Sign Language recognition and text to speech conversion

Meraz Hossain
Department of Computer Science and Engineering
BRAC University
Dhaka, Bangladesh
meraz.hossain@g.bracu.ac.bd

Mushfiqur Rahman
Department of Computer Science and Engineering)
BRAC University
Dhaka, Bangladesh
mushfiqur.rahman6@g.bracu.ac.bd

Suvarthi Chowdhury
Department of Computer Science and Engineering
BRAC University
Dhaka, Bangladesh
suvarthi.chowdhury@g.bracu.ac.bd

Shashwata das
Department of Computer Science and Engineering
BRAC University
Dhaka, Bangladesh
shashwata.das@g.bracu.ac.bd

***Abstract*—The majority of the population does not know sign language, and qualified interpreters are hard to come by; therefore, we developed a real-time method based on neural networks for American Sign Language (ASL) finger spelling. In our approach, the hand is filtered, and then the filtered hand is run through a classifier to determine what category the hand gestures belong to. For all 26 letters, our method achieves a 95.7**

***Index Terms*—component, formatting, style, styling, insert**

I. Introduction

Among the many sign languages used today, American sign language is the most widely used. People with DM only have a communication disability, so sign language is the only way for them to communicate. The term ”communication” refers to the process of conveying information from one person to another using a variety of media, including words, gestures, body language, and images. People who are deaf and/or mute (DM) often rely on hand gestures as a means of communication. Nonverbal communication takes place through the use of gestures, which are deciphered visually. Deaf and dumb people use sign language to communicate nonverbally. Our project's central goal is to develop a model that can recognize Fingerspelling-based hand gestures and combine them into a complete word.

II. Motivation

Normal people and DM people can't talk to each other because the structure of sign language is different from that of normal text. So they can only talk with each other through their eyes. If there is a common interface that turns sign language into text, it will be easy for other people to understand the gestures. So, research has been done on a vision-based interface system that would let DM people talk to each other without really understanding each other's languages. The goal is to make human-computer interfaces (HCI) that are easy to use and let the computer understand sign language. There are different sign languages all over the world, such as American Sign Language (ASL), French Sign Language, British Sign Language (BSL), Indian Sign Language, Japanese Sign Language, and others that are still being worked on.

III. Methodology

The way the system works is based on a vision. All the signs are made with the hands, so there is no need to use any artificial devices to communicate.

A. Data Set Generation

During the course of the project's development, we looked for pre-existing datasets but were unable to locate any raw image datasets that satisfied the needs of the task at hand. We could only find the datasets as RGB values. As a result, we went ahead and compiled our own set of numbers. The following are the procedures we used to compile our data set. In order to generate our dataset, we relied on the Open computer vision(OpenCV) library. To begin, we photographed roughly 800 examples of each ASL symbol for use in training and roughly 200 examples for use in testing. We begin by recording every single image that is displayed on our computer's webcam. We have outlined a ROI in blue in each of the images below to indicate where we want to focus our analysis. The following illustration shows how we take the RGB original and extract the ROI before converting it to a grayscale image. We finish by applying a gaussian blur filter to the image, which aids in feature extraction. Here is what the image looks like after being blurred using the gaussian method.

B. Gesture Classification

- ***The strategy we employed for this project is as follows:*** To predict the user's final symbol, our strategy employs two layers of algorithm.

- ***Algorithm Layer 1:***
  1) After feature extraction in opencv, the image is processed by first applying a gaussian blur filter and a threshold.
  2) The transformed image is fed into a convolutional neural network (CNN) model for prediction; if a letter is recognized for more than 50 frames, it is used in the final word.
  3) By inserting a blank symbol between words, we account for the space between them.
- ***Algorithm Layer 2:***
  1) Several groups of symbols are detected with consistent detection results.
  2) We then use classifiers developed specifically for these subsets to categorize data between them.
- ***Layer 1***
  CNN Model:
  1) ***1st Convolution Layer :*** The input image is 128x128, so the first convolution layer has to deal with that. The first convolutional layer uses 32 filter weights to process the data (3x3 pixels each). It will produce a 126x126 pixel image, one for each of the Filter-weights.
  2) ***1st Pooling Layer :*** The images are down sampled using max pooling of 2x2, which means that the highest value in each 2x2 square of the array is retained. This results in a reduced resolution of 63x63 pixels for our image.
  3) ***2nd Convolution Layer :*** The output of the first pooling layer, a 63-by-63 matrix, is used as input for the third layer, a convolutional neural network.
  4) ***2nd Pooling Layer :*** At the second pooling layer, the resulting images are downsampled once more, this time with a maximum pool size of 2x2. This brings the final image resolution down to 30x30.
  5) ***1st Densely Connected Layer :*** The output of the second convolutional layer is then reshaped into an array of 30x30x32 =28800 values for the first densely connected layer, which receives the images as input. This layer receives a 28800-valued array as input. This layer's output is sent to the second densely connected layer. To prevent overfitting, we have implemented a dropout layer with a value of 0.5.
  6) ***2nd Densely Connected Layer :*** The outputs from the first densely connected layer are then fed into the second densely connected layer, a fully connected 96-neuron network.
  7) ***Final layer :*** The final layer receives its input from the output of the second densely connected layer, and its number of neurons is equal to the number of classes being classified (alphabets plus blank symbol).
- ***Activation Function :***
  In each of the layers, we have implemented the Rectified Linear Unit, abbreviated as ReLu (convolutional as well as fully connected neurons). ReLu calculates max(x,0) for each input pixel. This contributes nonlinearity to the formula and assists in the learning of features that are more complex. It helps to eliminate the problem of vanishing gradients and speeds up the training process by reducing the amount of computation time required.
- ***Pooling Layer :***
  We use the Max pooling algorithm on the image that was provided to us, with the pool size set to (2, 2) and the relu activation function. This results in a reduction in the total number of parameters, which in turn results in a reduction in the total cost of computation and a reduction in overfitting.
- ***Dropout Layers:***
  Overfitting, where the network's weights are so tuned to the training examples that they don't perform well with new examples. This layer zeroes a random set of activations in that layer. Even if some activations are removed, the network should classify or output a specific example[1].
- ***Dropout Layers:***
  Adam optimizer updates the model based on loss function output. Adam combines the benefits of two stochastic gradient descent algorithm extensions: adaptive gradient algorithm (ADA GRAD) and root mean square propagation (RMSProp).
- ***Layer 2***
  To get as close as we can to correctly identifying the symbol being displayed, we are employing a two-layer algorithm to verify and predict symbols that are most similar to each other. While conducting tests, we discovered that the following symbols were misdisplaying and giving other symbols instead:
  1) For D: R and U
  2) For U: D and R
  3) For I: T,D,K and I
  4) For S: M and N

  As a result, we developed three distinct classifiers to handle the aforementioned scenarios:
  1) D,R,U
  2) T,K,D,I
  3) S,M,N
- ***Finger spelling sentence formation***
  1) **Implementation :**
     We print the letter and add it to the current string whenever the count of a letter detected exceeds a specific value and no other letter is close to it by a threshold. (In our code, we kept the value as 50 and the difference threshold as 20.) In that case, the current dictionary, which contains a count of the number of detections of the present symbol, is purged in order to eliminate the possibility of an incorrect letter being predicted. Whenever the count of a blank (plain background) detected is greater

than a particular value and the current buffer is empty, no spaces are detected. A blank is defined as an area with no pattern or texture. In any other scenario, it prints a space after the word it thinks will be the last one, and the current sentence is appended to the one that comes after it.

2) **Autocorrect Feature :**
A python library called Hunspell suggest is used to suggest correct alternatives for each (incorrect) input word, and we display a set of words matching the current word from which the user can select a word to append to the current sentence. If the user selects a word that matches the current word, it will be added to the current sentence. This helps to reduce the number of spelling errors that are made and provides assistance in predicting difficult words.

C. Training and Testing :

To filter out unwanted details, we apply a gaussian blur to our grayscale input images after converting them from RGB. The images are resized to 128x128, and an adaptive threshold is used to separate the hand from the background. After performing all of the aforementioned operations on the input images, we then feed them into our model for training and testing. The prediction layer makes an educated guess as to which category the image belongs to. This means that the output is scaled so that the sum of each class's values equals 1. The softmax function was instrumental in allowing us to accomplish this. Initially, the prediction layer's output will differ significantly from the true value. We've improved things by training the networks on labeled data. When classifying data, the cross-entropy is used as a performance metric. It is a continuous function that is nonnegative when equal to its labeled value and exactly zero otherwise. Accordingly, we made sure the cross-entropy was as close to zero as possible. We do this by modifying the neural networks' weights within the network layer. The cross entropy can be easily computed with a built-in function in TensorFlow. After determining the cross entropy function, we used Gradient Descent to fine-tune it; in particular, we used the Adam Optimizer, which is currently the best gradient descent optimizer available.

IV. Challenges Faced :

In the course of this project, we encountered a number of obstacles. The lack of a complete dataset was the first challenge we encountered. Since it was much more manageable to work with only square images, we preferred dealing with raw images as CNN in Keras. We looked everywhere, but we couldn't find a suitable dataset, so we compiled one ourselves. The second challenge was deciding which filter to apply to the images so that useful features could be extracted and the resulting image used as input for the CNN model. We experimented with a number of filters, such as binary threshold and canny edge detection, but ultimately decided on gaussian blur. We had more problems with the accuracy of the model we trained in earlier stages, but we solved them by making the input images bigger and by using a better dataset.

V. Results :

With only layer 1 of our algorithm, our model achieves an accuracy of 95.8%; by combining layers 1 and 2, we achieve an accuracy of 98.0%, which is higher than the accuracy of most of the current research papers on American Sign Language. The vast majority of these studies examine the effectiveness of hand detection hardware like the Kinect. In [2], the authors use convolutional neural networks and Kinect to create a system that can recognize Flemish sign language with an error rate of only 2.5%. It is shown in [3] that a recognition model can be constructed with a vocabulary of 30 words and an error rate of 10.90% using a hidden markov model classifier. Average accuracy for 41 static Japanese sign language gestures is 86%, as reported in [4]. Map [5], which made use of depth sensors, was 99.99% accurate for previously observed signers and between 83.58% and 85.49% accurate for new signers. CNN was used in their recognition software as well. Unlike some of the other models presented here, ours doesn't employ a background subtraction algorithm. There may be some inaccuracies once we try to use background subtraction in our project. While most of the aforementioned work relies on Kinect sensors, we set out to develop a solution that could be implemented with commonplace equipment. Our model's use of a regular laptop's webcam is a huge advantage because sensors like Kinect are not only uncommon but also prohibitively expensive for the vast majority of potential users. Our confusion matrices are provided below.

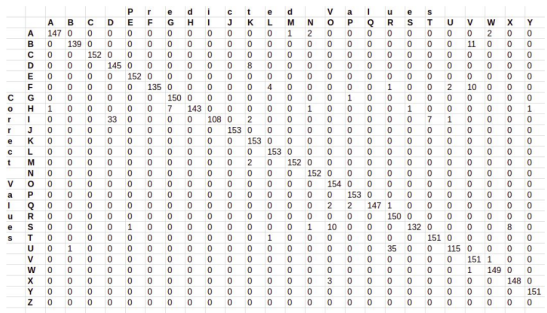

Fig. 1. **Algorithm 1**

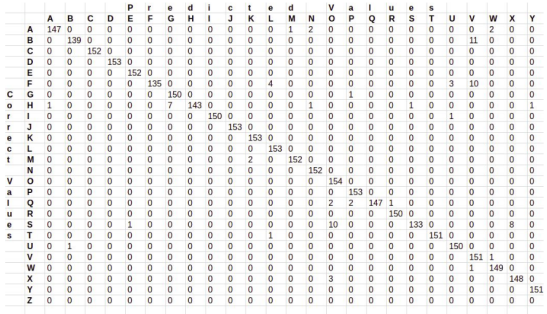

Fig. 2. **Algorithm 1 + Algorithm 2**

VI. Future Scope :

By experimenting with a variety of algorithms for background subtraction, we intend to increase the level of accuracy that can be achieved regardless of the complexity of the background. Additionally, we are contemplating the possibility of enhancing the preprocessing in order to improve the accuracy with which gestures can be predicted under low-light conditions.

VII. Conclusion :

The purpose of this report is to describe the development of a functional real-time vision-based American Sign Language recognition system for DM people using ASL alphabets. On our dataset, we ended up achieving an accuracy rate of 98.0%. Following the implementation of two layers of algorithms, in which we verify and predict symbols which are more similar to one another, we were able to improve our ability to make accurate predictions. By operating in this manner, we are able to recognize almost all of the symbols, provided that they are displayed accurately, there is no noise in the background, and the lighting is sufficient.

References

[1] Aeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convol utional-Neural-Networks-Part-2/

[2] Pigou L., Dieleman S., Kindermans PJ., Schrauwen B. (2015) Sign Language Recognition Using Convolutional Neural Networks. In: Agapito L., Bronstein M., Rother C. (eds) Computer Vision - ECCV 2014 Workshops. ECCV 2014. Lecture Notes in Computer Science, vol 8925. Springer, Cham.

[3] Zaki, M.M., Shaheen, S.I.: Sign language recognition using a combination of new vision based features. Pattern Recognition Letters 32(4), 572–577 (2011).

[4] N. Mukai, N. Harada and Y. Chang, "Japanese Fingerspelling Recognition Based on Classification Tree and Machine Learning," 2017 Nicograph International (NicoInt), Kyoto, Japan, 2017, pp. 19-24. doi:10.1109/NICOInt.2017.9.

[5] Byeongkeun Kang , Subarna Tripathi , Truong Q. Nguyen "Real-time sign language fingerspelling recognition using convolutional neural networks from depth map" 2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR).