Sign Language Classification Using Webcam Images

Ruslan Kurdyumov, Phillip Ho, Justin Ng

December 16, 2011

Abstract

Immediate feedback on sign language gestures can greatly improve sign language education. We seek to classify the English sign language alphabet using webcam images. We normalize and scale the images to 20 x 20px, and use the binary pixels as features. We can successfully classify up to $92 - 93\%$ of the signs using a linear or Gaussian kernel SVM, outperforming k-nearest neighbor classification by about 10%. Using a custom set of features we developed, we can successfully classify $81 - 82\%$ of the signs with an SVM.

1 Introduction

Implementing techniques in machine learning and image processing, we hope to obtain a high level of accuracy in distinguishing between the letters in the English sign language alphabet. Our system receives input from webcam images and classifies them based on features defined by post-processing the images.

The primary application for sign language recognition is to improve sign language education. A person who wants to learn and practice sign language today does so by hiring a sign language instructor or watching instructional videos online while practicing in front of a mirror. The former method can be expensive and inconvenient for people with busy schedules and the latter does not provide the user with necessary feedback on correctness.

Our recognition system can be implemented into a desktop or browser application that can provide immediate feedback to a user's hand gesture. Such a tool would be inexpensive and convenient, requiring the user to practice in front of their webcam rather than attending expensive sessions with an instructor. Unlike the mirror, the system can also provide instant feedback on the correctness of the hand gesture.

The alphabet we're classifying introduces several interesting challenges. First, many letters (e.g. a, m, n, s, and t) are represented as very subtle nuances of a clenched fist. Since we use the silhouette of the hand shape to classift, these letters are likely to be commonly misclassified. Second, the letter j and z are signaled by motion in the hand, which cannot be represented in a single image. The starting pose of the letter j is identical to the letter i, so we removed this letter from our data set. The letter z, which starts in a distinct pose with the pointer finger, is represented by its starting pose.

Our choice of using webcam images as our input format also creates some difficulty in correctly classifying hand shapes. Depending on the quality of the webcam, captured image data typically contains noise from compression. Since we cannot constrain where the user's hand will be in relation to the camera, our recognition system needs to be invariant to scale and position. Rotational invariance is less important because the user's arm will typically be pointing vertically upward to make hand gestures, but this is a feature we hope to include in the future. We tackle these challenges in post-processing, explained in a later section, of the image captures.

Current solutions incorporate a variety of machine learning techniques to classify hands. Using a Pyramid of Histogram of Oriented Gradients as a feature for a SVM with 70-30 cross validation, Knight is able to distinguish between a hand and non-hand [1]. Shariff applies a combination of (2d) shape-based and size-based features to recognize the configuration of the hand in the scene [2]. Tracking data gathered over time with a 3d-camera, Marx devises a SVM-based system to classify between one finger, two fingers, a closed fist, and open palm gestures [3]. Hidden Markov models, commonly used in handwriting recognition, achieve a 97%+ accuracy in classifying 40 words in American Sign Language [4].

2 Data Collection

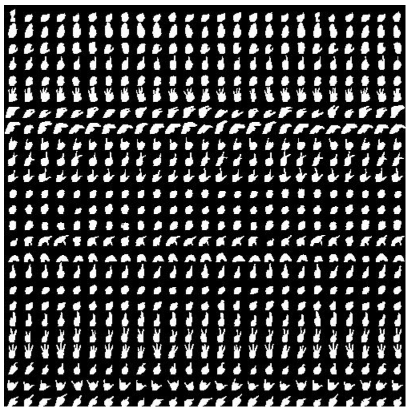
Figure 1: A sampled set of image processed training data.

Our dataset comprises over 1800 webcam images of hand gestures (see Figure 1). The images were taken from a standard laptop webcam at 640 x 480px resolution. Using a custom MATLAB script to expedite the process, each team member took 20-30 images of his right hand signing each of the 25 letters (skipping the letter "j"). While capturing images, the hand gestures were moved or rotated slightly to avoid taking sets of images that were too similar. By introducing some variance into the dataset, we help ensure that any new data we test on does not need to look exactly like the training data in order to be correctly classified.

A single image of the background without a hand in the foreground is also taken for both the training and test data. This background plate allows our system to extract the foreground from the images much more easily by performing background subtraction, which we describe later.

3 Image Processing

We apply image processing to better extract features from our input images. In particular, the features that we extract from our images should be invariant to background data, translation, scale, and lighting condition.

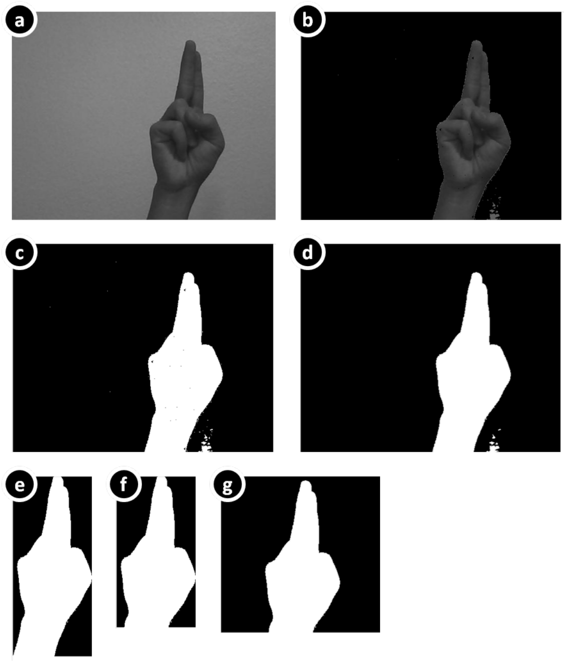

Figure 2: Image processing timeline. We start with a raw image (a), subtract the background (b), binarize (c), remove small white and black holes (d), remove white patches and crop the hand (e), cut out the wrist (f), normalize the hand size (g).

First, images are converted to grayscale. While this makes other operations much more simple and quicker to compute, we lose color information from the original images. Since our system needs to be lighting invariant, the color information should actually be ignored because we cannot rely on the coloration of the hand to be consistent between test and training image data.

Since the background data is not relevant and should not be trained nor tested against, we first remove the background from the foreground. By subtracting the background image from an input image, we find the intensity change of each pixel. Intensity changes above some threshold are classified as foreground pixels and the remaining pixels are background pixels. The threshold was set to ensure that hand pixels would not be subtracted out.

The next step is to get the silhouette of the hand. The image is binarized by setting all non-black pixels to white (1) and leaving the remaining pixels as black (0). We remove holes and remove all but the largest connected components to get a single, solid silhouette of the hand.

We normalize the translation and scale by relying on some assumptions about the wrist. First, we locate the wrist by starting from the bottom-most row, assuming the hand protrudes from the bottom edge of the image, and checking the width of the arm in each row until the width reaches its minimum, which is assumed to be the wrist location. The hand is repositioned such that the wrist is centered at the bottom of the image and padded by a factor dependent on the wrist width to fit into a square. The final image is resized into a 20 x 20px image, forming a 400-member feature vector. The full effect of processing can be seen in Figure 2.

4 Approach

Our overall approach to the classification problem is summarized in Figure 3.

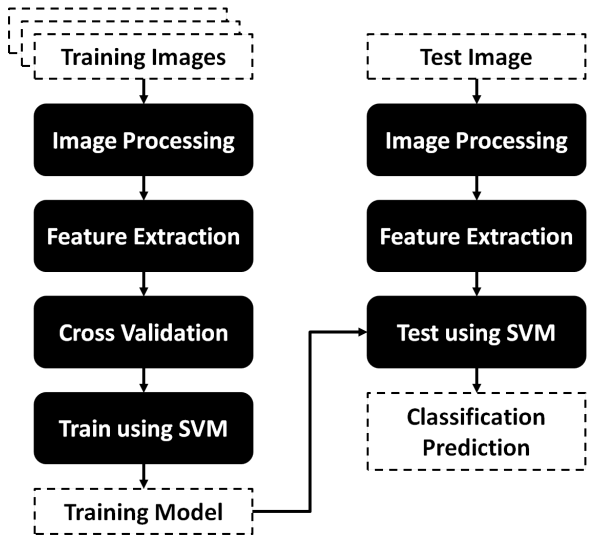

Figure 3: The classification process.

After obtaining our full data set, we shuffle the data and split into 85%/15% train/test, which still leaves us with over 250 test examples. Since it's not obvious which image size will provide the most distinguishing features, we varied the resized image dimension and examined the 10-nearest neighbor test error. We find that a 20 x 20px image is optimal, and these results are summarized in the Cross Validation subsection.

Once we have an optimal image size, we run 10-fold cross-validation on the train examples to determine optimal SVM parameters C and γ (for the Gaussian kernel). Given an optimal set of parameters, we then train on the full training set, forming a training model, and test on the test set.

5 Results

5.1 Pixel features

5.1.1 Image Scaling

To determine the optimal image dimensions to use for classificiation, we ran k-nearest neighbor (with $k = 10$) classification on our full test set with pixel dimensions = [10 20 30 40 50]. We chose to use k-nearest neighbor classification rather than SVM since using an SVM would require tuning model parameters for each pixel dimension, significantly complicating the cross validation. The results can be seen in Figure 4.

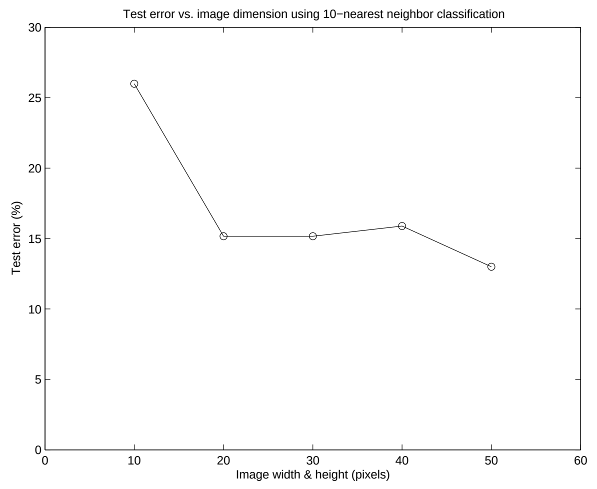

Figure 4: Image resize dimension test error using 10-nearest neighbor classification.

We find that 10 pixels is not enough to robustly classify among the signs, as expected. When scaling the image down that severely, many distinguishing features are lost. 20 and 30 pixel image dimensions have improved performance (error around 16%, and the 50 pixel image dimension has the best performance. We chose to use 20 x 20 pixel images because enough distinguishing features should remain and we can filter out some noise by scaling down to that size. In addition, with a training set of around 1500 images, we could ensure that our number of features (400) was still significantly smaller than the training set size.

5.1.2 Cross Validation

After creating the feature vectors for all of our test and train examples, we ran 10-fold cross validation to determine the optimal soft margin parameter C and the Gaussian parameter γ (for the Gaussian kernel). We used a log2 space search (in either 1 or 2 dimensions) and picked the parameter (or parameter pair) with the highest cross validation accuracy. In general, we first run a coarse search, and if necessary, a fine search. For example, the search for optimal C, γ parameters is shown in Figure 5.

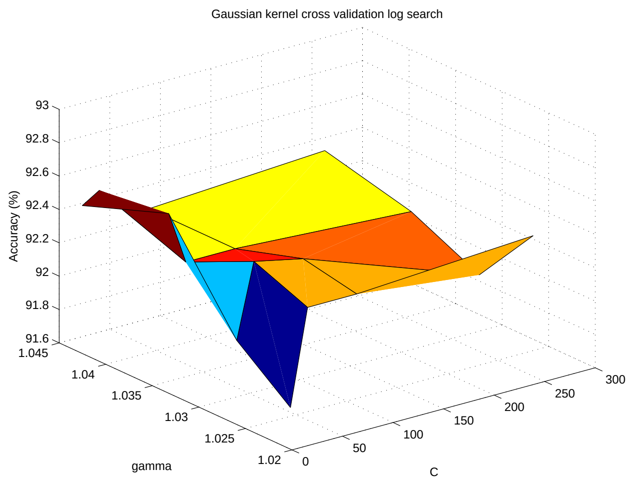

Figure 5: The cross-validation accuracy for a range of Gaussian kernel SVM parameters.

5.1.3 Classification Results

The summary of the classification results can be seen in Table 1.

Table 1: Pixel Feature Classification Results

Classifier	C.V. accuracy 12 signs	C.V. accuracy 25 signs	Test accuracy 12 signs	Test accuracy 25 signs
Linear kernel	97.2%	90.8%	98.6%	92.4%
Gaussian kernel	98.3%	92.4%	98.6%	93.5%
k-nearest-neighbor	N/A	N/A	93.0%	84.8%

The Gaussian kernel slightly outperforms the linear kernel SVM when testing the full 25 sign set, but they are both 98.6% accurate on the condensed 12 sign set. They both outperform the baseline k-nearest neighbor classifier (with $k = 10$). Our test error is actually slightly lower than the cross validation error, which seems somewhat anomalous. The likely explanation is that when we randomly shuffled our data and split into training and test sets, the training set happened to get a disproportionate amount of poor examples.

5.1.4 Learning curve

To get an idea of how well our algorithm was learning, we plotted the learning curve, seen in Figure 6.

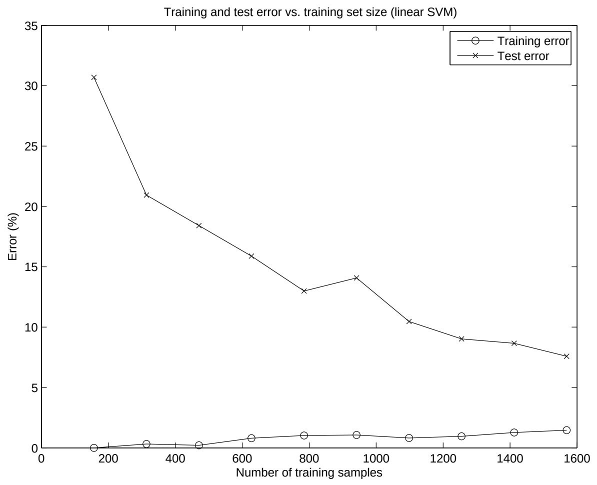

Figure 6: Learning curve. The test error appears to be asymptotically decreasing, suggesting a high variance situation.

The plot shows an asymptotic decrease in the test error, coupled with a slow increase in the training error as the training set size increases. It appears that we have a high variance situation - lots of features and not enough training examples. Since our train error is fairly low, high bias is not a problem.

5.2 Custom features

To compare the performance of our pixel-based feature vector, we also created a 13-feature vector from the images. To collect the features, we used the normalized (but not resized) image, as in Figure 2 (g), except with the hand pixels in grayscale. Three features we collected were the relative area, height and width of the normalized hand. We also determined the width of the top of the hand and the gap in the top the hand. These two features could be useful in telling us how many fingers that the user was holding up. We also obtained features by running Fourier transforms on various rows in the image and saving the magnitude at an optimal frequency. Running the Fourier transforms provides a rough alternative to edge detection. The edges are primarily caused by the fingers, so the position of the fingers in the hand gives a unique spectral signature which can be used to recognize the sign.

The classification results using our custom features can be seen in Table 2.

Table 2: Custom Feature Classification Results

Classifier	C.V. accuracy		Test accuracy	
	12 signs	25 signs	12 signs	25 signs
Linear kernel	94.8%	78.8%	99.2%	82.3%
Gaussian kernel	95.8%	85.7%	87.1%	81.2%
k-nearest-neighbor	N/A	N/A	92.7%	78.0%

With our custom features, the Gaussian kernel is actually less effective than the linear kernel, suggesting that some overfitting is taking place in the higher dimensional feature space. We have a higher test accuracy than C.V. accuracy for the linear kernel, but not for the Gaussian kernel, which is difficult to explain. The linear kernel SVM outperforms the 10-nearest neighbor classifer for both sign sets.

6 Further Work

There are several key extensions of our work. While our method works well on webcam images, ideally, we would like to classify live webcam video of sign language. This would require a different image processing scheme to isolate the hand in the image. In particular, our background subtraction algorithm would have to be modified, since the background would be dynamic. An alternate approach would be to use scale invariant features derived from webcam stream images, derived from an algorithm such as SIFT [5], which would be less sensitive to background noise and rotations of the hand.

References

[1] Knight, D., Tang, M., Dahlkamp, H., and Plagemann, C. ”A Framework for Recognizing Hand Gestures”, CS229 Final Project Paper, 2010.

[2] Shariff, S. and Kulkarni, A. ”Identifying Hand Configurations with Low Resolution Depth Sensor Data”, CS229 Final Project Paper, 2009.

[3] Marx, M., Fenton, M., and Hills, G. ”Recognizing Hand Gestures with a 3D Camera”, CS229 Final Project Paper.

[4] Starner, T. and Pentland, A. ”Real-Time American Sign Language Recognition Using Desk and Wearable Computer Based Video”, *IEEE Transactions on Pattern Analysis and Machine Intelligence* Vol. 20, No. 12, December 1998

[5] Lowe, D.G. ”Object recognition from local scale-invariant features”, *The Proceedings of the Seventh IEEE International Conference on Computer Vision, 1999* Vol. 2