Vision based Hand Gesture Recognition using Color, Depth, and Confidence Information

Berat Biçer
Department of Computer Engineering
Bilkent University
Ankara, Turkey
berat.bicer@bilkent.edu.tr

Alper Şahıstan
Department of Computer Engineering
Bilkent University
Ankara, Turkey
alper.sahistan@bilkent.edu.tr

***Abstract*—Despite being studied for a long time, hand gesture recognition in unconstrained environments, in practice, still relies heavily on hardware support for accurate segmentation and classification due to difficulties in accurate segmentation and hand tracking. In this project, we present some experiments that study the effects of these difficulties using computer vision based approaches with an in the wild dataset.**

***Index Terms*—hand gesture, color image, depth image, computer vision, machine learning**

I. Introduction

Hand gesture recognition is a crucial field in computer vision with many applications in fields of augmented reality, and human-computer interaction. Recently, sophisticated hardware is developed for automated hand gesture recognition with high accuracy such as LeapMotion [1]. However, in software domain, obtaining performance and accuracy in hardware levels is extremely difficult. The main challenge in software domain is the lack of robust hand tracking, posture recognition, and segmentation algorithms [2]–[8] as shown in Figure 1. This is a major drawback of software-dependent hand gesture recognition systems and is an active field of research.

In this project, our aim is to study the solutions to difficulties in software based hand gesture recognition task. Our dataset [9], [10] consists of 1320 samples, each belonging to 11 different gestures repeated 30 times by 4 unique subjects. We plan to apply convolutional neural networks(CNN) and principal component analysis(PCA) for feature extraction and support vector machines(SVM) and neural networks(NN) for classification. For performance evaluation, we shall compare our results to those in [9], [10] for external validation.

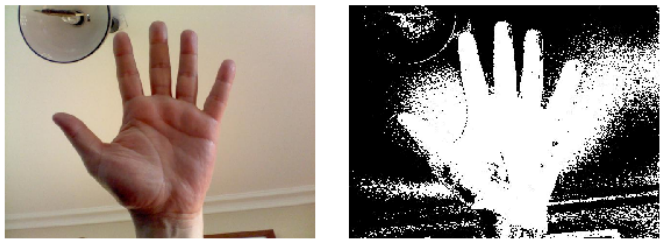

Fig. 1. An example of a poor performance for hand segmentation [11].

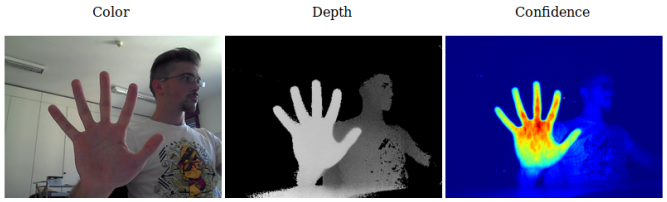

Fig. 2. A visualized example from our dataset [9], [10].

II. Related Work

In out project we will look at mainly two approaches first one being the approach by Bretzner et al. [2] and the other being approach by Liu et al. [6]. First approach is based on the paper "Hand Gesture Recognition using Multi-Scale Colour Features, Hierarchical Models and Particle Filtering" which requires RGB images without any special sensors [2]. Second approach on the other hand is based on the "Hand Gesture Recognition using Depth Data" paper and this one does require a special camera with dept sensor [6]. Although we mainly focus on there techniques we will incorporate methodologies mentioned in the section I.

1) Hand Gesture Recognition using Multi-Scale Colour Features, Hierarchical Models and Particle Filtering: This method focuses on classifying well defined couple of gestures rather than estimating whole hand's posture. They use color-based feature detection method that uses scale-space extrema of normalized differential invariants.The input RGB image is firs transformed into an Iuv colour space:

$$I = \frac{(R+G+B)}{3} = f_1 \tag{1}$$

$$u = R - G = f_2 \tag{2}$$

$$v = G - B = f_3 \tag{3}$$

These features are calculated and further processed for multi-scale pyramid structure of the RGB image.This process allows the detection of blobs(fingers, palm...) and ridges. Then these detected blobs and ridges are used to estimate a Hierarchical Model that roughly reassembles a hand that

consists of (i) the palm as a coarse scale blob, (ii) the five fingers as ridges at finer scales and (iii) finger tips as even finer scale blobs see Figure 3. In addition to these they make the hand model morediscriminative in cluttered by scenes including skin colour information in the form of a probabilistic prior [2].

2) Hand Gesture Recognition using Depth Data : Liu et al. uses a 4 stage algorithm along side with dept sensor images to detect the gesture of the hands. 4 stages consist of:

1) Hand detection
2) Hand shape analysis
3) Hand trajectory analysis
4) Classification of <shape, location, trajectory, orientation>

Since images are dept images separating the hand can be done through a simple threshold(assuming hand is the closest body part to camera). Hand shape analysis is done by matching an unknown hand shape against hand shape in a data set, using Chamfer Distance (CD) as the similarity measure. Trajectory analysis is done by decoupling X,Y,Z datas stored in the images to estimate a curve that is created by the movement of the hand. After collecting all the data classification can be performed. G=¡shape, location, trajectory, orientation, speed¿ G can contain several dozen gestures due to many degrees of freedom. The tricky part is to set the gestures apart. The proposed solution for this is introducing delimiters. Thus allowing a classifiation [6].

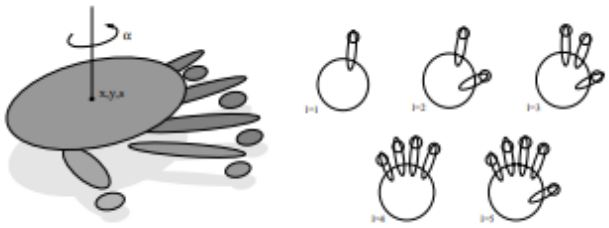

Fig. 3. Feature-based hand models in different states. The circles and ellipses correspond to blob and ridge features [2]

III. Methodology

A. Data

We mainly utilized "Hand Gesture Datasets" by Multimedia Technology and Telecommunications Laboratory at University of Padova [9] [10] as our train and test data. It contains 1320 samples of 11 gestures of 4 subjects repeated 30 times each. Each sample contains color, depth, and confidence information available.

B. Algorithms

For feature extraction, CNN and PCA are the two major algorithms we have been employed. For classification, a transfer learning approach for a pretrained CNN with a multilayer perceptron for classification is our primary method. The second approach is to train a MLP on PCA features. This is less promising than the first approach, however, is easier to implement and is useful for comparison. Thirdly, we implement an Gaussian Mixture Model(GMM) approach on PCA features. This is not also very promising however it provides a good base line for our previous results.

C. Implementation Details

We have used PyTorch [12] and Scipy Python libraries for implementation. PyTorch is a deep learning library supporting design, train, and test of neural networks. Scipy is Python scientific computing library, providing many functionalities such as PCA ready to use along with image processing functionality. Some of our experiments made use of Matlab.

IV. Experiments and Results

Experiments focuses on three aspects of our classifiers: the effects of feature extraction, network architecture, and fusion on classification accuracy. First, the effects of feature extraction has been studied by comparing classifiers which use PCA features with classifiers which use CNN features. Next, the effects of network architecture has been studied by comparing the results of classifiers which uses a MLP to those using a clustering approach. Lastly, we will follow a similar approach for fusion, by comparing classifiers with different fusion strategies. The results are provided in form of tables and graphs along with discussions texts elaborating on them.

A. GMM on PCA features

We implemented and experimented a GMM based classifier that acted on PCA features that we obtained from our data. As previously mentioned in the III-A. section our data has 5 different channels(RGB, Dept and Confidence) when looked at as an image. To conduct our experiments we flattened these 5 channels into 1-D arrays then concatenated to each other. Without any re-scaling loading each image channel as a feature occupied 933 MB. This was very storage heavy for our devices therefore we had to make some re-scaling on our image data before we flattened them. The images were scaled down to 70x70 pixel resolution. Then flattened.

From there we applied PCA on those features. Before we applied PCA our data's dimensionality was 24500 (70x70x5). Since going over 24500 PCA's takes too much time to experiment we decided to go over first 120 features of the PCA(sorted in the order that yielded the highest variance). When looked at fig.IV-A first 120 features correspond to maximum of 90% variance. Our script goes over these features 2-by-2 (including 2 more each iteration).

After reducing the dimensionality we train GMMs for each class separately. For each subset of PCA features we experiment on different GMMs with different component counts between 1 and 10. And for the distance metric for classification results we used two different metrics; maximum of PDFs and minimum of mahalanobis distance. Maximum classification accuracy we achieved with PDF was 0.13 with 1 PCA feature and 6 GMM components (see fig. IV-A) where

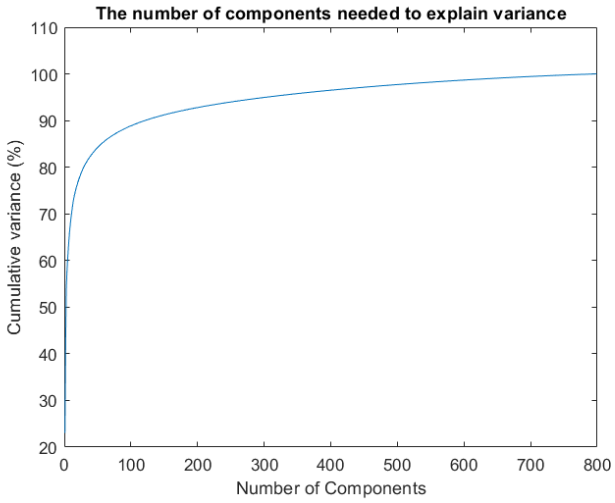

Fig. 4. Variance percentage explained by number of PC's from our dataset [9].

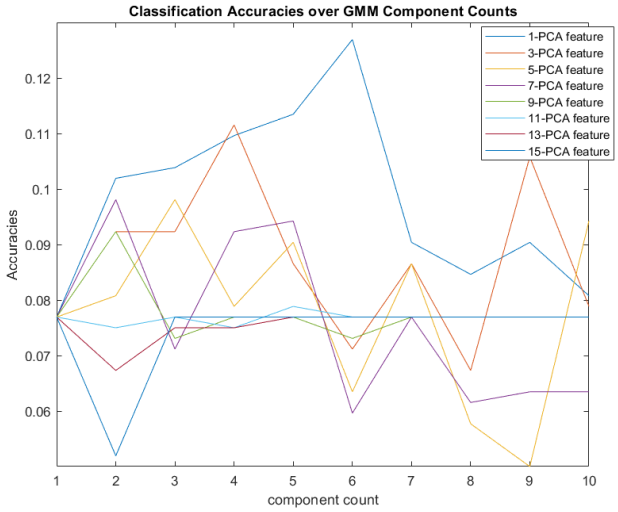

Fig. 5. Test set accuracy for class specific GMMs over PCA features.

as mahalanobis distance failed to go above 0.09 accuracy for any of the cases. These accuracies are obtained from a test set that contains unobserved samples. 800 images where used for training where as remaining 520 images were used for testing. Summary of the accuracy results are given in the tables I.

There few things to note for this experiment. Firstly we had to change the default algorithm of the Matlab's PCA function to eigenvalue decomposition (EIG) of the covariance matrix [13] since data dimensionality were larger than the number of samples. Also for the same reason we could not experiment more than 127 PCA-features since it failed to converge for 1100+ iterations. Since we were picking random indices for train and test set some times sample count for a certain class in the training phase was not enough for a GMM to be trained for those cases we duplicated some of the rows of that class's data until number of rows for that data became larger or equal to the number of columns(feature count).

TABLE I
TEST SET ACCURACY FOR GMM CLASSIFIERS OVER PCA FEATURES.

Comp. counts	1	2	3	4	5	6	7	8	9	10
1 PCA feature	0.08	0.10	0.10	0.10	0.11	0.13	0.09	0.09	0.09	0.08
3 PCA features	0.08	0.09	0.09	0.12	0.08	0.07	0.09	0.07	0.11	0.08
5 PCA features	0.08	0.08	0.10	0.08	0.09	0.06	0.09	0.06	0.05	0.09
7 PCA features	0.08	0.10	0.07	0.07	0.09	0.05	0.08	0.06	0.06	0.06
9 PCA features	0.08	0.9	0.08	0.07	0.08	0.07	0.08	0.08	0.08	0.08
11 PCA features	0.08	0.07	0.08	0.08	0.08	0.08	0.08	0.08	0.08	0.08
13 PCA features	0.08	0.07	0.07	0.07	0.08	0.08	0.08	0.08	0.08	0.08
15 PCA features	0.08	0.05	0.08	0.08	0.08	0.08	0.08	0.08	0.08	0.08
17 PCA features	0.08	0.09	0.08	0.08	0.08	0.08	0.08	0.08	0.08	0.08
19 PCA features	0.08	0.08	0.08	0.08	0.08	0.08	0.08	0.08	0.08	0.08

B. Multi-layer Perceptron over PCA features

For this section we implemented an experiment that utilizes a shallow Neural Network with maximum of 10 layer size for a single hidden layer. We used the patternet implementation of Matlab [**?**]. The experiments we conducted first went over learning rate which we found 0.001 to be optimum. Network terminates after 100 epochs. Then we conducted a similar experiment as in the IV-A that goes over number of PCA features while increasing the size of the hidden layer up to 10. Maximum accuracy we obtained is 0.39 for layer size of 3 over 68 PCA features. Fig. IV-B illustrates test set accuracies for varios number of PCA features. Due to time constraints we were not able to experiment with more layers however our CNN experiments touch on that hyper-parameter although data is processed not as flatted 1D data like the experiment in this section.

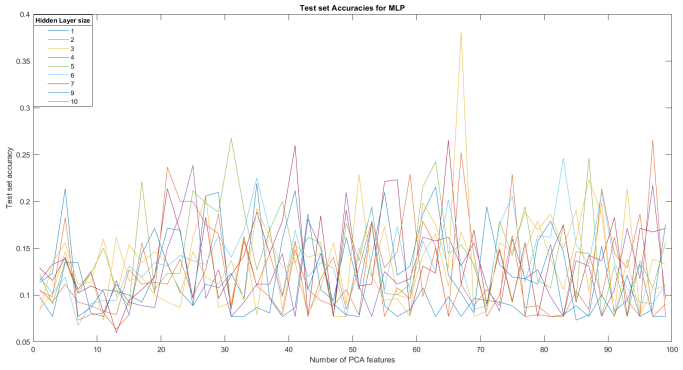

Fig. 6. Test set accuracy for MLP for various hidden layer sizes.

C. Convolutional Neural Networks

We implemented three different neural networks consisting of four convolution groups consist of convolutional layers, batch normalization after every convolutional layer, one max pooling and one dropout after the groups. Architectures have the same input and output shape for group input and output, they only vary in terms of number of convolutional layers and batch normalization applied right after it. In first architecture, named CNN1, one; in second architecture, named CNN2,

two; in third architecture, named CNN3, groups contain three convolutional layers, each followed by batch normalization. Each network accepts 3x30x40 images (downscaled from 3x320x240) containing grayscale, depth, and binary channels. Inputs are passed through the convolutional groups described above and flattened before being fed into fully connected layers having 512, 128, and 32 neurons. Dropout is applied after the first two layers. Output of the third layer is given to the softmax layer where number of output classes is 11.

TABLE II
MODEL CONFIGURATION PERFORMANCE.

Model	Learning Rate	Momentum	Epochs	Test Accuracy	Test Loss
CNN1	0.005	0.95	100	0.436	0.071
CNN1	0.005	0.9	90	0.341	0.075
CNN1	0.005	0.9	90	0.33	0.076
CNN2	0.005	0.9	80	0.58	0.067
CNN2	0.005	0.9	100	0.557	0.067
CNN2	0.005	0.95	100	0.519	0.069
CNN3	0.005	0.9	90	0.545	0.068
CNN3	0.005	0.9	90	0.561	0.067
CNN3	0.005	0.85	90	0.614	0.066

Experiments performed focused on two aspects of the networks: hyperparameter tuning and network capacity. Evaluated hyperparameters are learning rate, momentum, and train epochs. Note that since the hyperparameter hyperspace is large, and extensive parameter search is infeasable, hence, we selected a reasonable subset of values and trained a model for each combination. In terms of model capacity, we repeated the same experiments for hyperparameter tuning for each model. For ease of presentation, we share the results for three top performing models in Table 1. We experimented with learning rates 0.005, 0.001, 0.0005; momentum 0.85, 0.9, 0.95; and train epochs 50, 60, 70, 80, 90, 100. Results show that learning rate 0.005 gives the best test accuracy in all cases. Momentum varies between experiments implying it isn't a major factor in performance. For epochs, we see that high training epochs give better results. Network capacity also affects the performance, as CNN1 performs significantly worse than the others where CNN2 and CNN3 perform similarly. Train loss over 90 epochs for the best network is given in Figure 6, which suggests that training may continue after 90 epochs since loss is still improving.

V. DISCUSSIONS

GMM over PCA features results failed mostly due to variety of reasons. Those reasons include lack of computational power and data. GMM approach required more data samples to combat the need of high dimensionality. However if one tries to increas the number of sample one must also consider obtaining more expensive equipment since this amount of samples with the given reduction scheme in IV-A. section was using nearly 6GB of RAM on Matlab.Despite these obvious improvements one can try other things to get better result with this scheme;

1) To reduce dimensionality before the PCA is applied one can experiment the ways to fuse 5 channels(multiply, average ...) especially RGB channels. This would allow hardware to handle larger re-scaling of the images since we are reducing the image 3/5 of its presented size in this paper.
2) Rather than re-scaling the image one could have cropped the image using confidence or dept maps or even using a simple Neural Network that detects hands. Doing so would have made the in-the-wild dataset some-what regular where we could have fidled with a GMM that converges better.

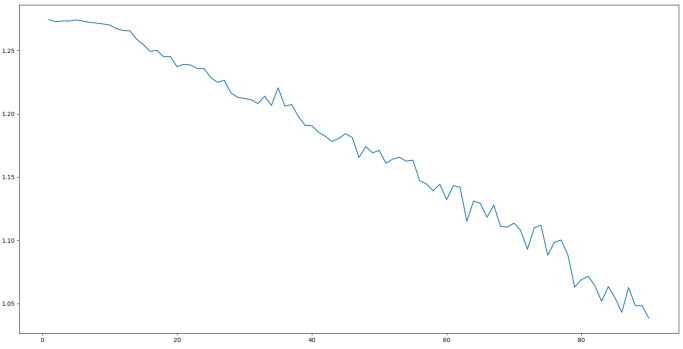

Fig. 7. Train loss for the best configuration CNN3, lr=0.005, mom = 0.85, epoch=90.

Mainly we were bounded by our hardware for this part.

MLP on PCA results got better test accuracy compared to GMM approach since it was able to predict data's trend more accuretly. This is due to gradient descent and weight learning rather than trying to cluster a high dimensional data(unlike GMM approach). One can claim that data distribution cannot be explained by a gaussian or a combination of gaussians since our GMM approach yielded 4 times worse results than MLP results. Yet there are many things that can be done to fine tune this approach if time constraint was not upon us;

1) One can increase the number of layers to make the network learn more indirect non-linear relations. While experimenting on size of PCA features.
2) As stated before one could have tried to crop the gestures out of the image using a network trained to detect bounding boxes for hands.
3) Different loss functions migth have been utilized rather than Matlab's default which is mean square error.

CNN results are satisfactory compared to other approaches in this report, obtaining %61.4 percent accuracy over 11 classes, yet it is inferior to reference paper [9] where the authors obtain an average test accuracy of %90. Our network, however, can be improved in following ways to close the gap in test accuracies:

1) Due to time restrictions we chose to apply only feature level fusion. This is problematic since raw binary and depth images are fed as input rather than to be used in preprocessing for segmentation, unlike in [9]. In this sense, future work can focus on generating segmented images and train the same networks on these along with greyscale images. Hand segmentation can also be

applied to the greyscale images as the dataset contains complex background in natural settings (such as other people, different illimunation, subject's other body part such as their head, etc.).

2) Hyperparameter optimization, due to the size of the parameter hyperspace, is restricted to a small subset described previously. Future work can focus on enlarging this hyperspace, specifically for larger train epochs since train loss curve still hasn't flattened even after 90 epochs.

The dataset we use can be found at https://lttm.dei.unipd.it/downloads/gesture/, namely, Creative Senz3D dataset.

VI. Contributions of each member

A. Berat Biçer

Berat proposed the project topic and found the dataset we used. He coded the convolutional neural networks and the relevant experiments. Other than contributing to the report overall such as content editing and spell-check, he also wrote the discussion chapter related to CNNS.

B. Alper Şahıstan

Alper has implemented the GMM on PCA and MLP on PCA experiment. He summarized the II. Related Works section in the proposal and in the final report. He also made some edits on both of the previously mentioned documents. Finally he provided results for the experiments he did on this report.

References

[1] W. Lu, Z. Tong, and J. Chu, "Dynamic hand gesture recognition with leap motion controller," *IEEE Signal Processing Letters*, vol. 23, no. 9, pp. 1188–1192, 2016.

[2] L. Bretzner, I. Laptev, and T. Lindeberg, "Hand gesture recognition using multi-scale colour features, hierarchical models and particle filtering," in *Proceedings of fifth IEEE international conference on automatic face gesture recognition*. IEEE, 2002, pp. 423–428.

[3] D. Del Vecchio, R. M. Murray, and P. Perona, "Decomposition of human motion into dynamics-based primitives with application to drawing tasks," *Automatica*, vol. 39, no. 12, pp. 2085–2098, 2003.

[4] T. B. Moeslund and L. Nørgaard, "A brief overview of hand gestures used in wearable human computer interfaces," *Computer Vision and Media Technology Lab., Aalborg University, DK, Tech. Rep*, 2003.

[5] M. Kolsch and M. Turk, "Fast 2d hand tracking with flocks of features and multi-cue integration," in *2004 Conference on Computer Vision and Pattern Recognition Workshop*. IEEE, 2004, pp. 158–158.

[6] X. Liu and K. Fujimura, "Hand gesture recognition using depth data," in *Sixth IEEE International Conference on Automatic Face and Gesture Recognition, 2004. Proceedings.* IEEE, 2004, pp. 529–534.

[7] B. Stenger, A. Thayananthan, P. H. Torr, and R. Cipolla, "Model-based hand tracking using a hierarchical bayesian filter," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, no. 9, pp. 1372–1384, 2006.

[8] A. Erol, G. Bebis, M. Nicolescu, R. D. Boyle, and X. Twombly, "Vision-based hand pose estimation: A review," *Computer Vision and Image Understanding*, vol. 108, no. 1-2, pp. 52–73, 2007.

[9] L. Minto and P. Zanuttigh, "Exploiting silhouette descriptors and synthetic data for hand gesture recognition," 2015.

[10] A. Memo and P. Zanuttigh, "Head-mounted gesture controlled interface for human-computer interaction," *Multimedia Tools and Applications*, vol. 77, no. 1, pp. 27–53, 2018.

[11] J. M. Saavedra, B. Bustos, and V. Chang, "An accurate hand segmentation approach using a structure based shape localization technique," in *VISAPP*, 2013.

[12] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[13] Matworks, "Matlab-pca," 2020, last accessed 11 January 2020. [Online]. Available: https://www.mathworks.com/help/stats/pca.html