CS231A Course Project Final Report
Sign Language Recognition with Unsupervised Feature Learning

Justin Chen
Stanford University
justinkchen@stanford.edu

Abstract

This paper focuses on experimenting with different segmentation approaches and unsupervised learning algorithms to create an accurate sign language recognition model. To more easily approach the problem and obtain reasonable results, we experimented with just up to 10 different classes/letters in the our self-made dataset instead of all 26 possible letters. We collected 12000 RGB images and their corresponding depth data using a Microsoft Kinect. Up to half of the data was fed into the autoencoder to extract features while the other half was used for testing. We achieved a classification accuracy of 98% on a randomly selected set of test data using our trained model. In addition to the work we did on static images, we also created a live demo version of the project which can be run at a little less than 2 seconds per frame to classify signed hand gestures from any person.

1. Introduction

The problem we are investigating is sign language recognition through unsupervised feature learning. Being able to recognize sign language is an interesting computer vision problem while simultaneously being extremely useful for deaf people to interact with people who don't know how to understand American Sign Language (ASL).

Our approach was to first create a data set showing the different hand gestures of the ASL alphabet that we wished to classify. The next step was to segment out only the hand region from each image and then use this data for unsupervised feature learning using an autoencoder, followed by training a softmax classifier for making a decision about which letter is being displayed. Many different segmentation approaches were tried until we discovered that the skin color and depth segmentation techniques worked most consistently.

2. Methodology

2.1. Problem Statement

2.1.1 Background

The CVPR gesture workshop from 2011 provides a great information on modern gesture recognition models as well as how to incorporate different learning algorithms. There is some past work[1] related to our project that we initially looked at such as segmentation-robust modeling for sign language recognition [3] and sign language and human activity recognition [1], but we ended up using mostly our own approach to sign language recognition. Inspiration for our learning model was drawn from the MNIST handwritten digits recognition problem[2] which also used a similar unsupervised feature learning and classification approach [2]. We also investigated the use of convolutional neural networks for feature learning based on the visual document analysis paper

[1] http://clopinet.com/isabelle/Projects/CVPR2011/
[2] eblearn.sourceforge.net/demos/mnist/index.shtml

by the Microsoft Research group [4]. We felt that a simpler approach, however, would still yield competitive accuracy results.

2.1.2 Dataset

The data used for training and testing came from our own self-made dataset. 1200 samples of each of the 10 signed letters (a, b, c, d, e, f, g, h, i, l) were collected using the Kinect including the corresponding depth data. Each sample consists of a person signing the corresponding letter while facing directly at the Kinect camera. This dataset consists of 6000 images used for training and another 6000 images used for testing. The following is a visualization of some of the unprocessed raw images in the dataset:

Figure 1: Example Image in Dataset of (Top Left) letter 'F', (Top Right) letter 'L', (Bottom Left) letter 'I', (Bottom Right) letter "D"

The depth data that was collected along with the RGB data overlayed perfectly on top of the original image and was helpful during the segmentation section of the model. An example is shown below:

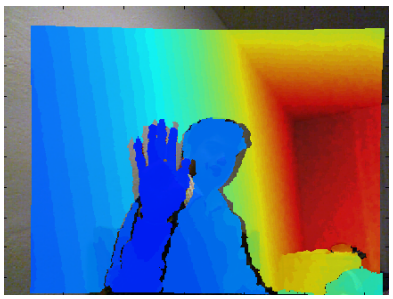

Figure 2: Depth data overlayed on top of RGB data

2.2. Technical Approach

2.2.1 Segmentation Methods

We tried implementing many different versions of quick image segmentation, in particular, hand segmentation.

Edge Segmentation

One of the methods we tried was using a Canny edge detector to find relevant "objects" in the field of view of the camera. The edges were then dilated, and then all remaining holes in the mask were filled to create a solid, continuous mask. Once this was done, only the largest areas were taken in order to remove all the background clutter objects. This approach makes the simplifying assumption that the biggest objects seen in segmentation are typically of the most interest as well.

Figure 3: (Left) Edge detection results, (Right) Edge detection mask applied to image shown by black outline

However, this approach did not always work since there would be instances where the largest "object" with distinguishable edges was not the hand (or there was a busy background which cluttered the output) and therefore segment poorly. We decided to move on to a simpler, but perhaps more accurate method of segmentation using just color.

Skin Color Segmentation

We tried out the two approaches for skin segmentation using only color information. The first approach involved modeling the skin color by a 2D Gaussian curve and then using this fitted Gaussian to estimate the likelihood of a given color pixel being skin. First, we collected skin patches from 40 random images from the internet. Each skin patch was a contiguous rectangular

skin area. Skin patches were collected from people belonging to different ethnicities so that our model is able to correctly predict skin areas for a wide variation of skin color. The colors were then normalized as follows :

$$r = \frac{R}{R+G+B}, b = \frac{B}{R+G+B}.$$

The g component is ignored as it is linearly dependent on the other two. The mean and covariance matrix of the 2D Gaussian (with r, b as the axes) is estimated as follows :

$$\text{Mean } m = E[x], \text{ where } x = [r, b]^T$$
$$\text{Covariance } C = E[(x-m)(x-m)^T].$$

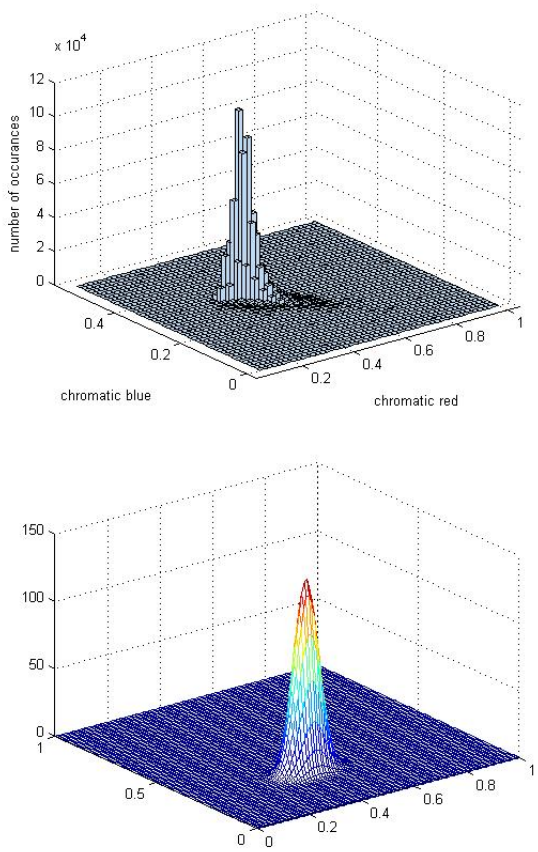

Figure 4: (Top) Histogram of color distribution for skin patches, (Bottom) Gaussian model fit

With this Gaussian fitted skin color model, the likelihood of skin for any pixel of a given test image can be obtained. If the pixel, has a chromatic pair value of (r, b), then the likelihood of skin for this pixel is given by:

$$Likelihood = e^{[-0.5(x-m)^T C^{-1}(x-m)]}, \text{ where } x = [r, b]^T.$$

Finally, we thresholded the likelihood to classify it as skin or non-skin. However, this approach did not give significantly good or consistent results and failed to detect dimly illuminated parts of skin. These poor results did not meet the quality standard of hand segmentation we needed to get a consistent feature extraction from the learning layer of the model.

The second approach which we used is motivated by the paper [5], in which the authors first transform the image from the RGB space to the YIQ and YUQ color spaces. The authors then compute the parameter

$$\Theta = tan^{-1}(V/U)$$

and combine it with the parameter I to define the region to which skin pixels belong. Specifically, the authors called all pixels with $30 < I < 100$ and $105^o < \Theta < 150^o$ as skin. For our experiments, we tweaked these thresholds a bit, and found that the results were significantly better than our Gaussian model in the previous approach. This might have been because of two reasons:

1. The Gaussian model was trained using data samples of insufficient variety and hence was inadequate to correctly detect skin pixels of darker shades

2. Fitting the model in the RGB space performs poorly as RGB doesnt capture the hue and saturation information of each pixel separately.

Figure 5: (Left) Skin detected using Gaussian model, (Right) Skin detected using YIQ and YUV color spaces

Having detected the skin regions quite accurately, we then further filtered out the hand region and eliminated the face/other background pixels that might have been detected by using the corresponding collected depth data. We assumed that in any given frame, the hand was the object of interest and therefore the closest skin-colored object in the camera's view. We then created a secondary dataset out of this segmentation model which consisted of hand gestures for ten letters of the ASL alphabet. Each image was cropped and resized to be a square 32x32 bounded area.

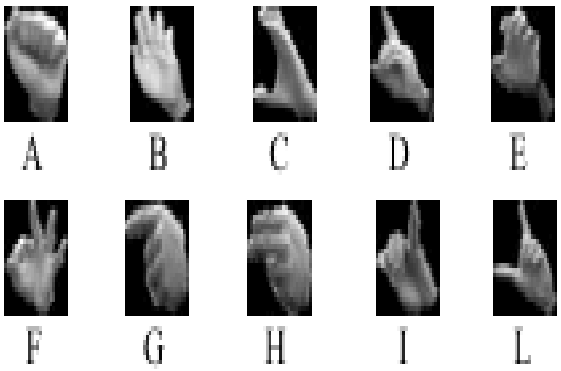

Figure 6: Visualization of processed database images

Further Segmentation Work

We explored further segmentation methods, but eventually deemed them of less importance than the original skin-color hand segmentation to get an accurate ASL recognition system. One of the other approaches was attempting to properly segment pointed fingers within the detected hand. In order to segment out the fingers, we used convex hull detections to find the possible "fingers" after the hand has already been segmented out. The fingers will ideally be oriented along the direction from the convex hull point to the centroid of the hand as seen in Figure 7.

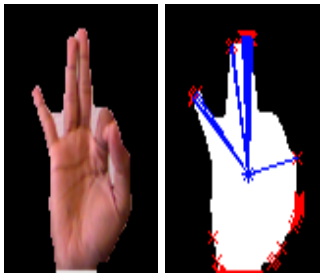

Figure 7: (Left) Skin model segmentation, (Right) Using convex hull detection to find potential "fingers"

However, we never ended up using the detected fingers as part of the classification approach because of the slightly unpredictable detections of convex hulls in a hand image.

2.2.2 Feature Learning and Classification

The extracted data of hand images were fed into an autoencoder in order to perform the actual recognition training part of the project. This stage implements an unsupervised learning algorithm. We feed all the data samples into the sparse autoencoder. The input data from the segmentation block are images of size 32x32 pixels. A sparse autoencoder is chosen initially with an input layer with 32x32 nodes and one hidden layer of 100 nodes. We used L-BFGS to optimize the cost function. This was run for about 400 iterations to obtain estimates of the weights. Now the autoencoder has learnt a set of features similar to edges. A visualization of the learned features can be seen below:

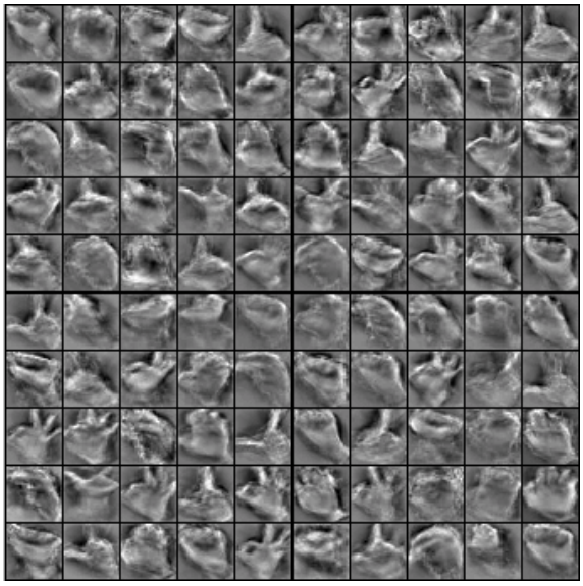

Figure 8: Visualization of sparse autoencoder features

The next step is to classify the 10 different letters based on the features learnt by the autoencoder training. The output of the hidden layer of the autoencoder is fed into a softmax classifier

to now classify the data into 10 categories. The softmax classifier again learns using the L-BFGS optimization function. This algorithm converges after about 40 iterations. We tested the system accuracy by using the remaining 600 images per letter (for a total of 6000 images) as our test set.

An overall view of the system's block diagram can be seen here:

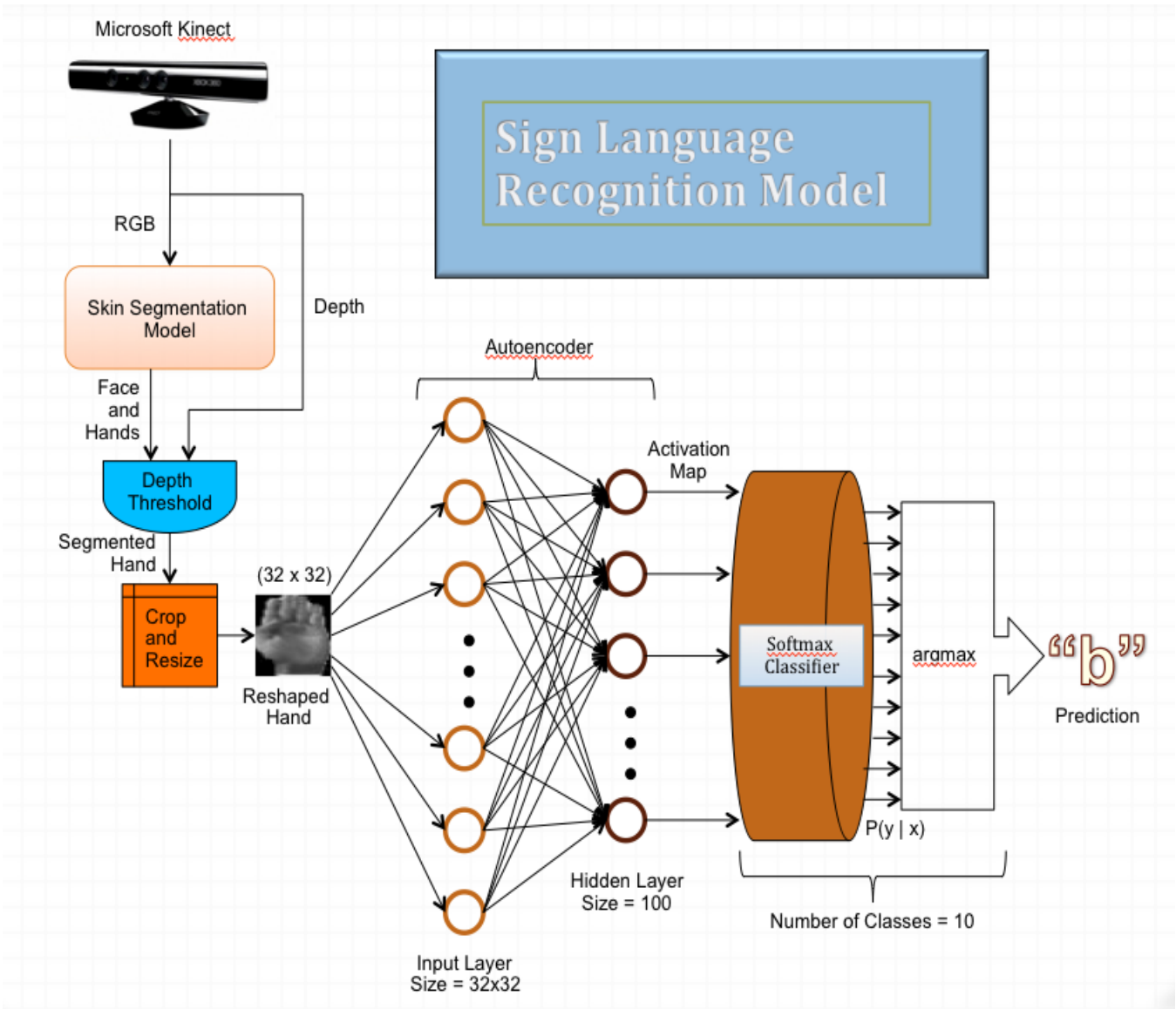

Figure 9: Block diagram summarizing our approach for sign language recognition

3. Experimental Results and Discussions

In the previous sections, we have mentioned details of our implementation of the hand segmentation, unsupervised feature learning and classification sub-blocks. In this section, we report the performance of our system through tables and figures. Our primary evaluation metric is based on classification accuracy. Given an unsegmented image of a person signing a letter, we want to see what the accuracy of our model is in classifying/predicting the signed letter in the image. Achieving a classification accuracy around 98% similar to many MNIST digit recognition scores is desired.

As a preliminary diagnostic, we plotted a learning curve showing the training error and the test error as a function of the size of the training set. The following plot shows the learning curve we obtained:

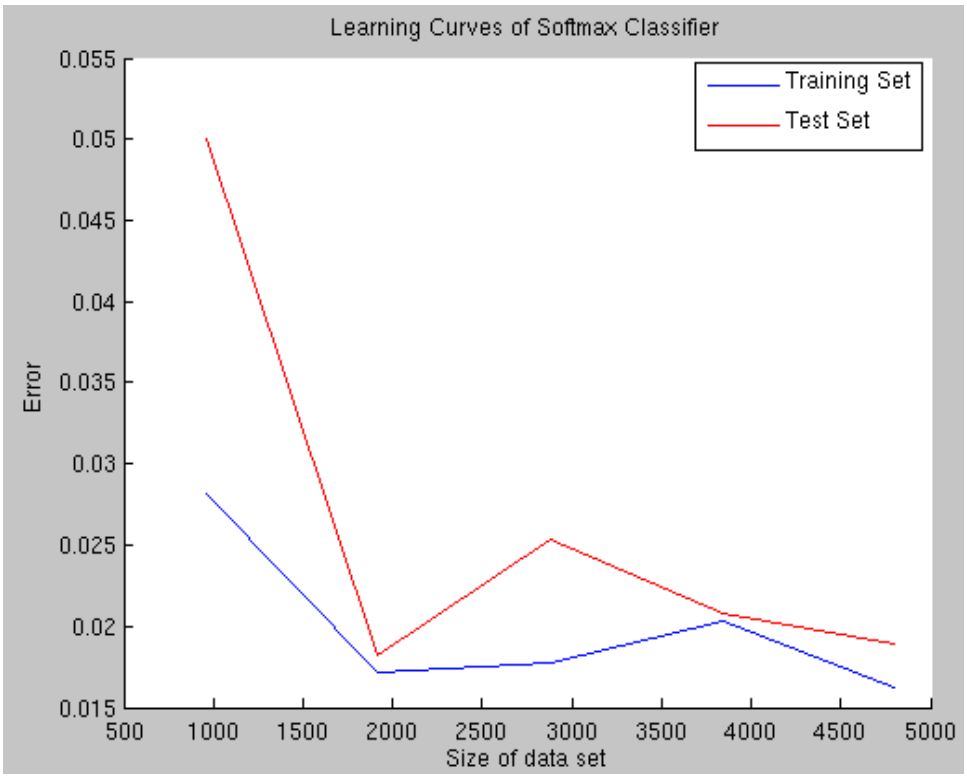

Figure 10: Learning curve of sign language recognition system

In our milestone report, we had used 50 hidden units for our autoencoder. Analyzing our learning curve, we observe that the training error and the test error are close to each other (except for one aberration at training set size 3000), and even the training error is more than 1.5%, which is somewhat high. Hence suspecting that we might be in the high bias region, we decided to increase the size of our features by increasing the number of hidden units of our autoencoder to 100. Our final classification accuracy on the test set achieved using this 100 length feature vector was 98.2%.The following table summarizes all our implementation details and reports the accuracy obtained :

Size of training set	Size of feature vector	Number of classes (letters)	Accuracy of classification (%)
1200	100	10	95
2400	100	10	98.18
3600	100	10	97.47
4800	100	10	97.92
6000	100	10	98.20

Table 1: Classification Accuracy Evaluation Results on Test Set

The accuracy results that we obtained are comparable to the accuracy of digit recognition on the MNIST dataset and therefore we believe that our

approach works relatively well.

As a finishing step to our project, we have successfully created a real time implementation of our entire system, so that hand gestures made in front of the Kinect connected to our computer directly displayed the image captured by the kinect, the segmented hand gesture and the output of our classifier, which is one of the ten letters in our dataset. The evaluation process takes less than 2 seconds per frame. The following figures show screenshots of our real-time implementation and the results obtained. In each screenshot, the original image is shown with the result of the segmentation and the predicted result to the right of it.

Figure 11: Example screenshots of real-time live demo sign language recognition system interface

4. Conclusion and Future Work

In this project, we have implemented an automatic sign language gesture recognition system in real-time, using tools learnt in computer vision and machine learning. We learned about how sometimes basic approaches work better than complicated approaches. Despite trying to use a smart segmentation algorithm, the relatively basic skin segmentation model turned out to extract

the best skin masks. We also realized the time-constraints and difficulties of creating a dataset from scratch. Looking back, it would have been nice to have had a dataset already to work off of. Some letters were harder to classify in our live demo such as ”a” vs ”i” since they only differ by a very small edge (the ”i” has the pinky pointing up). Although our classification system works quite well as has been demonstrated through tables and images, theres still a lot of scope for possible future work.

Possible extensions to this project would be extending the gesture recognition system to all alphabets of the ASL and other non-alphabet gestures as well. Having used MATLAB as the platform for implementation, we feel that we can also improve upon the speed of our real-time system by coding in C. The framework of this project can also be extended to several other applications like controlling robot navigation using hand gestures and the like.

References

[1] D. Metaxas. Sign language and human activity recognition, June 2011. CVPR Workshop on Gesture Recognition.

[2] M. Ranzato. Efcient learning of sparse representations with an energy-based model, 2006. Courant Institute of Mathematical Sciences.

[3] S. Sarkar. Segmentation-robust representations, matching, and modeling for sign language recognition, June 2011. CVPR Workshop on Gesture Recognition, Co-authors: Barbara Loeding, Ruiduo Yang, Sunita Nayak, Ayush Parashar.

[4] P. Y. Simard. Best practices for convolutional neural networks applied to visual document analysis, August 2003. Seventh International Conference on Document Analysis and Recognition.

[5] X. Teng. A hand gesture recognition system based on local linear embedding, April 2005. Journal of Visual Languages and Computing.

5. Appendix

This project is done in combination with the CS229 Machine Learning final project. The CS231A Computer Vision primary component is the hand and finger segmentation using 3D camera.