Detecting Hand Direction for Computer Interaction

Ben Mattinson and David Gaddy
Massachusetts Institute of Technology
77 Massachusetts Ave. Cambridge, MA 02139
bmatt@mit.edu, dgaddy@mit.edu

Abstract

We present a method for detecting the direction of a pointing hand in an image for the purpose of creating a computer interface based on pointing. Given an image that is dominated by a pointing hand, we use regression to determine where it points. We collected a set of images to use for comparing performace on this task. We evaluated the performance of two different features for the regression: convolutional neural networks and histograms of oriented gradients. We also compared two different methods of performing the regression: K-nearest neighbors and least squares linear regression. This work differs from many previous works in that it focuses on detecting the angle of a single finger in relation to the screen rather than on estimating hand poses or recognizing gestures.

1. Introduction

Gesture recognition is an important problem for Human Computer Interaction because hands provide a very natural way to interact with the computer. Users are used to using their hands to manipulate the world and often use their hands when communicating ideas to other people. Giving computers the ability to recognize gestures would allow people to manipulate objects on a computer in a way that mimics the way they manipulate the world and communicate with others, instead of forcing them to use an artificial form of manipulation like a computer mouse. There are also other related applications for this technology, such as the interpretation of sign language.

The task of gesture recognition is often divided into two parts: first detecting hands in an image, and second recognizing the gesture a hand is making. Detecting the hands deals with finding local patches of pixels where a hand appears in an image. In order to do this, the computer must be able to differentiate hands from other objects in the background and from other parts of the body. Recognizing hands differs from many object recognition tasks, such as face detection, because the hands have many degrees of freedom that allow them to appear very different in images based on the gesture being made. For this reason, hand detection is often done by modeling skin color, but it has also been done by classifying shapes and combinations of the two methods.

Hand gesture recognition deals with the problem of, given a patch of pixels that represents a hand, determining the configuration the hand is in. One approach to this problem is modeling the joints of the hand and using image data to determine the position of each joint, but because of the complexity of this model, an example based approach is often used instead. In the example based approach, hand images are compared against examples using various features to find an example with a similar hand configuration. This approach is similar to the problem of object classification since different configurations of a hand look like different objects when projected onto a two dimensional image.

2. Previous work

2.1. Hand detection in images

Because the shape of hands varies greatly based on the position and configuration of a hand, color is often used as a primary feature for detecting hands in images. In [11], Bergh et al. modeled skin colors with a Gaussian Mixture Model of skin color and combined this with a color histogram of faces found with a face detector. By combining both the general skin color model with a scene specific skin color, they were able to detect skin accurately. To distinguish hands from other skin, they took the largest connected regions of skin that were not the face (determined using the face detector). Their method was reported to be more robust to lighting and different users than more basic color models, though they still encountered problems when faces and hands overlapped in images.

Other methods for detecting hands involve looking at shape. Ong and Bowden [8] trained a tree of boosted classifiers to find hands in images. To deal with the many different possible shapes of the hands, the first level of the tree proposed possible hand locations that were then put through classifiers lower in the tree that were trained to detect spe-

cific hand configurations. They reported a 99.8% success rate on hand detection.

One last cue used for detecting hands is depth information. To deal with the problems of detecting hands with color alone (namely overlapping skin regions), Bergh and Gool used depth images taken with an infrared depth camera in addition to color in [12]. By placing the restriction that the hands be held a certain threshold in front of the face (determined using a combination of a face detector and depth image) they were able to accurately detect hand regions, even when they overlapped with other skin.

2.2. Hand gesture recognition

One approach to recognizing hand gestures is to build a model of the joints of the hand and to try to determine the location of these points on an image. Erol et al. give a survey of such techniques in [3]. The full kinematic model of a hand has 27 degrees of freedom. To deal with all of these degrees of freedom, constraints of the ranges of motion for each joint and joint angle dependencies are considered. In addition, methods such as placing markers on the hands or tracking the movement of hands through time have been used to make detecting the joint locations possible.

Another approach involves comparing the hand image to a set of example images for different gestures. This approach builds a classifier that attempts to put a given image into a class that represents the type of gesture being made. To do this, a vector of features is extracted from the image and compared against the features on the given examples. A variety of features and classifier types have been used to do this.

One approach, used by Freeman and Roth [4], used a histogram of gradient orientations in the image as a feature vector. Using gradients made these features invariant to lighting and using a histogram made it invariant to translation. By finding the training feature vector that is most similar to the vector for a given image, they were able to distinguish between 10 different hand gestures.

More recently, neural networks have been employed for accurate continuous hand pose estimation. Tompson et al. [10] used a deep convolutional neural network to identify the positions of key points on hands in RGB+Depth images. They trained the neural network to recognize certain hand features and used the results to infer the hand pose.

2.3. Finger detection and gesture recognition

Several different methods of detecting fingers in images exist. Most involve first detecting a hand in the image and using information about the hand position and components to identify fingers. Hands can be detected using some of the aforementioned techniques, e.g. color or shape. Lee and Lee [7] used color and information about consecutive motion between frames of video for hand detection. After isolating the hand from the image, they examined the curvature of the hand region for sharp curves to identify fingertips. To measure the direction of each fingertip, Lee and Lee selected two points along the edge of the finger, one on each side a set distance from the fingertip. They then measured the direction of the vector between the average of these two points and the fingertip.

3. Methods

3.1. Problem setting

For this work, we decided to focus on the problem of finding the orientation of a hand given a region where the hand is known to be instead of detecting the hand in a larger image. To do this, we made several assumptions about the images we processed.

First, we assume that the images have a clear background. Although this assumption will often be broken in the target application of computer interaction, additional preprocessing could be done in a complete system to segment out the background of an image. Methods for hand detection such as those described in the previous work section that use hand color, depth, or other cues could be used for this purpose.

Second, we assume that the hand is roughly at a constant distance from the camera and at roughly the same location in the image. For the application of pointing at a computer screen, this assumption is fairly reasonable since a user's distance to the screen will usually vary by only a few feet at most and is likely to hold their hand directly in front of the screen to point at it. To make the system more robust or for other applications, the location of the hand in an image could be detected using the methods described in previous work and our system could be passed only the region around a detected image. Additionally, we could make our system more robust to distance and location by training on images that contained hands at different distances and locations.

3.2. Data collection and evaluation

We collected a set of images to evaluate the relative effectiveness of different methods. These images were collected from a standard webcam directly above a computer screen. Each picture contained a pointing hand to recognize on a blank background. The hand was approximately one foot away from the camera and screen. To keep hand location consistent, the user was shown a video of their hand as they recorded and attempted to keep their hand on a crosshair overlayed at the center of the image. A dot was displayed on the screen and the user pointed at the dot as it moved across the screen. The dot moved along a seven by seven grid with spacing of 80 pixels and for each location a picture was taken of the user pointing at that location from the webcam. We repeated this grid collection procedure 15

times, collecting a total of 735 images.

For evaluation, we used 15-fold cross validation, except the groups were chosen based on the grid collection iteration instead of a random sample. For each data group, we trained on the other 14 groups and evaluated performance on the group, then we took the average performance for all 15 groups. The reason for choosing groups based on collection iteration instead of random sampling is that images collected during the same scanning process tended to be more similar so training with images in the same iteration as the test data artificially increased scores.

It is important to note that the data collected has inherent noise because people are limited in precision for pointing at a location on a screen. The location a user thinks they are pointing at may depend on the relative orientations of their eyes, their hands, and the screen. For this reason, no system will be able to achieve perfect results.

3.3. Overview of methods

Created feature vectors to describe image. We evaluated two different methods of extracting these feature vectors: convolutional neural networks and histograms of gradient orientations. We then trained regressors from these features to the ground truth location. We experimented with several different types of regressors and report here the results for least squares linear regression and K-nearest neighbors regression. Other regression methods did not provide any significant improvement. We used the scikit-learn library implementation of these methods [9].

3.4. Convolutional neural networks

One source of features we evaluated was convolutional neural networks. Convolutional neural networks have been used to get state of the art results in object recognition tasks such as the ILSVRC-2012 competition [6]. They are made up of many layers of convolutional filters which are learned from training images, with each layer learning a higher representation of the image. It has been found that the values of hidden units in the intermediate layers of a network provide good features for a variety of computer vision tasks, even tasks that the network was not trained for [2]. For our features, we used the Caffe deep learning framework[5] running a pretrained model architecture that is a variant of Krizhevsky's ILSVRC network[6][1]. We ran their model and extracted the outputs of the final fully connected layer before the softmax layer, which gave us a 4096 dimensional feature vector.

3.5. Histogram of Gradients

We also evaluated the Histogram of Oriented Gradients (HOG) feature descriptor [1] as a means to detect the location and orientation of the pointing hand. This expands on work done by Freeman and Roth, who used gradients to detect hand gestures. We used a variant of the R-HOG algorithm described by Dalal and Triggs in [1]. We divided the image into an 8×12 grid, yielding approximately 53×60 pixel cells. We divided orientations into 9 bins from $0°$-$360°$. In contrast to the block-normalization scheme used by Dalal and Triggs, all cells were scaled by the same value so the l^2 norm of the cells was 1.

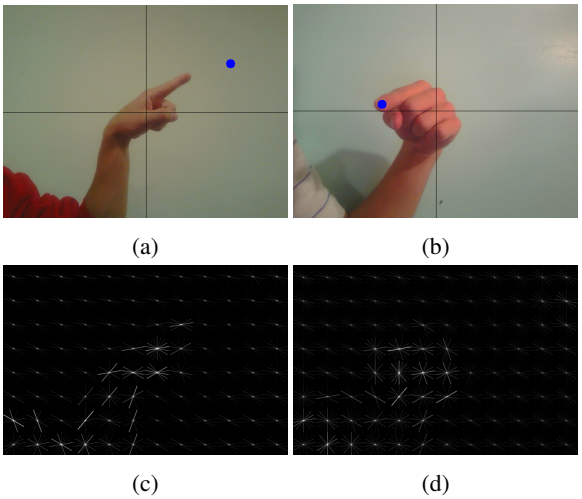


Figure 1: An image (on the left) and our corresponding HOG output (on the right). The location calculated by our algorithm is illustrated by the blue dot on the left image.

3.6. Regression

We experimented with several different including k-Nearest Neighbors and linear regression. We found that a combination of k-nearest neighbors worked best across a wide variety of data. We trained the k-nearest neighbors and linear regression models from sci-kit learn separately on 15 sets of test data comprised of the HOG features computed from each image. These models each predicted a location based on HOG input. These locations were averaged together and fed into a smoothing algorithm to compute the final location.

3.7. Smoothing

In order to make the system more robust to noise, we both smooth the location and eliminate outliers. Let $x[n]$ be the location at frame n and $u[n]$ be the location output from the k-NN and linear regression combination at frame n. Outliers are eliminated if

$$\sqrt{\sum_{i=n-l}^{n} (u[n] - x[i])^2} \geq \epsilon \quad (1)$$

[1]The pretrained model is downloadable from the Caffe Model Zoo under the name 'BVLC Reference CaffeNet'

where l and ϵ are parameters representing the amount of history to remember for computing outliers and the sensitivity to outliers, respectively. We used $l = 10$ and $\epsilon = 200$. Smoothing is implemented by the formula

$$x[n+1] = \begin{cases} \lambda x[n] + (1-\lambda)u[n] & \text{if not an outlier} \\ x[n] & \text{otherwise} \end{cases} \quad (2)$$

where λ is a smoothing parameter between 0 and 1. We used $\lambda = 0.5$ to balance quick responsiveness with robustness to noise.

4. Results

4.1. Performance on test data

4.1.1 Performance of Different Regressions

We evaluated the performance of both convolution neural networks and HOG with k-Nearest Neighbors and Linear Regression. The results from our tests our given in figure 2, which is a histogram of the error of the different classifiers. The error is the euclidean distance measured in pixels between the predicted point and the actual point.

Errors for Convolution Neural Nets	
Regression Type	**Average Error (Pixels)**
k-Nearest Neighbors	47.8
Linear Regression	114.2

Errors for Histogram of Oriented Gradients	
Regression Type	**Average Error (Pixels)**
k-Nearest Neighbors	32.0
Linear Regression	79.7
Combined	47.0

4.2. Evaluation as a Computer Interaction Device

To test our system's effectiveness for computer interaction, we used the HOG feature based system to predict where a user was pointing at a computer screen. We used the prediction to move a dot on the screen, which simulated the movement of a computer mouse. Something about effectiveness fdjslfj

The time to process an image using convolutional neural networks was too high to allow for real time computer interaction, since each frame took several seconds to process. Using a GPU to process frames would greatly increase the speed and should allow for the possibility of using these features in a real time system.

5. Conclusion

References

[1] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005.

[2] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. *arXiv preprint arXiv:1310.1531*, 2013.

[3] A. Erol, G. Bebis, M. Nicolescu, R. D. Boyle, and X. Twombly. Vision-based hand pose estimation: A review. *Computer Vision and Image Understanding*, 108(1):52–73, 2007.

[4] W. T. Freeman and M. Roth. Orientation histograms for hand gesture recognition. In *International Workshop on Automatic Face and Gesture Recognition*, volume 12, pages 296–301, 1995.

[5] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[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.

[7] D. Lee and S. Lee. Vision-based finger action recognition by angle detection and contour analysis. *ETRI Journal*, 33(3):415–422, 2011.

[8] E.-J. Ong and R. Bowden. A boosted classifier tree for hand shape detection. In *Automatic Face and Gesture Recognition, 2004. Proceedings. Sixth IEEE International Conference on*, pages 889–894. IEEE, 2004.

[9] 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.

[10] J. Tompson, M. Stein, Y. Lecun, and K. Perlin. Real-time continuous pose recovery of human hands using convolutional networks. *ACM Transactions on Graphics (TOG)*, 33(5):169, 2014.

[11] M. Van den Bergh, E. Koller-Meier, F. Bosché, and L. Van Gool. Haarlet-based hand gesture recognition for 3d interaction. In *Applications of Computer Vision (WACV), 2009 Workshop on*, pages 1–8. IEEE, 2009.

[12] M. Van den Bergh and L. Van Gool. Combining rgb and tof cameras for real-time 3d hand gesture interaction. In *Applications of Computer Vision (WACV), 2011 IEEE Workshop on*, pages 66–72. IEEE, 2011.

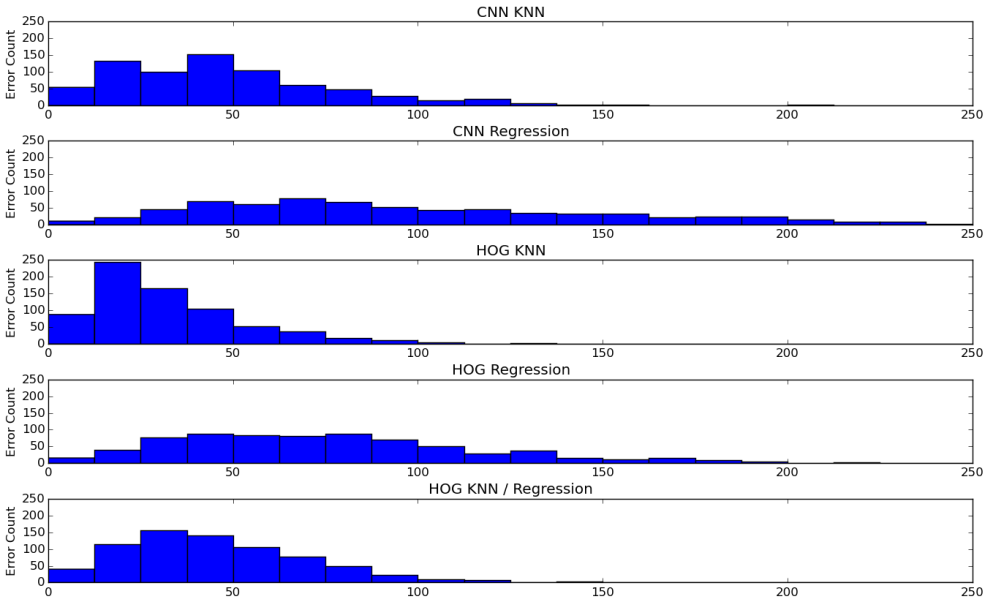
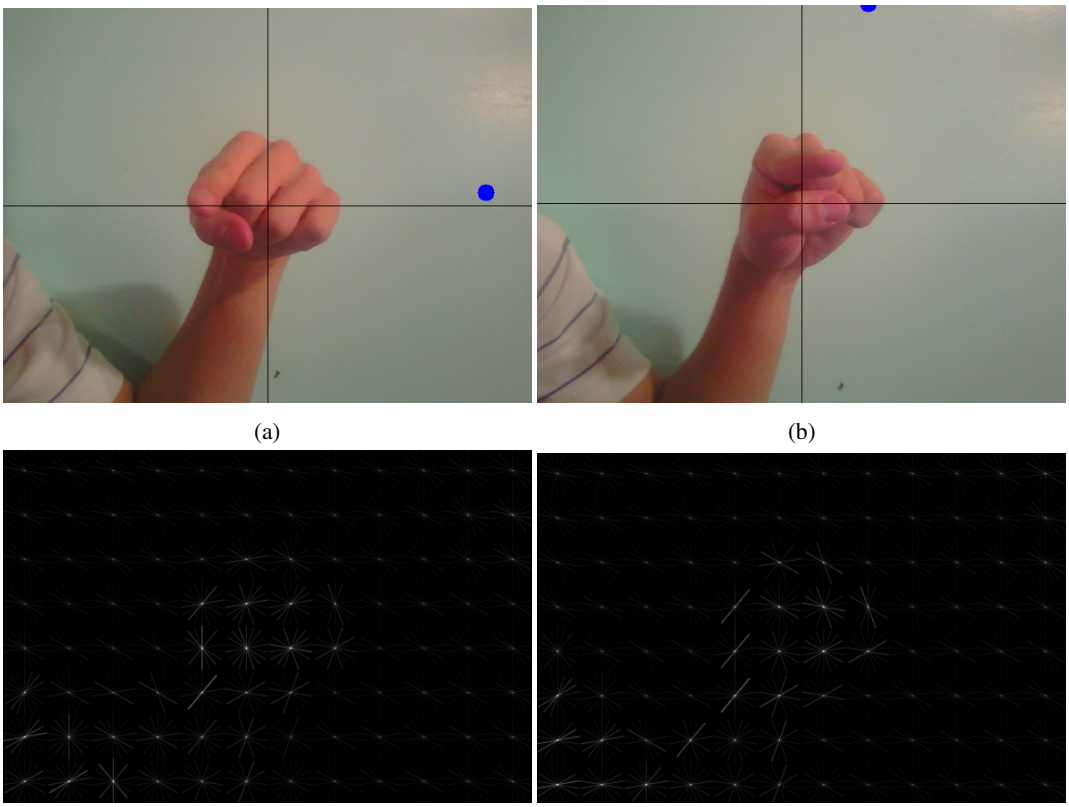

CNN KNN
CNN Regression
HOG KNN
HOG Regression
HOG KNN / Regression
Error Count
0
50
100
150
200
250

Figure 2



(a) (b)

(c) (d)

Figure 3: Examples of images in which our algorithm does not perform as well. The hand is pointing to different points on the screen in images 3a and 3b. However, the HOG outputs in 3c and 3d are almost identical. This happens because the HOG algorithm mostly detects the outline of the hand and not the pointing finger which is facing mostly toward the camera. Because the HOG images are so similar, the regression does not train properly, causing the blue cursor in 3a and 3b to be in the incorrect location.