Segmentation and Pose Estimation of Swimmer using SAM and YOLOv7

Soubhik Majumdar, MSEE
San Jose State University, soubhiksinha.majumdar@sjsu.edu

***Abstract* - This project presents a novel framework for the segmentation and pose estimation of swimmers using two advanced deep learning models: Segment Anything Model (SAM) and YOLOv7. The integration of these models aims to enhance the precision and efficiency of analyzing swimmer movements, a critical aspect in sports performance optimization. Meta's SAM, Segment Anything Model, known for its segmentation capabilities, accurately isolates swimmers from aquatic environments. YOLOv7, a state-of-the-art object detection model, is employed to identify and track key body joints, enabling detailed biomechanical analysis. The pipeline combines SAM's segmentation output with YOLOv7's pose estimation to achieve robust and real-time analysis of swimmer actions. The proposed framework successfully segmented and estimated the pose of underwater SJSU swimmers from various video datasets with high accuracy. The outcomes of this project contribute to the advancement of Computer Vision technology and its application in sports analytics by offering a reliable tool for swimmer analysis.**

Index Terms – Convolutional Neural Networks, Pose estimation, SAM, Segmentation, YOLOv7

1. INTRODUCTION

The advancement of computer vision technologies has opened new possibilities in sports analytics, enabling precise and efficient analysis of athlete performance. In swimming, the body, joint and skeletal orientation and technique are central to measuring performance. To enable performance analysis in this regard, this project focuses on developing a robust framework for the segmentation and pose estimation of swimmers, combining the Segment Anything Model (SAM) and YOLOv7. By leveraging the unique strengths of these models, the framework addresses challenges in aquatic environments, in real-time.

The primary goals of this project are to utilize SAM for accurate swimmer segmentation from the aquatic environment and employ YOLOv7 for real-time pose estimation, while also labelling key joints in the skeletal frame. SAM's segmentation capabilities ensure precise isolation of swimmers from aquatic backgrounds encountering air bubbles and splashes, while YOLOv7 provides detailed tracking of key body joints, enabling comprehensive biomechanical analysis. Together, these models form a cohesive pipeline optimized for real-time performance.

This project holds significant importance in sports science and safety. Detailed analysis of swimmer movements can assist coaches and athletes in performance optimization, injury prevention, and technique refinement. Additionally, automated swimmer detection and tracking can enhance safety monitoring in pools and open waters. Beyond immediate applications, this work contributes to advancements in computer vision research, particularly in integrating segmentation and pose estimation tasks on human biomechanics.

My contribution to this project is successfully implementing pre-trained SAM and YOLOv7 models in a Google Colab environment under GPU and TPU constraints on multiple video datasets and coding and implementing the automation of identification and reduction of skeleton data points to key joints of the swimmer across multiple video datasets.

2. DESCRIPTION OF THE DESIGN

2.1 METHODOLOGY

The proposed framework integrates the Segment Anything Model (SAM) for segmentation and YOLOv7 for pose estimation to produced video data that is easy to analyze for sport analysts. The process begins with frames of the video being captured and stored in a database (it is written into a google drive folder). These frames are then fed into Meta's SAM model where the swimmer is segmented from the background. These segmented frames are written into another database in preparation for the pose estimation process. The YOLOv7 model takes these segmented frames as input and provides an accurate pose estimation skeleton with 17 joints. From the model output data, the key joints are labelled for clarity. These frames are stored in a database. The final process converted the segmented and pose estimated frames and converted them back in a video format with a predefined FPS (Frames per Second). This process is applied for up to 4 video datasets where the distance between the camera and swimmer varies and the movements and body orientation of the swimmer changes.

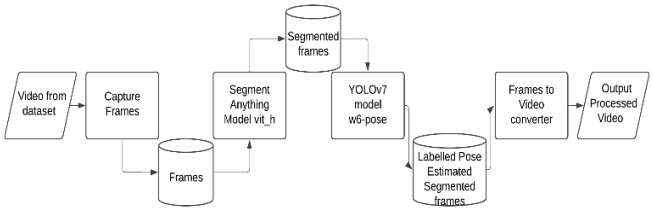

Fig. 1. Block Diagram of Segmentation and Pose Estimation Process

2.2 SEGMENTATION USING SAM

SAM is a foundation model for image segmentation that utilizes a Vision Transformer (ViT) backbone. It is designed for generalization, enabling zero-shot segmentation of objects across diverse datasets using sparse prompts such as bounding boxes, points, or text. SAM employs an attention mechanism to focus on relevant regions within the image, producing high-precision segmentation masks. Its architecture consists of a multi-scale ViT encoder for feature extraction and a mask decoder that integrates encoded image features with prompt embeddings to generate segmentation outputs. These capabilities make SAM particularly suitable for isolating swimmers in complex aquatic environments.

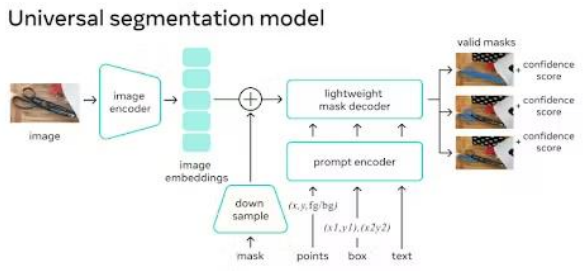

Fig. *2*. Architecture of the Segment Anything (SA) universal segmentation model [5]

The Segment Anything Model employs a powerful image encoder, a prompt encoder, and a lightweight mask decoder. This unique architecture enables flexible prompting, real-time mask computation, and ambiguity awareness in segmentation tasks. SAM also allows for auto annotation. Auto-annotation is a key feature of SAM, allowing users to generate a segmentation dataset using a pre-trained detection model. This feature enables rapid and accurate annotation of many images, bypassing the need for time-consuming manual labeling.

Segmentation processes visual data at the pixel level, using various techniques to annotate individual pixels as belonging to a specific class or instance. “Classic” image segmentation techniques determine annotations by analyzing inherent qualities of each pixel (called “heuristics”) like color and intensity, while deep learning models employ complex neural networks for sophisticated pattern recognition. The outputs of this annotation are segmentation masks, representing the specific pixel-by-pixel boundary and shape of each class, typically corresponding to different objects, features or regions, in the image. [1]

```
# Load a SAM model
device = "cuda" if torch.cuda.is_available() else "cpu"
model_type = "vit_h"  # Model type can be 'vit_h', 'vit_l', or 'vit_b' (larger models may perform better but are slower)
sam = sam_model_registry[model_type](checkpoint="/content/gdrive/MyDrive/CompVision/sam_vit_h_4b8939.pth")
sam.to(device);
```

Fig. 3. SAM is loaded in Colab environment

SAM is loaded in a Google Colab environment. The model used is the “vit_h” model. Due to the computational demands of this model, even in inference, CUDA, Compute Unified Device Architecture, is utilized, to accelerate GPU and process data faster than traditional CPUs. This model type is stored on google drive and is accessible when the drive is mounted.

```
box = widget.bboxes[0] if widget.bboxes else default_box
box = np.array([
    box['x'],
    box['y'],
    box['x'] + box['width'],
    box['y'] + box['height']
])

mask_predictor = SamPredictor(sam)
mask_predictor.set_image(image_rgb)

masks, scores, logits = mask_predictor.predict(
    box=box,
    multimask_output=False
)
```

Fig. 4. Code for Prediction Masks on Bounding boxes region of frame

In the above code snippet, ‘widget.bboxes’ is a list of bounding boxes which are provided through a user interface on colab, this is done for every frame under consideration for segmentation and further processing. The bounding box is converted into an array format. ‘SamPredictor’ is a class in SAM that is prompted for segmentation on the image that it is set for. ‘masks’ are the predicted segmentation masks, ‘scores’ are the confidence for each mask, ‘logits’ are raw model outputs before applying activation functions. ‘multimask_output’ is set to ‘False’ as the desired output is to return a singular mask, one that is the mask of the swimmer. This ensures that only the most confident mask is used. As the bounding box is drawn by the user, it was noticed that the swimmer’s mask was obtained for every frame processed as the confidence score of the swimmer was always highest within the box.

```
box_annotator = sv.BoxAnnotator(color=sv.Color.RED, color_lookup=sv.ColorLookup.INDEX)
mask_annotator = sv.MaskAnnotator(color=sv.Color.RED, color_lookup=sv.ColorLookup.INDEX)

detections = sv.Detections(
    xyxy=sv.mask_to_xyxy(masks=masks),
    mask=masks
)
detections = detections[detections.area == np.max(detections.area)]

source_image = box_annotator.annotate(scene=image_bgr.copy(), detections=detections)
segmented_image = mask_annotator.annotate(scene=image_bgr.copy(), detections=detections)
```

Fig. 5. Code for detecting, annotation and visualization to output the segmented image

The above code snippet focuses on visualizing segmentation and detection results by overlaying bounding boxes and masks on an input image. Two annotator objects, 'sv.BoxAnnotator' and 'sv.MaskAnnotator', are initialized to handle the visualization. The 'sv.BoxAnnotator' is responsible for drawing bounding boxes around detected objects, using red as the specified color (sv.Color.RED). Similarly, 'sv.MaskAnnotator' overlays segmentation masks in red. Both annotators use 'sv.ColorLookup.INDEX' to map colors to indices, ensuring consistent annotation when multiple objects are present.

The 'sv.Detections' object is created to store detection information, including bounding box coordinates and segmentation masks. The bounding box coordinates are derived from the masks using 'sv.mask_to_xyxy(masks=masks)', which computes the smallest rectangle enclosing each mask. The mask=masks parameter retains the raw mask data for mask annotation. Once detections are created, only the largest detection is retained by filtering based on the area property. This is achieved by comparing each detection's area with the maximum area 'np.max(detections.area)' to ensure that the annotations focus on the most prominent region in the image.

Finally, annotations are applied to copies of the input image 'image_bgr.copy()' to create two separate visual outputs. The 'box_annotator.annotate()' method draws bounding boxes from the detections onto the image, resulting in 'source_image', which highlights the detected object's enclosing area. The 'mask_annotator.annotate()' method overlays the segmentation mask onto another image copy, producing 'segmented_image', which highlights the segmented region. These outputs provide clear visualizations of the detected regions, combining object localization through bounding boxes and precise region identification using segmentation masks. This approach is essential for understanding segmentation and detection tasks in computer vision. The segmentation process is performed on multiple frames of the video datasets and the output segmented frames are stored in a Google drive folder for the next process, namely, pose estimation.

2.3 POSE ESTIMATION USING YOLOv7

YOLOv7 is a state-of-the-art object detection model extended for pose estimation tasks. It features an anchor-based detection mechanism that predicts object bounding boxes and keypoints with high accuracy. The architecture employs a convolutional backbone combined with feature pyramids for multi-scale detection, ensuring robust performance across different swimmer positions and sizes. For pose estimation, YOLOv7 includes a regression head that localizes key body joints. Innovations such as dynamic label assignment and sparsity-aware training improve efficiency and accuracy, making YOLOv7 suitable for real-time applications. Its lightweight design, coupled with sparsity-aware training and pruning techniques, enhances computational efficiency without sacrificing accuracy, making it suitable for real-time applications. By integrating YOLOv7, the framework achieves precise detection and tracking of swimmer poses, enabling advanced performance analysis.

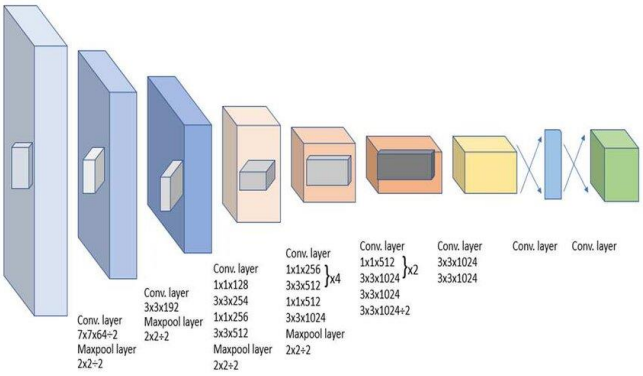

Fig. 6. YOLOv7 model architecture [3]

The YOLO system inputs an image and employs a deep convolutional neural network to recognize the objects within it. The figure above displays the architecture of the CNN model that acts as the foundation of the YOLO. It divides an image into an S x S grid, where each grid cell is responsible for detecting an object if it appears in that cell. The grid cells then produce predictions for bounding boxes and confidence scores for those boxes. During training, only one bounding box predictor is assigned per object and the confidence scores show the model's certainty and accuracy in predicting an object within the box.[3]

```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
weigths = torch.load('/content/gdrive/MyDrive/CompVision/yolov7-w6-pose.pt', map_location=device)
model = weigths['model']
_ = model.float().eval()

if torch.cuda.is_available():
    model.half().to(device)
```

Fig. 7. YOLOv7 model yolov7-w6-pose.pt is loaded in colab environment

The above code snippet initializes the device to either a GPU or CPU using CUDA if available, enabling efficient hardware utilization. The model weights are loaded from the specified file (yolov7-w6-pose.pt). The loaded weights include a key 'model', which contains the trained YOLOv7 pose estimation model. The model is set to evaluation mode using '.eval()' to deactivate layers such as dropout. If CUDA is available, the model is converted to half-precision '.half()' to optimize memory usage and computational speed and is transferred to the GPU for acceleration.

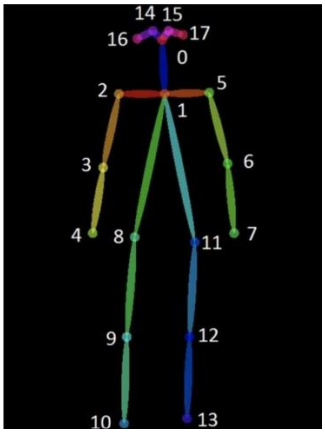

Fig. 8(a). All 17 joints defined by YOLO v7 pose model [2]

```
# Indices for the grouped joints to label
joint_groups = {
    "knees": [joint_names.index("left_knee"), joint_names.index("right_knee")],
    "hips": [joint_names.index("left_hip"), joint_names.index("right_hip")],
    "elbows": [joint_names.index("left_elbow"), joint_names.index("right_elbow")],
}
```

Fig. 8(b) Code labelling key joints to be overlayed on output frame

The 'joint_groups' dictionary organizes key body joints into meaningful groups, such as knees, hips, and elbows, for easier interpretation and visualization. Each group contains indices corresponding to the specific joint names in the 'joint_names' list.

```
for image_filename in input_folder.glob("*.jpg"):
    # Read the image
    image = cv2.imread(str(image_filename))  # Convert Path object to string path
    # Apply letterbox resizing
    image = letterbox(image, 960, stride=64, auto=True)[0]
    # Copy image for processing
    image_ = image.copy()
    # Convert image to tensor
    image = transforms.ToTensor()(image)
    image = torch.tensor(np.array([image.numpy()]))
    if torch.cuda.is_available():
        image = image.half().to(device)
    # Get model output
    output, _ = model(image)
    # Perform non-max suppression on keypoints
    output = non_max_suppression_kpt(output, 0.25, 0.65, nc=model.yaml['nc'], nkpt=model.yaml['nkpt'], kpt_label=t
    # Convert output to keypoints
    with torch.no_grad():
        output = output_to_keypoint(output)
```

Fig. 9. Code for Image preprocessing and YOLOv7 model inference

The script processes segmented images for pose estimation through several steps. First, each segmented image is loaded using OpenCV (cv2.imread) and resized using a letterbox method, ensuring that the aspect ratio is maintained while adapting the image dimensions for the model. The resized image is then converted into a tensor using 'transforms.ToTensor()' and further encapsulated in a batch-like structure. If CUDA is available, the tensor is converted to half-precision and moved to the GPU. The YOLOv7 model processes the input image, producing predictions for bounding boxes and keypoints. Only when confidence is above 0.4 is pose estimation and labelling allowed, and therefore only those predictions are considered for further processing. These predictions undergo non-maximum suppression (NMS) using 'non_max_suppression_kpt', which eliminates overlapping detections and retains the most confident ones. NMS starts by selecting the box with the highest objectness score. It then estimates the IoU of this box with other predicted boxes that have lower objectness scores. In this project, IoU is set to 0.65. Finally, the predictions are converted into keypoint coordinates using 'output_to_keypoint', ready for further analysis or visualization.

```
# Prepare image for saving
nimg = image[0].permute(1, 2, 0) * 255
nimg = nimg.cpu().numpy().astype(np.uint8)
nimg = cv2.cvtColor(nimg, cv2.COLOR_RGB2BGR)

# Plot keypoints on the image
for idx in range(output.shape[0]):
    plot_skeleton_kpts(nimg, output[idx, 7:].T, 3)
# Plot grouped labels on the image
for idx in range(output.shape[0]):
    nimg = plot_grouped_labels(nimg, output[idx, 7:].T.flatten(), joint_groups)
```

Fig. 10. Code for Visualization and Keypoint Annotation

The processed image is prepared for visualization by converting the tensor back into a NumPy array. The image, originally in RGB, is converted to BGR format using OpenCV (cv2.cvtColor) for compatibility with most image viewers. Keypoints are plotted on the image using 'plot_skeleton_kpts', which connects detected keypoints to form a skeletal representation. Additionally, 'plot_grouped_labels' annotates the image with labels corresponding to predefined joint groups, such as "knees" or "hips," by mapping the detected keypoints to their respective groups. This step creates a visually informative output, showing both individual keypoints and grouped labels, which can be used for applications such as athletic performance analysis or injury prevention.

```
# Release GPU memory after processing each image
del image, image_, output, nimg  # Delete tensors and arrays
torch.cuda.empty_cache()  # Clear GPU memory
gc.collect()  # Force garbage collection to release RAM
```

Fig. 11. Code for Releasing GPU memory after each image

This code ensures efficient memory management during image processing, especially when working with large datasets and GPU-based inference, while also forcing python's garbage collector to reclaim unused memory. This is especially useful for releasing memory allocated to CPU variables, further ensuring efficient resource utilization.

The above YOLOv7 model process is performed for every considered frame that was successfully segmented. These frames which are stored in google folder after pose estimation are converted to produce the processed video, that is successfully swimmer- segmented and pose estimated, with a predefined FPS.

3. RESULTS AND ANALYSIS

Frames are extracted from the video datasets to be segmented and pose estimated. Due to GPU/TPU constraints every 30th frame is considered for segmentation and pose estimation.

Fig. 12(a). Frame 700 from Swim_InputVideo.mp4

As shown in the figure above, the frames are underwater and contain bubbles, splashes, degrees of distortion and/or obstruction. The position and body orientation of the swimmer(s) change with respect to the underwater cameras. It is under these conditions that the swimmer nearest to the camera are to segmented, and subsequently pose estimated.

Fig. 12(b). Annotated Image from Frame 700

The preliminary stage involves using SAM prediction model to obtain all the annotated masks from the frames. These frames do not isolate and segment the swimmer from the background, but it does give us an idea of how the frames are being annotated into segments.

Using the bounding box as explained in the methodology, we can isolate the swimmer from the background completely. This is essentially the use of the mask detected within the bounding box, and it's overlayed over the original image and colored with a translucent red color. The result is the figure below. This is performed for all considered frames from video datasets and stored in a google folder for pose estimation.

Fig. 12(c). Segmented frame 700 where the swimmer is segmented completely from the aquatic background

The segmented frame is then processed by the YOLOv7 model. Pose estimation is performed and a 17 joint skeleton is overlayed over the swimmer. The coordinates of these joints are stored in a .csv file for each video dataset. Of these joints, the ones that relate to the "elbow", "knees" and "hips" are identified as key joints and are labelled on the skeleton.

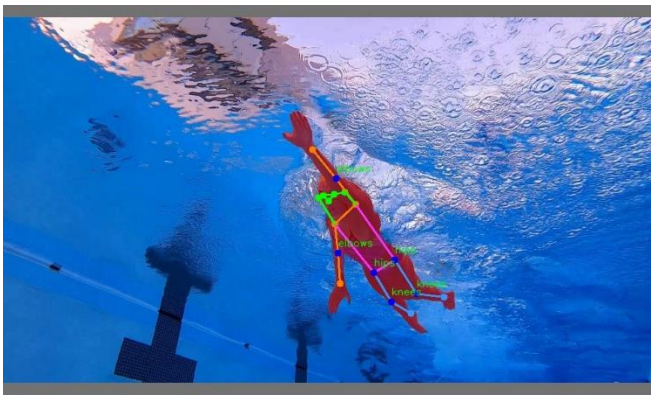

Fig. 12(d). Pose estimated frame 700 with labelled key joints

Unlike segmentation, the pose estimation accuracy was impacted by body orientation and position and distance with respect to the underwater camera. Despite this, over 70% of

frames considered across 4 video datasets performed pose estimation accurately.

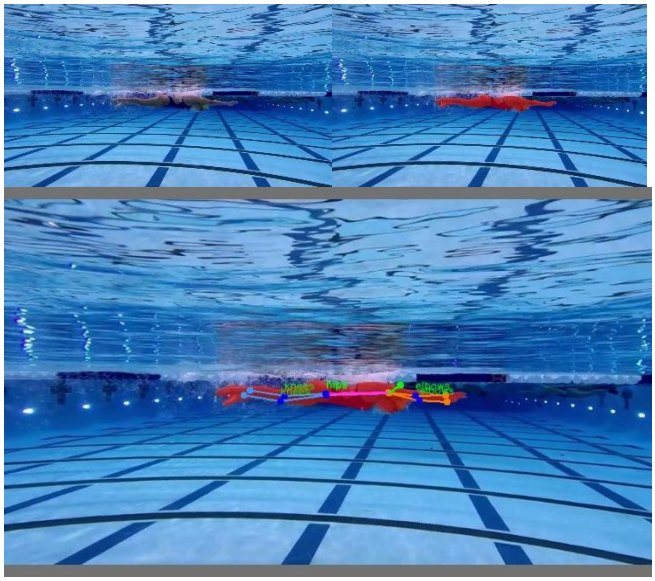

Fig. 13. Segmentation and Pose estimated results from BR.mp4 frame 120

The same process was conducted for the other video datasets, above, are the results for the BR.mp4 video dataset. The perspective of the camera is different, nevertheless, segmentation was performed without errors and pose estimations were orientally accurate, if not perfect.

4. CONCLUSION

The implemented framework successfully achieves swimmer segmentation and pose estimation using SAM and YOLOv7, fulfilling the project's primary goals. SAM demonstrated its effectiveness in isolating swimmers from complex aquatic environments, handling challenges such as water reflections and overlapping objects with precision. YOLOv7 accurately detected and tracked key body joints, enabling detailed skeletal pose reconstruction essential for performance analysis. Visual results further underscore the framework's ability to generate, segmentation masks, and accurate labeled skeletal key joints. However, there is scope for improvement. Integrating transfer learning techniques could enhance robustness by addressing challenges such as bubbles, splashes, and underwater distortions.

Using pre-trained models, SAM and YOLOv7 has enabled us to segment and estimate the pose of swimmers in an environment that these technologies are not typically applied to. As a result, it has deepened our understandings of pre-trained convolutional neural network and deep learning models and also deepened our understanding of the limitations and potential scope for improvement in these domains.

5. REFERENCES

[1] IBM, "What is image segmentation", [Online] Available: https://www.ibm.com/topics/image-segmentation [Accessed: Dec. 06, 2024].

[2] E. Odemakinde, "Human Pose Estimation - Ultimate Overview in 2025", Oct 04, 2023 [Online] Available:https://viso.ai/deep-learning/pose-estimation-ultimate-overview/ [Accessed: Dec 07, 2024]

[3] K. Kolachi, M. Khan, S. A. Sarang and A. Raza, "Fault Detection and Quality Inspection of Printed Circuit Board Using Yolo-v7 Algorithm of Deep Learning," 2023 7th International Multi-Topic ICT Conference (IMTIC), Jamshoro, Pakistan, 2023, pp. 1-7, doi: 10.1109/IMTIC58887.2023.10178512.

[4] encord.com, "Meta AI's Segment Anything Model (SAM) Explained: The Ultimate Guide", April 06, 2023 [Online] Available: https://encord.com/blog/segment-anything-model-explained/ [Accessed: Dec 07, 2024]

[5] docs.ultralytics, "Segment Anything Model (SAM)", [Online] Available: https://docs.ultralytics.com/models/sam/ [Accessed: Dec 06, 2024]

Soubhik Majumdar, MSEE.

A. Appendix

Link for zip file containing whole project:

CompVision.zip

Link to Google colab code for segmentation: https://colab.research.google.com/drive/1te0jZtdPxZT8P3Gy1SoHnXRKGMpVs9J5?usp=sharing

Link to Google colab code for pose estimation: https://colab.research.google.com/drive/1XPHkUcIYP5Nzi_DxHErbs7PL0V6p2let?usp=sharing

Link to Google drive Folder CompVision: https://drive.google.com/drive/folders/17TKFMhLaWqlZrZvsrkRgnaj7Q8Bx8Rgi?usp=sharing