Fall 12-14-2015

The Performance of Random Prototypes in Hierarchical Models of Vision

Kendall Lee Stewart
Portland State University

The Performance of Random Prototypes in Hierarchical Models of Vision

by

Kendall Lee Stewart

A thesis submitted in partial fulfillment of the
requirements for the degree of

Master of Science
in
Computer Science

Thesis Committee:
Melanie Mitchell, Chair
Jonathan Walpole
Feng Liu

Portland State University
2015

Abstract

I investigate properties of HMAX, a computational model of hierarchical processing in the primate visual cortex. High-level cortical neurons have been shown to respond highly to particular natural shapes, such as faces [3]. HMAX models this property with a dictionary of natural shapes, called prototypes, that respond to the presence of those shapes. The resulting set of similarity measurements is an effective descriptor for classifying images. Curiously, prior work [27] has shown that replacing the dictionary of natural shapes with entirely random prototypes has little impact on classification performance. This work explores that phenomenon by studying the performance of random prototypes on natural scenes, and by comparing their performance to that of sparse random projections of low-level image features.

Acknowledgements

This work would not have been possible without the advice and support of my advisor, Melanie Mitchell, and the members of the Mitchell Research Group at Portland State University: Sheng Lundquist, Anthony Rhodes, Max Quinn, Efsun Sarioglu, and Jordan Witte. Credit is also due to Mick Thomure for his PhD dissertation, which forms the foundation of this work.

A special thanks goes to Jonathan Walpole for his mentorship and support during my time at Portland State. I'd also like to thank Feng Liu for introducing me to a wide spectrum of computer vision techniques.

Many thanks to Garrett Kenyon, Pete Schultz, Wesley Chavez, Max Theiler, Brian Broom-Peltz, Xinhua Zhang, Will Shainin, and the other members of the PetaVision research group at the New Mexico Consortium, for taking in a wandering graduate student and broadening his horizons.

Additional thanks the other Portland State faculty that have helped to shape my fledgling academic career: Sergio Antoy, Karla Fant, Jim Hook, Mark Jones, Karen Karavanic, Bart Massey, Tim Sheard, Andrew Tolmach, and Bryant York.

Finally, a big thanks to all my friends and family, and a planet-sized thanks to my partner Dot for her love and forbearance.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This work concerns properties of the hierarchical vision model known as HMAX [20, 25], which attempts to provide a quantitative implementation of well-known properties of object recognition in the primate visual cortex. In addition to its use as a tool in neuroscience, HMAX has found more general applications in computer vision systems, particularly for object recognition tasks [18, 19, 25].

An important part of the structure of HMAX is a set of learned features, known in the literature as *prototypes*, which are designed to measure the presence of a particular shape somewhere in a target image. These prototypes mimic neurons in the primate cortex that have shown to be selective to particular specific shapes, such as faces [3].

This set of prototypes is sometimes called a *dictionary* of shapes. The HMAX architecture proposed by Serre et al. [25] learns this dictionary by extracting patches from a dataset of natural images (a process called "imprinting"). Serre et al. claim that the resulting shape-tuned units are key to providing a "rich representation" for object recognition [23].

Other methods of prototype learning beyond imprinting are explored in prior work by Thomure [27]. One important and perplexing result of Thomure's study is that a learned dictionary can be replaced with a dictionary of entirely random values, with little impact on performance. This contradicts the claim of Serre et al. that learned features are central to the model's performance. I seek to explore and explain this phenomenon.

The central questions of my work are: (1) whether the discriminative ability of random prototypes hold for larger and more diverse datasets than the ones used by

Thomure, and (2) whether the mechanism behind the discriminative ability of random prototypes can be sufficiently explained.

My contribution to the first question is to extend Thomure's result on small synthetic datasets to the larger and more diverse PASCAL VOC 2012 dataset [8] of natural images. I find that, for the most part, random prototypes still have relatively good performance even on this large dataset. Furthermore, I observe that some random prototypes do seem to be have a limited preference to particular natural shapes, contrary to what Thomure observed.

My contribution to the second question is to investigate the hypothesis, posed by Thomure, that random prototypes can be explained by random projection [1, 17], which is a technique for dimensionality reduction that has found recent popularity in machine learning and data processing.

The rest of this thesis is laid out as follows. In Chapter 2, I give some additional background on the history of HMAX, as well as details of the prior work by Thomure. In Chapter 3, I outline the operation of Glimpse, the HMAX-like model developed and used by Thomure. In Chapter 4, I discuss the methods I used to test Glimpse on the PASCAL VOC 2012 dataset, and present results from those tests. Chapter 5 describes *feature selection*, a method of improving model performance with task-based feedback, and gives results. Chapter 6 explores random projection, including methods and experimental results. Chapter 7 presents conclusions and possibilities for future work.

Chapter 2

Background and Related Work

2.1 The Task of Object Recognition

The models under investigation in this work are designed for the task of visual object recognition. This task involves presenting an image that contains some object, and asking a yes-no categorization question, e.g., "does this image contain an animal or not?" Humans and non-human primates excel at this task, performing it quickly and accurately [10, 29].

A computer vision system that performs this task typically does so by first performing *feature extraction*, where images are transformed from their low-level pixel representations to high-level feature vectors. These high-level features should be robust against variation and clutter, in order to assist separating images into categories. Usually, feature extraction is also a *dimensionality reduction*, where the length of the feature vector is less than the number of pixels in the input image.

The categorical separation itself is performed by a classifier such as a support vector machine (SVM) [6]. The dataset is first split into two subsets, training data and testing data. The labels of the training data, along with their feature vectors, are fed to the classifier, which attempts to create a mapping from feature vectors to labels. The classifier is then given the feature vectors of the testing data (without labels), and it outputs prediction values which can be thresholded to obtain binary labels.

My work concerns the comparison of different methods of feature extraction for the task of object recognition. To evaluate the effectiveness of a particular feature

extraction method, I will measure the performance of a classifier (a linear SVM) that is trained and tested using this method. Metrics for assessing the performance of a classifier will be described in a later section.

2.2 HMAX

The computational model investigated in this work is HMAX [20, 25], a feature extractor that attempts to provide a quantitative implementation of the "standard model of object recognition", which is a set of well-known properties about object recognition in the primate visual cortex [24].

2.2.1 Background

According to the standard model of object recognition, the ventral stream of the primate visual cortex consists of a feed-forward hierarchy in which higher-level neurons encode increasingly invariant and abstract representations of visual stimuli, resulting in a representation that the brain can use for rapid categorization [23].

HMAX implements this visual pathway with a layer-wise processing model consisting of alternating "simple" and "complex" layers, following the terminology of Hubel and Wiesel [13], who first discovered a hierarchy of increasing invariance in the responses of neurons in the cat's visual cortex.

"Simple" cells in the visual cortex are highly *selective* for particular low-level visual features. For instance, one simple cell might respond highly only when presented with a vertical bar of light in the center of the visual field, while a different simple cell responds highly only when presented with the same vertical bar shifted to the left or right.

"Complex" cells, on the other hand, add a level of *invariance*: a complex cell receiving input from the two simple cells described above would respond highly to all

vertical bars, regardless of position within the visual field.

In HMAX, layers that model "simple" cell operations perform a selective operation, where the output is a spatial map of the similarity between the layer below and some set of features, e.g. prototypical edges or shapes.

One way to implement this selectivity is to represent the features as vectors, and then compare them against the input with an operation that measures similarity between vectors. In this context, features are usually called *kernels* or *filters*. At all locations in the input, a neighborhood of values (which I will refer to as a *patch*) that is the same size as the filter is centered at that location, and the operation is applied.

An example is given in Figure 2.1. Here, the filter to the right of the $*$ symbol (not rendered to scale) is compared with every 11×11 patch in the image to the left, using a rectified dot product. For an input patch $\mathbf{x} = x_1 \dots x_n$ and kernel $\mathbf{d} = d_1 \dots d_n$, this is defined as:

$$|\mathbf{x} \cdot \mathbf{d}| = \left| \sum_{i=1}^{n} x_i d_i \right| \tag{2.1}$$

The output is shown on the right. It is smaller than the input by 10 pixels in each dimension because the patches are 11×11 and thus must be centered 5 pixels away from the edges of the image.

Note that this particular filter acts as an edge detector. Locations in the input that contain edges of a particular orientation and size have higher values in the output. The effect of the rectification (taking the absolute value) is to make the filter *phase-invariant*. It responds equally to light edges on a dark background and dark edges on a light background.

In the context of image processing, the operation where a filter is applied to all locations in an input using a dot product is called a *convolution*, denoted by

the $*$ symbol. In a general convolution, the dot product is not rectified; however, rectification can be applied after the convolution with the same effect.

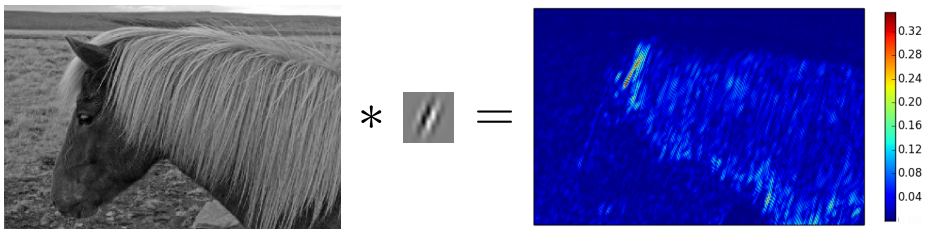


Figure 2.1: Illustration of a selective operation. The output is the result of comparing every patch in the input with the feature, using a rectified dot product (Equation 2.1). Here, the feature is an edge filter (enlarged to show detail). The resulting output is a "feature map" indicating the degree to which each input patch contains the slanted edge represented by the feature. Figure best viewed in color.

Layers that model "complex" cell operations add invariance, meaning that two different inputs could potentially map to the same output. HMAX implements invariance with an operation called *max-pooling*. Max-pooling also considers every patch in the input, but the result is simply the maximum value within that patch. The output is a spatial map of local maxima from the input. An example is given in Figure 2.2, where the output from Figure 2.1 is max-pooled with an patch size of 11×11. The overall effect is that the output is invariant to small translations: shifting the image around a little bit will not change the result.

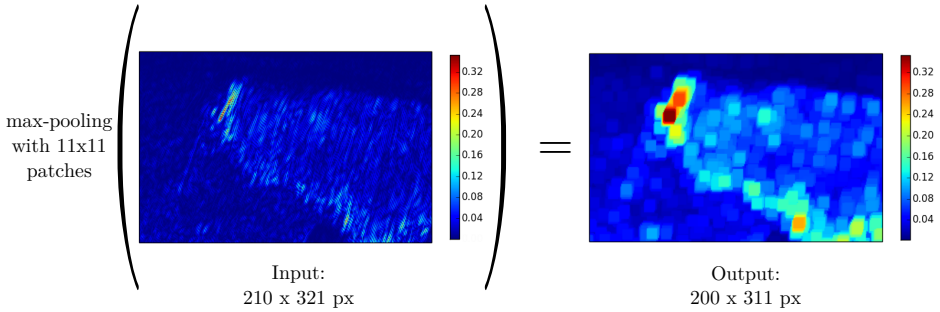


Figure 2.2: Illustration of max-pooling. The output is a map of the maximum values in each 11×11 patch from the input. Figure best viewed in color.

2.2.2 Implementation History

In Riesenhuber and Poggio's original implementation of HMAX [20], there are four layers, alternating between selectivity and invariance.

The first layer is selective, and is called *S1* (after "simple"). Following Hubel and Weisel's evidence that neurons early in the ventral stream are selective to edges, Riesenhuber and Poggio design S1 to apply a set of edge filters to the image using the rectified dot product (Equation 2.1). Each filter is sensitive to one of four orientations at a particular scale. This results in a layer composed of S1 *units* for each spatial location in the input, organized into *bands* for each scale and each of four orientations.

The second layer is called *C1* (after "complex"). The units in this layer max-pool independently over each orientation band in S1, using values from neighboring locations *and* scales. Thus C1 provides invariance not only to translation of edges, but also to edge size.

The third layer, *S2*, corresponds to neurons higher up in the ventral stream that are selective to particular shapes. Riesenhuber and Poggio's S2 layer models shapes as configurations of edges. Since each C1 unit measures the presence of an edge of a particular orientation at that location, each S2 unit takes input from a different configuration of C1 units in a 2×2 patch, and combines them with a Gaussian transfer function.

They restrict the choice of C1 units in the configuration to be from one orientation band at each location. With four orientations and a 2×2 patch, this results in $4^{2\times 2} = 256$ possible ways to combine them, and thus there are 256 types of S2 units.

Finally, the *C2* layer max-pools over the values for each type of S2 unit at all locations and scales, resulting in just one C2 value for each type of S2 unit (in this case, 256). Each value in C2 thus measures the presence of a particular shape (as

a configuration of C1 values) somewhere in the input image. The C2 vector is the result of the feature extraction process, and could be used as input to a classifier.

While Riesenhuber and Poggio showed that their model effectively reproduced the selectivity and invariance properties of cortical neurons, their goal in creating HMAX was mostly scientific. The main problem with applying their implementation to more general computer vision tasks is that the complexity of computing the S2 layer grows exponentially with the patch size.

Serre et al. [25] improved and generalized the model by redefining S2. Their model avoids the need for an exhaustive search of the configuration space by sampling random patches of C1 activity from images in a training set to create a set (or "dictionary") of shape *prototypes*. They refer to this learning process as *imprinting* [23]. These prototypes are then used as the basis for S2's selectivity, as follows:

For every location $\mathbf{x}$ in C1, each prototype $\mathbf{p}$ is compared against the patch using a radial basis function:

$$\mathrm{S2}(\mathbf{x}, \mathbf{p}) = \exp(-\beta ||\mathbf{x} - \mathbf{p}||^2) \tag{2.2}$$

Where $||\mathbf{x} - \mathbf{p}|| = \sqrt{\sum_i (x_i - p_i)^2}$ denotes the Euclidean distance (dissimilarity) between $\mathbf{x}$ and $\mathbf{p}$. Higher values of β will lead to a larger penalty for dissimilarity.

The result will be a set of spatial maps, one for each prototype, indicating how well each patch of C1 matched the configuration of that prototype. An example is given in Figure 2.3. Since a prototype is a patch of C1 values, it can be difficult to visualize. This figure instead shows the *receptive field* of an imprinted prototype, which is the set of pixels from the image that contributed to the C1 values in the prototype. A few sample images that cause high S2 responses to this prototype are shown to the right, annotated with the corresponding S2 activation map. This demonstrates the

selectivity of the prototype. It responds highly to the same natural shape (horse legs), as well as to similar natural shapes (dog legs), but also to similar configurations of edges (tree branches). Note that the response is invariant to changes in location and scale.

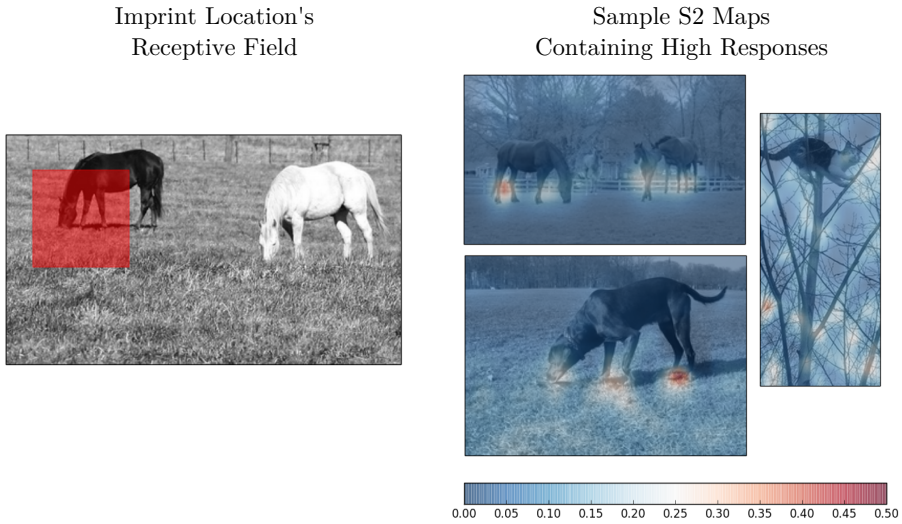


Figure 2.3: Illustration of S2 prototype selectivity. The receptive field of the imprinted prototype is shown on the left. On the right are S2 maps for a few sample images that cause high responses for this prototype.

These S2 maps are then max-pooled over all locations to produce a single C2 value for each prototype. Thus each prototype's C2 value measures how well it matched some patch in an input image. In the example in Figure 2.3, the C2 value for this prototype would measure the presence of a horse-leg-like shape somewhere in the input.

If a large number of positive training examples (those that contain horses) have high C2 values for this prototype, it will help push a classifier to predict that a horse-leg-like shape somewhere in an unlabeled image probably means that the image contains a horse, and it will give it a positive label.

As shown in Figure 2.3, this prototype also causes high responses in images that

do not contain horses. However, C2 values of additional prototypes should provide more information that can improve predictions.

An ideal dictionary of prototypes for the object recognition task will produce C2 representations for each image that highlight information necessary for classification, while discarding irrelevant variation between examples.

2.3 The Role of Shape Prototypes

Serre et al. [25] showed that their version of HMAX does well as a feature extractor for the task of object recognition, and they claim that prototype learning by imprinting from natural images "builds a generic dictionary of shape-tuned units which provides a rich representation" for object recognition tasks [23].

The effectiveness of learning by imprinting is a central question addressed by Thomure [27, 28], who studies prototype learning by imprinting, clustering, and feature selection, as well as the performance of S2 prototypes with random values.

One surprising result of Thomure's work is that features extracted by a dictionary of entirely random prototypes perform about as well on object recognition tasks as those extracted by a dictionary of imprinted prototypes. This holds true even for difficult tasks that HMAX is supposedly well-suited for.

For example, Pinto et al. [19] show that HMAX (specifically, the Sparse Localized Features (SLF) variant of Mutch and Lowe [18], which also uses imprinted prototypes) outperforms other models on invariant recognition tasks, in which objects are presented at varying degrees of translation and rotation. Pinto et al.'s study controls for this effect using synthetic images where the variation is a parameter of the image-generation process. Samples from each variation level of the "Cars vs. Planes" recognition task are shown in Figure 2.4.

Thomure showed that the performance results in Pinto et al. could be reproduced

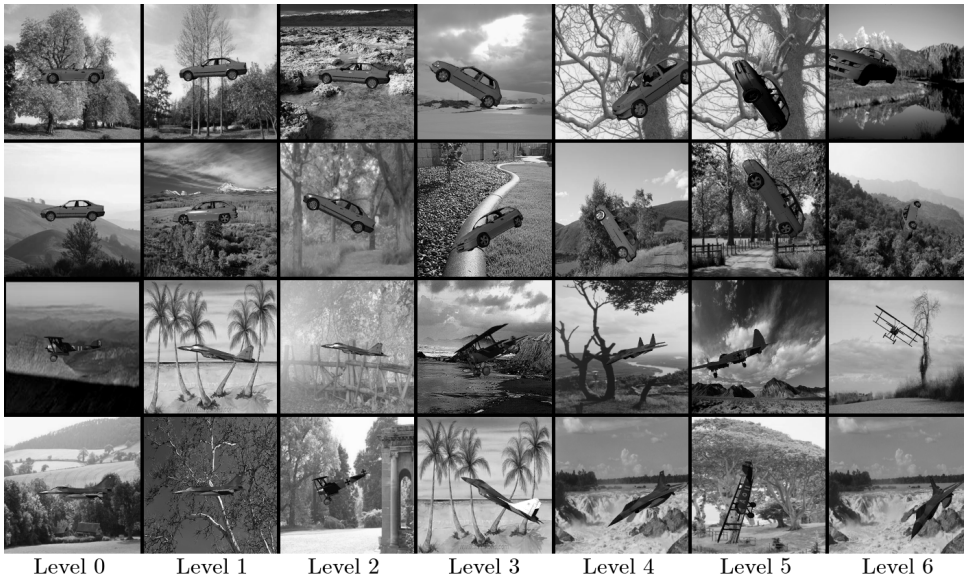


Figure 2.4: Sample images from each variation level of the "Cars vs. Planes object recognition task in Pinto et al. [19]

even if the S2 prototypes were replaced with random values. Figure 2.5 shows the performance of Glimpse, Thomure's HMAX-like model, on each variation level of the "Cars vs. Planes" task.

The vertical axis indicates classifier performance, as measured by the area under the classifier's receiver operating characteristic curve (commonly abbreviated ROC AUC, or just AUC). ROC AUC is a common performance metric for binary classifiers that will be detailed further in Chapter 3.

The solid blue line indicates classifier performance with features extracted by a dictionary of 4,075 prototypes imprinted from the training set. The dashed red line indicates the performance of a dictionary of 4,075 prototypes whose values are drawn from a uniform random distribution. It is remarkable that random features perform about as well or better than imprinted features for every variation level, suggesting that they have some underlying discriminative ability.

The solid gray line is a baseline where the features fed to the classifier are simply

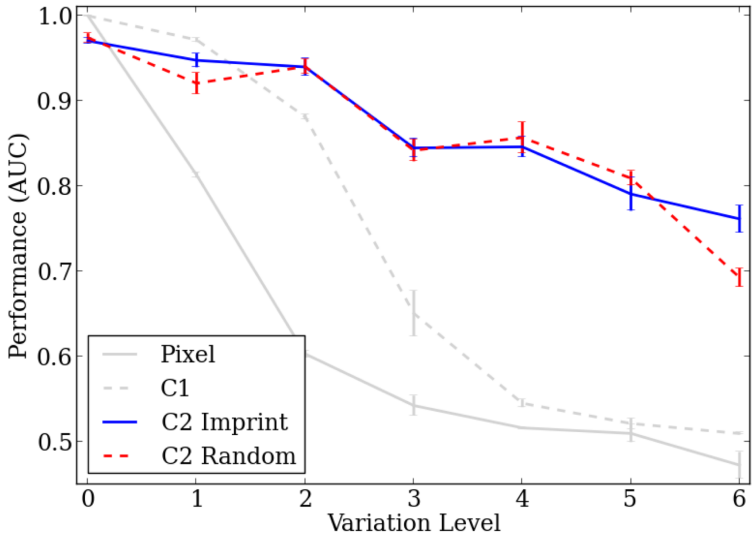


Figure 2.5: Performance of Glimpse on the Cars vs. Planes task of Pinto et al. [19]. Figure used with permission. Best viewed in color.

the concatenation of all of the pixel values in the input image. The dashed gray line is another baseline where the features are the concatenation of all C1 values extracted from the image.

Pinto et al. call this second baseline "V1-like", after the area of the visual cortex that is highly selective for low-level edge features. V1-like features such as C1 are actually quite successful on many tasks; one of the goals of Pinto et al.'s study was to create a benchmark that would highlight the difference between these simplistic models and more sophisticated ones like HMAX.

This is one reason why the performance of random prototypes is so surprising: on the surface, using random prototypes instead of learning shapes from natural images seems unsophisticated, yet it still creates a representation that is effective for a difficult visual task like invariant object recognition.

Despite these experimental results, the underlying mechanism behind the performance of random prototypes is still not well-understood.

2.4 A Word on Deep Learning

Deep learning, a field of machine learning concerned with with so-called deep neural networks (DNNs), is enjoying a surge of popularity [16], thanks to record-breaking object-recognition performance from a DNN implementation by Krizhevsky et al. in 2012 [15]. Unlike HMAX, deep neural networks are not intended to be models of the visual cortex. However, they have drawn attention from neuroscientists because their alternating multi-layer structure is similar to cortical models like HMAX, and their performance on classification tasks has been compared with of primate vision [4].

Deep learning is a rich field of contemporary research; however, this work does not attempt to compare the performance of HMAX or other cortical models to that of DNNs.

Chapter 3

Glimpse

For image processing and feature extraction, I used Glimpse [27, 26], a system developed by Mick Thomure for his PhD thesis. In addition to being a general system for exploring layer-wise image processing, Glimpse provides an implementation of an HMAX-like model very similar to that of Serre et al. [25]. Thomure showed that Glimpse's performance is similar to that of HMAX on object recognition tasks described by Serre et al. to test their implementation.

Thomure presents a comprehensive description of Glimpse's functionality in his work [27]. I provide a brief summary of each layer's computation in the sections below.

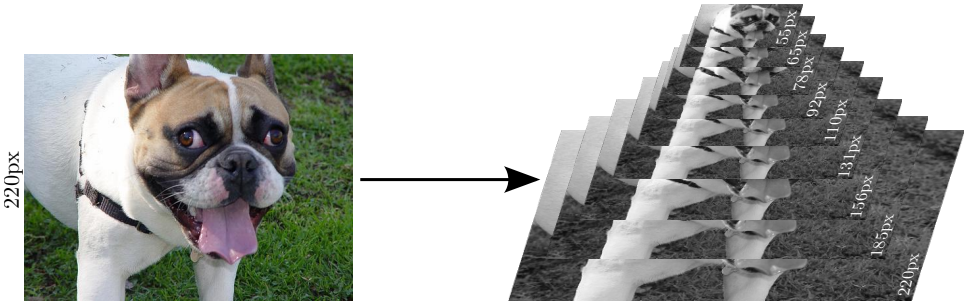
All model and parameter selections described in this chapter and used in my experiments follow those in Thomure's work. These selections are provided by default by Glimpse's application interface [26].

3.1 Image Preprocessing

All images are converted to grayscale, and downsampled to be 220 pixels on the short edge; the long edge is allowed to be as long as needed to preserve the aspect ratio of the original image.

In general, there are two ways to detect the same feature at different scales. One approach is to keep the input the same size and re-scale the filter; this is done by Riesenhuber et al. [20] and Serre et al. [25] in their version of HMAX. Another approach is to keep the filter the same size and re-scale the image. This is the approach taken by Mutch et al. [18] in their version, as well as by Glimpse.

The latter approach is called an *image pyramid* (or scale pyramid), and is a common technique in image preprocessing. Glimpse uses a pyramid of 9 scales. Starting with the downsampled image, each scale subsequent scale is downsampled again to be $2^{1/4}$ times smaller than previous scale. An example is given in Figure 3.1.



Original Color Image
Short Edge: 220px

Grayscale Image Pyramid
9 Scales (Ratio: $2^{1/4}$: 1)

Figure 3.1: Illustration of image preprocessing technique used by Glimpse.

3.2 S1 Layer

To compute the S1 layer's activity, the image is convolved at each scale with a set of Gabor filters. These are sinusoidal Gaussian kernels, each of which respond highly to edges of a particular orientation. For an orientation θ, phase ϕ, aspect ratio γ, kernel size w, wavelength $\lambda = \frac{w}{4}$, and scale $\sigma = \frac{\lambda}{2}$, the location u, v offset from the center of the Gabor filter $\mathbf{d}$ is defined as:

$$\begin{aligned} d(u,v) &= \exp\left(-\frac{u_0^2 + \gamma^2 v_0^2}{2\sigma^2}\right) \sin\left(\phi + \frac{2\pi u_0}{\lambda}\right) \\ \text{where: } u_0 &= u\cos\theta + v\sin\theta \\ v_0 &= -u\sin\theta + v\cos\theta \end{aligned} \tag{3.1}$$

Glimpse uses four Gabor filters, with $\theta = \left(\frac{\pi}{8}, \frac{3\pi}{8}, \frac{5\pi}{8}, \frac{y\pi}{8},\right)$, $\phi = 0$, $\gamma = 0.6$, and $w = 11$px. These are depicted in Figure 3.2 with $w = 200$ to show detail.

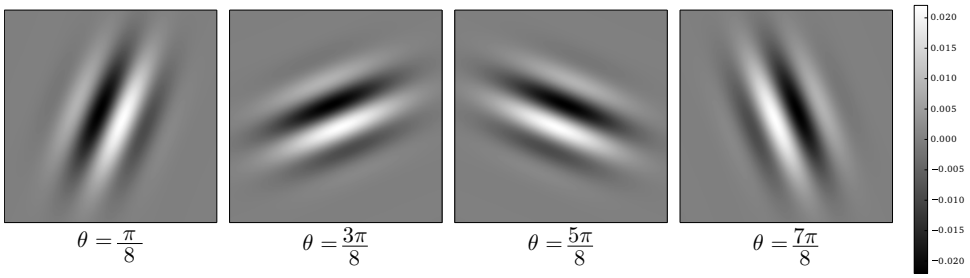


Figure 3.2: The set of Gabor filters used by Glimpse, shown with $w = 200px$. For all experiments, $w = 11$px.

During the convolution, each patch is normalized to have a Euclidean norm[1] of 1 to provide contrast invariance (where a low-contrast edge provides a similar response to a high-contrast edge), but this normalization is thresholded to prevent noise amplification in low-light image regions. The result is also rectified to provide phase invariance (where a light edge on a dark background gives the same response as and a dark edge on a light background).

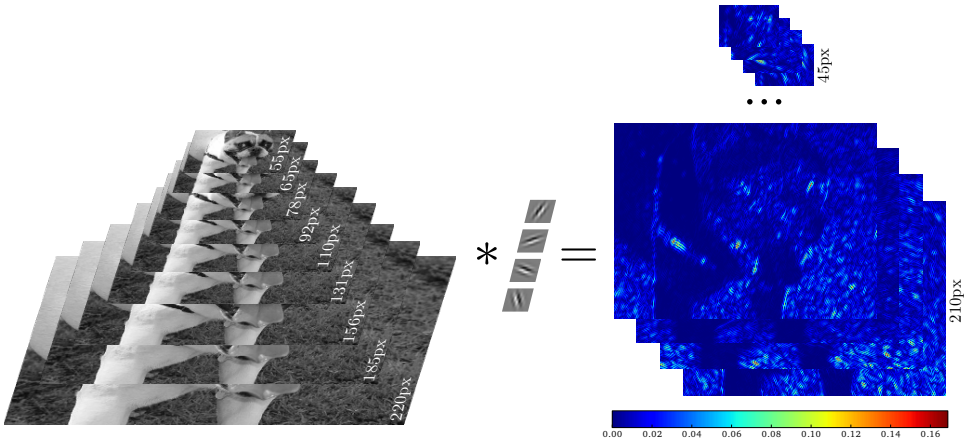
Thus, for each input patch $\mathbf{x}$ at all locations and scales, and each Gabor filter $\mathbf{d}$, the activity of the corresponding S1 unit is defined as:

$$\text{S1}(\mathbf{x}, \mathbf{d}) = \frac{|\mathbf{x} \cdot \mathbf{d}|}{\max(||\mathbf{x}||, \tau) \times ||\mathbf{d}||} \tag{3.2}$$

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

The resulting S1 layer's activity is a set of 9 feature maps (one for each scale) that each contain 4 *orientation bands*. An example of S1 activity is given below in Figure 3.3.

[1]The Euclidean norm $||\mathbf{x}|| = \sqrt{\sum_i x_i^2}$.



Input:
9 Scales (Ratio $2^{1/4}:1$)

Filters:
11 x 11 px
4 Orientations

Output:
9 Scales, 4 Orientations

Figure 3.3: Illustration of S1 activity in Glimpse. Input is an image pyramid of 9 scales. Output is the response to 4 Gabor filters at each location and scale. Best viewed in color.

3.3 C1 Layer

C1 activity is computed by max-pooling over S1 independently for each scale and orientation band, followed by downsampling by a constant factor N.

Thus for every Nth location in S1, denoted by $\mathbf{x}$, the corresponding C1 unit's activation is the maximum value from an 11×11 patch centered at that location:

$$\text{C1}(\mathbf{x}) = \max(\mathbf{x}) \tag{3.3}$$

Max-pooling has the effect of making the C1 activity invariant to small translations of S1. Downsampling speeds up computation in later steps by removing redundancies caused by overlapping patches with the same maximum. In my experiments I use the downsampling factor $N = 5$.

An example of C1 activity is given below in Figure 3.4. Downsampling and max-pooling make C1 features more difficult to interpret visually than S1 features.

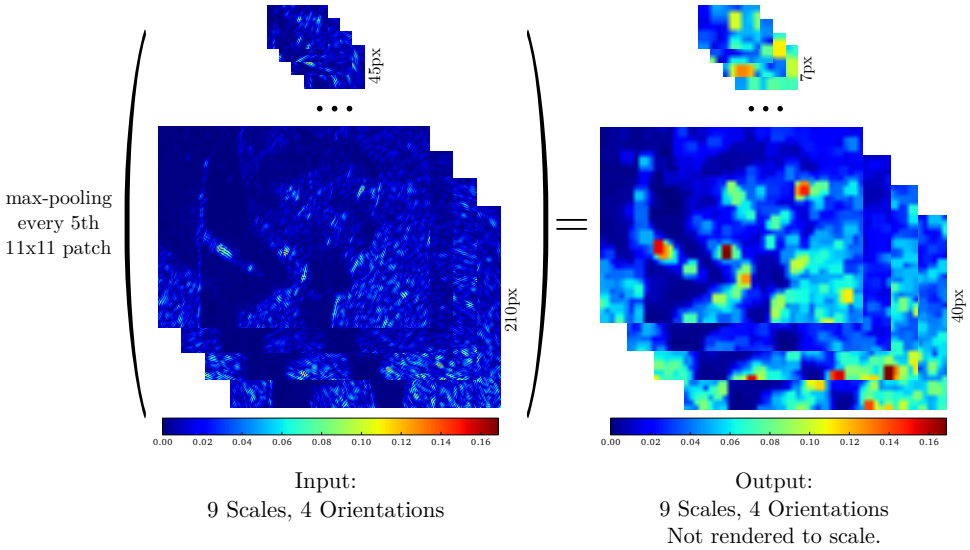


Figure 3.4: Illustration of C1 activity in Glimpse. Input is S1 feature maps. Output is the local max of every 5th 11×11 neighborhood. Best viewed in color.

3.4 S2 Layer

3.4.1 Imprinted Prototypes

Imprinted S2 prototypes are created by sampling random C1 patches from images in the training set (both positive and negative images). The patches are $7 \times 7 \times 4$ (a 7×7 patch across all four orientations) but can come from any of the 9 C1 scales.

An input image's S2 activity is computed by treating each prototype as a radial basis function, and then applying it to C1 patches at all locations and scales. For every C1 input patch $\mathbf{x}$ and prototype $\mathbf{p}$, the corresponding S2 unit's activation is defined as:

$$\mathrm{S2}(\mathbf{x}, \mathbf{p}) = \exp\left(-2\beta \left\|\mathbf{x} - \mathbf{p}\right\|^2\right) \tag{3.4}$$

If the input $\mathbf{x}$ and prototype $\mathbf{p}$ are identical, then the S2 activity is at its maximum, 1.0. The more dissimilar the input and prototype, the lower the activity. Higher

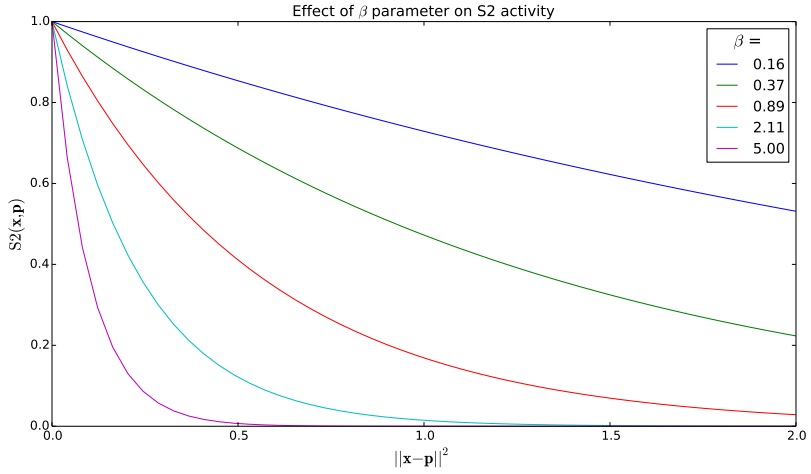


Figure 3.5: Illustration of the effect of β on the radial basis function (Equation 3.4) that determines S2 activity.

values of β result in sharper tuning to the prototype. An illustration of this property is given in Figure 3.5. In my experiments, β is set to 2.5.

The resulting S2 layer's activity is again a set of 9 feature maps, but each unit contains a separate band for the response to each of N prototypes. An example of S2 activity at the first scale for a single test image is given in Figure 3.6.

The first and third columns show each of 10 imprinted prototypes as their *receptive fields*. The red box indicates the set of pixels in the training image that provided input to the prototype.

The second and fourth columns show the S2 activity for the corresponding prototype, where the color indicates the strength of the similarity at that location. The scale reference at bottom indicates the scale at which each prototype was applied to the test image.

Note for instance that the responses to a dog's nose (column 3-4, row 4) are higher than the responses to the front of a bus (column 3-4, row 2). Such responses may provide useful information for classification, while others such as that to a uniform patch of sky (column 3-4, row 5) may not be as informative.

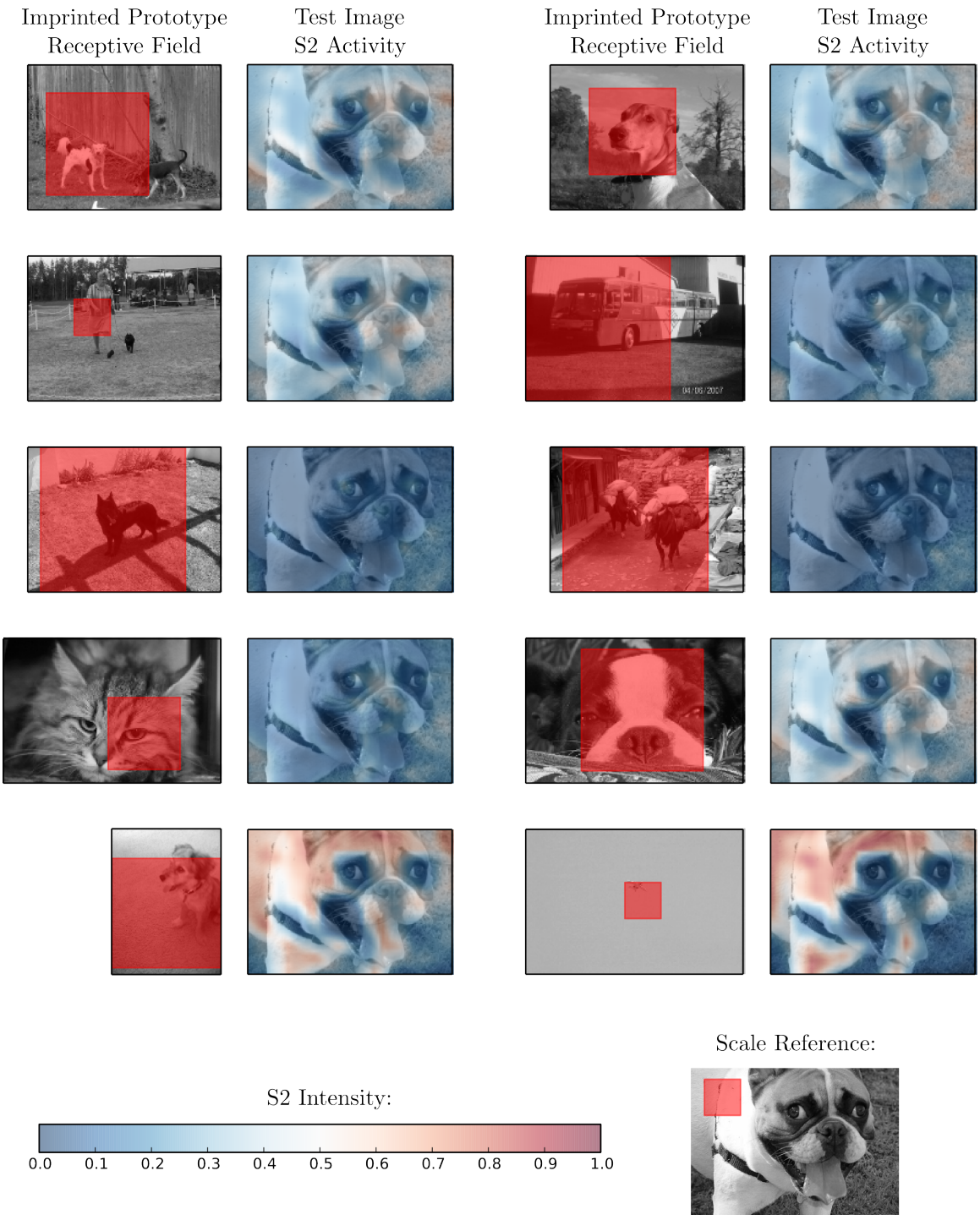


Figure 3.6: Illustration of S2 activity in Glimpse. The first and third columns show the receptive fields of 10 prototypes from the test image. The second and fourth columns show the S2 activity for each of those prototypes for the first scale in a single test image, where color indcates the strength of the similarity. A reference is shown at bottom indicating at what scale the prototype was applied. Figure best viewed in color.

Figure 3.7: Illustration of per-patch normalization of image pixels with a patch size of 7×7. Every location in the image is scaled to have roughly equal intensity.

3.4.2 Random Prototypes

Random S2 prototypes are created by drawing each value in the $7 \times 7 \times 4$ prototype from a uniform distribution in the range $[0, 1]$.

When using random prototypes, Glimpse computes the S2 layer's activity with a normalized version of Equation 3.4:

$$\hat{\text{S2}}(\mathbf{x}, \mathbf{p}) = \text{S2}\left(\frac{\mathbf{x}}{||\mathbf{x}||}, \frac{\mathbf{p}}{||\mathbf{p}||}\right) \tag{3.5}$$

Per-patch normalization makes the prototype's activation *gain invariant*. That is, two input patches that contain similar shapes but a different relative intensity should have similar activations.

An example of this kind of normalization at the pixel level is shown in Figure 3.7.

Thomure [27] found that using a normalized radial basis function (Equation 3.5) was necessary to get good classification performance from a dictionary of random features, but negatively impacted the performance of imprinted features. The mechanism behind this phenomenon is not fully understood.

3.5 C2 Layer

Finally, the C2 layer implements global max-pooling for each S2 prototype. The activation of each C2 unit is the maximum activity among the corresponding S2 units at all locations and scales.

Intuitively, a high C2 activity for a particular prototype indicates that the shape encoded by that prototype is present somewhere within the input image.

Since objects are composed of shapes, a set of C2 values from different shape prototypes should provide some information whether a particular object is likely to be present in that image.

This makes C2 features a good candidate for feeding into a classification algorithm.

3.6 Classifier

In this work, the C2 values for each image are used as feature vectors to train and test a classifier for the task of object recognition (described in Section 2.1).

For this task, a dataset of images with binary "positive" and "negative" labels (e.g., "contains a horse" vs. "contains no horses") is split into a *training set* and a *test set*.

During training, the classifier's job is to build a mapping from a C2 feature vector to a binary label. This mapping is called the classifier's *decision function*.

The classifier I use in my experiments is a linear Support Vector Machine (SVM) [6], specifically an implementation called `liblinear` [11]. The basic idea of training a linear SVM is to find a hyperplane (a plane generalized to a high-dimensional space) between the positive and negative training examples that maximizes the *margin* (distance) between the training examples and the hyperplane. As a preprocessing step, values for each feature (across examples) are scaled to have a mean of 0 and a standard deviation of 1, to prevent a single feature with a much higher variance from

having a magnified effect on the position of the hyperplane.

The hyperplane method assumes that the data is *linearly separable* in the feature space. This is a guarantee we hope our feature extractor has provided: that a positive image can be distinguished from a negative image using only the representation it has created. Thus, the performance of a linear classifier trained on the representation of a feature extractor is a good indicator of the performance of the feature extractor itself.

The hyperplane in a linear SVM is defined by a set $\mathbf{w}$ of coefficients (or *weights*) for each feature. Its equation (for some threshold τ) is the familiar linear form $w_1x_1 + w_2x_2 + \ldots + w_nx_n = \tau$, which I will write in vector notation as $\mathbf{w} \cdot \mathbf{x} = \tau$.

During testing, the label for a particular example depends on which side of the hyperplane its feature vector $\mathbf{x}$ falls on. If $\mathbf{w} \cdot \mathbf{x} > \tau$, then the label is positive. If $\mathbf{w} \cdot \mathbf{x} < \tau$, the label is negative. Thus, the classifier's decision function can be expressed as:

$$f(\mathbf{x}) = \operatorname{sgn}(\mathbf{w} \cdot \mathbf{x} - \tau) \tag{3.6}$$

I will refer to the dot product $\mathbf{w} \cdot \mathbf{x}$ itself as the *decision value* or *confidence value*.

The performance of a trained classifier is measured on the testing data, which consists only of examples not seen during training. A simple performance metric is accuracy, which is the percentage of examples given a correct label.

Another common metric for the performance of a binary classifier is the area under its Receiver Operating Characteristic curve, commonly abbreviated ROC AUC or just AUC. To create an ROC curve for a linear SVM, the threshold τ in the decision function is varied. When the threshold is decreased, more images will be classified as positive, leading to increased *recall* (the rate of true positives), but also increased

fall-out (the rate of false positives). When the threshold is increased, more images will be classified as negative, thus fall-out will decrease, but so will recall.

The ROC curve is a plot of recall vs. fall-out as the threshold is varied, and the AUC is the area under this curve. A classifier making random predictions will have both labels uniformly distributed amongst the range of decision values, so the recall and fallout are likely to be the same. The area under this curve will be 0.5. AUC values higher than 0.5 indicate that the classifier, on average, predicts more true positives than false positives. Unless otherwise noted, I will use AUC for all performance measurements.

Chapter 4

Random Prototype Performance

To investigate the relative performance of models with imprinted and random prototypes, I ran a series of object recognition experiments with Glimpse (as described in Chapter 3) on datasets constructed from the 2012 PASCAL Visual Object Classes (VOC) Challenge [8, 9] in which I varied the number and type of S2 prototypes used. Model performance was evaluated by using the extracted C2 features to train and test a linear SVM.

4.1 Methods

4.1.1 Dataset

The PASCAL Visual Object Classes (VOC) Challenge [8, 9] ran from 2005 until 2012. The set of tasks in the challenge grew over the years, but always contained two main tasks related to object recognition: classification and detection.

The classification task is essentially the object recognition task described in Section 2.1. Presented with an image, a system should answer the question "does this image contain any objects in class X?" by providing a decision value (for an SVM, this is the value $\mathbf{w} \cdot \mathbf{x}$ in Equation 3.6).

For the detection task, a system should answer the question "*where* are all instances of class X in this image?" by providing the location and size of a *bounding box* that encompasses each object, along with an associated confidence value.

In this work, I present experiments on the classification task only. According to Serre et al. [23], HMAX is designed to model the *ventral stream* (the "what" path-

Class	Images	Class	Images
Aeroplane	670	Dining Table	538
Bicycle	552	Dog	1286
Bird	765	Horse	482
Boat	508	Motorbike	526
Bottle	706	Person	4087
Bus	421	Potted Plant	527
Car	1161	Sheep	325
Cat	1080	Sofa	507
Chair	1119	Train	544
Cow	303	TV/Monitor	575

Table 4.1: Number of images in the PASCAL VOC 2012 dataset (out of 11,540) that contain at least one object from each corresponding class.

way) of the primate visual cortex, which is responsible for object recognition. This is distinct from the *dorsal stream* (the "where" pathway), responsible for localizing objects in space.

Since 2007, the challenge has included 20 object classes: Aeroplane, Bicycle, Bird, Boat, Bottle, Bus, Car, Cat, Chair, Cow, Dining Table, Dog, Horse, Motorbike, Person, Potted Plant, Sheep, Sofa, Train, and TV/Monitor. A sample image from each class is shown in Figure 4.1. Images are 500 pixels on the long side, but may be of any aspect ratio. Note that objects may be at any location or scale within the image, and may be occluded or surrounded by clutter, including objects from other categories.

Each year, the organizers released a labeled dataset where the bounding boxes and class labels for each object in every image the dataset are provided. In 2012, the classification and detection tasks included 11,540 images from this dataset, containing a total of 27,450 distinct objects. The classes are not distributed evenly among the images. Table 4.1 gives the number of images out of the 11,540 in the dataset that contain at least one object of a particular class.

Competitors were expected to train and validate their models on this dataset,

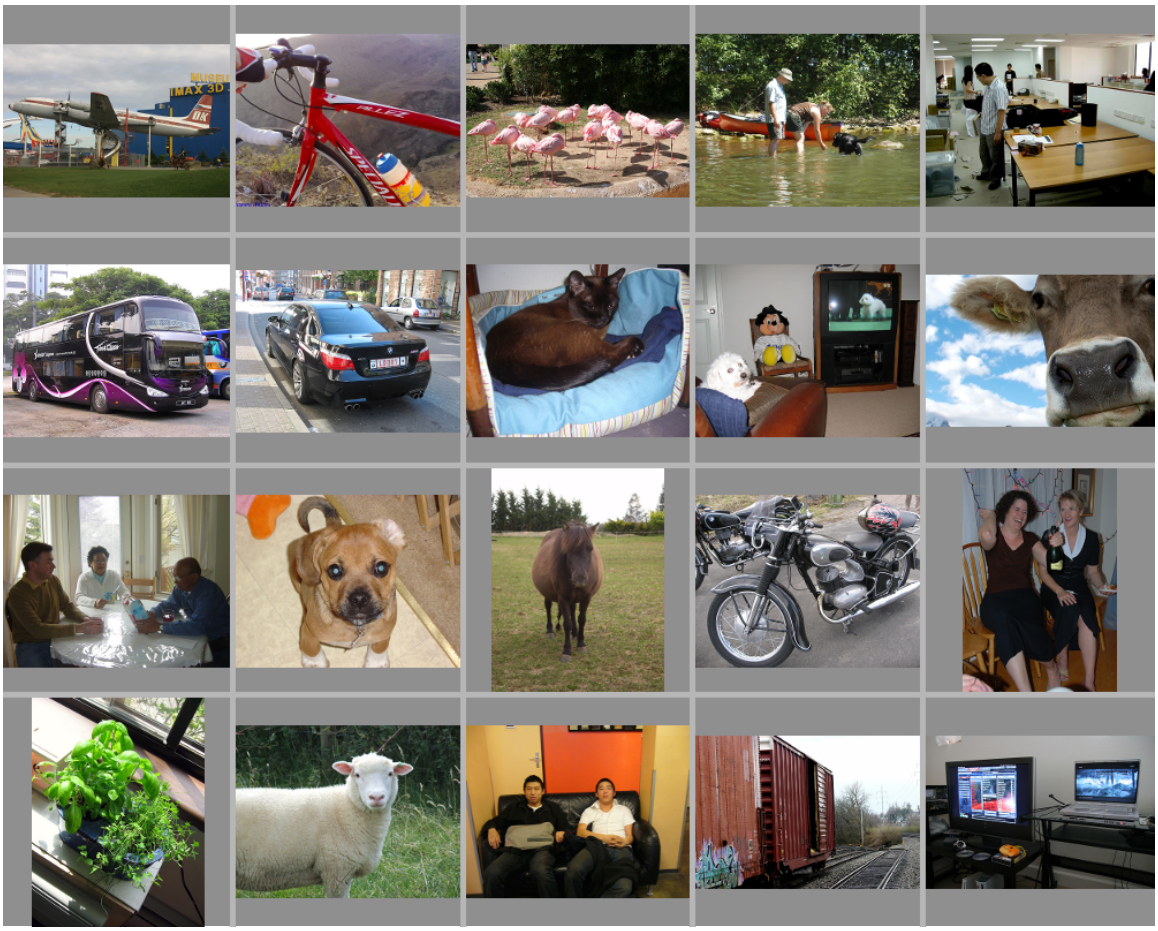


Figure 4.1: Samples images from the PASCAL VOC 2012 dataset that contain at least one object from a class in the classification task. From left to right and top to bottom: Aeroplane, Bicycle, Bird, Boat, Bottle, Bus, Car, Cat, Chair, Cow, Dining Table, Dog, Horse, Motorbike, Person, Potted Plant, Sheep, Sofa, Train, and TV/Monitor.

whereas formal evaluation was conducted on an unlabeled test set. The labels for this test set have never been publicly released, therefore my experiments present results on various training/testing splits of the of the labeled set only.

For my experiments, I organized the classification dataset into "positive" and "negative" sets for each of the 20 object classes, where an image was in the "positive" set if it contained at least one object of the image class in question. Since there are 20 classes, and most images contain objects from only a few of them, the size of each negative set for each class was much larger than that of the positive set.

To control the ratio of positive to negative examples for each class, I created a *balanced* dataset for that class by taking all of the positive images for that class, and then a random draw of equal size from the negative images with respect to that class. Each class received a different draw of negative images. These balanced datasets did not change, and were used for all experiments on the PASCAL VOC 2012 data presented in this work. Variance between experiments is thus limited to the choice of a training and testing split and the choice of prototypes.

4.1.2 Performance Evaluation

To evaluate the performance of Glimpse using imprinted and random prototypes, I ran 8 trials of the following experiment for each class, type of S2 prototype (imprinted or random), and number of S2 prototypes N in the set $\{10, 50, 100, 500, 1000, 4075\}$.

First, I randomly split the class's balanced dataset into training and test sets of equal size. Then, if using imprinted prototypes, I created prototypes by randomly extracting N patches of C1 activity from images in the training set. If using random prototypes, I drew each component of N prototypes independently from a uniform random distribution in the range $[0, 1]$.

I then used these prototypes to compute a C2 feature vector for each image in the dataset. The labels and C2 feature vectors of images in the training set were used to train a linear SVM.

I used the trained SVM to get the decision values for the C2 feature vectors of images in the test set, and used these along with their labels to get the AUC score for the classifier.

Pseudocode for this procedure is given in Figure 4.2.

Algorithm Evaluating Prototype Performance

```
1: procedure EVALUATE(dataset, s2type)
2:     for N ← [10, 50, 100, 500, 1000, 4075] do
3:         for each of 8 trials do
4:             Randomly split dataset 50/50 into trainSet and testSet.
5:             if s2type = imprint then
6:                 Randomly imprint N prototypes from images in trainSet.
7:             else if s2type = random then
8:                 Randomly draw components of N prototypes from Uniform(0,1).
9:             end if
10:            Use Glimpse with prototypes to extract C2 features of
               images in dataSet.
11:            Train a linear SVM on C2 features and labels of trainSet.
12:            Compute decision values from the C2 activity of testSet.
13:            Output ROC AUC score based on decision values and labels of testSet.
14:        end for
15:    end for
16: end procedure
```

Figure 4.2: Pseudocode for the experimental setup used to evaluate Glimpse with imprinted and random prototypes.

4.1.3 Baseline Method

The baseline method used by Thomure [27] (described in Section 2.3) was the concatenation of all of the C1 features (edge responses) in the input image. This is similar to Pinto et al.'s "V1-like" baseline [19], which also used low-level edge features. These features do well on a variety of object recognition tasks, but fail at the invariant object recognition tasks described by Pinto et al., where HMAX succeeds. The hypothesis behind HMAX's performance is that it can extract high-level shapes rather than just low-level edges. It is worth attempting to reproduce this baseline in my experiments.

However, this baseline is not possible with the data from PASCAL VOC 2012, because the images are of different sizes, and concatenating the activity of every C1 unit would result in a feature vector of a different dimensionality for each image. To use an SVM, each feature vector must be of the same dimensionality.

Instead, I approximate it with the following histogram-based approach: at each C1 scale, divide the space into a 4×4 grid as evenly as possible. For each of the 16 grid locations at each scale and orientation, compute a histogram of C1 values with 50 bins in the range $(0, 0.35)$. I then use a feature vector that is the concatenation of the counts in each bin. The dimensionality of this feature vector is 50 bins $\times$ 16 locations $\times$ 9 scales $\times$ 4 orientations = 28,800, regardless of the size of the input image. I then use these feature vectors to train and test an SVM as described above, with a different random training and testing split for each of 8 trials.

4.2 Results

A summary of performance results on the VOC 2012 classes is given in Figure 4.3. This figure shows the classifier performance of C2 feature vectors created by dictionaries of 4,075 imprinted and random prototypes, as well as the performance of the

histogram-based C1 baseline method described above. The object classes are sorted by baseline performance, under the intuition that a classifier's performance given a "naive" feature extractor is an indication of the difficulty of the dataset for that class.

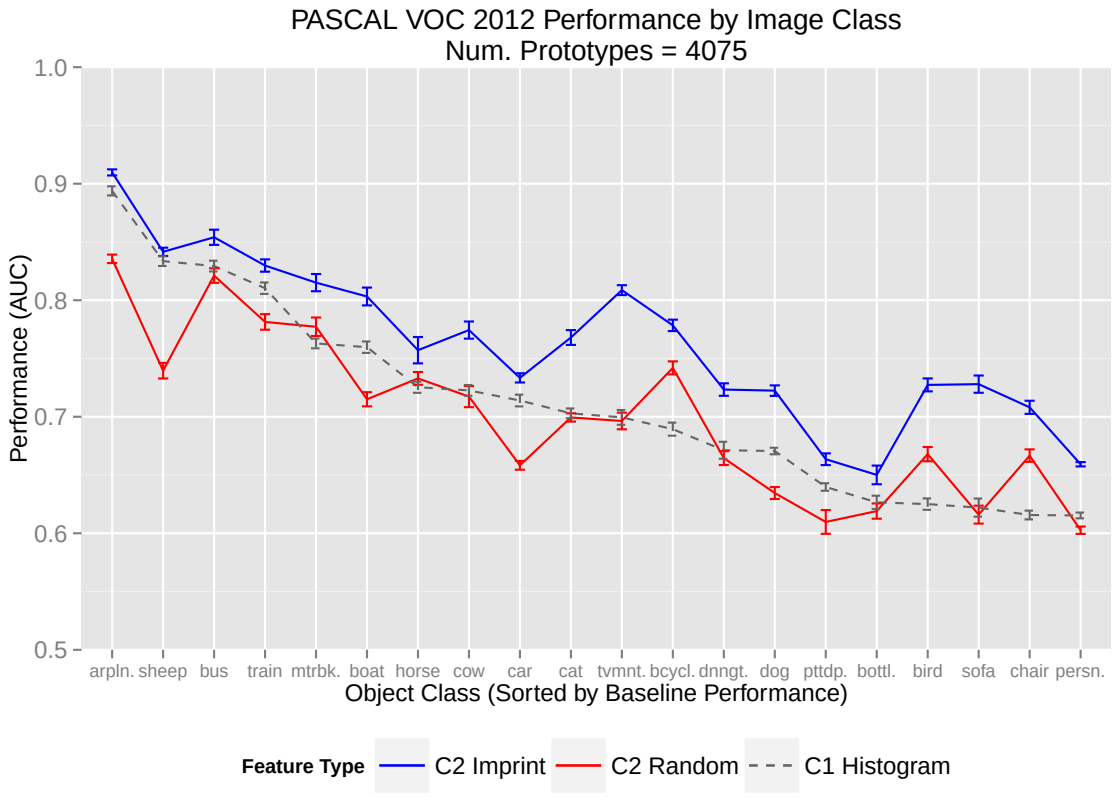


Figure 4.3: Summary of performance results for all 20 PASCAL VOC 2012 object classes. Classifier performance is given for a baseline feature vector consisting of C1 histograms (dashed gray line), and for C2 features extracted by 4,075 imprinted prototypes (solid blue line) and 4,075 random prototypes (solid red line). Error bars indicate standard error over 8 trials.

The classifier performs well above chance (AUC 0.5) for every type of feature. Features using imprinted prototypes consistently perform above the baseline, consistent with prior results by Thomure [27], and Pinto et al. [19].

Random prototypes perform at baseline in 10 out of the 20 classes, above baseline in 3 classes, and below baseline in 7 classes. This is in contrast to Thomure's observations on Pinto et al.'s "Cars vs. Planes" task (see Figure 2.5, page 12), in

which random prototypes consistently performed above baseline, and about as well as imprinted prototypes.

Thomure stated that datasets for which HMAX performs close to baseline "are of little use in the study of invariant object recognition" [27], because the success of edge features indicates that contextual cues about texture and background could be providing a lot of information about object identity. Consider the "cow" object class: a naive way to separate positive from negative images would be to look for grass rather than for cows; if most of the image consists of grass, then a concatenated (or histogram-binned) C1 map will have a characteristic pattern which the classifier can use.

However, even if contextual cues are dominating the classification task on PASCAL VOC 2012, I believe it is still remarkable that features derived from random prototypes are able to perform well above chance in every category.

It is also interesting that the performance of the random prototypes appears to be correlated with that of the imprinted prototypes, relative to the baseline: many of the classes below the baseline for random prototypes are closer to the baseline for imprinted prototypes (such as "car"), and every class above the baseline for random prototypes is further above the baseline for imprinted prototypes (bicycle, bird, and chair).

This suggests that the upper layers of the model have characteristics that the lower layers lack regardless of the choice of prototype, even for tasks on which a "naive" model does well. This idea has also been proposed by Saxe et al. [22], who find that convolutional pooling architectures can be inherently selective and invariant, even when using random filters.

A full chart of results for the performance of imprinted and random prototypes for all classes and dictionary sizes is given in Figure 4.4. This figure depicts performance

as a function of dictionary size (the number of prototypes), and includes the baseline performance as a horizontal line.

Curiously, the relationship between performance and dictionary size is not always increasing over the range of sizes selected: almost every class has a dip in the performance around 100-500 prototypes, for both imprinted and random prototypes. For some classes (e.g., "car" and "person"), the performance of 4,075 random prototypes is actually worse than the performance of 10 random prototypes.

I suspect that these phenomena are due to *overfitting*, where the classifier begins to model noisy variations in the feature vectors as the number of prototypes increases. When the number of prototypes is sufficiently low, there is not enough information to overfit. When the number of prototypes is sufficiently high, the SVM can overcome overfitting by finding the most relevant features. This suggests the idea of using only the subset of features determined to be relevant by the SVM. This is the basis of *feature selection*, and is discussed further in Chapter 5.

4.3 Discussion

4.3.1 Baseline Performance

An alternative explanation for the performance of HMAX being so close to baseline on the PASCAL VOC 2012 data is that my baseline technique is somehow more sophisticated than the baseline used by Thomure (the raw concatenation of all C1 features).

To test this hypothesis, I ran all methods described above on the same "Cars vs. Planes" dataset that was used to create Figure 2.5, and also ran Thomure's original baseline method.

The results are shown in Figure 4.5. I am able to reproduce Thomure's results

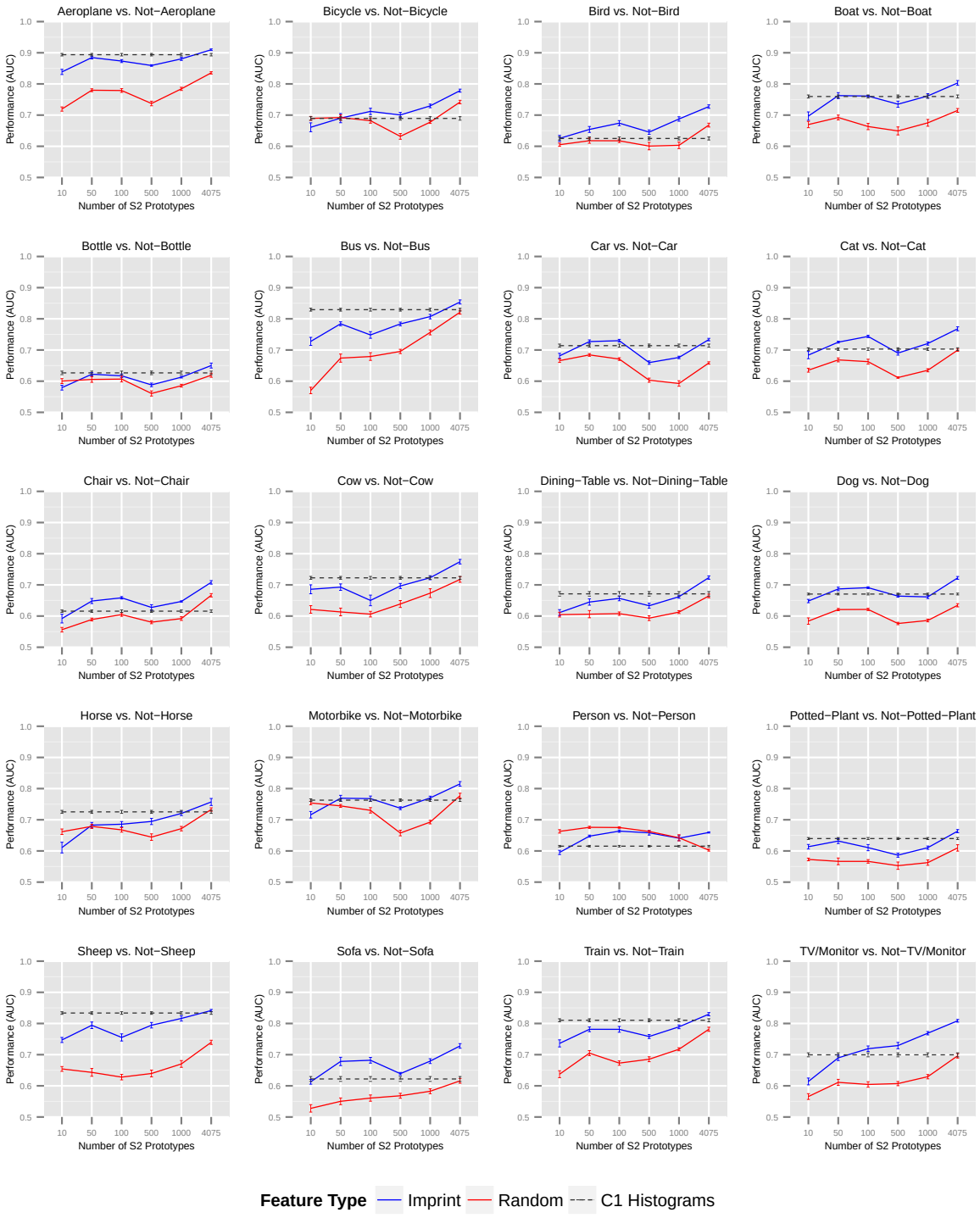


Figure 4.4: Performance results for all 20 VOC 2012 object classes. Measurements are given for classifier performance using dictionaries with various numbers of imprinted (blue solid), and random (red solid) prototypes. The baseline (dashed gray) indicates classifier performance on histograms of C1 features. Number of prototypes is shown on a categorical axis. Error bars indicate standard error over 8 trials. Best viewed in color.

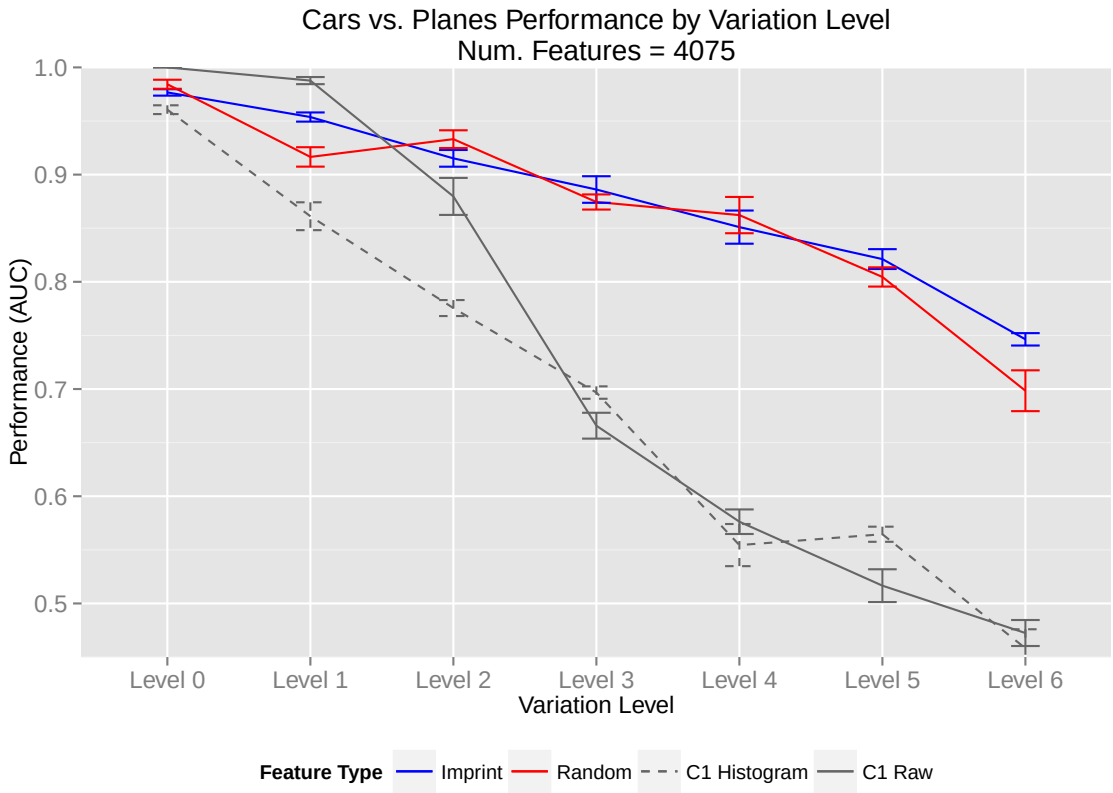


Figure 4.5: Performance of raw C1 features (solid gray), C1 histogram features (dashed gray), 4,075 imprinted C2 features (solid blue), and 4,075 random C2 features (solid red) on the "Cars vs. Planes" task of Pinto, et al. [19]. Error bars indicate standard error over 5 trials.

for imprint, random, and raw C1 features. My histogram-based C1 features perform similarly to raw C1 features at the higher variation levels, and a little worse at the lower levels. I conclude that my representation is not more sophisticated than raw C1.

Thus, the invariant recognition properties of HMAX may not be necessary for solving the classification task as given in PASCAL VOC 2012, given that C1 histograms are incredibly cheap to compute compared to C2 features. Nevertheless, we can still use results from this task to study the properties of imprinted and random prototypes for large, diverse datasets composed of natural scenes.

4.3.2 Shape Selectivity

One such property is shape selectivity (or shape-tuning), described as a key component of HMAX by Serre et al. [23]. To explore this claim, Thomure [27] investigated the selectivity of imprinted and random prototypes that are highly discriminative in isolation; that is, they can be used to extract one-dimensional feature vectors that are nonetheless useful for separating classes in an object recognition task. He found that such highly discriminative prototypes do exist, regardless of whether they were created randomly or through imprinting.

Thomure also found that while discriminative imprinted prototypes are selective for particular natural shapes (such as faces), discriminative random prototypes appear to have no shape selectivity whatsoever. This is a surprising result, as it contradicts the hypothesis of Serre et al. that shape-tuning is necessary for creating a rich, discriminative representation useful for object recognition.

However, Thomure's result was only run on the small datasets of synthetic images from Pinto et al. [19], where it may have been difficult to observe a diverse set of natural shapes. I attempt to reproduce his result on each class of the VOC 2012 dataset.

To find highly discriminative prototypes for each object class, I first create a training and testing split for that class, then create a set of 10,000 prototypes either by imprinting from the training set or by sampling components from a uniform random distribution.

For each of these 10,000 prototypes, I compute the corresponding C2 value for every image in the dataset. Then I use that single C2 value to train and test a linear SVM, retrieving its AUC score. I then rank the prototypes by their AUC score.

The results of this ranking process for each class are shown in Figure 4.6. Compar-

ing these results with Figure 4.4, I reproduce Thomure's finding that single prototypes (whether imprinted or random) can be surprisingly discriminative. The best single prototypes are often as discriminative as a set of 1,000 arbitrarily chosen imprinted prototypes, or 1,000 randomly generated prototypes. This may indicate that only a small subset of a large dictionary is actually useful for object recognition on this task.

To observe the selectivity of the most discriminative prototypes for each class (i.e., the prototypes at Rank 1 in Figure 4.6), I follow Thomure's technique of visualizing the input patches in the test set that generate the highest S2 activity for that prototype.

The top 10 responses for the best performing imprinted and random prototypes for each class (from a single trial set of 10,000 per class and feature type) are shown in Figure 4.7.

The "Weight" column indicates the weight assigned to that prototype by the SVM. A positive weight means that if the C2 value for that prototype is above the SVM's threshold, it will predict that the object in question is present. A negative weight means that if the C2 value is above threshold, it will predict that the object is *not* present.

The borders around each patch then show whether or not that input image was classified correctly (green) or incorrectly (red) by the SVM.

Imprinted Prototype Selectivity

These results support Thomure's observation that imprinted prototypes are highly selective to natural shapes, and the best imprinted prototype for a class often has a positive weight: every patch that responded to the best imprinted prototype for "person" contains a face, even the misclassified example (which is a dog's face). Almost every response to the top imprinted prototype for "motorbike" contains a motorcycle

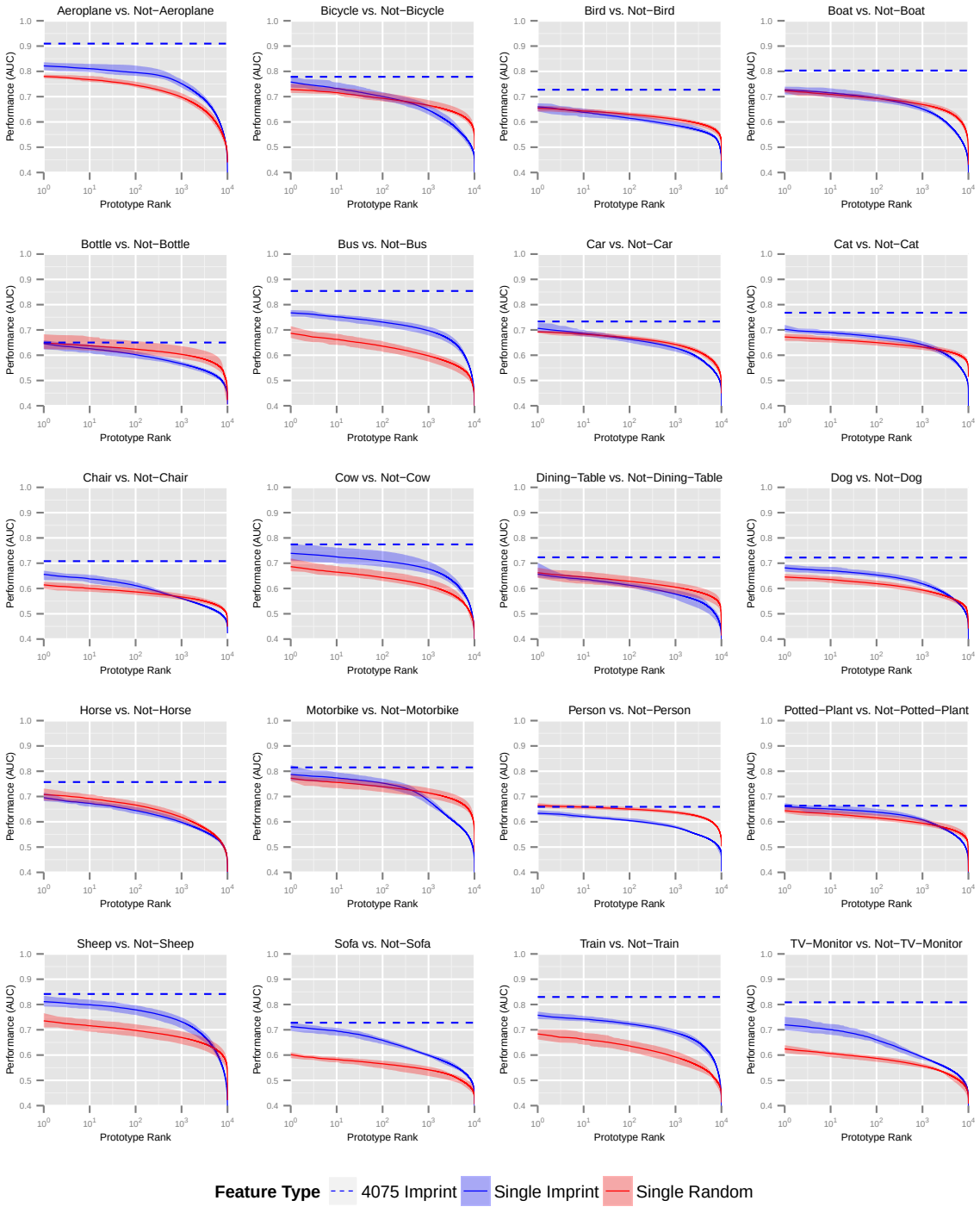


Figure 4.6: Performance of 10,000 individual imprinted and random prototypes on each of the 20 object classes. The x-axis indicates the prototype rank by its AUC value, and the y-axis indicates classification performance when using only the prototype of that rank. The solid line indicates the mean performance for that rank over 8 trials, while the shaded area indicates the range of values observed for that rank. Mean performance for a dictionary of 4,075 imprinted prototypes (dashed blue line) is added for comparison. Figure best viewed in color.

wheel. For "car", the responses contain car wheels. For "tv/monitor", we see the edges of a monitor, and so on.

In fact, only the best imprinted prototypes for "aeroplane" and "bird" are assigned a negative weight. On the surface, the responses to these prototypes lack the specificity to a single object observed with the positively weighted imprinted prototypes, but there is definitely a similarity in the response to the prototype for "aeroplane": almost all contain flowing, natural textures like hair, fur, fabric, and grass, which are highly unlikely to co-occur with aeroplanes.

Random Prototype Selectivity

On the other hand, I find evidence to refute Thomure's claim that discriminative random prototypes completely lack shape specificity. The specificity may be more mysterious because random prototypes are less likely to repeatedly match shapes from the same object (such as motorcycle wheels), but it is still visible.

For instance, the best performing random prototype for "cow" is weighted negatively, and the responses all contain cluttered scenes with lots of edges. Almost all of the responses have a similar edge configuration: the lower half is a diagonal edge extending up from the lower left, while the upper half contains a flat edge, or one angled slightly towards the lower right. Indeed, this is a random configuration of edges, but it is unlikely to co-occur with a cow standing in a grassy field.

Additionally, almost all of the top responses for the best random prototype for "sofa" contain a diagonal edge running from the top left to the lower right, and also perhaps a diagonal edge extending from the lower left to about the middle of the image. This prototype is weighted negatively, so perhaps this configuration is unlikely to co-occur with sofas in this dataset.

One of the more striking examples is the best random prototype for "train": every

single top response contains a set of parallel lines extending towards the upper right of the patch. This prototype is weighted positively, and this makes sense, because parallel lines are likely to occur around train tracks. The top random prototype for "bicycle" also appears fairly specific: three of the four top responses contain bicycle wheels in exactly the same position and scale.

All of that being said, this is still a purely qualitative, visual inspection of a small sample of data, and is thus highly variable and subject to human cognitive biases. I believe that a quantitative exploration of shape specificity on large datasets is necessary future work.

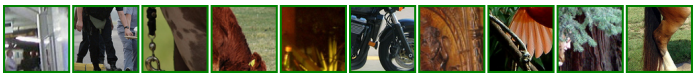
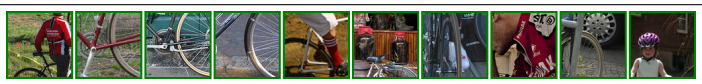
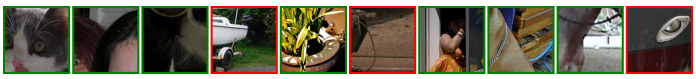
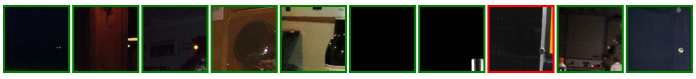
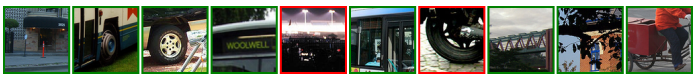
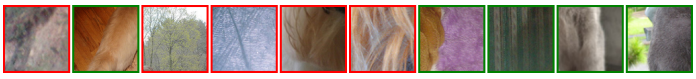
Class	S2 Type	Weight	Top Responses
Aeroplane	Imprint	Negative	
	Random	Negative	
Bicycle	Imprint	Positive	
	Random	Positive	
Bird	Imprint	Negative	
	Random	Negative	
Boat	Imprint	Positive	
	Random	Negative	
Bottle	Imprint	Positive	
	Random	Positive	
Bus	Imprint	Positive	
	Random	Positive	
Car	Imprint	Positive	
	Random	Positive	
Cat	Imprint	Positive	
	Random	Negative	

Figure 4.7: Characterization of the most discriminative single prototypes out of a set of 10,000 for each object class and type of prototype. "Weight" indicates the weight assigned to that prototype by the SVM. "Top Responses" shows the top 10 patches from the test set with the highest S2 activation for that prototype, in order of response. Each patch in a row is from a different image. A green border indicates a correct prediction for that image; a red border indicates an incorrect prediction. Best viewed in color. Continues on next page.

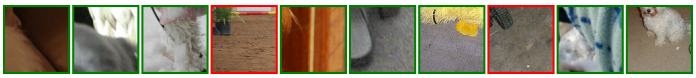
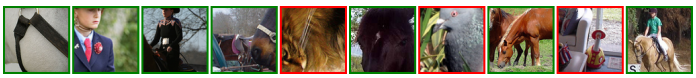
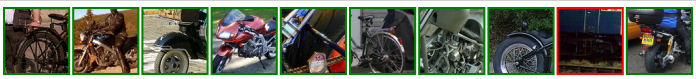
Class	S2 Type	Weight	Top Responses
Chair	Imprint	Positive	
	Random	Positive	
Cow	Imprint	Positive	
	Random	Negative	
Dining Table	Imprint	Positive	
	Random	Positive	
Dog	Imprint	Positive	
	Random	Negative	
Horse	Imprint	Positive	
	Random	Positive	
Motorbike	Imprint	Positive	
	Random	Positive	
Person	Imprint	Positive	
	Random	Positive	
Potted Plant	Imprint	Positive	
	Random	Positive	

Figure 4.7: Continued from previous page. Continues on next page.

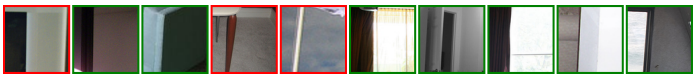
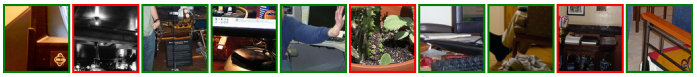
Class	S2 Type	Weight	Top Responses
Sheep	Imprint	Positive	
	Random	Negative	
Sofa	Imprint	Positive	
	Random	Negative	
Train	Imprint	Positive	
	Random	Positive	
TV / Monitor	Imprint	Positive	
	Random	Positive	

Figure 4.7: Continued from previous page.

Chapter 5

Feature Selection

In this chapter I present results from experiments with *feature selection*. Feature selection, as used in this thesis, is a method for learning a dictionary of prototypes using *task feedback*, where the choice of prototypes is informed by their performance on the classification task.

Feature selection can be contrasted with task feedback methods that would modify the prototype's values themselves, such as learning by gradient descent, which is a common technique for training large multi-layer neural networks [16]. Instead, learning by feature selection begins by generating a large set of candidate prototypes, then ranking each prototype by its "quality".

In my experiments, quality is determined by the magnitude of the weight assigned to the corresponding C2 feature during the SVM's training phase (see Section 3.6). Features with lower weights are ones that are likely to have similar values in both positive and negative images, and thus have less impact on the outcome of the decision function (Equation 3.6).

Thomure [27] found that dictionaries formed by feature selection performed much better than their unselected counterparts. In other words, a selected dictionary can be much smaller than an unselected dictionary with the same performance. This helps offset some of the computational burden of generating a large candidate set, and makes dictionaries learned by feature selection more practical for systems where prediction on a novel image must be quick, but expensive training can be done ahead of time.

5.1 Methods

I repeat Thomure's feature selection experiments [27] with imprinted prototypes, and I also conduct feature selection experiments with random prototypes.

My experimental procedure begins similarly to the one described in Section 4.1.2. For each of 8 trials for each class, I create a random training set and test set of equal size. I then generate a candidate set of 10,000 prototypes, either by imprinting from the test set or by drawing uniform random values. I use these prototypes to extract C2 features from the data set, and then train an SVM on the training set.

I retrieve the feature weights from the trained SVM, and then rank the features in descending order of their magnitude (absolute value). Then, for each of the top $N = \{10, 50, 100, 500, 1000, 4075\}$ features in the ranked candidate set, I filter the C2 feature vectors to contain only those top N features.

I then train a new SVM using the labels and filtered feature vectors of the training set, and use the trained SVM to predict decision values for the filtered feature vectors of the test set. The labels and decision values of the test set are used to compute the classifier's AUC score.

Pseudocode for this procedure is given in Figure 5.1.

5.2 Results

Detailed performance measurements of feature selection for all 20 VOC 2012 classes is given are Figure 5.2. This figure includes the data on unselected imprinted and random prototypes from Figure 4.4. I also add a point for performance on the full 10,000 prototype candidate set without any feature selection.

Curiously, I observe that feature selection generally only helps performance for dictionary sizes in the range of 500 to 1,000 prototypes, where the performance of

Algorithm Evaluating Feature Selection

```
1: procedure FEATURE SELECTION(dataset, s2type)
2:     for each of 8 trials do
3:         Randomly split dataset 50/50 into trainSet and testSet.
4:         if s2type = imprint then
5:             Randomly imprint 10,000 prototypes from images in trainSet.
6:         else if s2type = random then
7:             Randomly draw components of 10,000 prototypes from Uniform(0,1).
8:         end if
9:         Use Glimpse with prototypes to extract C2 features of images in dataSet.
10:        Train a linear SVM on C2 features and labels of trainSet.
11:        Retrieve the feature weights from the SVM.
12:        Sort the C2 features by the magnitude of their weights,
           in descending order.
13:        for N ← [10, 50, 100, 500, 1000, 4075] do
14:            Filter the C2 features to contain only the top N of the sorted features.
15:            Train a linear SVM on the filtered features and labels of trainSet.
16:            Compute decision values from the filtered features of testSet.
17:            Output ROC AUC score based on decision values and labels of testSet.
18:        end for
19:    end for
20: end procedure
```

Figure 5.1: Pseudocode for the procedure used to evaluate the performance of dictionaries created through feature selection.

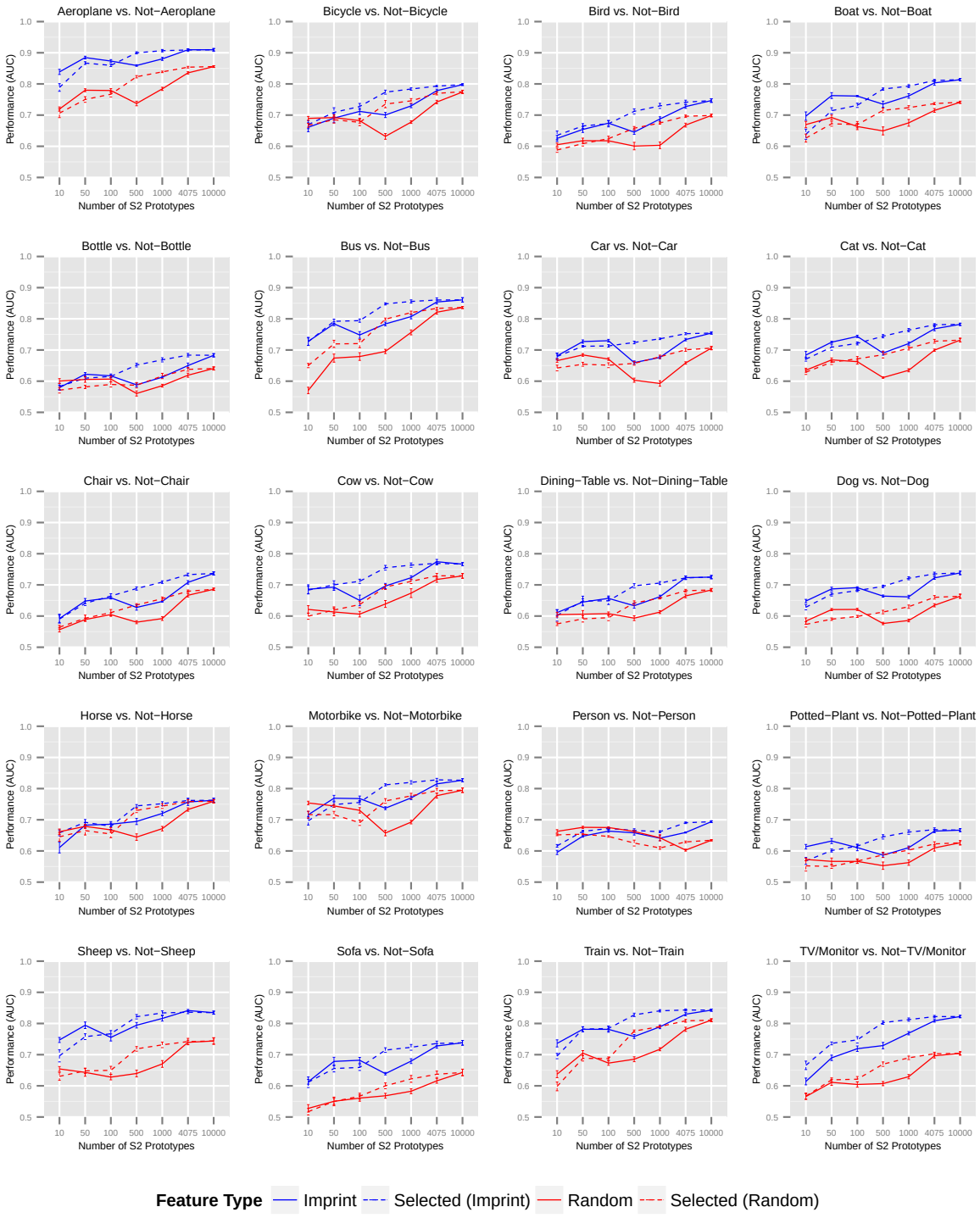


Figure 5.2: Performance results for all 20 VOC 2012 object classes. Measurements are given for classifier performance using dictionaries formed by various numbers of selected (blue dashed) and unselected (blue solid) imprinted prototypes, as well as selected (red dashed) and unselected (red solid) random prototypes. Number of prototypes is shown on a categorical axis. Error bars indicate standard error over 8 trials. Best viewed in color.

unselected dictionaries has a characteristic dip. At 100 prototypes and below, selected dictionaries perform the same or worse than their unselected counterparts. At 4,075 prototypes, feature selection only marginally improves performance. Of course, at 10,000 prototypes, there is no difference between a selected and unselected dictionary, so performance is the same. These observations hold true for both imprinted and random prototypes.

5.3 Discussion

Thomure [27] observed that dictionaries formed through feature selection tended to reach a "saturation point" which he posited could be an indicator of a particular dataset's complexity; the more broad the variation present in a particular object class, the larger a dictionary must be to capture features relevant for classification.

I find that very few dictionaries formed through feature selection for the PASCAL VOC 2012 data reach such a saturation point before hitting the limit of the performance of the entire candidate set. On the contrary, for many classes, there is still an upward trend in performance.

One potential explanation is that the candidate set is simply not big enough; given the huge diversity in the PASCAL VOC 2012 data, this is certainly a possibility. Redoing this study with a larger candidate set may prove fruitful for increasing performance.

Another possible explanation is that Thomure's choice of classification algorithm was more appropriate for feature selection. For consistency and clarity, I use a linear SVM everywhere in this work, including for feature selection.

Thomure's experiments on feature selection used another linear classification algorithm called *logistic regression*, which uses the decision function:

$$f(\mathbf{x}) = \operatorname{sgn}\left(\frac{1}{1 + e^{\tau - \mathbf{w}\cdot\mathbf{x}}} - \frac{1}{2}\right). \tag{5.1}$$

Here, $\mathbf{x}$ is a feature vector, $\mathbf{w}$ is the weight vector, and τ is the threshold. The weight vector $\mathbf{w}$ can be used to rank features just as with the weights assigned by a linear SVM, and in general, linear SVMs and logistic regression classifiers generally produce equivalent performance on the same set of feature vectors, since they both assume that the data is linearly separable.

Thomure used a variant called *sparse* logistic regression that attempts to set as many values of the weight vector to zero as possible during the training phase. I hypothesized that this may give sparse logistic regression unique properties when used for feature selection, which depends on the values of these weights.

To test this idea, I re-ran the algorithm from Figure 5.1 on the "dog" dataset, using sparse logistic regression instead of an SVM for both training steps. For a consistent comparison, I skipped the creation of a new training and testing split and candidate set, instead using the same 16 splits and candidate sets created for the SVM.

Figure 5.3 gives the results of this comparison. Overall, linear SVMs and sparse logistic regression (SLR) classifiers display very similar performance when used for feature selection. However, the SLR classifier reaches the "saturation point" described by Thomure at 500 prototypes, while the SVM continues to improve up to the limit set by the candidate set.

My conclusion is this saturation point is due to the sparsity of the weight vector. Recall that the sparse logistic regression learning algorithm attempts to set as many features as possible to have zero weight; if we select more prototypes for the dictionary

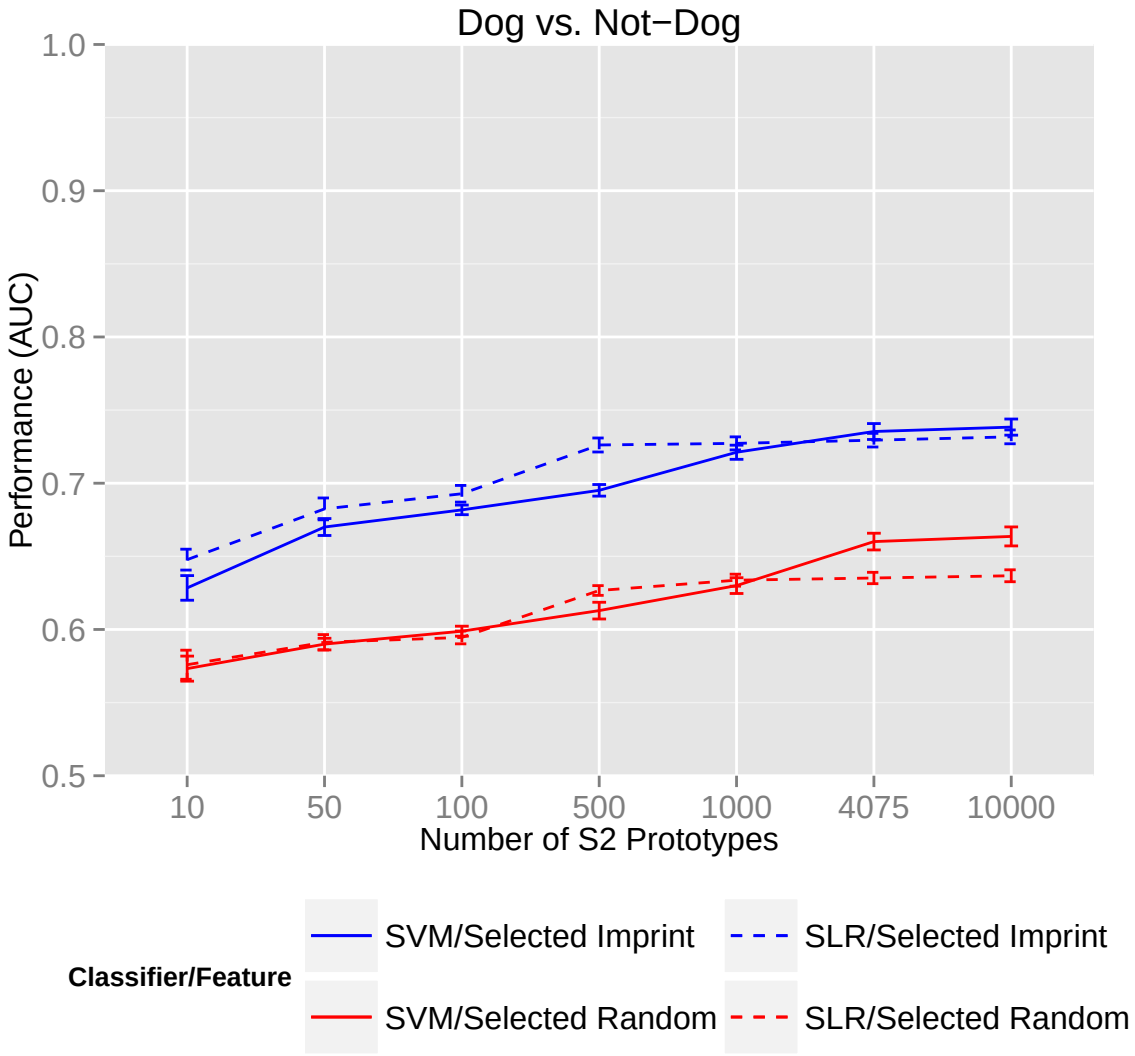


Figure 5.3: Comparison of SVM and Sparse Logistic Regression (SLR) classifiers when used for creating dictionaries with feature selection. The blue lines represent dictionaries created from imprinted prototypes, while the red lines represent dictionaries created from random prototypes. Solid lines indicate SVM performance; dashed lines indicate sparse logistic regression performance.

than there are non-zero weights, the remaining features will be the ones that had zero weights to begin with. Thus adding additional features will have absolutely no additional impact on classification performance.

Chapter 6

Random Projection

Thomure [27] found that for certain synthetic datasets, high-level image features extracted by HMAX with a dictionary of random prototypes performs just as well as those extracted with a dictionary of imprinted prototypes. My experiments have shown that random prototypes do not perform as well as imprinted prototypes on the diverse natural dataset of PASCAL VOC 2012; however, a dictionary of random prototypes still performs well above chance. Furthermore, experiments with single prototypes and feature selection show that some random prototypes are more discriminative than others.

This leads to one of the central open questions posed in Thomure's work: what is the underlying mechanism for the discriminative ability of random prototypes?

Thomure proposes an investigation into the methods of *random projection* and *compressive sensing*. These methods involve using matrices of random values to transform high-dimensional data or high-frequency signals into efficient low-dimensional representations that nevertheless preserve important properties of the input [1, 5, 7, 21].

Among many other applications, these methods have proven useful on the tasks of on facial recognition [12] and visual object categorization [2].

Given that random prototypes are essentially a matrix of random values, it makes sense to study these methods in the context of HMAX. In this chapter I describe a technique called *sparse random projection* [1, 17], and present preliminary experimental results.

6.1 Background

In general, random projection takes a set of points in a high d-dimensional space and *embeds* those points in a low k-dimensional space by multiplying each point by a $d \times k$ projection matrix $\mathbf{R}$, where each component of $\mathbf{R}$ is chosen randomly.

To see how this might be related to HMAX, recall from Section 3.4 that the activity of a single S2 unit is computed by taking a $7 \times 7 \times 4$ patch of C1 units (7×7 across 4 orientations) and applying each of k prototypes to the values in that patch using a radial basis function (Equation 3.4). The result is a $1 \times k$ vector of responses, one for each prototype.

My hypothesis is that if we treat the $7 \times 7 \times 4$ input patch as a point in 196-dimensional space, then perhaps a set of k random prototypes is acting similarly to a $196 \times k$ random projection matrix.

If this is the case, we may be able to replace the computation of k prototype responses with a simple multiplication by a $196 \times k$ random projection matrix. Below, I describe a method for doing so.

6.2 Methods

To evaluate the effectiveness of random projection in HMAX, I introduce a new way of computing S2 activity. At all locations and scales in C1, I take a $7 \times 7 \times 4$ patch centered at that location and flatten it into a 1×196 vector $\mathbf{x}$. I then multiply that vector by a $196 \times k$ projection matrix $\mathbf{R}$, resulting in a $1 \times k$ S2 response at each location. The same $\mathbf{R}$ is used for all locations and scales.

Thus, for every C1 input patch $\mathbf{x}$ at all locations and scales, the corresponding S2

unit's activation is defined as:

$$\mathrm{S2}(\mathbf{x}) = \mathbf{x} \cdot \mathbf{R} \tag{6.1}$$

The output is a set of 9 feature maps (one for each scale), where each S2 unit contains a separate band for each of the k dimensions in the random projection.

The definition of C2 (Section 3.5) is left unchanged. That is, for each element in the k-dimensional projection, I conduct a max-pooling over all locations and scales, resulting in a $1 \times k$ C2 vector describing each input image.

To create the projection matrix $\mathbf{R}$, I use a technique called *sparse random projection* [1, 17], which chooses components such that most of them are equal to zero. This can vastly speed up computation by using a sparse matrix representation. For an embedding dimensionality k and a sparsity parameter s, the components of $\mathbf{R}$ are chosen independently as follows:

$$r_{ji} = \frac{\sqrt{s}}{\sqrt{k}} \cdot \begin{cases} 1 & \text{with probability } \frac{1}{2s} \\ 0 & \text{with probability } 1 - \frac{1}{s} \\ -1 & \text{with probability } \frac{1}{2s} \end{cases} \tag{6.2}$$

I follow Li et al.'s suggestion that s be set to the square root of the original dimensionality. In this case, $s = \sqrt{196} = 14$. This results in a matrix composed of approximately $1 - \frac{1}{14} \approx 92.9\%$ zeros.

Using the method described above, I conducted experiments on the PASCAL VOC 2012 exactly as described in Section 4.1.2 for random prototypes, replacing the creation of a new dictionary of random prototypes for each trial with the creation of a new random projection matrix $\mathbf{R}$. I varied my choice of k (the size of the projection)

Algorithm Evaluating Sparse Random Projection

```
1: procedure EVALUATE(dataset)
2:     for k ← [10, 50, 100, 500, 1000, 4075] do
3:         for each of 8 trials do
4:             Randomly split dataset 50/50 into trainSet and testSet.
5:             Create a projection matrix R according to Equation 6.2, with s = 14.
6:             Use Glimpse with R (Equation 6.1) to extract C2 features
               of images in dataSet.
7:             Train a linear SVM on C2 features and labels of trainSet.
8:             Compute decision values from the C2 activity of testSet.
9:             Output ROC AUC score based on decision values and labels of testSet.
10:        end for
11:    end for
12: end procedure
```

Figure 6.1: Pseudocode for the experimental setup used to evaluate Glimpse with sparse random projections.

over the same dictionary sizes chosen for random and imprinted prototypes, and used the same classifier (a linear SVM), and the same performance metric (the classifier's AUC score).

Pseudocode for my experimental procedure using sparse random projections is given in Figure 6.1.

6.3 Results

Results for $k = 4,075$ are given in Figure 6.2. This figure also shows the results from Figure 4.3 for the C1 histogram baseline as well as dictionaries of 4,075 imprinted and random prototypes. I find that a 4,075-dimensional sparse random projection performs nearly identically to or better than a dictionary of 4,075 random prototypes for every class.

A full performance chart is given in Figure 6.3. This figure shows that the size of a

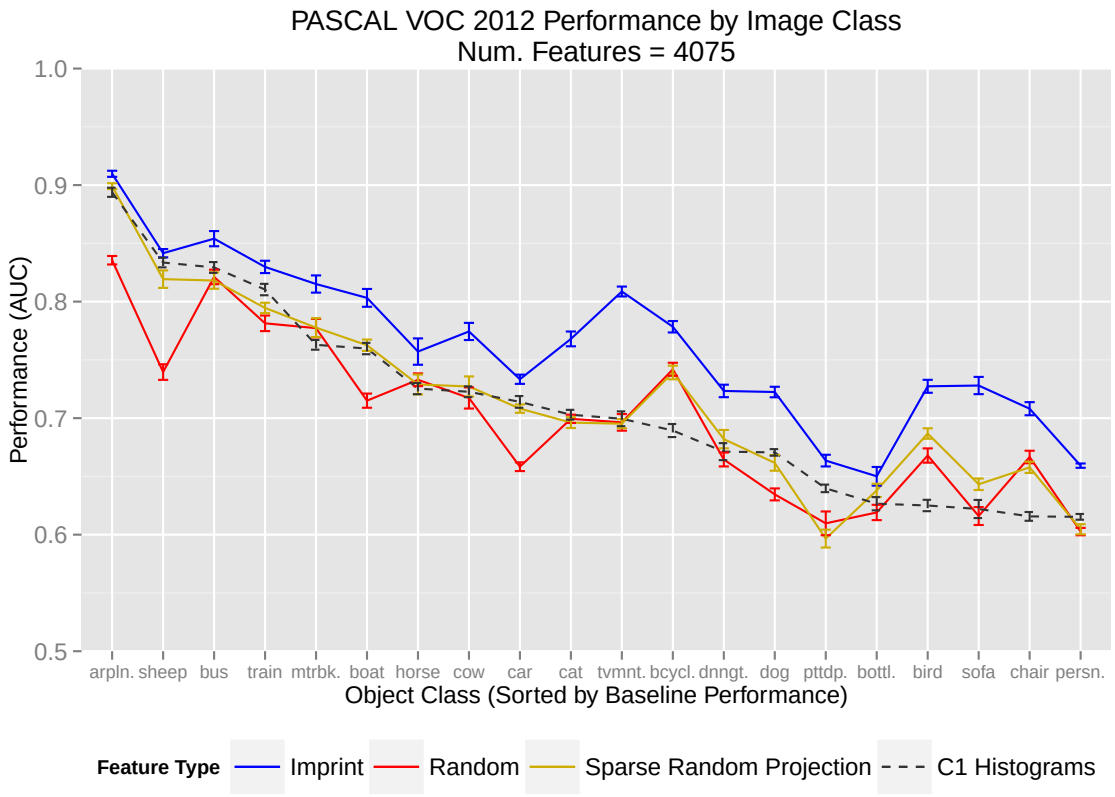


Figure 6.2: Performance results for all 20 PASCAL VOC object classes. Classifier performance is given for baseline features (dashed gray line), features extracted using a dictionary of 4,075 imprinted prototypes (solid blue line), a dictionary of 4,075 random prototypes (solid red line), and a 4,075-dimensional sparse random projection (solid gold line). Error bars indicate standard error over 8 trials. Figure best viewed in color.

dictionary of random prototypes and the dimensionality of a sparse random projection are related, as classifier performance follows roughly the same trend for each as the size of the representation is varied.

As discussed previously in Section 4.2, the best observed performance of HMAX is not far above baseline. Imprinted prototype performance varies from about 1% to 10% above baseline, and random prototype performance varies from about 10% below baseline to 5% above.

Sparse random projections vary from about 5% below baseline to 5% above. This raises the possibility that using sparse random projections within HMAX is simply another baseline method to compare against.

To test this, I also ran the sparse random projection method described above on the "Cars vs. Planes" dataset of Pinto et al. [19], which has proven difficult for baseline features (see Section 4.3.1).

Results are shown in Figure 6.4. I find that a 4,075-dimensional sparse random projection performs nearly identically to a dictionary of 4,075 random prototypes on this dataset as well, which is well above both a C1 histogram baseline and "raw" C1 baseline.

6.4 Discussion

Taken together, these results show that the performance of random prototypes may be explainable as an instance of random projection.

The mechanisms and mathematical formulations behind the performance of random projections are beyond the scope of this work. There are detailed theoretical and technical results [5, 7, 14, 21] in the fields of mathematics, signal processing and information theory that should form the foundation of any future work along these lines.

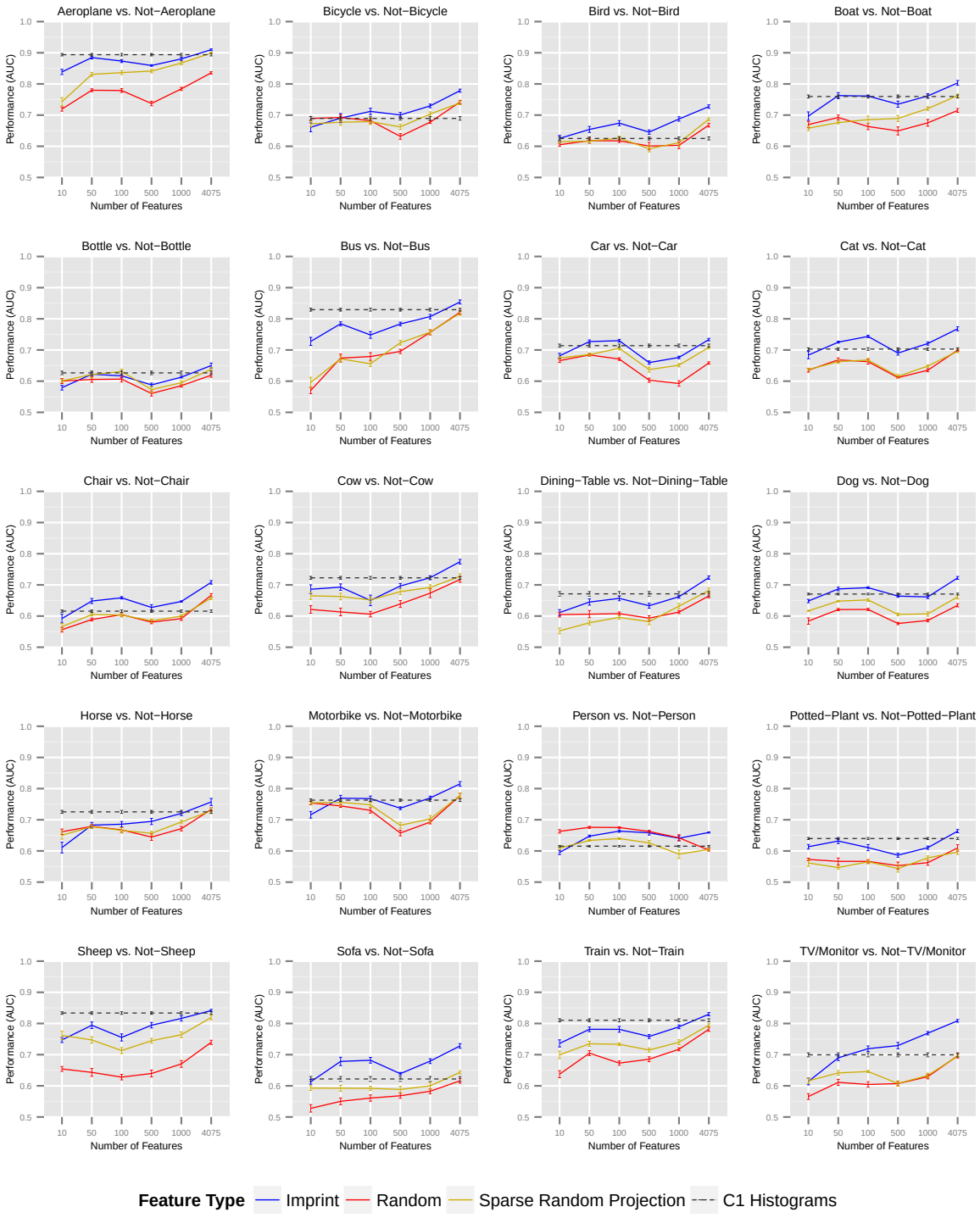


Figure 6.3: Performance results for all 20 PASCAL VOC 2012 object classes when using baseline features (dashed gray line), imprinted prototypes (solid blue line), random prototypes (solid red line), or sparse random projections (solid gold line). For prototype-based features, the x-axis indicates the number of prototypes in the dictionary. For random projections, it indicates the dimensionality of the projection matrix. Error bars indicate standard error over 8 trials. Best viewed in color.

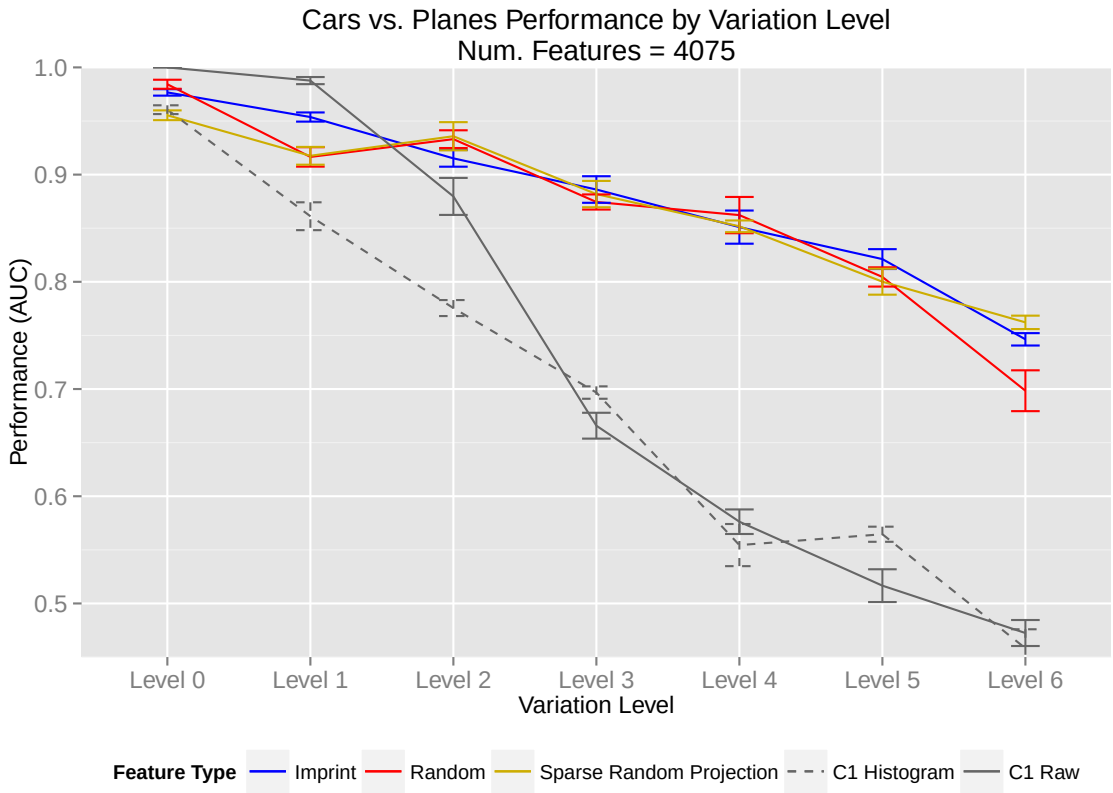


Figure 6.4: Performance on the "Cars vs. Planes" task for features extracted by 4,075 imprinted prototypes (solid blue line), 4,075 random prototypes (solid red line), and 4,075-dimensional sparse random projection (solid gold line). Baselines include C1 histograms (dashed gray line) and concatenated C1 (solid gray line). Error bars indicate standard error over 5 trials. Figure best viewed in color.

One of the more curious aspects of these results is that random projection is usually used to *reduce* the dimensionality of an input, but I find that the best performing projection matrices for this task actually produce a representation of *greater* dimensionality than that of the 196-dimensional input patch. Perhaps the max-pooling operation in C2 fundamentally changes the mathematical formulation of random projection; this should be explored further.

It is also yet unknown what effect different implementations of random projection will have on these results; sparse random projection is only one of a family of related techniques. Another common method is so-called "dense" random projection, which

simply draws each component of the projection matrix $\mathbf{R}$ independently from a zero-mean, unit-variance normal distribution. Sparse random projection was originally proposed as a more computationally efficient alternative [17].

However, sparse random projection is different from dense random projection not just in its sparsity, but also the fact that the projection matrix is composed of discrete values (see Equation 6.2). Each column thus takes a different discrete linear combination of the input patch. This may induce the kind of shape selectivity desired by Riesenhuber and Poggio [20] in their original formulation of HMAX: their S2 layer computed different discrete combinations of C1 input patches (see Section 2.2.2). More theoretical and experimental work is required to explore this hypothesis.

At the very least, the technique I have described above shows that the complicated mathematical operations for calculating S2 activity with random prototypes (Equations 3.4 and 3.5) can be replaced with a much more computationally efficient sparse weighted sum without impacting classification performance. As a coarse preliminary benchmark, computing C2 features for the 260 images in the "Cars vs. Planes" task takes approximately 3 minutes when using a dictionary of 4,075 random prototypes, but only 1 minute when using a 4,075-dimensional sparse random projection. Feature selection (Chapter 5) may improve computational efficiency and classification performance even further. More controlled benchmarks with different numbers of features are necessary future work.

Chapter 7

Conclusions and Future Work

7.1 Performance of Random Prototypes

Thomure [27, 28] showed that a dictionary of random prototypes in Glimpse, an HMAX-like model similar to that of Serre et al. [25], can extract features that are as discriminative as those extracted by a dictionary of imprinted prototypes, even on a difficult task such as Pinto et al.'s [19] invariant object recognition. This provides evidence contrary to the claim of Serre et al. [23] that imprinting is critical to the success of HMAX on object recognition tasks.

In this work, I extend Thomure's result on small datasets to the task of image classification on the large, diverse PASCAL VOC 2012 dataset (Chapter 4). I find that the performance of Glimpse with imprinted and random prototypes is close, and well above chance, for every object class. However, imprinted prototypes carry an advantage in all cases, and random prototypes usually perform at or below a simple baseline constructed from histograms of response to edge filters.

One possible explanation for this is that since my selection of parameters for Glimpse comes from Thomure's work on synthetic datasets, it may not be appropriate for PASCAL VOC 2012, leading to poor performance.

Another possibility is that the additional invariance provided by higher layers is simply unnecessary to be successful on this task, and low-level features are adequate.

Nevertheless, the large diverse dataset of PASCAL VOC 2012 still provides an environment to explore why random prototypes are discriminative at all. I find that the most discriminative random prototypes (i.e., those that perform the best when

used as the only dictionary element) do appear to express some shape selectivity (Section 4.3.2). This is contrary to Thomure's finding that discriminative random prototypes are not selective for any shapes. I believe this supports Serre et al.'s [23] theory that shape-tuning (selectivity) is a key factor in HMAX's performance, but not necessarily their claim that imprinting is necessary for shape-tuning.

I also find that task feedback through feature selection increases performance for both imprinted and random prototypes (Chapter 5). I believe that task feedback is a promising area to conduct further experiments on the shape selectivity of random prototypes.

I hypothesize that random prototypes have a shape selectivity that is "built-in" when they are created. For instance, consider the best performing random prototype for bicycle, shown in Figure 4.7 (Page 41). Three of the four top responses contain a bicycle wheel at the same position in the right half of the patch. A form of task feedback that modifies random prototypes directly, such as through gradient descent, might accentuate the values in the prototype that make it respond to this particular shape, thereby increasing selectivity and potentially increasing the system's performance. Modifying features through gradient descent is one of the techniques behind the success of deep neural networks (see LeCun et al. [16] for a recent overview).

7.2 Random Projection

In this work, I also explore the possibility that the underlying mechanism behind the discriminativity of random prototypes is random projection (Chapter 6). I find that a dictionary of random prototypes can be replaced by a sparse random projection matrix with no degradation in classification performance, while also being significantly more efficient to compute.

Further work is necessary to explore these results in depth. The technique I have

outlined in this work should be applied to additional datasets to ensure that its performance is not a quirk of the ones used here. I have coarsely verified that computing features from sparse random projections is faster than computing features from random prototypes for 4,075 features, but more detailed and controlled benchmarks are necessary to see how each of these methods scale with different dictionary sizes. Future work should also include experiments to determine the effect of feature selection on sparse random projections.

Crucially, however, while I have found that random prototypes and sparse random projections appear to be interchangeable for the tasks outlined in this work, I have not yet found an explanation for why sparse random projections themselves can be used to extract discriminative features for object recognition. Further experimental and theoretical study in the fields of random projection and compressive sensing is an essential next step.

Bibliography

[1] Achlioptas, D. Database-friendly Random Projections: Johnson-Lindenstrauss with Binary Coins. *Journal of Computer and System Sciences 66*, 4 (2003), 671–687.

[2] Arriaga, R. I., Rutter, D., Cakmak, M., and Vempala, S. S. Visual Categorization with Random Projection. *Neural Computation 27*, 10 (2015), 2132–2147.

[3] Bruce, C., Desimone, R., and Gross, C. G. Visual Properties of Neurons in a Polysensory Area in Superior Temporal Sulcus of the Macaque. *Journal of Neurophysiology 46*, 2 (1981), 369–384.

[4] Cadieu, C. F., Hong, H., Yamins, D. L. K., Pinto, N., Ardila, D., Solomon, E. A., Majaj, N. J., and DiCarlo, J. J. Deep Neural Networks Rival the Representation of Primate IT Cortex for Core Visual Object Recognition. *PLoS Computational Biology 10*, 12 (2014), e1003963.

[5] Candes, E., and Tao, T. Near-Optimal Signal Recovery From Random Projections: Universal Encoding Strategies? *Information Theory, IEEE Transactions on 52*, 12 (2006), 5406–5425.

[6] Cortes, C., and Vapnik, V. Support-Vector Networks. *Machine learning 20*, 3 (1995), 273–297.

[7] Do, T. T., Gan, L., Nguyen, N. H., and Tran, T. D. Fast and Efficient Compressive Sensing Using Structurally Random Matrices. *Signal Processing, IEEE Transactions on 60*, 1 (2012), 139–154.

[8] Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J., and Zisserman, A. The Pascal Visual Object Classes (VOC) Challenge. *International Journal of Computer Vision 88*, 2 (2010), 303–338.

[9] Everingham, M., and Winn, J. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Development Kit. `https://web.archive.org/web/20151026070603/http://host.robots.ox.ac.uk:8080/pascal/VOC/voc2012/htmldoc/index.html`.

[10] Fabre-Thorpe, M., Richard, G., and Thorpe, S. J. Rapid Categorization of Natural Images by Rhesus Monkeys. *Neuroreport 9*, 2 (1998), 303–308.

[11] Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., and Lin, C.-J. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research 9* (2008), 1871–1874.

[12] Goel, N., Bebis, G., and Nefian, A. Face Recognition Experiments with Random Projection. *Proceedings of SPIE 5779* (2005), 426–437.

[13] Hubel, D. H., and Wiesel, T. N. Receptive Fields, Binocular Interaction and Functional Architecture in the Cat's Visual Cortex. *The Journal of Physiology 160*, 1 (1962), 106–154.

[14] Johnson, W., and Lindenstrauss, J. Extensions of Lipschitz Maps into a Hilbert Space. *Contemporary Mathematics 26* (1984), 189–206.

[15] Krizhevsky, A., Sutskever, I., and Hinton, G. E. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 25*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds. 2012, pp. 1097–1105.

[16] LeCun, Y., Bengio, Y., and Hinton, G. Deep Learning. *Nature 521*, 7553 (2015), 436–444.

[17] Li, P., Hastie, T. J., and Church, K. W. Very Sparse Random Projections. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2006), pp. 287–296.

[18] Mutch, J., and Lowe, D. Multiclass Object Recognition with Sparse, Localized Features. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on* (2006), vol. 1, pp. 11–18.

[19] Pinto, N., Barhomi, Y., Cox, D. D., and DiCarlo, J. J. Comparing State-of-the-Art Visual Features on Invariant Object Recognition Tasks. *2011 IEEE Workshop on Applications of Computer Vision (WACV)* (2011), 463–470.

[20] Riesenhuber, M., and Poggio, T. Hierarchical Models of Object Recognition in Cortex. *Nature Neuroscience 2* (1999), 1019–1025.

[21] Romberg, J. Compressive Sensing by Random Convolution. *SIAM Journal on Imgaging Sciences 2*, 4 (2009), 1098–1128.

[22] Saxe, A., Koh, P. W., Chen, Z., Bhand, M., Suresh, B., and Ng, A. On Random Weights and Unsupervised Feature Learning. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)* (2011), L. Getoor and T. Scheffer, Eds., pp. 1089–1096.

[23] Serre, T., Oliva, A., and Poggio, T. A Feedforward Architecture Accounts for Rapid Categorization. *Proceedings of the National Academy of Sciences 104*, 15 (2007), 6424–6429.

[24] Serre, T., Wolf, L., Bileschi, S., Riesenhuber, M., and Poggio, T. Robust Object Recognition with Cortex-Like Mechanisms. *Pattern Analysis and Machine Intelligence, IEEE Transactions on 29*, 3 (2007), 411–426.

[25] Serre, T., Wolf, L., and Poggio, T. Object Recognition with Features Inspired by Visual Cortex. In *Computer Vision and Pattern Recognition, 2005 IEEE Computer Society Conference on* (2005), vol. 2, pp. 994–1000.

[26] Thomure, M. D. Glimpse, a General Layer-wise Image Processing Engine. Code: `http://dx.doi.org/10.5281/zenodo.32091`.

[27] Thomure, M. D. *The Role of Prototype Learning in Hierarchical Models of Vision*. PhD dissertation, Portland State University, 2013. `http://pdxscholar.library.pdx.edu/open_access_etds/1665`.

[28] Thomure, M. D., Mitchell, M., and Kenyon, G. T. On the Role of Shape Prototypes in Hierarchical Models of Vision. In *Neural Networks (IJCNN), The 2013 International Joint Conference on* (2013), pp. 1–6.

[29] Thorpe, S., Fize, D., and Marlot, C. Speed of Processing in the Human Visual System. *Nature 381*, 6582 (1996), 520–522.