Hand in the Dark:

Standardized Hand Gesture Recognition

Course Project of Northwestern EECS 349, Spring 2015 (Instructor: Prof. Doug Downey)

Jerry Li, Ivy Zheng, Yiyao Fu

*Computer Science and Material Science, Northwestern University*

jiamingli2017@u.northwestern.edu

*Computer Science and Mathematical Methods in the Social Science, Northwestern University*

xunzheng2017@u.northwestern.edu

*Mathematics and Political Science, Northwestern University*

annieyf@u.northwestern.edu

1. Introduction of Task

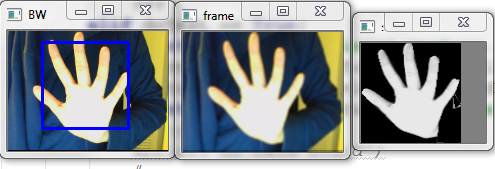
Gesture recognition is to interpret human motion after capturing the image. In this project, the input is a hand gesture picture, and the output is type of gesture as pre-defined in the dataset.

This task is meaningful and desirable for its wide range of applications, including various commands, UI’s and communication programs. For example, people can use gesture to send simple commands to the computer via a camera without touching the keyboard.

1. Information of Dataset

In this project, we focus on gesture recognition of standardized pictures, i.e. how to take standardized hand gesture pictures, and how to classify them.

To create the dataset, we used an OpenCV based public hand HAAR cascade xml file trained for hand detection[[1]](#footnote-1). This file can find the general location of a hand in an image, and around the general location we use the Canny edge detector in OpenCV2 to find the accurate contour of the hand. A gray-scaled, 100x100 image is created around the contour of the hand, with the pixels outside of the contour blackened, as shown in Fg. 1.



F.g. 1: Picture Preprocessing

The complete dataset include 3130 processed pictures of 8 gestures from 7 people, with an average of 391 pictures for each gesture, and 56 pictures per gesture per person. For the purpose of future training, the images are taken by gesture, so that pictures in the same folder belong to the same category and the same person. We rename each folder so it shows the gesture category of the images and the person the gestures are from.

1. Training and Testing

The attribute we use is the grayed pictures, encoded as a list of pixels. Each picture is labeled with a gesture ID, which is also what the algorithm needs to predict. To train the dataset, our code first reads directories’ names as a list of categories, read each picture as a list of pixels, and then label the type of each picture respectively. Lastly we convert the image to binary by coloring all pixels inside the hand white. This action will lessen the noise and improve our performance, as shown in the next section.

We use k-nearest neighbor as our classifier. We think kNN would be a good classifier because of the nature of the dataset: there’s no low dimension linear relation, a lot of noise, strong correlation between attributes, and low human involvement during training. Specifically, we subtract two image matrices and square the difference, find the picture that has the least difference with the input image, and return the gesture type of this picture as the predicted gesture type for the input.

To test our classifier, we use cross validation across different people. Specifically, the cross validation function shuffles our dataset, take images from one persona as testing set, and the rest as training set. Then, our function predicts the gesture type for testing set, validates it, and records the accuracy in a csv file.

1. Experiment Results

On the progress report, we mentioned that we reached a 0.998 accuracy when we tested with only one person’s hand. However, it can only predict another person’s gesture with 0.374 accuracy. The test with high accuracy above is not robust, because gesture varies a lot from person to person. An image of five from one person is usually a good neighbor of his other five’s, but not always a good neighbor of other people’s five.

Aware of this problem, we used hands of seven people in our training set, and we do the validation across different people’s hands. This makes sure that a picture will not find its nearest neighbor another picture of the same hand.

We then tested different k values for kNN method. We tested k from 1 to 10, and reached our best result at k=2, with a precision of 0.630.

We then implemented a function that converts gray-scale images to binary. As mentioned, this would lessen the noise and thus increased the precision to 0.844, which we think is a satisfactory result. After filtering the failed images, we can see that their contours usually have poor quality, as shown below in F.g. 2.

F.g.2: Some examples of failed images

Finally, we test the data using different classifiers on Weka. As a benchmark, we first ran 2 Nearest Neighbor on Weka, with 66% training and the rest testing. We got 63.91% instances classified correctly. We then ran several other classifiers and got the following result:

TABLE 1

Performance of Various Classifiers on Weka

|  |  |
| --- | --- |
| **Classifier** | **Correctly Classified** |
| 2NN | 58.50% |
| Decision Tree | 54.97% |
| Attributed Selected Classfier | 42.41% |
| Neural Net | 46.90% |
| AdaBoost | 23.16% |

This shows how nearest neighbor is a better classifier.

1. Conclusion and Future Work

From the experiment, we can see that our proposed method work well in predicting the hand gesture, and give a better result than many other classifiers, and can potentially run better with a larger dataset.

Considering the potential of this project, there are several parts that can be further improved or developed given more time and energy on it.

1. Since videos are just sequences of images, this project can be modified and extended to recognize moving gestures.
2. Right now, the preprocessing only works well when the background is of uniform color and the contour of the hand is closed. It can potentially be improved and become robust on stronger noise.

1. Details of this xml file can be found in Envision, “HAAR xml file”, at http://nmarkou.blogspot.com/2012/02/haar-xml-file.html

   2http://docs.opencv.org/modules/imgproc/doc/feature\_detection.html [↑](#footnote-ref-1)