Live Sign Language Translation

Brendan Baker Justin Ceresa Klaas van Kempen
Matt Moriarty

2023-12-04

Table of contents

1 Introduction/Background

Automated translation of sign language is a complex problem that has generated extensive research effort in the field of deep learning (DL). While early efforts to decipher sign language algorithmically stemmed from classical temporal modeling techniques such as hidden markov models, deep learning models have enabled researchers to encode better and more flexible representations of sign language (Camgoz, Hadfield, Koller & Bowen, 2017). Despite these improvements, difficulties in accomplishing machine translation of sign languages stem from a broad variety of considerations, such as the input modality, feature extraction method, and model architecture.

There are over 137 different documented sign languages in the world (Fenlon & Wilkinson, 2015; Ethnologue), many of which are not sufficiently documented and codified to build useful DL models. For sign languages with enough source data for modeling, challenging decisions must be made regarding the input modality and feature extraction method. Researchers have utilized both static and dynamic input modalities such as grayscale, RGB, and infrared images, video, and scene flow (Rastgoo, Kiani, & Escalera, 2021). Since many sign languages communicate with posture and facial expressions in addition to hand signals, three main types of features have been explored: (1) hands only, (2) hands and face, and (3) full body.

The present paper focuses on the detection of letters from the american sign language (ASL) alphabet, which can be seen below in Figure 1. Using these signs is referred to as "finger-spelling," or dactylology, and consists of about 8.7% of signs in casual ASL (Morford & MacFarlane, 2003). Deep learning models have previously been used to detect and classify ASL characters, including LeNet (LeCun, Bottou, Bengio, & Haffner, 1998; Bilgin & Mutludoğan, 2019). Additionally, some researchers have combined convolutional neural networks (CNNs) with Long Short-Term Memory (LSTM) models in a pipeline to accommodate both the gesture representation and the temporal component of word formation (Elboushaki, Hannane, Afdel, & Koutti, 2020).

This project has several primary goals. First, we hope to utilize object detection models such as You Only Look Once (YOLO) to detect and isolate the image of the hand sign. Then, we plan to utilize CNNs to classify the signed characters, exploring different model architectures. Finally, we hope to package this model and deploy it to a web application in which users can test the model themselves by signing letters into their webcam.

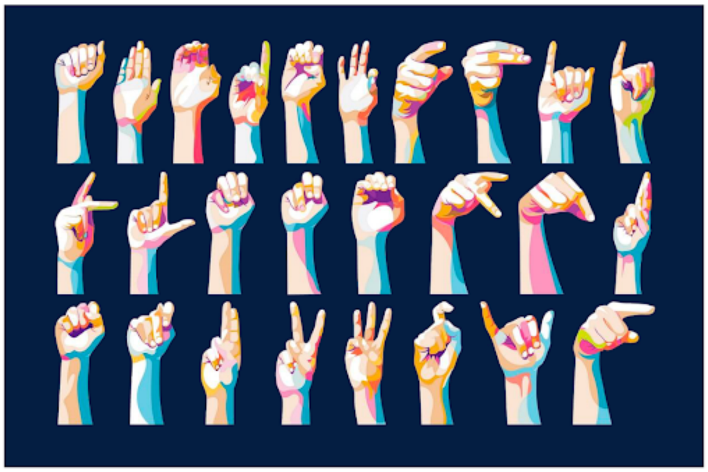

Figure 1: Hands depicting sign language signs for each letter of the alphabet.

2 Methods

2.1 Image Classification

The data for this project is collected from kaggle, and consists of 34,627 28x28 grayscale images. The images were originally collected as rgb and converted to grayscale. Each image consists of a single hand, which is signing a particular letter in the English language. Please see Figure 2 below for some examples. Each letter/sign is represented between 1,400 and 1,500 times in the dataset, with the exception of J and Z which require motion to sign and are therefore not represented in the dataset. Some of the images are synthetic and have been pre-augmented, which includes random pixelation, brightness adjustments, and small rotations.

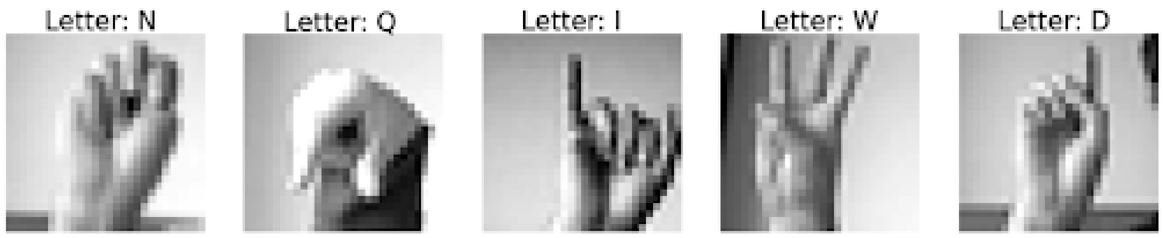

Figure 2: Five examples of the images in the training dataset and the letters that they represent.

The image data is already well prepared for modeling, but a few additional steps can help achieve better results. While augmentation has already been applied, since the goal of this project is to deploy an application with prediction for live images, we anticipate that the images being used in the application will have a lot of variation and therefore more image augmentation of the training images could be useful. We applied random rotation up to 10 degrees, random horizontal and vertical shifts, and random zooming. Other steps to preprocess the data include normalizing the pixel values to [0-1] and splitting the training data into training and validation data. This new set of training images plus augmented images will be the input for the model.

2.2 Model Training

Convolutional Neural Networks (CNN's) are the gold standard in image prediction. Here, a CNN is implemented with an input layer that takes in 28 x 28 images, 2 convolutional layers, 2 max pooling layers, and a dense feed forward layer, and finally an output layer with 24 classes. Since this is a multiclass classification problem, sparse categorical cross entropy is the loss function for the model. Using a hyperparameter search the optimal model based on validation loss uses the adam optimizer with a learning rate of 0.001 and a batch size of 16. Implementing dropout did not improve the model. The model trained for 20 epochs to identify the best model (lowest validation loss). Figure 3 below shows both the training and validation losses experienced throughout our training session.

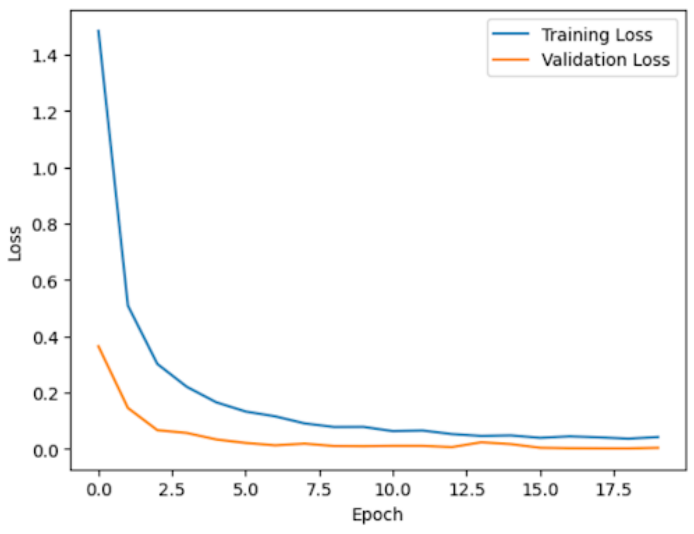

Figure 3: Training (blue) and validation (orange) loss plotted over each epoch during the training process.

2.3 YOLO Object Detection

With our CNN model prepared for translating pre-processed images in our testing set, we turn to live image processing for the purpose of live sign language translation. For this, we need to construct a pipeline to take us from a live image to a predicted translation. For this, we turn to a pre-trained object detection model, known as the YOLO model.

The YOLO object detection model is capable of detecting objects within an image, assigning a bounding box to each one. This technique is helpful to us, as we aim to detect a hand within a live image in order to isolate it from the rest of the features of the image, such as background objects and noise. By sending a live image through the YOLO model, we obtain a detected bounding box and use it to crop the image to that detected region.

2.4 Live Image Processing

Once we isolate a detected hand within a live image, there are still more processing steps we must take in order to prepare the image for our image classification CNN model. For this, we must convert the image to grayscale and reduce the image down to the appropriate size.

Converting the image to grayscale is simple: we just need to average out the color channels of the image. By averaging the red, green, and blue, color channels within the image, we obtain a single color channel that captures our desired grayscale color palette.

Reducing the image down to the appropriate size (28 x 28) is a bit more complex: we must average out “patches” of pixels in order to reduce the size of the image without losing any information contained within the image itself. For instance, if our image has size 280 x 140, we average out patches of size 10 x 5 in order to obtain 28 “patches” in both width and height, giving us our reduced image. More generally, we construct patches with width equal to the total width of the image divided by 28, and height equal to the total height of the image divided by 28. Finally, by averaging out the pixel intensities within each patch, we obtain our reduced 28 x 28 image. At this point, we can send our processed live images into our CNN the same way in which we send the pre-processed images from our testing dataset.

Figure 4 below visualizes our pipeline for live image sign language translation.

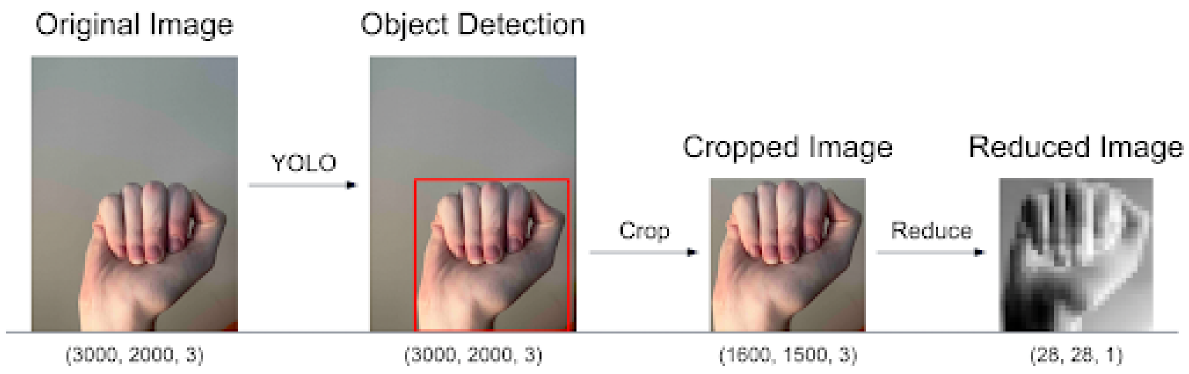

Figure 4: Live image translation pipeline, from object detection (YOLO) to image reduction.

2.5 Streamlit Application

With our live image translation pipeline complete, we look to employ our work through an accessible application using Streamlit. Here, our live images come directly from the user’s camera or webcam, should they choose to allow access to the application. The idea for our app is for the user to sign a letter, click a button to translate, and have that letter appear at the top of the screen. The internal python script works so that as each letter is signed and translated, they are appended to the top, allowing for the gradual spelling out of what someone is trying to communicate. We’ve also set up an additional tab on the app to demonstrate our model’s performance on the test data. The user selects a letter, the test data is then limited to photos of signs of that letter, and the predicted and actual letters are displayed. The live application can be visited here.

3 Results/Discussion

In the first section of these results we will focus on the evaluation of the training and test set of the image classification model. After that we will focus on the results of deploying this model and its performance on live images.

Table 1 below shows the test and training results of the optimal CNN model. The model has 99.9% accuracy over the training set and 99.7% accuracy over the test set, indicating that the model is able to predict sign language images with high accuracy and is not overfitting to the training data.

Table 1: The model training and test accuracy and loss in the image classification problem.

	Accuracy	Loss
Training	99.9%	0.0011
Testing	99.7%	0.0126

The confusion matrix on the test set, which can be seen in Figure 5 below, continues to show the strong performance of the model, predicting every class correctly with the exception of a few mistakes of predicting the letter “i” for true “y” images.

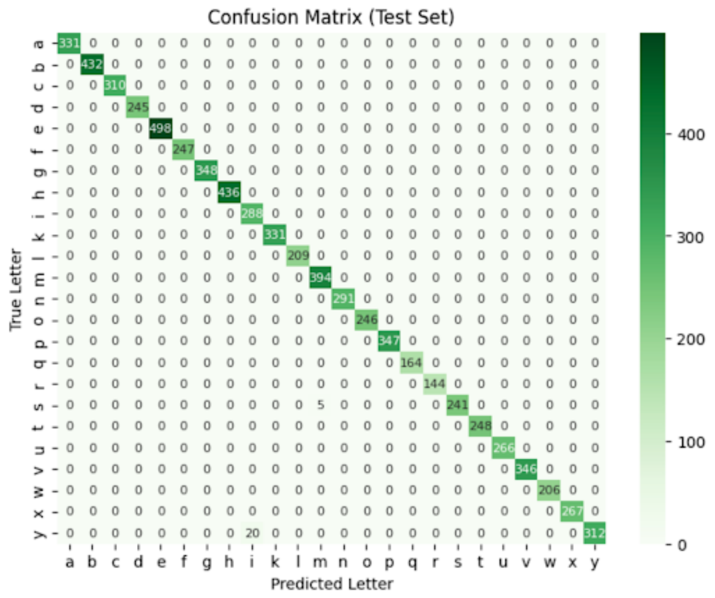

Figure 5: Confusion matrix of the true vs predicted values for the model over the test set.

After deploying our model in a streamlit application, we discovered that our CNN model did not perform as well on live images as it did on the testing set. The YOLO model was successful in detecting the hand, and we were able to separate the hand from the rest of the scene. However, the CNN model was often inaccurate when classifying signs from the user-input webcam images. In fact, through a test of using one live image per alphabet character, our application was successful in identifying only half of the 24 characters. In many cases, the incorrectly predicted letter was similar in shape to the actual letter. Reasons for the discrepancy in accuracy could include different lighting conditions and backgrounds between test images and live images.

4 Conclusions

Translation of sign languages with deep learning models is a complex task, and success is heavily dependent on the context of the images being translated. As we discovered in our project, well-annotated images that have similar backgrounds and lighting conditions can be used to train a model that translates signs very well on other images of similar quality. Our dataset that was used to train our CNN model performed exceptionally on testing images from the same dataset, achieving 99.7% accuracy. Few mistakes were made in the thousands of test images, and the majority of mistakes were between two similar characters (“i” and “y”).

Live images are more challenging to classify accurately. Our streamlit application utilized a pretrained YOLO model to separate the hand from a user-input webcam image, then processed the images to be consistent with those in the training data (grayscale, 28 x 28). Despite utilizing significant data augmentation in our training phase, the CNN model still had difficulty accurately predicting the signed letter in live images. Creating models that perform well on images from different cameras, backgrounds, and lighting conditions is a common challenge for computer vision applications. In future studies, we would suggest implementing further processing in our live image application. One possibility would include masking the background after detecting the hand and replacing it with a consistent white background before running the image through the CNN.

Our exploration into the translation of sign languages utilizing deep learning models has demonstrated the potential and challenges of this technology. While promising applications such as translation, interpreting services, and human-computer interaction are within reach, innovative methods to address the inherent variability in real-world images must be developed. We can be proud of the step we took in this direction, as we have developed an application that is not only useful, but accessible to those who may need it.

5 References

Bilgin, M., & Mutludoğan, K. (2019, October). American sign language character recognition with capsule networks. In 2019 3rd International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT) (pp. 1-6). IEEE.

Cihan Camgoz, N., Hadfield, S., Koller, O., & Bowden, R. (2017). Subunets: End-to-end hand shape and continuous sign language recognition. In Proceedings of the IEEE international conference on computer vision (pp. 3056-3065).

Elboushaki, A., Hannane, R., Afdel, K., & Koutti, L. (2020). MultiD-CNN: A multi-dimensional feature learning approach based on deep convolutional networks for gesture recognition in RGB-D image sequences. Expert Systems with Applications, 139, 112829.

Fenlon, J., & Wilkinson, E. (2015). Sign languages in the world. Sociolinguistics and Deaf communities, 1, 5.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.

Morford, J. P., & MacFarlane, J. (2003). Frequency characteristics of American sign language. Sign Language Studies, 213-225.

Rastgoo, R., Kiani, K., & Escalera, S. (2021). Sign language recognition: A deep survey. Expert Systems with Applications, 164, 113794.