SignNet: Recognize Alphabets in the American Sign Language in Real Time

Zeqiang Lai
Beijing Institute of Technology
laizeqiang@outlook.com

Zhiyuan Liang
Beijing Institute of Technology
secondauthor@i2.org

Kexiang Huang
Beijing Institute of Technology
thirdauthor@i2.org

Abstract

We propose an end to end deep learning approach called SignNet for American sign language recognition task. Specifically, our deep learning architecture employs VGG network to extract latent representation of gesture images. We shrink and remove some layers in VGG so that it can run in real time on a single CPU. To address the generalization problem, we record a dataset without background using Average Background Subtraction algorithm. Evaluation results show that our method could accurately recognize gestures in the environments with clean backgrounds. The code of this work has been made available at `https://github.com/Zeqiang-Lai/SignNet`

1. Introduction

With the continuous development of computer vision technology, new interactive technologies such as gesture recognition, face recognition, and human posture recognition have been widely used. Among them, gesture recognition is a research hotspot in the field of human-computer interaction (HCI). It is mainly used by a computer to detect, track and recognize user gestures through video input devices (cameras, *etc.*) to understand human intentions. The robust gesture recognition system has a positive impact on the development of many applications such as sign language recognition, robot control, and human-computer interaction.

Sign language recognition is an important application field of gesture recognition. Compared with dynamic gestures, static gesture recognition is less difficult and has higher real-time performance, which can meet the real-time requirements of sign language recognition. The research object of static sign language recognition is a gesture image at a certain point in time, and the recognition result is related to the appearance features of the hand in the picture, such as position, contour, and texture. We chose American Sign Language (ASL) as our sign language recognition standard, which includes 26 different gestures of English letters (as shown in Figure 1), and also includes two different gestures for delete and space commands.

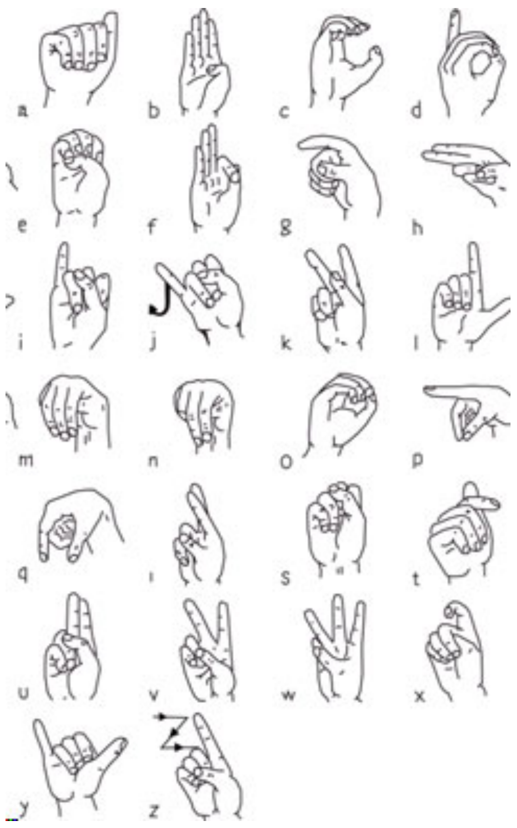

Figure 1. Sign Language Reference

From the perspective of computer vision, our gesture recognition is mainly divided into four stages: 1. Collection of gesture images; 2. Segmentation of gesture images;

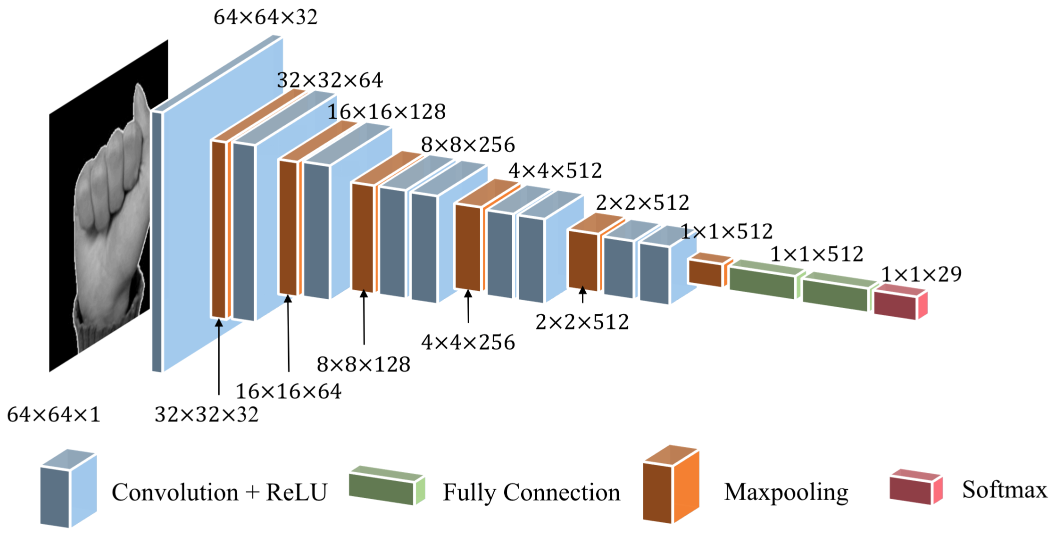

Figure 2. Network architecture

3. Feature extraction; 4. Gesture recognition.

Through the overall framework of these four steps, we used the VGG16 Network to complete the sign language recognition task under the ASL mark.

2. Related Work

ASL recognition is not a new computer vision problem. In the past two decades, researchers have used classifiers from various categories, which can be roughly divided into linear classifiers, neural networks and Bayesian networks.

Although linear classifiers are relatively simple models and therefore easy to use, they require complex feature extraction and preprocessing methods to succeed [9] [3]. Singha and Das used Karhunen-Loeve transform [4] to obtain 96% accuracy on 10 categories of one-hand gesture images.

Bayesian networks like Hidden Markov Models have also achieved high accuracy [10] [5]. These are particularly good at capturing temporal patterns, but they require well-defined models that have been defined before learning.

Some neural networks have been used to process ASL translation [3, 6]. It can be said that the biggest advantage of neural networks is that they learn the most important classification features. So far, the most relevant work is L. Pigou et al. applied CNN to classify 20 Italian gestures in the ChaLearn 2014 humanoid gestures competition [7]. They used Microsoft Kinect on the full-body image of the person performing the gesture, and the cross-validation accuracy reached 91.7%. As in the case of the aforementioned 3-D gloves, Kinect can capture depth features, which helps to classify ASL signs.

3. SignNet

As it is mentioned before, we formulate sign language recognition task as a classification problem and we use a modified version of VGG Network [8] to train a classifier for images with different signs.

In detail, our modifications can be summaried as followed. First, we reduce the size of last fully connected network from 4096 to 512. Secondly, we reduce the number of filter of the intermediate convolutional blocks. We make these modifications based on the requirement of live recognition and the assumption that VGG is way to large for the sign language recognition task, which was shown to be correct by the experiment on the ASL Dataset.

The network architecture is shown in Figure 2. The total number of parameter is 10m approximately and the model can achieve nearly 15 fps on an Intel Core I7 CPU.

4. Vanilla Experiments

In this section, we introduce the vanilla experiments we have conducted in the early stage of the project.

4.1. ASL Dataset

ASL Alphabet[1] is a collection of images of alphabets from the American Sign Language, which is public available on Kaggle. The training data set contains 87,000 images which are 200×200 pixels. There are 29 classes, of which 26 are for the letters A-Z and 3 classes for SPACE, DELETE and NOTHING. The test data set contains a mere 29 images, which is too small for an effective test.

For testing, we use another dataset - ASL Alphabet Test[2]. This dataset is collected by a different person and it

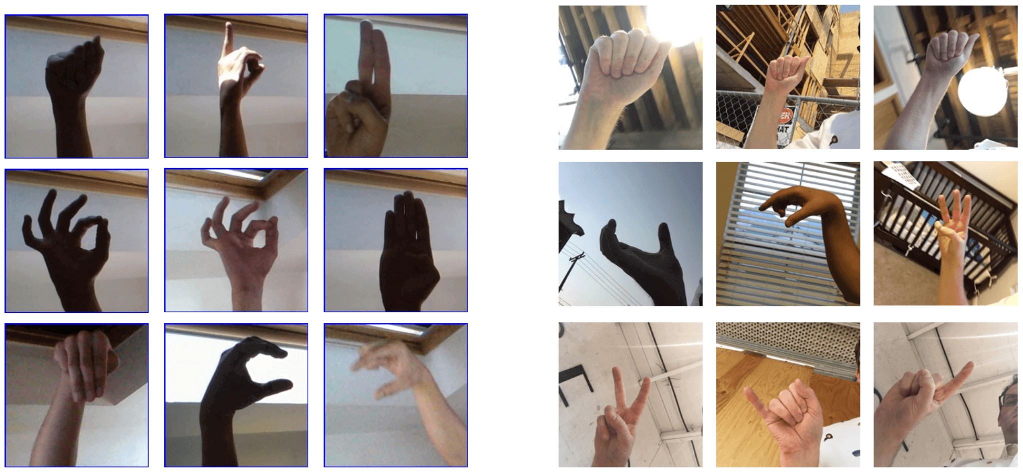
Figure 3. Preivew of ASL Alphabet (Left) and ASL Alphabet Test (Right)

consists of 870 images with various background. There are 30 images for each symbol, A-Z, delete, space, and nothing and every image is 200×200 8-bit photo, which is the same as it is in ASL Alphabet.

See Figure 3 for a preview of ASL Alphabet and ASL Alphabet Test.

4.2. Training Detail

We implement our model with Pytorch, and train it with cross entropy loss and Adam optimizer[6] with default parameters. The learning rate is set to 1×10^{-4} for the first epoch, and decayed by a ratio of 0.7 every other epoch.

During training, we randomly flip images horizontally. After that, the images are converted into grayscale and resized to 64×64 to reduce computational complexity.

4.3. Results

The training and testing results are shown in table2. We can see that the model can overfit on the training set, but it works badly on test set. This poor performance might be the result of the lack of background variation in the training set, or the regularization on the model. In any cases, current model cannot work properly.

Dataset	Accuracy
ASL Alphabet Train	86906/87000 (100%)
ASL Alphabet Test	145/870 (17%)
Our ASL Test Set	1087/21418 (5%)

Table 1. Results on ASL dataset.

5. Improvement

The ASL Alphabet dataset has small background variations and it is hard to train a model that works well in real environment, which is shown in the experiments mentioned in the previous section.

As a result, we decide to make a dataset on our own. In order to reduce the negative effect of background on prediction, we use background subtraction technique to remove it while recording dataset, and we use the same method to remove background in testing and inference stages. In this way, we can basically obtain a model that is able to work under limited environment.

5.1. Custom Dataset

Instead of collecting more data with different backgrounds, we choose to make a dataset without background directly, because the former is generally time-consuming and laborious.

Similar to the ASL Alphabet, our custom dataset also consists 29 types of gestures where each type includes 900 and 300 images (200px×200px) for training and testing, respectively. Background of each video is removed using a background subtraction algorithm, as it is shown in Figure 4.

5.2. Background Subtraction

We choose Running Average Background Subtraction algorithm to detect and remove background, because it is fast and easy to implement.

The key idea of this algorithm is to compute a reference background image using running average of first few frames in a video, and subtract the current image with respect to the reference background, and finally compare it with to the

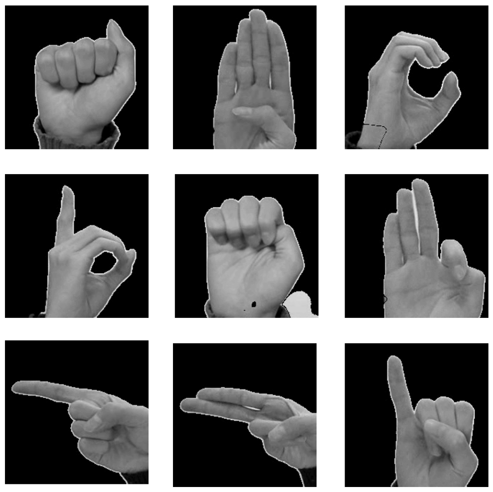

Figure 4. Preview of custom dataset.

certain threshold. Pixels that exceed threshold are considered to be foreground, while all the others are background.

After comparison, we could get a background mask. Instead of using binary mask directly, we use the mask to remove background to get a grayscale hand image, because some signs require counting fingers to be recognized correctly. See Figure 4.

5.3. Experiment

Training Setting With the new custom dataset, we use the same loss, optimizer, and data preprocessing techniques to train a new recognition model. (See previous section 4.2 for more training detail).

Quantitative Results It takes about 5 epochs for the recognition model to converge and another 15 epochs for further tuning. As it is shown in Table 2, our model works perfectly on training set, but the test accuracy is slightly lower.

Dataset	Accuracy
Custom Train	25804/25809 (100%)
Custom Test	7364/8700 (85%)

Table 2. Results on custom datasets.

Error Analysis Figure 5 shows the confusion matrix on the custom test set. We can observe from the figure that the majority of gestures can be recognized correctly, but there are some gestures are frequently misclassified to certain types. For example, almost all the gesture M are misclassified to N. If we have a look on the gestures of M and N (See Figure), it can be understood that our model work bad on these two ones, because they are similar to each other. In summary, our model work well on the most part of cases, but can be confused with some similar gestures, which might be improved with more training examples.

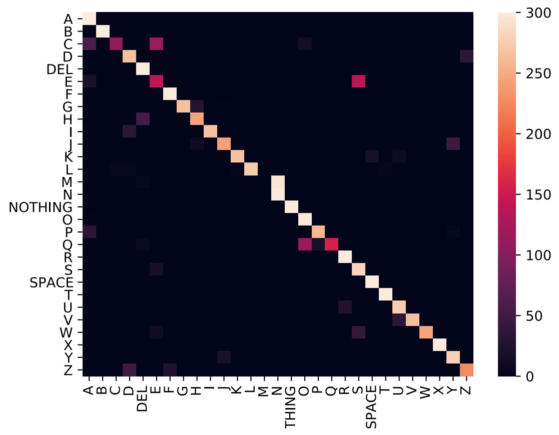

Figure 5. Confusion matrix on custom test set.

Speed Test To evaluate the run-time performance of our approach, we perform a set of tests under different hardwares and platforms. As it is shown in Table 3, our model can easily achieve real time performance both GPU and CPU where tests are conducted on a series of static images (rows of type static). To demonstrate the performance on live stream, we also implement a live script using OpenCV. The test result with such script achieve 400fps on GPU and is approximately real-time on CPU.

Hardware	Platform	Type	FPS
Intel Core i7	macOS 10.15.7	live	15.73
		static	81.00
Nvidia RTX 2060	Windows 10	live	31.05
		static	534.12
Nvidia GTX 1070	Ubuntu 20.04.1	static	453.78

Table 3. Average FPS of our model in different settings. Type ”live” means it is tested using our live_demo script which uses OpenCV to record video in real time, and type ”static” means it is tested using static images.

6. Conclusion

In this work, we propose SignNet for recognizing American sign language in real time. To address the problem of lacking background variation, which encounters poor performance, we use the background subtraction technique to remove the background in each training and testing sample.

By combining the original ASL Alphabet dataset with the custom dataset collected by ourselves, our SignNet achieves the satisfactory performance within a short response time.

In future, it is worth to investigating how to increase the accuracy of some confusing gestures, and improve human-computer interaction experience.

References

[1] ASL Alphabet.

[2] ASL Alphabet Test.

[3] Dewinta Aryanie and Yaya Heryadi. American sign language-based finger-spelling recognition using k-nearest neighbors classifier. In *2015 3rd International Conference on Information and Communication Technology (ICoICT)*, pages 533–536. IEEE, 2015.

[4] Jason Atwood, Matthew Eicholtz, and Justin Farrell. American sign language recognition system. *Artificial Intelligence and Machine Learning for Engineering Design. Dept. of Mechanical Engineering, Carnegie Mellon University*, 2012.

[5] Maher Jebali, Patrice Dalle, and Mohamed Jemni. Extension of hidden markov model for recognizing large vocabulary of sign language. *arXiv preprint arXiv:1304.3265*, 2013.

[6] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[7] Lionel Pigou, Sander Dieleman, Pieter-Jan Kindermans, and Benjamin Schrauwen. Sign language recognition using convolutional neural networks. In *European Conference on Computer Vision*, pages 572–578. Springer, 2014.

[8] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[9] Joyeeta Singha and Karen Das. Hand gesture recognition based on karhunen-loeve transform. *arXiv preprint arXiv:1306.2599*, 2013.

[10] Thad Starner and Alex Pentland. Real-time american sign language recognition from video using hidden markov models. In *Motion-based recognition*, pages 227–243. Springer, 1997.