American Sign Language Alphabet Recognition Using Computer Vision

Ramisa Tahsin Rahman, Yu Wati Nyi, Hong Anh Nguyen

Abstract:

This application is an innovative solution that tries to leverage computer vision and translation technologies to bridge communication gaps between individuals using American Sign Language (ASL) and those who may not be proficient in sign language. While the goal of the project, in the beginning, was to create an application that could recognize and translate common ASL phrases, we ended up narrowing the scope of our project to work with ASL alphabets. Our current application provides real-time recognition of ASL alphabets captured through a device's camera, translating them into text for a seamless and inclusive communication experience. We apply the hand recognition model from MediaPipe to detect users' hand signs in real-time. In addition, we processed the photos with Gaussian Naive Bayes classification to identify which alphabet letters the users demonstrated. At the moment, the model performs well on some specific words like A, B, C, I, and Y but sometimes gets mixed up between similar letters like H and U.

Introduction

American Sign Language (ASL) is a natural language that serves as a predominant means of communication for speech-impaired people. However, the gap in understanding between sign language users and those unfamiliar with ASL remains a persistent challenge. This project is our attempt to bridge this communication gap through the implementation of an ASL alphabet recognition and translation model driven by machine learning and computer vision. Our project leverages a machine learning model that is trained to recognize and translate ASL alphabets in real time. Using computer vision techniques, the model captures the location of the user's hand providing a robust framework for accurate ASL alphabet recognition. The implementation includes combining key point information to enhance the model's precision in interpreting ASL alphabets. With our project being able to detect alphabets (not including J and Z) from a live webcam, our project has the potential to detect instances of using sign language for words and expressions. If we continue to develop the dataset and the model further to be able to predict labels for video datasets, we will be able to have a more useful and accurate translator that will be able to detect hand movements instead of just instances, such as being able to detect ‘J’ and ‘Z’.

Related work

Sign Language Recognition is a well-studied topic and a very interesting project to apply the machine learning model. With the growing popularity of computer vision and convolutional neural networks, there are multiple works online where machine learning was used to create a real-time/image translator for various sign languages. Completely translating sign language using breakthrough technology is an ongoing research. One of the best developments so far has been Google's Project Shuwa. It is a collaboration between Google, the Chinese University of Hong Kong, the Nippon Foundation, and Kwansei Gakuin University. The AI model used in the project detects every part of the human upper body. This is because sign language is not just communicated through hand movements, but also through facial expressions and body language. SignTown, the final product of Project Shuwa, aims to 'authentically solve for the deaf community and increase awareness' by teaching sign language through real-time video and the AI model. Our project focuses more on capturing instances of hand gestures through each image frame in a video. It does not detect hand movements, body language, and facial expressions. To get to this point, we have reviewed multiple tutorials and projects.

Our first inspiration was Nicholas Renotte's YouTube tutorial on *Real Time Sign Language Detection with TensorFlow Object Detection with Deep Learning and Python*. Renotte walks through the process of building a sign-language detector and he introduces the use of importing pre-trained models and retraining them with a new set of manually created data. Our first trial to create a machine-learning model that detects American sign languages follows similar steps to this tutorial. Similar to the tutorial, we created our dataset and then we imported a pre-trained model to retrain it. What we did differently was that in our trial, we imported a pre-trained model from TensorFlow Hub while Renotte used a pre-trained model from TensorFlow Model Zoo. The main difference between the two repositories is that TensorFlow Hub is modular. The user can customize the model. Whereas the model from TensorFlow Model Zoo is a complete model without the need to customize. Another difference between our works was the way we collected and used our data. Renotte collected his image data and assigned labels to each image by using LabelImg, an image labeling tool by Label Studio. The user can use the tool to create manual descriptive labels for multiple parts of one image. In our project, we collected our own data and created code for each image to automatically save in a folder with the label name.

In addition, we examined a research paper titled "American Sign Language Recognition using Deep Learning and Computer Vision" as it aligns closely with the objectives of our project. The paper focused on collecting data from YouTube, featuring different artists performing the same piece of music in ASL. The diversity in their dataset, stemming from the inclusion of various artists, became an important insight for us. This realization emphasized the importance of curating a diverse dataset for our project, recognizing that a varied dataset contributes significantly to enhancing the accuracy of the model.

Another research paper that we referred to is “Sign Language Recognition Using the Fusion of Images and Hand Landmarks through Multi-headed Convolutional Neural Networks.” This paper helped us to lay out the concept of our project using hand landmarks - which is what our current model is based on. The paper introduced a novel approach involving using kernels to sharpen images before incorporating them into the model for training. This helped us connect our classroom learning with our project even though we didn’t end up doing this in our project. Additionally, the study utilized Google’s Mediapipe for hand landmark detection, forming the backbone of their convolutional neural network. Given time and technical constraints, our project focused on building a neural network using the hand landmark data, omitting the additional complexity introduced by the multi-headed structure.

Dataset Exploration:

Previous Model:

In the initial model, we curated a dataset comprising 15 common phrases, with 20 images captured for each phrase. Out of the 300 images, 240 images (80%) were used to train the model, and 60 images (20%) were used for validation.

Fig: Visualization of the raw dataset with label and image for each image

Since this model did not perform well with real-time images trained on that dataset, we swapped our dataset with publicly available data from Kaggle, focusing on the 24 alphabets (excluding J and Z) and 3 inputs (delete, space, nothing) with 20 images for each. We had 580 images belonging to 29 classes where every class represents a different alphabet. 464 images (80%) were used for training and 116 images (20%) were used for validation.

This dataset contained images of only one hand performing different alphabets and interpretations in ASL in different lighting and a variation of backgrounds. This dataset brought diversity to the training set and had the potential to enable a comprehensive evaluation of the model's performance.

Fig: Visualization of the (new) raw dataset with label and image for each image

Current Model:

The improved and our current model introduces a refined dataset structure, segregating data into two folders: data_keypoints and dist_data. The “data_keypoints” folder in our code stores coordinates of 21 landmarks for each alphabet in different CSV files, each file containing 306 records. Each CSV file represents a specific alphabet, with 21 columns corresponding to landmark coordinates. In total, we have landmark records of 24 alphabets excluding “J” and “Z” since these two alphabets cannot be detected via image. The “dist_keypoint” folder contains the distances between key points that were calculated and stored for specific point pairs, representing distinctive parts of the hand. Distances included ["dist_20_0", "dist_16_0", "dist_12_0",

"dist_8_0", "dist_4_0", "dist_20_16", "dist_16_12", "dist_12_8", "dist_8_4"] where "dist_20_0" means distance between point no 20 and point no 0. All the distances are stored in separate CSV files for each alphabet and the resulting CSV files in this folder feature 9 columns and 306 rows. For all the points that we are calculating distances between, each point represents a specific part of the hand. For example, in a 21-point model, point 0 represents the lower center of the palm, while other points represent specific joints of the fingers. The distance between point 20 and point 0 represents the distance from a specific finger joint to the center of the palm. These values between selected points can provide valuable information about the shape and orientation of the hand, which is used to recognize different hand gestures.

After storing all the raw data, we use the Isolation Forest algorithm to remove outliers from the distance data. The Isolation Forest Algorithm is used on the raw data of "dist_keypoints," where the algorithm is used to identify and eliminate outliers from the calculated distances, focusing on anomalies rather than typical data points.

	dist_20_0	dist_16_0	dist_12_0	dist_8_0	dist_4_0	dist_20_16	dist_16_12	dist_12_8	dist_8_4	label
0	49.024784	31.035207	49.443554	110.101010	129.158331	26.182262	26.409017	61.235759	19.586999	0
2	48.290193	35.370848	49.501759	99.344606	136.790507	27.421204	25.415429	50.800759	37.518808	0

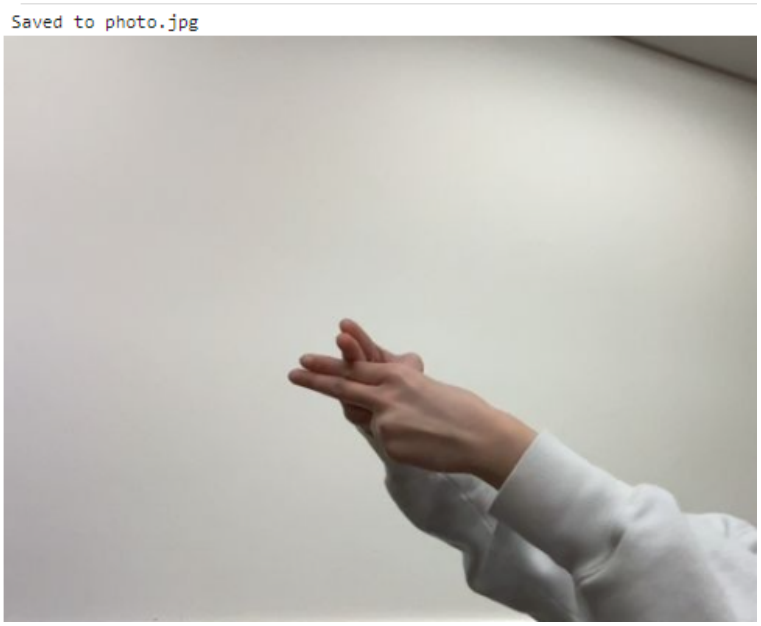
Fig: Snippet of the database after using Isolation Forest Algorithm on the raw dataset

The "explore_data.ipynb" script manages the intricate pre-processing tasks, storing the refined data in dist_data for subsequent model utilization. Now the dataset is ready to be fed into the model for training & validation.

User Inputs:

Previous Model:

The old model lets users use this model in 2 different methods. The first method is Image Processing, in which users can initiate ASL recognition by capturing a picture using their laptop's webcam. A JavaScript script facilitates webcam access, captures an image, and saves it as a JPEG/PNG file in a Colab environment. The saved image was then provided as input to the model for ASL prediction. Example of an image being captured and saved as a jpg file:



In addition, we also use a live video recognition model to support live ASL detection through a JavaScript script accessing the laptop's webcam. For each frame in the live video stream, the script converts the JavaScript object into an OpenCV image. Hand and ASL detection were performed on each frame, and the model updated the result for every image frame in the live video.

Current Model:

The new model streamlines user interactions by utilizing OpenCV's cv.VideoCapture(0) to access the laptop's webcam. This implementation allows users to engage in real-time ASL recognition seamlessly.

The cv.VideoCapture(0) function accesses the default camera (index 0), establishing a connection with the laptop's webcam. Each frame from the live video stream is processed in real time through the ASL recognition model. The model provides instantaneous feedback on hand gestures, enabling users to observe ASL translation as they perform signs. The model detects hand landmarks and bounding boxes - which are visualized on each frame. Recognized ASL gestures are converted into letters, forming a dynamic word input system, and are displayed in real-time on the video feed, providing immediate feedback to the user.

Users can use the “backspace” key to delete the last entered letter or space. The script terminates when the 'Esc' key is pressed.

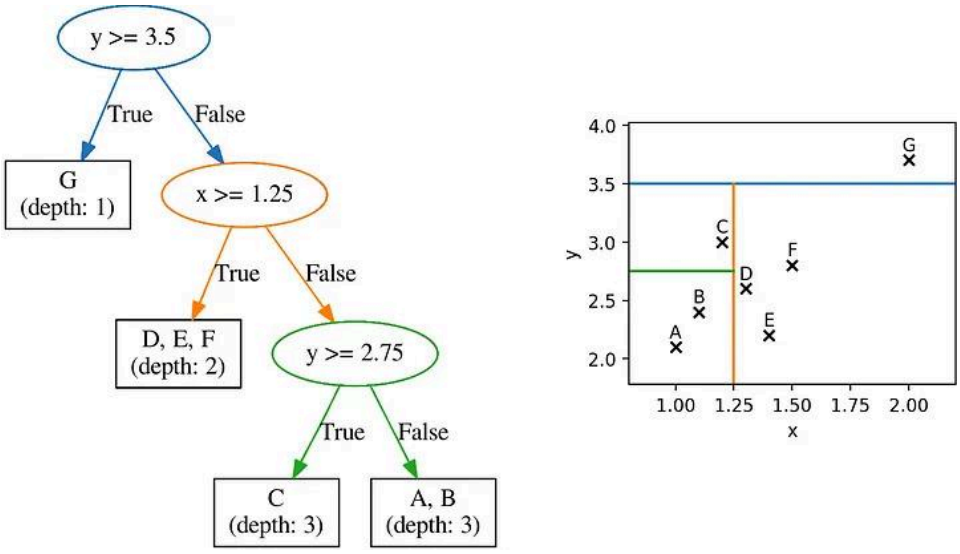
Methods

Mediapipe:

We used Google's Mediapipe for hand landmark detection in our project. Mediapipe is a powerful computer vision library that facilitates the extraction of intricate details from images and videos, specifically designed for tasks such as hand tracking. Using machine learning models, Mediapipe identifies and precisely locates 21 key landmarks on a hand, encompassing critical points like fingertips, joints, and the palm's center. Using these key landmarks helps us achieve better accuracy in ASL alphabet recognition and is more efficient than using images since image files can be very heavy to load. Moreover, calculating the distance between landmarks makes it more generalized so that when the hand is flipped or rotated the distance between the points remains the same.

Isolation Forest:

In addition to the hand recognition method, which is based on MediaPipe, we implemented another algorithm called Isolation Forest. The concept is similar to Random Forest, but in this case, we focus on abnormality detection. Since the MediaPipe object detection can sometimes incorrectly predict joints depending on the background, brightness, and other external factors, we attempt to mitigate this using Isolation Forest. Unlike other anomaly detection algorithms, which define ‘normal’ instances first and label anything else as outliers, Isolation Forest isolates anomalous data points from the beginning.



In the image above, we want to detect outliers from 7 points, each with x and y coordinates. We first split the nodes with random conditions like on the left-hand side, which has corresponding separating lines on the right. The process is repeated until we separate all outliers, which are G and C in this case. Then we take an average of the depth of each node over different trees, which

represents the ‘average depth’ of a node or an equivalent measure of ‘outlierness.’ The outliers tend to be separated earlier on, and thus, have lower ‘average depth,’ while other clustering points tend to have higher scores.

The two images below show histograms of 9 different Euclidean distances before and after applying the Isolation Forest algorithm. From the image, we can see that the distribution of the distance resembles that of a normal distribution after the transformation. Specifically, for dist_20_16 and dist_4_0, which are heavily skewed right and left respectively, become unimodal after applying the algorithm.

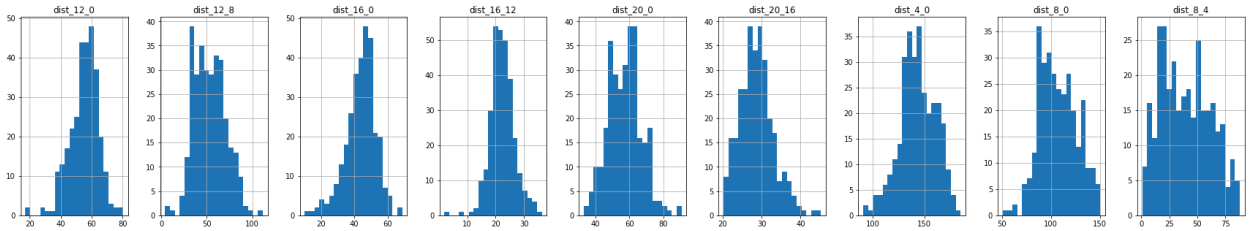


Fig: Distance between different key points for alphabet “A” **before** Isolation Forest Algorithm

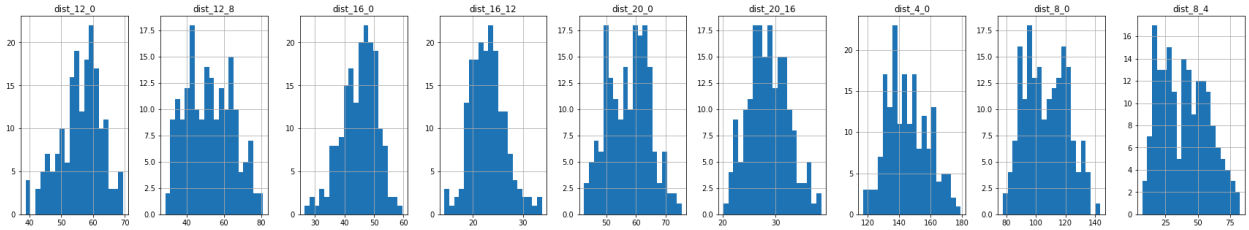


Fig: Distance between different key points for alphabet “A” **after** Isolation Forest Algorithm

GaussianNB

In order to predict whether the real-time hand sign belongs to any of the 23 alphabets, we carried out Gaussian Naïve Bayes (GassianNB) classifiers. The Naïve Bayes classifier is a supervised machine learning algorithm, which is used for classification tasks, like text classification. The algorithm is known to be based on Bayes’ Theorem, which can be represented as

$$P(y|x1, x2, x3..xN) = \frac{P(x1|y).P(x2|y).P(x3|y)...P(xN|y).P(y)}{P(x1).P(x2).P(x3)...P(xN)}$$

where X = x1, x2, x3, …, xN are list of independent predictors, y is the class label, and P(y|X) is the probability of label y given the predictors X. In our problem, we are trying to find the probability a hand sign belongs to any of the alphabet classes given each Euclidean distance pair. From that, we find the label with the highest probability and assign the sign to that class. The process is repeated for each frame the camera captures.

Results and evaluation:

Previous Model:

As mentioned above, we split the data into training and testing sets and trained them using the TensorFlow Sequential model. The dataset we are using comprises common ASL phrases that are constructed using images recorded from our computer webcam. Overall, the model performs well in the training and validation set, the orange and blue lines respectively. In the first image, the loss function declines as the number of epochs increases. Concurrently, accuracy increases as we train the dataset on more epochs.

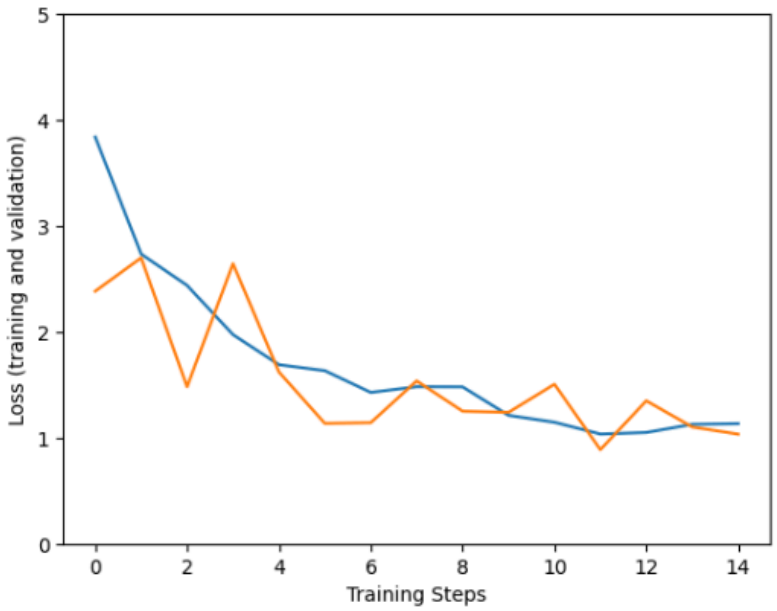


Fig: ** fill later**

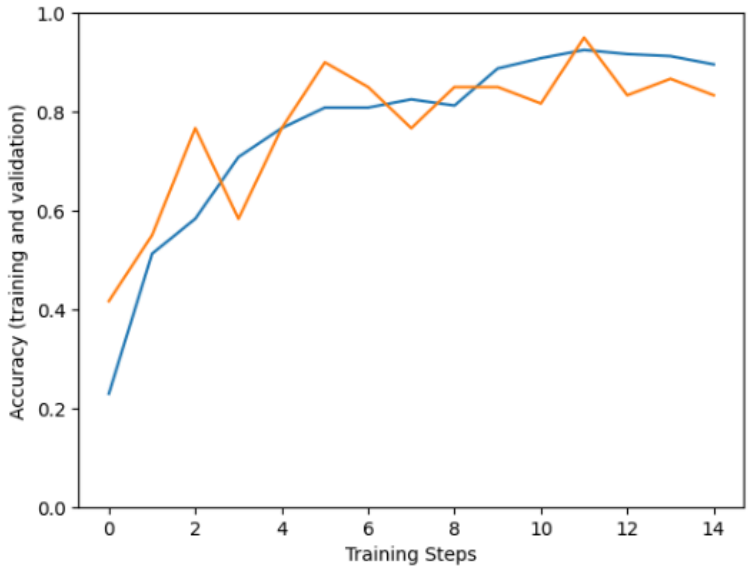


Fig: ** fill later**

The image below belongs to one of our test sets while training the model. In this case, the trained model has the correct prediction for the hand sign “congratulations” as the signer did it.

```
1/1 [==============================] - 0s 113ms/step
True label: congratulations
Predicted label: congratulations
```

Fig: Result of testing the model on an image from Validation dataset

However, when we tried to take pictures outside the dataset using the same webcam, the model failed to give the right prediction results. Here, the model predicted the label as “sorry” while we were doing the sign language “name.”

```
1/1 [==============================] - 0s 133ms/step
Predicted label: sorry
```

Fig: Original Label: “name”, predicted label by model: “sorry”

We tried to improve the prediction by cutting out the hand sign only when training and rotating the images in different directions. However, the model still performs not well on novel images outside the training and validation sets. Thus, we moved on to use a pre-built dataset online, consisting of 26 alphabet letters and 3 additional words. The result of the training is presented below

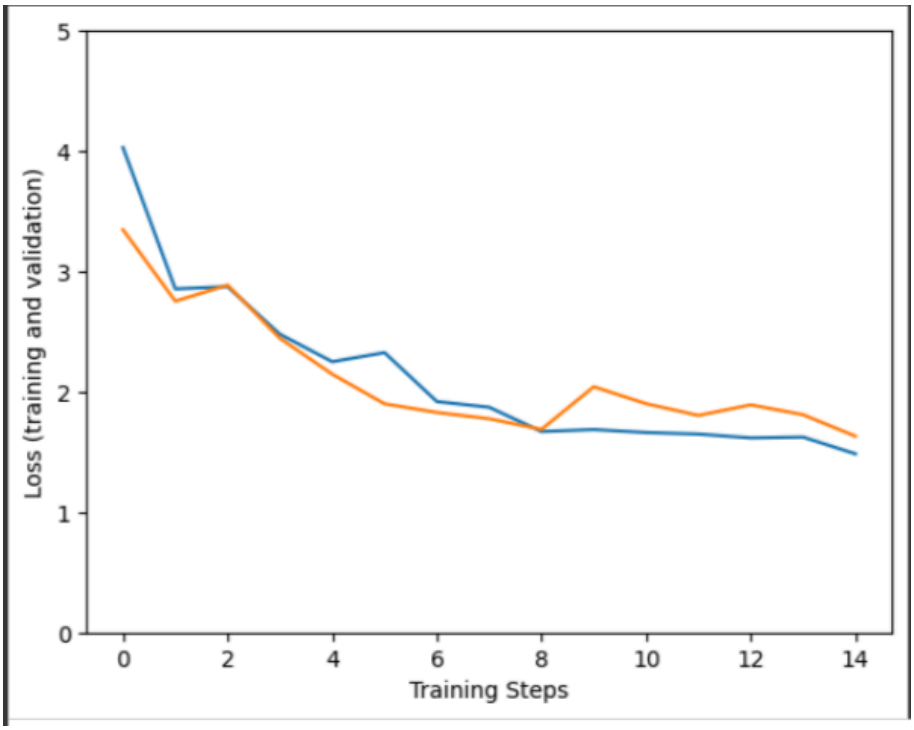
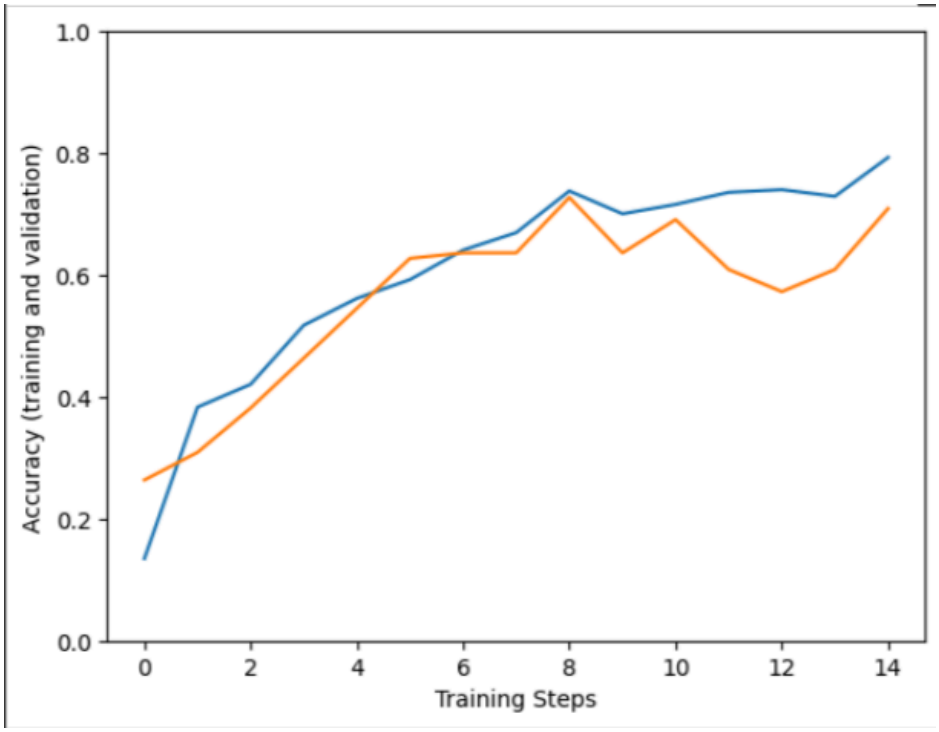


Fig: **fill later**



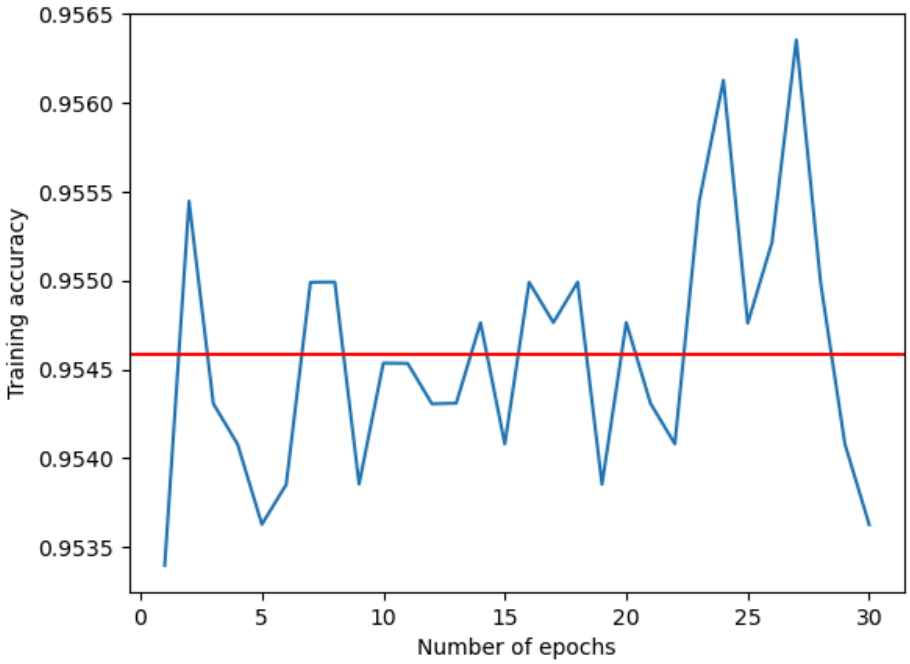
Similar to the previous dataset, as the number of epochs increases, the loss value decreases and the accuracy rises, which signals that the model performs well on the training data. Thus, we proceeded to add new images from our webcam to check the performance of the model on real-time data.

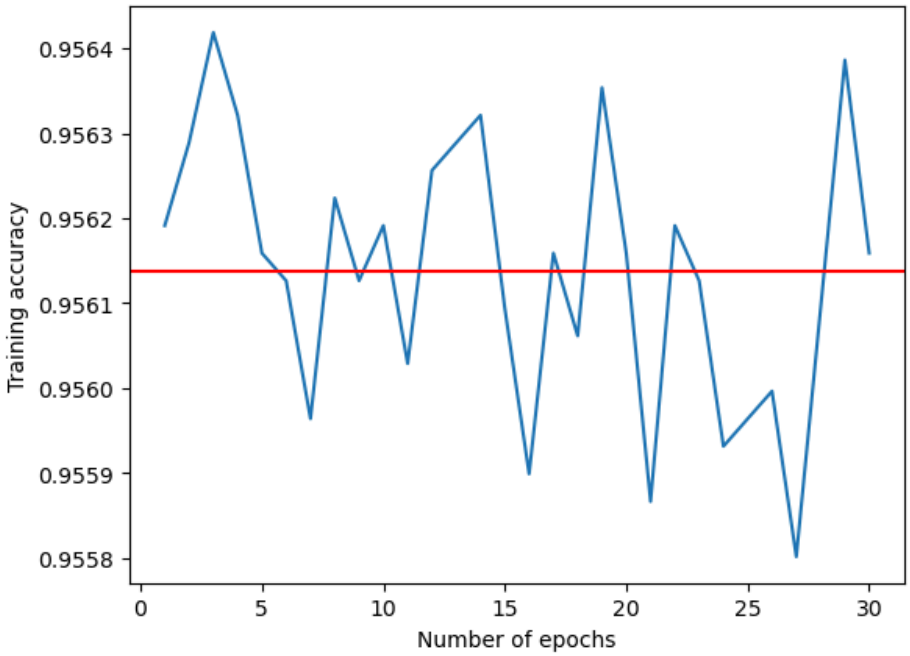
Fig: Original Label: “U”; Predicted label by the model: “Z”

However, even with a wide variety in the dataset, this model fails to give the right prediction results on real-time images. In the image above, the photos show a hand sign of the letter “U”, but was predicted as “Z.”

Current Model

Due to the setbacks of the previous model, we decided to try a different approach to processing the input images. Instead of using the original photos, we extracted the coordinates of each landmark or knuckle point from the hand using MediaPipe and used that as the input for the machine learning algorithm. In this case, as mentioned in the Methods section, we cleaned the dataset and eliminated abnormality using Isolation Forest and then put them into the Gaussian Naive Bayes model. This method yields a really consistent result in terms of accuracy for both the training and validation sets. In the two images below, the model has a consistent accuracy of 95% for both the two set types. When we run the model on real-time video, it gets a consistently accurate prediction for distinct letters like A, B, C, V, and W. However, for words with similar hand signs like H, U, and R, it gets mixed up between the letters.





Conclusion:

This project has delivered an ASL recognition and translation model that contributes to breaking down communication barriers and offers educational value to people who would like to learn more about ASL. While the model is not 100% accurate and there are areas to improve on, the model coupled with a user-friendly interface does a fairly great job in recognizing and interpreting ASL alphabets.

Throughout this project, we learned valuable insights into the intricacies of machine learning models, delving into the principles governing their behavior. While working with the database for this project - which took a significant amount of time while building this application- we got to explore different aspects of computer vision. This helped us gain a nuanced understanding of how computers perceive images and how we can manipulate the images to build a product and serve a purpose. Additionally, we also learned and used popular libraries such as Mediapipe, Tensorflow, and OpenCV. Recognizing the importance of datasets in training machine learning models was a fundamental learning point for us while working on this project.

So far we managed to build a real-time recognition application that can recognize basic alphabet letters. The model performs well on some distinct letters like A, B, C, I, and Y but gets mistaken between similar letters like M and N. Thus, some future goals for this project can be improving

the accuracy of the model by adding more data or augmenting the photos so that they are rotated in different directions. Furthermore, we hope to implement common ASL phrases similar to the original dataset we have built like ‘Hello’, ‘I love you’, etc. As part of the scope, we hope to turn this project into a web application or extension that can be widely used by people around the world.

References:

1. K. Bantupalli and Y. Xie. "American Sign Language Recognition using Deep Learning and Computer Vision." In Proceedings of the Conference on Neural Information Processing Systems (NeurIPS), 2018.

2. Refat Khan Pathan, Munmun Biswas, Suraiya Yasmin, Mayeen Uddin Khandaker, Mohammad Salman, and Ahmed A. F. Youssef. "Sign language recognition using the fusion of image and hand landmarks through multi-headed convolutional neural network." In Proceedings of the Conference on Neural Information Processing Systems (NeurIPS),2023.

TensorFlow Hub Tutorials

TensorFlow Hub Tutorials are a set of tutorials created by Google to help users get started on how to use the pre-trained models saved in the TensorFlow Hub repository. To get started on our project, we used the tutorial template 'Retraining an Image Classifier' to get started on our project. The tutorial consisted of:

a) Setting up imports on Google Colab, including an import of the TensorFlow Hub Repository
b) Setting up an imported model to be compiled and retrained
c) Using TensorFlow.keras to augment the data and divide the dataset into training and validation and also the creation of class labels.
d) Using Matplotlib to track the accuracy and loss over each epoch

What we added to this initial setup was:

a) visualization of random 6 images and classes from our dataset with labels so that we know what the dataset currently looks like.
b) A file organizer for a large dataset. To be able to use a dataset with over 1,000 images on Google Colab, we had to make a smaller randomized version of it.
c) More specific data augmentation: cropping images in a dataset by a specific amount to remove the blue bounding box on the datasets; normalizing the images in a dataset to black and white.
d) Javascript code to take a photo with Google Colab and save it for model testing purposes

e) Real-time video capture with javascript