5-1-2024

Portable ASL Translator Using Raspberry Pi

Brooklyn Kelly
University of Alabama in Huntsville

Portable ASL Translator Using Raspberry Pi

by

Brooklyn Kelly

An Honors Capstone

submitted in partial fulfillment of the requirements

for the Honors Diploma

to

The Honors College

of

The University of Alabama in Huntsville

Spring 2024

Honors Capstone Project Director: Kevin Preston

Brooklyn Kelly 05/01/24

Student Date

Kevin Preston 1-May-2024

Project Director Date

Letha Etzkorn Digitally signed by Letha Etzkorn
Date: 2024.05.02 09:54:57 -05'00'

Department Chair Date

Sean Lane Digitally signed by Sean M Lane
Date: 2024.05.02 16:43:00 -05'00'

Honors College Dean Date

Honors College

Frank Franz Hall

+1 (256) 824-6450 (voice)

+1 (256) 824-7339 (fax)

honors@uah.edu

Honors Thesis Copyright Permission

This form must be signed by the student and submitted with the final manuscript.

In presenting this thesis in partial fulfillment of the requirements for Honors Diploma or Certificate from The University of Alabama in Huntsville, I agree that the Library of this University shall make it freely available for inspection. I further agree that permission for extensive copying for scholarly purposes may be granted by my advisor or, in his/her absence, by the Chair of the Department, Director of the Program, or the Dean of the Honors College. It is also understood that due recognition shall be given to me and to The University of Alabama in Huntsville in any scholarly use which may be made of any material in this thesis.

Brooklyn Kelly

Student Name (printed)

Student Signature

05/01/2024

Date

Table of Contents

Table of Contents **3**
Introduction **4**
Importance of the Project **4**
Collecting Images **5**
Summary of the Project Process **6**
Methodology **6**
1. Data Collection: 6
Figure 1: “Collecting Images Code” 7
2. Dataset Creation: 7
Figure 2: “Preprocessing Image Data” 8
3. Model Training: 8
Figure 3: “Range Generator” 9
4. Real-Time Gesture Recognition: 9
Figure 4: “Translated ‘A’ “ 11
Figure 5: “Translated ‘B’” 11
Figure 6: “Translated ‘C’” 12
Performance Self-Assessment and Lessons Learned **12**
Raspberry Pi Conversion **13**
Comparing Specs **14**
Figure 7: “Time Difference Formula” 15
Figure 8: “Laptop Training Example” 15
Figure 9: “Raspberry Pi Training Example” 15
Recommendations For Future Work **16**
Conclusions **17**
References **18**

Introduction

The American Sign Language (ASL) Translator project addresses the communication barriers faced by individuals with hearing impairments. This is done by developing a real-time system that translates ASL alphabet gestures into text. Traditional methods of communication for individuals with hearing impairments, such as lip reading and written communication, can be challenging and inconvenient. The portable translator could also be used for the hearing impaired to communicate with those unfamiliar with ASL. The ASL Translator project seeks to provide an intuitive and accessible means of communication for individuals who use ASL by leveraging machine learning and computer vision techniques.

Importance of the Project

The ASL Translator project significantly enhances accessibility and inclusivity for individuals with hearing impairments. Existing ASL recognition systems may lack real-time capabilities or demand expensive hardware. This project aims to surmount such limitations by employing machine learning techniques and optimizing for resource-constrained environments like the Raspberry Pi 4B. Its goal is to develop a system running on a low-cost, portable platform, thereby increasing access to ASL translation technology.

Collecting Images

I attempted to implement a loop in a previous iteration of the image collection script. It cycled through a fixed number of classes and gathered a predetermined number of images per class. However, this initial approach proved inadequate. Especially when faced with a substantial dataset comprising 87,000 images for analysis. The limitations became evident as the script struggled to handle the volume of data efficiently. It became clear that a more scalable solution was necessary to manage such a large dataset effectively.

One significant issue was the inefficiency caused by creating directories for each class, even if the dataset size was not fully utilized. This leads to unnecessary overhead and clutter. Additionally, the program was not flexible enough to add new classes dynamically during runtime, limiting its scalability. Processing many frames in real-time significantly slows down the program, especially on devices with limited computational resources. This slowdown impacts the user experience and hinders the collection process, mainly when dealing with a high frame rate or large image sizes.

The script can be optimized to address these issues by implementing dynamic class creation and more efficient frame processing instead of predefining the number of classes. The program prompts the user to input the class label during runtime, allowing for the creation of new classes as needed. Additionally, implementing batch processing or frame subsampling techniques reduces processing time and improves program efficiency. Especially when dealing with large datasets. These optimizations would

enhance the program's flexibility, scalability, and performance. Making the image collection process more efficient and user-friendly.

Summary of the Project Process

Hand gesture recognition is a technology that enables computers to interpret and understand human hand movements. It has various applications, such as human-computer interaction, sign language recognition, and virtual reality systems. This paper presents a hand gesture recognition system implemented using Python and several libraries such as OpenCV, MediaPipe, TensorFlow, and scikit-learn. The system comprises several components: data collection, feature extraction, model training, and real-time gesture recognition.

Methodology

1. **Data Collection**:

   Data collection is crucial in developing a robust hand-gesture recognition system. To begin this process, the system sets up a dictionary structure to store the image data collected. As shown in Figure 1, the camera is then initialized using OpenCV's VideoCapture function to capture frames for each hand gesture class specified by the user. A data collection loop guides the process, prompting

user interaction to initiate data capture. Upon receiving the command, the system captures and saves a predetermined number of frames for each class, organizing them into subdirectories within the primary data directory. The system then releases the camera and closes OpenCV windows. This concludes the process and ensures the computer resources are managed efficiently.

```
img = cam.read()[1]
img = cv2.flip(img, 1)
imgHSV = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
dst = cv2.calcBackProject([imgHSV], [0, 1], hist, [0, 180, 0, 256], 1)
```

Figure 1: “Collecting Images Code”

2. **Dataset Creation**:

The system uses the MediaPipe Hands library to create a dataset of hand gesture images. It extracts hand landmark data by configuring the MediaPipe Hands model with specific parameters to ensure accurate landmark detection, as shown in Figure 2. It then navigates through each class directory within a designated dataset, processing individual image files to detect hand landmarks. The system reads and converts each image to RGB (red, green, and blue) format before utilizing the MediaPipe Hands model to identify hand landmarks. If the system successfully identifies the landmarks, it extracts and normalizes the landmark coordinates, associates them with respective class labels, and stores the data in lists. Finally, it serializes the extracted data and labels into a pickle

file, creating a structured repository of hand landmark data for further analysis and model training.

```
blur = cv2.GaussianBlur(dst, (11,11), 0)
blur = cv2.medianBlur(blur, 15)
thresh = cv2.threshold(blur,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)[1]
thresh = cv2.merge((thresh,thresh,thresh))
thresh = cv2.cvtColor(thresh, cv2.COLOR_BGR2GRAY)
thresh = thresh[y:y+h, x:x+w]
```

Figure 2: “Preprocessing Image Data”

3. **Model Training**:

Training a hand gesture recognition model begins by creating a Convolutional Neural Network (CNN) architecture to extract hierarchical features from input images. This architecture is tailored to suit the dataset's requirements and consists of annotated hand gesture images. The dataset is integrated into the training pipeline using TensorFlow. Before the images are fed into the model, they undergo preprocessing steps using OpenCV to ensure they are suitable for analysis. Data augmentation techniques improve the model's condition and general performance. This technique is implemented through TensorFlow's ImageDataGenerator, as shown in Figure 3, which generates augmented images efficiently. The CNN architecture uses TensorFlow's Keras API, which comprises convolutional, pooling, and dense layers. These layers are designed to capture intricate patterns within the input data. Optimization parameters and evaluation

metrics are specified, which are vital in refining the model parameters during training iterations. During each epoch, the model undergoes parameter adjustments, guided by the minimization of the defined loss function, aimed at improving performance and learning representations inherent in the data. After completing training epochs, the model undergoes post-training evaluation, which assesses on a separate test dataset. This validation step corroborates the trained model's efficacy in accurately recognizing hand gestures, providing insights into its generalization capabilities and readiness for real-world deployment.

```
datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)
datagen.fit(train_images)
```

Figure 3: “Range Generator”

4. **Real-Time Gesture Recognition**:

The system demonstrates real-time hand gesture recognition through a combination of a pre-trained model and the MediaPipe Hands library, alongside

live camera input. As illustrated in Figures 4, 5, and 6, the system loads a pre-trained machine-learning model explicitly designed for hand gesture recognition. Upon initialization of the camera, real-time video input is captured. The MediaPipe Hands library identifies landmarks on the hand for each frame captured. Subsequently, the system processes each frame from the camera feed, detecting hand landmarks and extracting hand gesture data. Predicted gestures, such as those depicted in the screenshots (e.g., A, B, C), are then mapped to corresponding characters and overlaid on the frame for visualization. Annotated frames are displayed using OpenCV. The system operates continuously until the user exits, ensuring proper resource cleanup and system closure.

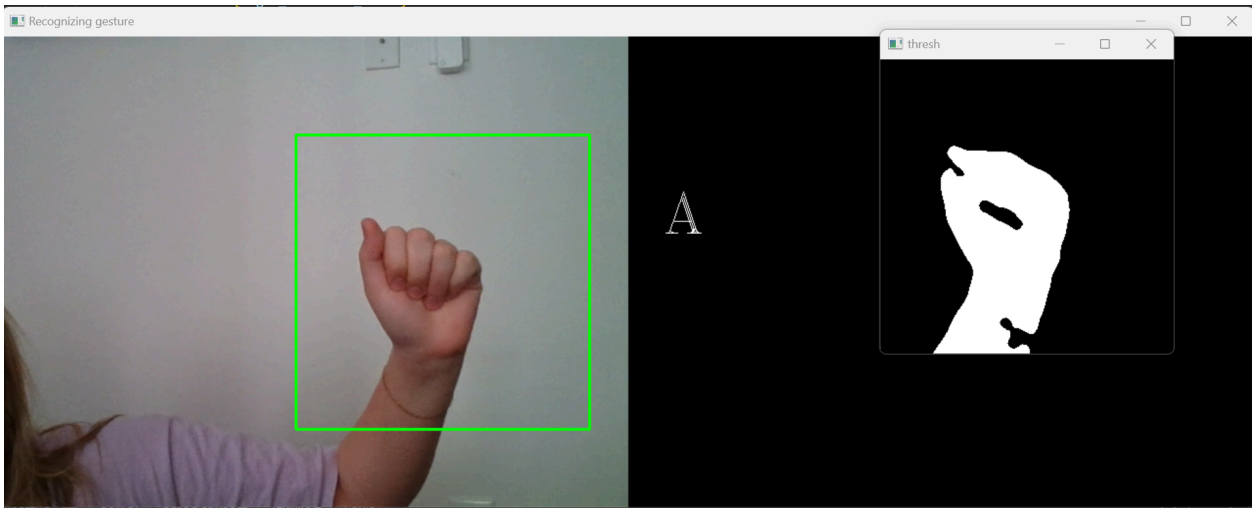

Figure 4: “Translated ‘A’ “

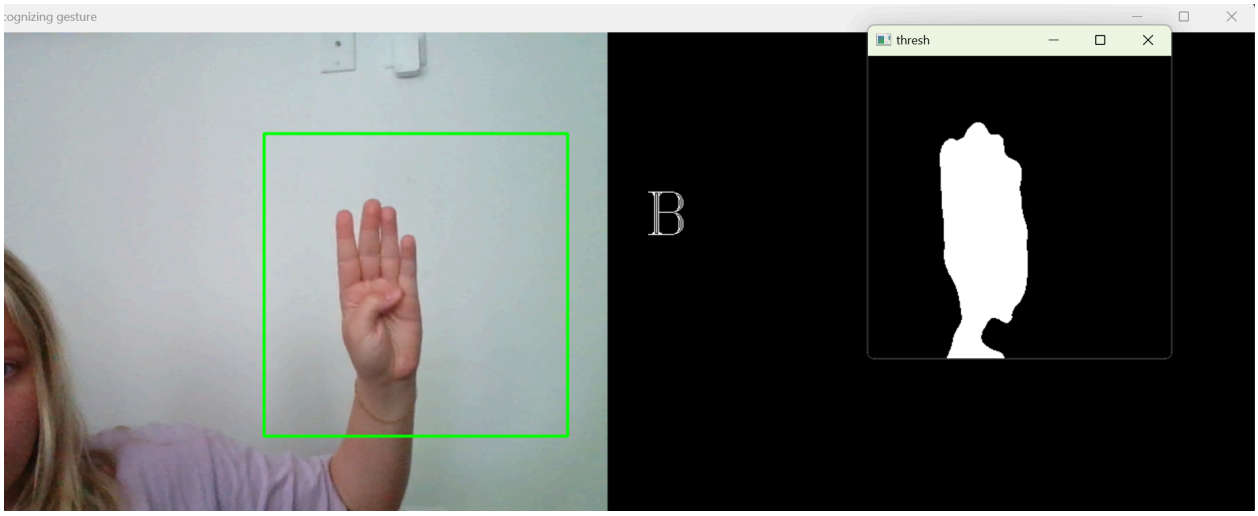

Figure 5: “Translated ‘B’”

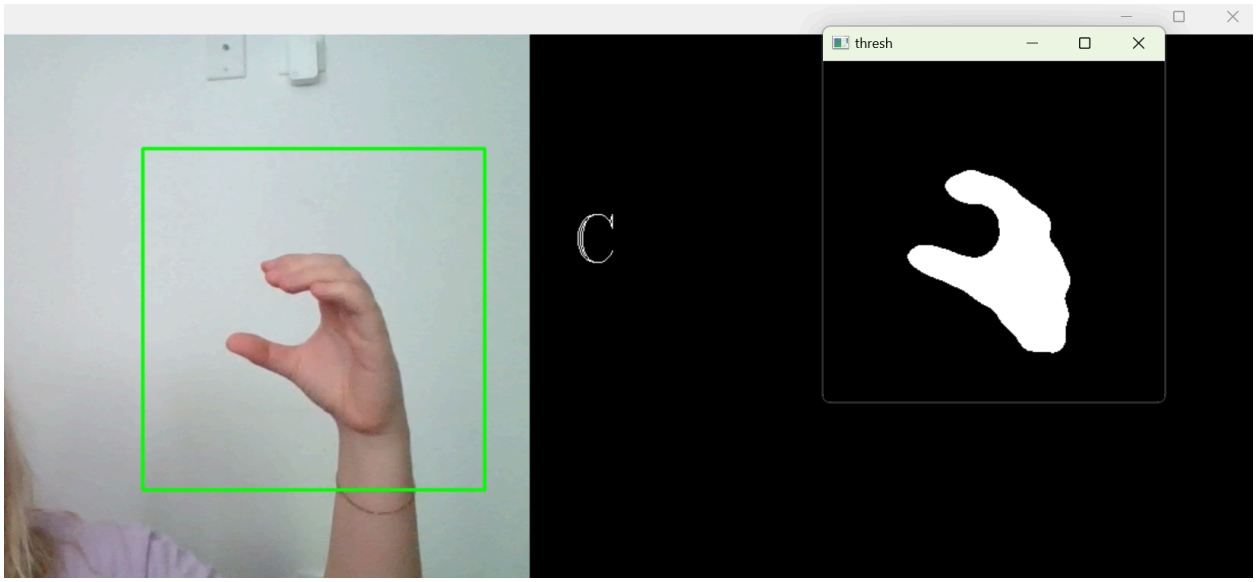

Figure 6: “Translated ‘C’”

Performance Self-Assessment and Lessons Learned

The ASL Translator system achieved accurate real-time recognition of ASL alphabet gestures, meeting or surpassing expectations. However, optimizing it for the Raspberry Pi 4B's limited computational resources posed significant challenges. Transitioning from standard computing platforms to the Raspberry Pi introduced processing power and memory constraints, algorithmic fine-tuning, and model optimization for efficient operation. Additionally, ensuring variations in lighting, hand orientations, and background clutter was challenging but crucial for reliable performance across diverse real-world scenarios. Techniques like data augmentation played a pivotal role in overcoming these challenges. Transitioning from the Kaggle ASL Alphabet dataset (Nagaraj, Akash) to a custom dataset creation approach significantly enhanced

efficiency and flexibility. Performance and adaptability are improved by tailoring data collection to the system's needs. Looking ahead, lessons learned will inform future iterations, aiming to enhance further accessibility for individuals with hearing impairments and promote greater communication and understanding in society.

Raspberry Pi Conversion

Moving the hand gesture recognition system from a conventional computer to a Raspberry Pi presents several challenges and benefits. One of the main difficulties lies in the hardware limitations of the Raspberry Pi compared to a typical computer. The Raspberry Pi has less processing power, memory, and storage capacity, which affects the system's performance, especially in real-time image processing tasks. Additionally, the compatibility of external peripherals, such as cameras, posed challenges, requiring adjustments or alternative hardware solutions. Furthermore, optimizing the code and algorithms to run efficiently on the Raspberry Pi's ARM architecture and limited resources is crucial for maintaining acceptable performance.

Despite these challenges, there are several benefits to moving the system to a Raspberry Pi. Firstly, the compact size and low power consumption of the Raspberry Pi make it suitable for applications, enabling hand gesture recognition systems deployment in various environments and devices with minimal power requirements. Additionally, the affordability of the Raspberry Pi makes it an accessible platform for hobbyists, educators, and developers to experiment with and implement gesture recognition

projects without significant financial investment. Finally, using the Raspberry Pi will provide access to many resources, tutorials, and libraries, facilitating development and troubleshooting efforts. For optimal performance, the Raspberry Pi requires a 64-bit operating system tailored for ARM architecture, ensuring compatibility and efficient utilization of resources. While shifting the system to a Raspberry Pi presents challenges, the potential benefits in terms of versatility, affordability, and accessibility make it a compelling platform for deploying hand gesture recognition solutions.

Comparing Specs

Comparing the performance of an application on a laptop with an AMD Ryzen (Microsoft R Edition) and a Raspberry Pi 4B highlights a substantial difference. This discrepancy stems from variations in CPU power, memory, storage, and graphics capabilities between the two platforms. Notably, the laptop's Ryzen processor boasts superior processing power and multitasking capabilities compared to the ARM-based CPU in the Pi 4B. Based on processing time, the comparison reveals that the application completes its task significantly faster on the laptop. Specifically, while the laptop takes less than 50 seconds (Figure 8) to process 812 files, the Raspberry Pi 4B requires approximately 360 seconds (6 minutes) (Figure 9) for the same workload.

To quantify this performance difference, we calculate the performance improvement factor for the laptop compared to the Raspberry Pi 4B:

$$P = \frac{\text{Time taken on Pi 4B}}{\text{Time taken on laptop}} = \frac{360 \text{ seconds}}{50 \text{ seconds}} \approx 7.2$$

Figure 7: “Time Difference Formula”

This indicates that the application runs approximately 7.2 times faster on the laptop than on the Raspberry Pi 4B. While this comparison provides valuable insights, it is essential to consider that projections for future hardware iterations, such as a hypothetical Pi 5, would depend on various factors, including advancements in hardware specifications, software optimizations, and workload characteristics. Below are the epochs shown to the user during training, which depict the accuracy, losses, remaining files, and remaining time left for that training batch.

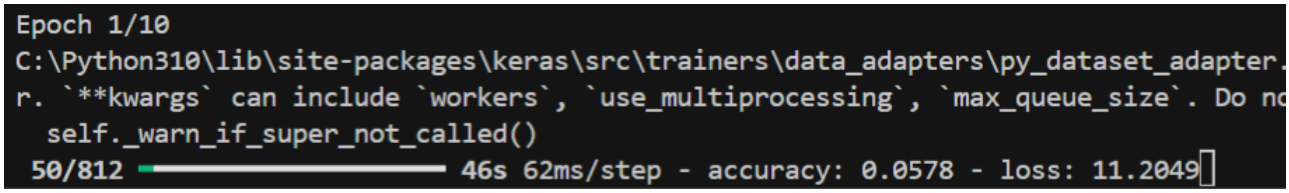

Figure 8: “Laptop Training Example”

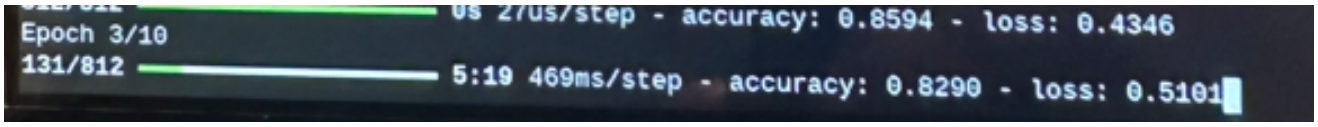

Figure 9: “Raspberry Pi Training Example”

Recommendations For Future Work

There are several ways to focus on improving the ASL Translator system. One way to make the system more versatile and applicable to a broader range of communication scenarios is to expand the vocabulary of recognized gestures beyond the ASL alphabet. Expanding the library of hand gestures is possible with this current model, but it is a lengthy process. Therefore, a more user-friendly development process is a crucial goal for future projects.

Another critical goal is fine-tuning the model's hand histogram, which depicts the lighting and shapes of hand gestures. Fine-tuning is important because the lighting is very sensitive. Creating something that can automatically update to the best version of the histogram can make this project more user-friendly.

Efforts could also be made to improve the user interaction experience with the system. This could involve implementing intuitive user interfaces, incorporating feedback mechanisms, and exploring ways to add to the library to cater to diverse user preferences and needs.

A promising idea for future expansion is the integration of video frame comparison techniques. By analyzing sequences of video frames, the system could detect dynamic hand gestures and motions, allowing for the recognition of gestures beyond stagnant poses. This would enable the system to recognize complex gestures, signs, and expressions, significantly expanding the system's vocabulary and applicability.

Conclusions

In conclusion, the ASL Translator project represents a groundbreaking initiative in accessibility technology, leveraging state-of-the-art machine learning and computer vision techniques to empower individuals with hearing impairments. By developing a real-time system capable of recognizing and translating ASL alphabet gestures, the project significantly enhances accessibility and inclusivity for all individuals, regardless of their hearing abilities. Originally based on the ASL Alphabet dataset from Kaggle, the project has evolved to incorporate a more efficient approach to creating custom datasets for hand gesture recognition. This shift allows for greater flexibility and adaptability in data collection, leading to improved performance and scalability.

As the project advances, it holds the potential to revolutionize communication for individuals with hearing impairments and foster greater understanding and inclusivity in society. Future iterations of the ASL Translator system could explore further enhancements in gesture recognition accuracy, user interaction experiences, and dynamic gesture recognition capabilities. With ongoing technological advancements and collaborative efforts, the ASL Translator project stands poised to make a meaningful impact on the lives of individuals with hearing impairments, creating a more inclusive and accessible world for everyone.

References

Nagaraj, Akash. 2018. "ASL Alphabet." Available at: https://www.kaggle.com/grassknoted/aslalphabet_akash. DOI: 10.34740/KAGGLE/DSV/29550.

Computer Vision Eng. "Sign Language Detector Python." GitHub, https://github.com/computervisioneng/sign-language-detector-python.

“Sign Language” GitHub, EvilPort2/Sign-Language: A very simple CNN project. (github.com)

UserBenchmark CPU. UserBenchmark. https://cpu.userbenchmark.com/.