Sign Language Recognition from Video using Geometrical and Transfer Learning Techniques

David Chen[1,*,‡] and Thomas Li[2,*,‡]

[1]*Harvard-Westlake School, 3700 Coldwater Canyon Ave, Los Angeles, CA 91604, USA*
[2]*Weston High School, Weston, MA 02493, USA*
[*]*Corresponding authors: dchen1561@gmail.com (DC), dbbested@gmail.com (TCL)*
[‡]*Equal contribution: These authors contributed equally to this work*

September 14, 2023

Abstract

We aim to develop an American Sign Language (ASL) recognition system to bridge the communica tion barrier for the deaf and hard-of-hearing communities. Some previous projects utilized specialized hardware, while this study focuses on purely 2-D video stream recognition due to its accessibility. In this paper, we use the fine-tuning method, which involves fine-tuning a neural network model trained on public datasets for specific individuals in a data-efficient manner. Challenges such as image background interference and occlusion are discussed. The algorithms are tested on teenager, adult and senior male and female hands and the accuracies are comparatively better than other previous models, with the results average testing accuracy being 96.696%.

1 INTRODUCTION

The deaf and hard-of-hearing population has faced communication barriers throughout history, leading to significant challenges in their day-to-day lives. American Sign Language (ASL), a visual language used by millions of people worldwide, has become an essential means of communication for this population. However, the majority of the general population remains unable to understand ASL, exacerbating the communication divide. In recent years, machine learning and artificial intelligence have demonstrated immense potential in addressing such challenges by creating systems capable of understanding and interpreting sign language.

The primary aim of this research project is to develop and test new ASL recognition systems, capable of identifying and interpreting sign language gestures in real time. By developing a reliable ASL recognition system, this project aims to break down communication barriers for the deaf and hard-of-hearing commu nities, while promoting accessibility and social integration. In the long term, the successful deployment of sign language recognition systems in various applications such as education, healthcare, and public services, will contribute to creating a more inclusive and supportive society for all. While some projects have used a variety of hardware systems that normal people would not use, like special gloves, multiple cameras, etc. We wanted to test recognition systems with a single purely 2-D video stream, with a variety of

background interference, lighting and hand conditions.

After reviewing the field, we have decided to employ two methods. The first is the 3-D Hand Geome try(HG) method, which uses an open source library Mediapipe to identify the 3-dimensional joint positions (landmarks) of the hand. We then created a non machine learning python code which computes the ge ometric relations between the landmarks, and sorts the video stream into different gestures. The second method is the Transfer Learning(TL) method, which utilizes a pre-trained neural network based on stock photo data. The model is then fine tuned based on live captured video footage of a new individual user's

hand, by optimizing a small portion of the neural network weights while fixing the rest. The aim of this project is to compare the two methods for the purpose of detecting sign language that is effective at working with all types of hands and environments.

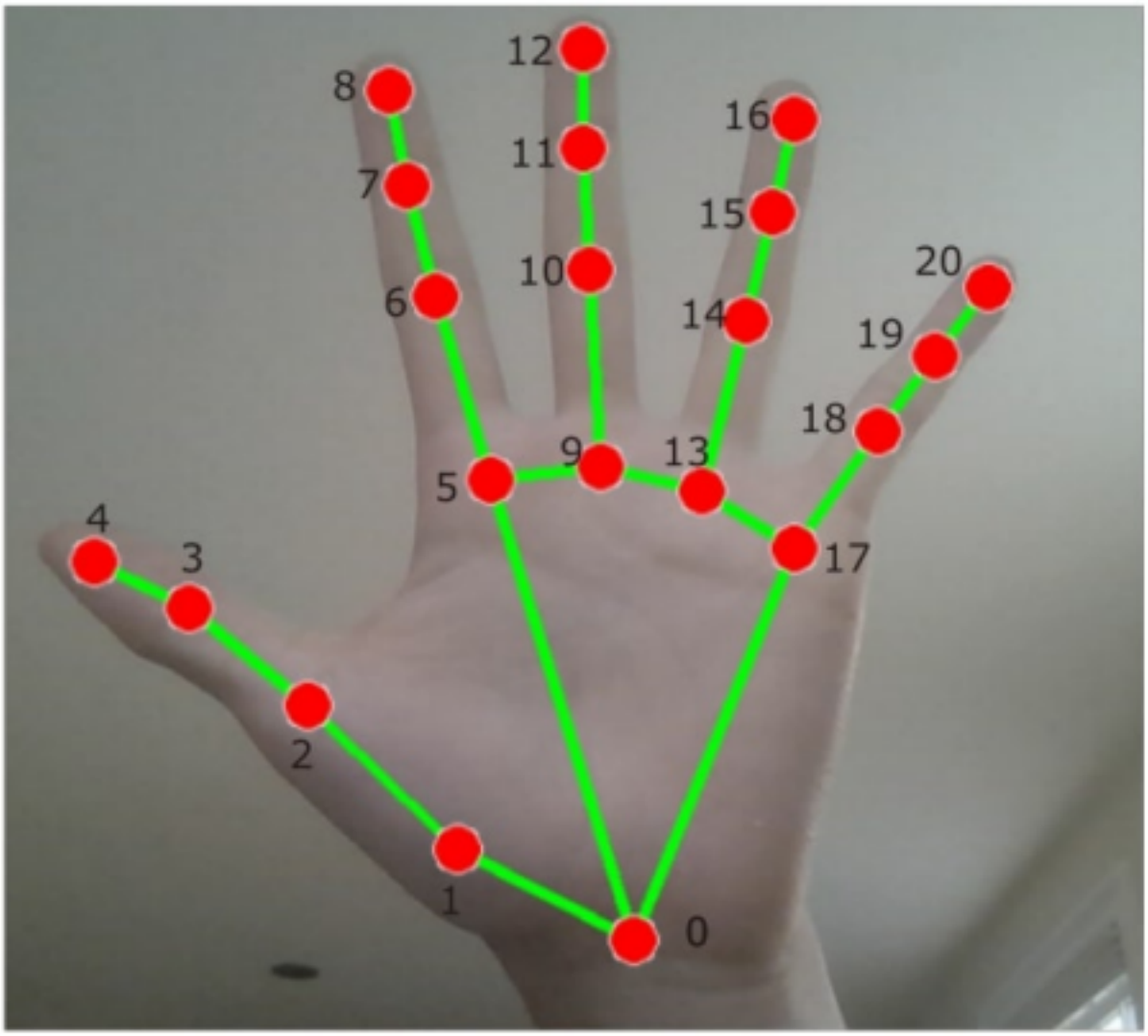


Figure 1: Visualization of Mediapipe hand landmark locations based on hand joints. Labels 1-4 are along the thumb, 5-8 are along the pointer finger, 9-12 are along the middle finger, 13-16

are along the ring finger, and 17-20 are along the pinky finger. In total there are 20 landmarks to represent the 3-D geometric state of the hand in any image.

2 Hand Geometry

In this section, we describe the Hand Geometry(HG) method, which recognizes ASL signs through the 3- D geometric relations between different joints or landmarks in the hand. This method is similar to past research on ASL methods, which used glove-like devices to render the hand in three dimensions and identify the gestures from there.[1] In contrast to these methods, our 3-D Hand Geometry method only requires 2-D

pictures and videos, which makes it more applicable to general users.

2.1 Landmark Identification

The objective of the Hand Geometry method is to utilize the Mediapipe hand landmarker library [2] which numerizes each finger joint into position values *x, y, z*. Each frame in a computer vision image can be represented by its three RGB values such that $F = \{\{F_{Ri}, i = 0, ..., n \cdot m\}, \{F_{Gi}, i = 0, ..., n \cdot m\}\}, \{F_{Bi}, i = 0, ..., n \cdot m\}\}$ where n and m are the dimensions of the image. The set of of hand landmarks is represented as $L = \{L_i = (x_i, y_i, z_i), i = 0, .., 20\}$ where each (x_i, y_i, z_i) tuple is defined by the Mediapipe model M such that

$$(x_i, y_i, z_i) = M(F, i) \quad (1)$$

where each of the *xyz* tuples represent the 3 dimensional location of the landmarks.

2.2 Landmark Geometry

Given the three dimensional position of all the landmarks in the hand, we then used a Python program which tests all the geometric relations between each landmark in order to determine the sign output. For example, the orientation of the three landmarks L_0, L_5, L_{17} give the direction of the palm depending on whether L_0 is the highest landmark of the three in terms of the y coordinate. Furthermore, the relation of different joints could be used to determine whether two fingers were in contact with each other or if they were inside the palm area.

```
def palm(self,pt):
    # form a triangle from the key points indexed by 0, 5, 17
    self.triangle = np.array([self.xyz[0], self.xyz[5], self.xyz[17]])
    assert len(self.triangle) == 3
    vecs = self.triangle - pt
    cosines = np.zeros((3))
    cosines[0] = np.sum(vecs[0] * vecs[1]) / (np.linalg.norm(vecs[0]) *
    np.linalg.norm(vecs[1])) cosines[1] = np.sum(vecs[0] * vecs[2]) / (np.linalg.norm(vecs[0])
    * np.linalg.norm(vecs[2])) cosines[2] = np.sum(vecs[1] * vecs[2]) /
    (np.linalg.norm(vecs[1]) * np.linalg.norm(vecs[2])) count = np.sum(cosines < 0)
    if count >= 2:
```

```
            return True
        else:
            return False

...

def letter_M(self):
    if self.touching(self.middle_tip,self.thumb_ip,self.accuracy):
        if self.touching(self.index_tip,self.thumb_ip,self.accuracy):
            if self.palm(self.pinky_tip):
                return True
    return False

...

def letter_C(self):
    if not self.palm(self.pinky_tip) and not self.palm(self.index_tip)
    and not self.palm(self.ring_tip) and not self.palm(self.middle_tip) and
    not self.touching(self.thumb_tip, self.middle_tip, self.accuracy):
        if self.touching(self.pinky_tip, self.ring_tip, self.accuracy) and
    self.touching(self.index_tip,self.middle_tip,self.accuracy):
            if not self.palm(self.thumb_tip):
```



```
                if self.palm_direction(self.wrist, self.index_mcp, self.pinky_mcp)[1] == "up":
                    return True
    return False
...
```

The above shows the Python code snippets of the 3-D HG method. A lot of "if else" statements are required to be programmed and it is quite time consuming to debug the code, but we are able to finish it in reasonable shape.

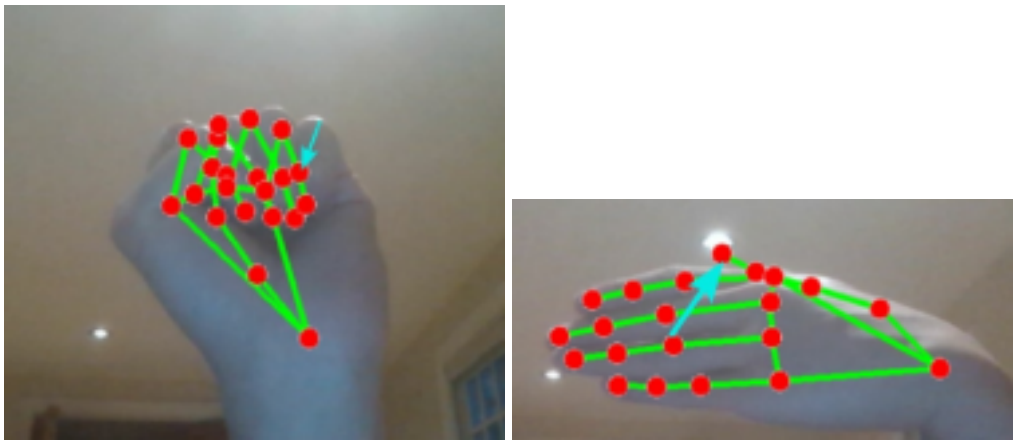
2.3 Implementation errors

One of the challenges in using Mediapipe's hand landmarks for identifying sign language is the varying geometric tolerances for different hands. Each person has different hand sizes, shapes and colors, leading to variations in the decision statements about landmark relationships. As a result, the accuracy of identifying gestures can differ between individuals. This discrepancy poses a challenge when creating a program that aims to accurately recognize sign language across a diverse range of users. Adjusting the accuracy thresholds to accommodate various hand variations becomes crucial but also adds complexity to the implementation.

Another source of inaccuracy when using Mediapipe's hand 3-D landmarks for sign language identification is the image background interference that can affect landmark detection. The hand tracking algorithm relies on distinguishing the hand from the background, and any elements in the environment that resemble or overlap with the hand can interfere with accurate landmark

labeling. For instance, if the background for the picture or video is a similar color to the user's hands, it will cause identification errors where the background becomes identified as a hand. These background factors can introduce errors in recognizing sign language gestures, compromising the overall accuracy of the system.

In sign language, different joints and regions of the hand are utilized to form specific signs or gestures. However, when performing certain signs, it is possible that some hand joints or landmarks get occluded or blocked from the camera's view. This occlusion can occur when the hand crosses over itself, when fingers overlap, or when certain hand configurations obscure specific landmarks. As a result, the missing or obscured landmarks can lead to inaccuracies in identifying the intended sign language gesture. Dealing with occlusion scenarios becomes a significant challenge in leveraging Mediapipe's hand landmarks for sign language recognition, as it requires additional techniques or algorithms to handle partial or incomplete landmark information.



(a) Error Example 1 (b) Error Example 2

Figure 2: Error Example 1 shows landmark errors when joints are close together. Error Example 2 shows landmark errors with background objects identified as joints.

3 Transfer Learning Method

In this section, we describe the second methodology of recognizing ASL signs from 2-D RGB video streams. We start with briefing the problem definition of ASL alphabet recognition in Section 3.1. Next we introduce the fine tuning technique applied in Section 3.3, which is a common technique in neural network based deep learning. In Section 3.4, we present our software design on mobile device that assists few shots data collections on a new domain (i.e. the hand of a new user) for the purpose of model fine tuning. Related implementation details are provided in Section 3.5.

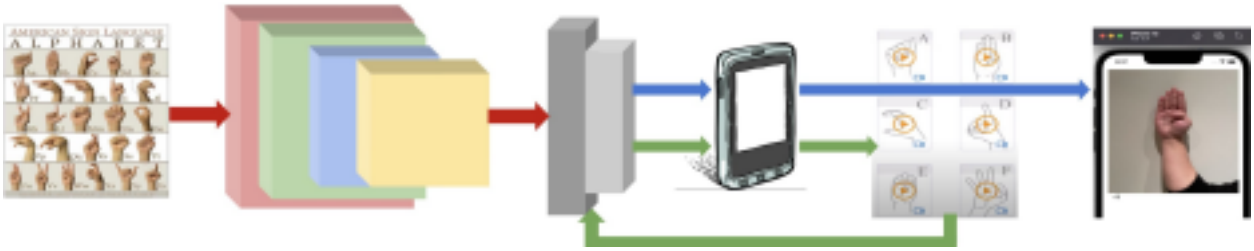


Figure 3: The proposed ASL recognition system featuring neural network model fine tuning on mobile device. Red arrow flow: offline ASL model training pipeline using public datasets. Green arrow flow: ASL model fine tuning using individualized collected datasets. Blue arrow flow: fine tuned ASL model online inference on mobile end. Our ASL recognition model takes more than 85,000 video frames and predicts ASL letters given mobile captured video stream in 30 fps.

3.1 Problem Definition

The objective is to determine a single ASL letter out of 26 alphabets and 3 special characters [3] from streamed video frames. Because some of the signatures in the alphabet such as "Z" and "J" are motion based, the frames in computer vision's terminology are represented as a sequence X of RGB images in terms of $X = \{x_{RGB,t}\}^{T}_{t=1}$, where the length of video frames is denoted by T. The set of ASL alphabets is represented as $Y = \{y_i, i = 1, .., 29\}$. Our ASL recognition problem hence can be defined as learning a model M such that

$$y_t = M(x_{RGB,t}) \quad (2)$$

where $y_t \in Y$ given the timestamp t is the recognized ASL letter. Because the prediction of the highest likelihood letter is the optimal one from 29 ASL characters, this problem is considered a classification problem, and therefore M would be a classification model with neural network weights to be optimized. For all timestamps $t = 1, ..., T$, $M(x)$ would represent the model ouput and $y(x)$ would be the vector representation of the label, with the vector value being 1 at the index where the classification is correct and 0 everywhere else. Therefore if $M(x)$ has the highest value at the same index where $y(x)$ is equal to 1, and furthermore if this highest value is very positive, the model would be more accurate for that one instance.

3.2 Model Learning

Given the classification problem stated in Eq. 2, we apply a similar modeling approach with traditional image classification deep neural networks [4] in order to address object classification on image domain. The model we trained can be decomposed into the following:

$$M = \{M_{BB}, M_{FC}\} \quad (3)$$

where M_{BB} represents the backbone network layers (BB) that encode the RGB images to multi dimensional features. M_{FC} is a fully connected layer (FC) placed at the end of M. It maps the encoded features to the

layer of classification output. The objective is to minimize the so-called cross-entropy loss[5]

between the classification predictions and target: $\sum_{n=1} l_n$

$$L(x, y) = \quad {}_N \quad (4)$$

P_N
where

$$l_n = -X^C c=1 \omega_c \log \exp(M(x)_{n,c}) \sum_{i=1} \exp(M(x)_{n,i}) y(x)_{n,c} \quad (5) \; P_C$$

where $P^{\exp(M(x)_{n,c})}$ $\sum_{i=1} \exp(M(x)_{n,i})$ is the probability predicted by the model of the label being in class c, and $y(x)_{n,c}$

C

is either 0 or 1 depending on whether the label is of class c. ω_c is the individual weight for each class, which is used because of the different number of appearances for each class in the training dataset. Because ω_c is there to counteract the training set inequality, it is a fixed value and does not change during model learning. As the model at the moment is a classification model for classifying between 29 classes, C is 29.[5]

In this work we used different optimizers, including RMSProp(RMSP), Adam, and AdamW, to obtain the optimal neural network parameters for $M(x)$ that minimizes cross entropy loss.

3.3 Model Fine Tuning

Following the learning objective defined in Eq. 4, model M can be trained by iterating through the dataset points $\{x, y(x)\}_D$. Evaluation of the model can be performed on the subset $D_{sub} \in \{x, y(x)\}_D$ which is unseen during the training. In real-world applications, ASL video frames are collected *in-the-wild* with diverse backgrounds and hand size, shape and color profiles. Those frames may fail at classification for initial M because the image data, in appearance, can be quite different from the dataset applied to model training. And as we can see in Sec. 2, these background factors can wildly affect the output results in a negative way.

To attain a more applicable model, we apply the fine tuning technique [6]. The core strategy of fine tuning a neural network is to continue the training referring to Eq. 4 on a fixed set of layers in model M while keeping the rest of layers weights unchanged. Specifically, the model weights of M are categorized as:

$$\{\omega, \omega \in M\} = \{\omega_{fx}, \omega_{fx} \in M_{BB}\} \cup \{\omega_{ft}, \omega_{ft} \in M_{FC}\} \quad (6)$$

where ω_{fx} are backbone layers in which the weights are fixed for feature extraction while ω_{ft} are weights of the final fully connected layer updated for fine tuning. The weights update of fine tuning layers M_{fc} follows the learning objective defined in Eq. 4 and 5 as to solve the same classification problem.

3.4 Mobile Application

The approach described in Sec. 3.3 is suitable for use cases where the new image frames are unseen to the base model but are adapted through fine tuning to achieve the

same learning objective. To assist with getting a more user-friendly interface of our proposed method, we take advantage of an Apple iPhone IOS based app *Translitero* and port the pre-trained model weight $\{\omega_{fx}, \omega_{ft}\}$ based on public hand-gesture dataset to the mobile end in Fig. 3, to provide the portability. The user interface allows the data collection process of the new use cases to be natural and fast.

In Fig. 3 we sketch the core components that supports the fine tuning in a data-efficient manner:

- Data Collection. The user interface opens up the camera to collect live video stream of the new user's hand gesture for up to ten seconds. The user is prompted to make hand gestures of the 29 ASL letters, and the recorded video clips are automatically labeled.
- Fine Tune Backend. The collected data in video clips(MP4) are uploaded to a desktop computer server where a pipeline takes the RGB video frames and then follows the procedure detailed in Sec. 3.3 to optimize and obtain the fine tuned neural network weights. The fine-tuned neural network weights ω^{new}_{ft} are then sent back to the mobile phone.

- Live Inference. The real-time video stream captured from mobile camera is projected to the handheld screen. In the meantime, the ASL model $\{\omega_{fx}, \omega^{new}_{ft}\}$ takes the raw frames and predicts the most likely ASL letter based on model inferences directly on the smartphone.

The session time of data collection is set as one minute maximum per ASL letter. The collected video clip from iPhone device has the frame rate of 30Hz. As demonstrated in section 4, 10 seconds recording (~300 frames) per ASL letter applied to model fine tune could attain an acceptable recognition accuracy, thus requiring less than 5 minutes recording time in total. It justifies our proposed approach in realizing in-the-wild ASL recognition as well as the mobile user interface that provides a user-friendly and rapid data collection method.

3.5 Implementation Details

The fine-tuning method of ASL letter classification, proposed in Sec. 3.2, is implemented under PyTorch deep learning code framework [7]. The architectural details are below:

- Back Bone. Following the definition of M_{BB} in Equ. 3, we adopt MobileNetV2 [8] as the back bone architecture with pretrained weights that takes RGB image frames in size of 224 x 224.
- Classfication. Following the definition of M_{FC} in Equ. 3, the classification layer is a fully connected (FC) one with dimension of 29 corresponds to the size of ASL letters introduced in this work.

For model training on M, we apply Adam optimizer [9] with the learning rate of 0.0005, and other optimizers. A learning rate scheduler is also utilized during the training. The

same optimization process is also adapted for the model fine tuning.

Translitero integrates multiple view controllers as the user interfaces developed in XCode 14.0 and IOS 14.5 to satisfy data collection or online ASL inference. Portability of the ASL model is supported by LibTorch 10.0 framework installed on an iPhone.

Methods	AA (%)	TM 5s (%)	TM 10s (%)	TM 20s (%)
HG MASL MASL+FT (Pro posed)	78.60 97.76 N/A	85.50 20.03 90.76	85.44 20.00 95.45	85.43 19.99 95.89

Table 1: ASL letter recognition accuracy in percentages. AA: results on ASL Alphabet dataset. TM: results on Translitero Mobile dataset. 5/10/15 seconds correspond to frame time length used for model fine tuning. The model fine tuning approach we proposed demonstrates the better accuracy.

Optimizers	RMSP [10] (10s) (%)	Adam [9] (10s) (%)	AdamW [11] (10s) (%)
Accuracy	95.33	95.45	95.48
Blurred Background	MASL + FT (5s)	MASL + FT (10s)	MASL + FT (20s)
Accuracy	89.99	95.20	95.56

Table 2: Ablation study of fine tuning based ASL letter recognition accuracy in percentage. Row 1-2: results of adapting different optimizers on model fine tuning under the same data collection time (10s); Row 3-4: results of applying image frames with blurred background but different data collection time during model fine tuning.

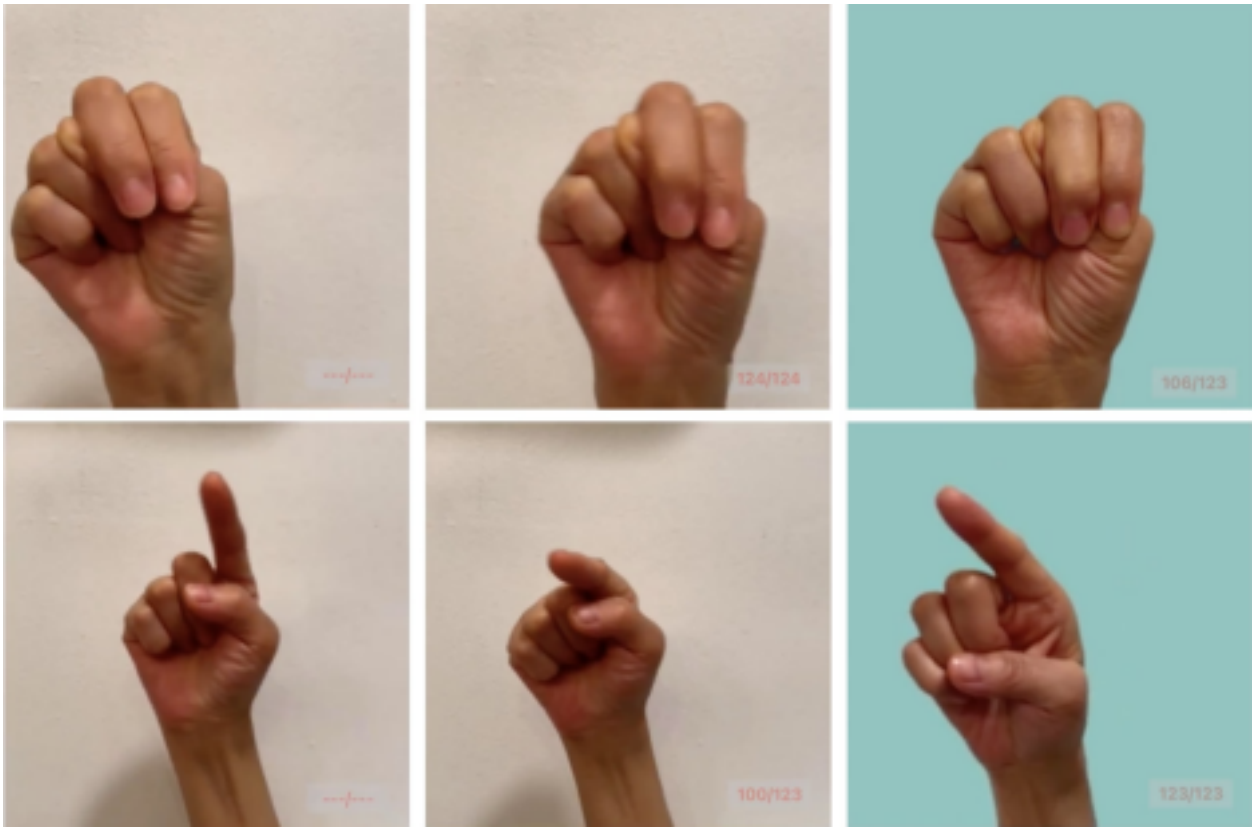


Figure 4: Visualization of ASL letter recognition qualitative examples. Column 1: ASL gesture frame samples captured from mobile camera using the app. Column 2: ASL letter recognition accuracy (bottom right red fonts) tested on pre-recorded ASL clips. Column 3: verification on recognition accuracy while applying virtual background to the clips. Row 1 and Row 2 correspond to ASL letters N and Z.

4 Experiments

We have carried out ASL recognition experiments on teenager, adult and senior male and female hands in the wild and the accuracies obtained are compared.

4.1 Datasets

- ASL Alphabet. The training data contains 87, 000 images in 29 ASL letter categories, are 26 letters A-Z and 3 classes for SPACE, DELETE and NOTHING, which are necessary for spelling words and sentences.
- Translitero Mobile. Short-duration ASL alphabets video frames are captured using an iPhone. Those video clips are 1 minute in length containing 29 ASL letter categories to validate the model's ability of generalization.

The ASL Alphabet dataset is further divided into training and testing splits for model learning and evaluation purposes. Video segments from the Translitero mobilephone-collected dataset are sampled in fixed length for model fine tuning and transfer learning. The rest of the video segments are used for evaluation. Variation of

the sample video frame length used for fine tuning is discussed in Sec. 4.3

8

4.2 Methods

Among the various methods tested, the most effective methods for ASL letter recognition are:

- Hand Geometry(HG). This method extracts 3-D hand key points (landmarks) based on video image frames utilizing the Mediapipe [12] library. The geometry-based classifier on 29 ASL letters are derived based on hand key points geometric relations.
- MobileNet ASL(MASL). The vanilla ASL classification model developed according to Sec. 3.2, 3.5 using public datasets with no customization. The model is trained end-to-end from image frames to 29 ASL classes upon ASL Alphabet dataset.
- Fine Tuned MASL(MASL+FT). The ASL classification model which uses the fine tuning technique (Sec. 3.3) using new individualized data from Translitero Mobile clip frames.

4.3 Results

4.3.1 Quantitative Results

We summarize the quantitative results in Table 2 of the recognition accuracies evaluated on ASL Alphabet and Translitero datasets. The percentages are averaged over the 26 ASL letters A-Z. HG accuracies are all below 90% which indicates that the 3-D geometric cues extracted from hand key points are not precise enough to attain exceptional recognition accuracy. Results of MobileNet ASL(MASL) trained on ASL Alphabet dataset demonstrates the great performance if tested as well on the ASL Alphabet public dataset. However, the low recognition accuracies on Translitero Mobile dataset collected on new individuals indicates the model does not have the ability of generalization. During test time, a few of the ASL letters recognition accuracies are observed to be nearly zero. On the other hand, after employing the model fine tuning technique to MobileNet ASL model(MASL+FT), the issue of generalization gets largely mitigated. The 10s video frame length per ASL letter applied for fine tuning could achieve the test accuracy above 95%. Those quantitative results validate our transfer learning approach on improving performance of neural network models with high data efficiency.

4.3.2 Qualitative Results

In Fig 4, we selected two ASL letter recognition cases, "*N* " (row 1) and "*Z*" (row 2). The selected two cases are challenging according to our observations. Letter *N* is similar to *M* in appearance, with the only difference being in the position of the thumb tip relative to the fist. *Z* is a dynamic letter, which means that it involves the hand moving during the duration of the gesture. Since our modeling approach predicts per frame, there is a

chance that the dynamic letters could suffer from lack of data. Despite that, most of our testing results showed 100% accuracies, which proves the quality when executing the fine tuned model.

4.3.3 Ablation Study

In machine learning, an ablation study is when we study the effects of parts of the model by removing or silencing certain layers. We conducted this type of study upon our fine tuned model to explore more alternatives that could obtain the optimal performance in terms of ASL letter recognition accuracy. In this work, we investigate the following variants:

- Optimization Algorithm. During the model fine tuning phase, the weights of fine tune layers, ω_{ft}, are updated according to the gradient descent algorithm. Choices of optimization could result in different efficiency and accuracy.
- Virtual Background. Considering the high background diversity of the video image frames captured on the mobile device, we additionally develop a functionality that enables attaching a customized background to the video stream. Inside the camera view, the background is blurred resulting in only

the moving hands being visible. We apply this setup on both the fine-tuning data collection stage and real-time ASL inference stage.

In Table 2, among the listed optimization algorithms, the Adam optimizers [9, 11] attains better accuracy than RMSP [10]. And AdamW [11] could achieve the best accuracy given 10 seconds of fine-tuning data collection. Further, we observe that the ASL recognition accuracy is not having much performance change with various physical backgrounds (curtains, lighting, wallpaper) after applying the virtual background. For all the fine tuning collection time tested on 5, 10, 20 seconds, the accuracy results are close to the uniform background ones. Those are also demonstrated in the qualitative results in Fig. 4. These results prove the effectiveness of developing the virtual background feature in our mobile APP such that the impact of physical backgrounds to the ASL model could be mitigated to greatly enhance the usability.

Letter/ Demo graphic	Teenage Male	Teenage Fe male	Adult Male	Adult Female	Senior Male
C	100%	73.2%	100%	100%	100%
E	100%	100%	100%	100%	100%
H	89%	100%	100%	100%	100%
O	100%	89.4%	100%	100%	100%

R	94.2%	91.1%	100%	100%	63%
SPACE	100%	100%	100%	100%	100%

Table 3: The above shows the hand gesture data collected from the mobile app. ASL finger spelling recog nition was analyzed to assess the accuracy across categories of a teenage male, a teenage female, an adult male, an adult female, and a senior male. The accuracy is determined by the correct number of hand gesture predictions divided by the total frames, with the number of frame ranging from 80 to 123 depending on the device. The recognition system achieved high accuracy for teenage males and females as well as adult males and females. However, the accuracy is still less satisfactory for a senior male, likely due to factors such as hand tremors and variations in hand shape or size.

5 CONCLUSIONS

In conclusion, our research translates ASL to English with greater than 95% accuracy using only 2D video image-based inputs. The transfer learning method proved to be highly data-efficient during the training stage and was a lot more effective during testing due to its adaptiveness to different hands and environments. Furthermore, the mobile app with the transfer learning algorithm can be used in practical circumstances with different physical backgrounds that occur in real life. This could mean that ASL learning for the sign language community could be much easier in the future.

The ASL algorithms and app may be further expanded into more areas. The app can be made to accommodate more than just the ASL alphabet, such as complete words. In addition, the sign language transfer learning method can be applied to more sign language systems such as the Spanish / French / Russian Sign Languages.

Acknowledgement

We acknowledge the patient teaching and guidance by Mr. Yingchun Chen and Mr. James Ma.

References

[1] Zhihao Zhou, Kyle Chen, Xiaoshi Li, Songlin Zhang, Yufen Wu, Yihao Zhou, Keyu Meng, Chenchen Sun, Qiang He, Wenjing Fan, Endong Fan, Zhiwei Lin, Xulong Tan, Weili Deng, Jin Yang, and Jun

Chen. Sign-to-speech translation using machine-learning-assisted stretchable sensor arrays. *Nature Electronics*, 3:571–578, 2020.

[2] Google. Hand landmarks detection guide.

https://developers.google.com/mediapipe/solutions/ vision/hand_landmarker.

[3] Asl alphabet. https://www.kaggle.com/dsv/29550.

[4] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[5] Pytorch. Crossentropyloss. https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html.

[6] Joanne Quinn, Joanne McEachen, Michael Fullan, Mag Gardner, and Max Drummy. *Dive into deep learning: Tools for engagement*. Corwin Press, 2019.

[7] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[8] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mo bilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

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

[10] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Cited on*, 14(8):2, 2012.

[11] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[12] Camillo Lugaresi, Jiuqiang Tang, Hadon Nash, Chris McClanahan, Esha Uboweja, Michael Hays, Fan Zhang, Chuo-Ling Chang, Ming Guang Yong, Juhyun Lee, et al. Mediapipe: A framework for building perception pipelines. *arXiv preprint arXiv:1906.08172*, 2019.

12