ASL Fingerspelling Interpretation

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 give us the corresponding meaning as output.

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. A graphic illustrating this system is found in figure 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'.

In terms of the bigger picture, hand gesture recognition technology can be applied to other problems as well, such as interfacing with computers or machines through gestures. Interpreting hand movements could also become a way of logging and analyzing human behavior.

On the particular problem of fingerspelling interpretation, there has been work done in the recent past. For instance, [1] describes a method of fingerspelling translation using a Kinect sensor. This is the case of a feature-based method 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, a goal of this project is a real-time mobile implementation of the translation system. Although this increases the difficulty of the problem compared to the use of better hardware (such as a Kinect system of camera and time-of-flight sensor), the tradeoff will greatly improve the practical usability of the system.

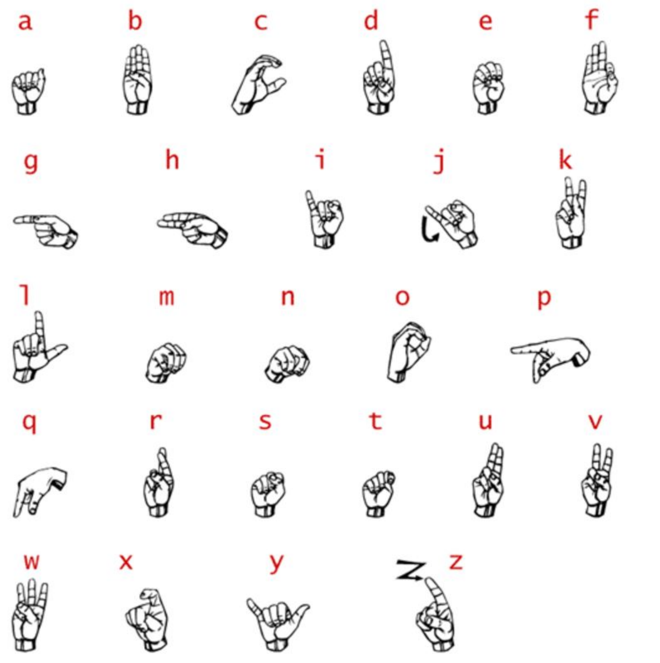


Fig. 1. ASL Fingerspelling alphabet as used in this project. Note that 'j' and 'z' are excluded from our considerations here, since they are not static symbols.

II. Theory

This whole section presents the research and theoretical development that represents the bulk of the work that was done for this project. First we present a basic overview of the structure of our algorithm, before going into further detail on specific aspects of our research.

A. *Algorithm Overview*

Here we briefly present the structure of our proposed system as seen in figure 2. The two main tasks of feature extraction and classification are each discussed in detail in the following two parts of the paper.

The structure of the pipeline that is used in the final implementation is described in detail in section III, including the specific methods chosen to solve these two main tasks.

B. *Feature Extraction*

This part describes various feature extraction methods that were researched as part of this project. In each case we

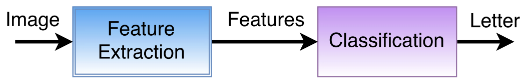


Fig. 2. Basic structure of our translation system.

provide a brief introduction into the principle of the method, before going into detail on how the actual feature extraction works.

1) HOCD: The Histogram of Centroid Distances (HOCD) is a very natural feature extractor which captures the information of the shape of the hand and converts it to a feature representation which is as long as the number of bins used in the histogram. This method works on the binary mask of the segmented hand in the image. The input to this feature extractor is thus, a bounding box surrounding the connected component corresponding to the hand, with ideally, no holes inside it. The algorithm then works as follows-

- Compute the centroid of hand's binary mask
- Extract an edge map of the hand by subtracting an eroded copy of the binary mask from it's dilated one.
- Calculate distances of each edge pixel in this edge map from the centroid and then normalize them by the median distance.
- Compute a histogram of these distances with n bins.

This histogram is then the equivalent representation of the the hand in the feature space where n, denotes the dimensionality of the feature vector. Figure 3a shows the binary mask (along with centroid marked by cross symbol) computed using morphological image processing operations on an image (denoting the letter 'y') from the dataset [3] used in this project. Figure 3b shows the histogram computed by the HOCD algorithm on this input image. Figures 3c and 3d show the binary mask and HOCD histogram for the same letter on one of our own hand images, that were used to test our algorithm. The closeness of these two histograms, one corresponding to 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 these features. This motivated us to lookout for other options for feature extraction.

2) Gabor Filters: One feature extraction method that has been applied in previous work is the use of the Gabor filter. This filter is similar in function to the common gradient operators, but with expanded flexibility in terms of scale and orientation of the kernel. The feature extraction system that we propose here is taken from [1]. The feature extraction principle here is to process the image with a set of filters and then sample the filter responses to create image features.

Gabor filters are linear filters defined by the convolution kernel

$$g(x,y,\lambda,\sigma,\theta) = \exp\left(\frac{x'^2+y'^2}{2\sigma^2}\right)\cdot\exp(i\ ) \qquad (1)$$
$$x' = x\cos(\theta)+y\sin(\theta)\ ,\ y' = -x\sin(\theta)+y\cos(\theta)$$

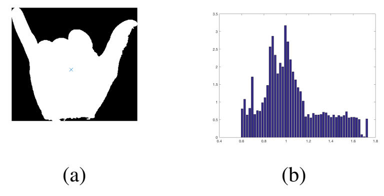
This filter responds to intensity gradients with respect to the orientation specified by θ and the scale specified by λ. In order to detect gradients of many different scales and orientations, we employ a bank of 16 filters that are all applied to the image, resulting in 16 complex filter response images. 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. Before filtering we first resize and zero-pad the image to fit a 128 pixel square frame in order to regularize the feature extraction process. Gabor-filtering the image in this way leads to a suitable basis for detecting the different letter signs.

Since individual pixel values are unsuitable as actual features, one more step is needed for this feature extraction method. Using an 8-by-8 grid of 64 two-dimensional Gaussian functions, we extract 64 local averages of the complex magnitude of each of the 16 Gabor filter responses. Each Gaussian is defined by

$$G_{ij}(x,y) = \frac{1}{\sqrt{2\pi\tau^2}}\exp\left(-\frac{(x-\mu_{xi})^2+(y-\mu_{yj})^2}{2\tau^2}\right)$$
$$\mu_{xi} = (16i-8)\ ,\ \mu_{yj} = (16j-8) \qquad (2)$$

The resulting grid of 64 functions is shown in figure 4. We choose the variance τ^2 of the Gaussians such that the overlap between the channels is quite low. The local averages are computed by weighting the whole image by the ij-th Gaussian function and then summing up all weighted pixel values. 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



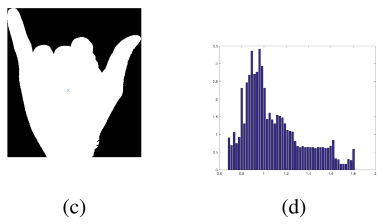


Fig. 3. (a) Binary mask for dataset image of letter 'y';(b) Histogram computed by HOCD algorithm on the binary mask of dataset image; (a) Binary mask for test image of letter 'y';(b) Histogram computed by HOCD algorithm on the binary mask of test image

changes. Scaling and translation can be compensated easily enough as soon as the hand has been correctly segmented. However, rotation differences are difficult to detect on the segmented hand and compensate and so it remains a problem.

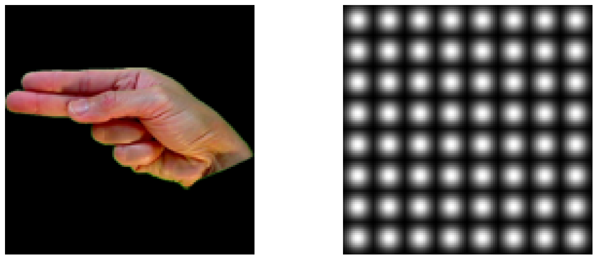


Fig. 4. To the left is a test image used to evaluate the Gabor filter. To the right is a showcase of the grid of 64 Gaussian averaging channels that is used to extract features from the images.

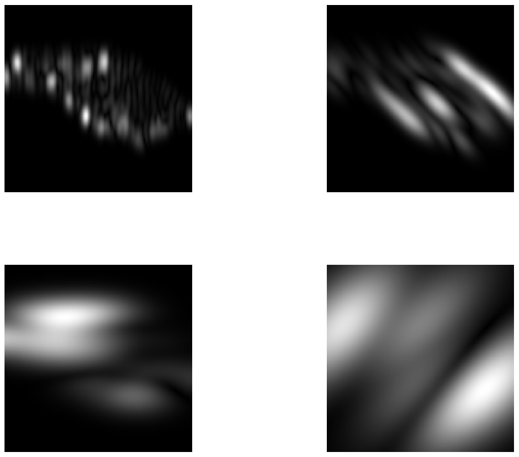


Fig. 5. Responses of four of the 16 use Gabor filters to the test image. There are four different scales and four different orientations used across the filter bank.

3) SURF and Bag of Words: Key-points on the static hand gesture images can be extracted via either the SIFT [4] or SURF [5] methods as a scale-invariant method of points which can be repeatedly extracted on images corresponding to the same letter symbol. Local descriptors for each key-point are also created such that a keypoint detected in another hand image of the same letter will have a similar descriptor. These descriptors are vectors of length 128 for SIFT and length 64 for SURF. In the bag of visual words method [6], descriptors of SIFT/SURF keypoints, extracted from each image and are concatenated together to form a matrix containing vectors of either 64 or 128 elements (one vector for every keypoint in each of the images). To understand the ”words” that are contained in the visual dictionary that makes up these hand images, we apply a k-means clustering algorithm with $k = 200$ cluster centroids. From this we get k vectors of length 64 or 128, which we treat as the ”words” in our visual vocabulary. We now assign every descriptor vector in each of the images to the nearest cluster centroid (or ”word”). For each image, this gives us a histogram with k bins and bin values equal to the number of descriptors which are closest to that centroid (i.e. the number of times that ”word” appears in the image). The histogram for each image can then be used as the feature vector for that image (i.e. training example). However, when this method (tried for SURF) was used for feature extraction on training and test images independently, completely different key-points were detected in the two images. It was reasoned out that these features are not suitable for actual implementation, because they fail to capture enough information from the images because of illumination differences playing a major role in different key-points extracted on train and test images. This difference is illustrated in Figure 6

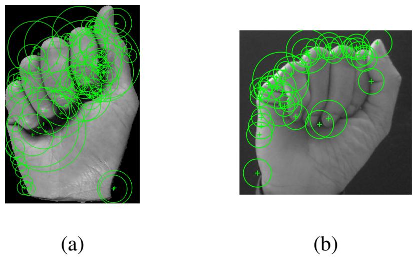


Fig. 6. Difference between SURF key-points extracted on dataset image and real test image (corresponding to letter ’a’) due to different illumination conditions

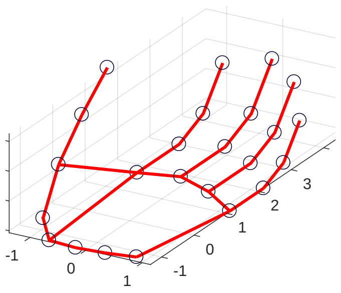
4) Hand Model: A method that has been extensively researched is a posture detection method based on a mathematical 20-degree-of-freedom (DoF) model of the hand skeleton. This method is inspired by [], but no direct references for this approach have been found. Instead, all work has been built from the ground up. The principle of feature extraction is as follows: For each image, we segment out the hand to get its silhouette. Then we use an optimization algorithm to find a parameter set for the hand model that aligns its 2D-projected silhouette with that of the detected hand. Finally, these optimized parameters are used as the image features and can be used to classify images.

The model defines the hand with 24 points, whose relative position is parametrized by the angles of the finger joints. In the simplified model used here, each finger has one ball joint (two rotation axes) and two hinge joints (one rotation axis each) - so the total amount of parameters comes to 20. Again, this 20-DoF model does make some simplifications, so we may consider employing more complex models if errors occur. The lengths of the individual parts of the skeleton were estimated to produce a good visual similarity to the shape of real hands. One configuration of this model can be seen in figure

The projection of the 3D hand model onto a 2D image surface was accomplished with perspective projection. This adds further parameters for scale as well as x- and y-position in the 2D frame. Before the projection is applied the 3D model can also be rotated around all three axes, resulting in a total of 26 parameters for the 2D-projected 20-DoF hand model.

The key to the actual feature extraction process now lies in manipulating this established model to align with a detected hand silhouette. The parameters of the optimal fit represent the estimate of the hand's posture and are theoretically well suited to distinguish all of the hand symbol postures. The approach explored here involves defining an analytical quality function to the position of each point of the hand model in the 2d projection. This quality function has so far been defined as the sum of a set of bumps centered on all hand pixels. The optimization of this quality function can now in theory be implemented with the gradient ascent method.

This did however prove difficult within the given time constraints. Also, the pixelwise definition of the quality function - while theoretically feasible - 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 its biggest drawback.



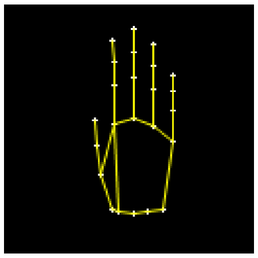


Fig. 7. To the left is a test of the 3D hand model with the circles indicating the joints. To the right is a projection of that hand into a 128 pixel square image.

5) Skin Segmentation: Although it is not a feature in and of itself, skin segmentation can be used to separate the hand from the background prior to implementing any of the feature extraction methods mentioned above. Several color models have been analyzed for the purpose of skin pixel classification. 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 be 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 (making sure to include shadows on the palm and highlights on the fingers). The normalized RGB values were calculated as follows:

$$r = \frac{R}{R+G+B}; g = \frac{G}{R+G+B}; b = \frac{B}{R+G+B}$$

Since $r + g + b = 1$, we can discard the b value and represent normalized RGB values in the rg chromaticity space, thereby removing the pixel intensity information. In this space, a pixel with the same ratio of R:G:B will be identical to another pixel with a scaled ratio of R:G:B. The r and g values are thus calculated for each pixel in each skin sample. A 2-dimensional histogram is then created for the r and g values as seen in Figure 8.

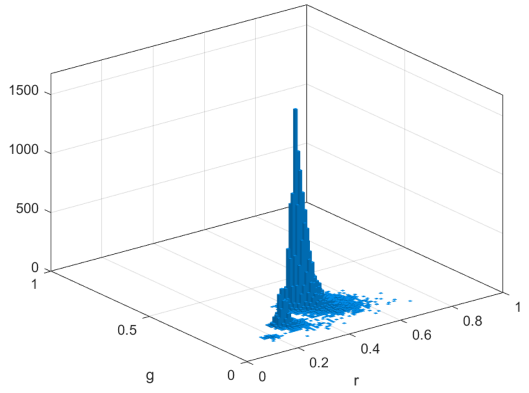


Fig. 8. Histogram of skin pixels in rg chromaticity space

To then detect the hand in each image, we calculated the r and g values for each pixel in the 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 threshold the likelihood map to create a binary image, and use various morphological image processing techniques to fill in holes and reduce other noise created by pixels that appear to be skin but are not (Figure 9). This procedure does not create a perfect mask and tends to miss areas of the hand with particularly harsh illumination or deep shadow. It should be noted that this method is very inaccurate at segmenting out the hand when other skin such as the face or neck is also present in the image. However, 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 training and testing images that consisted of a hand in front of a black background, thereby forgoing the need to use skin segmentation.

C. Classification

For classification we have considered the simple yet powerful, non-parametric K-nearest neighbors (KNN) algorithm which is built into MATLAB. In the training phase, this algorithm just stores the feature vectors (for each or a combination of the above feature extraction techniques) along

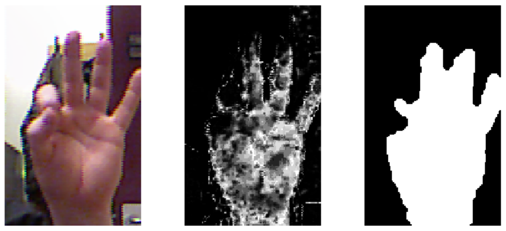


Fig. 9. Original image (left), likelihood map (middle), and binary mask of hand (right)

with corresponding class labels for each training example (data-set hand image) 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 test image's feature vector. Other methods such as Multiclass SVM and Discriminant Analysis were also considered. Although they gave comparable or sometimes even better accuracy on training set as compared to KNN, they offered much lower test errors as compared to KNN.

III. Implementation

A. Training phase

1) Data-sets: In the training phase, we considered two different data-sets, particularly, [2] and [3]. The former has hand images of very low resolution and are captured in cluttered backgrounds, while the later has good resolution segmented hand images. Although the first dataset would generalize well, for training our model, we considered the second dataset to ensure, a much accurate prediction of the live images captured using the Android smart-phone. This is because, while performing the live demo we made an assumption that we capture images against a black background to avoid hand segmentation problems and focus on the more important task of classifying the hand gesture correctly with a good choice of features. In a separate experiment, we did attempt to relax this assumption by trying to extract a hand mask from the cluttered background by using skin color histogram segmentation, more on which is elaborated later in the report.

2) Building the model: The data-set [3] contains approximately 70 images per letter from 5 different users. The feature vector for each image in this data-set is constructed by concatenating the feature vectors from the HOCD method and Gabor Filter methods that were described in the above section. Among other options such as only HOCD and only Gabor, this combination was particularly chosen as it gave the best classification performance on the real-test images that were taken with the Android phone. SURF features using the bag of words model weren't considered because even though they offered a not so poor generalization error on the training set itself, the repeatability of key-point detection in test images was very poor because of illumination variance of SURF. Also, before the actual test images were passed to the individual feature extractors, they were morphologically pre-processed as follows-

- Gray-level thresholding using Otsu's method to get a binary mask of the actual image
- Dilation and hole filling of this binary image to obtain a smoother and more well segmented hand mask.
- Small region removal to segment out erroneously detected foreground elements (due to illumination conditions) and ensure that the hand is the only connected component in the entire binary image.
- Crop the image based on a bounding box around the hand area.
- Give the cropped binary mask as input to HOCD and original image cropped using the computed bounding box as input to the Gabor Filter feature extractor

These steps ensure, that the features computed are independent of re-scaling and bad lighting conditions (which will be more important when testing, but to ensure same method for feature extraction, applied to training as well). The combination of these two features and corresponding class labels is then passed to a KNN classifier to train a classification model which is later used directly to predict every single test image captured using the Android phone.

B. 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 10.

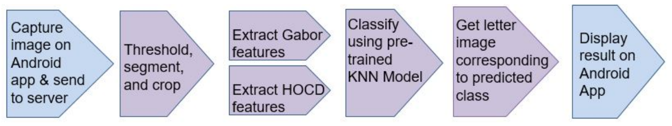

Capture image on Android app & send to server
Threshold, segment, and crop
Extract Gabor features
Extract HOCD features
Classify using pre-trained KNN Model
Get letter image corresponding to predicted class
Display result on Android App

Fig. 10. Steps in prediction pipeline with Android platform in blue and server in purple

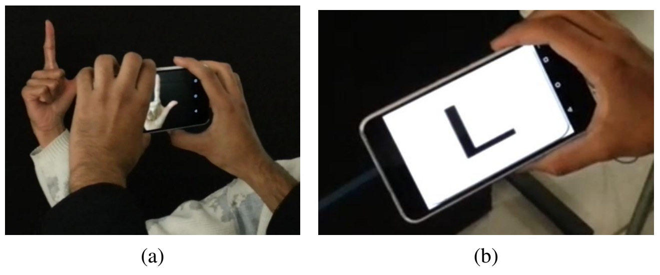


Fig. 11. The a) first and b) last steps of the prediction pipeline

1) Android Client-Server Communication: In Figure 11, the first and last steps in blue were performed on a mobile Android application, 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

TABLE I

ERRORS MEASURED ON TRAINING SET AND TEST IMAGES

Feature Extractors	Generalization error on training set (using 5 fold cross validation)	Test-error on images captured using Android phone
HOCD	0.1940	0.6
Gabor Filter	0.009	0.5
HOCD + Gabor Filter	0.0107	0.2
SURF with Bag of Words	0.1403	-

in [7], we set up an Android application as the client which takes a photo and sends it to the server (specifically, to the URL of a PHP script on the server). We would then like the PHP file to save this input image, generate a file indicating that the input image is ready for processing, and invoke the overarching MATLAB code loop (predictASLLoop.m) that will begin the process of interpreting the image. However, the corn server does not allow MATLAB to be launched from our PHP file, which means we have to make sure that predictASLLoop.m is running on the server while we are trying to process an input image. While it is running, the overarching loop (predictASLLoop.m) continuously checks for the PHP-generated file indicating that an input image is ready for processing. If an input indicator file is present, predictASLLoop.m will read the input image, remove the input indicator file, pass the input image for processing by predictASL.m, save the image result from predictASL.m, and generate a file to indicate that the image result is ready to be sent to the Android app. The PHP file will be continuously checking for this result indicator file. When the result indicator is detected, the PHP file will automatically push the image result back to the Android app.

2) MATLAB Prediction Steps: The predictASL.m file implements the purple prediction steps seen in Figure 10. First, it thresholds and segments the image using Otsu's method, while filling in holes and performing small region removal. 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. This predicted letter image is then returned to predictASLLoop.m.

IV. Results

As discussed before, the training of the classification model was performed with four different combinations of feature extraction methods and generalization error using 5-fold cross validation on the training set and test-errors on the real-images captured using Android phone were computed three of these combinations (test errors weren't computed for SURF with Bag of Words because, even though generalization error was not too bad on training set, the key-points computed had a very high dependence on illumination condition and this method wasn't suited for real time implementation as discussed before). The results are given in Table I.

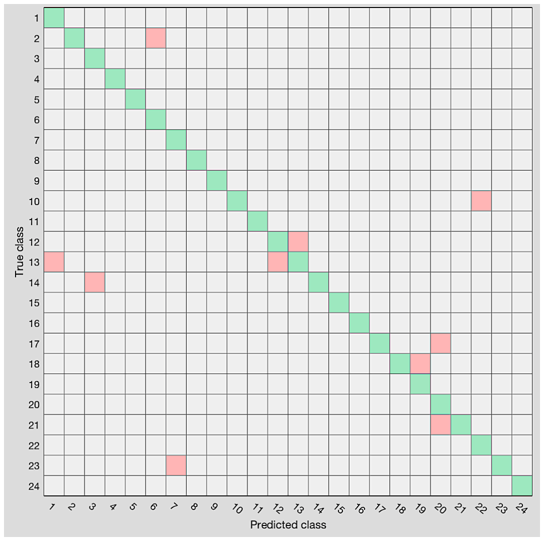


Fig. 12. Confusion Matrix for generalization on training set using HOCD and Gabor Filters in combination as feature extractors

As it can be seen from the table above, Gabor filter clearly over fits the training dataset but performs much worse than the combination of HOCD and Gabor on the test set. HOCD is inherently more robust to rotation and scale changes, but it's generalization error is not as good. The combination of both HOCD and Gabor gives the lowest test error of 0.2. To give a sense of the generalization error using the combination features, a confusion matrix for the 24 class labels is shown in Figure 12.

V. Conclusion

To evaluate the work on this project, we look to the insight gained from the research performed and to the test results offered by our completed prototype system.

As we saw in section II, the effort to find useful features proved quite difficult for this problem, which meant that a range of different approaches needed to be explored. In the end it is clear that for the constraints of this project in terms of time and hardware used, the combination of HOCD and Gabor filter feature extraction proved to be most effective and reliable. The SURF-based method and the hand-model based approach were both discarded for now along with the ML skin segmentation method. The difficulties encountered mean that further exploring new solutions and methods may be a fruitful way of improving on the current system.

As seen in sections III and IV, 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 difficult ones - that are quite often misclassified.

All in all, the project shows that this difficult problem can require much effort to solve optimally. However, the progress made is encouraging and demonstrates that the feature-based approach can be made to work even in a mobile application.

Also, with the right training sets, this method could be applied to interpret any set of static hand gesture symbols.

Next steps to improve the current system would focus on further work within hand model feature extraction and and skin segmentation. The former - finding an effective implementation of the model-based detection - promises to add some distinctive features to the classification process that could help make better decisions, especially with the most difficult symbols. The latter - an effective skin segmentation system - would allow the system to be used on images with cluttered backgrounds of almost any color, which would be a significant improvement. Connected to this effort we would also suggest developing a system that can distinguish between different types of skin regions in order to avoid interpreting non-hand regions by accident.

Acknowledgment

The authors would like to thank Prof. Gordon Wetzstein , Jayant Thatte, Jean-Baptiste Boin and Stanford University for their support and mentorship throughout this project.

References

[1] N. Pugeault, R. Bowden, Spelling It Out: RealTime ASL Fingerspelling Recognition,

[2] N. Pugeault, ASL Finger Spelling Dataset,

[3] Barczak, A.L.C., Reyes, N.H., Abastillas, M., Piccio, A., Susnjak, T. (2011), A new 2D static hand gesture colour image dataset for ASL gestures, Research Letters in the Information and Mathematical Sciences, 15, 12-20

[4] D. Lowe, Object Recognition form Local Scale-Invariant Features,

[5] H. Bay, T. Tuytelaars, L. Gool, SURF: Speeded Up Robust Features,

[6] Y. Zhang, R. Jin, Z. Zhou, Understanding Bag-of-words Model: A Statistical Framework,

[7] http://web.stanford.edu/class/ee368/Android/Tutorial-3.pdf