# Object Detection and Recognition in Natural Settings

by

George William Dittmar

A thesis submitted in partial fulfilment of the
requirements of the degree

Master of Science
in
Computer Science

Thesis Committee:
Melanie Mitchell, Chair
Fei Xie
Feng Liu

Portland State University
2012

**Abstract**

Much research as of late has focused on biologically inspired vision models that are based on our understanding of how the visual cortex processes information. One prominent example of such a system is HMAX [17]. HMAX attempts to simulate the biological process for object recognition in cortex based on the model proposed by Hubel & Wiesel [10].

This thesis investigates the ability of an HMAX-like system (GLIMPSE [20]) to perform object-detection in cluttered natural scenes. I evaluate these results using the StreetScenes database from MIT [1, 8]. This thesis addresses three questions: (1) Can the GLIMPSE-based object detection system replicate the results on object-detection reported by Bileschi using HMAX? (2) Which features computed by GLIMPSE lead to the best object-detection performance? (3) What effect does elimination of clutter in the training sets have on the performance of our system?

As part of this thesis, I built an object detection and recognition system using GLIMPSE [20] and demonstrate that it approximately replicates the results reported in Bileschi's thesis. In addition, I found that extracting and combining features from GLIMPSE using different layers of the HMAX model gives the best overall invariance to position, scale and translation for recognition tasks, but comes with a much higher computational overhead. Further contributions include the creation of modified training and test sets based on the StreetScenes database, with removed clutter in the training data and extending the annotations for the detection task to cover more objects of interest that were not in the original annotations of the database.

## Dedication

Dedicated to my parents and family for their love and support while pursuing my education. Love you all.

## Contents

## List of Tables

## List of Figures

## Chapter 1 Introduction

Computer vision has been an area of research for many years, tackling the problem of developing algorithms and systems to recognize and understand visual information. Such algorithms can range from trying to recognize pictures of particular objects to more recent work on finding and understanding what the underlying context and meaning in a picture is. Naturally, much research focused on trying to understand how we as humans use our eyes and brains to understand and recognize visual information rather than trying to develop completely "new" algorithms. Hubel and Wiesel pioneered work on understanding how the visual cortex does recognition in cats, and it is from this work that HMAX was born. HMAX was also inspired by the Neocognitron proposed by Fukushima [9] and further extensions of HMAX were developed by Serre et al. [17, 18, 19].

In computer vision, object recognition and object detection are sometimes considered separate (though related) tasks. Recognition deals with the task of feeding an image of a single object into some recognition system and seeing if it is correctly identified in the image or not. As an example, we might show the system an image of a car and an image of a tree and see if our system can recognize which one is the image of the car.

Object detection, though similar to recognition, is a much harder task because we are asking our algorithm to find all the locations of some object in an unconstrained setting. These settings can be anything from finding a pair of glasses on a table full of objects to trying to develop software for security cameras to detect the location of all people walking by in real time. The task addressed in this thesis is

to find the location of all cars in a given outdoor image (a "street scene"). Figure 1.1 shows an example of object recognition while Figure 1.2 shows an example of object detection.

Figure 1.1: Sample image from the training set showing an ideal case of the problem of object recognition. Notice that the object is fairly centred in the image and that there is not much else to potentially add clutter when extracting features.

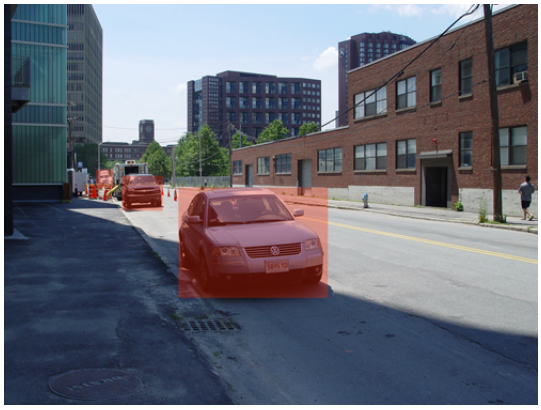


Figure 1.2: Sample image from the test set showing the type of natural settings in the dataset. In a perfect scenario for object detection our system would find all the bounding boxes that surround the cars while returning no incorrect detection locations.

Figure 1.1 has a car centred in the image and the car is the only object, besides some background clutter at the edges. Given enough training data showing this object class, here ,"car", modern computer vision systems can learn a model that describes a car. We can then show the model a new "car" image and it should be able to tell that the image is that of a car. Figure 1.2 illustrates the much harder task of finding the location of cars and surrounding each car with a bounding box. Now the task has changed from identifying a single object (without explicitly

locating that object in the image) to finding the locations of all instances of the object in a cluttered scene. I use the term "cluttered" to mean images taken in natural settings that contain many different types of objects in the foreground and background.

Humans perform this sort of task fairly easily because we are able to learn generalities about certain types of objects such as cars and other people. This generalization helps us to be able to tell that a Cadillac and a Subaru are both cars of some sort. On top of being able to identify common features between different instances of the same object categories, our brains are able to recognize these objects given different translations and even recognize them if they are occluded. Current-day computer vision systems are ,as yet, unable to come close to matching human performance on such tasks.

The work presented in this thesis looks at the application of the HMAX vision model to recognition and detection tasks in natural settings. When I say natural settings I mean images that have not been "staged" or setup in a way that might make the task easier. Essentially this means just taking images that were shot on the street and using them for training and testing of HMAX.

Bileschi [1] applied HMAX to the object detection task for three categories: Person, Bike, and Car. The object-detection system I built is based on Bileschi's StreetScenes system. Bileschi's code is freely available online [8], but is written in MatLab.

My system is implemented in Python and uses Thomure's GLIMPSE implementation of HMAX [2, 20] (written in C++ and Python). Reimplementing Bileschi's system has allowed me to attmept to verify his published results as well as to modify the system for additional experiments not currently supported in the original source code.

## 1.1 Thesis Contributions

The contributions of this thesis include:

- The development of a Python implementation of object recognition and detection algorithms based on Bileschi's StreetScenes project [1]. My implementation consists of the recognition and detection scripts, and uses Thomure's GLIMPSE HMAX system [20].

- Experiments to test the model's recognition capabilities and test the detection algorithms ability to find object locations.

- Modification of the training and testing sets from the original StreetScenes database to create new training and testing sets to be used with our experiments. Training sets were created from the original StreetScenes database for each of the three object classes and then I manually removed training images that contained "clutter". What I call clutter in the training sets is when a negative example contains an object of interest for that class. For example a negative crop contains part of a car for the car classes training set.

- Additional annotations of 100 plus test images used in our car detection experiments, in order to make up for the lack of object annotations in the original StreetScenes database. I found that many of the images in the original database contained up to a dozen objects of interest but only two or three annotations. This was noted as an area for future work and improvement by Bileschi [1]. In Section 5.3 I discuss the main differences between my version of the data set and Bileschi's original data set.

All code and training and testing sets developed for this thesis will be made available for anyone to use on GitHub.

## 1.2 Background

Much work has been done to develop algorithms to recognize and detect objects in a way that is invariant to position, scale, and translation. The field of computational neuroscience has tried to fuse the world of neuroscience and computer science to help understand how humans process visual information. This has lead to biologically inspired computer algorithms based on the work of Hubel and Weisel [10] which has come to be known as the *standard model*. Hubel and Wiesel demonstrated the existence of *simple* and *complex* cells in cortex that are stacked in a hierarchical fashion so as to build a cascading layered network of recognition. Simple cells are cells that respond to oriented edges and act as filters on the input. Complex cells are cells as well respond to oriented edges but have a certain degree of invariance for their responses. The complex cells are an aggregate of many simple cells and respond to information from the simple cells [10]. This alternating structure of simple then complex cells has become the basis for many computer vision systems. One of the first computational models of this neural network, was the Neocognitron [9].

The Neocognitron, developed by Fukushima, is a hierarchical multi-layered neural network designed to learn patterns that are invariant to position and scale [9]. The Neocognitron was influential for the computer vision field because it showed that a network could be built using alternating layers of simple and complex cells to perform recognition tasks such as character recognition[9]. Further work from Poggio et al. at MIT extended the model by taking a Neocognitron style hierarchical model and focusing on improving the complex cell layer of the network. Their work modified the computation at the complex layer by changing from a linear (SUM) to a non-linear MAX operation, selecting the input to the complex cell that had the strongest activation value. Using a MAX function, verses say a SUM, helps to determine preferred features when extracting object

information from images [17]. Even further extensions to the HMAX model came with the addition a learning phase that learned combinations of edges, or prototypes, at the higher levels of the model [18]. This addition morphed the model into what it is today which allows for a final vector of features to be sent to a support vector machine for classification tasks.

## Chapter 2 The HMAX Model

HMAX is a biologically inspired computer model using feedforward neural networks to process an image to extract relevant features. The network architecture is a simulation of the model proposed by Hubel and Wiesel [10] that alternates layers of simple cells with complex cells and was first demonstrated in the Neocognitron by Fukushima [9]. HMAX was further extended to include a learning step [18] which extends the model to build object feature vectors that can be used to learn an object class.

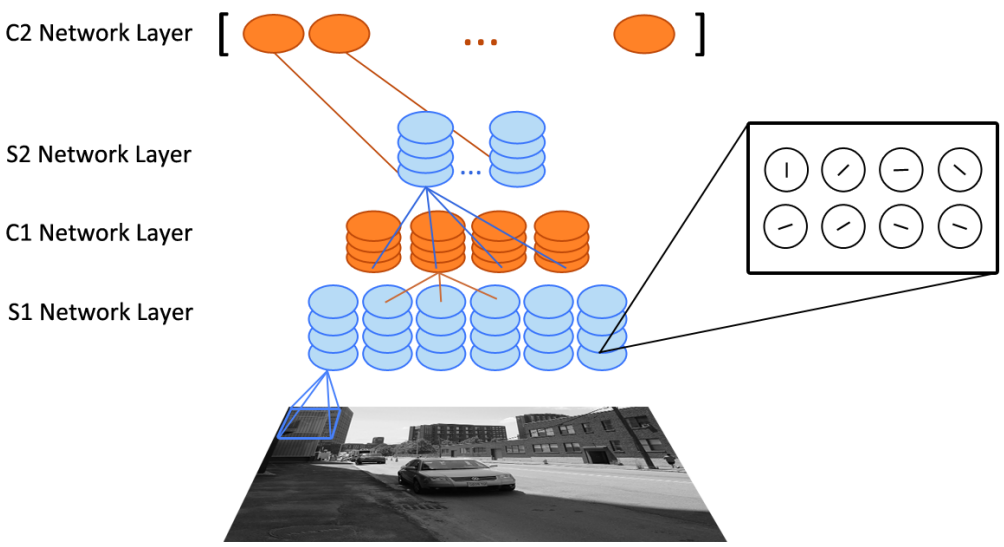



Figure 2.1: A generalized illustration of the HMAX computer vision model proposed by Serre et al. [18]. This model consists of simple and complex layers that first find a series of edges at the S1 layer and then learn prototypes at S2. The C2 layer is an N-component vector, where N is the number of learned prototypes in the model. This vector can then be sent to well-known classification algorithms such as a Support Vector Machine to build a classification model of an object.

HMAX at its simplest consists of four network layers, see Figure 2.1, consisting of simple and complex units, which we shall from now on just call S and C layers. The job of these layers is to extract interesting image features that are invariant to position, scale, and orientation [1, 2, 15, 19]. In our system the layers are known, respectively, as S1, C1, S2,and C2.

**The S1 layer**: This is the initial layer of the model and consists of an array of S cells, each one being activated by a particular edge at a given scale and orientation (See Figure 2.1). Each S1 cell has a receptive field associated with its particular edge detector. Another way to look at this layer is that a receptive field can be thought of as associated with a "column" of S1 cells that each look for a particular activation at a given scale and orientation. The number of scales and orientations used in the model are parameters that can be set in the GLIMPSE framework very easily [20]. In our experiments we are using the default parameters of 4 scales and 8 orientations at S1.

**The C1 Layer(Pooling stage):** At this layer of the network each C1 unit pools over a neighborhood of S1 units at a particular position and scale. Pooling is the act of learning some invariance in the features by finding the maximum activation over the neighborhood of S1 units.

**The S2 Layer:** S2 is a layer that learns shape prototypes. A prototype is essentially a particular combination of edges. An S2 unit has high activation if similar combinations of edges are found in the input image as were learned from training data. Following [18], "learning" in the S2 layer consists of "imprinting", where patches are sampled from C1 activations on training set images. GLIMPSE has the ability to learn multiple prototypes as well as use different prototype learning methods in addition to imprinting [20].

**The C2 Layer (Pooling stage):** The C2 layer is the final pooling stage of the model. Each C2 unit pools over an S2 prototype that has been applied to all parts of an image. The output from the C2 layer is a one-dimensional vector that

captures the generalized object's shape information [16]. These features are then passed to a classifier, in our case a linear support vector machine.

There are many different variants of HMAX and each one has its own strengths and weaknesses. The GLIMPSE system, used in this thesis, allows for the easy implementation of computer vision tasks in Python and comes with a parametrizable HMAX-like framework[20] and additional libraries.

# Chapter 3 Methodology

In this chapter I discuss the implementation details of my object detection and recognition code, and the evaluation metrics I used. I also describe the reasons behind certain design choices I made. Most of the work discussed in this chapter involves my direct port of Bileschi's StreetScenes object detection and recognition algorithms [1] to use with GLIMPSE [20]. Even though Bileschi provided his original Matlab code for his algorithms [8], I found that many design choices were not easy to understand until I wrote my own version and then corresponded with Bileschi.

## 3.1 Dataset Creation

To test and train my object detection system I built additional support code similar to that used in the original StreetScenes system [1] to create training and testing sets from the larger image database. See Figures 3.5 and Section 3.4 for a detailed explanation on how my system extracts training examples and trains the SVM classifier. The StreetScenes database consists of 3500+ annotated images that were taken in and around the Boston area. The images were then hand annotated by Bileschi by drawing polygons around objects of interest given certain criteria, e.g., no objects are occluded by other objects [1]. All of the annotations are then stored in associated XML files which contain the $x$ and $y$ coordinates of all objects of interest in any of the 3500 images. For a more in-depth explanation on how StreetScenes training and testing sets were created, see [1].

For my implementation, since I did not use any of the original source code,

I had to build my own training and testing sets from the larger StreetScenes database. To build these sets the scripts took all 3500 annotated images from the database and split them in half, resulting in about 1750 images each used for training and for testing. These will be known as our training and testing database splits. From the training split of the database, I then wrote scripts to extract grayscale crops to be used for training sets for the car, bike, and pedestrian object classes. This code follows the same methodology that Bileschi used for building his training sets [1].

Figure 3.1: Sample positive class training crops taken from the StreetScenes Database for Car, Pedestrian, and Bike. During the extraction process I attempt to keep the object of interest as centred as possible in the crop. All crops are taken from the training split in the database and are scaled to 128x128 grayscale images.

Figure 3.2: Sample negative class training crops taken from the StreetScenes Database for Car, Pedestrian, and Bike. All crops are resized to 128x128 and randomly sampled from the StreetScenes database making sure to not include objects of interest.

My training set creation scripts extract positive examples from the training split by using the associated XML annotations to crop out all objects of interest, such as a car. My code takes these polygonal annotations, and calculates a bounding box that centers the object in the bounding box, or as best as possible. Each bounding box is additionally resized to have the same width and height. The

image is then cropped at the location of the bounding box. All extracted crops are resized to 128x128 so as to have the same size and aspect ratio, to simplify training the model and ensure that the number of features extracted remains the same. Some examples of positive and negative crops are shown in Figures 3.1 and 3.2.

Negative class examples ("distractors") are randomly taken from all images sampled from the training split. I randomly generate square bounding boxes over the image and randomly set the scale of the bounding-box from 128 to 920 pixels. This simulates sampling from the same set of scales that objects are extracted from, since objects in the database are all different sizes. My code ensures that none of the generated bounding boxes intersect or contain an object annotation by more than 10%.

If the code does extract a negative bounding box that contains part of an object that was annotated, it throws out that box and extracts another random bounding box from the image. This is done until $n$ instances for the negative set have been extracted. I made sure to have the same number of positive and negative crops for the training sets. For the training sets that were constructed for each class there were 1000 positive and negative examples for car; 742 positive and negative examples used for Pedestrian; and 102 positive and negative examples for Bicycle.

It should be noted here that there are still some instances where a partial object of interest may be contained in a negative crop (clutter) due to lack of annotations for all object classes in an image. Due to the process described in [1], the number of objects that are annotated in each image is small compared to the total number of actual objects in an image by visual inspection. Thus by using the above mentioned method, to build our training sets we are prone to potentially adding some background clutter into our negative sets which can add some confusion into the model. I will explain further how clutter may affect our

results in the chapter 5.

## 3.2 Object Recognition Task

To evaluate the learned object models on recognizing objects that are not in "clutter", I performed a series of object recognition tasks. Clutter is a term that Serre et al. used to describe natural images where objects are not cropped out or localized already[1, 19]. I wrote scripts to perform cross validation experiments as described in 3.5.3 for all three object classes. The scripts randomly split the object classes training set into $\frac{2}{3}$ training, and $\frac{1}{3}$ validation splits. From the $\frac{2}{3}$ split, I trained GLIMPSE to learn an object class and then evaluated the trained model on the $\frac{1}{3}$ validation split. This is done for k times ("folds"), where at each "fold" a new randomly generated training and validation split are trained and evaluated. In my experiments we do this for five runs, so I generate five different training and validation sets in total. The scripts then calculate the average accuracy and average Area under the Curve from the ROC curve from all five experimental runs. AUC and ROC curves are explained in section 3.5.1.

### 3.2.1 Feature Extraction

In addition to wanting to see how well our models perform on a simple recognition task (i.e. object "present" or "absent") I also wanted to get an idea of which features are best for representing the object classes of interest. I extracted image features from the C1 layer as well as the C2 layer to determine if vital object features are more pronounced at lower layers than higher network layers. Additionally I was interested in the performance of a combination of C1 and C2 features.

Extracting features from the C1 layer of GLIMPSE produces a multi-dimensional feature vector of about 18,000 plus features for a 128x128 pixel example crop. The reason for a high number of features is because C1 still contains explicit feature

information for each orientation and scale. I ensure that all examples used for training are 128x128 pixels so the feature vectors extracted at C1 are consistent in size.

The C2 layer extraction generates a single one-dimensional feature vector of prototype activation's that represent an object. Prototypes are combinations of edges to form shape parts learned from the training set. For my experiments I use the default prototyping rule explained in Section 2, learning 1000 prototypes. Finally for the combination of features, I extract features from the C1 and C2 layers and then concatenate the two vectors together. This process does involve flattening the C1 vector so I end up with a final one-dimensional vector that contains about 19,000 features per example.

### 3.3 Object Detection

Object detection consists of code to locate objects in much larger images (1280 by 960 pixels) from the test set. The test set consists of images that were not used for extracting training examples. Object detection involves the implementation of a well-known, but exhaustive technique of sliding a fixed size window over the entire image. This involves taking a fixed size square box and moving it over every position extracting crops from the image, then processing all of the extracted crops with GLIMPSE and classifying the resulting features with a trained support vector machine (SVM) [13, 16, 19, 20]. This is done for multiple scales of the image by downsampling the input image after each round of extracting crops. Downsampling is the act of resizing an image to be smaller while maintaining the aspect ratio.

To find object locations in an image the algorithm goes through the following steps:

1. Compute C1 activation's by running the whole image through GLIMPSE. This produces a multi-dimensional feature vector representation of the im-

age.

2. Extract sub-sections out of the feature vector at every position. Each sub-section corresponds to a bounding box of a particular size at a particular location in the image.

3. Send sub-section feature vectors from image to trained SVM for classification. Get back a series of decision values from the SVM.

4. Save decision values and associated bounding boxes.

5. Repeat the first step, but downsample the input image on this round. Downsampling consists of resizing the image by a certain percentage while maintaining aspect ratio. The purpose of downsampling the input image is to be able to extract crops at a different scale than before, without having to resize the window.

6. Once all scales of the image have been processed, apply the local neighborhood suppression algorithm, explained in Section 3.3.1, on all of the bounding boxes extracted from the image.

The above steps are performed on all images in the test set. Section 3.3.1 discusses in more detail how the detection scripts find potential locations in the image. For my code to make a positive detection, the detection's decision value must be greater than zero. Additionally I limit the number of detection's extracted per image, so if the script finds a new detection but it is over the maximum allowed number, the system stops looking. For my experiments I set the detection limit to 20 per image.

The problem with this strategy is that since the system is evaluating bounding boxes at all possible locations and several different bounding box scales, the search for object locations becomes a very computationally expensive task when applied on a test set of large images. Since this is can be an exhaustive technique, the run

times for processing a single image can be long. Bileschi states his code was able to process an entire image in around 300 seconds [1]. I found that my implementation can process an image in under 3 minutes, but can take much longer depending on the number of scales and bounding-box step size parameters that are set for an experiment. The parameters in Bileschi's thesis were not explicitly stated which unfortunately means there is much leeway in how the system will perform compared to the original StreetScenes code.

Further communication with Bileschi about StreetScenes confirmed similar issues with trying to perform a search at a very fine level. For my implementation it can take up to 10 hours total to process 100 images using 65 scales and a bounding box step size of 15. This makes the detection step a very computationally expensive task to perform when dealing with many large images and thus some fine tuning must take place to balance computation time with system performance.

### 3.3.1 Local Neighborhood Suppression

When performing object detection using a sliding window, the likelihood for detections to cluster around a particular object are very high [1, 15]. This is not necessarily a flaw of the system or classifier itself but more due to objects that are detected showing similar decision values from bounding boxes that also contain parts of an object. Given that our task is to find a bounding box that best describes where an object is, returning results that are stuck in a particular region of an image does not tell much about where other objects might be. To try to overcome this issue, Bileschi implemented a local neighborhood suppression algorithm to reduce the chance for duplicate detections of the same object [1]. A similar suppression algorithm was used by Mutch & Lowe in their HMAX variant, but was more fine tuned to specific detection tasks and datasets [15].

My version of the algorithm is modeled after Bileschi's and works by taking the bounding boxes and decision values extracted by my windowing code, and finds a

global maximum over all decision values, checking that the decision value is greater than zero. The algorithm then picks the associated bounding box containing a possible detection location. I then take the bounding box and build a matrix of size $N$ x $N$ where N is the size of the bounding box plus 35% of the length of a side of the bounding box. This neighborhood is filled with Gaussian values generated by 1 - G(x,y) (see Equation 3.3,) at that $x$,$y$ position in the matrix. Any detection's that then have their center points fall inside this matrix get multiplied by the value of the associated Gaussian suppression value in the neighborhood map.

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Figure 3.3: Equation for the 2D Gaussian function used in my neighborhood suppression algorithm to build the suppression neighborhood map. $\pi$ and $\sigma$ are tunable parameters that are set in each experiment. The $x$ and $y$ values correspond to $x$ and $y$ coordinates from the origin of the matrix.

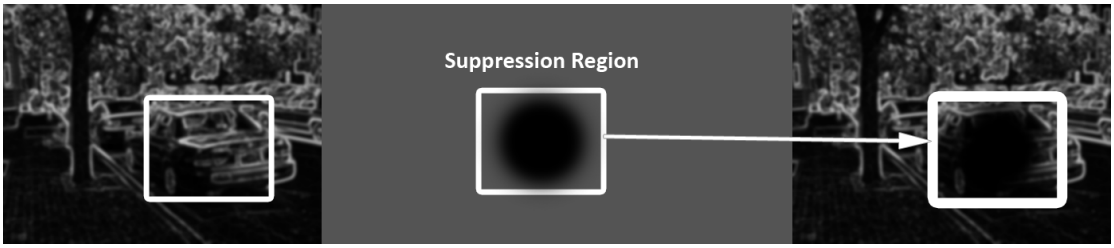



Figure 3.4: A visual rendering of what happens with the local suppression algorithm if you were looking at it with your eyes as a region of the image is suppressed. The suppression region within the bounding box will be blacked out so that the system cannot look in that area again because as far as the system knows, nothing is there.

The algorithm repeats this step until all possible detection locations are found, or until some pre-set stopping point is reached, such as number of detection locations returned or a decision value threshold. The advantage of this suppression

technique over just flattening large regions to zero is that there might be objects of interest close or partially inside the local region being suppressed. Using the Gaussian gives us the ability to suppress detection's by a greater amount if their center point is closer to the center of the neighborhood, while suppressing bordering detection's far less. Flattening a region to zero is an easy approach to this task but is too "destructive" to the search space for our needs for objects that might be right next to each other.

### 3.4 Training and Testing workflow

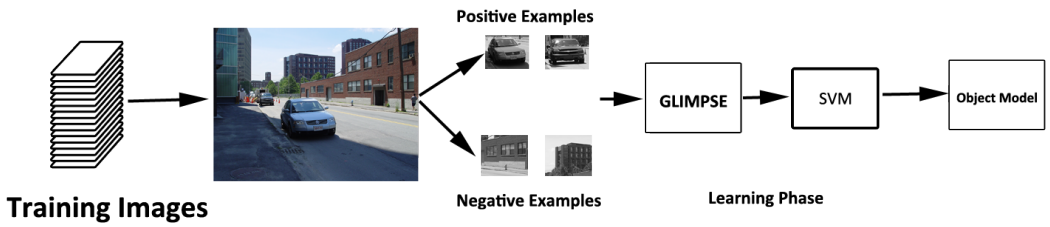



Figure 3.5: Object classification training stage workflow. The system first extracts objects, such as cars, from the training images split using the annotations for the image. The system extracts negative crops by randomly cropping sections of the image at random scales between 128 and 960 pixels. All crops extracted are resized to 128 by 128 pixels. Learning occurs by sending the image crops through GLIMPSE and then sending the feature vectors to a support vector machine. The final stage is the saving of the svm model used for classification in the recognition and detection experiments.

The workflow of my system can be broken down into two main stages: the training stage, Figure 3.5, and testing stage, Figure 3.6. To train an object model, my system extracts positive and negative image crops, as described in Section 3.1. The system takes the extracted positive and negative example crops and then

processes all of them inside GLIMPSE. The processing in GLIMPSE can be done to any layer of the network, S1 through C2, so we can specify what features we are wanting to learn. Each crop's feature vector is then sent to the support vector machine where the learning of the object model occurs. After the model is learned, the GLIMPSE code then saves the learned SVM model to disk for reuse in the testing stage.

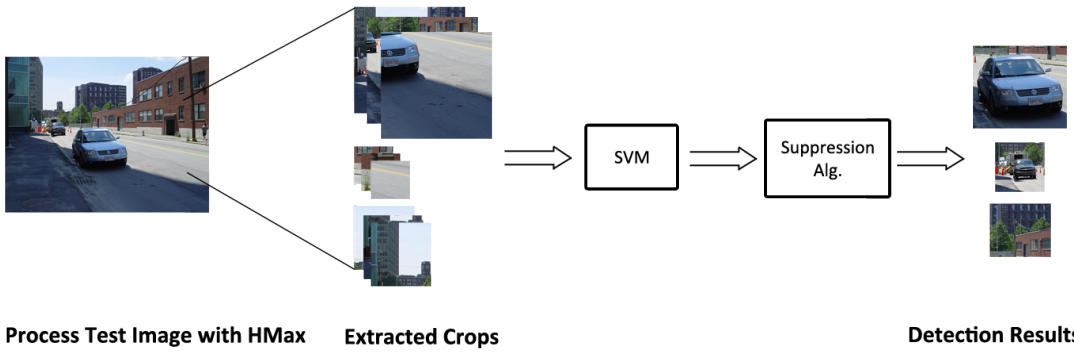



Figure 3.6: Image processing work flow used for the detection of object locations in clutter.

The testing stage is the application of the object detection steps to multiple images as described in Section 3.3. After all images are processed by GLIMPSE, the system takes the returned detection locations and checks to see if any of them intersect with ground truth annotations by more than 50 percent in area. From this evaluation check stage I generate precision recall graphs that give, for each detection threshold the system's precision and recall for the set of objects that were detected at that threshold.

### 3.5 Evaluation Metrics

I follow the same set of evaluation metrics as used in Bileschi's thesis but shall explain them in further detail here for the reader. The two main evaluation metrics I used are the ROC curve and precision-recall curves. Both of these metrics have been used before for vision challenges and have been used in the PASCAL computer vision challenge for the last six years [6].

#### 3.5.1 ROC and AUC Metrics

The Receiver Operator Characteristic (ROC) curve is used as a measure of results for a binary classification problem [5]. ROC curves plot the changes in the true-positive rate verses the false-positive rate of a classifier. TP-rate is the number of true positives over the number of true positives plus false negatives and the FP-rate is the number of false positives over the number of false positives plus the number of true negatives. This curve is used in determining the best performing classification models based on the TP-rate and FP-rate from the validation data. Perfect classification is found in the top left corner of the curve while worse classification is towards the bottom right.

From the ROC curve a single value can be calculated that will tell us how well our classifier is performing. The area under the curve (AUC) is a score of the discrimination quality of the classifier to label a random example correctly. The higher the AUC score is, the more likely the classifier is to rate a randomly selected positive instance as a positive class and a randomly selected negative instance as a negative class. Using accuracy results alone may be misleading because it does not really say anything about how a classifier performs as the number of incorrect classifications increases [7]. Because of this the AUC has become a standard metric used with the ROC curve to evaluate classification algorithms.

### 3.5.2 Precision-Recall Metrics

Precision-Recall is a standard evaluation metric used mostly in the information retrieval field to test how well a system does at returning relevant information at different levels of recall [5]. Precision is calculated as the number of true positives over the number of retrieved detections and recall is the number of true positive detections over the number of objects in the test set. For my detection experiments this is a good measure of how many objects are being detected by my system as the support vector machines decision value threshold is lowered. The precision-recall graph tells us the trade off between the loss of precision with the increase in recall.

### 3.5.3 Recognition Evaluation

For the evaluation of Bileschi's object recognition code, he employed a cross-validation technique known as repeated random sub-sampling. Repeated random sub-sampling works by using an objects training set and creating fixed sized, random training and testing splits from the original training set. This is repeated a certain number of times, for instance five, and then AUC scores are calculated for each run. From the AUC of all these runs we can then take the average AUC score from the five runs and have a single value that tells how good of an object classifier GLIMPSE was able to learn.

In my experiments I perform repeated random sub-sampling on each object classes training set. To perform random sub-sampling validation, an objects training set is randomly split into training and validation splits, the size of each being $\frac{2}{3}$ training, $\frac{1}{3}$ validation. The flow of the algorithm is described below:

1. Repeat for five runs.
    (a) Split original object class training set into $\frac{2}{3}$ training and $\frac{1}{3}$ validation splits.

(b) Train GLIMPSE on the training split.

(c) Evaluate the GLIMPSE model using the remaining validation split.

(d) Plot the ROC curve on this run and calculate the AUC score for this run.

2. Calculate the average AUC from all five runs.

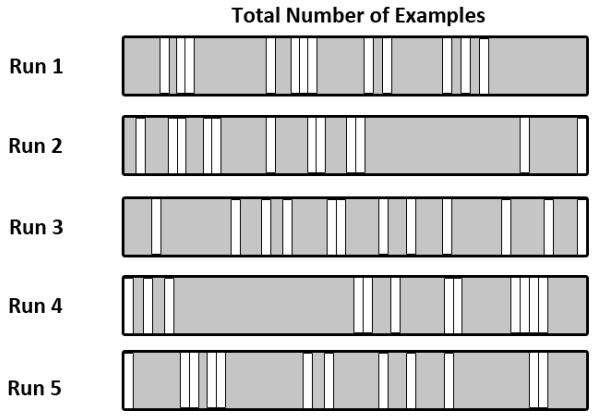



Figure 3.7: Visual example of cross-validation performed for the evaluation of my system. Gray areas represent the training split while the white areas represent the validation split for the round.

After all five runs have been performed, I then have an average AUC score which predicts how well the learned object classifier from GLIMPSE should perform on new data. This is used to help evaluate not only object recognition performance, but also which features from GLIMPSE give the best classification performance.

### 3.5.4 Detection Evaluation

Object detection in my system consists of ranking and evaluating a set of crops out of a series of test images. In our test set we use 100 images so the problem then

becomes the evaluation of a set of crops from 100 images. For each image used in the test set I made sure that there is at least one annotation that represents the object class that we are testing for. Evaluation of this task breaks down to a simple classification problem that is very similar to an information retrieval problem from the database field. Essentially this detection task becomes a ranked retrieval task once all the crops have been evaluated by the SVM. I can rank all the crops by their SVM decision value's, from highest to lowest, and calculate a precision-recall graph on this data.

By sorting the list of detection locations by each detection's decision value, I calculate a running precision-recall score that uses the decision value of each detection as a threshold in the graph. This generates a very detailed plot from which I can see, in general, what level of system performance I should get depending on how much relevant information the system retrieves.

This metric was used in the original StreetScenes work but is also a key metric used in the Pascal Vision Challenge [6]. In the Pascal challenge they take an interpolated precision recall graph of all the detection results, which is easier to visualize and understand, but evaluates at a much coarser grain.

To calculate the precision and recall values we need to know which detection's are true positive and which ones are false positive. The Pascal challenge gives fairly straight forward guidelines as to what to consider a true or false positive classification. For a true positive classification of a crop I calculate the intersection of the ground truth , A in equation 3.1, and the crop's bounding box, B in equation 3.1. Then the evaluation scripts find the area of the intersection between the ground truth and bounding box and divide that by the area of the union of the ground truth and bounding box. A true positive detection occurs if equation 3.1 is greater than or equal to $\beta$.

$$\frac{area(A \cap B)}{area(A \cup B)} \geq \beta \tag{3.1}$$

In my experiments, $\beta$ is set to .5 which is the same value used by Bileschi and in the Pascal challenge [1, 6]. I found that changing $\beta$ to be low, $\beta = .35$, did not make a huge difference in the detection tasks results.

# Chapter 4 Results

The work described in this thesis consisted of building recognition and detection algorithms based on Bileschi's StreetScenes project. My focus was on checking for comparable results as well as testing which features work best when using GLIMPSE to perform recognition and detection tasks on natural images. I looked more deeply than Bileschi at how different prototypes and combinations of features may affect performance. My results contribute to the overall knowledge of how to best use HMAX systems for recognition and detection tasks.

## 4.1 Object Recognition Task

I performed a series of object recognition tasks looking to see how well GLIMPSE's HMAX models do at classifying image crops as containing an object of interest or not. I discuss the experimental setup in section 3.2 above.

### 4.1.1 Object Recognition at C1

I performed a series of object recognition experiments for the car, pedestrian, and bicycle object classes using the method described in Section 3.5.3. For this first experiment I used my scrubbed training sets to extract features from the C1 layer in GLIMPSE. The number of features extracted from each 128x128 example was over 18,000, due to all the scale and orientation information that is retained in the feature vector.

Figure 4.1 and Table 4.1 shows the ROC curves and average AUC for five random sub-sampling validation runs for each object class. In Section 3.5.3, I

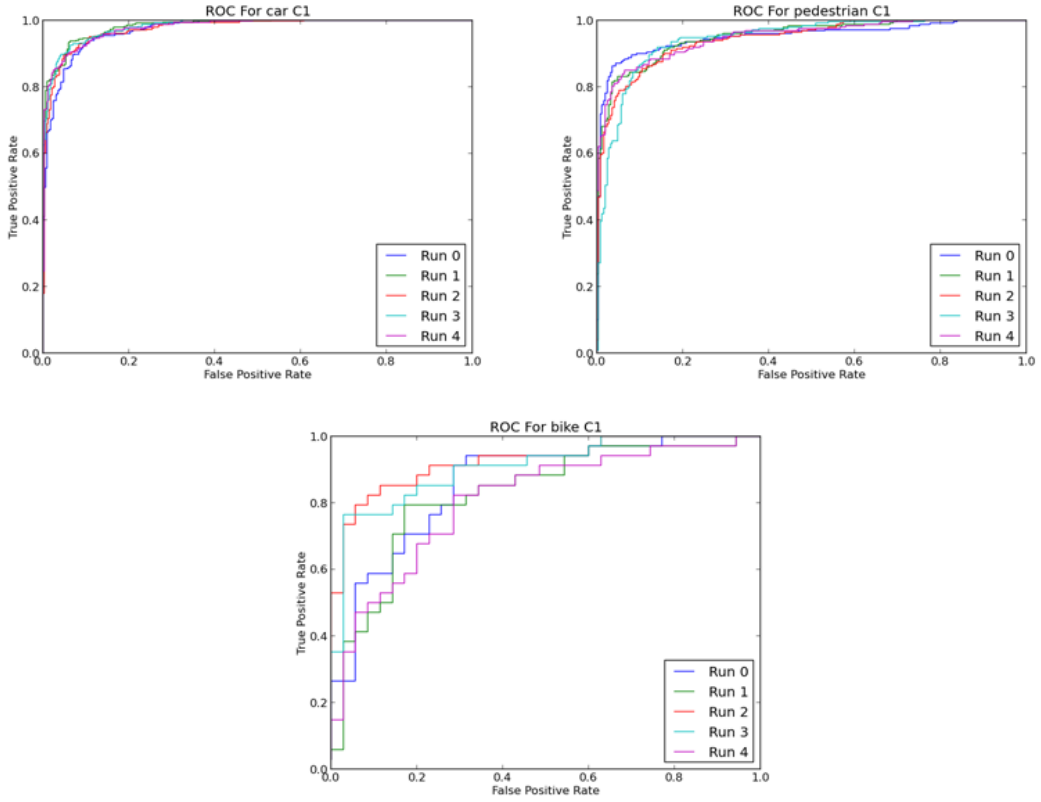



Figure 4.1: ROC Curves for five cross validation runs of car, pedestrian, and bicycle classes using features from the C1 layer of GLIMPSE. Each curve in the graph shows the plotted true-positive and false-positive rate of the object classifiers on the testing split described in section 3.5.3.

discuss the details of how I generate the ROC curves and their use in the evaluation of my classification results. The ROC curves show the classification performance of GLIMPSE on the car, pedestrian, and bicycle object classes (see Figure 4.1).

Looking at Figure 4.1 and Table 4.1, out of the three object classes, GLIMPSE performs the best with the car class, achieving the largest average AUC and the most consistent ROC curves for each of the five experimental runs. For the Car class this could easily be attributed to having the largest number of training examples available. Classification performance on the other two classes, was rather high as well, with the worst performing class being Bicycle, with an average AUC of 0.883.

| Model | Avg. AUC C1 | Bileschi C1 |
|---|---|---|
| Car | 0.978 (0.0017) | 0.99 |
| Bike | 0.883 (0.006) | 0.908 |
| Pedestrian | 0.957 (0.0004) | 0.983 |

Table 4.1: Average AUC for five random sub-sampling validation runs using C1 layer features. Values in parenthesis are standard error. Bileschi does not state the standard error in his results.

Bileschi's results showed similar average AUC results for the car class, but differing results for pedestrian, and bicycle classes. In Bileschi's thesis, he noted that for C1 object recognition, Pedestrian was the worst performing class with an average AUC of .908 [1], while my results show Pedestrian performing second best with an average AUC of .954. For the classification results of the Bicycle class, Bileschi was able to get an AUC of .98, but GLIMPSE gets an AUC of .883.

It should be noted that my system was only able to get comparable performance to Bileschi after I manually removed clutter from the negative examples in the object class training sets. Originally when performing object recognition with the non-scrubbed sets, my system was only able to achieve around .95 average AUC when classifying cars. After the scrubbing of the sets the average AUC rose to .976 which is much closer to what Bileschi was able to achieve. The differences in AUC results most likely are due to parameter differences in the underlying HMAX model of StreetScenes compared to GLIMPSE. Unfortunately the exact parameters of Bileschi's HMAX system are not noted in his thesis.

### 4.1.2 Object Recognition At C2

I performed another round of recognition experiments using the experimental method described in section 3.5.3. Instead of using features from the C1 layer of GLIMPSE, I extract features from C2 and compare these features to the features used at C1. I then did experiments that combined C1 and C2 features to see if a combination of features from different layers help classification performance

in GLIMPSE.

| Class | Avg. AUC C2 | Bileschi C2 |
|---|---|---|
| Car | 0.970 (0.0021) | 0.978 |
| Bike | 0.859 (0.0243) | 0.96 |
| Pedestrian | 0.913 (0.0059) | 0.944 |

Table 4.2: Average AUC for five cross-validation runs at C2 using 1000 imprinted prototypes for the three object classes car, pedestrian, and bicycle. Values shown in parenthesis are standard error.

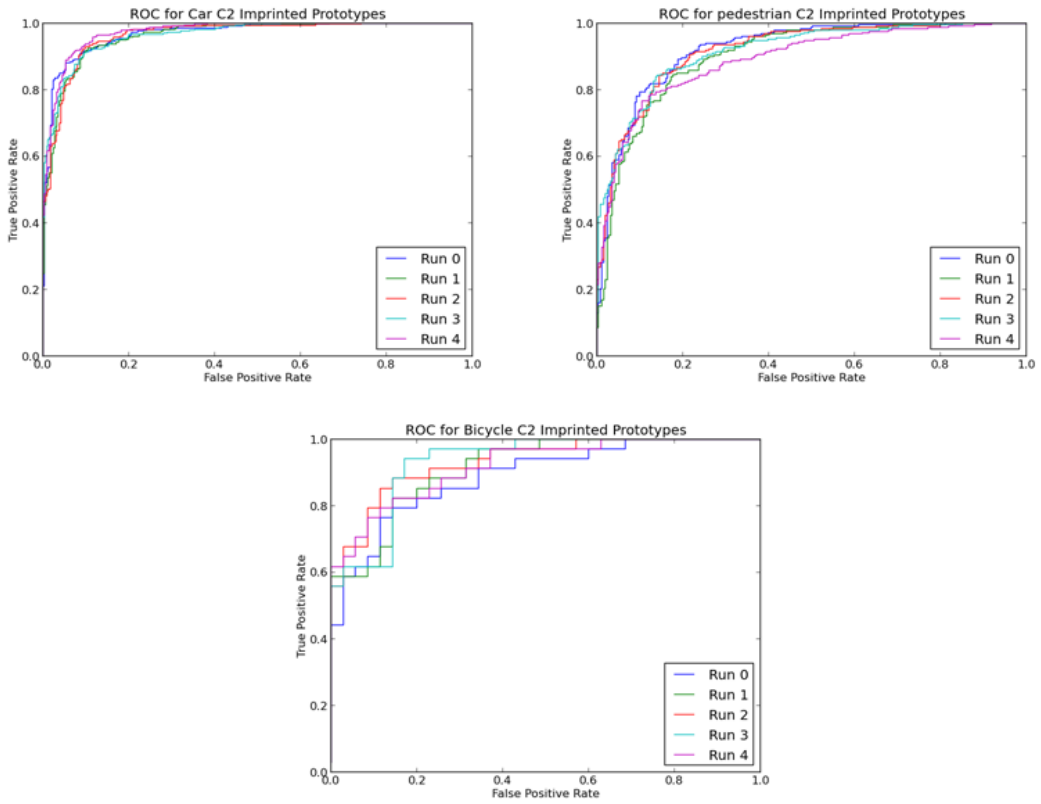
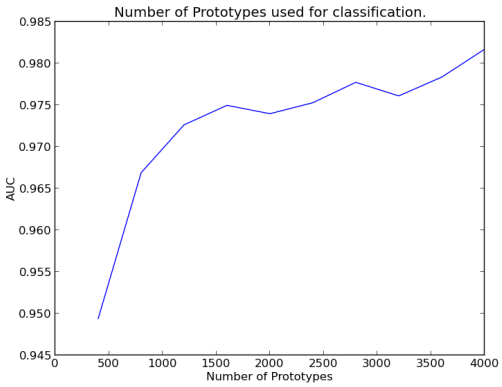
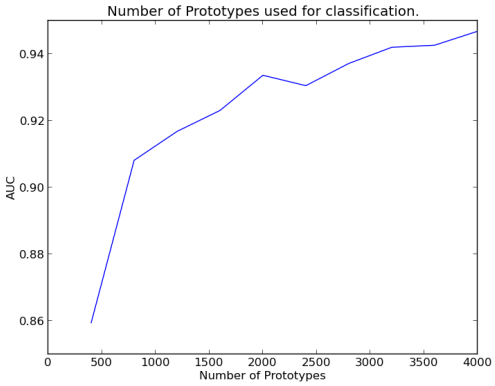



Figure 4.2: ROC curve showing recognition performance of five-fold cross-validation on all three object classes using 1000 imprinted prototypes at the C2 layer. Each curve in the graph shows the plotted true-positive and false-positive rate of the object classifier's performance on the testing split described in section 3.5.3.

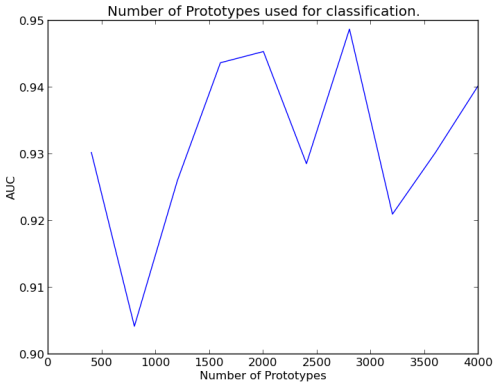
The AUC results of the C1 layer and C2 layer models, seems to show that there is some loss in classification accuracy using C2 with 1000 prototypes. This was




(a) Car Object Class




(b) Pedestrian Object Class




(c) Bicycle Object Class

Figure 4.3: Average AUC for object classes verses the number of imprinted prototypes learned at C2.

noticed in Bileschi's results [1] and could be due to C2 being a highly generalized feature vector, where as C1 still contains raw scale and orientation information in the feature vector. I should note that even though the results for C1 performing better have been noted in previous papers, I did further experimentation by seeing how the performance of the classifier changed as we increased the number of prototypes learned at S2 and noticed from Figure 4.3 that there is an increase in classification performance for 2000 or more prototypes learned.

## 4.2 Object Recognition C1+C2

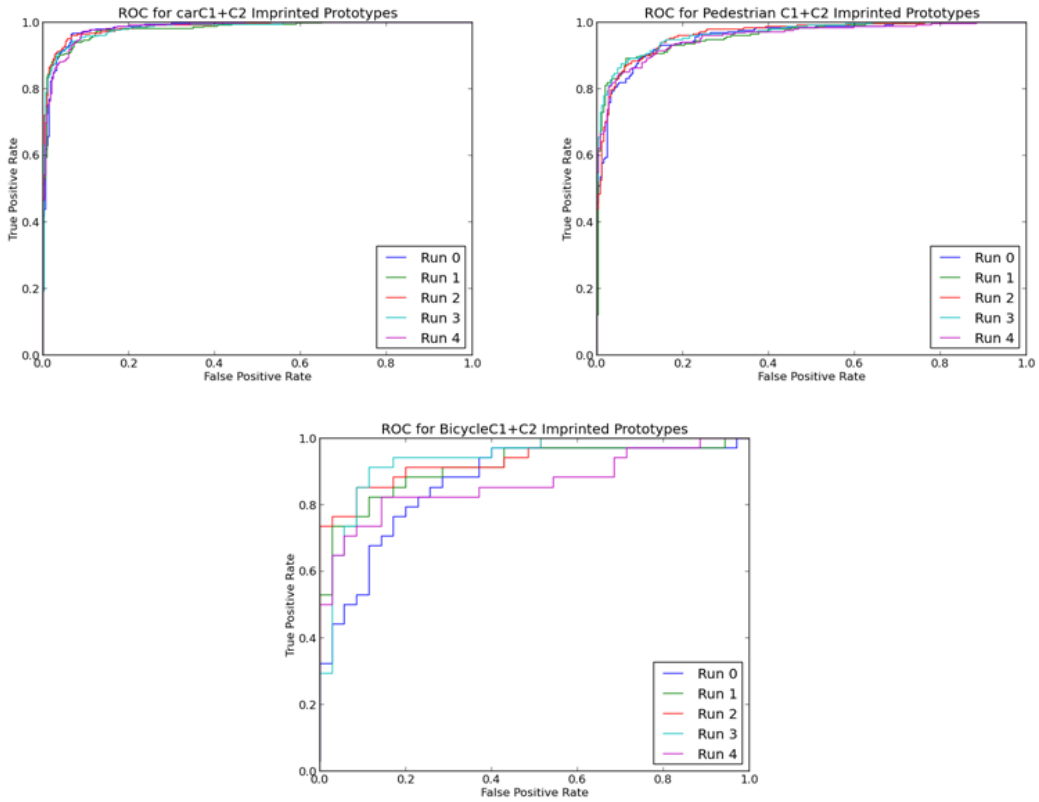



Figure 4.4: Object Recognition of Cars using C1 layer features concatenated with 1000 C2 imprinted prototypes.

By combining C1 and C2 features, there is an increase in recognition performance compared to just using C1 or C2 alone. For the car and pedestrian classes

| Class | C1 + C2 |
|---|---|
| Car | 0.982 (0.0008) |
| Bike | 0.904 (0.0162) |
| Pedestrian | 0.9602 (0.002) |

Table 4.3: Average AUC of five cross validation runs using C1 features concatenated with C2 features. An increase is noted in the AUC scores for all prototyping schemes compared to just C2 alone (Table 4.2).

there is only a boost in classification performance by about less than a percent, while the bicycle class sees an improvement of nearly 2 percent. While there is increased performance combining these features, the downside to using this feature combination is an increase in computational overhead by having to learn and process S2 prototypes in GLIMPSE. Again the car object class has the best performance and pedestrian is second best.

### 4.3 Object Detection At C1

For the object detection experiments (see Section 3.3), I focused on the car object class due to limited occurrences of the other object classes in the test set. Cars are the most frequent object in the database since all of the images were taken on streets during busy times of the day [1].

The precision-recall graph in Figure 4.5 shows how well the detection system did at finding the objects of interest out of the set of possible detection's found in all of the test images. I noticed from the graph that the detection results are far below those reported in Bileschi's thesis [1]. My detection system is only getting about 20 percent precision at about 10-15 percent recall and then drops off dramatically. Additionally the system is only finding about 50 percent of the objects in the whole test set, which is rather disappointing considering Bileschi's system could find all objects eventually [1].

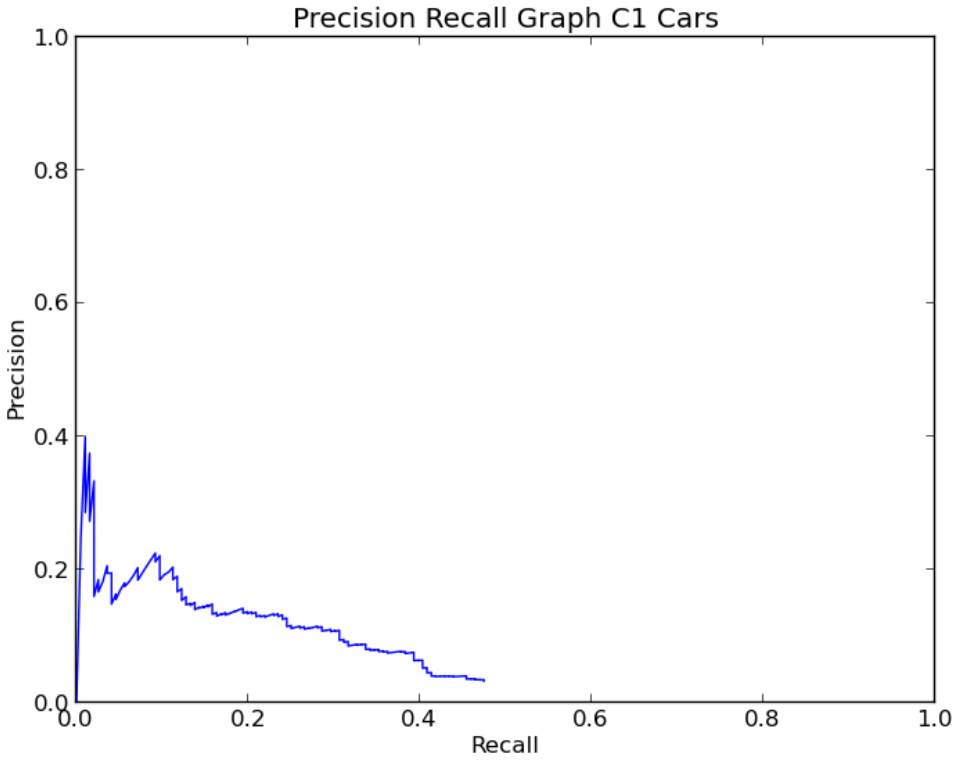



Figure 4.5: Precision-Recall graph for detecting cars in 100 images using C1 layer features.

### 4.4 Object Detection At C2

I performed another round of detecting cars in the test set, but this time extracted features from GLIMPSE at the C2 layer using 1000 imprinted prototypes. Figure 4.6 is the resulting precision-recall graph from the experiment. What is interesting about the graph is that there is a loss in both recall and precision, which indicates that not only is the system doing poorly at classifying crops that are extracted from the images, but that most of the detection's that are being returned do not even contain the actual objects.

From the graph we see that we are only finding about 30 percent of the total objects and at most getting 10 percent precision for about 5 percent recall. These results are not totally surprising though, considering the results obtained when

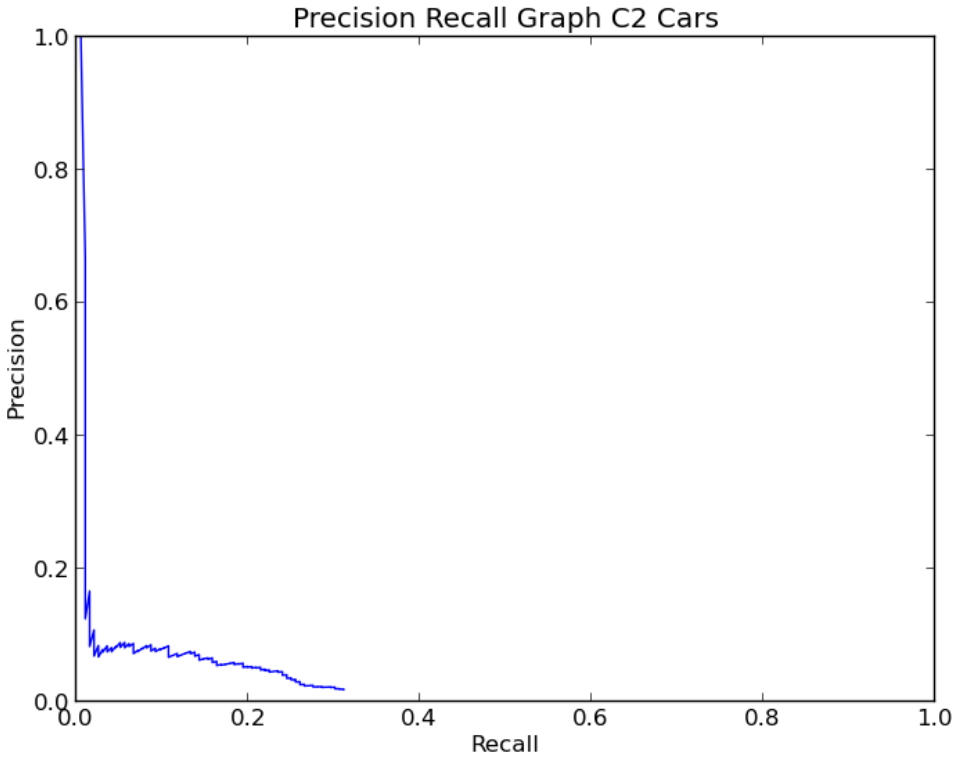



Figure 4.6: Precision-Recall graph for detecting cars in 100 images using C2 layer features. C2 features were generated using 1000 imprinted prototypes.

classifying crops using C2 features alone in as seen in Section 4.1.2.

# Chapter 5 Discussion

## 5.1 Recognition and Detection Discussion

In Chapter 4 I presented results on how well my object recognition and detection algorithms performed using GLIMPSE's HMAX implementation. From the results I was able to look at which layers in the HMAX network would be best for extracting object information. This contribution distinguishes my work from that of Bileschi, who focused more on trying to detect objects in images rather than what feature layers best work for HMAX [1].

### 5.1.1 Object Recognition Experiments

My object recognition experiment results showed comparable AUC results to those of Bileschi's thesis when using my manually scrubbed training set on the Car object class [1] for both C1 and C2 layers (Tables 4.1 and 4.2). The data indicates that GLIMPSE builds models that are similar to those of the StreetScenes system for cars, and builds a better classification model for the pedestrian class than reported by Bileschi. For the Bicycle class there is a loss of classification performance by about 10% from Bileschi's .98 average AUC to my systems .88 average AUC [1].

Between the recognition experiments extracting C1, C2 and a combination of C1 and C2 features, I found that concatenating the C1 and C2 layers performed the best for all three object classes, as seen in the results in Tables 4.1, 4.2, and 4.3. However this came with the downside of extra computational overhead in terms of image processing times due to having to learn and process C2 prototypes. C2 performed fairly well but was a few percentage points less than C1 for each of the

object classes.

What is interesting from these results is that it would seem C1 is more invariant and robust to orientation and scale changes than C2 (using 1000 prototype features). This most likely is due to C1 still containing orientation and scale information while C2 has tried to generalize the features to higher level representations by learning shape prototypes. My results seem to show that GLIMPSE is comparable to that of StreetScenes for the Car object class and performs better than StreetScenes on the Pedestrian object class. The Bicycle object class in my experiments perform the worst, but I attribute this to having fewer training examples than what Bileschi had for training. The Bicycle class had in total in my training set 102 positive and negative examples, and Bileschi had about 60 additional positive and negative examples [1]. Further examples of bicycles would no doubt help to improve model performance.

Both my results and those of Bileschi [1], seem to show that the C2 features alone lose some invariance in the object models when applied to new examples. I found that the only solution to gaining similar or better performance at the C2 layer alone is to learn thousands of prototypes, upwards of four thousand (Figures 4.3). This however again increases the computation time of GLIMPSE.

The results from testing the classification performance of GLIMPSE using different numbers of prototypes ( Figure 4.3 ) differ somewhat from the conclusions of Bileschi for his recognition experiments using C2 features. It appears that if enough prototypes are learned, recognition performance will not degrade as previously noted in Bileschi, and in fact may improve as more prototypes are learned. This could then mean C2 would produce better performance than C1 if enough prototypes are learned for an object class. One argument against learning so many prototypes at C2 is that there is an increase in computation time as the number of prototypes increase. The overhead happens from GLIMPSE having to learn prototype information from the training data before GLIMPSE can finish

building the feature vector and process the images to C2. This overhead may not be an issue for some tasks, but having to process images to a higher network layer and using a much larger number of prototypes will increase processing times. More experimentation will likely be needed to see how much of an overhead on average could be expected when using C2 features verses C1 features. The final recognition experiment I performed examined GLIMPSE's performance when combining C1 and C2 layer features into one feature vector. Figure 4.4 and Table 4.3, the system does gain some performance over just using C1 or C2 by themselves. However what was noticed in GLIMPSE is only a marginal increase in model performance over just using C1 and the extra computation time needed to build this vector for every extracted crop does not seem to make the use of combined layer features viable for close to real time tasks. It seems that the boost in performance mostly comes from the C1 layer doing most of the "heavy lifting" because it contains the most rich feature information which is supplemented by the imprinted prototype features of the C2 layer.

### 5.1.2 Object Detection Discussion

For the object detection task I found worse performance than that of [1, 15] using both C1 and C2 features. Mutch and Lowe [16] noted that this task is highly dependent on the construction and implementation of the neighborhood suppression algorithm [16]. I experimented with different suppression methods, including zeroing out whole regions or setting larger parameters in the Gaussian to help make the suppression algorithm remove more redundant crops. I found through my parameter experimentation that the best way to handle this task was to set the neighborhood to the size of the bounding box plus some extra amount so as to reduce the number of crops that are overlapping in a detection region. However, as seen in the precision recall graph, this still did not produce great results since the system is unable to find all objects across the test set and has

very poor precision at all recall levels.

Even with a somewhat tuned suppression algorithm, I still found poor detection results compared to Bileschi. His results showed 50 percent precision for 50 percent recall [1], when my algorithm achieves 20 percent precision for 10-15 percent recall using C1 and only about 10 percent precision at 20 percent recall. This is fairly disheartening because the object models and methods used should be comparable based on our recognition results. However there can be many reasons for the poor detection performance of my system compared to that of Bileschi. What I found was that for many of the images that contained car the system tended to detect bounding boxes around or near these objects (Figure 5.4). In fact most of the bounding boxes found by my algorithm clustered around the car objects. Unfortunately most of the cars in the images were not annotated and thus counted as false-positive detections against my system when I evaluated my results. This is an unfortunate symptom of the database when used to evaluate detection algorithms and was noted in Bileschi's thesis [1].

## 5.2 Computational Issues

Both StreetScenes and my system run into the problem of how to efficiently handle an exhaustive search task when trying to detect object locations. This has been a noted concern for object detection in general, not just with HMAX itself [1, 13]. Some computer vision algorithms, such as SIFT [14], have been able to avoid the need to perform a search at every position and scale of the image, but SIFT does not show as much robustness to invariance in object orientation as compared to HMAX-based recognition [18].

To detect objects in natural settings, without any prior expectation for their locations, a detection algorithm has to perform the sliding window technique, mentioned earlier in the thesis, over the input image at many different positions and scales. Sliding a window over the image is a fairly standard practice for almost

all object detection algorithms proposed [1, 13, 16].

The biggest hindrance to using a sliding window for detection is that it has to search over the entire image and rerun the same set of tests for each extracted window from an image. The search technique clearly turns into a brute force exhaustive search task that can explode in computation time as the number and size of images increase. A paper by Lampert et al. [13] on object localization mentions that sliding a window in a small image over all positions and scales, in their example 320x240, can yield up to two billion crops that would have to be examined. My system does not extract crops at such a fine-grained level, but Lampert et al.'s calculation shows how much potential information there is to process per image.

To put it in the context of my object detection experiments, if it takes a little less than a second to process one crop from an image to the C1 layer, and the detection algorithm cropped out 500,000 possible locations from the input image, that would end up taking nearly 5 days to fully process all the crops in that one image alone. Granted the previous example is on the extreme end of what would be extracted by my system, but it shows how bad of a computational issue we risk with this brute force technique. While developing the initial version of my system I ran into this exact problem were it took 20 to 30 minutes to process each image due to having to process each individual crop through GLIMPSE.

To combat the long compute times, I employed an easy optimization into the detection algorithm that processes the entire input image through GLIMPSE first and then extract "crops" out of the feature vector that GLIMPSE returns. The algorithm can then extract sub-sections from the image vector which correspond to crops in the original input image. This optimization then brings the running time down to hours in the worst case, rather than days. In my experiments I set a lower bound on the smallest crop the algorithm can take out of an image, 128 by 128 pixels, so this helps reduce the search space.

Bileschi argued that one way to combat the long running times of the detection algorithm is to just perform the detection algorithm by using fewer scales and a larger spacing between crops, arguing that this small change should not hinder overall performance [1]. However I have not been able to verify these claims concretely. When building the detection system and trying to figure out a good number of scales and step size for extraction, I noticed that my system can shave off a great deal of time by decreasing the number of scales at which the system extracts crops at. Increasing the step size between each extracted crop saves some time when processing an image, but not as much as lowering the number of scales does. More work will need to be done to figure out what number of scales and steps are optimal for detecting objects.

### 5.3 Dataset

The StreetScenes database is a decent resource for computer vision experiments and a lot of hard work went into its creation [1]. Image collection and annotation can be a labor intensive and time consuming task. Being able to go through all images and label objects of interest alone takes much time and effort. In addition unless there are clearly defined rules for annotations, issues of ambiguity on when to annotate an object in an image becomes a problem. StreetScenes was no exception: such issues arose in the process of taking photographs in the Boston area and annotating the photos with object, textural, and contextual information [1].

For my experiments on the object recognition task, the dataset is fairly easy to work with and I can pull any object I need from it to build training and testing sets for all object classes. I found however that for the detection experiments, the dataset was much harder to use and ran into some pitfalls. StreetScenes, it turns out, is a fairly hard dataset to test object detection algorithms on since there is much clutter in the background of these images (see Figure A.1). This background

clutter tends to throw off the detection system and is an issue resulting in clutter appearing in the training sets for the objects.

### 5.3.1 Training Set Clutter

Clutter is the addition of background visual information that appears in the training sets for the object class models. Clutter in the training sets for our objects is hard to remove due to needing to extract positive and negative crop examples from the larger StreetScenes database. Examples of some crops from the car classes training set that contain clutter can be seen in Figures 5.1 and 5.2.

Figure 5.1: Examples of clutter present in the positive examples of the car object class training set. Clutter in the positive examples happens when background information takes up portions of the image, such as trees, buildings, or when other classes appear in the image. This last issue is seen with the two pedestrians who are walking by cars. Removing these cluttered examples appears to help the classification model, but becomes a very labor intensive task when dealing with thousands of examples.

In my thesis I attempted to reduce the amount of clutter present in all training sets, but only was really able to focus on the negative sets for each object class. The reasoning behind removing as much clutter as possible from the negative examples was for two reasons: (1) By removing examples of cars from the negative class it should help reduce the confusion on the model as to what class a car really falls into. (2) It is much easier to find and delete negative examples that contain the

Figure 5.2: Examples of clutter present in the negative examples of the car object class training set. Clutter in the negative examples happens when parts of the car object class are still present in the negative set of examples. Removing these cluttered examples appears to help the classification model, but becomes a very labor intensive task when dealing with thousands of examples.

object class than trying to determine if an object part will hinder the classification model or not. Bileschi's training sets appear to run into the same problems with clutter, but he does not appear to attempt to remove clutter as far as I can tell from either the negative or positive examples.

When dealing with training sets that contain thousands of examples, a lot of time and effort has to go into looking at each individual image to effectively remove clutter. Thus to remove hundreds of example images with clutter takes hours per training set, which is why I focused on the negative examples due to time constraints. Unfortunately even with the manual scrubbing of the negative training examples, I was not able to eliminate all clutter, only a majority from the negative examples. For the car class close to 200 negative examples were thrown out due to containing clutter. More work will need to be done in the future to remove more clutter from the negative examples as well as the positive examples.

What was apparent from removing most clutter from the negative class examples was an increase in recognition performance, though a modest one at best. The results reported in my thesis are that of models trained using the cleaned training sets with reduced clutter in the negative examples. The initial performance of my system for the recognition task when I was debugging, had car classification at

.95 average AUC without any clutter removed, compared to the .978 reported in Table 4.3. As shown in my results the increase in performance was not as big as I was expecting but still showed how clutter can cause confusion in the model. If more cluttered example images are removed from the training sets then it should be possible to increase recognition performance even further to fall in line with that of Bileschi's work [1].

### 5.3.2 Testing Set Annotations

One issue that was apparent when testing my detection system is that there are not very many annotations of objects in the original StreetScenes database. According to Bileschi this was due to strict rules for when to annotate objects and when not to. Bileschi attempted to only annotate objects that where not occluded by other objects or clutter and objects that were not next to other objects [1]. Even with these rules in place there was ambiguity between annotators as to when it is best to annotate an object and when not to and this is a draw back to the current annotations [1].

I found that the average number of annotations for an object such as car in StreetScenes is only about 1.9 annotations per image. This means that on average there will be two labeled cars per image even though by visually examining the images for detection quality, there are typically more than two cars present per image (see Figure 5.3). The lack of annotations seem to be contributing to my system's low precision-recall scores because the system is finding "false-positive" locations based on the annotations, but these detections really are not actual false positives. This makes the detection task using StreetScenes original object annotations very tough because the object model that GLIMPSE has learned will always try to return the location of cars whether the cars have an associated annotation or not. Because of the lack of annotations, I manually selected 100 images as a test set and attempted to annotate extra instances of cars. The


Figure 5.3: Example of a test image with only one car annotated, while there are clearly more cars in the image. For many of the annotations in the StreetScenes database, there was much ambiguity as to what to actually label as an object class or not [1]. In this case only the least occluded vehicle was chosen leaving any other detection's that find the other cars as false positives.

hope being that with more annotations, there is a greater chance of reducing the incorrect "false-positive" detections that was seen with the original annotations.

I attempted to rerun the C1 layer detection task using the new annotations I created, however I noticed a reduction in the recall of my system and no real increase in detection precision. The format of the new annotations are different than that of the original StreetScenes database so I had to rewrite some of my annotation code. There may be some error in this portion of my system that needs to be corrected to be sure there is not some software issue lowering the system's performance. The lowering of the recall amount though is somewhat reasonable since with the new annotations there is an increase in the number of objects for my system to look for. If the original test set when performing detection using C1 only gave us 50 percent recall at most then it makes sense that if we double the number of objects to return, the same system may have 25 percent recall given

the same experimental parameters. This appears to be what might be happening if there is not a bug in the system.

Looking at Figure 5.4, many of the detections my system makes appear to be clustered around cars in some of the test images. From a qualitative analysis, visually inspecting the detection results on the images, this seems very good because the top detections returned by the algorithm are indeed cars. However from the quantitative analysis done using the precision and recall graphs, Figure 4.5, a different story emerges. From the precision-recall graph, it appears that detection algorithm does very poorly on the test data. I attribute this discrepancy partially to the lack of annotations in the original StreetScenes dataset. The lack of annotations of objects is noted by Bileschi, with the future work of StreetScenes having more annotations or object classes [1] to help counter this issue with the dataset.

Having said that about the original StreetScenes annotations, that doesn't mean that all the false-positive detections returned by my system are not actual errors. There are unfortunately still many instances where actual false-positives are being labeled as a car. Figure A.3 shows a few test images that have a majority of their detections being false positives. This is not completely surprising due to the background clutter in the test images potentially having extracted features that are similar to that of the learned car model.

Another reason my system may be losing performance is that the local neighborhood suppression algorithm (described in Section 3.3.1) may not be suppressing image regions well enough. What appears to happen in some cases is that when suppressing the region around a global maximum detection (the potential object location), if the detection crop is small, it leaves more room for duplicate detections in that same region to be returned as another possible object detection. Looking at Figure A.3, there are many detection windows that not only find incorrect locations, but also overlap with smaller windows. Unfortunately even with the images that have detections clustering mostly around cars, the issue of

overlapping larger with smaller detections happens as well (see Figure A.2 and 5.4). Most of these issues stem from needing to determine adequate parameters for building the suppression neighborhood, for instance neighborhood size as some function of the detection location size.

In the end I cannot easily determine if the poor performance for the detection system results are mainly due to a dataset lacking annotations or a combination of the lack of annotations and a non-optimal suppression algorithm. There are many parameters used in the original StreetScenes experiments that were not given in the StreetScenes publications [1]. It becomes apparent that to increase system performance for object detection, additional experiments will be needed to determine a set of tuning parameters that work best to increase performance in the suppression algorithm.

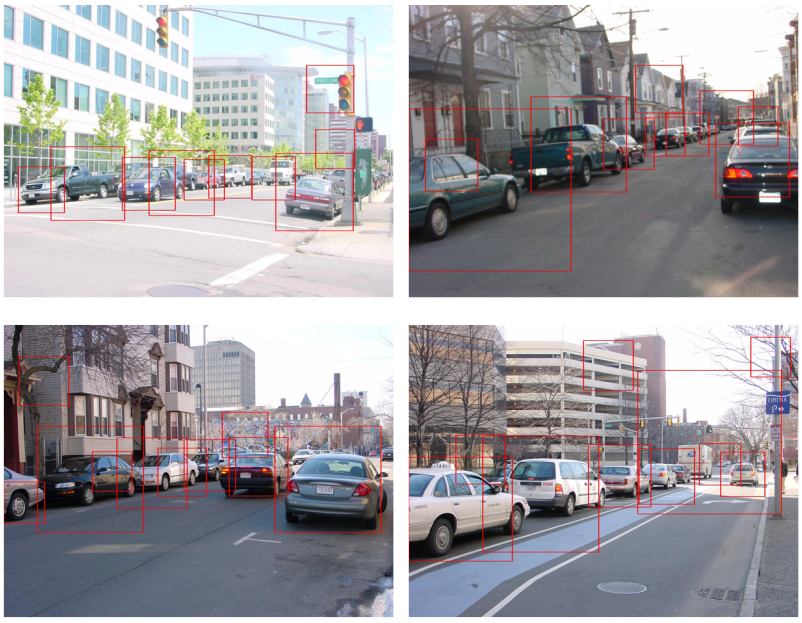


Figure 5.4: Results for the detection experiments for car using 100 images from the StreetScenes database. Consistent detection of cars in the image with a few stray detections.

## Chapter 6 Conclusion and Future Work

In my thesis I focused on a few key problems about how to use HMAX-like systems for object recognition and detection in natural settings. First, my thesis explored whether my GLIMPSE-based object detection system could replicate the results of Bileschi's StreetScenes system. Second, I explored what features from GLIMPSE gave the best performance. Finally I created new training and test sets used for experimentation by removing clutter from each of the object classes training sets and hand annotated 100 test images. Unfortunately due to time limitations I do not have detection results using the new annotations. My system potentially has a bug when parsing the new annotation files that still needs to be fixed so future work will need to address this issue.

The results of my thesis are somewhat mixed. Regarding the first question of comparability, I found that for the recognition experiments we could get similar or better results than Bileschi's system. This was good because it showed that GLIMPSE produces similar object models to that of the ones used in StreetScenes. However the detection code that was developed to use these learned models did not perform as well. There were many factors that could have played a role, such as variances in the HMAX models, suppression algorithms, and annotations for the test set.

From the results answering the first question, I was able to answer the second with a little bit of extra work. Generally I found that when learning the object models, one can pull features from either C1 or C2 or even a combination of C1 and C2 and get fairly good recognition performance. This was the general trend

for all three object classes that I tested.

C1 was generally seen as the ideal choice by Bileschi [1], though my system was able to get similar or better results by using C2 with 2000 or more prototypes, or by using a combination of C1 and C2 features. In the end C1 still appears to be the ideal layer for feature extraction since it contains the most rich feature information for all object classes, and is not too computationally expensive to process per image. C2 features do not show the same performance until they have learned thousands of prototypes, which has a high computational overhead. Combining C1 layer and C2 layer features performed the best out of all features, but again this has additional computational overhead which make these features expensive to work with.

Finally my work consisted of manually adding additional car object annotations to the test set and manually removing clutter from the negative examples in the training sets. When originally validating the recognition models, I saw lower average AUC scores compared to Bileschi when using the training sets that still contained clutter in the negative examples. I saw an increase in recognition performance by a few points after I removed these bad examples, which allowed for the GLIMPSE models to have similar performance to that of Bileschi (see Table 4.3). By using the cleaned up training sets we ended up with increased classification performance. It becomes clear that having a training set that has little clutter in the negative examples leads to improvement in building object models.

## 6.1 Future Work

Much of the work contributed towards my thesis went into the development and experimental comparison of my recognition and detection algorithms to StreetScenes. Initially I had very ambitious goals wanting to implement not only the StreetScenes code in Python, but additionally try to implement some localization step to answer the question left by Bileschi, of how to deal with the computational issues of

exhaustive windowing over a large input image [1, 15, 19]. However due to time constraints for development and testing, one must always reign in the scope of what can be done on such a project. Unfortunately due to a lack of time, I could not get to testing my detection system using the new test set annotations. Also due to the need to get the main experiments running I was not able to spend as much time fine tuning parameters in my local suppression and detection code.

Developmental extensions to my detection and recognition system would be in adding parallelization to help combat the long compute times for the detection code. The current detection process is sequential due to processing images one at a time, so an easy speed up would be to throw multiple images at the same time onto different CPU cores and have each image process at the same time. This step is essential for efficient detection of objects when using C2 features, or a combination of image features from GLIMPSE.

One of the most helpful areas for additional research would be into how to cut down on the exhaustive search problem I discussed in section 5.2. One extension could be in adding a branch and bound algorithm that was proposed in [13]. A by-product of having a robust localization step as a preprocessor to an image, is that it may allow for more accurate detection's because the system can easily throw out areas that are not likely to contain an object of interest [13]. This would be a very useful area to research as a next step and is probably the most critical.

Additional future work includes performing detection tasks for bicycle and pedestrians for both C1 and C2 layers, after creating newly annotated test sets for those two classes. One last area of work should additionally focus on seeing if feature reduction techniques such as PCA [3] or ICA [4, 11] may help increase the quality of features that are extracted from GLIMPSE, in terms of improving GLIMPSE's ability to recognize and detect objects.

## Appendix A Additional Figures and Detection Examples

The figures in this chapter show examples of the test set used for my experiments and some detection results. Figures A.2 and A.3 show examples of detection locations from my system, both good and bad. As well I include images showing the differences between detections using C1 and detection's using C2 features. Figure A.4 shows some of the differences in my detection experiments from using C1 features and C2 features.


Figure A.1: Sample test images from the StreetScenes database showing the task of finding objects in clutter.

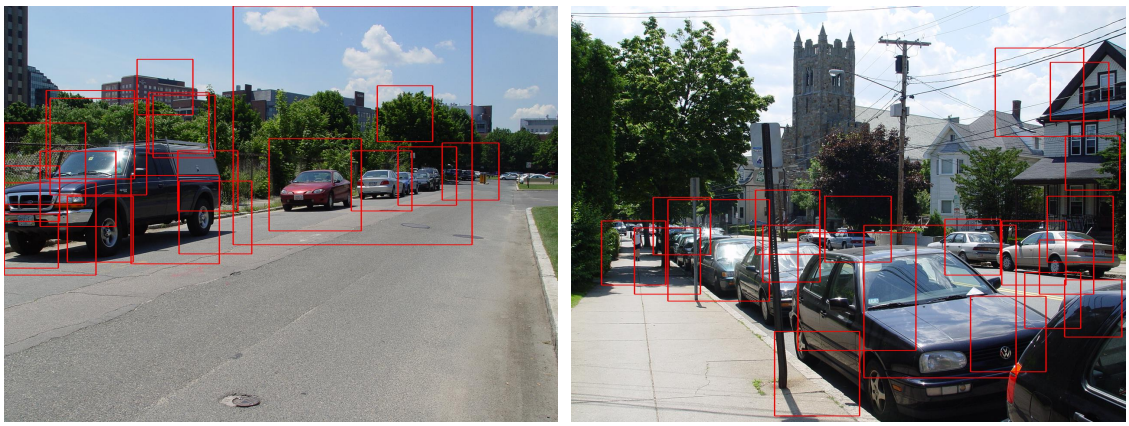


Figure A.2: Object detection results showing my system's performance on finding Cars in the image using C1 features.

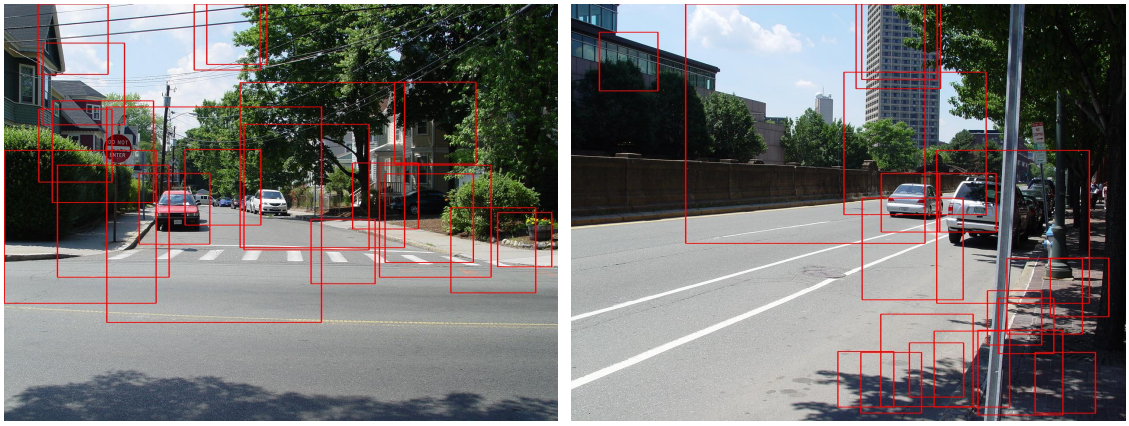
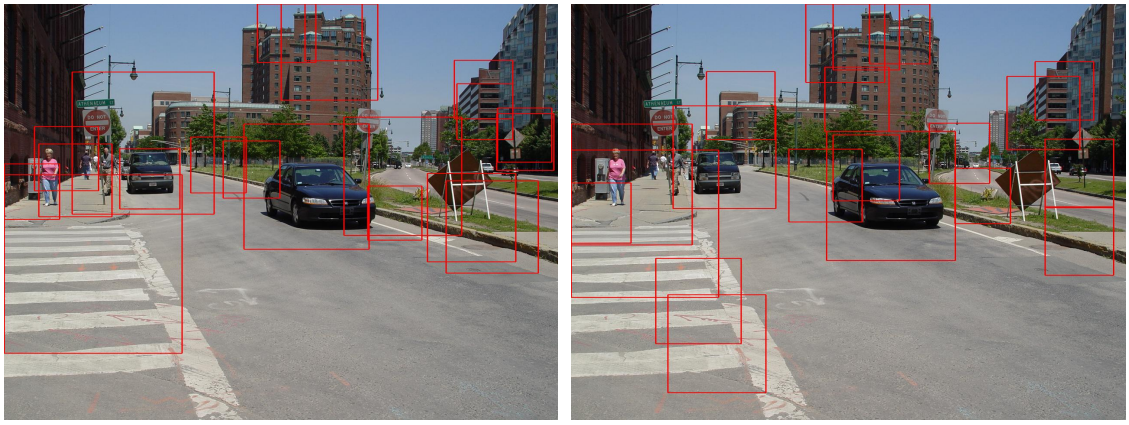


Figure A.3: Object detection results showing my systems performance on finding Cars in the image using C1 features. These images show examples from the test set where my detection system was highly confused in finding the locations of cars.



(a) Detection using C1.

(b) Detection using C2.

Figure A.4: Object detection results using C1 features and object detection results using C2 features. For the C2 features, my system gets confused on some of the paint on the street and corners of the buildings. The C1 detection results tend to get more confused on the signs and some of the pedestrians on the sidewalk.

## Bibliography


[1] Stanley Bileschi. Streetscenes: Towards scene understanding in still images. Master's thesis, MIT, 2006.

[2] Steven P. Brumby, Garrett Kenyon, Will Landecker, Craig Rasmussen, Sriram Swaminarayan, and Luis M. A. Bettencourt. Large-scale functional models of visual cortex for remote sensing. In *Applied Imagery Pattern Recognition*, 2009.

[3] Miguel Carreira-Perpin. A review of dimension reduction techniques, 1997. Technical report CS-96-09, Dept. of Computer Science, University of Sheffield, UK.

[4] Pierre Comon. Independent component analysis, a new concept? *Signal Processing*, 36(2):287–314, 1994.

[5] Jesse Davis and Mark Goadrich. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd international conference on Machine learning*, ICML '06, pages 233–240, New York, NY, USA, 2006. ACM.

[6] Mark Everingham, Andrew Zisserman, Chris K. I. Williams, and Luc Van Gool. The pascal visual object classes challenge 2006 (VOC2006) Results. http://www.pascal-network.org/challenges/VOC/voc2006/results.pdf.

[7] Tom Fawcett. An introduction to ROC analysis. *Pattern Recogn. Lett.*, 27(8):861–874, June 2006.

[8] The Center for Biological and Computational Learning. Street Scenes. `http://cbcl.mit.edu/software-datasets/streetscenes/`, 2006. [Online; accessed 1-November-2011].

[9] Kunihiko Fukushima. Neocognitron: A self organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36:193–201, 1980.

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

[11] Aapo Hyvrinen. Independent component analysis. *Neural Computing Surveys*, 2, 2001.

[12] Ron Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. pages 1137–1143. Morgan Kaufmann, 1995.

[13] Christoph H. Lampert, Matthew B. Blaschko, and Thomas Hofmann. Beyond sliding windows: Object localization by efficient subwindow search. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1 –8, june 2008.

[14] David Lowe. Object recognition from local scale-invariant features. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 2, pages 1150 –1157 vol.2, 1999.

[15] David Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 2(60):91–110, 2004.

[16] Jim Mutch and David Lowe. Object class recognition and localization using sparse features with limited receptive fields. *International Journal of Computer Vision*, 2008.

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

[18] Thomas. Serre, L. Wolf, and Tomaso Poggio. Object recognition with features inspired by visual cortex. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 2, pages 994 – 1000 vol. 2, june 2005.

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

[20] Mick Thomure. GLIMPSE - The General Layer-wise IMage ProceSsing Engine. `https://github.com/mthomure/glimpse-project/`, 2010. [Online; accessed 24-Septemer-2011].