CS229 Project Final Report

Sign Language Gesture Recognition with Unsupervised Feature Learning

Justin K. Chen, Debabrata Sengupta, Rukmani Ravi Sundaram

1. Introduction

The problem we are investigating is sign language recognition through unsupervised feature learning. Being able to recognize sign language is an interesting machine learning 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 wish 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. Our final dataset consists of ten letters of the ASL alphabet, namely a, b, c, d, e, f, g, h, i and l. The following block diagram (Figure 1) summarizes our approach.

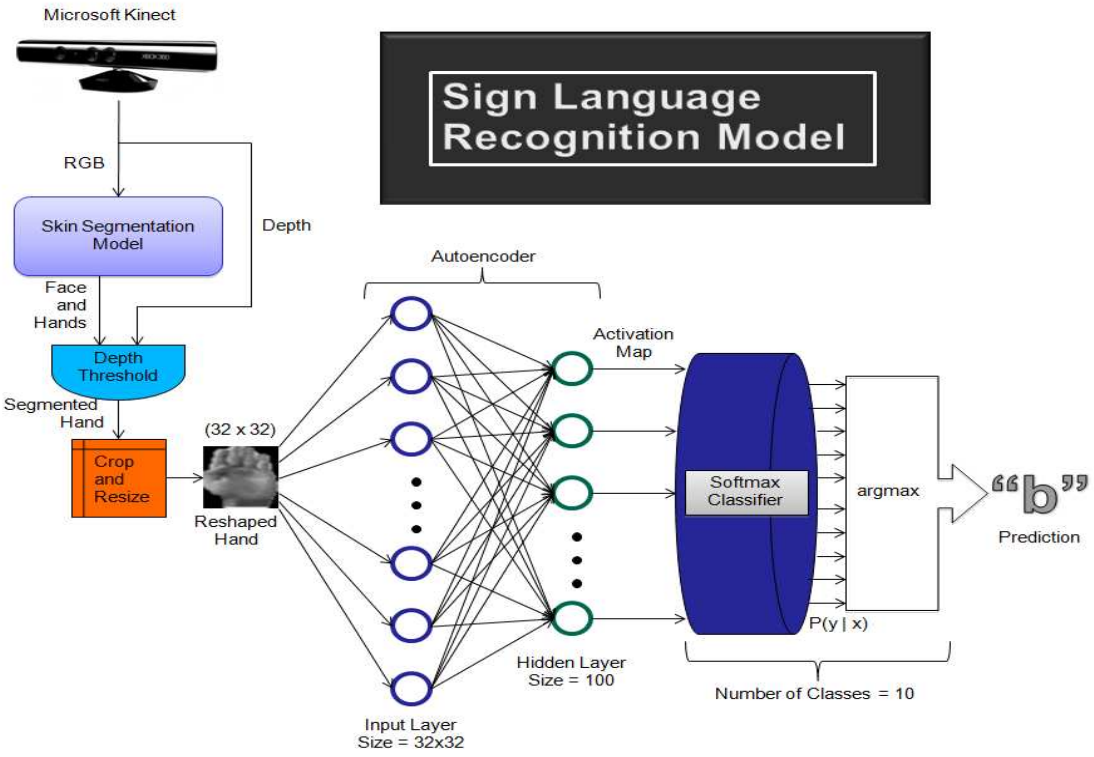

Fig. 1: Block diagram summarizing our approach for sign language recognition

2. Methodology

2.1 Data collection and Hand segmentation

The data that we used was collected off of a Microsoft Kinect 3D depth camera. Videos were taken of the test subject's hands while forming sign language letters. Frames showing individual letters were extracted. For segmenting out only the hand, we tried several approaches as described below.

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. We first collected skin patches from 40 random images from the internet. Each skin patch was a contiguous rectangular skin area. The 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. We first normalized each color as follows : r=R/(R+G+B), b=B/(R+G+B). We ignored the *g* component as it is linearly dependant on the other two. We then estimated the mean and covariance matrix of the 2D Gaussian (with r, b as the axes) as Mean $\mu = E(x)$, Covariance $C = E(x - \mu)(x - \mu)^T$, where x is the matrix with each row being the r and b values of a pixel.

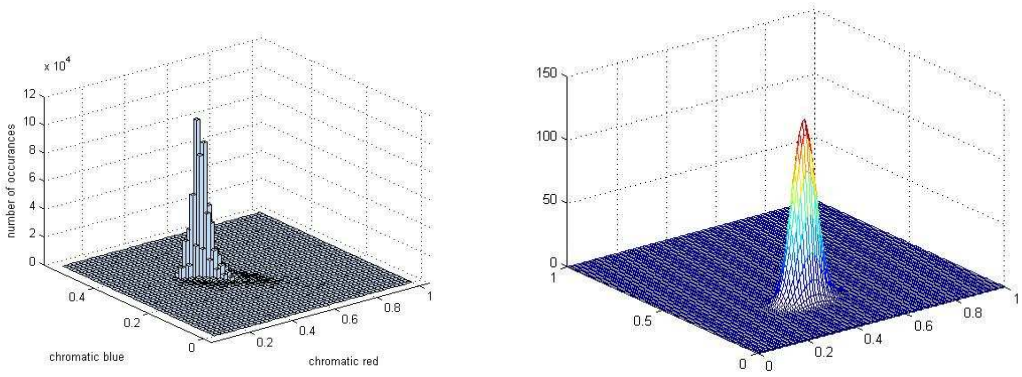

Fig 2: Histogram of the color distribution for skin patches and the corresponding Gaussian model that was fit to it.

With this Gaussian fitted skin color model, we computed the likelihood of skin for any pixel of a given test image. Finally, we thresholded the likelihood to classify it as skin or non-skin. However, this approach did not give significantly good results and failed to detect dimly illuminated parts of skin. The results of using this algorithm for skin segmentation are shown in Fig 3 (b).

The second approach which we used is motivated by the paper [1], in which the authors first transform the image from the RGB space to the YIQ and YUQ color spaces. Then the 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 – (i) our Gaussian model was trained using data samples of insufficient variety and hence was inadequate to correctly detect skin pixels of darker shades (ii) fitting the model in the RGB space performs poorly as RGB doesn't capture the hue and saturation information of each pixel separately. The results of this second approach is shown in Fig 3(c).

After having detected the skin regions, the next task was to filter out only the hand region and eliminate the face and other background pixels that might have been detected. For this, we used the depth information that we

obtained from the Kinect, which returns a grayscale depth image with objects having lower intensity values being closer to the camera. We assumed that in any frame, the hand was the object closest to the camera, and used this assumption to segment out only the hand. Our final dataset consists of hand gestures for ten letters of the ASL alphabet bounded in a 32x32 bounding box, as shown in Fig 4. We have 1200 samples for each letter, giving us a total size of 12,000 images for our entire dataset of ten letters.

Fig 3(a) Original Image

Fig 3(b) Skin detected using the Gaussian model

Fig 3(c) Skin detected using YIQ and YUV color spaces

A B C D E F G H I L

Fig 4. Samples of each letter from our final dataset

2.2 Unsupervised Feature Learning and Classification

The extracted data of hand images was fed into an autoencoder in order to learn a set of unsupervised features. We used 600 images of each letter (so, a total of 6000 images) as training data samples and fed them into the sparse autoencoder. Each input image from the segmentation block are images of size 32x32 pixels. A sparse autoencoder is chosen 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. Visualization of the learned weights of the autoencoder is shown in Fig.5. From this figure, it becomes evident that the autoencoder learns a set of features similar to edges.

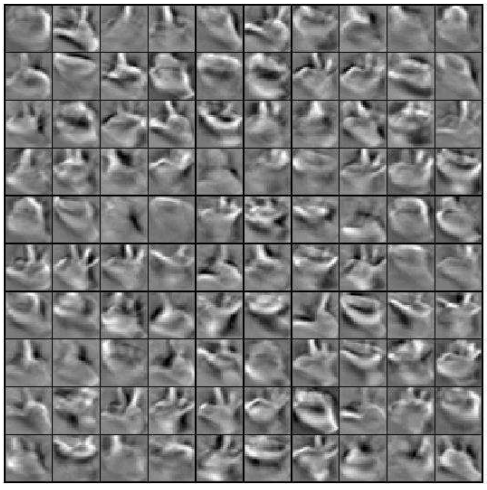

Fig 5 : Visualization of the learned weights of the autoencoder

The next step was to train a softmax classifier in order to classify the 10 different letters based on the features learnt by the autoencoder. The activations of the hidden layer of the autoencoder was fed into a softmax classifier. The softmax classifier again learns using the L-BFGS optimization function. This algorithm converges after about 40 iterations. We tested the system classification accuracy by using the remaining 600 images per letter (so a total of 6000 images) as our test set.

3. 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.

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

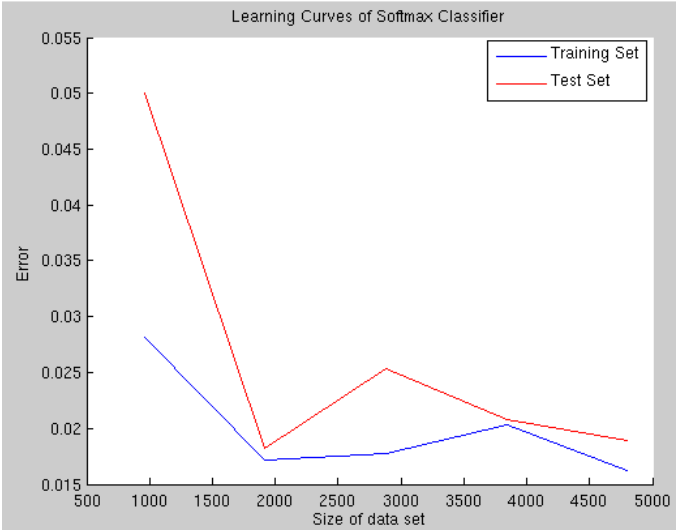

Fig. 6: Learning curve

In our milestone report, we had used 50 hidden units for our autoencoder. Analyzing our learning curve, we observed 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 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	95.62
2400	100	10	98.18
3600	100	10	97.47
4800	100	10	97.92
6000	100	10	98.20

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 examples of our real-time implementation and the results obtained.

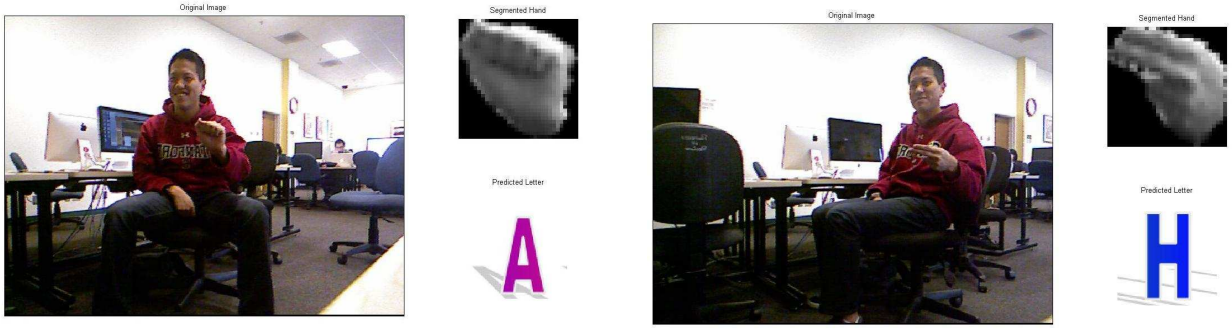

Fig. 7 : Examples of prediction from our live demo

4. Concluding Remarks and Future Work

In this project, we have implemented an automatic sign language gesture recognition system in real-time, using tools learnt in CS229. Although our system works quite well as has been demonstrated through tables and images, there's 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 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.

5. Acknowledgements

We would like to thank Prof. Andrew Ng and the TA's of CS229 for the valuable advice they provided from time to time.

References

[1] Xiaolong Teng, Biani Wu, Weiwei Yu and Chongqing Liu. A Hand Gesture recognition system based on local linear embedding, April 2005.

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

[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

Appendix

This project was being done in combination with Justin Chen's CS231A Computer Vision project.