Real-Time Sign Language Recognition

Jake Stringfellow
Khoury College of Computer Sciences
Northeastern University
Boston, USA
stringfellow.j@northeastern.edu

Abstract

Millions of people routinely use sign language as a form of communication. Despite this, American Sign Language datasets are scarce compared to spoken language datasets, and as a result this creates a unique problem to be solved: the recognition and translation of a visual language with limited data.

We attempt to solve this problem using deep learning techniques, specifically a convolutional neural network, to recognize and translate sign language letters displayed in real time video frames. Experimental model building and image processing through these methods produced effective, yet inconsistent results., showing promise for future research in the area.

I. Introduction

American Sign Language (ASL) is one of the predominant forms of communication used by deaf, disabled, or hard of hearing people in the United States and parts of Canada. For those who communicate strictly through ASL, communicating with a person who does not know it may prove to be very difficult. American Sign Language translation would pave the way for greater communication between the hearing-impaired and the non-impaired, and thus translation of this language would be a meaningful and important development.

Sign languages rely on expression through motion and gestures primarily by a person's hands. Sign Language Recognition (SLR) is a fundamental task in visual sign language research, and is considered to be a challenging classification problem. Since they are visual languages, they convey information using features including hand shape, movement, and pose.

The Real-Time Sign Language Recognition program looks to provide a solution to this problem through deep-learning techniques. A sign language recognition model is trained on an American Sign Language based dataset comprised of hand gestures signing letters in the ASL alphabet. This model then predicts a presented hand pose/gesture frame-by-frame from a live video feed.

We conclude the paper in section V by discussing our findings and possible future work. I hope that this simple approach to Sign Language Recognition provides some insight to this interesting and important topic, and one day can help ease the burden on those with hearing disabilities.

II. Related Work

In this section, we will briefly review the related topics, including sign language recognition, pre-training strategy and hand-modeling technique.

2.1 Sign Language Recognition

Lack of scale in data is commonly seen as a hindrance in the progress of sign language translation[1][2][4]. The difference in scale between datasets available for Sign Language and datasets available for other language based recognition is immense. In efforts to mitigate this setback, computer vision researchers working with sign language recognition systems have developed unique ways of interpreting data and training models.

The study by Chen *et al*[1] proposed a transfer learning baseline for sign language translation as a way to deal with the bottleneck the lack of sign language data available places on recognition model training. Their proposal was the development of a system that has a model pretrained on the general domain of human actions, another pretrained on sign to gloss datasets, and then a module to link the two networks together to create a baseline model for future research.

The study by Hu *et al*[2] introduced a self-supervised pre-trainable model that specializes in recognizing hand pose for the purpose of sign language translation. Their approach to create effective models while working with less data was to pretrain the model by masking and reconstructing the data they have.

The study by Camgoz *et al* in 2020[3] introduces a novel transformer based architecture that jointly learns continuous sign language recognition and translation while being trainable in an end-to-end manner[3]. Their approaches had significant performance increases across challenging datasets. Their research placed emphasis on the task of recognizing features of input along with the comprehension of signer interactions and movements in their 3D space.

The study by Camgoz *et al* in 2017[4] proposed a novel deep learning approach to solve simultaneous alignment and recognition problems. By dividing the recognition problem into SubUNets, they modeled the relationships between the SubUNets to solve the task. Their approach mimicked human learning techniques, and they learned that by better constraining the problem they could improve performance.

The study by Koller *et al*[5] took a novelty approach to weakly supervised video learning. Since this method was relevant to sequence learning problems[4] and experimented with sign language, it was directly relevant to the current study, similarly exploiting sequence constraint as the 2017 Camgoz *et al* study[4]. They embed powerful CNN-LNSTM models following a hybrid approach to discover attributes that were normally not distinct enough to be identified by models. They then applied this approach to sign language by recognizing mouth shape, hand shape, and sign language in parallel.

2.2 Sign Language Translation

Many sign language recognition and translation efforts involve identifying a single gloss word label for a given video clip[1]. A gloss label is another word for the transcription of sign language, where the “gloss label” is the word associated with the meaning of the sign[3]. While many successful studies test their models on video clips, our work focuses on analyzing individual frames given by live video[1].

Translating a raw video sequence to a spoken language sentence is no simple task. Some works treat this as a neural machine problem, but the problem with this is that those types of systems benefit from huge amounts of data to work with, something that sign language translation does not have[1].

2.3 Hand Modeling Technique

Common approaches to sign language recognition unsurprisingly involve the hands, usually as a key visual landmark to build the system upon. Where these methods differ from my work, however, is the recognition of multiple frames of video translated to a single sign[1]. The program created for this project is a 1 to 1 mapping of an image frame to a sign label.

Recognizing hand gesture is a difficult task due to the small area it takes up in input streams and the huge variety of backgrounds based on hand movement and location[2]. Many deep learning based methods learn feature representations adaptively from the cropped RGB hand sequence. The model in this project is trained and tested on similarly cropped images, separating the hand from the rest of the image in order to achieve recognition.

The hand modeling technique in the study by Hu *et al* inspired the use of the hand-tracking module implemented with the use of MediaPipe’s hand landmarks detection guide. The MANO statistical model and the model used in this study are both notably capable of representing hand geometric changes in a low-dimensional shape and pose space.

While this model was only used for the user’s ability to see the pre-trained hand model’s detection of the user’s hand, and did not factor into the recognition task at the time of writing, the hand tracking module will prove invaluable in the next stages of developing this program.

III. Methods

3.1 Dataset

In this section, we introduce our method for sign language recognition and translation. The goal was to develop a convolutional neural network that can predict the associated letter directly from the input of a video frame featuring a signing hand.

In order to be able to recognize sign language the model would have to be trained on a dataset featuring plenty of signs (in this case only letters) in a variety of backgrounds and poses. I chose the Sign-MNIST dataset because it was open sourced through Kaggle, and was modeled after the classic MNIST data set which I was familiar with. This dataset follows the same CSV format and represents the American Sign Language alphabet.

Each training and test case has a label that maps to an alphabetic letter, unfortunately there is no data in this set for J or Z due to those signs requiring motion along with the gesture to be accurate. It is worth noting that while there is data for “T”, the data presented does not accurately represent the sign that represents that letter. While the dataset format is patterned to match closely with MNIST it contains roughly half of the amount of data.

Each data case represents a 28x28 pixel image with grayscale values between 0-255. The original image data represents multiple users represented different users gesturing different signs against different backgrounds to provide variety to the set. In an effort to increase the scale of the dataset, the data was augmented to crop to hands-only, grayscale, resize, and creating many different variations of existing data cases. Modifications include different types of filtering (including increasing and decreasing contrast and brightness), and rotations of the images.

While the data augmentation was effective in drastically increasing size of the dataset, it presented challenges with preprocessing image frames later in the process of recognizing signs in live image frames.

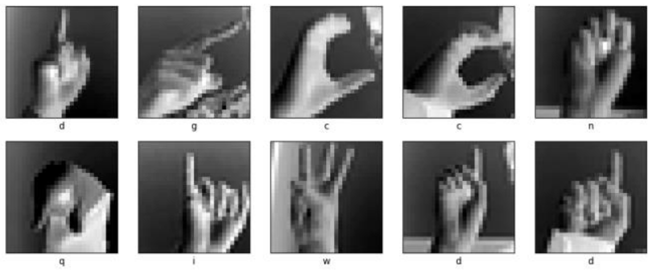
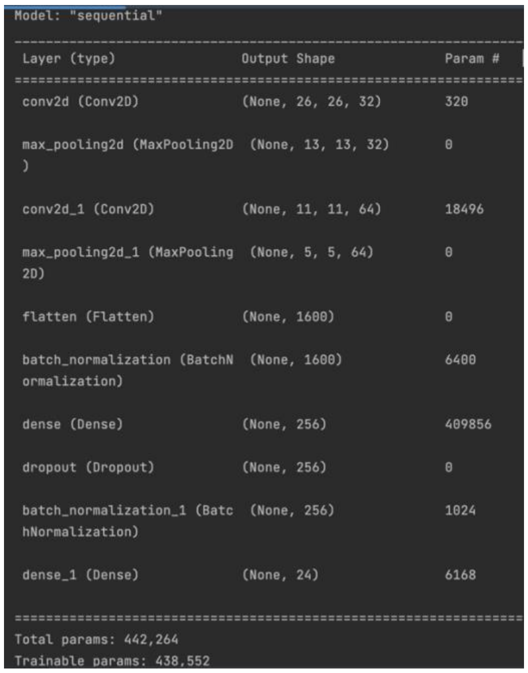


Figure 1: The first 10 images of the training data, and their corresponding sign letter.

3.2 CNN Model

To load our training and test data we created a function that used pandas and numPy to read the csv provided and store the image data within X and labels within Y variables. To create and train the convolutional neural network model we used the TensorFlow library and Keras framework.

We implemented a sequential model in a neural network class with the architecture ordering as follows: a convolutional layer with 32 3x3 filters, max-pooling layer with a 2x2 window, another convolutional layer with 64 3x3 filters and a ReLU function applied, another max-pooling layer with a 2x2 window, a flatten layer (fix output of convolutional layer), a batch normalization layer (allow for faster training), a dense layer, a dropout layer (to avoid overfitting), a batch normalization layer then another dense layer to output the softmax probablilty of the classes



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320

 max_pooling2d (MaxPooling2D  (None, 13, 13, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 11, 11, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 5, 5, 64)         0
 2D)

 flatten (Flatten)           (None, 1600)              0

 batch_normalization (BatchN  (None, 1600)             6400
 ormalization)

 dense (Dense)               (None, 256)               409856

 dropout (Dropout)           (None, 256)               0

 batch_normalization_1 (Batc  (None, 256)              1024
 hNormalization)

 dense_1 (Dense)             (None, 24)                6168

=================================================================
Total params: 442,264
Trainable params: 438,552
```

Figure 2. The summary of our model's architecture provided by the keras framework's summary() function.

To create a model in our training program we call the constructer the neural network class and initialize the model variable to the model attribute of the neural network class, then compiled it using the keras framework's "compile()" function. To train and test the model we used the "fit()" function provided by the keras framework given the training image and label data, test image and label data, and 5 epochs to train. We store this training information in a "history" variable.

Once the model has been trained and tested, we save the state of the model for use in the main file.

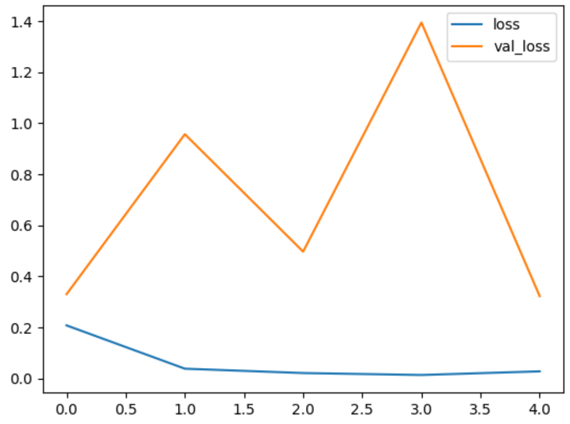


Figure 3. Plot of loss and accuracy for training epochs.

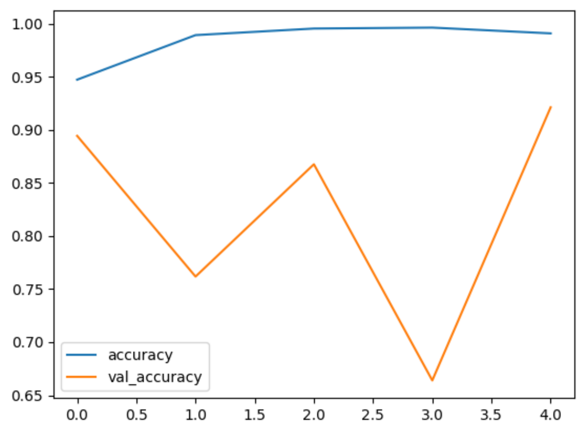
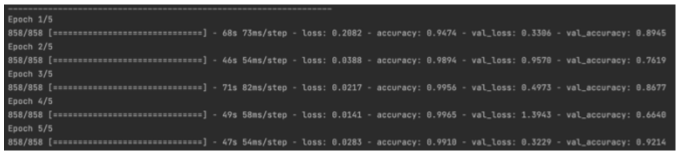


Figure 4. Plot of loss and accuracy for validation (testing) epochs.



```
---------------------------------------------------------------
Epoch 1/5
858/858 [==============================] - 68s 73ms/step - loss: 0.2082 - accuracy: 0.9474 - val_loss: 0.3306 - val_accuracy: 0.8945
Epoch 2/5
858/858 [==============================] - 46s 54ms/step - loss: 0.0388 - accuracy: 0.9894 - val_loss: 0.9570 - val_accuracy: 0.7619
Epoch 3/5
858/858 [==============================] - 71s 82ms/step - loss: 0.0217 - accuracy: 0.9956 - val_loss: 0.4973 - val_accuracy: 0.8677
Epoch 4/5
858/858 [==============================] - 49s 58ms/step - loss: 0.0141 - accuracy: 0.9965 - val_loss: 1.3943 - val_accuracy: 0.6640
Epoch 5/5
858/858 [==============================] - 47s 54ms/step - loss: 0.0283 - accuracy: 0.9910 - val_loss: 0.3229 - val_accuracy: 0.9214
```

Figure 5. Training data per epoch given by the console.

3.3 Main Application

The main file of our program begins by initializing the dimension values for a region of interest rectangle that will later be drawn on the live user video. The user video is opened, then we initialize our hand-tracker object, which continuously draws landmarks, based on the MediaPipe hand landmarks detection guide, on detected hands in the frame. We then load our trained model from the support file, then open the class names file and print the available recognizable signs to the user.

Within the main application loop we initialize the region of interest rectangle. This region of interest will be a cropped version of the user's live video frame, and will allow the neural network model to begin making predictions on a more focused live input. The region of interest is drawn to the screen as a cyan square to guide the user's hand sign placement.

The portion of the image inside the region of interest is stored separately from the full live video frame, and is then preprocessed by converting it to grayscale, normalizing the values, then decreasing the brightness and raising the contrast of the image slightly. After the image is filtered it is reshaped to a 28x28 image with one color channel. This preprocessing is done to make the test frame as close to the training data as possible.

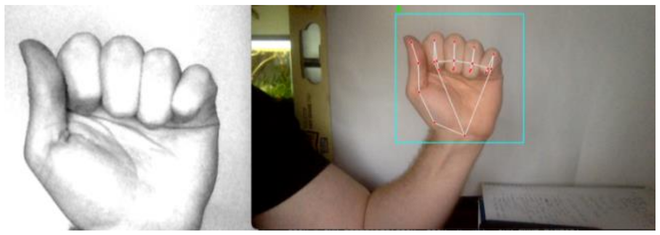


Figure 5. Output of the program when presented with the sign for "A". (Left) Preprocessed image fed to the CNN. (Right) Live video stream from user capture device.

We then have the model predict the sign given in the frame, and display the prediction label above the region of interest. We also call the handsFinder() method of the handTrackingModule to display the tracked landmarks of the user's hands, as well as coordinates of the base of their middle knuckle, as a representation of the general location of their hand in the image.

To set up the user's environment for more accurate sign recognition and translation, the user should run the program in a well lit room, with a light background. Upon running the main function the user needs to present their hand in the region of interest, in the pose of the sign they wish to have translated.

3.4 Hand-Tracking Module

The hand-tracking module is a module based on the hand landmarks detection guide provided by MediaPipe. The module features objects and functions also provided by MediaPipe. It is used to localize key points of hands by using a machine learning model to operate on image data and output landmarks as image coordinates.

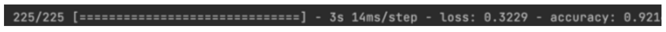
The constructor of the handTracker class initializes all attributes of the handTracker including the maximum detectable hands, the minimum detection confidence, and tracking confidence. The handsFinder() method tracks the hands in an image (used with frames in the main file) and draws the hand landmarks and connections onto the frame. The positionFinder() method finds the x and y coordinates of each hand point (landmark) in the given image, it returns a list of all landmarks found and can be accessed in the main file.

While the handTrackingModule is only being utilized as a way for the user to better detect visualize the hand tracking and pose, it has many implications on the future work of the project.

IV. Experiments and Results

Most of the experimenting within the project took place within the building of our neural network model architecture and preprocessing the live video frames for model recognition. After considering convolutional neural networks modeled for similar object recognition tasks, I had experimented with the presence and amount of different layers within the CNN until I received an accuracy I was happy with in the amount of time given for the project.

Choosing how to pre-process the live video feed for model recognition was a difficult task, since there were so many environmental factors that could differ my new test data (the live video frame) from the limited data that the model was trained on, since the processing effects were not completely consistent with the classic MNIST dataset, and there were augmentations made to increase the dataset that were not specified (such as contrast/brightness values, inversion of pixels, etc). After considering and experimenting with various combinations of filtering, we came to the preprocessing method described in section 3.3. We evaluate our model's performance using the keras framework's "evaluate()" method.



```
225/225 [==============================] - 3s 14ms/step - loss: 0.3229 - accuracy: 0.921
```

Figure 6. The console output of the keras evaluate() method. Displays how long evaluation took, as well as the loss and prediction accuracy of the model on the given validation data.

The model performs well on the with a 92% accuracy on the validation set. With this accuracy we can have confidence when testing the model on data that closely resembles the dataset. Unfortunately, live video presents some difficulties for our model as well, such as motion blur. Due to dataset quality(Features the wrong sign for "T") and constraints (such as not including multiple frames when attempting to recognize signs, does not include "J" and "Z") the total amount of "detectable" signs is 23.

Of these 23 possibly recognizable signs, 20 of them can be repeatedly recognized and translated. Some do work better than others, however. Of the 20 signs that can be repeatedly recognized, 11 of them are recognized easily without much movement or reposing required, the signs in this category are as follows: A, B, C, G, H, , L, M, O, R, W, and Y. 6 of the repeatedly recognized signs require specific conditions for accurate translation, the signs in this category are as follows: D, F, Q, S, V, X. 3 of the repeatedly recognized signs require very specific conditions for accurate translation, including close/far distance from camera, location in frame, and pose, the signs in this category are as follows: I, K, P. The signs that are not recognized in live video (but are recognized within the test data) are as follows: E, N, U. Each of the signs that were not recognized in real-time have a common factor: they share a very similar pose to a recognizable sign. "E" is confused for "A", "N" is confused for "M", and "U" is confused for "R". As a reminder, J, T, and Z were not represented in the dataset for consideration.

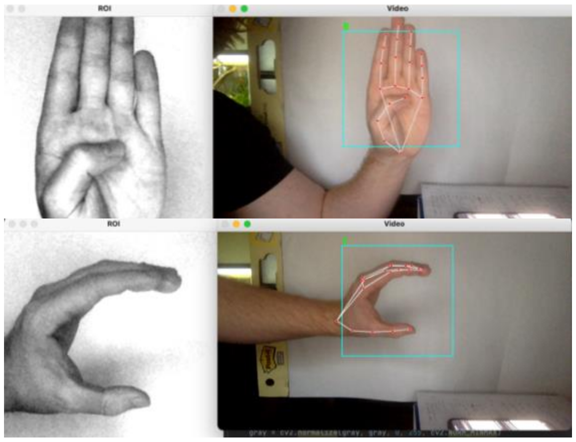


Figure 7. Most signs are easily recognized and translated, such as B and C.

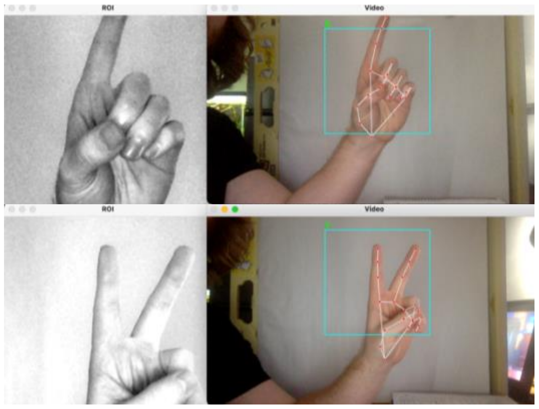


Figure 8. Some signs require specific poses that, if deviated from, alter the model's ability to recognize and translate the sign.

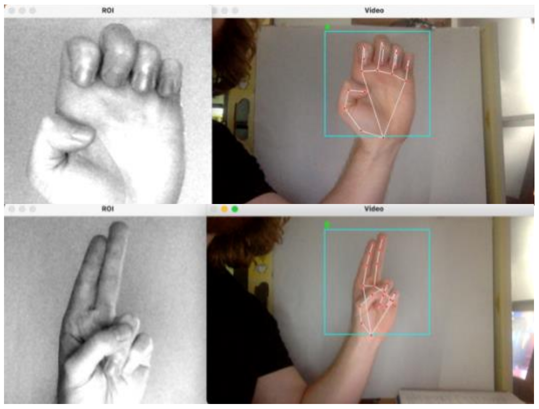


Figure 9. Some signs are not able to be recognized regardless of pose or location in frame, these signs all have a similar sign in the dataset they are mistaken for, such as E being mistaken for A, or U being mistaken for R.

This data translates to 86.96% of dataset signs being recognizable in real time by our model, and 13.04% not being recognizable at all. This also translates to 47.83% of dataset signs being recognized easily, 26.10% of dataset signs being recognized in certain poses, 13.04% of dataset signs being recognized in very specific situations, and 13.04% not being recognized at all.

V. Discussion and Summary

Considering the timeframe the project was completed in and the lack of quality, extensive data available for training, we were pleasantly surprised by the ability of the model to accurately and consistently recognize and translate the sign in the live video for 47.38% of available signs and recognize and translate the majority of the other signs, although less consistently, for a total of 86.96% recognition across all signs in the set. While we strive for as close to 100% accuracy as possible, these results show that we are able to recognize signs in real-time using this model, and given more time and data, would be able to progress and increase accuracy even further.

Going forward, we intend to continue experimenting with the model, and develop a second model that recognizes signs based on the hand landmarks presented by the handTracker module. We could also extend the project by allowing for the region of interest to follow the user's hands using the coordinates returned by the handTracker module. We hope our simple CNN and program can help inspire people to continue sign language recognition and translation research.

References/Bibliography

[1] Y. Chen, F. Wei, X. Sun, Z. Wu, S. Lin, "A Simple Multi-Modality Transfer Learning Baseline for Sign Language Translation," Tsinguhua University, 2022, Microsoft Research Asia, Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition(CVPR) pp.5120-5130.

[2] H. Hu, W. Zhao, W. Zhou, Y. Wang, H. Li, SignBERT: Pre-Training of Hand—Model-Aware Representation for Sign Language Recognition, Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV). CAS Key Laboratory of GIPAS, EEIS Department, University of Science and Technology of China (USTC), Institute of Artificial Intelligence, Hefei Comprehensive National Science Center, 2021, pp.11087-11096.

[3] N. C. Comgoz, O. Koller, S. Hadfield, R. Bowden, "Sign Language Transformers: Joint End-to-End Sign Language Recogntion and Translation," CVSSP, University of Surrey, Guildford, UK, Microsoft, Munich, Germany, 2020.

[4] N. C. Camgoz, S.Hadfield, O. Koller and R. Bowden, "SubUNets: End-to-End Hand Shape and Continuous Sign Language Recognition," 2017 IEEE International Conference on Computer Vision (ICCV), Venice, Italy, 2017, pp. 3075-3084

[5] O. Koller, N. C Camgoz, H. Ney and R. Bowden, "Weakly Supervised Learning with Multi-Stream CNN-LSTM-HMMs to Discover Sequential Parallelism in Sign Language Videos," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 42, no. 9, pp. 2306-2320, 1 Sept. 2020, doi: 10.1109/TPAMI.2019.2911077

[6] Huang, J., Zhou, W., Zhang, Q., Li, H., & Li, W. (2018). Video-Based Sign Language Recognition Without Temporal Segmentation. Proceedings of the AAAI Conference on Artificial Intelligence, 32(1).