Winter 2021

Biologically Inspired Computer Vision/ Applications of Computational Models of Primate Visual Systems in Computer Vision and Image Processing

Reza Hojjaty Saeedy
University of New Hampshire

Biologically Inspired Computer Vision

Applications of Computational Models of Primate Visual Systems in Computer Vision and Image Processing

By

Reza Hojjaty Saeedy

B.Sc. Mathematics, University of Tabriz, 2006

M.Sc. Mathematics, Urmia University, 2009

DISSERTATION

Submitted to the University of New Hampshire

in Partial Fulfillment of

the Requirements for the Degree of

Doctor of Philosophy

in

Integrated Applied Mathematics

December, 2021

This dissertation was examined and approved in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Applied Mathematics by:

Dissertation Director, Richard A. Messner, Associate Professor
Department of Electrical and Computer Engineering

Gregory Chini, Professor
Department of Mechanical Engineering

Mark Lyon, Associate Professor
Department of Mathematics and Statistics

John LaCourse, Professor
Department of Electrical and Computer Engineering

Wayne Smith II, Principal Lecturer
Department of Electrical and Computer Engineering

On December 10, 2021

Approval signatures are on file with the University of New Hampshire Graduate School.

Acknowledgements

I would like to thank my advisor Professor Richard A. Messner for his support, guidance and advice during the research and writing of this dissertation. He helped me to work on subjects that I liked most and his insights provided me with enough confidence to finish this thesis.

I also like to thank Professor Mark Lyon, as he is the main reason that I joined the IAM program in the Mathematics department. During my years in UNH, whenever I needed help or support he was ready to offer that.

I want to thank Professor Greg Chini, for smoothing the way for me both as a member of my committee and as the chair of the IAM program.

Also, my thanks goes to Professor. John LaCourse and Dr. Wayne Smith for being members of my committee, for their advice and insights.

And last but not least I must thank my wife Niloufar for his patience and kindness through all these years.

Contents

List of Figures

List of Tables

Biologically Inspired Computer Vision

Applications of Computational Models of Primate Visual Systems in Computer Vision and Image Processing

Reza Hojjaty Saeedy

Abstract

Biological vision systems are remarkable at extracting and analyzing the information that is essential for vital functional needs. They perform all these tasks with both high sensitivity and strong reliability. They can efficiently and quickly solve most of the difficult computational problems that are still challenging for artificial systems, such as scene segmentation, 3D/depth perception, motion recognition, etc. So it is no surprise that biological vision systems have been a source of inspiration for computer vision problems.

In this research, we aim to provide a computer vision task centric framework out of models primarily originating in biological vision studies. We try to address two specific tasks here: saliency detection and object classification. In both of these tasks we use features extracted from computational models of biological vision systems as a starting point for further processing.

Saliency maps are 2D topographic maps that catch the most conspicuous regions of a scene, i.e. the pixels in an image that stand out against their neighboring pixels. So these maps can be thought of as representations of the human attention process and thus have a lot of applications in computer vision. We propose a cascade that combines two well-known computational models for perception of color and orientation in order to simulate the responses of the primary areas of the primate visual cortex. We use these responses as inputs to a spiking neural network(SNN) and finally the output of this SNN will serve as the input to our post-processing algorithm for saliency detection.

Object classification/detection is the most studied task in computer vision and machine learning and it is interesting that while it looks trivial for humans it is a difficult problem for artificial systems. For this part of the thesis we also design a pipeline including feature extraction using biologically inspired systems, manifold learning for dimensionality reduction and self-organizing(vector quantization) neural network as a supervised method for prototype learning.

Chapter 1

Introduction

It is a central question that why should we be inspired by biology at all? or how biologically-motivated studies could be useful for constructing artificial systems?

Living systems are engineered to perfection by evolution, and thus they provide a seemingly inexhaustible source of inspiration for engineering tasks. All species were optimized by evolution over large time scales. This has led to efficient solutions to many of the challenges they face. And due to this excellence in terms of performance and robustness, it is therefore not surprising that humans always make efforts to use nature as a model for innovation and problem solving.

1.1 Statement of the problem

The central issue addressed in this thesis is efficient exploitation of specific biological functionality observed in primate visual systems and their respective popular computational models. We leverage such functionalities and models toward solving some problems in computer vision. The most important aspect of the work done here is to design a framework to extract salient regions of digital images. The process of finding salient regions or pixels in digital images is closely related to the attention process in Human Visual System(HVS). Attention refers to the process by which organisms select a subset of available information for further processing. This process is implemented by a network of brain areas that includes the superior colliculus, the pulvinar nucleus of thalamus, and the posterior parietal cortex. It is believed that evolution chose attention to address the issue of limited resources available to human sensory.

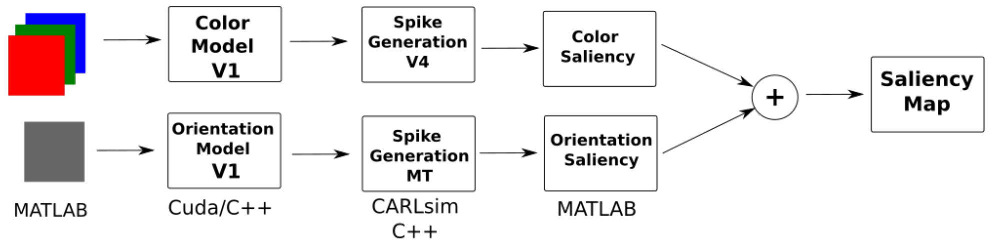


Figure 1.1: Our designed pipeline for saliency detection in digital still images.

The block diagram shown in Figure 1.1 depicts the strategy we used herein to extract salient regions in a digital still image. This pipeline consists of two separate pathways. One inspired by a simplistic view of ventral pathway $V1 \longrightarrow V4$ to extract color features and the other one imitates the dorsal pathway $V1 \longrightarrow MT$ that aims to extract orientated bars/edges features. The ventral pathway takes in three RGB channels of a color image and uses a known computational model for the $V1$ area implemented in CUDA/C++. In this stage of the model the main computational calculations are achieved using a DoG scheme [63] on color channels(see also 2.5 for details). The next step is to simulate the connectivities in the neural network of the visual cortex in general and model the functionality of neurons in the $V4$ area in particular. This stage is done in the CARLsim5 library(see section2.8 for details of implementation). After generating the spikes of area $V4$ the results are passed to our post-processing algorithm in MATLAB to extract the color saliency regions.

The dorsal stream is modeled in a similar manner except that here we start by intensity profile of an image as input and then use a Gabor-like filter bank in V1 area to extract oriented bars and edges. These features are later passed to CARLsim where we simulate area MT. Finally the spikes generated in this step are used to produce the orientation saliency.

It must be noted that the purpose of this research was not to propose a new computational algorithm for HVS but to leverage the existing models to design a framework that is useful for engineering applications. Also the goal for this framework is not to outperform every single model but to study and investigate the practical use of computational models of live systems in computer vision problems such as object classification and pattern recognition.

1.2 Importance of the problem

In the realm of computer vision a central question is *why should we be inspired by biology at all? or how artificial vision systems can benefit from biologically-motivated studies?*

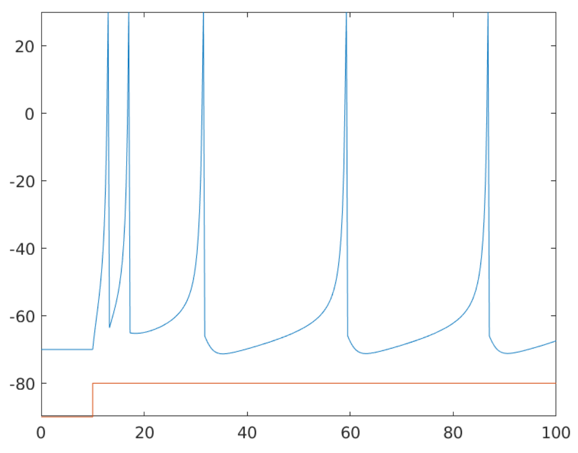
The answer relies partly on the evolution of living systems. Through the long time of evolution living systems engineered to nearly perfection and hence it is not surprising that they provide a seemingly inexhaustible source of inspiration. Over these large time scales, all species were optimized and this has led to efficient solutions to many of the challenges they face. Due to this excellence in terms of performance and robustness, it is therefore not surprising that humans always make efforts to use nature as a model for innovation and problem solving.

Chapter 2

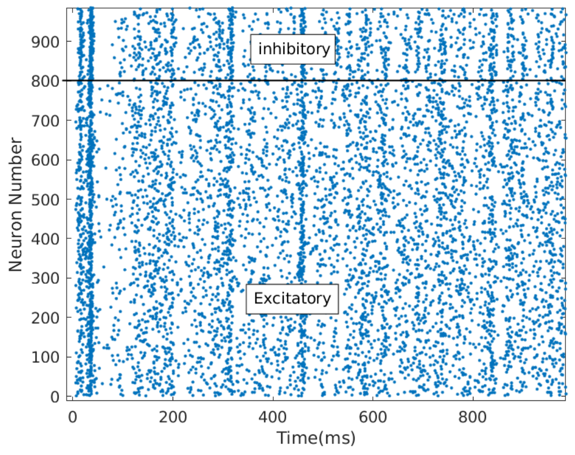
Human visual system and its computational models

2.1 Spiking Neural Networks

The main information processing workload of the brain is carried by nerve cells, also called neurons. Estimates of the number of neurons in the brain typically vary between 10^{10} and 10^{11}. Each neuron has one very long formation called an axon which connects it to other cells. Axons have sophisticated biochemical machinery to transmit signals over such relatively long distances. The machinery is based on a phenomenon called action potential. An action potential is a very short(1ms) electrical impulse traveling via the axon of the neuron. Due to their typical shapes (Figure 2.1), action potentials are also called spikes. Broadcasting the action potentials or spike trains to downstream neurons is the main means of communication between neurons. These individual spikes are sparse in time and so each spike has high information content. Spikes constitute the signals by which the brain receives, analyzes, and conveys information. Action potentials are all-or-none, in the sense that they always have the same strength (about 100mv) and shape. So the meaning of a spike is determined by when and where it happens. The activity of a cell is characterized by a single



(a)



(b)

Figure 2.1: (a):The typical shape of a spike. This specific spike is generated according to Izhikevich model. (b): 1s Simulation of 1000 randomly coupled Izhikevich neuron. The top 200 neurons are inhibitory and 800 lowers are excitatory. Both figures reproduced from [40]

scalar called firing rate, i.e. the number of spikes fired (emitted) by a neuron per second. The neurons and their connectivity, as well as their interactions play a central role in this thesis but since this is not a thesis in computational neuroscience we do not investigate the details of neuronal activity in the brain or even their computational models. In what follows we only focus on parts of the neural system that we employed to design our framework for computer vision application.

A spiking neural network is a two-layered feed-forward network with lateral connections in the second hidden layer that is heterogeneous in nature [60]. Therefore spiking neural networks (SNNs) are a class of artificial neural networks (ANNs) that mimic biological neural networks. In addition to neuronal and synaptic status, SNNs incorporate time into their working model and hence are more potent than non-spiking counterparts because they can encode temporal information in their signals. The idea is that unlike traditional neural networks (e.g. multilayer perceptron), neurons in the SNN do not transmit information at the end of each propagation cycle, but only when a membrane potential reaches a certain value, known as the threshold. Knowledge representation in time and space makes SNNs unique to perform brain-like computations and to understand the brain activity in spatio-temporal pattern [43]. Thus, the neuron fires when the membrane potential hits the threshold, sending a signal to neighboring neurons, which increase or decrease their potentials in response to the signal, depending on them being excitatory or inhibitory. Action potentials, spikes, and pulses are all terms used to describe these signals. This results in the key difference between ANNs and SNNs: SNN works with discrete events that happen at defined times instead of working with continually changing time values as ANN. As a result, SNN takes a set of spikes as input and produces a set of spikes as output (a series of spikes is usually referred to as spike trains). These processes can be summarized as follows:

- Each neuron corresponds to a value that is equivalent to the electrical potential of biological neurons at any given time.

- The value of a neuron can change according to its mathematical model; for example, if a neuron gets a spike from an upstream neuron at a specific time, its value may rise (excitatory neuron) or fall (inhibitory neuron).

- If a neuron's value surpasses a certain threshold, the neuron will send a single impulse (which is similar for all neurons) to each downstream neuron connected to the first one, and the neuron's value will immediately drop below its average.

- As a result, the neuron will go through a refractory period similar to that of a biological neuron. The neuron's value will gradually return to its average over time.

Spiking neurons and their linking synapses are described by configurable scalar weights in an SNN architecture called synaptic weights. The analogue input data is encoded into the spike trains using either a rate-based technique, some sort of temporal coding or population coding as the initial stage in building an SNN. As explained earlier, a biological neuron in the brain (and a simulated spiking neuron) gets synaptic inputs from other neurons in the neural network. However, the network dynamics of artificial SNNs are much simplified as compared to actual biological networks.

Similar to any other ANN, spiking neural networks learn by altering scalar-valued synaptic weights but being spiking, SNNs are capable to replicate a form of bio-plausible learning rule that is not possible in non-spiking networks. Many variations of this learning rule have been uncovered by neuroscientists under the umbrella term spike-timing-dependent plasticity (STDP). The main characteristic of STDP is that the weight (synaptic efficacy) connecting a pre- and post-synaptic neuron is altered based on their relative spike times within tens of millisecond time intervals. The weight adjustment is based on information that is both local to the synapse and local in time [19].

There are some advantages and disadvantages for SNNs that are summarized below:

Advantages

- SNN is a dynamic system. So it excels in processes of dynamic nature like speech and dynamic picture identification.
- The sparseness of spikes in time has the advantage that these spikes consume lower energy.
- To train an SNN, you simply need to train the output neurons.
- Because the neurons send discrete impulses rather than a continuous value, SNNs can work incredibly quickly.
- SNNs can boost information processing and noise immunity due to their leverage of temporal presentation of information.

Disadvantages

- SNNs are difficult to train because the transfer function of spiking neurons is usually non-differentiable which prevents back-propagation.
- As of now, there is no learning algorithm built specifically for this task.
- It is not practical to design a small SNN.

In recent years SNNs gained popularity in different fields of science and engineering and multiple variants of them are designed (see [97] for a review).

2.1.1 Neuron models

Neurons make contacts with each other through synapses. The presynaptic neuron sends its output to the synapse via an axon and post synaptic neuron receives it via its dendritic tree (see Figure 2.2).

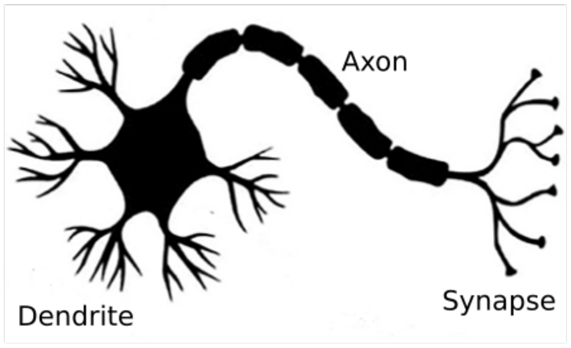


Figure 2.2: Schematic of a neuron with its major parts(image adapted from [94])

Most neurons produce their output in the form of virtually binary events(action potentials or spikes). The most detailed neuron model is Hodgkin-Huxley (H-H) neuron that covers neurophysiological details but one disadvantage of the H-H model is its high computational complexity. It requires about 1200 floating point operations (FLOPS) for the simulation of 1ms of time. So for simulating a large network with many neurons we need to use simpler models such as the Integrate-and-Fire (IF) model or the Izhikevich model. The Izhikevich [40] model offers the same rich spiking dynamics as the full H-H model, while having a computational complexity similar to the IF model. The importance of the Izhikevich model for this thesis belies in its implementation in CARLsim library where we do all our spiking neural simulation in. The Izhikevich neuron is a dynamical systems model that can be described by a two-dimensional system of ordinary differential equations:

$$\frac{dv}{dt} = 0.04v^2 + 5v + 140 - u + I \tag{2.1}$$
$$\frac{du}{dt} = a(bv - u) \tag{2.2}$$

Here, (2.1) describes the membrane potential v for a given current I, where I is the sum of all synaptic and external currents; that is, $I = I_{syn} + I_{ext}$. (2.2) describes a recovery variable u; the parameter a is the rate constant of the recovery variable, and the parameter, b, describes the sensitivity of the recovery variable to the subthreshold fluctuations of the membrane potential. All parameters are dimensionless; however, the right-hand side of (1) is in a form such that the membrane potential v has mV scale and the time t has ms scale. The parameters a, b, c, d are open parameters that have different values for different neuron types. The inclusion of u in the model allows for the simulation of typical spike patterns observed in biological neurons. The action potential downstroke is modeled using an instantaneous reset of the membrane potential whenever v reaches the spike cutoff, plus a stepping of the recovery variable:

$$v(v > 30) = c \tag{2.3}$$
$$u(v > 30) = u + d \tag{2.4}$$

The inclusion of u in the model allows for the simulation of typical spike patterns observed in biological neurons. In our work, all four parameters are chosen according to default parameters in [40]. thus, $a = 0.02$, that describes the time scale of the recovery variable u. $b = 0.2$ that describes the sensitivity of the recovery variable u to sub-threshold fluctuations of the membrane potential v. $c = -65mV$ that describes the after-spike reset value of the membrane potential v. and $d = 8$ or 2 is the after-spike reset value of the recovery variable u for excitatory and inhibitory neurons respectively.

2.1.2 Poisson model for spike generation

Successive action potentials in the cortex show a highly irregular timing. The interpretation of this irregularity has led to different divergent views of cortical organization. Here, we follow [35] assuming that the irregularity might arise from stochastic forces and thus the irregular interspike interval reflects a random process and so the precise timing of individual spikes conveys little information. An immediate result of this assumption is that an instantaneous estimate of the spike rate can be obtained by averaging the pooled responses of many individual neurons.

In doing so, suppose that the generation of each spike depends only on an underlying driving signal, $r(t)$, that we will refer to as the instantaneous firing rate. In this manner the generation of each spike is independent of other spikes and thus the spike train can be completely described by Poisson process. The Poisson distribution is a discrete probability distribution that expresses the probability of a given number of events occurring in a fixed interval of time or space provided that these events occur independently of previous events and with a known constant mean rate. Such events are called Poisson processes and are important in a variety of problems involving rare, random events in time or space. Later on, when we model our network in CARLsim we use the `PoissonRate` class that allows us to create spike trains whose inter-spike interval follows a Poisson process. To see the details of Poisson spike generation please refer to [35].

2.2 Visual cortex

2.2.1 Visual area V1

The primary visual area (V1) of the cerebral cortex is the first stage of cortical processing of visual information. Area V1 contains a complete map of the visual field covered by the eyes. It receives its main visual input from the lateral geniculate nucleus of the thalamus

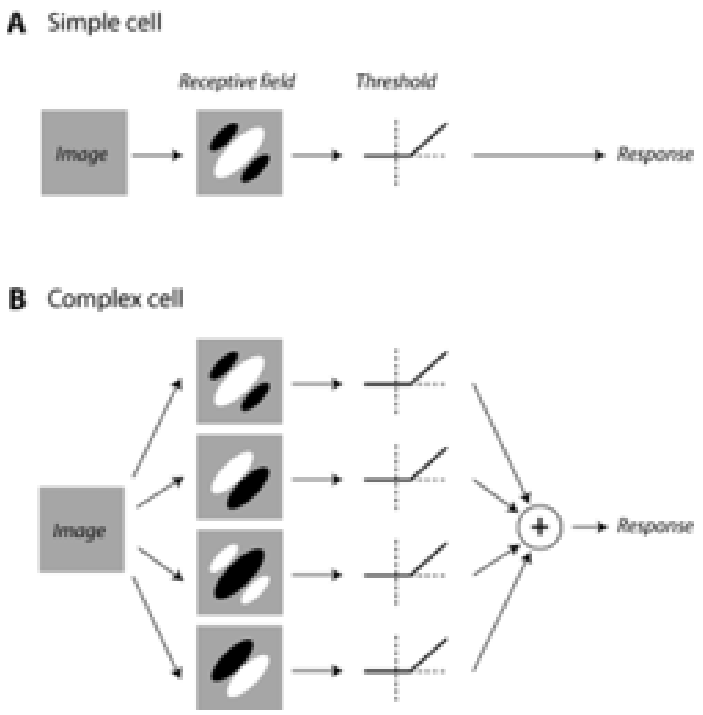


Figure 2.3: A: Simple cell's receptive field shape, B: Complex cell's receptive field shape. Also the functionality of these cells is shown. The image is taken from [20]

(LGN), and sends its main output to subsequent cortical visual areas. V1 is one of the best understood areas of the cerebral cortex, and constitutes a prime workbench for the study of cortical circuits and of computations. Numerous factors contribute to this fortunate position: we understand the nature of its main inputs, we know what stimuli make its neurons fire, and we can easily make those stimuli thanks to computer displays. The main task of V1 is to process visual inputs from the LGN and send the results of this processing to higher visual areas and subcortical structures. In primates, these include areas V2, V3, V4, MT, etc. Neurons in area V1 are classically divided into two types: simple and complex ([37, 38]), based on the structure of their receptive field. In simple cells, receptive fields have separate ON and OFF subregions (Figure 2.3 part A). ON and OFF subregions differ in their responses to the onset of stimuli on a gray background: ON subregions respond white bars, and OFF subregions respond to black bars. In complex cells, instead, ON and OFF regions are superimposed, i.e. every location in the receptive field responds both to white and black bars (Figure 2.3 part B). V1 cells are commonly classified as simple or complex based on their responses to drifting visual gratings. In simple cells the responses are periodic, whereas in complex cells they are steady in time. In addition to stimulus position, V1 neurons are selective for a number of attributes, including orientation, direction of motion, spatial and temporal frequency. In many species they are also selective for binocular depth and color. Out of all above attributes we are interested in their selectivity to orientation and color.

- **orientation**: A key characteristic of the responses of V1 neurons is their high selectivity for stimulus orientation. This selectivity was discovered by Hubel and Wiesel [37], and must arise from computations that take place within cortex, because LGN responses are not selective for orientation. Orientation selectivity is arguably the most studied example of cortical processing. The orientation selectivity of simple cells derives directly from the shape of their receptive field, (Figure 2.3, part A): ON and OFF subregions are elongated, so their preferred stimulus is similarly elongated. For complex cells, orientation selectivity cannot be directly predicted from the profile of the receptive field, (Figure 2.3, part B), but it is no less pronounced than for simple cells. This follows straightforwardly from the descriptive model of complex cells : all the simple cells providing input to the complex cell are selective for the same orientation, endowing the complex cell with the same orientation selectivity.

- **color**: In primates, the retina contains cones responsive to three bands of wavelengths (trichromacy). Retinal ganglion cells rearrange these responses along one of three “cardinal directions”, known informally as red-green, blue-yellow, and black-white.

We are mainly interested in objectives of two information processing streams originating in the occipital cortex, dorsal (which goes to parietal cortex) and ventral (which goes to temporal cortex), which exhibit relative specialization in object recognition (what) and spatial vision (where). This configuration sometimes referred as what-and-where proposal. To see a more detailed explanation please see [100]

2.2.2 Functional role of V1

The basic operation that V1 is thought to perform on images is simple filtering to enhance edges and contours. Simple cells are thought to perform linear filtering, i.e. weighted sums of the intensity values in an image, with weights given by the receptive field profile [68]. Complex cells in turn are thought to sum the rectified output of simple cell-like filters, thus computing the overall energy of an image in a band of frequency and orientation [67].

The result of these image processing operations can be summarized by taking an example image and producing a "neural image" that summarizes the output of an array of visual neurons (Figure 15). The original image is filtered by center-surround receptive fields in retina and LGN to obtain a neural image that enhances contours. This enhancement is much more pronounced in the output of V1 cells, which enhance oriented contours.

2.2.3 Simple cells and complex cells in V1

V1 neurons have been classified into two primary categories: simple cells and complex cells. Simple cells can respond in a way that can be noted from arrangements of excitatory and inhibitory regions in their receptive fields (RF). Essentially, this means that the RFs are *simple* because there appears to be a relationship between the response of the cell and the RF mapped with small spots. There is a long tradition in which simple cell responses have been characterized using linear RFs. In such a model, the neuronal response is a weighted sum of the local stimulus contrast. Linear models are commonly used to explain simple cell selectivity for stimulus orientation and spatial frequency. Typical linear receptive fields can have negative response, since they combine inputs using both positive and negative weights, but extracellular response measurements(firing rates) are, by definition, positive. This deficiency is usually addressed by imposing a form of rectification on the linear output.

The Simoncelli and Heeger model uses a half-squaring operation (halfwave-rectification followed by squaring) Figure 2.4. For these purposes. Half-squaring does not drastically alter the tuning properties of the model neuron, which are primarily determined by the underlying

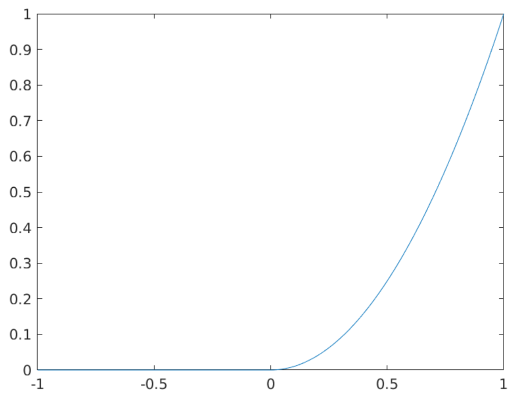


Figure 2.4: Typical shape of a half-square rectification function that we used in this thesis.

linear receptive field. This particular form of rectification is chosen for mathematical convenience but other nonlinearities can also be used. Rectified linear receptive field models do not account for several important simple cell nonlinearities, such as response saturation and cross-orientation inhibition. Many of these behaviors can be accounted for by incorporating response normalization.

Complex cells are similar to simple cells, in that they are selective for spatio-temporal orientation. However, their responses are relatively independent of the precise stimulus position within the receptive field. Complex cells and their RFs, have a more complex response that unlike simple cells does not exhibit the simple relationship between RF and its response. It is widely believed that complex cells combine the responses of a set of underlying linear receptive field sub-units.

2.3 Visual area V4

Neurons in V4 continue to integrate lower-level responses into higher-level responses and therefore increase invariance. V4 neurons respond selectively to orientation, color and simple shapes. Color coding cells in V4 differ from those in V1 in that they code for hue, rather than color opponency along the two principal color axes. An important characteristic of V4 is its role in color constancy which refers to the case of perceiving the constant color for an object under different illumination condition. In our research, we model neurons in area V4 as spiking neurons in CARLsim and consider six types of color sensitive cells, each responding to six colors, red, green, blue, yellow as primary colors and cyan and magenta as secondary colors. For more information about computational models of V4 please refer to [36, 4, 18, 84]

2.4 Visual area MT

Many neurons in area MT are tuned for retinal image velocity [13]. They respond vigorously to a visual stimulus moving with a particular speed and direction, and somewhat indifferent of the stimulus spatial pattern [85]. The computational models for area MT are used extensively for motion and velocity detection [104]. An empirical link has been established between neural activity in are MT and the perception of motion [64]. The computational model that we use in our research consists of two primary stages corresponding to cortical areas V1 and MT [99]. The basic form of computation is identical in each of these two stages: a weighted sum of input values followed by rectification, squaring and response normalization. Although, the known functionality and role of MT neurons is toward detection of motion, we modify the existing models to leverage them for orientation detection in still images.

2.5 A Computational model for early vision

One of the first and most influential works done in computationally modeling the early steps of human vision is reflected in the seminal works of David Marr et al. [62, 63] This model suggests a simple yet effective way to detect boundaries in a digital image. Detecting the boundary of objects in a natural scene is known to be an essential step in object detection and scene recognition [91]. This task is believed to take place in first areas of the visual cortex and studies have shown that people with lesions in this area lost their ability in object detection [73]. Different variants or modifications of their model were developed and used in later years but most of them share the main idea behind their method. We also use their model and some of its close alternatives later in this thesis when we try to use computational models for color and orientation perception in the visual cortex. Thus, here we briefly explain their model. For further details and biophysical and philosophical implications of their model please refer to [62].

It is clear that boundaries of objects are located at intensity or color changes in a digital image. So in order to find boundaries of objects or regions in a digital image one needs to to find the locations of abrupt changes in intensity. Intensity changes occur at different scales in an image and so their optimal detection requires the use of operators of different sizes; and that a sudden intensity change will give rise to a peak or trough in the second derivative. These idea suggest that in order to detect intensity changes efficiently, one should search for a filter that has two salient characteristics. Such an operator should be a differential operator that takes either first or second spatial derivative of the image. Second, it should be capable of being tuned to act at any desired scale, so that large filters can be used to detect blurry shadow edges and small ones to detect sharply focused fine detail in the image.

Marr and Hildreth suggestion was the filter $\nabla^2 G$ where $\nabla^2 = (\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2})$ is the Laplacian operator and G stands for the two-dimensional Gaussian distribution:

$$G(x, y) = e^{-\frac{x^2+y^2}{2\pi\sigma^2}} \tag{2.5}$$

with standard deviation σ. The idea behind the choice of the filter $\nabla^2 G$ is that, first, the Gaussian part G, blurs the image, thus wiping out all structures at scales much smaller than the space constant σ. Second, the ∇^2 operation, is computationally economic and its zero-crossing corresponds to intensity change in image. Hence, in practice, the most satisfactory way of finding intensity changes at a given scale in an image is first to filter with the operator $\nabla^2 G$, and then to locate the zero-crossing in the filtered image. Or mathematically, given image intensity function $I(x, y)$ for each pixel location (x, y) we need to first convolve I with

Gaussian G and then apply operator ∇^2 like $\nabla^2(G \star I) = (\nabla^2 G) \star I$.

One can even make a more economic approximation of $\nabla^2 G$ by difference of two Gaussian(DoG). $\nabla^2 G$ can be approximated closely by a DoG, and according to [63], the best approximation from an engineering viewpoint being achieved when the two Gaussian that form the DoG have space constant ratio 1:1.6.

2.6 Linear models of visual neurons

Much of neuroscience has been concerned with measuring the firing rates of cells as a function of some properties of visual input. In the early visual system, the response of a typical neuron depends only on the intensity pattern of a very small part of the visual field. This area, where light increments or decrements can elicit increased firing rates, is called the (classical) receptive field (RF) of neuron.

2.6.1 Simple cells and linear models

Cells in V1 have more interesting receptive fields than those in the retina or LGN. The so called simple cells typically have adjacent elongated (instead of concentric circular) regions of excitation and inhibition. This means that those cells respond maximally to oriented image structure. A linear model for a visual neuron means that the response of a neuron is modeled by a weighted sum of the image intensities, as in

$$r_j = \sum_{x,y} w_j(x,y) I(x,y) + r_0 \tag{2.6}$$

where $w_i(x,y)$ contains the pattern of excitation and inhibition for light for the neuron j in question. The constant r_0 is the spontaneous firing rate.

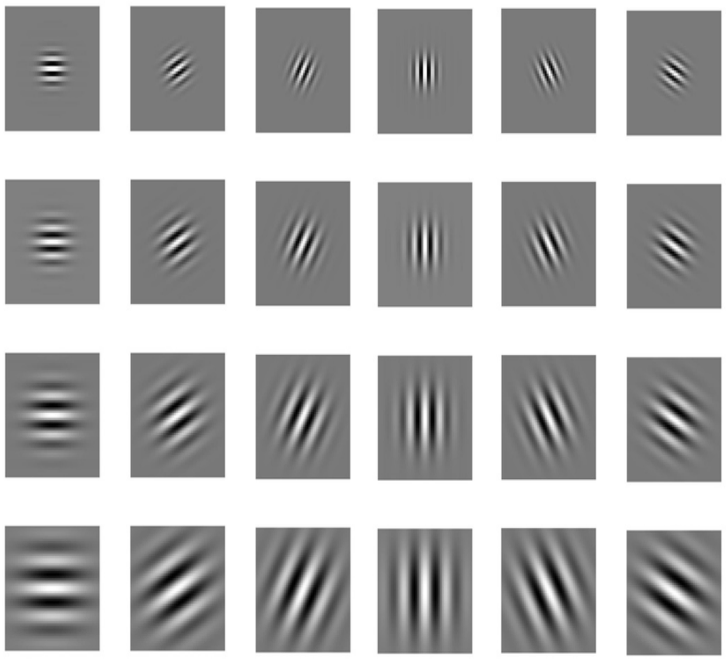


Figure 2.5: Filter banks of Gabor function. left to right: $\theta = 0, \pi/6, \pi/3, \cdots, 5\pi/6$ and top to bottom: different σ. In all rows $\phi = 0$ and $\gamma = 1$.

2.6.2 Gabor functions and receptive fields of simple cells

Typically we model the receptive fields of simple cells by Gabor functions. A Gabor function consists of an oscillatory sinusoidal function which generates the alternation between the excitatory and inhibitory areas and a gaussian envelope function which determines the spatial size of the receptive field,

$$w_{ij} = \exp(-\frac{(u^2 + \gamma^2 v^2)}{2\sigma^2}) \cdot \cos(\frac{2\pi}{\lambda} u + \phi) \tag{2.7}$$

for $u = i\cos(\theta) + j\sin(\theta)$ and $v = -i\sin(\theta) + j\cos(\theta)$. Here θ is the orientation of the filter, γ is the aspect ratio, σ is the effective width, ϕ is phase and λ represents the wavelength. Figure 2.5 represents a collection of filter banks of Gabor functions for different parameters. The Figure 2.6 shows how Gabor filter or its alternatives like Gaussian derivatives (see appendix A). An edge, bar or a moving particle that is oriented at the same direction as filter lobe, can activate the filter. On the other hand if it is perpendicular to the filter lobes

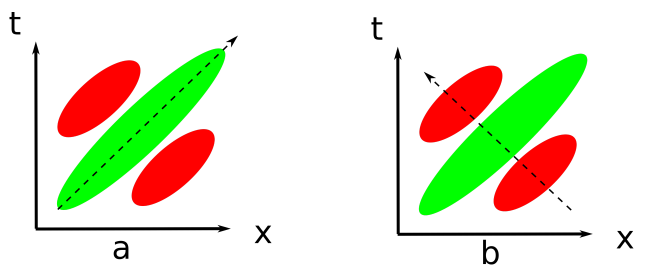


Figure 2.6: The functionality of gabor filter in edge extraction. The center green lob is excitatory and surround red lobes are inhibitory. An edge oriented at the same direction as center lobe will fire the maximum response(image reproduced from [6]).

filter's excitatory part (green lobe) will be nullified by filter's inhibitory part (red lobes) and thus the edge will not elicit a response.

2.6.3 Non-linearity in simple-cell responses

Real neurons exhibit different kinds of nonlinear behavior. The most basic non-linearities can be handled by adding a simple scalar non-linearity to the model, which leads to what is simply called a linear-nonlinear model. In these kind of models, a linear stage is followed by a static non-linearity f:

$$\tilde{r}_j = f\left(\sum_{x,y} w_j(x,y) I(x,y)\right) \tag{2.8}$$

One reason to employ non-linearity is to avoid negative firing rate. If a neuron has a relatively low spontaneous firing rate, the firing rate predicted by the linear model may then tend to be negative, which is impossible by definition. Thus non-linearities like half-wave rectification offer one way to work around this problem.

2.6.4 Computational model for complex cells

Simple cells are not the only cells present in V1. Complex cells are another group of cells that exist in V1. Similar to simple cells, complex cells respond selectively to bars and edges at a particular location and of a particular orientation, however, they do not show any clear spatial zones of excitation or inhibition.

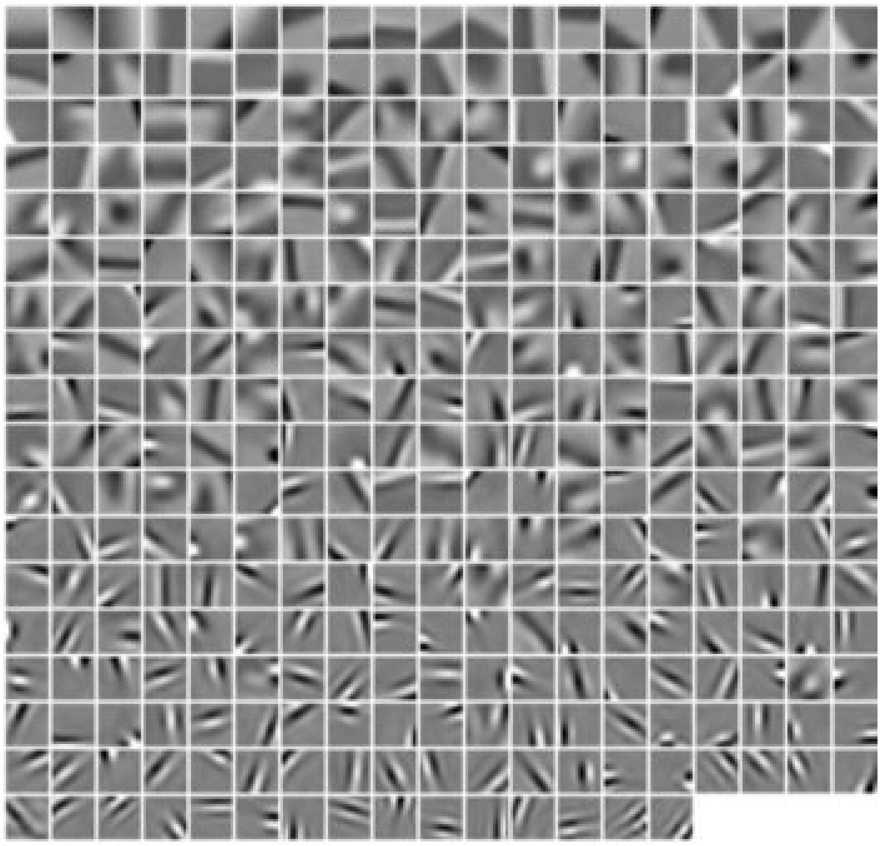


Figure 2.7: Sparse features learned from a dataset of natural images. Notice their similarity to patches of Gabor functions in Figure 2.5

2.7 Sparse coding in the brain

Mammalian brains consist of a large network of billions of neurons, each of them capable of fire independently. The pattern of activation of this large network is a means of information representation and is called neural code. In other words, the neural code defines what pattern of neural activity corresponds to each represented information item [8]. Neurons are binary in the sense that they are either active or inactive and once they are active, they show a fixed level of activity. For any set of N such binary neurons, the average (or expected value) of active neurons is called the density of the code and a code with low density is called a sparse code.

Brain employs sparse coding as a computational strategy to encode sensory information, where at a given time only a small number of simultaneously active neurons are used. There are compelling evidence for brains using sparse coding in sensory systems and there are convincing theoretical reasons for why brain should use sparse coding among which the following are more popular:

1. more memories can be stored
2. make use of statistical structure of natural scenes
3. Represent data in a convenient way for further processing
4. save metabolic energy, by decreasing neural firing rate.

2.7.1 Mathematical meaning of sparseness

Mathematically, sparseness is defined as the case that the random variable is very close to zero most of the times or alternatively, the random variable is active only rarely. But in order to speak of sparseness of a random variable we need to have a baseline distribution for comparison. The gaussian normal distribution is one that works here. By definition, a random variable is sparse if it is less active than a gaussian with the same variance and mean zero.

In vision science one way to learn features is by maximizing the sparseness. So for example, we consider a single feature s as weighted sum of pixel intensities:

$$s = \sum_{x,y} W(x,y)I(x,y) \tag{2.9}$$

and try to learn the weights that maximize the sparseness with respect to some sparseness measure. Figure 2.7 shows a few weights W_i obtained by finding a local maximum of sparseness, using the sparseness measure $h(s^2) = -\log\cosh(s)$. One can see the similarity between these learned weights and Gabor functions and simple cell receptive fields(see Figures 2.5 and 2.6).

2.8 Color perception

Light can vary in both wavelength and intensity. Color vision is the ability to make discrimination based on the wavelength composition of the light independent of its intensity. Color vision is used to determine the location and shapes of objects (e.g., fruit among foliage) and their identity and characteristics (e.g., what kind of fruit and whether it is ripe). It is particularly useful in cluttered natural scenes, where intensity variations may arise from either shadows or object borders.

The retina has two types of receptors, rods and cones. The three types of cone found in the human retina support color vision, which makes human trichromats [83]. The three are called S-cones, M-cones, and L-cones where the capital letters refer to their peak sensitivities in the short, medium, and long wavelength regions (respectively) of the visible spectrum. The output of each type of cone provides measurements of incoming light intensity over a broad range of wavelengths. It is however, important to bear in mind that the cones provide just the starting point for color vision, and that their outputs are not simple correlates of our perception of color.

2.8.1 Opponent color theory

Hering proposed his opponent color theory in 1872. Hering had noted several anomalies which seemed to be incompatible with a simple trichromacy theory of color. These anomalies were that it does not seem possible to experience a red-ish green or a blueish yellow color. Accordingly he proposed that colors were represented in two opponent pairs, red-green and blue-yellow. The cone classes do not correspond to our perception of red, green, and blue; rather, our perception of color requires multiple stages of L,M,S input integration [4]. An important early stage is the generation of color-opponency. Red-green neurons detect differences in L and M cone inputs, blue-yellow neurons compare S and L+M inputs.

About 5-10% of V1 cells are dedicated to color-coding and are called double-opponent

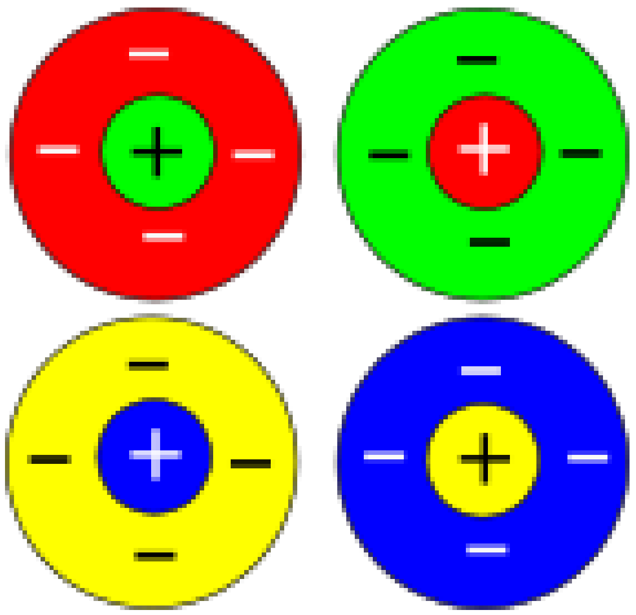


Figure 2.8: Color-opponent cells in visual cortex. The center marked with + means excitatory and the surround marked with - means inhibitory.

cells. double-opponent cells respond well to a spot of one color on a background of its opponent color and they form the basis of color contrast and color constancy. Four types of double-opponent cells are characterized in V1: Red+/Green-, Green+/Red-, Yellow+/Blue- and Blue+/Yellow- for excitatory(+) and inhibitory(-) effects. Mathematically these opponent cells are modeled using a DoG scheme (see Figure 2.8).

Chapter 3

A saliency detection method using spikes generated in visual cortex

In this chapter we design a framework to detect salient regions in still images. We build our model based on existing computational models for primate visual systems. Specifically, we are interested in two models as the backbone for feature extraction. The first one is a known algorithm for color feature detection which works based on center-surround scheme and second one is a popular algorithm for orientation detection which is based on well-known Gabor filters or its variants. This chapter is organized as follows: First we discuss the attention process in the human brain, its importance and also its relation to saliency maps, then we explain the details of color and orientation feature extraction in our model. Please note that none of these algorithms are novel and thus for mathematical details, the reader is referred to appendix A or the given references. Then we combine all these visual features in two post-processing algorithms to derive the final saliency map. Finally we evaluate our model over ground truth and some other well-known models. We used various evaluation metrics where all of them are briefly explained.

3.1 Attention and saliency

Visual attention is one of the most important features of the human visual system. visual attention serves as a mediating mechanism involving competition between different aspects of the visual scene and selecting the most relevant areas to the detriment of others. Nevertheless, our environment presents far more perceptual information than can be effectively processed. In order to keep the essential visual information, humans have developed a particular strategy. This strategy, confirmed during the last two decades, involves two mechanisms. The first refers to the sensory attention driven by environmental events, commonly called bottom-up or stimulus-driven. The second one is the volitional attention to both external and internal stimuli, commonly called top-down or goal-driven [56].

Primates' visual cortex is capable of interpreting complex dynamical scenes in clutter. This process is thought to involve attention shifting; i.e., selecting circumscribed regions of visual information to be preferentially processed and by changing the processing focus over the time course. There have been several approaches in the literature for dynamic attention along the ventral and dorsal pathways, including both scene-dependent (bottom-up) and/or task-dependent (top-down) strategies (see [11] for a review). Many computational models of human visual search have embraced the idea of a saliency map to accomplish pre-attentive selection. This representation contains the overall neural activity elicited by objects and non-objects, which compete for processing space, and includes salience for primary visual features such as intensity, orientations, colors, and motion. In some theoretical and experimental studies, saliency maps have been attributed to early visual areas. Itti and Koch introduced a bottom-up model [39], in calculating multiple visual features and linearly combining the feature maps to obtain a representation for describing the overall saliency of the objects on the static image [79].

Image saliency detection is an important and fundamental research problem in neuroscience and psychology to investigate the mechanism of human visual systems in selecting regions of interest from complex scenes. Recently it has also been an active topic in computer vision, due to its applications to object detection and image editing techniques [92].

Visual saliency is the perceptual quality that makes an object, person, or pixel stand out relative to its neighbors and thus capture our attention. Visual attention results both from fast, pre-attentive, bottom-up visual saliency of the retinal input, as well as from slower, top-down memory and volition based processing that is task-dependent [1].

The most important function of selective visual attention is to direct our gaze rapidly towards objects of interest in our visual environment. This ability to orientate rapidly towards salient objects in a cluttered visual scene has evolutionary significance because it allows the organism to detect quickly possible prey, mates or predators in the visual world. Attention addresses an information-processing bottleneck by allowing only a small part of the incoming sensory information to reach short-term memory and visual awareness. Instead of attempting to fully process the massive sensory input (estimated to be on the order of 10^7–10^8 bits per second at the optic nerve) in parallel, a serial strategy has evolved that achieves near real-time performance despite limited computational capacity. Attention allows us to break down the problem of understanding a visual scene into a rapid series of computationally less demanding, localized visual analysis problems.

The first explicit, neurally plausible computational architecture for controlling visual attention was proposed by Koch and Ullman in 1985 [44]. Koch and Ullman's model was centered around a *saliency map*, that is, an explicit two-dimensional topographical map that encodes stimulus conspicuity, or saliency, at every location in the visual scene. The saliency map receives inputs from early visual processing, and provides an efficient control strategy in which the focus of attention simply scans the saliency map in order of decreasing saliency. The first processing stage in any model of bottom-up attention is the computation of early visual features. Neurons at the earliest stages are tuned to simple visual attributes such as *intensity contrast*,*color opponency*, *orientation*, *direction* and *velocity of motion*, or *stereo disparity* at several spatial scales [30]. Neural tunning becomes increasingly more specialized with the progression from low-level to high-level visual areas, such that higher-level visual areas include neurons that respond only to corners or junctions, shape-from-shading cues or views of specific real-world objects [52].

In this chapter we explain our proposed algorithm for saliency detection in still images. Similar to most saliency detection methods we start with extracting features from input images. We are specifically interested in color and orientation features. The novelty of our work is that instead of directly calculating features, we use a spiking neural network simulator to simulate the neural connectivity in two different pathways in the striate cortex. Then we use action potentials (spikes) generated by neurons in this area as visual features. Our results show that while the responses of these neuron form sparse matrices, they carry enough information for our post-processing algorithm to extract salient pixels in the image and on the other hand their sparsity help us to make the post-processing step simple and fast.

3.2 Color feature extraction in our model

We use the model by Livingston & Huble [59] as implemented in [7]. The color sensitive receptive fields of V1 and their connectivity to color sensitive cells in V4 is depicted in Figure 3.1. Note that this setting is specific to our framework and not in general. This setting, as well as most parameters that we used in CARLsim configuration and setup, is the result of a trial and error process over the benchmark images in Figure 3.2.

In V1 there are four double-opponent cells: red center-green surround, green center-red surround, yellow center-blue surround and blue center-yellow surround. These types of cells that are known to be present in area V1 [90] respond well to a spot of one color on its opponent color and thus form the basis of color contrast and color constancy. The standard way to model them is to use the Difference of Gaussian (DoG), i.e. the difference of two Gaussian functions with different widths where the width of the center is smaller or

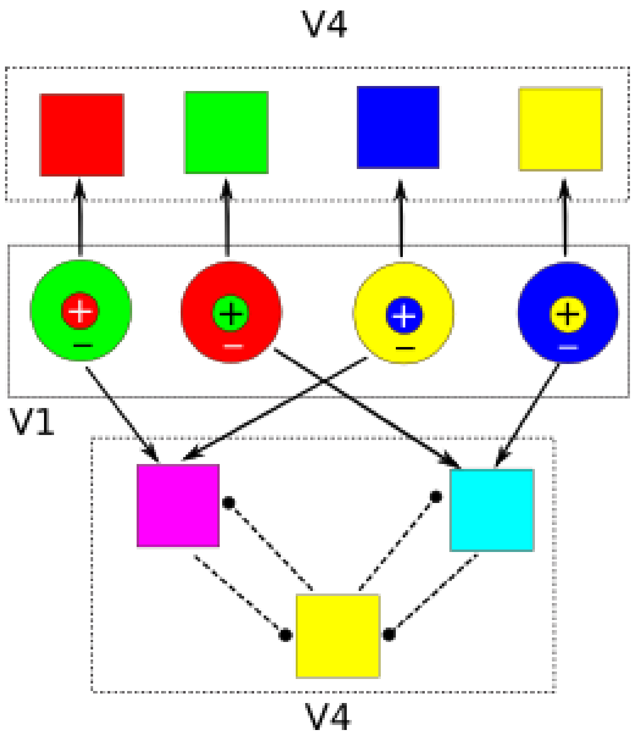


Figure 3.1: Connections between center-surround(double-opponent) cells in area V1 and hue sensitive cells in area V4. The arrow heads are excitatory connections and circle heads are inhibitory connections. Every cell selective to four primary colors in V4 is connected to the cell in V1 that its center has the same color as it. However Magenta and Cyan are connected to 2 different cells in V1 as well inhibitory connections to yellow sensitive cell in V4. This setting is specific to our model. To obtain that we used benchmark image in Figure 3.2.

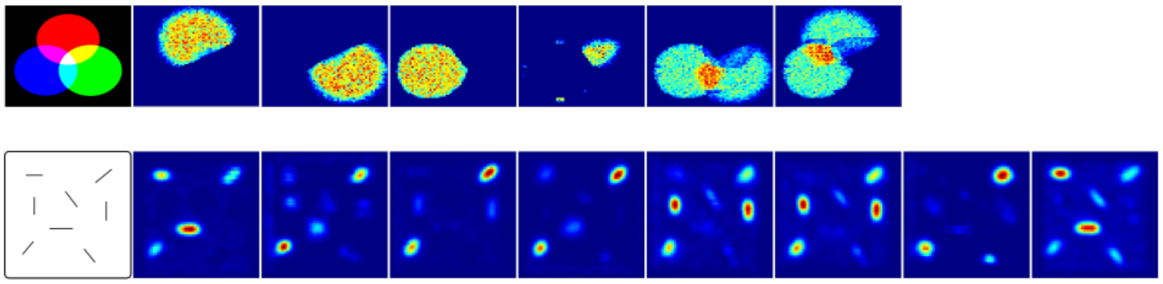


Figure 3.2: The benchmark images that we used to tune the parameters and set the connections in CARLsim. top: a colormix image including all 6 colors in V4 area in our model. The darker pixel means more spikes were generated by neuron corresponding to that pixel. bottom: a collection of bars oriented differently. The responses are from 0 to $7\pi/8$ left to right.

mathematically:

$$I_{\text{c-s}} = I_c \star G_{\sigma_{cen}} - I_s \star G_{\sigma_{sur}} \tag{3.1}$$

where (c,s) $\in$ {(red,green),(green,red),(yellow,blue),(blue,yellow)}, $\sigma_{cen} < \sigma_{sur}$ are the width of the Gaussian kernel and $\star$ denotes the convolution operation. Here each of red, green and blue colors are one of the channels in the input RGB stimuli and we form the yellow color according to the following formula:

$$I_{\text{yellow}} = \left[\frac{I_{\text{red}} + I_{\text{green}}}{2} - \frac{|I_{\text{red}} - I_{\text{green}}|}{2} - I_{\text{blue}}\right]^{\geq 0} \tag{3.2}$$

where $[\cdot]^{\geq 0}$ means we zero out potential negative values. In our setting $\sigma_{cen} = 1.2$ and $\sigma_{sur} = 1.6$. There are also color sensitive cells in V4 but unlike cells in V1 their receptive fields code for hue rather than color opponency. Similar to [7] we considered six hue sensitive cells in area V4 both with excitatory and inhibitory cells. In Figure 3.1 the arrow-heads show excitatory connections while the circle-heads denote inhibitory connections. Each center-surround cell in V1 has an excitatory connection to one hue sensitive cell in V4 that is sensitive to the same color as its center. The exception is for cyan and magenta that are secondary colors and are connected to two different double-opponent cells.

3.3 Orientation feature extraction in our model

To model the orientation selective cells in V1 we follow [7] which is a modern implementation of the one in [93]. Here we only sketch the plan of calculations and skip the mathematical details and refer the interested reader to above references. This process can be summarized in 3 steps: first we compute a weighted sum of input stimuli. Then we apply a form of rectification (in this case half-square rectification) which is performed to address the deficiency of having negative firing rates. Finally, response normalization is applied to account for nonlinearities. The first 2 steps calculate the linear response of V1 cells and can be described mathematically as:

$$L = \alpha_{v1lin} M b, \tag{3.3}$$

where α_{v1lin} is a scaling factor, M and b are 28×28 and 28×1 matrix and vector respectively. Each element of b is a third derivative of Gaussian that is used as a good approximation to Gabor filters that are normally used for orientation feature extraction and each row and column of M address one spatio-temporal component of one of 28 vectors on a dome. Note here that we are using the V1-MT connectivity as an orientation detection tool while in the above references it codes for direction and speed of motion as is known from biological studies. Thus we made some modification to the original MATLAB script `projectV1toMT.m` in [93]. This function computes the projections from V1 to MT. In their setting V1 has 28 different space-time oriented filters at each pixel location. Those filters need to be projected onto MT neurons selective to 8 directions and 3 non-zero speeds. Since those directions are direction of motion they set $\theta \in \{0, \pi/4, \pi/2, \cdots, 7\pi/4\}$ but we need to detect oriented bars or edges so we change it to $\theta \in \{0, \pi/8, \pi/4, \cdots, 7\pi/8\}$. this way we cover a wide range of oriented bars or edges and also we only consider the speed of 0 since we are dealing with still images. The last step is to normalize this response.

3.3.1 Post processing algorithms for saliency detection

Almost all saliency detection algorithms, no matter what category they belong to, either being biologically inspired or purely computational, look for rarity or distinctiveness in images [15, 11, 31, 80]. In our framework this process is accomplished using our post processing algorithms outlined in algorithms 1 and 2. For color feature extraction we consider six color sensitive cells in V4, namely, red, green, blue, yellow, cyan and magenta. After generating the spikes in V4 and MT area as color and orientation features respectively, we pass them to our post processing algorithms to extract color and orientation salient regions. So, in order to find the distinction of response to a specific color in area V4 compared to other colors, we subtract the weighted sum of responses to all other colors from the target color as in line 4 of algorithm 1. This way the remaining values denote the amount of distinctiveness in that specific color compared to other color responses. Note that we are also using a constant α_c, for c being one of those six colors in order to give more weight to some colors in final saliency map. In algorithm 1 $\alpha_{red} = 0.8$ and $\alpha_{yellow} = 0.9$ and $\alpha_c = 1$ for c any other color and hence red and yellow will have stronger roles in saliency [28]. The process of choosing rare pixels is done in line 10. N_c denotes the number of pixels that their value is larger than a threshold(here 0.2 times of maximum value). We multiply the saliency of each color by $\frac{1}{N_c+1}$ and thus we mitigate the effect of a very large area of one color in an image, because such regions are not rare at the end. Also notice that we zero out values that are lower than some threshold (line 6 of Algorithm 1). The reason for this is that if you run the CARLsim simulation for long enough then almost every neuron corresponding to each pixel will fire eventually and so we must guarantee that we are only considering the strong responses. Also notice that we smooth the results using a convolution kernel multiple times (lines 7 and 11 of Algorithm 1). This is because the operations before them will introduce some discontinuity in the response matrix, however, we expect that if a region is salient it is continuous enough so in this way we smooth the remaining regions. Finally we normalize the map to get the color saliency S_c.

Similarly for orientation saliency we look for rare and distinct regions. The process is the same in general with some minor differences. For example in line 7 when we try to down weight the large regions we multiply by $(\frac{1}{N_\theta})^2$ where N_θ is the number of pixels greater than some threshold. Moreover, in line 9, that we aim to extract the distinct regions we add back half of the current saliency map for two immediate before and after orientations. The reason for this is that the angles are only $\pi/8$ apart so it is reasonable to see strong response to an edge of some orientation, say $\pi/4$ from cells selective to $3\pi/8$ and $5\pi/8$. similar to color saliency we normalize the orientation saliency S_o at the end.

Once we have the color(S_c) and orientation(S_o) saliency maps, we can compute the saliency map of the image by smoothing the average of S_c and S_o as $S = \frac{S_o+S_o}{2} \star K$. The kernel K that we use here, is a simple average kernel like $K = \frac{1}{9}\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$, but it can be any other smoothing kernel depending on dataset or image itself.

Algorithm 1 Computing Color Saliency Map

```
1: $C = \{\text{red, green, blue, yellow, magenta, cyan}\}$
2: input: $R_c$                                   ▷ Response of V4 cells for each $c \in C$
3: for $c \in C$ do
4:     $R_c \leftarrow R_c - \alpha_c \sum_{c' \neq c} R_{c'}/5$
5:     $M_c \leftarrow \max(R_c)$
6:     $R_c(R_c < 0.7M_c) = 0$                    ▷ Only keep the highest 30% values
7:     $S_c \leftarrow R_c \star K$               ▷ Convolution with an averaging kernel $K$
8:     $m_c \leftarrow \max(S_c)$
9:     $N_c \leftarrow |S_c(S_c > 0.2m_c)|$▷ Number of elements in $S_c$ that are greater than 0.2 times of
   the maximum value
10: $S_c \leftarrow \sum_c \frac{1}{1+N_c} S_c$
11: $S_c \leftarrow S_c \star K$                 ▷ Convolution with an averaging kernel $K$
12: $S_c \leftarrow S_c / \max(S_c)$            ▷ Normalizing the result
```

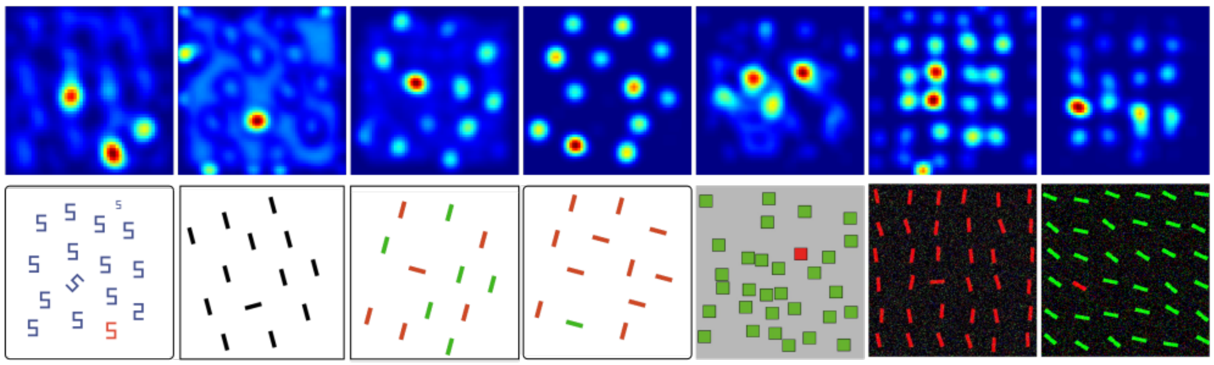


Figure 3.3: The output of our algorithm on some synthetic image made specially for saliency detection algorithms. The first 4 images are from Toronto dataset[15] and last 3 are from pattern category of CAT2000 dataset [12].

Algorithm 2 Computing Orientation Saliency Map

```
1: $\Theta = \{0, \pi/8, \pi/4, 3\pi/8, \pi/2, 5\pi/8, 3\pi/4, 7\pi/8\}$
2: **input:** $R_\theta$                                  ▷ Response of MT cells for each $\theta \in \Theta$
3: **for** $\theta \in \Theta$ **do**
4:     $s_\theta = R_\theta \star K$
5:     $M_\theta \leftarrow \max(s_\theta)$
6:     $N_\theta \leftarrow |s_\theta(s_\theta \geq 0.5M_\theta)|$     ▷ Number of elements that are at least half of maximum value
7:     $s_\theta \leftarrow \left(\frac{1}{N_\theta}\right)^2 s_\theta$
8: **for** $\theta \in \Theta$ **do**
9:     $S_\theta = s_\theta - \sum_{\theta' \in \Theta, \theta' \neq \theta} s_{\theta'} + 0.5 s_{\theta+\pi/8} + 0.5 s_{\theta-\pi/8}$
10:    $S_\theta(S_\theta < 0) = 0$
11: $S_o \leftarrow \sum_\theta S_\theta$
12: $S_o \leftarrow S_o \star K$                              ▷ Convolution with an averaging kernel $K$
13: $S_o \leftarrow S_o / \max(S_o)$                         ▷ Normalizing the result
```

3.4 Results

In general, two types of images are used to tune and evaluate the saliency detection algorithms. Some of them contain regions where their conspicuity is clear. These images are either hand crafted synthetic images (see Figure 3.3) or hand picked natural images [15]. These kinds of images are ideal to examine the performance of an algorithm by eye. On the other hand datasets also contain random natural images that do not have trivial salient regions. To evaluate the saliency algorithms over these datasets one needs ground truth (see section evaluation). In this paper we evaluate our model on different datasets. Salient regions in an image might be the result of different color, orientation/curvature, luminance, density, etc and the use of each of these features depends on the design of the algorithm. In our algorithm we used spikes generated in the V4 and MT area in the visual cortex in response to color and orientation and thus our method is more targeted at multi color images with objects oriented differently. For example in images like the second one in Figure 3.3 the color pathway in our framework is redundant and does not add more accuracy to the final result.

3.5 Model evaluation

To evaluate our model's performance we used five quantitative evaluation metrics: *Similarity*(SIM) that ranges in $[0, 1]$ where 0 means no overlap between ground truth and prediction and 1 indicates complete similarity. *Normalized Scan-path Saliency* (NSS) with theoretical values in $[-\infty, \infty]$ where 0 means chance and any positive value indicates performance above chance. *Correlation Coefficient* (CC) assumes values in $[-1, 1]$ with positive value indicating correlation between model prediction and fixation maps and negative values denoting decorrelation. *Information Gain* (IG) that is similar to NSS and finally *Kullback-Leibler divergence* (KL) with values in $[0, \infty]$ where lower values means better performance by the model. The first 4 metrics are similarity metrics in the sense that the higher score means the saliency map and fixation map are more similar and hence better performance by the model. The last one (KL) is a dissimilarity metric, i.e. the lower score means better performance by the model. We skip the mathematical definition of these metrics here and refer the reader to [17]. We also used the MATLAB implementations of these metrics provided by the above reference. They also cover a thorough discussion of various evaluation metrics. figure 3.7 shows the best and worst performance of our model on the Toronto dataset [15] for three similarity and KL dissimilarity metrics. Note that in the case of KL-divergence lower score means better performance. In the following we explain each of these measures briefly. To see a thorough review and comparison see [17].

Normalized scanpath saliency(NSS)

Given a fixation map F and a saliency map S, NSS can be defined as:

$$\begin{aligned} \text{NSS}(S,F) &= \frac{1}{N}\sum_m \sum_n \bar{S}_{mn}\bar{F}_{mn} \\ \text{where,} \qquad \bar{F} &= \frac{F-\mu_F}{\sigma_F} \end{aligned} \tag{3.4}$$

and m, n are spatial locations. N is the total number of fixation locations, i.e. the nonzero pixels of fixation density map. μ_F and σ_F are the mean and standard deviation of the fixation map respectively, so $\bar{F}$ is the normalization of F. The theoretical range for NSS score is $(-\infty, \infty)$, but its empirical range depends the data set and is much smaller [17]. Positive NSS values indicate that the model performs above chance. All false positives contribute to lowering the NSS and thus NSS is very sensitive to false positives.

Similarity(SIM)

Similarity is defined as the sum of the minimum values of normalized fixation map F and saliency map S:

$$\begin{aligned} \text{SIM}(S,F) &= \sum_m \sum_n \min(\bar{S}_{mn}, \bar{F}_{mn}) \\ \text{where,} \qquad & \sum_m \sum_n \bar{F}_{mn} = \sum_m \sum_n \bar{F}_{mn} = 1 \end{aligned} \tag{3.5}$$

and m, n are pixel locations. SIM ranges in $[0, 1]$ where 1 indicates F and S have identical distribution. SIM is very sensitive to missing values and penalizes for every ground truth value that is not detected by the model.

Pearson's Correlation Coefficient(CC)

CC is a statistical measure that can compute how correlated two variables are. If we assume saliency and fixation maps S and F as random variables then:

$$\text{CC}(S,F) = \frac{\sum_m \sum_n (S_{mn}-\mu_S)(F_{mn}-\mu_F)}{\sqrt{(\sum_m \sum_n (S_{mn}-\mu_S)^2)(\sum_m \sum_n (F_{mn}-\mu_F)^2)}} \tag{3.6}$$

CC is equally sensitive to both false negatives and false positives. It ranges between $[-1, 1]$, higher positive CC value means S and F have more pixel locations with similar magnitude.

Kullback-Leibler divergence(KL)

KL-divergence measures the difference between two probability distributions. We use a variant of it used and provided in [17] and defined as:

$$\mathrm{KL}(S, F) = \sum_m \sum_n F_{mn} \times \log(\epsilon + \frac{F_{mn}}{\epsilon + S_{mn}}) \tag{3.7}$$

where, S and F are normalized such that $\sum_m \sum_n F_{mn} = \sum_m \sum_n S_{mn} = 1$. ϵ is a regularization constant (here MATLAB's built-in ϵ). KL is a dissimilarity measure ranging from $[0, \infty)$ and thus a lower score means less dissimilarity.

3.5.1 Information gain(IG)

Information gain first introduced in [51] evaluates a saliency models performance versus a baseline map. In other words, IG measures the information gained by the output of a model over some baseline. The baseline could be a center prior, such as a 2D Gaussian centered at the center of the image or an image with random pixel values, see Figure 3.5. Let F be a binary fixation map, S the saliency map, and B the baseline map, then IG is defined as:

$$IG(S, F) = \frac{1}{N} \sum_m \sum_n F_{mn}(\log_2(\epsilon + S_{mn}) - \log_2(\epsilon + B_{mn})), \tag{3.8}$$

where, m, n are spatial locations, N is the total number of fixated pixels, ϵ is regularization constant. IG ranges between $(-\infty, \infty)$. A positive score means that the model predicts the fixated locations better than baseline map. In Figure 3.4 the output of our model over nine images from different categories of CAT2000 train set is compared with the fixation maps of the data set . The visualizations in D show the performance of the model over center prior baseline. The numbers in E and F are corresponding IG scores for center prior and chance maps respectively.

Table 3.1 shows the average score of all five metrics on the Toronto dataset and also the pattern

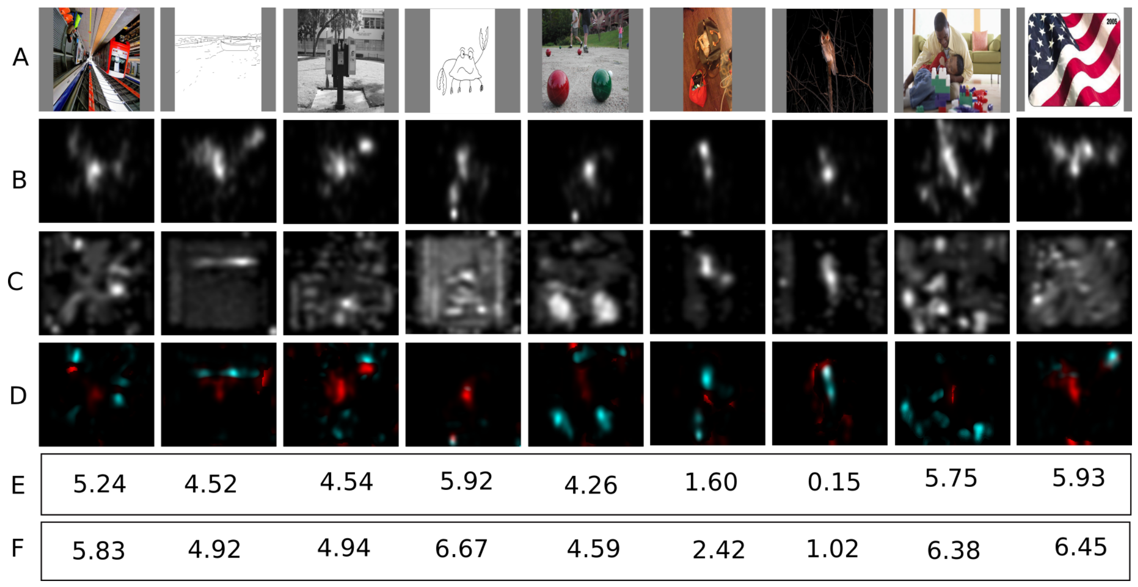

A
B
C
D
E 5.24 4.52 4.54 5.92 4.26 1.60 0.15 5.75 5.93
F 5.83 4.92 4.94 6.67 4.59 2.42 1.02 6.38 6.45

Figure 3.4: Evaluation of our model over 9 images from different categories of CAT2000 dataset. A: original images, B:fixation maps, C: output og our model, D:performance of our model over center-prior baseline(The green area means better performance compared to baseline and red regions means the model is inferior to baseline), E and F: IG score compared to two baseline maps in Figure 3.5.

category of CAT2000 dataset [12]. Note that for information gain(IG) metric, a baseline map must be provided, say center prior or a random map. IG measures the amount of information captured by the saliency model compared to the baseline map, see Figure 3.5 and a positive score indicates that the saliency model is predicting the fixated locations better than the baseline.

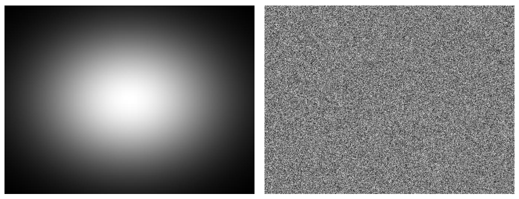


Figure 3.5: Baseline maps used to measure information gain (IG) score. Left: Center prior, 2D Gaussian. Right: An image with random pixel values.

Figure 3.6 shows a visual comparison between the output of our model and Itti's model [39] on seven images from MIT300 dataset [16]. The ground truth for this dataset is not released. Please note that Itti's model is using a larger blurring kernel.

Figure 3.7 represents the best and worst performances of our model on the Toronto dataset. As can be seen, whenever the image has clear salient regions that show significant distinction in color

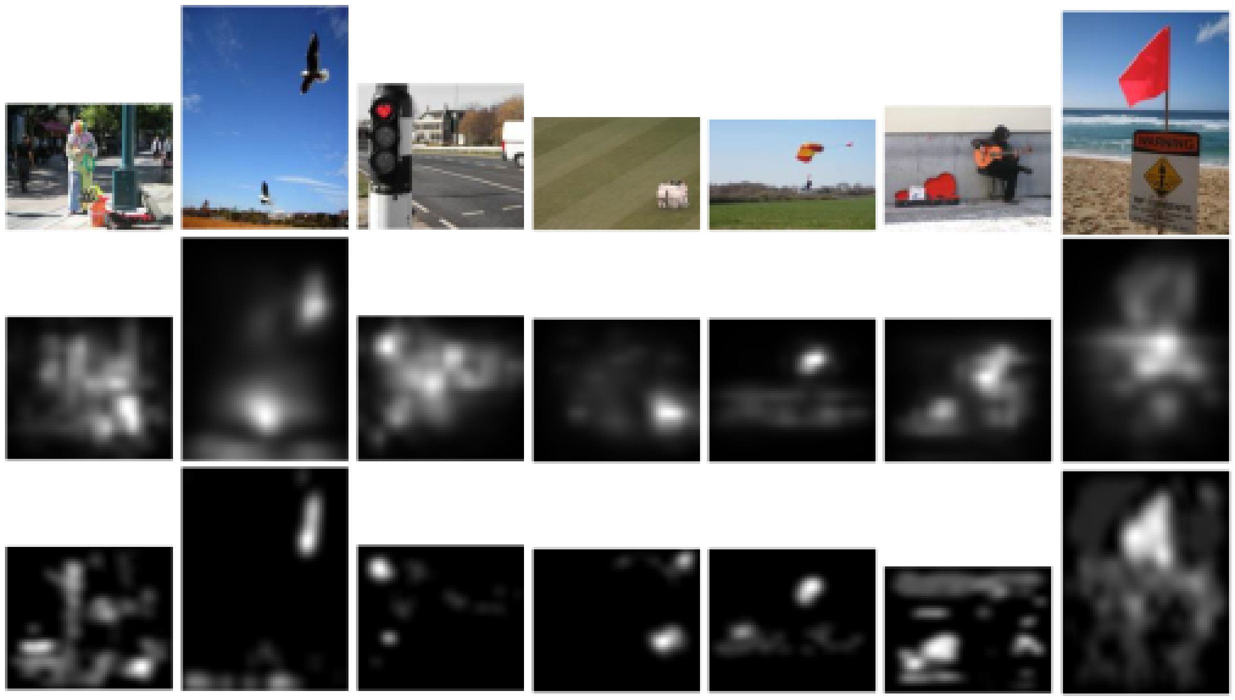


Figure 3.6: A comparison between the output of our model and Itti's model [39]. First row: seven pictures from MIT300 dataset [16]. Second row: output of Itti's model with default parameters. third row : the output of our model. Please notice that in our model's setup we removed the least 10% of values and used a small smoothing kernel.

or orientation, then our model performs very well. Like the piece of cloth on the turf in the third image in the left panel. But in images that these visual cues are not trivial enough, our model shows weakness.

	SIM↑	NSS↑	CC↑	KL↓	IG(center prior)↑	IG(chance)↑
CAT2000[1]	0.33	0.24	0.24	6.73	3.96	3.33
Toronto	0.33	0.21	0.16	3.60	4.93	3.77

Table 3.1: The average score of 4 saliency metrics on 2 datasets. The first row shows the score for pattern category in CAT2000 dataset. Second row shows average scores for toronto dataset.

Figure 3.8, compares our model with the fixation maps side by side on nine images of the Toronto dataset. One characteristic of the Toronto dataset is that while images are taken from natural scenes, they have elements that clearly stand out against their neighboring regions.

Figure 3.9 gives a side-by-side comparison of our model and fixation maps over nine images

[1]only pattern category

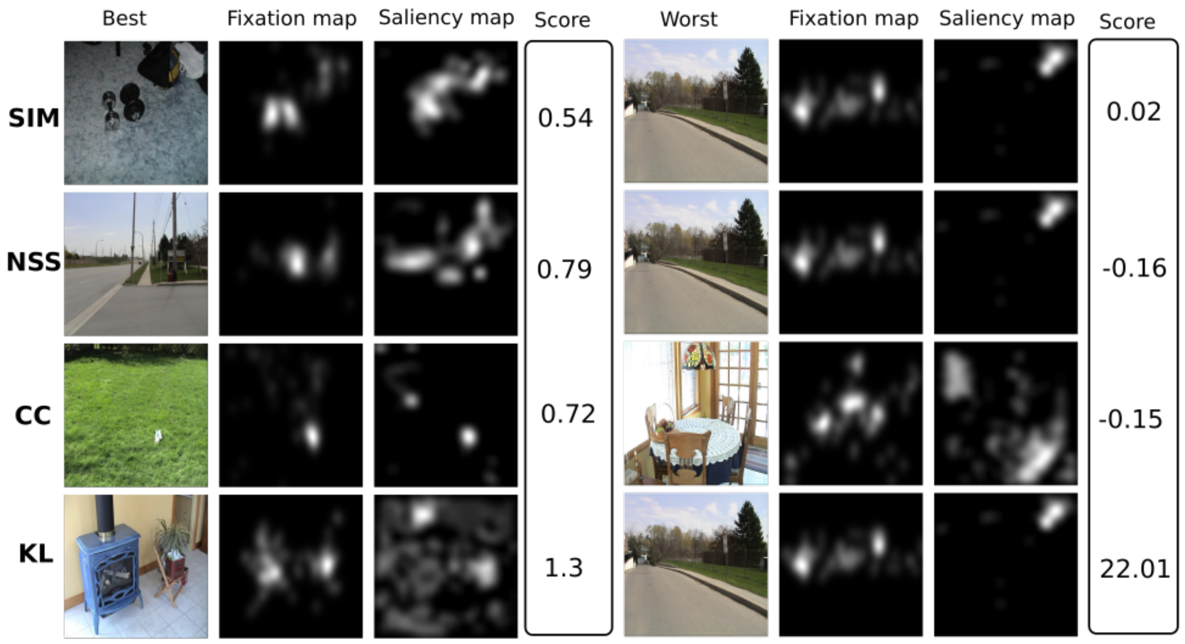


Figure 3.7: The best and worst result of our algorithm on Toronto dataset. The first column in left panel consists of images that our model performed best according to each metric. The second and third columns are ground truth and prediction respectively. The fourth column shows the corresponding scores. The right panel is similar to left but consists of images that our model did worst. Notice that in three metrics images are the same.

from the Toronto dataset. In the last column the information gain is visualized with respect to the center prior baseline (see Figure 3.5). The green areas represent the regions that our model is outperforming the baseline map and red areas represent the pixels that are predicted better by the baseline map. For example in the fifth image the baseline is catching the salient regions better but it is surprising as the only salient region in that image is the door knob that is located almost at the center.

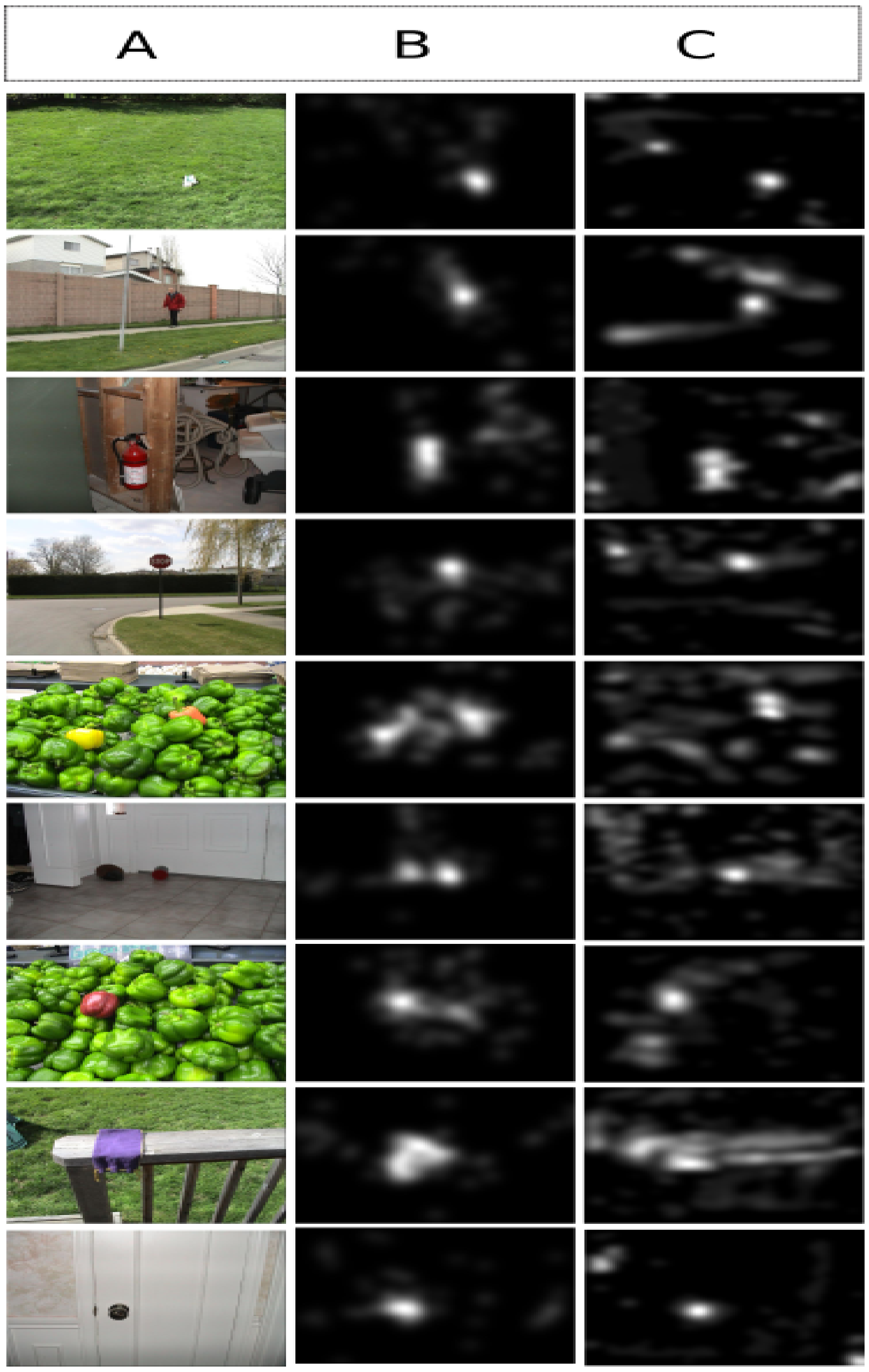


Figure 3.8: A: Nine images from Toronto data set. B: Fixation maps (ground truth). C: Output of our model. The score for four evaluation metrics NSS, SIM, CC, and KL are given in Table 3.1

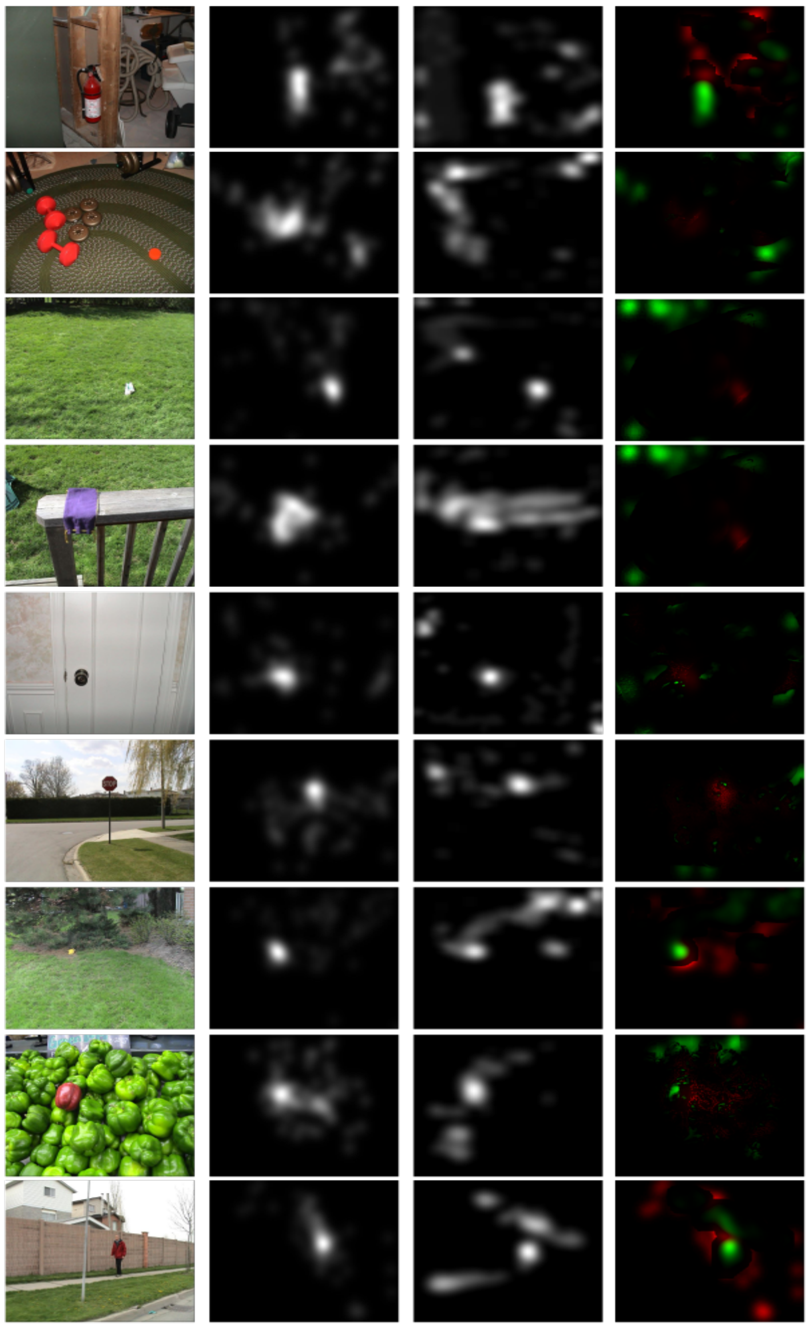


Figure 3.9: Nine sample images from Toronto dataset [15]. salient regions in these images are easily discernible. 1st column: original images, 2nd column: ground truth, 3rd column: the output of our framework, 4th column: visualization of IG w.r.t center prior (see Figure 3.5). The green regions are pixels that our model predicted better than baseline and red regions are pixels that our model performed worse than baseline.

Chapter 4

Object recognition and classification

The visual system rapidly and effortlessly recognizes a large number of diverse objects in cluttered, natural scenes. In particular, it can easily categorize images or parts of them, for instance as faces, and identify a specific one. Despite the ease with which we see, visual recognition, one of the key issues addressed in computer vision, is quite difficult for computers and is indeed widely acknowledged as a very difficult computational problem [75]. The problem of object recognition is even more difficult from the point of view of neuroscience, since it involves several levels of understanding from the information processing or computational level to the level of circuits and of cellular and biophysical mechanisms. After decades of work in striate and extrastriate cortical areas that have produced a significant and rapidly increasing amount of data, the emerging picture of how cortex performs object recognition is in fact becoming too complex for any simple, qualitative mental model [88]. The key computational issue in object recognition is the *specificity-invariance* trade-off: recognition must be able to finely discriminate between different objects or object classes while at the same time be tolerant to object transformations such as scaling, translation, illumination, viewpoint changes, change in context and clutter, non-rigid transformations (such as a change of facial expression) and, for the case of categorization, also to shape variations within a class. Thus the main computational difficulty of object recognition is achieving a very good trade-off between selectivity and invariance [77].

Human visual recognition is a far from trivial feat of nature. The light patterns projected on the human retina are always changing, and an object will never create exactly the same pattern twice. Objects move and transform constantly, appearing under an unlimited number of aspects. Yet, the human visual system can recognize objects in milliseconds. It is thus natural for computer vision models to draw inspiration from the human visual cortex.

4.1 Computational models of object recognition

We have amazing ability at recognizing objects by sight and it involves any variation of object representation. We recognize and classify objects even by seeing a few strokes of pen done by a talented artist. Also, different angles of view, different intensities or occlusion have minimum effect on our perception ability. Thus it worth to talk a little about computational theories of object recognition.

The task that we will discuss in this thesis usually is called object classification. Classification can be considered as a subtask of recognition. However, in the former one we are interested in assigning class labels rather than determining the instances of a class. There are various theories aimed at explaining how our brain does the recognition tasks. We do not intend to give a complete review of existing models here but only briefly discuss the one that is closely related to our approach in this thesis. For more details please refer to [30].

Many of the existing models of object recognition can be traced back to some of the original ideas developed by Hubel and Wiesel [37, 38] and the influential works of David Marr [63, 62]. The approach that we take here can be categorized in the models that are based on Marr and Nishihara's model.

4.1.1 Marr and Nishihara's model

In order to judge theoretical proposals regarding different aspects of object recognition theories, Marr and Nishihara set out three criteria that must be met by a good representation of a shape:

1. **Accessibility:** Can the representation be completed easily?

2. **Scope and Uniqueness:** Is the representation suitable for the class of shapes with which it has to deal (scope), and do the shapes in that class have one or more canonical represen-

tations(uniqueness)?

3. **Classification and identification:** The term that Marr and Nishihara have used originally was *stability* and *sensitivity*. It states that can the representation be used to capture both the similarities (for deciding class membership) and the differences (for identifying specific instances) between the objects it has to deal with?

Their proposal is an attempt to address a very important question: should there be a flat organizational structure or should there be a hierarchical organization, in which features in lower levels are combined in higher levels. A hierarchical scheme has shown to be more successful in classification/recognition tasks [77, 89, 78].

4.1.2 Learning and plasticity

In this short section we briefly revisit the learning theory in spiking neurons, specifically we are interested in Hebbian learning as it is the backbone of the learning process and plasticity that is behind the spiking neural network that are integral to our research. The interested reader is referred to [23].

In 1949, Donald Hebb conjectured that if input from neuron A often contributed to the firing of neuron B, then the synapse from A to B should be strengthened [34]. Hebb suggested such synaptic modification could produce neuronal assemblies that reflect the relationships experienced during training. Hebb rule is the basis of many researches in the role of synaptic plasticity in learning and memory. The simplest plasticity rule that is also close to the Hebb idea can be expressed as:

$$\tau_w \frac{d\mathbf{w}}{dt} = \nu \mathbf{u} \tag{4.1}$$

where, τ_w is the time constant that controls the rates at which the weights change. w is the vector of all synaptic weights. u is the presynaptic activity and ν is the post-synaptic activity. Synaptic plasticity is generally modeled as a slow process that gradually modifies synaptic weights over a time period.

4.2 Prototype learning and dimension reduction

4.2.1 Dimension reduction

Dimensionality reduction, or dimension reduction, is the transformation of data from a high-dimensional space into a low-dimensional space so that the low-dimensional representation retains some meaningful properties of the original data, ideally close to its intrinsic dimension. Working in high-dimensional spaces can be undesirable for many reasons; raw data are often sparse and analyzing the data is usually computationally intractable. Dimensionality reduction is common in fields that deal with large numbers of observations and/or large numbers of variables, such as signal processing, speech recognition, neuroinformatics, and bioinformatics.

Methods are commonly divided into linear and nonlinear approaches. Approaches can also be divided into feature selection and feature extraction. Dimensionality reduction can be used for noise reduction, data visualization, cluster analysis, or as an intermediate step to facilitate other analyses.

The curse of dimensionality refers to various phenomena that arise when analyzing and organizing data in high-dimensional spaces that do not occur in low-dimensional settings such as the three-dimensional physical space of everyday experience. The expression was coined by Richard E. Bellman [5] when considering problems in dynamic programming. Dimensionally cursed phenomena occur in domains such as numerical analysis, sampling, combinatorics, machine learning, data mining and databases. The common theme of these problems is that when the dimensionality increases, the volume of the space increases so fast that the available data become sparse. This sparsity is problematic for any method that requires statistical significance. In order to obtain a statistically sound and reliable result, the amount of data needed to support the result often grows exponentially with the dimensionality. Also, organizing and searching data often relies on detecting areas where objects form groups with similar properties; in high dimensional data, however, all objects appear to be sparse and dissimilar in many ways, which prevents common data organization strategies from being efficient.

4.2.2 Uniform Manifold Approximation and Projection(UMAP)

UMAP is a highly scalable, novel manifold learning technique for dimension reduction. Similar to all manifold learning algorithms it assumes that the observed data lie on a low-dimensional manifold embedded in a higher-dimensional space [66]. Intuitively, this assumption, which is known as the manifold assumption or sometimes the manifold hypothesis, states that the shape of our data is relatively simple. UMAP is constructed from a theoretical framework based in Riemannian geometry and algebraic topology. Here we follow [65] and briefly explain the UMAP algorithm from a computational perspective but also warn that there is an extensive mathematical theory behind it. The interested reader is referred to the original report cited above.

Any algorithm that employs a mathematical structure similar to graph neighborhood for approximating a manifold that fits the data must follow some basic flow that can be put into two stages: *Graph Construction* and *Graph Layout*. In the first stage, the algorithm must form a weighted k-neighborhood graph, then apply some transform on the edges of the graph and deal with the inherent asymmetry of the k-neighbor graph. In the second stage, the algorithm must define an objective function in a way that preserves the characteristics of the k-neighborhood graph and finally find a low dimensional representation of the original dataset that optimizes this objective function. In the following we explain in steps in a more formal way but we skip all mathematical justification as well as the implementation details of the algorithm. To see a thorough explanation with complete proof of all theorems see [65].

- **Graph construction:** Let $X = \{x_1, \cdots, x_N\}$ be the input dataset, equipped with a metric $d : X \times X \to \mathbb{R}^{\geq 0}$. For a given hyperparameter k for each x_i we compute the set $\{x_{i_1}, \cdots, x_{i_k}\}$ of k nearest neighborhood of x_i according to metric d. The choice of this nearest neighborhood algorithm is not important. For each x_i define ρ_i as: $\rho_i = \min\{d(x_i, x_{i_j}) | 1 \leq j \leq k, d(x_i, x_{i_j}) > 0\}$ and find σ_i that satisfies the following equation:

$$\sum_{j=1}^{k} \exp\left(\frac{-\max(0, d(x_i, x_{i_j}) - \rho_i)}{\sigma_i}\right) = \log_2(k) \tag{4.2}$$

The selection of ρ_i ensures that x_i connects to at least one other data point with an edge of weight 1 and aims at improving the representation on very high dimensional data. The σ_i is a normalization factor that smooths out the manifold structure so keep it Rimannian at least locally around x_i. Now we have enough tools to define the weighted directed graph $\bar{G}(V, E, w)$. The vertices V are simply the set X. The set of directed edges is $E = \{(x_i, x_{i_j}) | 1 \leq j \leq k, 1 \leq i \leq N\}$ and the weight function can be defined as:

$$\exp\left(\frac{-\max(0, d(x_i, x_{i_j}) - \rho_i)}{\sigma_i}\right) \tag{4.3}$$

- **Graph layout:** The definition of objective function and then solving for a low dimensional representation that optimizes it, is based on attractive-repulsive forces applied along the edges and among the vertices respectively. The algorithm applies these forces iteratively and local minima is achieved by slowly decreasing the attractive and repulsive forces. In UMAP the attractive force on an edge connecting two vertex i and j at coordinates $\mathbf{y}_i$ and $\mathbf{y}_j$ respectively, is determined by:

$$\frac{-2ab\|\mathbf{y}_i - \mathbf{y}_j\|^{2(b-1)}}{1 + \|\mathbf{y}_i - \mathbf{y}_j\|^2} w((x_i, x_{i_j}))(\mathbf{y}_i - \mathbf{y}_j) \tag{4.4}$$

  where, a and b are hyperparameters. The repulsive force is given by:

$$\frac{2b}{(\epsilon + \|\mathbf{y}_i - \mathbf{y}_j\|^2)(1 + a\|\mathbf{y}_i - \mathbf{y}_j\|^{2b})}(1 - w((x_i, x_{i_j})))(\mathbf{y}_i - \mathbf{y}_j) \tag{4.5}$$

  ϵ is a small number to avoid division by zero. Note that this repulsive force is only applied to a sampled vertices of an edge that an attractive force is already applied to it.

4.2.3 Prototype-based learning

The main concept of the prototype-based methods is to represent previously observed data in terms of a handful of data so-called prototypes, which reflect typical properties of the data. Equipped with a suitable, similarity or dissimilarity measure, prototypes can be used for the classification of complex and possibly high-dimensional data.

Prototype-based models constitute a very successful family of methodological approaches in machine learning (see e.g. [10, 9]). They are appealing for a number of reasons: The extraction of information from previously observed data in terms of typical representatives, so-called prototypes, is particularly straightforward and intuitive The same is true for the working phase, in which novel data are compared with the prototypes by use of a suitable (dis-)similarity or distance measure.

Prototype systems are often employed for unsupervised learning where they are used for detection of underlying structures, such as clusters (see for instance [9]). Vector Quantization or the well-known K-means algorithm are prominent examples for the use of prototypes in the context of unsupervised learning ([27, 41]).

Among the many frameworks developed for supervised machine learning, prototype-based systems are particularly intuitive, flexible, and easy to implement. Various prototype-based classifiers have been considered in the literature, some of them are derived from well-known unsupervised schemes like the Self-Organizing Map or the Neural Gas [45]. which can be equipped with a posterior labeling of prototypes. Here, our focus is on the so-called Learning Vector Quantization (LVQ), a supervised framework which was originally suggested by Kohonen [45]. But, prior to that, As a starting point for the discussion, we briefly revisit its unsupervised predecessor, Self Organizing Map(SOM) as we leverage the former one for object classification.

- **Self-Organizing Maps(SOM):** The Self-Organizing Map(SOM) is an automatic data-analysis method [48] that is related to the classical vector quantization(VQ) methods and is extensively applied to clustering problems in a wide range of applications in science and engineering [49]. SOMs are a type of artificial neural network that implement a characteristic nonlinear projection from a high-dimensional space of input signals onto a low-dimensional(usually 2-dimension) array of neurons. They are able to map a structured high-dimensional signal manifold onto a much lower dimensional network in an orderly fashion. The mapping tends to preserve the topological structure of the original data in signal domain.

  In section 2.2, we talked about single neural cells in the brain that respond selectively to some specific sensory stimuli. The local assemblies of these cells, called brain maps, are tuned to be sensitive to some feature value of a specific stimulus. It is found that these brain maps are not genetically determined but depend on sensory experiences [38]. So this question was inevitable: whether feature-sensitive cells could be formed also in artificial systems automatically, during the learning process (i.e. as a result of adaptation to stimuli). SOMs were one of the first successful answers to this question. Self-organizing maps were first introduced during 1981-82 by Teuvo Kohonen [46]. As mentioned previously, SOMs are some form of vector quantization. In vector quantization, the space of input feature vectors is partitioned into a finite number of contiguous regions, and each region is represented by a single model vector, or codebook vector. Note that this representation is optimal in the sense that the codebook vectors are constructed in a way that the mean distance between some input data and the best-matching codebook vector is minimized according to some metric.

The working process of an SOM can be explained in an intuitive manner: assume that we want to map a set of n-dimensional real vectors $\mathbf{x}$ to a 2-dimensional vector space, then we Consider a 2-dimensional grid of neurons(nodes) and to each node we assign a weight vector $\mathbf{m}$ of the same dimension as $\mathbf{x}$. At the beginning these weight vectors can be initialized in any way, say randomly. In the first iteration when the first input arrives, all nodes compete for winning the input according to their distance to the input vector. The winner node (also called Best-Matching Unit or BMU) updates its weight and some of its neighboring nodes- according to some neighborhood function- to move closer to the input signal. This process continues recursively until the algorithm converges to the optimal limit. Now we explain this algorithm formally but before that, we must point out that there are various implementations for SOM [48] and we only explain the original algorithm which is also the computationally simplest one.

Let $\{\mathbf{x}(t)\}$ be a sequence of input data items and each $\mathbf{x}$ is a real n-dimensional vector and t an integer signifies a step in the sequence. Also assume that $\{\mathbf{m}_i(t)\}$ is another real sequence of n-dimensional vectors that represent the codebooks computed successively. Here i is the spatial index of the grid node associated to $\mathbf{m}_i$. The update rule for recursively computing models $\mathbf{m}_i$ is:

$$\mathbf{m}_i(t+1) = \mathbf{m}_i(t) + h_{ci}(t)\left[\mathbf{x}(t) - \mathbf{m}_i(t)\right] \tag{4.6}$$

where, $h_{ci}(t)$ is neighborhood function. The subscript c is the index of the winner node, i.e. the node with the codebook $\mathbf{m}_c(t)$ that has the smallest Euclidean distance from $\mathbf{x}(t)$ or:

$$c = \underset{i}{\operatorname{argmax}}\{\|\mathbf{x}(t) - \mathbf{m}_i(t)\|\} \tag{4.7}$$

It is worthwhile to note that the distance measure need not be Euclidean(to see a variant of SOM that uses other distance measures refer to [2]). The neighboring function is a Gaussian-like function most of the times:

$$h_{ci}(t) = \alpha(t)\exp(\frac{\|c-i\|^2}{2\sigma^2(t)}) \tag{4.8}$$

where, $\sigma(t)$ and $\alpha(t)$ are both monotonically decreasing functions. SOMs have been employed in numerous computer vision applications. In the following we briefly review some of these.

Lampinen et.al. [55, 54] used SOMs to cluster features extracted from images using Gabor filters and then used a multi-layer perceptron [32] to label these clusters. Their approach showed success in face recognition as well as wood defect detection. Our approach here is similar to theirs as we use Gabor-like receptive fields to extract features in the visual cortex. In [58], authors proposed a statistical SOM by fitting a one-dimensional density map to each voronoi cell [24] of nodes. Their motivation was to eventually compare the distribution of newly arrived data with trained prototypes instead of directly using a distance measure. This method could be more reliable in tasks like outlier detection. Later in [50], authors extended this idea by formulating the probability density models for which SOM training gives the maximum likelihood estimate [27], based on the local error function. The similar approach was taken in [53] for feature extraction and object retrieval, however they used the entropy as a measure of randomness of BMUs. In [72], authors designed a hierarchical two stages neural network based on SOMs for color image segmentation. Toivanen et.al. [98] used a regular SOM to order the pixels of a multispectral image and then used an edge detection method to extract edges. In [96] authors combined SOM with a k-nearest neighborhood(k-NN) for recognizing partially occluded face images from a single training image. Other than regular SOMs, there are lots of innovations in literature that aim at stacking SOMs in layers in order to improve their performance. In [57] and [102] the idea is to form deep architecture by stacking alternating layers of SOM and sampling, so when SOM finds BMU, the sampling layer picks BMUs and form another map for next SOM layer. Other instances of deep or stacked SOMs are proposed in [61, 29, 105, 26, 71]. Also, combining SOMs with convolutional neural networks(CNN) [32] is another novel approach in computer vision [95, 33] that has received some attention recently. And finally, another recent effort in extending SOMs is to add a supervised layer or leverage some supervision in the training phase to improve the performance [76]. These categories that are usually known as semi-supervised SOMs have been used for clustering and classification [14, 42] in general and remote sensing [82] in particular.

- **Learning Vector Quantization(LVQ):** The LVQ algorithms are related to other competitive learning algorithms such as SOMs, and k-means which themselves are based on the winner-takes-all learning rule [23] and algorithms in which only certain elements or neighborhoods are updated during the learning process. Like SOMs the aim of LVQ algorithms is to learn prototypes(codebook vectors) representing class regions, where each class region is determined by the Voronoi cell [24]of the prototype representing it. However, the major distinction is that unlike SOMs, LVQ algorithms are supervised. The LVQ algorithms are also introduced by Kohonen in the late 80's. Over several years, various versions of LVQ were introduced by Kohonen and his research team, mainly characterized by a number, like LVQ1, LVQ2, etc. [45, 47]. These classifiers are particularly intuitive and easy to understand. One advantage of the LVQ over so called black box methods like multilayer perceptron or SVM is that it is simple and fast as its speed depends on the number of prototypes which is a fixed number in contrast to SVM which is a function of dataset size [101]. However, this heuristic nature of them might also be a drawback as there is no mathematical justification that guarantees their convergence [70].

In order to address this issue, different variants of the LVQ algorithms have been developed that solve the problem of convergence by adding an explicit cost function from which to derive the learning rule via gradient descent. The LVQ methods can be categorized into three different families [70]: Kohonen's original method(heuristic), methods based on margin maximization, and methods based on likelihood ratio maximization. In this section we focus on the GLVQ [86] method. Later in this thesis when we employ LVQ for object classification on the Fashion-MNIST dataset, GLVQ is the version that we use. So, in the following we begin by explaining the LVQ algorithm in general and then introduce the cost function that is specific to GLVQ. Let $\{(\mathbf{x}_i, y_i) \subset \mathbb{R}^N \times \{1, \cdots, C\} | i = 1, \cdots, M\}$ be a training dataset, where $\mathbf{x} = (x_1, \cdots, x_N) \in \mathbb{R}^N$ are N-dimensional input samples, with cardinality $|\mathbf{x}| = M$ and $y_i \in \{1, \cdots, c\}, i = 1, \cdots, M$ are the sample labels, and c is the number of classes. The neural network consists of a number of prototypes, which are characterized by vectors $\mathbf{w}_i \in \mathbb{R}^N$, for $i = 1, \cdots, M$ and their class labels $c(\mathbf{w}_i) \in \{1, \cdots, C\}$ with $y = \{c(\mathbf{w}_i) \in \{1, \cdots, C\} | j = 1, \cdots, M\}$. The classification scheme is based on the best matching unit(BMU). The receptive field of prototype $\mathbf{w}_i$ is defined as: $\mathcal{R}^i = \{\mathbf{x} | \forall \mathbf{w}_j (j \neq i) \Rightarrow d(\mathbf{w}_i, \mathbf{x}) \leq d(\mathbf{w}_j, \mathbf{x})\}$. The GLVQ cost function is defined as:

$$E_{\text{GLVQ}} = \sum_{i=1}^{M} \phi(\mu(\mathbf{x}_i, \mathbf{w})), \tag{4.9}$$

where $\phi(x) = \frac{1}{1+e^{-x}}$ is the sigmoid function, and $\mu(\mathbf{x}_i, \mathbf{w}) = \frac{d^+ - d^-}{d^+ + d^-}$ is the relative distance difference. Here, $d^+ = d(\mathbf{x}_i, \mathbf{w}^+)$ is the Euclidean distance of data point $\mathbf{x}_i$ from its closest prototype $\mathbf{w}^+$ having the same class label, and $d^- = d(\mathbf{x}_i, \mathbf{w}^-)$ is the Euclidean distance of data point $\mathbf{x}_i$ from the closest prototype $\mathbf{w}^-$ having a different class label. The term $d^+ - d^-$ constitutes the hypothesis margin of an LVQ classifier according to the WTA, and that is why GLVQ is characterized as a margin optimization learning algorithm.

The following learning rules for GLVQ are obtained using stochastic gradient descent(SGD) on Equation 4.9:

$$\begin{aligned} \mathbf{w}^+(t+1) &= \mathbf{w}^+(t) + 2 \cdot \alpha \cdot \phi'(\mu(\mathbf{x}_i)) \cdot \mu^+ \cdot (\mathbf{x}_i - \mathbf{w}^+), \\ \mathbf{w}^-(t+1) &= \mathbf{w}^-(t) - 2 \cdot \alpha \cdot \phi'(\mu(\mathbf{x}_i)) \cdot \mu^- \cdot (\mathbf{x}_i - \mathbf{w}^-) \end{aligned} \tag{4.10}$$

where, $\mu^+ = \frac{2d^-}{(d^-+d^+)^2}$, $\mu^- = \frac{2d^+}{(d^-+d^+)^2}$ and $\alpha \in (0,1)$ is the learning rate.

Other than GLVQ, there are other successful implementations of LVQ in the literature that each is different from GLVQ either in cost function or in distance measure [25, 87]. A review of these models is given in [70].

4.3 Our model for object classification

In this section we propose a model for object classification that uses spike trains generated in the MT area of the visual cortex using CARLsim library. The framework that we use is depicted in Figure 4.1.

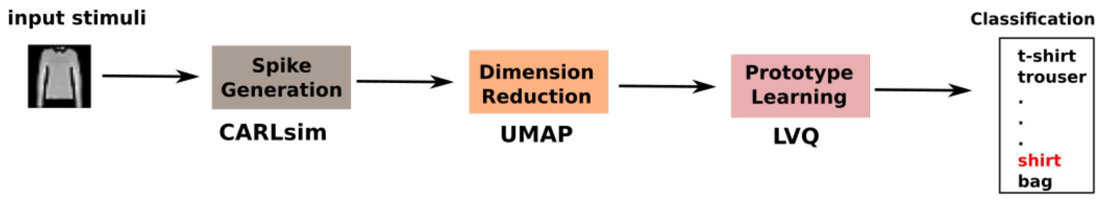


Figure 4.1: The framework that we used for object classification on Fashion-MNIST dataset. We send the images to CARLsim to generate spike trains simulated by neurons in V1 and MT area. The binary files then are fed to UMAP algorithm to reduce the dimension. Then using these new low dimensional vectors we produce prototype using supervised LVQ method. These prototypes later are used for classification using some similarity measure.

The first step in this framework is like the saliency detection model introduced in chapter 3 except that here we only consider the response of neurons located in area MT that their receptive fields are sensitive to bars and edges oriented in eight different orientations. So for each input stimuli we will have a response vector of size $M \times N \times 8$ where M and N are the size of the input stimuli. Thus the output will fall in a very high dimensional vector space. This is the point that we use the non-linear dimensionality reduction technique UMAP that we introduced in section 4.2. We project the space of all eight responses to a 3-dimensional embedding and so reduce the dimension to three in total which is a significant reduction. And finally we employ the LVQ method (see section 4.2.3) to classify the objects.

In next subsections we explain the details of our model for object classification and also statistical analysis regarding its performance. The pipeline that we used is shown in the block-diagram of Figure 4.1. We test our model on the Fashion-MNIST dataset that is explained in the following.

- **The dataset(Fashion-MNIST):** Fashion-MNIST is a dataset of Zalando's article [103] images, consisting of a training set of 60,000 examples and a test set of 10,000 examples. But instead of images of 10 hand-written digits, Fashion-MNIST contains images of 10 cloth categories, namely, *T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag,* and *Ankle boot.* Each example is a 28x28 grayscale image, associated with a label from 10 classes. Figure 4.2 shows an example of how the data looks (each class takes two-rows).The authors intended for Fashion-MNIST to serve as a direct drop-in replacement for the original MNIST dataset for benchmarking machine learning algorithms. It shares the same image size and structure of training and testing splits as original MNIST.

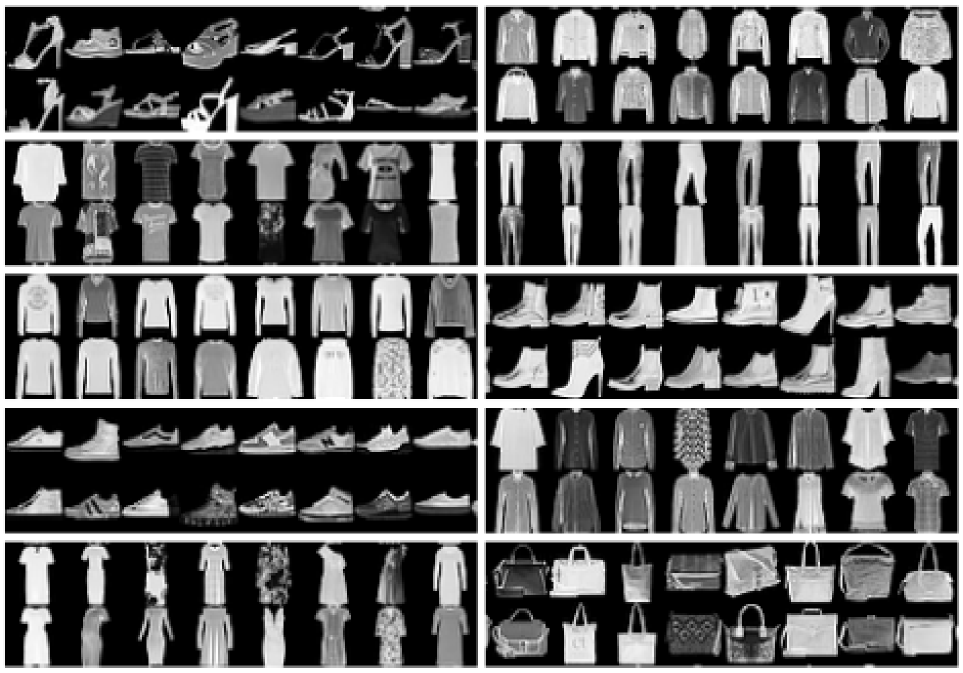


Figure 4.2: Samples of 10 classes of Fashion-MNIST dataset

- **Response of MT area in CARLsim:** Similar to our model for saliency detection, we first pass the images to CARLsim. However, unlike saliency detection models, we are only interested in the response of neurons located in the MT area. This is consistent with most

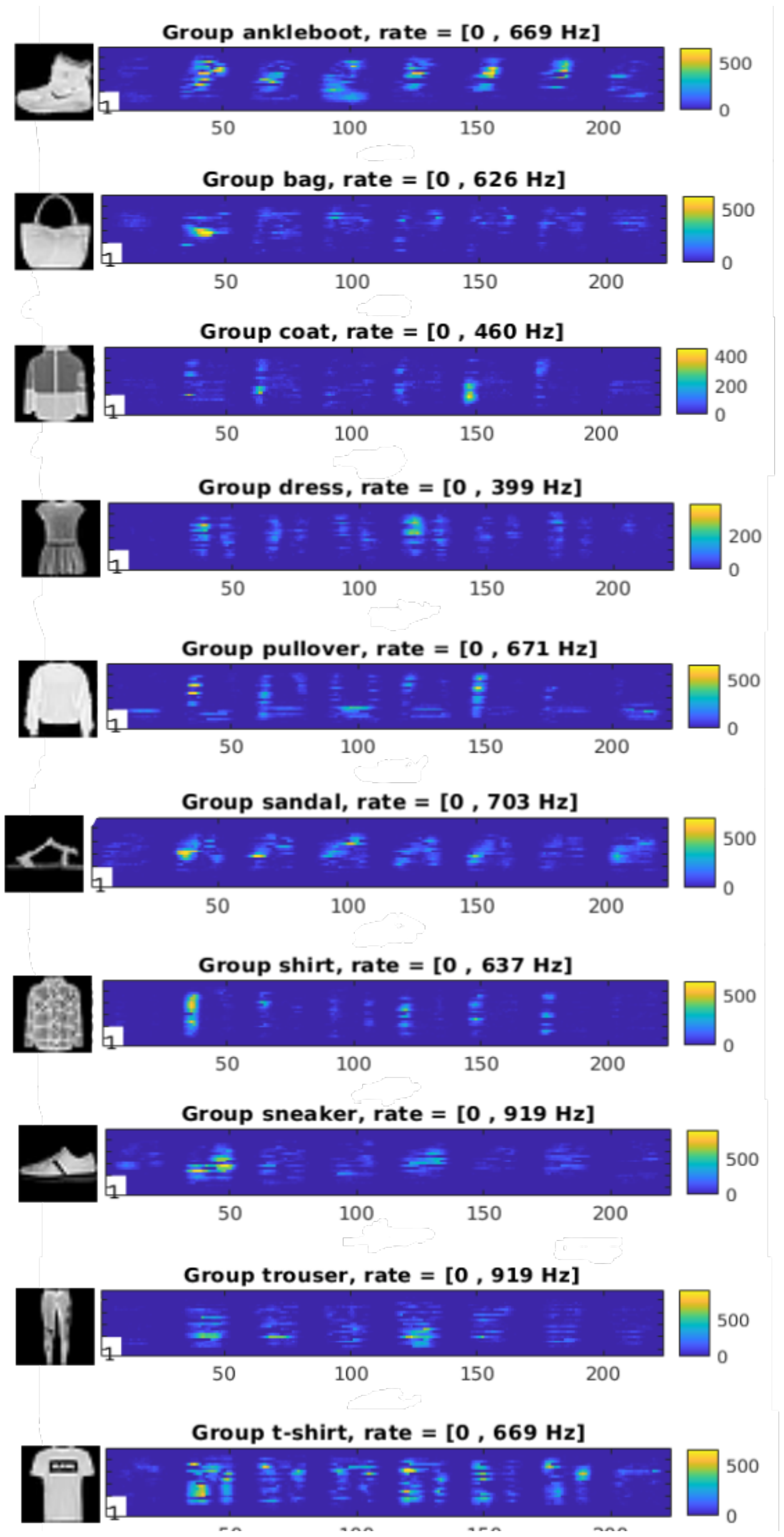
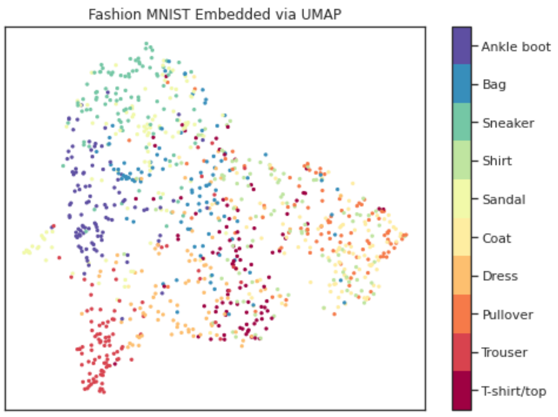


Figure 4.3: Spike trains generated in MT area using CARLsim simulator. Each row corresponds to its a gray scale image from fashion-mnist dataset shown next to it. Each image is 28×28 pixels and responses are calculated for neurons selective to one of 8 orientations.

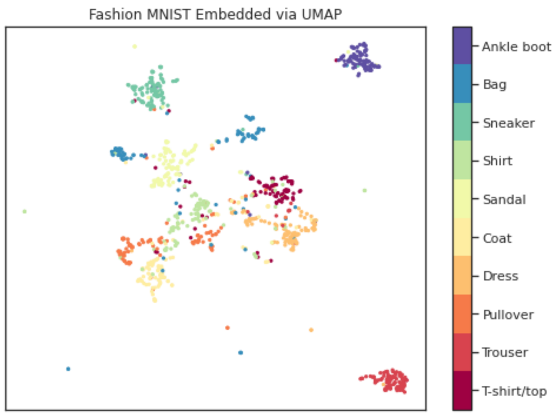
object detection/classification algorithms that the main role is played by oriented bars as visual features. Every pixel in input images corresponds to a neuron in the visual cortex area MT and for every 28×28 grayscale image, we will have eight response images of the same size as the original image (see Figure 4.3). Every pixel location in these response images represent the number of times that the corresponding neuron fired during simulation time (2 seconds in our case). We treat these responses as the visual features for further processing. But for using them in next steps of the model we flatten these 28×28 images, hence the response corresponding to each image would be a vector of size $28 \times 28 \times 8 = 6272$ which is a very high dimensional vector to work with. Thus in the next step we employ UMAP to reduce the dimension of these vectors.

- **Reducing the dimension of generated spikes:** In this stage of the model we use UMAP to reduce the dimension of the response vector. For each response we calculate a 3-dimensional embedding, so we replace a 6272 dimensional vector with a 3-dimensional one. The reason for choosing 3 as the dimension of embeddings is mainly ad hoc. We obtained the best result with three dimensions. UMAP is capable of both supervised and un-supervised dimension reduction. Since our final goal for these embeddings is to serve as input to LVQ for prototype learning, it is important that these lower dimensional projections retains the most properties of the original high dimensional data. Thus we train UMAP using labels. These 2-dimensional embeddings are depicted in Figure 4.4 for both supervised and un-supervised case over Fasion-MNIST dataset. The number of training set used to train UMAP was 1000 in total, or 100 for each category. Also you can see the plot for 3D embeddings that we have used for prototype learning in LVQ in Figure 4.5.

  It is obvious that supervised UMAP can separate the classes much better than unsupervised but even in unsupervised cases the clusters are discernible and this is despite the huge reduction in dimension. another interesting point is that while the responses in MT area do not look like they follow a pattern or even have any meaning in many cases (see Figure 4.3), the fact that UMAP is able to separate them is an evidence that the sparse coding in the brain is a really effective way to remove redundancy while keeping the most important information



(a) un-supervised



(b) supervised

Figure 4.4: 2D embeddings of high dimensional spike trains similar to Figure 4.3 generated in CARLsim, using both supervised and un-supervised UMAP

intact.

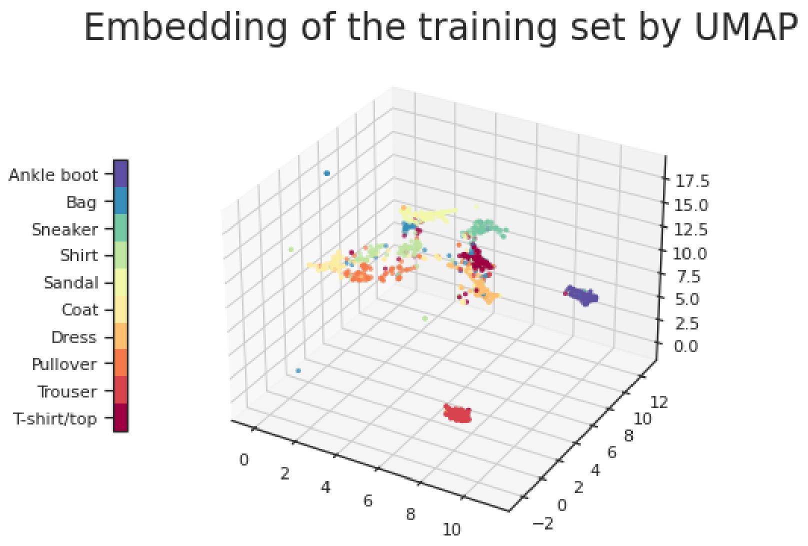


Figure 4.5: 3D embeddings of high dimensional spike trains similar to Figure 4.3 generated in CARLsim. These are the embeddings that we used to train LVQ for prototype learning

- **Training LVQ using 3D embeddings** The last step in our object classification pipeline is to train the LVQ over 3D embeddings generated by UMAP to produce prototypes. We used the GLVQ version of LVQ for this purpose. Before sending data to GLVQ we also make sure to transform them to z-score space. For training, we used `squared-euclidean` as distance measure, the activation function was `swish`, $\beta = 2$ and the solver was `steepest-gradient-descent` with step size equal to 0.1. The matrix P is an example of a prototype learned for Fashion-MNIST dataset with 100 sample for each class.

$$P = \begin{bmatrix} \text{T-shirt} & 6.94547305 & 2.94324863 & 6.2745214 \\ \text{Trouser} & 10.78699397 & 3.0205616 & 12.13141956 \\ \text{Pullover} & 2.13637348 & 3.86036774 & 7.19052154 \\ \text{Dress} & 5.70099298 & 1.66877899 & 9.22073466 \\ \text{Coat} & 4.29084838 & 9.19994977 & 6.72747327 \\ \text{Sandal} & 2.75782748 & 4.18813301 & 8.67307002 \\ \text{Shirt} & 2.12428563 & 9.19854548 & 9.19886386 \\ \text{Sneaker} & 2.75782748 & 4.18813301 & 8.67307002 \\ \text{Bag} & 2.62066067 & 6.95872302 & 8.87659434 \\ \text{Ankleboot} & 7.02562854 & 10.10105448 & 7.0030404 \end{bmatrix}$$

Each row in this matrix corresponds to a prototype vector representing one of 10 classes. Having these prototype, classifying the new data is just the matter of comparing them with these prototypes based on some similarity/dis-similarity measure.

Table 4.1 shows precision, recall and f-score of GLVQ on training samples including 1000 images, 100 for each class.

class	precision	recall	f1-score
T-shirt	0.78	0.81	0.80
Trouser	0.98	0.92	0.95
Pullover	0.86	0.75	0.80
Dress	0.75	0.79	0.77
Coat	0.82	0.69	0.75
Sandal	0.86	0.90	0.88
Shirt	0.70	0.75	0.72
Sneaker	0.89	0.97	0.93
Bag	0.73	0.77	0.75
Ankle boot	0.95	0.94	0.94

Table 4.1: Performance evaluation of GLVQ over 200 test samples of Fashion-MNIST dataset.

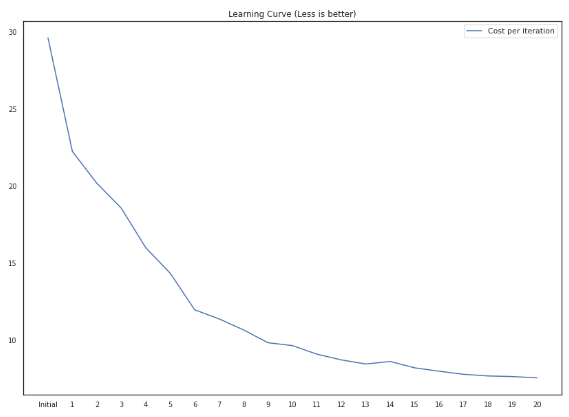
To test our results further, we randomly selected 200 samples from 10000 images from test data, 20 samples from each category and tested the performance of our framework for three levels

	1% data			2% data			3% data		
	Precision	Recall	F-score	Precision	Recall	F-score	Precision	Recall	F-score
T-shirt	0.22	0.1	0.14	0.43	0.45	0.44	0.46	0.47	0.46
Trouser	0.88	0.7	0.78	0.73	0.80	0.76	0.78	0.81	0.79
Pullover	0.20	0.30	0.24	0.43	0.15	0.22	0.42	0.25	0.31
Dress	0.34	0.50	0.41	0.29	0.35	0.32	0.32	0.52	0.40
Coat	0.25	0.05	0.08	0.37	0.70	0.48	0.38	0.72	0.50
Sandal	0.54	0.35	0.42	0.47	0.45	0.46	0.55	0.48	0.51
Shirt	0.24	0.40	0.30	0.22	0.10	0.14	0.25	0.40	0.31
Sneaker	0.63	0.60	0.62	0.52	0.55	0.54	0.60	0.55	0.57
Bag	0.57	0.65	0.60	0.55	0.55	0.55	0.60	0.67	0.63
Ankle boot	0.57	0.65	0.60	0.68	0.65	0.67	0.71	0.69	0.70
Accuracy			0.43			0.48			0.52
Average	0.44	0.43	0.42	0.47	0.47	0.46	0.51	0.56	0.52

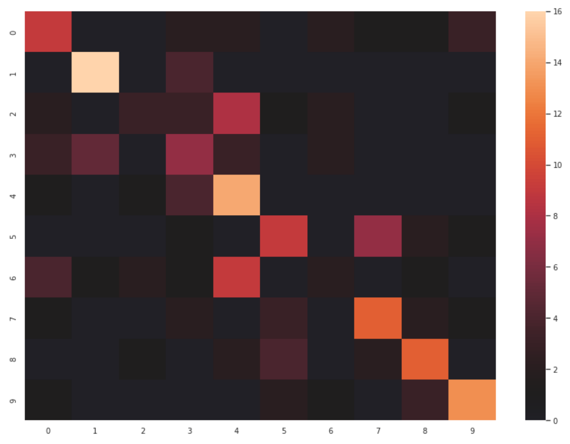
Table 4.2: Result of our model over 200 samples from Fashion-MNIST dataset. The model was trained for 1%, 2%, and 3% of train set. Precision, Recall, F-score, and accuracy are reported and show improving performance by increasing the size of training set.

of training on 1%, 2%, and 3% of train data. The results are shown in Table 4.2. It can be seen that the best performance is for 3% of data and 2% is performing better than 1%. While this cannot prove that our model can perform necessarily better with adding more data in the training phase seeing this trend is promising for any future extension.

The learning curve as well as the confusion matrix are shown in Figure 4.6. Looking at the confusion matrix it is clear that the most energy is on main diagonal. Although it is far from perfect but we believe that this result is still promising given the small training set (only 3% of whole training set) that we used. The majority of wrong detections belong to classes that are very hard to classify even for a human, like shirt from T-shirt or Pullover. This is also verified in Figure 4.7 that shows the Receiver Operating Characteristic (ROC) curve for individual classes as well as the macro-average curve. Here, macro-average curve simply represent the average performance for the model when considering all classes with the same statistics. We are representing macro-average (in contrast to micro-average) because our classes have balance in the number of elements.



(a) confusion matrix



(b) learning curve

Figure 4.6: Left: confusion matrix that shows the higher energy on main diagonal. Right: the curve of learning rate of LVQ over 3D embeddings in 20 epochs.

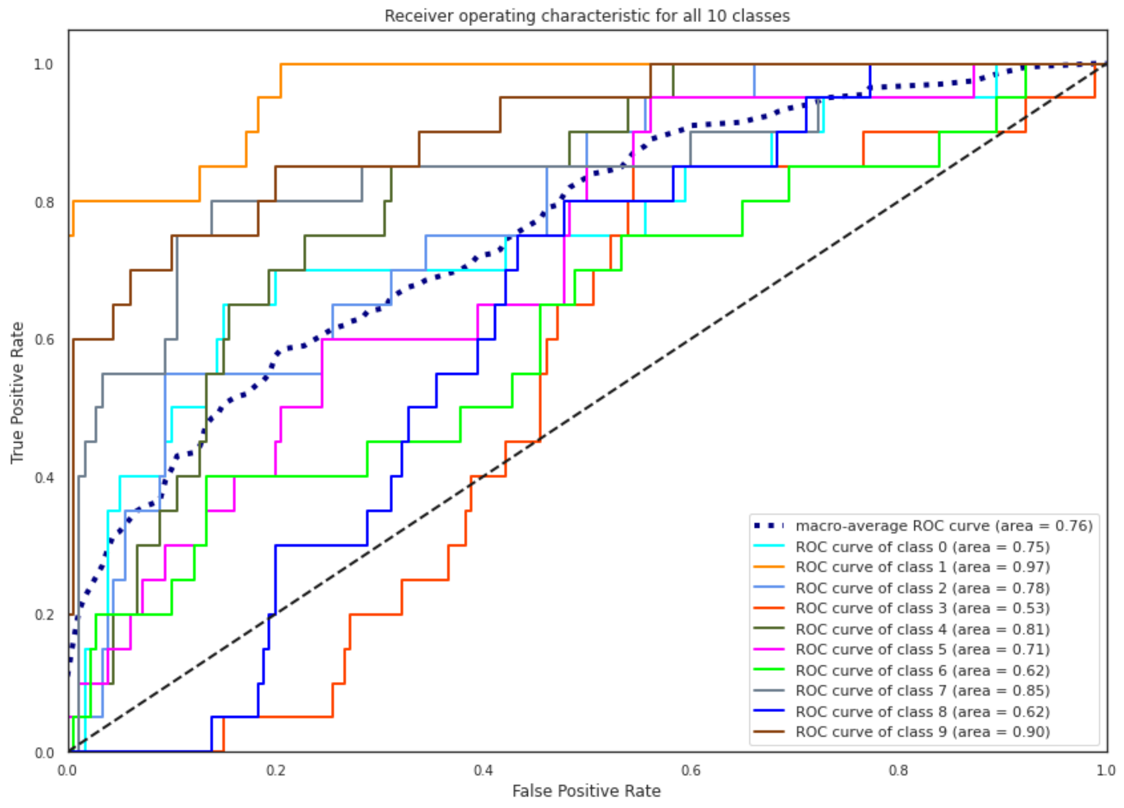


Figure 4.7: The ROC curve for 10 individual classes. The worst performance is over class labeled 3(dress) and class labeled 8(bag) with 0.53 and 0.62 respectively. The area under macro-average curve is 0.76.

Chapter 5

Conclusion and future works

5.1 Conclusion

As pointed out previously in the abstract and introduction, the main goal of this thesis was to investigate the usefulness of some of the most popular computational models for human visual systems in order to employ them for computer vision and machine learning applications. We mainly were interested in the known ability of brain in efficiently analyzing and perceiving the complex visual scenes despite its limited resources.

It is believed that our brain addresses this issue by coding the visual scene sparsely. So it only samples parts of incoming data that carry the maximum information in order to decrease the redundancy.

In order to simulate this behavior, we used available models for color and orientation coding by V1 area in the primate visual cortex first and then reduced the results significantly by simulating the spiking neural network of the human visual cortex. This way we generated spike trains that are really sparse(almost 90% of pixels for each image are 0). However, these responses are able to retain and carry much of the useful information.

We tested these results in two different case studies, namely, saliency detection and object classification. In both of these computer vision tasks our frameworks showed reasonable results. Although, the performance is not competitive compared to some modern and more sophisticated existing algorithms, their simplicity and speed proves one more time that nature could be a great source of inspiration for computer vision and machine learning problems. For example, it only takes approximately 5 seconds to generate 3-dimensional embeddings using UMAP for 1000 images from Fashion-MNIST data set and another 6 seconds to train the GLVQ. All of these parts are done in Python using scikit-learn library [74] and in a HP ProBook 470 G4.

5.2 Future work

There is still room for improvements and extensions to our model. As mentioned earlier, one problem with our proposed pipeline is that it is not fully integrated. An improvement would be to redesign the framework in an input-output style. This would provide the ability to feed the model an image directly and receive the output without the need to use separate coding and libraries for steps in the pipeline. For this to be time efficient it will be necessary to have high performance graphical processing units.

Another improvement would be to prepare additional training data and extend the training phase to include more than just 3% of the data set. This would allow the ability to see where the improvement trend tapers off in the recognition task.

Finally, for the object classification part we used only gray scale images and relied on the responses of the MT area of the visual cortex alone for orientation coding. It would be of interest to take the same approach for color images and employ the color responses of the V4 area of the visual cortex.

Bibliography

[1] Radhakrishna Achanta, Sheila Hemami, Francisco Estrada, and Sabine Süsstrunk. Frequency-tuned Salient Region Detection.

[2] Saleh Aly. Learning invariant local image descriptor using convolutional Mahalanobis self-organising map. *Neurocomputing*, 142:239–247, 2014.

[3] Adarsha Balaji, Prathyusha Adiraju, Hirak J. Kashyap, Anup Das, Jeffrey L. Krichmar, Nikil D. Dutt, and Francky Catthoor. PyCARL: A PyNN Interface for Hardware-Software Co-Simulation of Spiking Neural Network. *Proceedings of the International Joint Conference on Neural Networks*, jul 2020.

[4] A. Bartels and S. Zeki. The architecture of the colour centre in the human visual brain: New results and a review. *European Journal of Neuroscience*, 12(1):172–193, 2000.

[5] Richard Ernest Belman. *Dynamic Programming.* Courier Corporation, 2003.

[6] Michael Beyeler, Kristofor D. Carlson, Ting Shuo Chou, Nikil Dutt, and Jeffrey L. Krichmar. CARLsim 3: A user-friendly and highly optimized library for the creation of neurobiologically detailed spiking neural networks. *Proceedings of the International Joint Conference on Neural Networks*, 2015-Septe, 2015.

[7] Michael Beyeler, Micah Richert, Nikil D. Dutt, and Jeffrey L. Krichmar. Efficient spiking neural network model of pattern motion selectivity in visual cortex. *Neuroinformatics*, 12(3):435–454, 2014.

[8] Michael Beyeler, Emily L. Rounds, Kristofor D. Carlson, Nikil Dutt, and Jeffrey L. Krichmar. Neural correlates of sparse coding and dimensionality reduction. *PLoS Computational Biology*, 15(6), jun 2019.

[9] Michael Biehl, Fthi Abadi, Christina Göpfert, and Barbara Hammer. Prototype-based classifiers in the presence of concept drift: A modelling framework. 976, mar 2019.

[10] Michael Biehl, Barbara Hammer, and Thomas Villmann. Prototype-based models in machine learning. *Wiley Interdisciplinary Reviews: Cognitive Science*, 7(2):92–111, mar 2016.

[11] Ali Borji and Laurent Itti. State-of-the-art in visual attention modeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(1):185–207, 2013.

[12] Ali Borji and Laurent Itti. CAT2000: A Large Scale Fixation Dataset for Boosting Saliency Research. pages 2–5, 2015.

[13] Richard T. Born and David C. Bradley. Structure and function of visual area MT. *Annual review of neuroscience*, 28:157–189, 2005.

[14] Pedro H.M. Braga and Hansenclever F. Bassani. A Semi-Supervised Self-Organizing Map for Clustering and Classification. *Proceedings of the International Joint Conference on Neural Networks*, 2018-July, 2018.

[15] Neil D.B. Bruce and John K. Tsotsos. Saliency based on information maximization. *Advances in Neural Information Processing Systems*, (June 2014):155–162, 2005.

[16] Zoya Bylinskii, Tilke Judd, Borji Ali, Laurent Itti, Frédo Durand, Aude Oliva, and Antonio Torralba. MIT Saliency Benchmark.

[17] Zoya Bylinskii, Tilke Judd, Aude Oliva, Antonio Torralba, and Fredo Durand. What Do Different Evaluation Metrics Tell Us about Saliency Models? *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(3):740–757, 2019.

[18] Charles Cadieu, Minjoon Kouh, Anitha Pasupathy, Charles E. Connor, Maximilian Riesenhuber, and Tomaso Poggio. A model of V4 shape selectivity and invariance. *Journal of Neurophysiology*, 98(3):1733–1750, sep 2007.

[19] Natalia Caporale and Yang Dan. Spike timing-dependent plasticity: a Hebbian learning rule. *Annual review of neuroscience*, 31:25–46, 2008.

[20] Matteo Carandini. What simple and complex cells compute. *The Journal of Physiology*, 577(Pt 2):463, dec 2006.

[21] Kristofor D. Carlson, Jayram Moorkanikara Nageswaran, Nikil Dutt, and Jeffrey L. Krichmar. An efficient automated parameter tuning framework for spiking neural networks. *Frontiers in Neuroscience*, 8(8 FEB):1–15, 2014.

[22] Ting Shuo Chou, Hirak J. Kashyap, Jinwei Xing, Stanislav Listopad, Emily L. Rounds, Michael Beyeler, Nikil Dutt, and Jeffrey L. Krichmar. CARLsim 4: An Open Source Library for Large Scale, Biologically Detailed Spiking Neural Network Simulation using Heterogeneous Clusters. *Proceedings of the International Joint Conference on Neural Networks*, 2018-July:1158–1165, 2018.

[23] Peter Dayan and L. F. Abbott. *Theoretical Neuroscience.* The MIT Press.

[24] Mark De Berg, Otfried Cheong, Mark Van Kreveld, and Mark Overmars. *Computational Geometry, Algorithms and Applications.* Springer, 2008.

[25] Virginia R de Sa, Dana H Ballard, David Nova, Pablo A. Estévez, Thomas Villmann, Andrea Bohnsack, Marika Kaden, Barbara Hammer, Thomas Villmann, Petra Schneider, Michael Biehl, Barbara Hammer, Kerstin Bunte, Petra Schneider, Barbara Hammer, Frank Michael Schleif, Thomas Villmann, Michael Biehl, Yuan Yuan Shen, Yan Ming Zhang, Xu Yao Zhang, Cheng Lin Liu, Petra Schneider, Kerstin Bunte, Han Stiekema, Barbara Hammer, Thomas Villmann, and Michael Biehl. Regularization in matrix relevance learning. *Neural Networks*, 21(8-9):831–840, 2012.

[26] Yang Du, Chunfeng Yuan, Bing Li, Weiming Hu, and Stephen Maybank. Spatio-temporal self-organizing map deep network for dynamic object detection from videos. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017-Janua:4245–4254, 2017.

[27] Richard O. Duda, Peter E. Hart, and David G Stork. *Pattern Classification.* Wiley, 2nd edition, 2000.

[28] Sergio Etchebehere and Elena Fedorovskaya. On the role of color in visual saliency. *IS and T International Symposium on Electronic Imaging Science and Technology*, (February):58–63, 2017.

[29] Florent Forest, Mustapha Lebbah, and Hanene Azzag. Deep Embedded SOM: Joint Representation Learning and Self-Organization.

[30] P. Frisby, John and V. Stone, James. *Seeing: The Computational Approach to Biological Vision.* The MIT Press, second edition, 2010.

[31] Antón Garcia-Diaz, Xosé R. Fdez-Vidal, Xosé M. Pardo, and Raquel Dosil. Saliency Based on Decorrelation and Distinctiveness of Local Responses. pages 261–268, 2009.

[32] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* MIT Press, 2016.

[33] Richard Hankins, Yao Peng, and Hujun Yin. SOMNet: Unsupervised Feature Learning Networks for Image Classification. *Proceedings of the International Joint Conference on Neural Networks*, 2018-July, 2018.

[34] Donald Hebb. *The organization of behavior, A neurophysiological theory.* Psychology Press, 1949.

[35] David Heeger. Poisson Model of Spike Generation. *Handout*, pages 1–13, 2000.

[36] C. A. Heywood, A. Gadotti, and A. Cowey. Cortical area V4 and its role in the perception of color. *Journal of Neuroscience*, 12(10):4056–4065, 1992.

[37] D . H . Hubel and T . N . Wiesel. Receptive Fields of Single Neurons In The Cat ' s Striate Cortex. pages 574–591, 1959.

[38] D. H. Hubel and T. N. Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of Physiology*, 160(1):106–154, 1962.

[39] Laurent Itti, Christof Koch, and Ernst Niebur. Short Papers. *Journal of Tissue Viability*, 8(2):24, 1998.

[40] Eugene M. Izhikevich. Simple model of spiking neurons. *IEEE Transactions on Neural Networks*, 14(6):1569–1572, 2003.

[41] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. An Introduction to Statistical Learning. 2021.

[42] Youki Kamiya, Toshiaki Ishii, Shen Furao, and Osamu Hasegawa. An online semi-supervised clustering algorithm based on a self-organizing incremental neural network. *IEEE International Conference on Neural Networks - Conference Proceedings*, pages 1061–1066, 2007.

[43] Nikola K. Kasabov. Time-Space, Spiking Neural Networks and Brain-Inspired Artificial Intelligence. 7, 2019.

[44] C. Koch and S. Ullman. Shifts in selective visual attention: Towards the underlying neural circuitry. *Human Neurobiology*, 4(4):219–227, 1985.

[45] Teuvo Kohonen. Improved versions of learning vector quantization. pages 545–550, 1990.

[46] Teuvo Kohonen. Learning Vector Quantisation and the Self Organising Map. pages 235–242, 1992.

[47] Teuvo Kohonen. Learning Vector Quantization. pages 175–189, 1995.

[48] Teuvo Kohonen. Essentials of the self-organizing map. *Neural Networks*, 37:52–65, 2013.

[49] Tuevo Kohonen, Erkki Oja, Olli Simula, Ari Visa, and Jari Kangas. Engineering Applications of the Self-Organizing Map. *Proceeding of the ieee*, 84(10), 1996.

[50] Timo Kostiainen, Timo Kostiainen, and Jouko Lampinen. On the generative probability density model in the self-organizing map, Neurocomputing 48. *NEUROCOMPUTING*, 48:217—-228, 2002.

[51] Matthias Kümmerer, Thomas S.A. Wallis, and Matthias Bethge. Information-theoretic model comparison unifies saliency metrics. *Proceedings of the National Academy of Sciences of the United States of America*, 112(52):16054–16059, 2015.

[52] Itti L and Koch C. Computational modelling of visual attention. *Nature Reviews Neuroscience*, 2(3):194–203, 2001.

[53] Jorma T. Laaksonen, J. Markus Koskela, and Erkki Oja. Class distributions on SOM surfaces for feature extraction and object retrieval. *Neural Networks*, 17(8-9):1121–1133, 2004.

[54] Jouko Lampinen and Senior Member. Distortion Tolerant Pattern Recognition Based on Self-organizing Feature Extraction. 6(3), 1995.

[55] Jouko Lampinen and Seppo Smolander. Self-organizing feature extraction in recognition of wood surface defects and color images. *International Journal of Pattern Recognition and Artificial Intelligence*, 10(2):97–113, 1996.

[56] Olivier Le Meur, Patrick Le Callet, Dominique Barba, and Dominique Thoreau. A coherent computational approach to model bottom-up visual attention. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(5):802–817, 2006.

[57] Nan Liu, Jinjun Wang, and Yihong Gong. Deep Self-Organizing Map for visual classification. *Proceedings of the International Joint Conference on Neural Networks*, 2015-Septe, 2015.

[58] Jukka Livarinen and Ari J. E. Visa. Unsupervised image segmentation with the self-organizing map and statistical methods. *Intelligent Robots and Computer Vision XVII: Algorithms, Techniques, and Active Vision*, 3522(October 1998):516–526, 1998.

[59] M. S. Livingstone and D. H. Hubel. Anatomy and physiology of a color system in the primate visual cortex. *Journal of Neuroscience*, 4(1):309–356, 1984.

[60] Wolfgang Maass. Networks of spiking neurons: The third generation of neural network models. *Neural Networks*, 10(9):1659–1671, dec 1997.

[61] Laura Manduchi, Matthias Hüser, Julia Vogt, Gunnar Rätsch, and Vincent Fortuin. DPSOM: Deep Probabilistic Clustering with Self-Organizing Maps. oct 2019.

[62] D. Marr and E. Hildreth. Theory of edge detection. *Proceedings of the Royal Society of London - Biological Sciences*, 207(1167):187–217, 1980.

[63] David Marr. *Vision A Computational Investigation into the Human Representation and Processing of Visual Information.* The MIT Press, 1982.

[64] J. H.R. Maunsell and D. C. Van Essen. The connections of the middle temporal visual area (MT) and their relationship to a cortical hierarchy in the macaque monkey. *The Journal of neuroscience : the official journal of the Society for Neuroscience*, 3(12):2563–2586, 1983.

[65] Leland McInnes, John Healy, and James Melville. UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. 2018.

[66] Luke Melas-Kyriazi. The Mathematical Foundations of Manifold Learning. 2020.

[67] J. A. Movshon, I. D. Thompson, and D. J. Tolhurst. Receptive field organization of complex cells in the cat's striate cortex. *The Journal of Physiology*, 283(1):79, oct 1978.

[68] J. A. Movshon, I. D. Thompson, and D. J. Tolhurst. Spatial summation in the receptive fields of simple cells in the cat's striate cortex. *The Journal of Physiology*, 283(1):53, oct 1978.

[69] Jayram Moorkanikara Nageswaran, Nikil Dutt, Jeffrey L. Krichmar, Alex Nicolau, and Alexander V. Veidenbaum. A configurable simulation environment for the efficient simulation of large-scale spiking neural networks on graphics processors. *Neural Networks*, 22(5-6):791–800, 2009.

[70] David Nova and Pablo A. Estévez. A review of learning vector quantization classifiers. *Neural Computing and Applications*, 25(3-4):511–524, 2014.

[71] Christian O'Connell, Andrea Kutics, and Akihiko Nakagawa. Layered self-organizing map for image classification in unrestricted domains. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8156 LNCS(PART 1):310–319, 2013.

[72] S. H. Ong, N. C. Yeo, K. H. Lee, Y. V. Venkatesh, and D. M. Cao. Segmentation of color images using a two-stage self-organizing network. *Image and Vision Computing*, 20(4):279–289, 2002.

[73] Giuseppe Papari and Nicolai Petkov. Edge and line oriented contour detection: State of the art. *Image and Vision Computing*, 29(2-3):79–103, feb 2011.

[74] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825—-2830, 2011.

[75] Nicolas Pinto, David D. Cox, and James J. DiCarlo. Why is Real-World Visual Object Recognition Hard? *PLOS Computational Biology*, 4(1):e27, jan 2008.

[76] Ludovic Platon, Farida Zehraoui, and Fariza Tahi. Self-Organizing Maps with supervised layer. *12th International Workshop on Self-Organizing Maps and Learning Vector Quantization, Clustering and Data Visualization, WSOM 2017 - Proceedings*, pages 1–8, 2017.

[77] Tomaso Poggio and Maximilian Riesenhuber. Hierarchical models of object recognition in cortex. *Nature Neuroscience*, 2(11):1019–1025, 1999.

[78] Tomaso Poggio and Shimon Ullman. Visual Cortex Models for Object Recognition. *Computer Vision*, pages 862–866, 2014.

[79] David F. Ramirez-Moreno, Odelia Schwartz, and Juan F. Ramirez-Villegas. A saliency-based bottom-up visual attention model for dynamic scenes analysis. *Biological Cybernetics*, 107(2):141–160, 2013.

[80] Nicolas Riche, Matei Mancas, Bernard Gosselin, and Thierry Dutoit. Rare: A new bottom-up saliency model. *Proceedings - International Conference on Image Processing, ICIP*, (May 2014):641–644, 2012.

[81] Micah Richert, Jayram Moorkanikara Nageswaran, Nikil Dutt, and Jeffrey L. Krichmar. An Efficient Simulation Environment for Modeling Large-Scale Cortical Processing. *Frontiers in Neuroinformatics*, 5, sep 2011.

[82] Felix M. Riese, Sina Keller, and Stefan Hinz. Supervised and semi-supervised self-organizing maps for regression and classification focusing on hyperspectral data. *Remote Sensing*, 12(1), 2020.

[83] R W Rodieck. *The First Steps in Seeing.* Sinauer Associates, Inc., Sunderland, MA, 1998.

[84] Anna W. Roe, Leonardo Chelazzi, Charles E. Connor, Bevil R. Conway, Ichiro Fujita, Jack L. Gallant, Haidong Lu, and Wim Vanduffel. Toward a Unified Theory of Visual Area V4. *Neuron*, 74(1):12–29, apr 2012.

[85] Nicole C. Rust, Valerio Mante, Eero P. Simoncelli, and J. Anthony Movshon. How MT cells analyze the motion of visual patterns. *Nature Neuroscience*, 9(11):1421–1431, 2006.

[86] Atsushi Sato and Keiji Yamada. Generalized Learning Vector Quantization. *Proceedings of the 8th International Conference on Neural Information Processing Systems*, pages 423–429, 1996.

[87] Petra Schneider, Kerstin Bunte, Han Stiekema, Barbara Hammer, Thomas Villmann, and Michael Biehl. Regularization in matrix relevance learning. *IEEE Transactions on Neural Networks*, 21(5):831–840, 2010.

[88] Thomas Serre, Minjoon Kouh, Charles Cadieu, Ulf Knoblich, Gabriel Kreiman, and Tomaso Poggio. A theory of object recognition : computations and circuits in the feedforward path of the ventral stream in primate visual cortex This version replaces the preliminary " Halloween " CBCL paper from Nov . 2005 . *Artificial Intelligence*, (December):1–130, 2005.

[89] Thomas Serre, Lior Wolf, and Tomaso Poggio. Object recognition with features inspired by visual cortex. *Proceedings - 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2005*, II:994–1000, 2005.

[90] Robert Shapely and Michael J. Hawken. Color in the Cortex: single- and double-opponent cells — Elsevier Enhanced Reader. *Vision Research*, 51:701–717, 2011.

[91] R. M. Shapley and D. J. Tolhurst. Edge detectors in human vision. *The Journal of Physiology*, 229(1):165, feb 1973.

[92] Xiaohui Shen and Ying Wu. A unified approach to salient object detection via low rank matrix recovery. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 853–860, 2012.

[93] Eero P Simoncelli and David J Heeger. A Model of Neural Responses in Area MT. *Vision research*, 38(5):743–761, 1998.

[94] Athira Sivadas and Kendal Broadie. How Does My Brain Communicate With My Body? *Frontiers for Young Minds*, 8, oct 2020.

[95] Victor Skuratov, Konstantin Kuzmin, Igor Nelin, and Mikhail Sedankin. Application of Kohonen self-organizing map to search for region of interest in the detection of objects. *EUREKA, Physics and Engineering*, 2020(1):62–69, 2020.

[96] Xiaoyang Tan, Songcan Chen, Zhi Hua Zhou, and Fuyan Zhang. Recognizing partially occluded, expression variant faces from single training image per person with SOM and soft κ-NN ensemble. *IEEE Transactions on Neural Networks*, 16(4):875–886, jul 2005.

[97] Amirhossein Tavanaei, Masoud Ghodrati, Saeed Reza Kheradpisheh, Timothée Masquelier, and Anthony Maida. Deep learning in spiking neural networks. *Neural Networks*, 111:47–63, 2019.

[98] P. J. Toivanen, J. Ansamäki, J. P.S. Parkkinen, and J. Mielikäinen. Edge detection in multispectral images using the self-organizing map. *Pattern Recognition Letters*, 24(16):2987–2994, 2003.

[99] James M.G. Tsui, J. Nicholas Hunter, Richard T. Born, and Christopher C. Pack. The role of V1 surround suppression in MT motion integration. *Journal of Neurophysiology*, 103(6):3123–3138, jun 2010.

[100] Leslie G. Ungerleider and Luiz Pessoa. What and where pathways. *Scholarpedia*, 3(11):5342, nov 2008.

[101] Thomas Villmann, Andrea Bohnsack, and Marika Kaden. Can learning vector quantization be an alternative to SVM and deep learning? - Recent trends and advanced variants of learning vector quantization for classification learning. *Journal of Artificial Intelligence and Soft Computing Research*, 7(1):65–81, 2017.

[102] Chathurika S. Wickramasinghe, Kasun Amarasinghe, and Milos Manic. Deep Self-Organizing Maps for Unsupervised Image Classification. *IEEE Transactions on Industrial Informatics*, 15(11):5837–5845, 2019.

[103] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. 2017.

[104] Parvin Zarei Eskikand, Tatiana Kameneva, Anthony N. Burkitt, David B. Grayden, and Michael R. Ibbotson. Pattern motion processing by MT neurons. *Frontiers in Neural Circuits*, 13, may 2019.

[105] Zhenjie Zhao, Xuebo Zhang, and Yongchun Fang. Stacked multilayer self-organizing map for background modeling. *IEEE Transactions on Image Processing*, 24(9):2841–2850, 2015.

APPENDICES

A Computational model for orientation detection in visual cortex

In this appendix section we discuss the computational model that we leveraged to calculate the response of orientation selective cells in visual cortex. The model was first proposed by Simoncelli and Heeger in [93] to model the receptive fields of neurons in area V1 and MT that are known to be tunned for detecting the direction of motion. This model later was re-implemented in [7] using C++/CUDA. Here we discuss the details of both implementations and we also explain the minor changes that we made to fit it to our use.

Assume that $I(x, y, t)$ is the light intensity of a visual stimulus projected on the retina, x, y are spatial dimension and t represents temporal dimension. We also can characterize visual stimulus by its local contrast:

$$A(x, y, t) = \left[I(x, y, t) - \bar{I}\right] / \bar{I} \tag{A.1}$$

where, $\bar{I}$ is the average light intensity over space and time. The linear response of the n-th simple cell is then a weighted sum of the local contrast:

$$L_n(t) = \int\int\int s_n(x, y, T)\ A(x, y, T)\ dx\ dy\ dT + \alpha_1 \tag{A.2}$$

where, α_1 is a constant and $s_n(x, y, T)$, are a set of directional third derivatives of a Gausian, with 28 different space-time orientations, replicated at all spatial locations. These functions are very similar to Gabor functions but they are computationally more efficient as they are separable. The response of the n-th simple cell, $s_n(t)$ is expressed as:

$$S(t) = \frac{K_1 \lfloor L_n(t) \rfloor^2}{\sum_m \lfloor L_m(t) \rfloor^2 + \sigma_1^2} \tag{A.3}$$

where, σ_1 is the semi-saturation constant of normalization, K_1 determines the maximum attainable response, and $\lfloor \cdot \rfloor^2$ denotes the half-squaring operation $\lfloor L_n(t) \rfloor^2 \equiv \max\{0, L(t)\}$ V1 complex cell responses are computed as local averages of simple cell responses:

$$C_n(t) = \sum_m c_{mn} s_m(t) \tag{A.4}$$

with c_{mn} weights all positive. The underlying linear velocity-selective response $Q_n(t)$ of an MT pattern cell is then:

$$Q_n(t) = \sum_m p_{nm}(t) c_m(t) + \alpha_2 \tag{A.5}$$

α_2 determines the spontaneous response level and p_{nm} are a set of weights and finally the n-th pattern MT neuron's response, $P_n(t)$ is expressed as:

$$P_n(t) = \frac{K_2 \lfloor Q_n(t) \rfloor^2}{\sum_m \lfloor Q_m(t) \rfloor^2 + \sigma_2^2} \tag{A.6}$$

here, σ_2 and K_2 have the same meaning as σ_1 and K_1 but for MT neurons. Now, we explain the newer implementation of the same model by Bayeler et al.[7]. The CUDA implementation of [7] also starts with visual stimulus intensity $I(x, y, t)$ but the stimulus is processed at three different spatio-temporal resolutions, r. The first scale, $r = 0$, is equivalent to the original image and two other scales are achieved by successively blurring the image with a Gaussian kernel. Thus:

$$\begin{aligned} I_0(x,y,t) &= I(x,y,t) \\ I_1(x,y,t) &= \exp\left(\frac{-(x^2+y^2+t^t)}{2}\right) \star I_0(x,y,t) \\ I_2(x,y,t) &= \exp\left(\frac{-(x^2+y^2+t^t)}{2}\right) \star I_1(x,y,t), \end{aligned} \tag{A.7}$$

In accordance to S&H, Beyeler et al. modeled V1 simple cells as linear space-time-oriented filters whose receptive fields are third derivatives of a Gaussian. The full set of V1 linear receptive fields consisted of 28 space-time orientations that are evenly distributed on the surface of a sphere in the spatio-temporal frequency domain. A unit vector $\hat{u_k} = (\hat{u_{k,x}}, \hat{u_{k,y}}, \hat{u_{k,t}})$ describes the k-th space-time-oriented filter in the V1 population for $k = 1, \ldots, 28$ and $\hat{u_k}$'s are parallel to filter orientation. Input stimuli in three different scales are first filtered with a Gaussian kernel of width

$\sigma_{V1simple} = 1.25$ pixels corresponding to the receptive field size of V1 simple cells:

$$f_r(x, y, t) = \exp\left(\frac{-(x^2 + y^2 + t^2)}{2\sigma^2_{v1simple}}\right) \star I_r(x, y, t) \tag{A.8}$$

this convolution also guaranties that the stimuli is smooth enough to be three times differentiable.

Given above circumstances, then, the third-order derivative in the direction of $\hat{u}_k$ will define the linear response of a simple cell at spatial location (x, y) and scale r with space-time orientation k as follows:

$$L_{kr}(x, y, t) = \alpha_{v1lin} \sum_{T=0}^{6} \left[\sum_{Y=0}^{6-T} \left[\frac{6!}{X!Y!T!} (\hat{u}_{k,x})^X (\hat{u}_{k,y})^Y (\hat{u}_{k,t})^T \frac{\partial^3 f_r(x, y, t)}{\partial x^X \partial y^Y \partial t^T}\right]\right], \tag{A.9}$$

where, $X = 6 - Y - T$, and $\alpha_{V1lin} = 6.6084$ is a scaling factor. These two sums combined yield exactly 28 summands. We can also express this equation in vector notation $L_r = \alpha_{V1lin} M b_r$ where M is 28×28 matrix and b_r is a 28×1 vector. Each element of b_r is one derivative at scale r, and each element of M is a number $\frac{6!}{X!Y!T!} (\hat{u}_{k,x})^X (\hat{u}_{k,y})^Y (\hat{u}_{k,t})^T$. Finally, the simple cell response can be derived by half-squaring the linear response and then normalizing it within a large Gaussian envelope with half-width $\sigma_{V1norm} = 3.35$ pixels:

$$S_{kr}(x, y, t) = \frac{\alpha_{filt \to rate,r} \alpha_{v1rect} L_{kr}(x, y, t)^2}{\alpha_{v1norm} \exp\left(\frac{-(x^2+y^2)}{2\sigma^2_{v1norm}}\right) \star \left(\frac{1}{28} \sum_{k=1}^{28} L_{kr}(x, y, t)^2\right) + \alpha^2_{v1semi}}, \tag{A.10}$$

The scaling factors $\alpha_{v1rect} = 1.9263$ and semi-saturation constant $\alpha_{v1semi} = 0.1$. $\alpha_{filt \to rate,r} = 15Hz$ and $\alpha_{V1norm} = 1.0$ were set simultaneously to map the unit-less filter responses at each scale r onto more meaningful mean firing rates. V1 complex cells response were computed as local weighted averages of simple cell responses:

$$C_{kr}(x, y, t) = \alpha_{v1comp} \exp\left(\frac{-(x^2 + y^2)}{2\sigma^2_{v1comp}}\right) \star S_{kr}(x, y, t), \tag{A.11}$$

here, $\sigma_{v1comp} = 1.6$ is the half-width of the Gaussian and $\alpha_{v1comp} = 0.1$ is a scaling factor. The responses $C_{kr}(x, y, t)$ are the output of CUDA implementation we used and were passed to spiking neural network simulated in CARLsim.

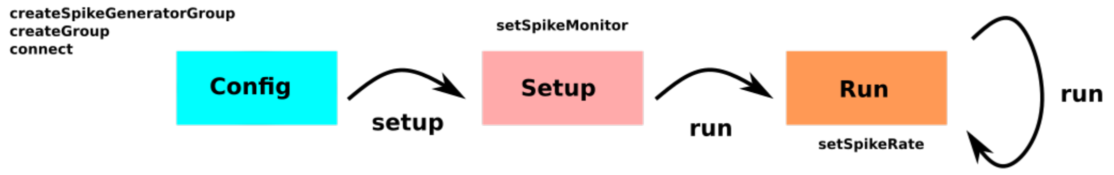


Figure 1: States of CARLsim that we used in our model. Note that this is specific to our setup. In practice CARLsim can occupy more states and setup.

In [7], MT cells respond preferentially to motion in one of eight different directions (in 45° increments) and three different speeds at any pixel location. It is done in `projectV1toMT.m` script. However, we were interested in orientation detection rather than direction of motion, so we adjusted the setting in this MATLAB script, so that MT cells respond preferentially to eight different orientations in $[0, 7\pi/8]$ (in 22.5° increments) and also, since we were working with still images we set the speed to zero.

B CARLsim and its setup in our research

CARLsim is an efficient, easy-to-use, GPU-accelerated library for simulating large-scale spiking neural network (SNN) models with a high degree of biological detail . CARLsim allows execution of networks of Izhikevich spiking neurons with realistic synaptic dynamics on both generic x86 CPUs and standard off-the-shelf GPUs. The simulator provides a programming interface (C/C++), which allows for details and parameters to be specified at the synapse, neuron, and network level [69, 81, 21, 7, 6, 22, 3].

In this thesis we used CARLsim5[1] on linux Ubuntu 20.04. A simplified workflow of CARLsim is given in Figure 1. Note that this figure is specific to our application and practically it might include more settings. A more complete figure showing more details of workflow can be found in the web address given in the footnote.

- **The config state:** CARLsim uses groups of neurons and connections as an abstraction to aid defining synaptic connectivity. We used a 2-dimensional topology of neurons as we assigned each pixel location in input stimuli to a neuron. For the $V1 \longrightarrow MT$ pathway, we

[1]https://uci-carl.github.io/CARLsim5/index.html

have two kinds of neurons: neurons located in the V1 area which are spike generators and receive their input from MT objects. These neurons are connected to neurons in MT that are regular(not spike generators) in a Gaussian connection, so basically, MT neurons sum over V1 neurons.

- **The set up state:**In this state, a number of spike monitors is set up to record generated spikes in binary files for off-line analysis. We use these binary spike files in our post processing algorithm for saliency detection or in an object classification pipeline.
- **The run state:** Once we configured and set up the network, we can run the network to start simulation. Our run state was 2 sec in this experiment. Input was generated via spike injection.

Following code snippets represent a typical implementation of above flow in CARLsim:
The following sample code creates a center-surround cell in V1 area. The size of the receptive field is the same size as the input image nrX*nrY.

```
sim.createSpikeGeneratorGroup("red-green-cells", nrX*nrY,

EXCITATORY_NEURON);
```

The following code snippet creates a group of neurons in V4 area:

```
sim.createGroup("Ev4red", nrX*nrY, EXCITATORY_NEURON);
```

The following sample code connects a neuron in V1 with a red-center green-surround receptive field to an excitatory neuron in V4 area sensitive to red hue. The connection is of Gaussian type with RF radius $(1.5, 1.5)$ which means the presynaptic neurons will connect to postsynaptic neuron if $(x_{\text{pre}} - x_{\text{post}})^2 + (y_{\text{pre}} - y_{\text{post}})^2 \leq (1.5)^2$ holds true where x, y are spatial locations of neurons. Also notice that the probability is 1.0f which means the neurons will definitely connect if the above condition is satisfied.

```
sim.connect(v1Cells[RED_GREEN], v4CellsExc[RED_v4], "gaussian",

RangeWeight(5.0f), 1.0f,RangeDelay(1), RadiusRF(1.5,1.5,0), SYN_FIXED);
```

For orientation coding we used following configuration in CARLsim:

```
int MT = sim.createGroup("MT", gridMT, EXCITATORY_NEURON);

sim.setNeuronParameters(MT, 0.02f, 0.2f, -65.0f, 8.0f);

sim.connect(V1, MT, "gaussian", RangeWeight(1.0f), 1.0f, RangeDelay(1),

RadiusRF(.5,.5,0));
```