Hand gesture recognition for Sign language

Cong T. Le, Vuong Q. Nguyen, Tuan T. Nguyen

***Abstract*—Hand gesture recognition is a challenging topic but its applications have a strong influence on many fields. Because of the difficult communication between the deaf and the others, the authors propose a novel sign language recognition system by recognizing hand gestures with webcam, processing images through thresholding to find contour features, and using a convolutional neural network classifier. Experimental results show that this system can recognize ASL alphabet with the accuracy rate over 96% in the best condition. With the result of this method, it both helps the deaf in communication and brings about many tendencies for developing more human-computer interaction systems using gestures to control in the future.**

***Index Terms*—Hand gesture, gesture recognition, sign language, ASL alphabet, CNNs.**

I. Introduction

Hand gestures have been getting more and more researched since the early years of the 21st century. Researchers have made many promising and innovative proposals in this field. The inventions which seem to be just fictions are gradually realized. For instance, people can use speech or body language to input information, to control the device, to play games, to assist the deaf or the dumb in communicating with the community through gesture recognition, or speech-to-text devices.

Hand gesture recognition for sign language is not a new topic. Many researchers have proposed different identification techniques and improved little by little identification accuracy. William T. Freeman and Michel Roth presented handwriting recognition based on the pattern recognition technique[1] developed by McConnell, who employed histograms of local orientation. They use the orientation histogram as a vector feature for gesture classification and interpolation. They can distinguish a small vocabulary of about 10 different gestures. After that, there are more approaches in hand gesture recognition such as vision based approach, approach based on instrumental gloves, approach based colored gloves. However, human life increasingly requires precision, convenience and customization capabilities. Therefore, gesture recognition should be easy to use, no need to wear physical equipment on hands and can be easily updated in the future.

Almost everyone does not understand the sign language. Therefore, communication between the deaf and the community needs to be done in a convenient and fast way instead of using conventional methods such as handwriting or interpreter. In addition, sign language can be interpreted and conveyed quickly without writing traditionally. Moreover, human and machine interactions tend to grow in the future. The gestures need to be accurately and rapidly identified so that they can be more natural for human interaction and control of the device. For example, computer control with virtual keyboard, virtual mouse, interactive gaming, screen control via hand gesture,...

The authors hope to describe successfully the sign language recognition system through hand gesture recognition with high accuracy. This research can help the deaf and dumb more naturally in communicating with the community. Furthermore, this will be the premise for further studies later in the field of gesture recognition as well as other fields of recognition.

The authors use a training set of 44 gestures and for each gesture consisting of 2400 images which are 50x50 pixels and in grayscale. The training dataset is used to train a Convolutional Neural Network classifier to classify the gestures. The authors use Keras[2] to conduct training that model. Then the experiments are carried out in a variety of cases which are in different conditions such as complex background, full of brightness, lack of brightness, wearing gloves,... The authors choose randomly 10 from 26 letters of ASL alphabet for the experiment, for each letter, there are various types of hand from many people participating in this experiment to make it more objective.

After experiments, the results illustrate that the sign language recognition system achieves over 80% accuracy in a variety of environmental conditions, the best accuracy is over 96% in simple background with proper illumination. Response time is about 1 second for this system to recognize successfully a gesture and print corresponding letter on screen. However, each time the environment changes, hand histogram needs to be set again. The disadvantage of this sign language recognition system that needs fixing is improving accuracy, being directly identified without setting hand histogram, and changing the environment also does not need to re-install but still retain the accuracy of the system.

The paper is organized as follows. Section II mentions several related works so far. Section III describes the proposed system with its architecture. Section IV comprises the experimental results and discussion. Section V concludes with future works.

II. Related works

The motivation of researching hand gesture recognition is to help the Deaf and Dumb community communicate with others conveniently throughout sign language, to enhance interactivity for convenient controls, gaming, and user experience. There are many studies in hand gesture recognition because of its promising motivation. Hand Gesture Histogram[1] of McConnell was published in 1995 and has been developed in the field of hand gesture recognition for use in sign language and control of physical devices. This research is based on recognizing hand patterns and almost static gestures to identify small vocabulary and control basic computer graphics. They apply a simple pattern recognition technique to the problem of hand gesture recognition. For static hand

gestures, they uses the histogram of local orientations as a feature vector for identification. In the paper "Hand Gesture Recognition for Sign Language Recognition: A Review"[3] presents that Hand gesture recognition is increasingly being studied, and more and more approaches are emerging, not only on computer vision but also on high-tech equipment such as gloves with sensors. Moreover, artificial intelligence is prevalent and machine learning is applied to solve many data problems. As a result, neural networks are more widely used in the field of hand gesture recognition. In addition, the paper "Sign Language Recognition using Convolutional Neural Networks"[4] presents the built-in recognition system that uses Microsoft Kinect, convolutional neural networks and GPU enhancements. Thus, the most common approach in hand gesture recognition is to use a computer vision to capture an image or a sequence of commonly used images and then train machine learning with CNNs. When the actual hand gesture is detected, the camera captures the image or sequence of images and uses what it has learned to categorize gestures, which is usually done by ANNs. The accuracy of hand gesture recognition has been greatly improved and tested in a variety of conditions.

III. Proposed system

The proposed system consists of five main stages which includes Dataset collecting, Image processing, CNN classifier training, Hand histogram setting, and Hand gestures recognizing that uses trained CNN model. The general architecture of the proposed hand gesture recognition system is shown below in Fig.1.

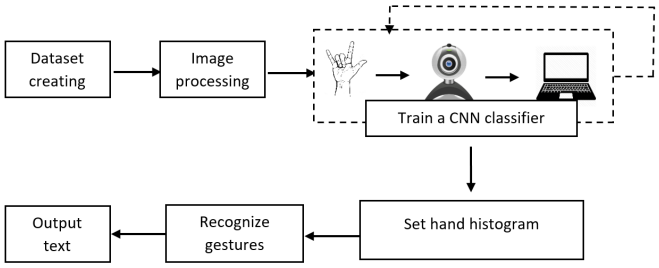

Fig. 1. Diagram of the sign language recognition system

A. Dataset structure

In this work, the first basic dataset is created for American Sign Language Alphabet which comprises 26 letters and 1-9 numbers in Fig.2. However, the dataset later easily add new gesture such as 10 gesture for number and some other gestures for a string. The total gestures of the dataset we use is 44 gestures. All static hand gestures which represent these letters are captured 2400 times from a standard laptop webcam with the resolution at 720x1280 pixels. The total of samples is 105600, 88000 samples for training, and 17600 samples for validating. When hand gestures are captured, the authors define the area of the hand on the screen with size 50x50 pixels. Then, the authors use OpenCV to crop that area of the hand gesture, turn it to greyscale, and save to dataset in Fig.3. The stage of normalizing the size and turning images to greyscale is image processing.

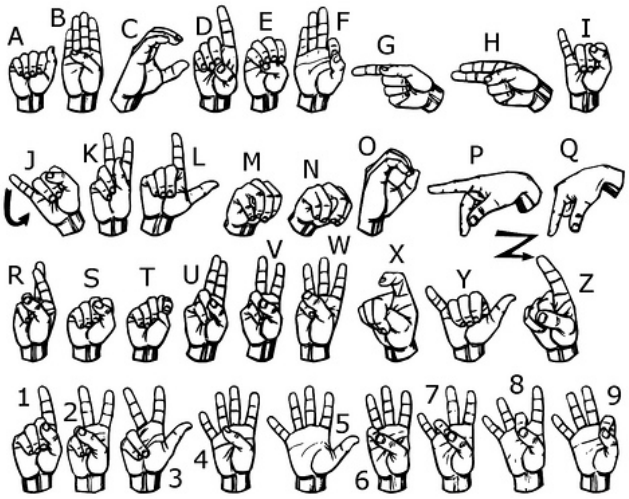

Fig. 2. American Sign Language Alphabet with 26 letters and 1-9 numbers

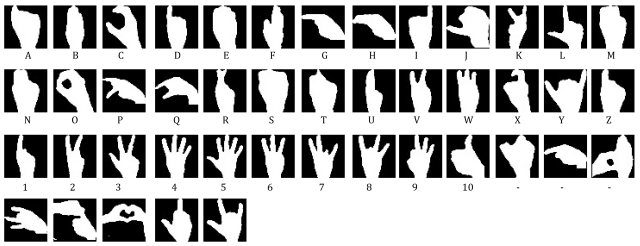

Fig. 3. Sign language with 44 gestures in the database

The images are captured with different signers, different lighting conditions and with different orientation. The experimental setup requires a simple plain background because it is easier to extract the hand gesture from a simple background. The signer is required to maintain a distance of less than 1 m from the webcam. The webcam is fixed at a predetermined position to capture the sign which is shown by the signer, and preventing the vibration of the camera. Proper illumination is to be given during the image acquisition so as to capture a clear hand image.

All the gestures in the American Sign Language database are static in nature except the letter 'J' and 'Z'. Since these two gestures are dynamic which involve movement of fingers, they are out of focus of our work, so we define it as a static gesture.

B. Image processing

The stage of processing images is to make the input images normalize and easily extract features before training CNN classifier and testing hand gesture later. The images we captured are in RGB color mode. As we mentioned, the square covering the hand gesture is set up when capture these gestures. The

raw image is cropped into 50x50 pixels and the new size image cover full area of only the hand gesture. The processing image is firstly turning the new cropped image into gaussian blur and then median blur, which makes other opearations much more exact and quicker to carry out, also for Otsu thresholding technique we will mention. Besides, gaussian blur and median blur can not only enhance the results of further processing but also smoothen the hand image which is used for thresholding in the next stage.

Next, we convert this 50x50 pixels image into a binary image by thresholding. There are many thresholding techniques existing, but the authors use Otsu thresholding after Gaussian filtering. This technique brings about a binary image that is very clear and distinct between black and white area, specifically white hand gesture histogram and the black background. We use this thresholding technique depending on Hand histogram setting which we will mention later.

Fig. 4. Processing images through these steps. (1)-(2): Gaussian blur the input image. (2)-(3): This image continues to be applied Median blur. (3)-(4): This step is applying Otsu thresholding technique[5] to the image to convert it into binary image which has just white and black. (4)-(5): This step is carried out to find contours of the hand when the signer recognizes gestures

With the simple background, this thresholding technique differentiates the foreground and background pixels easily and segments the hand from the image with high accuracy.

C. CNN classification

In machine learning, a convolutional neural network[6] is a class of deep, feed-forward artificial neural networks, most commonly applied to analyzing visual imagery. Convolutional networks were inspired by biological processes in that the connectivity pattern between neurons resembles the organization of the animal visual cortex. Individual cortical neurons respond to stimuli only in a restricted region of the visual field known as the receptive field. The receptive fields of different neurons partially overlap such that they cover the entire visual field. CNNs use relatively little pre-processing compared to other image classification algorithms. This means that the network learns the filters that in traditional algorithms were hand-engineered. This independence from prior knowledge and human effort in feature design is a major advantage. They have applications in image and video recognition, recommendation systems, and natural language processing.

In this work, we use CNN to classify hand gestures, and make it possible to recognize sign language exactly by this CNN classification. CNNs apply a series of filters to the raw pixel data of an image to extract and learn higher-level features, which the model can then use for classification. CNNs contains three components:

- Convolutional layers, which apply a specified number of convolution filters to the image. For each subregion, the layer performs a set of mathematical operations to produce a single value in the output feature map. Convolutional layers then typically apply a ReLU activation function to the output to introduce nonlinearities into the model.
- Pooling layers, which downsample the image data extracted by the convolutional layers to reduce the dimensionality of the feature map in order to decrease processing time. A commonly used pooling algorithm is max pooling, which extracts subregions of the feature map (e.g. 2x2-pixel tiles), keeps their maximum value,

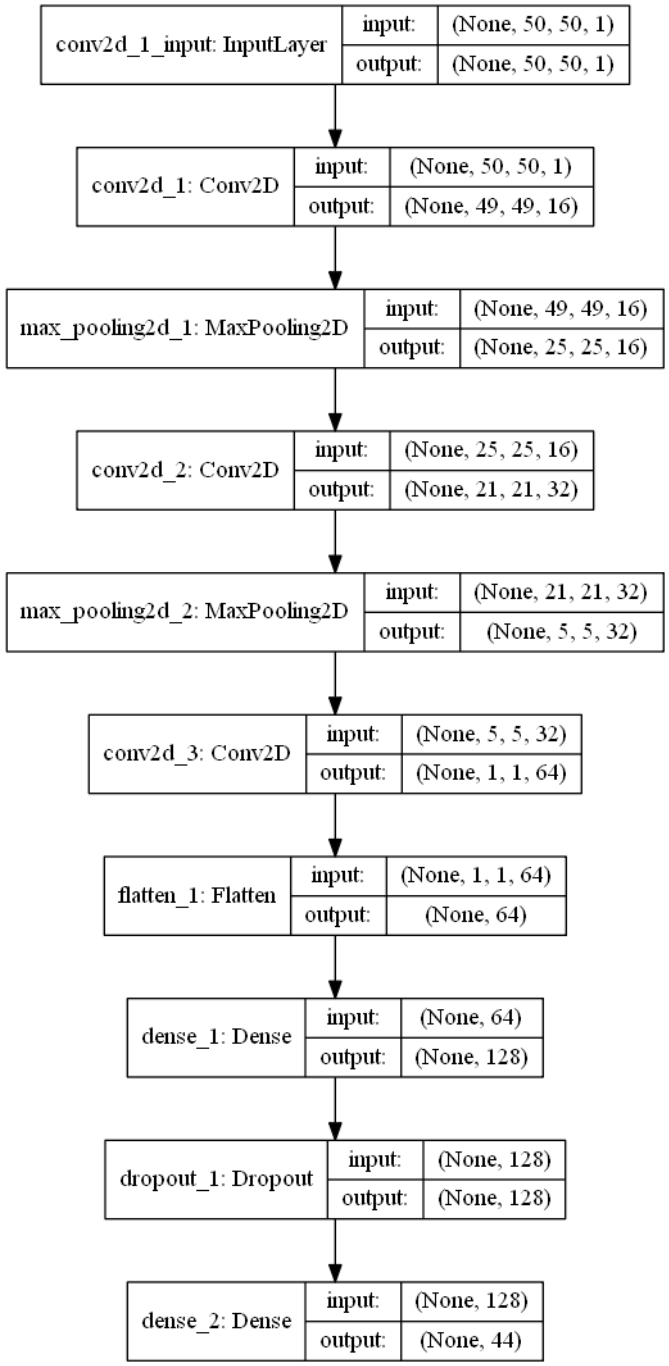

Fig. 5. The model of CNN Classifier

and discards all other values.
- Dense (fully connected) layers, which perform classification on the features extracted by the convolutional layers and downsampled by the pooling layers. In a dense layer, every node in the layer is connected to every node in the preceding layer.

Typically, a CNN is composed of a stack of convolutional modules that perform feature extraction. Each module consists of a convolutional layer followed by a pooling layer. The last convolutional module is followed by one or more dense layers that perform classification. The final dense layer in a CNN contains a single node for each target class in the model with a softmax activation function to generate a value between 0-1 for each node. We can interpret the softmax values for a given image as relative measurements of how likely it is that the image falls into each target class.

In this work, we use Keras to train CNN classifier. Keras is an open source neural network library written in Python. It is capable of running on top of TensorFlow[7], Microsoft Cognitive Toolkit[8], Theano[9], or MXNet[10]. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible. With Keras, we only need to add input layer, convolution layer, pooling layer, dense layer with the parameters as designed in the model Fig.5.

Input layer: The methods in the layers module for creating convolutional and pooling layers for two-dimensional image data expect input tensors to have a shape of [batch-size, image-height, image-width, channels] by default.

- Batch-size: Size of the subset of examples to use when performing gradient descent during training.
- Image-height: Height of the example images.
- Image-width: Width of the example images.
- Channels: Number of color channels in the example images. For color images, the number of channels is 3 (red, green, blue). For monochrome images, there is just 1 channel (black).

Our dataset is composed of monoschrome 50x50 pixel images, so the desired shape for our input layer is [batch-size, 50, 50, 1]

Convolution layer: conv2d_1, in our first convolutional layer, we want to apply 16 2x2 filters to the input layer, with a ReLU activation function. The filters argument specifies the number of filters to apply (here, 16), and kernel-size specifies the dimensions of the filters as [height, width] (here, [2, 2]).

Pooling layer: We connect our first pooling layer to the convolutional layer we just created. Here, our input tensor is conv2d_1, the output from the first convolutional layer, which has a shape of [batch-size, 49, 49, 16]. The pool-size argument specifies the size of the max pooling filter as [height, width] (here, [2, 2]). The strides argument specifies the size of the stride. Here, we set a stride of 2, which indicates that the subregions extracted by the filter should be separated by 2 pixels in both the height and width dimensions. Our output tensor produced by max_pooling2d_1 has a shape of [batch-size, 25, 25, 16]: the 2x2 filter reduces height and width by 50% each. We can connect a second convolutional and pooling layer to our CNN. For convolutional layer 2, we configure 32 5x5 filters with ReLU activation, and for pooling layer 2, we use the specs as a 5x5 max pooling filter with stride of 5. Note that convolutional layer 2 takes the output tensor of our first pooling layer (pool1) as input, and produces the tensor conv2d_2 as output conv2d_2 has a shape of [batch-size, 21, 21, 32], and 32 channels for the 32 filters applied. Pooling layer 2 takes conv2d_2 as input, producing pool2 as output pool2 has shape [batch-size, 5, 5, 32] (80% reduction of height and width from conv2d_2). conv2d_3 takes the output tensor of our second pooling layer (pool2) as input, and produces the tensor conv2d_3 as output. conv2d_3 has a shape of [batch-size, 1, 1, 64], and 64 filters applied.

Dense layer: Next, we want to add a dense layer (with 1,024 neurons and ReLU activation) to our CNN to perform classification on the features extracted by the convolution/pooling layers. Before we connect the layer, however, we flatten our feature map (pool2) to shape [batch-size, features], so that our tensor has only two dimensions. The inputs argument specifies the input tensor: our flattened feature map, pool2-flat. The units argument specifies the number of neurons in the dense layer (1,024). The activation argument takes the activation function; again, we add ReLU activation. To help improve the results of our model, we also apply dropout regularization to our dense layer, using the dropout method in layers. The rate argument specifies the dropout rate; we use 0.6, which means 60% of the elements will be randomly dropped out during training. Our output tensor dropout has shape [batch-size, 1024]. The final layer in our neural network is the logits layer, which will return the raw values for our predictions. We create a dense layer with 44 neurons (one for each target class 0-43). Our final output tensor of the CNN, logits, has shape [batch-size, 44].

D. Hand histogram setting

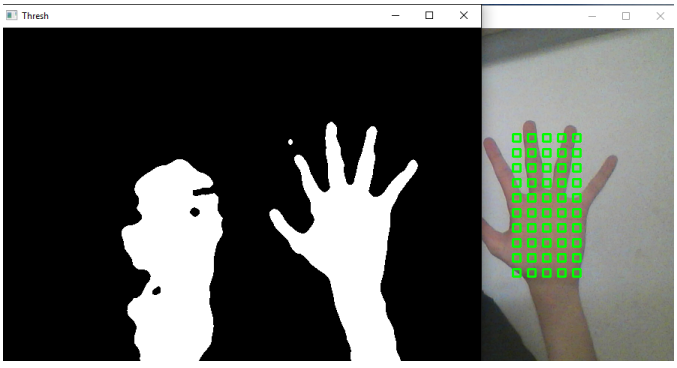

Fig. 6. Set hand histogram stage

This stage is mainly to get hand features and save it to database. The more clearly features are captured, the more accurate recognition gestures are. The signer do not need to do it again if the signer has already done it. But the signer needs to do it if the lighting conditions change.

The most important thing to use Otsu thresholding technique effectively is hand histogram setting. Before the signers want

to recognize their sign language, they need to set their hand histogram in the static square area on the screen, the best condition to set hand histogram is in proper illumination, and distance to webcam less than 1 meter. The signer need to capture their hand until the thesholding is clear and white hand is seperated from black background. This stage also uses Image processing that we mentioned above. When the signer save their hand histogram, it will be saved to database. The main saved data is the information of color of the hand. When the signer want to recognize hand gestures, the data of hand histogram will be used to process image and then classify hand gestures.

E. Hand gestures recognizing

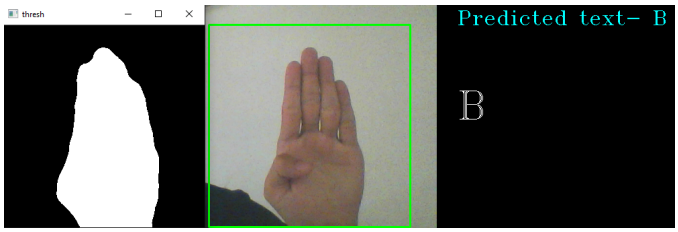

Fig. 7. Recognize hand gestures for sign language

After setting hand histogram, the signers only need to run recognizing hand gestures for sign language. This stage uses processing image, hand histogram data and CNN classifier to get correct letters or a string from static hand gestures. We can individually identify each gesture and then pair it up into a complete sentence.

IV. Experimental results and discussion

A. Performance CNN classifier training

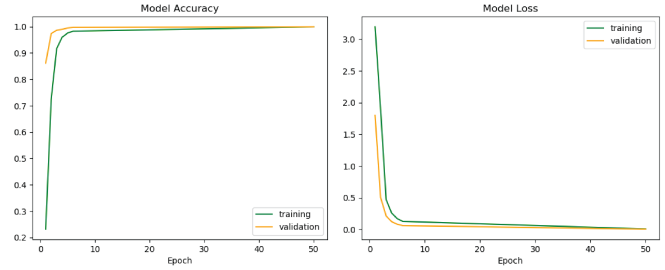

Fig. 8. Line charts show model accuracy and mode loss of training and validation during CNN training by Keras

When we load dataset to train CNN with Keras, we seperate it into 2 subsets: 88000 samples for training and 17600 samples for validation. The line chart above show that the model loss rate of validation is less than that of training and the model accuracy rate of validation is more than that of training. The model loss is asymptotic to 0 and the model accuracy is asymptotic to 1. The total CNN error during training is 0.05%.

B. Results of experiments in 6 conditions

In our experiments, we choose randomly 10 gestures from 44 available gestures in database which consist of 26 ASL alphabet from A to Z, 10 numbers from 0 to 9, and 8 others gestures. For each gesture, we carry out 26 tests, which are tested by random people in 6 condition we defined. We realize that our system can not recognize gestures in flare light condition because it makes webcam very poor in capturing gestures or some objects. Therefore, we test in 6 conditions such as simple background and good brightness; simple background and bad brightness; simple background, bad brightness, and high quality camera; simple background, bad brightness, high quality camera, and flash; simple background, good brightness, and wearing gloves; complex background and good brightness. In Fig.9, the confusion matrix shows the frequency of the gestures that can be recognized correctly and some gestures are easily mistaken with each other.

1) Simple background and good brightness: The result of recognizing hand gestures in this condition (1) shows that the accuracy is very high. In Fig.10, the average accuracy in condition (1) is 96.92%.

The accuracy in this condition is high because of the proper illumination and simple background. Webcam can capture clearly static hand gestures and then Image processing will be going well. Besides, with proper illumination, the color of the hand is identified correctly.

2) Simple background and bad brightness: In Fig.10, the average accuracy in condition (2) is 79.23%, it is so low compared with other conditions.

In condition with lack of brightness, webcam can not capture clearly static hand gestures, which results in not finding contour features after thresholding to match up with hand gesture value. In addition, the hand is unevenly colored causing confusion with the surrounding environment, so thresholding images will mistake hand with background.

3) Simple background, bad brightness, and high quality camera: In Fig.10, the average accuracy in condition (3) is 83.08%, it is just a little higher than condition (2).

High resolution camera with higher image quality do not still resolve the problem of light on the uneven hand in case of lack of brightness resulting in the fact that the skin is not recognized and some gestures are still mistaken such as 'A' and 'L'.

4) Simple background, bad brightness, high quality camera, and flash: In Fig.10, the average accuracy in condition (4) is 94.62%, it is so high as well as condition (1).

Using the flash helps increase the illumination of the hands. Thanks to that, the webcam recognizes exactly the hand gestures, from which the image processing is more accurate. In this case, light is added into this condition that are nearly identical to condition (1).

5) Simple background, good brightness, and wearing gloves: In Fig.10, the average accuracy in condition (5) is just 83.85%. It is so low compared with other conditions.

When using gloves, the size of hands is greater than normal, the color of hands is different. Depending on the type of gloves that give different colors are not uniform. Of course, each time we use different gloves, we have to set hand histogram again.

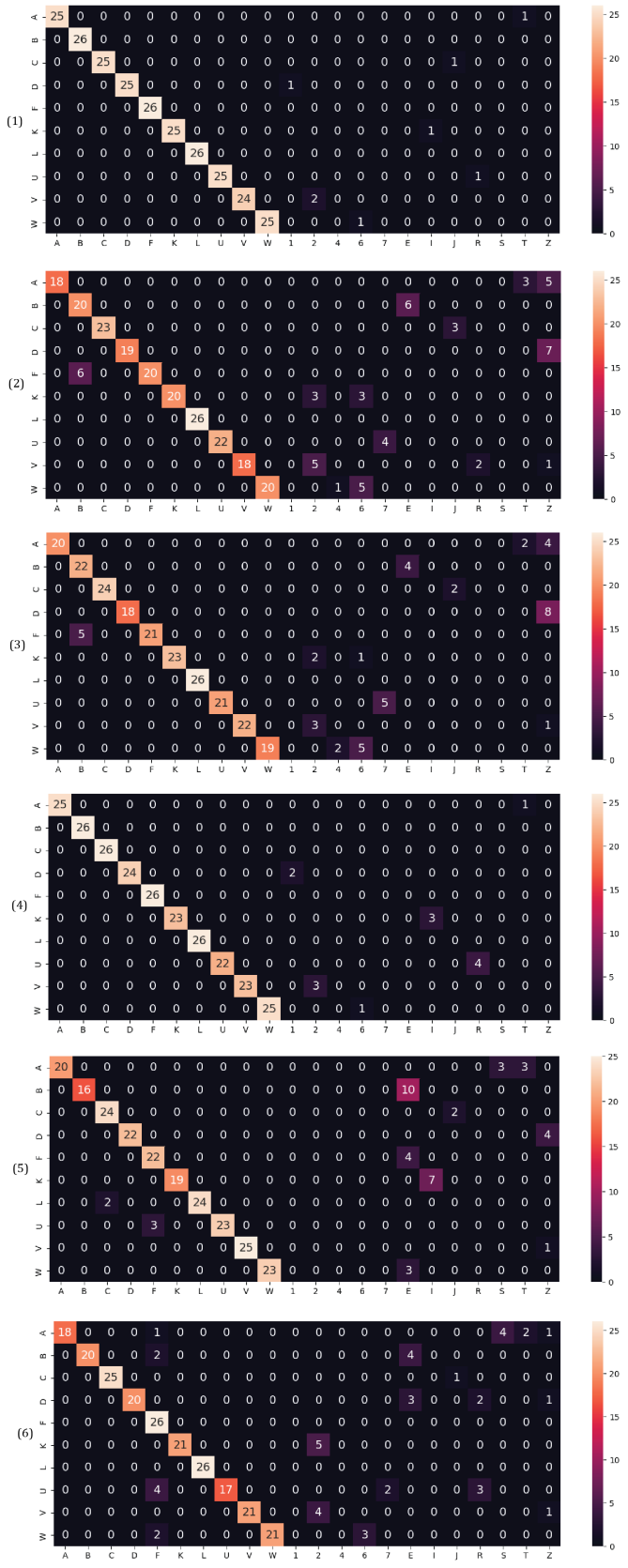

Fig. 9. Confusion matrix of the experiments with 10 gestures in conditions from (1) to (6). The y-axis is actual value and the x-axis is predicted value

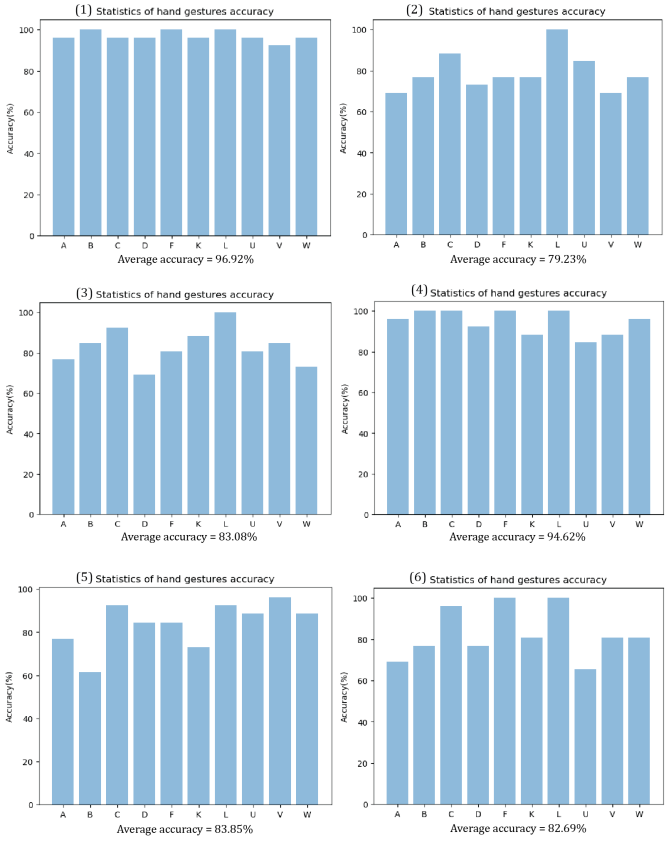

Fig. 10. The accuracy of 10 gestures in conditions from (1) to (6). The average accuracy is below the chart

Anyway, the test results are just fine but not high because the color of the gloves is confusing in the image processing.

6) Complex background and good brightness: In Fig.10, the average accuracy in condition (6) is 82.69%. It is also so low with the others.

The background which is not too complex with a few details can still be identified. The background has so many confusing details in analyzing and localizing the colors that correspond to the skin color. The 'A' gesture is a gesture with a high false rate. The most important thing to recognize hand gestures in this condition is locating the skin color to get rid of the background and then recognizing the gesture well.

C. Comparative study with other related works

TABLE I
COMPARATIVE STUDY WITH RELATED WORKS

Approach	Accuracy rate	Difficulties
Edge Oriented Histogram + Support Vector Machine[11]	93.75%	Only works for simple background
Perceptual Colour Space[12]	100%	Only 5 gestures
ANN based[13]	94%	Needs hand gloves with 13 sensors
PCA, Gabor filter and SVM[14]	95.2%	Only 11 gestures

Compared with the related studies in the field of hand gesture recognition for sign language, our research has also

TABLE II
RESULTS OF OUR WORK

Our work	Accuracy rate	Advantages
Contour based + CNN classifier	96.92% in the best condition 79.23% in the worst condition	- Flexible 44 gestures and new gestures can easily be added. - Do not need gloves just use bare hand to recognize. - Can work in complex background and poor light, but accuracy is reduced.

yielded very high accuracy in comparison to those studies. The positive side of our research is: flexibility, each time you want to add new gestures just add images of the gesture and train again; no need to use gloves with sensors, just use bare hands. However, our limitation is that: dynamic gestures are not recognized; failing in recognize gestures in night conditions or flare light; there is still a confusion among gestures that are nearly identical because of the oriented histogram of the hand.

V. CONCLUSION AND FUTURE WORKS

Processing image using Otsu thresholding after gaussian blur to check contour features by OpenCV and training a CNN classification to recognize hand gestures for sign language by Keras is investigated and analyzed in this paper. The experimental results show that the proposed system give an overall of accuracy rate over 80% in a variety of light and other conditions, specially over 96% in the good condition with simple background and proper illumination. This work brings about the flexible feature in adding new gesture easily, interpreting sign language to text for helping the deaf communicate with the others.

However, some letters of the alphabet are misclassified due to the shape of the gestures in front of the webcam as well as the similarity of some gestures such as A, E, S. When the light condition or color of the hand changes, the signers need to set hand histogram again because the webcam will capture badly the feature, color of the hand, then results will be fail in recognizing hand gestures. Besides, the complex background and harsh light condition prevent the proposed system from recognizing hand gestures. However, the proposed system has satisfactory results compared to other existing approachs.

In the future, the proposed system needs to improve the accuracy rate in a variety of conditions. An infrared camera can be used instead of webcam to be able to recognize hand gesture in lack of brightness condition. Machine learning is more and more developing, so this system can be applied with artififfical intelligent and no need to train this system with a large dataset because it can study and improve the accuracy rate little by little until it masters. This system need to develop to recognize dynamic gestures in the future. With this system and related works, the deaf and the others can easily communicate with each other, and the controlling devices in the future will be replaced with hand gestures to make life more fast and convenient.

VI. ACKNOWLEDGEMENT

The authors thankfully wish to acknowledge EvilPort[15] who is a B.Tech student of Computer Science and Engineering for his works in acquisition of input hand gestures in real time, preparing the image database of ASL alphabet, creating other gestures for some words in a system that can convert sign language into text. The authors honor EvilPort for his significant works in this system ”Hand gesture recognition for Sign language”.

REFERENCES

[1] William T. Freeman and Michal Roth, Mitsubishi Electric Research Labs 201 Broadway, ”Orientation Histograms for Hand Gesture”.
[2] F Chollet , ”Xception: Deep learning with depthwise separable convolutions”.
[3] Pratibha Pandey and Vinay Jain, ”Hand Gesture Recognition for Sign Language Recognition: A Review”.
[4] Lionel Pigou, Sander Dieleman, and Pieter-Jan Kindermans, Benjamin Schrauwen Ghent University, ELIS, Belgium, ”Sign Language Recognition using Convolutional Neural Networks”.
[5] JianGong, Liyuan Li, and WeinanChen, ”Fast recursive algorithms for two-dimensional thresholding”.
[6] Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner, ”Gradient-Based Learning Applied to Document Recognition”, Proc. IEEE, Nov 1998.
[7] Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, and Andy Davis ”TensorFlow: A system for large-scale machine learning”, 12th USENIX Symposium on Operating Systems Design and Implementation, USENIX Association 2016, pp. 265-283.
[8] Dennis Gannon, ”CNTK Revisited. A New Deep Learning Toolkit Release from Microsoft”.
[9] James Bergstra, Olivier Breuleux, Pascal Lamblin, Razvan Pascanu, Olivier Delalleau, Guillaume Desjardins, and Yoshua Bengio, ”Theano: Deep Learning on GPUs with Python”.
[10] T Chen, M Li, Y Li, M Lin, and N Wang, ”Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems”.
[11] S.Nagarajan, Assistant Professor, Department of CSE, Annamalai University, Annamalai nagar, Tamil Nadu, India, ”Static Hand Gesture Recognition for Sign Language Alphabets using Edge Oriented Histogram and Multi Class SVM”.
[12] S.Saengsri, V.Niennattrakul, and C.A.Ratana mahatana, ”TFRS: Thai Finger-spelling Sign Language Recognition System”, IEEE, 2012, pp 457-462.
[13] Rafael C.Gonzalez, and Richard E.Woods, ”Digital Image Processing”, Second Edition, 2005.
[14] Ali Karami, Bahman Zanj, and Azadeh Kiani Sarkalesh, ”Persian sign language recognition using wavelet Transform and neural Networks”, Journal of Expert Systems with Applications 38, 2011, 2661-2667.
[15] Github: EvilPort, a B.Tech student in Computer Science and Engineering, ”https://github.com/evilport2”.