Farhan Rahman Khan & Trishir Kumar Singh

COMP-3301
Steph McIntyre

Project Report

4th December 2023

GESTOVISION

INTRODUCTION

GestoVision is an innovative application designed for empowering sign language communication with the power of Machine Learning. The app accurately recognizes ASL (American Sign Language) hand gestures in real-time from a live video feed, and with a smart and neat overlay displays it over the hand in the video itself.

ASL allows communication with the hearing-impaired. Even though ASL helps with inclusion and communication, not everybody is proficient in this gesture-based communication method or feels the need to understand it. The motivation behind this project and aim is to bridge this communication gap by creating a machine learning software that recognizes ASL, and in the process of doing so, ASL alphabet recognition was recognized as the crucial first step towards breaking down this barrier. GestoVision serves as a bridge, aimed towards facilitating communication in various fields such as education, sign-to-text conversion, healthcare, and more.

In the initial stages of the project, various solutions were explored to address the challenges of ASL gesture recognition. Multiple methodologies were considered, each with its distinct advantages and drawbacks. Among the solutions investigated were traditional computer vision approaches, deep learning models, and existing frameworks tailored for hand detection. In the quest for an optimal solution, the capabilities of cutting-edge technologies such as MediaPipe (a versatile library for hand detection) emerged as the backbone of the project.

The implemented software takes live video feed as input, employs hand detection using MediaPipe, and identifies the specific ASL alphabet. We used Scikit-Learn to train a machine learning model on a Kaggle dataset which is saved and used to detect the alphabets at high speed.

TOPIC DESCRIPTION

The project focuses on solving the challenges associated with ASL gesture recognition, as an elementary step towards bridging the communication gap with the hearing-impaired. Key elements in the topic description include:

Image Processing: The video feed is usually going to be from a webcam, which more often than not is of questionable quality and framerates. This raises a lot of concerns regarding image quality, and the possibility of blurry/unfocussed hands. Again, the feed might also have noise and in need of color correction. Some frames might also be overshadowed or have too high exposure. While finding a solution, all of these must be kept in mind.

Hand Detection: Determining an effective method for detecting hands in a dynamic video environment can be challenging. Detecting a hand is a big concern to begin with, not to mention there might be more than one hand in the video. The project would have to either accommodate multiple hands or be restricted to just one.

Continuous Tracking: Even if detecting a hand in a still frame is possible, developing a mechanism for continuous tracking of the detected hand throughout the video feed after the initial detection is a big concern. If the app is always detecting a hand from the entire frame, it is going to be very computationally expensive. For efficiency and seamless hand detection in the final stage of the application, a method to track the hand after the initial detection needs to be implemented.

Data Acquisition: Obtaining relevant data for training, which considers all the different permutations of backgrounds, lighting conditions, skin tones etc. is necessary. Training a machine learning model has its advantages but it also has a big disadvantage, for it to accurately understand any new scenario, it must be trained on that specific one. So, when detecting hand gestures, the model needs to be trained on various scenarios, like different lighting, gestures with people of different skin tones, different hand types, or even different hand gesture accents.

Data Representation: One of the most important parts of creating a machine learning model is choosing the right data representation. Choosing an appropriate representation for the acquired data(photos), proved to be challenging at first. If you train the model on an entire frame, the machine will have to read every pixel, every frame and it would be insanely computationally heavy which will possibly cause a lot of damage to the learning algorithm as well. Consequently, coming up with good data representation was imperative to solving the problem.

SOLUTION DESCRIPTION

The solution involves the utilization of MediaPipe for hand recognition, Kaggle datasets supplemented by custom data for training, Random Forest Classifier from Scikit-Learn library for the machine learning algorithm, and a combination of image processing techniques for optimal performance.

MediaPipe for Hand Recognition: Leveraging the capabilities of MediaPipe, a robust library for hand detection, the process of identifying hands in real-time was streamlined. After going through various methods of object detection and hand detection, MediaPipe was a game changer. MediaPipe accurately detects hand in multiple scenarios, even in different lighting and background. This library is also very computationally efficient which leads to our code being more streamlined and faster. Additionally, MediaPipe addresses the concern of continuous tracking in the video feed. It uses a very smart algorithm to follow the hand after an initial hand detection method, and if the hand is no longer detected in the video, the hand detection algorithm is initiated again followed by the tracking algorithm. The switching between a hand detection algorithm and a tracking algorithm significantly improves performance and makes it so that the app is not unnecessarily doing the search algorithm every frame.

Kaggle Dataset and Custom Data: We incorporated a Kaggle dataset along with custom images to ensure a diverse and comprehensive training set. A Kaggle dataset was found with 3000 images for each letter with different lighting conditions, backgrounds and skin tones. A few hundred images from GestoVision creators for each alphabet were also used to ensure diversity.

MediaPipe Hand Landmarks to Represent the Data: MediaPipe library also has a method to create landmarks (points at each joint/junction of the hand) for the hands which can be used as an appropriate data representation of the hand positions. This ensures we are not using thousands of pixels as data to train the model and only have a few (x, y) coordinates, 21 points to be exact. The coordinates found on the training images are then translated to the lower left part of a graph plot while keeping relative positions intact. Then when the model is trained, they are compared to the coordinates of the hand in the live video, also translated to the lower left part, to accurately make a guess of the alphabet being shown.

Scikit-learn Random Forest Classifier for Training the Model: In the training phase, the power of machine learning was harnessed through the implementation of the Random Forest Classifier from the Scikit-Learn library. This algorithm, known for its robustness and versatility, played a pivotal role in training the model to recognize ASL alphabets. Leveraging the Kaggle dataset, we employed RandomForest to build a predictive model capable of efficiently categorizing hand gestures in real-time. The decision-making capabilities of the RandomForest algorithm, coupled with its ability to handle complex data patterns, contributed to the accuracy and speed essential for our ASL recognition framework. The model would not only provide the ASL alphabet that was recognized, but also a percentage which says how close the gesture is to the gesture used to train the data.

Image Processing: Implementing a sharpening kernel for enhancing hand features and blurring for noise reduction. These two image processing methods were what could fit into the app while maintaining a careful balance between performance and accuracy without compromising the real-time nature of the application. A lot of other methods were experimented with but cut off due to performance issues. Besides, MediaPipe already deals with most pre-processing concerns.

EXPERIMENTS

Various machine learning methods were considered for the project. TensorFlow was initially planned to be used, but due to the complexity of its implementation, a much simpler library Scikit-Learn was employed. And for hand detection, various methods for object recognition were researched, such as– R-CNN Model to detect objects, however, towards the end, the MediaPipe library seemed to be the most efficient.

During the initial stages of the project, various image processing methods were experimented with, such as– Sobel-Edge Detection for sharpening, Histogram Stretching and Gamma Correction. A lot of the photos generated during data gathering ended up being blurry. We decided to use Sobel for increasing the detail. Again, the webcam used during data gathering produced very poor results. Histogram Stretching was just one of the steps in our quest for image processing. Finally, the Gamma Correction technique was particularly valuable when dealing with diverse lighting conditions, ultimately contributing to a more robust and accurate machine learning model.

While various image processing methods were explored, a trade-off was established to maintain real-time performance. The use of a sharpening kernel and blurring for noise reduction was found to be efficient, considering that MediaPipe already addresses several image processing concerns. Additionally, manipulating the lighting conditions in the dataset inhibited the predictive performance of the model.

REFERENCES

1. B., A., Krishi, K., M., M., Daaniyaal, M., & H. S., A. (2020). Hand gesture recognition using machine learning algorithms. *Computer Science and Information Technologies, 1*(3), 116-120. doi:https://doi.org/10.11591/csit.v1i3.p116-120
2. Google. (n.d.). *MediaPipe | google for developers*. https://developers.google.com/mediapipe/
3. Sharma, A., Mittal, A., Singh, S., & Awatramani, V. (2020). Hand gesture recognition using image processing and feature extraction techniques. *Procedia Computer Science*, *173*, 181–190. https://doi.org/10.1016/j.procs.2020.06.022

CONTRIBUTIONS

Trishir & Farhan: Equal Contribution

1. ASL Alphabet Dataset: https://www.kaggle.com/dsv/29550