Eyeglass Segmentation Report

Name – Manish. S

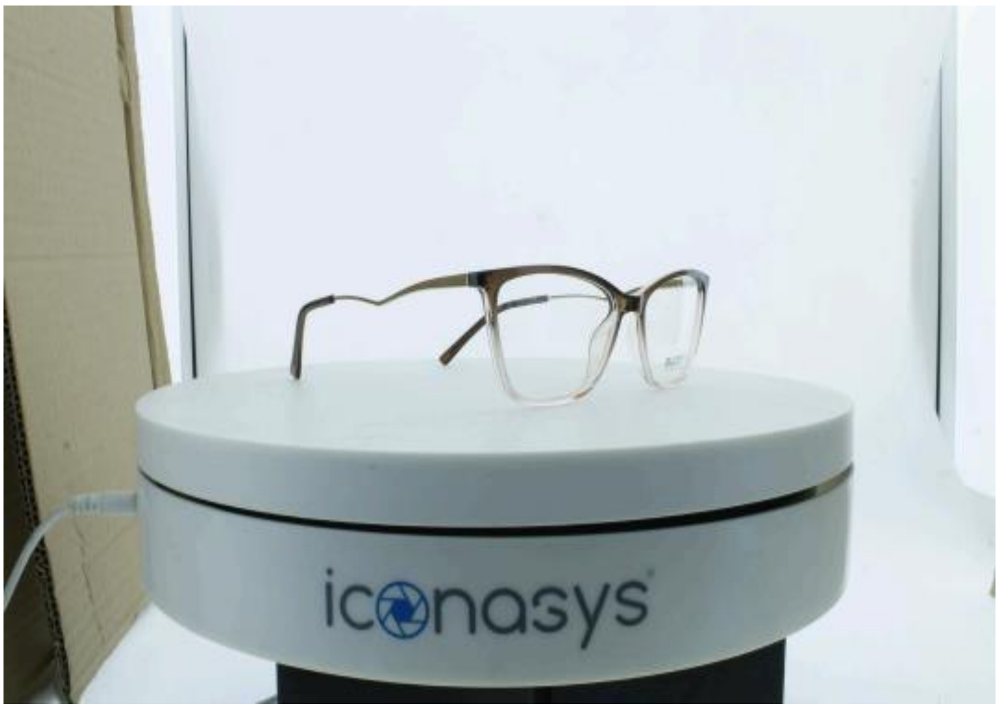
Reg. No – 20BCE7228

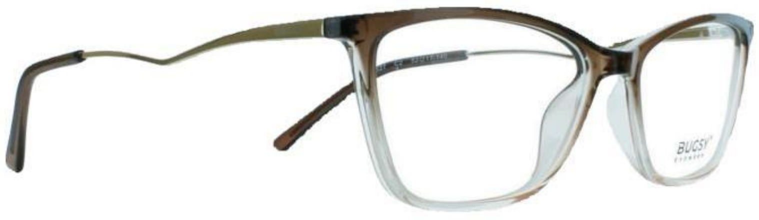
Mail - manish.20bce7228@vitap.ac.in

Eyeglass Segmentation Project Assignment

Objective

Develop an image Segmentation solution that accurately segments Eyeglasses from images, ensuring the edges are smoothed, the segmentation is precise, and maintaining a white background for the segmented output. This assignment will test your ability to apply and possibly retrain machine learning models for the task of image segmentation.





Task Description

You are required to develop a program that applies a segmentation algorithm to accurately extract Eyeglasses from a set of images. The program should account for variations in lighting, scale, and orientation of glasses. The edges of the segmented glasses must be smooth, and the background of the output images should be white.

Technical Stack

Programming Language: Python

Segmentation Model: Segment Anything

Libraries and Tools: OpenCV, PyTorch, MatplotLib etc.

Theoretical Concept

The Segment Anything Model (SAM) is an instance segmentation model developed by Meta Research and released in April, 2023. Segment Anything was trained on 11 million immages and 1.1 billion segmentation masks.

Using Segment Anything, you can upload an image and:

1. Generate segmentation masks for all objects SAM can identify;

2. Provide points to guide SAM in generating a mask for a specific object in an image, or;

3. Provide a text prompt to retrieve masks that match the prompt (although this feature was not released at the time of writing).

SAM has a wide range of use cases. For instance, you can use SAM:

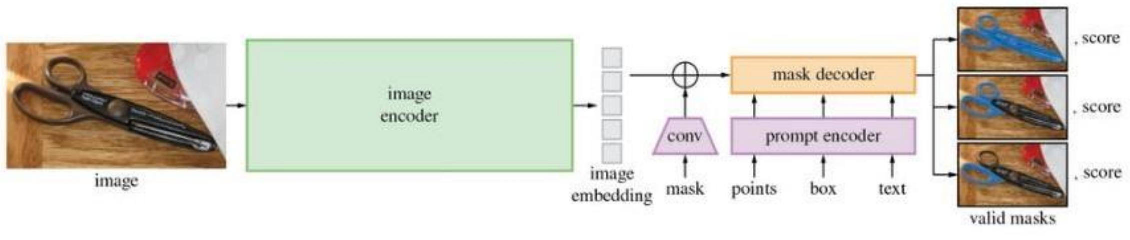
4. As a zero-shot detection model, paired with an object detection model to assign labels to specific objects;

5. As an annotation assistant, a feature made available in the Roboflow Annotate platform, and;

6. Standalone to extract features from an image, too. For example, you could use SAM to remove backgrounds from images.

How it works?

The Segment Anything model is broken down into two sections. The first is a featurization transformer block that takes and image and compresses it to a 256x64x64 feature matrix. These features are then passed into a decoder head that also accepts the model's prompts, whether that be a rough mask, labeled points, or text prompt (note text prompting is not released with the rest of the model).



Segment Anything model diagram

The Segment model architecture is revolutionary because it puts the heavy lifting of image featurization to a transformer model and then trains a lighter model on top. For deploying SAM to production, this makes for a really nice user experience where the featurization can be done via inference on a backend GPU and the smaller model can be run within the web browser.

Repository

https://github.com/Wild-Manish/Glasses-Segmentation/tree/main

The repository contains:

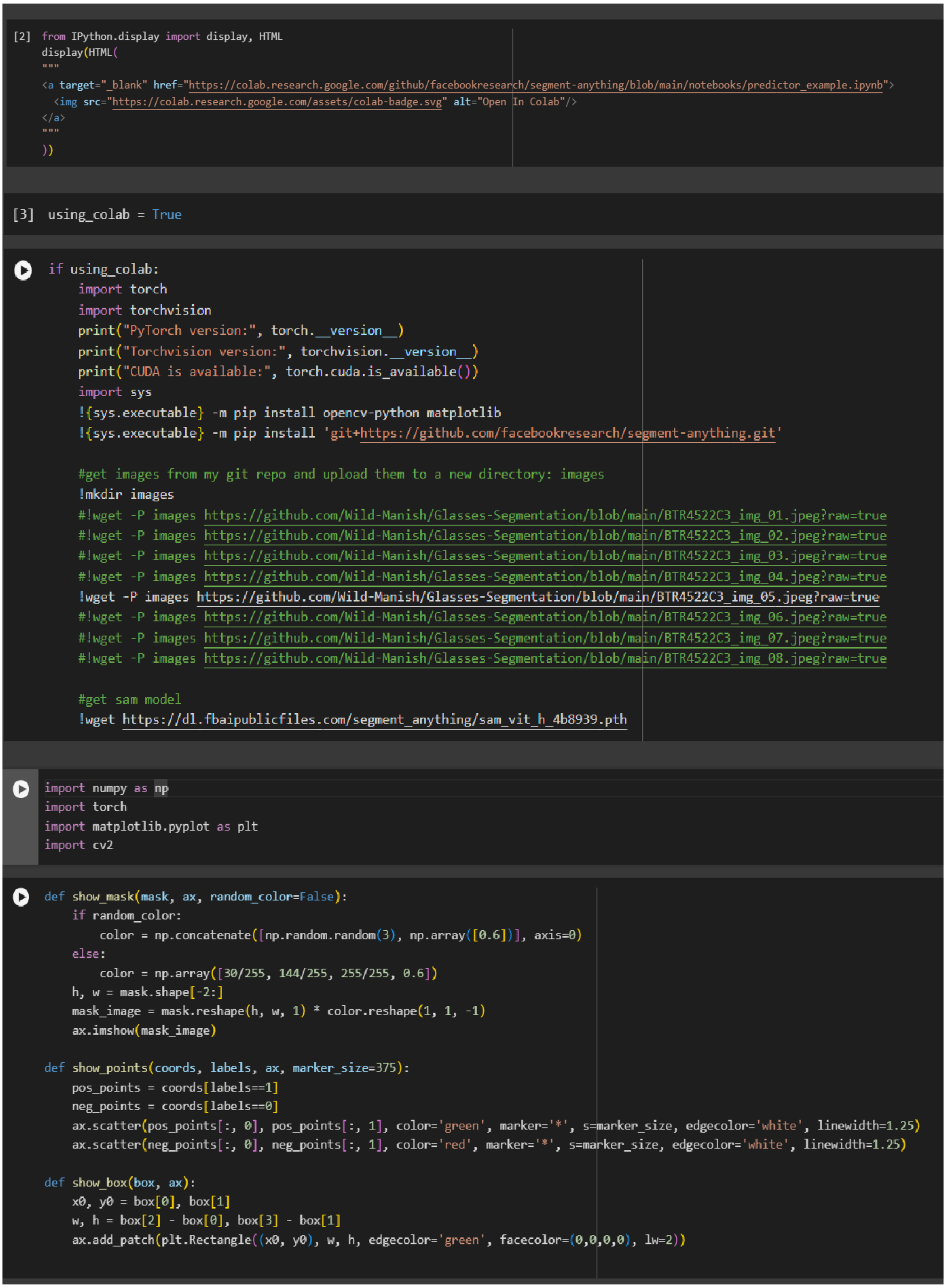
1. images of one model of the dataset

2. copy of the colab notebook for reference

3. the documented report

Approach

1. Starting off with the basic environment setup, necessary imports and helper functions for displaying points, boxes and masks.



```
[2] from IPython.display import display, HTML
    display(HTML(
    """
    <a target="_blank" href="https://colab.research.google.com/github/facebookresearch/segment-anything/blob/main/notebooks/predictor_example.ipynb">
      <img src="https://colab.research.google.com/assets/colab-badge.svg" alt="Open In Colab"/>
    </a>
    """
    ))
```

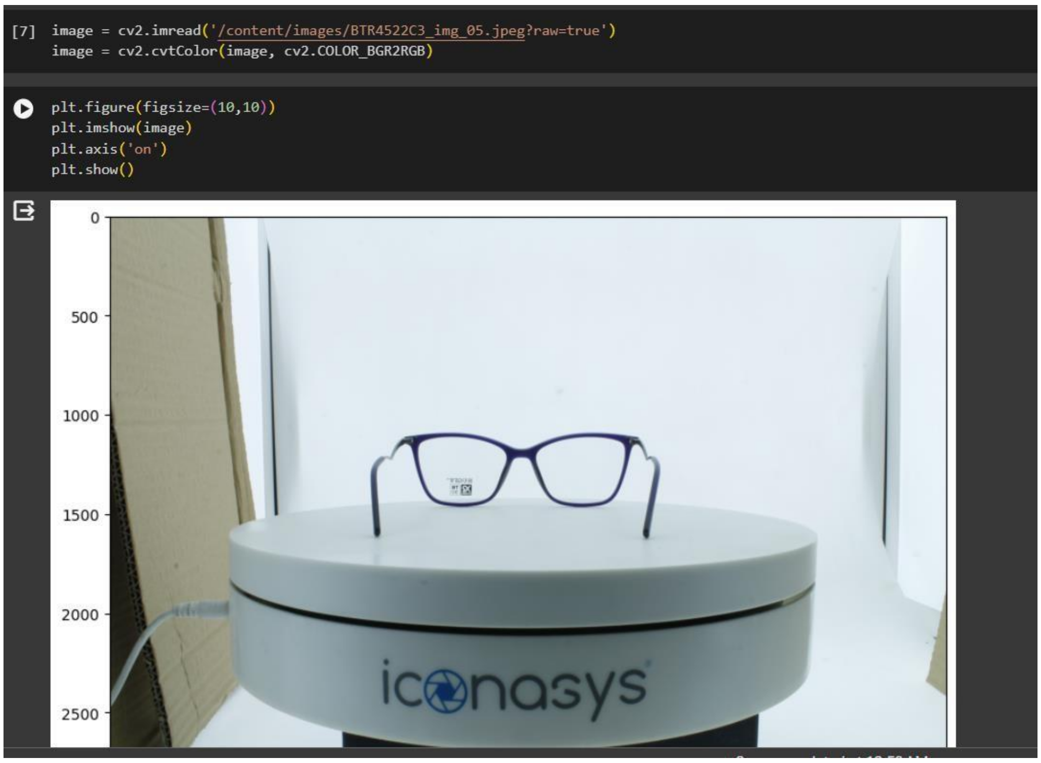
```
[3] using_colab = True
```

```
if using_colab:
    import torch
    import torchvision
    print("PyTorch version:", torch.__version__)
    print("Torchvision version:", torchvision.__version__)
    print("CUDA is available:", torch.cuda.is_available())
    import sys
    !{sys.executable} -m pip install opencv-python matplotlib
    !{sys.executable} -m pip install 'git+https://github.com/facebookresearch/segment-anything.git'

    #get images from my git repo and upload them to a new directory: images
    !mkdir images
    #!wget -P images https://github.com/Wild-Manish/Glasses-Segmentation/blob/main/BTR4522C3_img_01.jpeg?raw=true
    #!wget -P images https://github.com/Wild-Manish/Glasses-Segmentation/blob/main/BTR4522C3_img_02.jpeg?raw=true
    #!wget -P images https://github.com/Wild-Manish/Glasses-Segmentation/blob/main/BTR4522C3_img_03.jpeg?raw=true
    #!wget -P images https://github.com/Wild-Manish/Glasses-Segmentation/blob/main/BTR4522C3_img_04.jpeg?raw=true
    !wget -P images https://github.com/Wild-Manish/Glasses-Segmentation/blob/main/BTR4522C3_img_05.jpeg?raw=true
    #!wget -P images https://github.com/Wild-Manish/Glasses-Segmentation/blob/main/BTR4522C3_img_06.jpeg?raw=true
    #!wget -P images https://github.com/Wild-Manish/Glasses-Segmentation/blob/main/BTR4522C3_img_07.jpeg?raw=true
    #!wget -P images https://github.com/Wild-Manish/Glasses-Segmentation/blob/main/BTR4522C3_img_08.jpeg?raw=true

    #get sam model
    !wget https://dl.fbaipublicfiles.com/segment_anything/sam_vit_h_4b8939.pth
```

```
import numpy as np
import torch
import matplotlib.pyplot as plt
import cv2
```

```
def show_mask(mask, ax, random_color=False):
    if random_color:
        color = np.concatenate([np.random.random(3), np.array([0.6])], axis=0)
    else:
        color = np.array([30/255, 144/255, 255/255, 0.6])
    h, w = mask.shape[-2:]
    mask_image = mask.reshape(h, w, 1) * color.reshape(1, 1, -1)
    ax.imshow(mask_image)

def show_points(coords, labels, ax, marker_size=375):
    pos_points = coords[labels==1]
    neg_points = coords[labels==0]
    ax.scatter(pos_points[:, 0], pos_points[:, 1], color='green', marker='*', s=marker_size, edgecolor='white', linewidth=1.25)
    ax.scatter(neg_points[:, 0], neg_points[:, 1], color='red', marker='*', s=marker_size, edgecolor='white', linewidth=1.25)

def show_box(box, ax):
    x0, y0 = box[0], box[1]
    w, h = box[2] - box[0], box[3] - box[1]
    ax.add_patch(plt.Rectangle((x0, y0), w, h, edgecolor='green', facecolor=(0,0,0,0), lw=2))
```

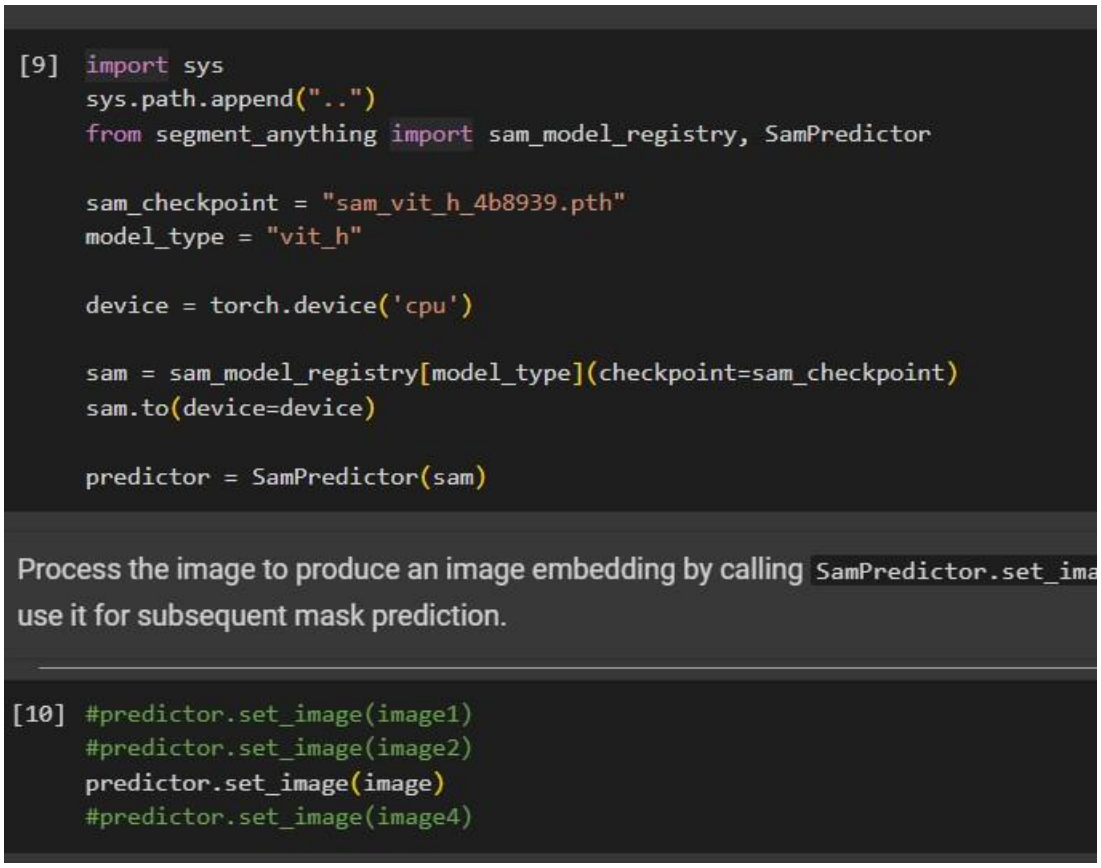
2. plotting an example image for view



```
[7] image = cv2.imread('/content/images/BTR4522C3_img_05.jpeg?raw=true')
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
```

```
plt.figure(figsize=(10,10))
plt.imshow(image)
plt.axis('on')
plt.show()
```

3. First, load the SAM model and predictor. Change the path below to point to the SAM checkpoint. Running on CUDA and using the default model are recommended for best results. Process the image to produce an image embedding by calling SamPredictor.set_image. SamPredictor remembers this embedding and will use it for subsequent mask prediction.

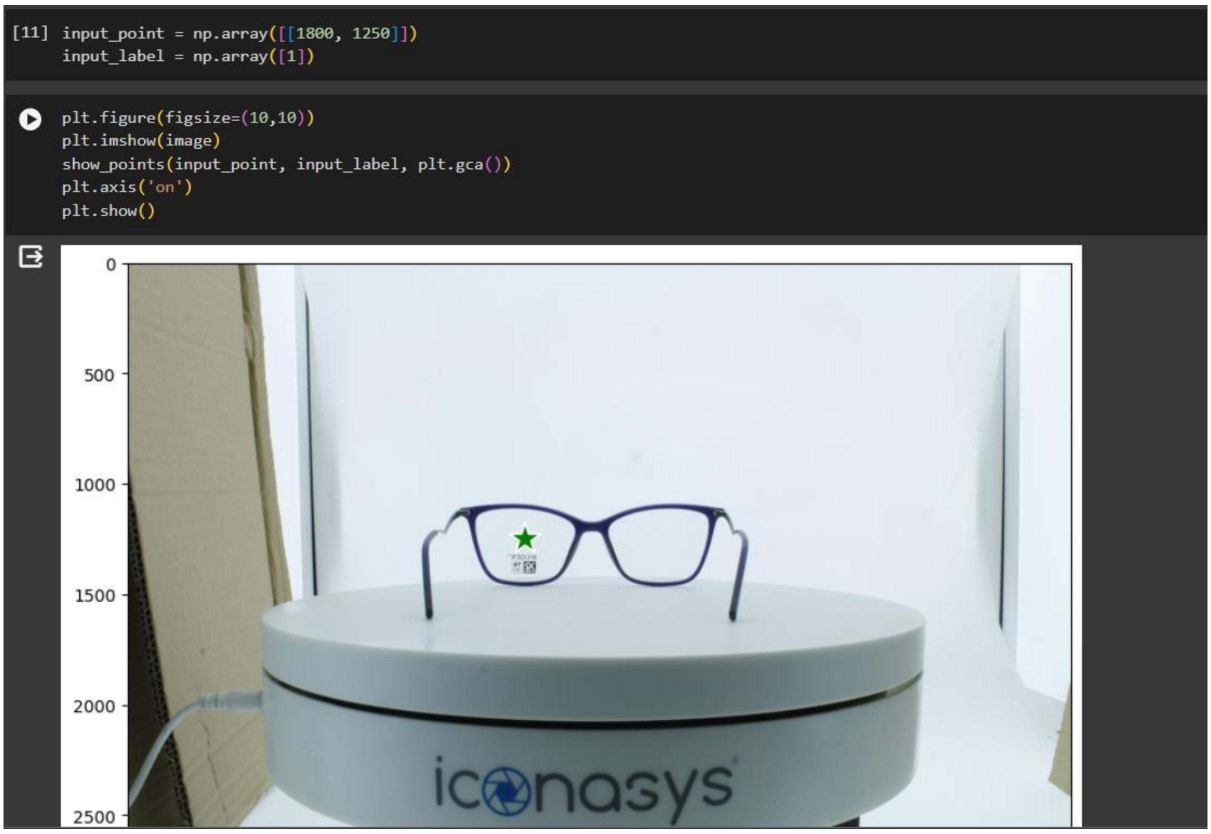


```
[9] import sys
    sys.path.append("..")
    from segment_anything import sam_model_registry, SamPredictor

    sam_checkpoint = "sam_vit_h_4b8939.pth"
    model_type = "vit_h"

    device = torch.device('cpu')

    sam = sam_model_registry[model_type](checkpoint=sam_checkpoint)
    sam.to(device=device)

    predictor = SamPredictor(sam)
```

Process the image to produce an image embedding by calling `SamPredictor.set_ima`
use it for subsequent mask prediction.

```
[10] #predictor.set_image(image1)
     #predictor.set_image(image2)
     predictor.set_image(image)
     #predictor.set_image(image4)
```

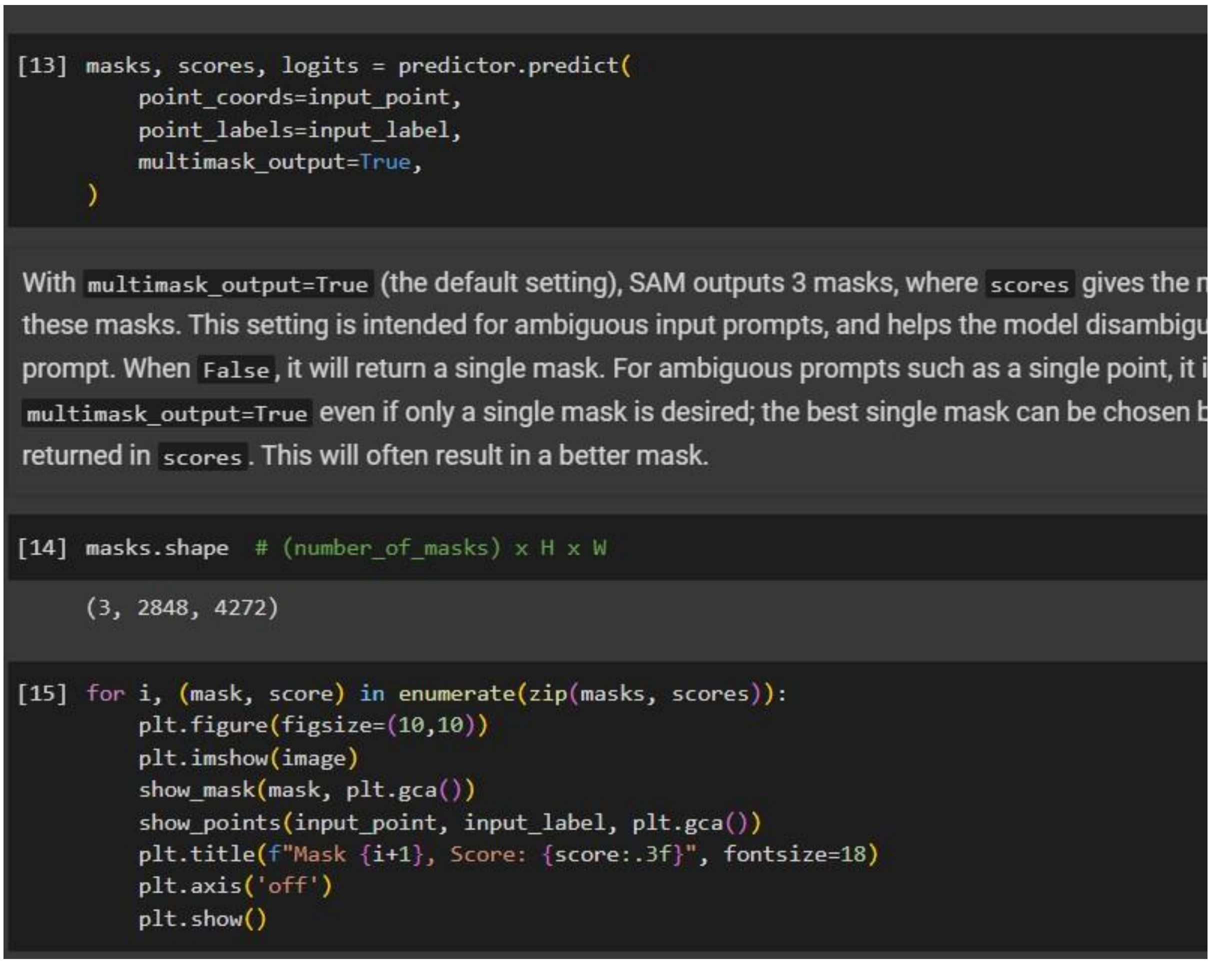
4. To select the truck, choose a point on it. Points are input to the model in (x,y) format and come with labels 1 (foreground point) or 0 (background point). Multiple points can be input; here we use only one. The chosen point will be shown as a star on the image.



```
[11] input_point = np.array([[1800, 1250]])
     input_label = np.array([1])
```

```
plt.figure(figsize=(10,10))
plt.imshow(image)
show_points(input_point, input_label, plt.gca())
plt.axis('on')
plt.show()
```

5. Predict with SamPredictor.predict. The model returns masks, quality predictions for those masks, and low resolution mask logits that can be passed to the next iteration of prediction. With multimask_output=True (the default setting), SAM outputs 3 masks, where scores gives the model's own estimation of the quality of these masks. This setting is intended for ambiguous input prompts, and helps the model disambiguate different objects consistent with the prompt. When False, it will return a single mask. For ambiguous prompts such as a single point, it is recommended to use multimask_output=True even if only a single mask is desired; the best single mask can be chosen by picking the one with the highest score returned in scores. This will often result in a better mask.



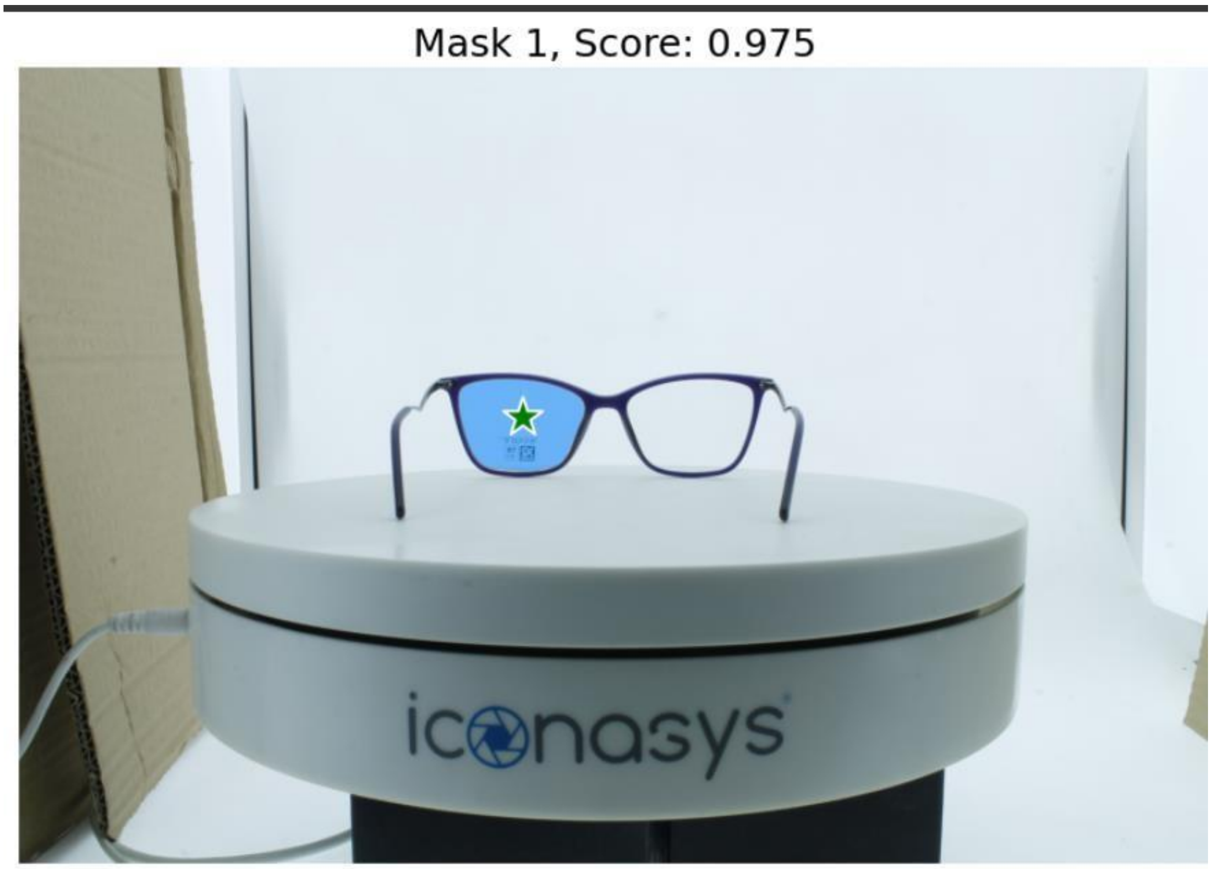
```
[13] masks, scores, logits = predictor.predict(
         point_coords=input_point,
         point_labels=input_label,
         multimask_output=True,
     )
```

With `multimask_output=True` (the default setting), SAM outputs 3 masks, where `scores` gives the r these masks. This setting is intended for ambiguous input prompts, and helps the model disambigu prompt. When `False`, it will return a single mask. For ambiguous prompts such as a single point, it i `multimask_output=True` even if only a single mask is desired; the best single mask can be chosen b returned in `scores`. This will often result in a better mask.

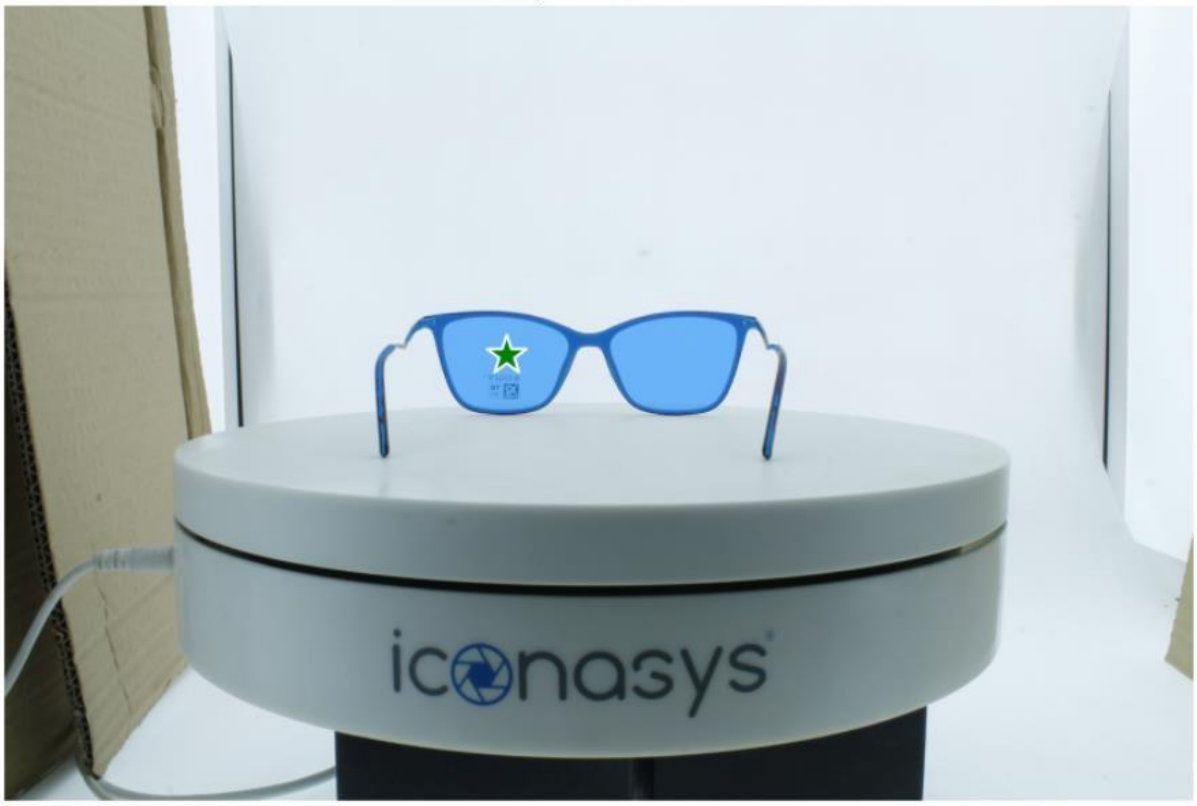
```
[14] masks.shape  # (number_of_masks) x H x W

     (3, 2848, 4272)
```

```
[15] for i, (mask, score) in enumerate(zip(masks, scores)):
         plt.figure(figsize=(10,10))
         plt.imshow(image)
         show_mask(mask, plt.gca())
         show_points(input_point, input_label, plt.gca())
         plt.title(f"Mask {i+1}, Score: {score:.3f}", fontsize=18)
         plt.axis('off')
         plt.show()
```

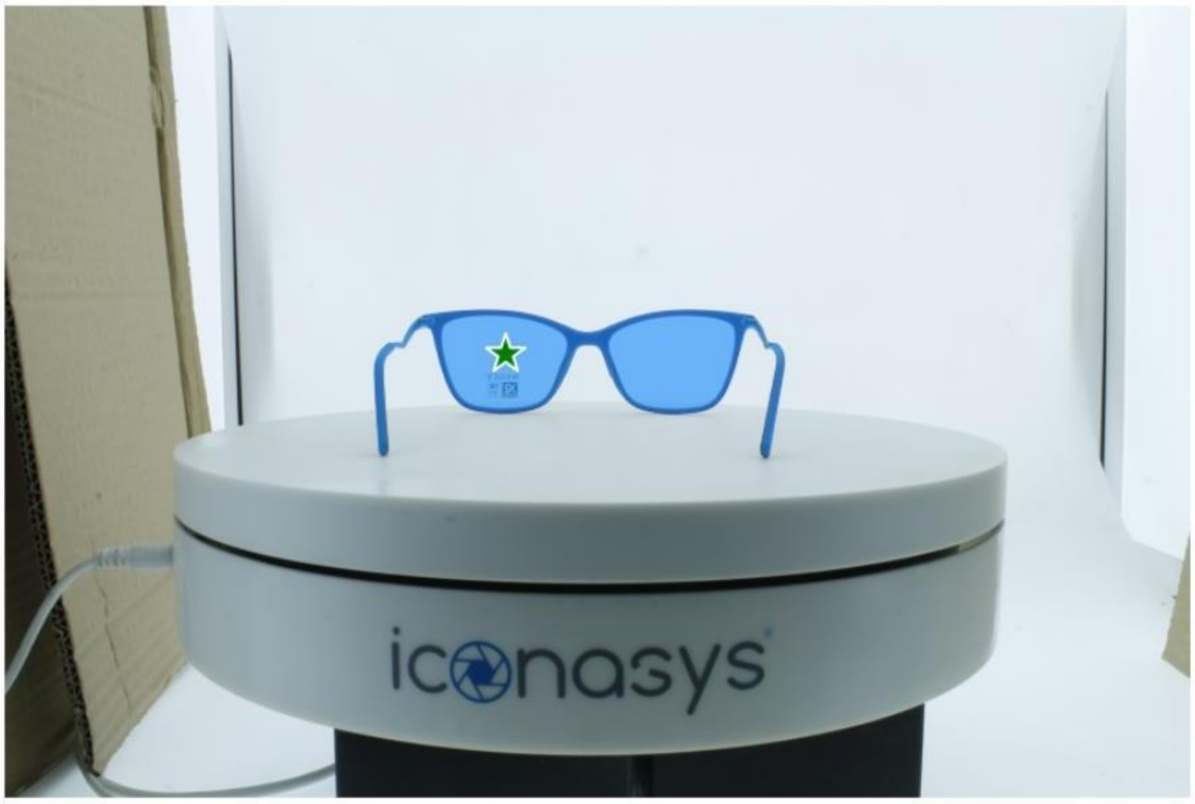
6. And the following masks are:



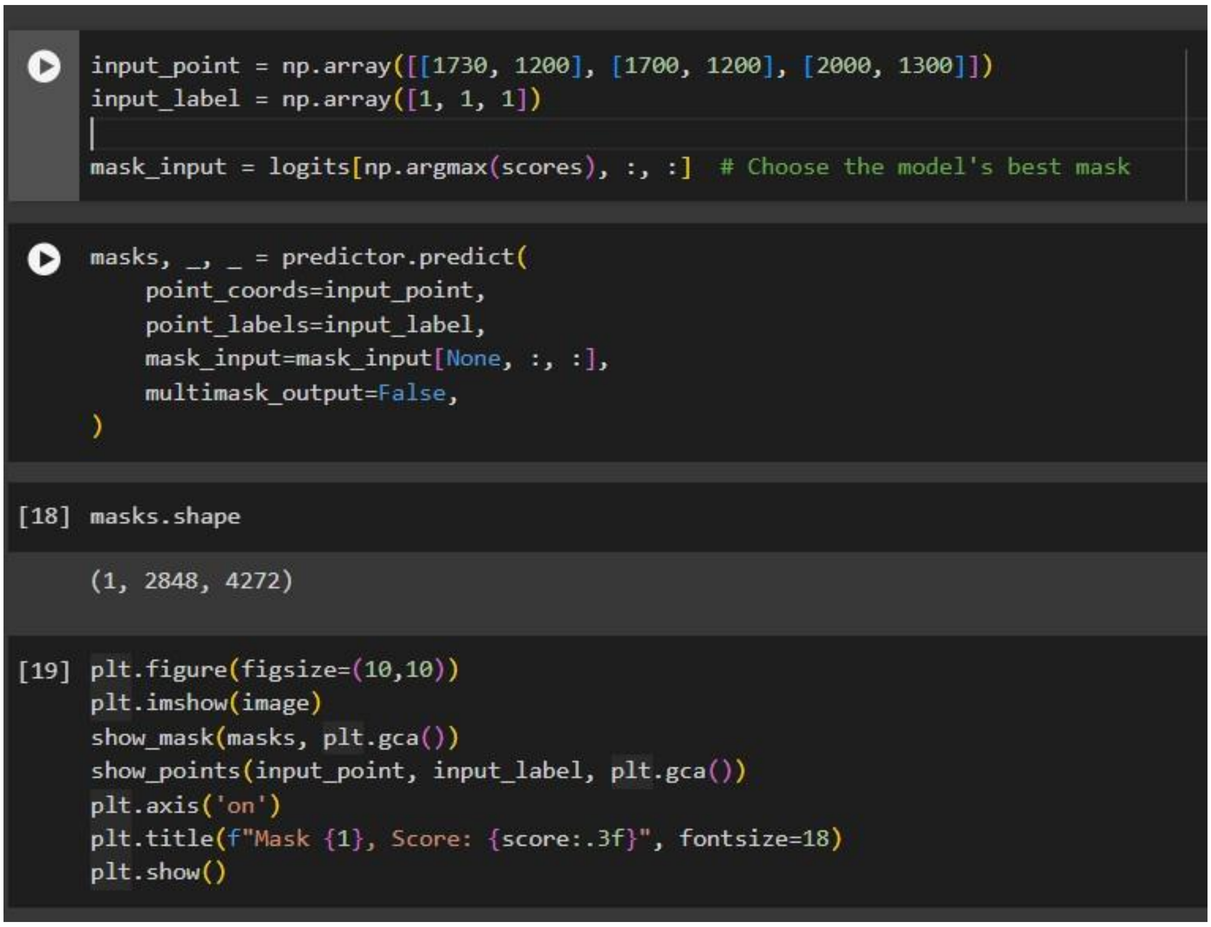
Mask 2, Score: 0.991



Mask 3, Score: 0.953



7. Even though the accuracy was high but with point masking not every image I could mask properly cause the positions/angles differ for every image. So I came up with an another solution; to mask with multiple points and the results are shown below. The single input point is ambiguous, and the model has returned multiple objects consistent with it. To obtain a single object, multiple points can be provided. If available, a mask from a previous iteration can also be supplied to the model to aid in prediction. When specifying a single object with multiple prompts, a single mask can be requested by setting multimask_output=False.

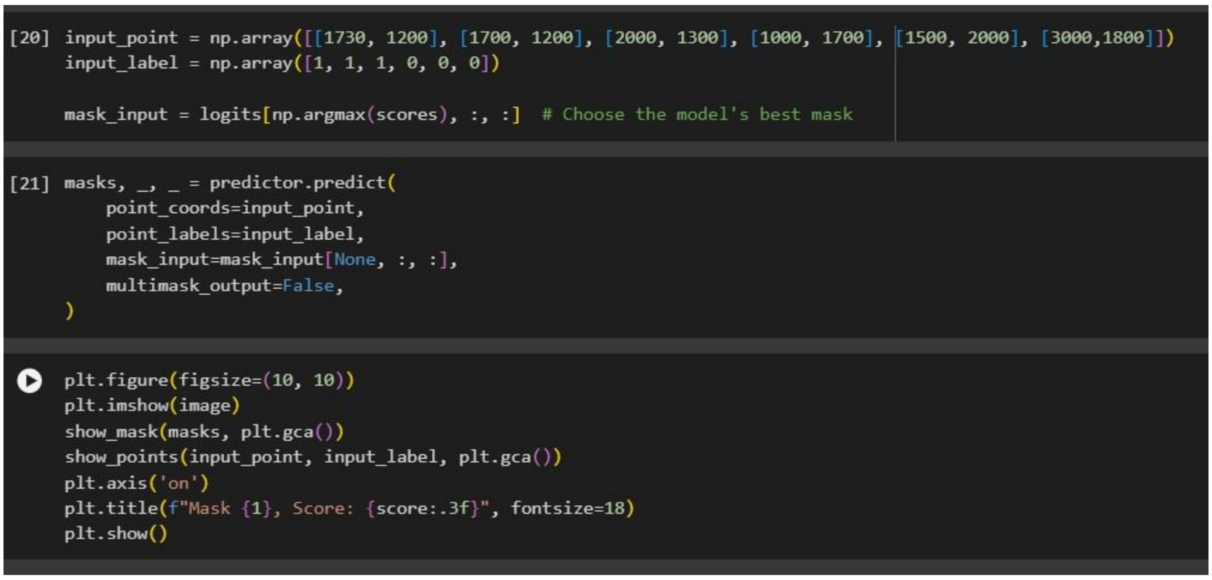


```
input_point = np.array([[1730, 1200], [1700, 1200], [2000, 1300]])
input_label = np.array([1, 1, 1])

mask_input = logits[np.argmax(scores), :, :]  # Choose the model's best mask
```

```
masks, _, _ = predictor.predict(
    point_coords=input_point,
    point_labels=input_label,
    mask_input=mask_input[None, :, :],
    multimask_output=False,
)
```

```
[18] masks.shape

     (1, 2848, 4272)
```

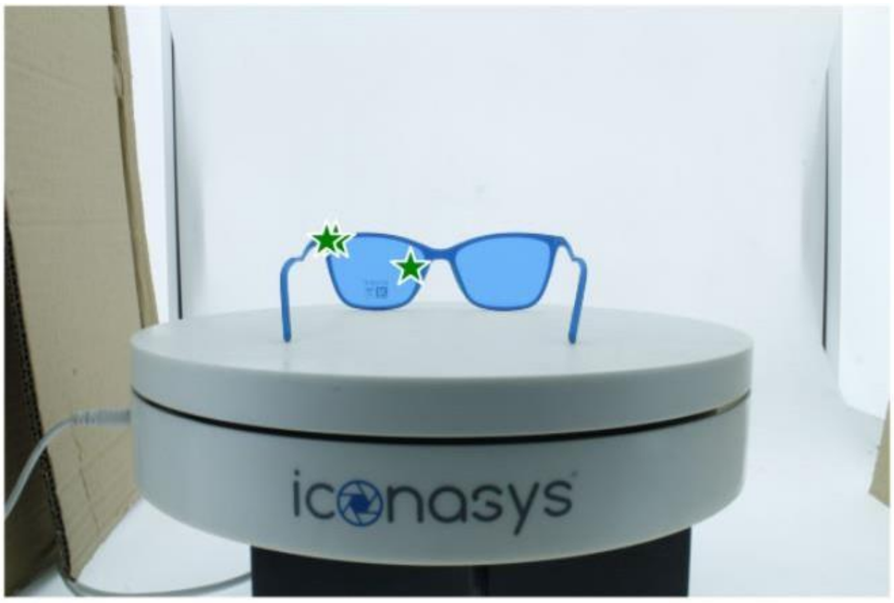
```
[19] plt.figure(figsize=(10,10))
     plt.imshow(image)
     show_mask(masks, plt.gca())
     show_points(input_point, input_label, plt.gca())
     plt.axis('on')
     plt.title(f"Mask {1}, Score: {score:.3f}", fontsize=18)
     plt.show()
```

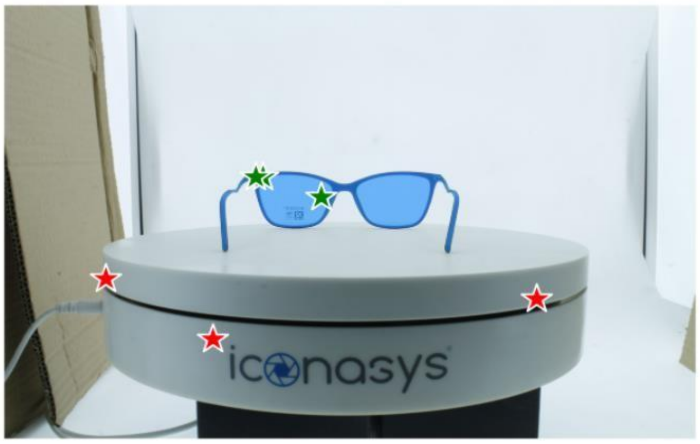


```
[20] input_point = np.array([[1730, 1200], [1700, 1200], [2000, 1300], [1000, 1700], [1500, 2000], [3000,1800]])
     input_label = np.array([1, 1, 1, 0, 0, 0])

     mask_input = logits[np.argmax(scores), :, :]  # Choose the model's best mask
```

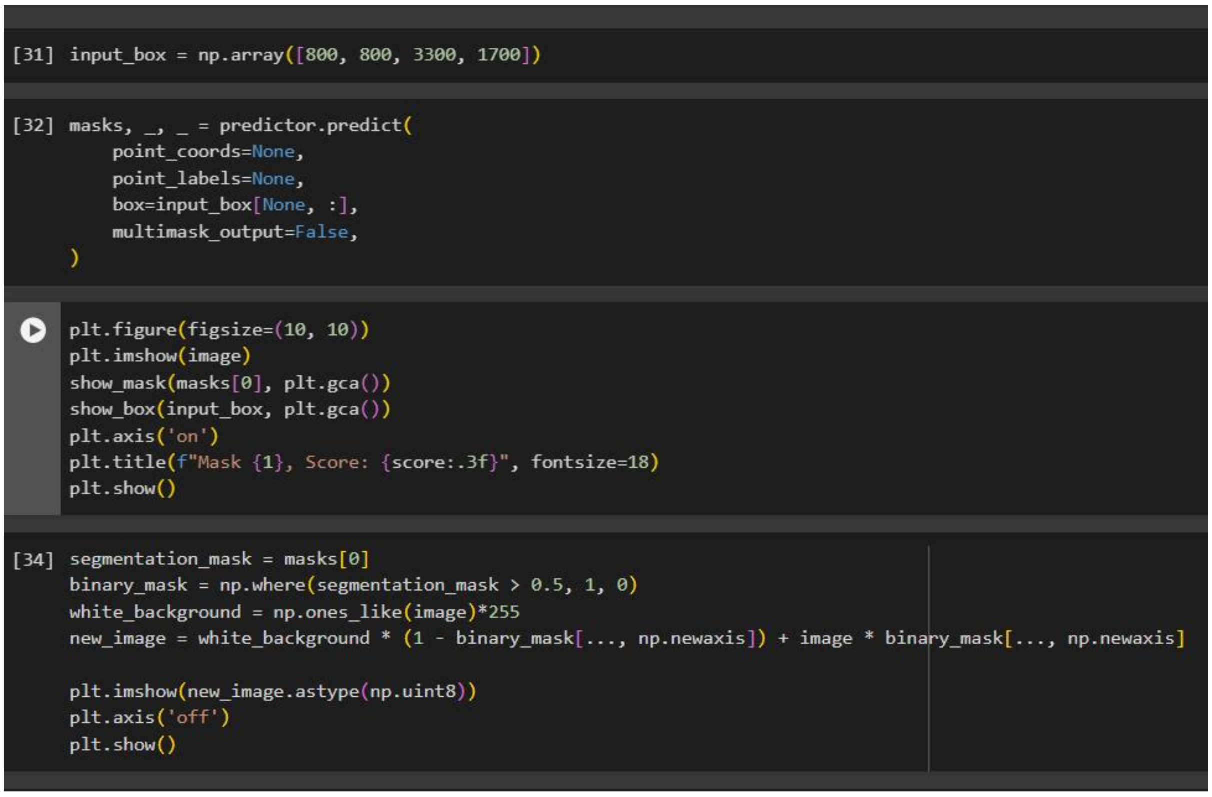
```
[21] masks, _, _ = predictor.predict(
         point_coords=input_point,
         point_labels=input_label,
         mask_input=mask_input[None, :, :],
         multimask_output=False,
     )
```

```
     plt.figure(figsize=(10, 10))
     plt.imshow(image)
     show_mask(masks, plt.gca())
     show_points(input_point, input_label, plt.gca())
     plt.axis('on')
     plt.title(f"Mask {1}, Score: {score:.3f}", fontsize=18)
     plt.show()
```





8. The final and the generalized solution that would work for almost every position of a frame is to specify an object with a box. The model can also take a box as input, provided in xyxy format and the results are shown below.

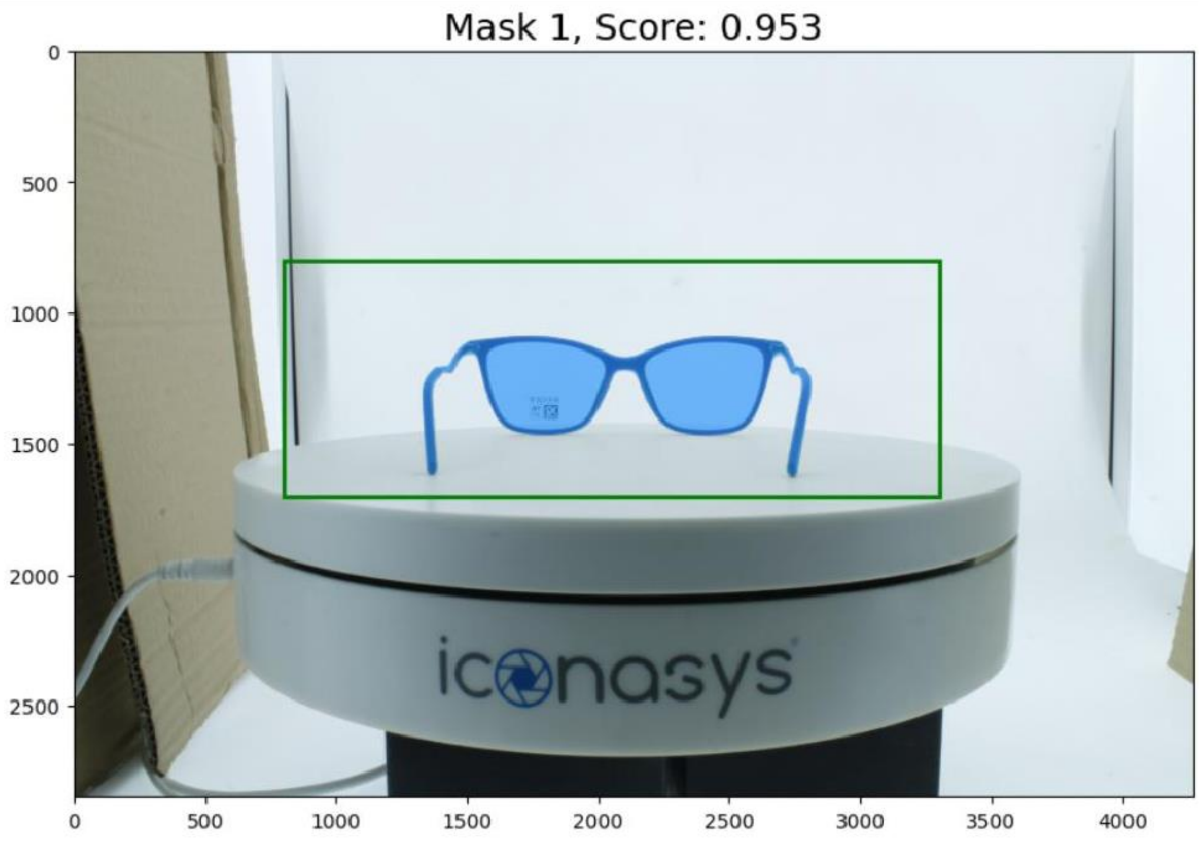


```
[31] input_box = np.array([800, 800, 3300, 1700])
```

```
[32] masks, _, _ = predictor.predict(
         point_coords=None,
         point_labels=None,
         box=input_box[None, :],
         multimask_output=False,
     )
```

```
plt.figure(figsize=(10, 10))
plt.imshow(image)
show_mask(masks[0], plt.gca())
show_box(input_box, plt.gca())
plt.axis('on')
plt.title(f"Mask {1}, Score: {score:.3f}", fontsize=18)
plt.show()
```

```
[34] segmentation_mask = masks[0]
     binary_mask = np.where(segmentation_mask > 0.5, 1, 0)
     white_background = np.ones_like(image)*255
     new_image = white_background * (1 - binary_mask[..., np.newaxis]) + image * binary_mask[..., np.newaxis]

     plt.imshow(new_image.astype(np.uint8))
     plt.axis('off')
     plt.show()
```



Desired Output:

Negotiation:

The dataset has not been trained and tested with the whole given data. And also that it requires a high level computational power and the time being a factor.

Conclusion

The Segment Anything Model offers a powerful and versatile solution for object segmentation in images, enabling you to enhance your datasets with segmentation masks. With its fast processing speed and various modes of inference, SAM is a valuable tool for computer vision applications. To experience labeling your data with SAM, you can use Roboflow Annotate which offers an automated polygon annotation tool, Smart Polygon, powered by SAM.

References

https://blog.roboflow.com/how-to-use-segment-anything-model-sam/