Development of a Sign Language Recognition System

Ray Huda, Mitchell Zinck
School of Information Technology,
Carleton University, Ottawa, Canada
rayhuda@cmail.carleton.ca, mitchellzinck@cmail.carleton.ca

***Abstract*—This paper presents the development of a Sign Language Recognition System utilizing convolutional neural networks (CNNs) to facilitate communication for individuals with hearing impairments. With over 430 million people globally living with disabling hearing loss, the need for accessible tools for learning American Sign Language (ASL) is critical. Our approach involves two CNNs: one for detecting static ASL signs and another for recognizing dynamic signs, including challenging letters like J and Z. Leveraging datasets such as Sign Language MNSIT (SMNIST) and EMNIST Letters (LMNIST), coupled with finger tracking techniques, our system enables users to practice ASL through accessible means like phones or computers. Experimental results demonstrate high accuracies, paving the way for improved inclusivity and accessibility in communication for the hearing impaired. Further research could focus on enhancing robustness and expanding the system's capabilities.**

***Index Terms*—SMNIST, MNSIT, EMNIST, SLD, Sign Language Detection, Computer Vision**

I. Introduction

According to the World Health Organization, Over 430 million people are currently living with disabling hearing loss[1]. This represents 5.73% of the global population, less then half of that amount of people know how to communicate using American sign language (ASL)[2]. A common method of learning ASL is through attending classes, however, this may not be possible for many people do to things like scheduling issues, cost, or lack of accessibility. Being able to practice and learn ASL through the use of a phone or computer would allow for many more people to have access to the language and increase communication for people with disabling hearing.This program could function similar to applications like Duolingo where a user is presented with a sentence then, instead of speaking or writing the answer, they would sign it into their camera.

This paper shows the implementation of a convolutional neural network (CNN) for the purpose of detecting ASL signs within 28x28 images. The dataset used is the Sign Language MNSIT dataset featuring 27,455 images across the signs for A to Y with J and Z being omitted due to them requiring motion to convey[3]. In order to provide coverage for J and Z, we used a combination of finger tracking and a separate convectional neural network trained on the EMNIST letters dataset[4] which allows us to not only detect J and Z, but also all other letters when drawn in the air with a finger. The EMNIST dataset contains a total of 88,800 which we used in a 70/30 train/test split. Enabling the in air drawing of other letters opens the path to other potential methods of communication between those who are unable to speak or hear but do not yet know ASL.

II. Steps of Proposed Method

To begin, we need to acquire a data set, set up our convolutional neural network, and train. Python version 3.10 was selected for the environment, and Pycharm was used for the interpreter due to its general versatility for python and the debugging tools it provides. For the datasets, both can be downloaded from Kaggle.com[3][4] and come as CSV files split into a training and testing file in a 70/30 split. the sign language MNSIT (SMNIST) data set was able to be loaded easily however for the letters EMNSIT (LMNIST) dataset, the testing data was not configured in a usable form for our purposes leading to us splitting off 30% of the training data and reserve it for testing. The data is originaly stored in CSV format as a 784x1 array of values representing each pixel so reshaping it to 28x28 when loading is required. A CNN was then set up using TensorFlow and Keras based on a format from Abhishek Singh[5] as seen in the figure below.

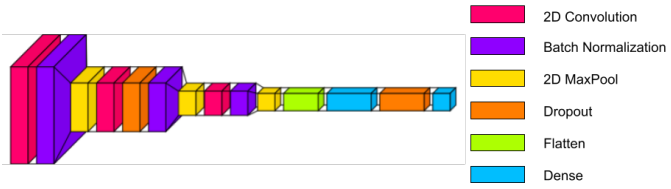

Figure 1: Convolutional Neural Network Layout

Both the CNN for SMNIST and LMNIST used the same layout other than the outputs having 24 and 26 values respectively.The size for the convolutions was 3x3 for all of the convolution layers and 2x2 was used for the pool size for MaxPool. both of the networks were then trained for 2000 epochs taking 8-10 hours each.

The next step in the sign language detection is to find a hand in our scene then extract it to feed into our neural network. This is accomplished through the use of the libraries opencv[6] and mediapipe[7]. Opencv is used to read data from the webcam and process the images, and mediapipe is use to determine the key points of a hand found in the image. A

square centered on the middle of the hand is then created from these key points. This square is used to extract our region of interest that encompasses the hand while removing extraneous data that is not needed. The region of interest is then down sampled to 28x28 and converted to a 1x28x28x1 tensor to be used with our SMNIST trained CNN. The CNN will then output a 24x1 array of integers with on value being 1 (the predicted value) and all others are 0. The index of the predicted value can be used to determine the letter equivalent of a predicted sign.

In order to get the dynamic letters (J and Z) working, we need to now track the finger and convert that motion to an image we can pass into our LMNIST CNN. We need to first determine when we should be tracking either the pointer finger or the pinky finger. Thanks to mediapipe's hand tracking, we already have a list of 21 points that make up the hand. The greatest outlier can then be found and used to determine which point is sticking out the most. We also remove point 0 from this calculation as it is often found as the greatest outlier when we don't want it and it is unnecessary for the calculations.

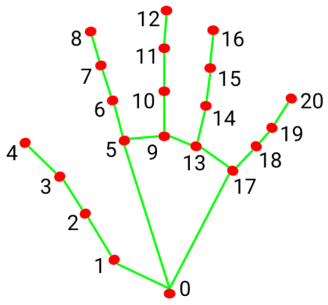

Figure 2: Mediapipe Hand Detection Key Points[8]

When we detect that either point 8 or 20, the fingers used to draw Z and J respectively, is the greatest outlier we begin to draw circles on a blank canvas at the given point's location. This is done continuously allowing us to draw the path the finger follows. Once no outlier/finger is detected for 1.5 seconds we assume that the user is done performing the sign and we start passing it to the LMNIST CNN. This is done by first downsizing the image to 28x28 and normalizing it from 0-1. Surprisingly, we then need to reflect the image along the Y-axis and rotate our image 90 degrees as this is how the images in our dataset are originally stored. We can then use our LMNIST detector with the transformed data to show what letter was drawn giving us not only the ability to detect the signs for J and Z, but also all other letters when drawn in the air.

III. Set Up and Experimental Results

All required code can be downloaded from: https://github.com/Zinckle/OSS4009_SignLanguageRecognition

A. CNN Training Set Up

when training the convolutional neural networks using TrainLetterDetect.py or TrainSLD.py, start by ensuring that the following libraries are installed:

- matplotlib
- seaborn
- keras
- sklearn
- pandas

we then need to ensure that the variables train_data and test_data are set to the relevant CSV files as strings depending on which detector is being trained. The files can be located anywhere but the path to the files needs to be included in the variable. The variable for epochs can be increased or decreased depending on the training duration desired and the parameters for datagen can be tweaked to increase robustness but this will come at the cost of training time and potentially overall accuracy.The CNN can then be run and the resulting network with trained weights will be saved as either lmnist.h5 or smnist.h5 depending on which network has been trained.

B. Detection Set Up

In order to run the SignLanguageDetection.py code, a web-cam and a computer capable of running tensorflow is required. Start by ensuring that the following libraries are installed:

- cv2
- mediapipe
- time
- math
- numpy
- tensorflow

and that the lmnist.h5 and smnist.h5 models are available to be loaded and placed in the same directory as the Sign-LanguageDetection.py file. When attempting to detect sign language, ensure that the background is relatively clear with balanced lighting in the scene to get best results.

C. Results

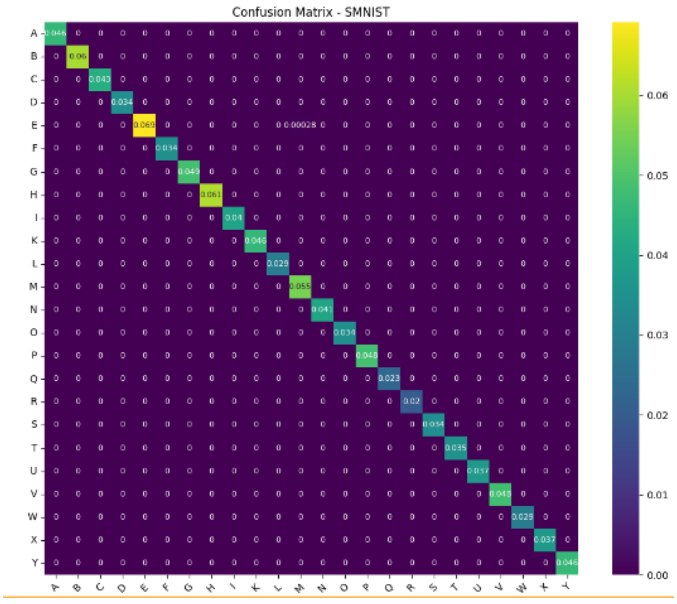

Figure 3: SMNIST Confidence Matrix

When we look at the confidence matrix from SMNIST trained CNN, we see that there is very little confusion between entries. the only one we see for our test cases is between the signs for E and M which makes sense as they are similar signs. The testing accuracy that we found was 99.972% however, this is only on the ideal case and when attempt to detect from the live feed the results were not as accurate due to differences in things like lighting and backgrounds.

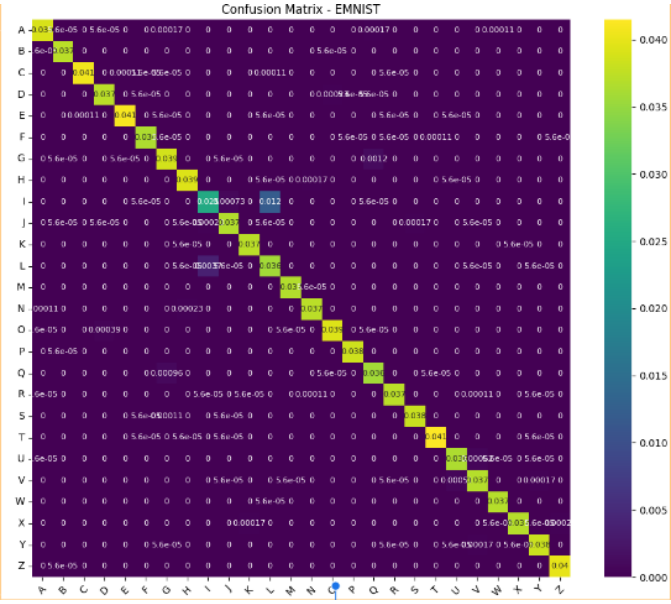

Figure 4: LMNIST Confidence Matrix

The confidence matrix for our LMNIST trained CNN shows slightly different results which aligns with it's 97.308% accuracy. The greatest source of confusion is between the drawn letters for L and I which makes sense as the way some people write their Ls and Is look very similar to each other and without the context of the word it is difficult to determine which is which.

IV. Our Contributions

A. Overall Contributions

The overall contributions we made in this paper include the method for extracting the hand for processing with our SMNIST CNN, and our method for the detection for dynamic signs. Throughout our research, we found very little documentation covering simple solutions for the implementation of the signs for J and Z and our method of utilizing finger tracking and LMNSIT is a conceptually straight forward method. Furthermore, many of the sign language detection methods that we found during research only functioned within a specific region of the screen where as ours allows the sign to be positioned anywhere it is fully visible to the camera.

B. Individual Contributions

For the individual contributions made to this paper, Mitchell Zinck was responsible for the majority of the code however much of the development was done collaboratively over discord calls. Ray Huda was responsible for a majority of the presentations and reports made throughout the term. The final paper was a combined effort split between both parties.

V. Conclusion

In conclusion, this paper presents a comprehensive approach to the development of a sign language recognition system using convolutional neural networks (CNNs). The system aims to bridge communication barriers for individuals with hearing impairments by enabling them to practice and learn American Sign Language (ASL) through accessible means such as phones or computers.

The proposed method involves the utilization of two CNNs: one for detecting static ASL signs and another for recognizing dynamic signs. Our contributions extend beyond implementation by introducing novel techniques for hand extraction and dynamic sign detection enabling users to sign anywhere visible to the camera, unlike many existing solutions confined to specific regions of the screen.

In summary, this work represents a significant step towards developing accessible tools for sign language learning and communication, with potential implications for enhancing inclusivity and accessibility for individuals with hearing impairments. Further research could explore methods for improving robustness under diverse environmental conditions and expanding the system's vocabulary and capabilities.

VI. References

References

[1] "Deafness and hearing loss," World Health Organization, https://www.who.int/health-topics/hearing-loss#tab=tab_2.

[2] M. R. TA;, "How many people use sign language? A national health survey-based estimate," Journal of deaf studies and deaf education, https://pubmed.ncbi.nlm.nih.gov/36423340/.

[3] Tecperson, "Sign language mnist," Kaggle, https://www.kaggle.com/datasets/datamunge/sign-language-mnist.

[4] C. Crawford, "EMNIST (extended mnist)," Kaggle, https://www.kaggle.com/datasets/crawford/emnist.

[5] A. Singh, "Sign-language MNIST problem(american sign language)," Medium, https://medium.com/@abhkmr30/sign-language-mnist-problem-american-sign-language-48896ea960e0.

[6] "OpenCV-python tutorials," OpenCV, https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html.

[7] "MediaPipe — google for developers," Google, https://developers.google.com/mediapipe.

[8] Mediapipe Hands: 21 landmarks. — download scientific diagram, https://www.researchgate.net/figure/MediaPipe-Hands-21-landmarks-13_fig1_362871842.