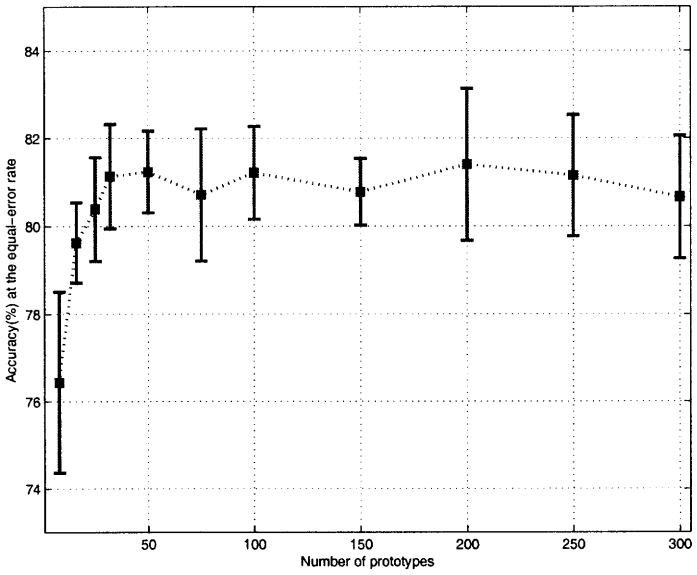

Figure 4-2: *Performance of random sampling as a function of the codebook size.* This plot show the accuracy on the testing set evaluated on $S2$ maps using an SVM classifier.

as input to the classifier whereas $S2$ maps are used for evaluating the codebooks in this chapter. Figure 4-3 clearly shows that using $S2$ maps produces better results. However, this is not an indication that the standard model's performance is poor. Recall that $S2$ maps have position information which $C2$ maps do not have. This effectively increases the number of features available to SVM by a factor, the size of the $S2$ maps. Additionally, the standard model is scale-invariant, but our datasets do not offer scale- or position- invariance, so in this case $C2$ do not have real advantage.

4.4.2 GentleBoost Selection

Boosting is a popular machine learning technique that improves the performance of simple classifiers, called *weak learners*, by creating a linear combination of them. In their simplest form, weak learners are thresholding classifiers of the form $X_f > th$ or

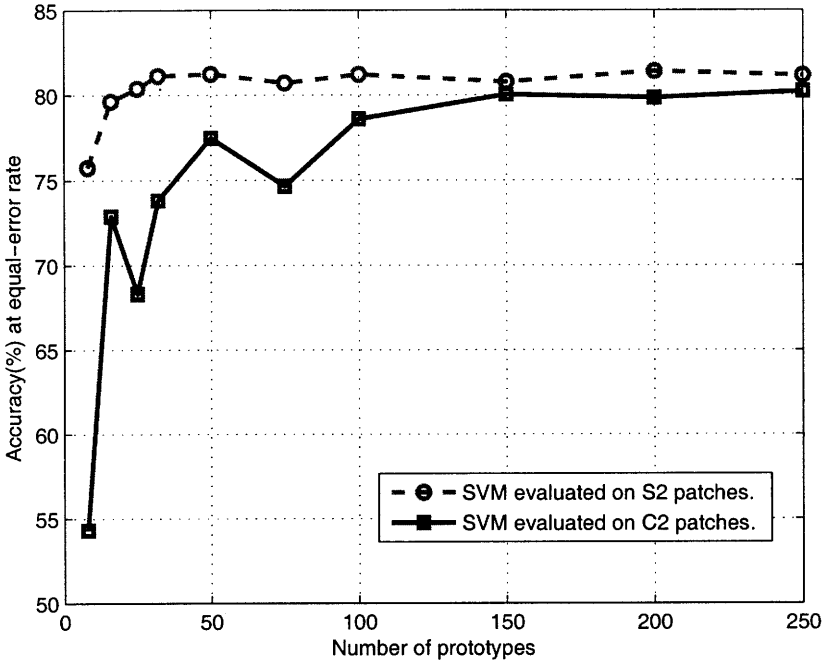

Figure 4-3: *Comparing the performance of $S2$ and $C2$ maps.* Here we show the accuracy at the EER of both $S2$ and $C2$ units evaluated on codebooks of varying size. Note that the combination of an SVM classifier, random codebook selection, and the use of $C2$ units is exactly what the standard model uses.

$X_f < th$ that predict positive class when the value of a certain feature, f, is above (resp. below) the threshold th. It is not difficult to show that for each training data vector, there exists a weak learner with at least 50% accuracy on the data. The power of boosting is that it incrementally adds to classifier weak learners with at least 50% accuracy and assigns voting weight to each learner, the coefficient in the linear combination. It also reweights training examples in such a way as to put more emphasis on examples that were miscategorized by the current set of weak learners. Thus, each subsequent weak learner is good on the examples that previous learners miscategorized. More detailed description of boosting is available in [10, 11].

Different implementations of boosting vary in the details. The two most commonly

used implementations are AdaBoost[10] and GentleBoost[11]. For our experiments we choose the GentleBoost algorithm.

We use boosting to augment our random sampling method. As we mentioned above, selecting more random patches diversifies the codebook and increases the chance of coming upon good visual features. However, having a lot of patches just for the few good ones is not desirable – it increases the computational complexity and, more importantly, the size of the parameter space of the Bayesian network at the next stage.

We use GentleBoost to select the best n patches from a big pool of $N(N >> n)$ randomly select patches. In particular, we use the $C2$ maps generated on the N patches on the training set. In our experiments we use $N = 1500$.

The selection process exploits the fact that GentleBoost assigns weights to weak learners and since each learner operates on one feature, the weight of the learner indicates the importance of the feature. This gives us a direct mapping between weak learner weights and patch importance. In the end, the top n patches are selected for the final codebook.

Note that while it makes sense to extract patches only from positive training examples, evaluating the quality of those patches needs to be done on both the positive and negative training set – boosting is a classifier itself and needs at least two classes to work with.

4.4.3 Jurie-Triggs Codebooks

The final method for generation of codebooks was presented by Jurie and Triggs [16]. It combines ideas from all of the methods presented here and tries to overcome some of their limitations.

In explaining the idea of the algorithm it is useful to refer to the k-means problem. The k-means problem can be formulated this way: given a set of points, $x_i \in \mathbb{R}^d, i = 1, \ldots, m$, find their disjoint partition in k clusters, $x_i \in S_j, j = 1, \ldots, k$, such that the

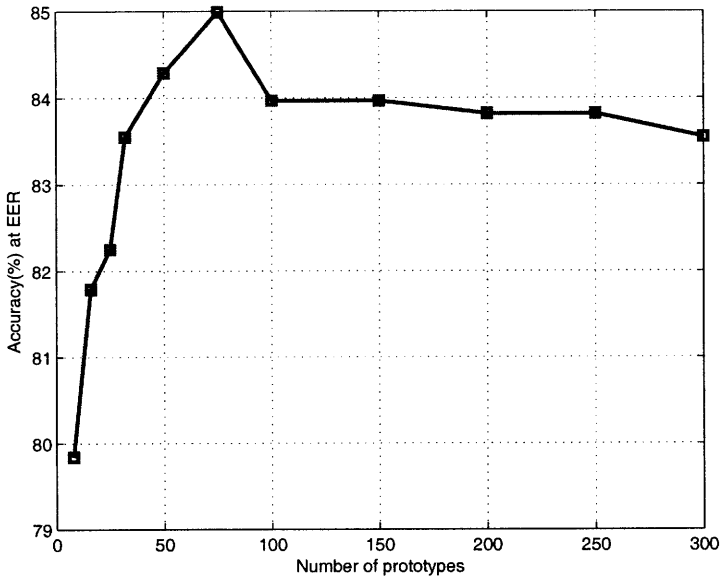

Figure 4-4: *Performance of GentleBoost codebooks as a function of the number of prototypes.* This plot show the accuracy on the testing set evaluated on $S2$ maps using an SVM classifier.

sum of the L_2 distances from each point to the mean of its assigned cluster is minimized. That is, minimize the function $\sum_{j\in[k]} \sum_{i,x_i\in S_j} ||x_i - \mu_j||^2$, where $\mu_j = \sum_{i,x_i\in S_j} x_i$. This problem is known to be NP-hard, so in practice the cost function is approximated. What relates k-means to our work is that the GMM is a generalization of it in some respect. Consider the distance function. In k-means the distance function, L_2 distance, is symmetric in all directions around the cluster center, i.e. all points at a given distance form a high-dimensional sphere. In GMM, all points "equidistant" from the center form an ellipse and its semi-axes are determined by the covariance matrix.

What is common between k-means and GMM is that both spread out to infinity. In case there is a point far from the rest, it will affect the center of the cluster it is assigned to. Worse, it may be assigned an individual cluster. This is what Jurie and Triggs try to solve by limiting the radius of the cluster to a fixed value r. Their algorithm works

as follows:

1. The input images are densely sampled. In our implementation we consider all possible crops of the given patch size.

2. At each iteration N patches are randomly selected from the pool. A mean shift algorithm (described below) is performed on them and a cluster center is found.

3. Once the algorithm finds one cluster center, it never changes it. All unassigned patches that are within radius r from the center are assigned to the cluster and removed from the pool. After that no new members can join the cluster because of the limit on its radius. The algorithm proceeds to the next cluster.

4. After k iterations or until there are no more patches in the pool, the cluster finding process stops. Usually, k is chosen to be larger than the number of patches. This requires a selection process to pick the best n patches.

5. The authors suggest several approaches. The simplest one is to train an SVM classifier on the $C2$ maps and use the weighted sum of the support vectors to find which patches are important. Our suggestion is to use GentleBoost. Figure 4-5 shows there is a slight gain in using GentleBoost selection.

The Jurie-Triggs (JT) algorithm is incremental in nature and once a cluster center is selected it and its cluster are never changed afterwards. The process can continue for as long as there are input patches left. In practice, though, we set a predefined limit k to use for the pool size.

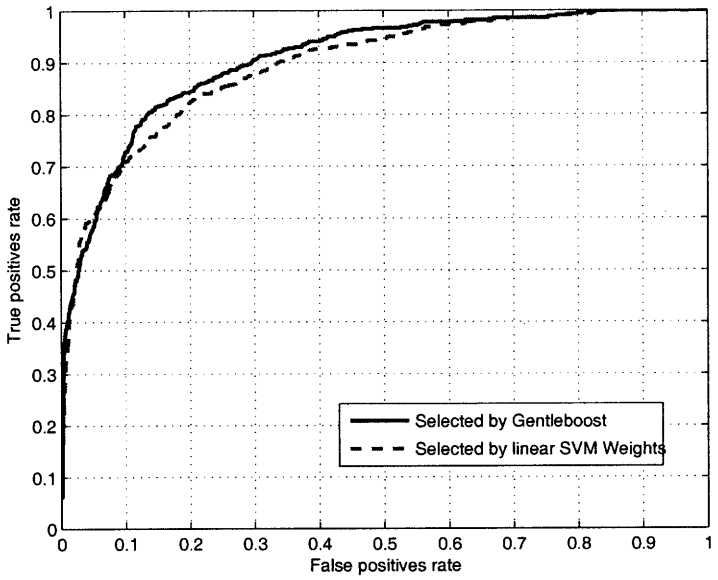

Figure 4-5: *ROC curve comparison between GentleBoost and SVM weights in the selection step of the Jurie-Triggs algorithm.*

Mean Shift Clustering

The mean shift procedure by Comaniciu and Meer [4] is a density estimator that given a kernel $K(x)$ and set of datapoints $x_i \in \mathbb{R}, i = 1, \ldots, N$ computes the density function:

$$\hat{f}(x) = \frac{1}{Nh^2} \sum_{i=1}^{N} K\left(\frac{x - x_i}{h}\right)$$

In loose terms, the value of the function is a good indicator of how "populated" the region around the point x is. Overall, the peak of the function is the place with the maximum density and is the best candidate for a cluster center.

The algorithm we use for finding the maximum of $\hat{f}(x)$ is a hill climbing along the gradient vector [4, 13]. This is the algorithm that we use. Figure 4-6 shows a mean shift iteration done by this algorithm.

There are two paramaters that need to be selected: the bandwidth parameter, h and

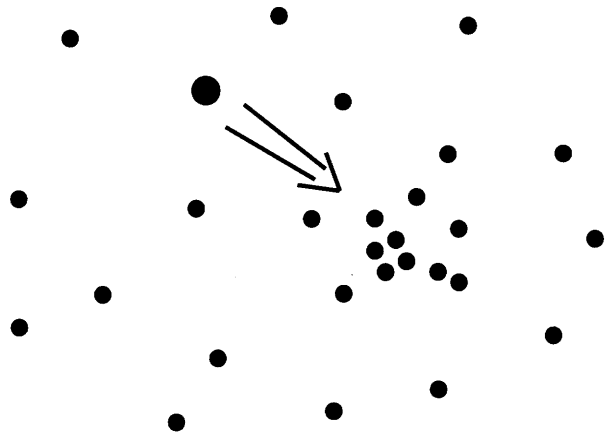

Figure 4-6: *An illustration of a mean shift iteration..*

the kernel function. While the bandwidth is a true parameter, Comaniciu and Meer suggest the use of radially symmetric kernel. The kernel that Jurie and Trigs use and that we adopt is the multivariate Gaussian (normal) kernel:

$$K(x) = (2\pi)^{-d/2} \exp\left(-\frac{1}{2}||x||^2\right)$$

Evaluation

The main four parameters of the algorithm are r, the cluster radius; N the number of patches sampled from the pool; h, the mean shift bandwidth; and k, the pool size.

The paper [16] suggests setting $h = r$. In one of the examples, Jurie and Triggs use $h = r = 0.8, N = 1000, k = 2500$. We use a very similar setup, but we decrease k to about 1500 because instances with larger values for k occasionally ran out of computer memory.

We also decided to change the mean shift bandwidth to $h = 0.1$. In our experiments using $h = r = 0.8$ proved to produce poor codebooks because the Gaussian kernel has very high spread for $h = 0.8$. This yielded cluster centers that captured the entire set of N vectors resulting in the same patch being selected over and over again.

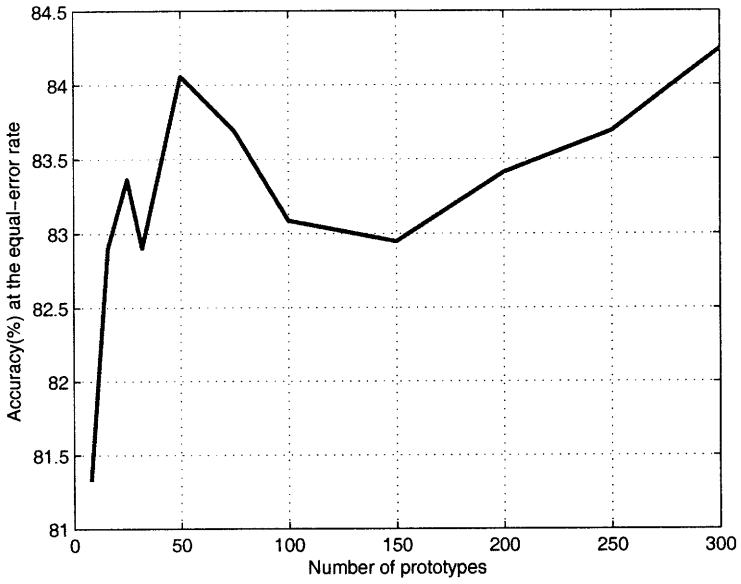

Figure 4-7: *Performance of JT codebooks.* This plot show the accuracy on the testing set evaluated on $S2$ maps using an SVM classifier.

4.5 Comparison of Codebooks

What is left at the end is to compare the three algorithms and decide on which one to use.

Figure 4-9 show how performance improves with the number of prototypes and with the algorithm. As expected, the random sampled codebooks underperformed the other two learned codebooks, though, not by a large margin as one would have expected.

Selection method	Training time	Accuracy (at EER)
Random Sampling	< 1 min	80.7% (averaged)
GentleBoost	≈ 15 minutes	83.5%
JT codebooks	≈ 330 minutes	84.2%

Table 4.1: *Running time in constructing the codebooks.* The table also shows the accuracy at the EER. Comparison is done on 300 prototypes to exhibit the steady-state performance of the codebook learning algorithm.

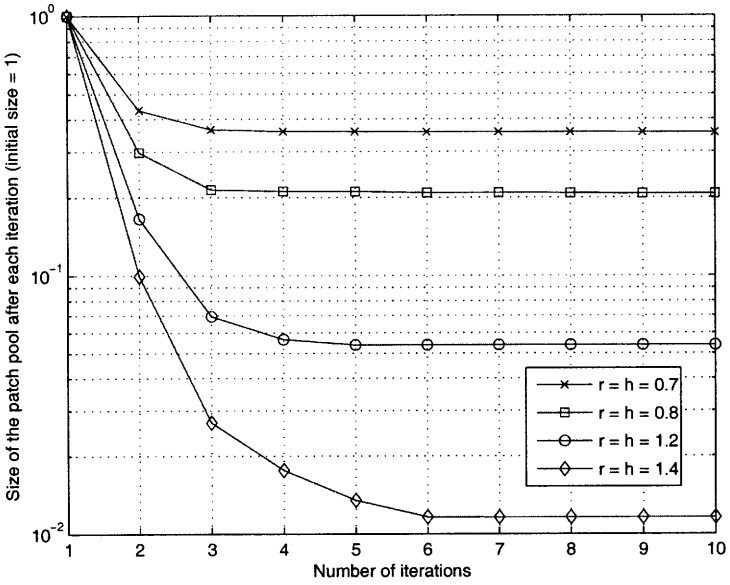

Figure 4-8: *JT codebook training for different values of* r. This figure shows the size of the pool with the number of iterations for different values of r.

As for the other two methods, it is a bit disappointing that JT codebooks, being one level more sophisticated, do not outperform GentleBoost selected codebooks for higher number of prototypes. It is expected, though, that with small number of prototypes JT codebooks outperform GentleBoost because of the extra computational effort put into finding each prototype. Evidently, after about 50 prototypes the amount of effort put in the pool selection process has diminishing returns.

To conclude this chapter and our analysis of codebook algorithms, we can claim that overall GentleBoost is the better method for codebook selection given its performance and running time.

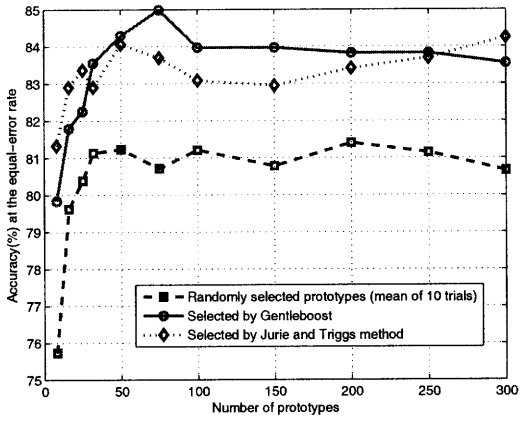

Figure 4-9: *Comparison between codebook generation algorithms.* This figure shows the accuracy at the equal error rate as a function of the size of the codebooks for each algorithm.

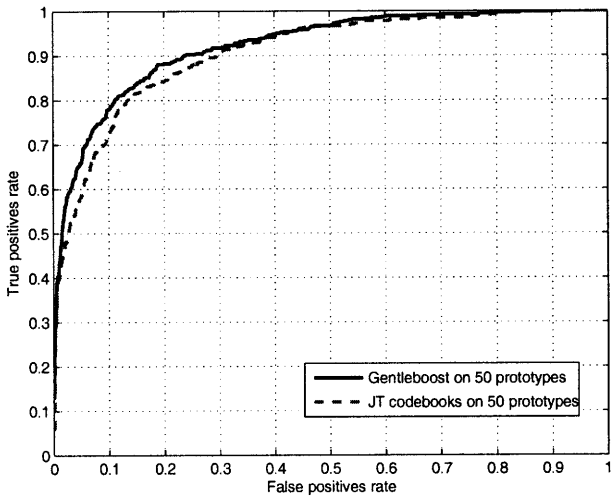

Figure 4-10: *ROC curves to compare GentleBoost and JT codebooks evaluated with S2 maps on a linear SVM classifier.* The number of prototypes in both cases is 50.

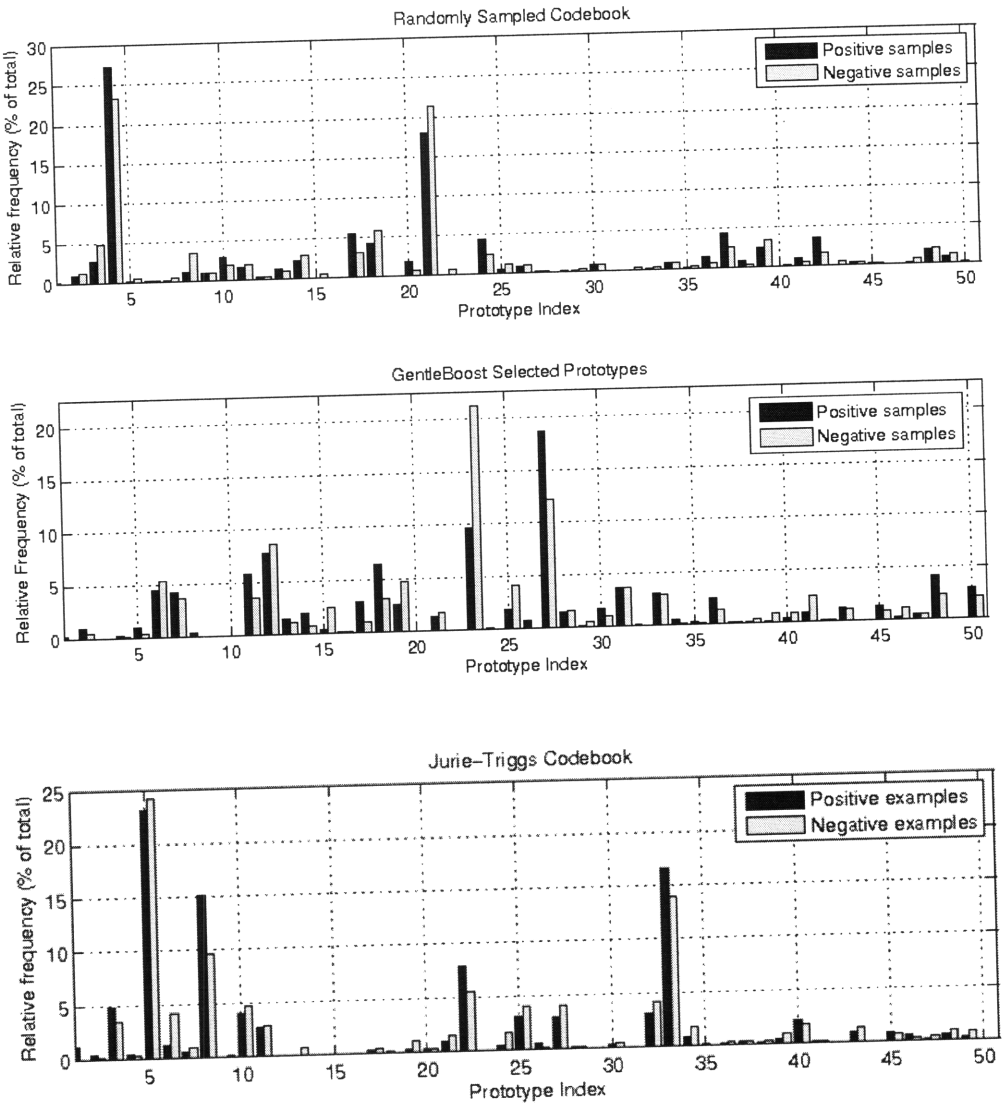

Figure 4-11: *Relative Frequency of occurrence of each prototype in the codebook.* Data is shown for all three codebook selection algorithms: random sampling (top), GentleBoost selection (middle), JT codebooks (bottom). Appendix B contains a more detailed version showing the actual patches.

Chapter 5

Tuning the Pyramidal Bayesian Network

This chapter advances our work from Chapter 4 where we reached to a decision for a codebook selection algorithm. Here we examine the use of this codebook algorithm in the Bayesian network model.

5.1 Bayesian Network Layout Design

All experiments in this chapter use one of the two Bayesian network layouts presented here. This is because the number of degrees of freedom in choosing the network layout is substantial and is beyond the scope of this thesis work.

The first layout, *Layout A* , was directly adapted from the car dataset experiments in Chapter 3. Ultimately it traces back to Dean's MNIST example. The second layout, *Layout B* , is the one we designed with the goal of overcoming some of the problems of the first layout.

The three main considerations in designing the layouts are the width of the receptive field, the total size of the parameter space of the probability distribution, and the shape

of the dataset images. These problems are explained in the next few paragraphs.

5.1.1 Receptive Field Size

The Pyramidal Bayesian network takes its name from the pyramidal shape of the receptive fields. Each layer of the network except the input one at the bottom consists of units with receptive fields in the layer below. Units respond based on the response patterns in their receptive fields. The number of possible response combinations each unit has to learn is directly affected by the receptive field size. However, this is not the only variable here. In the PBN model nodes/units are probability distributions over a finite domain. The size of the domain is the second direct factor in the number of combinations. For a receptive field of size $W \times W$ and domain size n, the total number of combinations for the receptive field is n^{W^2}.

5.1.2 Parameter Space Estimation

The total size of all probability mass tables in the network is what we call the *parameter space* of the network. This is because the network has to learn all of these values in order to be able to perform inference. This number, however, is not the straightforward sum of the domain sizes of each variable because of the complex conditional dependencies among the nodes of the network with each conditional dependence increasing the size of the probability table by a factor of the domain size of the conditioned variable. For example, while two random variables x, y over the domain [10] require 10 values each to describe their probability mass functions (in fact only 9 values are required, because all probabilities must sum up to 1), describing their conditional dependence $x|y$ requires a table of size 100, the values of $p(x|y)$ for each x and y. This does not imply that conditional dependencies have negative impact because they increase the parameter space. As we saw in Chapter 2, Figure 2-3 the parameter space of the joint probability distribution is orders of magnitudes larger than the parameter space of the network with

conditional dependencies.

5.1.3 Dataset Shape Consideration

Finally, the network should not change the aspect ratio of the input significantly. The pedestrian database has an aspect ratio of $2:1$. The $C1$ maps, apart from having an extra dimension for the orientations, are downsampled to size 30×14 for an aspect ratio of ≈ 2.14. This is the ratio we try to maintain.

5.1.4 Network Layouts

The two network layouts that we describe consist of three layers each: one root/output, one hidden, and one input layer. The presence of lateral connections is a separate parameter of the experiment. With the input layer containing independent (for the Bayesian network) units and the root layer consisting of only one node, the hidden layer is the only appropriate place for lateral connections.

Following is a description of the layouts along with a table that summarizes their properties.

Layout A

This layout is inherited from the original layout for the MNIST database in Dean's PBN code and from the very similar layout used in Chapter 3 for the car dataset. The original layout features a 7×7 input layer divided into receptive fields of 3×3 overlapping by 1 unit in each dimension (see Figure 2-5). The hidden layer has a size of 3×3 and is the receptive field for the root node in the input layer.

To keep the aspect ratio close to that of the $C1$ maps that serve as input for the filtering stage, we modify the above network by removing 2/3 of the units in the hidden layer. This makes the size of the input layer 3×7 (still keeping the overlap along the

Network	Lateral connections	Codebook size	Parameter space size	RF combinations	
				Hidden Layer	Root node
Layout A	No	16	12062	$16^9 = 2^{36}$	$10^3 \approx 2^{10}$
Layout A	No	50	37562	$50^9 \approx 2^{50}$	$10^3 \approx 2^{10}$
Layout A	Yes	16	12422	$16^9 = 2^{36}$	$10^3 \approx 2^{10}$
Layout A	Yes	50	37922	$50^9 \approx 2^{50}$	$10^3 \approx 2^{10}$
Layout B	No	16	2642	$16^4 = 2^{16}$	$4^{10} = 2^{20}$
Layout B	No	50	8082	$50^4 \approx 2^{22}$	$4^{10} = 2^{20}$
Layout B	Yes	16	4778	$16^4 = 2^{16}$	$4^{10} = 2^{20}$
Layout B	Yes	50	10218	$50^4 \approx 2^{22}$	$4^{10} = 2^{20}$

Table 5.1: *Description of all layouts and their parameters.*. *Parameter space size* is the size of the probability distribution tables, the actual number of free parameters the Bayesian network has to learn. *Receptive field (RF) combinations* refers to the number of different input configurations the units in each layer can observer from their receptive fields.

longer dimension) and the size of the hidden layer 1×3, Figure 5-1. The aspect ratio is ≈ 2.33.

For the domain size of the hidden layer we choose 10, we tried smaller values such as 2 and 4 without success.

Layout B

One problem with *Layout A* is the high number of parameters. It turns out that the overlap region quadruples the number of parameters. Another problem with *Layout A* is the enormous number of receptive field combinations for the hidden layer. We designed *Layout B* to address these two problems.

In this layout, the input layer is a rectangle of size 4×10 that is visualized as a grid of 2×5 receptive fields of size 2×2 tiled without overlap as shown on Figure 5-2. This yields a hidden layer of size 2×5, which is also the receptive field for the root node.

We choose to use random variable with domain size of 4 for the nodes in the hidden layer. This balances the number of combinations for the receptive field in both the hidden and the root layer in addition to decreasing it well below the corresponding

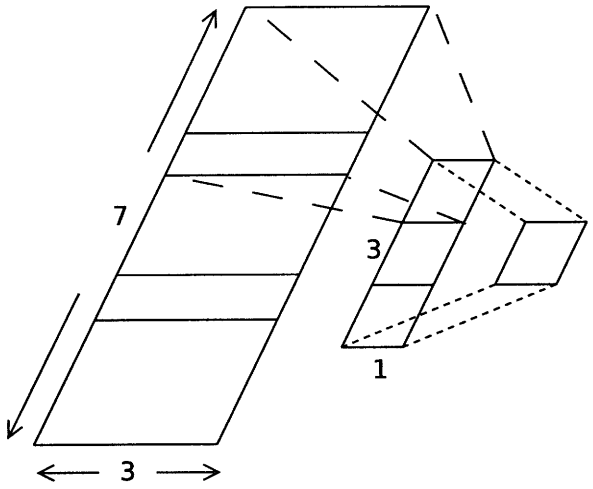

Figure 5-1: *Schematic of network* Layout A . Note the overlap of the receptive fields in the input layer.

figures for *Layout A* .

The aspect ration of the input layer is 2.5.

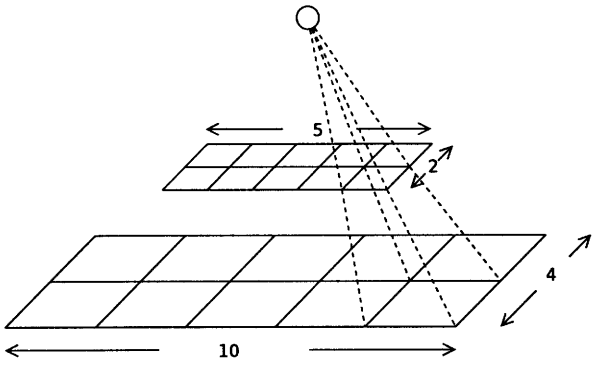

Figure 5-2: *Schematic of network* Layout B .

5.2 Software Systems

5.2.1 Bayesian Network Toolbox for MATLAB

Murphy's Bayesian network toolbox[23] (BNT) is a plug-in module written in MATLAB that implements a variety of graphical model techniques and algorithms including learning and inference of Bayesian networks. This is the library Dean uses in his PBN code. In order to increase the speed of the computations he augments it slightly with the idea of *hierarchical expectation refinement* (or subnetworks) that we discussed earlier.

5.2.2 OpenPNL Library by Intel

The OpenPNL library is a C++ analog of Murphy's toolbox with a very similar set of functionality. Unlike MATLAB, which is a script language, C++ is compiled and compiler-optimized. In the most cases this means

OpenPNL was released under an open source license by Intel and has since been supported by the open source community. Although its active development has ceased, the library is in mature state and supports the functionality that we need for this project.

5.2.3 Comparison

Our experiments showed a significant performance improvement using the OpenPNL library. Table 5.2 summarizes the results on some experiments used to help us with our choice. Based on this data, we decided to use the OpenPNL library for all our tests. We also decided not to run the data-parallel version of the library, but to provide parallelism through multiple problem instances running concurrently.

Description	Implementation	Resource Usage		Accuracy
		Running time	Memory usage	
Test 1	OpenPNL	18 MB	2 min	64.8%
Test 1	BNT	1750 MB	> 250 min	66.4%
Test 2	OpenPNL	20 MB	2 min	67.8%
Test 2	BNT	1560 MB	90 min	69.9%

Table 5.2: *Benchmarking the OpenPNL and Murphy's BNT toolbox implementations.* We have taken various samples from our experiments to benchmark the two systems both in terms of accuracy, but also of running time. Evaluations are performed on a single processor on similar settings.

5.3 Experiments Performed

5.3.1 Evaluation on Index $S2$ Maps

We first start by applying our results from the previous chapter to the PBN. Recall that we chose GentleBoost as the codebook of choice.

In Table 5.3 we show comparison results for different regime of parameters. There does not seem to be clear-cut setting that achieves optimal results, therefore, for subsequent experiments we use only codebooks of size 16 and networks with no lateral connections.

Figure 5-3 shows a comparison to the standard model as a function of the number of training examples. We see that the performance of our model increases, but we cannot see more of the trend because the dataset contains only 14000 examples. In Appendix C we describe a workaround for this limitation. The results on a larger dataset are shown on Figure 5-6 and described in the last part of this section.

5.3.2 Separate Codebook at Each Location

On Figure 5-4 we show the relative frequency of appearance of each prototype index in the dataset for two particular location: a corner of the image, and the center. As expected there is not much activity in the corner and the histogram is dominated by a

(a) *Layout A*

Lateral	16 Prototypes	50 Prototypes
Yes	71.06%	71.61%
No	71.43%	71.06%

(b) *Layout B*

Lateral	16 Prototypes	50 Prototypes
Yes	71.61%	71.72%
No	71.43%	71.25%

Table 5.3: *Summary of results on index $S2$ maps under different regimes of parameters.* Experiments conducted on 2000 positive and 6000 negative examples. Results show the accuracy at the EER.

single patch. In the center, however, we can see at least ten prototypes that play equal role and none of them is the one that peaked in the corner. This suggests that the total number of prototypes may not be enough to capture the variety at all locations especially when the predominant prototypes for each location are disjoint. To test this hypothesis we propose a new paradigm in which a separate codebook is trained for each location. To justify the validity of this setup, note that prototype indexes at different location are independent. They do not even have to belong to the same codebook.

Our experiments, however, do not show any significant improvement over a unified codebook. (see Figure 5-5)

5.3.3 Evaluation on Index $C1$ Maps

Augmenting the Domain of Random Variables

So far in our experiments we have considered filtering methods that for each location assign the index of the prototype with the strongest response. This scheme, however, has a drawback. It assumes there exists a prototype that can describe the input data at each location. This assumption does not necessarily hold true. One argument for this is that prototypes are $8 \times 8 \times 4 = 256$ dimensional vectors, but they are very few in

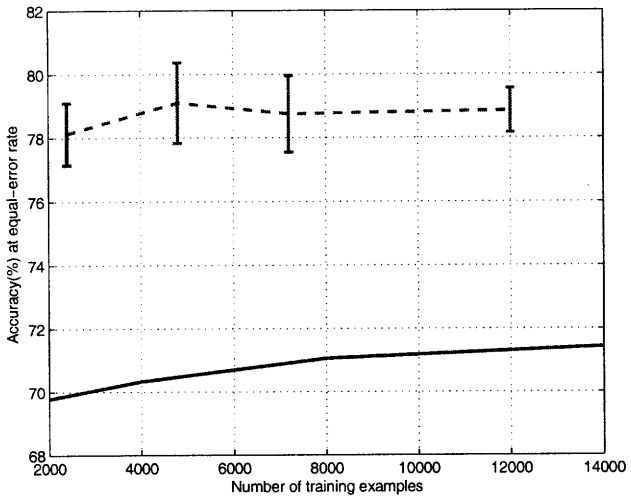

Figure 5-3: *Performance of index S2 maps with the number of training examples.* Performance is evaluated on *Layout B* on a codebook of size 16 produced by the GentleBoost approach.

Network	Lateral	Orientations	Accuracy at EER
Layout A	No	$4+1$	70.88%
Layout A	No	$8+1$	76.74%
Layout A	No	$16+1$	77.66%
Layout B	No	$4+1$	72.53%
Layout B	No	$8+1$	75.27%
Layout B	No	$16+1$	73.81%

Table 5.4: *Summary of results on index C1 maps.* Note that the "+ 1" in the *Orientations* field indicates an augmented domain. Evaluated on the complete original dataset of 2000 + 12000 images.

number. The immense volume of the 256-dimensional space, even when restricted to a cube of small side, cannot be captured by so few prototypes.

The solution we propose is to augment the domain size of each random variable in the input with a value which signifies that the strongest response is too weak for the prototype to be considered close to the input at that location. In other words, none of the prototypes is close enough.

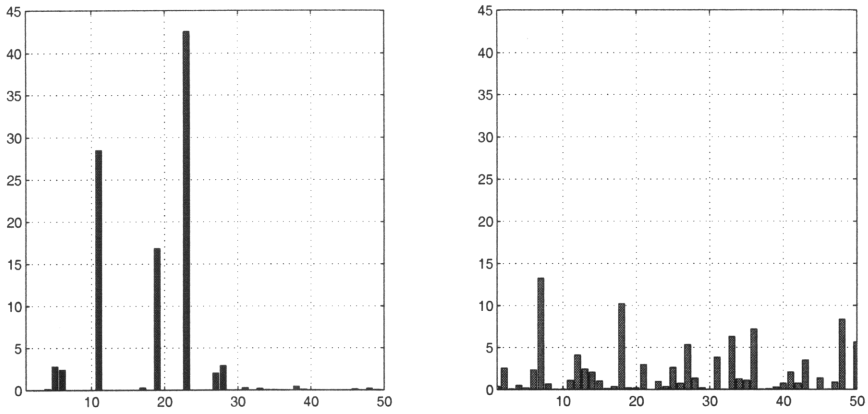

Figure 5-4: *Histograms of prototype frequency at two locations.* Shown is the relative occurrence(%) of each prototype at a location in the bottom-left corner (left) and in the center (right). The codebook is of size 50.

This method is applicable for both the index $C1$ and $S2$ maps. For $C1$ maps a threshold value is picked in advance and an extra "orientation" layer is created with values equal to the threshold. Therefore, at each location the $arg\max$ operator will automatically select the dummy orientation when the responses at the other orientations are below the threshold.

For $S2$ maps it is not feasible to set a threshold because the prototypes are independent of each other and may require different threshold values. Instead, we implement a scheme that automatically sets the threshold at twice the standard deviation of the peak responses for the prototype. In this implementation, all locations with patches too far from their closest prototype, as determined by the standard deviation criterion, are assigned the special value that augments the domain of the random variable. Effectively, this marks the patch under the location as not being close to any prototype. Note that in this particular paragraph we consider the response to be the L_2 distance and not the RBF value. Recall the inverse relationship between the two.

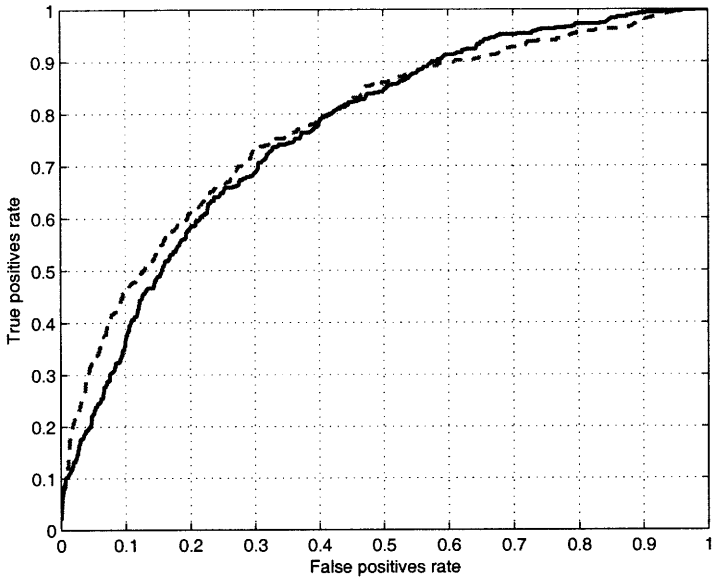

Figure 5-5: *ROC curves comparing single/unified codebook performance and separate codebook per location.* Comparison is done on a GentleBoost generated codebook with 16 prototypes and a *Layout A* network.

5.3.4 Improving the Index $S2$ Maps

The final set of experiments accounts for all our observations we have made so far. The model includes the following:

- A codebooks of size 16, generated by GentleBoost.
- No lateral connections.
- An augmented domain.
- Up to 48,000 training samples (24,000 positive and 24,000 negative). We generated additional *virtual examples* to increase the size of the dataset. For more information about virtual examples refer to Appendix C.

Network	Number of samples	Accuracy at EER	
		Non-augmented	Augmented
Layout A	2000	67.56%	68.86%
Layout A	4000	68.86%	70.88%
Layout A	8000	71.25%	71.06%

Table 5.5: *Comparison between augmented and non-augmented domains on index S2 maps.*

- Both positive and negative samples were split in 5 equal parts. Five instances of the model were ran, each trained on 4 parts of the data and tested on the fifth.

Figure 5-6 shows the performance of this method when compared to the standard model. In this case both the integrated model and the standard model used the same codebooks generated by GentleBoost. The standard model uses only one scale band (band 1), 4 orientations, and a patch size of 8×8.

5.4 Discussion

It is unfortunate that the integrated model underperforms the standard model on the tests we conducted.

An important advantage of the standard model is that its performance builds up as more scales, more orientations, and more patch sizes are added. But even when reduced to its bare minimum, the standard model is still better than a system with complex and sophisticated codebook evaluation and multilayered mesh of variables, which the PBN system is. The difference between the two is that in the standard model $C2$ maps measure the amount of relevant visual information present in the image, whereas Bayesian networks try to learn relationships between the visual features.

On the other hand, we admit that due to the very large parameter space in the configuration of the model, we may not necessarily be operating the PBN model at the regime of parameters that extracts the most value out of it. In particular, we left the

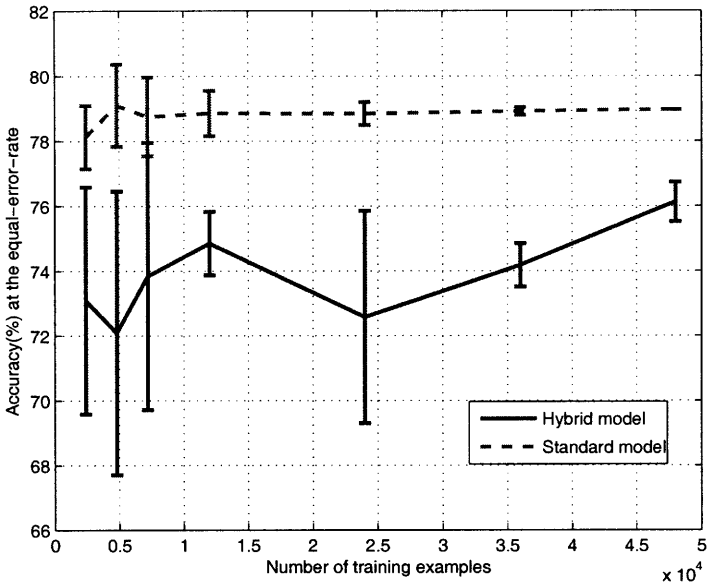

Figure 5-6: *Comparison between the standard model and the Bayesian network using index S2 maps on GentleBoost selected codebooks.* Evaluation is done on *Layout B* with random splits of training and testing data in 4 : 1 ratio. The two models use the same codebook of size 16. The error bars represent the standard deviation across the 5 splits.

space of possible network layouts largely unexplored. Yet, a comprehensive research of the network structure is beyond the scope of this Master's thesis.

Finally, Figure 5-6 shows the promise of the model to be in the number of training samples. As we increase the size of the dataset, the performance of the model improves consistently, while that of the standard model flattens out. We believe that with even larger datasets, with hundreds of thousands of examples, the model will show its true performance. Our argument is that the number of possible combinations in the receptive field require substantial amount of training data that we do not have available as of this time.

Chapter 6

Conclusion and Future Work

6.1 Future Work

In conclusion, we suggest two ideas that expand on this work.

6.1.1 Training on Larger Datasets

In our view, training the Bayesian network on very large datasets is the logical next step in this project. Evidence from performance measures suggests that the performance of the model continues to improve as more training examples are added to the system.

Ultimately, we expect to achieve better performance, for training sets with millions of examples. Unfortunately, we are not presently aware of a suitable dataset of real-world images with this many examples. One suggestion is the replacement of huge datasets with sets of short videos as a bulk source of still images. These images, however, need to have the variety required for the comprehensive training of the model.

6.1.2 Exploring the Network Layout

Another idea for future work is the exploration of the parameter space of network layouts. In this thesis we present experiments done on just two of the numerous possible layouts. We did not observe any significant difference in performance between the two layouts and so we do not wish to speculate as to how important the layout is for the performance.

In attempting various layouts it may be helpful to consider the biological point of view. It is believed that units at different levels behave in a similar manner and have receptive fields of similar size. This consideration alone will drastically reduce the size of the search space.

6.2 Conclusion

In this thesis work we explored one approach towards augmenting an existing biologically plausible system with feedback processing using Bayesian networks. We started with a simplistic model and we sequentially improved its components. First, we proposed three algorithms for generating visual codebooks for the model. Two of the algorithms improved upon the corresponding version in the initial model. Next, we proposed several suggestions for improvements on the Bayesian network component. In the end, we compared the resulting system back to the original.

Our model underperforms the initial system by a small margin, but as we increased the training data we noticed a trend of improvement which surpasses that of the original model. Unfortunately, we hit a limitation on the amount of available data we have.

Our belief is that the model has potential for improvement and we conclude with the two suggestions for future work outlined above.

Appendix A

Problems with GMM

Empirical evidence from the PBN model in Chapter 3 suggests numerical instability problems in the Gaussian mixture model for high-dimensional spaces. In particular, using $C1$ maps with patch size of 8×8, yields 256-dimensional patches. From our evidence, we know that 16-dimensional vectors do not trigger the problem, while 64-dimensional do. The next paragraphs try to explain why this is the case.

By definition, the GMM tries to learn n d-variate Gaussian functions of the form:

$$G_i = \frac{1}{(2\pi)^{d/2}\sqrt{|\Sigma_i|}} \exp -\frac{(x-\mu_i)^T {\Sigma_i}^{-1}(x-\mu_i)}{2}$$

where μ_i are the mean vectors and Σ_i are the covariance matrices, each of size $d \times d$. The error function for a given set of points, $X = \{x_i | i = 1, \ldots, m\}$, may vary from implementation to implementation, but we can assume a working definition that is as good as any other in practical terms:

$$\sum_{j \in 1 \ldots m} || \max_{i \in 1 \ldots n} G_i(x_j) ||^2$$

In other words, this is the sum of the squared value of the "closest" Gaussian for each

point. Note the similarity between this definition and the error measure for the k-means problem. In GMM, the goa is to select the parameters for G_i, the covariance matrix Σ_i and the mean vector μ_i, in such a way as to minimize the error. The size of the parameter space is then $O(nd^2)$.

Let us examine the scalability of the model in terms of its two main variables: the number of Gaussians, n, and their dimension, d. The number of Gaussians is essentially the size of the codebooks since each Gaussian represents one visual prototype. In the GMM approach, the n Gaussians have to be selected in such a way as to account for as many of the d-dimensional patches as possible. This is easy if there are n distinguishable clusters of points in the space. If, however, the pattern of points/patches in $\mathbb{R}^d$ is more complex and spread out, the Gaussians have to stretch to cover more points, thus becoming overly general. However, there is no cut-off range in the Gaussian exponential function which stretches to infinity, so spread-out Gaussians are influential on a lot of points. This trouble is avoided by increasing n. But the total number of parameters is linearly proportional to n and to learn all of them we need proportionally more training examples. From a technical standpoint, this indicates at least quadratic running time dependence on n.

On the other hand, increasing d poses a purely technical problem – a numerical under- or overflows in machine words. The learning process involves numerous evaluations of Gaussian functions, G_i, and those in turn involve the computation of the determinant of Σ_i. Σ_i is a d-dimensional vector and in the simplest case of a diagonal matrix, its determinant is the product of all d elements across the diagonal. (Diagonal elements are important even for non-diagonal covariance matrices as they represent the non-correlated component of the variance along each dimension.) However, storing a product of d numbers can easily lead to overflows or underflows. The reason being that all numbers represent variances and are of similar order of magnitude, which in turn is proportional to the dynamic range of the input images. Consider 8-bit images with pixel values in

the range $0 \ldots 255$. Taking the product of d such numbers may lead to an overflow. If, however, the dynamic range of the input is shifted to the interval $[0, 1]$ the product may lead to an underflow. This is especially true for large enough values of d and is precisely what was observed during some of the experiments with values of $d \geq 100$. Unfortunately, there is no simple workaround that guarantees absolute avoidance of the problems. To make it even worse, underflows or overflows are not detectable by a machine method during the process of learning, at least not without serious modifications to the code.

Appendix B

Codebook Prototypes and Their Relative Occurrence

These three figures each show the relative frequency of occurrence of prototypes from corresponding codebooks on positive and negative examples. The frequencies are taken from the index $S2$ maps built on the training set. The figures also show the actual $C1$ prototypes in all four orientations.

This is the expanded version of Figure 4-11.

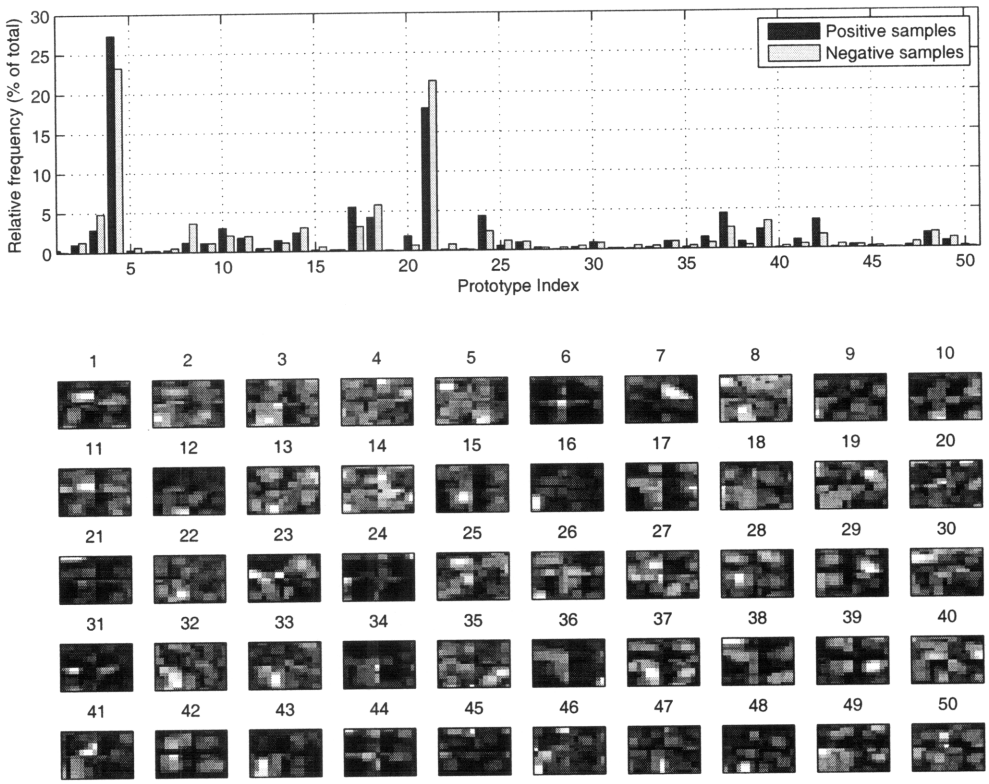

Figure B-1: *Frequency of occurrence of 50 randomly sampled prototypes.* Each pair of bars shows the relative frequency of occurrence of the corresponding prototype in positive(left, dark-colored) and negative(right, light-colored) examples.

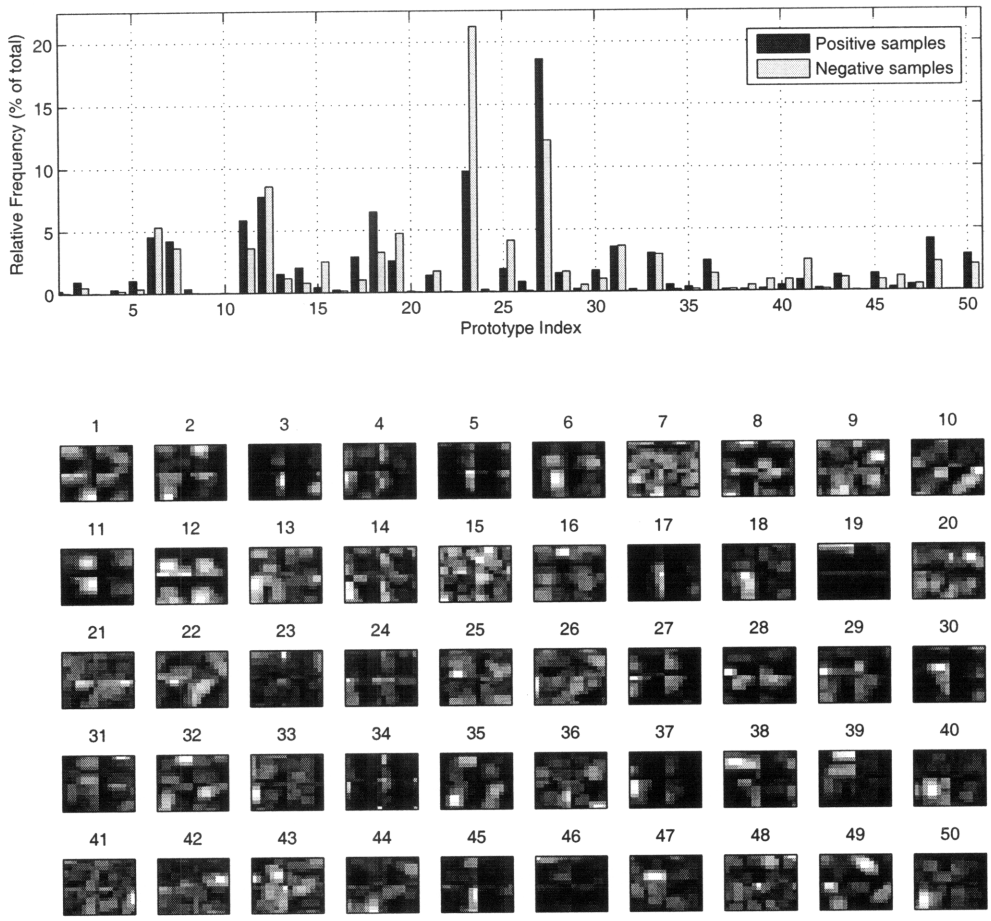

Figure B-2: *Frequency of occurrence of each GentleBoost prototype.* Each pair of bars shows the relative frequency of occurrence of the corresponding prototype in positive(left, dark-colored) and negative(right, light-colored) examples.

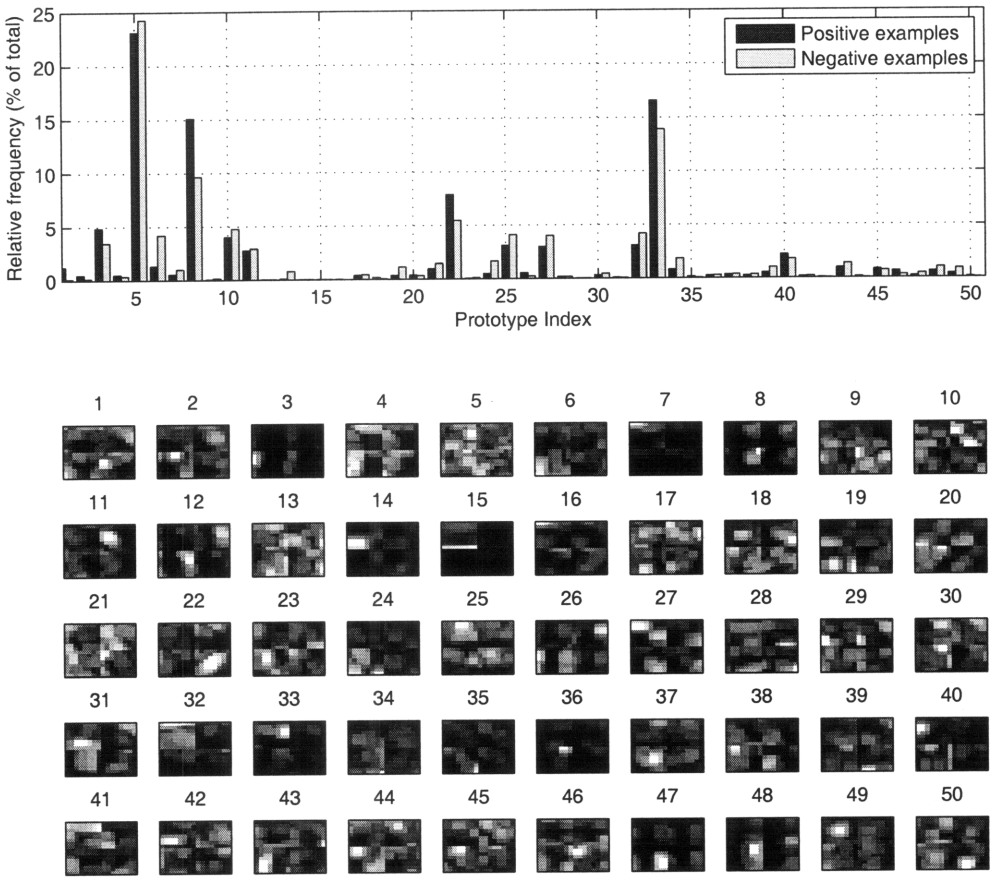

Figure B-3: *Frequency of occurrence of each JT prototype.* Each pair of bars shows the relative frequency of occurrence of the corresponding prototype in positive(left, dark-colored) and negative(right, light-colored) examples.

Appendix C

Virtual Examples

Virtual examples are training input samples that are not part of the original dataset, but are subsequently generated in order to reduce the variety of the training data or the total number of training examples.

Eleven virtual examples were generated for each positive training image, thereby increasing their number by a factor of 12, to 24000 images. A combination of the following procedures were used to generate the images:

- Flipping the image left-to-right.
- Rotating the image to ± 2 degrees around its center.
- Equalizing the intensity histogram of the image. This procedure effectively remaps the intensity values.

Given the relatively large pool of negative examples, only one virtual example was generated for each negative image by flipping the image left-to-right.

Figure C-1 shows a set of generated virtual examples on a single input image.

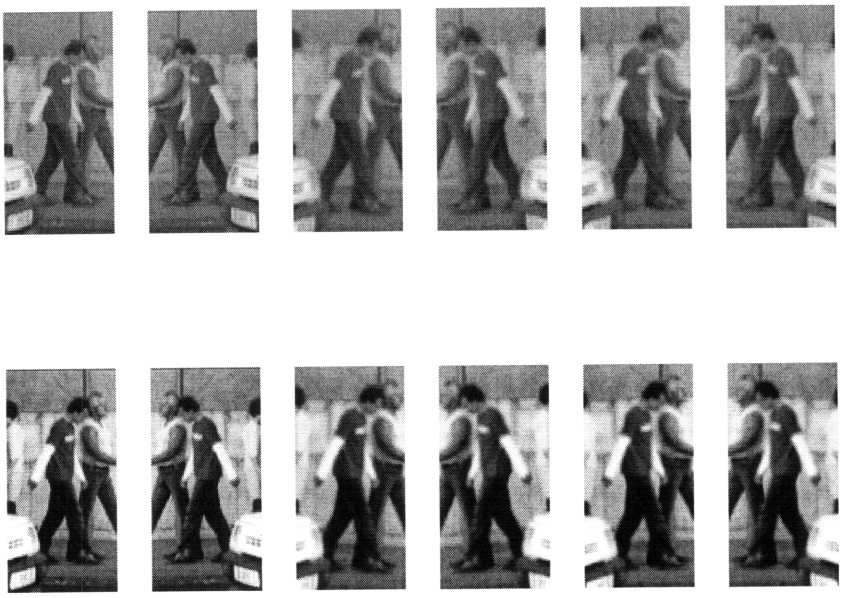

Figure C-1: *Example of virtual training samples generated from the pedestrian database..* The top left image is the original image. The bottom row contains the histogram adjuste images from the top row.

Bibliography

[1] Intel's open-source probabilistic networks library (pnl). http://www.intel.com/technology/computing/pnl/.

[2] Stanley Bileschi. *StreetScenes: Towards Scene Understanding in Still Images.* PhD thesis, Massachusetts Institute of Technology, 2006.

[3] Christopher Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

[4] Dorin Comaniciu and Peter Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(5):603–619, 2002.

[5] Sanjoy Dasgupta. Learning mixtures of gaussians. In *FOCS '99: Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, page 634, Washington, DC, USA, 1999. IEEE Computer Society.

[6] Thomas Dean. Hierarchical expectation refinement for learning generative perception models. Technical Report CS-05-13, Brown University Department of Computer Science, 2005.

[7] Thomas Dean. Scalable inference in hierarchical generative models. In *Ninth International Symposium on Artificial Intelligence and Mathematics*, 2006.

[8] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.

[9] Daniel Felleman and David Van Essen. Distributed Hierarchical Processing in the Primate Cerebral Cortex. *Cereb. Cortex*, 1(1):1–a–47, 1991.

[10] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *European Conference on Computational Learning Theory*, pages 23–37, 1995.

[11] J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting, 1998.

[12] Donald Geman and Stuart Geman. Stochastic relaxation, gibbs distribution, and the bayesian restoration of images. *IEEE Trans. Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 6:721–741, 1998.

[13] Bogdan Georgescu, Ilan Shimshoni, and Peter Meer. Mean shift based clustering in high dimensions: A texture classification example. *iccv*, 01:456, 2003.

[14] David Heckerman. A tutorial on learning with bayesian networks. Technical Report MSR-TR-95-06, Microsoft Research, March 1995.

[15] D. H. Hubel and T. N. Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *J Physiol.*, 160:106–154, 1962.

[16] Frederic Jurie and Bill Triggs. Creating efficient codebooks for visual recognition. In *ICCV '05: Proceedings of the Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*, pages 604–610, Washington, DC, USA, 2005. IEEE Computer Society.

[17] S. L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society. Series B (Methodological)*, 50(2):157–224, 1988.

[18] Yann LeCun and Corinna Cortes. The mnist database of handwritten digits. http://yann.lecun.com/exdb/mnist/.

[19] T. Lee and D. Mumford. Hierarchical bayesian inference in the visual cortex, 2002.

[20] T. Masquelier, T. Serre, S. Thorpe, and T. Poggio. Learning complex cell invariance from natural videos: A plausibility proof. Technical Report CBCL Paper 269/AI Technical Report 2007-060, Massachusetts Institute of Technology, December 2007.

[21] Mortimer Mishkin, Leslie Ungerleider, and Kathleen Macko. Object vision and spatial vision: two cortical pathways. *Trends in Neurosciences*, 6:414–417, 1983.

[22] David Mumford. On the computational architecture of the neocortex. *Biological Cybernetics*, 66(3):241–251, 1992.

[23] Kevin Murphy. The bayes net toolbox for MATLAB, 2001.

[24] Kevin Murphy, Yair Weiss, and Michael Jordan. Loopy belief propagation for approximate inference: An empirical study. In *Uncertainty in AI*, pages 467–475, 1999.

[25] Judea Pearl. Bayesian networks: A model of self-activated memory for evidential reasoning. In *Proceedings of the 7th Conference of the Cognitive Science Society, University of California, Irvine*, pages 329–334, August 1985.

[26] Maximilian Riesenhuber and Tomaso Poggio. Models of object recognition. *Nature Neuroscience*, 3:1199–1204, 2000.

[27] B.C. Russell, A. Torralba, K.P. Murphy, and W.T. Freeman. LabelMe: A Database and Web-Based Tool for Image Annotation. *International Journal of Computer Vision*, 77(1):157–173, 2008.

[28] Thomas Serre. *Learning a Dictionary of Shape-Components in Visual Cortex: Comparison with Neurons, Humans and Machines.* PhD thesis, Massachusetts Institute of Technology, 2006.

[29] Thomas Serre, Lior Wolf, Stanley Bileschi, Maximilian Riesenhuber, and Tomaso Poggio. Robust object recognition with cortex-like mechanisms. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 29(3):411–426, 2007.