Learning to Play *Hearthstone*

J David Smith
University of Kentucky
Lexington, KY, USA
emallson@cs.uky.edu

Abstract

One of the goals of computer vision is to develop techniques that allow computers to understand and interact with our world through imagery. A classic challenge in this area is choosing a problem that is both sufficiently constrained to be tractable and popular enough to have significant available data. I claim that the task of automatically playing video games ('botting' them) is such a problem. I make use of well-understood machine-learning and computer vision techniques to develop a 'bot' which is capable of playing Blizzard Entertainment's Hearthstone effectively. I conclude with a discussion about the potential for such bots as vehicles for computer vision research.

1. Introduction

The ability to understand the world through images is one of the primary goals of computer vision. However, the world is enormously large and unimaginably complex. As a result, tackling that task directly is intractable. The research community has instead focused on important individual pieces: object detection and classification, scene classification, and geometric understanding, among other things. I propose to take a step closer to interaction with the real world by interacting with limited mini-worlds in the form of video games.

The inspiration for this paper is the observation that video games form natural 'mini-worlds'. The size and scope of even the largest video game is significantly smaller than that of the real world. Therefore, successfully interpreting the imagery presented by a game is a considerably smaller problem than that of doing so in the real world, although it requires very similar methods. The limited forms of interactions available in video games may also be seen as a benefit. This reduces the set of tasks any successful bot must be able to perform from nearly all of them to a small subset in a natural way. Further, games often provide concrete ways to measure success. This helps to avoid the problem of determining how successful a machine is at interacting with the world. Video games also often demand that action be taken in real time. While a broad genre of turn-based games exists, it is only one part of the larger ecosystem. This places a valuable time constraint on vision algorithms that is often neglected.

The problem of collecting data for experiments is also significantly easier to tackle with video games. Many players stream their games online, allowing others to watch them play in real time. Many of these streams are preserved as videos which are viewable on-demand. This presents a large corpus of available data indexed in a logical fashion. Annotating this data remains a problem. When this problem is not the subject of research, it is possible to sidestep through careful choice of games. When annotation is the subject of research, researchers may take advantage of the nature of games to obtain annotations automatically. This is discussed further in section 7.

In this paper I demonstrate that it is feasible to automatically collect data for and play Blizzard Entertainment's game *Hearthstone*. This game was chosen for several reasons:

- It was popular, which allowed large quantities of data to be collected easily.
- It was at a reasonable medium between visual complexity and simplicity: most visuals are composed of non-deforming objects being moved. This simplified annotation.
- It had low timing requirements: turns in *Hearthstone* may last up to 90 seconds. This was seen as important because I was unsure of the speed of existing methods. Other games have much tighter requirements. For example: *DotA 2* players need to react in less than 300ms in order to be remotely competitive.
- It was easy to model the decision process with existing research.

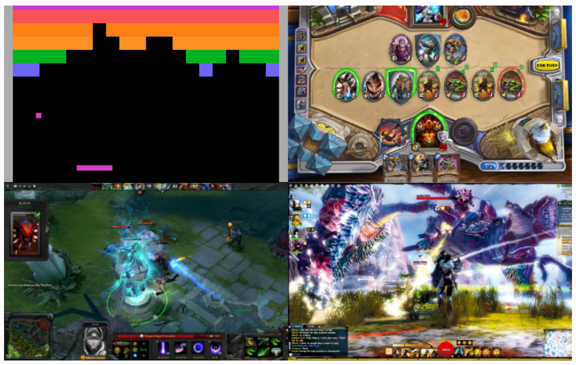


Figure 1. A Classic video game (top left; played by Google[8]) versus Modern video games. Games shown: Breakout, Hearthstone, DotA 2, Guild Wars 2

2. Related Work

There is relatively little work related to the specific problem of interpreting the visuals of video games. Among the only ones I found is the Mnih et al. paper from Google[8], which presents an unsupervised method for learning to play classic video games.

However, this work makes significant use of methods for scene classification, object detection and similarity matching. There has been significant progress in recent years on these tasks, especially since the introduction of deep learning into computer vision. For example, Zhou et al.[14] presented a state-of-the-art scene classification result with 205 different classes running on the 'Places' database. Meanwhile, Krizhevsky et al.[6] presented state-of-the-art results for object classification.

However, for this project my needs were simpler: binary scene classification, fast object detection, and unbounded object classification (which is better attacked as a similarity matching problem). Deep networks, of the kind used by [14] and [6] are powerful but have uncertain performance in terms of time. In the past, Linear SVM has been successfully used [2][13]. A variety of fast descriptors also exists, including SIFT[7], SURF[1] and Tiny Images[12].

The decision to use older methods was also motivated by time constraints on the project: implementing the latest methods would take non-trivial amounts of time. Older methods already have good, fast implementations. For example: scikit-learn[10] includes a fast and easy-to-use implementation of SVM with a linear kernel, while OpenCV[3] has efficient implementations of SURF and SIFT. Additionally, dlib[5] has an efficient and simple HOG-based cascade classifier.

3. Problem Statement

The objective of this paper is to demonstrate that video games are effective 'mini-worlds' which represent the intersection of many areas of computer vision research. To demonstrate this proposition, three things are necessary:

1. It must be shown that a vision-based bot is feasible.
2. It must be shown that a bot is more demanding than any individual task, but less demanding than the real world.
3. It must be shown that the evaluation of a bot can be derived naturally from the game it plays.

To accomplish this, I seek to develop an effective vision-based bot for *Hearthstone*. A bot is *effective* if it can play the game at least well enough to be mistaken for a human player. A bot is *vision-based* if the only inputs considered in its decisions are derived from visual information presented by the game.

In this paper, I present the beginnings of such a bot. This would show that such bots are feasible and, as will be seen throughout the rest of the paper, that they incorporate a significant amount of vision research. Upon completion of the bot, it could be shown trivially that evaluation comes naturally from the task at hand.

4. Approach

While deep-learning approaches are popular at present, I chose to construct the *Hearthstone* bot out of older, well-understood components rather than trying to train a deep network to play it directly from video. This was motivated by practicality: fast implementations exist for older ideas like SVM, and debugging individual pieces is easier than debugging the whole. The use of older techniques has an additional benefit: research has advanced since Linear SVMs were state of the art, so it stands to reason that more advanced games are playable *right now* if sufficiently fast implementations exist for newer methods.

The task of playing hearthstone was broken down into several pieces: data collection, annotation, play prediction, and input prediction.

4.1. Data Collection

One of the largest streaming websites on the Internet is *Twitch.tv*. At the time of writing, this website reported hundreds of 'channels' presently online. Many of these channels have all or part of their streams stored long-term for on-demand viewing. I tapped this resource, collecting 40 GB of videos tagged as *Hearthstone*.

The videos collected from Twitch are not perfect, however. Streamers may miss-tag videos or overlay other information on top of them. Figures 2 and 3 show how *Hearthstone*'s main menu normally looks and how it looks on a randomly chosen stream, respectively. To tackle the miss-tagging problem, I train a SVM to determine whether a frame is of *Hearthstone* or not. The overlay problem ultimately is one of generalization, and the SVM is shown to handle it well in section 5.1.

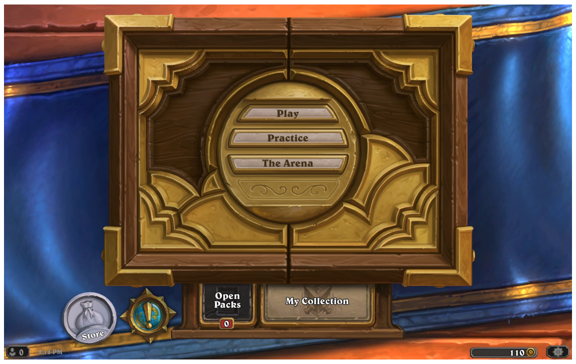


Figure 2. The *Hearthstone* main menu as it normally appears.

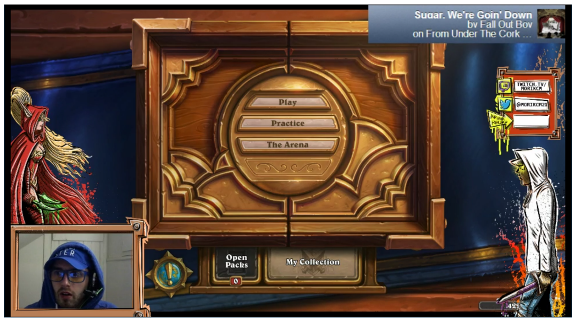

Play
Practice
The Arena
Open Packs
My Collection
Sugar, We're Goin' Down by Fall Out Boy on From Under The Cork

Figure 3. Many streamers overlay graphics including webcams and other video feeds on their streams. This is the Hearthstone main menu as it appeared on *morikcm*'s stream.

4.2. Data Annotation

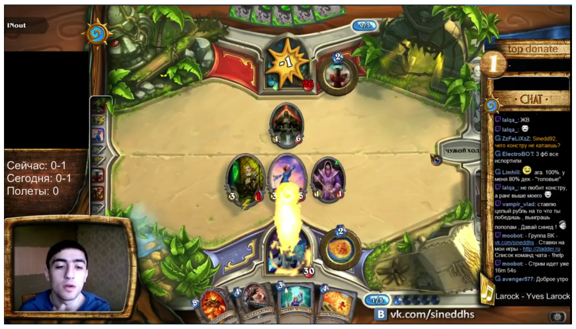


Figure 4. A game in progress. The streamer is playing a Mage, has 30 Health and 0 Armor, has 1 Mana out of 5 Maximum, and has 3 cards on the board and 4 in his hand. His opponent is playing a Shaman, has 26 Health and 0 Armor, has 0 Mana out of 5 Maximum, and has 1 card on the board and 5 in his hand.

In annotating the videos, we seek to extract all important information for understanding the scene. For this paper, it is done with a collection of detectors and classifiers. In *Hearthstone*, we want to know the following information:

1. Is this frame of a *Hearthstone* game?

   Sometimes videos are miss-labeled or streamers switch games without changing what their stream is tagged as.

2. What is each player's:

   - Class: There are 8 classes, each of which has access to a different set of extra cards
   - Mana & Maximum Mana: Mana is the resource players use to take actions. The maximum value varies throughout the game.
   - Health & Armor: Health is a resource which determines the winner. First player to run out of Health loses. Armor is temporary added Health.

3. What cards are in play, where, and who owns them?

   Cards may exist in a player's hand, on the board, or as a weapon. Some cards also have effects that depend on location on the board.

4. What effects and attributes does each card have?

   There are special effects that can be placed on cards including *Taunt*, which prevents attacking any card but the one with *Taunt*, and *Divine Shield*, which absorbs 1 attack completely without draining the card's Health. Each card also has its own Health and Attack values, which may change as a result of other cards being played.

4.2.1 Determining *Hearthstone*-ness

This is the simplest piece of the project. I extracted SURF and Tiny Images features from frames of randomly chosen, manually annotated videos. I trained an SVM with a linear kernel using scikit-learn[10]. Evaluation is in section 5.1. The Tiny Images features were chosen as their all-around performance was significantly higher than SURF.

4.2.2 Extracting Metadata

I did not have the opportunity to complete this part of the project. However, my initial experiments indicate that combining dlib's cascade classifier (which performed quite well at extracting minions) with an OCR tool like Tesseract[11] will give high-quality results.

4.2.3 Detecting Cards in Play

There are two distinct sets of cards that are in play at any given time: cards in a player's hand, and cards (more specifically: *minions*) on the board.

I trained a cascade classifier using dlib to detect minions. I manually annotated 40 frames of *Hearthstone*, then took

30 for a training set and left 10 for a test set. Evaluation is in section 5.2.

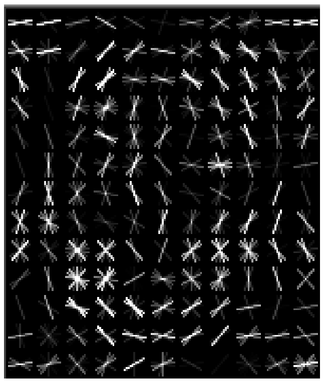


Figure 5. Visualization of the HOG detector learned by dlib.

I did not have the opportunity to train a detector for cards in the player's hand. I initially attempted to do so using a collection of card images downloaded from the *Hearthstone* sub-reddit. However, this detector performed extremely poorly. I believe the cause of the poor performance is that the 'synthetic' data (the card images) lacked the context which the image frames provided. However, the excellent performance of detection of minions gives me confidence that applying this same method would result in good results.

4.2.4 Distinguishing Cards

One might be tempted to phrase the problem of distinguishing cards as multi-class classification, to be done in a one-vs-all fashion with a collection of SVMs. However, that approach has several issues:

1. There are more than 500 kinds of cards. This results in 500 classes to be tested on. By itself this is not a significant problem, but given that my data is largely unlabeled an approach that doesn't require labeling hundreds of frame cutouts would be ideal. I am not confident that basing a classifier on data collected from a fan-site would have good performance, given the poor performance of my initial dlib card detector.

2. More cards are added with a reasonable frequency. Over the past year, 3 new sets of cards (1 expansion pack and 2 player-vs-computer challenges which reward cards) have been introduced. The ideal situation would be to be able to distinguish these new cards from existing cards, something which a normal multi-class classification approach does not handle.

Therefore, I approach this problem instead as one of *similarity matching* and *clustering*. I began by extracting detected minions from a subset of *Hearthstone* frames. Then, I ran several experiments on different ways of matching cards. A qualitative evaluation of these methods is given in section 5.3.

1. SURF + FLANN

   My initial test used SURF descriptors with FLANN[9] matching, both of which are easy to compute with OpenCV. However, no matter what parameters I tried I could never get a sufficient number of matches for any method (manual or otherwise) to use for clustering.

2. SIFT + FLANN

   I next tried SIFT with FLANN matching. I expected equally poor results, but was pleasantly surprised when not only did the method produce a *significantly* higher number of matching points (a full 10x increase), but it also proved to work well for clustering. I used this method to cluster the minion cutouts in 2 ways.

   First, I manually assigned a threshold of 30 matching points. This threshold was chosen based on observations of the number of matching points between two instances of the same minion. If two minions A and B had greater than 30 matching points, they were considered to be the same and grouped together. The first example for each cluster was used as the exemplar for the cluster, against which future instances were compared. This produced surprisingly good results.

   I next tried using Frey & Dueck's Affinity Propagation clustering algorithm[4]. This algorithm was chosen because it learns the number of clusters along the way, and it allows a flexible definition of similarity. Given $n = 4289$ minion cutouts, I constructed an $n \times n$ similarity matrix with similarity defined as 'number of matching points'.

   The manual method had split a few minions into multiple clusters. Each of these clusters appeared to be consistently occluded. The affinity propagation method did not do this, but occasionally grouped a minion with an incorrect class.

3. SIFT Bag of Visual Words

   Following SIFT + FLANN, I attempted to cluster the minions using a Bag of Visual Words based on SIFT descriptors with Euclidean distance as the similarity metric. A vocabulary was constructed by taking SIFT descriptors of 1500 cutouts and clustering them with k-means into 100 clusters. Then, I clustered the cutouts using Affinity Propagation. However, this method produced results which were seemingly random.

4.2.5 Detecting Effects & Attributes

I did not have the opportunity to complete this part of the project. It could be accomplished in a similar method as in

4.2.2. Effects are naturally classification problems, while attributes are numbers which require detection, localization and then OCR.

4.3. Play Prediction

I did not have the opportunity to experiment on predicting the actions the bot should take. However, I made significant progress on building a description of frames so that an algorithm would – in theory – be able to do so.

I believe that phrasing this portion of the project as a multi-agent stochastic game would allow me to take advantage of the significant body of research on multi-agent Markov decision processes to learn to play. *Hearthstone* is naturally a stochastic game because many of its elements (such as drawing cards) are random in nature.

4.4. Input Prediction

The final piece for an actual working bot is the ability to make inputs to the game in order to act on the predictions made by the component described in section 4.3. However, I did not have the opportunity to develop this piece.

In practice, many bots have hard-coded methods for generating input. However, these are complex to develop and prone to breaking on edge cases. I believe that it is possible to apply learning-based methods for this task rather than developing input generation by hand.

Since *Hearthstone* is a mouse-driven game, the general plan for implementing this would be to take a representation of mouse clicks and drags in association with state information given by the above to learn how to input maps to state transitions.

5. Evaluation

I quantitatively evaluate the methods described for determining if a frame is of *Hearthstone* and for detecting minions. Due to a lack of labeled data, I did not quantitatively evaluate the method for distinguishing minions. I also qualitatively evaluated all three methods.

5.1. Deciding if a Frame is of *Hearthstone*

I labeled 16 videos of *Hearthstone*, assigning each frame in the video either a 1 or -1 for *Hearthstone* or *Not Hearthstone*, respectively. I trained on 50% of the videos and tested on 50%. The performance of each method is summarized as an ROC curve.

Bag of Visual Words with Upright-SURF did not perform as well as expected, having an AUC of only 84.7%. 16 × 16 Tiny Images performed significantly better than expected, achieving 99.7% accuracy on the same task. I hypothesize that this is because the *Hearthstone* window is typically stationary in a consistent position.

Because Tiny Images had such high performance, I felt the need to do an extensive qualitative evaluation to insure

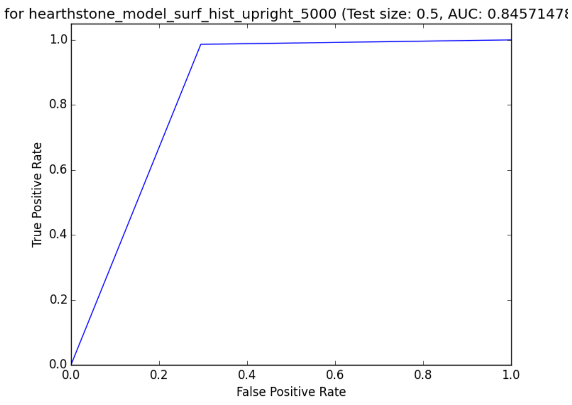


Figure 6. Linear SVM + SURF Histogram performance on frame type classification

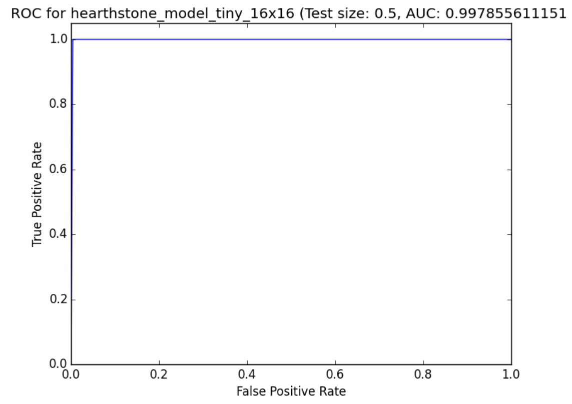


Figure 7. Linear SVM + 16 × 16 Tiny Images performance on frame type classification

there were not bugs in my evaluation. I watched 5 unlabeled videos and looked for incorrect predictions. I did not notice any. Examples are in figure 8.

Note the last example (row 2, column 3): the frame has *Hearthstone* in the background, but has more than half of the screen occluded. I count this as a true negative, because the state information recovered from this frame would be incomplete and likely incorrect.

5.2. Detecting Minions

I labeled 40 frames positively identified as *Hearthstone* by Tiny Images + Linear SVM. I trained a HOG-based cascade classifier using dlib on 30 of these frames, and tested it on 10. I evaluated it by comparing the pixels labeled as 'minion' by dlib versus the true values. This method achieves a pixel-labeling accuracy of 0.92 (see figure 9). Sample cutouts are displayed in figure 10.

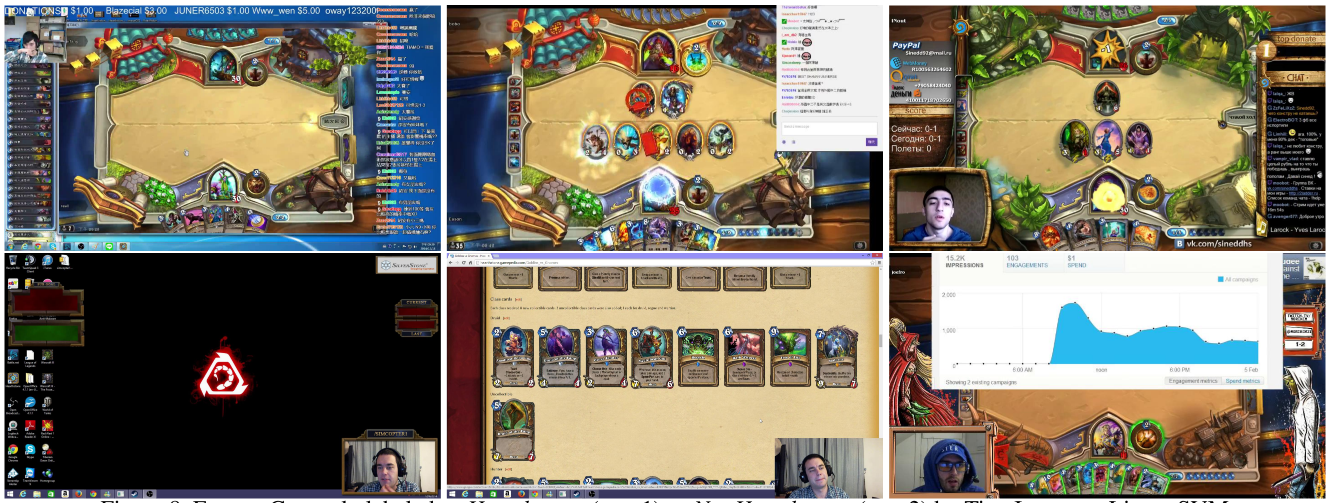


Figure 8. Frames Correctly labeled as *Hearthstone* (row 1) or *Not Hearthstone* (row 2) by Tiny Images + Linear SVM

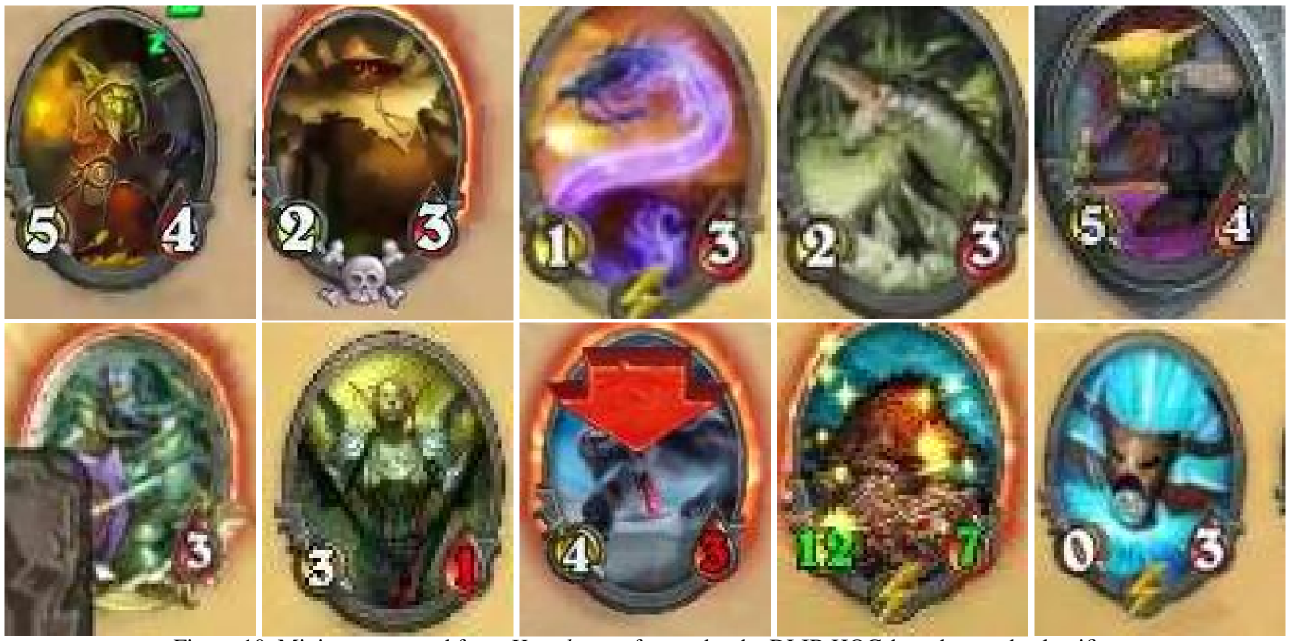


Figure 10. Minions extracted from *Hearthstone* frames by the DLIB HOG-based cascade classifier

5.3. Qualitative Evaluation of Minion Similarity Matching

Since I do not have ground truth labels for minion clustering, I conducted only a qualitative evaluation of the results.

5.3.1 SIFT+FLANN with a Manual Threshold

This clustering seemed to perform the best of the methods I tried. While in terms of raw performance it seems tied with SIFT + FLANN with Affinity Propagation, this method is biased towards exclusion (when unsure, split the cluster) while Affinity Propagation is biased towards inclusion (when unsure, leave the cluster alone). Examples of clusters found with this method are in figure 11.

5.3.2 SIFT+FLANN with Affinity Propagation

While this method occasionally incorrectly identifies minions, my qualitative analysis shows that it does remarkably well. Example clusters are shown in figure 12.

5.3.3 Bag of Visual Words (SIFT) with Affinity Propagation

This method performed very poorly, with only 5 cluster centers chosen when tested on 800 cutouts. Because of pro-

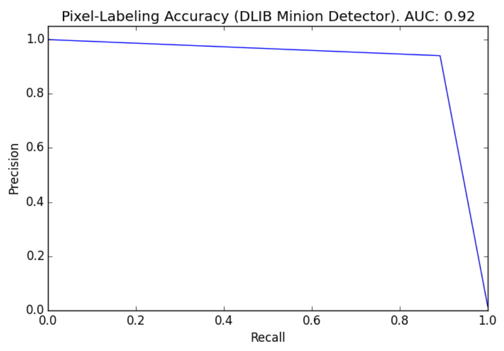


Figure 9. Precision-Recall curve of pixel-labeling accuracy of the DLIB Cascade Classifier

hibitive computation time, it was not tested on the full 4289 cutouts. An example of one of these clusters is in figure 13.

6. Applications

While the thrust of this paper has been towards demonstrating that video games provide viable mini-worlds, there are direct applications of the work presented.

6.1. Building Better In-Game Bots

Currently, *Hearthstone* has some bots to practice against. However, their most advanced ones are pretty trivial to win against. A better bot could provide a more advanced option for testing new strategies against strategies learned by watching real players.

6.2. Outlier Detection

By looking at the learned transition function (see section 4.3), it may be possible to better identify cards and card combinations which are more powerful than desired (outliers). While the Hearthstone developers already have access to information on how people are playing the game, watching a bot play from contrived scenarios may give further insight into why players play the way they do.

6.3. Outsider Statistics

Third-party sites are ever-popular for online games. Many games (including *Hearthstone*) have third-parties interested in keeping and displaying statistics about their game (card statistics, common decks, win percentages, etc), but do not have direct access to this information. A good classifier could provide this information by watching streams and providing annotations to these third parties.

7. Discussion

In this work, I have shown that it is feasible to build bots that learn play a game by looking at the visual information that is presented to the player. While the tools I used were not state of the art, I still demonstrated excellent performance at the task of understanding the game's visual information. This particular game is relatively simple by modern standards; more complex games would require more effort to get similar performance.

The proposal that video games fill an evaluation role in-between performance on a single task and performance on interacting with the entire world is supported by this. While I did not reach to the point that I could give performance results of a complete *Hearthstone*-playing system, I did show that this is a reasonable goal.

Evaluating performance on *Hearthstone* would be natural: what is the bot's win/loss rate over a few hundred games? This represents the bots capability when playing against a human opponent. A value of 1 would imply that the bot is approximately as good as a human, while higher values would indicate that the bot is better than human.

Similar natural evaluation methods exist for more complicated games, such as *League of Legends*, *DotA 2* and *Counter Strike*. Such evaluations would perform the role of 'system tests' in computer vision. We know that it is possible to accurately perform many tasks independently, but how well can we do when performing several under time constraints? Further: how do the errors of various algorithms interact? How do the errors matter, and how can they be reduced? These questions will appear when building robots that interact with the natural world. Investigating them could open up interesting new avenues of research within computer vision.

Much of my time on this project was spent on building annotations, and it suffered from lack of labeled data. While many game companies (including *Blizzard Entertainment*) have countermeasures to prevent users from reading a game's memory or connection information, they may be willing to provide academics with tools to build annotated datasets. Regardless of cooperation, for many games it is still possible to build such datasets. This could provide academics with a wealth of information on which to test.

I have shown that it is feasible to build effective vision-based bots to play video games, and that doing so would provide meaningful information about the performance of computer vision algorithms. I have shown that the video game data available is sufficient for such a bot, and briefly described how the problem of obtaining annotations could be side-stepped. Finally, I have discussed how the study of vision-based bots can offer answers to several research questions which will have application towards one of the goals of computer vision: building systems that can understand and interact with the world through imagery.

Source code available at `https://github.com/emallson/learn2play`

Special thanks to Connor Greenwell.

References

[1] H. Bay, T. Tuytelaars, and L. Van Gool. Surf: Speeded up robust features. In *Computer vision–ECCV 2006*, pages 404–417. Springer, 2006. 2

[2] M. R. Boutell, J. Luo, X. Shen, and C. M. Brown. Learning multi-label scene classification. *Pattern recognition*, 37(9):1757–1771, 2004. 2

[3] G. Bradski. *Dr. Dobb's Journal of Software Tools*. 2

[4] B. J. Frey and D. Dueck. Clustering by passing messages between data points. *science*, 315(5814):972–976, 2007. 4

[5] D. E. King. Dlib-ml: A machine learning toolkit. *Journal of Machine Learning Research*, 10:1755–1758, 2009. 2

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 2

[7] D. G. 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. Ieee, 1999. 2

[8] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015. 2

[9] M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. *VISAPP (1)*, 2, 2009. 4

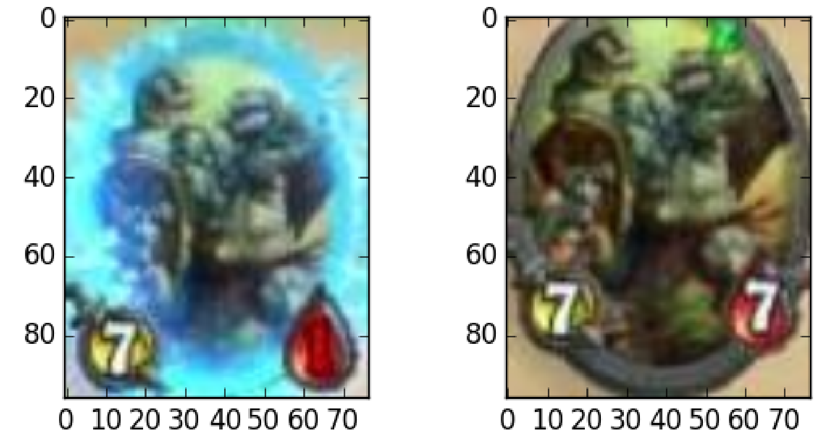
[10] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. 2, 3

[11] R. Smith. An overview of the tesseract ocr engine. In *Proceedings of the Ninth International Conference on Document Analysis and Recognition - Volume 02*, ICDAR '07, pages 629–633, Washington, DC, USA, 2007. IEEE Computer Society. 3

[12] A. Torralba, R. Fergus, and W. T. Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 30(11):1958–1970, 2008. 2

[13] J. Yang, Y.-G. Jiang, A. G. Hauptmann, and C.-W. Ngo. Evaluating bag-of-visual-words representations in scene classification. In *Proceedings of the international workshop on Workshop on multimedia information retrieval*, pages 197–206. ACM, 2007. 2

[14] B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, and A. Oliva. Learning deep features for scene recognition using places database. In *Advances in Neural Information Processing Systems*, pages 487–495, 2014. 2



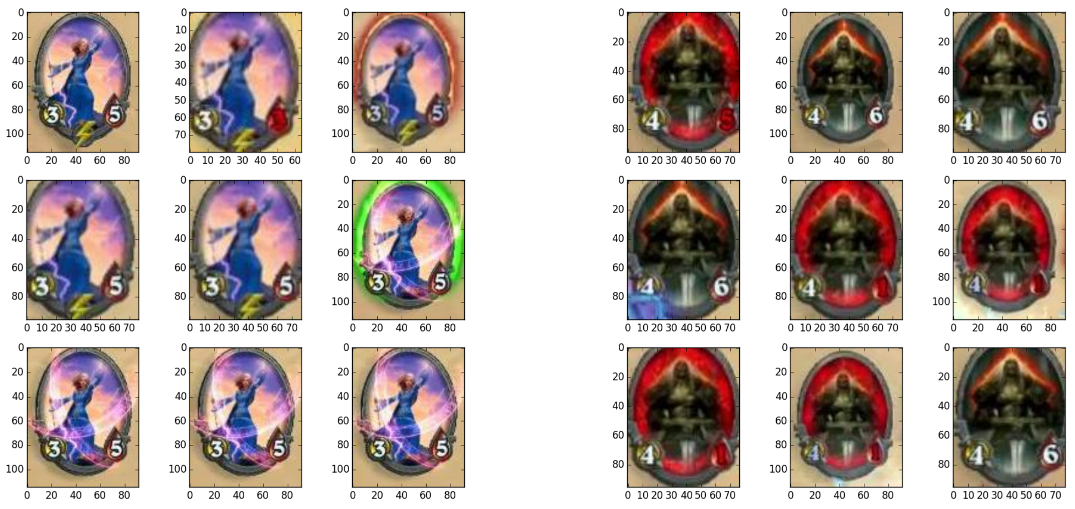


Figure 11. Clusters identified by manual threshold with SIFT+FLANN

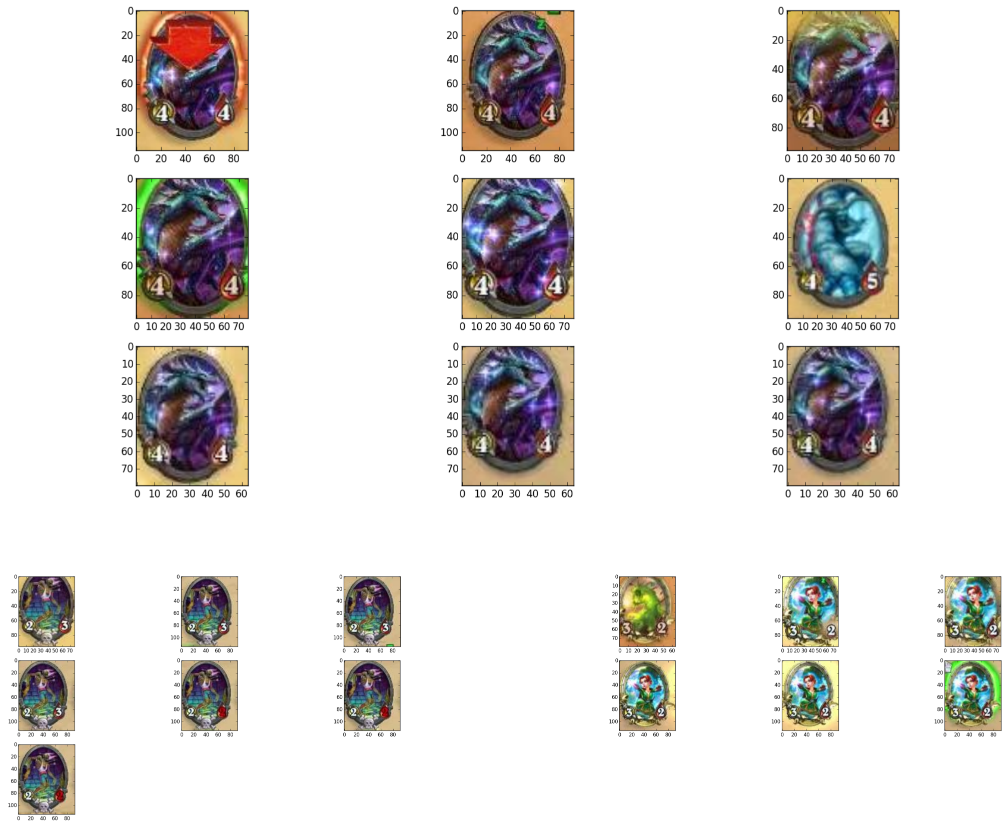


Figure 12. Clusters identified by Affinity Propagation with SIFT+FLANN

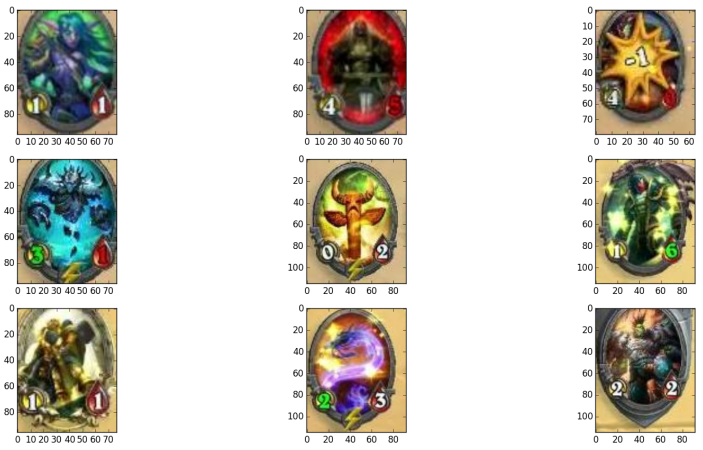


Figure 13. A cluster chosen using Affinity Propagation with a Bag of Visual Words build using SIFT descriptors.