

American Sign Language (ASL) Fingerspelling Detection

Shoaib Mohammed

01 June 2021

Abstract

Approximately 70 million people around the world are deaf-mute. While translation services have become easily accessible for about 100 languages, sign language is still an area that has not been explored. Our goal is to detect & translate the letters of ASL in real-time.

Keywords — ASL, fingerspelling, Convolutional Neural Network (CNN), transfer learning, computer vision

1 Introduction

1.1 Background

According to the Communication Service for the Deaf (CSD) [1], there are 360 million deaf people worldwide. Another report by the World Health Organization (WHO) [2] bumps up the number to 466 million people or over 6% of the world's population suffering from disabling hearing loss. But even in this age of technology and communication, we are yet to see a universal translation system that helps bridge the gap between people that can and cannot speak. ASL is a natural language meaning it was not created and was spread by the people who employ the signs by the movement of hands, facial expressions, and body posture. The goal of this project is to detect and accurately translate the letters in ASL.

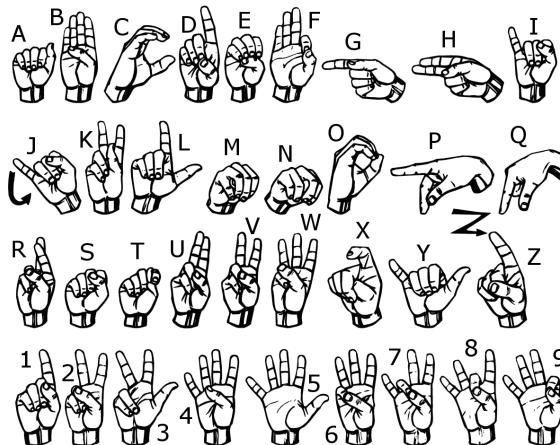


Figure 1: Alphabets in ASL

1.2 Geographical Distribution

The true count for the number of sign languages is still unknown given the vast majority, however, *Ethnologue* [3] lists this number to be 137 [4]. Given the vast majority, ASL is still the most popular sign language and is being widely used around the globe. In addition to being the primary source of communication for a sign language in the United States, ASL is being used throughout most of the provinces in Canada [5].



Figure 2: ASL being used around the world

Variations of ASL are also being used worldwide. Sign language similar to ASL is being used throughout Africa in places such as Nigeria, Ghana, Guyana, Central African Republic, Jamaica, Zimbabwe, and Kenya [6].

1.3 Scope & Limitations

The bottom line is that there is no universal sign language. For instance, the British Sign Language (BSL) differs by a great margin from ASL, this is clear when comparing Figure 1 with Figure 3. Generally speaking, a person in the US can understand spoken English in the UK but this is not the case with sign language. Even though there is a multitude of sign languages being used across the world, if an accurate model was developed to recognize a sign language, in our case, ASL, the same methodology could be applied to recognize other sign languages.

One big limitation for classification is that most signs require motion and are not static. Even more so as there are signs which are formed by the same motion but the repetition or the number of times a motion is repeated differs. Things get even more challenging as

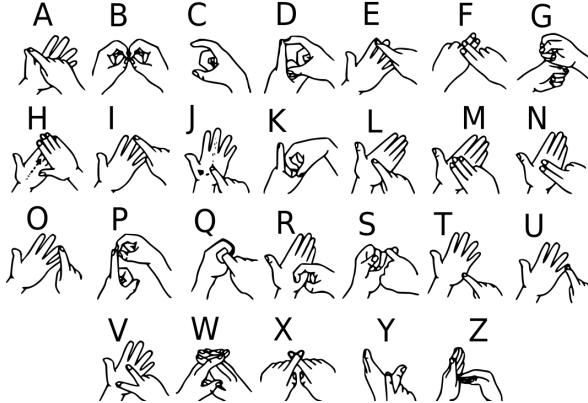


Figure 3: Alphabets in BSL

facial expressions are important in sign languages which is akin to the vocal tone of a person's voice when speaking. Two signs may be exactly the same visually but the face gesture of the signer makes them different.

Perhaps the major limitation is classifying each sign from the sheer corpus of signs in ASL. A dictionary on ASL contains illustrations for more than 1,600 signs [7]. However, our focus is classifying the alphabets alone which limits our range to **24** alphabets since we exclude the alphabets J & Z because these require motion.

2 Literature Survey

2.1 Population Statistics

A combined study in [8] estimates there were more than 250,000 deaf people and as many as 500,000 people who used ASL in 1972. Over the years, this number has been on the rise. Based on their research for in year 2006, fewer than 1 in 20 Americans or 10,000,000 people suffered from *hard of hearing* and close to 1,000,000 were classified as *functionally deaf*.

In 2016, the National Institute on Deafness and Other Communication Disorders (NIDCD) [9] released statistics about hearing loss in the United States. About 2 to 3 out of every 1,000 children have hearing loss. Approximately 15% of adults (37.5 million) aged 18 and over report trouble hearing and 18% of adults aged 20-69 have speech-frequency hearing loss in both ears. Furthermore, about 2% of adults aged 45-54, 8.5% of adults aged 55-64, 25% of adults aged 65-74, and 5% of adults aged 75 and

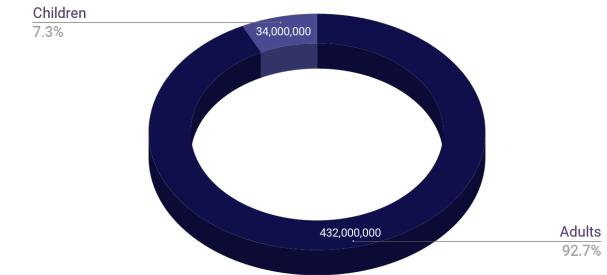


Figure 4: Distribution of the population

older have disabling hearing loss.

In terms of the distribution of people suffering from hearing loss [2], 432 million people are adults and 34 million are children. Future projections estimate 630 million people by 2030 and over 900 million people by 2050.

2.2 Existing Models

There has been ongoing research in this area with the growth in machine learning and datasets being made available publicly. Many of the models use Kinect to recognize the hand gestures and map them to ASL.

Earlier versions of work used a sensory glove. In 2005, Cemil and Ming [10] worked on recognizing the alphabets in ASL by using a sensory Cyberglove and a motion tracker to extract the gestures. By processing the data of the fingers on the strain gauges, the trajectory, and orientation from the motion tracker through an Artificial Neural Network (ANN), high accuracy results were achieved. Building upon this in 2011, Cemil and Ming [11] used feature extraction with noise reduction and were able to classify 50 ASL words. A similar approach was used in 2014 [12] where the output of the sensor glove gets sent to a microcontroller through an Analog-to-Digital Converter (ADC). The glove uses flex sensors which capture the bending of each finger. The predicted letter gets displayed on an LCD. Although this worked well, it is impractical in real-life. The whole system requires several high components to be connected each time someone needs to sign a letter.

Due to the sophisticated components required by using sensor gloves, there's been lots of research on translating



Figure 5: Cyberglove and Flock of Birds by Ascension

ASL by using depth cameras. In 2015, we had a major breakthrough successfully translating the 24 static ASL alphabet signs using a Microsoft Kinect [13]. A latex color glove allowed for easier hand segmentation and localizing the joint positions. The prediction was done through a Random Forest (RF) classifier on the depth data. The Kinect has also been used in conjunction with hidden Markov models [14] for recognition. Even with good results, the Kinect is not something people just carry around. Furthermore, depth cameras are also known to perform badly under poor lighting conditions.

Color glove	Raw color image	Segment ground truth	Raw depth image

Figure 6: A color glove allowed easier hand segmentation

In 2015, a hand gesture recognition model was developed using depth and intensity channels with 3D CNN [15]. A similar methodology was used to recognize sign language by extracting spatio-temporal features [16]. To improve the performance, multi-channels of

video streams including color, depth, and body joint information are used as input to the 3D CNN. In 2017, a features extractor with deep behavior was used to deal with the Arabic Sign Language [17]. By combining the features extractor with a 3D CNN, the recognition system was fed with data from depth maps and could recognize 25 gestures.

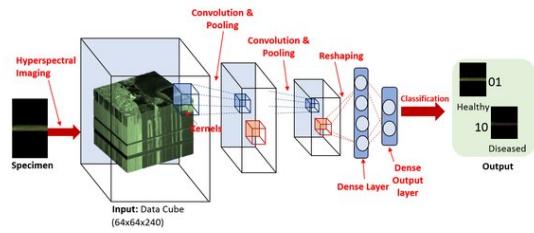


Figure 7: 3D Convolutional Neural Network

In 2018 a different solution was proposed, SignFi [18] which uses Wi-Fi to recognize sign language. SignFi can recognize 276 gestures involving the head, arm, hand, and finger gestures with high accuracy. SignFi works based on Channel State Information (CSI) which describes how a signal propagates from the transmitter to the receiver at a certain carrier frequency. SignFi uses Wi-Fi packets as input and a 9-layer CNN as the classification algorithm.

Realizing the challenges faced by the above mentioned methods, our goal is to create a machine learning model with transfer learning to classify the alphabets of ASL in real-time. By using *four* transfer learning models, namely, MobileNetV2[19], Xception[20], InceptionV3[21], and InceptionResNetV2[22], we achieve a high accuracy.

3 Problem Description

3.1 Data Gathering

Looking for available datasets, the *ASL Finger Spelling Dataset* [23] is perhaps the most popular one. There were others too, but they did pose a few problems. There were many outliers present in the dataset including the faces of the subjects. This causes difficulties while training the model since our focus is the hand. The other datasets which focus on the hand contain less outliers but do not contain a lot of variation causing the model to overfit. Hence, we decided to create our own dataset fixing the aforementioned problems by focusing on the

hand and adding variation by using more subjects in different locations.

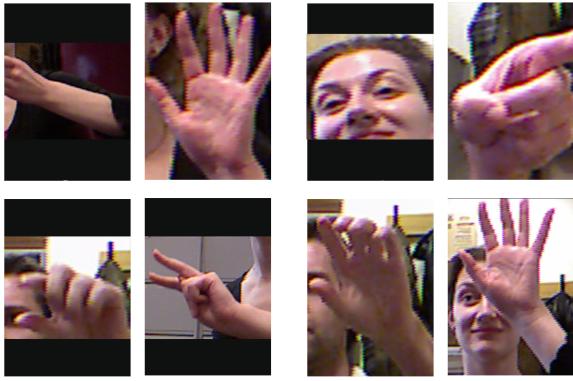


Figure 8: Outliers present in ASL Fingerspelling dataset [23]

Using `python`, a helper script was created to generate data to train the model and has the following features:

- Subjects: 5 different people with varying skin colors and features
- Shape: $[250 \times 250 \times 3]$ having reasonable parameters for the input image
- Variability: taken with both good & bad lighting
- Size: 4,000 images for each alphabet for a total 96,000 images



Figure 9: ASL alphabets from our dataset

3.2 Dependencies

We have used the following dependencies:

- Python 3
- Numpy
- Keras
- Kaggle API
- TensorFlow

All the codebase is written in the `python` programming language due to its wide support for machine learning algorithms with many different packages.

3.3 Data Preparation

The first step is to define the labels which in our case is a set of categorical labels. A helper script was created to accomplish just this task alone, given a string representing a set of labels it creates a `mapping.json` file with each label representing a numeric value. Due to the amount of data being processed for the model, it takes time to load and pre-process all of it. We can speed things up by using a common technique called *serialization*. Serialization is the process of converting an object into a stream of bytes and saving that in the binary format in memory. When required, we can load this stream of bytes back as our object, this process is called *deserialization*. Serialization allows us to have faster training as preprocessing the dataset on each run can take time. In `python`, serialization is commonly referred to as *pickling*.

The preprocessing phase consists of 3 parts:

1. Load the data – We load the `xs` which is a list of images each representing the pixel values in RGB format and the `ys` which is a list of labels each representing the alphabet. We can load the images using OpenCV which is a popular computer vision library.
2. Preprocess the data – The input image size may not match the size required as input for the model. We resize all the images to the shape `[SHAPE × SHAPE × CHANNELS]` where the constants are collected from a configuration file, it is also a good idea to one-hot encode the labels. Lastly, it is recommended to shuffle the data since in the case of the data being organized together, it may hinder the learning process of the model during training.

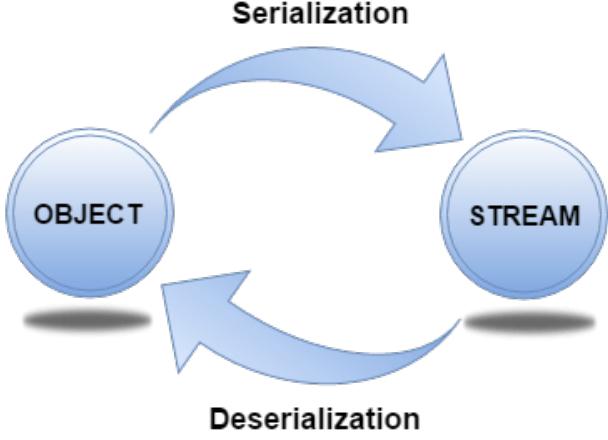


Figure 10: Process of serialization & deserialization

3. Split the data – We need to divide the data into train and test sets. There is a common split called the *80/20 split* which is derived from the Pareto principle. This means 80% of the data or 76,800 images are used for the training set and the rest 20% of the data or 19,200 images are used for the testing set.

3.4 Configuration

There are certain parameters that can be tweaked by the user to fit to his needs:

- Epochs: number of passes through the training set, *default = 64*.
- Shape: the shape of the square image, *default = 250*.
- Channels: number of color channels, *default = 3*.
- Batch Size: number of training samples used in one iteration, *default = 128*.

3.5 Transfer Learning

We have used transfer learning to get a foundation for the model configuration. Transfer learning makes use of knowledge gained while solving one problem and applying it to a different but related problem. It is a popular approach in deep learning as it provides a strong

foundation and a starting point given the vast amount of time and resources required to solve the related problem from scratch. In our case, we will use transfer learning models that have been trained to recognize patterns in images. As we are focusing on fingerspelling ASL, we do not require motion (since we exclude J & Z making it a related problem of finding patterns).

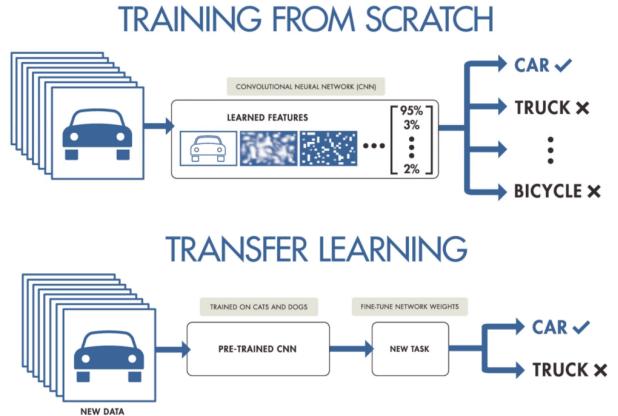


Figure 11: Training from scratch vs. transfer learning

The python framework `keras` has many available models to choose, we have made available the following:

- MobileNetV2: lightweight and fast model that runs on a smartphone [19]
- Xception: deep CNN with depth-wise separable convolutions [20]
- InceptionV3: refined from GoogLeNet with batch normalization and factorization [21]
- InceptionResNetV2: variation of InceptionV3 by adding residual connections and a simplified architecture [22]

These were chosen carefully due to their ability to perform well on images and computer vision related tasks.

We pass the image as input to our transfer learning which is well-trained to figure out *patterns*. The last layer in each transfer learning model is used to get the model's prediction of the category/label. We will be using the 2nd last layer which contains the activations of the transfer learning model.

Getting the activations of the transfer learning model takes time, hence, we store the model's activations on the entire dataset by using the technique of serialization

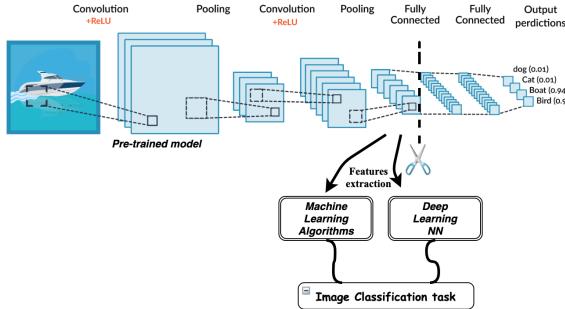


Figure 12: Transfer learning process

as mentioned previously. These can be huge and consume memory, if space is an issue, then it's better to compromise with longer training times.

3.6 Model Architecture

Keras allows two ways to create models:

- *Sequential*: allows to create a layer-by-layer model where the inputs flow straight down to the output.
- *Functional*: connect a given layer to any other layer resulting in complex networks thus providing flexibility.

We use the **Sequential** model since we do not need to manually define and connect each layer of our model.

Next, we need to define our layers. Each layer in our network tries to build onto the work extracted from the previous layer. Layers extract useful information from the data fed into them. The goal is to have meaningful representations for our problem. Take a problem of recognizing parts of a human face from a selfie. The first layer may extract the person from the background. The second layer may extract parts of the human body from the person such as the arms, legs, face and so on. The third layer may extract parts of the face itself like the eyes, nose, mouth, hair and ears. The fourth layer may extract more in-depth features such as the wrinkles, dark circles, dimples and more. The activation function is used to decide which features are relevant. Essentially, by deciding if a neuron fires (activates) or not, we evaluate the importance of a feature in affecting the overall output.

Working on large images can be a problem. For example, take a high-resolution color image of width 1080 px and height 720 px. Since a color image contains 3 channels (RGB), the total number of parameters is

$1080 \times 720 \times 3 = 2,332,800$ parameters just for the input image alone! As we keep adding more layers, the number of parameters goes on increasing. It is recommended not to exceed 10 million parameters for the entire model as this can have computation issues as well as other known issues like over-fitting.

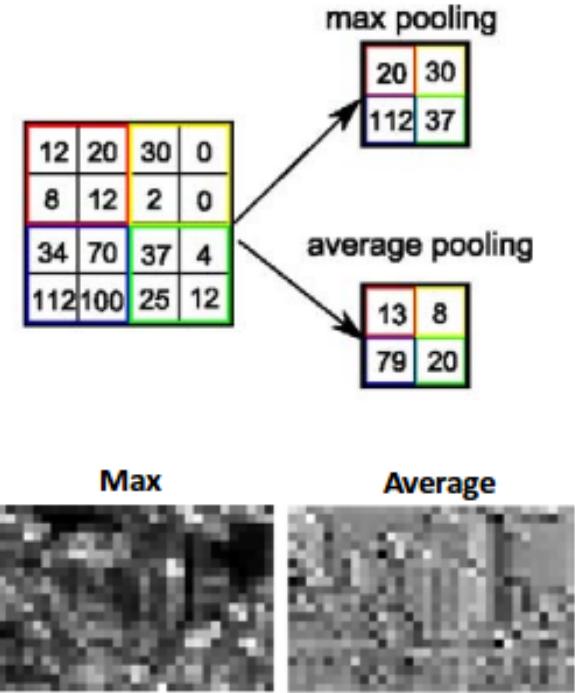


Figure 13: Max pooling vs. Average pooling

A resolve to decrease the number of parameters is a technique called pooling. Pooling layers allow us to downsample our feature map by summarizing the features. For image data, max pooling tends to work better because there could be some extreme and important features (like edges) which get smoothed out in average pooling. These two common pooling methods are described below:

- Average Pooling: replaces each patch by its average value.
- Max Pooling: replaces each patch by its max value.

Before passing the output to the final layer which is a fully-connected layer, it is a good idea to flatten this output. The flatten layer unrolls the input to a single

long one-dimensional array. Most of the work has been done at this point by all the previous layers in our model allowing us to figure out the patterns that are present. The work left now is to figure out the label, in our case, the ASL alphabet. We use fully-connected layers to help link the processed features together and predict the label.

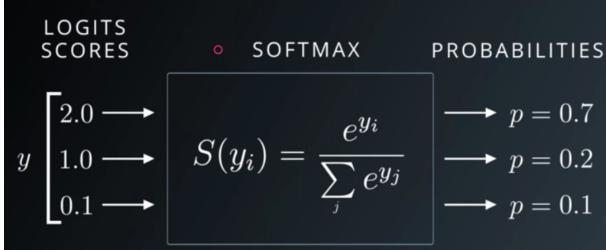


Figure 14: Softmax function pushing high scores close to 1 and low scores close to 0

The final layer in our model is also a fully-connected layer but it is special. The number of units is the number of classes, in our case, 26. The activation function used is softmax activation, it gives a probability distribution wherein a value is pushed close to 1 if it is large or close to 0 if it is small.

3.7 Train Phase

The training phase consists of four steps:

1. Input: get Region Of Interest (ROI) of shape $[250 \times 250 \times 3]$ from image
2. Transfer learning: get the model activations from last layer
3. Train model: recognize ASL alphabet using previous activations
4. Repeat: process batches by resuming the training in each iteration

4 Experimental Results

4.1 Predict Phase

The predict phase is exactly the same as the train phase except for the last step:

- Predict: display the alphabet with the highest score

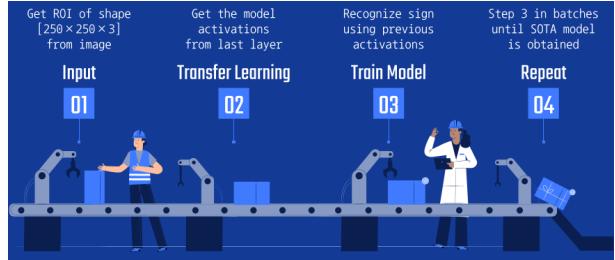


Figure 15: Process of training the model

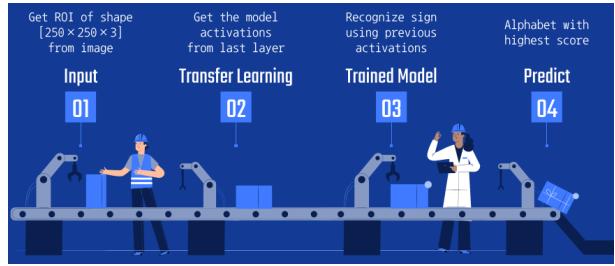


Figure 16: Process of testing the model

4.2 Model Statistics

When we see the performance for the trained model, we see InceptionV3 outperforms them all. This again can be explained by its very efficient architecture which is unlike its predecessors. Peculiarly, InceptionResNetV2 which had the best overall performance on the ImageNet dataset, has the worst performance on our dataset. The reason for this is unclear, perhaps an in-depth analysis of the weights can give a valid explanation. Due to the small range of labels to classify, i.e., 24, MobileNetV2 performs very well as its light-weight architecture can easily handle the small corpus of signs. Xception also works well but its better to use MobileNetV2 due to its computational advantage.

Model	Size	Accuracy	Parameters
MobileNetV2	91 MB	0.918	7,934,872
InceptionV3	99 MB	0.924	8,598,424
Xception	100 MB	0.908	8,721,304
InceptionResNetV2	93 MB	0.848	8,074,136

Table 1: Trained model statistics using varying transfer learning models

The overall performance of the model on the test dataset is **92.36%** which works well. The performance can further be improved by using the idea of model ensembling which uses the results of the multiple models together to get a combined prediction. This often works better than using an individual model since each model has its own best-side or data that it can recognize well leading to good performance.

4.3 Model Performance

The model was evaluated and tested on all 24 alphabets of ASL. From the results shown in Figure 17 it is clear that the model performs well. On further testing, it was found that specific signs such as the letter *F* are easily recognizable on the InceptionV3 or the letter *U* on the Xception model. This could just be due to the difference in the architecture. MobileNetV2 works well overall and is fast when compared to the other models. All the models work in real-time and work even better with good lighting and clear backgrounds.

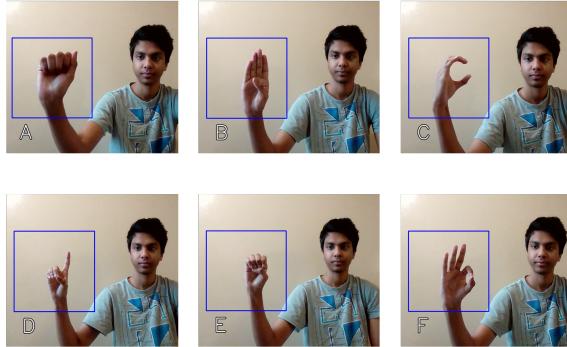


Figure 17: Prediction of the alphabets A, B, C, D, E, and F

But things are not always perfect as shown in Figure 18, some of the letters in ASL are similar causing discrepancies and confusion when given as input to the model. The distribution is shown below:

- M, N, T
- D, R, U
- G, H
- A, E, S

- C, O, X

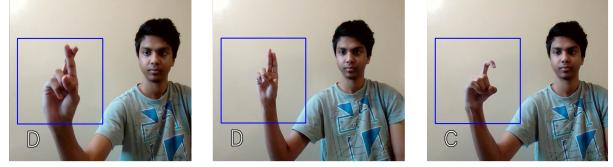


Figure 18: Incorrect prediction of R as D, U as D, and X as C

Most of the time, the incorrect results fall in the similar categories due to the confusion caused by high similarities. Looking closely at the distribution of M, N, T, we see all of them just differ in the position of the thumb in between the other fingers. Again, the performance can be improved by adding more variability into the dataset and improving the training model architecture.

References

- [1] C. Soukup, “Communication service for the deaf,” 1975. <https://www.csd.org/>, Last accessed on 2021-05-31.
- [2] R. E. C. David W Dowdy, “World health organization. office of library and health literature services,” 1988. <https://www.who.int/>, Last accessed on 2021-05-31.
- [3] S. International, “Ethnologue languages of the world,” 2000. <https://www.ethnologue.com/>, Last accessed on 2021-05-31.
- [4] J. Fenlon and E. Wilkinson, “Sign languages in the world,” *Sociolinguistics and Deaf communities*, pp. 5–28, 2015.
- [5] K. Al-Fityani and C. Padden, “Sign language geography in the arab world,” *Sign languages: A Cambridge survey*, vol. 20, 2010.
- [6] V. Nyst, “Sign languages in west africa,” *Sign languages*, pp. 405–432, 2010.
- [7] R. A. Tennant, M. Gluszak, and M. G. Brown, *The American sign language handshape dictionary*. Gallaudet University Press, 1998.
- [8] R. E. Mitchell, T. A. Young, B. Bachelder, and M. A. Karchmer, “How many people use asl in the united states? why estimates need updating,” *Sign Language Studies*, vol. 6, no. 3, pp. 306–335, 2006.

- [9] N. I. on Deafness and O. C. Disorders, “Quick statistics about hearing,” 1988. <https://www.nidcd.nih.gov/health/statistics/quick-statistics-hearing>, Last accessed on 2021-05-31.
- [10] C. Oz and M. C. Leu, “Recognition of finger spelling of american sign language with artificial neural network using position/orientation sensors and data glove,” in *International Symposium on Neural Networks*, pp. 157–164, Springer, 2005.
- [11] C. Oz and M. C. Leu, “American sign language word recognition with a sensory glove using artificial neural networks,” *Engineering Applications of Artificial Intelligence*, vol. 24, no. 7, pp. 1204–1213, 2011.
- [12] K. Patil, G. Pendharkar, G. Gaikwad, and S. Phule, “American sign language detection,” *International Journal of Scientific and Research Publications*, vol. 4, no. 11, 2014.
- [13] C. Dong, M. C. Leu, and Z. Yin, “American sign language alphabet recognition using microsoft kinect,” in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pp. 44–52, 2015.
- [14] S. Lang, M. Block, and R. Rojas, “Sign language recognition using kinect,” in *International Conference on Artificial Intelligence and Soft Computing*, pp. 394–402, Springer, 2012.
- [15] P. Molchanov, S. Gupta, K. Kim, and J. Kautz, “Hand gesture recognition with 3d convolutional neural networks,” in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pp. 1–7, 2015.
- [16] J. Huang, W. Zhou, H. Li, and W. Li, “Sign language recognition using 3d convolutional neural networks,” in *2015 IEEE international conference on multimedia and expo (ICME)*, pp. 1–6, IEEE, 2015.
- [17] M. ElBadawy, A. Elons, H. A. Shedeed, and M. Tolba, “Arabic sign language recognition with 3d convolutional neural networks,” in *2017 Eighth International Conference on Intelligent Computing and Information Systems (ICICIS)*, pp. 66–71, IEEE, 2017.
- [18] Y. Ma, G. Zhou, S. Wang, H. Zhao, and W. Jung, “Signfi: Sign language recognition using wifi,” *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 2, no. 1, pp. 1–21, 2018.
- [19] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “Mobilenetv2: Inverted residuals and linear bottlenecks,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510–4520, 2018.
- [20] F. Chollet, “Xception: Deep learning with depthwise separable convolutions,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1251–1258, 2017.
- [21] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the inception architecture for computer vision,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826, 2016.
- [22] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, “Inception-v4, inception-resnet and the impact of residual connections on learning,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31, 2017.
- [23] N. Pugeault and R. Bowden, “Spelling it out: Real-time asl fingerspelling recognition,” in *2011 IEEE International conference on computer vision workshops (ICCV workshops)*, pp. 1114–1119, IEEE, 2011.

List of Figures

1	Alphabets in ASL	1
2	ASL being used around the world	1
3	Alphabets in BSL	2
4	Distribution of the population	2
5	Cyberglove and Flock of Birds by Ascension	3
6	A color glove allowed easier hand segmentation	3
7	3D Convolutional Neural Network	3
8	Outliers present in ASL Fingerspelling dataset [23]	4
9	ASL alphabets from our dataset	4
10	Process of serialization & deserialization	5
11	Training from scratch vs. transfer learning	5
12	Transfer learning process	6
13	Max pooling vs. Average pooling	6
14	Softmax function pushing high scores close to 1 and low scores close to 0	7
15	Process of training the model	7
16	Process of testing the model	7
17	Prediction of the alphabets A, B, C, D, E, and F	8

18	Incorrect prediction of R as D, U as D, and X as C	8
19	ASL alphabets from our dataset	11
20	Process of training the model	11
21	Process of testing the model	11

List of Tables

1	Trained model statistics using varying transfer learning models	7
2	Model architecture and number of parameters in each layer	12
3	Shape of each layer in trained model for different transfer learning models	13

Additional Resources

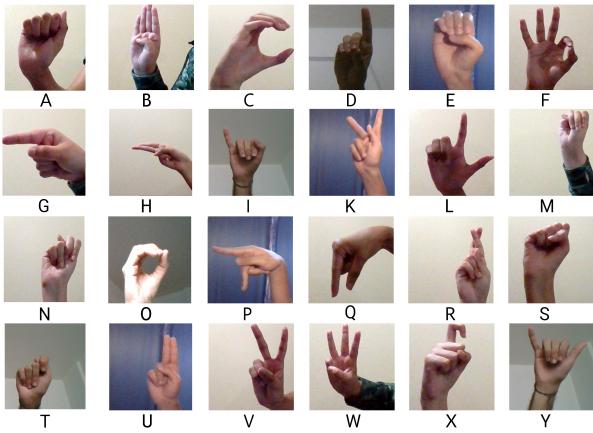


Figure 19: ASL alphabets from our dataset

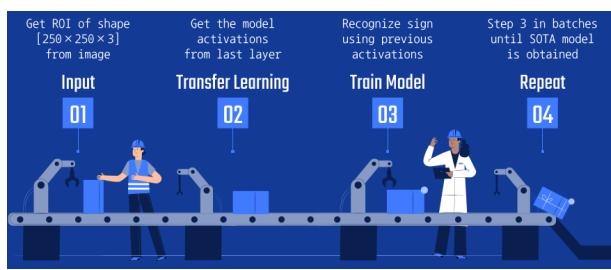


Figure 20: Process of training the model



Figure 21: Process of testing the model

ID	Layer (Type)	Number of Parameters
1	dense_1 (Dense)	<i>dependent</i>
2	dense_2 (Dense)	524,800
3	dense_3 (Dense)	131,328
4	dense_4 (Dense)	32,896
5	up_sampling2d_1 (UpSampling2D)	0
6	conv2d_5 (Conv2D)	131,136
7	depthwise_conv2d_1 (DepthwiseConv2D)	1,088
8	up_sampling2d_2 (UpSampling2D)	0
9	depthwise_conv2d_2 (DepthwiseConv2D)	1,088
10	conv2d_6 (Conv2D)	65,600
11	dense_5 (Dense)	8,320
12	dense_6 (Dense)	33,024
13	dense_7 (Dense)	131,584
14	dense_8 (Dense)	525,312
15	dense_9 (Dense)	2,099,200
16	dense_10 (Dense)	2,098,176
17	dense_11 (Dense)	524,800
18	dense_12 (Dense)	131,328
19	dense_13 (Dense)	32,896
20	max_pooling2d_1 (MaxPooling2D)	0
21	flatten_1 (Flatten)	0
22	dense_14 (Dense)	<i>dependent</i>

Table 2: Model architecture and number of parameters in each layer

ID	Layer (Type)	Number of Parameters
1	(None, 8, 8, 1024)	(None, 6, 6, 1024)
2	(None, 8, 8, 512)	(None, 6, 6, 512)
3	(None, 8, 8, 256)	(None, 6, 6, 256)
4	(None, 8, 8, 128)	(None, 6, 6, 128)
5	(None, 16, 16, 128)	(None, 12, 12, 128)
6	(None, 13, 13, 64)	(None, 9, 9, 64)
7	(None, 10, 10, 64)	(None, 6, 6, 64)
8	(None, 20, 20, 64)	(None, 12, 12, 64)
9	(None, 17, 17, 64)	(None, 9, 9, 64)
10	(None, 14, 14, 64)	(None, 6, 6, 64)
11	(None, 14, 14, 128)	(None, 6, 6, 128)
12	(None, 14, 14, 256)	(None, 6, 6, 256)
13	(None, 14, 14, 512)	(None, 6, 6, 512)
14	(None, 14, 14, 1024)	(None, 6, 6, 1024)
15	(None, 14, 14, 2048)	(None, 6, 6, 2048)
16	(None, 14, 14, 1024)	(None, 6, 6, 1024)
17	(None, 14, 14, 512)	(None, 6, 6, 512)
18	(None, 14, 14, 256)	(None, 6, 6, 256)
19	(None, 14, 14, 128)	(None, 6, 6, 128)
20	(None, 7, 7, 128)	(None, 3, 3, 128)
21	(None, 6272)	(None, 1152)
22	(None, 24)	

Table 3: Shape of each layer in trained model for different transfer learning models