Gesture Recognition and Background Manipulation in Live Camera.

Group Members - Pavan Kumar Jonnadula (pj1098), Ramprasad Kokkula (rk1668), Aakanksha Padmanabhan(al7275), Vaishnavi Patil (vp2156)

Project overview

Abstract:

This project delves into the realm of computer vision with a focus on image processing, feature extraction, and object recognition, centering on the understanding of gestures and background changes in images. Using existing libraries like the cvzone handTracking module, key gesture points will be extracted. The exploration extends to various machine learning frameworks for testing classifier models, with the selection of the most effective one. OpenCV is employed for image processing tasks, including background manipulation and segmentation, while PyTorch and existing CNN models are fine-tuned for segmentation. The project's overarching goal is to combine gesture recognition with background changes, offering users an interactive experience.

Learning Objectives:

In our project, we want to explore the concepts of computer vision, focusing on image processing, feature extraction, and recognizing objects. Our main focus is on understanding gestures and changing backgrounds in images. We'll dive into Convolutional Neural Networks (CNNs), to become skilled at and using models for accurate gesture recognition.

Using existing libraries such as the mediapipe hand Landmark Detection module, we will extract key points of gestures. We will learn various machine learning frameworks and test different classifier models to compare and select the most effective one. We will use PyTorch and existing CNN models for segmentation and fine-tune them to meet our requirements.

Approach and Technologies used:

OpenCV: We will use OpenCV for various image processing tasks, including background manipulation and segmentation. We are using existing libraries like mediapipe hand Landmark Detection module to extract the key points of the gesture.

Machine Learning Frameworks: We will be trying some existing classifier models to train and test, comparing the results of each classifier. Then we will choose the classifier with the best result.

PyTorch: We are using existing PyTorch CNN models to implement segmentation and fine-tune them to our specific requirements which will expedite our progress.

Technical Implementation:

1. **Gesture Recognition**

   In the initial phase of our gesture recognition system, we focused on data preprocessing to lay a solid foundation for subsequent stages. The primary objective during this phase was to extract and process relevant information from gesture data, enabling efficient and accurate analysis.

Data Preprocessing:

1. **Keypoint Extraction:**

   The cornerstone of our preprocessing approach involved the extraction of 21 key points from the gesture data. Key Points serve as pivotal landmarks that capture the intricate details of hand movements and configurations. This selection was deliberate, aiming to encapsulate the essential elements for effective gesture recognition.

2. **Data Representation:**

   To streamline further analysis, the extracted key points were organized and stored in a structured JSON format. This format facilitates seamless data handling and retrieval, laying the groundwork for subsequent stages of the gesture recognition pipeline.

MediaPipe's Hand Detection (Algorithm for keypoint generation):

Google's MediaPipe is a state-of-the-art framework designed for the extraction of intricate hand and body movements. Leveraging machine learning and computer vision techniques, MediaPipe precisely identifies and locates 21 key points on a given hand, forming a comprehensive representation of its configuration.

Gesture Prediction with K-Nearest Neighbors (KNN) and Support Vector Classification (SVC)

Initially, our gesture recognition system employed the K-Nearest Neighbors (KNN) algorithm for predicting gestures based on the extracted key points. However, during the initial stages, it became evident that the performance of KNN was not achieving the desired accuracy. In response to this, we pursued alternative approaches to enhance the predictive capabilities of our system.

Challenges with KNN:

The initial choice of KNN, while a widely used algorithm, encountered challenges in classification speed while classifying gestures, particularly when faced with complex hand configurations. The need for a more robust solution prompted us to explore alternative models.

Transition to Support Vector Classification (SVC):

To address the accuracy concerns and the speed, our system underwent a transition to Support Vector Classification (SVC). SVC is a machine learning algorithm well-suited for classification tasks, offering a powerful framework for discerning intricate patterns within the data.

Advantages of SVC:

The decision to adopt SVC has the ability to handle complex decision boundaries and adapt to diverse patterns of data. Through proper training, the SVC model learns to differentiate between various gestures, optimizing accuracy and increasing the overall performance of our gesture recognition system.

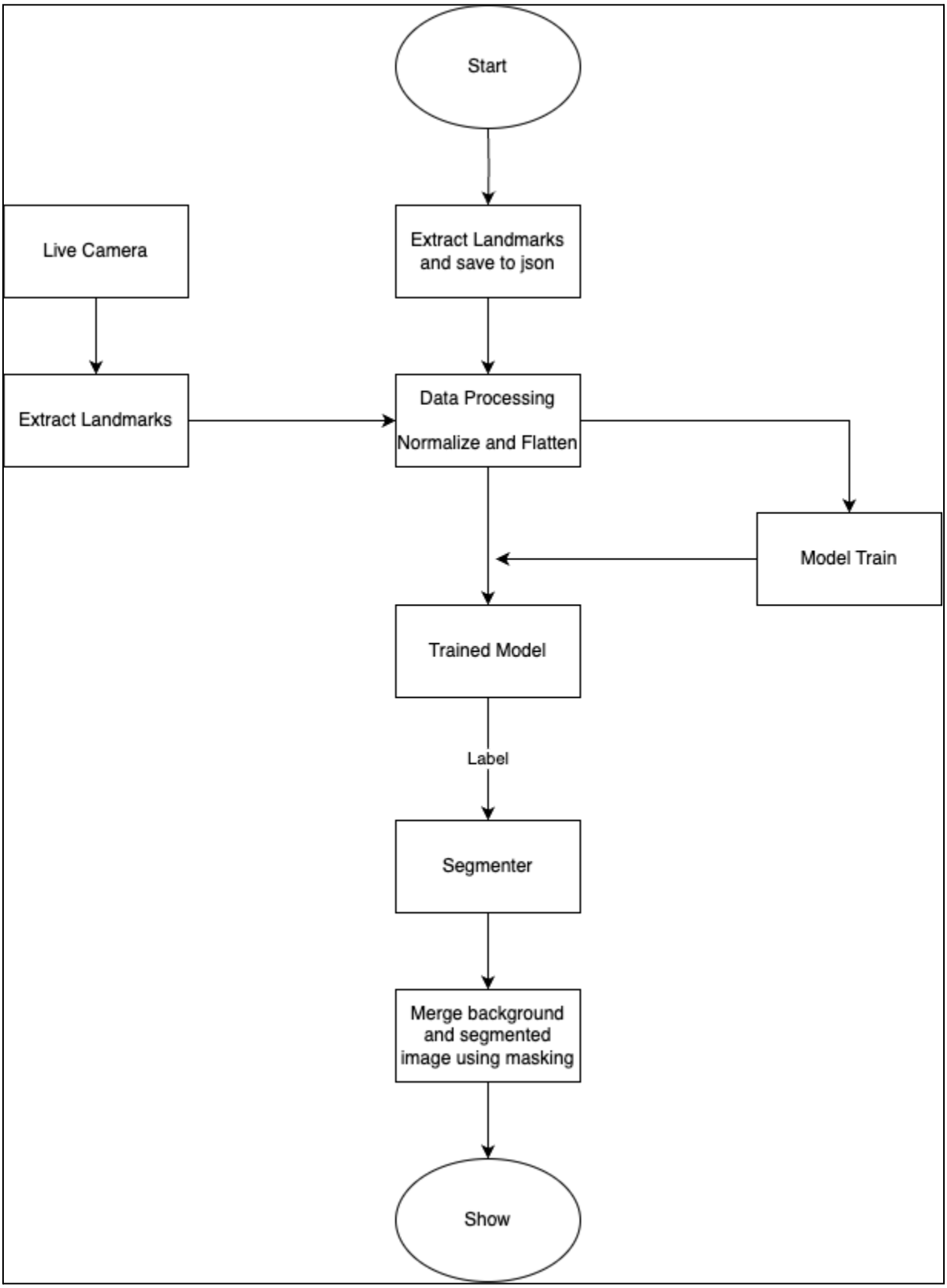
The current accuracy of approximately 90% underscores the success of transitioning from K-Nearest Neighbors (KNN) to Support Vector Classification (SVC). Future considerations may involve fine-tuning SVC hyperparameters and implementing data augmentation techniques for potential performance enhancements.

Detailed description of pipeline:

The pipeline is initiated by data processing, with a significant focus on extracting hand detection landmarks from the dataset. These landmarks are subsequently stored in a JSON file. Prior to being input into the Support Vector Classification (SVC) classifier, these points undergo normalization and flattening. The classifier is then trained based on these preprocessed keypoints.

In the subsequent stage, new keypoints are captured from the live camera feed. These newly detected points are normalized and flattened before being utilized as input to the trained model predictor. The result is a label that signifies the classification based on the live camera's detected key points.

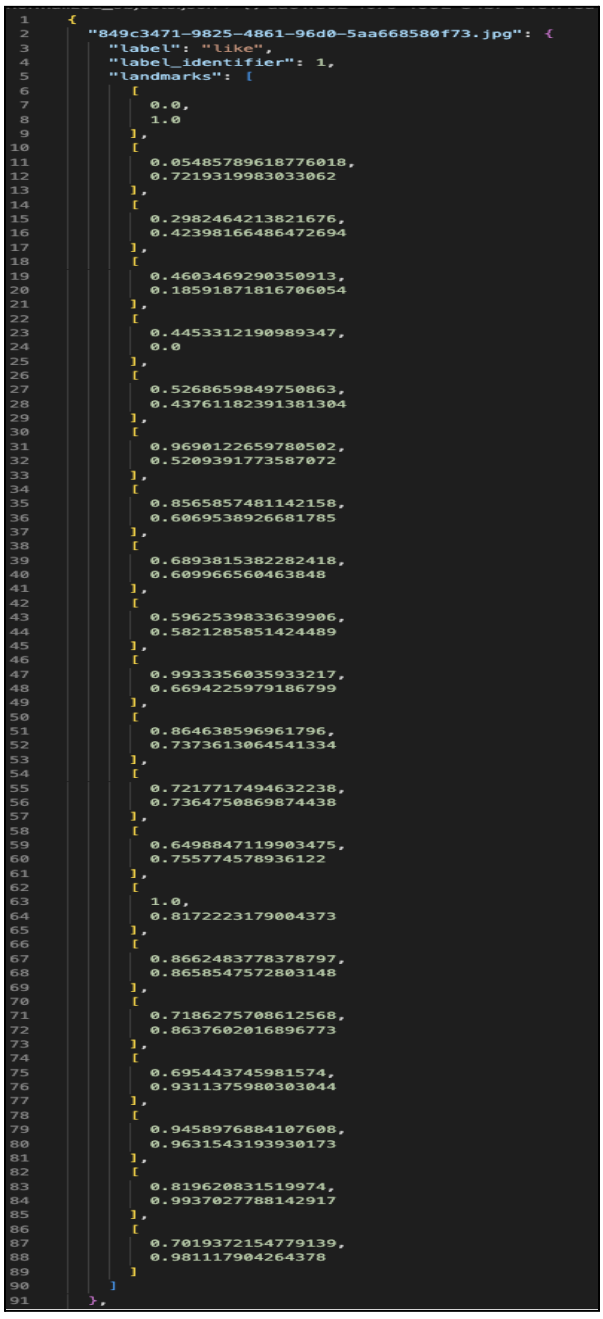
Flowchart for Project Implementation:



Gesture Recognition

MileStone 1:

Below is the screenshot of the data preprocessing and after getting the key points and saving in json format



Milestone 2 and Milestone 3:

Segmentation and Background merging

In this project we use three pre-trained models to segment the human from the background to perform merging of background. The pretrained models are used

1. **Human Segmenter**
   It identifies and segments the human in a video frame. It utilizes a pre-trained deep learning model for image segmentation. In this case, we are using deeplabv3_resnet101 from torchvision. We convert the input image to a tensor using ToTensor() transformation. We then normalize the tensor using the mean and standard deviation values. We pass the preprocessed image through the deep learning model and extract the segmentation map from the model's output. We then use the binary mask to segment the human from the original image. We then create a new image where the human is isolated from the background.

2. **DeepLabV3**
   DeepLabV3 is another image segmentation method that can be used for semantic segmentation tasks. We apply the necessary preprocessing steps, including converting the input image to a tensor and normalization. We pass the preprocessed image through the DeepLabV3 model. We then extract the segmentation map, which contains class predictions for each pixel. We colorize the segmentation map to visualize different semantic classes in the image.

3. **Semantic Segmenter**
   Semantic segmentation involves assigning a class label to each pixel in an image, providing a more detailed understanding of the scene. We use a pre-trained model designed for semantic segmentation. We apply preprocessing techniques to prepare the input image for the model. We pass the preprocessed image through the semantic segmentation model and obtain the class predictions for each pixel in the image. We overlay the segmented regions on the original image using a color mapping for different classes. This provides a detailed visualization of the semantic information in the scene.

After performing the segmentation process, we use the segmented image to overlay it on a background.

Background Merging Process:

The background merging is achieved by combining the results of semantic segmentation and human segmentation, where the background is replaced with a specified color. The blending of the original image and the segmented result is controlled by the parameter alpha in both segmenters.

Algorithms used:

We used three main pertained models for segmentation :

1. **DeepLabV3:**

   The segmenter DeepLabV3segmenter uses the ResNet101 as backbone which in pretained on a large dataset. The input image is preprocessed using torchvision transforms where tensor is to a tensor and normalizing its pixel values. The `transforms` attribute transforms include converting the image to a tensor and normalizing its pixel values. The `segment_image` method performs semantic segmentation on the input image using the DeepLabV3 model. It converts the input image to a PyTorch tensor and applies the predefined transforms which passes the preprocessed image through the DeepLabV3 model, obtaining a segmentation result. It extracts the class predictions from the result and converts them to a NumPy array. The `colorize_segmentation` method maps each class in the segmentation mask to a specific color. The colors are obtained using the 'tab20' colormap. It iterates through unique class IDs in the segmentation mask.

   For each non-background class, it retrieves a color from the 'tab20' colormap and assigns that color to the corresponding pixels in the output image.

   The detect_and_change_background` method replaces the background of the input image with a specified color. It then blends the original image with the segmented image using the specified alpha value. The code utilizes the OpenCV library for video capture, image processing, and display. The blending of the original and segmented images is done using the `cv.addWeighted` function.

2. **Human Segmentor:**

   We implement a Human Segmentation algorithm using the DeepLabV3 model with a ResNet101 backbone. The algorithm begins by initializing the segmentation model and preparing the input images through a series of preprocessing steps, including conversion to a PyTorch tensor and normalization. Subsequently, the model is applied to the input image to generate a pixel-wise segmentation map, where each pixel is assigned a class label indicating its semantic category, such as human or background. The segmentation map is then colorized to enhance visual interpretation, and a binary mask is created to isolate the human subject.

   The core of the algorithm involves replacing the background in the original image with a specified color, defaulting to normal background. This is achieved by identifying pixels corresponding to the human class in the segmentation map and modifying the original image accordingly. The blending of the modified image with the original, controlled by an alpha parameter, results in a final composition where the human subject is separated from the background. The algorithm operates in a real-time video feed, continuously

processing frames and updating the display to provide users with a live view of the human segmentation and background replacement.

User interaction is facilitated through a loop that captures video frames and displays the processed images. The loop continues until the user triggers an exit command by pressing the 'q' key. This design allows for a seamless and interactive experience, enabling users to observe the algorithm's segmentation and background replacement in action. Overall, the algorithm provides an effective and visually compelling method for isolating and highlighting human subjects in a dynamic video stream.

In this, Human Segmentor encapsulates the functionality of human segmentation. It initializes the DeepLabV3 model with a ResNet101 backbone, sets up image transformations for preprocessing, and allows customization of the blending alpha parameter. We use video input and apply the defined transformations, and perform human segmentation using the DeepLabV3 model. The resulting segmentation map, represented as class predictions, is returned as a NumPy array. The segmentation mask, the original image, and an optional background color is sent as an input, which produces a colorized mask by replacing non-human regions with the specified background color, where like (thumbs up symbol) has green color and dislike (thumbs down symbol) is blue.

3. **Semantic segmenter :**
   It uses the LRASPP_MobileNet_V3_Large model for semantic segmentation, which defines image transformations using the weights' default transforms. We then set the blending parameter `alpha` and prepare for the further color mapping using color mapping for each class based on the 'tab20' colormap. The image transformation is done using `torchvision.transforms. It later takes the image and converts it into pytorch tensor and does the transformation. It extracts class predictions from the result and colorizes the segmentation using the `colorize_segmentation` method and resizes the output image if its dimensions do not match the input image. Then we create a colorized segmentation mask based on the class predictions. It uses the 'tab20' colormap to assign distinct colors to different classes. Handles background class (class_id=0) separately. It maps class ID to an RGB color using the 'tab20' colormap. It later calls “segment image” to obtain a segmented image which creates a mask (`person_mask`) for the person by identifying pixels corresponding to the background in the segmented image. It replaces the background in the original image with a specified color. Later it blends the original and segmented images using the `cv.addWeighted` function with the specified alpha value.

Milestone 1

Run existing models and compare speed and qualitative goodness

Our first milestone in this project was to run pre-existing models, compare their speed and qualitative goodness.

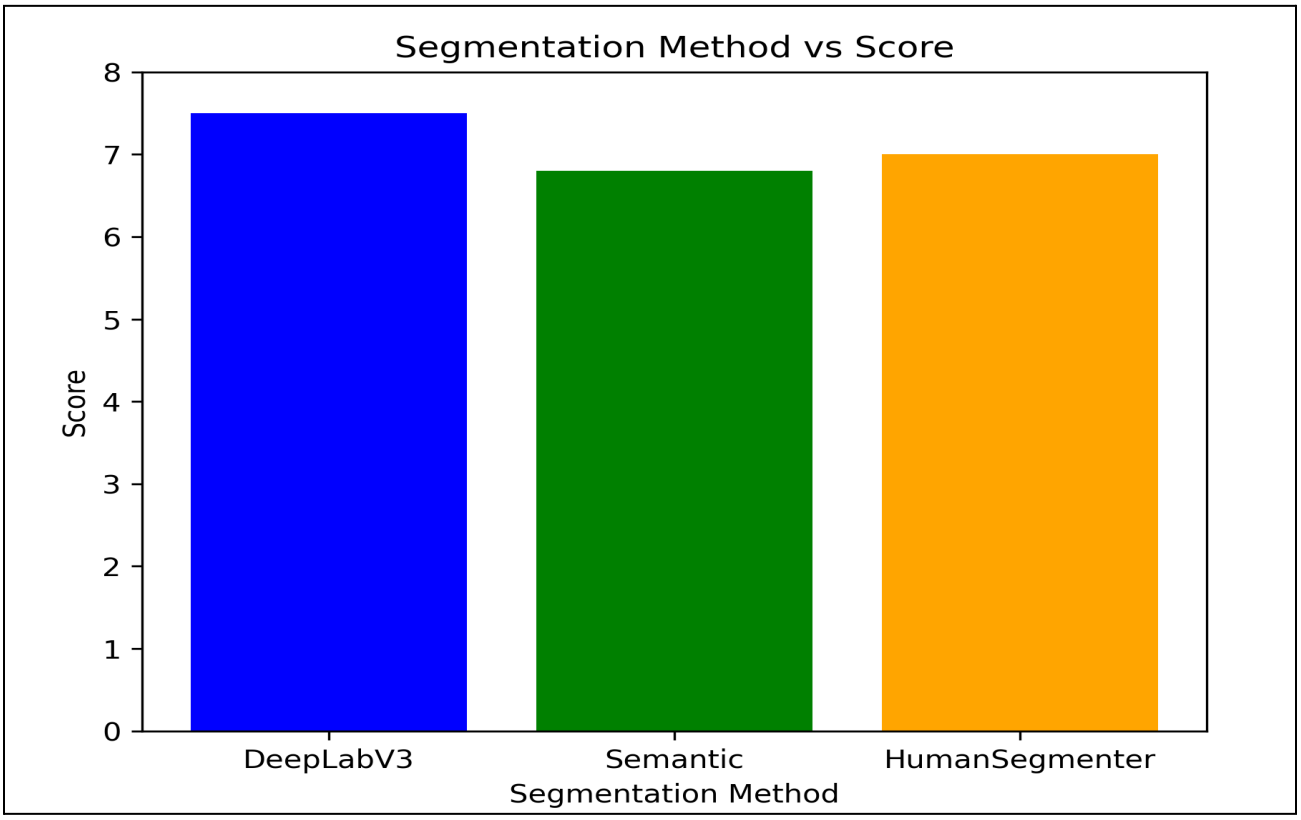
This is the image of segmenting still images. We tested it on 35 different pictures.

HumanSegmenter

Semantic segmentation

DeepLabV3

Quality Goodness :



We compared all the three pre-built segmentation models and we concluded that DeepLabv3 was the best according to all the test images. We compared it across 35 images and the result of one of the images is shown above.

As DeepLabV3 proved to be victorious , we decided to go ahead with the further segmentation and background merging using that.

Milestone 2:

As seen in the above outputs, we observed that DeepLabV3 was the best pre-trained model for segmentation. Using DeepLabV3, we then decided to complete our Milestone 2, which involved changing our background color.

Milestone 3

We then worked on milestone 4 which involved capturing the person and segmenting the image using a live video camera.

Milestone 4

We then worked on milestone 4 which involved capturing the person and segmenting the image using a live video camera and applying green background..

Milestone 5

This is an additional part, which we decided to complete a milestone which involved using a live background which is using floating balloons with a live video camera being used. However, the speed of the image and background is very slow because of which we could not go ahead and use this for our final project. Hence, we went ahead with using live video capture with static background color.

Milestone 6

We integrated gestures, such that backgrounds change on the basis of the gesture.

Link of implementation:
https://drive.google.com/drive/folders/1u-LSBhIHQmTtqr9msPmmUCJJXu-i-KOL?usp=sharing

Setbacks and Problem-solving

The hurdles we faced during this project were:

Gesture Recognition:

1. We had bottlenecks in data preprocessing; only few images in the data were processing initially, to overcome this we found out that new object is getting created for each image so we corrected our code to create only one object so that one object is used for all images and program processed all the images in the dataset.
2. We were trying to use KNN classifiers initially, but while training it was taking a long time. So, to overcome this we had to switch SVC to get the optimized way to get training done.

Segmentation and Background Changing:

1. The balloons that we used if we detected a "thumbs-up" were not supposed to be transparent and were supposed to appear from behind the person.
2. The code takes a lot of time to run because it captures one frame in approximately 3 seconds. We can't optimize it because this happened because of hardware restrictions as well.

Use of Course Material

Gesture Recognition

1. Inspired from Pose estimator used in the assignment 8.
   In the assignment we have used keypoint detection using an inbuilt library, in the project we have used mediapipe mode.

2. We have tried using two different classifier algorithms to classify the user image.

3. Using opencv functions, handling image resolutions.

Segmentation and Background Changing:

1. **Semantic Segmenter used in the assignment 8.**
   In this assignment, we used semantic segmentation to highlight a dog in the image.

2. **Masking for applying background to segmented images**
   In assignment 8 and assignment 2, we understood the concept of masking. This is the same thing we used to apply our background over our segmented images.

Ethical Considerations:

We made sure to consider ethical concerns at every stage of our project. The following key ethical considerations will guide our approach:

1. **Privacy Protection:**
   Our dataset is collected from Kaggle website, where privacy protection is considered all ready.

2. **Bias Mitigation:**
   This has been taken care of in the pre-processing stage, as we will be dealing with points and not the image. In our project, we convert the complete image into landmarks of the key points which makes sure that we do not cause any bias or discrimination from our outputs.

3. **Misuse of the model**
   Our model is trained only to recognize likes and dislikes and thus , our model won't be misused in gesture recognition.

Learning Objectives

In our project, we wanted to explore the concepts of computer vision, focusing on image processing, feature extraction, and recognizing objects. We did manage to meet and complete our learning objective. We performed feature extraction, to recognize gestures which are like (thumbs up) and dislike (thumbs down).
Our main focus was on understanding gestures and changing backgrounds in images. We were able to achieve the same. We understood gestures which were like and dislike. We also were able to perform segmentation of humans and change background color to green or red depending on the gesture.

We wanted to dive into classifiers, and become skilled at and using models for accurate gesture recognition. We achieved it using SVC to train and classify gestures. We can scale it to use more complex classifier models to improve the classification for recognition of more gestures.

We wanted to learn more about pre-trained models for segmentation. We achieved the same, using multiple pre-trained models such as DeepLabV3, Human Segmentor and Semantic Segmentation, which are used for segmentation. We also used masking to change the background.

Conclusion:

In conclusion, our project successfully combines gesture recognition with background manipulation in live camera feeds, providing users with an interactive experience. We started by exploring computer vision concepts, focusing on image processing, feature extraction, and object recognition. The key milestones in our project involved robust data preprocessing for gesture recognition, transitioning from K-Nearest Neighbors (KNN) to Support Vector Classification (SVC) for improved accuracy, and integrating sophisticated segmentation models like DeepLabV3 for background manipulation.

Our approach leverages existing libraries like mediapipe for hand tracking and pre-trained models for segmentation, demonstrating a practical application of machine learning and computer vision techniques. We encountered challenges such as speed and transparency issues with the background, but these were addressed through careful algorithm selection and problem-solving.

The ethical considerations were an integral part of our project, ensuring privacy protection, bias mitigation, and preventing misuse of the model. We utilized concepts from the course material, such as semantic segmentation and masking, to implement key components of our project.

Future work could involve optimizing the speed of the live video capture, exploring more advanced segmentation models, and addressing transparency issues in the background. Despite the challenges, our project showcases the potential of combining gesture recognition and background manipulation, opening avenues for interactive applications and creative expression.

Implementation video:
https://drive.google.com/drive/folders/1u-LSBhIHQmTtqr9msPmmUCJJXu-i-KOL?usp=sharing

References:

1. MediaPipe Library: (https://google.github.io/mediapipe/)
2. OpenCV Documentation: (https://docs.opencv.org/)
3. PyTorch Documentation: (https://pytorch.org/docs/stable/index.html)
4. cvzone Documentation: (https://github.com/cvzone/cvzone)
5. Kaggle Datasets: (https://www.kaggle.com/datasets)
6. DeepLabV3: (https://pytorch.org/hub/pytorch_vision_deeplabv3_resnet101/)
7. Image Segmentation: (https://pytorch.org/tutorials/beginner/deeplabv3_on_ios.html)