# Hand Sign Detection

Justin Liaoo
Khoury College of Computer Science
Boston, MA USA
liao.ju@northeastern.edu

***Abstract*—For the final project a hand sign detection application was implemented to demonstrate the basic ideas of image processing to extract features to store in a database to train a fully connected dense network. This paper describes the process of developing this application as well as design decisions made to iteratively improve predictive accuracy and the user experience (perceived framerate). The final product implements MediaPipe to extract hand landmarks representing the 21 joints of a hand projected into 3D space. This pre-processing through the MediaPipe network was selected for its efficient ability to segment the hand and process the features to accurately identify the landmarks, and ease of implementation. The points were normalized in respect to the wrist's joint point and used to train and test a fully connected network to classify a user's hand sign at the application run time. The resulting application produced consistently accurate predictions for a database developed with a specific user's hand. The limitation of this application was the ability to fully generalize model to analyze hands of other users but could be remediated by expanding the database to include different hand sizes and different levels of dexterity to represent a hand sign.**

***Keywords—image processing; hand sign; neural network; MediaPipe; preprocessing; segmentation; feature detection; classification***

## I. Introduction

To showcase the learned concepts from the CS5330 Pattern Recognition and Computer Vision course, a final project was completed to demonstrate some of the basic concepts of computer vision and machine learning to process video feed to extract hand gestures and label them. This report will visit and explain key concepts to achieve the end product as well as experiments that were conducted to improve the overall performance of the program.

The main goal of this project was to be able to detect hand signs. A simple premise with multiple approaches available.

A major component of achieving reliable predictions of the program when given a target image with a hand is to identify the hand prior to identifying it. This segmentation of the hand from the background is the core concept to begin preprocessing of the input prior to identification. The use of the open source MediaPipe [1] library was used to identify 21 hand landmarks representing joints on the hand. These points represent 3D points homogenized in respect to the camera's world point being the origin.

These 3D landmarks are then cleaned and normalized in respect to the wrist point as the origin to make the data scale and translation invariant. Since hand gestures such as American Sign Language (ASL), provide meaning based on the movement of the hands and the orientation along with actual shape of the hands, rotational was not considered.

Lastly, the cleaned data is used to train and test a simple fully connected network with no convolutional layers for classification. This pipeline of using MediaPipe to identify landmarks that are then used to train and test a TensorFlow model resulted in high accuracy predictions during real time deployment of the trained model.

The report will highlight the experiments to achieve the end resulting product and why some decisions were made.

## II. Related Work

### A. *Research on the Hand Gesture Recognition Based on Deep Learning [2]*

In this paper written by Sun et al, the authors described their method of hand gesture recognition. They accomplish this by creating a model that segments the hand by establishing the skin color and using AdaBoost classifier based on Haar features from frame-to-frame changes. The segmented hand then uses a CamShift algorithm to track the target to be used for gesture recognition of the 10 common digits in a convolution neural network. The general steps of isolating the hand to begin processing the hand gestures was the inspiration of the changes made in this project to improve the classification predictions in the iterative improvement process.

### B. *MediaPipe: A Framework for Building Perception Pipelines [3]*

In this paper written by Lugaresi et al, the authors described the MediaPipe framework and the general features/tools available in the library to allow for iterative improvements on applications that is cross-platform and reproducible. The general ideas of how this framework operated was used to better understand how to implement it for this project.

### C. *Real-time Hand Gesture Detection and Classification Using Convolutional Neural Networks [4]*

In this paper written by Köpüklü et al, the authors described a hierarchical structure that enables offline-working convolution neural network (CNN) architecture for real-time hand gesture detection. The architecture consisted two models: a lightweight CNN to detect gestures and a deep CNN for classifying the detected gestures. The learning of this paper helped understand the complexity of implementing a solution to classify a solution for ASL letters “j” and “z” which required gestures. A simple solution would have been possible but to fully develop a robust solution would had been out of scope of the project.

## III. METHODS

### A. *Programming Language*

This project was created using Python as the main programing language for its usage in the data science community which has many supporting libraries and resources to quickly create sandbox environments and test out new libraries. Since this project is to showcase the basic concepts of computer vision and machine learning for computer vision, the real time performance was not a priority to consider. There will be some instances of slower frame rates during run time but will not hinder the overall experience.

### B. *Supporting Libraries*

#### 1) *MediaPipe [5]*

Lightweight library of tools that can be used to apply artificial intelligence (AI) and machine learning (ML) to programs. It contains ready to use solutions for hand detection using its own built in machine learning models to identify different types of known landmarks. For this project, the “Hands” solution was used to identify the features to create learning model.

#### 2) *OpenCV [6]*

A cross platform library specifically created to tackle computer vision tasks. For this project, this library was used to create the main user interface (UI) as well as simple filtering.

#### 3) *TensorFlow [7]*

An open-source library to aid in the development of AI and ML applications by simplifying the creation of complex models for use in applications such as predictive analytics, natural language processing, and image processing. For this project, this library was used to create the final model for its ease to create a ML model without knowing too much about the semantics of the underlying framework.

#### 4) *scikit-learn [8]*

An open-source library in Python that provides tools for data analysis and predictive ML models. For this project, it was used to split a single dataset to a train and test set for the classification model.

#### 5) *PyTorch [9]*

An open-source library developed to be a flexible and efficient platform for building and training deep learning models. For this project, this library was initially used to develop the model but was later switched to TensorFlow for its simpler setup.

#### 6) *Pandas [10]*

An open-source library built on Python that is used for data analysis and data manipulation. For this project, this library was used to aid in accessing and manipulating the raw data and collect it into a single file.

#### 7) *NumPy*

A scientific computation library in Python that provides support for multidimensional array objects. This library is serves as fundamental frameworks for many Python based libraries.

### C. *Operating System and Hardware*

This project was developed using MacOS with the M1 Apple Silicon. Though the libraries mentioned in this section are open source and cross-platform, Apple Silicon is still catching up in terms of support. Notably, MediaPipe and TensorFlow require some specific set up parameters. For MediaPipe, the library only supports Python 3.7 to 3.10 for non-Apply Silicon. Therefore, to use this library, the mentioned Python version must be used and a third-party version must be used: MediaPipe-silicon. For TensorFlow the installation requires installing a specific metal version [11].

### D. *Initial Approach*

Upon first investigation to solving the problem, American Sign Language (ASL) was considered for the initial training dataset because it easily consists of 24 different hand signs to start developing hand recognition model. The two missing letters, “j” and “k”, are motioned based and was out of scope at the time. Fig. 1 contains the ASL alphabet as described

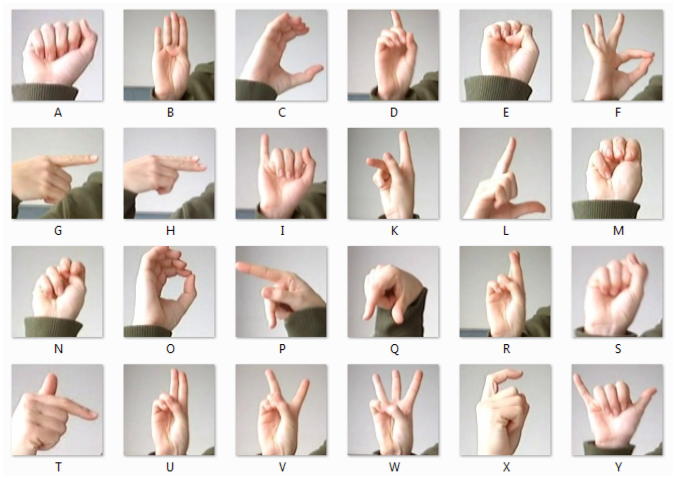


*Fig. 1: American Sign Language Alphabet [12]*

Several data sets exist on Kaggle that allow for easy accumulation of data. The first of these data sets consisted of 27,455 training cases and 7172 test cases in 28x28 square 1 channel pixels which is similar in format to classic MNIST datasets [12]. An example of the loaded data was provided in Fig. 2.

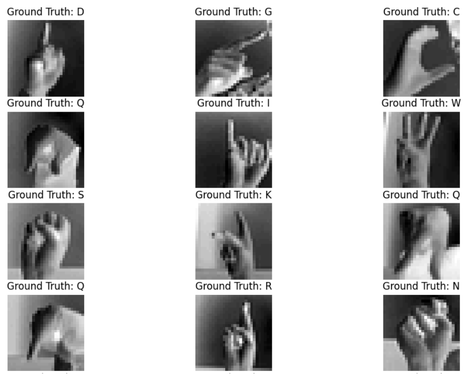


*Fig. 2: Sample of Kaggle Sign Language Dataset*

With less developed knowledge of PyTorch, a solution from Vignesh [13] was used as an initial starting point to create and train a PyTorch network. A pictural representation of the network is shown in Fig. 3. An image from the data set of 28x28 square pixels was enlarged to 244x244 square pixels, retaining the single channel status. By using five convolution layers and an enlarged image, the goal was to allow the network to learn more weights to detect useful features from the dataset. To train this network, a training batch size of 64 was used with 10 epochs. The validation was completed with a test batch size of 1000.

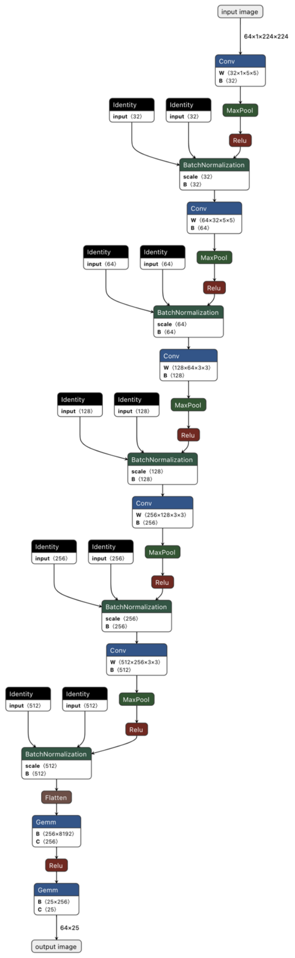


*Fig. 3: Sign Language Neural Network*

At runtime testing with video feed was conducted using MediaPipe to identify a bounded square area of a right hand at a given frame. This bounded area was then cropped and processed by converting to a gray scale image and scaling it down to 28x28 square pixels to match the original data set. The purpose of this processing prior to the actual pre-processing was to homogenize the data.

## *E. Realtime Prediction Improvement*

In the case of the model unable to generalize the real time data, the dataset was expanded to process larger more detailed images. The intent of using larger images was to provide the model with better pixel resolution for hand details to differentiate similar hand signs such as "a" and "e" where the difference is simply the placement of the thumb. The dataset that was tested was from tecperson [14]. The higher resolution images were resized to fit the input size of the original model.

## *F. Landmark Extraction*

Instead of using MediaPipe to crop the image to put into a neural network, MediaPipe was used to extract the landmarks to be used as an input vector into another neural network. Fig. 4 shows an image of the landmarks detected on a hand. There are 21 points that represents a hand in 3D space. This information can be flattened into a 63-element long feature vector after normalization at the wrist point. Each point will then have a relative 3D space around the wrist. The limitation of this method was that the data that was generated for training and testing will be relative to the current user's hand. Another individual with a different hand size will experience sub-optimal predictions.

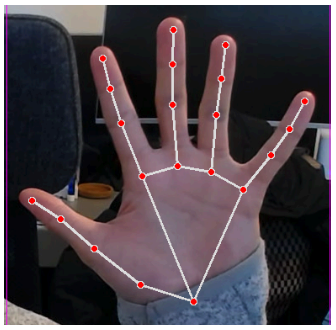
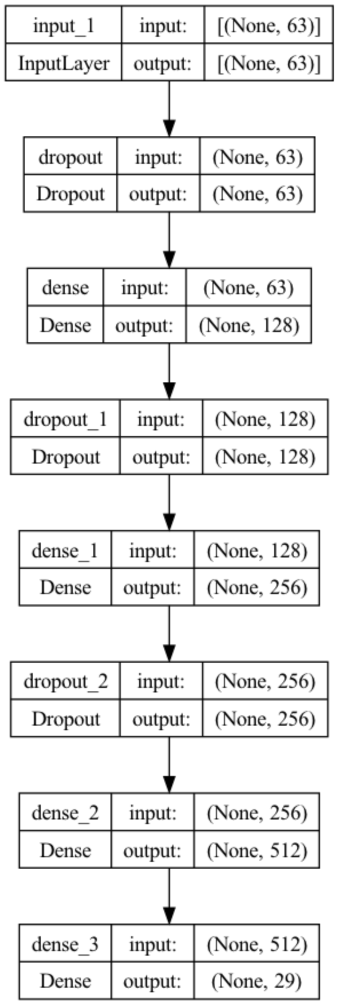


*Fig. 4: Hand Landmarks*

By using a feature vector of landmarks, this method eliminates extra work the neural network has to do because the hand is already segmented and analyzed by the MediaPipe model. Essentially, with this method, the task was now transformed from a segmentation and classification problem to simply a classification problem. This method also has the added benefit oof the ability to isolate strictly the hand in the learning process in the model development because it extracts the hands in the frame and ignores all other features in the frame from the model. A new network would have to be developed to classify the landmarks with a 1D input into the network. Fig. 5 depicts this new model. A GitHub repository by Kninivi was used as reference to generate the TensorFlow model [15]. A

TensorFlow model was adopted due its simpler setup process and simpler data setup.


input_1 | input: | [(None, 63)]
InputLayer | output: | [(None, 63)]
dropout | input: | (None, 63)
Dropout | output: | (None, 63)
dense | input: | (None, 63)
Dense | output: | (None, 128)
dropout_1 | input: | (None, 128)
Dropout | output: | (None, 128)
dense_1 | input: | (None, 128)
Dense | output: | (None, 256)
dropout_2 | input: | (None, 256)
Dropout | output: | (None, 256)
dense_2 | input: | (None, 256)
Dense | output: | (None, 512)
dense_3 | input: | (None, 512)
Dense | output: | (None, 29)
*Fig. 5: TensorFlow Landmark Network*

## *G. Defining Dataset*

Upon closer inspection of the higher resolution dataset, the MediaPipe hands solution had difficulty isolating the hands from the background for some images as shown in Fig. 6 where is depicts the number of vectors for each Unicode letter that the MediaPipe hands solution was able to detect. The data suggested that Unicode 116 or the letter "t" was hardly identifiable. Therefore, instead of opting to use the dataset from Kaggle, a dataset was generated during run time of the application to collect 50 data points of the landmarks and use that to train the model. Each entry in the data set would be represented as a normalized value with the wrist point as the center of origin in 3D space. The data was stored in a CSV based data store. The biggest limitation at this point was that the data collected would work best for individuals who created the database with their hands. Prediction performance would be hindered if a hand that was not used in the training was used due to a person's hand size and dexterity.

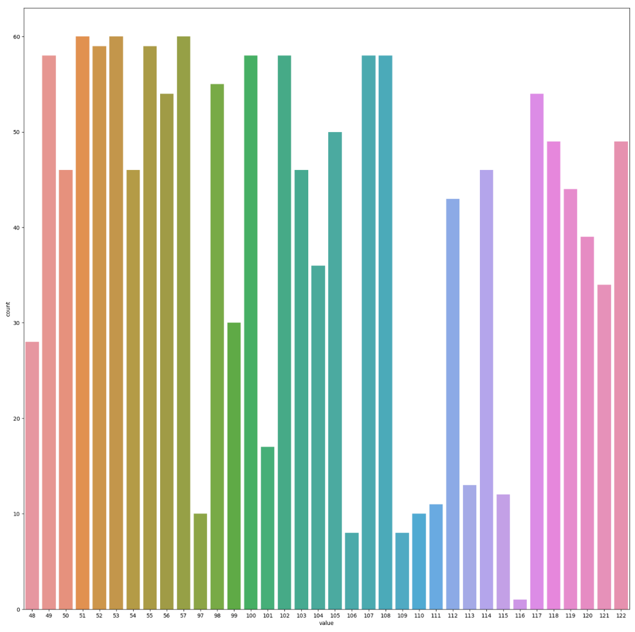


*Fig. 6: High Resolution Data Map*

## IV. EXPERIMENTS

In the initial design of the network model, Vignesh design performed well in the training and testing process, able to get up to 95% when being tested with the dataset. Fig. 7 indicated that the data trained within 1 epoch, where each epoch consisted of 1000 test cases. Further training passed 1 epoch did not improve the overall accuracy of the model. This initial model also indicated that the F1, which was a measure of the harmonic mean of precision, of 1, meaning the test set itself had good accuracy.

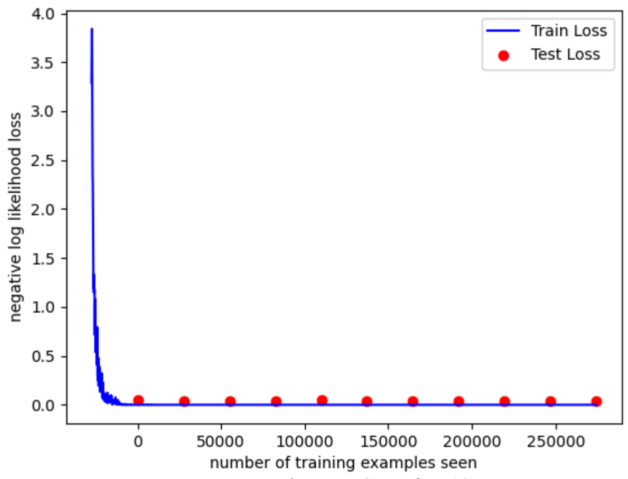


*Fig. 7: Initial Deep CNN for ASL*

When this model was implemented during real time prediction from a video feed, the results were not consistent. First, the input from the video required the target hand to have a solid light-colored background, meaning that any extra textures in the input image resulted in incorrect prediction values. Fig. 8 depicts what the application should be processing to predict the hand signs at run time. Fig. 9 depicts the proper input to be used in the model. This result was not optimal because the solution would require the user to place their camera in a specific location to allow the application to run properly which is not ideal.

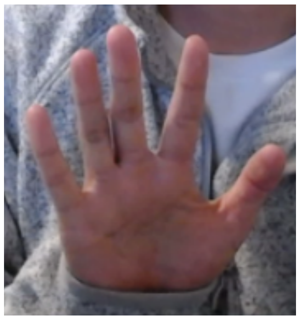


*Fig. 8: Target image to be processed*

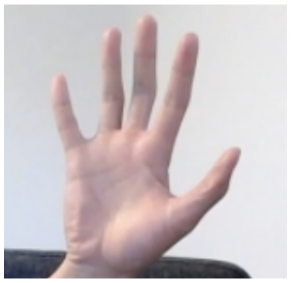


*Fig. 9: Actual image able that could be processed*

Second, there are some similar hand signs could not be differentiated easily at run time prediction. For instance, the letters “m” and “n” were hard to distinguish. The lower resolution 28x28 square pixels were often too low resolution to use at real time to correctly predict the true values.

Therefore, to remediate these issues, a new higher solution image set was used to train and test the model. Instead of training directly with the data set, the data set was passed through the MediaPipe hands solution to extract the data for the landmarks. The intent of using this method was to provide the MediaPipe to do the segmentation and extraction of points to produce the necessary feature vectors. The higher resolution, in theory, allows the MediaPipe algorithm to work properly.

Unfortunately, as mentioned in Section III G, the algorithm did not perform well because it had difficulty extracting points for some hand signs. For the hands signs that were detectable, the response at run time indicated that the model could detect distinct ASL hand signs such as the letter “b”. The main issue was that for slightly more difficult hand signs such as “f” or “k” were subject to incorrect classifications. A possible hypothesis would be the lack of ASL proficiency made it hard to categorize compared to someone who is proficient. These phenomena would be akin to different handwriting between individuals or a native speaker having better pronunciations than a non-native speaker.

Instead of using a pre-defined set of data, a database was created using the user’s defined hand signs. The resulting application would be able to predict better based on the individual’s distinct ASL style. Also, this method helps ensure that each entry in the dataset would yield landmarks unlike the method of attempting to find them in a pre-defined photo dataset.

As mentioned in Section III.E, the model was changed to from an all-encompassing feature learning and classification problem into strictly a classification problem. Therefore, the model was simply processing a flattened vector of 21 3D points. The result of this change in model resulted in consistently correct predictions from the model if the database was developed with that specific user’s hands.

By using the new model design above and the new type of dataset, the model still trained well. The train batch size was 128 cases using 1000 epochs. The training was conducted so that it allowed early termination after stability for 20 epochs. In many instances, the trained only lasted about 80 epochs with early termination with the entire training process taking approximately 5 minutes maximum. This quick turnaround actually allows for the user to collect data and be using it in quick succession with the help of parallel processing.

For this implementation of the data set and the model, the limitation is the ability to fully generalize the accuracy of the model to outside the scope the defined database. Several other users of the applications attempted to complete the entire ASL alphabet with 100% correct predictions without difficulty (i.e., taking more than 10 seconds) but were unable at first before their hands were used within the database. Therefore, this lack of generalization appeared to be solvable if more data was collected with different hands. Even though the hands were normalized in respect to the wrist point, the proportions of the fingers could differ from person to person. The ratios between the wrist and other joints were not consistently normalized which makes the model unable to generalize to different individuals.

## V. Discussion and Summary

By switching the overall pipeline of processing a frame from the video feed from processing an entire cropped image of a hand through a CNN to using two smaller networks allowed for simpler implementation for the application with better performance. In addition, through running the application, the base framerate without any processing was about 60 frames per second (fps) and dropped to about 15 fps when the MediaPipe detection was on. Though, the framerate decreased about a factor of 3, there were no other major drops in framerates from the classification model to hinder the overall application experience which was a major consideration in the design process.

The use of higher resolution images and explicitly creating the dataset allowed for equal distribution of categorical datapoints. This method of data collection allowed for more even training between categories but also allowed for the identification of categories that needed more data. The implementation allowed for quick turnaround from collecting landmark points to implementation of the retrained model. As mentioned before, the initial database was generated with the use of single user's hand which limits the performance accuracy because the ratio between the wrist landmark and the others are different based on individual. Therefore, for better prediction accuracy, the database and model require a larger pool of data from different individuals.

Overall, this pipeline depicted in this paper was effective in producing a growing model that could learn and classify more hand gestures that are not limited to ASL hand signs. More data would be needed to general the model more but the method of data collection allows for that expansion into generalization. This model that was developed worked well for static hand signs but the related work describes in the paper by Köpüklü et al provided means to expand this model to analyze and train hand gestures motions as well.

## References

[1] Mediapipe: https://developers.google.com/mediapipe/solutions

[2] J. -H. Sun, T. -T. Ji, S. -B. Zhang, J. -K. Yang and G. -R. Ji, "Research on the Hand Gesture Recognition Based on Deep Learning," *2018 12th International Symposium on Antennas, Propagation and EM Theory (ISAPE)*, Hangzhou, China, 2018, pp. 1-4, doi: 10.1109/ISAPE.2018.8634348.

[3] Lugaresi, Camillo, et al. "Mediapipe: A framework for perceiving and processing reality." *Third Workshop on Computer Vision for AR/VR at IEEE Computer Vision and Pattern Recognition (CVPR)*. Vol. 2019. 2019.

[4] Köpüklü, Okan & Gunduz, Ahmet & Kose, Neslihan & Rigoll, Gerhard. (2019). Real-time Hand Gesture Detection and Classification Using Convolutional Neural Networks. 1-8. 10.1109/FG.2019.8756576.

[5] *OpenCV*. OpenCV. (2023, April 19). Retrieved April 24, 2023, from https://opencv.org/

[6] *Tensorflow*. TensorFlow. (n.d.). Retrieved April 24, 2023, from https://www.tensorflow.org/

[7] *Pytorch*. PyTorch. (n.d.). Retrieved April 24, 2023, from https://pytorch.org/

[8] *Learn*. scikit. (n.d.). Retrieved April 24, 2023, from https://scikit-learn.org/stable/index.html

[9] *Pandas*. pandas. (n.d.). Retrieved April 24, 2023, from https://pandas.pydata.org/

[10] *NumPy documentation*. NumPy documentation - NumPy v1.24 Manual. (n.d.). Retrieved April 24, 2023, from https://numpy.org/doc/stable/index.html

[11] Inc., A. (n.d.). *Tensorflow plugin - metal*. Apple Developer. Retrieved April 24, 2023, from https://developer.apple.com/metal/tensorflow-plugin/

[12] Tecperson. (2017, October 20). *Sign language mnist*. Kaggle. Retrieved April 24, 2023, from https://www.kaggle.com/datasets/datamunge/sign-language-mnist?datasetId=3258&searchQuery=pytorch

[13] Vignesh, V. (2020, July 1). *CNN - Pytorch - 96%*. Kaggle. Retrieved April 24, 2023, from https://www.kaggle.com/code/vijaypro/cnn-pytorch-96

[14] Thakur, A. (2019, April 28). *American Sign Language Dataset*. Kaggle. Retrieved April 24, 2023, from https://www.kaggle.com/datasets/ayuraj/asl-dataset

[15] Kiselov, N. (n.d.). *hand-gesture-recognition-mediapipe*. GitHub. Retrieved April 24, 2023, from https://github.com/kinivi/hand-gesture-recognition-MediaPipe