Amish Singh
Northwood High School
Michelle Vela

Real-Time ASL Detection

Bridging the communication gap between ASL speakers using machine learning.

Abstract

The problem I aimed to solve was improving communication between ASL and English speakers through a real-time machine learning model. To achieve this, I used a 30,000-image dataset from Kaggle, which provided diversity and realism. The model leveraged MediaPipe Hands to detect hand landmarks, aiding in training a neural network. Additionally, a computer vision element was introduced using a camera, enabling real-time ASL letter translation.

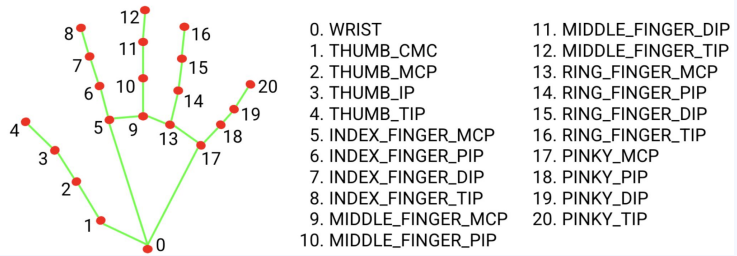
Our procedures involved preprocessing images to a uniform style, extracting landmarks with MediaPipe Hands, and encoding the data using One-Hot Encoding. The data was split into training, validation, and test sets to prevent overfitting and ensure optimal performance. The model was trained using TensorFlow/Keras and evaluated through various metrics and visualized using libraries like matplotlib. The CV2 library was used to capture images from a webcam and feed them into the model for predictions.

The model achieved 86% accuracy during testing, with performance improving over successive epochs. It initially struggled with similar signs and background noise, but these issues were mitigated through data augmentation and background filtering. Despite these challenges, the model demonstrated consistent performance across various conditions.

In conclusion, the solution was validated, effectively improving ASL sign detection accuracy. Key factors in its success were the use of diverse data and MediaPipe Hands's landmark extraction capabilities. Future work could focus on adding more classes and enhancing background removal techniques to further boost accuracy and reliability.

Introduction (Background Research)

Prior to starting my project, I conducted research to find a diverse dataset suitable for training my ASL detection model. Initially, I found a dataset containing 2,000 images, but to improve accuracy and confidence, I needed more diverse data. I later discovered a dataset with 27,000 images, offering greater variation in backgrounds and lighting. However, this introduced a challenge: the model began misidentifying background elements as significant features. To address this, I researched and chose MediaPipe, a framework from Google that focuses on extracting 21 key hand landmarks (x, y, z coordinates), effectively isolating the hand from the background. This allowed the model to learn the essential features for accurate predictions without interference from background noise. I also searched for other models, such as Convolutional Neural Networks (CNNs), but selected MediaPipe Hands due to its ease of use, efficiency, and suitability for real-time applications. The project criteria focused on accuracy, efficiency, and real-time performance, while constraints included ensuring data diversity and minimizing computational resources. MediaPipe Hands's ability to process data with less training while maintaining high accuracy made it an ideal choice for meeting these criteria and overcoming challenges.



A map of the 21 key hand landmarks that MediaPipe Hands detects in images.

Criteria

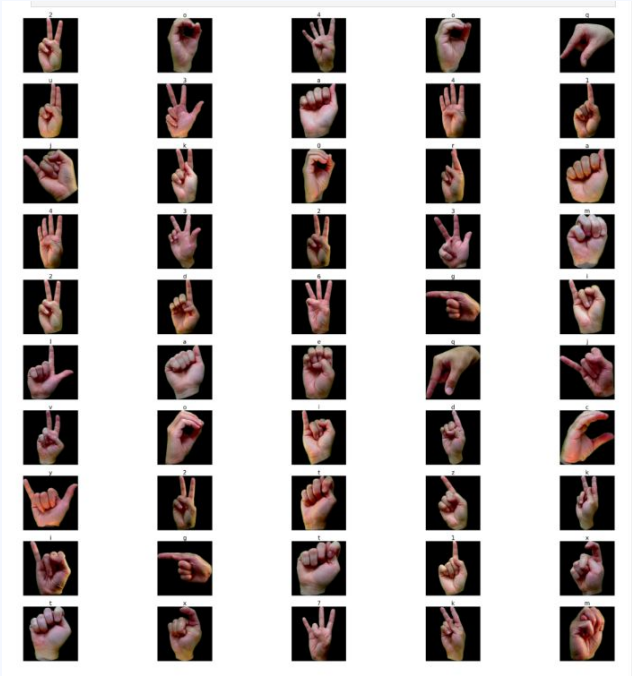
- Test Accuracy of 80% or higher
- The ability to process and predict signs from a live camera feed
- Efficiency of the model requiring minimal computational resources

Constraints

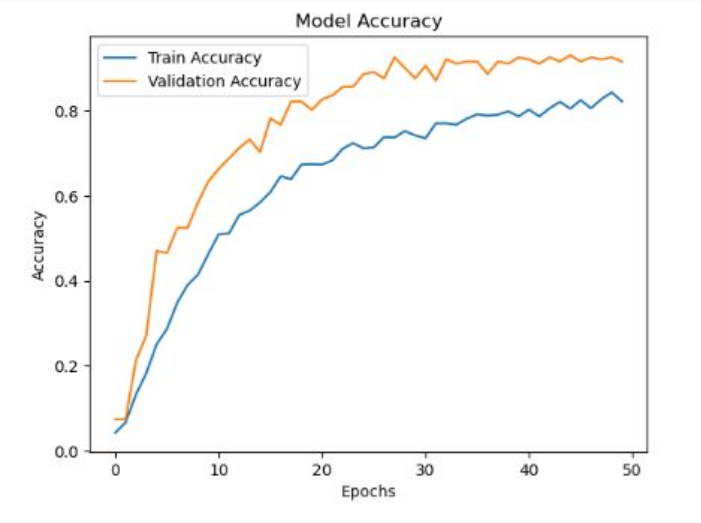
- Model must be able to run on standard devices
- Made using free open-source tools
- Uses limited memory and computational resources on devices

The Prototype

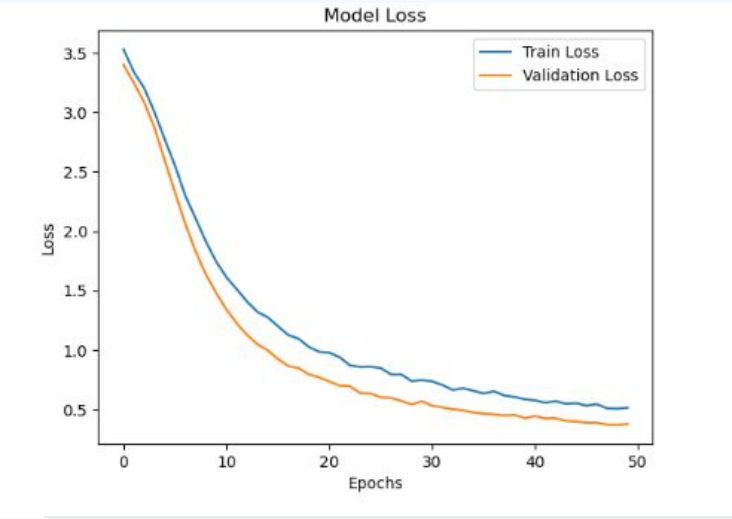
For my prototype model, I initially went with creating a simple model and dataset to determine if my code was functional. The dataset that I ran my initial model on consisted of only two thousand images and lacked diversity as well as realism, as shown on the left. In practical applications the model would not be looking at signs in front of a black screen; there would be various backgrounds, which is why I used another dataset to train the model. While the test accuracy is higher for the initial model than the secondary model, this is because the data is much simpler. When this model was individually tested by me, the accuracy in reality was much less because the model kept getting confused by the background noise. The accuracy and loss graphs also show the less accurate results when testing due to the lack of data.



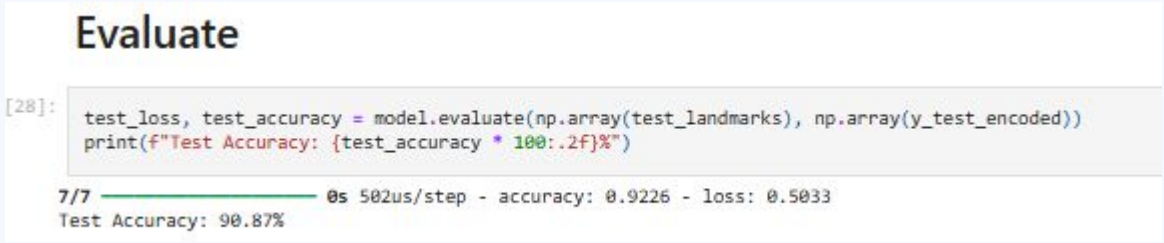
A random selection of 50 images from our initial dataset



The initial model's accuracy and loss graphed



The initial model's test accuracy as a percent



Materials

Hardware

- Webcam
- Computer

Software

- Python 3
- Anaconda
- Jupyter Notebook
- Github

Libraries

- TensorFlow/Keras
- MediaPipe
- OpenCV
- Matplotlib
- Numpy
- Scikit-learn

Datasets

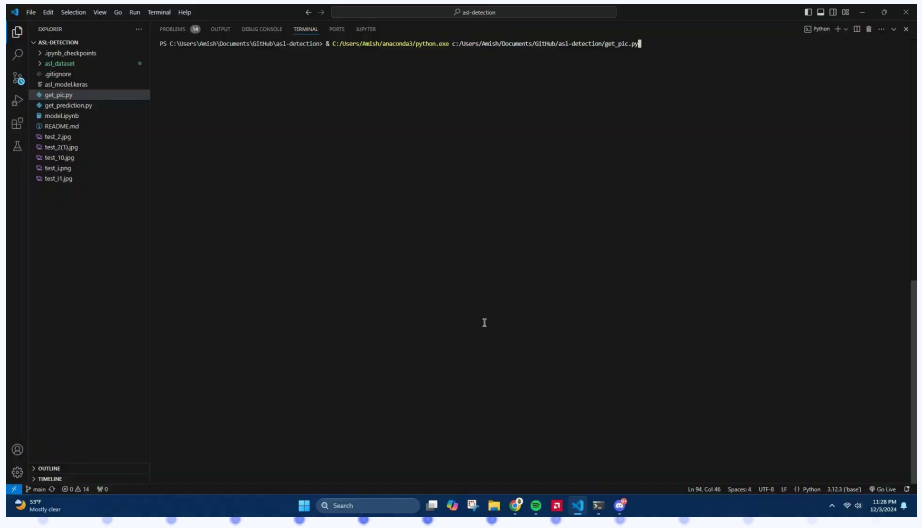
- Kaggle
  - https://www.kaggle.com/datasets/lexset/synthetic-asl-alphabet/data?select=Train_Alphabet
  - https://www.kaggle.com/datasets/ayuraj/asl-dataset/data

Procedure

Find and Upload Data (Images) to GitHub Repository	Locate a dataset containing labeled ASL images (e.g., signs for letters A, B, C, 0, 1, 2, etc.)
Map Each Image to Its Label (e.g., A, B, C)	Organize the dataset by associating each image with its corresponding label (e.g., an image of the sign for 'A' will be labeled 'A').
Preprocess Data	Resize all images to a uniform size (e.g., 224x224 pixels) to ensure consistency and convert all images to RGB format
Split Data into Training, Validation, and Test Sets	**Training Set (70%)**: Used for training the model. **Validation Set (15%)**: Used to finetune the model **Test Set (15%)**: Used to evaluate the model's performance after training
Separate the Images and Labels into Features (X) and Targets (Y)	**X (features)**: The image data (paths or pixel values). **Y (targets)**: The corresponding labels (e.g., 'A', 'B', 'C'), which are what the model is trying to predict.
One-Hot Encode Labels	Convert the labels (e.g., 'A', 'B', 'C') into binary using one-hot encoding, making it easier for the model to process multi-class classification.
Use MediaPipe Hands to Extract Landmarks	For each image, use MediaPipe Hands to extract landmarks (21 key points with 3D coordinates) from the hand sign images. These landmarks will be the input features used to train the model.
Remove Background Noise	Apply techniques to remove background noise from the images or landmarks to focus only on the hand and improve prediction accuracy.
Train the Model	Use the preprocessed training data (images/landmarks) to train the model. Validate the model during training using the validation data.
Visualize the Model's Performance	Plot the accuracy and loss curves to visualize how the model's performance changes over time during training and validation.

Revised Solution and Prototype/Model

During the creation of the initial model, I noticed a lack of diversity, realism, and quantity in the images. At that time there were only around two thousand five hundred images, which creates an underfed model that prevents it from accurately predicting the images. Furthermore, the dataset previously used had an all-black background and a lack of diversity in the images. To fix these issues, I looked for a new dataset to add onto the ones already previously used. The one I found on Kaggle made by Lexset contained around 27 thousand diverse images and in different backgrounds, allowing for more realism. The diverse backgrounds, however, caused another issue with MediaPipe Hands as the landmark detection began misinterpreting figures in the background as key points. In order to fix this, I implemented another function that loops through the images and prevents MediaPipe from recognizing the background. Another feature that I added was the implementation of computer vision into this project. By using the CV2 library to open a local camera, capture a picture, and input it into the model, I created a real-time ASL detection system. This allows for dynamic predictions directly from a live camera feed and sets the model up for real-world practical applications.

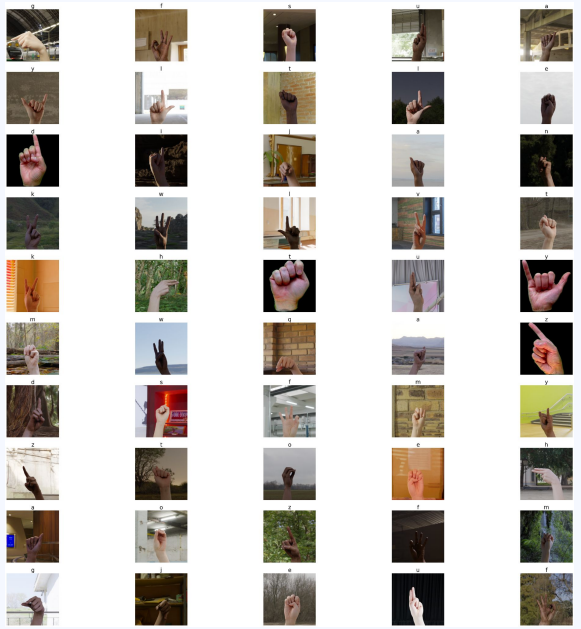


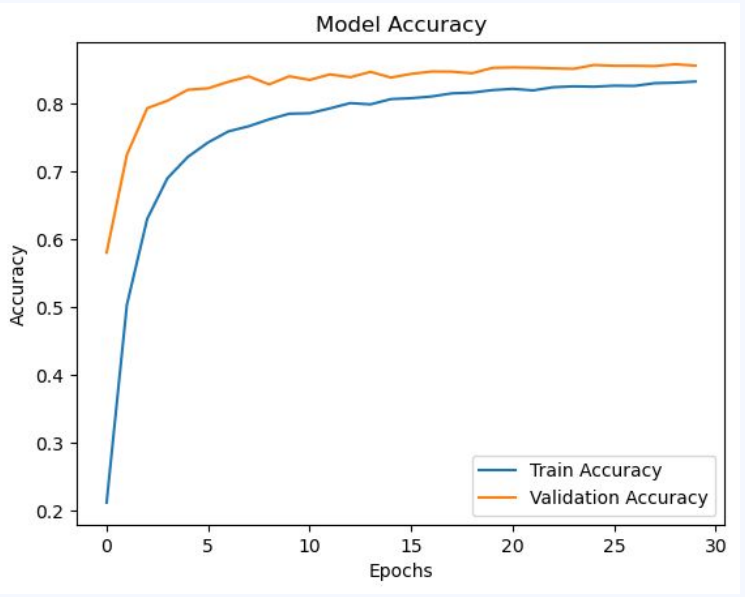
Demonstration of real-time ASL sign detection using computer vision, accurately recognizing the letter 'B'

Results - Data/Observations

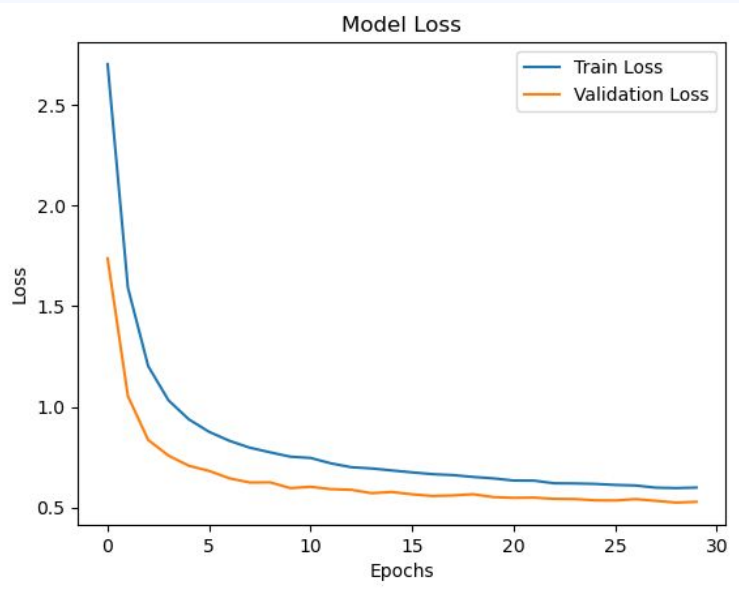
As I ran the model using the matplotlib library, I created the graphs on the right side of the screen to track the model's accuracy and loss. The model loss shows a numerical value representing the amount of error the model made compared to the actual result. This value should generally trend down as the model is trained and tested more. Model accuracy checks for the percentage of when the model correctly predicts the result. This value should trend upward as the model is trained and tested more. The model also outputted percentage showing the “Test_Accuracy” of the model, showing the percentage of times the model is right. My model showed a test accuracy of 86% which is great for a small scale model with limited resources and data. Another piece of information the model showed in the loss and accuracy as each epoch was trained. As you can see, the overall loss goes down while the accuracy trends upwards.

A random selection of 50 images from our secondary dataset

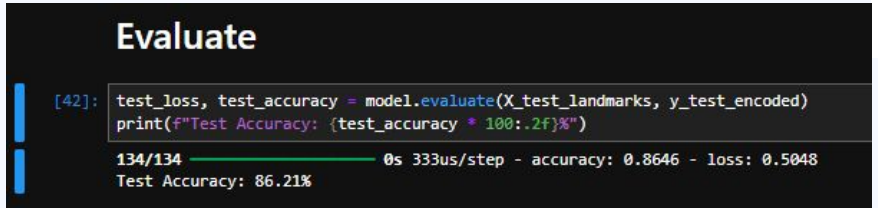




The secondary model’s accuracy and loss graphed



The secondary model’s test accuracy as a percent



Discussion

The data from the model displayed in the results slide shows that while the accuracy of the model is technically 86%, when taking real-time pictures through computer vision, the accuracy significantly drops due to the fact that everyone has different ways they position their hands when making these signs, and with only about 900 images per letter, it's impossible to get them all. An example of this is the letter k, which is very similar to the letter v. Depending on how you position your hand, the model will decide to predict one over the other.

Yes, my project turned out as expected, with a few challenges that ultimately enhanced its performance. Initially, the model struggled with limited data and uniform backgrounds, which impacted accuracy and generalization. After incorporating a more diverse dataset and implementing background filtering, the model's performance significantly improved. The use of MediaPipe for landmark detection proved effective, and integrating live camera input through OpenCV allowed for real-time predictions, which was a key goal. While some challenges arose with background interference and landmark misinterpretation, addressing these issues resulted in a more robust and accurate model. Overall, the final model met expectations, predicting ASL signs in various environments with an accuracy rate that corresponded to the work and data used.

Conclusion

The solution in the end that solved my problem was creating a computer vision-based model that uses a camera to take real-time pictures of ASL letters and convert them, allowing us to bridge the gap between ASL speakers and speakers of other languages. While at this time it is not a practical solution to spell out each word by hand, this model provides a valuable stepping stone into the future where I can add to our dataset and model and provide users the ability to now translate entire words and sentences in one clean swoop.

Some next steps I could take would be to find possible ways to solve the issue of inaccurate predictions. A way I could do this would be to find larger and more varying datasets or use different neural network models, such as CNN. However, when taking into account constraints such as time it takes to process the image, data available, and simply my own experience in the field, these solutions are not possible at this time but instead something I could slowly work towards in the future.

Further research could explore the impact of lighting and hand positioning on model accuracy. Variations in light intensity, shadows, and angles might cause the model to misinterpret gestures, highlighting the need for robust preprocessing techniques. Additionally, testing the model in real-world conditions, such as low-light environments or with users wearing gloves, could identify limitations and areas for improvement to enhance performance and usability.

Reflection

While the specific challenges and solutions in this project are tailored to the current dataset, the methods used to address them—such as retraining the model with diverse data—can be applied to other machine learning tasks. By using different datasets, the same approach can be adapted and applied to improve the performance of various machine learning models.

My next steps towards researching this problem would be to figure out ways to make the model more accurate by researching new computer vision models such as convoluted neural networks, recurrent neural networks, and more. This takes priority over other aspects of expanding the model into taking in more complex signs, such as words, since even if I found a database containing all the information, without an accurate model it would be essentially useless in real-world applications.

If I were to start this project from the beginning, some things I would've done differently include focusing on collecting a more diverse and larger dataset from the start instead of having to crunch and find one in the end. This would have helped in training a more accurate model by avoiding issues related to limited data diversity. Finally, I would have managed my time better in order to help test different models and be able to determine which one was the most effective and accurate. This also would have allowed me to spend less time on data preprocessing, as other models would not have the same problem as this one does.

References

01 Linear regression: Loss. (n.d.). Google for Developers. https://developers.google.com/machine-learning/crash-course/linear-regression/loss

02 Find open datasets and machine learning projects | Kaggle. (n.d.). https://www.kaggle.com/datasets/ayuraj/asl-dataset/data

03 Synthetic ASL Alphabet. (2022, June 17). https://www.kaggle.com/datasets/lexset/synthetic-asl-alphabet/data?select=Train_Alphabet

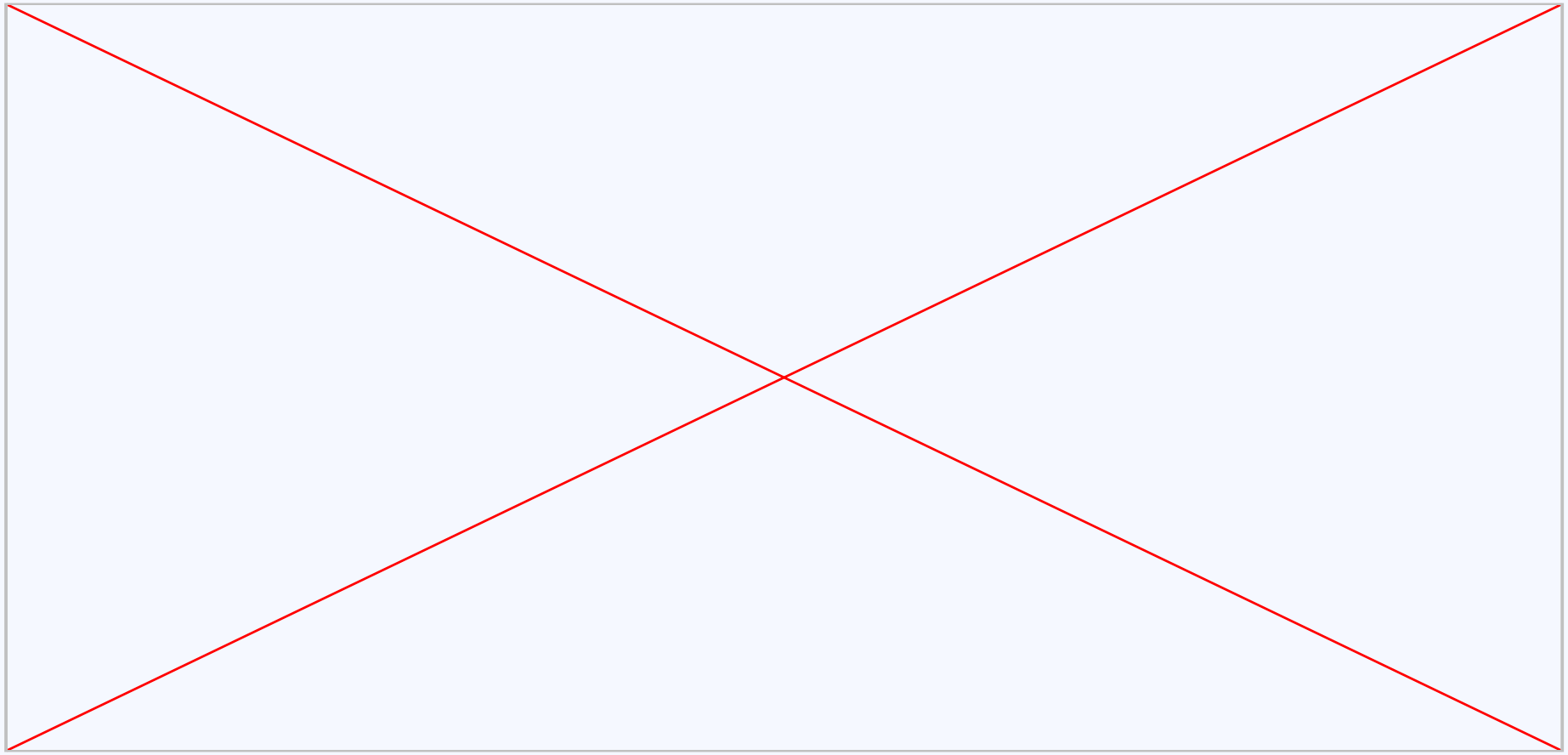
04 Using Matplotlib — Matplotlib 3.9.3 documentation. (n.d.). https://matplotlib.org/stable/users/index

05 GeeksforGeeks. (2024, November 2). One hot encoding in machine learning. GeeksforGeeks. https://www.geeksforgeeks.org/ml-one-hot-encoding/

Acknowledgements

I would like to sincerely thank my parents for their support and for providing the resources needed to complete this project. Their encouragement and belief in me made a significant difference throughout this journey. I would also like to acknowledge the open-source community for their invaluable contributions, including datasets and tools, which were essential in bringing this project to life.

Video



https://drive.google.com/file/d/1-Yz9pzQmuA2Vt5f7O2MGiyLfyaZkwij2/view?resourcekey

Logbook 1

Date	What was accomplished?	Additional Notes
9-4	Choose a topic	The problem I am attempting to solve is the communication gap between ASL speakers and other languages. I will solve this by creating a machine learning model that translates ASL signs in real time.
9-12	Completed Background Research	I researched different ways to create neural networks and different machine learning models. I decided on using MediaPipe over other models, such as CNN, due to it being easier and more beginner-friendly. I also found the initial dataset.

Logbook 2

9-15	Started work on the initial model	Started uploading and matching the files to their corresponding labels as well as downloading all the libraries and dependencies needed.
9-22	Completed preprocessing images and splitting data	Created the function to preprocess images into a uniform style and color. Split the images into training, validation, and testing datasets.
10-1	Prepared the data and extracted key hand landmarks.	Used one-hot encoding to prepare the data for the computer to handle. Then we used MediaPipe to extract and store the 21 key hand landmarks from the images.

Logbook 3

10-11	Created the neural network	Created the neural network from the initial dataset using TensorFlow/Keras.
10-15	Visualized initial model performance.	Fixed all bugs from the initial model and trained the model using the initial dataset consisting of 2700 images. Test accuracy was around 90.87%.
10-18	Completed the materials and procedure slides	Added materials used as well as procedures I used to train the initial model. This would also be the procedure I would use to train the secondary model.

Logbook 4

10-24	Found and trained the model with a larger dataset	I found another dataset to use for my secondary model that would contain more realistic and diverse images. With this, my accuracy went down significantly, and thus I hypothesized that the new background present in the images was causing the problem. The test accuracy using the initial model but with the larger dataset was around 50%.
11-3	Implemented a function to hide the background from MediaPipe	This was done to prevent confusion when finding key hand landmarks. Following this implementation, the model accuracy jumped to 86%.

Logbook 5

11-14	Created a real-time camera feed to translate signs	Used the CV2 library to open the default computer webcam, capture an image, and run it through the dataset.
11-20	Processed the results	Plotted graphs and made observations using the results from the secondary testing. Created the slides.
11-28	Completed the conclusion, discussion, and reflection slides	
12-1	Completed the slides	
12-3	Submitted the project	