The Role of Prototype Learning in Hierarchical Models of Vision

Michael D. Thomure

December 2, 2013

ABSTRACT: I conduct a study of learning in HMAX-like models, which are hierarchical models of visual processing in biological vision systems. Such models compute a new representation for an image based on the similarity of image sub-parts to a number of specific patterns, called prototypes. Despite being a central piece of the overall model, the issue of choosing the best prototypes for a given task is still an open problem. I study this problem, and consider the best way to increase task performance while decreasing the computational costs of the model. This work broadens our understanding of HMAX and related hierarchical models as tools for theoretical neuroscience, while simultaneously increasing the utility of such models as applied computer vision systems.

Contents

Chapter 1

Introduction

Hierarchical models of vision have been suggested repeatedly in the computational neuroscience literature to describe the functional organization of visual processing in biological systems. One of the best known of these models is the HMAX system [1, 2], which has garnered interest both in neuroscience and computer vision communities. This model can be understood as addressing the object recognition problem, in which the system takes an image and responds with the type of object present in that image. As an example of this problem, an object recognition system may be presented with images of animals, and its task is then to determine the type of animal present in each image. Models such as HMAX tackle this problem by first computing a number of abstract features for the image, and then applying methods from statistical machine learning to choose the object class that best matches those features.

To compute features for an image, such systems apply a hierarchy of local pattern detectors at various locations and scales across the image. At the bottom of the hierarchy, each detector looks for a simple oriented edge, while at higher levels, a given detector looks for a specific pattern—or *prototype*—in the activation of lower-level detectors. The maximum activation for each of these abstract detectors is then used as the value of a single feature, and the combination of these feature values forms the new image representation. It is the top-level feature-value representation that is used by a trained classifier to determine the type of object present.

The choice of prototypes clearly plays an important role in the performance of the model. However, it is unclear how to best choose these prototypes, given a particular instance of the object recognition problem. One promising approach is to learn a useful set of prototypes automatically by finding statistical regularities in a set of example images. A simple example of this approach—called "imprinting"—has recently resulted in the model achieving competitive (though still far below human) performance on multiple computer vision benchmarks. This performance has led to the theory [2] that prototypes composed of imprinted shape are responsible for the model's success. Given the simplicity of imprinting, it seems probable that more sophisticated learning methods can achieve even better performance on these tasks.

The thesis advanced in this dissertation is that hierarchical visual models can be improved by learning prototypes. An investigation of this topic first requires knowing what performance the model is able to achieve without learning, in order to have an effective baseline when evaluating learning methods. Using this baseline, we can then measure the effect of learning using imprinting as well as other methods found in the literature. Finally, the benefit of using task-specific information during learning can be measured by introducing a new method that uses classifier feedback.

The contributions of this dissertation include the following.

- A novel framework is developed that allows the expression of a wide range of hierarchical visual models. This framework is used to construct a new visual model called Glimpse, which achieves competitive performance (Chapter 3).

- Common benchmark datasets are analyzed, and many are shown to be uninformative for object recognition research (Section 4.2).

- The benefit of imprinting is investigated, leading to the conclusion that imprinted shape is unnecessary to account for the model's success (Section 4.3). An alternative representation for object recognition based on random prototypes is introduced.

- A study is conducted on the use of feedback in prototype learning, where results show a significant increase in performance (Chapter 5).

- A more sophisticated learning technique—one that is commonly used in similar visual models—is also investigated (Chapter 6), with the discovery of important limitations.

- A new feedback-driven learning method is introduced in (Chapter 7), which is computationally efficient. The method is flexible enough to accept many forms of feedback information.

The rest of the dissertation is organized as follows. Chapter 2 provides background on the family of HMAX-like models used in this work, and discusses how they have been used in the literature. Chapter 3 introduces an HMAX-like model called Glimpse, which I developed for this dissertation. Chapter 4 provides an analysis of prototype learning, and discusses the role of shape in such prototypes.

Chapter 5 investigates a method known as feature selection, and shows how task information can be used to increase model performance. Chapter 6 analyzes an existing approach for prototype learning by using a machine learning method called clustering. Chapter 7 introduces a novel extension to clustering that allows task information to be used when learning prototypes. Finally, I present my conclusions in Chapter 8, and discuss future work in Chapter 9.

Chapter 2

Background & Prior Work

This chapter provides context for the dissertation, and begins by outlining the problem domain in Section 2.1. Section 2.2 describes the family of hierarchical models used in the dissertation. Section 2.3 explains how an important component of these models, called prototypes, are learned from image data. Finally, Section 2.4 describes the methodology that is commonly used to evaluate such models.

2.1 Object Recognition

This work considers the task of visual object recognition. Given a previously unseen image containing an unlabeled object, the task of object recognition is to predict what that object is. This is a difficult prediction task, as the appearance of an object can change greatly due to lighting conditions and the relationship between object and observer. Often the goal of object recognition is to recognize an entire class of objects, rather than to identify a single instance. Inherent differences between the instances of the same class make the task even more difficult. Consequently, a successful object recognition system must be robust to such changes, a property that is called *invariance*. Of course, the system must not be too inclusive, or it risks "recognizing" the same object in every image. This property of being appropriately inclusive is called *selectivity*.

Advances in object recognition would dramatically affect how devices interact with their environment, and allow us to interact with those devices in a more natural way. Such advances could enable applications such as visual search, gesture-based interfaces, and robotic navigation, while impacting areas such as national security, transportation, consumer electronics, and medicine. Along the way, advances in object recognition could easily impact our understanding of the neuroscience of vision.

2.2 Alternating Multilayer Architectures

The focus of this dissertation is a family of object recognition systems that I call *alternating multilayer architectures*, which were popularized by the Neocognitron [3] and HMAX models [2, 1]. These systems employ artificial neural networks in a manner inspired by biological vision systems. The network is organized hierarchically into discrete layers, where the activity of one layer is used as input to the layer above.

A diagram of the architecture is shown in Figure 2.1. The image is processed by a layer of *S1 units*, which detect edges of different orientation and scale. The result is processed by a layer of *C1 units*, which provide some tolerance to changes in the scale or location of those edges. The names S1 and C1 refer to the so-called *simple* and *complex* cells in the brain, as discovered by Hubel & Wiesel [4].

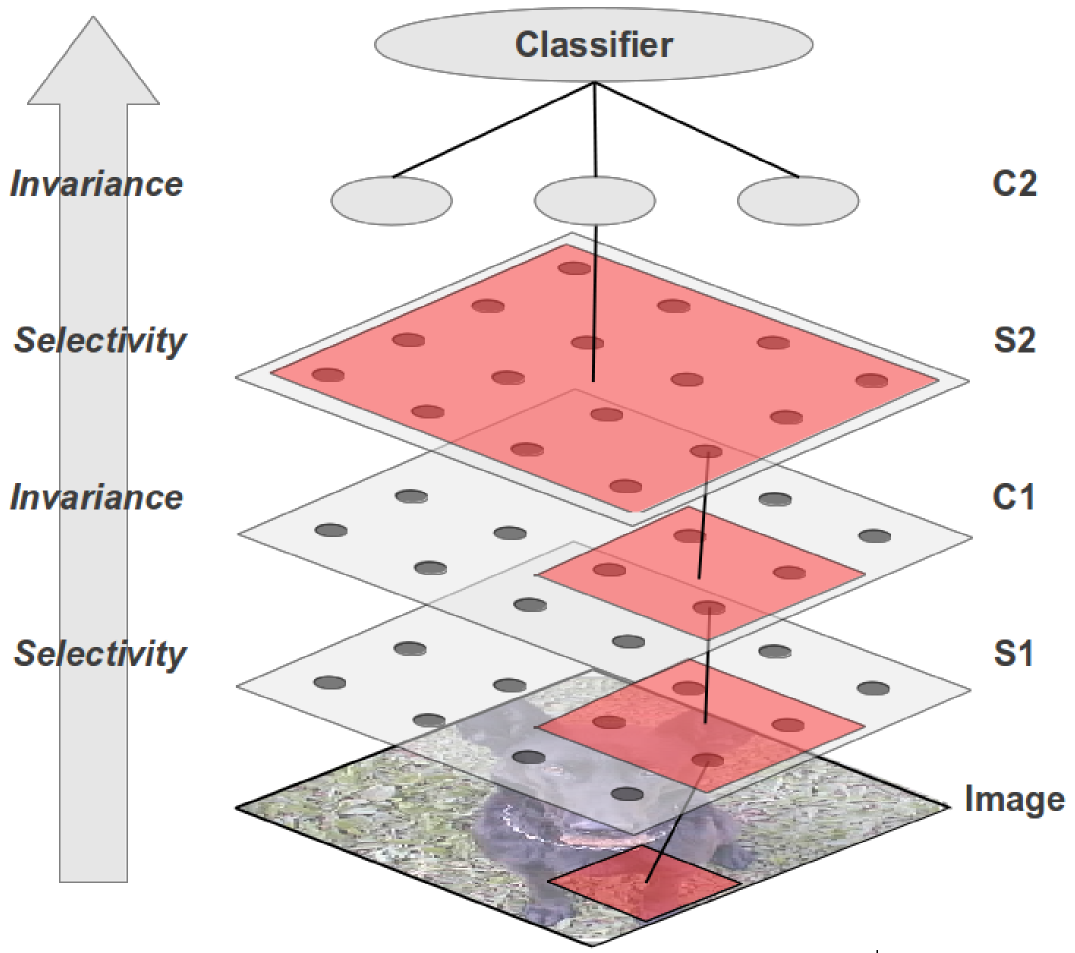


Figure 2.1: Diagram of the alternating multilayer architecture. An image is first processed by units in the S1 layer, each of which is selective for an edge at a particular orientation and scale. The result is processed by units in the C1 layer, which provide local invariance by pooling over a small neighborhood of S1 units. Units in the S2 layer are then applied, which become active when the input matches a stored shape template called a prototype. The result is passed to the C2 layer, in which units pool over the all S2 units for a given prototype. Activity of the C2 units is passed to a classifier, which predicts the class of object in the image (e.g., “dog”).

Activity of the C1 layer is processed by a layer of *S2 units*, which detect the presence of shape templates called *prototypes*. The system is connected hierarchically, with activity for multiple edge orientations fed into each S2 unit. The result is processed by a layer of *C2 units*, which provide tolerance to large changes in the size and location of objects. This alternation between S-units and C-units is argued to allow the model to balance the conflicting needs of selectivity and invariance [2]. Finally, the activity of the C2 layer is input to a classifier, which predicts the class of the object.

In short, the system uses a hierarchy to compute a new representation of the image, from which an object can be identified more easily than from raw pixel values. Critically, this representation is invariant to certain changes to the object's appearance, such as those caused by certain translations, rotations, and scalings.

2.2.1 S1 Layer

An S1 unit takes a neighborhood of image pixels as input, and responds to an edge at a particular orientation and scale. The unit becomes active if the given edge occurs at that location in the image. A battery of S1 units—corresponding to a range of edge orientations and scales—is applied at each location in the image. The same battery is replicated for each location, and the resulting activity defines a set of "edge maps". Note that the parameters of the edge detectors are constants that are specified as part of the model.

2.2.2 C1 Layer

The input to a C1 unit consists of a small region of S1 activity defined by a neighborhood of locations and scales. A C1 unit's activation is equal to its most active input. Thus, the C1 layer is intended to provide a small degree of tolerance to changes in the position and scale of the edges detected at S1.

2.2.3 S2 Layer

An S2 unit takes a neighborhood of C1 activity as input, and compares it to a stored shape template called a *prototype*. The activity of the S2 unit reflects the degree of match between the input and the prototype. There is a battery of S2 units applied at each location, where each unit is associated with a different prototype. This battery is replicated across all locations and scales at C1. Thus, the S2 layer provides specificity to particular shapes. Unlike the parameters at S1, the set of prototypes is not specified by the model. This will be discussed further in Section 2.3.

2.2.4 C2 Layer

The input to a C2 unit consists of activity from all S2 units for a given prototype. A C1 unit's activation is equal to the maximum input activation. There is one C2 unit for each prototype, and the activity of a C2 unit indicates the best match for that prototype anywhere in the image (and at any size). Thus, C2 activity provides an image representation that is invariant to changes in the object's position and scale.

2.2.5 Classifier

The input to the classifier consists of the activity for all C2 units in the network. Each activity value is called a *feature*, and the vector of activities of all C2 units is called a *feature vector*. Similarly, the class of object in the image is called the *label*. The classifier compares the feature vector to those it has seen in the past, and predicts a label for the image.

To perform this prediction, the classifier must have been exposed previously to the feature vectors and known labels for a set of example images. During this *training* phase, the classifier uses the examples to learn the relationship between labels and features.

2.2.6 Related Work

Perhaps the best known example of an alternating multilayer architecture is the HMAX model [1, 2]. HMAX was initially designed as a neuroscience tool to account for the behavior of biological vision systems, and its parameters were chosen to match observations from neurophysiology [5]. The model was later shown to be useful for computer vision problems, with researchers using it to demonstrate what was then state-of-the-art performance on common computer vision problems [6, 7]. However, it should be noted that the model's performance on visual tasks is well below the capabilities of humans. The model did match constrained human performance [2] on a so-called "speed of sight" task [8], in which the image is shown very briefly.

The design of HMAX was influenced by the work of Fukushima on the Neocognitron, and was itself the basis for the Sparse Localized Features (SLF) model of Mutch & Lowe [9]. Additionally, alternating multilayer architectures are closely related to Convolution Networks [10, 11] and other "deep" neural networks [12, 13], which have recently generated interest in both academic and industrial contexts. The model has been extended by a number of researchers, increasing its performance substantially [14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25]. Since its introduction, HMAX has been applied to tasks such as biometric analysis [26], face and facial expression recognition [27, 28], remote sensing [29], and the modeling of visual attention [30, 31, 32]. The alternating multilayer architecture is also a common theme in the neuroscience literature [33, 34, 35, 36, 37].

2.3 Prototype Learning

As discussed above, the choice of prototypes used by the S2 layer is not given as part of the model. Instead, prototypes are learned from images by a process known as *imprinting*. In this approach, illustrated in Figure 2.2, the model is applied to a set of example images and the C1 activity is recorded—or imprinted—for selected image regions. These regions are selected at random, and one prototype is created from each region.

The method of imprinting is argued to create a redundant "dictionary" of discriminative shape components [38]. This is thought [2] to be central to the model's success, and the learning of S2 prototypes via imprinting is the primary contribution of the extended HMAX model compared to the base model of Riesenhuber & Poggio [1]. This is explained by Serre et al. [2]:

> The major extension is a new unsupervised learning stage of the units in intermediate stages of the model. A key assumption in the new model is that the hierarchy [...] builds a generic dictionary of *shape-*

> *tuned* units which provides a rich representation for task-specific categorization [...] The resulting dictionary is generic and *universal* in the sense that it can support [...] the recognition of many different object categories. *(Emphasis added.)*

The prototypes in the dictionary are *redundant* if they encode the same shape more than once, and are *discriminative* if they yield feature values that help distinguish between different visual categories.

The performance of the HMAX model was shown to increase significantly when manually constructed prototypes were replaced with those learned by imprinting [39]. In practice, imprinting often leads to strong model performance even when the prototypes are learned from unrelated images. In one case, for example, the model successfully performed a multiclass object recognition task using prototypes learned from randomly-chosen natural images [7]. This supports the notion that imprinting can create dictionaries that are universal.

Due to the random selection of image regions, however, there is no guarantee that imprinted prototypes will be helpful for classification. In fact, this is often not the case, either because prototypes are redundant or because they lead to non-discriminative features. This is problematic for two reasons. The first is that such prototypes can decrease the performance of the model, since many common classifiers are sensitive to irrelevant features [40, 41]. The second reason is that the addition of these prototypes dramatically increases the model's computational complexity, which is already quite significant.

To address these problems, some researchers [39, 31, 16, 9, 19] have successfully used a process known as *feature selection* [42] to identify and remove extraneous prototypes. The approach has been very effective, increasing performance

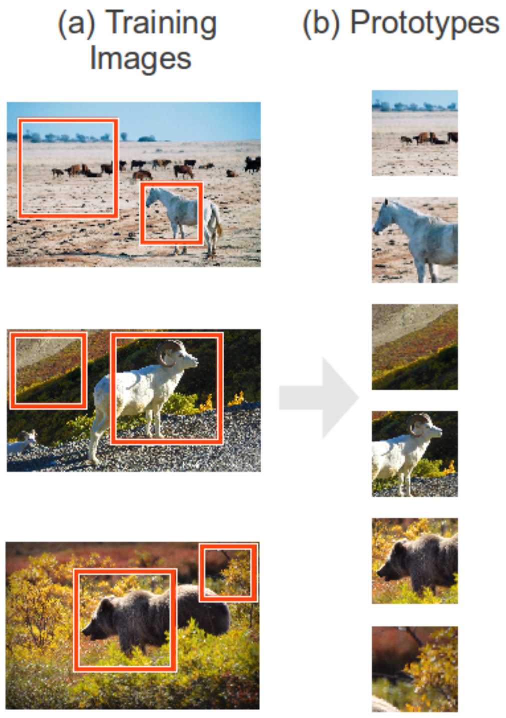


Figure 2.2: Illustration in which prototypes are constructed by imprinting for a hypothetical animal/no-animal task. (a) Image regions (shown as red boxes) are chosen at random, and (b) new prototypes are recorded from the model's C1 activity. In this example, six prototypes are created from three images.

while reducing the number of prototypes by as much as 75% [16]. Unfortunately, the computational expense of feature selection can be prohibitive, and this limits the number of imprinted prototypes that can be evaluated. Additionally, feature selection can only return a discriminative prototype if it was found during imprinting—that is, it can never synthesize a new prototype.

An alternative way to reduce the number of prototypes is called *clustering* [43]. Here, the set of imprinted prototypes is grouped into clusters of visually similar elements. A new prototype is created for each cluster, and only the new prototypes are used by the model. In the most common approach [44, 39, 19], the k-means algorithm [45] is used to create new prototypes that are the "average" of the elements in each cluster. The techniques of feature selection and k-means clustering will be discussed further in Chapters 6 and 7.

2.3.1 Learning of Invariance Properties

In addition to learning prototypes for S2 units, it is also possible to learn the connectivity patterns of C-units, which encode the model's invariance properties. The general problem of invariance learning has been considered in a handful of studies [46, 47, 48, 49, 50], and was applied to an HMAX-like system by Masquelier et al. [51]. See also [52, 53] for a discussion of invariance learning in biological systems. Note that I ignore the problem of invariance learning in this work, and instead focus only on the learning of selectivity. Thus, I apply fixed connectivity for C1 and C2 units.

2.4 Evaluation

The purpose of learning prototypes is to increase the accuracy of the classifier while decreasing the number of prototypes used at (and thus the compute cost of) the S2 layer. To evaluate a particular learning method, therefore, an obvious approach is to compute the accuracy using sets of learned prototypes, and compare that to some baseline accuracy. One baseline is the accuracy that would be achieved by chance—that is, if the model classified each feature vector by randomly guessing from the set of target classes. Given T target classes, the probability of correctly guessing the class for a single feature vector is $\frac{1}{T}$, and thus the accuracy due to chance for a binary classification task is 50%. Accuracy that is (statistically) significantly above 50% implies that the learned prototypes allow the classifier to form a useful decision boundary. Some studies, such as that of Serre et al. [39], compare the classifier accuracy and related measures for learned prototypes to that of manually constructed prototypes. In this case, the performance for manually-chosen prototypes provides a lower bound on the performance achievable in the absence of learning. In some cases, these manually-chosen prototypes lead to performance that would be expected if the classifier were guessing at random [44]. The difference between the two accuracies indicates the relative performance benefit of learning.

Note that accuracy indicates only whether prototypes led to discriminative features. It does not, however, explain why those features were useful for classification. To better understand this, it is common to ask what a given S2 unit is "looking for"—that is, what input pattern it responds to. Remember that a prototype represents a configuration of C1 activity. However, the invariance properties of the C1 layer mean that it produces the same activity for an entire

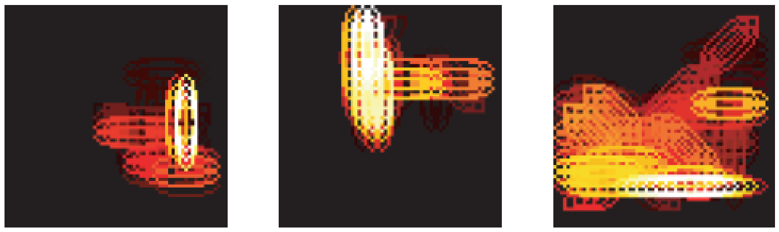


Figure 2.3: Example visualizations of prototypes taken from Serre et al. [6], corresponding to airplanes (left), faces (middle), and motorcycles (right). Here, an oval indicates the location and scale of an edge detector (i.e., an S1 unit), while color indicates the contrast of those edges.

class of related images. Thus, the prototype actually represents a class of related image patterns, which can be difficult to analyze.

Various ways to *visualize* a prototype have appeared in the literature. Since S1 units specify the presence of edges, one way to visualize the input pattern corresponding to a prototype is to show the activation of those edge detectors [39, 6, 35]. For clarity, each S1 detector is represented by an oval, which indicates the location and orientation of the detector's preferred edge. Some examples of this approach are shown in Figure 2.3. Another approach [9] applies an S2 unit at multiple locations and scales across each image in a corpus, and records the image locations that lead to the highest activity. This provides a collection of image patches that match the prototype, and these patches are inspected manually. I have also used this visualization approach in my work [54].

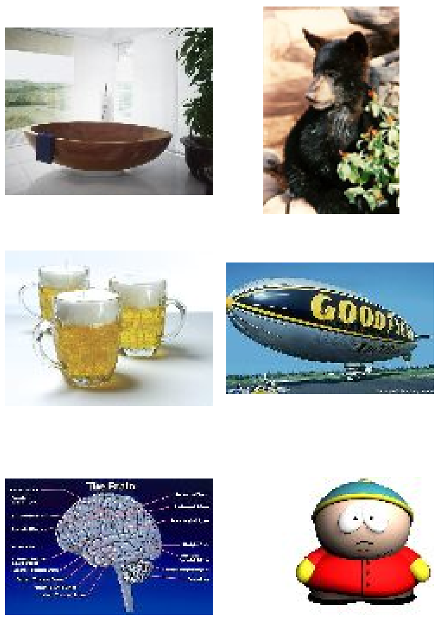
2.4.1 Datasets Used in This Work

The computer vision literature contains a wealth of public datasets, which provide a shared point of reference and allow the comparison of models with very different architectures. While we do not attempt an exhaustive list, this section provides a survey of some common object recognition datasets.

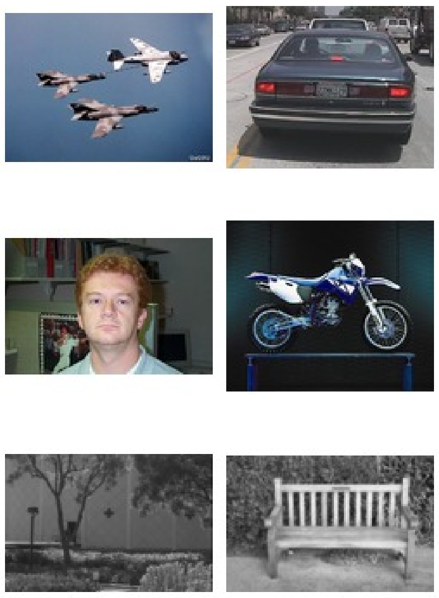
Two datasets used for research in HMAX-like visual models are the *Caltech 101* corpus [55] and the tasks of Fergus et al. [56]. The *Caltech 101* corpus includes examples of 102 categories (101 foreground categories and a background category), where each category contains between 30 and 800 examples. The tasks of Fergus et al. provide similar examples for a set of five (four foreground and one background) categories. These datasets have been used extensively in research, both in HMAX-like models [38, 29, 57, 58, 59, 14, 15, 60, 9, 20, 25, 23, 19, 16, 6, 7, 61, 62] and the broader computer vision literature [63, 64, 65, 66, 67, 68]. The related *Caltech 256* corpus [69] has also been used for work on HMAX-related models [62, 29, 61]. The *Animals* dataset of Serre et al. [2, 38, 70] presents an Animal/No-Animal task, and was used to compare behavior of their HMAX model with that of human subjects. Example images from these datasets are shown in Figure 2.4.

A number of authors [71, 17, 72, 73] have raised concerns about corpora based on unconstrained natural imagery of the kind described above. The first concern is simply that the tasks have become too easy, and thus fail to differentiate between models. This issue is somewhat positive, in that it reflects the substantial progress made since these datasets became available, and is being addressed with the introduction of significantly larger datasets. A more serious concern is that these datasets may be too easy simply because they lack real-world variation in the presentation of objects. If an object is always presented in the same way, and this way is different for different objects, then the model may end up recognizing the presentation rather than the object itself. The *Caltech 101* dataset has received particularly strong criticism. For example, Pinto et al. [17] showed that a simple model with no invariance properties could account for the dataset's best-reported performance.

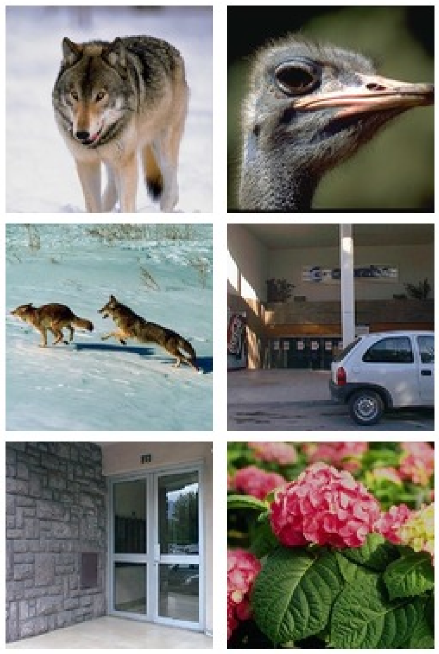
(a) *Caltech 101*



(b) *Caltech 256*



(c) Datasets due to Fergus et al. [56].



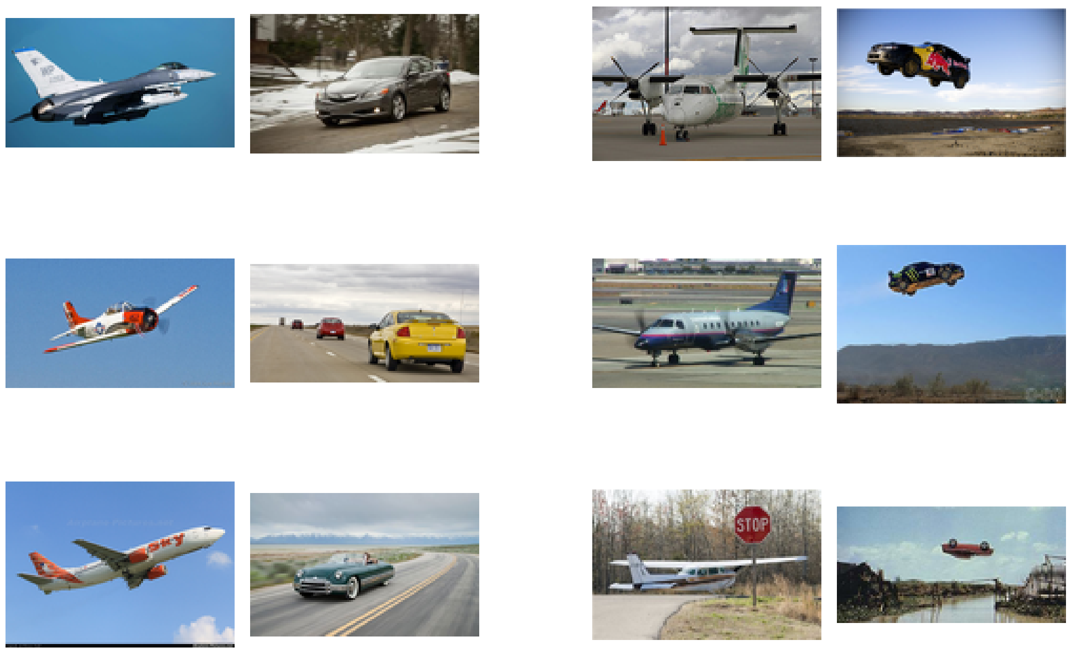
(d) *Animals*

Figure 2.4: Example images from reference corpora used in this work.

To demonstrate, imagine that the goal is to build a dataset consisting of cars and airplanes. This dataset could be composed of example images downloaded from the internet, such as those shown in Figure 2.5a. In these images, the object's context is highly predictable. Airplanes are set against a blue or cloudy sky, while cars are shown driving over asphalt. Furthermore, imagine that the visual model is evaluated on the dataset, and it performs spectacularly. Does this suggest that the model performs invariant object recognition? If so, it should be able to recognize the same objects in a new context, such as those in Figure 2.5b. An airplane spends a great deal of time resting on the ground, after all, and often appears on the same asphalt surface that composes the "car" context. There are even instances in which a car may appear in the sky! If the dataset does not include such examples, we have no way of knowing whether the model is simply performing "blue sky" detection for airplanes and "road" detection for cars.

This issue relates to what the machine learning community calls *generalization*, which is the ability to solve the general problem we care about, rather than exploiting specific regularities in the training data. Ideally, an object recognition system should mimic the abilities of natural vision systems. Thus, the system should generalize with respect to object presentation, which includes such attributes as pose, location, illumination, and background clutter or context.

In response to these concerns, some authors have chosen to create synthetic object recognition tasks. These tasks were designed to probe a system's ability to demonstrate viewpoint invariant object recognition, without using visual cues from the surrounding environment. The dataset is constructed by rendering a 3D object model from various points of view, and then composing the object with a randomly-chosen image background. The difficulty of each task depends on the



(a) (b)

Figure 2.5: (a) Example images of cars and airplanes for an object recognition task. In this hypothetical dataset, objects are strongly associated with a particular context, where airplanes appear on sky backgrounds, and cars on asphalt. (b) Examples of an "unexpected" context for the same task.

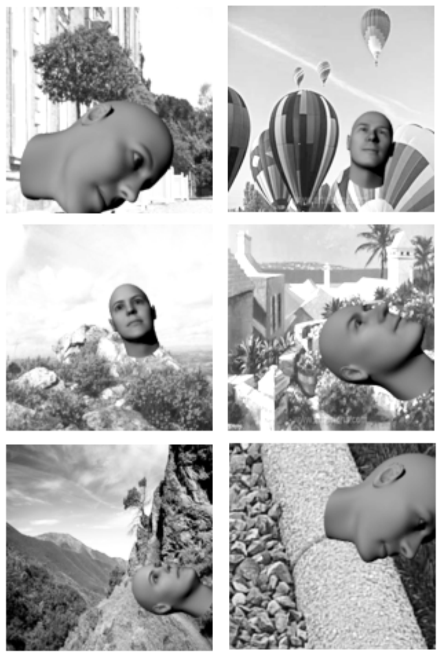
type of background and range of viewpoints from which an object is rendered.

Pinto et al. [18, 74] provide two such datasets of rendered objects. The first dataset contains rendered examples of cars and airplanes (*Car v. Plane*), and measures category-level discrimination (Figure 2.6a). The second dataset contains rendered examples of two different faces (*Face1 v. Face2*), and measures subordinate-level discrimination—that is, discrimination between examples of the same category (Figure 2.6b). The provided data is split into seven different variation levels—levels of variation in rotation, position, and scale of the objects of interest—and each level of variation defines a separate object-recognition task.

Additionally, a similar set of tasks was constructed by Brumby et al. [62], which contain both rendered and natural examples of cats and dogs (*Cats v. Dogs*; Figure 2.6c). These objects were composed with various backgrounds, including uniform gray, randomly-generated $\frac{1}{f}$ noise, and natural imagery selected randomly from the internet.

The use of computer-generated objects could be considered a source of concern, as they may present visual statistics unrepresentative of natural imagery. A possible solution to this problem could use images of real objects in conjunction with natural image backgrounds. (In fact, this was already done as part of the *Cats v. Dogs* dataset.) There exist a number of datasets containing real foreground objects with variation in orientation and illumination, including ALOI [75], ETH80 [76], NORB [77], and COIL [78]. Given pixel-wise object masks—that is, a labeling of pixels as “foreground” or “background”—the object can be easily extracted to create new corpora (see Figure 2.7 for an example). While object masks are available for the ALOI and ETH80 datasets, NORB and COIL lack this information. Unfortunately, the automatic generation of such masks is sometimes non-trivial, as illustrated in Figure 2.8 for an example COIL object.

(a) *Cars v. Planes*


(b) *Face1 v. Face2*

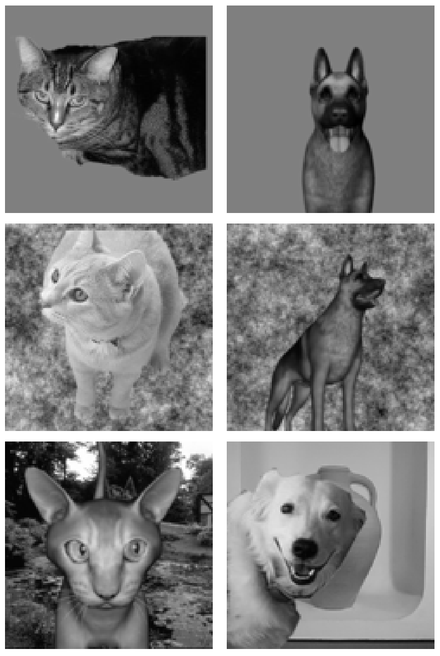

(c) *Cats v. Dogs*

Figure 2.6: Example images from synthetic corpora used in this work.

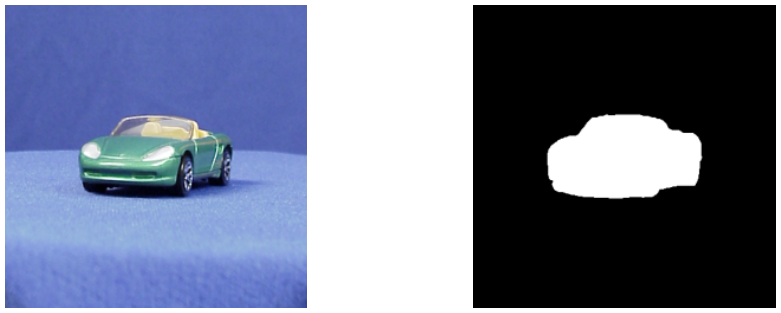


Figure 2.7: Example image from the ETH80 dataset (left) and corresponding object mask (right), as provided with the dataset.

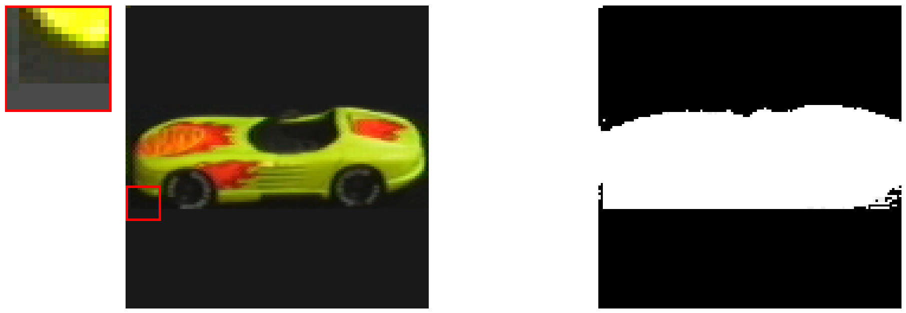


Figure 2.8: Example image from the COIL dataset (inner left) and the best corresponding object mask I was able to generate using a color threshold (right). A contrast-enhanced section of the object's boundary is shown (far left), which shows the presence of background artifacts.

It is worth noting that we often *do care* about the context in which an object appears, including its associated background information. In fact, "scene" recognition has become a fruitful line of research unto itself [79, 80], and has been suggested as a mechanism with which to "prime" object detection [81, 82]. For any given experiment, however, we want to know exactly *what* is being measured. We want to know that a model performs well because it solves the object recognition problem rather than relying on background artifacts and consistent presentation. In general, we want to know *why* a model performs the way it does, which drives the quest for explainable visual models [83].

Chapter 3

Glimpse

To support my research, I have created a novel system for the implementation and application of hierarchical visual models. I call this system the General Layer-wise IMage Processing Engine (GLIMPSE) [84]. The goal of the Glimpse Project is to allow a broad range of feed-forward, hierarchical models to be encoded in a high-level, declarative manner, with low-level details of the implementation hidden from view. This project combines an efficient implementation with the ability to leverage parallel processing facilities and is designed to run on multiple operating systems using only common, freely-available components.

Using this system, I have instantiated a particular hierarchical model that I call the Glimpse model. The rest of the chapter discusses this model, starting with the architecture in Section 3.1. In Section 3.2, I discuss the method used to choose some of the more significant model parameters. Finally, Section 3.3 provides a comparison of Glimpse behavior to that of similar hierarchical models from the literature, demonstrating that Glimpse effectively replicates the behavior of well-known models from the literature.

3.1 Architecture

The Glimpse model is an example of an alternating multilayer architecture introduced in Chapter 2. It applies multiple stages with alternating layers of S- and C-units, which provides a trade-off between selectivity (i.e., object specificity) and invariance (i.e., stability under image transformations). It uses six layers in total.

An image is first input to the model and is preprocessed. A layer of S1 units is then applied, which implement localized edge detectors across a range of scales and orientations. This is followed by a layer of C1 units, which provides a representation that is invariant to small changes in an object's location and scale. A layer of S2 units is then applied, which detect localized patterns of activity often representing shapes. These patterns are given by prototypes, and are detected at each scale independently. A layer of C2 units is applied to the result, which pools over the entire image and over all scales. The output is largely invariant to changes in location and scale of the target object. Finally, a classifier is applied to a feature vector composed of C2 activity, and a prediction is made regarding the class of object in the image. A diagram of the model is given in Figure 3.1, which also summarizes some of the model's more significant parameters. Below, I provide details for each layer of the model.

3.1.1 Preprocessing Layer

In the first layer, the input image is preprocessed. The image is converted to grayscale, and resized such that its shortest edge is 220 pixels (maintaining the image's aspect ratio). The result is split into a nine-band scale pyramid by down-sampling the image at progressively higher rates (using an anti-aliasing

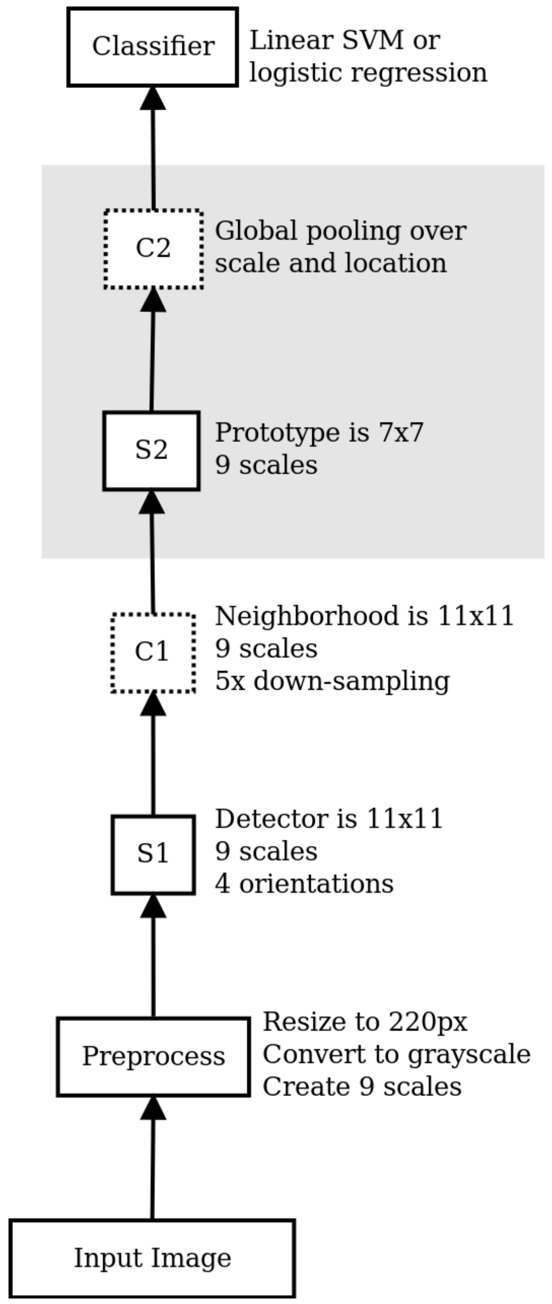


Figure 3.1: Overview of the Glimpse model. An image is presented at the bottom layer, and processing flows up the diagram. Layers of S-units are shown as solid-outline boxes, and C-unit layers are shown as dashed-outline boxes. The C2 layer generates a one-dimensional vector of features, with one feature per S2 prototype. At the top layer, those features are passed to a trained classifier, which predicts the object class.

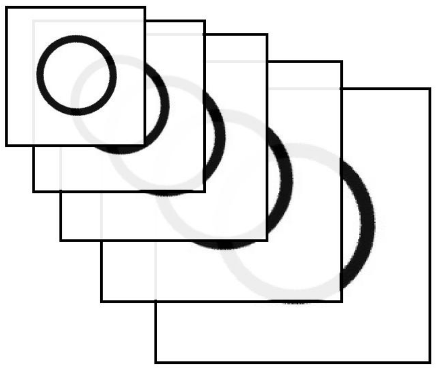


Figure 3.2: Illustration of a scale pyramid for an example image containing a circle. This pyramid has five scales, with a down-sampling ratio of $2^{1/4}$ between scales. Scale bands appear translucent for illustration.

filter). The ratio between neighboring scale bands is $2^{1/4}$. An example scale pyramid is shown in Figure 3.2 for an image containing a circle.

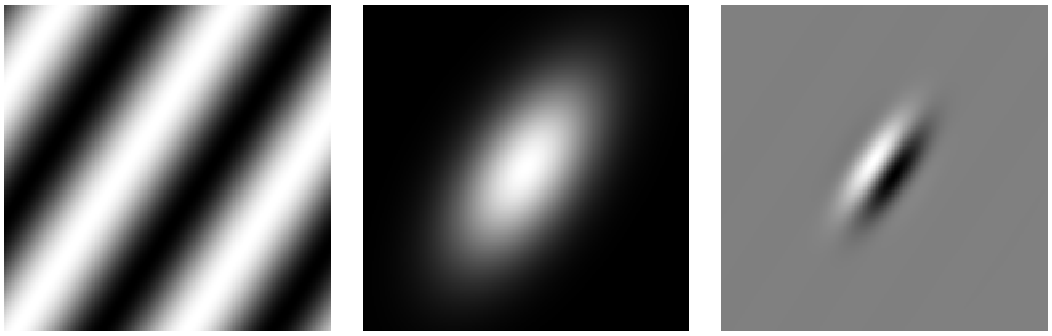
3.1.2 S1 Layer

The first stage of S-units applies localized edge detectors over each scale band of the preprocessed image. The detectors are implemented by first computing a normalized dot-product[1] and then applying the absolute value operator. That is, the activation of the S1 unit is given by

$$\mathrm{S1}(\mathbf{x}, \mathbf{d}) = \frac{|(\mathbf{x}, \mathbf{d})|}{\|\mathbf{x}\| \cdot \|\mathbf{d}\|} \tag{3.1}$$

where $\mathbf{d}$ is an S1 edge detector, and $\mathbf{x}$ is a patch of the input image, $(\cdot, \cdot)$ denotes the dot product, $|\cdot|$ denotes the absolute value operator, and $\|\cdot\|$ denotes the L^2 norm of the vector. Here, vectors are denoted in lowercase bold font ($\mathbf{x}$), and matrices in uppercase bold font ($\mathbf{X}$). Scalars will be denoted in lowercase (x). Edge detectors are defined by the Gabor function, given as

[1]The normalized dot-product is also called the *cosine similarity*.



(a) Sine wave (b) Gaussian (c) Gabor edge detector

Figure 3.3: A Gabor edge detector can be thought of as the combination of a sine wave with a two-dimensional, oriented Gaussian function.

$$\begin{aligned} d\left(u,v\right) &= \exp\left(-\frac{\left(u_0^2+\gamma^2 v_0^2\right)}{2\sigma^2}\right)\sin\left(\phi+\frac{2\pi x_0}{\lambda}\right), \text{ where} \\ u_0 &= u\cos\theta+v\sin\theta \\ v_0 &= -u\sin\theta+v\cos\theta\,. \end{aligned} \tag{3.2}$$

Here, u and v define the horizontal and vertical offset from the center of the detector. This defines a Gaussian window applied to a sinusoidal wave, as illustrated in Figure 3.3.

In this work, I use orientations $\theta=\left(\frac{\pi}{8},\frac{3\pi}{8},\frac{5\pi}{8},\frac{7\pi}{8}\right)$, phase $\phi=0$, aspect ratio $\gamma=0.6$, wavelength $\lambda=\frac{w}{4}$, and scale $\sigma=\frac{\lambda}{2}$, where $w=11$ is the detector size. Thus, there are four S1 detectors, corresponding to edges at four different orientations. This is shown in Figure 3.4. When these detectors are applied to the "circle" image used in Figure 3.2, this produces the result shown in Figure 3.5.

Note that the choice of λ and σ was made to ensure that one to two cycles of the sinusoidal wave would be present in the resulting detector. Overall, the

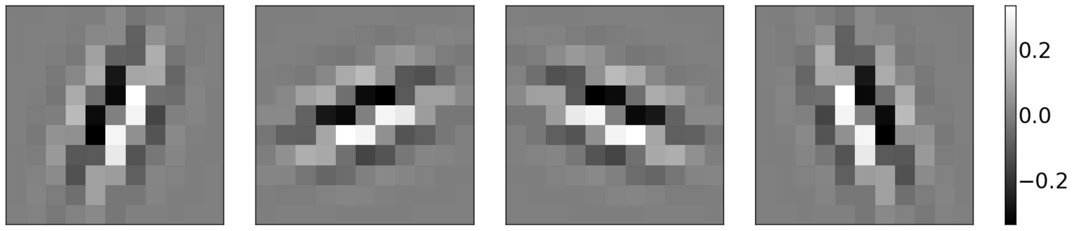


Figure 3.4: Visualization of the S1 detectors corresponding to edge orientations $\theta = \left(\frac{\pi}{8}, \frac{3\pi}{8}, \frac{5\pi}{8}, \frac{7\pi}{8}\right)$, given clockwise from the top-left corner.

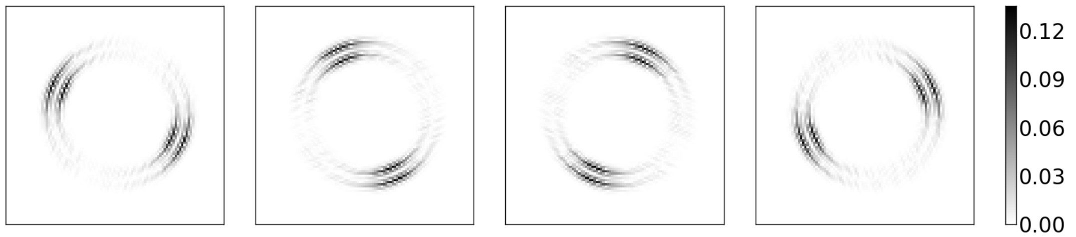


Figure 3.5: S1 activity at one scale for an image containing a circle. Activity is shown for all four orientations, with plots corresponding to the detectors in Figure 3.4.

Gabor parameters were chosen to provide a good trade-off between orientation and scale specificity, as discussed below in Section 3.2.

The behavior of the S1 activation function (Equation 3.1) has a number of desirable properties. First, the dot product provides a measure of similarity between the input patch and detector. Second, the normalization constraint provides a form of contrast gain control, in that a dark edge on a light background elicits a similar response regardless of the darkness of the edge or brightness of the background. However, note that this causes poor behavior for very dark image regions, since Equation 3.1 approaches infinity as the norm on the input patch shrinks to zero. Thus, I suppress activation in low-light regions by thresholding the input norm as

$$\mathrm{S1}(\mathbf{x}, \mathbf{d}) = \frac{|(\mathbf{x}, \mathbf{d})|}{\max\left(\|\mathbf{x}\|, \tau\right) \times \|\mathbf{d}\|} . \tag{3.3}$$

In my experiments, I use the threshold $\tau = 0.1$.

Another desirable property of Equation 3.1 is its invariance to an inversion of the image, which is provided by the absolute-value operation in the numerator. That is, the model responds identically when each white pixel is replaced with a black pixel and vice versa.

3.1.3 C1 Layer

The C1 layer implements local pooling over space, as well as a down-sampling in the spatial resolution. Each C1 unit pools over a small neighborhood of S1 units at one scale, where this neighborhood is 11x11 units in all experiments. Given a neighborhood of S1 activation denoted by the vector $\mathbf{x}$, the C1 activation is calculated as

$$\mathrm{C1}(\mathbf{x}) = \max_i x_i \,, \tag{3.4}$$

where x_i ranges over the elements of the input neighborhood. The result is then down-sampled by some constant factor, N. Thus, C1 activity is retained at every scale, but only for every N^{th} location. In my experiments, the down-sampling factor was set to $N = 5$, which results in each S1 unit contributing to the activation of exactly one C1 unit. Applying this processing to the S1 activity in Figure 3.5 results in the C1 layer shown in Figure 3.6. This has the effect of "blurring" the S1 edge maps.

3.1.4 S2 Layer

The S2 layer detects patterns in each scale and location of C1 activity. Each S1 unit compares a local neighborhood of C1 activity to a stored pattern called a *prototype*. The comparison is implemented as a radial basis function (RBF), and

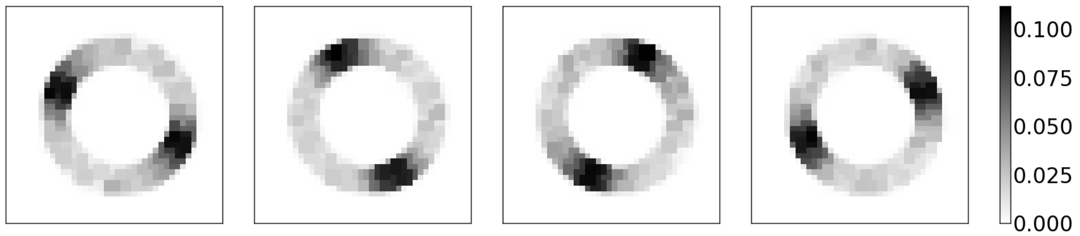


Figure 3.6: C1 activity at one scale for an image containing a circle. Activity is shown for the four orientations in Figure 3.5.

is given as

$$\mathrm{S2}(\mathbf{x}, \mathbf{p}) = \exp\left(-2\beta \left\|\mathbf{x} - \mathbf{p}\right\|^2\right) \tag{3.5}$$

for C1 input $\mathbf{x}$, S2 prototype $\mathbf{p}$, and parameter β. The S2 activity is maximal when the input and prototype are identical, and decreases as the input diverges from the prototype. This decrease is not linear, however, but follows a Gaussian function with width β^{-1}. Large values of β cause the S2 unit to be sharply tuned, such that the input and prototype must be nearly identical for the unit to become active. The unit becomes broadly tuned as β decreases. Note that the input and prototype contain activity for all four orientations bands. However, S2 prototypes are applied at each scale independently.

A prototype is constructed by imprinting the C1 data from a randomly selected patch in a randomly selected training-set image, as discussed in Chapter 2. This is illustrated in Figure 3.7a. Applying the prototype to the image from which it was imprinted results in S2 activity shown in Figure 3.7b. Notice the high activation for the region from which the prototype was imprinted.

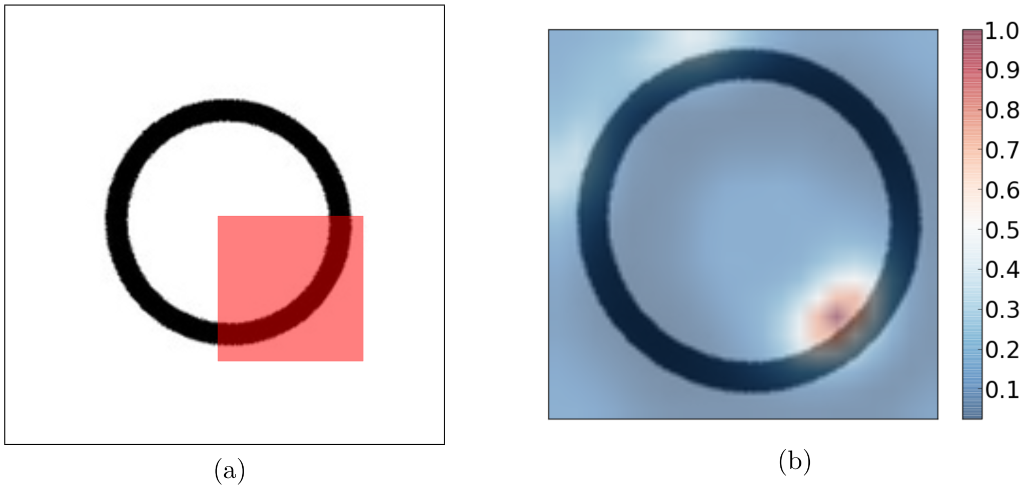


Figure 3.7: (a) Example in which a prototype is "imprinted" from an image, with the selected region shown in red. (b) The S2 activity resulting from applying this prototype to the original image. Red indicates high activity, while blue indicates low activity.

3.1.5 C2 Layer

The C2 layer applies a maximum-value pooling operation to the activity of all S2 units for a given prototype, including all locations and scale bands. That is, the activation of a C2 unit is given by

$$\mathrm{C2}(\mathbf{x}) = \max_i x_i \tag{3.6}$$

where $\mathbf{x}$ is the activation of all S2 units for a given prototype. Thus, the C2 layer has one unit for each S2 prototype. The activation of a unit indicates the degree to which the corresponding prototype was matched at any location and scale within the image.

3.1.6 Classifier

As the final step in the model, a trained classifier is applied to a feature vector that is constructed from C2 activity. The classifier analyzes the feature vector by applying a *decision function*, which decides which object label to return. A simple decision function might be

$$f(\mathbf{x}) = \operatorname{sgn}\left(\sum_{i=1}^{m} x_i \right) , \tag{3.7}$$

where $\mathbf{x}$ is the feature vector composed of C2 activity, and m is the number of features. If the sum of the features is greater than zero, then the "positive" class is chosen by returning $+1$. Otherwise, the "negative" class is chosen by returning -1. In practice, the values $+1$ and -1 would be associated with different object labels, such as "dog" and "person".

The linear decision function in Equation 3.7 is not very useful, because it assumes that all features are associated with the positive class. What if a strong match for one prototype indicates the presence of a "dog", while a strong match for another prototype indicates a "person"? To handle this case, a slightly more complex function is needed. An example of such a function is

$$f(\mathbf{x}) = \operatorname{sgn}\left(b + \sum_{i=1}^{m} \alpha_i x_i \right) , \tag{3.8}$$

where the feature weights α_i and bias b are parameters that are chosen during a training process. This allows, for example, a "dog" feature to be weighted negatively, assuming that -1 indicates the "dog" class.

The training process takes a collection of labeled feature vectors called the *training set*, and chooses the classifier's parameters such that the classifier predicts the correct label for as many training examples as possible. At the end of training, the parameters are fixed, and the classifier is evaluated on a set of labeled examples called a *test set* that were not part of the training set. The errors on this set determine the *performance* of the classifier, and thus the performance of the model.

The form of the decision function and the learning method used to choose its parameters are defined by the choice of classification algorithm. A common choice of classification algorithm in HMAX-like models is called a support vector machine (SVM). An SVM chooses a subset of the feature vectors as reference points, called *support vectors.* Given a new image, the decision function compares the feature vector to each support vector. The predicted label is the one associated with the most similar support vectors. Specifically, the decision function is defined as

$$f(\mathbf{x}) = \operatorname{sgn}\left[b + \sum_k \gamma_k \phi\left(\mathbf{x}, \mathbf{v}_k\right)\right] , \tag{3.9}$$

where $\mathbf{v}_k$ denotes the k^{th} support vector, γ_k denotes the importance of that vector, and b is a bias term. The function $\phi\left(\mathbf{x}, \mathbf{v}\right)$ is called the *kernel function*, and measures the similarity between the feature vector and the support vector. In the simplest case, called a linear SVM, $\phi\left(\mathbf{x}, \mathbf{v}\right)$ is just the inner product of the two vectors. In this case, Equation 3.9 can be rewritten as

$$f(\mathbf{x}) = \operatorname{sgn}\left[b + \sum_i \left(\sum_k \gamma_k v_{ki}\right) x_i\right] , \tag{3.10}$$

which takes the form of Equation 3.8 with feature weights given as $\alpha_i = \sum_k \gamma_k v_{ki}$.

In some instances, classification is performed using logistic regression [85].

This is an alternative approach[2] with a decision function given by

$$f(\mathbf{x}) = \operatorname{sgn}\left[\frac{1}{1+e^{-(b+\sum \alpha_i x_i)}} - \frac{1}{2}\right] , \tag{3.11}$$

where i is a feature index, b is a bias term, and the expression $1/\left(1+e^{-t}\right)$ is called the logistic function. In some cases, the classifier is encouraged to use as few features as possible, which is achieved by setting the remaining feature weights α_i to zero. This is called "sparse" logistic regression [86]. In my experiments, sparse logistic regression consistently resulted in performance that was similar to that of a linear SVM classifier, while being significantly faster to train.

It is sometimes useful to measure a feature's "importance", that is, the degree to which it influences the classification. In the case of a linear SVM, the importance of the i^{th} feature has been measured [40, 87] as the value α_i^2. Similar values have been used to measure feature importance in logistic regression [41].

3.2 Model Parameters

I have performed a number of experiments to investigate the optimal parameter settings for the Glimpse model. One significant choice is the method used to create scale bands in the S1 layer, which is considered in Section 3.2.1. The optimal size of the edge detector at the S1 layer is considered in Section 3.2.2, and Section 3.2.3 considers the best way to implement normalization in the S1 activation function.

[2]Despite the misleading terminology, logistic regression is actually an algorithm for classification problems rather than regression.

Scale	1	2	3	4
Correlation	0.95	0.89	0.83	0.81

Table 3.1: Similarity between the response maps for multiscale detectors compared with that for multiscale inputs (i.e., image scaling). The similarity is measured as the correlation coefficient for the response maps shown in Figure 1.9. The correlation coefficient takes values between zero and one, with larger values indicating more similar maps.

3.2.1 Scaling

In a hierarchical model such as HMAX, the activity of the S1 layer indicates the presence of edges at various locations and scales. Edges can be extracted at different scales using (at least) two alternative approaches. First, a battery of multiscale detectors can be applied to the original image, where each detector responds to an edge at a different scale. Second, the image may be repeatedly down-sampled, with a single detector scale applied to each layer of the resulting scale pyramid. Given the correct down-sampling ratio, the two alternatives produce equivalent results [88, 89].

To demonstrate this equivalence, I will use the ”dog walking” image shown in Figure 3.8. This image is first processed with a set of multiscale detectors[3], as shown in Figure 3.9, with results shown in Figure 3.10. The same image is then used to construct a scale pyramid, and only the smallest scale detector is applied to each level. Results of this latter step are shown in Figure 3.11. Notice that the two maps are nearly identical. Furthermore, the pixel-wise correlation between corresponding edge maps is shown in Table 3.1, with very similar output for the two methods. These results have been found for multiple images.

The equivalence can be seen in a more general way by investigating the

[3]Notice that the detectors in Figure 3.9 are much larger than we would use in practice. This is required so that large Gabor waves fit entirely within the detector window. This is not an issue when using a scale pyramid, because only the smallest scale detector is used.

Figure 3.8: Example image used for discussion of detector scaling in Glimpse.

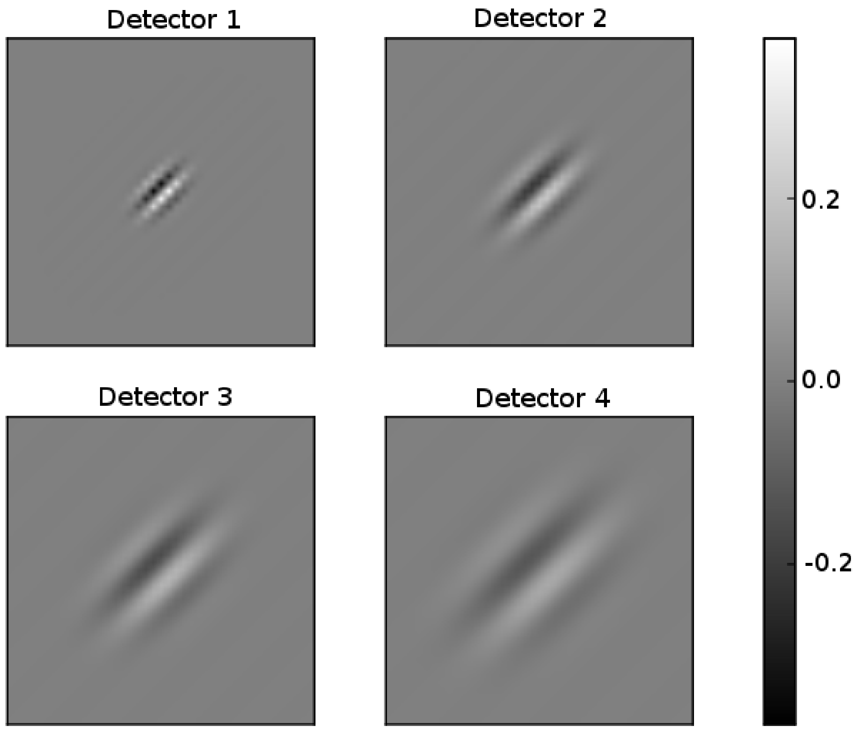


Figure 3.9: Multiscale edge detectors of size 41x41 pixels, as defined by Equation 3.2, were used to avoid clipping in large scale detectors. Color indicates the detector's preferred input, with black indicating low activity, white indicating high activity, and gray indicating no preference.

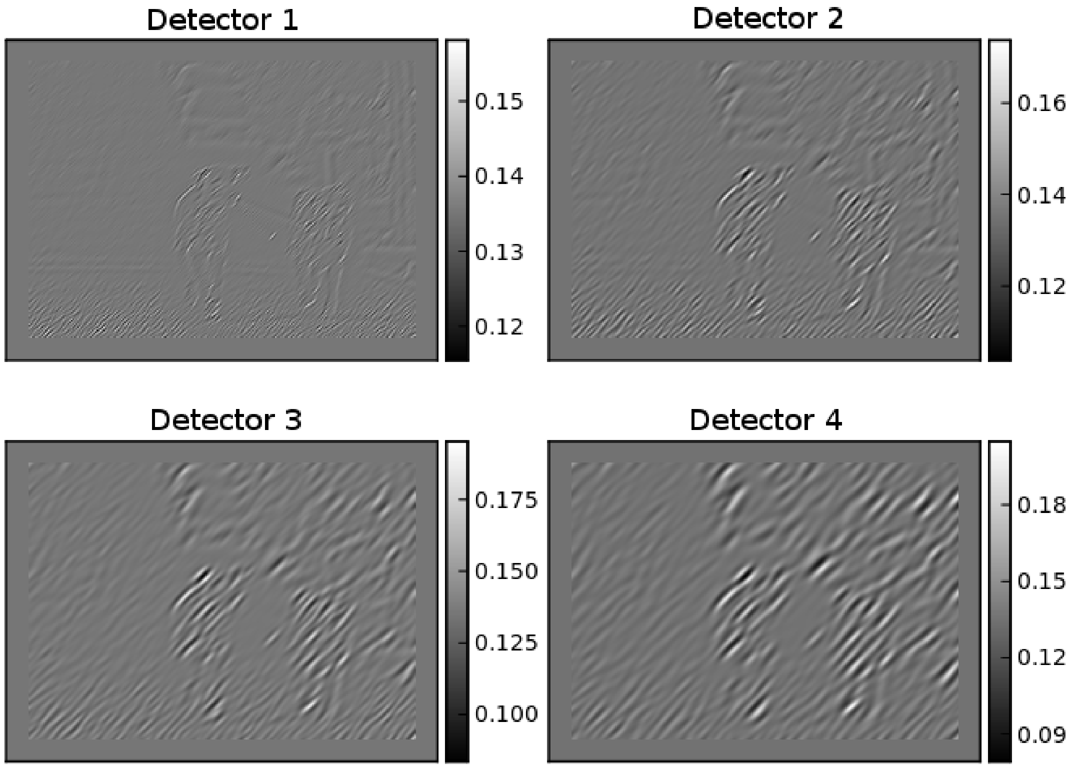


Figure 3.10: Edge maps for multiscale detectors, where the order of response maps corresponds to that in Figure 3.9. Brightness indicates response strength, with white indicating maximum response.

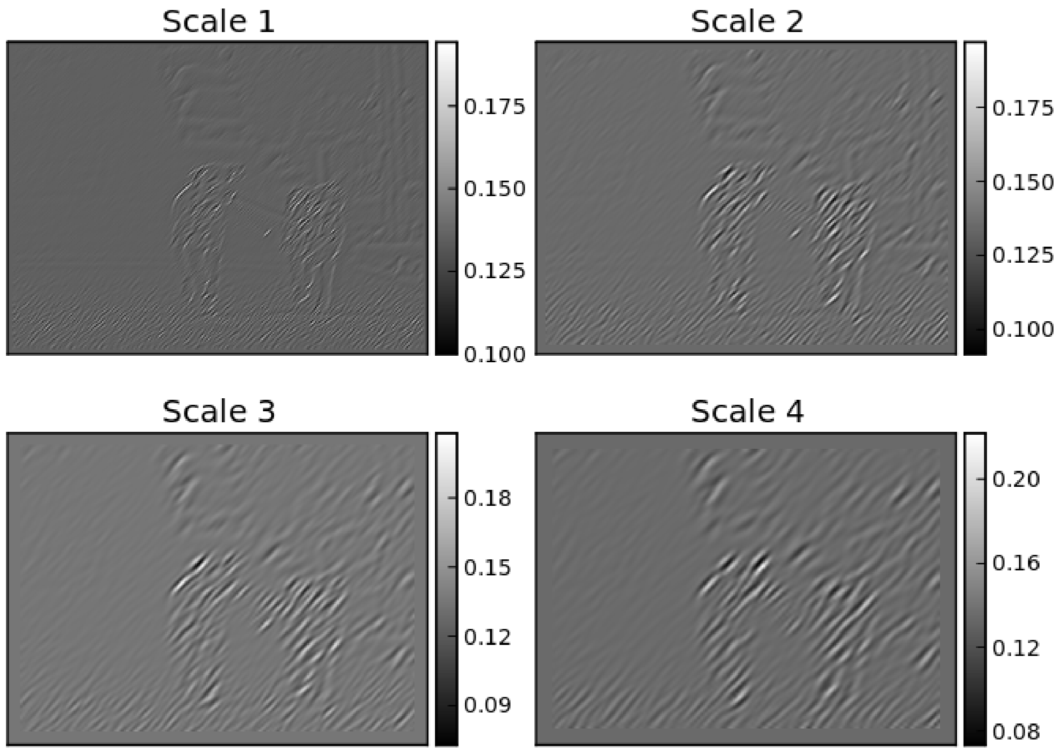


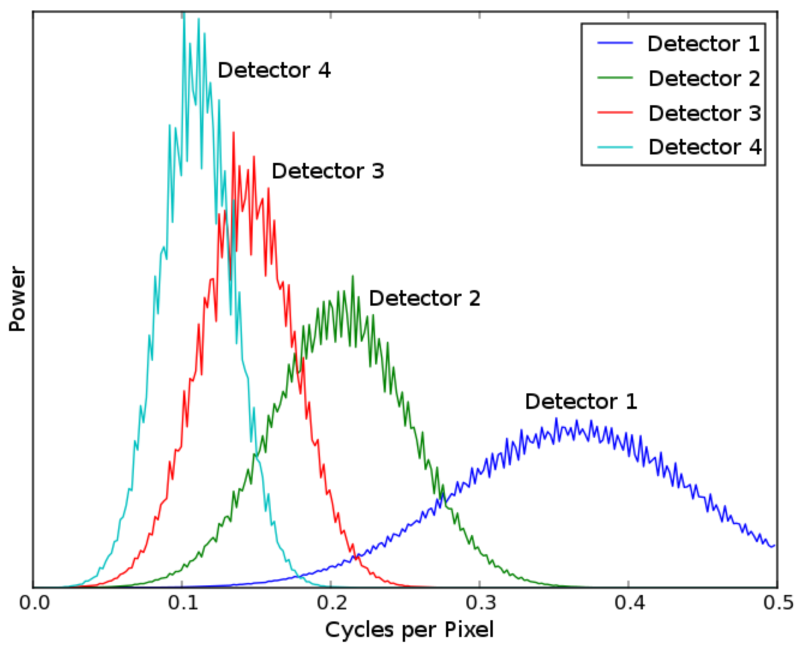
Figure 3.11: Edge maps for a small scale detector applied to a scale pyramid, formatted as in Figure 3.10. Notice that the corresponding maps are very similar between methods.

frequency response for the edge detectors used in each method. Figure 3.12 (top) shows the power spectrum of the four edge detectors shown in Figure 3.9. Figure 3.12 (bottom) shows the corresponding results for a high-scale filter applied to different layers of a scale pyramid. The similarity of the responses for both methods demonstrates that they are sensitive to edges in the same set of scale bands. Thus, edge maps generated by the two methods will be nearly identical, and this property holds regardless of the input.

The result of Glimpse's S1 layer is a set of edge maps, each indicating the presence of an edge at a specific orientation and scale. Ideally, the architecture should minimize runtime costs, allowing for fast "shallow" processing of large-scale content. That is, scale bands containing low-frequency information should incur lower computational cost during analysis, since low frequencies have low spatial resolution. Given the equivalence of scale pyramids to multiscale detectors, I argue that a scale pyramid better supports the two design goals given above. By scaling the image, I can choose a single detector scale that is well adapted to the size of the detector. This ensures that the entire Gabor pattern fits within the window (i.e., it avoids "clipping"), which greatly increases its orientation specificity. Furthermore, note that computing S1 feature maps for large scales is computationally cheaper under a scale pyramid, because the size of the input matrix is smaller for lower frequencies. Indeed, this approach is used in the SIFT [90] and SLF [9] models in the literature.

3.2.2 Edge Detector Size

Next, I consider the optimal size of the S1 edge detector. Since down-sampling always increases the frequency response of the system, the ideal solution is to



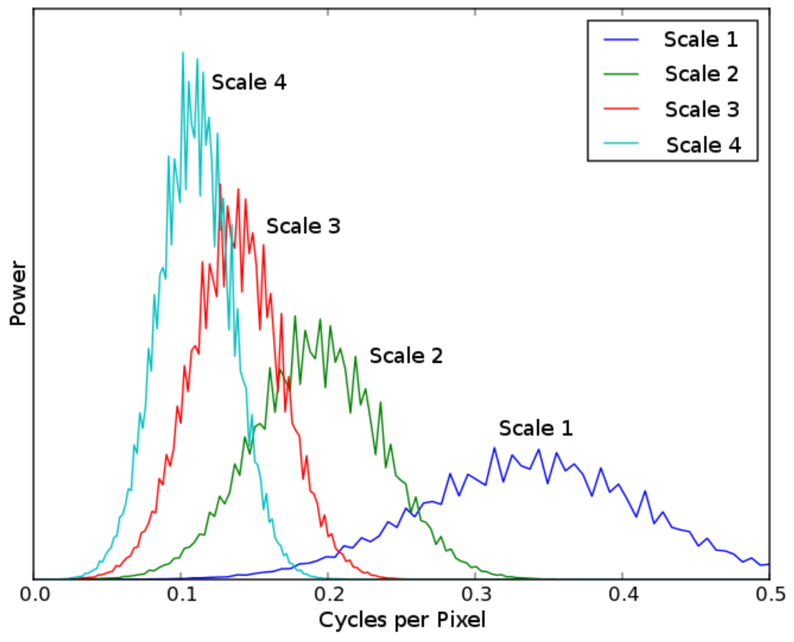
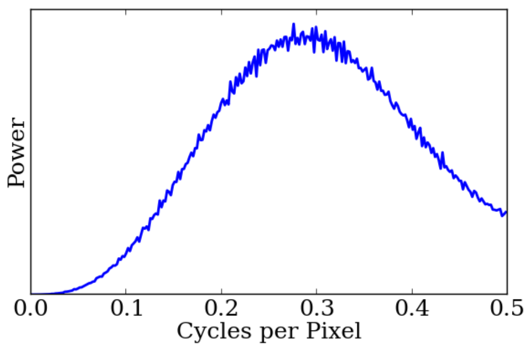


Figure 3.12: (top) Frequency response for multiscale detectors shown in Figure 3.9. The horizontal axis indicates frequency, and the vertical axis indicates the degree of response. Each detector responds to a range of frequencies. (bottom) Effective frequency response when applying a single detector to down-sampled versions of the same image. Notice the strong similarities in the response characteristics.

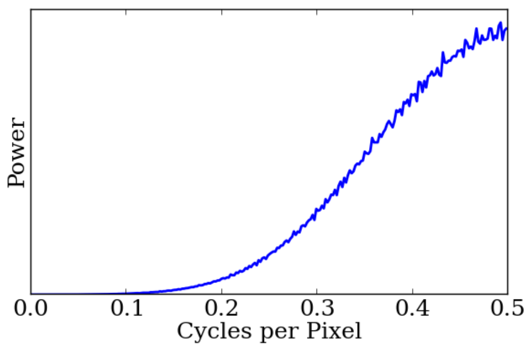
choose the detector with the highest possible frequency—and thus the smallest size. This has the added benefit of minimizing the system's run-time. Note, however, that the detector should not be too small, as aliasing will cause the frequency response to blur for some diagonal orientations. Thus, I measure the power spectrum of the multi-orientation detectors at different sizes[4], looking for the smallest size that maintains a sharp frequency distribution.

Figure 3.13 shows the power spectrum for edge detectors of various areas. As the area of the detector is increased, its frequency response decreases (that is, a larger detector matches a lower-frequency edge), and tightens to match a smaller range of frequencies. Although only a single Gabor orientation is shown here, results for other orientations are nearly identical. Based on the argument given above, the optimal detector will have a high-frequency response in a tight band, which corresponds to a tight peak near the right side of the plot. In this case, an 11x11 pixel window is suggested. Figure 3.14 shows the two-dimensional Fourier transform of the same set of Gabors, which shows the frequency response of the detector for each orientation. Here, frequency is plotted as the distance from the center of the plot, and Gabor orientation is given by the angle from the horizontal. In these plots, the radial width of the high power areas indicates the range of orientations to which the given detector responds. Thus, a patch that is far from the center of the image and which has a small radial width indicates a detector that responds to high-frequency input at a specific orientation. These results also suggest that an edge detector of 11x11 pixels provides the best trade-off between frequency and orientation selectivity.

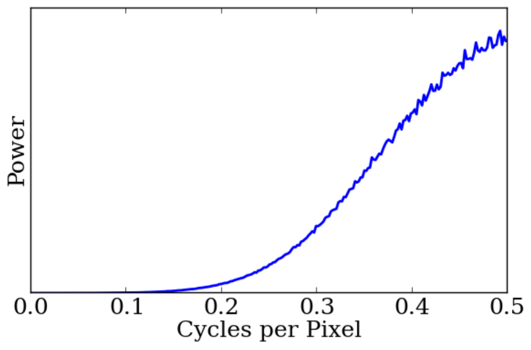
[4]The power spectrum of these small ”images” can be measured without edge artifacts, because detector values decrease to zero at the edges by design.



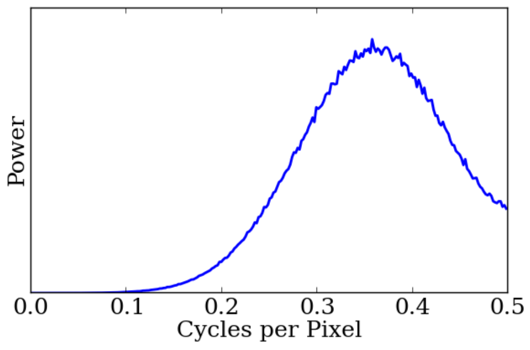
(a) $w = 5$



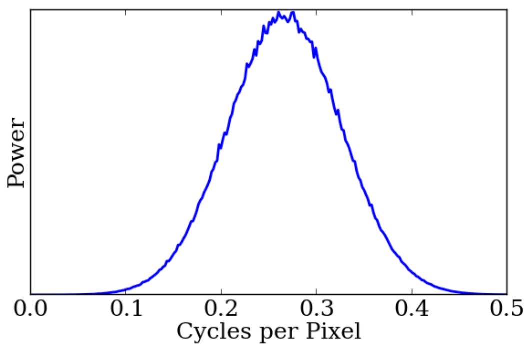
(b) $w = 7$



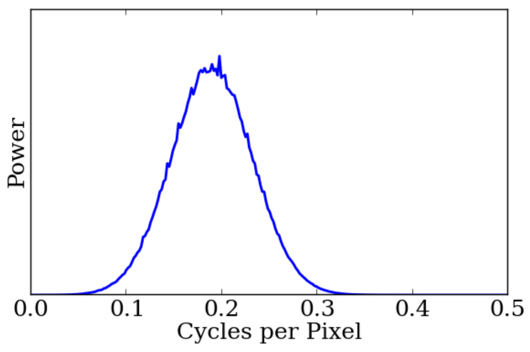
(c) $w = 9$



(d) $w = 11$

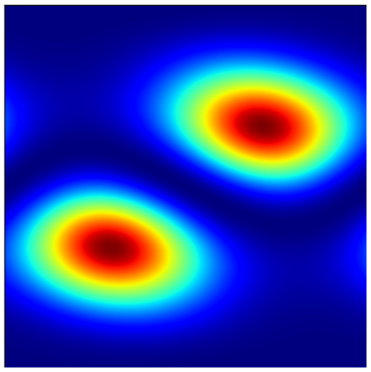


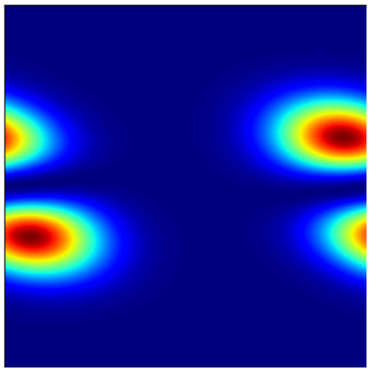
(e) $w = 15$

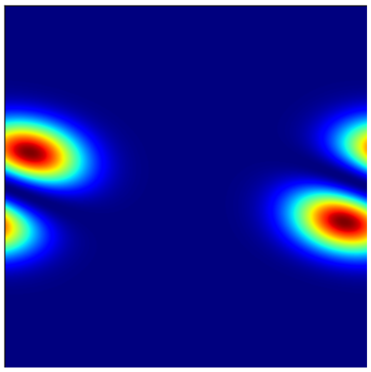


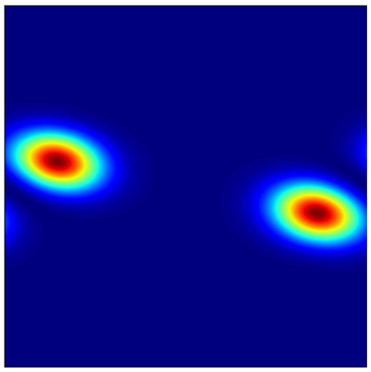
(f) $w = 21$

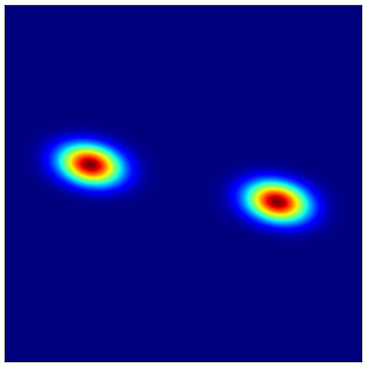
Figure 3.13: Frequency sensitivity for different detector widths w, summarized by the detector's power spectrum. The horizontal axis indicates frequency, and the vertical axis indicates the degree of response. Here, the Gabor wavelength is set to $\frac{1}{4}$ the detector width. Notice that frequency sensitivity drops dramatically for detectors smaller than 11 pixels.


(a) $w = 5$


(b) $w = 7$


(c) $w = 9$


(d) $w = 11$


(e) $w = 15$

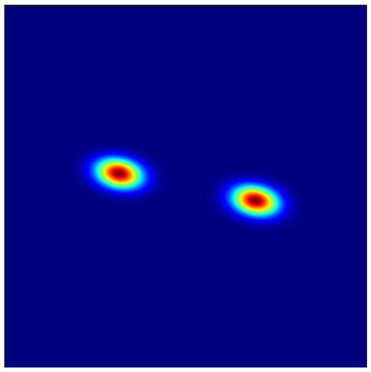

(f) $w = 21$

Figure 3.14: Frequency sensitivity for Gabor detectors of various size, shown as the two-dimensional power spectrum. The center of each plot indicates the detector's responsiveness to low frequency input, and the border of the plot indicates the same for high frequencies. The angle from the horizontal indicates the orientation selectivity, with 0° meaning an input of a horizontal line.

3.2.3 S1 Normalization

Finally, I consider the best way to implement normalization in the S1 activation function (see Equation 3.1). As noted above, normalization is useful to provide contrast gain control, meaning that the S1 unit will be somewhat invariant to a change in the contrast of the input. For example, this allows an edge detector to match well even in a region of low contrast. However, a direct normalization of the input as

$$x' = \frac{x}{\|x\|} \tag{3.12}$$

has the undesirable property of magnifying noise in regions with extremely low light (seeing "ghosts in the darkness").

To avoid amplifying this noise, I bias the denominator in Equation 3.12 to guarantee that it is bounded by some constant. In the simplest case, we can use an additive bias of the form

$$\mathbf{x}' = \frac{\mathbf{x}}{\|\mathbf{x}\| + b} \, . \tag{3.13}$$

This is similar to the proposed "divisive normalization" model of contrast gain control in cortex [91, 92]. The approach ensures that regions with gain less than b are not amplified, but fails to appropriately scale regions whose gain is larger than b. In fact, only regions with very high contrast will be mapped to have near-unit norm. This is illustrated in Figure 3.15a, which shows the behavior of an additive bias for different values of b.

As an alternative, a conditional bias takes the form

$$\mathbf{x}' = \frac{\mathbf{x}}{\max\left(\|\mathbf{x}\|, b\right)} \, , \tag{3.14}$$

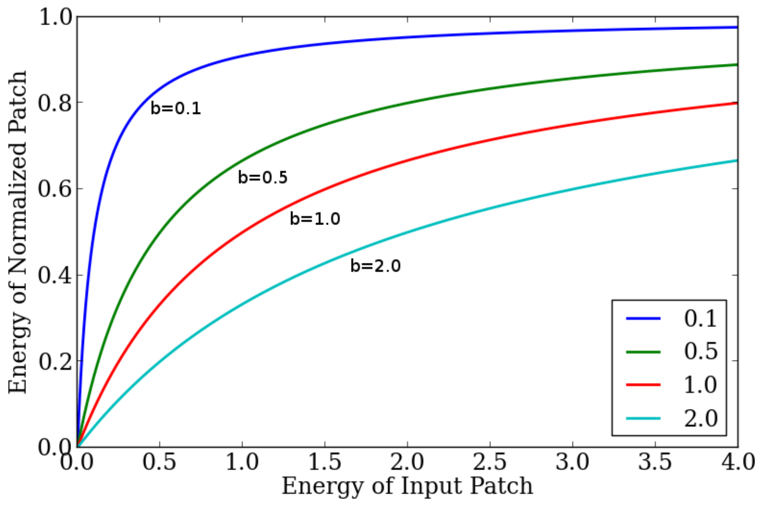
which maps all regions with gain larger than b to the surface of a spheroid with radius b (see Figure 3.15b). Only those regions lying within the sphere are suppressed. This achieves gain control for those regions lying on, or outside, the unit sphere, and treats those lying within the sphere as noise. This is illustrated in Figure 3.15b, which shows that the input patches are properly normalized even when the bias is large. Due to this behavior, I use a conditional bias (Equation 3.14) in the Glimpse model.

3.3 Comparison to Previous Models

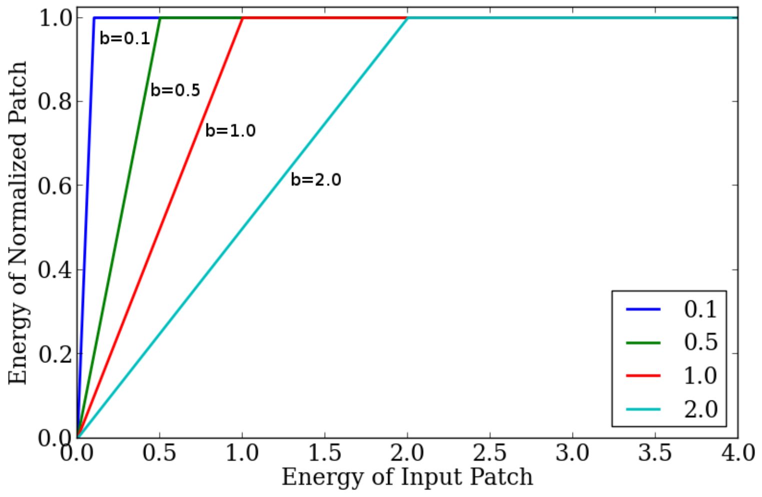
The goal of this work is to uncover general properties of alternating multilayer architectures. However, the use of a new model risks introducing a qualitative shift in behavior, and thus to non-generalizable results. As a result, I performed extensive validation to verify that the Glimpse model captures the qualitative behavior of similar hierarchical models found in the literature.

Implementations for the HMAX and SLF models were first downloaded from the internet [93]. I then measured the performance of all three models (Glimpse, SLF, and HMAX) using imprinted prototypes on tasks commonly used in the literature. The SLF model has been used in a number of studies (e.g., [18]), and has often been shown to out-perform the HMAX model [9]. Thus, SLF provides an additional reference point that is helpful for validating the Glimpse model.

Figure 3.16 shows a diagram of the HMAX model used in this work, with values for some of the more important parameters. This is an approximation of the model used by Serre et al. [6, 2]. Compared with the diagram of the Glimpse model (Figure 3.1), there are two significant differences. First, the HMAX S2 layer uses prototypes of six different sizes, while Glimpse uses a single prototype size. Second, the S2 prototypes used in the HMAX implementation



(a) Additive: $\mathbf{x}' = \frac{\mathbf{x}}{\|\mathbf{x}\|+b}$



(b) Conditional: $\mathbf{x}' = \frac{\mathbf{x}}{\max(\|\mathbf{x}\|,b)}$

Figure 3.15: Effect of normalization on S1 activity for two different approaches. (a) Input activity is bounded with an additive bias as $x' = \frac{x}{\|x\|+b}$, and the behavior is plotted for various values of the bias b. This causes the input to be suppressed even when its energy was initially large. (b) Input activity is bounded with a conditional bias as $x' = \frac{x}{max(\|x\|,b)}$. Low-energy inputs are suppressed, while the response to high energy patches is contrast invariant.

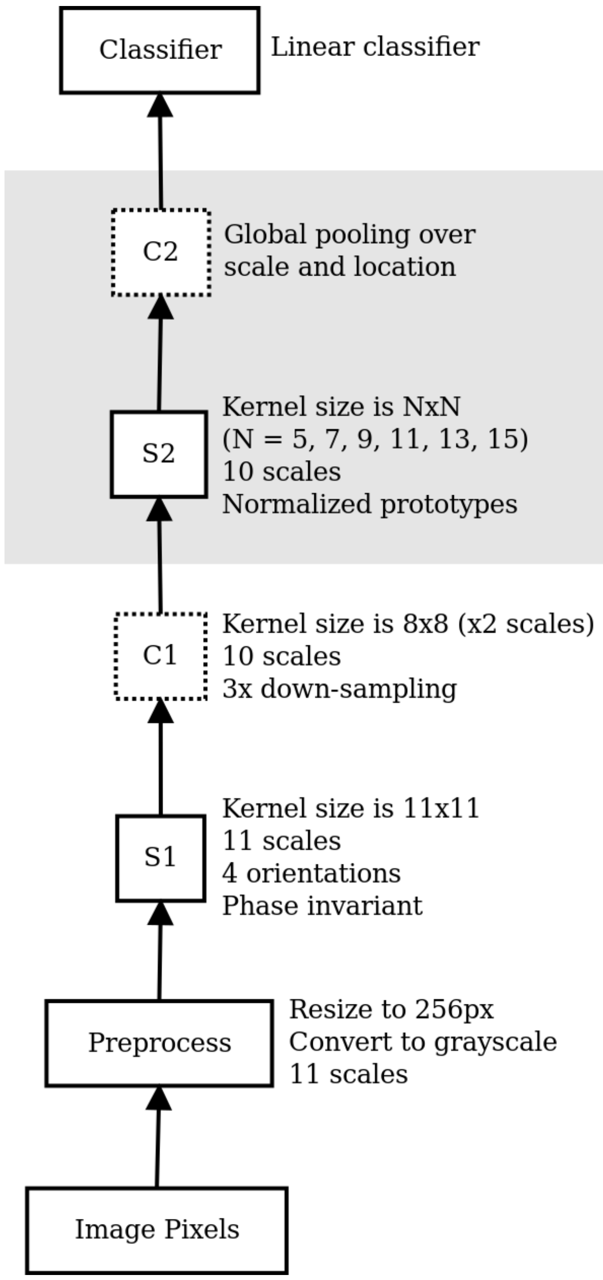


Figure 3.16: Architecture diagram for the HMAX model. The default parameter choices—which are used in this work—are shown to the right of the diagram.

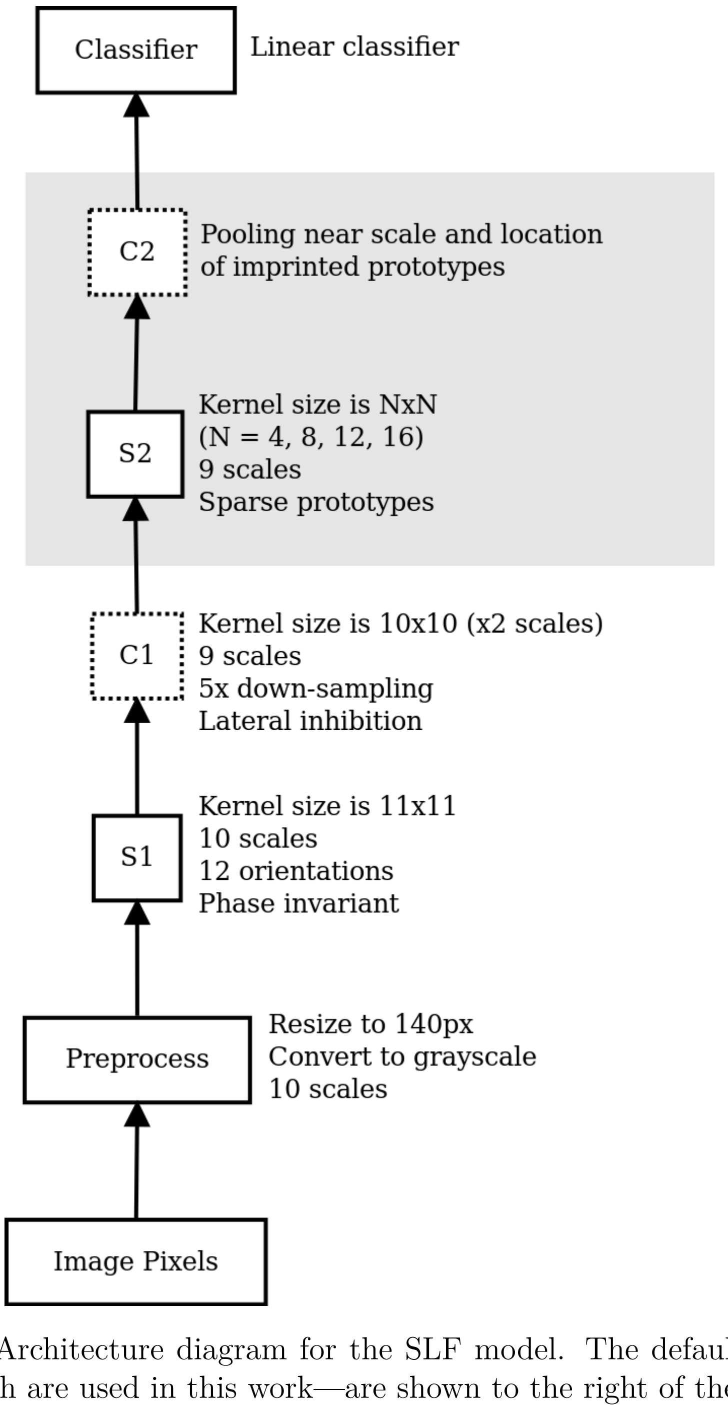


Figure 3.17: Architecture diagram for the SLF model. The default parameter choices—which are used in this work—are shown to the right of the diagram.

are “normalized”, meaning that the total activation within each prototype is scaled to have unit (L2) norm[5].

Similarly, Figure 3.17 shows a diagram of the SLF model used in this work. Differences between Glimpse and SLF are three-fold. First, note the use of “lateral inhibition” at the C1 layer, which means that C1 units at a given location *compete*. As a result, less active units have their output suppressed—or inhibited—by more active units. Second, the S2 layer in the SLF model uses “sparse prototypes”, which means that an imprinted prototype uses only the most active orientation at each location. When comparing such a prototype to an input patch, only these active orientations are considered. This is argued to increase the model’s robustness to clutter [9]. Third, the C2 layer of the SLF model pools over a limited area of S2 activity, rather than pooling globally over all scales and locations. This area is given by a small neighborhood around the prototype’s original location, and includes the S2 activity for the scale immediately above and below the imprinted scale.

Results are shown in Figure 3.18 for the HMAX, SLF, and Glimpse models on subsets of *Caltech101* [55], the *Animals* task of Serre et al. [2], and the synthetic tasks of Pinto et al. [18]. Following Serre et al. [6], each model uses 4075 C2 features learned by imprinting, and a linear-kernel SVM for classification. Performance is reported as the area under the ROC curve (AUC). I performed five independent trials for each model and corpus, and imprinted new prototypes in each trial. The height of each bar shows the mean performance across those trials, and error bars show one standard error. Figure 3.19 shows a similar comparison for the tasks of Fergus et al. [56], where Glimpse is compared with the SLF model. Performance for the HMAX model was omitted in this and later

[5]Note that this normalization process is not applied to the S2 unit’s input.

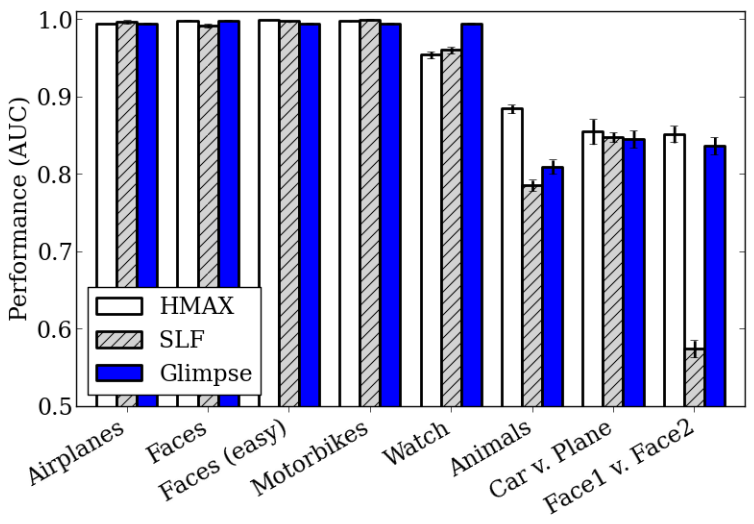


Figure 3.18: Performance comparison (AUC) for 4075 C2 features using HMAX (white), SLF (hatched gray), and Glimpse (blue) models. Datasets include subsets of Caltech101 categories—*Airplanes*, *Faces*, *Faces (easy)*, and *Watch*—the *Animals* dataset of Serre et al. [2], and the synthetic tasks of Pinto et al. [18]. Error bars indicate standard error over five independent trials.

experiments, since 1) the behavior of SLF and HMAX is often quite similar, while 2) applying the HMAX model takes considerably more time.

From these two figures, we see that the behavior of Glimpse appears to be consistent with that of the reference models across all tasks. This suggests that the model has the salient features of previous work. Thus, there is a good chance that interesting results found for the Glimpse model would also apply for other hierarchical models.

However, I do note differing results for two tasks. On the *Animals* task, the HMAX model displays performance that is significantly above that of the other models. This is unsurprising, as it is the task for which the model's parameters were optimized. Additionally, the SLF model displays performance that is significantly below that of the other models on the *Face1 v. Face2* task. This result is likely due to the use of localized pooling at C2 in the SLF model, which

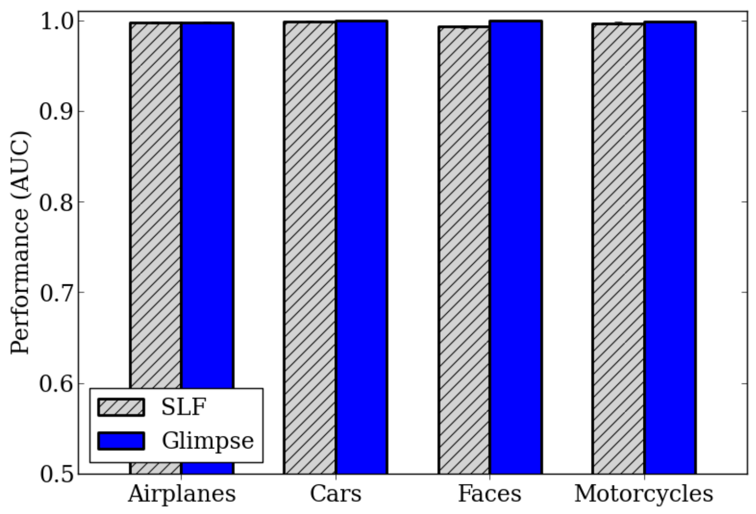


Figure 3.19: Performance comparison (AUC) for 4075 C2 features using SLF (hatched gray) and Glimpse (blue) models. Datasets are due to Fergus et al. [56]. Error bars indicate standard error over five independent trials. (Note that a lack of variation leaves the error bars difficult to see.)

is inappropriate in synthetic tasks that vary the object's location. Thus, it is surprising that SLF performance does not suffer on the *Cars v. Planes* task, as that too includes strong variation in object location. Interestingly, these results show that model performance is quite saturated across all *Caltech101* and Fergus et al. tasks, indicating that these tasks are of limited use for object recognition research. This result is investigated further in Chapter 4.

Figure 3.20 compares the SLF and Glimpse models on the *Cats v. Dogs* tasks. Glimpse performs better—often significantly better—than the SLF model on all tasks. As discussed above, this may be due to the localized pooling operation at C2 of the SLF model.

Thus far, the behavior of Glimpse has only been investigated for very large networks, that is, those employing a large number of prototypes. It is possible that two models could behave similarly in this case, while showing qualitatively different behavior for less complex S2 layers. To investigate this, I measure the

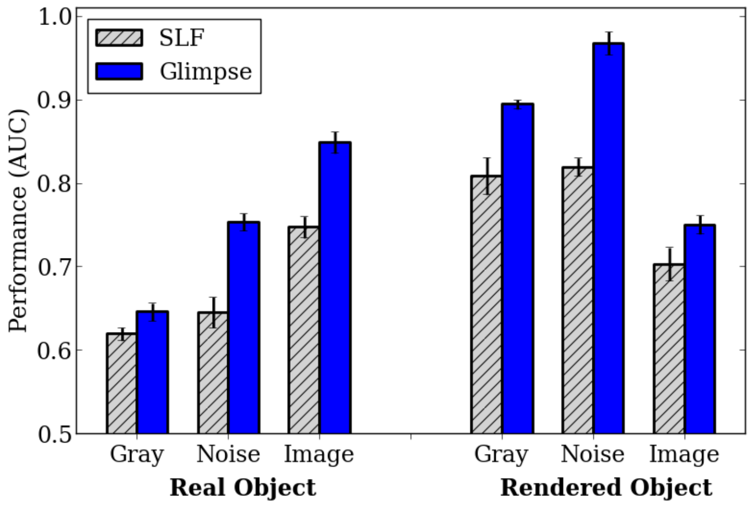
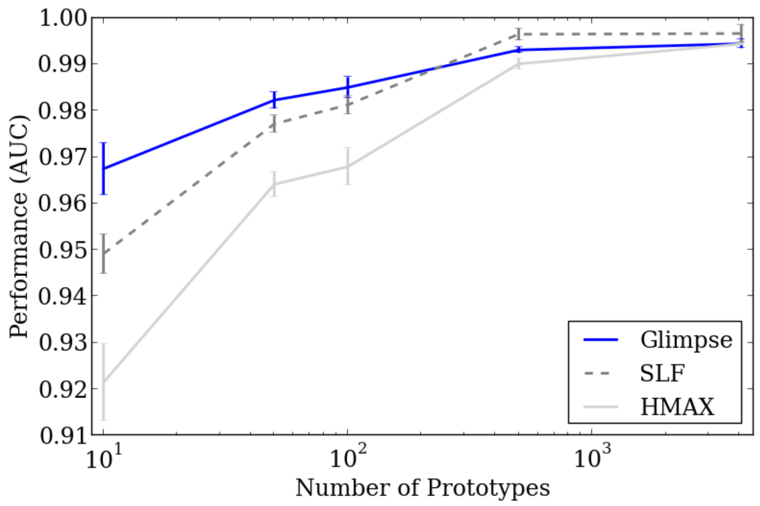


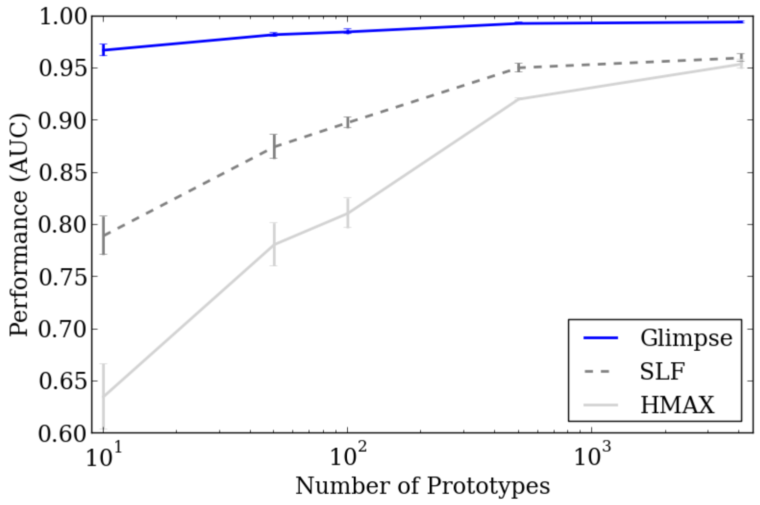
Figure 3.20: Performance comparison (AUC) for 4075 C2 features using SLF (hatched gray) and Glimpse (blue) models. Tasks are from the *Cats v. Dogs* dataset. Results on the left give performance for *photographic* foreground objects, while results on the right give the same for *rendered* foreground objects. Results are given for different types of backgrounds, including uniform color (Gray), randomly generated images following a $\frac{1}{f}$ frequency distribution (Noise), and randomly chosen photographs of outdoor scenes (Image). Error bars indicate standard error over five independent trials.

performance for each of the models as the number of prototypes is increased, and thus capture the "scaling behavior" of each model.

Results are shown in Figures 3.21-3.23 for a representative subset of the tasks in Figure 3.18. As before, these results show that Glimpse has captured the qualitative behavior reported in the literature, even with respect to changes in the number of units in the S2 layer. Additionally, it is interesting to note the lack of a consistent ranking for the three models, since each model shows superior performance on at least one of the four datasets. This is important to note for those developing their own models, so that no single model is considered a "gold standard".



(a) *Airplanes* (Caltech101)



(b) *Watch* (Caltech101)

Figure 3.21: Performance comparison (AUC) on *Caltech101* tasks for varying number of C2 features using HMAX (gray), SLF (dashed), and Glimpse (blue) models. Error bars show one standard error.

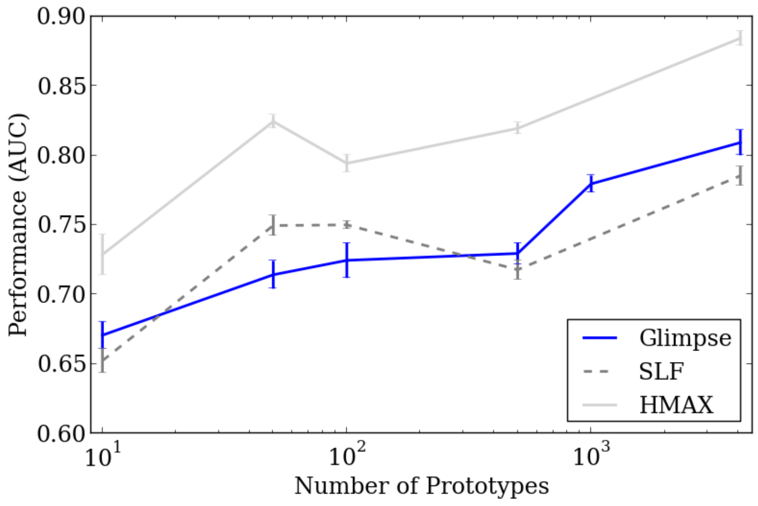


Figure 3.22: Performance comparison (AUC) on the *Animals* task for varying number of C2 features using HMAX (gray), SLF (dashed), and Glimpse (blue) models. Error bars show one standard error.

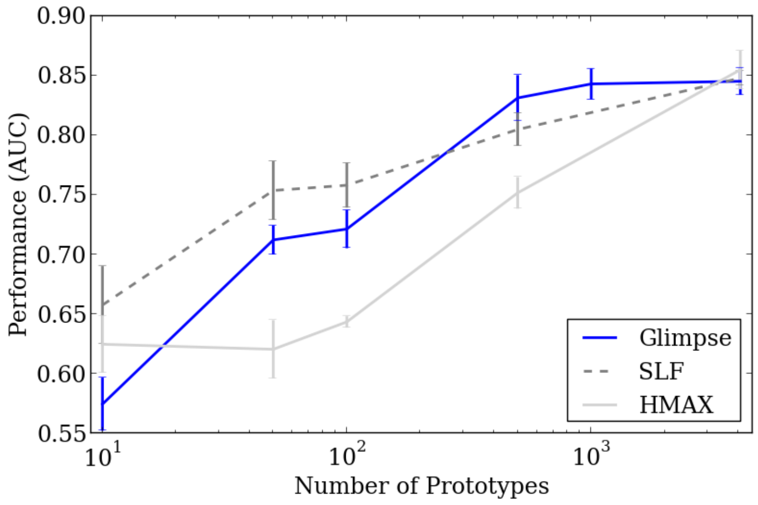


Figure 3.23: Performance comparison (AUC) on the *Cars v. Planes* task (variation level three) for varying number of C2 features using HMAX (gray), SLF (dashed), and Glimpse (blue) models. Error bars show one standard error.

Chapter 4

The Role of Shape Prototypes

In this chapter, I describe a series of detailed experiments to investigate how learned shape prototypes in alternating multilayer models affect classification performance. Surprisingly, I find that the classification performance of networks using randomly generated prototypes—with no apparent spatial structure—perform in a nearly identical way to networks using prototypes imprinted from natural images in a way so as to capture "useful" shape components.

The rest of this chapter is organized as follows. Section 4.1 discusses the basic methodology used in all experiments for this chapter. The benefit of invariant representations is considered in Section 4.2. The role of shape prototypes is investigated in Section 4.3. Finally, a discussion of the results is given in Section 4.4. This work has been published in a shorter format as [54].

4.1 Methods

Unless otherwise noted, all experiments in this work use the following experimental methodology. In each trial, a different subset of half the images was chosen for training, with the other half reserved for testing. Each trial chooses a different set of prototypes, and performance is reported on the test set. Results are somewhat stochastic, because the classifier is applied with a different set of features and a different set of testing images in each trial. Thus, each experiment is repeated five times, and the average performance is reported.

Beyond the features derived from C2 activity (see Chapter 3), the experiments below use two additional types of features. First, "pixel" features are computed by converting the image to grayscale, and concatenating image rows to form a single vector. Second, "C1" features are derived from the activity of all units in the C1 layer by concatenating units for all positions, scales, and edge orientations to form a single vector. Both cases result in a feature space of very high dimensionality[1]. Techniques for dimensionality reduction, such as PCA, were not used, because the goal was to give the invariant representation every opportunity to succeed.

Note that a fixed feature space is required for many classifiers, including SVMs. That is, the number of features representing each image must be constant. However, the dimensionality of a pixel or C1 representation depends on the size of the input image[2], and thus the size of the images was constrained in these experiments. For tasks derived from the *Caltech 101* dataset and the tasks of Fergus et al., the image size was constrained by removing border pixels as needed. Fortunately, this is likely to have little effect on Glimpse's performance, as foreground objects are intentionally placed in the center of each image. All remaining tasks employ images of the same size.

[1]The dimensionality of a C1 representation is at least 20,000 features. This size increases as the image becomes elongated. A pixel representation is even larger, with approximately 50,000 features for a square image.

[2]This is not the case for a C2 representation, which provides one feature per prototype.

4.2 Role of Invariant Representations

I first investigate the benefit of Glimpse's C2 features relative to a simpler image representation. Remember from Section 2.2 that Glimpse's C2 representation is described as *invariant*, because C2 activity is unaffected by changes to an object's location or scale. Here, I specifically investigate the role of this invariance property in the success of the model.

To do this, a given task is performed using a representation composed of C2 activity, and Glimpse's performance is measured. The same task is then performed using a representation composed of C1 features, and again using pixel features. These latter representations lack the strong invariance properties of the C2 layer, and thus provide a useful performance baseline.

The results are shown in Figure 4.1, which reports performance across a number of datasets for raw pixel features (light gray), C1 features (dark gray), and 4075 C2 features (blue). Performance is reported as the mean area under the ROC curve (AUC) across five independent trials, with error bars showing one standard error. The set of prototypes is imprinted independently for each trial.

Interestingly, the *Caltech 101* tasks do not appear to require an invariant representation. In most cases, performance for C1 features is almost identical to that using C2 features. In fact, many tasks can be performed using only raw pixel data, and I am thus forced to conclude that these tasks are of little use in the study of invariant object recognition. These results agree with similar findings of Pinto et al. [17]. Consequently, these datasets will not be used in the experiments described in the next sections.

In contrast, results for the remaining tasks showed a significant benefit for

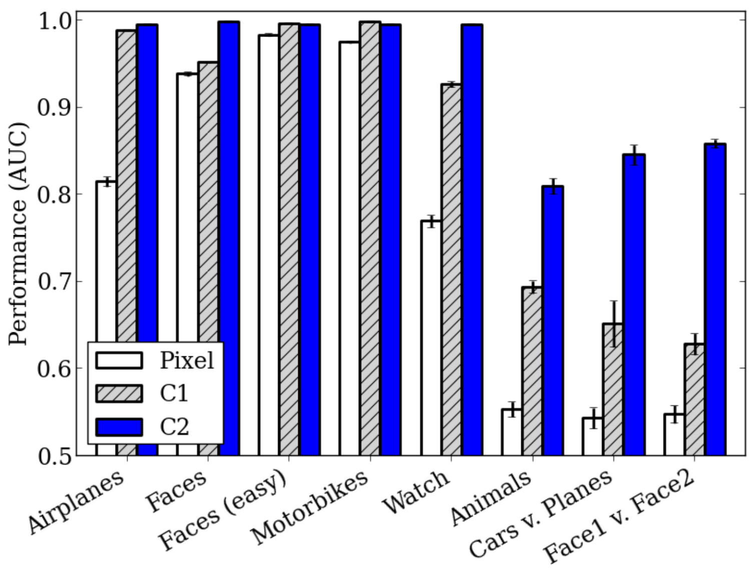
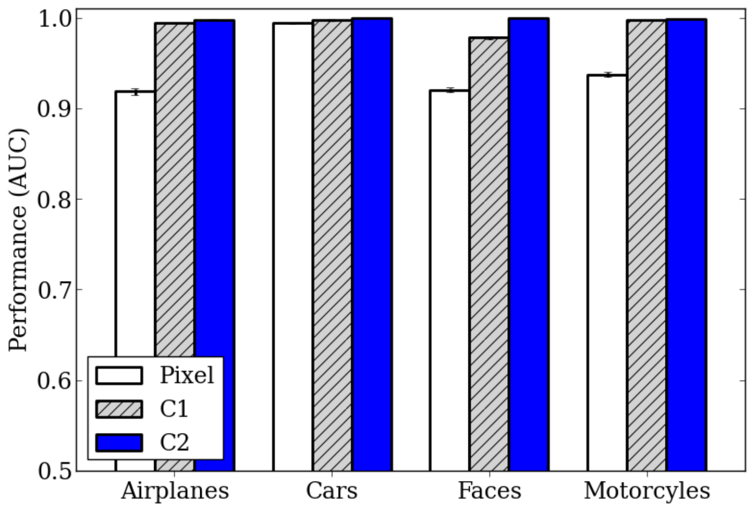


Figure 4.1: Performance comparison (AUC) for raw pixels (white), C1 features (hatched gray), and 4075 C2 features (blue). Datasets include subsets of *Caltech 101* categories—*Airplanes*, *Faces*, *Faces (easy)*, *Motorbikes* and *Watch*—the *Animals* dataset of Serre et al., and the synthetic tasks of Pinto et al.—*Cars v. Planes* and *Face1 v. Face2*. The vertical axis indicates the mean performance over five independent trials, and error bars indicate standard error over five independent trials.

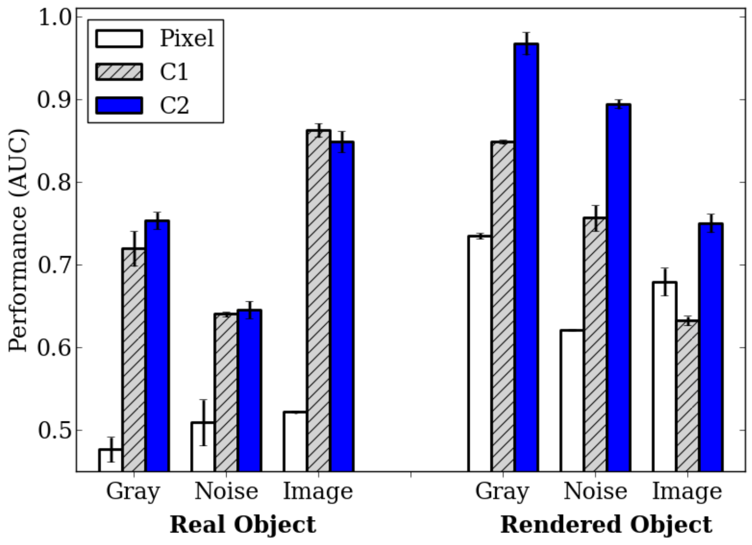
invariant representations. For the *Animals*, *Cars v. Planes*, and *Face1 v. Face2* tasks, pixel features were effectively useless for recognition. In addition, while C1 features provide a useful representation for these tasks, an invariant representation using C2 features is significantly more useful.

Figure 4.2a shows results for the tasks of Fergus et al. These results are quite similar to those for the *Caltech 101* tasks in Figure 4.1. Again, pixel features account for nearly all of the model's performance, and an invariant representation provides little additional benefit. Figure 4.2b shows results for the various *Cats v. Dogs* tasks. On the left half of the plot, results are shown for real objects—those captured from natural imagery—on the three background types. While the invariant C2 representation is clearly superior to raw pixel features, a C1 representation completely accounts for the performance increase. However, note the model's surprisingly high performance in the presence of natural image backgrounds (i.e., bars labeled "Image" on the left side of Figure 4.2b). It is possible that this is caused by unintentional regularities in the background for each object, such as a tell-tale difference in the edge statistics of "cat" backgrounds vs "dog" backgrounds.

The right half of Figure 4.2b shows results for the *Cats v. Dogs* tasks using rendered objects. In this case, an invariant C2 representation provides a clear benefit over C1 features. Additionally, the performance for C1 and C2 features decreases as the backgrounds become more complex, with the highest performance for simple gray backgrounds, lower performance for backgrounds containing randomly generated noise, and the lowest performance for backgrounds containing complex image natural images. However, note the strong performance for raw pixel features, which may indicate an insufficient variation in the presentation of objects or their backgrounds.



(a) Datasets from Fergus et al.



(b) *Cats v. Dogs*

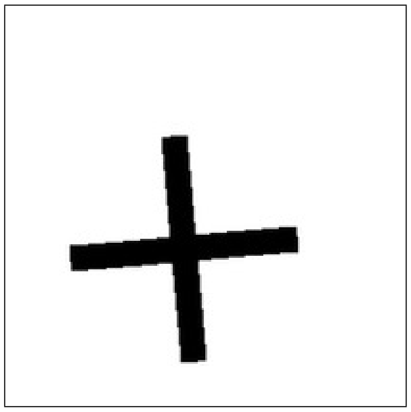
Figure 4.2: Performance as in Figure 4.1 comparing pixel (white), C1 (hatched gray), and 4075 C2 (blue) features.

In summary, these results suggest that Glimpse's C2 representation provides a significant benefit for performing object recognition. However, many tasks that are used often in the literature do not require an invariant representation, and thus provide little information about the efficacy of object recognition models.

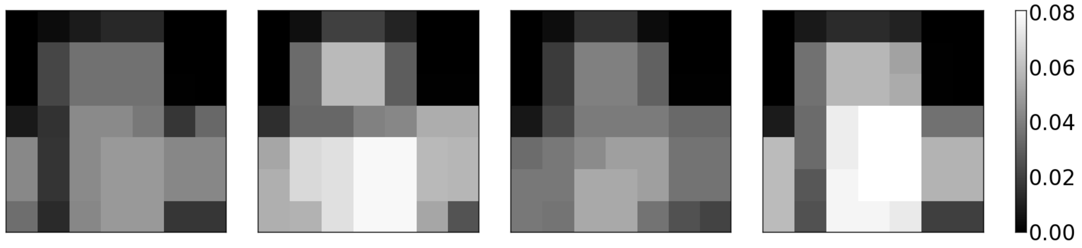
4.3 Importance of Shape

In this section, I test the "shape dictionary" hypothesis discussed in Section 2.3, which suggests that an imprinted representation is useful because it captures important "shape-based" properties of objects [2]. To isolate the benefit of learned shape features, I measure the impact on Glimpse's performance when this information is degraded. In the first step, a set of prototypes is imprinted as previously described. Performance is then measured on the same task when these prototypes are "shuffled", that is, when the order of activation values in each prototype are randomly permuted. This process is demonstrated in Figure 4.3. As a result, the shape information—that is, spatial and orientation configuration—is scrambled, while the basic activation statistics within each prototype are maintained.

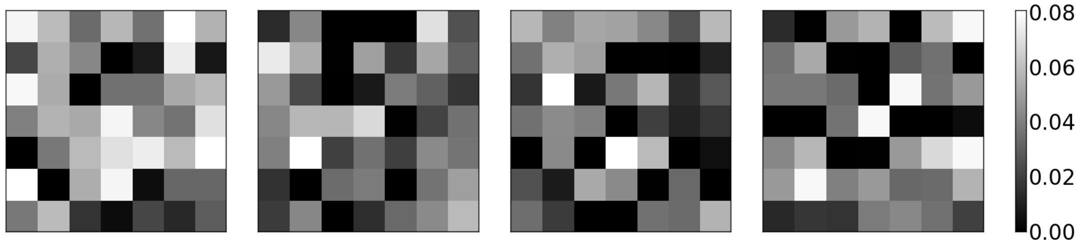
Finally, performance is measured for a set of unlearned, "shape-free" prototypes. This set is constructed randomly, where each prototype component is drawn independently from a uniform distribution over activation values. (This approach should not be confused with imprinting, in which randomness is used to choose the location of image regions.) Due to their construction, these prototypes capture neither the spatial information, nor the activation statistics of learned shape prototypes. Recent evidence suggests that various kinds of random features can be surprisingly useful in hierarchical networks [64, 94, 95], though



(a) Image patch.



(b) Imprinted activation.



(c) Shuffled activation.

Figure 4.3: An example in which (a) an image patch is used to construct (b) an imprinted prototype. The figure shows the activations—white denotes high activation, black denotes low activation—for a neighborhood of C1 units, with one plot for each edge orientation. The activation values within this prototype are then permuted to create (c) a shuffled prototype. Note that activation is permuted across orientation bands as well as locations.

the reasons for this behavior are still unclear.

Specifically, random prototypes are constructed as observations of a multivariate, independent, and identically-distributed random variable, with each component given by

$$p_i \sim \text{Uniform}(0, 1) \ .$$

An example of such a prototype matrix is shown in Figure 4.4.

Additionally, random and shuffled prototypes are both *sparse* and *gain invariant*, which are properties that imprinted prototypes lack[3]. Sparsity is enforced by lateral inhibition across orientation bands at each location, where the activity x_i for the i^{th} orientation at location ℓ is scaled as

$$x'_i = \frac{x_i}{a_\ell} \ ,$$

and where $a_\ell = \sqrt{\sum x_j^2}$ measures the total energy for all units at location ℓ. This has the effect of suppressing less active orientations. Gain invariance is achieved by constraining the total energy of the input and prototype, where S2 activation is computed as

$$S2\left(\frac{\mathbf{x}}{\|\mathbf{x}\|}, \frac{\mathbf{p}}{\|\mathbf{p}\|}\right)$$

for input $\mathbf{x}$ and prototype $\mathbf{p}$, and activation function $S2\left(\cdot, \cdot\right)$ as given in Equation 3.5 (Page 28). This allows the comparison between input and prototype to be unaffected by a change in contrast in $\mathbf{x}$.

I found that performance for shuffled and random prototypes was substantially lower without the sparse contrast-invariant activation function. In contrast, such an activation function significantly *decreased* performance when im-

[3]This approach was inspired by the architecture of the PANN model [29].

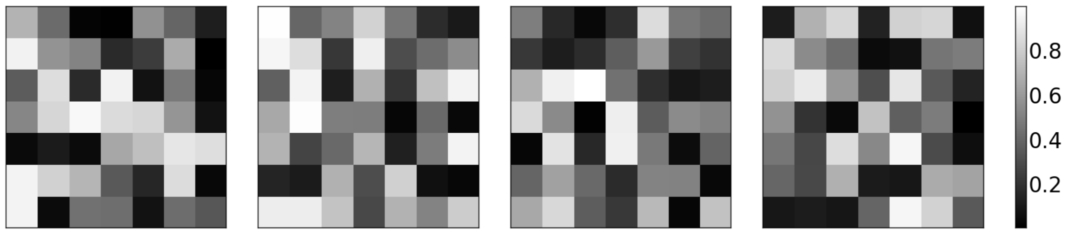


Figure 4.4: An example of a random prototype, with plots corresponding to the four orientation bands. Each component is chosen independently from a uniform distribution.

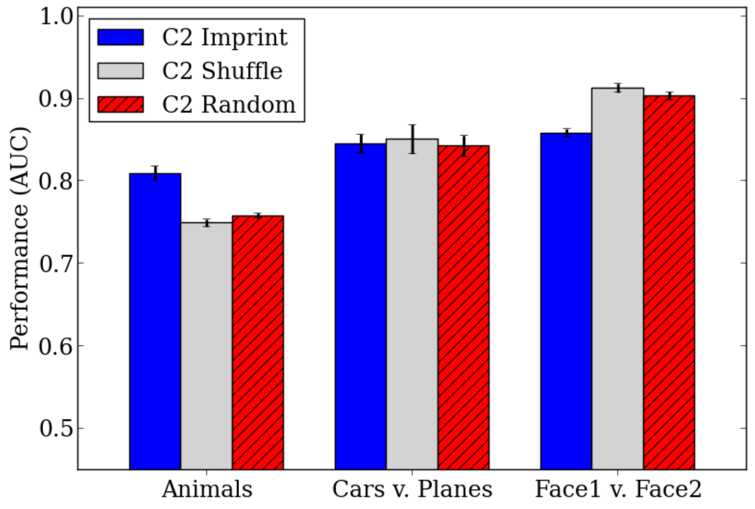
printed prototypes were used. The reason for this relationship is currently unknown. However, I hypothesize that this result indicates that imprinted and random prototypes operate in different ways. I believe that a "good" set of imprinted prototypes should contain examples of specific and discriminative shapes from the domain. These are qualities that a set of random prototypes will lack. In contrast, a "good" set of shuffled or random prototypes may need to simply contain prototypes that are sufficiently different from one another, which are qualities that a set of imprinted prototypes will lack.

Glimpse's performance for imprinted, shuffled, and random prototypes is shown in Figure 4.5. Performance is reported as mean AUC over five independent trials for the datasets identified in Section 4.2. Each feature vector is based on 4075 prototypes. Figure 4.5a shows this comparison for the *Animals* task, and for variation level three of the *Cars v. Planes* and *Face1 v. Face2* tasks. Figure 4.5b shows the same comparison for the *Cats v. Dogs* tasks. Across all datasets, I found that the degradation of shape information has surprisingly little impact on Glimpse's performance. In fact, this degradation led to an improvement in performance for the *Face1 v. Face2* task.

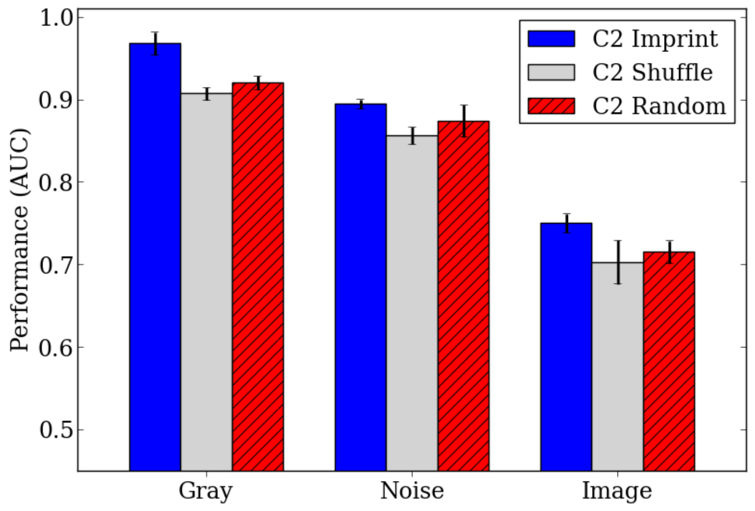
Figures 4.6a and 4.6b show a similar comparison (without "shuffled" prototypes) for the *Cars v. Planes* and *Face1 v. Face2* tasks, respectively, as the level of variation is increased. Following Pinto et al. [18], performance is plotted as the variation level is increased. As before, performance is plotted as mean AUC, with error bars showing one standard error. Results for imprinted prototypes were similar to those reported by Pinto et al. [18], with performance dropping as the variation level was increased. However, I find that random prototypes also perform this way, with behavior that is nearly identical to that of imprinted prototypes. Critically, I find that a (invariant) C2 representation based on random prototypes performs well even when a pixel or C1 representation does not. Thus, an invariant representation is crucial, while shape is not.

Taken together, these results seem to contradict the "shape dictionary" hypothesis. A number of possible explanations for these results were considered. I first considered the possibility that a sufficiently large network is simply robust to a bad choice of prototypes. That is, it is possible that any sufficiently large set of prototypes would lead to the behavior seen in Figure 4.6. To investigate this, I compare the performance of these two representations using different numbers of prototypes, with results shown in Figure 4.7. Performance was quite similar even when using only 10 prototypes. Regardless of the size of the network, I was unable to find a significant difference in performance between a representation based on random versus imprinted prototypes.

Alternatively, I considered the possibility that a random prototype provides a weakly discriminative feature when used in isolation, but that a *group* of random prototypes could be strongly discriminative. This might be the case if each feature provides an independent piece of information about the object, which when combined is enough for recognition. This is inspired by a machine learning technique called *boosting* [96], in which a number of "weak" classifiers are

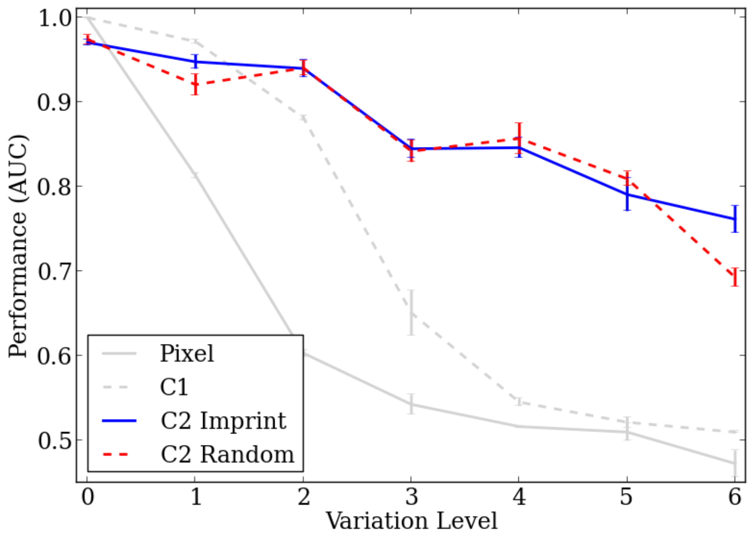


(a) *Animals* and synthetic tasks of Pinto et al. Variation level three is used for synthetic tasks.

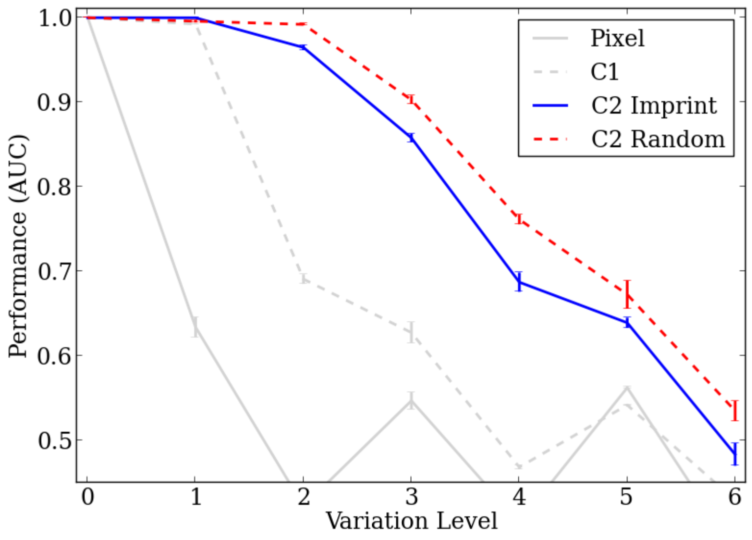


(b) Rendered *Cats v. Dogs* on various backgrounds.

Figure 4.5: Comparison of Glimpse's performance across different tasks, using 4075 imprinted (blue), shuffled (gray), and random (hatched red) prototypes. The vertical axis shows the mean AUC over five independent training and testing splits, and error bars show the standard error. Results for the *Cars v. Planes* and *Face1 v. Face2* tasks use variation level three.

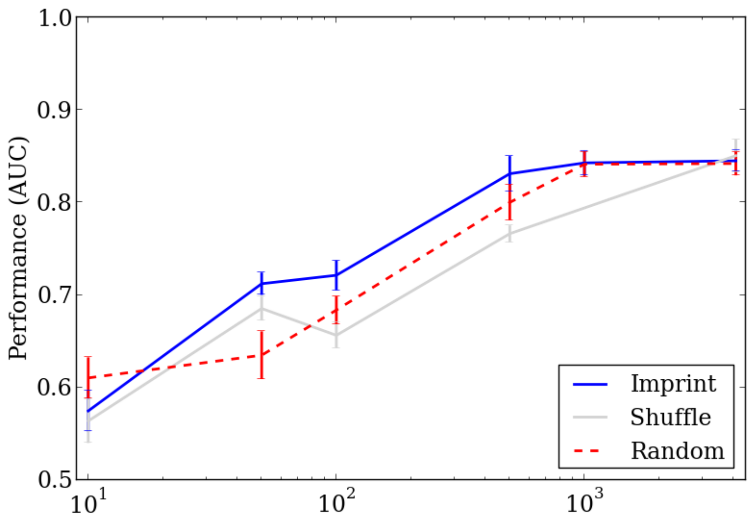


(a) *Cars v. Planes*

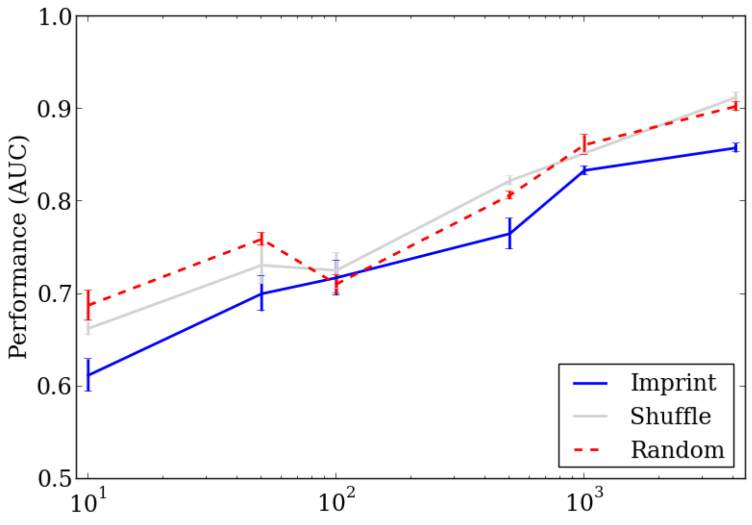


(b) *Face1 v. Face2*

Figure 4.6: Comparison of Glimpse's performance on two tasks, using 4075 imprinted prototypes (blue); and 4075 random prototypes (dashed red). The horizontal axis shows the variation level (over rotation, position, and scale) of the object of interest, and the vertical axis shows the mean AUC over five independent training and testing splits at each variation level. Error bars show the standard error. Results for raw pixel (gray) and C1 (dashed gray) features are shown for reference.



(a) *Cars v. Planes*



(b) *Face1 v. Face2*

Figure 4.7: Comparison of Glimpse's performance for different numbers of imprinted (blue), shuffled (gray), and random (dashed red) prototypes. Performance is reported as mean AUC over five trials, with error bars showing one standard error. Variation level three is used for each task.

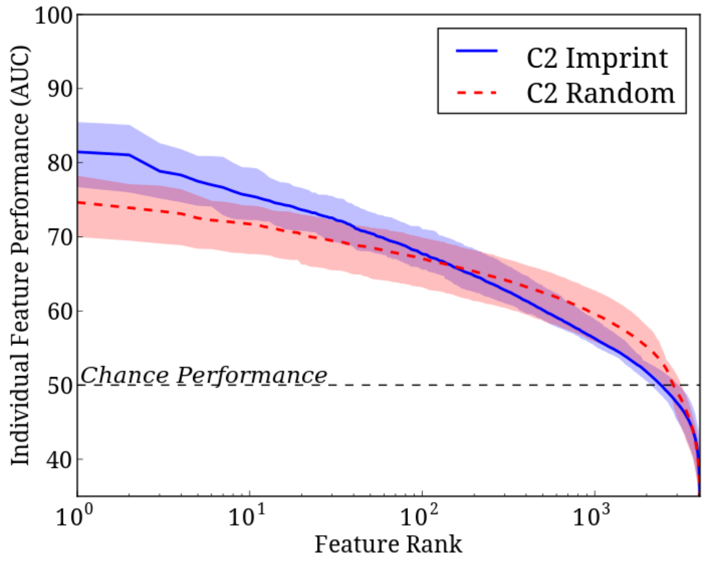
combined to create a single, highly discriminative group. In contrast, I expect imprinting to generate at least some prototypes that provide highly discriminative representations, even when considered in isolation.

To investigate this, I measured performance based on individual features. For each prototype generation method (imprinting or random), I generated 4075 prototypes as before, except here I used them one at a time to create a single value to represent each image in order to train and test the SVM. As before, I performed five independent training and testing splits using each prototype. Figure 4.8a shows the performance for single imprinted prototypes and single random prototypes on the *Cars v. Planes* task, where the prototypes are ranked by performance. Figure 4.8b shows the same values for the *Face1 v. Face2* task. I found very little difference between the two representations in terms of the occurrence of individually-discriminative features. In fact, it is striking how well the best random features perform when operating in isolation. In short, it appears that random prototypes are not limited to operating in ensembles.

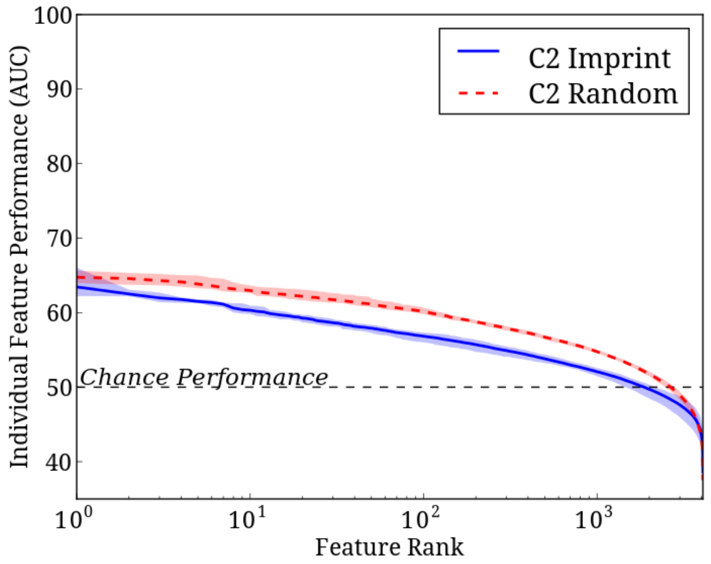
Lastly, I investigated the hypothesis that the imprinted and random prototype representations behave similarly because they code for similar visual features. It is possible, in theory, that the process of random prototype generation occasionally creates the kind of useful shape selectivity that would be expected under imprinting. In this case, I would expect these "lucky" random features to be among the most discriminative when used in isolation.

Due to the nature of these networks, it is difficult[4] to interpret the contents of a prototype directly. Instead, I attempt to characterize a given prototype by examining those input patches that provide the best match. Figures 4.9 and 4.10

[4]As one example, the invariance properties of the C2 layer may cause the same feature values to be produced for multiple images.



(a) *Cars v. Planes*



(b) *Face1 v. Face2*

Figure 4.8: Performance (mean AUC and range) using individual features from either imprinted (solid blue) or random (dashed red) prototypes for (a) the *Cars v. Planes* task, and (b) the *Face1 v. Face2* task. In both cases, the tasks use variation level three. The line shows the mean performance over five independent trials, while the shaded area shows the range of performance values.

shows this data for the most discriminative prototypes on the *Cars v. Planes* and *Face1 v. Face2* tasks, respectively. Each row in the figures corresponds to one of the five most discriminative prototypes, that is, those ranked 1–5 in Figure 4.8. The columns of each row give the 10 image patches from the dataset to which the corresponding prototype matched most closely, where each image is allowed at most one match. Although it may appear that patches in, say, the top row of Figure 4.10a are from slightly different positions of the same image, these patches are in fact from different images.

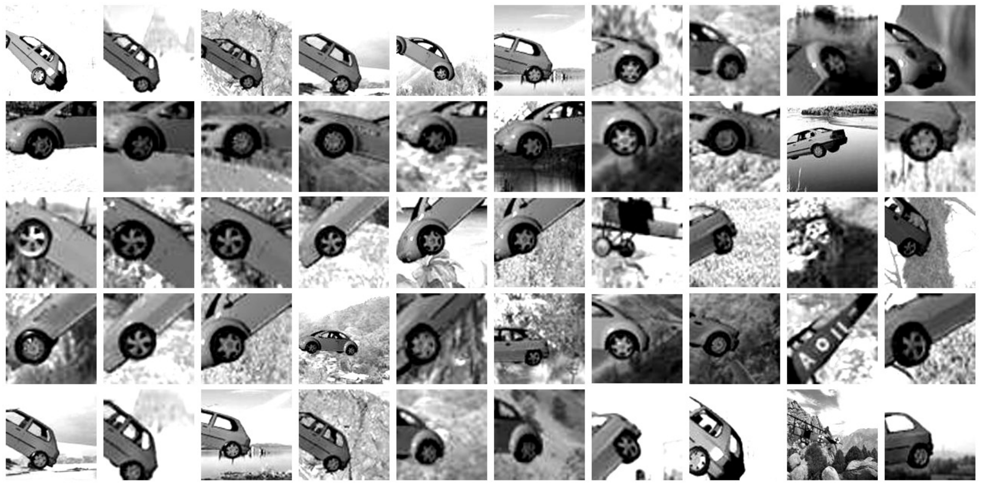
As expected, it appears that the five imprinted prototypes are responding preferentially to specific "shape-based" patterns relevant to faces, and are relatively robust to rotation and translation of those patterns. However, the five random prototypes display no obvious "shape" preference along each row, nor do their responses appear to be relevant to faces.

These results show that, while imprinted features are highly selective to shape and somewhat invariant to background clutter, random prototypes are not easily interpretable as shape templates. Although results were shown for one particular set of imprinted and random prototypes, this behavior was found to be qualitatively similar for other, independently generated, sets of prototypes.

4.4 Discussion

In this work, I investigated the hypothesis that shape-based prototypes are central to the ability of alternating multilayer networks to perform invariant object-recognition. To summarize the results:

- I apply Glimpse to challenging benchmarks for invariant object recognition, and find that learned "shape" prototypes are not necessary to achieve the


(a) Imprinted prototypes

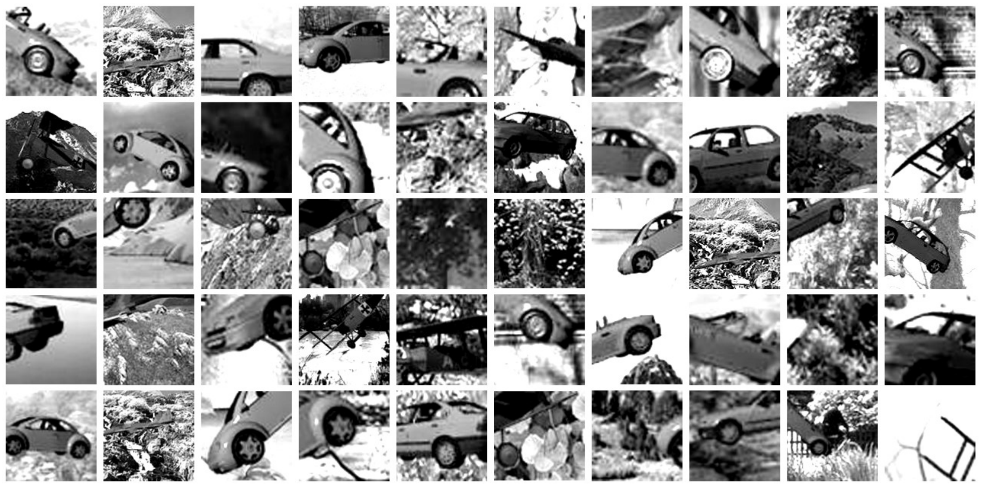
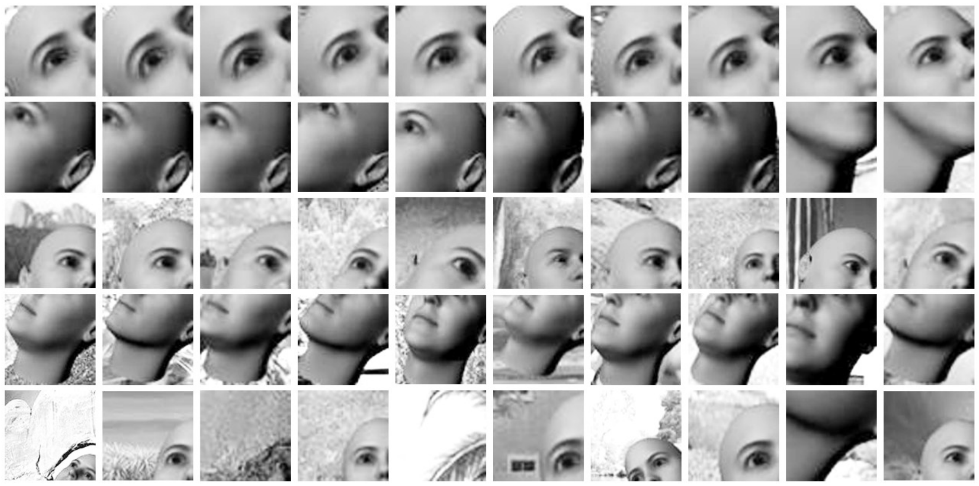

(b) Random prototypes

Figure 4.9: Characterization of best-performing prototypes for the *Cars v. Planes* task (cf. Figure 4.8a) based on the input patches to which they respond most strongly. (a): Each row corresponds to one of the top five imprinted prototypes (those ranked 1–5 in the imprinted set in Figure 4.8a). The 10 images in each row are the 10 image patches in the *Cars v. Planes* dataset to which the prototype matched most closely. All patches in a row are drawn from different images. (b): Same as part (a), but here the five top prototypes are those ranked 1–5 in the random-prototype set in Figure 4.8a. In contrast to part (a), there is a distinct lack of shape specificity along each row.


(a) Imprinted prototypes

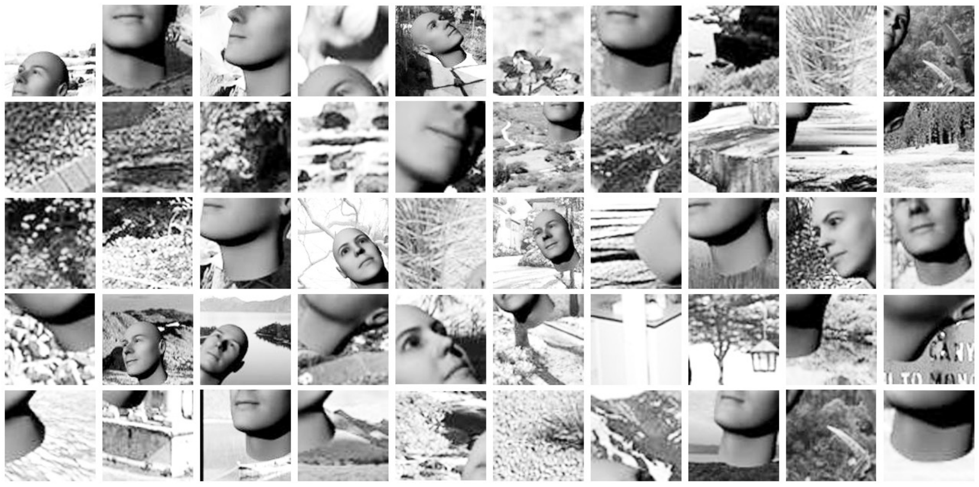

(b) Random prototypes

Figure 4.10: Characterization of best-performing prototypes for the *Face1 v. Face2* task (cf. Figure 4.8b) based on the input patches to which they respond most strongly. Results are shown as in Figure 4.9, with the best matches shown for (a) imprinted and (b) random prototypes. As before, random prototypes lack the shape specificity that is characteristic of imprinted prototypes.

performance seen in the literature. These benchmarks specifically emphasize viewpoint-invariance by including realistic variation in the presentation of objects. As such, the “shape-free” features based on random prototypes seem to provide an unlearned, unbiased (i.e., universal) dictionary.

- Upon analysis, I find evidence that (1) randomly-generated prototypes mediate performance that is on par with a learned shape dictionary (Figures 4.5 and 4.6), even for small networks (Figure 4.7) or single prototypes (Figure 4.8). Critically, I also find evidence that (2) those prototypes lack shape specificity (Figures 4.9 and 4.10), a characteristic that was thought to be central to the success of these networks.

Taken together, these results argue that our understanding of successful hierarchical visual models is far from complete, and that further analysis is warranted. Furthermore, my work suggests that—when used properly—random prototypes may have an important role to play in these hierarchical networks.

I am left with several questions that have yet to be answered. Chief among them are: (1) In what types of object-recognition tasks would a set of learned shape-based prototypes provide an advantage over randomly generated prototypes? Equivalently, for what sorts of tasks can we simply rely on random prototypes and thus avoid the cost of learning? (2) What are the mechanisms underlying the success of random prototypes in my experiments? For example, can this success be explained by mechanisms related to the methods of random projections or compressive sensing [97, 98]? The consideration of these questions is left for future work.

Chapter 5

Feature Selection

This chapter considers the problem of learning prototypes from data. In this sense, the method discussed here can be thought of as an extension of imprinting. Although the literature on hierarchical visual models contains many approaches to prototype learning, this chapter focuses on one particular approach called feature selection. The approach of feature selection has been reported to significantly increase performance in some models [9], even on complex tasks with many object categories.

In this chapter, I explore the benefits of feature selection for increasing the performance of the Glimpse model. I find that this method leads to a dramatic improvement in performance, but is limited by its prohibitive computational cost.

5.1 Background

Feature selection begins in a manner similar to imprinting. The process starts by selecting patches at random from training images. Glimpse is applied to

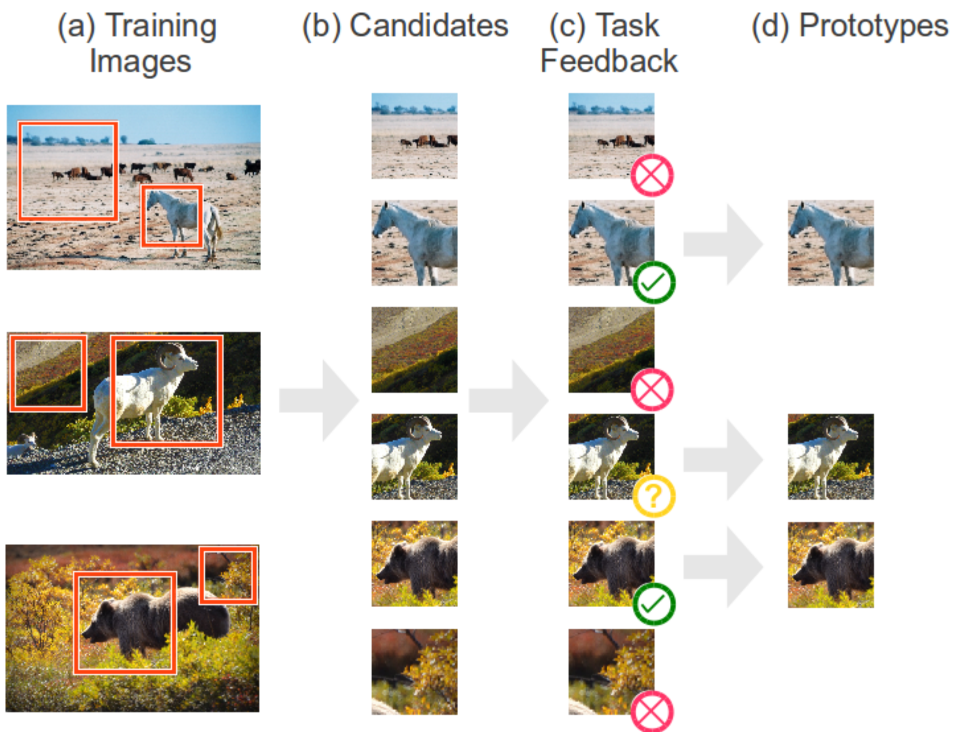


Figure 5.1: Illustration in which prototypes are learned by feature selection. (a) Image patches (shown as red boxes) are chosen at random, and (b) candidate prototypes are recorded from the model's C1 activity that is calculated from these patches. (c) Task feedback is used to weight each candidate prototype, illustrated here with high, medium, and low weight indicated by a green check mark, yellow question mark, and red "X", respectively. (d) Candidates are selected by weight to construct the final set of prototypes.

each patch, and the model's C1 activity for each patch is recorded as a *candidate* prototype. Each candidate is assigned a weight that reflects its estimated "quality". A candidate prototype's weight is estimated from task feedback in an application-specific way, as will be described below. Finally, candidates are selected by weight to form the final set of prototypes. This process is illustrated in Figure 5.1.

A more detailed description of feature selection is shown in Algorithm 5.1. The algorithm begins in Line 1 by choosing a large set of candidate prototypes from training images, as done in imprinting. Glimpse is then used with these

Algorithm 5.1 Prototype learning by feature selection.

Input: Number n of candidate prototypes, number k of selected prototypes, and set of training and testing images.

```
1. Select $n$ candidate prototypes from training images, as in imprinting.
2. Use Glimpse with candidate prototypes to compute $n$ features for each
   training image.
3. Select $k$ features:
   (a) Compute quality $J(f_i)$ of each feature $f_i$.
   (b) Rank features by quality.
   (c) Select the $k$ highest-ranked features.
4. Create set $\mathbf{P}$ of only those candidate prototypes whose feature was selected
   in Line 3c.
5. Evaluate Glimpse on testing images using prototypes in $\mathbf{P}$.
```

prototypes to extract features for each image in the training set. From the extracted features, a subset is selected in Lines 3a-3c. In Line 3a, the quality of each feature f_i is computed in an application-specific way. The quality measure is denoted as $J(\cdot)$. Features are then ranked according to $J(f_i)$ in Line 3b. The first k features in the ranked list are "selected" in Line 3c, where k is specified *a priori*. The prototype associated with each selected feature is identified in Line 4. Glimpse is used with the selected prototypes in Line 5, and the performance is evaluated on a set of testing images.

In the most common approach, the quality $J(\cdot)$ of each feature is computed using feedback from the classifier. This feedback indicates whether a given feature was discriminative. For a linear SVM, this value can be computed from the feature weights α_i that are assigned when training the classifier. As discussed in

Section 3.1.6, the decision function of a linear SVM can be written as

$$f(\mathbf{x}) = \operatorname{sgn}\left[b + \sum_i \left(\sum_k \gamma_k v_{ki}\right) x_i\right] , \tag{5.1}$$

where γ_k is the weight on the k^{th} support vector, v_{ki} is the i^{th} feature of the k^{th} support vector, and x_i is the i^{th} feature value. The weight of the i^{th} feature is given as $\alpha_i = \sum_k \gamma_k v_{ki}$, and feature quality is measured as $J(f_i) = \alpha_i^2$. This approach has been used successfully by a number of researchers [9, 16, 99] to increase model performance while minimizing the number of prototypes.

One drawback of feature selection is its high computational cost, which results from the fact that feature quality requires feature values to be computed for every candidate prototype on every training image. In practice, this significantly limits the number of candidate prototypes that can be evaluated.

As a result of this drawback, some researchers have suggested the use of other forms of task information [19, 25, 39]. In one example, candidate prototypes are selected only if they were created from the part of the image that contained the foreground object [39].

Another approach chooses the weight of a candidate based on the classifier's *estimated* feedback [15]. In this method, a small set of reference prototypes is recorded from training images, and classifier feedback is used as defined above to weight each prototype. When a weight is needed for a candidate prototype, it is not measured directly from classifier feedback. Instead, the weight is copied from the most similar reference prototype. If the reference prototype was useful for classification, then the candidate is weighted highly. Otherwise, the candidate is given a low weight.

The estimated feedback approach compares prototypes with respect to a number of properties of the activation values within the prototype's template, such as the mean and standard deviation of the activation values. These properties are intrinsic, in that they have nothing to do with the current task. The approach was demonstrated to select useful features for an HMAX model, while significantly reducing its computational cost [15].

5.2 Methods

As discussed in the previous section, feature quality has often been computed from weights assigned by a linear SVM. In my experiments, I use a similar approach, but instead use weights assigned by sparse logistic regression [86]. As discussed in Section 3.1.6, the decision function for logistic regression can be written as

$$f(\mathbf{x}) = \operatorname{sgn}\left[\frac{1}{1+e^{-(b+\sum \alpha_i x_i)}} - \frac{1}{2}\right] , \tag{5.2}$$

where α_i is the weight on the i^{th} feature. In my experiments, I compute feature quality as $J(f_i) = \alpha_i^2$. The parameters in Equation 5.2 have been chosen using a sparse optimization procedure, so that the fewest number of features are used.

Performance is measured as the classification accuracy[1] for five independent trials, and the mean and standard error of the trials is reported. Due to its prohibitive computational cost, however, only a single trial is reported for feature selection.

In each trial, half the images are chosen at random for training, and the other half are saved for testing. The training images are used to choose a new set of prototypes, and a sparse logistic regression classifier is trained on features

[1]Accuracy is used in this and later chapters that employ multiclass datasets, since AUC is a measurement of performance on binary tasks.

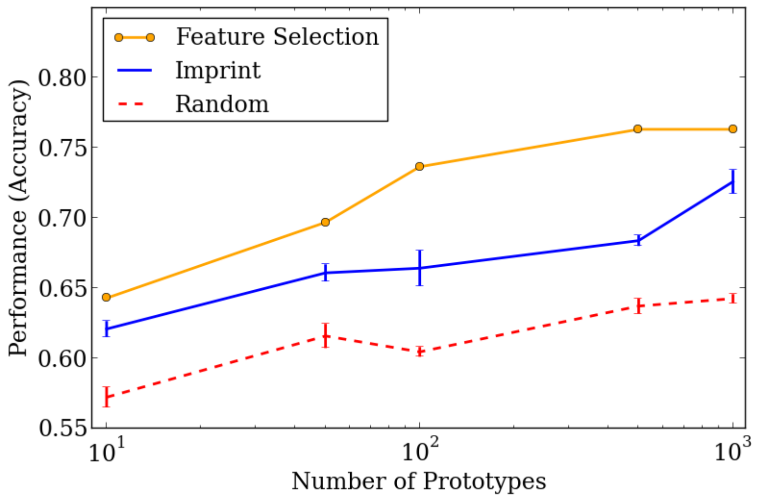


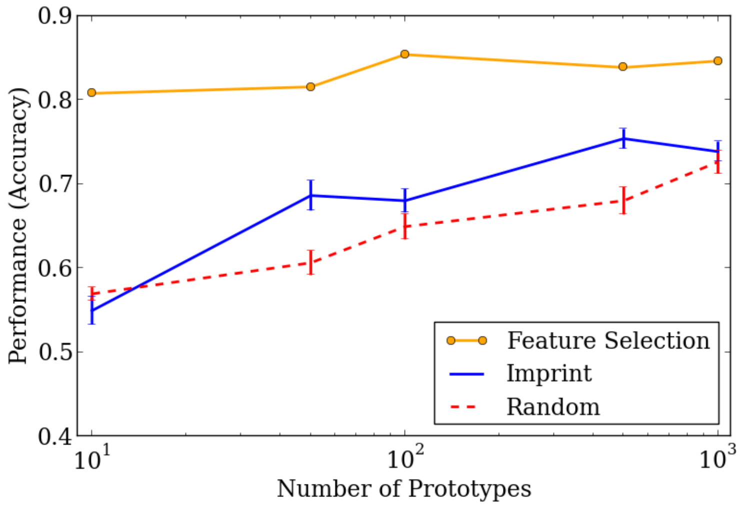
Figure 5.2: Performance on the *Animals* task for prototypes learned by feature selection (solid orange line), compared with performance for imprinted (solid blue line) and random prototypes (dashed red line).

composed of C2 activity. The datasets used in these experiments are those for which an invariant representation was found to be useful in Section 4.2, including the *Animals* task of Serre et al. [2], and the *Cars v. Planes* and *Face1 v. Face2* tasks of Pinto et al. [18].

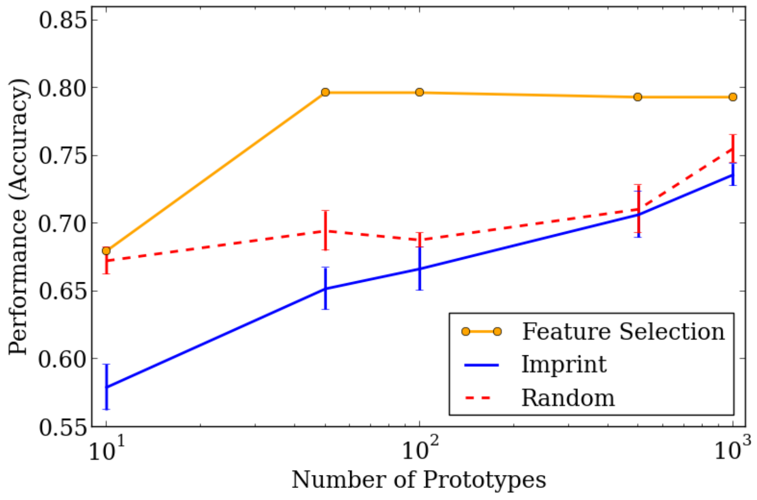
5.3 Results

I evaluated the performance for prototypes learned by feature selection, and compared this with the performance of imprinted and random prototypes. For each method, performance was computed for a range of network sizes (i.e., for different numbers of prototypes). Results are given for the *Animals* task in Figure 5.2, the *Cars v. Planes* task in Figure 5.3a, the *Face1 v. Face2* task in Figure 5.3b, and the *Caltech 256* task in Figure 5.4.

I found that feature selection consistently out-performs other methods. For the baseline methods of imprinting and random construction, performance im-



(a) *Cars v. Planes*



(b) *Face1 v. Face2*

Figure 5.3: Performance on synthetic tasks for prototypes learned by feature selection (solid orange line), compared with performance for imprinted (solid blue line) and random prototypes (dashed red line).

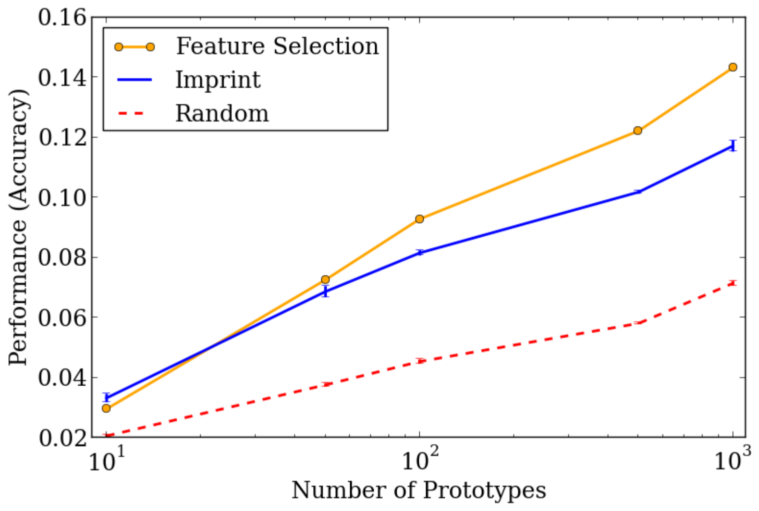


Figure 5.4: Performance on the *Caltech 256* task for prototypes learned by feature selection (solid orange line), compared with performance for imprinted (solid blue line) and random prototypes (dashed red line).

proves consistently with model size—that is, with an increase in the number of prototypes. In contrast, performance for feature selection generally saturates at a particular model size, with the best performance occurring even for small networks. This shows that task feedback can be used to find highly discriminative, task-specific prototypes. Interestingly, performance for feature selection is not decreased by including more prototypes. This suggests that the model is robust to prototypes that lead to irrelevant and redundant features, at least when sparse logistic regression is used for classification.

The benefit of feature selection depends heavily on the size of the model used. For example, the performance for prototypes learned by feature selection is no better than that of random prototypes on the *Cars v. Planes* task, if the model is allowed to use only 10 prototypes. However, the benefit of feature selection quickly becomes apparent when the model size increases.

In a sense, the "saturation point" for feature selection—that is, the smallest model with high performance—provides a measure of a dataset's complexity. In this sense, the *Face1 v. Face2* dataset is more complex than the *Cars v. Planes* dataset. This agrees with our intuition, since a single "wheel" prototype was enough to achieve high performance on the *Cars v. Planes* task (see Figures 4.8a and 4.9a). This measure also suggests that the *Animals* dataset is more complex than the *Face1 v. Face2* dataset, which could be explained by the broad variation of appearance in the "animal" category. Similarly, this measure suggests that the *Caltech 256* dataset is more complex than the *Animals* dataset, which could be explained easily by the significant difference in the number of object categories between the two datasets.

5.4 Conclusions

In this chapter, I explored the benefits of feature selection for increasing the performance of the Glimpse model. I found that feature selection led to a dramatic improvement in performance, although the method is limited by its prohibitive computational cost.

Prototypes learned by feature selection consistently out-performed those learned by baseline methods such as imprinting. The benefit of feature selection over other methods was often significant, but this result depended on the number of prototypes used in the model. Overall, the difference between methods only becomes apparent at a certain scale, and this scale appears to be task dependent. Thus, my results suggest that *it is critical to consider the scaling behavior of a model*, rather than simply measuring performance at a fixed number of prototypes. Interestingly, performance for prototypes learned by feature selection was very high on some datasets, even when very few prototypes were used. This result suggests that task feedback has been used successfully to find a small set of task-specific and discriminative prototypes.

Chapter 6

Learning by Clustering

As discussed in Section 2.3, imprinting has a number of drawbacks that result from the inclusion of redundant and non-discriminative prototypes. This increases the model's computational cost, and can decrease its performance as well. A number of approaches have been suggested in the literature to overcome these drawbacks.

This chapter investigates k-means clustering, which is a fast method for summarizing a large set of prototypes by creating a small number of representative examples. K-means has been used repeatedly in the literature to learn prototypes in hierarchical models. For example, this approach is suggested [100] as the best approach for "unsupervised" learning in convolutional networks, which are hierarchical models that bear a strong resemblance to HMAX. While k-means has been used to learn prototypes in HMAX [19, 39, 44], its benefit—compared with imprinting, for example—is still unclear.

Surprisingly, I find that k-means provides no performance benefit when compared to imprinting, but instead often hurts performance. I explore two hypotheses to explain the lack of improvement, but find that both hypotheses are contradicted by the evidence. Thus, finding an explanation for the behavior of k-means prototypes is left as an open problem.

The rest of this chapter is organized as follows. Section 6.1 provides background on k-means clustering, and discusses how this method can be used to learn prototypes. Section 6.2 outlines the experimental methodology used in the experiments of this chapter, and my results are presented in Section 6.3. Finally, Section 6.4 summarizes the findings and provides conclusions.

6.1 Background

The approach of clustering takes the choice of prototypes and casts it as a machine learning problem. Just as in imprinting, the process starts by randomly selecting image patches, applying a hierarchical model, and recording the model's C1 activity for each region. These prototypes are then treated as vectors that are partitioned into a number of clusters. Each cluster is summarized by a new prototype that is representative of the cluster members, and the model is evaluated using only these representatives as prototypes. This process is illustrated in Figure 6.1.

Clusters are chosen so that prototypes from the same cluster are similar. The overall quality of a cluster is measured as the variance of the cluster elements. Once clusters are chosen, each cluster is summarized by computing the average of all prototypes assigned to it. This is possible because the prototypes are treated as vectors, so the i^{th} component of the average is given by the average of the i^{th} component across all members of the cluster.

The above description can be written analytically as an *objective function* that measures how well (or poorly) a given partitioning meets the criteria. This

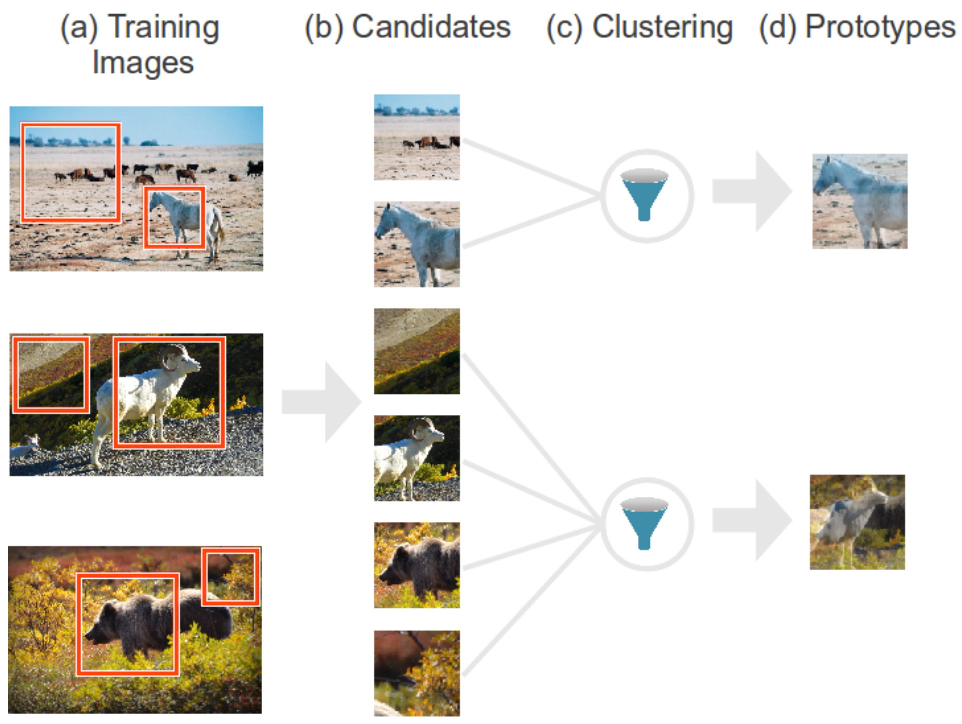


Figure 6.1: Illustration in which prototypes are learned by clustering. (a) Image patches (shown as red boxes) are chosen at random, and (b) prototypes are recorded from the model's C1 activity. (c) The prototypes are partitioned into clusters. (d) Each cluster is summarized by computing the average of its prototypes. This creates a new set of prototypes—given as vectors of C1 activations—and the model is evaluated using only these new prototypes.

is given as

$$\mathrm{Obj}(\mathbf{P}; \mathbf{C}) = \sum_{j=1}^{k} \sum_{i=1}^{n} c_{ij} \left\| \mathbf{x}_i - \mathbf{p}_j \right\|^2 , \tag{6.1}$$

where $\mathbf{P} = \{\mathbf{p}_j\}$ is the matrix of new prototypes, k is the number of prototypes in that matrix, n is the number of original prototypes, and $\mathbf{x}_i$ is the i^{th} such prototype. The matrix $\mathbf{C}$ encodes the cluster assignments, where the element $c_{ij} \in \{0, 1\}$ indicates whether the i^{th} example is assigned to the j^{th} cluster.

The goal of prototype learning by clustering is to find a new set of prototypes $\mathbf{P}$ that minimize the objective function. The inner summation of Equation 6.1 ranges over the elements of a given cluster, calculating their dissimilarity with the cluster's representative prototype. The higher this dissimilarity, the more the choice of prototypes is penalized. The outer summation ranges over the clusters, combining these penalties. Thus, the best prototypes are those that minimize the variance over all clusters simultaneously.

One approach to minimize $\mathrm{Obj}(\mathbf{P}; \mathbf{C})$, called *k-means* [101], is shown in Algorithm 6.1. The algorithm begins in Line 1 by choosing a large set of example prototypes from training images, as done in imprinting. Clustering is then performed in Lines 2a-2d. This begins by guessing the cluster centers $\mathbf{P}$ in Line 2a, and then repeatedly applies the following two steps. Cluster assignments are reviewed in Line 2b to ensure that each candidate is assigned to the cluster with the closest center. The center of each cluster is shifted in Line 2c to minimize its distance to all members, where the new center is chosen to be the average of all current members. These two steps are repeated until the objective function reaches a fixed point in Line 2d. Finally, Glimpse is evaluated on separate set of testing images in Line 3, where prototypes are given from the cluster centers $\mathbf{P}$.

Note that a reassignment in Line 2b will always decrease the combined

Algorithm 6.1 Prototype learning by k-means clustering.

```
Input: Number $n$ of initial prototypes, number $k$ of clusters, and set of training and testing images.

1. Select $n$ prototypes from training images, as in imprinting.
2. Partition these prototypes into clusters:
   (a) Choose initial cluster centers $\mathbf{P}$ arbitrarily.
   (b) Reassign each candidate to the cluster with the nearest center, given by
       $c_{ij} = \begin{cases} 1 & \text{if } j = \operatorname{argmin}_{j^*} \|\mathbf{x}_i - \mathbf{p}_{j^*}\| \\ 0 & \text{otherwise.} \end{cases}$
   (c) Update each cluster center according to the newly assigned points, given by
       $\mathbf{p}_j = \frac{1}{\sum_i c_{ij}} \sum_i c_{ij} \mathbf{x}_i \,.$
   (d) Go to Step 2b unless the objective Obj() in Equation 6.1 has converged.
3. Evaluate Glimpse on testing images using cluster centers $\mathbf{P}$ as prototypes.
```

penalty for the two clusters in the reassignment, that is, the cluster that loses a member and the cluster that gains it. Similarly, Line 2c will never increase the penalty for either cluster, since the cluster centers are chosen to minimize this penalty. Since the value of the objective function is never increased, k-means is guaranteed to converge to a (locally) optimal set of clusters.

6.2 Methods

In these experiments, Glimpse uses the sparse logistic regression classifier [86] that was introduced in Section 3.1.6. Clustering is performed using the Mini-Batch k-means algorithm [102], implemented in the Scikit-Learn Python package

[103]. In this chapter, performance is measured as the classification accuracy[1] for five independent trials, and the mean and standard error of the trials is reported. In each trial, half the images are chosen at random for training, and the other half are saved for testing. The training images are used to choose a new set of prototypes, and the classifier is trained on features composed of C2 activity. The datasets used in these experiments are those for which an invariant representation was found to be useful in Section 4.2.

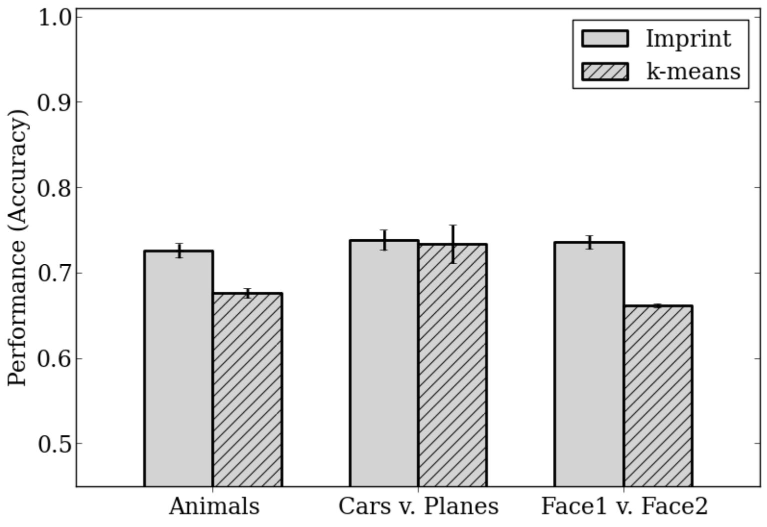
6.3 Results

For each dataset, the performance of Glimpse is measured using 1,000 prototypes learned by k-means clustering, which uses a large set of 10,000 candidate prototypes. Every set of k-means prototypes is learned from an independent set of candidates. For comparison, the performance is computed on the same dataset using 1,000 imprinted prototypes.

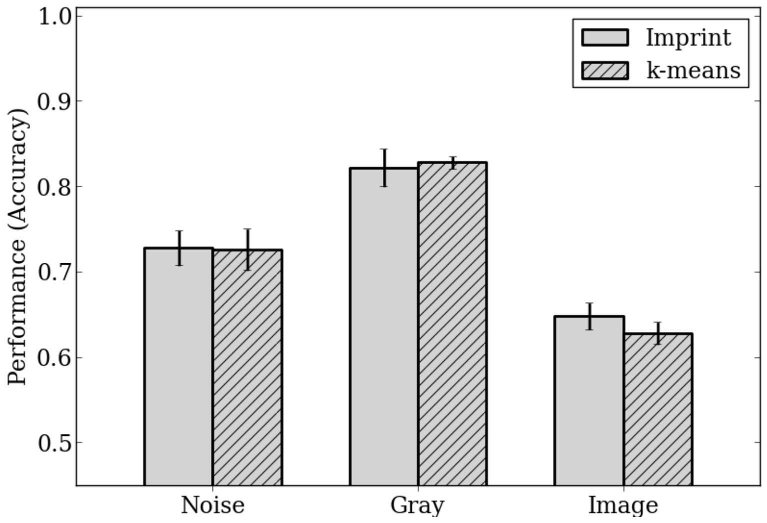
Results are shown in Figures 6.2 and 6.3. Surprisingly, imprinting led to a superior representation in these experiments, with performance that was at least as high as for k-means prototypes across all tasks.

As in Chapter 4, the performance for Glimpse is also analyzed for different number of prototypes. Results are shown in Figures 6.4 and 6.5, which shows the performance (mean accuracy) for different numbers of prototypes chosen by imprinting and k-means. These results suggest that the difference between representations is less significant in low-dimensional representations (i.e., for a small number of prototypes), but that imprinted representations have a superior scaling behavior.

[1]Accuracy is used in this and later chapters that employ multiclass datasets, since AUC is a measurement of performance on binary tasks.



(a) *Animals* and synthetic tasks of Pinto et al. Variation level three is used for synthetic tasks.



(b) *Cats v. Dogs* with rendered objects on various backgrounds.

Figure 6.2: Comparison of performance for C2 features using imprinted (gray) and k-means (dashed gray) prototypes. The vertical axis shows the mean performance (accuracy) over five independent trials for 1,000 prototypes, with error bars showing one standard error. Surprisingly, imprinting led to superior representations across nearly all tasks.

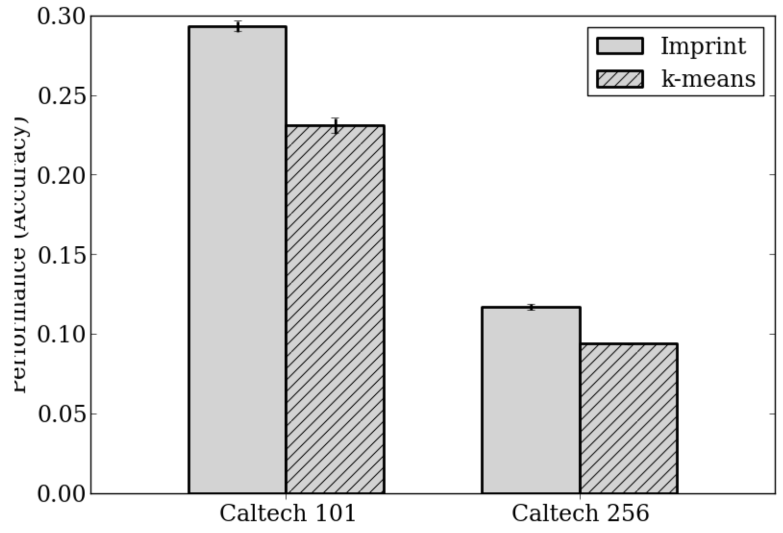
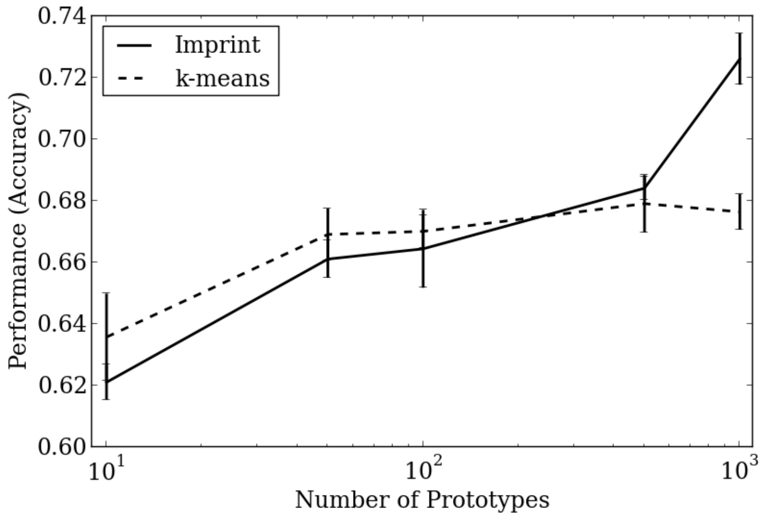


Figure 6.3: Comparison of performance on multiclass datasets for 1,000 C2 features using imprinted (gray) and k-means (dashed gray) prototypes, as in Figure 6.2.

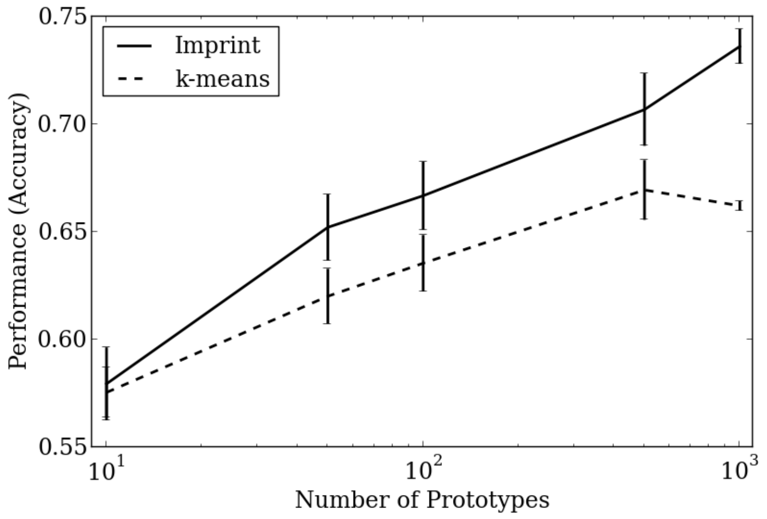
6.3.1 Effect of Training Size

Note that the experiment described in the previous section used the same number of candidates when learning k-means prototypes, regardless of the number k of prototypes to be learned. Consequently, the average number of candidates per cluster varies inversely with k. For example, the ratio of candidates to prototypes in Figure 6.4a is approximately 1,000:1 on the left side, but only 10:1 on the right. It is possible that this difference adversely affects the performance of learned prototypes. To test this, I measure the performance of Glimpse using k-means prototypes learned with different numbers of candidates.

Results are shown for two representative datasets in Figure 6.6, which shows the change in performance when k prototypes are learned with an average of 10, 100, and 1,000 candidates per cluster. This data is reported for the Animals and Cars v. Planes tasks. Interestingly, I see little evidence that more training samples result in better prototypes. In fact, the results for the Cars v. Planes



(a) *Animals*



(b) *Face1 v. Face2*

Figure 6.4: Comparison of Glimpse's performance for different numbers of imprinted (solid) and k-means (dashed) prototypes, for a representative sample of the datasets used in Figure 6.2. Plots show mean performance (accuracy) over five independent trials, with error bars showing one standard error.

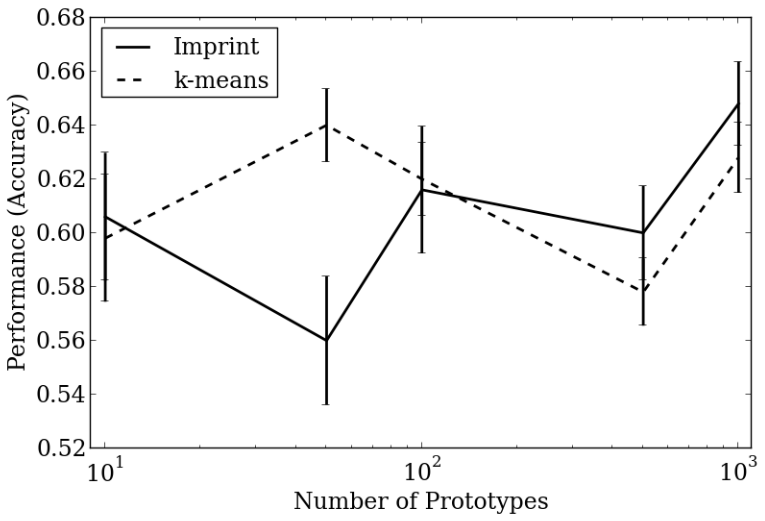


Figure 6.5: Comparison of Glimpse's performance on rendered Cats v. Dogs over image backgrounds for different numbers of imprinted (solid) and k-means (dashed) prototypes, as in Figure 6.4.

task shows a slight negative trend.

6.3.2 Effect of Activation Function

My results for prototypes learned using k-means are surprising, but not unprecedented. Although researchers have reported good performance for k-means in related visual models (e.g., see [100]), it is unclear whether these results also apply to HMAX-like models. Studies of prototype learning for HMAX-like networks have reported mixed results, and have roughly been in line with our findings. For example, Brumby et al. [29] report that prototypes found using a Hebbian learning rule (similar to an online version of k-means) often fail to outperform a simple imprinting procedure.

Here, I consider whether the poor results for k-means, relative to its use in other architectures, is related to Glimpse's S2 activation function. One of the unique characteristics of HMAX-like networks is their use of an RBF activation

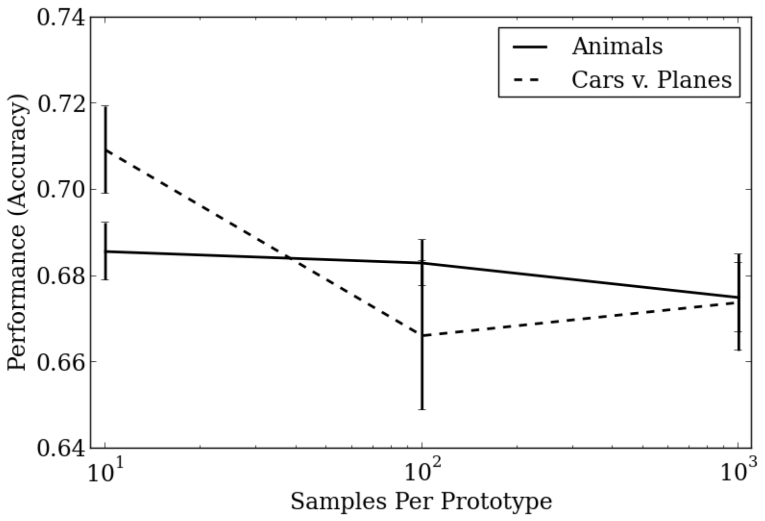


Figure 6.6: Effect of sample size on Glimpse's performance with k-means prototypes on the *Animals* (solid) and *Cars v. Planes* (dashed) tasks. Performance is shown for 128 prototypes. The horizontal axis shows the number of candidates (on a log scale) per prototype, the vertical axis shows the mean performance (accuracy) over five independent trials for sparse logistic regression. Error bars show one standard error.

function. However, one of the models in which k-means has been successful uses a dot-product activation function, similar to that used for Glimpse's S1 layer. This section investigates whether this difference can account for Glimpse's poor performance when using k-means prototypes.

To test this hypothesis, Glimpse's S2 activation function was modified to use a dot product in the following way. Remember that the RBF in Equation 3.5 is based on the distance between the input $\mathbf{x}$ and prototype $\mathbf{p}$. However, the distance is defined in terms of the inner product. Using this definition along with the bilinearity of the inner product, the distance can be rewritten as

$$\|\mathbf{x} - \mathbf{p}\| = \|\mathbf{x}\|^2 + \|\mathbf{p}\|^2 - 2\,(\mathbf{x}, \mathbf{p})\ , \tag{6.2}$$

where the term $(\cdot, \cdot)$ denotes the dot product. If contrast normalization is used—

that is, the input and prototype are constrained to have unit norm—then Equation 6.2 implies that

$$\|\mathbf{x} - \mathbf{p}\| \propto (\mathbf{x}, \mathbf{p}) \ .$$

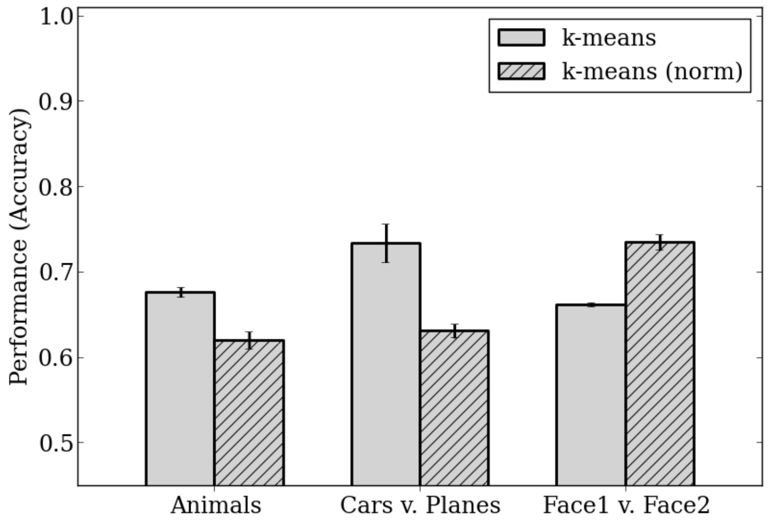
That is, the use of contrast normalization makes the distance function proportional to the dot product between the two vectors[2]. A method for contrast normalization was already introduced in Section 4.3, and was shown to be useful for random prototypes. That method is used here as well.

Results are shown in Figures 6.7 and 6.8 for 1,000 prototypes learned with k-means, where the S2 units use either an RBF (as in Figure 3.5) or a sparse, contrast invariant activation function. In nearly all tasks, the updated activation function resulted in a significant decrease in performance. The only exception to this decrease was the synthetic *Face1 v. Face2* task, in which a sparse contrast-invariant activation function led to an *increase* in model performance. These results suggest that the activation function is generally not responsible for Glimpse's poor performance when using k-means prototypes.

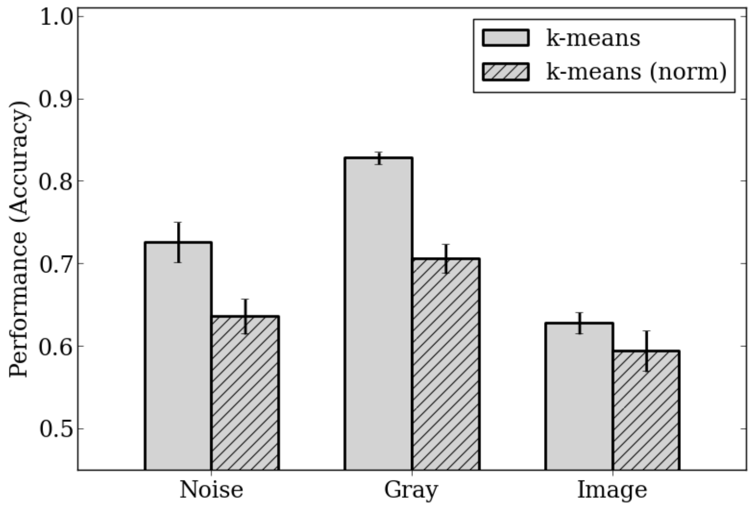
6.4 Conclusions

This chapter investigated the use of k-means clustering to learn prototypes in Glimpse. I hypothesized that clustering would significantly increase the model's performance, compared to the use of imprinted prototypes. However, my results suggest that k-means is poorly adapted to the problem of prototype learning. The performance for prototypes learned by k-means was consistently worse than that using an imprinted dictionary (Figure 6.2), and this relationship did not

[2]The connection between an RBF and normalized dot product is also discussed by Serre et al. [104].



(a) Animals and synthetic tasks of Pinto et al. Variation level three is used for synthetic tasks.



(b) Rendered *Cats v. Dogs* on various backgrounds.

Figure 6.7: Performance for k-means prototypes using the standard activation function (gray), compared to that using a sparse, contrast invariant activation function (hatched gray). The vertical axis shows the mean performance (accuracy) over five independent trials for 1,000 prototypes, with error bars showing one standard error. (The error is not visible for k-means prototypes on *Caltech 256*, due to low variation between trials.) These results suggest that the activation function is not the cause of the poor performance for k-means prototypes.

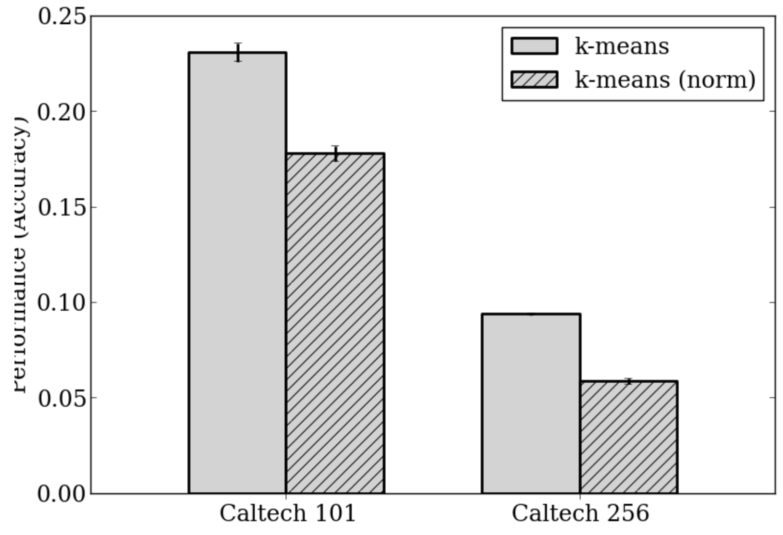


Figure 6.8: Performance on multiclass datasets for k-means prototypes using the standard activation function (gray), compared to that using a sparse, contrast invariant activation function (hatched gray).

depend on the number of prototypes used (Figure 6.2).

Surprisingly, I found that this result did not depend on the number of examples that were used during clustering (Figure 6.6). Additionally, the result does not appear to be caused by the use of an RBF activation function, as performance suffered further when the activation function used the equivalent of a dot product (Figure 6.7). Thus, the explanation for this poor performance is still unclear, and is left for future work.

Chapter 7

Clustering with Feedback

Chapter 5 showed that task information can be very useful to learn a small set of discriminative prototypes for an HMAX-like model. However, the method considered, called *feature selection*, has a high computationally cost. Alternatively, Chapter 6 showed that an alternative method, called *clustering*, has a low computational cost, but fails to find discriminative prototypes.

In this chapter I propose and test a new method for prototype learning that attempts to improve upon these existing approaches. The new method uses task information in a scalable way to learn a small set of discriminative prototypes from a large number of candidates. The method uses an approach based on clustering, but integrates a notion of feature quality to ignore candidates that are irrelevant to the task.

7.1 Background

In HMAX-like models, the approach of learning prototypes with task information has generally been limited to that of feature selection, as discussed in Chapter 5.

Perhaps the only other use of task information in the literature is the approach of category-dependent imprinting, which was used by Serre et al. (2002). Here, the model was applied to a *Faces v. Background* task, and prototypes were imprinted only from "face" images. The authors report a small increase in performance, compared with imprinting from both "face" and "background" categories.

Outside of HMAX-like models, a number of approaches have been suggested to include task information during clustering. These approaches have been introduced in the machine learning literature, and include methods such as Learning Vector Quantization [105] and Information Loss Minimization [106].

7.2 Weighted Clustering

In this section, I introduce a novel way to learn prototypes using task feedback. This method extends the clustering approach, described in Section 6.1, by integrating task information in the form of weights on the prototypes. I hypothesized that the introduction of task feedback would significantly improve the performance of prototypes learned by clustering.

As in other approaches, the method starts by generating prototypes from a set of training images, that is, by recording the C1 activity that results from applying Glimpse to randomly chosen image patches. Task information is used to assign a weight to each prototype, which indicates its estimated contribution to the model's performance. A weighted version of clustering is applied to 1) partition prototypes into groups and 2) create a new prototype from each group. Each new prototype is synthesized by taking the *weighted* average of all members of its group. This process is illustrated in Figure 7.1.

As before, clusters are chosen so that members of the same cluster are as

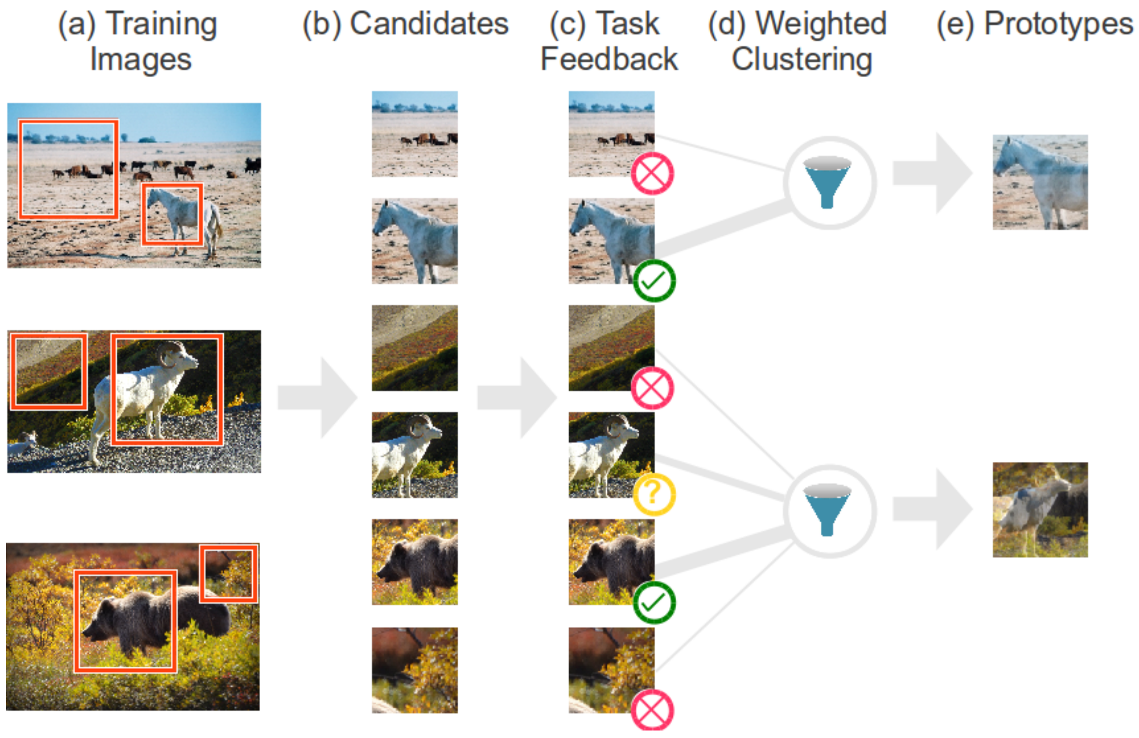


Figure 7.1: Illustration in which prototypes are learned with feedback using weighted clustering. (a) Image regions (shown as red boxes) are chosen at random, and (b) prototypes are recorded from the model's C1 activity. (c) Task feedback is used to weight each prototype, with high, medium, and low weight indicated by a green check mark, yellow question mark, and red "X", respectively. (d) The prototypes are partitioned into clusters. (e) One new prototype is created from each cluster, given by the *weighted* average of the prototypes in that cluster.

similar as possible. In this case, however, clusters are robust to outliers with low weight. Once a partition is chosen, each cluster is summarized by computing the weighted average of all cluster members. This can be captured analytically in the objective function

$$\text{Obj}_W(\mathbf{P};\mathbf{C}) = \sum_{j=1}^{k}\sum_{i=1}^{n} \alpha_i c_{ij} \left\|\mathbf{x}_i - \mathbf{p}_j\right\|^2 , \tag{7.1}$$

where $\mathbf{P}$ is a set of prototypes, $\mathbf{C}$ is a set of cluster assignments, and $\mathbf{x}_i$ is the i^{th} prototype, as in unweighted clustering. The additional term α_i gives the weight of the i^{th} example patch, which is assumed to be non-negative. It is not assumed that weights sum to unity.

As in Equation 6.1, the inner summation ranges over members of a given cluster, and calculates the degree to which those members differ from the cluster's center. The penalty for this cluster is proportional to the degree of difference. Unlike Equation 6.1, however, this penalty is scaled by the weight α_i of each prototype, such that deviations due to low-weight prototypes are penalized less than those for highly-weighted prototypes. The outer summation combines the penalties for all clusters.

To minimize Equation 7.1, I propose a simple modification to the k-means algorithm. The modified algorithm is shown in Algorithm 7.1. Note that the only difference between this and Algorithm 6.1 is the need to compute weights in Line 2 and the use of a weighted average to update cluster centers in Line 3c.

7.2.1 Illustrative Example

To demonstrate Algorithm 7.1, I investigate its behavior on an artificial clustering task with two-dimensional data. Here, training data is sampled from a

Algorithm 7.1 Prototype learning by weighted k-means clustering.

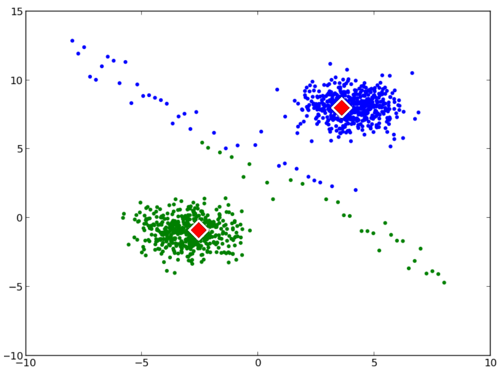
Input: Number k of clusters, and set of training and testing images.

1. Select prototypes from training images, as in imprinting.
2. Compute weight α_i for each prototype $\mathbf{x}_i$.
3. Partition prototypes into clusters:
    (a) Choose initial cluster centers $\mathbf{P}$.
    (b) Reassign each prototype to the cluster with the nearest center, given by
    $$c_{ij} = \begin{cases} 1 & \text{if } j = \operatorname*{argmin}_{j^*} \|\mathbf{x}_i - \mathbf{p}_{j^*}\| \\ 0 & \text{otherwise.} \end{cases}$$
    (c) Update each cluster center according to the newly assigned prototypes, given by
    $$\mathbf{p}_j = \frac{1}{\sum_i c_{ij}} \sum_i \alpha_i c_{ij} \mathbf{x}_i \,.$$
    (d) Go to Step 3b unless the objective $\text{Obj}_W()$ in Equation 7.1 has converged.
4. Evaluate Glimpse on testing images using cluster centers $\mathbf{P}$ as prototypes.

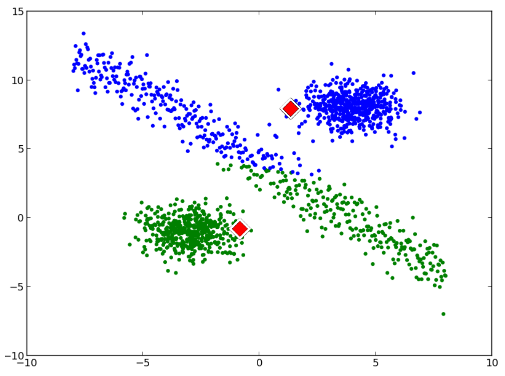
combination of three distributions, composed of two isotropic Gaussian distributions and a linear function with random noise. The goal is to estimate the parameters of the Gaussian data, while minimizing the effect from samples of the linear function. Thus, the linear function plays the role of image backgrounds, while the two Gaussian distributions provide examples of "image foregrounds".

Results are shown in Figure 7.2 for an increasing amount of background data. In the plots, we can see that the unweighted k-means algorithm becomes increasingly "distracted" by background data points, particularly as the training set becomes dominated by such data. This is significant, as unconstrained natural image corpora often contain many more examples of image backgrounds, compared with the amount of foreground data.

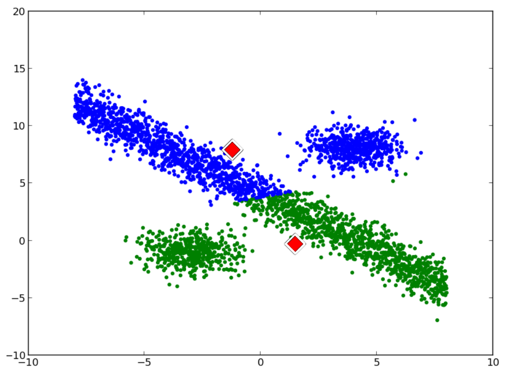
Next, I apply the weighted k-means algorithm to the most extreme case in Figure 7.2, in which a 4:1 ratio of foreground to background points is used. As the weight on background points is decreased relative to foreground points, we expect the recovered distribution parameters to shift toward their true values. Results are shown in Figure 7.3. As expected, the estimated means (shown as red diamonds) approach their true values as the weight on background patches increases. This demonstrates how non-uniform weighting can effectively compensate for background distractors. As with other forms of k-means clustering, I found that the algorithm was sensitive to the choice of initial cluster centers. However, I found that poor results occurred for weighted k-means much less frequently than for unweighted k-means.



(a) $M = \frac{1}{8}N$

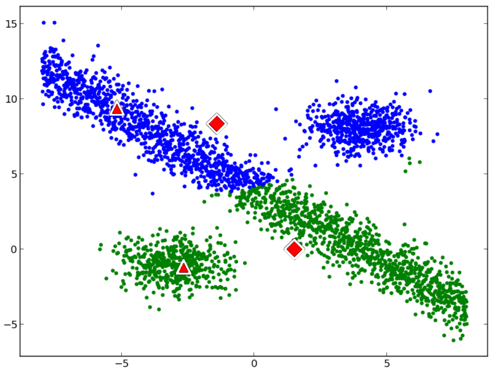


(b) $M = N$

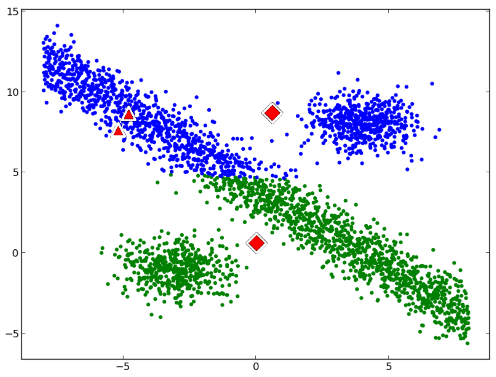


(c) $M = 4N$

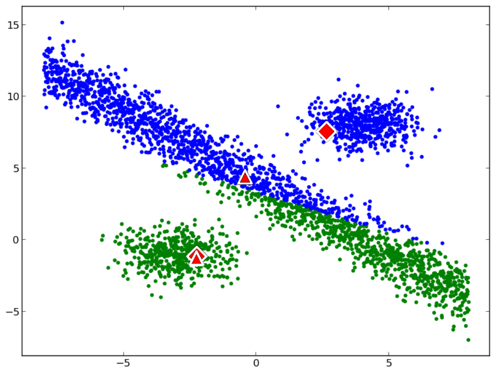
Figure 7.2: Results for unweighted k-means clustering on an artificial task, which illustrates the effect of "background" data when clustering prototypes. Data points are sampled from two "foreground" Gaussian distributions centered at $(-3, 1)$ and $(4, 8)$, and a "background" linear function $Y = 4 - X$ for normal random variable X. The goal is to recover the parameters of the two foreground distributions as the number M of background points is increased, while the total number $N = 2048$ of foreground points is held constant. Points are colored (blue and green) according to their cluster assignment. As more background points are added, the cluster centers (red diamonds) are pulled away from the foreground distributions.



(a) $\alpha_B = 1$

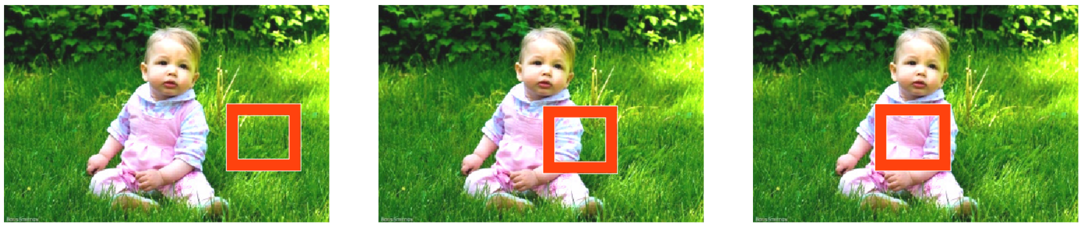


(b) $\alpha_B = \frac{1}{2}$



(c) $\alpha_B = \frac{1}{10}$

Figure 7.3: Results for weighted k-means clustering on the artificial task of Figure 7.2c for different weights α_B on the background points. The weight on foreground points is held constant at 1.0. The estimated parameters of the foreground distributions are shown (red diamonds) along with the algorithm's initial cluster centers (red triangles; see Line 1 of Algorithm 7.1). As the weight on background points is decreased, the effect of those points is reduced. The result is a significant improvement on the estimate of the foreground distributions.



(a) No overlap, $\alpha_i = 0.0$ (b) Partial overlap, $\alpha_i = .5$ (c) Total overlap, $\alpha_i = 1.0$

Figure 7.4: Example of weights computed from the degree of overlap between the prototype and the foreground object.

7.3 Methods

My experiments investigate the effectiveness of learning new prototypes when initial prototypes are drawn only from the foreground object. Specifically, this asks whether better prototypes are learned when initial prototypes drawn from image backgrounds are suppressed. This problem is approached using weighted clustering, where weights indicate whether the prototype was drawn from the foreground object. Specifically, weight is calculated as the fraction of the prototype—or rather, the image region from which it was recorded—that overlaps the foreground object. This is illustrated in Figure 7.4.

To compute these weights, we need to know which image pixels represent the foreground, and which are from the background. Such detailed segmentations are available for the *Animals* task, and were generated manually for the *Cars v. Planes* task. In my experiments, a constant value of 0.1 was added to all weights, which ensures that background prototypes (i.e., those drawn from the image background) will not be ignored entirely. This is intended to provide more examples of natural image statistics, while focusing the learner on examples of the foreground objects. However, the model's performance was qualitatively unchanged when this constant was removed.

Note that an entirely different source of task information was also investigated. In that investigation, a prototype was weighted heavily if it was discriminative—that is, if the prototype generated a feature that was useful for classification. However, the results for that investigation were very similar to that reported in Section 7.4.

The experiments used Glimpse with a sparse logistic regression classifier [86]. The implementation of prototype clustering used in Chapter 6 was modified according to Algorithm 7.1. Unless otherwise noted, each experiment is repeated in five independent trials, and the mean and standard error of the performance is reported. In each trial, half of the images are chosen for training, with the other half reserved for testing. The set of prototypes is then chosen from the training images, and the classifier is trained on the resulting feature vectors. Performance is reported as the accuracy on the set of test images. The datasets used in these experiments are those for which an invariant representation (i.e., a representation composed of C2 activations) was found to be useful, as described in Section 4.2, and includes the *Animals* task of Serre et al. [2] and the *Cars v. Planes* task of Pinto et al. [18].

7.4 Results

Results are shown in Figure 7.5, which reports performance for new 1,000 prototypes learned from 10,000 initial prototypes. These results show no significant improvement in performance when the learner is focused on the foreground object. As before, the model's scaling behavior is also analyzed by measuring the performance for different number of prototypes, with results reported in Figure 7.6. This analysis shows that learning from image foregrounds often hurts per-

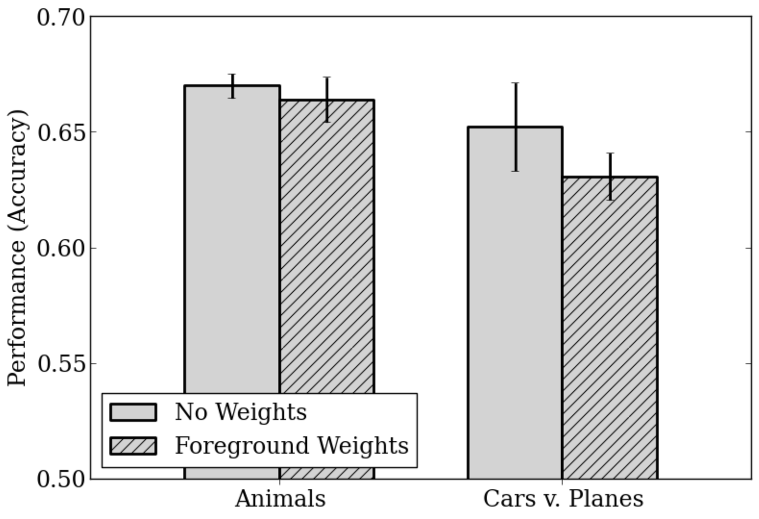
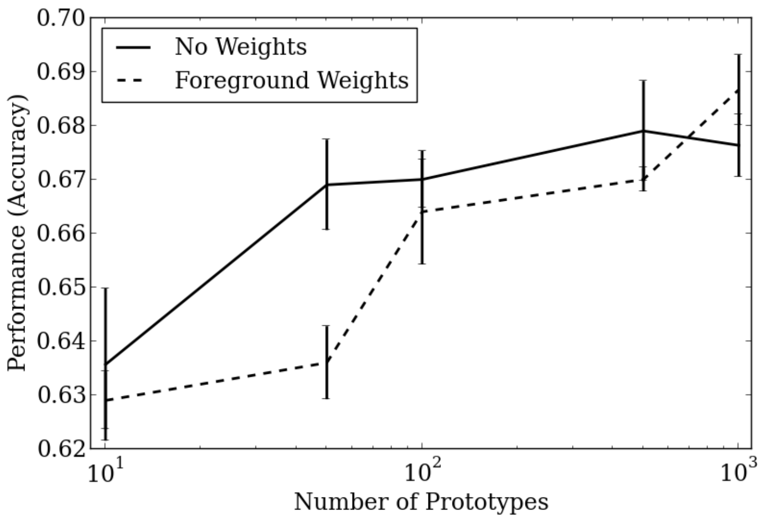


Figure 7.5: Comparison of performance for k-means prototypes learned without weights (gray) and with foreground weights (hatched gray). The vertical axis shows the mean performance (accuracy) over five independent trials, each using 100 prototypes with sparse logistic regression. Error bars show one standard error.

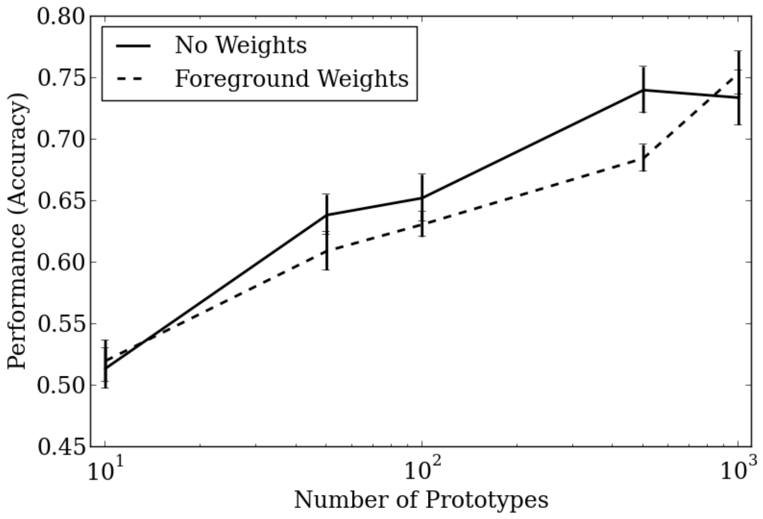
formance when Glimpse uses few prototypes. Thus, it appears that foreground weights have not helped the learner find an improved object representation.

7.4.1 Informative Backgrounds

This experiment assumes that only image foregrounds are useful for classification. However, it is possible that the poor results for learning from image foregrounds in Figure 7.5 are due to the presence of discriminative image backgrounds. To test this hypothesis, I ask whether prototypes drawn from the background are more or less useful than those drawn from the foreground object. To measure this, I generate prototypes, and record whether they overlap the object. Prototypes are weighted by classifier feedback using sparse logistic regression, and are said to be *used* by the classifier if their weight is non-zero. The event that a prototype has any overlap with a foreground object is modeled as a random



(a) *Animals*



(b) *Cars v. Planes*

Figure 7.6: Comparison of performance for different numbers of C2 features using k-means prototypes learned without weights (solid) and with foreground weights (dashed). Plots show mean performance (accuracy) over five independent trials, with error bars showing one standard error. Plot (a) shows results for the *Animals* task and plot (b) shows results for the *Cars v. Planes* task.

variable, with probability $P(\mathrm{fg})$. The event that a feature generated from a prototype is used by the classifier is modeled as another random variable, with probability $P(\mathrm{used})$. The *utility* of foreground prototypes is measured as

$$\mathrm{Utility}(\mathrm{fg}) = \log_2\left(\frac{P(\mathrm{used}\,|\,\mathrm{fg})}{P(\mathrm{used}\,|\,\neg\mathrm{fg})}\right), \tag{7.2}$$

where $P(\mathrm{used}\,|\,\mathrm{fg})$ and $P(\mathrm{used}\,|\,\neg\mathrm{fg})$ are the probabilities that foreground and background prototypes are used by the classifier, respectively. This measure is zero if foreground and background prototypes are equally likely to be useful for classification, and grows when foreground prototypes are more useful than background prototypes. A utility value of 1.0 indicates that foreground prototypes are twice as likely to be useful, compared with background prototypes.

Figure 7.7 reports the utility of foreground prototypes for the two datasets in Figure 7.5, where probabilities are estimated using 10,000 imprinted prototypes. The results suggest that recording prototypes from foreground objects should provide a significant benefit for the *Cars v. Planes* task, where foreground prototypes were used twice as often as background prototypes. In contrast, the benefit of foreground and background prototypes was nearly indistinguishable for the *Animals* task.

These results are somewhat surprising. Although experiments in Section 4.2 suggested that an invariant representation is useful for performing the *Animals* task, we see here that the prototypes composing this representation are often drawn from image backgrounds. Therefore, this dataset may be less informative for studies of object recognition than previously thought. Interestingly, these results agree with a previous study [83], which suggests that an HMAX-like model relies on the presence of "blurry" image backgrounds to perform the *Animals*

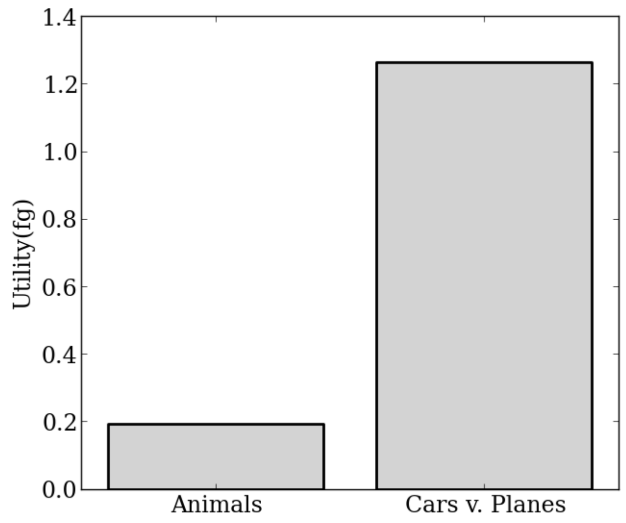


Figure 7.7: Utility of foreground prototypes for different datasets, where utility is defined as in Equation 7.2. This measure is zero if foreground and background prototypes are equally likely to be used for classification, with greater values indicating a preference for foreground prototypes. Here, a prototype is *used* if its feature is given a non-zero weight by the classifier, and is a *foreground* (fg) prototype if it was recorded from any part of a target object.

Corpus	Number Used
Animals	450
Cars v. Planes	110

Table 7.1: The number of prototypes given non-zero weight by the classifier, from a total set of 10,000 imprinted prototypes.

task.

Table 7.1 reports the total number of prototypes that were used by the classifier for each dataset. Note that this number varies considerably across the datasets, which appears to reflect the dataset's inherent difficulty. Relative to its behavior on the *Cars v. Planes* task, the classifier used four times more prototypes for the *Animals* dataset. This suggests that the "animal" category (or perhaps the "not animal" category) is harder to represent than rendered cars and planes.

7.5 Conclusions

In this chapter, I investigated a new method to learn prototypes from training images when task information is available. The new method was created by extending the clustering approach discussed in Chapter 6 to incorporate weights on prototypes. This approach was intended to capture the low computational cost of clustering, while using task information to learn prototypes that are discriminative.

Surprisingly, the addition of task information had very little effect on the performance of learned prototypes, with performance under the new approach being similar to—or worse than—a simple unweighted clustering. A follow-up experiment suggested that this poor performance is likely due to the method of weighted clustering, rather than the source of task information. Thus, it appears that weighted clustering is not effective for learning discriminative prototypes in HMAX-like models.

Chapter 8

Conclusions

This work investigated an influential family of hierarchical visual models, which have garnered significant interest in both the computer vision and neuroscience communities. Such models are applied to the problem of object recognition, in which the model must discriminate between different three-dimensional objects (such as automobiles) based solely on the visual input. The task is performed by exhaustively comparing patches of an image to a set of stored patterns, called prototypes, and the results are fed to a statistical classifier that predicts the category of object present. Prototypes are learned by imprinting patches from arbitrarily chosen training images.

While the choice of prototypes is thought to be crucial to the model's success, we lack a comprehensive understanding of the mechanisms by which prototypes mediate this success. This dissertation investigated those mechanisms, including the impact of prototypes on model performance, the benefits and limitations of adapting prototypes to new tasks, and the role of feedback in this adaptation. The contributions of this dissertation include the following.

- *A new hierarchical model is introduced (Chapter 3).* A novel framework was created, which allows the expression of a wide range of hierarchical visual models. This framework was used to implement a new visual model called Glimpse, the performance of which was shown to be competitive with existing HMAX-like models.

- *Limitations in common datasets are identified (Chapter 4).* I analyzed a number of benchmark datasets that have commonly been used in object recognition research, and find that many are inappropriate for testing viewpoint-invariant object recognition. Instead, I found that high performance on many of these datasets was possible by using only a simple representation that lacks viewpoint invariance.

- *The importance of imprinted shape is studied (Chapter 4).* I investigated a core assumption of HMAX-like models, namely that their success as widely reported in the literature was due to the learning of prototypes by imprinting. This was tested by comparing performance for imprinted prototypes and prototypes generated in a purely random fashion. Surprisingly, I found that performance was nearly identical for the two methods. While prototype learning is likely to be a crucial way to increase model performance, this result argues that the method of imprinting—assumed to be central to the model's success—may be entirely unnecessary.

- *The importance of feedback is demonstrated (Chapter 5).* I investigated the effect of feedback on model performance, and showed how feature selection can be used to choose a small set of discriminative prototypes. The results include an improvement in model performance, and a reduction in its computational cost.

- *Limitations in a common prototype learning method are found (Chapter 6).* The method of k-means clustering, often used in the hierarchical model literature, is shown to result in surprisingly poor performance in an HMAX-like model. The performance for prototypes learned by k-means clustering was found to be lower than those learned by imprinting.

- *A feedback-driven method for prototype learning is introduced (Chapter 7).* An extension of k-means clustering is described, which integrates task feedback into the learning process. Few assumptions are made of the form of feedback used, allowing the method to be applied in a wide range of situations. Compared with feature selection, the method requires far less computation and allows completely new prototypes to synthesized.

Chapter 9

Future Work

This work raises a number of important questions about the family of HMAX-like models, which should be addressed in future work. The most important goal is to construct a new theory, which describes how the family of HMAX-like models supports competitive performance on viewpoint-invariant object recognition tasks. Such a theory should account for the results of Section 4.3. Specifically, such a theory should explain how a randomly-constructed representation supports discrimination, despite its lack of shape specificity. Furthermore, such a theory should be able to predict when, if ever, a learned representation will out-perform random prototypes.

One source of inspiration for this theory may come from work on compressive sensing [98], in which random measurements are used to capture and store signals (such as visual or auditory signals) using a highly compressed representation. An important restriction is that the signals must be sparse. Here, signals are assumed to be generated by combining atoms of a dictionary[1], and are called *sparse* if each signal can be represented using a small (possibly different) subset

[1]For example, edge components form the atoms of one such dictionary for image data.

of those atoms.

The salient point in compressive sensing is the idea that random measurements have little effect on the "similarity" between sparse signals; if two signals are similar, then their random measurements will be similar. The questions that should be asked include the following. First, does the C1 layer provide a sparse representation—that is, is the C1 layer sparsely active—for natural images? This is very likely to be true, since past studies [107] suggest that a representation composed of localized edge filters does provide a sparse representation. Second, do features computed with random prototypes preserve the similarity between images? That is, do similar images result in similar feature vectors, if those feature vectors are computed using random prototypes. Under what definition of "similarity"? Third, how does a highly-nonlinear transformation, such as the pooling operation at C2, affect the results of compressive sensing?

Another interesting area of future work is to understand the role of additional layers of S-units in these models, which can be approached as an extension of the investigation in Section 4.2. Biological systems rely on many layers of processing, but it is unclear whether models such as HMAX derive any significant benefit from using, for example, an S3 layer. On one hand, a prototype in a high model layer can express more complex patterns than those in a low layer, since a given S-layer is defined in terms of the patterns captured in the S-layer below. This allows a deep model to use domain-specific patterns that are sparsely activated. On the other hand, a prototype in a high model layer is defined with respect to a larger image area than a prototype in a low layer, since each C-layer leads to more invariance. Thus, I expect model depth to be limited by the resolution of the input data (e.g., the number of pixels). Part of this future work should include an investigation of the role of depth in this family of models.

Finally, the study of prototype learning, both with and without the use of feedback, should be extended. The perplexing behavior for representations learned by k-means will be investigated, with a focus on the ways in which k-means prototypes differ from imprinted prototypes. In addition, I plan to extend my results for feedback-driven learning by comparing weighted clustering with alternative methods, such as supervised clustering [106], learned vector quantization [108], and the concurrent optimization of classifier and prototype dictionary as suggested by Boureau et al. [109]. Weighted clustering will also be compared with the general problem of instance-weighted learning [110], as discussed in the machine learning literature.

Bibliography

[1] M. Riesenhuber and T. Poggio, “Hierarchical models of object recognition in cortex,” *Nature Neuroscience*, vol. 2, pp. 1019–1025, Nov. 1999.

[2] T. Serre, A. Oliva, and T. Poggio, “A feedforward architecture accounts for rapid categorization,” *Proceedings of the National Academy of Sciences*, vol. 104, pp. 6424–6429, Apr. 2007.

[3] K. Fukushima, “Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position,” *Biological Cybernetics*, vol. 36, pp. 193–202, Apr. 1980.

[4] D. H. Hubel and T. N. Wiesel, “Receptive fields, binocular interaction and functional architecture in the cat’s visual cortex.,” *The Journal of Physiology*, vol. 160, pp. 106–54, Jan. 1962.

[5] T. Serre, G. Kreiman, M. Kouh, C. Cadieu, U. Knoblich, and T. Poggio, “A quantitative theory of immediate visual recognition,” *Progress in Brain Research, Computational Neuroscience: Theoretical Insights into Brain Function*, vol. 165, pp. 33–56, 2007.

[6] T. Serre, L. Wolf, and T. Poggio, “Object Recognition with Features Inspired by Visual Cortex,” *CVPR*, 2005.

[7] T. Serre, L. Wolf, S. Bileschi, M. Riesenhuber, and T. Poggio, "Robust object recognition with cortex-like mechanisms," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, pp. 411–426, Mar. 2007.

[8] S. J. Thorpe, D. Fize, and C. Marlot, "Speed of processing in the human visual system.," *Nature*, vol. 381, no. 6582, pp. 520–2, 1996.

[9] J. Mutch and D. G. Lowe, "Object Class Recognition and Localization Using Sparse Features with Limited Receptive Fields," *International Journal of Computer Vision*, vol. 80, pp. 45–57, Oct. 2008.

[10] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation Applied to Handwritten Zip Code Recognition," *Neural Computation*, vol. 1, pp. 541–551, Dec. 1989.

[11] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[12] Q. V. Le, M. A. Ranzato, R. Monga, M. Devin, G. S. Corrado, J. Dean, and A. Y. Ng, "Building High-level Features Using Large Scale Unsupervised Learning," in *Proceedings of the 29th International Conference on Machine Learning*, (Edinburgh, Scotland, UK), 2012.

[13] Y. Bengio, "Learning Deep Architectures for AI," *Foundations and Trends in Machine Learning*, vol. 2, pp. 1–127, Jan. 2009.

[14] Y. Huang, K. Huang, L. Wang, D. Tao, T. Tan, and X. Li, "Enhanced biologically inspired model," in *Proceedings of the 2008 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–8, Aug. 2008.

[15] E. Krupka, A. Navot, and N. Tishby, "Learning to Select Features using their Properties," *Journal of Machine Learning Research*, vol. 9, pp. 2349–2376, 2008.

[16] W. Zhu and L. Zhang, "Object Recognition with Task Relevant Combined Local Features," in *Proceedings of the International Conference on Intelligent Computing*, pp. 285–292, Springer Berlin / Heidelberg, 2008.

[17] N. Pinto, D. Cox, and J. J. Dicarlo, "Why is Real-World Visual Object Recognition Hard?," *PLoS Computational Biology*, vol. 4, Jan. 2008.

[18] N. Pinto, Y. Barhomi, D. D. Cox, and J. J. DiCarlo, "Comparing State-of-the-Art Visual Features on Invariant Object Recognition Tasks," in *Proceedings of the IEEE Workshop on Applications of Computer Vision (WACV 2011)*, 2011.

[19] Y. Wu, N. Zheng, Q. You, and S. Du, "Object Recognition by Learning Informative, Biologically Inspired Visual Features," in *Proceedings of the IEEE International Conference on Image Processing – ICIP 2007*, 2007.

[20] P. Moreno, M. J. Marín-Jiménez, A. Bernardino, J. Santos-Victor, and N. P. D. L. Blanca, "A Comparative Study of Local Descriptors for Object Category Recognition: SIFT vs HMAX," in *Pattern Recognition and Image Analysis*, pp. 515–522, Springer Berlin / Heidelberg, 2007.

[21] Q.-S. Lian and Q. Li, "Object Recognition Based on Biologic Visual Mechanisms," in *Proceedings of the 2008 Congress on Image and Signal Processing*, pp. 386–390, IEEE, 2008.

[22] J.-W. Woo, Y.-C. Lim, and M. Lee, "Obstacle Categorization Based on Hybridizing Global and Local Features," in *ICONIP 2009* (C. S. Leung, M. Lee, and J. H. Chan, eds.), (Berlin, Heidelberg), pp. 1–10, Springer Berlin Heidelberg, 2009.

[23] R. G. J. Wijnhoven and P. H. N. de With, "Advanced Concepts for Intelligent Vision Systems," in *Advanced Concepts for Intelligent Vision Systems 2009* (J. Blanc-Talon, W. Philips, D. Popescu, and P. Scheunders, eds.), pp. 410–421, Springer, 2009.

[24] S. Bileschi and L. Wolf, "Image representations beyond histograms of gradients: The role of Gestalt descriptors," in *Proceedings of the 2007 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–8, IEEE, June 2007.

[25] P. Mishra and B. K. Jenkins, "Hierarchical model for object recognition based on natural-stimuli adapted filters," in *Proceedings of the 2010 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 950–953, IEEE, 2010.

[26] K. Faez, S. Motamed, and M. Yaqubi, "Personal verification using ear and palm-print biometrics," in *Proceedings of the 2008 IEEE International Conference on Systems, Man and Cybernetics*, pp. 3727–3731, IEEE, Oct. 2008.

[27] Z. Yaghoubi, K. Faez, M. Eliasi, and S. Motamed, "Face recognition using HMAX method for feature extraction and support vector machine classifier," in *Proceedings of the 24th International Conference on Image and Vision Computing New Zealand – IVCNZ'09*, pp. 421–424, IEEE, Nov. 2009.

[28] W. Gu, C. Xiang, and H. Lin, "Modified HMAX models for facial expression recognition," in *Proceedings of the 2009 IEEE International Conference on Control and Automation*, pp. 1509–1514, IEEE, Dec. 2009.

[29] S. P. Brumby, G. Kenyon, W. Landecker, C. Rasmussen, S. Swaminarayan, and L. Bettencourt, "Large-Scale Functional Models of Visual Cortex for Remote Sensing," in *Applied Imagery Pattern Recognition 2009 (AIPR '09)*, 2009.

[30] D. Walther, L. Itti, M. Riesenhuber, T. Poggio, and C. Koch, "Attentional Selection for Object Recognition — A Gentle Way," in *Biologically Motivated Computer Vision* (H. H. Bülthoff, C. Wallraven, S.-W. Lee, and T. A. Poggio, eds.), pp. 251–267, Springer, 2002.

[31] S. S. Chikkerur, T. Serre, C. Tan, and T. Poggio, "What and where: A Bayesian inference theory of attention," *Vision Research*, vol. 50, pp. 2247–2233, May 2010.

[32] B. Han, X. Gao, V. Walsh, and L. Tcheang, "A saliency map method with cortex-like mechanisms and sparse representation," in *Proceedings of the ACM International Conference on Image and Video Retrieval - CIVR '10*, (New York, New York, USA), p. 259, ACM Press, July 2010.

[33] H. Wersing and E. Körner, "Learning optimized features for hierarchical models of invariant object recognition.," *Neural Computation*, vol. 15, pp. 1559–88, July 2003.

[34] W. Einhäuser, C. Kayser, K. Körding, and P. König, "Learning Multiple Feature Representations from Natural Image Sequences," in *Artificial Neural Networks - ICANN 2002*, p. 788, 2002.

[35] P. O. Hoyer and A. Hyvärinen, "A multi-layer sparse coding network learns contour coding from natural images," *Vision Research*, vol. 42, pp. 1593–1605, June 2002.

[36] R. Miikkulainen, J. A. Bednar, Y. Choe, and J. Sirosh, *Computational Maps in the Visual Cortex.* 2005.

[37] E. T. Rolls and T. Milward, "A model of invariant object recognition in the visual system: learning rules, activation functions, lateral inhibition, and information-based performance measures.," *Neural Computation*, vol. 12, pp. 2547–72, Nov. 2000.

[38] T. Serre, *Learning a Dictionary of Shape-Components in Visual Cortex: Comparison with Neurons, Humans, and Machines.* Phd thesis, Massachusetts Institute of Technology, Cambridge, Apr. 2006.

[39] T. Serre, M. Riesenhuber, J. Louie, and T. Poggio, "On the Role of Object-Specific Features for Real World Object Recognition in Biological Vision," in *Biologically Motivated Computer Vision* (H. H. Bülthoff, C. Wallraven, S.-W. Lee, and T. A. Poggio, eds.), pp. 209–218, Springer, 2002.

[40] J. Weston, S. Mukherjee, O. Chapelle, M. Pontil, T. Poggio, and V. Vapnik, "Feature Selection for SVMs," in *Neural Information Processing Systems 2000*, pp. 668–674, 2000.

[41] Andrew Y Ng, "Feature selection, L1 vs. L2 regularization, and rotational invariance," in *Proceedings of the Twenty-First International Conference on Machine Learning*, (Banff, Canada), 2004.

[42] I. M. Guyon and A. Elisseeff, "An Introduction to Variable and Feature Selection," *Journal of Machine Learning Research*, vol. 3, pp. 1157–1182, Mar. 2003.

[43] K.-L. Du, "Clustering: a neural network approach.," *Neural networks*, vol. 23, pp. 89–107, Jan. 2010.

[44] J. Louie, *A Biological Model of Object Recognition with Feature Learning.* Masters, Massachusetts Institute of Technology, 2003.

[45] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, vol. 1, pp. 281–297, Univ. of Calif. Press, 1967.

[46] P. Földiák, "Learning invariance from transformation sequences," *Neural Comput.*, vol. 3, pp. 194–200, June 1991.

[47] L. Wiskott and T. J. Sejnowski, "Slow Feature Analysis: Unsupervised Learning of Invariances," *Neural Computation*, vol. 14, pp. 715–770, Apr. 2002.

[48] W. Einhäuser, C. Kayser, P. König, and K. Körding, "Learning the invariance properties of complex cells from their responses to natural stimuli," *European Journal of Neuroscience*, vol. 15, no. 3, pp. 475–486, 2002.

[49] M. W. Spratling, "Learning viewpoint invariant perceptual representations from cluttered images," *IEEE transactions on pattern analysis and machine intelligence*, vol. 27, pp. 753–61, May 2005.

[50] S. M. Stringer, G. Perry, E. T. Rolls, and J. H. Proske, "Learning invariant object recognition in the visual system with continuous transformations.," *Biological cybernetics*, vol. 94, pp. 128–42, Feb. 2006.

[51] T. Masquelier, T. Serre, S. J. Thorpe, and T. Poggio, "Learning complex cell invariance from natural videos: A plausibility proof," tech. rep., Massachusetts Institute of Technology, Cambridge, MA, 2007.

[52] G. Wallis and E. T. Rolls, "Invariant face and object recognition in the visual system," *Progress in Neurobiology*, vol. 51, pp. 167–194, Feb. 1997.

[53] N. Li and J. J. DiCarlo, "Unsupervised natural experience rapidly alters invariant object representation in visual cortex.," *Science (New York, N.Y.)*, vol. 321, no. 5895, pp. 1502–7, 2008.

[54] M. D. Thomure, M. Mitchell, and G. T. Kenyon, "On the Role of Shape Prototypes in Hierarchical Models of Vision," in *International Joint Conference on Neural Networks (IJCNN)*, 2013.

[55] L. Fei-Fei, R. Fergus, and P. Perona, "Learning generative visual models from few training examples: an incremental Bayesian approach tested on

101 object categories," in *CVPR 2004, Workshop on Generative-Model Based Vision*, 2004.

[56] R. Fergus, P. Perona, and A. Zisserman, "Object class recognition by unsupervised scale-invariant learning," *Computer Vision and Pattern Recognition*, vol. 2, pp. 264–271, 2003.

[57] D.-S. Pham and S. Venkatesh, "Joint learning and dictionary construction for pattern recognition," in *Proceedings of the 2008 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–8, IEEE, June 2008.

[58] M. Ranzato, F. J. Huang, Y.-L. Boureau, and Y. LeCun, *Unsupervised Learning of Invariant Feature Hierarchies with Applications to Object Recognition*. IEEE, June 2007.

[59] R. Rigamonti, M. A. Brown, and V. Lepetit, "Are sparse representations really relevant for image classification?," in *CVPR 2011*, pp. 1545–1552, IEEE, June 2011.

[60] T. Masquelier and S. J. Thorpe, "Unsupervised Learning of Visual Features through Spike Timing Dependent Plasticity," *PLoS Computational Biology*, vol. 3, Feb. 2007.

[61] C. Thériault, N. Thome, and M. Cord, "Extended coding and pooling in the HMAX model.," *IEEE transactions on image processing : a publication of the IEEE Signal Processing Society*, vol. 22, pp. 764–77, Feb. 2013.

[62] S. P. Brumby, L. M. Bettencourt, M. I. Ham, R. A. Bennett, and G. Kenyon, "Quantifying the difficulty of object recognition tasks via scaling of accuracy versus training set size," in *COSYNE*, 2010.

[63] R. Raina, A. Battle, H. Lee, B. Packer, and A. Y. Ng, "Self-taught Learning: Transfer Learning from Unlabeled Data," in *Proceedings of the Twenty-Fourth International Conference on Machine Learning*, (Corvallis, Oregon), AAAI Press, 2007.

[64] K. Jarrett, K. Kavukcuoglu, M. A. Ranzato, and Y. LeCun, "What is the best multi-stage architecture for object recognition?," in *Proceedings of the 2009 IEEE 12th International Conference on Computer Vision*, pp. 2146–2153, IEEE, Sept. 2009.

[65] K. Kavukcuoglu, P. Sermanet, Y.-L. Boureau, K. Gregor, M. Mathieu, and Y. LeCun, "Learning Convolutional Feature Hierarchies for Visual Recognition," in *Advances in Neural Information Processing Systems 23* (J. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, eds.), 2010.

[66] F. Rodriguez and G. Sapiro, "Sparse Representations for Image Classification: Learning Discriminative and Reconstructive Non-Parametric Dictionaries," tech. rep., University of Minnesota, Minneapolis, Minnesota, 2007.

[67] J. Yang, K. Yu, Y. Gong, and T. Huang, "Linear spatial pyramid matching using sparse coding for image classification," in *Computer Vision and Pattern Recognition*, pp. 1794–1801, IEEE, June 2009.

[68] F. Perronnin, J. Senchez, and Y. L. Xerox, "Large-scale image categorization with explicit data embedding," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 2297–2304, IEEE, June 2010.

[69] G. Griffin, A. D. Holub, and P. Perona, "The Caltech-256," tech. rep., Caltech, 2007.

[70] S. M. Crouzet and T. Serre, "What are the visual features underlying rapid object recognition?," *Frontiers in Perception Science*, vol. 2, 2011.

[71] J. Ponce, T. Berg, M. Everingham, D. Forsyth, M. Hebert, S. Lea, M. Marszalek, C. Schmid, B. C. Russell, A. Torralba, C. Williams, J. Zhang, and A. Zisserman, "Dataset Issues in Object Recognition," in *Toward Category-Level Object Recognition*, pp. 29–48, 2006.

[72] Y. LeCun, D. G. Lowe, J. Malik, J. Mutch, P. Perona, and T. Poggio, "Object Recognition, Computer Vision, and the Caltech 101: A Response to Pinto et al.," Mar. 2008.

[73] A. Torralba and A. A. Efros, "Unbiased look at dataset bias," in *CVPR 2011*, pp. 1521–1528, IEEE, June 2011.

[74] N. Pinto, "Personal Communication."

[75] J. Geusebroek and A. Smeulders, "The Amsterdam library of object images," *International Journal of Computer Vision2*, vol. 61, no. 1, pp. 103–112, 2005.

[76] B. Leibe and B. Schiele, "Analyzing Appearance and Contour Based Methods for Object Categorization," in *International Conference on Computer Vision and Pattern Recognition (CVPR'03)*, (Madison, Wisconsin), 2003.

[77] Y. LeCun, F. J. Huang, and L. Bottou, "Learning methods for generic object recognition with invariance to pose and lighting," in *Proceedings*

of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004., pp. 97–104, IEEE, 2004.

[78] S. A. Nene, S. K. Nayar, and H. Murase, "Columbia Object Image Library (COIL-100)," technical report, Columbia University, 1996.

[79] A. Oliva and A. Torralba, "Modeling the Shape of the Scene: A Holistic Representation of the Spatial Envelope," *International Journal of Computer Vision*, vol. 42, pp. 145–175, May 2001.

[80] J. Xiao, J. Hays, K. Ehinger, A. Oliva, and A. Torralba, "SUN Database: Large-scale Scene Recognition from Abbey to Zoo," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2010.

[81] A. Torralba and A. Oliva, "Statistics of natural image categories," *Network: Computation in Neural Systems*, vol. 14, pp. 391–412, Aug. 2003.

[82] A. Torralba, "Contextual Priming for Object Detection," *Int. J. Comput. Vision*, vol. 53, pp. 169–191, July 2003.

[83] W. Landecker, M. D. Thomure, L. M. A. Bettencourt, M. Mitchell, G. T. Kenyon, and S. P. Brumby, "Interpreting Individual Classifications of Hierarchical Networks," in *Computational Intelligence and Data Mining - CIDM 2013, Special session on Interpretable Systems in Machine Learning.*, 2013.

[84] "Source code for the Glimpse model." `http://web.cecs.pdx.edu/~thomure/glimpse/`.

[85] C. Bishop, *Pattern Recognition and Machine Learning.* Springer, Oct. 2006.

[86] S. K. Shevade and S. S. Keerthi, "A simple and efficient algorithm for gene selection using sparse logistic regression," *Bioinformatics*, vol. 19, pp. 2246–2253, Nov. 2003.

[87] D. Mladenić, J. Brank, M. Grobelnik, and N. Milic-Frayling, "Feature selection using linear classifier weights: interaction with classification models," in *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2004.

[88] P. Burt and E. Adelson, "The Laplacian Pyramid as a Compact Image Code," *IEEE Transactions on Communications*, vol. 31, pp. 532–540, Apr. 1983.

[89] D. Forsyth and J. Ponce, *Computer Vision: A Modern Approach*. Prentice Hall, Aug. 2002.

[90] D. G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," *International Journal of Computer Vision*, vol. 60, pp. 91–110, Nov. 2004.

[91] D. J. Heeger, "Normalization of cell responses in cat striate cortex.," *Visual neuroscience*, vol. 9, pp. 181–97, Aug. 1992.

[92] O. Schwartz and E. P. Simoncelli, "Natural signal statistics and sensory gain control.," *Nature neuroscience*, vol. 4, pp. 819–25, Aug. 2001.

[93] "Source code for the HMAX and SLF models." `http://cbcl.mit.edu/software-datasets/pnas07/`.

[94] N. Pinto, Z. Stone, T. Zickler, and D. Cox, "Scaling up biologically-inspired computer vision: A case study in unconstrained face recognition

on facebook," in *CVPR 2011 Workshop on Biologically-Consistent Vision*, pp. 35–42, IEEE, June 2011.

[95] A. Saxe, P. W. Koh, Z. Chen, M. Bhand, B. Suresh, and A. Y. Ng, "On random weights and unsupervised feature learning," in *NIPS 2010 Workshop on Deep Learning and Unsupervised Feature Learning*, 2010.

[96] Y. Freund and R. Schapire, "Experiments with a new boosting algorithm," in *Proceedings of the Thirteenth International Conference on Machine Learning*, pp. 148–156, 1996.

[97] J. Wright, A. Y. Yang, A. Ganesh, S. S. Sastry, and Y. Ma, "Robust face recognition via sparse representation.," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, pp. 210–27, Feb. 2009.

[98] J. Wright, Y. Ma, J. Mairal, G. Sapiro, T. Huang, and S. Yan, "Sparse Representation for Computer Vision and Pattern Recognition," *Proceedings of the IEEE*, vol. 98, no. 6, pp. 1031–1044, 2010.

[99] S. S. Chikkerur, C. Tan, T. Serre, and T. Poggio, "An integrated model of visual attention using shape-based features," tech. rep., Massachusetts Institute of Technology, Cambridge, MA, June 2009.

[100] A. Coates and A. Y. Ng, "Learning Feature Representations with K-means," in *Neural Networks: Tricks of the Trade* (G. Montavon, G. B. Orr, and K.-R. M uller, eds.), Springer, 2nd ed., 2012.

[101] S. Lloyd, "Least squares quantization in PCM," *IEEE Transactions on Information Theory*, vol. 28, pp. 129–137, Mar. 1982.

[102] D. Sculley, "Web-scale k-means clustering," in *Proceedings of the 19th international conference on World wide web - WWW '10*, (New York, New York, USA), p. 1177, ACM Press, 2010.

[103] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[104] T. Serre, M. Kouh, C. Cadieu, U. Knoblich, G. Kreiman, and T. Poggio, "A Theory of Object Recognition: Computations and Circuits in the Feedforward Path of the Ventral Stream in Primate Visual Cortex," tech. rep., Massachusetts Institute of Technology, Cambridge, MA, 2005.

[105] T. Kohonen, "Improved versions of learning vector quantization," in *1990 IJCNN International Joint Conference on Neural Networks*, pp. 545–550, IEEE, 1990.

[106] S. Lazebnik and M. Raginsky, "Supervised learning of quantizer codebooks by information loss minimization.," *IEEE transactions on pattern analysis and machine intelligence*, vol. 31, pp. 1294–309, July 2009.

[107] D. J. Field, "Relations between the statistics of natural images and the response properties of cortical cells.," *Journal of the Optical Society of America. A, Optics and image science*, vol. 4, pp. 2379–94, Dec. 1987.

[108] P. Schneider, M. Biehl, and B. Hammer, "Distance learning in discriminative vector quantization.," *Neural computation*, vol. 21, pp. 2942–69, Oct. 2009.

[109] Y.-L. Boureau, F. Bach, Y. LeCun, and J. Ponce, “Learning mid-level features for recognition,” in *Computer Vision and Pattern Recognition 2010*, pp. 2559–2566, IEEE, June 2010.

[110] N. Karampatziakis and J. Langford, “Online Importance Weight Aware Updates,” in *Proceedings of Uncertainty in Artificial Intelligence (UAI-11)*, (Corvallis, Oregon), pp. 392—-399, AUAI Publishers, 2011.