2007

Biologically-Inspired Translation, Scale, and rotation invariant object recognition models

Myung Woo

Biologically-Inspired Translation, Scale, and Rotation Invariant Object Recognition Models

By

Myung Chul Woo

A Thesis Submitted in Fulfillment of the Requirements for the Degree of Master of Science in Computer Science

Supervised by
Dr. Roger S. Gaborski
Department of Computer Science
Golisano College of Computing and Information Sciences
Rochester Institute of Technology
Rochester, New York
May 2007

Approved By:

Date:

Dr. Roger S. Gaborski
Director, Laboratory of Intelligent Systems
Chairman

Date:

Dr. Joe Geigel
Reader

Date:

Thomas J. Borrelli
Observer

Contents

Abstract

The ventral stream of the human visual system is credited for processing object recognition tasks. There have been a plethora of models that are capable of doing some form of object recognition tasks. However, none are perfect. The complexity of our visual system is so great that models currently available are only able to recognize a small set of objects.

This thesis revolves analyzing models that are inspired by biological processing. The biologically inspired models are usually hierarchical, formed after the division of the human visual system. In such a model, each level in the hierarchy performs certain tasks related to the human visual component that it is modeled after. The integration and the interconnectedness of all the levels in the hierarchy mimics a certain behavior of the ventral system that aid in object recognition.

Several biologically-inspired models will be analyzed in this thesis. VisNet, a hierarchical model, will be implemented and analyzed in full. VisNet is a neural network model that closely resembles the increasing size of the receptive field in the ventral stream that aid in invariant object recognition. Each layer becomes tolerant to certain changes about the input thus gradually learning about the different transformation of the object. In addition, two other models will be analyzed. The two models are an extension of the “HMAX” model that uses the concept of alternating simple cells and complex cells in the visual cortex to build invariance about the target object.

1. Introduction

Modeling the human visual system has been an object of interest for many years. The vastness of information processed by the visual system and the complexity of the system has made modeling it difficult.

Over the years, there have been numerous studies done on the structural tasks of the human visual system. When a stimulus is presented, preliminary processing takes place in retina where basic features are extracted and an electrical representation of the image is transmitted as neural spikes along the optic nerve. The signal travels to the Lateral Geniculate Nucleus (LGN) via two pathways, parvoellular and magnocellular pathways. The LGN acts as a relay switch in the brain and routes the signal to the back of the brain towards the visual cortex. From there, the signal splits again to different areas of the visual system via the ventral pathway and the dorsal pathway. The object recognition takes place in the ventral pathway.

Studies like [Yaginuma *et al* 1993.; Lerner *et al* 2001] show that the ventral system is divided into different layers that aid in the recognition task. The ventral system is primarily divided into these components: V1(primary visual cortex), V2, V4, and different parts of the inferotemporal lobe (posterior inferotemporal, central inferotemporal, and anterior inferotemporal). Each visual area contains receptive fields of varying size. Each area performs different tasks and they are connected in such a way to represent the entire visual field.

In V1 and V2, low level features are extracted from the input such as edges, orientations, and gradient information. Then in V3 and V4, more complicated features are extracted such as color information and different perspectives of the input. This

information is then passed to the Inferior Temporal Cortex (IT) where the recognition takes place. It is in the IT where objects are labeled or associated with something familiar in the memory. These stages of functionalities are combined to form a model that is able to recognize an object regardless of its size, location, or rotation.

2. Background

In this section, different layers of the visual system, in particular the ventral stream will be briefly reviewed. The layers considered are V1, V2, V4, and the Inferior temporal cortex. The recognition process occurs in a hierarchical manner primarily involving these four regions as shown in [Felleman and Van Essen 1991; Barone *et al.* 2000] Furthermore, the interaction of visual information can be constructed by combining data from the human maps with anatomical and single-unit data from macaques [Tootell 1998]. Analyzing such data can lead to the structure and functionality of each of the region along the ventral stream.

2.1 V1 (Primary Visual Cortex)

About 90 percent of the information from the eye are relayed through the lateral geniculate nucleus to V1 [Tong 2003]. The primary visual cortex has been extensively studied in the past few decades. Several models have been developed to describe what kind of processing might happen in the primary visual cortex.

The primary visual cortex is further divided into layers comprised of different types of cells: primarily simple cells and complex cells. The simple cells have elongated receptive fields of varying orientation. Thus, they are maximally activated by a line of that particular orientation from a particular region of the retina. The complex cells have receptive cells that are similar to simple cells but the line can lie over a larger area of the retina. This makes the complex cells somewhat invariant to positions. Structurally, several simple cells of the same orientation converge onto a complex cell. The study

done in [Hubel and Wiesel 1968] discovered that cells in V1 are arranged in precise and orderly fashion. The cells are congregated based on the orientation tuning preferences thus making extraction of simple features possible at the V1 level.

From looking at V1 responses to simple stimuli, such as bars and grating, V1 can be modeled using linear filter responses, such as Gabor filters [Lee 1998; Jones and Palmer 1987], and Difference of Gaussian (DoG) filters. Thus these filters are used often to extract lines and edges of specified orientation and size from the target stimuli. These filters are formed with center-on surround-off, or center-off surround-on to extract the bar information from the image (Figure 2.1).

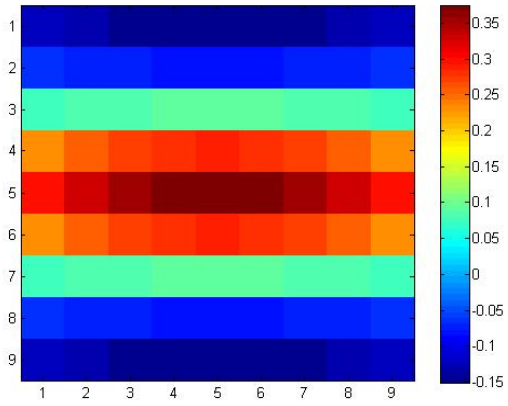

Figure 2.1a

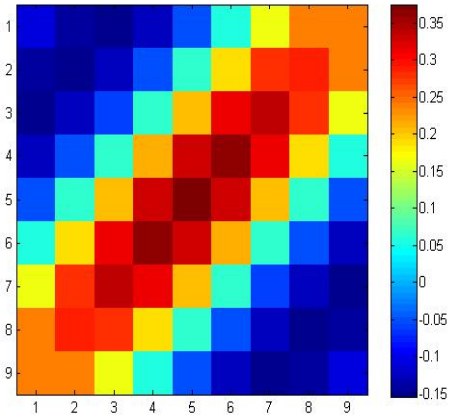

Figure 2.1b

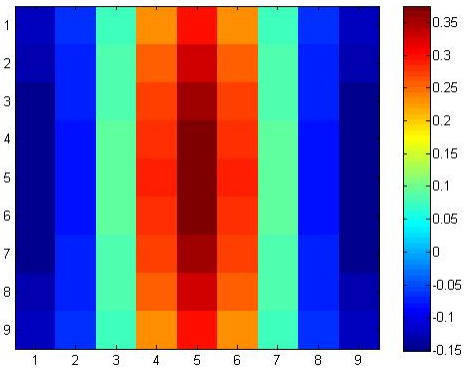

Figure 2.1c

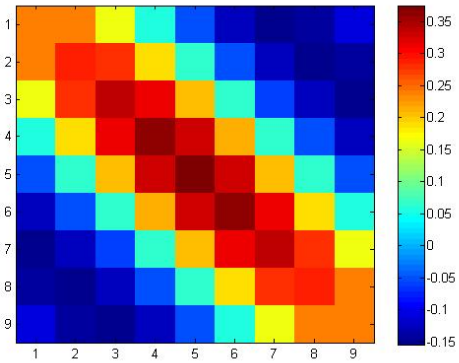

Figure 2.1d

Figure 2.1: Examples of Difference of Gaussian filter for different orientations. They are 0 (2.1a), 45 (2.1b), 90 (2.1c), and 135(2.1d) degrees in that order.

2.2 V2

V2 is rather less known compare to the primary visual cortex, but it is important in the visual processing nonetheless. The cells in V2 are selective for orientation similar to V1. In [Boynton and Hegde 2004], V1 simple cells might be constructed from series of

center-surround neurons in the lateral geniculate nucleus, thus it suggests that V2 might be constructed in a similar manner from V1 inputs. Thus, V1 and V2 aid in low level feature extraction and it won't explore any further than that in this segment.

2.3 V4

V4 receives strong connections from V1 and V2. Cells in V4 are also selective for orientation, length, and width of bar stimuli, as well as orientation of spatial frequency of gratings. They are also capable of responding to more complex shapes, varying in different curvatures [Pasupathy *et al* 2001]. More importantly, studies done in the past two decades [Reynods *et al* 2000; Connor *et al* 1997; Treue 2000] show that V4 is involved in forming recognition in early cortical processing.

2.4 IT

Inferior temporal cortex plays an important role in shape and object recognition. In [Booth and Rolls 1998; Desimonde *et al* 1984], the shape selectivity of inferior temporal neurons was studied by recording their responses to a set of shapes varying in boundary curvature. This supports the possibility that boundary curvature is an important stimulus dimension processed by inferior temporal cortex. Besides this, IT cells were found to be selective for color, texture, three-dimensionality, and more complex features such as faces and hands.

3. Previous Work

The human visual system is quite effective and efficient. It is able to distinguish and classify different objects regardless of its scale, rotation, view, perspective, and location at a glance. The models considered in this thesis are based on a hierarchical model, which uses different information about the input to build tolerance to its natural transformations of it (size, rotation, and translation). The models explained in Section 5 are an extension of the models described below. The fundamental basis for the base models are the same, however, important changes have been made to the models to be more robust in building tolerance to different aspects of the object.

The models described in this section can basically be divided into two categories: alternating simple and complex cells and neural networks using the trace rule. The trace rule allows the network to learn about the various aspects about the input by training the network with sequence of images. The first category of models described is a hierarchical model with several layers simulating the behaviors of simple and complex cells. The two models following it are neural networks approach to invariant object recognition. They incorporate the idea of trace learning rule to learn about the different aspects of the target object(s).

3.1 Feedforward model using Simple and Complex cells

The model is described in [Riesenhuber and Poggio (1999 and 2000)], and the inspiration for this model comes from the increasingly sophisticated representation of the processing in visual cortex in a hierarchical fashion. The model makes use of the

simple and complex cells that are present in the visual system to explore the feasibility of using biological scheme to aid in object recognition. The model is described in Figure 3.1.

From the results of studying done in [Bruce *et al* 1981; Ungerleider and Haxby 1994] of macaque inferotemporal cortex, the cells in this region are found to be tuned to view of complex objects such as faces but do not respond strongly to other objects. What's more interesting to note is that the neurons are able to respond to different views of the stimuli it is tuned to, such as position, scale, and different rotation of the stimuli. However, these cells respond differently to similar stimuli such as two faces.

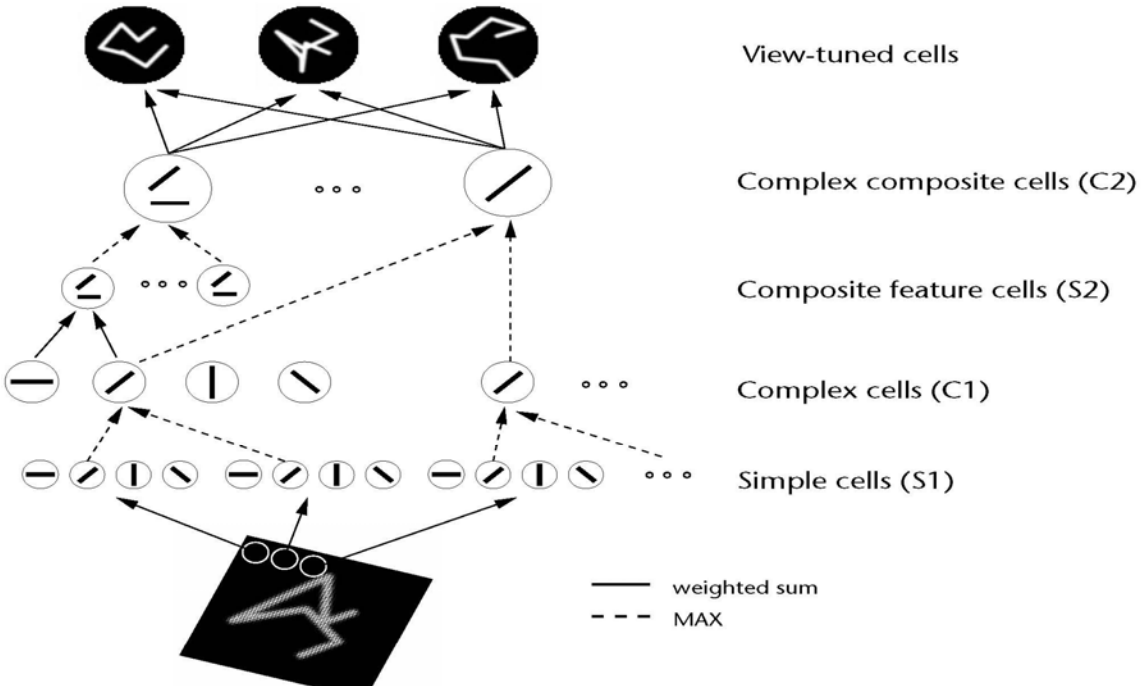

Figure 3.1 [From Riesenhuber and Poggio (1999)] This figure illustrates the HMAX model described in section 3.1. The model is comprised of layers of alternating simple and complex cells that help to build invariance in the model.

Utilizing this concept, the model is constructed with alternating simple cell layer and complex cell layer based on the hierarchical model in [Perret and Oram 1993]. To make the model tolerant to position and scale, it uses linear summation and maximum operations. The linear summation is used to define the specificity of the model, whereas the maximum operation allows a degree of invariance of the stimuli as long as it is contained within the receptive field. This works because the maximum operation considers the most active response from the incoming layer, thus as long as the target stimuli is visible within the receptive field, position or size of the stimuli would not affect the recognition process too greatly. In Figure 3.1, the simple cells are tuned to respond to bars of different orientations. These cells are then pooled in the next layer of complex cells over the same orientation but different scales. This allows the model to build tolerance to small shifts and small changes in size.

The stimulus is filtered through a layer of simple cell-like receptive fields, which can be commonly modeled using Difference of Gaussian or Gabor filters, varying in different orientations and sizes. The filtered output is fed into the next layer, composed of complex cells, that pool over simple cell units using a maximum operation over same orientations and different scales. The output from this layer is then combined to form the next layer, either by combining units from previous cells with different preferred orientations or over same orientation but differing in the size of the receptive field. However, more layers of simple and complex cells can be added in principle.

The alternating of simple and complex cell layers are used in this hierarchical model to build invariance by pooling over outputs resulting from different orientation and scales. This max operation allows position invariance within the receptive field, since it

will only consider the ‘best matching’ inputs. The model also builds tolerance to size by further pooling over larger receptive fields and limiting overlapping units.

This model is adequate to detect simple objects. However, it can be further enhanced by using more complicated features and adding additional layers of alternating simple and complex cells to be used in more complicated and sophisticated object recognition task as it will be analyzed in the later sections.

3.2 Neural Network using Modified Hebbian Rule

The model considered in [Foldiak 1991] uses a learning rule to learn about the different transformation of the object similar to [Klopf 1982; Sutton and Barto 1981]. The basic idea behind this model is that the visual system must contain knowledge about such transformations in order to be able to generalize correctly. The different experiments in [Foldiak 1991] suggest that the invariance properties of neurons may be due to the receptive field characteristics of individual cells in the visual system.

Many neural networks use Hebbian learning rule of [Hebb, 1949] to change the strength of the connection. The weights are adjusted so that each weight better represents the relationship between the nodes. So if two nodes are simultaneously active, then this learning rule will increase the connection strength between the two so that excitation of either one will induce the excitation of the other. On the other hand, when two neurons are active at opposite times, then the learning rule gradually decreases the connection strength between them. The model described in this section uses a modified Hebbian learning rule to learn about the different ‘shifts’ of the object. The idea is that by using the learning rule, it can pickup correlations between different

positions. The model implemented is a neural network, and the learning rule imposed is a modified Hebbian learning rule. The learning rule is slightly different from the Hebbian learning rule where the modification of the synaptic strength at a given time is proportional to the pre-synaptic activity and a trace of the postsynaptic activity. Thus whenever the weights are being updated, it always takes into consideration the previous activity of the cell. The usage of this modified Hebbian rule would allow the network to be able to learn about the different shifts of the object and build tolerance to these different shifts.

The neural network uses layers of simple and complex cells, similar to [Riesenhuber and Poggio 1999]. However, the architecture of the network is somewhat different. The simple layer is consisted of position-dependent line detectors, one unit for each of 4 orientations in the 8x8 grid. The complex layer is consisted of 4 units, fully connected to the simple layer. The simulation is done by moving a line across the simple layer. Thus, the cells in the simple layer would respond to different orientations of the line. The learning rule on top of that should train the network so that the cells in the complex layer are trained to respond to varying shifts of the input.

3.3 Hierarchical Feedforward network with modified Hebbian Rule

The model considered in [Wallis 1996] uses the modified Hebbian learning rule as shown in [Foldiak 1991], but uses a bit more extensive hierarchical model. Again, the basic idea behind is the same as before. The learning rule uses spatio-temporal correlations of stimuli as well as spatial ones to assemble stimulus categories, building on the assumption that images of object appear in short sequences.

The network, dubbed TraceNet, is comprised of two-layers with competition within the layer. The first layer of the network consists of 256 neurons arranged in 4x4 grid of 4x4 inhibitory pools. Each pool fully samples a corresponding 4x4 patch of a 16x16 input image. The competition that takes place is ‘winner-take-most’, or leaky learning. The second layer is centered about the input layer and the neurons are fully connected. The TraceNet is trained with a set of hand written digits. And for each digit, there are several training samples for them. The idea is to learn about each of the digit by presenting the different views of the digit using the learning rule. This way, trained TraceNet should be able to learn about the different views of the digit and the output layer should respond to for the digit regardless of its different views of it.

The models described in this section are intended to mimic certain functions of the human visual systems to incorporate invariance into the network. The modified Hebbian learning rule and the usage of simple and complex cells are intended to do just that. The learning rule and the usage of simple and complex cells are used to build position, scale, and rotation invariance throughout the network, so that the network is able to respond the same way even though the appearance of the object may have been changed. The models considered in Sections 4 and 5 are enhancements of the models described in this section.

4. Standard Model

The model [Serre, Kouh *et al* 2005] only considers the first 150 ms of the flow of information in the ventral stream. During this short time, "immediate recognition" tasks occur in the human visual system. Also, because the time duration considered is short, the connectivity is mainly feedfoward across visual areas. According to [Potter 1975], recognition is possible for stimuli viewed in rapid visual presentation that do not allow sufficient time for eye movements or shifts of attention. Furthermore, the object recognition task can be done by human within 150ms, which is backed by [Hung *et al* 2005], whose studies show that portion of the IT region contained accurate and robust information supporting a variety of recognition tasks.

Thus, the information processed by a layer is forwarded to the layer above the current layer. However, operations in which information is forwarded are different from layer to layer, which eventually aids in the network to be tolerant to different natural transformations of the target object. This is illustrated in Figure 4.2.

The model is explained in full in [Serre, Wolf, *et al* (2005, 2007)]. The model is based on using simple and complex cells to build some form of invariance throughout the network. The model contains several layers, which alternates between simple and complex cells. The simple cells in the visual system are responsive to bars of varying orientation and the complex cells get input from afferent simple cells and are tolerant to small shift in position and scale.

In the model, first two layers correspond to primate primary visual cortex, V1, (S1 and C1). The following two layers behave similarly to the first two layers but are more tolerant to position and scale changes of the target object. Additional layers can be added to the model by following the simple cell and complex cell properties.

4.1 S1

The S1 layer of the hierarchy corresponds to the simple cells found in the primary visual cortex. The cells respond to bars of a certain orientation. Thus, the S1 cells can be simulated using Gabor filter [Jones and Palmer 1987]:

$$G(x,y) = \exp\left(-\frac{(X^2+\gamma^2Y^2)}{2\sigma^2}\right) \times \cos\left(\frac{2\pi}{\lambda}X\right), \qquad \text{Equation 1}$$

$$\text{where } X = x\cos\theta + y\sin\theta \text{ and } Y = -x\sin\theta + y\cos\theta.$$

The parameters $\gamma, \theta, \sigma, \lambda$ are adjusted to match S1 units: γ controls the aspect ratio, θ adjusts the orientation (0^o, 45^o, 90^o, and 135^o), σ adjusts the effective width and λ controls the wavelength. To simulate the simple cell processes, a number of these filters are created ranging in orientation and scale and applied to the input image. The filter sizes varies from 7x7 to 37x37 increasing in steps of 2 pixels with 4 orientation per size thus resulting in 64 different filters. The shapes of all 64 filters are shown in figure 4.1. These filters will respond differently to bars and gratings of different sizes and

orientations. The stimulus is convolved with this set of Gabor filters and the output values are pooled to the C1 layer to allow small invariance to position and size.

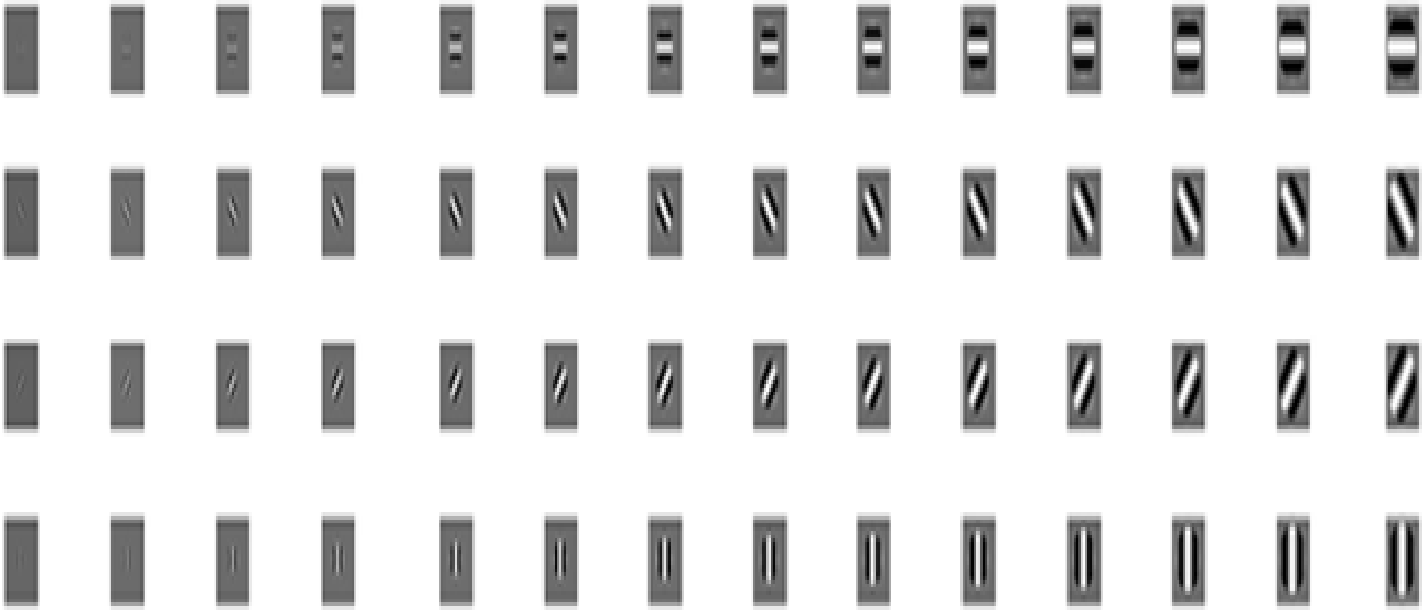

Figure 4.1 Shape of all of the Gabor filter used to compute S1 layer. The size of the receptive field increases from left to right. Each row corresponds to different orientation of the filter: 0^{o}, 45^{o}, -45^{o}, and 135^{o}.

4.2 C1

The output values from S1 layer are used in the C1 layer. This layer corresponds to the complex cells that are slightly invariant to locations and scales. Generally, the complex cells have larger receptive field than the simple cells. In [Hubel and Wiesel 1968], the complex cells have twice the size of receptive field than the simple cells and thus respond to bars anywhere within its receptive field making it more broadly tuned to

spatial frequency than simple cells. This makes the complex cells tolerant to small changes in location or in size.

To achieve this effect, the C1 layer pools over S1 units. The S1 units are first grouped into different bands. Each band contains two adjacent scales totaling in 8 different bands. Thus band 1 has two S1 output maps resulting from filters of two different scales 7x7 and 9x9, band 2 has 11x11 and 13x13, and so forth to band 8 which has 35x35 and 37x37.

Once the S1 maps are grouped into different bands (8 bands in total), C1 pools over a band of S1 units that are of the same orientation. This allows the C1 layer to be tolerant to position to a degree.

The pooling occurs in two stages. First, S1 units are pooled over a specified sample grid over different scales and different orientations. Then take the max from the sample grid for each orientation and scales, so that you have a value for four orientations and two scales. Once this is achieved, take the max over the two scales. For example, the grid size for the first band is 8x8. Thus for each location, there are 64 possible values. A maximum over the 64 values are chosen for each of the locations for the different orientations and different scales. Thus there are 2 scales and 4 orientation maps for each of the scales. Second max operation is applied to the resulting map across scale over the same orientation. After this process, the C1 layer contains four maps corresponding to different orientations. This pooling process and the softmax

operations allow the C1 to be slightly invariant over different scales and different shifts of the input.

The pooling occurs over a specific window depending on the band. For the first band, the window size is set at 8x8 and the size of the window increases by 2 pixels across different bands. Since the pooling occurs over a window, there will be overlapping values. To reduce the redundant information, the maps obtained through the pooling are sub-sampled.

The sub-sampling occurs such that, if NxN is the size of the pooling window, only N/2 S1 units overlap. So, the sub-sampling occurs by taking every N/2 values from the S1 maps. This will lead to the reduction of units from S1 layer to C1 layer quite significantly, which will make the computation at the next level more efficient.

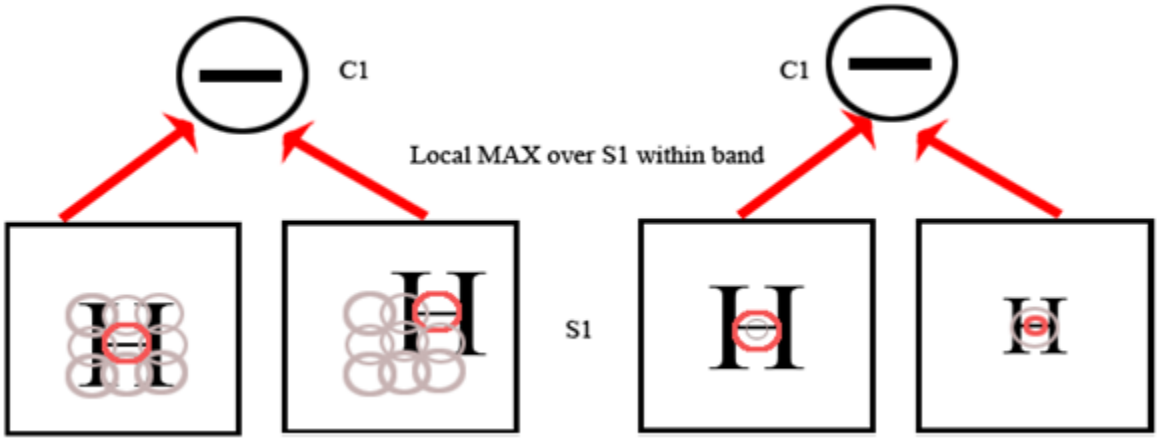

Figure 4.2 Adapted from [Serre, Wolf *et al 2005*]. From S1 to C1, the tolerance increases over shift and size. The figure on the left pools over different positions that falls within its receptive field. Since the maximum value is used for the C1 layer, the strongest value for the same input for two different position will still be the same, thus tolerant to small changes in location. Same scheme applied for different scales for the figure on the right. Since C1 pools over two different scales, it will be tolerant to small changes to size of the input.

4.3 S2

The values for S2 units are calculated using the output values of the C1 layer. The size of the receptive field is increased by sampling random C1 units over different filter scales and orientations. During the learning stage (Section 4.6), N number of prototypes is extracted from the C1 layer. The values for the S2 units are calculated by using a Euclidean distance between the prototype patch and trying to match it to all different samples of crops in the C1 layer using equation 2. Thus for each unit in band of C1, and for all of the four orientation for that particular location, a distance value is calculated between the samples in C1 with the prototype patches.

$$r = \exp(-\beta||X - P_i||^2) \qquad \text{Equation 2}$$

4.4 C2

The C2 layer pools over the S2 layer using the similar softmax operation of C1 units. This further allows the layer to be more tolerant to shifts and scale changes within its receptive field. The max pooling operations over different scales and orientations allow the size of the receptive field to grow. From the S2 space, it picks N values with the minimum distance from the prototype patch. Thus, if there are 250 samples per patch size, then C2 would have a vector containing 250 x 4 (number of patches) values that can be used to classification of different object.

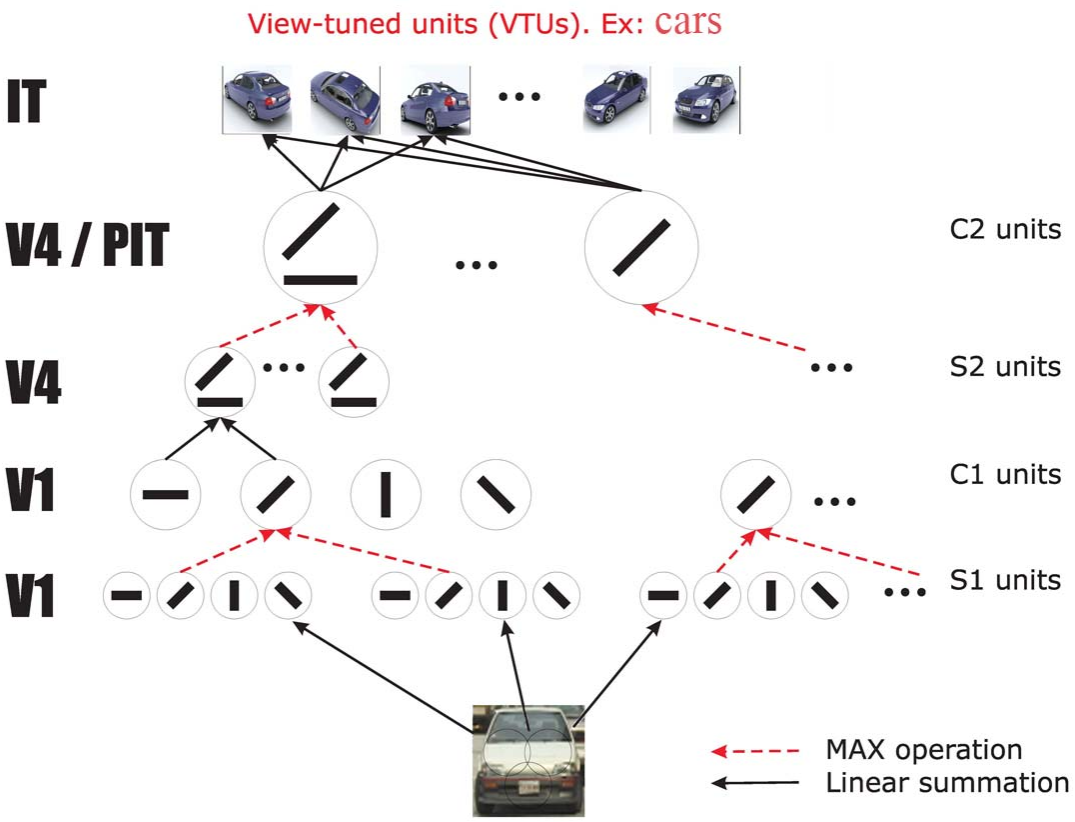

Figure 4.3 Adapted from [Serre, Wolf *et al* 2005]. The figure demonstrate the overall processes that occur within the model. The model is basically composed of two computations: linear summation (simple cells) and non-linear MAX (complex cells) operation. These functions alternate between the layers to achieve invariant recognition of the object.

4.5 Additional layers

Additional S and C layers can be added following the similar alternating technique. For example, S3 can be computed like S2 units, pooling over different scales and orientation with weights from C2 layer. Also, the C3 layer can perform the same softmax operation over S3 layer to achieve further increase in tolerance to shift and scale changes.

4.6 Learning

The learning in this model happens at S2 level and higher. The model is trained using random patches from the images. The patches extracted in [Serre, Wolf and Poggio 2005] are from the C1 layer. The patches vary in sizes (4x4, 8x8, 12x12, and 16x16) and are extracted from random image in the training set and at random location. Each of the patches is sampled across the different orientation. Thus, for a 4x4 patch since there are 4 different orientations (0, 45, 90, 135), it would contain 4x4x4, or 64 values. These patches are later used during the classification stage to calculate the distance between the patch and with different samples from the test set.

4.7 Classification

For each of the image in the test set, they are filtered through the model all the way up to the C2 layer. To obtain the values for C2, the test image must be first compared to the prototype samples that are obtained during the training phase. The values obtained in C2 are then put through an SVM or Nearest Neighbor classifiers to find the likeliness of the test image being belonging to the target object.

5. Standard Model with Sparse, Localized Features

In [Mutch and Lowe 2006], similar architecture of the previous model in Section 5 is considered and builds on it to add sparsification and lateral inhibition. Thus, the basic hierarchy of this model is similar to that of [Serre, Wolf, *et al* 2007; Serre, Wolf, and Poggio 2005]. They both are an extension of the HMAX model of [Riesenhuber and Poggio 1999], where the invariance is built using the alternating of simple cells and complex cells.

The model is built in two stages. First, a base model is created, which performs as well as the model described in [Serre, Wolf, *et al* 2007; Serre, Wolf, and Poggio 2005]. Base model is further improved by using sparsification of features and a form of lateral inhibition, which increased the performance of the model greatly.

5.1 Base Model

The model contains five layers: an initial image layer, and four alternating simple and complex cell layers. These four layers are built from the previous layer by alternating template matching and max pooling operation as described in the model from Section 4.2.

The image layer has fixed constraints: all images are converted to grayscale and the shorter edge is scaled to 140 pixels while maintaining the aspect ratio. Once the images are converted and scaled, an image pyramid is created which consists of 10

scales. The images are converted using bicubic interpolation (Equation 3) with each image $2^{1/4}$ smaller than the previous image.

$$\sum_{i=0}^{3}\sum_{j=0}^{3} a_{ij}x^i y^j$$ Equation 3

5.1a S1

The layer on top of the image layer is the Gabor filter or S1 layer. The S1 layer is computed by centering the two-dimensional array at each possible position, scale, and orientation for all the scales of the images in the image layer. Since the image layer is a three-dimensional pyramid, the resulting S1 layer is four-dimensional structure that has the three-dimensional pyramid shape but with maps for different orientation over each position and all scales.

The shape of the filter is a basic Gabor filter and the size of the filter is fixed at 11x11 [Equation 1]. The values for x and the values for y vary from -5 to 5 to create the 11x11 matrix and θ determines the orientation of the filter and it ranges from 0^o to 180^o. The γ (aspect ratio), σ (effective width), and λ (wavelength) are set at 0.3, 4.5, and 5.6 respectively and is taken from [Serre, Kouh *et al* 2005]. The values in S1 structure are then normalized such that the mean is 0 and the sum of the squares is 1.

$$R(X,G) = \left| \frac{\sum X_i G_i}{\sqrt{\sum X_i^2}} \right|$$ Equation 4

The responses are computed by using the Equation 4, given patch X and filter G. This is simple to do in MatLab. The image is convolved with the Gabor filter and the resulting values are normalized.

5.1b C1

Following the S1 layer is the local invariance, or C1, layer. The C1 layer pools over S1 units that are tuned to the same orientation to build tolerance over position and scale over local regions. This can be also done by sub-sampling the S1 units to reduce the number of units. To obtain the value for the C1 layer, the S1 pyramid is convolved with a three-dimensional max filter, whose dimension is 10x10x2. In essence, the max filter is also a pyramid. The individual values in C1 layer is obtained by simply taking the maximum of the values that lie within the three-dimensional max filter for that particular orientation. The dimension of the C1 layer is smaller than the dimension of the S1 layer because the max filter is moved around the S1 pyramid in steps of 5 pixels in position, but only 1 in scale.

The resulting pyramid is spatially smaller than the pyramid in S1 layer. However, number of features per position is still the same. Each position contains maximum value of the pooled units per orientation.

5.1c S2

The third layer in the model is the Intermediate feature, or S2, layer. The values for the S2 units are calculated by performing template matching operation between the patch of C1 units centered at that position and scale and each of the prototype patches (prototypes computed in Section 5.1e). The prototype patches are obtained during the learning stage by sampling random patches of size 4x4, 8x8, 12x12, or 16x16 at random position and scale in the C1 layer for the given image.

For each of the prototype patch, it is compared to the patch from the C1 layer and a distance is computed using a Gaussian radial basis function:

$$R(X,P) = \exp\left(-\frac{\|X-P\|^2}{2\sigma^2\alpha}\right)$$ Equation 5

X is the sample from C1 and P is the prototype patches. The size for both patches id nxnx4, where n € {4, 8, 12, 16}. σ or standard deviation is set to 1 for all computations. α is the normalizing factor for different patch sizes and set to be the ratio of the dimension of P to the smallest patch size, thus $\alpha = (n/4)^2$.

5.1d C2

The final layer is the Global Invariance (C2) layer. As the name suggests, global invariance is achieved at this layer. This layer consists of d (number of prototypes) -

dimensional vector consisting of maximum response to one of the d prototype patches. Since the maximum is taken over all the values in C1, the location and size information are removed.

5.1e Learning

Similar to the model in [Serre, Wolf, *et al* 2007], the learning happens at the S2 layer. During the training phase, d prototype patches are extracted from random images at random position. The prototype patches vary in size from 4x4 , increasing in increments of 4, to 16x16. It is suggested that the smaller patches could be seen to encode shape of the target, where the larger patches could be used to gather texture information. Since the patches are taken randomly from the layer, many of the patches might not have useful information about the target object. However, during the SVM step, the patches are weighted accordingly.

5.1f SVM Classifications

Once the C2 vector is obtained, it is classified using an all-pairs linear SVM. The data is first normalized to zero for the mean and one for the variance. The training images are first used to build the SVM, and then the test images are assigned to categories using the majority-voting method.

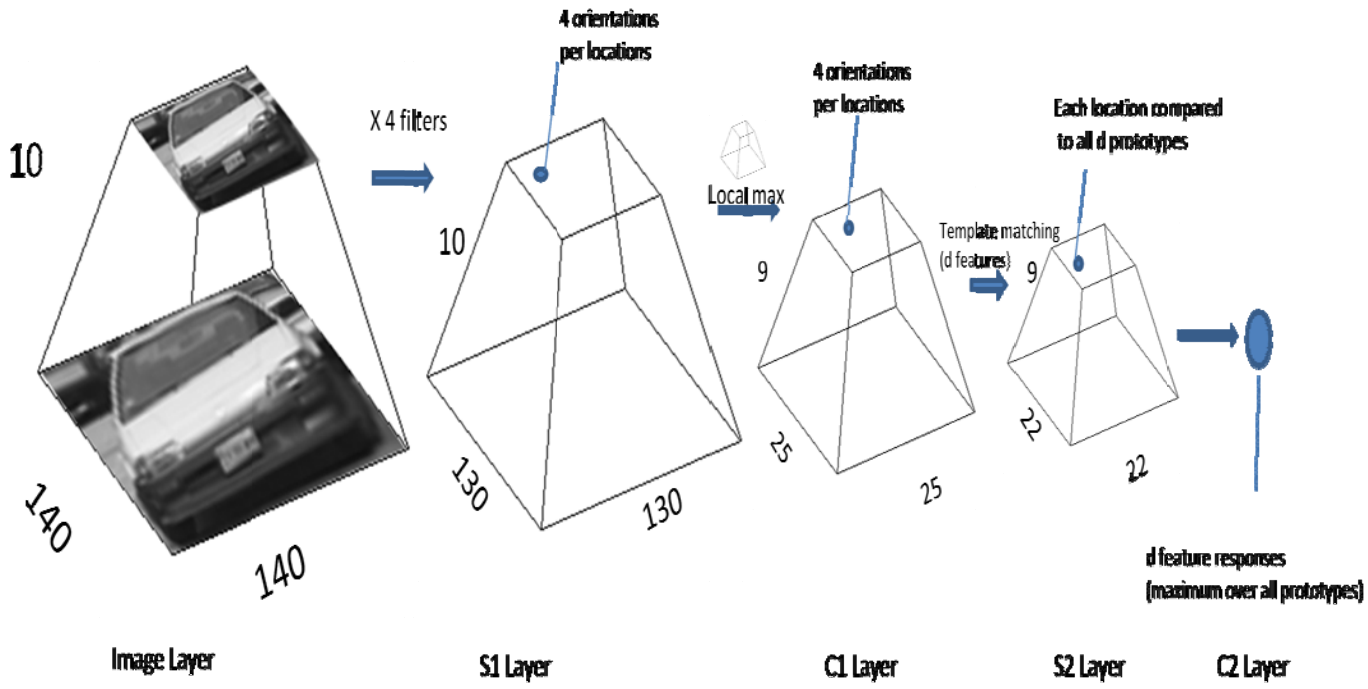

Figure 5.1 Adapted from [Mutch and Lowe 2005]. Overview of the base model. Each layer has three spatial dimensions at each of the locations. There are only 1 value for each of the location in the image layer. There are four values for each location in S1 and C1 layers, each values corresponding to a particular orientation. The upper two layers have d types per location, corresponding to number of prototype features. Each layer is computed by convolving a template matching or max pooling filters to the layer below and this can result in different sizes at each of the layer.

5.2 Sparsification

In the base model, the prototypes are extracted from the C1 layer by taking the C1 values that falls within the patch window and across all the orientations. This may not be the best approach because not all the values have pertinent information about the input. The approach taken in the next phase is to use sparse features as much as possible. This is done by taking fewer features from C1 into S2.

More specifically, instead of taking 4 (number of orientations) of features per position, only 1 feature per C1 position is sampled. Thus, only the most dominant response per position is used. While doing so, the magnitude and the identity of the dominant orientation are stored. This results in reduction of number of features from 64 to 16. Since the number of features is reduced, information containing combinations of different filter responses are lost. To supplement the loss, the number of orientations is increased from 4 to 16, having finer gradient over specificity. Now, the S2 features are based on correctness of orientations rather than different combination of them.

5.3 Lateral Inhibition

The lateral inhibition here is implemented by suppressing the outputs of S1 and C1 layers. This is achieved by encoding different orientations at the same position and scale. In essence, each unit is competing to describe the dominant orientation at the particular location. The competition is done using a global parameter *h,* to define the inhibition level, which can be set between 0 and 1 and represents the fraction of the response range that gets suppressed. So at each of the possible location, minimum, $R_{min,}$ and maximum, R_{max}, responses are calculated over all orientations. For each of the response R, if $R < R_{min} + h(R_{max} - R_{min})$ then the response is set to be zero. This allows some competition for that particular position in the same scale, since only the dominant responses will be passed up to the next layer.

5.4 SVM Weighting

The prototypes are sampled at random location. This would result in prototypes that may contain no useful information about the object of concern and could have unwanted cost. During the construction of the SVM, features with low weights are dropped. The training phase of the model consists of classifying S2 vectors of different categories. This means that all features are shared by all categories. Thus, the features are dropped based on the average weight of all the features in SVM. The features are reduced in a multi-round “tournament” fashion by setting the weights to zero.

The model then looks at the most dominant features for each category with the reduction of the features. This aids in elimination useless features that may be part of the background that we are not concerned with. Thus, the feature reduction improved the classification performance and made it more efficient to compute.

6. Feedforward Hierarchical Model with Trace Learning Rule

The model implemented in this thesis is a version of VisNet described in [Wallis and Rolls 1997; Rolls and Milward 2000], which is a four-layer feedforward network where small area of neurons in each layer converges to the neurons in the next level. In addition, there exists competition and trace learning rule for each neuron that aid in learning about the object and responding to the same object wherever it is located in the retina. The four layers in the model correspond to different layers that exist within the visual system. In particular the layers correspond to V2, V4, the posterior inferior temporal cortex and the anterior inferior temporal cortex of the ventral visual system.

The model described in this section is quite different from the models described in Sections 4 and 5. The previous models are formed with alternating simple and complex cells that are used to build tolerance to shifts or changes in the appearance of the object. The model implemented for this thesis is a neural network where the trace learning rule modifies the weights between connections to build tolerance to changes in size, location, or rotation of the object.

6.1 Architecture

Between each layer in the model, a Guassian distribution is used to determine the connection probabilities of feedforward connections to individual cells. The forward connections are made using the Guassian distribution with the radius as one standard deviation from the mean, thus 67% of the connections are made within the radius. The

radius of convergence increases as you move up the hierarchy. This allows the neurons in the top layer to have information about the overall input space which helps in the invariant object recognition. The general hierarchy of the network is described in the figure 6.1.

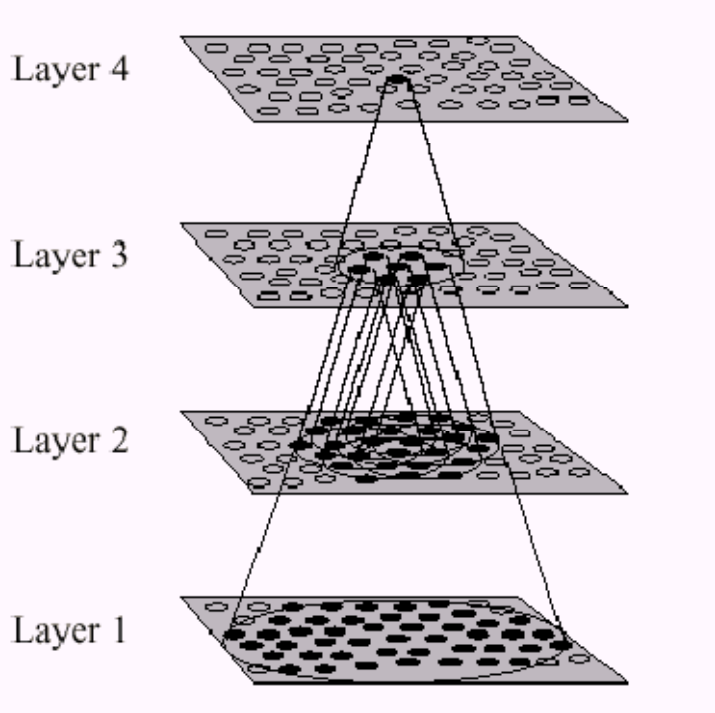

Figure 6.1 [Wallis and Rolls 1997] The conceptual view of the model. The model contains four layers with connections made between adjacent layers. Small area from the preceding layer converges to a neuron in the current layer. This convergence happens between all four layers. This allows a small set of neurons in the fourth layer to contain information about the entire retina.

The bottom layers are trained to learn about the simple characteristics of the input and the upper layers are trained to learn about the different perspective of the input. Thus, if trained properly, the top layer of the network could be used for invariant object recognition.

6.2 Input

The input to the network is pre-processed first with the filters that correspond to processing that occurs by the simple cells that are present in the V1 layer. To simulate this behavior filters were formed using difference of Gaussian weighted by a third Gaussian with the equation 6.

$$\Gamma_{xy}(\rho,\theta,f) = \rho\left[e^{\left(\frac{x\cos\theta+y\sin\theta}{\sqrt{2}/f}\right)^2} - \frac{1}{1.6}e^{\left(\frac{x\cos\theta+y\sin\theta}{1.6\sqrt{2}/f}\right)^2}\right] \times e^{\left(\frac{x\sin\theta-y\cos\theta}{3\sqrt{2}/f}\right)^2}$$

Equation 6

The three parameters ρ, θ, and f correspond to sign, orientation, and frequency. For the experiments done using this model, only positive filters are used thus ρ = 1 for all filters. The θ value determines to which orientation the filter would respond strongly to. The model uses filters varying in four orientations: 0^o, 40^o, 90^o, and 135^o. The indices x and y indicate the distance away from the center. The size of the filter used is 13x13, thus the values for x and y range from -12 to 12 with (0,0) as the center of the filter. The parameter f is used to determine the spatial frequency of the filter; there are also four values for f: 0.0625, 0.125, 0.25, and 0.5 cycles/pixel. The shape of the filters are depicted in the figure 6.2.

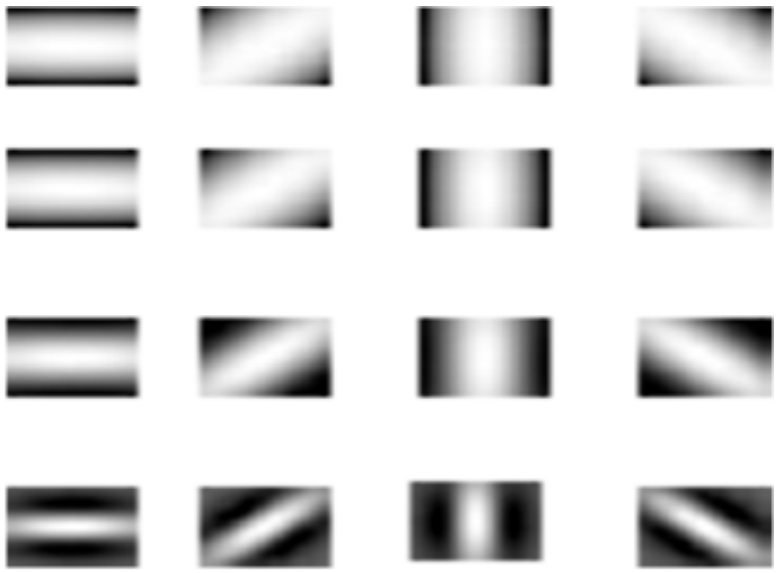

Figure 6.2 Shapes of all the filters used in the model by the pre-processing step. There are 16 filters in all, ranging in orientation and frequency. The filters differ in orientation from left to right (0^o, 45^o, 90^o, and 135) and the frequency is increased from top to bottom (0.0625, 0.125, 0.25, and 0.5 cycles/pixel).

The input image is convolved with the different filters. The filtered outputs are thresholded at zero, thus only the positive responses are considered. Additionally, the responses are normalized across all scales to compensate for the low-frequency bias in the images of natural objects [Elliffe *et al* 2002]. By using these different filters, general information about the input can be extracted from the stimulus.

The connections between the input layer and the first layer of the model are made randomly. The size of the input space is 128x128 and the size of the layers in the model is 32x32. Thus, the first layer pools its inputs such that the distribution of the information among the layer is even. This is achieved by taking each of the neurons in 32x32 space and distributing them evenly on top of the 128x128 input space. Thus, each neuron in the first layer becomes the center for the pooling radius. Then random

connections are made using this center point based on uniform connection probability. The radius used in all experiments is 6. The connections are made across all frequency and orientation and must take samples from each of the spatial frequencies. The number of connections made per frequency is given in table 1.

Frequency	0.0625	0.125	0.25	0.5
Number of Connections	8	13	50	201

Table 1 Connectivity information for layer1. Connections must be made across all frequencies. The number of connections made are 8, 13, 50, and 201 for frequencies 0.0625, 0.125, 0.25, and 0.5 respectively.

Each connection also has a weight associated with it. It is initialized with a random number and modified during training. Also, the weights are normalized such that the sum of the square of the weights equals 1. Once the connections are all set up, the input image can be passed to the hierarchy bottom up.

6.3 Connections between Layers in the Network

As described from the previous section, the layer receives its inputs from the preceding layer. The connections between the adjacent layers are also made randomly. Each neuron in the 32x32 space receives 100 connections from the previous layer. The way in which the connections are made is done randomly. However, each neuron receives its input based on a Gaussian probability with the given radius. This radius of convergence increases slightly from bottom layer to the upper layer. Also, each

connection carries a weight and it is initialized with a random value. The connection radius for each layer in the network is given in table 2.

	Dimensions	Num. of Connections	Radius
Layer 4	32x32	100	12
Layer 3	32x32	100	9
Layer 2	32x32	100	6
Layer 1	32x32	272	6
Input Layer	128x128x16	-	-

Table 2. Dimension for each layer in the network.

6.4 Activation of a Neuron

The activation rate h for the neurons in the first layer is calculated using the random weights assigned and the filtered output value. Each cell in the first layer receives 272 connections from the input space. The sum of the dot product of the weights and the filtered output determines the firing rate for the target neuron (Equation 7).

$$h = \Sigma_j x_j w_j$$ Equation 7

In equation 7, x_j indicates the firing rate of an input to the neuron and w_j is the weight associated with that input.

6.5 Local Competition

Within each layer, graded competition exists rather than winner-take-all competition, in principle by a lateral inhibition-like scheme. Lateral inhibition was used

to diversify the information between all the inputs. By using a local inhibition scheme, different area of the layer would fire to different stimulus thus distributing the information across all the neuron in the network. To be able to do this, first a spatial filter is obtained which basically contains a positive center surrounded by negative values. This filter is convolved with the current layer with the activation rate already calculated using equation 7. The filter is setup so that the resulting values do not alter the average activation rate.

The different lateral inhibition filters are used for each of the layers in the network. The size of the filters is about half the size of the radius of convergence. Actual shapes of the filters are given in figure 6.3.

The size of the filters increases as you go up the hierarchy so that the each neuron responds differently to different inputs. Thus in the fourth layer, each neuron or cluster of neurons should theoretically behave differently to different input that are being passed in to the network.

The values for the filters are created using the equation:

$$I_{a,b} = \begin{cases} -\delta e^{-\frac{a^2+b^2}{\sigma^2}} & \text{if } a \neq 0 \text{ or } b \neq 0, \\ 1 - \sum_{\substack{a \neq 0 \\ b \neq 0}} I_{a,b} & \text{if } a = 0 \text{ and } b = 0. \end{cases}$$

Equation 7

The lateral inhibition parameter pairs σ and δ for layers 1 through 4 are, 1.38 and 1.5, 2.7 and 1.5, 4.0 and 1.6, and 6.0 and 1.4 respectively.

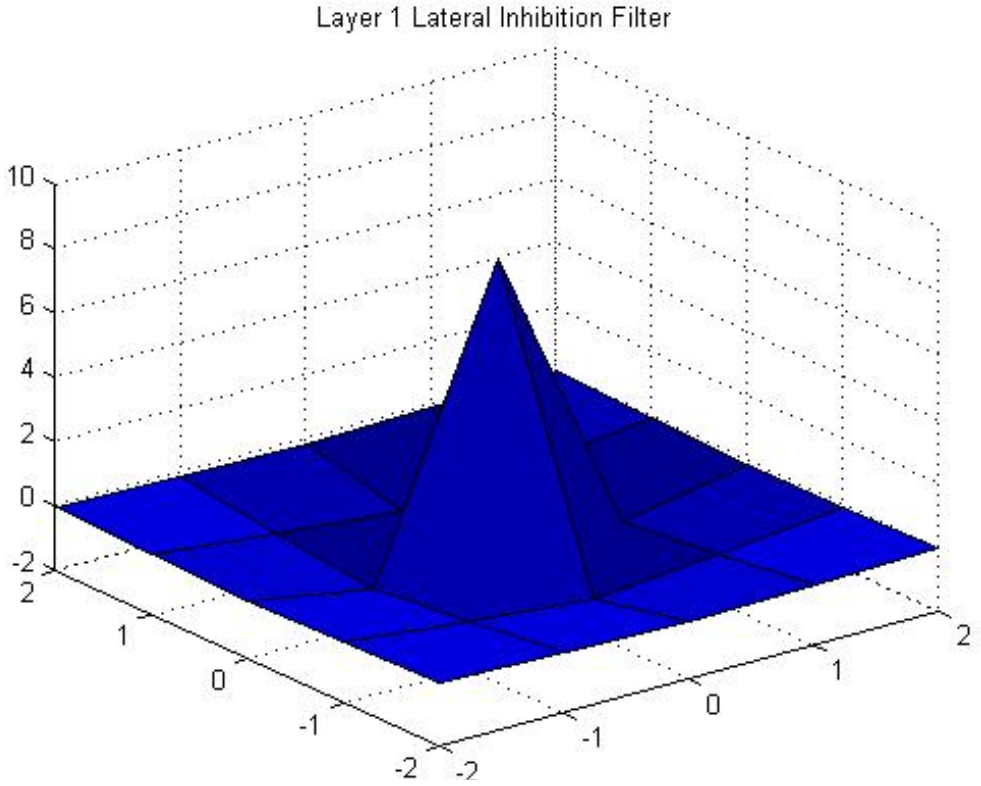

Figure 6.3a. Lateral inhibition filters used for the first layer. The pair σ and δ is 1.38 and 1.5 respectively.

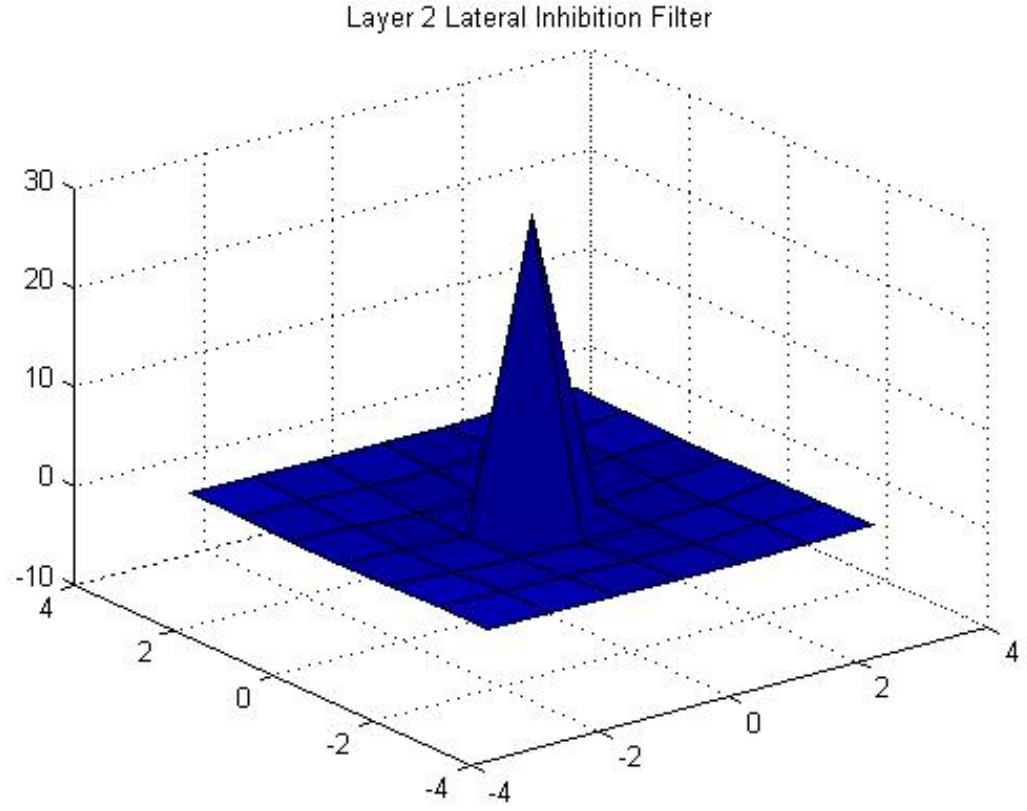

Figure 6.3b. Lateral inhibition filters used for the second layer. The pair σ and δ is 2.7 and 1.5 respectively.

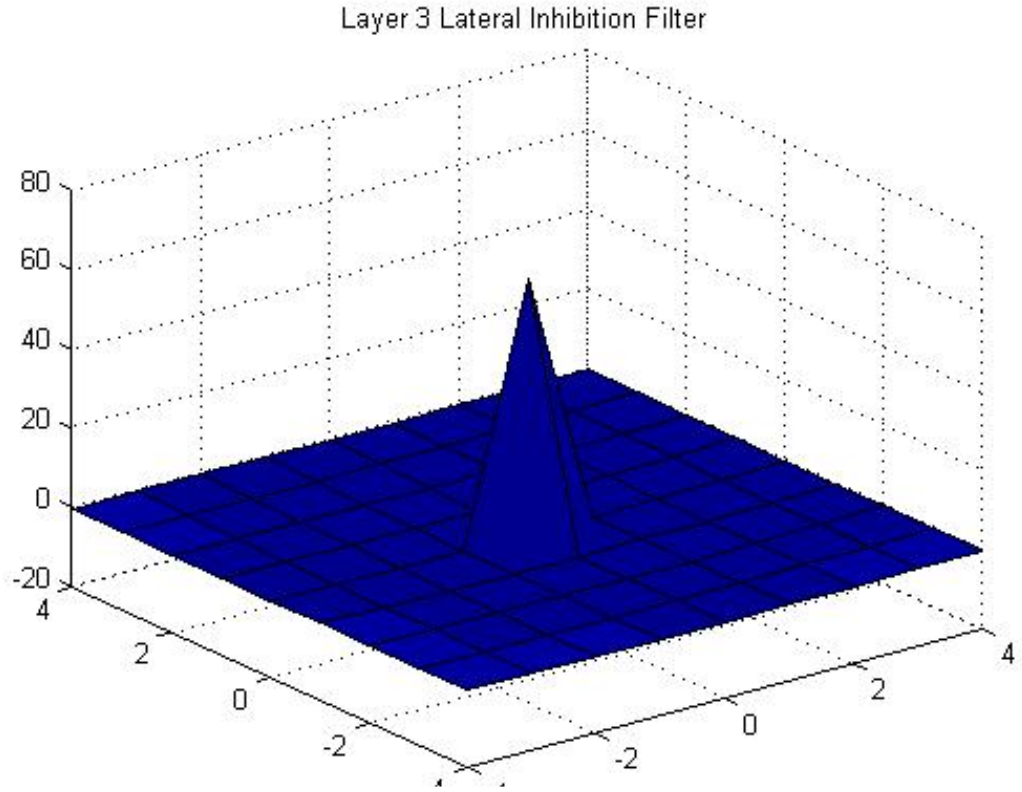

Figure 6.3c. Lateral inhibition filters used for the third layer. The pair σ and δ is 4.0 and 1.6 respectively.

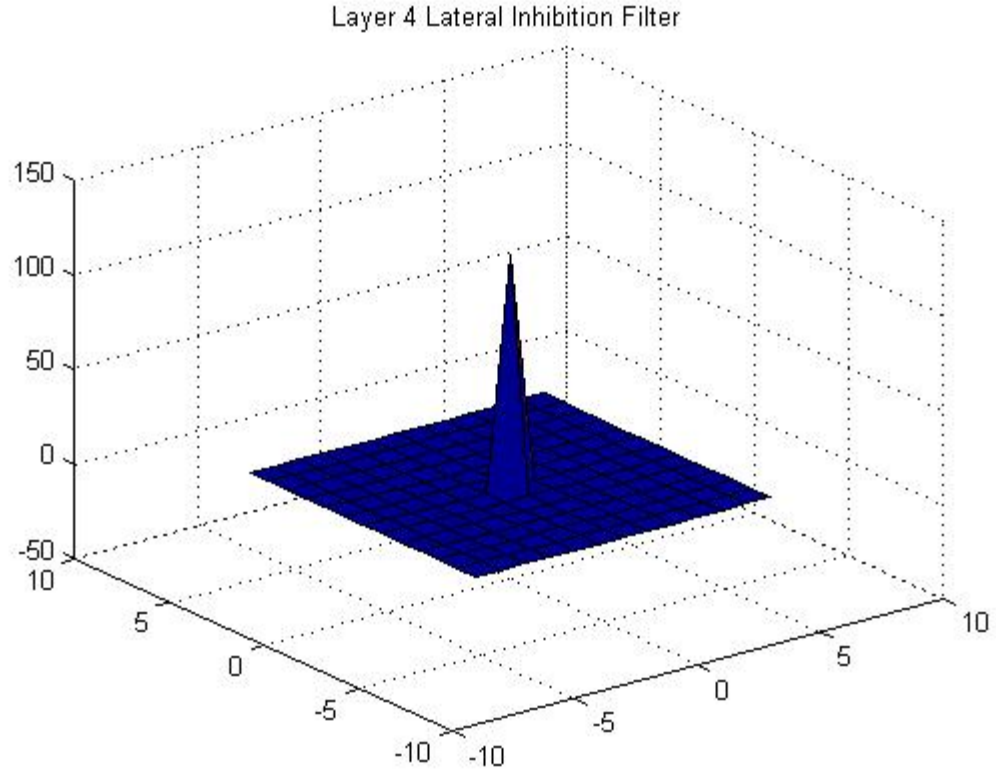

Figure 6.3d. Lateral inhibition filters used for the fourth layer. The pair σ and δ is 6.0 and 1.4

6.6 Output Calculation

Once the lateral inhibition filter is applied to a layer, a contrast enhancement function or sigmoid function is used to scale the values between 0 and 1. The sigmoid function is implemented so that it allows what percentage of the neurons should fire, explicitly setting the sparseness rate. The sigmoid function used is:

$$y = f^{\text{sigmoid}}(r) = \frac{1}{1 + e^{-2\beta(r-\alpha)}},$$

Equation 8

where r is the activation (or firing rate) after lateral inhibition, y is the firing rate after contrast enhancement, α is the function threshold and β is the slope. The α and β are held constant throughout each of the layer. The α value determines the percentile point for the activation of the layer. So for example, if the percentile is set at 95, then the neurons corresponding to the 95th percentile and higher are activated, i.e. top 5% of the neurons are activated. This is an explicit way of controlling the sparseness of the neurons. The values for α and β are 99.2 and 190 for layer 1, 98 and 40 for layer 2, 88 and 75 for layer 3, and 91 and 26 for layer 4.

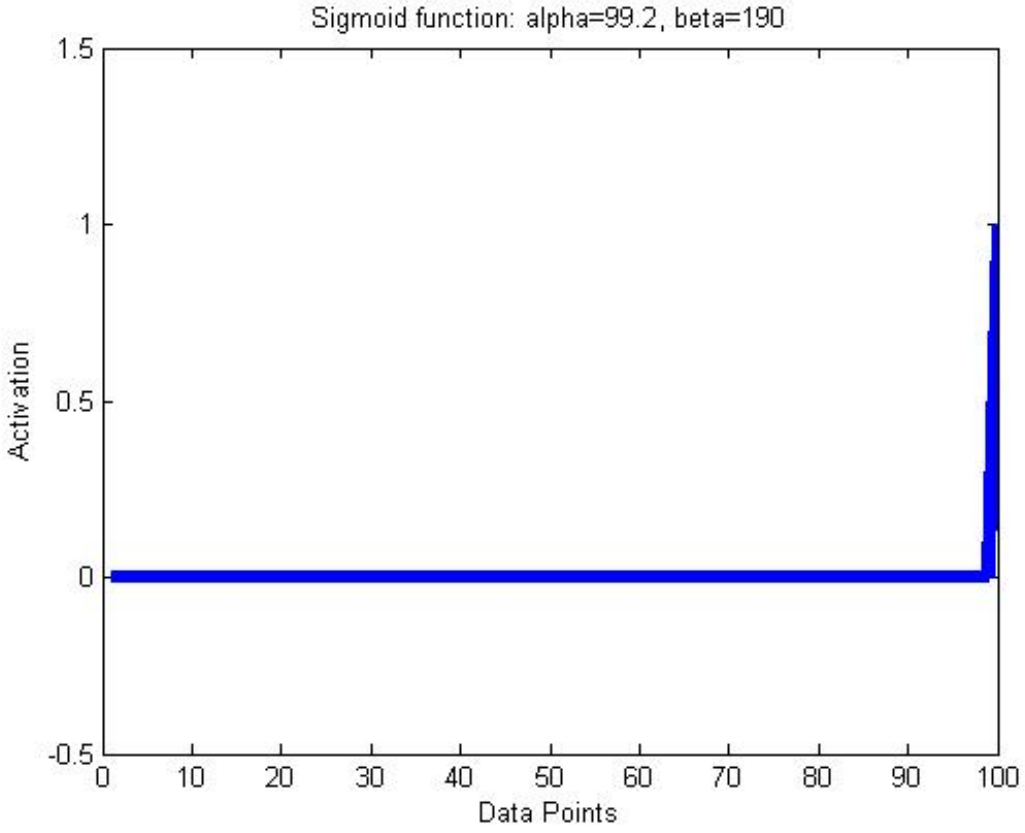

Figure 6.4a Sigmoid Activation Function for Layer 1

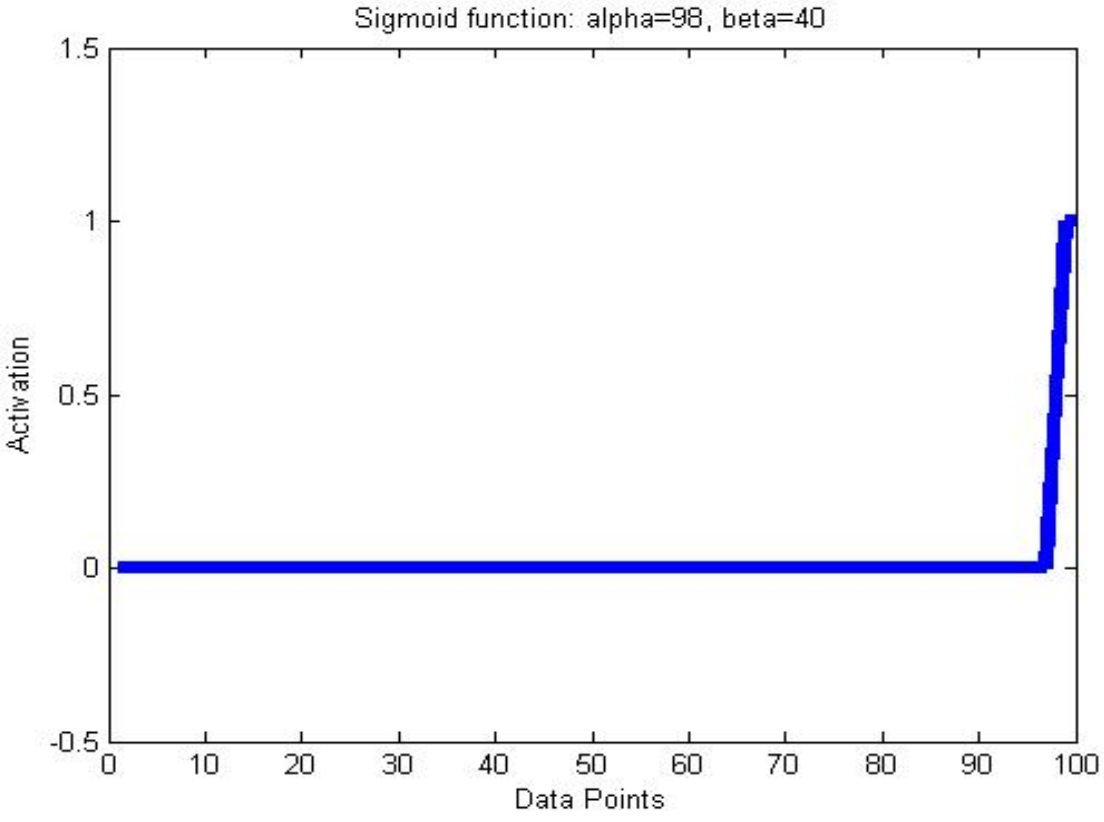

Figure 6.4b Sigmoid Activation Function for Layer 2

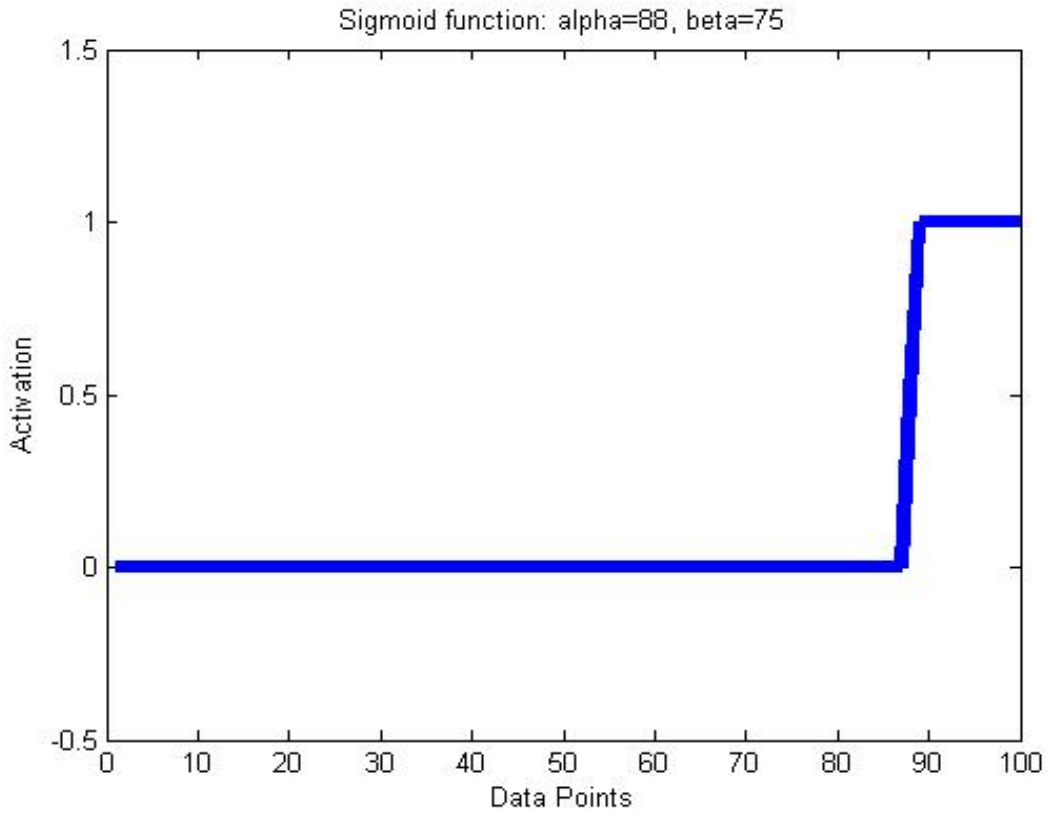

Figure 6.4c Sigmoid Activation Function for Layer 3

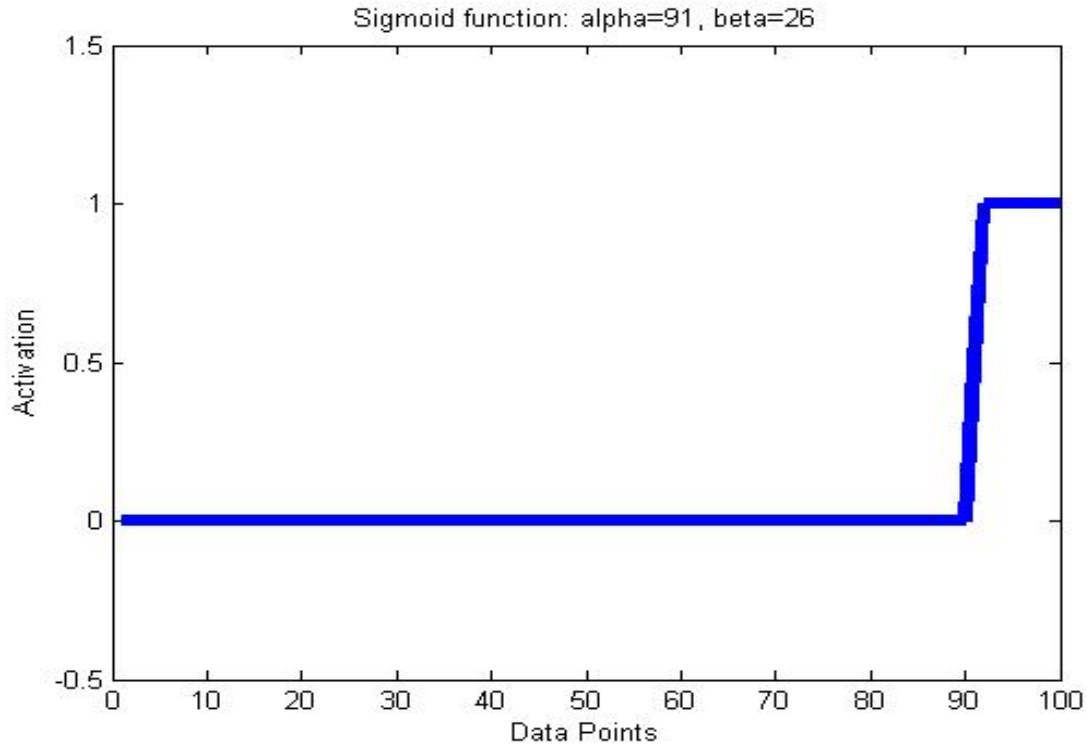

Figure 6.4d Sigmoid Activation Function for Layer 4

Thus, the contrast enhancement function determines whether the neuron should fire or not, depending on the local competition from the previous stage. The resulting values are then passed to the adjacent layer and the process is repeated for each of the layers in the network. The figures in 6.4 illustrate the idea of contrast enhancement function. For all the figures, it looks at data values ranging from 0 to 100 where no value is repeated. Thus, if the alpha value is set at 99.2, top .8% of the neurons will be set to 1. As a result, values that are greater than 99.2 are set to 1 and all the rest set to 0.

6.7 Training

Training of the network is implemented layer by layer. First, the bottom most layer is trained for a defined number of epochs. Followed by layer 2, 3, then 4. The idea behind this is that there is no sense in training layers in the upper layer if the bottom layers are still being modified.

The trace learning rule to train this model is a modified Hebbian learning rule where it incorporates traces of previous inputs in to the current calculation. The basis for this is that when we look at something for a very short period of time, say 0.5 seconds, it is highly likely that we'd looking at the same object for that time frame, maybe from different angles, rather than having different inputs being presented to us consistently. So this idea is implemented using a trace learning rule. The trace learning rule basically takes the information about the previously seen inputs when calculating

the weight changes for the current connection. This way, the network is able to learn invariant information about the input object in the way the network is trained.

The original trace learning rule in [Wallis 1996] is in equation 9.

$$\Delta w_j = \alpha \bar{y}^{-t} x_j^t$$

$$\bar{y}^t = (1 - \eta) y^t + \eta \bar{y}^{t-1} \qquad \text{Equation 9}$$

where x_j is the jth input to the neuron, y is the output from the neuron, $\bar{y}^t$ is the trace value of the output of the neuron at time step t, α is the learning rate, w_j is the synaptic weight between jth input and the neuron and η is the trace value. The learning is annealed between 0 and 1 using the cosine function. The trace value is set between 0 and 1. The optimal trace value is dependent on the presentation of the sequence length.

$$\Delta w_j = \alpha \bar{y}^{-t-1} x_j^t \qquad \text{Equation 10}$$

In [Rolls and Milward 2000; Rolls and Stringer2001], showed that the trace learning rule can be improved by including a trace value of activity from only the previous time step. Thus the firing rate of the current stimulus has no contribution to the current weight change. The modified rule is given in Equation 10.

During the training phase, the weights are updated using this trace learning rule. The trace learning rule allows the network to learn about the natural transformation of the objects [Wallis 1996]. As described in [Foldiak 1991; Wallis and Rolls 1997; Rolls 1994], the cells in the network can learn to respond to different transformations of the

same object by presenting consistent sequences of the target object in different transformations.

The network makes use of this rule by being presented with sequence of images that vary in either position, scale, rotation, or in any combination of them. The original network in [Wallis and Rolls 1997] was tested using simple objects such as T, L, and +. Each of the stimuli was swept across the input space in nine different positions. One training epoch consisted of presenting all stimuli across all nine positions. By training the network this way along with trace learning rule, it is able to learn about the different transformation of the object.

For each presentation of the stimulus, the weights are updated using the trace learning rule. The weight vectors are then normalized to limit the growth of the dendritic weight vectors. For each presentation, network adjusts its weights layer by layer. Thus, the first layer is trained with all the sequence of stimuli for given number of epochs. Once the layer's trained and weights are fixed, the second layer is then trained. This process is repeated for third and fourth layer.

6.8 Evaluation of the network

The performance of the network is evaluated differently than described in the original papers [Wallis and Rolls 1997; Rolls and Milward 2000] and rather trivial. The

evaluation is based on the invariant cell information at the fourth layer. When the network is trained, the set of neurons in the fourth layer becomes invariant to a particular stimulus. If the network was trained with location invariance in mind, the invariant cells will respond to trained stimulus regardless of its location in the input space (this will depend on how the network was trained).

To evaluate the network, first the positions of the invariant cells are noted for the particular stimulus. So, if the network was trained using 1 human face, 1 cat face, and 1 dog face, cells that respond to all of the human face location are noted, cells responding to all of cat face locations are noted, and cells responding to all of dog face locations are noted. Then the network is presented with a different stimulus that it was trained with and the responses of the invariant cells are noted. Thus, if we're interested in evaluating the human face stimulus, then the network is presented with dog face and the cat face at all the locations it was trained with. When the network is trained properly, the invariant cells should not respond to any other stimuli except for the one it was trained with. Ideally, the invariant cells noted for human face should not respond to any other stimuli except for the human face stimulus. Then this process is repeated for all other stimuli.

The performance is measured by looking at how these invariant cells respond to other stimuli. If the invariant cells respond to other stimuli other than the one they were trained with, then the number of such cells is recorded. This process is again repeated for all other stimuli. This can give us an estimate of how well the network performs.

7. Simulations

The model described in Section 6 is tested extensively using sets of variety of images. The model will be tested for invariance in translation and scale. Then it will be tested to see if the model is able to respond using different categories.

7.1 Translation

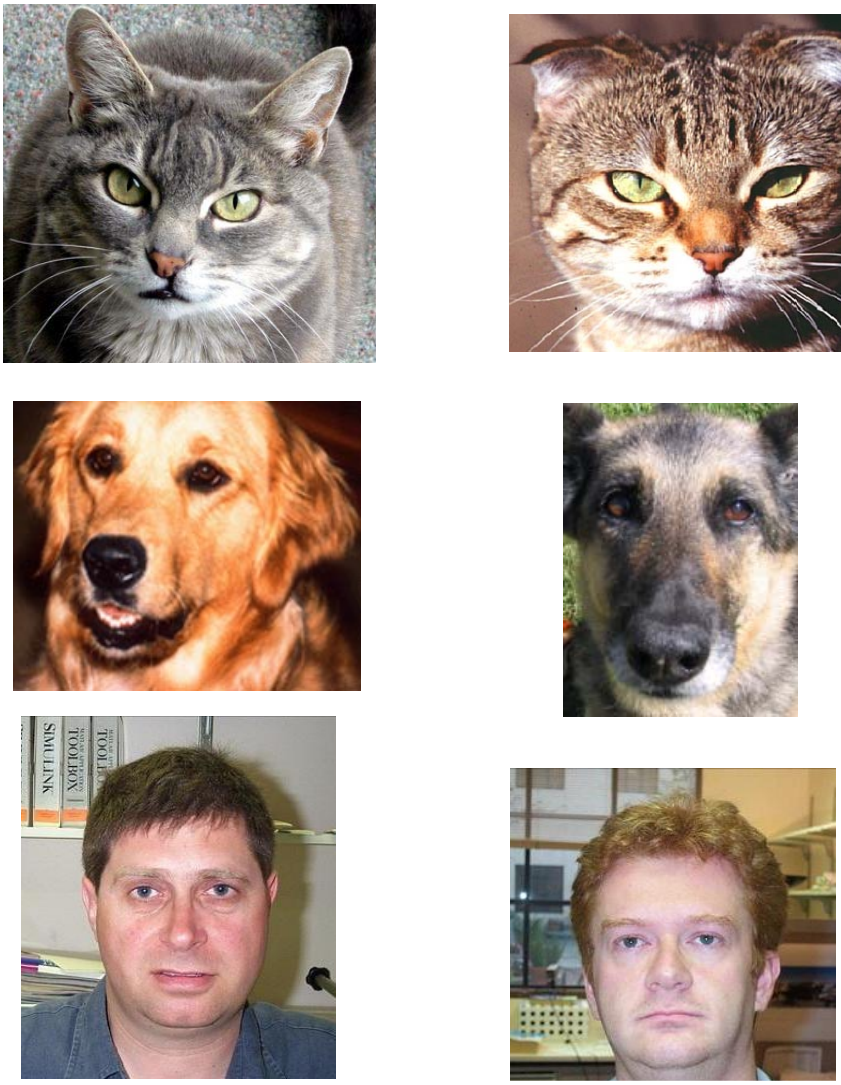

Figure 7.1 Stimuli used for translation simulation. Each image is rescaled to fit the input space and are converted to grayscale.

In the original paper [Wallis and Rolls 1997], the model was tested with stimuli that varied in position in the input space. The stimuli used for the initial experiment was T, L, and + and seven different human faces. The stimulus was moved across the input space covering different position, testing to see if the network is able to learn about the locations of the stimulus.

For this simulation, the 6 different stimuli were used: 2 cat faces, 2 dog faces, and 2 human faces (Figure 7.1). Each of the stimuli is 64x64 in dimension. The input space is divided into 9 even segments. So, for each stimulus, the network is presented with 9 different locations (figure 7.2). Presentation of all the stimuli at all the locations is considered one epoch. Each layer was trained for 300 epochs.

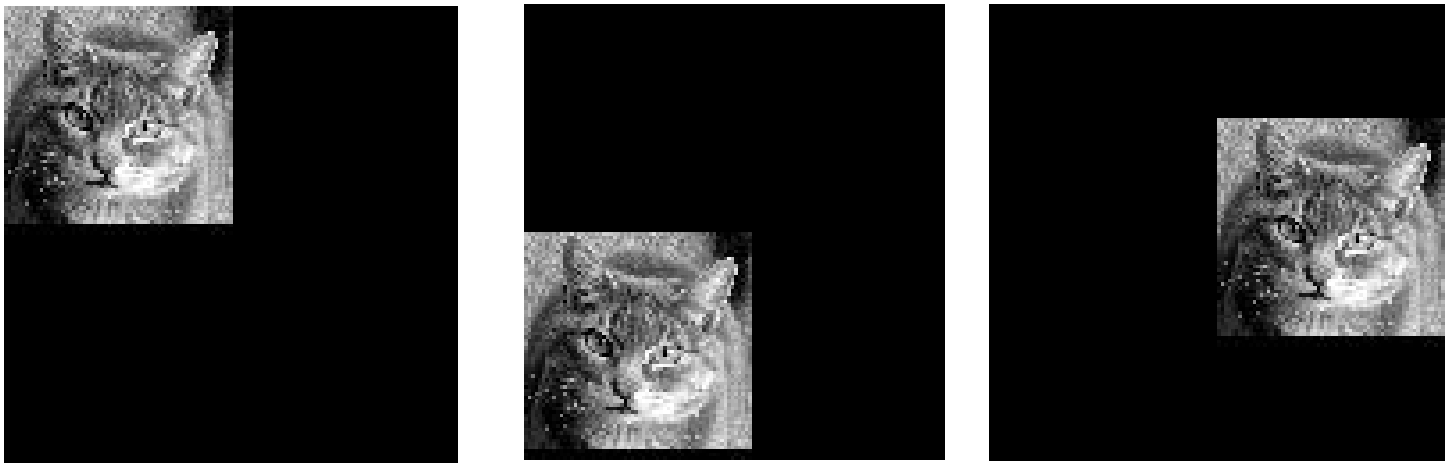

Figure 7.2 Different locations in the input space. The stimulus is presented with random location in the input space during the training phase.

Once the network is trained, it is presented with the inputs that it was trained with and is evaluated based on the number of neurons that fire maximally for one particular stimulus and minimally for all others. The network contains four layers, thus each input is presented to the network and the responses of the fourth layer are used to evaluate the performance.

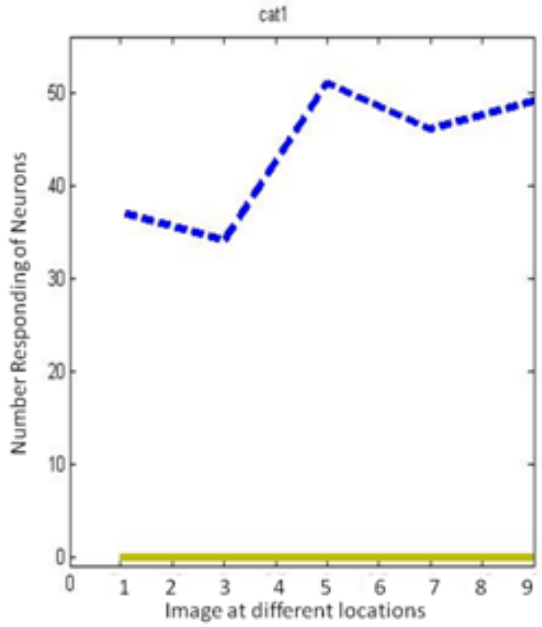

Figure 7.3a Cat1

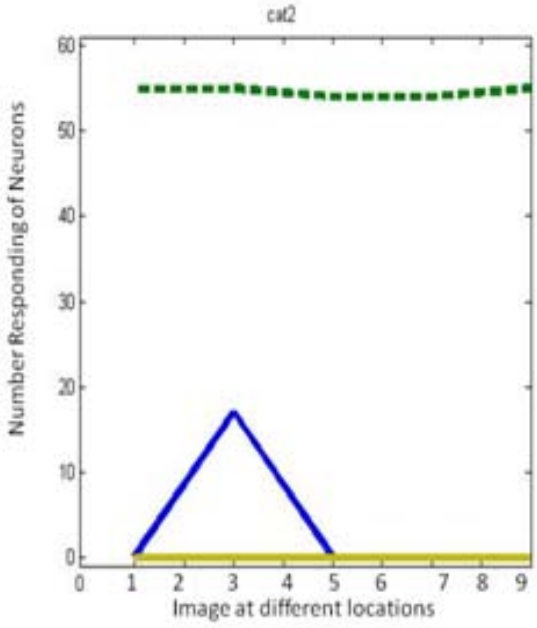

Figure 7.3b Cat2

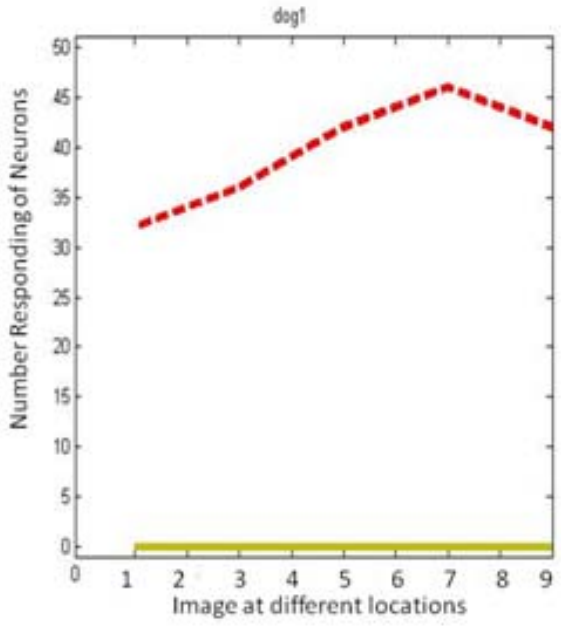

Figure 7.3c Dog1

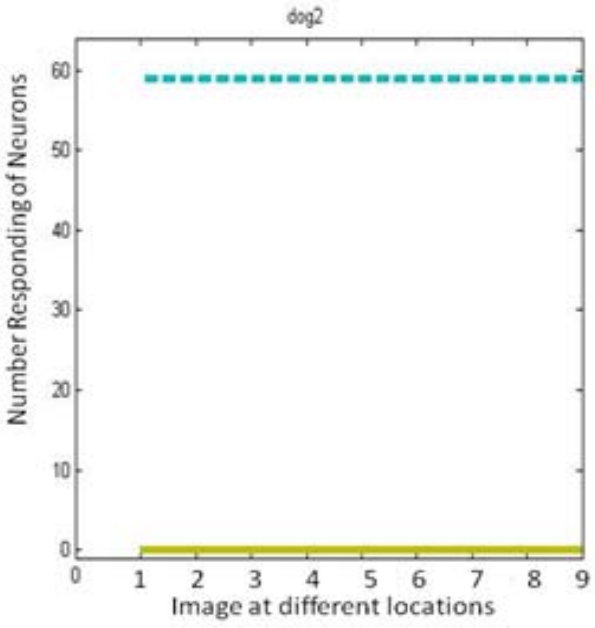

Figure 7.3d Dog2

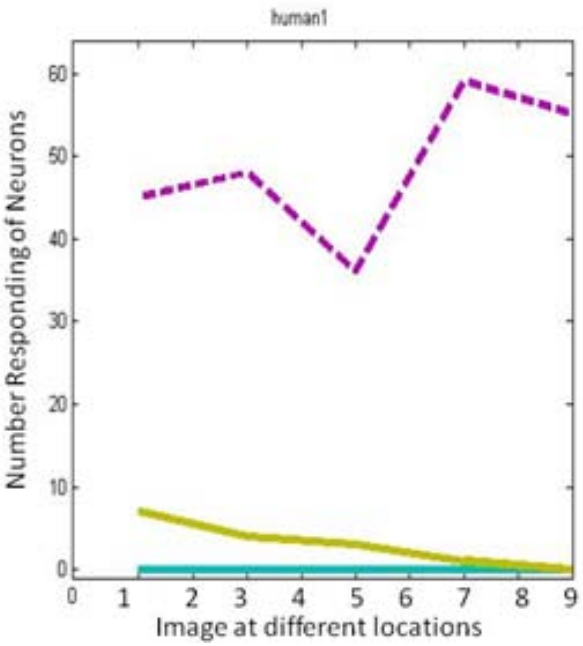

Figure 7.3e Human1

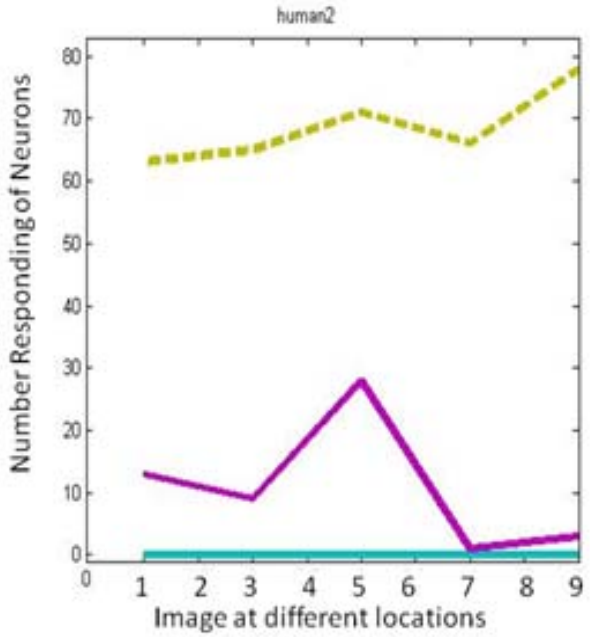

Figure 7.3f Human2

Figure 7.3. The plot for results of simulation. The invariant cells for each stimulus is compared against all the other stimuli to see how the network responds. Figure a,b, and c are for the three cat stimuli, Figures d e and f are for dog stimuli and figures g h and I are for the human face stimuli.

Figure 7.3 contains the plots for the results of this simulation. Each plot corresponds to the target stimulus compared to all other stimuli. Based on the results, the target stimulus has the highest response rate than the other stimuli. Thus, by thresholding it at the right level, it is feasible to classify different views of the stimulus as belong to one particular stimulus.

7.2 Scales

Humans are able to distinguish between different objects regardless of the size of the object, depending on how far away the object is from the viewer. Thus, the network is trained to respond to objects regardless of its size.

To achieve this, the network is trained using the same set of inputs as in section 7.1, but the network is presented in different scales. So, for each input, it is scaled to be half its original size and moved across the retina.

First, the network is trained using the same inputs (2 cat faces, 2 dog faces, and 2 human faces) as in Section 7.1, then the input is scaled to be half its size, and presented to the network in sequence across the input space. This is done for all the 6 faces (2 cat faces, 2 dog faces, and 2 human faces) and this represents one training epoch. Scaled samples of the input is listed in Figure 7.4.

Figure 7.4. Sample inputs to the image. The input consists of original inputs in Section 7.1 plus scaled versions of the inputs.

So the network is trained with each of the input in two different scales. By presenting the input in different scales, the network can learn to respond regardless of the size of the input. The result for this simulation is listed in Figure 7.5.

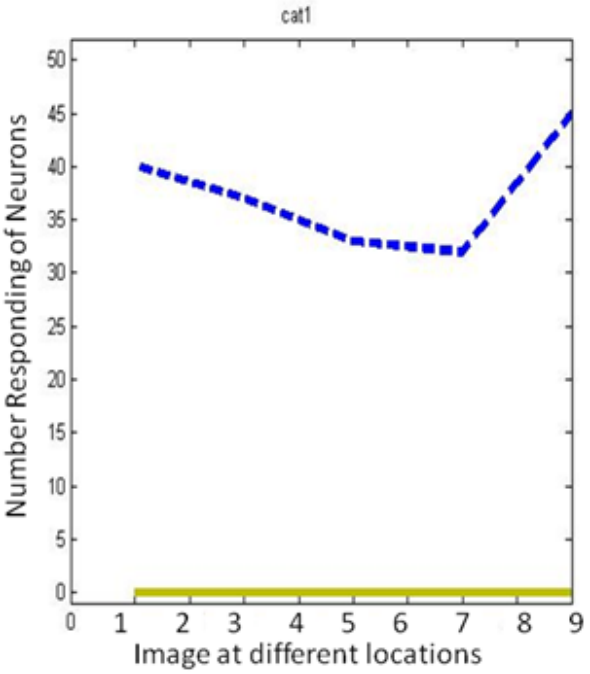

Figure 7.5a Cat1

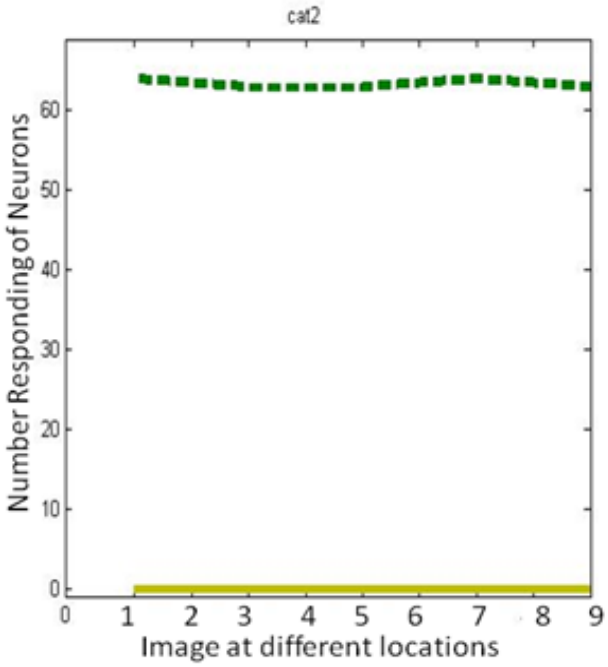

Figure 7.5b Cat2

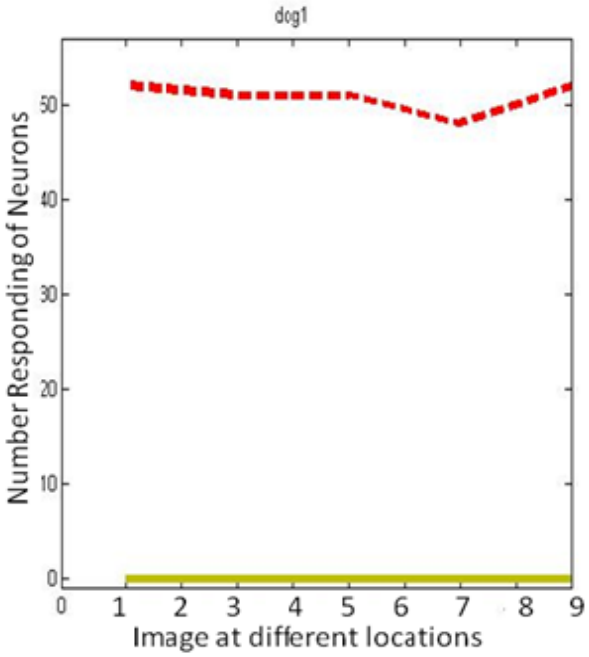

Figure 7.5c Dog1

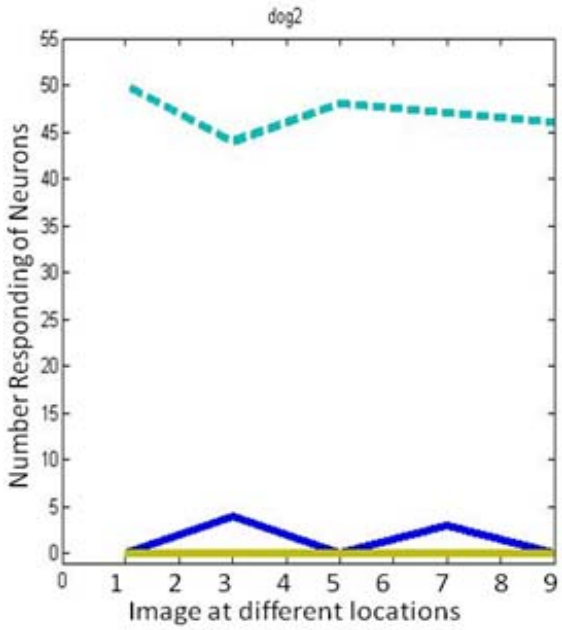

Figure 7.5d Dog2

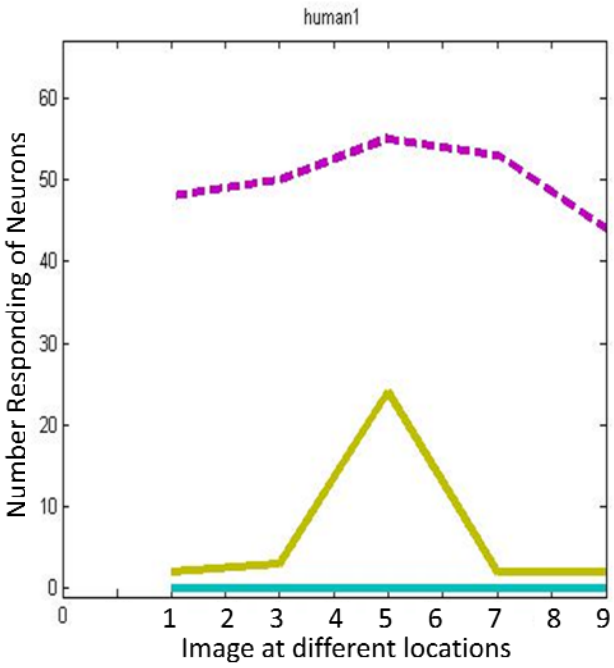

Figure 7.5e Human1

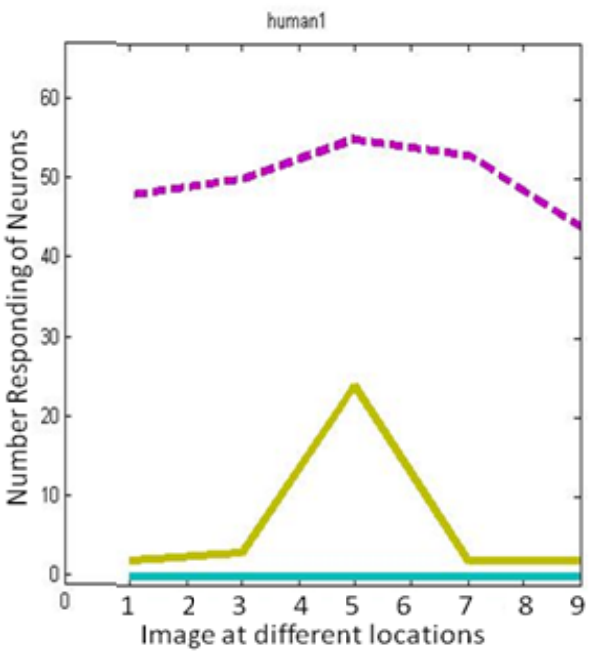

Figure 7.5f Human2

Figure 7.5 The plots for results of training the network with different scales and different locations.

Based on the results in Figure 7.5, the neurons at the fourth layer were trained to learn to respond to different scales of the input. Thus the network can respond to different scales of the input if it is trained with them. This simulation was done to illustrate that the network is capable of training the neurons to respond in certain ways in different situations. This isn't practical, however, since the network must be trained with different scales of the input to recognize it.

7.3 Categories

Looking at the results so far, the network is able to learn by being presented with sequence of images, containing the same object but different views of it. On the other hand, if the network is presented with sequence of images that are within the same

category, then it could be possible to have the network to respond to the categories (obtained from *Caltech101*) rather than specific objects. For the simulations in this section, network is trained with various categories. For the first simulation, the network is trained with two categories: cat face and non-cat faces. Then the network is presented with novel images of cat faces to figure out how the network behaves in the presence of images that it has never seen before. Then the network is trained with dog faces, cat faces, and non-related images to figure out its performance with a slightly larger input. For the last simulation, the network is trained with multiple categories to see the networks versatility.

7.3a 2 Categories: Cat faces and Non-Cat faces

The network is presented with 50 cat faces and 100 non-cat faces. Once the network is trained, the network is presented with 50 novel cat faces and several other non-cat faces that the network wasn't trained with. The idea behind this simulation is to figure out if the network can be used to recognize novel inputs of the same category that it was trained with. Thus by using the trace learning rule, instead of learning about the different views of the same object, it might be possible to train the network to respond similarly to a set of images that are in the same category. If this can be done, then the network can be further modified to be incorporated in a real world application. Sample images of the cat faces are shown in figure 7.6 and some images of non-cat faces are shown in figure 7.7.

Figure 7.6 There are two categories used for this simulation. The two categories are cat faces and dog faces.

Figure 7.7 Sample input used. There are 10 categories in all ranging from airplane to guitar to strawberry. There are 5 sample per category each slightly different from the rest.

Once trained, the responses of the network with presentation of novel inputs are recorded. The evaluation of the network is same as described in 6.8. The common fourth layer neurons are noted and their firing rate is used to determine the responsiveness of the network. The results are described in Table 7.1.

Category	Percentage
Cat	98% (56% Dog)
Accordion	0%
Panda	10.53%
Crayfish	4.29%
Brontosaurus	4.65%
Brain	2.04%
Platypus	11.76%
Hawksbill	6%
Dragonfly	14.71%
Flamingo Head	4.44%
Butterfly	10.98%

Category	Percentage
Kangaroo	6.98%
Pyramid	10.53%
Chandelier	10.28%
Motorbikes	2.76%
Bonsai	9.38%
Snoopy	8.57%
Garfield	29.41%
Tick	12.24%
Strawberry	8.57%
Stapler	24.44%
Average (False)	9.68%

Table 7.1 Experiment Results. The target object is cat faces.

As it can be seen in table 7.1, it is possible to train the model to recognize novel images. The trained network is presented with novel cat faces and its response rate was high at 98% of the time, but classified 56% of dog faces as dog faces. Meanwhile, the responses for non-cat face images were relatively low, though there were some false positives. The average false positive for this simulation was at 9.68% when the network was presented with inputs of different categories that the model hasn't seen before. However, when presented with inputs with similar features, such as dog faces, the model does not respond well to them. The dog faces have common high-responding features as cats, such as the eyes and the ears. These features might be enough to result in high firing rate of the dog face input to the model. The responses of the first layer of the model for some dog faces and cat faces are shown in figure 7.8. One way to

get around this is to train the network with both cat faces and dog faces so that it learns about the discriminating factors between the two sets. This idea is explored further in Section 7.3b.

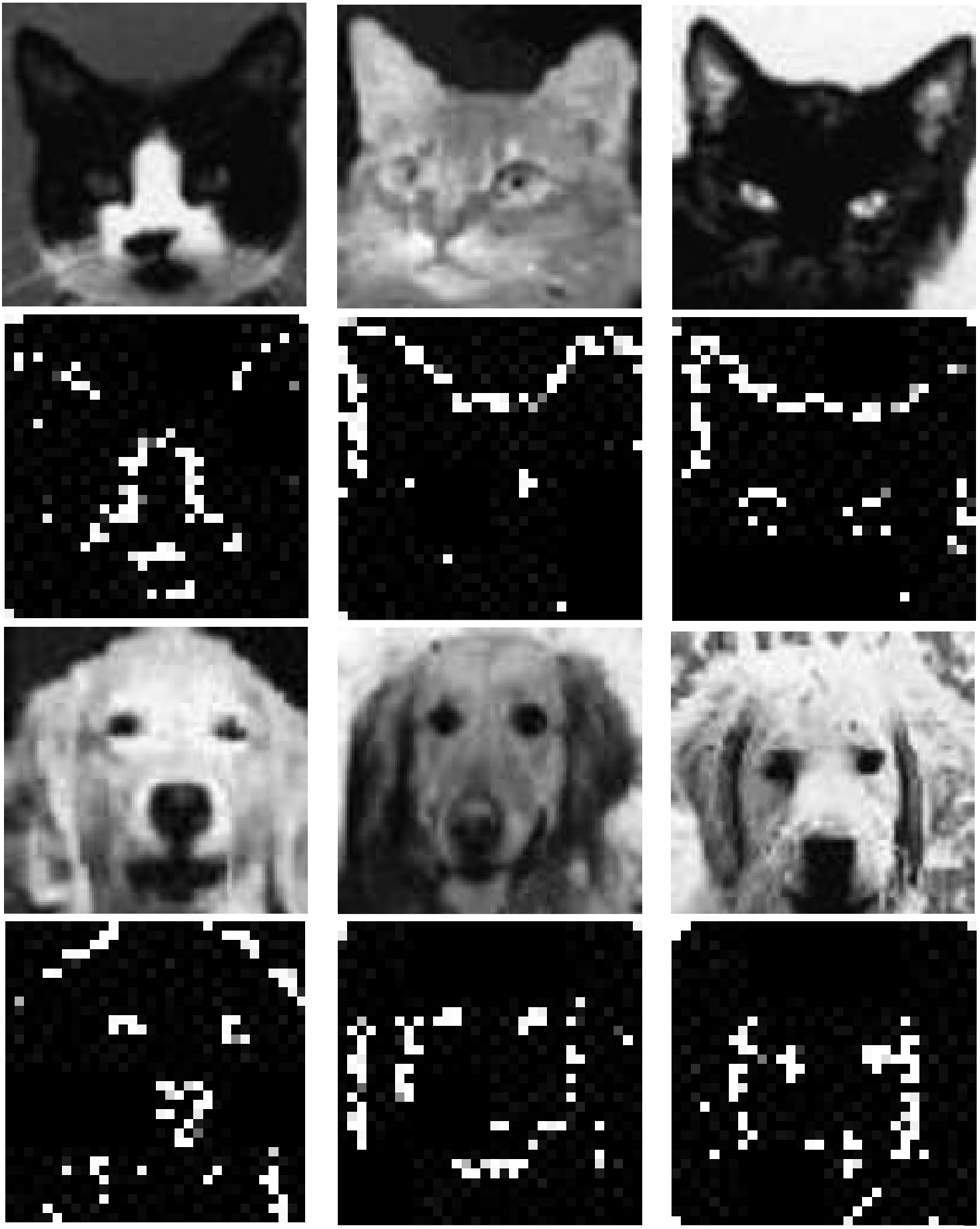

Figure 7.8 Example of Cat and Dog faces and their corresponding layer 1 output.

7.3b Cat Faces, Dog Faces, and Various Categories

In 7.3a, the results show that it's feasible to train the network to recognize objects in the same category, while responding minimally to different objects. In this simulation, the network is trained to recognize cat faces as well as dog faces. Thus, 50 cat faces and 50 dog faces are presented to the network and the network is trained to respond highly to cat faces and dog faces. Then the network is presented with 50 novel cat faces and 50 novel dog faces and 20 different categories of objects and the results are recorded in Table 7.2.

Category	Percentage
Cat	99% (14% Dog)
Dog	87% (12% Cat)
Platypus	8.82%
Cellphone	1.69%
Cougar Body	2.13%
Emu	7.55%
Octupus	14.29%
Camera	6%
Gramophone	7.84%
Menorah	9.82%
Dragonfly	2.94%
Dalamatian	1.49%

Category	Percentage
Cup	1.75%
Tick	4.08%
Leopards	0%
Headphone	4.76%
Stop Sign	6.25%
Scorpion	1.19%
Pizza	0%
Brain	2.04%
Starfish	13.95%
Ketch	1.75%
Average	4.92%

Table 7.2 Experiment Results. The target objects are cat faces and dog

In Section 7.3a, it was shown that the network responded poorly to novel inputs that contain somewhat similar features as the trained set of images, such as cat faces vs dog faces. The problem arises due to the quite a few similarities between the two set of images. This can be overcome, however, by training the network with both set of images and training the network to respond differently to the two set of images. The results show, that the network performed much better when trained with both cat and

dog faces when presented with novel cat and dog faces, than having the network be trained with just the cat faces. The network was able to classify novel cat faces 99% of the time, while classifying dog faces as cat faces 14% of the time, and classified novel dog faces correctly 87% of the time, while classifying cat faces as dog faces 12% of the time. It is a significant improvement from the previous experiment. Also, the false positive rate for other non cat/dog faces dropped at an average of 4.92%.

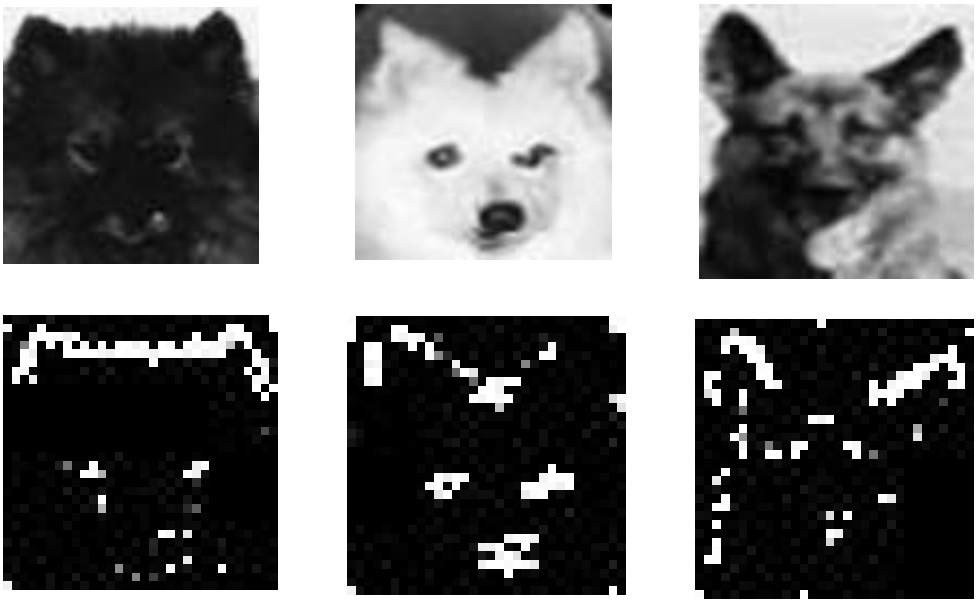

Figure 7.9 False Positive results for dog faces. Some of the dog faces have similar features as cat faces that result in them being classified as cat faces.

Figure 7.9 illustrates why some of the dog faces were classified as cat faces. As it can be seen in the figure, some of the dog faces have similar features as cat faces (i.e. pointy ears) thus the output of these faces may match closer to the cat invariance matrix rather than the dog invariant matrix resulting in false positive classification.

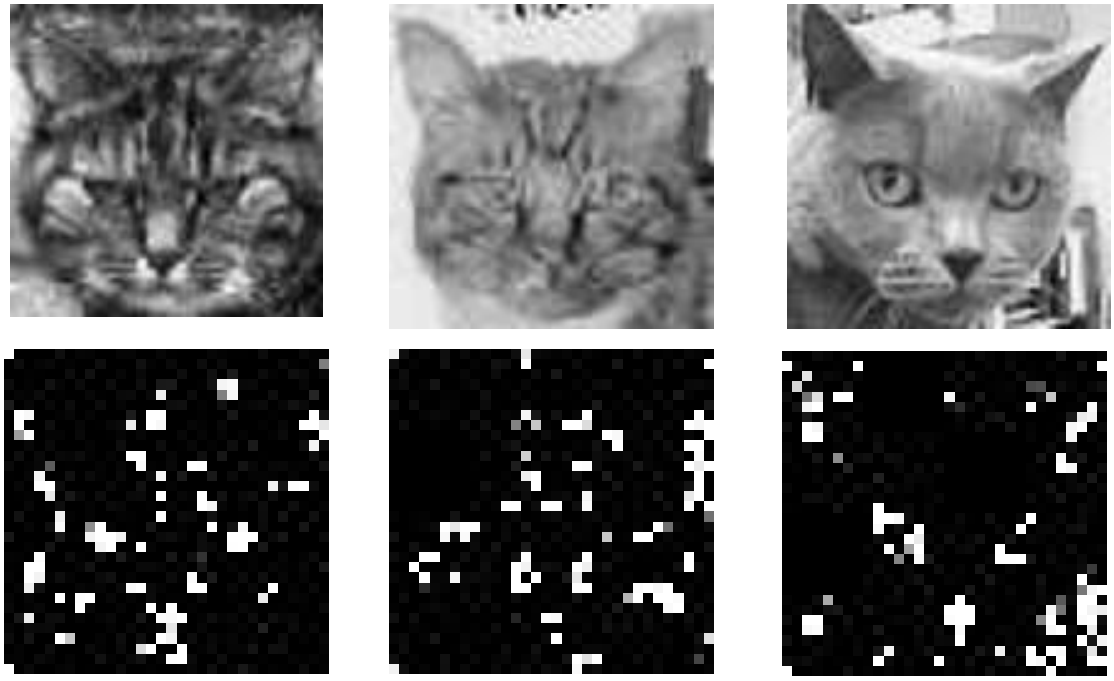

Figure 7.10 Some of the cat faces that were misclassified. Features extracted from these cat faces are rather ambiguous. No distinct features are extracted from the cats which can result in being classified as both cat face and dog face.

In figure 7.10, some of the cat faces that were classified as both cat face and dog face are illustrated. The features extracted from the cat faces are not very distinct. The ambiguity of the features extracted from them could match either the cat invariant matrix or the dog invariant matrix. The false positives in figure 7.9 and figure 7.10 might be alleviated by using better filters so that you are guaranteed to extract important features from the object presented to the network.

The features of different images in the same category are learned by the model using the trace learning rule. The trace learning rule allowed the model to take information about the different images in the same category into account while updating the weights for the current image. This trains the model to be tolerant to small changes to the input, thus allowing the model to respond highly to inputs of same type.

7.3c 5 Categories with 50 Samples per Category

In this simulation, the network is presented with sequence of images that belong to a same category. There are five categories all together. The samples of the categories shown are illustrated in Figure 7.11.

Figure 7.11 Categories of objects used to train the network. About 50 images per category are used to trained the network. Each category contains between 100 – 800 images.

The network is trained with 5 different categories: Airplanes, Car Sides, Bonsai, Watches, and Motorbikes. There are approximately 50 images per category in the training set. The network is presented with all the images in the training set per epoch. Once trained, the network is presented with novel images of that category. Number of images that are in the test set varies from 50 to 800. The input values range from 0 (dark) and 1 (light). The network's output values are between 0 and 1 as well, i.e. either the neuron fires or doesn't fire. Thus, for images with light background, the network

might consider the background pixels as one of the highest responding pixels and affect the weight updates. This is something we don't want. So, to prevent this from happening, images with light background are inverted (in grayscale) so that the background is darker (providing close to no contribution to the network) and the target object is lighter. One could also just extract the target object out of the image and provide a dark background for it, but this is a tedious task and just inverting the picture works for most cases. So for this simulation, the images are inverted and then fed into the network.

Once the network is trained, the network is presented with novel images and the network is evaluated. The primary goal of this test is to get an idea of how versatile the network is. The network was able to perform fairly well with just two categories as shown in Section 7.3b. If the number of categories is increased, then some of the invariant neurons in the fourth layer might overlap, since it's possible for different images of different categories to share similar features. The result of this simulation is shown in table 7.3.

Category/Percentage	Motorbikes	Airplanes	Bonsai	Car Side	Watch
Motorbikes	95.92%	78.85%	34.69%	64%	54%
Airplanes	95.92%	88.46%	40.82%	64%	54%
Bonsai	12.24%	30.77%	67.35%	4%	14%
Car Side	81.63%	46.15%	28.57%	60%	44%
Watch	93.88%	80.77%	32.65%	86%	78%

Table 7.3 Result of training 5 categories. The rows indicate the category it was trained with and the columns are the percentage of positive and false positive rates.

The results of the experiment are quite poor. The set of images in the categories vary slightly in position and in size. Because of this, the model is unable to produce accurate responses. The shift in the location may overlap responsive region of a different image in a different category. This would result in network responses to overlap, producing noisy results. Other possible reason for poor result might be that some of the target objects are embedded in a slightly cluttered scene. So far in the previous experiments, the model was trained with the target object centered in the input space. However, in this experiment that is not the case. The background scene might respond greater than the target object to the pre-processing filters. This results in a poorly trained model and leads to poor results. Possible solutions to the problems mentioned above are discussed briefly in Section 9.

8. Conclusion

As shown through experiments in Section 7, the model described in Section 6 can be used in variety of ways. The hierarchical structure of the model allows it to be versatile depending on the set of inputs that it is trained with. Through simulations, it was shown that the network can be used to train different translation and scales of the object simulating the invariant behaviors of the neurons in the upper levels of the ventral stream. The trace learning rule takes huge part in making the model work in an invariant manner. The learning rule can be further used to train the network with different categories of inputs. Instead of training the network with different transformations of a single object, it can be trained with a set of inputs that are in the same category. Thus, the network can be trained to respond to images that are in the same category. The results in Section 7.3 show that this is very feasible and the model responds pretty well to novel inputs.

This thesis revolves around the three recognition models that are built in a hierarchical structure simulating the divisions in the ventral stream. Each of the three models is quite different, yet they intend to achieve the same goal of simulating some mechanism in the visual cortex. Currently, no model can match the complexity and the efficiency of the human visual system. Thus, it seems natural to model the behaviors of human visual system in striving to come up with a robust and efficient object recognition system.

9. Future Work

The model described in Section 6 can be further modified to be used in a real world application. Through simulations, it has shown that it is possible to use the model to perform object recognition on a novel input. The model could be trained to recognize different aspects of a car and use it to recognize cars in a natural image. It could be also possible to extend the model so that it would recognize the target object anywhere in the input space, instead of being confined to locations it has seen before. Part of this could be implemented in the pre-processing layer. Instead of using just the Difference of Gaussian filters, the notion of simple and complex cells could be used so that it introduces some scale and position invariance before even being presented to the network. This could alleviate the problem discussed in Section 7.3c. The input images vary slightly in position and size. By using complex cells at the pre-processing layer, the effect of these slight variations might be mitigated. Also, if the image is contained in a cluttered scene, utility function could be implemented to simulate focus of attention to extract the target object that could be used to train and test the model.

In Section 7.3c, the model was overloaded with different types of categories. This resulted in the network to perform very poorly. This could be due to the overlapping responses throughout the layers. If this is resulted from different categories occupying the same space, then it is possible to increase the dimension of the layer and section it off to accommodate different types of categories. For example, if there are 6 categories, then the layers in the model could be divided into 6 sections and each section could be used to train a particular category. Once trained, novel inputs could be tested against different sections and responses of each section can be used to classify the novel input.

The hierarchical structure of the model allows it to be quite versatile. Additional layers could be added to further increase the size of the receptive field or to aid in interpretation of the responses in the fourth layer. These changes could substantially increase the robustness of this particular model.

Appendix: Matlab Code

setupNetwork.m

This function sets up the connection in the network. The connections are made based on the radius that is defined. The neurons in the first layer are distributed evenly over the input layer and receive connections randomly from a topologically similar area over a given radius. For the layers above it, it receives inputs from the preceding layer based on the radius given and the Gaussian probability.

```
function [ layers ] = setupNetwork(filterMethod)

connectionRadius = [5, 6, 9, 12];  %Radius of convergence of each layer
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

fprintf('Layer 1 Connection \n')

%Number of connections to a cell from different group of input responses
%based on spatial frequencie
if (filterMethod == 1)
    inputConnections = floor([8 13 50 201]);
else
    inputConnections = [201 50 13 8];
end

sizeLayer = 32;
layer1Connections = sum(inputConnections);
inputSize = 128;
layer1Steps = floor(inputSize/sizeLayer);
numScale = 4;

%The size of the input and the layers in the network are different. So, the
%neurons in the first layer is spread out evenly to cover the input space.
counterX = 1;
for i = 1:layer1Steps:inputSize
    fprintf('.')
    counterY = 1;
    for j = 1:layer1Steps:inputSize
        clear neuronConnections;
        clear neuronInputs;
        clear neuronWeights;
        inputCounter = 1;
        neuronConnections(1:layer1Connections) = 0;
        neuronInputs(1:layer1Connections) = 0;
        neuronWeights(1:layer1Connections) = 0;
        for k = 1:numScale
            counter = 0;
            %Connections from the input layer to the first layer in the
            %network are made randomly from a small region in the input
            %layer given the radius.
            while counter < inputConnections(k)
```

```
                if ( rand < .5 )
                    x = floor(rand*(connectionRadius(1)));
                else
                    x = -floor(rand*(connectionRadius(1)));
                end
                if ( rand < .5 )
                    y = floor(rand*(connectionRadius(1)));
                else
                    y = -floor(rand*(connectionRadius(1)));
                end

                connX = i + x;
                connY = j + y;
                %Wrap around if the connection trying to make is
                %out of range.
                if ( connX < 1 )
                    connX = mod(connX,inputSize);
                    if ( connX == 0 )
                        connX = inputSize;
                    end
                end
                if ( connY < 1 )
                    connY = mod(connY,inputSize);
                    if ( connY == 0 )
                        connY = inputSize;
                    end
                end
                if ( connX > inputSize )
                    connX = mod(connX , inputSize);
                    if ( connX == 0 )
                        connX = 1;
                    end
                end
                if ( connY > inputSize )
                    connY = mod(connY, inputSize);
                    if ( connY == 0 )
                        connY = 1;
                    end
                end

                f = ceil(rand*numScale*k);
                key = (connY-1)*inputSize+connX + (f-
1)*(inputSize*inputSize);

                if ( sum(neuronConnections == key ) == 0 )
                    neuronConnections(inputCounter) = key;
                    neuronWeights(inputCounter) = rand;
                    counter = counter + 1;
                    inputCounter = inputCounter + 1;
                end
            end
            layerValues(counterX, counterY).connections = neuronConnections;
            layerValues(counterX, counterY).weights =
normaliz(neuronWeights);
            layerValues(counterX, counterY).inputs = neuronInputs;
        end
```

```
        counterY = counterY + 1;
    end
    counterX = counterX + 1;
end

layer1.values = layerValues;
layer1.activation = ones(sizeLayer)*0;
layer1.output = ones(sizeLayer)*0;

%Keep a small history of trace values
layer1.trace(:,:,1) = ones(sizeLayer)*0;
layer1.trace(:,:,2) = ones(sizeLayer)*0;

layers(1) = layer1;
fprintf('\n')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Random connections for layers 2-4
for layerCounter = 2:4
    clear currentLayer;
    clear layerValues;
    conns = 150;
    fprintf('Layer %d Connection \n', layerCounter)

    for i = 1:sizeLayer
        fprintf('.')
        for j = 1:sizeLayer
            clear neuronConnections;
            clear neuronInputs;
            clear neuronWeights;
            counter = 0;
            neuronConnections(1:conns) = 0;
            neuronInputs(1:conns) = 0;
            neuronWeights(1:conns) = 0;
            inputCounter = 1;
            %Forward connections between adjacent layers are made
            %randomly from a small region based on the radius using a
            %gaussian distribution of probabilities wiith 67% of the
            %connections coming from the radius.
            while counter < conns
                 x = floor(randn*(connectionRadius(layerCounter)));
                 y = floor(randn*(connectionRadius(layerCounter)));
                 if ( rand < .5 )
                    %x = floor(rand*(connectionRadius(layerCounter)));
                else
                    %x = -floor(rand*(connectionRadius(layerCounter)));
                end
                if ( rand < .5 )
                    %y = floor(rand*(connectionRadius(layerCounter)));
                else
                    %y = -floor(rand*(connectionRadius(layerCounter)));
                end

                 connX = i + x;
                 connY = j + y;
```

```
                if ( connX < 1 )
                    connX = mod(connX, sizeLayer);
                    if ( connX == 0 )
                       connX = sizeLayer;
                   end
                end
                if ( connY < 1 )
                    connY = mod(connY, sizeLayer);
                    if ( connY == 0 )
                       connY = sizeLayer;
                   end
                end
                if ( connX > sizeLayer )
                    connX = mod(connX, sizeLayer);
                    if ( connX == 0 )
                       connX = 1;
                   end
                end
                if ( connY > sizeLayer )
                    connY = mod(connY, sizeLayer);
                    if ( connY == 0 )
                       connY = 1;
                   end
                end
                key = (connY-1)*sizeLayer + connX;
                if ( sum(neuronConnections == key ) == 0 )
                   neuronConnections(inputCounter) = key;
                   neuronWeights(inputCounter) = rand;
                   counter = counter + 1;
                   inputCounter = inputCounter + 1;
                end
            end
            layerValues(i, j).connections = neuronConnections;
            layerValues(i, j).weights = normaliz(neuronWeights);
            layerValues(i, j).inputs = neuronInputs;
        end
    end

    currentLayer.values = layerValues;
    currentLayer.activation = ones(sizeLayer)*0;
    currentLayer.output = ones(sizeLayer)*0;

    currentLayer.trace(:,:,1) = ones(sizeLayer)*0;
    currentLayer.trace(:,:,2) = ones(sizeLayer)*0;

    layers(layerCounter) = currentLayer;

    fprintf('\n')
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

trainNetowork.m

This script is called to start all the training of the network. It first sets up the network then obtains images used for training in a vector format. Then it trains one layer at a time with the given number of epochs per layer. Once the network is trained, the invariant cells in the fourth layer are obtained which will be used during simulation.

```
tic
clear

%There are two ways of pre-processing the input
%filterMethod = 1   Use DoG filters used for VisNet
%filterMethod = 2   Use Gabor filters used for Standard Model (But only
%looks at the 1st band

filterMethod = 1;

invertImage = true;

%Create the connections between the layers
layers = setupNetwork(filterMethod);
untrained = layers;

%Set the directory of training data
%trainRootDir = './train_categ_5';
%trainRootDir = './train_categ_10';
%trainRootDir = './train_trans';
%trainRootDir = './train_scale';
%trainRootDir = './train_catdog';
%%trainRootDir = './recognition_train';
%trainRootDir = './train_catdog_50';
trainRootDir = './train_3_categ';

trainDirs = dir(trainRootDir);
trainDirs = trainDirs(3:end);

%Go through all the images and go through the pre-processing stage and put
%the resulting outputs into a vector
fprintf('Reading Images\n')
for i = 1:length(trainDirs)
    fprintf('%d: Vectorizing %s  ', i, trainDirs(i).name);
    trainDir = [trainRootDir '/' trainDirs(i).name];
    imageRead{i} = readAllImages(trainDir);
    for j = 1:length(imageRead{i})
        fprintf('.')
        if (invertImage)
            centeredImage{i}{j} = centerImage(InvertIm(imageRead{i}{j}));
        else
            centeredImage{i}{j} = centerImage(imageRead{i}{j});
        end

        if (filterMethod == 1)
            vectImage{i}{j} = vectorizeImage(centerImage(imageRead{i}{j}));
```

```
            %vectImage{i}{j} = vectorizeImage(imageRead{i}{j});
        else
            vectImage{i}{j} = vectorizeImageC1(centerImage(imageRead{i}{j}));
            %vectImage{i}{j} = vectorizeImage(imageRead{i}{j});
        end
    end
    fprintf('\n')
end

%Set the trace rate value here
traceRate = .8;

%Pick which trace rule to use, 1 = only the previous trace, 2 = previous
%trace and current information.
traceRule = 1;

%Define how many epochs each layer should go through
epochs = [100 100 100 100];

numCategories = length(vectImage);

%This might be useful depending on what the network is being trained
%with... present the network with same input set (randomly within the set)
%as defined by the value per training epoch
numRepeat = 1;

%Train one layer at a time, starting with the bottom layer
for currentLayerNum = 1:4
    %Learning rate is annealed between 1 and 0 using the cos(0) to
    %cos(pi/2)
    learningRate = cosd(0:90/(epochs(currentLayerNum)-1):90);
    %learningRate = 1;

    for numEpochs = 1:epochs(currentLayerNum)
        fprintf ('Layer %d Epoch %d \n', currentLayerNum, numEpochs)
        for categ_index = randperm(numCategories)
            t = 0;
            images = vectImage{categ_index};
            sizeImages = length(images);
            for repeat = 1:numRepeat
                for index = randperm(sizeImages)
                    %If it is the bottom layer, then get the input
connections
                    %from the response filters for the image
                    if (currentLayerNum == 1)
                        previousLayer = images{index};
                        layers(currentLayerNum) =
updateLayer(layers(currentLayerNum), previousLayer, currentLayerNum,
learningRate(numEpochs), traceRate, traceRule, t);
                    else
                        for prevLayer = 1:currentLayerNum
                           if (prevLayer == 1)
                                previousLayer =
layerOutput(layers(prevLayer), images{index}, prevLayer);
                            elseif (prevLayer == currentLayerNum)
```

```
                                layers(prevLayer) = updateLayer(
layers(prevLayer), previousLayer(:)', prevLayer, learningRate(numEpochs),
traceRate, traceRule, t);
                            else
                                previousLayer = layerOutput(
layers(prevLayer), previousLayer(:)', prevLayer);
                            end
                        end
                    end
                    t = t + 1;
                end
            end
        end
    end
end

%Find out which neurons in the forth layer should belong to which stimulus
for i = 1:length(trainDirs)
    fprintf('%d Simulating %s: ', i, trainDirs(i).name);
    for j = 1:length(centeredImage{i})
        fprintf('.');
        currentResponse = simulate(layers, centeredImage{i}{j},4);
        responses{i}(:,:,j) = currentResponse(:,:,4);
    end
    layerSum{i} = sum(responses{i},3);
    percent = percentile(layerSum{i}(:),95);
    invarMatrix{i} = layerSum{i} >= percent;
    fprintf('\n');
end

trained = layers;
time = toc;
save('.\save\trained3', 'untrained', 'trained', 'centeredImage',
'invarMatrix');
```

updateLayer.m

During the training phase, the layers are updated accordingly in this function. First the activation rate is calculated, followed by lateral inhibition, and contrast enhancement. Depending on the trace learning rule used, the order of these may be different. At the end of the function, the weights are updated using the trace value calculated using the specified trace learning rule.

```
function [layer] = updateLayer( layer, previousLayer,
currentLayerNum,learningRate,...
                                    traceRate, traceRule, t)
%This function is used to calculate the weight changes during the training
%phase. The weights are modified based on the trace learning rule, where it
%incorporates the previous trace of the cell's activity to current cell's
%activation.
%Two different trace learning rule are implemented.
%One uses the previous trace and current information to calculate the
```

```
%change. The other uses only the previous information to calculate the
%change, thus current stimulus don't have any influence in the current weight
%change.

%parameter values
gamma = [1.38, 2.7, 4.0, 6.0];  %Radius for lateral inhibition filter
sigma = [1.5, 1.5, 1.6, 1.4];   %Contrast for lateral inhibition filter
alpha = [96, 95, 88, 91];     %Percentil for contrast enhancement function
beta = [190, 40, 75, 26];       %Slope for contrast enhancement function

layerValues = layer.values;

[x y] = size(layerValues);
%Calculate the activation rate for each neuron
for i = 1:x
    for j = 1:y
        %Retrieve the connections made to the layer from the previous layer
        neuronWeights = layerValues(i,j).weights;
        neuronConnections = layerValues(i,j).connections;
        inputs = previousLayer(neuronConnections(:));
        layerValues(i,j).inputs = inputs; %update input
        layer.activation(i,j) = sum(sum(inputs .* neuronWeights));
    end
end

layer.values = layerValues;

%currentTrace is based on the previous output and the trace rate at t - 2
if ( traceRule == 1)
    if ( t == 0 )
         currentTrace = ones(x).*0;
         layer.trace(:,:,t+1) = currentTrace;
    elseif ( t == 1)
        currentTrace = (1-traceRate) .* layer.output;
        layer.trace(:,:,t+1) = currentTrace;
    else
        t = mod(t,2) + 1;
        currentTrace = (1-traceRate) .* layer.output +
(traceRate).*layer.trace(:, :, t);
        layer.trace(:,:,t) = currentTrace;
    end
end

%currentLayerNum
layerInhibition = lateralInhibition(gamma(currentLayerNum),
sigma(currentLayerNum), layer.activation, x);
%layerInhibition = imfilter(layer.activation, lateral);

percent = percentile(layerInhibition(:), alpha(currentLayerNum));

%Current output
layer.output = contrastEnhancement(percent, beta(currentLayerNum),
layerInhibition);

if (traceRule == 2)
```

```
    if ( t == 0 )
        currentTrace = (1-traceRate) .* layer.output;
        layer.trace(:,:,1) = currentTrace;
    else
        currentTrace = (1-traceRate) .* layer.output + traceRate .*
layer.trace(:,:,1);
        layer.trace(:,:,1) = currentTrace;
    end
end


%Update the weight using the trace value and the learning rate (annealed
between unity and zero).
for i = 1:x
    %fprintf('_')
    for j = 1:y
        inputs = layerValues(i,j).inputs;
        weightChange = learningRate .* currentTrace(i,j) .* inputs;
        layerValues(i,j).weights = normaliz(layerValues(i,j).weights +
weightChange);
    end
end

layer.values = layerValues;
```

layerOutput.m

This function calculates the output of a given layer. The output is figured out using the activation function, lateral inhibition, and contrast enhancement function. The output of a layer is defined between values 0 and 1. The parameters for each layer are different and predefined in the function.

```
function [layerOutput] = layerOutput(layer, input, layerNum)
%This function is used to calculate the output of the layer with the
%current input. The input could come from the image filtered layer or any
%of the layers in the network.
%The resulting output is the firing rate of the neurons

gamma = [1.38, 2.7, 4.0, 6.0];  %Radius for lateral inhibition filter
sigma = [1.5, 1.5, 1.6, 1.4];   %Contrast for lateral inhibition filter
alpha = [96, 95, 88, 91];     %Percentile for contrast enhancement function
beta = [190, 40, 75, 26];       %Slope for contrast enhancement function

layerValues = layer.values;

[x y] = size(layerValues);
for i = 1:x
    for j = 1:y
        %Calculate the activation rate for each neuron
```

```
        neuronWeights = layerValues(i,j).weights;
        neuronConnections = layerValues(i,j).connections;
        connections = input(neuronConnections(:));
        outputMatrix(i,j) = sum(sum(connections .* neuronWeights));
    end
end

%Local competition using lateral inhibition
layerInhibition = lateralInhibition(gamma(layerNum), sigma(layerNum),
outputMatrix, x);
%Find the threshold value using the percentile value
percent = percentile(layerInhibition(:), alpha(layerNum));
%Simoid function to calculate the firing rate
layerOutput = contrastEnhancement(percent, beta(layerNum), layerInhibition);
```

inputFilter.m

The input filters used for this network is difference of Gaussian weighted by third Gaussian. This is calculated using three parameters: rho – sign of the filter, theta – orientation of the filter, and freq – spatial frequency of the filter.

```
function [filter] = inputFilter(rho, theta, freq)
%This function is used to build the filters that will be used for the input
%image.
%
%rho - sign of the filter
%theta - orientation of the filter
%freq - frequency of the filter

matrixSize = 13;    %Size of the filter

%The filter is calculated using different of Gaussian weighted by a third
%Gaussian.
for x = 1:matrixSize
    gaussX = x - ceil(matrixSize/2);
    for y = 1:matrixSize
        gaussY = y - ceil(matrixSize/2);
        firstExp = (gaussX*cos(theta)+gaussY*sin(theta))/(sqrt(2)/freq);
        firstGauss = exp(-1*(firstExp*firstExp));

        secondExp = (gaussX*cos(theta)+gaussY*sin(theta))/(1.6*sqrt(2)/freq);
        secondGauss = (1/1.6)*exp(-1*(secondExp*secondExp));

        thirdExp = (gaussX*sin(theta)-gaussY*cos(theta))/((3*sqrt(2))/freq);
        thirdGauss = rho.*exp(-1*(thirdExp*thirdExp));

        filter(x,y) = (firstGauss - secondGauss)*thirdGauss ;
    end
end
```

inputResponses.m

```
function [responses] = inputResponses( img )
%This function is used to apply the DoG filters to the image.
%It will go through all the filters different orientation and frequencies.
%There are four orientations: 0, 45, 90, 135 and
%four frequencies: 0.0625, 0.125, 0.25, 0.5.

frequencies = [0.0625, 0.125, 0.25, 0.5];    %Spatial frequency of the filter
rho = [1];
theta = [0, pi/4, pi/2, 3*pi/4];            %Orientation of the filter
counter = 1;
for x = 1:size(frequencies')
    for y = 1:size(theta')
        for z= 1:size(rho')
            responseFilter = inputFilter(rho(z),theta(y),frequencies(x));
            responses(:,:,counter) = imfilter(img, responseFilter);
            zeroThresh = responses(:,:,counter) > 0;
            responses(:,:,counter) = mat2gray(zeroThresh .*
responses(:,:,counter));
            counter = counter + 1;
        end
    end
end
```

lateralInhibition.m

This function is used to perform lateral inhibition. Once the activation rates are calculated, lateral inhibition is imposed to simulate local competition.

```
function [ ret ] = lateralInhibition(gamma, sigma, output, s)
%This function is used to perform local competition within the layer.
%First a filter is built using gamma(width) and sigma(contrast).
%Then the filter is applied to the layer simulating lateral inhibition.

centerValue = 0;
filter = ones(ceil(gamma)+1*2)*0;
radius = ceil(gamma);

%Build lateral inhibition filter
for i = -radius:radius
    for j=-radius:radius
        if ( i == 0 & j == 0 )
        else
            filter(i+radius+1,j+radius+1) = -sigma*(exp(-
1*(i^2+j^2)/gamma^2));
        end
    end
end
```

```
%Center value is 1 - sum(all other values)
filter(ceil(gamma)+1, ceil(gamma)+1) = 1 - sum(sum(filter));

%Applay the filter to the layer
padded = padarray(output, [radius,radius], 'circular');
filtPadded = imfilter(padded, filter);


ret = filtPadded(radius+1:radius+s, radius+1:radius+s);
```

contrastEnhancement.m

Contrast enhancement function scales the values resulting from lateral inhibition to be between 0 and 1.

```
function [ output ] = contrastEnhancement( alpha, beta, rate )
%Simple sigmoid function
%alpha is the percentile, beta is the slope of the function
%Rate is the data to use
output = 1 ./ (1 + exp(-2*beta*(rate-alpha)));
```

References

Barone P, Batardiere A, Knoblauch K., and Kennedy H. Laminar distribution of neurons in extrastriate areas projecting to visual areas V1 and V4 correlates with the hierarchical rank and dindicates the operation of a distance rule. J NeuroSci 20: 3263-3281, 2000.

Booth M, and Rolls E. View-invariant representations of familiar objects by neurons in the inferior temporal visual cortex. Cereb Cortex 8: 510-23, 1998.

Boynton G, and Hegde J. “Visual Cortex: The Continuing Puzzle of Area V2”. Current Biology, Vol. 14, July 13, 2004.

Bruce C, Desimone R, and Gross C. Visual properties of neurons in a polysensory area in the superior temporal sulcus of the macaque. J. Neurophys. 46, 369-384, 1981.

Caltect101. http://www.vision.caltech.edu/Image_Datasets/Caltech101/Caltech101.html.

Connor C, Preddie D, Gallant J, and Van Essen D, Spatial attention effects in macaque area V4. Journal of Neuroscience 17(9): 3201-3214, 1997.

Desimone R, Albright T, Gross C, and Bruce C. Stimulus-selective properties of inferior temporal neurons in the macaque. J Neurosci 4: 2051-62, 1984.

Elliffe M, Rolls E, and Stringer, S. Invariant recognition of feature combinations in the visual system. Biological Cybernetics 86: 59-71, 2002.

Felleman D, and Van Essen D. Distributed hierarchical processing in the primate cerebral cortex. Cereb Cortex 1: 1-47, 1991.

Foldiak, Peter. .Learning Invariance from Transformation Sequences. Neural Comput 3: 194-200, 1991.

Hebb, D. The Organization of Behavior. John Wiley, New York, NY, USA. 1949.

Hubel D, and Wiesel T. Receptive fields and functional architecture of monkey striate cortex. Journal Physiology, 1968, March; 215-43

Hung C, Kreiman G, Poggio T, and DiCarlo, J. Fast read-out object identity from macaque inferior temporal cortex. 2005

Jones J, and Palmer. An Evaluation of Two-Dimensional Gabor Filter Model of Simple Receptive Fields in Cat Striate Cortex. J. Neurophysiology, 58: 122301258, 1987.

Klopf, A. The Hedonistic Neuron: A theory of memory, learning, and intelligence. 1982.

Lee T, Mumford D, Romero R, and Lamme, V. The role of the primary visual cortex in higher level vision. Vision Research 38: 2429-2454, 1998.

Lerner Y, Hendler T, Ben-Basha D, Harel M, and Malach R. A hierarchical axis of object processing stages in the human visual cortex. Cereb Cortex 11: 287-297, 2001.

Mutch, J. and Lowe, D. G. Multiclass Object Recognition with Sparse, Localized Features. In *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 1*:June 17 - 22, 2006.

Pasupathy A, and Connor C. Shape Representation in Area V4: Position-Specific Tuning for Boundary Conformation. 2001.

Perrett D, and Oram M. Neurophysiology of shape processing. Img. Vis. Comput. 11, 317-333, 1993.

Potter, M. Meaning in Visual Search. Science, 187:565-566. 1975.

Reynolds J, Pasternak T, and Desimone R, Attention increases sensitivity of V4 neurons. Neuron 26(3): p. 703-714, 2000.

Riesenhuber M, and Poggio, T. Hierarchical models of object recognition in cortex 1999.

Riesenhuber M, and Poggio T. Computational Models of Object Recognition in Cortex: A Review. 2000.

Rolls E, and Milward, T. A Model of Invariant Object Recognition in the Visual System: Learning Rules, Activation Functions, Lateral Inhibition, and Information-Based Performance Measure. Neural Computation 12: 2547-2572, 2000.

Rolls E, Stringer S. Invariant object recognition in the visual system: learning rules, activation functions, lateral inhibition and information-based performance measures. Neural comput12: 2547-2572, 2001.

Rolls E. Brain mechanisms for invariant visual recognition and learning. Behav Proc 33: 113-138, 1994.

Serre T, Kouh M, Cadieu C, Knoblich U, Kreiman G, and Poggio T. A Theory of Object Recognition: Computations and Circuits in the Feedforward Path of the Ventral Stream in Primate Visual Cortex. 2005.

Serre T, Wolf L, and Poggio T. Object Recognition with Features Inspired by Visual Cortex. 2005.

Serre T, Wolf L, Bileschi S, Riesenhuber M, and Poggio T. Robust Object recognition with Cortex-Like Mechanisms. 2007.

Sutton R, and Barto A. Toward a modern theory of adaptive networks: Expectation and prediction. Psychol. Rev. 88, 135-170, 1981.

Tong F. Primary Visual Cortex and Visual Awareness. Nat Rev Neurosci: 4(3): 219-29. 2003

Tootell R, Hadjikhani N, Mendola J, Marrett S, and Dale, A. From Retinotopy to Recognition: fMRI in Human Visual Cortex. Trends in Cognitive Science – Vol. 2, No. 5, May 1998.

Treue S. Neural correlates of attention in primate visual cortex. Trends in Neurosciences, 24(5): 295-300, 2000.

Ungerleider L,and Haxby J. What and Where in the human brain. Op. Neurobiol. 4: 157-165, 1994.

Wallis G, and Rolls E. Invariant Face and Object Recognition in the Visual System. Progress in Neurobiology 51: 167-194, 1997.

Wallis, Guy. Using Spatio-Temporal Correlations to Learn Invariant Object Recognition. 1996

Yaginuma S, Osawa Y, Yamaguchi K, Iwai E. Differential functions of central and peripheral visual field representations in monkey prestriate cortex. In Brain Mechanisms of Perceptron and Memory: from Neuron to Behavior. 1993 : 1-33.