Peekaboom: A Case Study in Human Computation

Roy Liu

August 2006

School of Computer Science
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Luis von Ahn, Chair
Manuel Blum

Submitted in partial fulfillment of the requirements
for the degree of Master of Science

Keywords: object recognition, image segmentation, interactive systems, human computation, Peekaboom

Thesis Committee

Luis von Ahn (Chair)
Computer Science Department
Carnegie Mellon University

Manuel Blum
Computer Science Department
Carnegie Mellon University

Abstract

We present Peekaboom, a web-based game that harnesses human cycles to produce training examples for computer vision algorithms. From an original motivation for the game, we go on to discuss the rationales for the design decisions behind it.

Although the primary focus will be on Peekaboom, our game is one instance of human computation, a class of algorithms that utilizes the human mind as computing module. Along with other so-called 'games with a purpose', we propose a set of features desirable for any system intended to solicit feedback from humans. Thus, in addition to addressing the specific problem of locating objects in images, we show how the ideas behind our work generalize to a broader approach for tackling hard artificial intelligence problems.

Finally, we give experimental results confirming the intuition that Peekaboom improves the discriminativeness of object recognition methods, and propose future uses for the accumulated data.

Table of Contents

CHAPTER 1

Human Computation

1.1 Motivation

Without much deliberation or difficulty, humans are able to perform tasks that computers cannot. We distinguish objects from each other without great difficulty; computers have a hard time telling cats from dogs. We think and reason; computers 'think' in a loose sense, but what they really do is run algorithms that deal with problems on a case by case basis.

One may safely say that, from the perspective of everyday life, computers are not very smart. They may be great at face detection, but that's because such algorithms [16] are specific to faces and result from years of research. A natural question arises: How to make computers smarter? One approach would be to teach them examples from our world. Gathering computer-friendly training examples, however, is time consuming. Current approaches in computer vision such as [22] and [9] are limited to a few hundred, hand-labeled examples each.

Machine learning algorithms, or those that improve their performance over time from feedback, stand to benefit greatly from increased training set size. Researchers attribute the success of face detectors, for example, to the thousands of images in databases devoted exclusively to faces. Ideally, we would like to have classes of a few thousand instances each for common things like cars, dogs, and people. How do we go about collecting them? To address the question, von Ahn [17] introduces the concept of **human computation**, where humans, as part of a larger system, carry out the hard computational tasks *that computers currently do not know how to perform.* More specifically, von Ahn proposes games as a way to carry these computations. The purpose is to get humans to participate, and, whether they intend it or not, produce desirable output. We summarize the three properties for a human computation algorithm first laid out in [17]:

1. **Incorporates the input-output behavior of humans.**
2. **Ensures, by its construction, that its human participants exhibit desired input-output behavior. One should assume that the parties involved are self-interested and, in the absence of an rewards structure, unwilling to cooperate.**
3. **Has a means to detect and discount the contributions of malicious participants.**

Figure 1.1: *The three properties of human computation.*

1.2 Organization of the Thesis

In what follows, we show how existing systems with a human interaction component are not well suited to collecting large, high quality sets of training examples meant for use by computers. We then present Peekaboom, a web-based computer game that incorporates human computation in an effort to collect training examples for computer vision algorithms.

We first identify the problem Peekaboom is intended to solve; then, we propose a solution in the form of a game; finally, we enumerate Peekabom's features and point out how they help the game fulfill its intended purpose of locating objects in images. After laying out Peekaboom's internal details, we discuss the similarities and differences among Peekaboom and three other examples of using games to collect useful information. Consequently, we derive a set of principles useful for future, similarly themed projects.

In the final chapters, we qualitatively and quantitatively evaluate the accumulated Peekaboom repository. First, we run user studies scoring the object bounding boxes derived from player actions. Then, we build and evaluate an object recognition algorithm trained on images augmented with extra information provided by the game. Thus, we lay the groundwork for a fully automated system for training computer vision algorithms – from collection to analysis to the goal of image understanding.

1.3 Previous Work

There are many examples of computer systems built prior to and shortly after the collection of games surveyed in this thesis. They include, but are not limited, to:

- The Flickr Photo Sharing Service [1] and Facebook [2] – Allows do-it-yourself annotations of photos with bounding boxes around objects and faces.
- The Open Mind Initiative [14] – Solicits users for common sense facts. Asks fill-in-the-blank questions like "Milk is a kind of _".

- LabelMe [12] – Asks the user to hand label ground truth training sets of images and the object categories contained in them.
- Wikipedia [21] – An extensive online encyclopedia compiled and maintained by a large base of dedicated volunteers.

When considering the appropriateness of the examples collected for training machine learning algorithms, each of the above approaches is insufficient. For example, Open Mind and LabelMe produce computer readable training examples under ideal circumstances – there are no safeguards or incentives for participants to enter faulty information. Wikipedia, on the other hand, has enormous amounts knowledge created and dutifully maintained by volunteers. Unfortunately, machine learning researchers don't have control over Wikipedia's output, which consists of articles written in natural language. A new system would be timely: one that combines the specialized know-how contributed by LabelMe participants with the large scale and self-managing properties of Wikipedia.

1.4 Proposing Games as a Solution

In a study done in 2000 [3], 35% of all Americans identified computer and video games as the source of greatest fun, over surfing the internet, reading books, and even watching television. Multiplying that percentage by the number of Americans and projecting into the present, one estimates that a game system could reach a potential audience of $95,000,000$.

Games are a natural fit for carrying out human computation. By design, they are fun and engaging, two qualities that work to the system designer's advantage. We summarize the key points from [18]:

- Players interact with the system's subroutines and produce valuable output; from their point of view, they're just playing a game.
- People don't play the game out of their desire to help – they do it because of the exhiliration of the gameplay and the accumulation of points. Of course, we structure the gameplay in a way that players' actions tell us information about images or abstract concepts.
- The problem of detecting and dealing with malicious participants in games reduces to the issue of cheating. We draw upon existing wisdom on the subject and develop our own ways of weeding out dishonest players and/or bots.

Since Peekaboom builds upon, and is built upon by, a body of work consisting of other games (ESP [18], Phetch [19], Verbosity [20]), we give a brief summary of each in Table 1.1. In the following chapter, we discuss the design specifications of Peekaboom specifically.

	what it does
Peekaboom	collects information about the location of objects in images
The ESP Game	attaches descriptive labels to images
Phetch	attaches descriptive sentences to images, when just a label is ambiguous
Verbosity	collects common sense facts

Table 1.1: *A one sentence overview of each game.*

CHAPTER 2

Peekaboom

2.1 Understanding Images

We designed Peekaboom to answer questions left open by the ESP Game [18]. Before discussing the former, however, some explanation is needed for the latter. The ESP Game does its job of annotating images very well. If thought of as a black box, the game takes images harvested by a web crawler, and for each image it outputs a set of annotations, or **labels**. A label, as we define it, is some word pertaining to its associated image in some way. Figure 2.1 shows a picture with some labels.

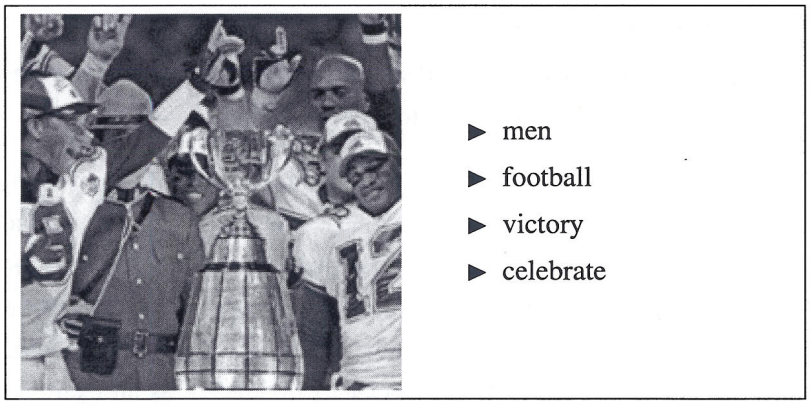

Figure 2.1: *A picture with many possible labels.*

Since its release in 2004, the ESP Game has collected 15 million labels (# images $\times$ # average number of labels per image) over $57,000$ images, which have been shown to be quite accurate in user studies. Although the ESP Game addresses the problem of attaching words to images, it does not, for example, tell

us the locations of the objects being labeled. If one of the labels to an image is 'dog', then we expect a dog to be in the image; however, the ESP Game just collects player responses in the form of words, and does not give us a clue as to where the dog actually is.

Having location information would be very useful to computer vision researchers. Whereas annotations probably suffice for content-based image retrieval systems like [4], local descriptor approaches like [11] would greatly benefit from knowing the regions contained in the object being labeled. One could, in principle, adapt the ESP Game to collect location information by cleverly showing pairs of players different sections of the same image, aggregating the kinds of responses for each part, and inferring the locations of the referred to objects. Such an approach, however, would be cumbersome – ESP is a game of words played on images, and we would like to design a game where the images are manipulated directly. The former game will still be of use to us; namely, it provides us with an annotated dataset to work off of. It also encourages us to formulate a new idea similar to the realization that *a word two players have agreed upon is a good descriptor for the image.*

We call our new idea **Peekaboom**, and we use it to locate objects in images that the ESP Game says are there, but cannot find. For its input, Peekaboom "bootstraps" off of ESP's output, presenting images with labels to players and monitoring the actions over those image-label pairs. While ESP depends on agreement between players as a way to ensure the quality of collected data, Peekaboom uses a completely different idea:

- Players cooperate with each other.
- One person, call him **Boom**, starts out seeing an image and a label. The other person, call her **Peek**, starts out seeing no image and no label.
- Boom has no means of communication with Peek, other than the ability to show her, piecemeal, parts of the image. Boom's objective is to get Peek to guess his label.

Clearly, to increase his own score and his partner's, Boom has an incentive to show Peek parts of the image relevant to the label. If the label corresponds to an object, he is effectively highlighting the regions pertaining to the object, which is what we designed Peekaboom to capture. Thus, the game works on the insight that we can gather valuable machine vision training examples *by monitoring dissemination of information from one player to another.* In the next section, we discuss the design and implementation of the game in much greater detail. Before that, however, we refer the reader to Figure 2.2 as an example of how our game can annotate scenes deemed difficult for computer vision techniques. Finally, to help the reader avoid later confusion, we include a definition of terms in the appendix.

Figure 2.2: *George Seurat,* A Sunday on La Grande Jatte. *Humans have little trouble in picking out the target objects, but object recognition algorithms become confused by a combination of occlusion (objects partially covering up other objects); texture (Seurat used a dot painting technique); and clutter (lots and lots of objects).*

2.2 Game Features

In this section, we go over the features of Peekaboom, which have a utilitarian and an aesthetic aspect. On the utilitarian side, the players' use of the game's functions give us useful information about images; on the aesthetic side, these features actually make the game more fun. The realization that the gaming aspect need not be compromised by building in collection mechanisms generalizes to the design of three other games, for which we defer discussion to a later part.

2.2.1 Basic Gameplay

A game of Peekaboom is played cooperatively between two people as follows:

- The roles of Peek and Boom are assigned randomly among the two players for the first image.
- For each image in succession, Boom starts out seeing the image and its label, while Peek starts out seeing nothing. Boom shows Peek parts of the image by clicking on it. Every click reveals to Peek the region centered on that click. See Figure 2.3 for an illustration of the two players' perspectives.
- Peek can enter guesses. Once she enters a guess that matches Boom's label, the players get points and switch roles – Peek now plays the Boom role, and vice versa.

- The game ends when either of the players fill up the score meter (Peek guesses 12 images correctly), or time runs out. Players can also pass, or opt out on, images, if both agree by hitting the 'pass' button. Passing does not make the game end sooner; it goes on until the former conditions are met.

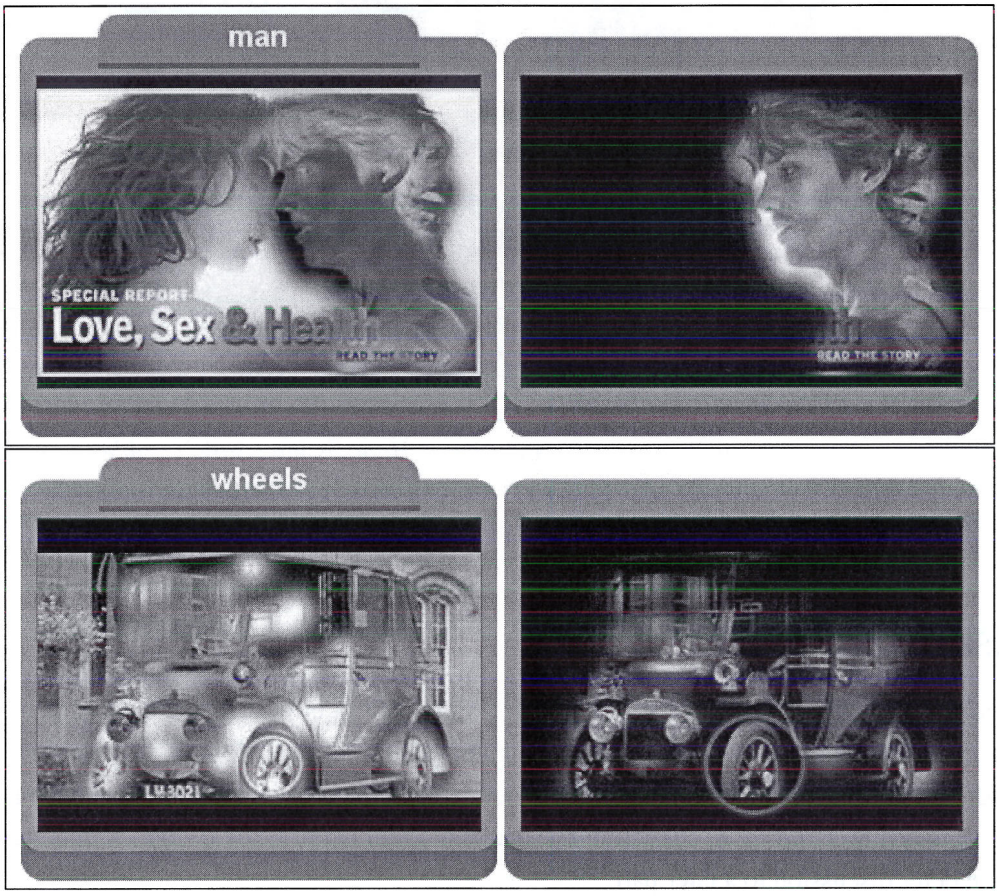

Figure 2.3: *Shown are two pairs of Peek and Boom players. The left is Boom, and the right is Peek. The parts where the image is negated are the places that Boom clicked. The corresponding regions are visible to Peek. Pings are demonstrated on the car image.*

As a further reference, we provide screenshots for the Peek and Boom interfaces in figures 2.4 and 2.5, respectively.

2.2.2 Blobs

Gaussian blobs, or just 'blobs', are the building blocks for the regions of the image revealed by Boom, as well as those regions seen by Peek. If the reader looks at Figure 2.3 closely, he will notice that the revealed

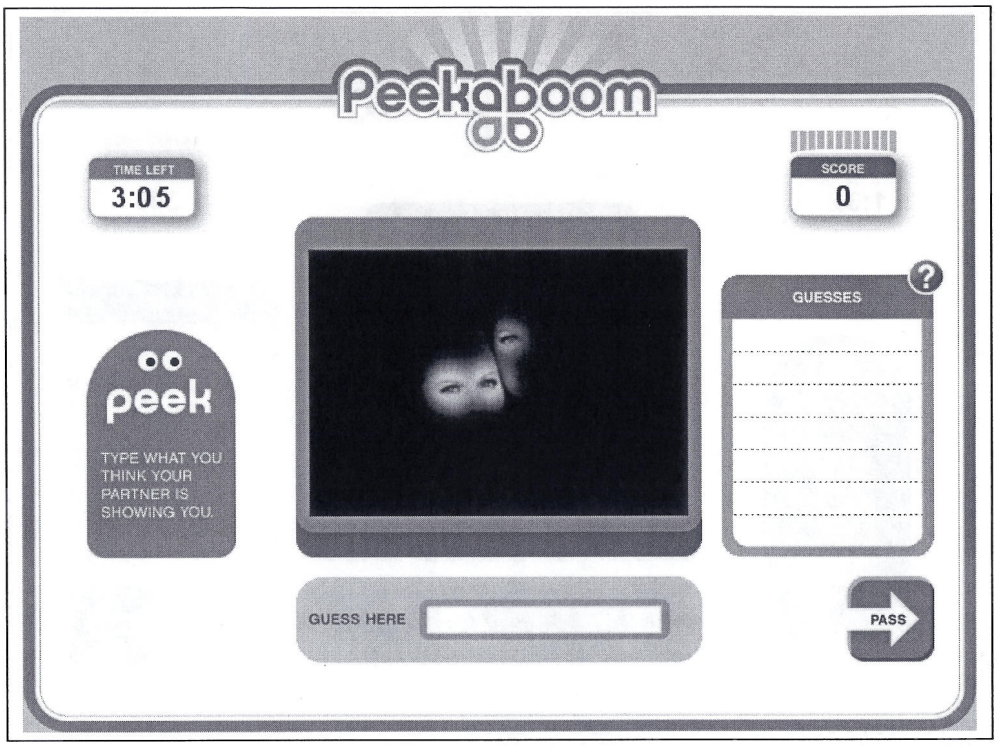

Figure 2.4: *Peek's interface – notice the guess entry tab and the list of guesses sent to Boom.*

regions look blurry on their boundaries. This is the blob effect, where, technically speaking, one image is composited onto another using Porter Duff rules with a varying compositing α value $g(x, y)$ for each point (x, y), where $g(x, y)$ is a Gaussian distribution with the blob center as its mean and the identity matrix as its covariance. Technicalities aside, blobs encode uncertainty – the closer a point is to the actual blob center that Boom clicked, the more prominently it appears on Peek's screen. Thus, we want to encourage the Boom player to *make every click count* through precise actions.

A final fact to note about blobs is that the regions clicked by Boom become negated on his screen. We chose such a marking scheme over darkening parts of the image for the reason that the played on image might be a dimly lit scene.

2.2.3 Pings

Pings give Boom a way to help Peek distinguish the target object from its context. For example, take a look at the 'wheel' image in Figure 2.3. We say that the Boom player is pinging, or pointing, to the wheel when he right clicks on a point to create purple ripples that propagate out from it. Pings are useful when the target

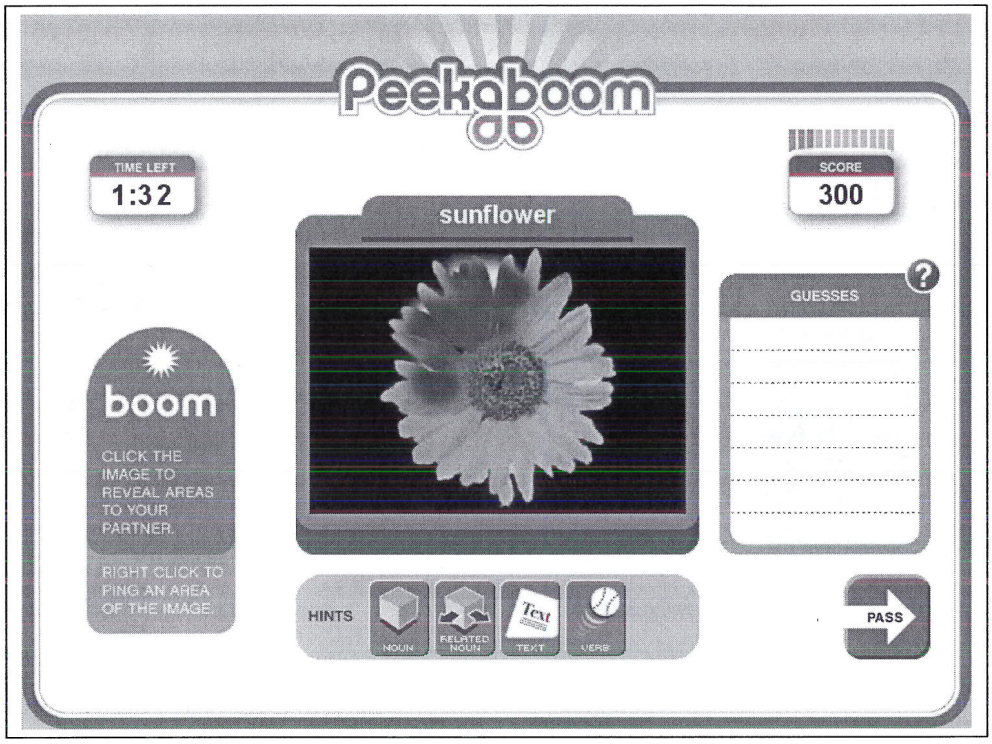

Figure 2.5: *Boom's interface – notice the hints tab and the list of guesses made by Peek.*

object can be identified more accurately with knowledge of its surrounding context than not – the wheel is part of the car, and seeing it along with other sections of the car helps Peek guess 'wheel'. When Boom shows the wheel along with its context to Peek, however, ambiguity arises as to which is the correct label: car, headlight, antique, or wheel? Pings help alleviate the Peek player's possible confusion by giving the Boom player an unambiguous way to specify which object he is referring to.

2.2.4 Hints

Similar in function to pings, hints, shown in Figure 2.5, are another tool that Boom can use to help Peek disambiguate among many possibly correct labels. Whereas pings *pinpoint* the target object, hints say *how* the label assigned to Boom appears – 'noun' when the label refers to a physical object in the image; 'verb' when it's an action of some sort; 'text' if it appears as actual text; and 'noun related' if it pertains to the image in some superficial way. For example, in Figure 2.1, Boom could hint to Peek 'noun' if his label was 'men', 'verb' if his label was 'celebrate', and 'noun related' if his label was 'victory'. The set of hint buttons is shown in Figure 2.6 for the reader's reference.

Another feature meant for Boom to help Peek guess correctly is the use of hot/cold hints. For every guess entry Boom sees, he or she can signal to Peek whether it is hot (close to the label) or cold (unrelated to the label).

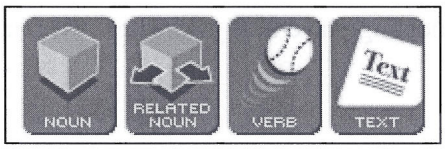

Figure 2.6: *The set of hint buttons.*

2.2.5 Cheating

The prevention and detection of cheating is crucial if we intend to derive correct data from Peekaboom. We assume, in the worst case, that malicious parties will try to maximize their points with complete disregard for possibly corrupted data derived from their actions. One can imagine, for example, two colluders who have some alternate channel of communication, like instant messaging.

We deal with undesirable behavior using a combination of in-game deterence and out-of-game detection. We give a list of the game features that deal with cheating:

- **Player pool** – When players hit the 'start playing' button, they are not immediately matched to an available partner, and instead inserted into a pool of players waiting to start a game of Peekaboom. A timer counts down on 10 second intervals, and once it reaches 0, all players in the pool are *randomly* paired. We intend such a measure to prevent colluders from cleverly timing their game start requests and getting each other as partners.
- **Seed images** – Since Peekaboom is web-based, we not only have to contend with malicious human users, but also bots (web capable programs). To detect automated agents, we introduce seed images into the system; in other words, images alread annotated by a trusted source. By occasionally observing the actions of a suspect player on a seed image, we can tell if he/she/it is honestly playing the game. If their behavior is not satisfactory, then we throw out all existing actions generated by that player and merely ignore all of his/her/its future contributions.
- **Limited scope of player actions** – In a passive way, the design of Peekaboom limits the amount of cheating that can happen. The Boom player cannot communicate words to the Peek player, and the Peek player may type single words back. Still, one could imagine that, even in this regimented setting, players exchange e-mail addresses or instant messages for the purpose of future collusion. We've found that the contrived scenario does not happen in practice, however, and even if it did, we have plenty of other anti-cheating mechanisms at our disposal.

- **Aggregation of data across multiple players** – If the data corruption rate is as high as $1/n$ on the same image-label pair, taking into account all n pieces of data, for sufficiently large n, will mitigate the outlier.
- **IP address checks** – By validating players' IP addresses, we can very crudely tell if they originate from the same network.

2.2.6 Peekabots

When people click the 'I'm ready, let's play!' button, they expect to start playing, and start playing soon. Since a game of Peekaboom requires at least two people, sometimes a real person isn't there for them. To give the semblance of an actual game of Peekaboom for the lone player, we pair him or her with a Peekabot – in other words, the game server takes the opposite role. From a usability standpoint, our system appears as always available to anybody who wants to play. This reduces frustration, and, although we don't collect any actions from simulated games, Peekabots keep people engaged until we can begin collection in a real game.

Since we intend the bot to convincingly simulate a real person, the bot has to play the Peek and Boom roles. The Boom role is exceptionally simple; we take the Boom player's clicks, pings, and hints and play them back to the human on the other end. Playing Peek in a way that is indistinguishable from humans, however, is an exceptionally hard problem – if computers could do this, then they would be able to locate objects in images, and there would be no need for Peekaboom in the first place. Thus, some approximation for playing Peek is in place. In particular, we make the assumption that *areas clicked by Boom are necessary to guess the label*. As the Peek player, the computer has access to a list of guesses made by the human, as well as Boom's revealed area, call it A. As the human player on the other end reveals parts of the image that cover more and more of A, the computer makes each of the guesses in turn. Once all of A is covered by the areas revealed by the human, the computer merely gives the final, or correct, guess.

2.2.7 High Scores

People like to play computer games because of the rush of excitement. They'd also like to feel a sense of accomplishment in the long term. While Peekaboom satisfies the former need with fast action gameplay, it uses a scoring system to satisfy in the latter category. A player's score is persistent – the more he plays, the higher his score. To give him more incentive to play the game, we impose a ranking system, where, to achieve the next rank, the player's score must surpass a fixed threshold. Furthermore, we post two lists of high scores: one for the day, compiled on the hour; and one for all time, compiled every day. We hope that fostering competitiveness among players provides them an incentive to play the game more and produce greater quantities of data under the collection mechanism. Shown below in Figure 2.7 is a screenshot of the daily high scores list.

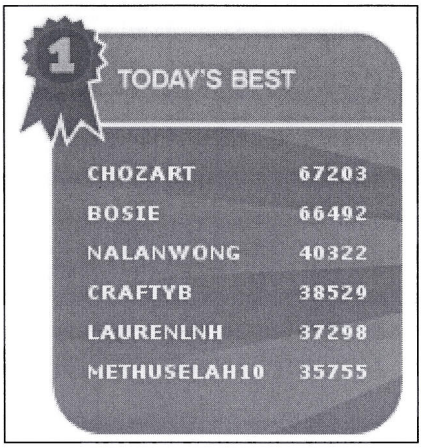

Figure 2.7: *The daily high scores list.*

2.3 Usage Statistics

We released Peekaboom to a general audience on August 1. At the time of writing on August 20, over 20,000 different people have tried out the game and generated at least 2 million pieces of data (one play on one image of one game). A quick back of the envelope calculation shows that each person contributed on average 100 images. A game of Peekaboom lasts 3.5 minutes, and players go through 8.7 images on average, so we estimate that each person spent at least 40 minutes in our system, on average.

Over 90% of the usernames in our system qualify as second-timers or more, and, judging by their scores in the millions, every player featured on the top scores list played at least 1,500 games. Thus, the averages in the first paragraph may be a little misleading: a small, dedicated core of volunteers contributed at least 40% of our repository.

Above all, Peekaboom is *fun*. Below are some comments from devotees' emails:

- "The game itself is extremely addictive, as there is an element of pressure involved in beating the clock, a drive to score more points, the feeling that you could always do better next time, and a curiousity about what is going to come up next. I would say that it gives the same gut feeling as combining gambling with charades while riding on a roller coaster. The good points are that you increase and stimulate your intelligence, you don't lose all your money, and you don't fall off the ride. The bad point is that you look at your watch and eight hours have disappeared!"
- "One unfortunate side effect of playing so much in such a short time was a mild case of carpal tunnel syndrome in my right hand and forearm, but that dissipated quickly".
- "This game is like crack. I've been Peekaboom-free for 32 hours. Unlike other games, Peekaboom is cooperative (rather than competitive)".

CHAPTER 3

Design Patterns for Human Computation

3.1 The Peekaboom Dataset

With a large collection of post-processed player actions, how do we make sense of it all? For starters, we show how players' blob, ping, hint, and pass actions say a lot about the location of the target object, its context, the way it appears, and about the relevancy of ESP generated labels to their respective images. Next, we showcase Peekasearch, a direct and preliminary application of the Peekaboom repository in the absence of any complicated computer vision algorithms.

3.1.1 Properties Guaranteed by Game Features

In the Game Features section, we *described* our design of Peekaboom. Now, we *justify* our choices for including the features we did; as noted earlier, many exist to fulfill a practical purpose, in addition to contributing to the fun factor. In the list below, we enumerate certain desirable properties of the data and exhibit how the features passively enforce them:

- **Objects and their context** – Recognition of objects does not happen in a vacuum; sometimes, given the exact pixels for a target object, one cannot properly classify it. Figure 3.1 below is just one such instance of ambiguity. Incidentally, blobs and pings seem tailor-made to extract target object contexts, as well as the target objects themselves. On one hand, blobs yield the *big picture*, which is however much the Boom player feels is necessary to coax the Peek player into guessing his label. Thus, the regions he clicks on would no doubt contain contextual information, in addition to the target object.

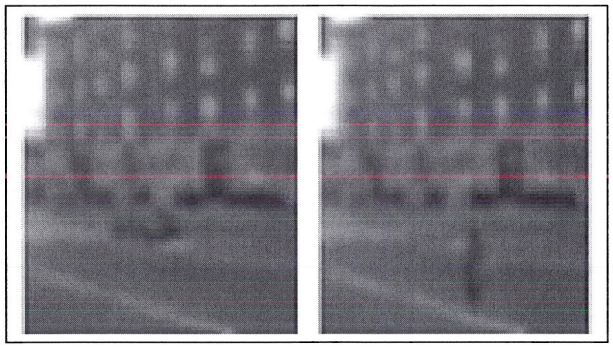

Figure 3.1: *Consider the two different orientations of the dark object across the left and right images. On the left, it's a car driving down the road; on the right, it seems to be a person standing upright.*

On the other hand, pings tell us what the Boom player *really* means, by precisely identifying the pixels contained in the target object.

- **Temporal Saliency** – By exploiting the fact we record the order of the Boom player's blob and ping actions, we are able to derive the most salient, or important, parts of an image. We have observed, for example, that given an image with a man, players almost always click on the face. This should be no surprise, since the most efficient way for Boom to play the game is to reveal the parts of the image that give away his secret label.
- **Quality** – We are able to, with high confidence, determine how relevant labels are to their associated images. Moreover, with a very shallow inspection of the collected player actions, we can assess their intrinsic quality. In the list below, we present criteria currently used to accept or reject the derived data for further consideration:

  - **Type of Hints Given** – Hints, as mentioned earlier, tell us *how* labels appear in their respective images. For most purposes, one need not further consider image-label pairs with a hint other than 'Noun,' because almost certainly the labels will not refer to actual objects. That's not to say that data with other hints given are useless; one could conceivably use those images branded 'text' to train optical character recognition systems.
  - **Peek Player Correctness** – If the majority of player records show Peek guessing Boom's label correctly, then that signifies the target object is easy to identify. Contrarily, if the records recall many pairs opting to pass, that likely means the underlying image makes a poor training example for computer vision algorithms.
  - **Counting** – For a given image-label pair, counting the number of plays is a very rudimentary way of determining whether we should consider it for inclusion in our final dataset. For each play, we can also count the number of blobs created by the Boom player. Thus, the pickier we

are about quality, the higher we should set accept/reject thresholds on counts.

3.1.2 Displaying the Data – Peekasearch

After having collected over a million pieces of data, we wanted a simple way to display our results. Without attempting to extract objects pixel for pixel, we instead compute bounding boxes around them. This is a simple and effective way of *qualitatively* verifying that the game's output is desirable – the user enters a search for a commonplace object (e.g. man, car, dog) into a bar, and Peekasearch returns page after page of images with bounding boxes around that object. Figure 3.2 is a graphic of the search system.

Figure 3.2: *Sample results from a Peekasearch query 'car'.*

3.2 Peekaboom as Human Computation

In an earlier part, we reviewed human computation as a framework for tapping the collective knowledge of humans. As such, we rely on intuition and empiricism to reason about and measure the effectiveness of systems like Peekaboom. We do, however, revisit the three properties of human computing stated at the very beginning of the thesis, and reflect on how the design of Peekaboom addresses them in Figure 3.3.

1. **Incorporates the input-output behavior of humans.**
   Humans are at the heart of our game. One could see Peekaboom as merely a sieve and mediator for their cycles.

2. **Ensures, by its construction, that its human participants exhibit desired input-output behavior. One should assume that the parties involved are self-interested and, in the absence of an rewards structure, unwilling to cooperate.**
   The core Peek and Boom mechanism behind our game ensures that we learn about the target object in an image. In addition, by properly crafting and introducing the concepts of blobs, pings, and hints, we are able to refine our ideas of the target object: where it is, its context, and how it appears. If the exhiliration of playing the game is not enough, we also provide a scoring system. That is to say, we indulge peoples' sense of long-term achievement. Thus, we obtain a guarantee of desired output intrinsically built into the system.

3. **Has a means to detect and discount the contributions of malicious participants.**
   In our game, malfeasance is equivalent to cheating, which we take measures to prevent in-game and detect out-of-game. See the section on cheating to peruse the variety of techniques at our disposal.

Figure 3.3: *Revisiting human computation in the context of Peekaboom.*

3.3 Common Themes

As one instance of using games as a vehicle for human computation, Peekaboom is the product of design decisions that are universal. With this realization, we propose a set of properties that hold for *any* game with a purpose. In the interests of brevity, we discuss two of the many possible key ingredients for games, and mention how the four examples provided (Peekaboom, ESP, Phetch, Verbosity) utilitize each one.

	key fun feature	dual purpose
Peekaboom	Peek and Boom	Boom reveals to Peek regions pertinent to his label
The ESP Game	"try to guess what your partner is thinking"	agreement on a label tied to the image
Phetch	search through a database of images for the one that is being described	the describer communicates with seekers with rich sentences and phrases
Verbosity	the guesser tries to deduce the narrator's secret word from a series of clues	the narrator's use of sentence templates generates common sense facts

Table 3.1: *How our features fulfill a purpose.*

3.3.1 Double Purpose Features

In subsection 3.1.1, we noticed how Peekaboom's features are simultaneously an important part of the gameplay and a passive way to ensure data quality. The pings, blobs, and hints of the game direct players' actions to tell us information about images, yet they are unobtrusive and actually make Peekaboom what it is. Thus, the very components that make a game unique are engineered to fulfill a serious, dual purpose. From experience, we've found that determining the 'correct' combination of features is tricky: favor fun, and lose sight of the original purpose; favor purpose, and the system becomes tedious and annoying to interact with. Using Table 3.1, we illustrate how games with a purpose strike a balance.

3.3.2 Approaches to Game Design

The design of games leverages ingenious ideas already polished from the trials and errors of others. For example, we initially hammered out the mechanics of Peekaboom with the game of Pictionary in mind. Pictionary is a game where, similar to Peek and Boom, one player has to draw a picture and another player has to guess it. As we worked out more details, Peekaboom quickly diverged from the former and became a novel concept in its own right, but the original gist of the design remains. The game of Verbosity has a similar origins story – the core idea originates from the board game Taboo, in which one player describes his or her secret word to a group of guessers without using that word or any of its similar 'taboos'. By taking the mechanics of widely known, fun, fast-paced games and gradually integrating a collection mechanism, one can come up with games that have a more serious, underlying motivation.

Another game design returns to a fundamental question: What would users like? The designers of the ESP Game [18] and Phetch [19] did exactly this – they came up with an idea and ran mockups on paper. Test runs may seem like an obvious step, but they suggest *why* certain ideas are fun and while others are not. As a result, one can understand what kinds of gameplay people would embrace in a final product.

CHAPTER 4

Experiments

4.1 User Studies

User studies are the primary way to validate hypotheses in the field of human-computer interaction. After all, our games solicit information from people, and anything that they produce has correctness in a subjective sense. The notion is defined relative to the behavior of people in the setting of a controlled experiment – we say that the data collected is 'correct' if it achieves a high score under some measure of similarity to what participants in the experiment generate. To see that Peekaboom collects the kinds of training examples we intend, we conduct user studies showing that bounding boxes and pings derived from the game are very close to their in-experiment counterparts.

4.1.1 Evaluation of Bounding Boxes

Before testing bounding boxes, we have to calculate them first. We consider an image-label pair I and all of the n data D_i associated with it. Each D_i contains information about a set B_i of blobs made by the Boom player in his efforts to reveal the image to the Peek player. Now, think of I as a set of its pixels (i, j), and the B_i's as circles with centers and fixed radius 16 (this value was chosen to be a fraction of 24, the radius of Gaussian blobs implemented in the game). We assign each pixel (i, j) a count $c(i, j)$, which is

$$\left|\left\{k \in n \middle| (i,j) \in \bigcup B_k\right\}\right|.$$

In other words, $c(i,j)$ is the number of distinct players whose blobs cover (i,j). Now, for every (i,j), we apply a thresholding rule

$$c(i,j) \leftarrow \begin{cases} 1 & c(i,j) \geq 2 \\ 0 & - \end{cases}.$$

Said a different way, we're interested in those pixels on which data for at least two pairs of players agree. Such a requirement would help discount outliers produced during gameplay. Finally, we compute connected components on what could be thought of as the resulting matrix of 0's and 1's. For each component, we conservatively estimate its bounding box by taking the topmost, bottommost, leftmost, and rightmost points. Notice that an image-label pair may yield multiple bounding boxes, a consequence of the fact that the thresholding step may create multiple connected components. For a visual illustration of how bounding boxes computed with the above method turns out, please refer to the earlier mention of Peekasearch.

To empirically assert that Peekaboom-derived bounding boxes are close to those generated by volunteers in controlled settings, we define an overlap metric between two surfaces A and B as

$$\text{overlap}(A,B) = \frac{\text{area}(A \cap B)}{\text{area}(A \cup B)}.$$

In our case, the surfaces are just regions delimited by their respective bounding boxes. Note that this similarity measure is quite reasonable: perfectly overlapping regions have measure 1; non-intersecting regions have measure 0; and large differences in areas will also result in measure 0.

For the experimental setup, we selected a common pool of 50 image-label pairs found to have singleton bounding boxes and to contain a noun referred to by the label. Each of four volunteers received this same set of images. A volunteer was then asked, for each of the 50 pairs, to draw a box around the part(s) of the image pertaining to the label. An ensemble average taken over the $200 = 50 \times 4$ overlap measurements of volunteer-generated boxes against Peekaboom-derived boxes revealed a mean of 0.754 and a standard deviation of 0.109. For the pair with the *lowest* score averaged over the four volunteers, we refer the reader to Figure 4.1.

4.1.2 Evaluation of Pings

The second of our user studies concerns pings. Recall that a ping is a Boom player action meant to disambiguate a target object to the Peek player. As in the previous bounding box trials, we selected 50 image-label pairs where the label refers to a noun in the image. We then extracted ping coordinates from the data associated with each pair. Three volunteers participated; we briefed them beforehand on examples where we specified what points qualified as 'inside the object' and as 'near or outside of the object'. A volunteer was

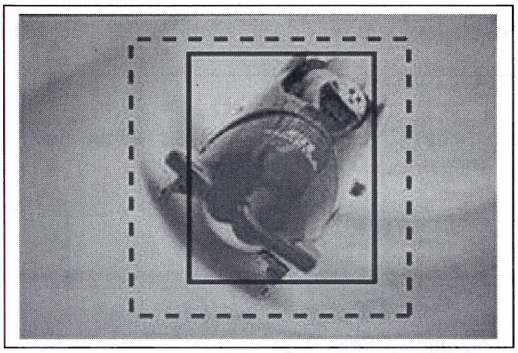

Figure 4.1: *An illustration of our worst result, suggesting that bounding boxes derived from Peekaboom data are quite accurate. The volunteer drew the solid line, and we derived the dashed line.*

then asked, for each pair, whether or not an associated ping chosen uniformly at random was inside the target object. Incidentally, 100% of the 150 ratings agreed with the description 'inside the object'.

4.2 Validation Through Computer Vision

Experiments like the user studies in subsections 4.1.1 and 4.1.2 assert that Peekaboom's output isn't far off from what humans would generate under controlled conditions; they do not, however, directly imply that the data improves the accuracy of computer vision algorithms. To show that having the Peekaboom data does indeed improve the discriminativeness of such algorithms, we design and run an experiment making use of computer vision techniques throughout. Before delving into details, we provide the reader with some background knowledge in image segmentation and object recognition.

Image segmentation, or **segmentation** for short, is the process of partitioning images into internally similar pieces in terms of texture, color, and possibly other features. See Figure 4.3 for the segmentation of a rose into 256 pieces using the always popular normalized cuts heuristic developed by Shi and Malik [13].

Object recognition, or just **recognition**, is the process of identifying objects in images. The *classification* problem in recognition tries to assign correct labels to the most prominent objects in test images – given an image, a dog-cat classifier would return 'dog', 'cat', or 'neither'. On the other hand, the *detection* problem tries to find objects of a specific class in test images – given an image, a car detector would either try to draw a bounding box around the region it believes to be a car or signal that a car isn't present.

We can think of the Peekaboom data as a collection of object detection **priors** (standing for prior knowledge) on images. The data may give us information about the locations of target objects, but how do we convert it into something suitable for trials and measurement? To this an end, we design an experiment

as in Figure 4.2. In stage 1, Peekaboom outputs data (oval 2) derived from the actions of players, along

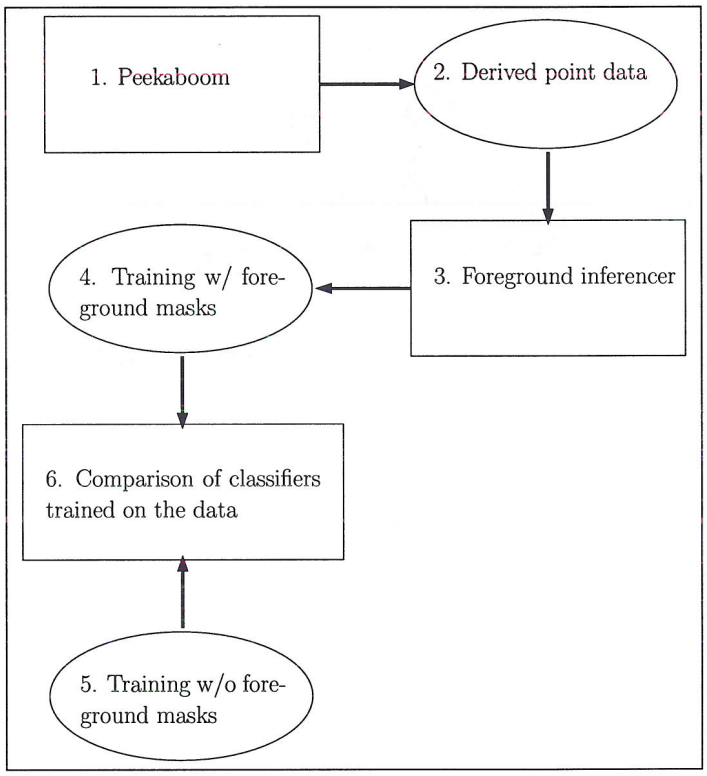

Figure 4.2: *The schematics for our proposed experiment. The squares represent programs, and the ovals represent inputs and outputs to those programs.*

with associated images. The data mostly consist of clicks and pings made by the Boom player. In stage 3, we perform **object inference** – given an image, we independently compute its segmentation and estimate the density distribution of the point data; then, we try to infer the regions belonging to the target object. This stage results in a collection of original images with corresponding **target object masks** (oval 4) whose pixels are white if they are inside the target object, and black otherwise. Lastly, we evaluate the accuracy of two **object classifiers** trained and tested on different pools in stage 6: original images with target object masks (oval 4) versus just original images (oval 5).

4.2.1 The Object Inference Step

Given an image and a target object, before a computer can 'learn' from the Peekaboom data, it needs to know what parts of the image are inside the target and what points are outside. Equivalently, for each pixel (x, y), we assign a `true` or a `false` (or a probability) to the predicate $(x, y) \in T$, where T is the region

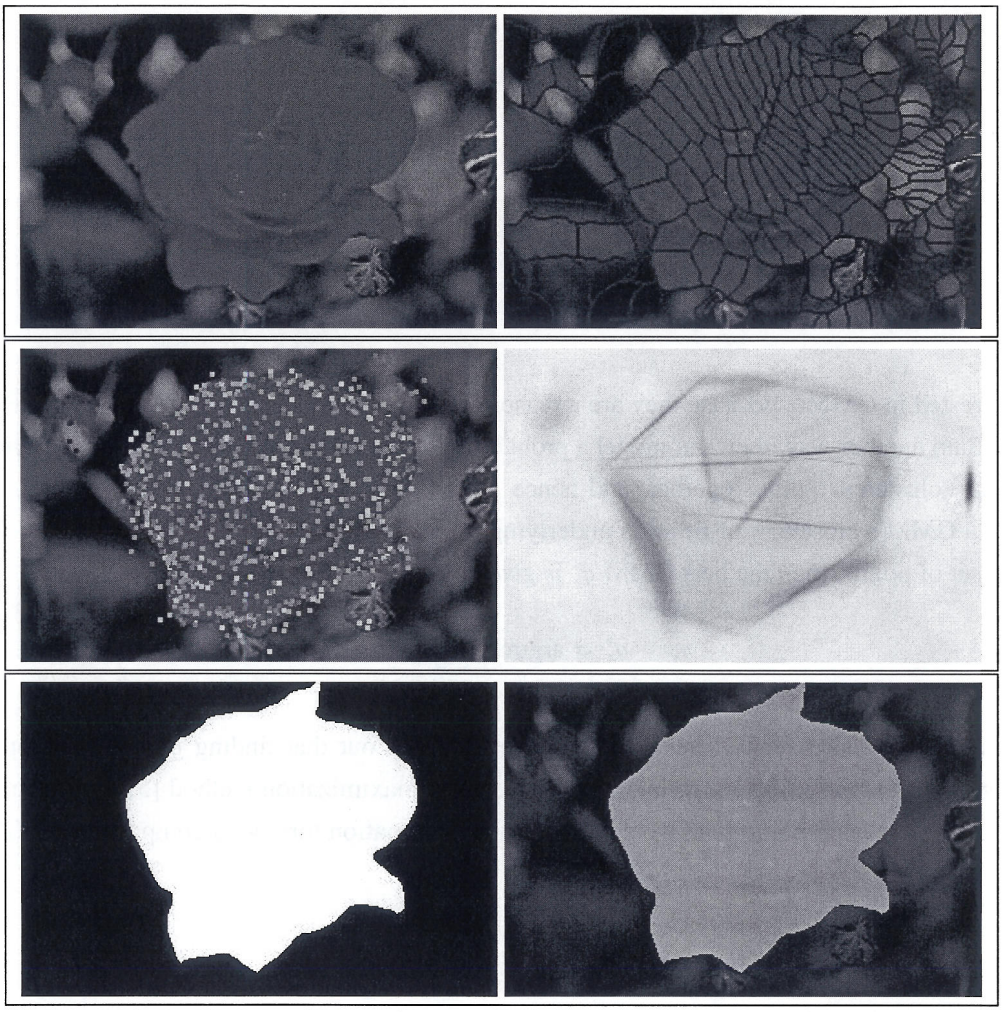

Figure 4.3: *The various stages of our inference scheme. Top left and right: the original image and its segmentation. Middle left and right: points derived from blob and ping actions and the resulting density estimate. Bottom left and right: object mask and the inferred object.*

encompassed by the target object. Using point data directly out of the game is not feasible – the number of Boom player clicks, on the order of 10, is too sparse cover the space of pixels, on the order of 10000. To get around such a hurdle, we employ a Gaussian mixture model based point density estimation and combine it with image segmentation to make a best guess at the region that is the target object. Our discussion focuses on Gaussian mixture models, image segmentation, and then a combination of the two.

Gaussian mixture models [6] are probability distributions consisting of a weighted average of Gaussians. More formally, a mixture model of N Gaussians is characterized by a probability density function f

satisfying

$$f(\mathbf{x}) = \sum_{i \in [N]} w_i \cdot g_i(\mathbf{x})$$
$$\sum_{i \in [N]} w_i = 1$$
$$g_i \sim N(\mu_i, \sigma_i) \qquad \forall i \in [N].$$

We are interested in GMM's because they are a reasonable estimate for the density induced by Peekaboom point data. Said another way, they can model a probability distribution with concentration in regions where the points are clustered tightly together, and hence assign more importance to those regions. A natural measure of a GMM's closeness of fit to its underlying points is the maximum likelihood score. If P is the underlying set of points, then the best GMM g^* is given by

$$g^* = \operatorname*{argmax}_g \sum_{p \in P} g(p),$$

where the argmax is taken to be over all GMM's g. It turns out that finding g^* is a hard optimization problem, and so we approximate it through the expectation maximization method [5]. Given an image label pair and its associated point data, we create a density estimation through a composition of GMM's (1).

Let $\{P_1, \ldots, P_n\}$ be the associated data, and let P_i be the points of the ith datum. [Recall that each datum represents one play by a distinct pair of players.];
for $i \in [n]$ **do**
    Let g_i be the maximum likelihood GMM computed for P_i with $|P_i|/8$ Gaussians. [An extra Gaussian for every 8 points.];
end
Let g be the weighted sum of the g_i's, or

$$g = \sum_{i \in [n]} \frac{1}{n} \cdot g_i$$

[The contribution of every pair of players is weighted equally.];
Normalize g so that it sums to 1 when evaluated on all pixel coordinates of the image. [Notice that, before normalization, g was originally a probability distribution on the plane, an infinite set of points.];
return g

Algorithm 1: Computing a Gaussian mixture model.

Figure 4.3 (see middle left and middle right pictures) has a visualization of the probability distribution gotten from such a procedure.

Independently of the point density estimation, we compute a segmentation of the image using the Berkeley Segmentation Engine [9], requiring an output consisting of 256 **superpixels**, or pieces (see top right picture of Figure 4.3). One important fact to know about the Berkeley Segmentation Engine is that it outputs, as a side effect, a **pixel affinity graph**. Their construction is an area of study in itself, so we will just view pixel affinity graphs as weighted graphs (V, E, w). Each pixel of the image is a vertex that has edges connecting it to every other vertex within a radius of 5. Consequently, each edge e's ($e = \{u, v\}$) weight $w(e)$ is determined by the 'affinity' of the pixels u, v. The reader need only think of affinity as a similarity measure among pixels – do they have similar surrounding colors and textures? With a point density estimation and a segmentation in hand, we now proceed to combine the two.

The point density estimation, call it g, induces a natural distribution over the segmentation superpixels, call them $\mathcal{S} = \{S_1, \ldots, S_k\}$. Since g is a probability distribution over the pixels of the image, we can define a distribution h and its inverse h^{-1} (with some constant normalization factor Z) as follows:

$$h(S_i) = \sum_{(x,y) \in S_i} g(x, y)$$
$$h^{-1}(S_i) = (1 - h(S_i))/Z.$$

Said differently, the probability measure of a super pixel is the sum of the probabilities of image coordinates it encompasses. Likewise, we extend the pixel affinity graph (V, E, w) to a superpixel affinity graph $H = (\mathcal{S}, \{\{u, v\} \mid u, v \in \mathcal{S}\}, w')$ ($\mathcal{S}$ is overloaded to represent the vertex set and the S_i's overloaded to represent the vertices) by setting for all $S_0, S_1 \in \mathcal{S}$:

$$D := \{\{u, v\} \in E \mid u \in S_0, v \in S_1\}$$
$$w'(\{S_0, S_1\}) = \sum_{e \in D} w(e).$$

Dispensing with formality, the weight of an edge between two superpixels S_0, S_1 (now thought of as vertices of H) is the sum of the pixel affinity weights of edges crossing between S_0 and S_1. Before constructing the final graph I, let us take stock of what we have so far:

- A probability distribution h over superpixels that assigns mass to each superpixel based on the total point density contained inside it.
- An inverse probability distribution h^{-1} assigning mass to where h assigns little.
- An affinity graph H on superpixels gauging the similarity among pairs of superpixels.

We construct the final graph I by incorporating all three of the aforementioned concepts. To start with, we introduce two vertices u, v that we call the **source** and the **sink**, respectively. We connect the source and sink to all the vertices of H and set their weights accordingly, like so (let c be the new edge weight function):

$$V(I) = V(H) \cup \{u, v\} = \mathcal{S} \cup \{u, v\}$$
$$E(I) = E(H) \cup \{\{u, S\} \mid S \in \mathcal{S}\} \cup \{\{v, S\} \mid S \in \mathcal{S}\}$$
$$c(e) = \begin{cases} w'(e) & e = \{S_0, S_1\} \wedge S_0, S_1 \in \mathcal{S} \\ \alpha \cdot h(e) & e = \{u, S\} \wedge S \in \mathcal{S} \\ \alpha \cdot h^{-1}(e) & e = \{v, S\} \wedge S \in \mathcal{S}. \end{cases}$$

Note that α is some scaling constant we will choose later. One can think of I as the result of augmenting H with a new set of edges, each of whose weight is proportional (or inversely proportional in the case of h^{-1}) to the probability mass of the superpixel incident on it. See Figure 4.4 for an intuitive pictorial interpretation.

We can think of I as a **min-cut** problem. More specifically, we want to find a partition of the graph into two sets of vertices $T, \overline{T}$ such that $u \in T$, $v \in \overline{T}$ such that the total weight (according to the function c) of

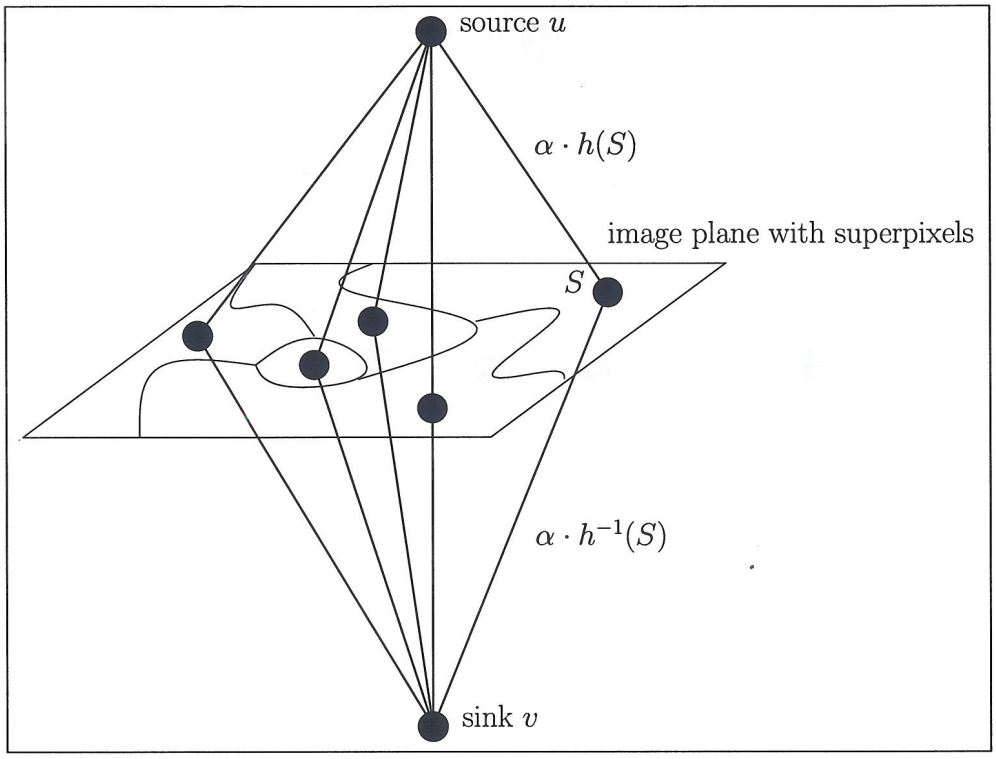

Figure 4.4: *A diagram of how we plan to construct the graph I. The edges incident on the source and the sink have their weights determined by the distributions h and h^{-1}, respectively, and scaled by a factor α. The subgraph induced on vertices of the image is exactly the superpixel affinity graph.*

edges crossing the cut is minimized. In mathematical notation:

$$\langle A, \overline{A} \rangle = \{\{a, b\} \in E(I) \mid a \in A, b \in \overline{A}\}$$
$$\langle T, \overline{T} \rangle = \operatorname*{argmin}_{\langle A, \overline{A} \rangle} \sum_{e \in \langle A, \overline{A} \rangle} c(e).$$

The problem of finding minimum cuts [8] is one of most extensively studied topics in computer science. As such, there are efficient, polynomial time algorithms to solve it. Figure 4.5 illustrates what a cut in I would look like. Keeping the picture in mind, we claim that the vertices belonging to the same partition as the source u *are a good estimate for the target object region.* To see this, call the edges with u as one endpoint 'prior' edges and call the edges induced from the original affinity graph H 'affinity' edges. We emphasize how the two types of edges are weighted differently: prior edges have their weight determined by point data density, while affinity edges have their weight set by the image segmentation algorithm. Thus, the minimum cut trades off between cutting prior edges as opposed to affinity edges, while trying keep its total weight

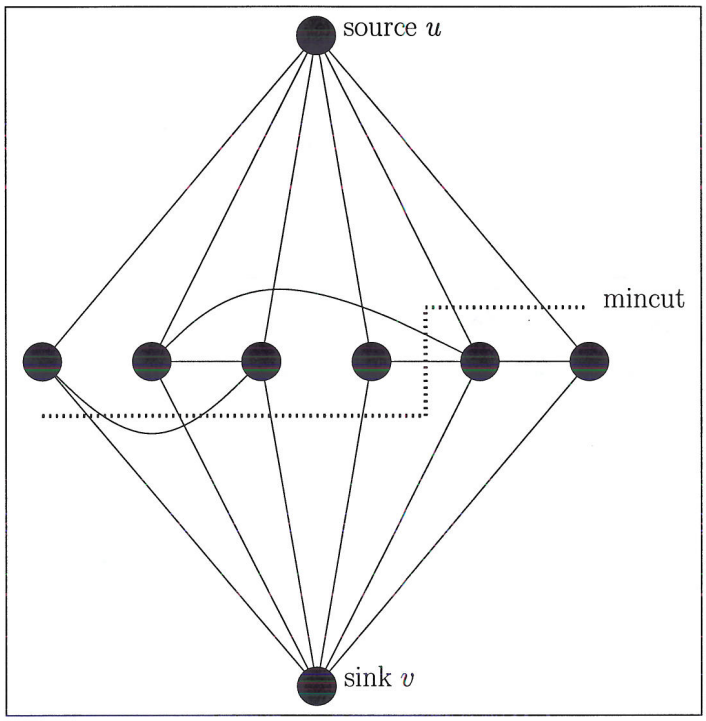

Figure 4.5: *The min-cut of the graph separates foreground superpixels from background.*

small. In effect, it has take into account the regions that Peekaboom says are important, as well as those that mutually have high affinity according to the Berkeley Segmentation Engine. Looking upon the vertices assigned to the source partition as the superpixels they represent, one can estimate the target object region by taking their union. See the bottom right frame of Figure 4.3 for a sample end result.

We remind the reader that we go through all of the above trouble to derive target object masks as in the bottom left frame of Figure 4.3, that is, white and black images that denote membership and non-membership of pixels in the target object. The masks in turn will help improve the accuracy of classifiers trained on them.

4.2.2 The Object Classification Step

Does Peekaboom improve intraclass discriminativeness? In other words, given a concept class, does the game pick out parts of images intrinsic to that class? We demonstrate how integrating the information in the form of target object masks improves recognition rates by as much as 10%. Before giving experimental findings, however, we provide background on the kind of algorithm used and exactly how we plan to shoehorn in our masks.

Our repository consists of 7 classes (airplane, dog, face, flower, man, tree, water) of 400 images each, for a total of 2800 images originally fetched from the web for the purposes of running the ESP Game. We ensure that each image has a sufficient number of plays on it to estimate its target object mask. From the training set, and target object masks, we make two groups: **augmented**, the training set *and* masks; and **unaugmented**, which is just the training set.

To build our classifiers, we make extensive use of the texton based feature representation of images. Our discussion will be self-contained, but the interested reader is referred to [15] for a foundational view and [22] for an application. We provide an overview of our procedure (2) with accompanying diagrams in the appendix.

1 Given an image, we convert it into a set of **filter responses** by convolving it with a **filter bank** [appendix entry];
2 From a pool of images, we construct a space of filter responses by selecting, say, 256 responses from each image. Cluster the space using K-means with $K = 2048$. Usually the cluster centers summarize the whole space well; call the set of centers our **visual dictionary** D [appendix entry];
3 Represent images as histograms of size $|D|$, where each entry of the histogram corresponds to a word in the dictionary. Take an image, and convolve it with our filter bank to get a set of filter responses. For each response, find its closest visual word, and increment the entry of the histogram mapped to that word. After all filter responses have been counted, the resulting histogram is a $|D|$-dimensional vector encoding the image [appendix entry];
4 Compute the histogram representation of training and test set images. Classification proceeds like so – given a test image and its histogram, find the histogram in the training set that matches it most closely. The match is tied to a training image, with class x. Consequently, the classifier infers that the test image's class is also x [appendix entry];

Algorithm 2: Building a texton based classifier.

Of course, the pseudocode makes no mention of how we treat the augmented and unaugment experimental groups differently throughout the whole process, and so step 3 deserves closer scrutiny.

The convolution of a filter bank with an image produces a 17-dimensional (17 is the number of filters in our experiment) feature for each pixel in the image. For the unaugmented group, *all* of the responses contribute to the resulting histogram. For the augmented group, however, only the responses on pixels that fall within the mask are counted; we include a filter response only if we believe it to be inside the target object. Essentially, our testing setup consists of two parallel tests: one in which classifiers are trained (and tested) on the full set of filter responses, and one in which they are restricted to looking at a set designated with information provided by the game. Thus, we will be able to say *whether use of the Peekaboom data helps computer vision algorithms discriminate between classes of objects.*

We measure a classifier's accuracy by taking the fraction of the test images for which it guesses correctly.

	airplane	car	dog	face	flower	tree	water
airplane	**311**	34	12	12	11	14	12
car	32	**222**	29	16	47	33	21
dog	16	34	**154**	56	64	53	23
face	4	18	72	**179**	84	33	10
flower	9	23	35	31	**224**	61	17
tree	18	20	43	26	58	**199**	36
water	32	31	49	41	35	72	**140**

Table 4.1: *The confusion matrix for the augmented case.*

	airplane	car	dog	face	flower	tree	water
airplane	**288**	29	11	7	14	19	31
car	26	**202**	32	14	51	51	24
dog	8	36	**167**	53	67	46	23
face	4	29	119	**109**	85	24	30
flower	9	30	49	33	**201**	55	23
tree	22	53	39	19	67	**134**	66
water	37	59	51	22	45	107	**79**

Table 4.2: *The confusion matrix for the unaugmented case.*

Notice that we don't set aside a test set, and instead perform leave one out cross validation on the training set. This approach gives us the maximum number of images to train on.

We present our results in confusion matrix form. A **confusion matrix** is a matrix $\{\Omega_{i,j}\}$ whose rows and columns are the object classes we train and test over: an entry $\Omega_{i,j}$ is the number of times the input was of class i and the output was of class j. Clearly, the numbers on the diagonal i count the times objects of class i were classified correctly, and vice versa for numbers not on the diagonal.

In Tables 4.1 and 4.1, we give the confusion matrices for experiments on our seven concept classes in the augmented and unaugmented cases, respectively. The experiment where we considered information from Peekaboom (augmented case) achieved an accuracy of 51%, while the experiment where we didn't (unaugmented case) achieved an accuracy of 42%. From the higher accuracy, we infer that taking into account player actions from the game helped distinguish images of one class from another. More specifically, the classification process benefited significantly from knowing regions of images likely to contain objects of that image's class.

CHAPTER 5

Concluding Thoughts

5.1 Promise and Limitations

At the beginning of the thesis, we summarized the notion of human computation: utilizing the unique capabilities of humans as a subroutine of some overarching algorithm. We then proposed computer games as a way to channel human understanding into training examples usable by machine learning algorithms. In particular, we laid out the internal workings of Peekaboom for all to see, from collection to post processing to learning.

That human computation deserves further investigation is not in doubt – humans interact with computers as a part of daily life now, so for the benefit of both parties, computers can improve their performance by soliciting humans for insights not otherwise available to them. What of computer games, however? Are they the optimal way to aggregate human cycles? The concept of 'games with a purpose' and its many instantiations has addressed many difficult problems already, like locating objects in images, annotating images with descriptions, and gathering common sense facts. For Peekaboom in particular, we've managed to achieve positive empirical results. Despite the promise of this new way of thinking, something needs to be said of the limitations.

First, we note that ideas behind games are intrinsically unique – they make each game what it is. There is no cookie cutter method, but in its lieu are general design guidelines. The work done so far showcases some of these principles, and surely there are other solutions to hard AI problems waiting to be discovered.

Second, games are sometimes appropriate remedies for situations requiring human feedback, but not always. In many cases, the best way to tap into the expertise of humans is simply to *ask* them. Imagine, for example, a biomedical program that displayed images to technicians and asked them to pick out cancer cells. While enjoyability, incentives for the participants and guarantees of correctness are desirable properties, they are not strictly necessary for the task described. Lab technicians, as opposed to self-interested web users

catered to by our games, are experts and have a job to do.

Finally, a game, as a level of indirection, invariably obfuscates the training examples we intend to collect. Directly asking users to paint an outline of the target object would, under *ideal* circumstances, yield higher quality data: we could specify to participants exactly how we wanted them to pick out the target object from background clutter. Of course, doing the collection from within a game is far more preferable than not, as we wouldn't be able to gather millions of examples in the first place!

5.2 New Directions

Much more can be said and done with the simple experiment we proposed, implemented, and tested in section 4.2. We focus on two areas for future improvement and exploration.

5.2.1 Making More Use of Limited Information

What if a computer could 'refine' sparse or noisy knowledge provided to it by making many passes over the same training set? We make the observation that the object inference step of our experiment takes into account a single image at a time, oblivious to other images of the same object class. On the other hand, our classifiers, by construction, simultaneously take into account sets of images, class by class. It stands to reason that if classifiers can discriminate among classes, they can also say what makes a class special. In particular, for our texton based method, a 'car' classifier would be able to say which visual words appear with a high frequency in cars. Thus, given an image labeled 'car' by the ESP Game, such a classifier would be able to assert, with different degrees of confidence, whether or not each pixel belongs to the target car.

To see how a learning algorithm would refine an initial training set provided to it, consider the tree image in figure 5.1. As is often the case, experienced Boom players only show minimal but salient parts of the image; they have no need to reveal the other trees in the picture. For our purposes, however, we would like all the trees in the image. Towards this end, we use a texton-based tree classifier (very much like one featured in our earlier experiments) to 'hallucinate' a set of Peekaboom points that most likely belong in the 'tree' class. We treat the fake points just as if a real player created them, and rerun the object inference procedure on the union of both real and fake points. Thus, we hope to iteratively improve object inference from a qualitative standpoint and ultimately boost classification accuracies.

Figure 5.1: *The Boom player only needs to reveal one tree among many in this scene to get the Peek player to guess 'tree'.*

5.2.2 Improving the State of Computer Vision

Although our experiment shows that Peekaboom picks out class-specific parts of images, it's still a very rudimentary step. Our plan going forward is to exhibit that data from the game can make a difference in cutting edge computer vision algorithms. Virtually all training sets for computer vision come from hand labeling, and therein we intend to leverage our greatest strength: size. Our repository of one million examples, although large, is disparate in the breadth of the object classes contained within. Focusing our collection efforts on a handful of classes would greatly increase the size of the training sets we can work with. Currently, our largest classes contain 600 fully annotated (by ESP and Peekaboom) images each, as exhibited in the experiment in section 4.2.

With more training examples in hand for common classes like 'bicycle', 'person', and 'car', we can test our repository with state of the art recognition algorithms found in submissions to the 2005 Pascal Challenge [7], for example. Only then can we definitively show that our game helps advance computer vision by making the collection of large training sets on the order of thousands to tens of thousands feasible.

Appendix

APPENDIX A

- **Player actions** – The actions that we observe players taking during gameplay. We record all their actions so that we can recreate a game fully.
- **Data** – Computer-usable training examples inferred from player actions.
- **Datum** – One piece of data arising from one play of one image for a distinct pair of players.
- **Dataset** – A collection of data.
- **Collection mechanism** – The process of recording player actions, storing them, and converting them into data.
- **Correctness** – The concept of correctness is very subjective, so we when refer to data as 'correct,' we merely mean that it will perform well against hand-labeled examples in controlled experiments and measurably improve the performance of computer vision algorithms trained on it.
- **Target object** – For an image-label pair, the object referred to by the label.

Figure A.1: *Some useful definitions to keep in mind while reading.*

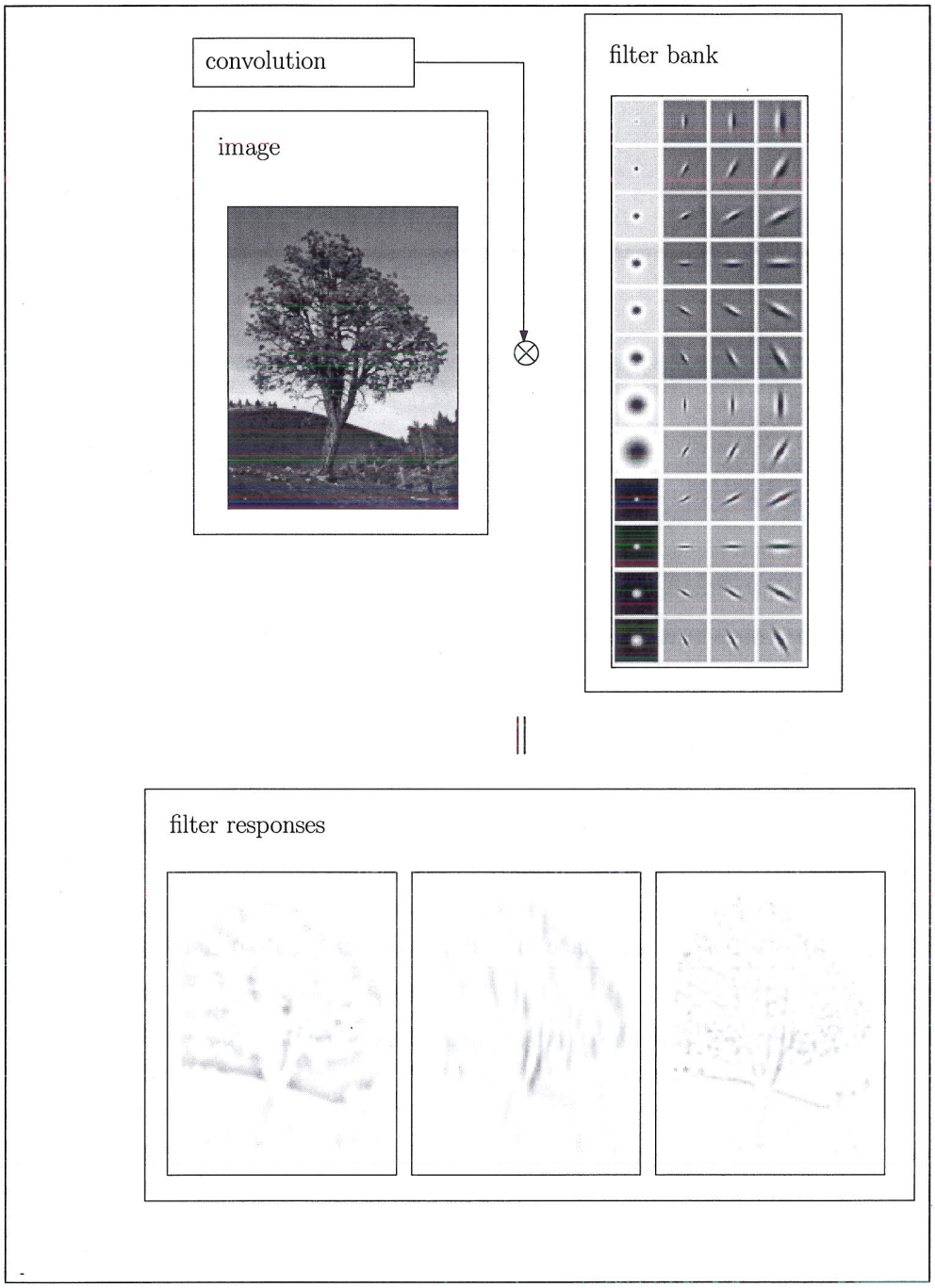

Figure A.2: *When we say that an image is 'convolved' with a filter bank in 1, we really mean that we convolve each filter in turn with an image to produces a series of convolution results. The pictured filter bank is due to Leung Malik [10]; ours is similar to it, and follows the construction in [15].*

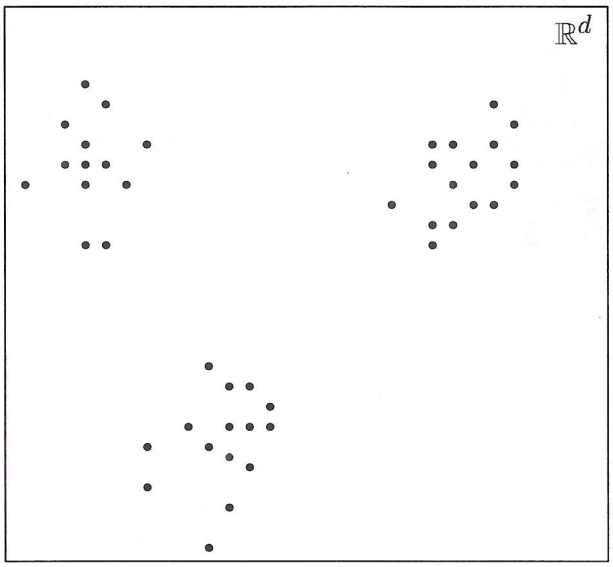

Figure A.3: *In step 2, we 'summarize' a space of d-dimensional points by estimating cluster centers (color coded in red). We interpret the centers as the words in our visual dictionary of filter responses.*

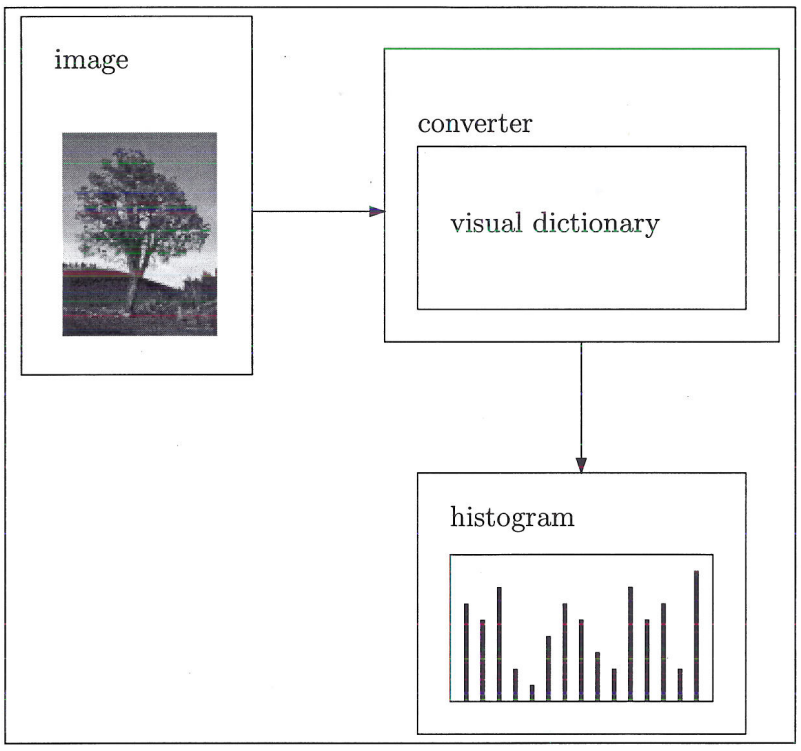

Figure A.4: *We reduce images into histograms by running them through a converter, which has access to the visual dictionary generated in step 3.*

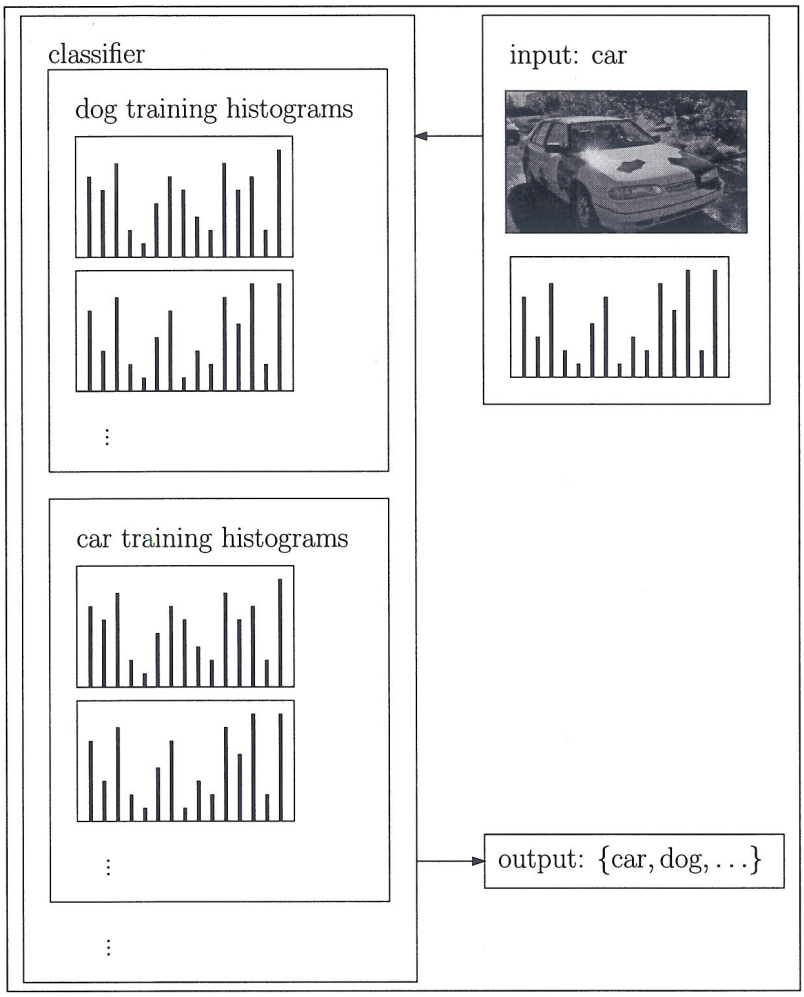

Figure A.5: *Given an image and its associated histogram, the classifier in step 4 finds the closest match from its set of training examples and returns the label of the closest match.*

Bibliography

[1] Flickr photo sharing service. `http://www.flickr.com`.

[2] The facebook. `http://www.thefacebook.com`.

[3] Interactive digital software association. `http://www.idsa.com`.

[4] Chad Carson, Megan Thomas, Serge Belongie, Joseph M. Hellerstein, and Jitendra Malik. Blobworld: A system for region-based image indexing and retrieval. In *Third International Conference On Visual Information Systems (VISUAL '99)*, Amsterdam, The Netherlands, 2 1999.

[5] A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1):1–38, 1977.

[6] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification (2nd Edition)*. Wiley-Interscience, 2000.

[7] Mark Everingham et al. The 2005 pascal visual object classes challenge. In *PASCAL Workshop '05*, number 3944 in Lecture Notes in Artificial Intelligence, pages 117–176, Southampton, UK, 2006.

[8] L.R. Ford and D.R. Fulkerson. *Flows in Networks*. Princeton University Press, Princeton, NJ, 1962.

[9] Charless Fowlkes, David R. Martin, and Jitendra Malik. Learning affinity functions for image segmentation: Combining patch-based and gradient-based approaches. In *CVPR (2)*, pages 54–64, 2003.

[10] Thomas Leung and Jitendra Malik. Representing and recognizing the visual appearance of materials using three-dimensional textons. *International Journal of Computer Vision*, 43(1):29–44, 2001.

[11] David G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the International Conference on Computer Vision (ICCV)*, pages 1150–1157, 1999.

[12] Bryan Russell, Antonio Torralba, and William T. Freeman. Labelme: A database and web-based tool for image annotation. Technical report, MIT AI Lab, September 2005.

[13] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000.

[14] David G. Stork. Open data collection for training intelligent software in the open mind initiative.

[15] Manik Varma and Andrew Zisserman. A statistical approach to texture classification from single images. *International Journal of Computer Vision*, 62(1-2):61–81, 2005.

[16] Paul Viola and Michael Jones. Robust real-time object detection. *International Journal of Computer Vision*, 2002.

[17] Luis von Ahn. Human computation. Technical Report CMU-CS-05-193, 2005. Carnegie Mellon Computer Science Department PhD Dissertation.

[18] Luis von Ahn and Laura Dabbish. Labeling images with a computer game. In *CHI '04: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 319–326, New York, NY, USA, 2004. ACM Press.

[19] Luis von Ahn, Shiry Ginosar, Mihir Kedia, Roy Liu, and Manuel Blum. Improving accessibility of the web with a computer game. In *CHI '06: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, New York, NY, USA, 2006. ACM Press.

[20] Luis von Ahn, Mihir Kedia, and Manuel Blum. Verbosity: A game for collecting common-sense facts. In *CHI '06: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, New York, NY, USA, 2006. ACM Press.

[21] Wikipedia. Wikipedia – wikipedia, the free encyclopedia, 2004. [Online; accessed 6-April-2006].

[22] J. Winn, A. Criminisi, and T. Minka. Object categorization by learned universal visual dictionary. In *ICCV '05: Proceedings of the Tenth IEEE International Conference on Computer Vision*, volume 2, pages 1800–1807, Washington, DC, USA, 2005. IEEE Computer Society.