American sign language alphabet recognition using deep learning

Geer Ma, Kendall Reed, Mike Ebrahimi, Reihaneh Derafshi, Lin Guo

University of Calgary

Abstract

Over 5% of the world's population have moderate to severe hearing loss, and much of this population in North America learns American Sign Language (ASL) to communicate. The learning and usage of sign language are not only useful for those who are 'hard of hearing' or 'deaf', but also for their family members and friends.

Unfortunately, sign language interpretation is not always available - and some technologies that classify sign language use expensive equipment like gloves with embedded sensors. A deep learning model that can classify ASL through images taken from a camera lens is a convenient solution to this problem. This report explores the use of both pre-trained and untrained deep learning models such as ResNet, VGG-16, GoogleNet, and a custom CNN model for an ASL fingerspelling dataset.

Our findings suggest that we received a testing accuracy of 96.7%, 95.6%, and 94.6% for ResNet, VGG-16, and GoogleNet respectively. Additionally, our custom CNN model resulted in 90.8% testing accuracy.

GitHub Link:
https://github.com/geerma/asl-recognition-deep-learning

***Index Terms*—** American Sign Language (ASL), Deep Learning, Convolution Neural Network (CNN), Fully Connected Neural Network (FCNN)

1. Introduction

The World Health Organization [1] states that over 430 million individuals have disabled hearing loss, meaning they require rehabilitation for their hearing. They estimate that the numbers are expected to grow to 700 million in 2050. The National Association of the Deaf mentions that American Sign Language (ASL) is a visual language that uses the shape, placement, and movement of the hands to convey information [2]. This language is used by those who are 'hard of hearing' or 'deaf'. Sign language is not a universal language and it can change and grow with time, but ASL in particular is predominantly used in the USA and Canada.

A study by M. Saleem from Sheridan College mentions that there are currently technologies available to classify ASL such as ones that use glove-based techniques, which facilitate human-computer interaction via embedded sensors on the glove [3]. However, the drawback is that they are dependent on a product that may not always be readily available, and there are high costs associated with obtaining the technology, which limits it from being an effective solution for individuals with or without hearing impairment.

One instrument available to almost everyone in the modern world is the camera. With that in mind, a deep-learning model that can receive visual input, and then output a classification of ASL would greatly improve the quality of life for individuals who have disabled hearing loss. This would also benefit those who do not have hearing loss but want to better connect with them - such as family members and family, as well as businesses.

The study from M. Saleem mentions that retail locations and walk-in environments struggle with barriers associated with accessibility, and thus end up losing potential customers [3]. A business that is readily able to communicate with hard-of-hearing / deaf individuals can foster long-term business relationships.

Alfonso Castillo, a journalist from Newsday, mentions in an article that sign-language interpreters are scarce at debates and news conferences, which is another area that could be addressed with a readily-accessible classification application that only requires a camera [4]. All-in-all, image classification for ASL is an invaluable research topic that can benefit everyone involved.

For this final project, we will be using the American Sign Language Alphabet dataset from Kaggle [5]. The goal of this project is to classify real-world images of signing as a letter in the alphabet. Note that in general conversations, people will typically sign "words" - for example, they would sign "hello" instead of five letters. However, "fingerspelling" is the act of spelling words, and it is useful when first learning the basics of ASL [6]. It is also used when the person does not know the word, or the sign for the word does not even exist.

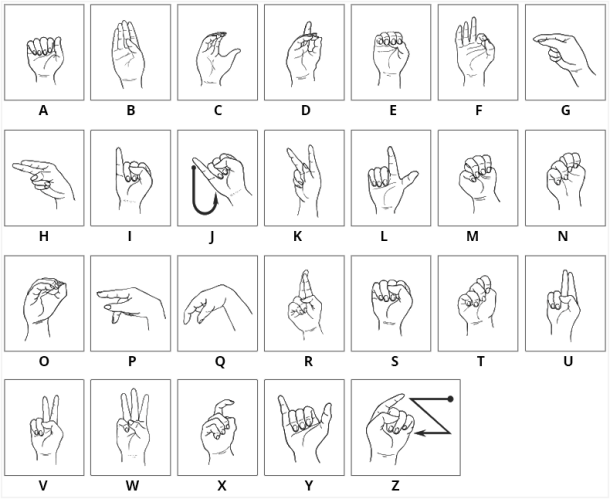

Fig. 1. American Sign Language Alphabet [7]

Creating/inventing a new sign is not usually done because it may violate the grammatical rules of ASL, or it may be unintentionally offensive. Therefore, knowing the alphabet remains general and all-purpose. Thus, a deep learning program that can classify the ASL alphabet is still relevant and can have practical uses. The completion of this project will lay the foundation for an improved version of the deep learning model which can support the classification of non-alphabetical ASL "words".

2. Related Works

Recognizing sign language gestures using machine learning algorithms is an active area of research that has seen significant progress in recent years. Traditional methods of automatic sign language recognition have relied on basic camera technology to generate datasets of simple images with no depth or contour information available, just the pixels present [8]. However, recent advancements in depth-sensing technology and the development of custom-designed color gloves [9] have allowed researchers to explore new approaches for sign language recognition that go beyond analyzing pixel data, incorporating depth and motion profiles for each gesture [3]. This has resulted in a significant improvement in the recognition accuracy of sign language gestures, particularly for dynamic sign recognition [8, 11].

Several studies have investigated the use of Convolutional Neural Networks (CNNs) to handle the task of classifying images of ASL letter gestures, using pre-trained models such as GoogLeNet [3, 8,11].

Bheda and colleagues presented a method for classifying images of the letters and digits in American Sign Language (ASL) using deep convolutional networks [11]. Their approach achieved recognition rates of 67% and 82.5% for the alphabet and digits of ASL, respectively [11].

Agarwal and Thakur proposed a sign language recognition system utilizing depth images captured by a Microsoft Kinect® camera [10]. They generated a feature matrix using computer vision algorithms that capture the characteristic depth and motion profile of each sign language gesture and trained a multi-class SVM classifier to recognize gestures for the digits 0-9. Results were compared with existing techniques, showing promising potential for the use of depth-sensing technology in sign language recognition systems.

Real-time sign language translation is crucial for facilitating communication between the deaf community and the general public. Garcia and Viesca presented their work on an American Sign Language (ASL) fingerspelling translator that utilizes a pre-trained GoogLeNet architecture and transfer learning on the ILSVRC2012 dataset, as well as the Surrey University and Massey University ASL datasets [8]. Their model produced robust results in consistently classifying letters a-e correctly with first-time users and a majority of cases for letters a-k. Although further research and more data are needed, their work presents a promising step toward a fully generalizable translator for all ASL letters.

In another study, Suk et al. proposed a new method for recognizing hand gestures in a continuous video stream using a dynamic Bayesian network (DBN) model [12]. They developed a gesture model for one- or two-hand gestures, which is used to define a cyclic gesture network for modeling a continuous gesture stream. The proposed DBN-based hand gesture model and the design of a gesture network model achieved a recognition rate of upwards of 99.59% with cross-validation and 84% with a precision of 80.77% for the spotted gestures in the case of recognizing a continuous stream of gestures. Although the proposed method is a bit more complicated and requires the analysis of hand shapes, it is believed to have strong potential for successful applications to other related problems such as sign language recognition.

3. Materials and Methods

The dataset used in this project comes from the American Sign Language Alphabet dataset from Kaggle [5]. It contains 87, 000 images, each with a size of 200x200 pixels. These images are divided equally into 29 classes, 26 classes consisting of the letters A-Z, and 3 additional classes of SPACE, DELETE, and NOTHING. Each image has 3 channels, red, green, and blue. This dataset was compiled using personal images that were captured with the intent of creating a dataset.

In order to utilize this dataset to create a machine learning model that could accurately predict a sign being made by a hand gesture captured by a camera, the images were split into three sets. The training set consists of 70% of the dataset, the validation set 20%, and the test set 10%. This split allowed for 60900 images to be used in testing

and training, 17400 images to be used to validate the training, and 8700 images to test the models final performance on unseen data. The images also had to be resized to perform correctly with the pre-trained models.

Convolutional neural networks (CNNs) were utilized to process the images and create an algorithm to differentiate the images from one another. Additionally, PyTorch [13] was used as a framework to facilitate the neural network, and ResNet50 [14], VGG-16 [15], and GoogleNet [16] were used as the pre-trained models. An additional untrained CNN model was used as well.

A CNN has many strengths and weaknesses compared to fully connected neural networks (FCNNs) or other traditional machine learning techniques. One of the largest strengths of neural networks in general is their ability to identify important aspects and features of the dataset without human intervention. Another strength of CNNs is their ability to accurately recognize and classify images, as shown in the related works section of this report. Compared to a FCNN, CNNs require less computational power, especially when supplemented with a pretrained model such as ResNet50 or GoogleNet.

Although a CNN did appear to be the right choice of training algorithm for this project, it has some weaknesses. One weakness of the CNN is that it requires a large amount of training data to be effective. By using a large percentage of our dataset as the testing set, in addition to using pretrained models, this weakness was minimized.

Table. 1 summarizes the characteristics of pretrained models we used in this project. Note that the number of trainable parameters are rounded to only show the magnitude order.

Table. 1. Characteristics of pretrained models

model	# Conv / inception layer	# pooling layer	# FC layer	Trainable params
ResNet50	50	2	1	23M
GoogleNet	22	5	1	6M
VGG16	13	5	3	138M

Input image size is the same for all these three pretrained models and is equal to 224 x 224. We had to resize our images in the database to conform with the requirements of these models. When creating our custom CNN, we used this same input size to reduce the preprocessing steps required.

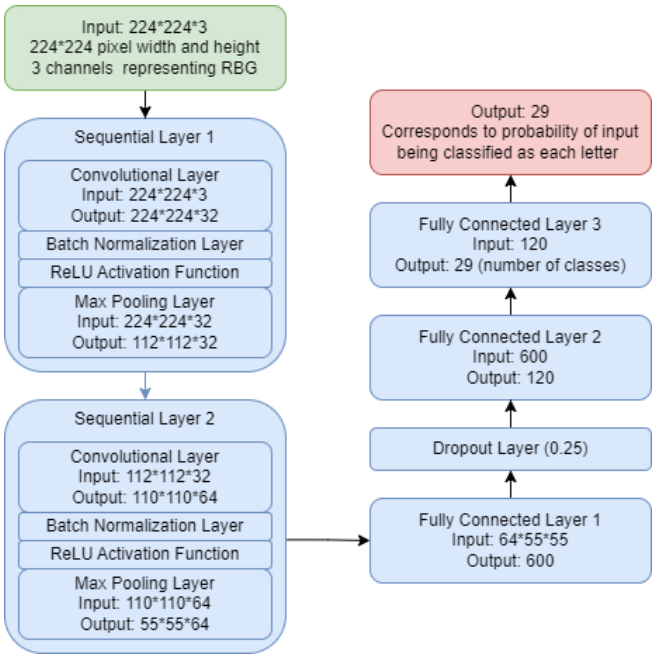

Fig. 2. Architecture of our custom CNN

The custom CNN was adapted from a previous assignment, and it is loosely based on a VGG-16 architecture of neural network layers.

4. Results

After preprocessing the data, we initialized the different models as discussed in the previous section. The first three models are pre-trained models loaded from PyTorch, while the last model is a custom CNN model.

4.1. ResNet50

The ResNet50 model containing 50 layers performed quite well. The training accuracy increased from 80% to 95%, and the validation accuracy increased from 92% to 96.5%.

Using the model on the test dataset yielded an accuracy of 96.7%.

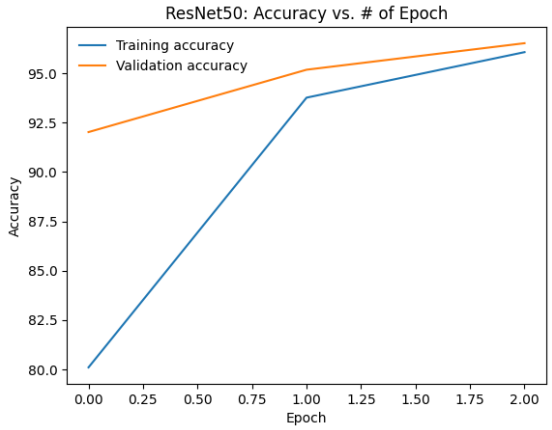

Fig. 3. Training and Validation Accuracy for ResNet50

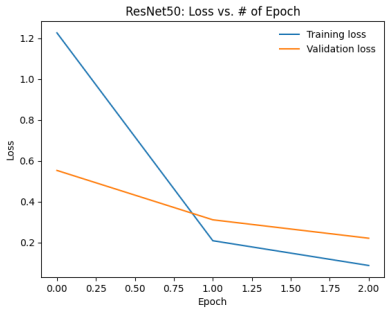

Fig. 4. Training and Validation Loss for ResNet50

4.2. VGG-16

The VGG16 model containing 16 layers performed quite well. The training accuracy increased from 73.4% to 83.8%, and the validation accuracy increased from 91.7% to 95.1%.

Using the model on the test dataset yielded an accuracy of 95.6%.

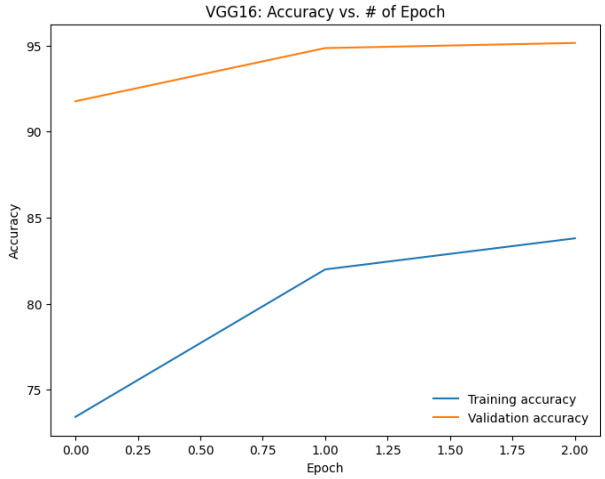

Fig. 5. Training and Validation Accuracy for VGG16

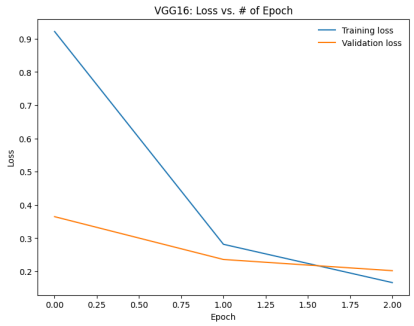

Fig. 6. Training and Validation Loss for VGG16

4.3. GoogleNet

The GoogleNet model containing 16 layers performed quite well. The training accuracy increased from 76% to 91.3%, and the validation accuracy increased from 90.8% to 94.9%.

Using the model on the test dataset yielded an accuracy of 94.6%.

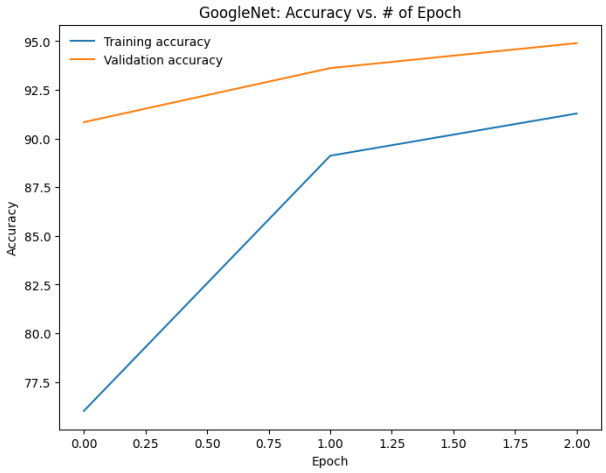

Fig. 7. Training and Validation Accuracy for GoogleNet

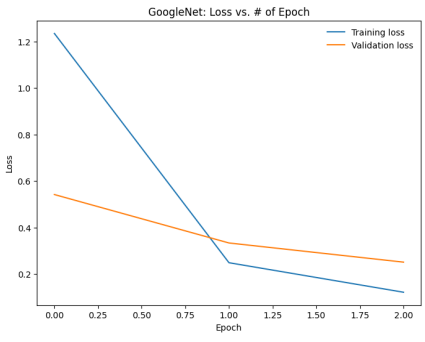

Fig. 8. Training and Validation Loss for GoogleNet

4.4. Custom CNN

The GoogleNet model containing 16 layers performed quite well. The training accuracy increased from 54% to 90%, and the validation accuracy increased from 82.5% to 91.5%.

Using the model on the test dataset yielded an accuracy of 90.8%.

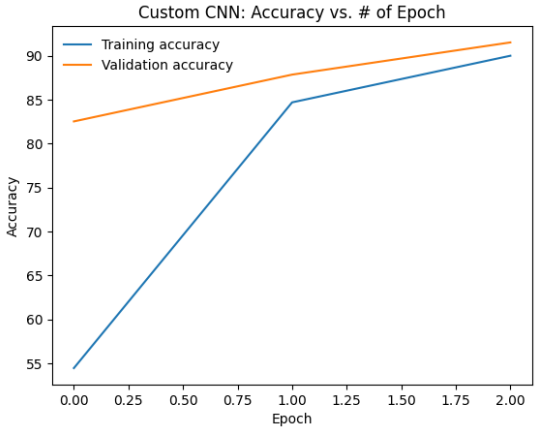

Fig. 9. Training and Validation Accuracy for Custom CNN

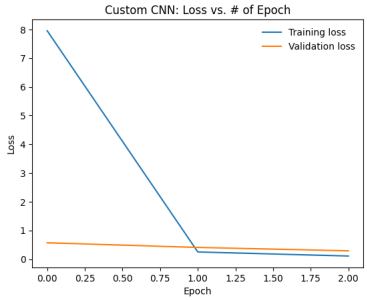

Fig. 10. Training and Validation Loss for Custom CNN

5. Discussion

After testing each model on a test dataset, we can compile the test accuracy and training time into a table.

Table. 2. Summary of Model Performance

Model	Test Accuracy	Training Time (s)
ResNet50	96.7%	1407
VGG16	95.6%	1878
GoogleNet	94.6%	960
CustomCNN	90.8%	1150

We notice that ResNet achieved the highest test accuracy while having a moderate training time. VGG16 had moderate test accuracy while having a long training time. Although GoogleNet had the lowest test accuracy, it also had a substantially low training time. Test accuracy would be the most important metric because it is paramount to classify ASL correctly - but training time could still be an important criterion to consider for training deep learning models.

The following figures show the misclassified images for ResNet50 and Custom CNN. The misclassifications for all four models can be found in the Jupyter Notebook.

Some common misclassifications we can find in these figures are that classes 20, 21, and 22 are frequently confused with each other. They correspond to the letters U, V, and W. The common denominator for those three letters is that they look similar to each other - they involve having 2 or 3 straight up in the air. Therefore, it is reasonable that the ResNet50 would confuse those letters together.

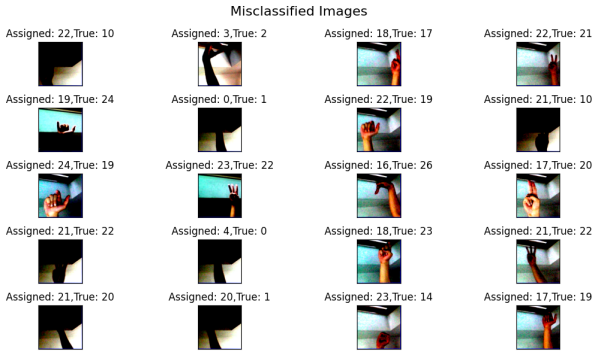

Fig. 11. Misclassified Images for ResNet50

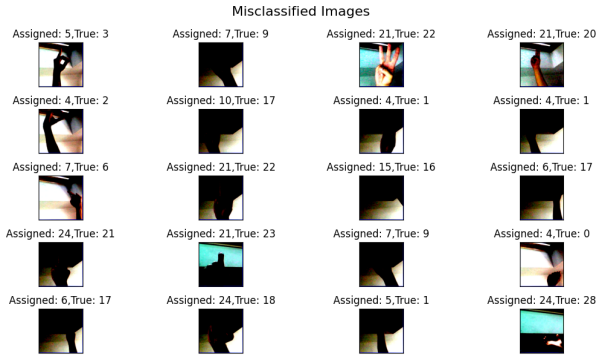

Fig. 12. Misclassified Images for Custom CNN

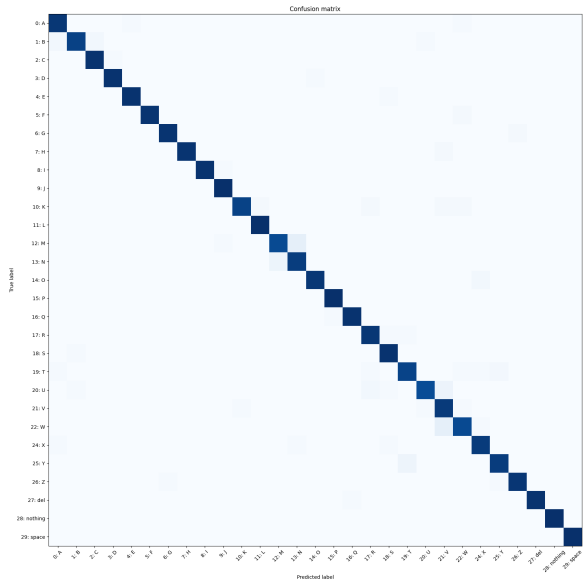

Fig. 13. Confusion Matrix for ResNet50

In the confusion matrix for ResNet 50, we notice the confusion of letters U, V and W which we mentioned when we mentioned in the previous section with the misclassification images. We also notice M and N being misclassified.

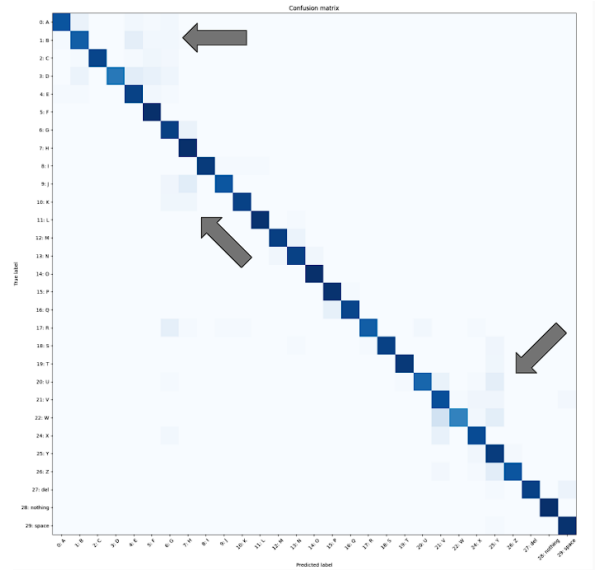

Fig. 14. Confusion Matrix for Custom CNN

The confusion matrix for our Custom CNN has the same common misclassifications as ResNet, but there are additional locations in the confusion matrix that are noteworthy to point out. The misclassification rate of U, V, and W further increased as well. The letters surrounding E and H also were frequently misclassified, as well as V.

6. Conclusions

This report investigated the usage of deep learning techniques to classify the American Sign Language alphabet through images. We tried different pre-trained models such as ResNet50, VGG16, and Google; as well as a custom CNN model trained from scratch. We discovered a testing accuracy of 96.7%, 95.6%, and 94.6% for the pre-trained models, and 90.8% for our custom CNN model.

The high accuracy score percentage for the pre-trained models demonstrates that deep learning is a valid solution for classifying the ASL alphabet from images. Given further experimentation, we would be able to create a solution that can correctly classify ASL words from a live camera feed with a high level of confidence.

From our experimentation, we can likely say that the best model for this topic is ResNet50 due to having the highest test accuracy.

7. References

[1] “Deafness and hearing loss,” World Health Organization. [Online]. Available: https://www.who.int/news-room/fact-sheets/detail/deafness-and-hearing-loss

[2] “What is American sign language?,” National Association of the Deaf. [Online]. Available: https://www.nad.org/resources/american-sign-language/what-is-american-sign-language/

[3] M. Saleem, “Deep Learning Application On American Sign Language Database For Video-Based Gesture Recognition,” Sheridan College. [Online]. Available: https://source.sheridancollege.ca/cgi/viewcontent.cgi?article=1006&context=fast_sw_mobile_computing_theses

[4] A. Castillo. “Deaf frustrated by scarcity of sign language interpreters at news briefings,” Newsday. [Online]. Available: https://www.newsday.com/news/health/coronavirus/interpreters-for-deaf-d92075

[5]“ASL Alphabet,” ASL Alphabet | Kaggle. /datasets/grassknoted/asl-alphabet. https://www.kaggle.com/datasets/grassknoted/asl-alphabet

[6]“National Association of the Deaf - NAD,” National Association of the Deaf - NAD. https://www.nad.org/resources/american-sign-language/learning-american-sign-language/

[7] “American Sign Language ABC and number stories,” APSEA. [Online]. Available: https://apsea.ca/families-students/home-learning/deaf-or-hard-hearing/self-determination-and-advocacy/asl-abc-and

[8] B. Garcia and S. Viesca, "Real-time American Sign Language Recognition with Convolutional Neural Networks," in Convolutional Neural Networks for Visual Recognition at Stanford University, 2016.

[9] Cao Dong, M. C. Leu, and Z. Yin, “American Sign Language alphabet recognition using Microsoft Kinect,” 2015 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), Jun. 2015, doi: 10.1109/cvprw.2015.7301347.

[10] A. Agarwal and M. K. Thakur, “Sign language recognition using Microsoft Kinect,” 2013 Sixth International Conference on Contemporary Computing (IC3), Aug. 2013, doi: 10.1109/ic3.2013.6612186.

[11] V. Bheda and D. Radpour, "Using deep convolutional networks for gesture recognition in American Sign Language," arXiv preprint arXiv:1710.06836, 2017.

[12] H.-I. Suk, B.-K. Sin, and S.-W. Lee, “Hand gesture recognition based on dynamic Bayesian network framework,” Pattern Recognition, vol. 43, no. 9, pp. 3059–3072, Sep. 2010, doi: 10.1016/j.patcog.2010.03.016.

[13] “PyTorch,” PyTorch. https://www.pytorch.org

[14] K. Team, “Keras documentation: ResNet and ResNetV2,” ResNet and ResNetV2. https://keras.io/api/applications/resnet/#resnet50-function

[15] K. Team, “Keras documentation: VGG16 and VGG19,” VGG16 and VGG19. https://keras.io/api/applications/vgg/

[16] C. Szegedy, et al., “Going deeper with convolutions,” 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015.