ASL Fingerspelling Interpretation on Android

Hans Magnus Ewald
Department of Electrical Engineering
Stanford University
Email: hmewald@stanford.edu

Ishan Patil
Department of Electrical Engineering
Stanford University
Email: iapatil@stanford.edu

Shalini Ranmuthu
Department of Electrical Engineering
Stanford University
Email: shalinir@stanford.edu

***Abstract*—In this project, we develop a real time ASL Fingerspelling translator using image processing and supervised machine learning. More particularly, in the finished system an Android smartphone is used to capture an image of a hand gesture, which is the interpreted on an online server. The final system is able to classify 24 different symbols with reasonable accuracy.**

I. Introduction

American Sign Language (ASL) is one of the main forms of communication among the deaf communities in United States and Canada. This motivates us to develop a translator that can recognize hand symbols and find the corresponding meaning. This would be useful aid for communication between deaf or mute people and those unfamiliar with sign language.

A subset of ASL is the American Manual Alphabet - referred to as fingerspelling - which is used to spell out the 26 different letters of the English language using unique hand gestures [1]. Of these symbols, 24 are static hand postures - the exceptions being 'j' and 'z'. In order to avoid excessive complexity, this project will restrict itself to the reduced 24-letter alphabet excluding 'j' and 'z'.

Our goal is to develop a complete system and implement that system in an mobile Android application. However, this project is also exploratory in the sense that methods beyond what is used in the test system will be researched and evaluated. The structure we are using takes an RGB image as input and gives one of the 24 letters as output.

Since this project is combined from CS 229 and EE 368 (Digital Image Processing), we declare here that all work on feature extraction and client-server communication to be considered part of the image processing project only. The sections discussing these topics are a brief summation included only for a better understanding of the whole system.

II. Related Work

On the particular problem of fingerspelling interpretation, there has been work done in the recent past. For instance, [2] describes a method of fingerspelling translation using a Kinect sensor. [5], the paper accompanying our dataset, also includes a description feature extraction using moment invariants. These are cases of feature-based methods of translation. Other approaches are known to employ methods based on deep learning with convolutional neural networks (CNNs), where feature extraction is unnecessary since CNNs are end-to-end solutions. Recently these deep learning approaches take up a dominant position in sign language detection and other computer vision classification problems.

The approach taken here represents an attempt to create an effective feature-based detector and validate its effectiveness as compared to the deep learning methods. Furthermore, the mobile implementation of the translation system imposes different circumstances than many other projects. Although the difficulty of the problem is increased compared to the use of better hardware (such as a Kinect system which has both a camera and a time-of-flight sensor), the tradeoff will greatly improve the practical usability of the system.

As for how we will implement classification, our sources also provide some information. [2] suggests the use of Random Decision Forests, as it is a viable method of combining many weaker features. However, there is a number of commonly used classifiers that we will also consider, such as k-nearest-neighbors and Gaussian Discriminant Analysis.

III. Data-set and Features

This section introduces the popular publicly available data-sets for the static ASL hand symbols and then delves into the two main feature extraction techniques used in this project.

A. Data-sets

We have considered two different data-sets, particularly, [4] from the University of Surrey and [5] from Massey University. The former has $\approx$ 400 images per letter per user for the 24 static letter symbols and 5 users under varying illumination and hand rotation conditions. The latter has $\approx$ 14 images per letter per user for the 24 static letter symbols and 5 users under varying illumination only. A sample image from each can be seen in figure 1. The major difference other than size of images and hand rotation variance is that the large data-set has images of very poor resolution and there is no segmentation of the hands from the cluttered background. On the other hand, the second data-set has good resolution and segmented hand images. Although the first data-set would generalize well, we ultimately trained our model using the second data-set to ensure a more accurate prediction of the live images captured using the Android smart-phone.

B. Feature Extraction

For extracting features from both the training/test images, we employ a combination of HOCD and Gabor filters. Gabor filters provide a very accurate and sensitive feature extraction,

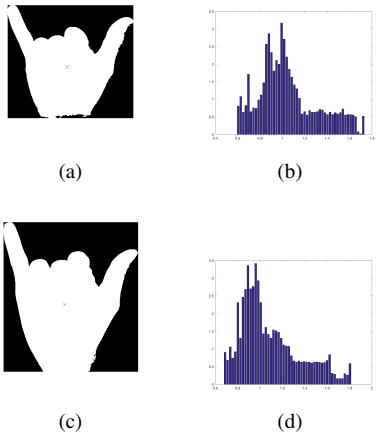
(a) (b)

Fig. 1. (a) Sample image of letter 'a' from data-set [4] (b) Sample image of letter 'a' from data-set [5].

while HOCD adds a robustness to rotation and scale changes of the input images.

1) HOCD: The Histogram of Centroid Distances (HOCD) captures the information of the shape of the hand and converts it to a feature representation. Morphological image processing is performed to isolate a bounding box around the binary mask of the segmented hand in the image. The distance from the centroid of the hand to each pixel on the edge of the hand is computed and then normalized. A histogram of $n = 50$ bins is generated from these distances and is the equivalent representation of the the hand in the feature space (where n denotes the dimensionality of the feature vector).

The closeness of the two output histograms in Figure 2, one corresponding to a training image and other corresponding to the real test image show that HOCD features indeed capture good shape information. But, for some fingerspelling gestures like 'm' and 'n' which have almost similar shape, HOCD would give a similar feature vector and hence, we would have a bad classification using only this features. This motivated us to look for other options for feature extraction.



(a) (b)

(c) (d)

Fig. 2. (a) Binary mask for dataset image of letter 'y';(b) Histogram computed by HOCD algorithm on the binary mask of dataset image; (a) Binary mask for test image of letter 'y';(b) Histogram computed by HOCD algorithm on the binary mask of test image

2) Gabor Filters: Another feature extraction method is the Gabor filter used in [2]. The principle is to process the image with a set of filters and then sample the filter responses to create image features. This filter is similar in function to the common gradient operators, but with expanded flexibility in terms of scale and orientation of the kernel.

In order to detect gradients of many different scales and orientations, we apply a bank of 16 differently parameterized filters to the image. The scale and orientation of these filters are drawn from a vector of four scales and a vector of four orientations in all possible combinations. Using two-dimensional Gaussian functions, we then extract 64 local averages of the complex magnitude of each of the 16 Gabor filter responses on an 8 by 8 grid. This results in 1024 features per image which are effective at determining image similarities. However, it should be noted that this feature extraction system is not invariant to rotation, scaling, and translational changes. This problem can be addressed by combining both feature extraction methods seen in this section.

IV. Methods

A. Skin Segmentation

We originally began our project by using data-set [4], which required segmentation of the hand from a cluttered background. In order to detect the hand and create a binary segmentation mask for each image, we chose to implement a likelihood model of the rg chromaticity values of skin pixels. A maximum a priori model could also have been used if segmented hand maps were available for each image used in creating the model. For the likelihood model, we identified 50 samples of skin sections from the training images set. The normalized red and green channel (r and g) values were calculated, and a 2-dimensional histogram was then generated. The model can be represented as $\mathcal{L}(\theta|x) = P(x|\theta)$ where θ is the r and g pixel values and x is whether the pixel is skin or background.

To then detect the hand in a new image, we calculate the r and g values for each pixel in the input image and look up the corresponding likelihood value from the histogram in order to create a likelihood map of the image (higher values indicating skin). We then threshold the likelihood map and use various morphological image processing techniques to create a more accurate binary image (Figure 3).

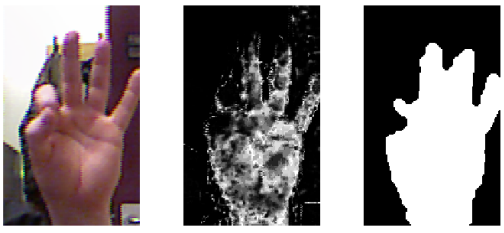


Fig. 3. Original image (left), likelihood map (middle), and binary mask of hand (right)

This procedure does not create a perfect mask and tends to miss areas of the hand with particularly harsh illumination or deep shadow. The imperfect nature of this resulting segmentation mask reduced the ability of our classification

methods to effectively separate and classify different ASL letters. Therefore, we chose to implement our algorithm on images from data-set [5] that consisted of a hand in front of a black background, thereby forgoing the need to use skin segmentation. However, we include this discussion of skin segmentation as it is critical in achieving a more practical implementation of a mobile ASL interpreter, and could easily be improved if a priori segmentations were available.

B. Classification Algorithms

Four different classification models were tried, namely, Random Forests, Support Vector Machine (SVM), Linear Discriminant Analysis (LDA) and K-nearest neighbors (KNN). The implementations of each are described below:

1) Random Forests: For feature vectors of length n, the Random Decision Forest is a set of 50 decision trees composed of a set of n decision stumps. We arrived at this number of trees through a few comparative tests. Each tree uses a randomly permuted sequence of the features, optimizing each decision limit to best separate the classes in the training set. Each branch of the tree leads to a decision on one of the 24 classes. Predictions on new data are decided by simple majority among the trees. In essence this is the same as using boosting with decision stumps, except that we apply the algorithm with 50 random sequences of features. The idea is that averaging over 50 results will lower variance and help regularize our classifier.

2) SVM: The Round Robin Classification method [6] with Error Correcting Codes (ECOC) as implemented in `MATLAB` is used to perform multi-class classification using SVM. More particularly, $\frac{24(24-1)}{2} = 276$ binary SVM models (for 24 unique class labels) are fit using the one-versus-one coding design. This means, for each of the 276 binary learners, one class is positive, another is negative and the rest are ignored. A new observation is then assigned to a class that minimizes the aggregation of the losses for the 276 binary learners.

3) Linear Discriminant Analysis: In discriminant analysis, the model assumes that the data comes from a Gaussian mixture distribution. In linear discriminant analysis especially, the assumption is that the co-variance matrix remains the same for the Gaussian distribution for data from each class. Only the mean of the distribution changes as per the particular class. A new observation in this model is then assigned to the class that minimizes the classification cost-

$$y_{obs} = \underset{y=1,2...24}{\operatorname{argmin}} \sum_{i=1}^{24} P(i|x)C(y|i)$$

where $P(i|x)$ is the posterior probability of class i given the observation x and $C(y|i)$ is the cost of classifying an observation as y when its class is i.

4) K-Nearest Neighbors: In the training phase, this model just stores the feature vectors along with corresponding class labels for each training example that is shown to it. In the classification (prediction) phase, it assigns that class label which is most common among the K nearest neighbors of the observation's feature vector. The algorithm is called Fine KNN when the value of K is selected to be 1.

C. Hand Model

A method that has been extensively researched is a posture detection method based on a mathematical 20-degree-of-freedom (DoF) model of the hand skeleton. This method is similar to the one described by [3], but all work here has been built from the ground up. The designed 3D model and its projection onto a 2D surface is showcased in figure 4

The key to the actual feature extraction process now lies in manipulating this established model to align with a detected hand silhouette from an input image. The parameters of the optimal fit represent the estimate of the hand's posture and are theoretically well suited to distinguish all 24 of the hand symbol postures. The fit is a learning problem, and for the solution we have explored the use of gradient ascent on a quality function we define on the detected hand shape:

$$\theta^{(t+1)} = \theta^{(t)} + \alpha \nabla_\theta G(\theta)$$

The preliminary quality function used for the position of each skeleton point (with $B[x, y]$ a binary mask of the segmented hand) is

$$Q(x_p, y_p) = \sum_{i}^{N_x} \sum_{j}^{N_y} S_{ij}(x_p, y_p) \cdot B[i, j]$$

$$S_{ij}(x, y) = \begin{cases} \frac{a_S}{2}\left(\cos\left(\frac{\pi}{d_S} r_{ij}(x, y)\right) + 1\right) : r_{ij}(x, y) \leq d_S \\ 0 : r_{ij}(x, y) > d_S \end{cases}$$

$$r_{ij}(x, y) = (x - i)^2 + (y - j)^2$$

This results in a quality function with a continuous gradient and where only pixels withing a radius of d_S need to be considered. The complete quality function comes together as a sum over the individual quality of all skeleton points, the location of each point resulting from the parameters θ:

$$G(\theta) = \sum_{p} Q(x_p(\theta), y_p(\theta))$$

This system did however prove difficult to realize fully in the given time constraints. Also, the pixel-wise definition of the quality function resulted in very inefficient computations for both the value of the error function and its gradient. As such, it would seem that the complexity of the optimization problem required by this feature extraction method is so far its biggest drawback.

V. Experiments and Discussion

A. PCA Visualization

Reducing the dimensionality of our features would allow us to visualize the data and observe the distribution of the 24 different letter classes in a lower-dimensional feature space. Principal component analysis (PCA) reduces dimensionality by finding principal components that maximize the variance of the data while remaining orthogonal to all other principal

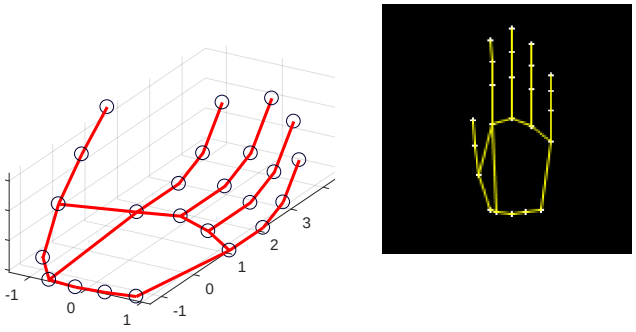


Fig. 4. To the left is a test of the 3D hand model with the circles indicating the joints. To the right is a projection of that hand into a 128 pixel square image.

components. The direction of the first component will thus satisfy the following equation (where u is the unit vector, m is the number of training examples, and $x^{(i)}$ is a feature):

$$\arg \max_{||u||_2=1} \frac{1}{m} \sum_{i=1}^{m} (x^{(i)} u)^2$$

We implemented PCA by pre-processing the data to normalize mean and variance, and then performing singular value decomposition (SVD) to generate the principal components. For visualization, we chose to reduce our 1074-dimensional data to only the first two principal components as seen in Figure 5. We can see that the classes are not cleanly separated in this two-dimensional space, but do form clusters which is promising considering that we will use more than two-dimensional features for our classification results.

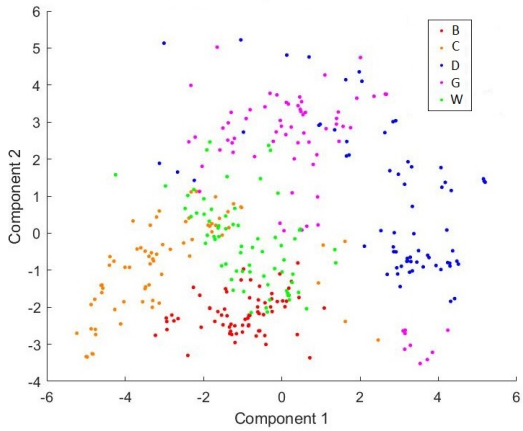


Fig. 5. Two-dimensional PCA visualization on subset of 5 letters

In addition to visualization, PCA can also be used for data compression prior to classification. We experimented with this by reducing the 1074-dimensional data to 34 dimensions in order to retain 95% variance of the data. However, when we implemented classification with PCA, the performance was dramatically reduced as seen in Figure 6. For this reason, used PCA only for data visualization.

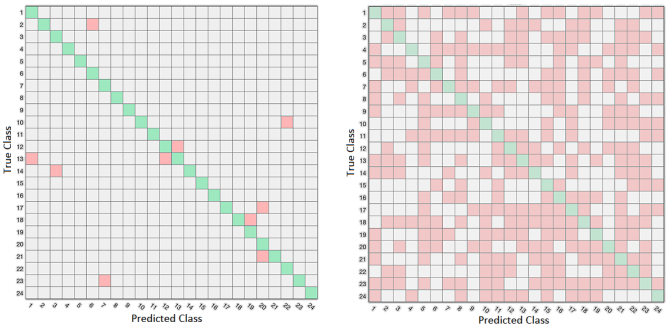


Fig. 6. Confusion matrices without PCA (left) and with PCA (right) generated using Fine KNN

B. Results and Error Analysis

To train the model for classification, both Gabor and HOCD features were extracted from each of the training images (from the data-set [4], which contains $\approx$ 1675 images from $\approx$ 70 images per letter for each of the 24 letters). For each image, both sets of features were concatenated into a single feature vector and fed into each of the classification models discussed above along with the corresponding class (letter) labels. Each of these models were then tested on a random combination of test images, which were taken by the Android Phone of our own hands against a black background (to mirror the performance in the final setup). The results for this analysis are presented in Table I. The classification model was generated by performing a 5-fold cross validation on the training set. The precision and recall metrics were then computed for each class and then the average of all these individual values is reported in the table. As can be seen, the two top performing classifiers on the training set are SVM and KNN with very high precision and recall values. But, when these models (SVM and KNN) were used to predict the class of the actual test images, KNN outperforms SVM. Hence, KNN is chosen for final implementation of the system. It should be noted that the fine KNN algorithm is used where $K = 1$, as discussed in the above section.

TABLE I
ERROR ANALYSIS WITH DIFFERENT COMBINATIONS OF FEATURES

Classifier	Precision	Recall	Accuracy on Test Images
Random Forest	0.9844	0.9840	60%
SVM	0.9931	0.9928	50%
LDA	0.9503	0.9498	40%
KNN	1	1	80%

In a separate set of experiments, the KNN classifier was also tested on different combination of features as seen in Table II. HOCD is inherently more robust to rotation and scale changes, but its generalization error is the highest of the three. The generalization error on the training set is lowest when using Gabor filters, but the combination of both HOCD and Gabor gives the lowest test error of 0.2. This indicates that the Gabor features themselves overfit the training set and don't

generalize well to the actual implementation. In short, the experiment serves to justify our intuition that using both shape information via HOCD and gradient information of the hand features in tandem give the best performance. To give a sense of the generalization error using the combination features, a confusion matrix for the 24 class labels is shown in Figure 7. Letter like ”m” and ”n” in the very middle are often confused (which is understandable given that they look very similar).

TABLE II
ERROR ANALYSIS WITH DIFFERENT COMBINATIONS OF FEATURES

Feature Extractors	Generalization error	Test error
HOCD	0.1940	0.6
Gabor Filter	0.009	0.5
HOCD + Gabor Filter	0.0107	0.2

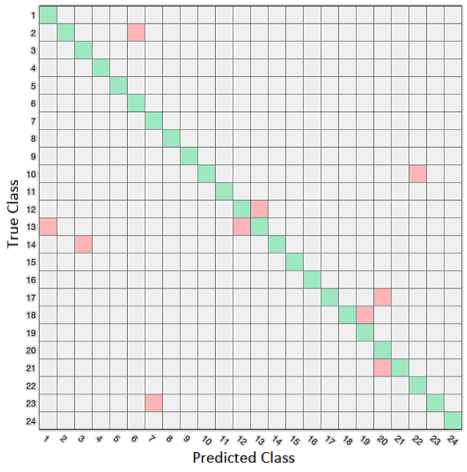


Fig. 7. Confusion Matrix for generalization on training set using HOCD and Gabor Filters in combination as feature extractors

C. Implementation of Prediction Pipeline

The main step in the implementation was to be able to classify the user's ASL images using the pre-trained KNN classification model. We accomplished this by using the steps shown in Figure 8.

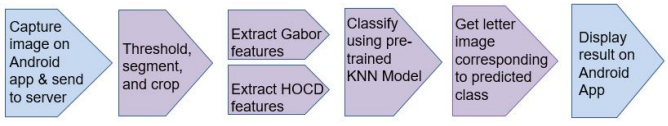
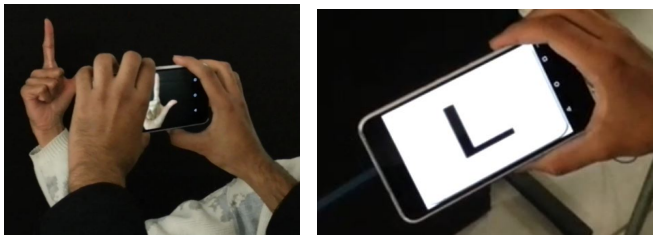


Fig. 8. Steps in prediction pipeline with Android platform in blue and server in purple

In Figure 8, the first and last steps in blue were performed on a mobile Android application (as seen in Figure 9), while the intermediate steps in purple were performed by MATLAB code that was running on a remote server. We chose to run our MATLAB code on Stanford University's FarmShare corn server, given that it had an easily accessible MATLAB module. Using the framework suggested in [7], we set up an Android application as the client which takes a photo and sends it to the



(a) (b)

Fig. 9. The a) first and b) last steps of the prediction pipeline

URL of a PHP script on the server. The PHP file then saves this input image and invokes the MATLAB code loop that will begin the process of interpreting the image. The MATLAB code will process the input image and return the image result, and the PHP file will automatically push the image result back to the Android app when it is detected.

The MATLAB code implements the purple prediction steps seen in Figure 8. First, it thresholds, segments, and performs morphological processing steps to produce a clean binary mask. The image and the binary mask of the image are both cropped by the bounding box of the largest region in the mask (the hand). The Gabor features are then extracted using the cropped image, and the HOCD features are extracted using the cropped binary mask. Both of these feature sets are fed into the predict function of the pre-trained KNN classifier. The resulting prediction is used to select a stock image of the letter that was predicted, which is then returned by the code.

VI. CONCLUSION AND FUTURE WORK

The effort to find useful features for this problem proved difficult, yet ultimately quite successful with the combination of the HOCD and Gabor filter features. Based on these features, the most effective classifier of the four methods tested proved to be Fine KNN. As seen in section V, this current system has proved to be effective in early tests. However, the test scenario is still somewhat limited, so the results must be taken in that context. It is also to be noted that there are some symbols - the more similar ones such as ”m” and ”n” - that are quite often mis-classified. All in all, the project shows that this difficult problem can require much effort to reach optimal performance. However, the progress made is encouraging and demonstrates that the feature-based approach can be made to work even in a mobile application.

In terms of next steps, finding an effective implementation of the hand model promises to add some distinctive features to the classification process that could help make better decisions, especially with the more difficult symbols. An effective skin segmentation process would also allow the system to be used in environments with cluttered backgrounds, which would drastically improve the practicality of the solution.

ACKNOWLEDGMENTS

The authors would like to thank Profs. Andrew Ng and John Duchi for the opportunity to learn and apply modern Machine Learning, and Nihit Desai for his guidance in our project.

REFERENCES

[1] J. A. Lapiak, "Sign language alphabet," in Handspeak, 1996. [Online]. Available: http://www.handspeak.com/spell/index.php?id=spell-asl.

[2] N. Pugeault, R. Bowden, "Spelling It Out: RealTime ASL Fingerspelling Recognition" in 2011 IEEE International Conference on Computer Vision Workshops, IEEE, 2011.

[3] B. Stenger, P. R. S. Mendonca, R. Cipolla, "Model-based 3D tracking of an articulated hand," in Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, IEEE, 2001, pp. 11-310.

[4] N. Pugeault, "ASL Finger Spelling Dataset," in College of EMPS. [Online]. Available: http://empslocal.ex.ac.uk/people/staff/np331/index.php ?section=FingerSpellingDataset.

[5] A. L. C. Barczak, N. H. Reyes, M. Abastillas, A. Piccio, and T. Susnjak, "A new 2D static hand gesture colour image dataset for ASL gestures," Research Letters in the Information and Mathematical Sciences, vol. 15, pp. 12-20, 2011. [Online]. Available: http://mro.massey.ac.nz/handle/10179/4514.

[6] J. Frnkranz, "Round robin classification," Journal of Machine Learning Research, vol. 2, pp. 721-747, 2002.

[7] "Tutorial on Client-Server Communications," in Stanford Univrsity. [Online]. Available: http://web.stanford.edu/class/ee368/Android/Tutorial-3.pdf