Raindrop Mask Generation

Project Review Presentation

Justin Schlag

23 April 2025

UNIVERSITY OF South Carolina

College of Engineering and Computing

INTEGER TECHNOLOGIES

January: Image Labeling App

Purpose:

•Created to manually label raindrops on lens-distorted images.

•Designed to support dataset creation by getting the coordinate points of each raindrop.

Key Features:

•Interactive GUI with point-and-click segmentation.

•Delete mode for removing unwanted selections.

Image Labeling App

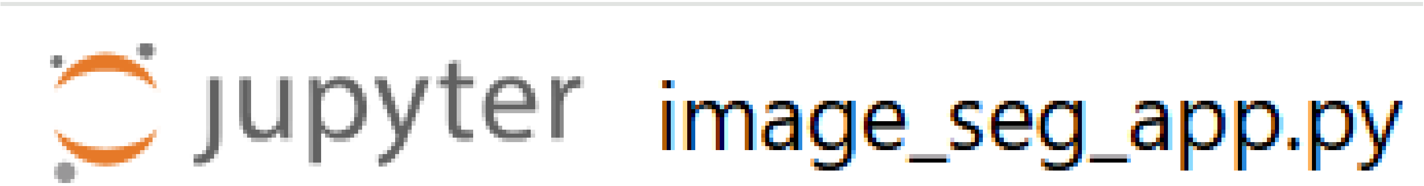
Image Labeling App

```
PROBLEMS 2    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\justi\OneDrive\Desktop\ImageLabelingProject> & C:/Users/justi/anaconda3/envs/sam2_env_new/python.exe c:/Users/just
ject/image_label.py
Marker Coordinates: [(137, 262), (228, 266), (318, 219), (389, 369), (738, 280), (545, 423), (170, 87), (46, 48), (416, 144)]
Segmentation result saved at: segmentation_results\6136_cam3_result.png
```

Image Segmentation using SAM2

- **First Attempt at creating masks**
- **Using Jupyter Notebook**



February: Image Segmentation App

First Successful Image segmentation from my app
Using the SAM2 model: a foundation model for image segmentation, meaning it can identify and outline *any object* in an image

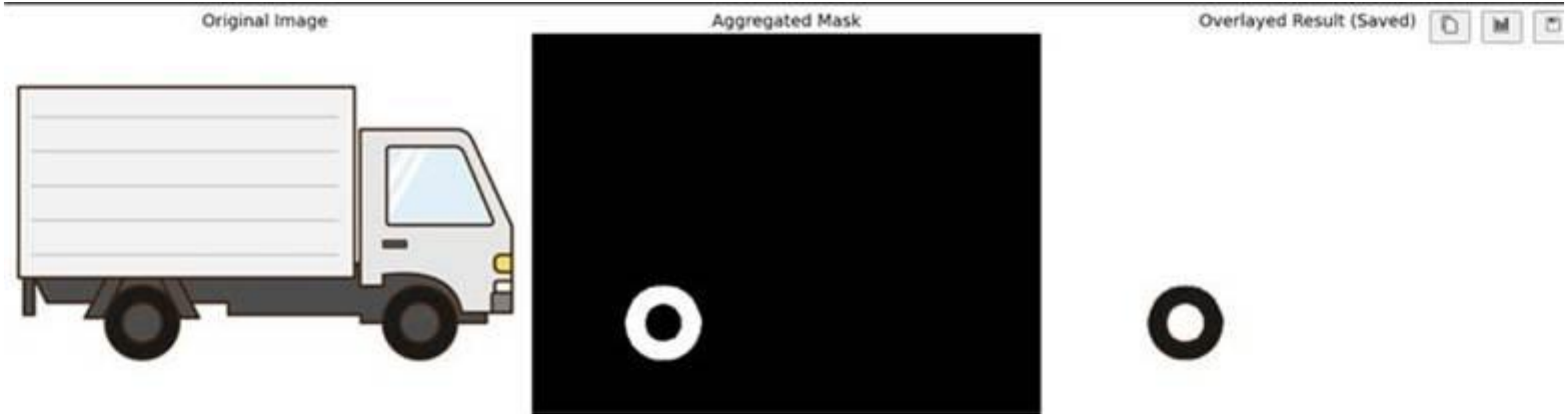


Image Segmentation App

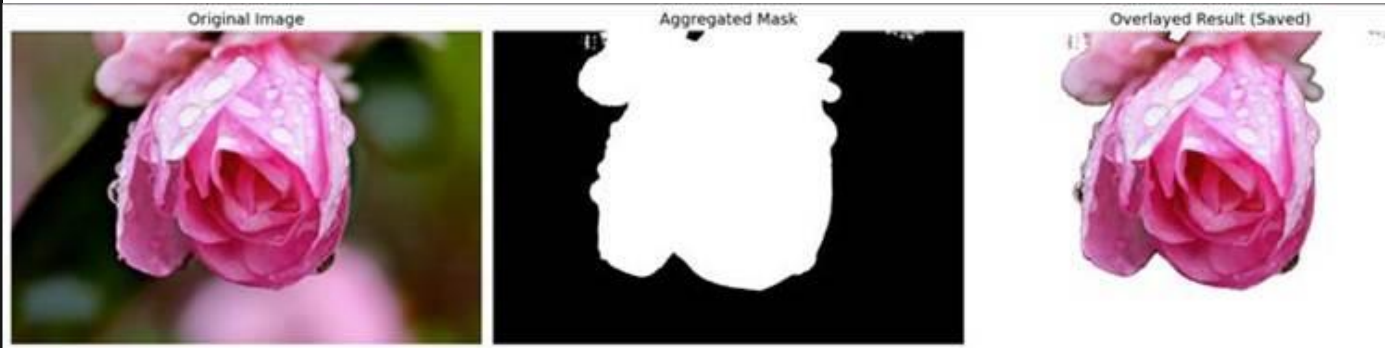
Downfalls:
I had to use my Image Labeling app first, gather all the coordinate points of raindrops, then go to my image segmentation app, and type in each point manually, then submit and wait for results

```
Enter raindrop coordinates in format: x y
Press Enter without typing to finish.

Enter point 1 (x y), or press Enter to finish: 239 238
Enter point 2 (x y), or press Enter to finish: 283 943
Enter point 3 (x y), or press Enter to finish: 193 2393
Enter point 4 (x y), or press Enter to finish: 283 2843
Enter point 5 (x y), or press Enter to finish: 2849 12
Enter point 6 (x y), or press Enter to finish: 4392 23
```

Very Lengthy Process:
Average Mask Time: 5 minutes

Wasn't great with raindrops either:



Updated Data Generation

- **Streamlined Way to get data**
- **I connected my image labeling and my segmentation app into one app, so I no longer had to manually input coordinate points**

Was very good at gathering masks for a few raindrops:

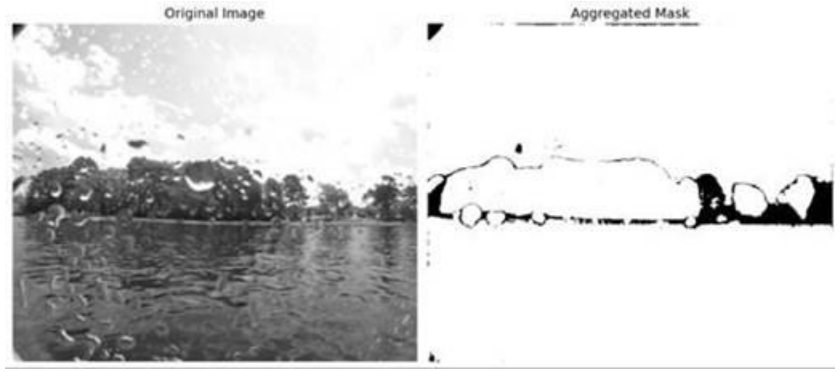
Original Image

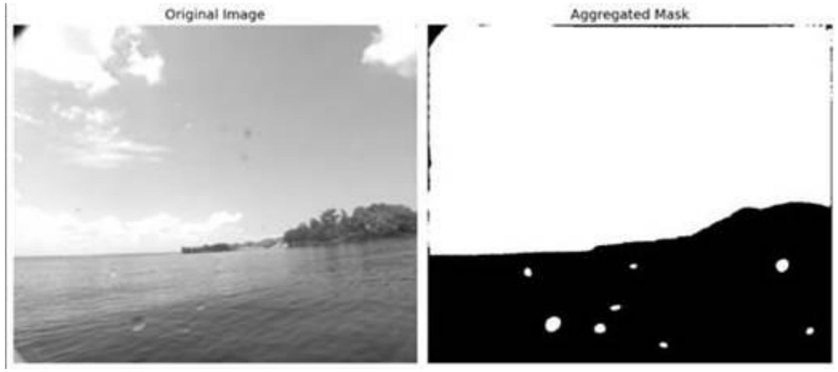
Aggregated Mask

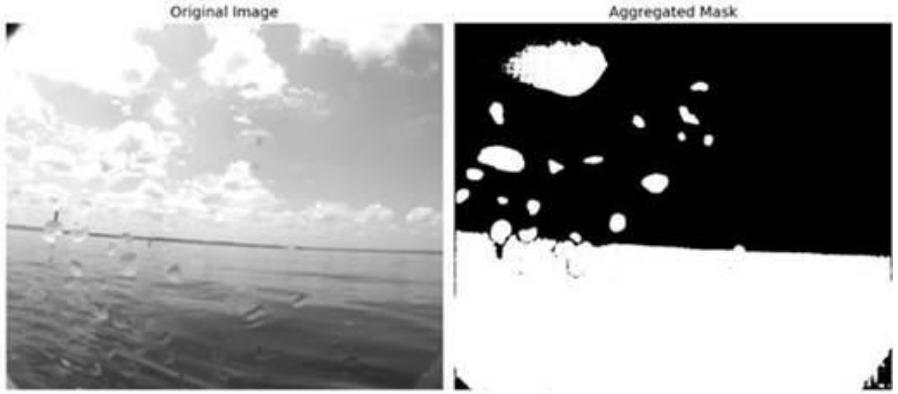
Overlayed Image

Downfalls of the updated app:

- **Bad at masking ALL raindrops**
- **Average time: 1 minute per mask (pretty fast)**





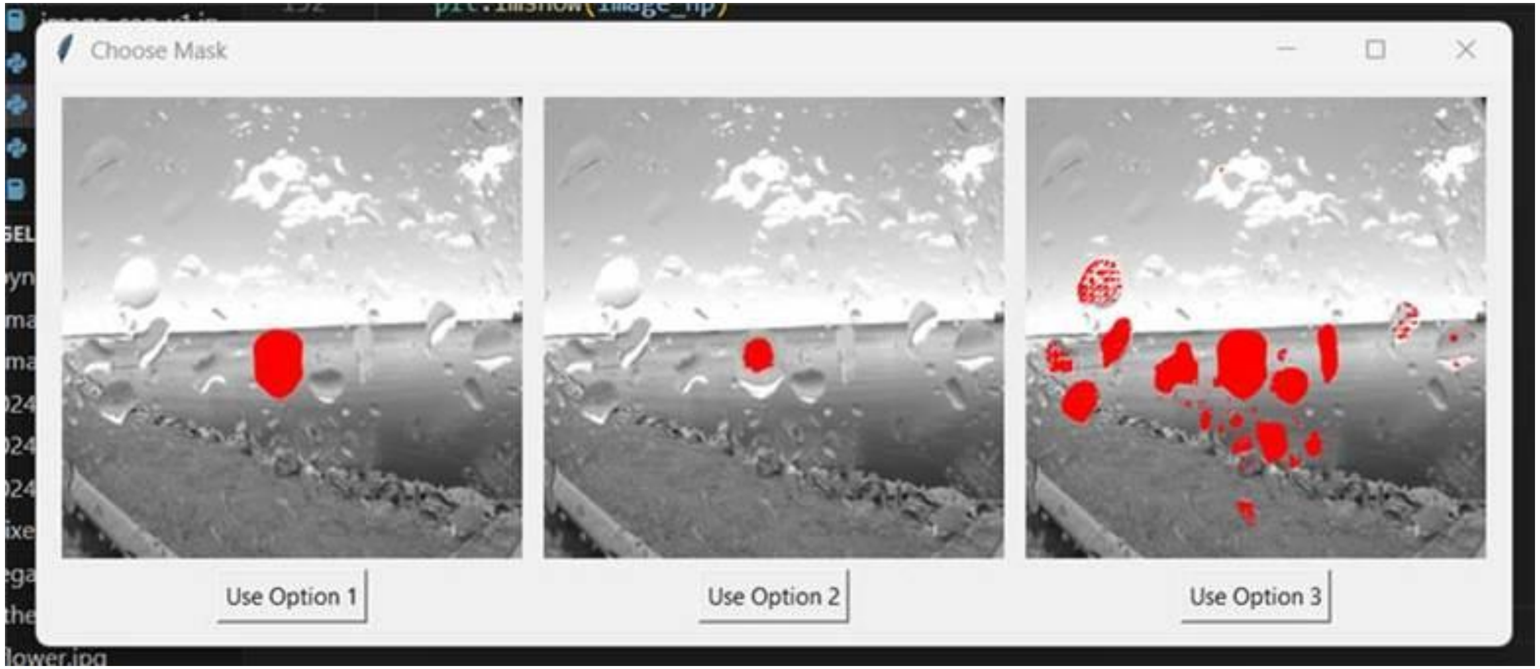


March: Updated (again) Data Generation

I talked to Xeerak, and he made me aware that SAM2 creates 3 masks, and picks the one that is “highest scoring”

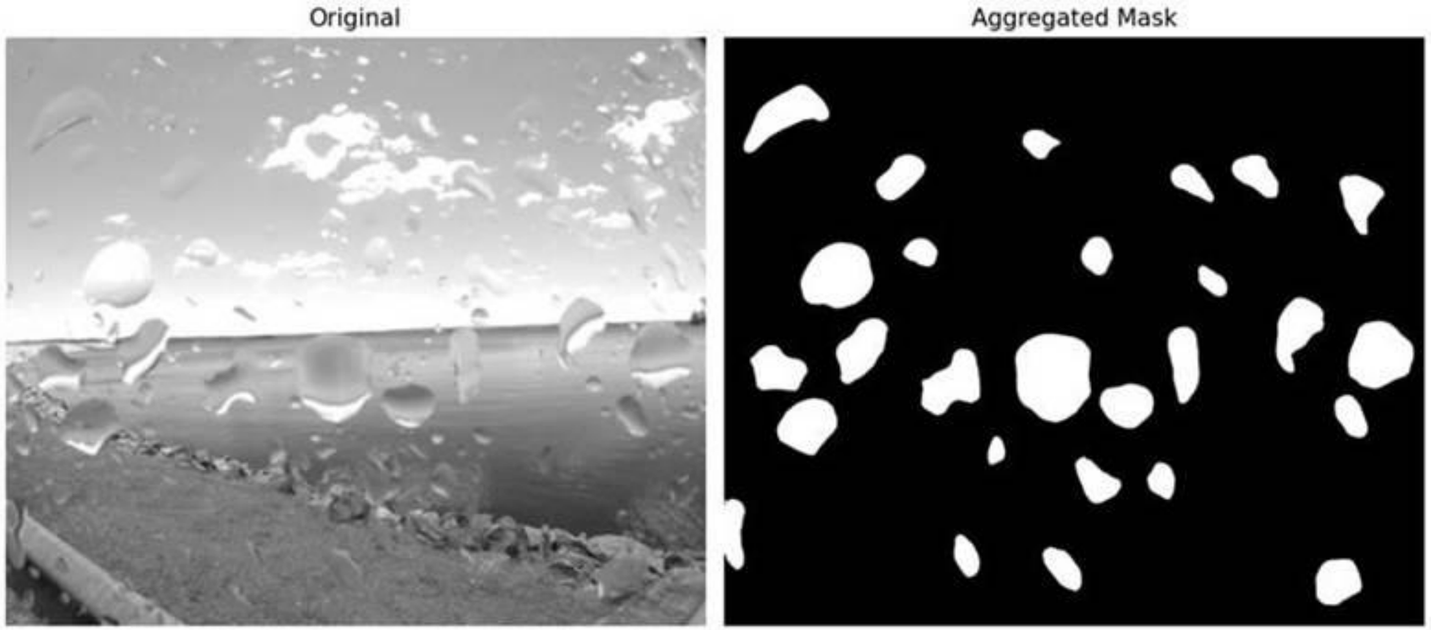
He told me the highest scoring mask wasn’t always the best, leading to a large amount of degradation in my data

So, I updated my program to run through each and every point individually, allowing me to manually choose the best mask for each raindrop.



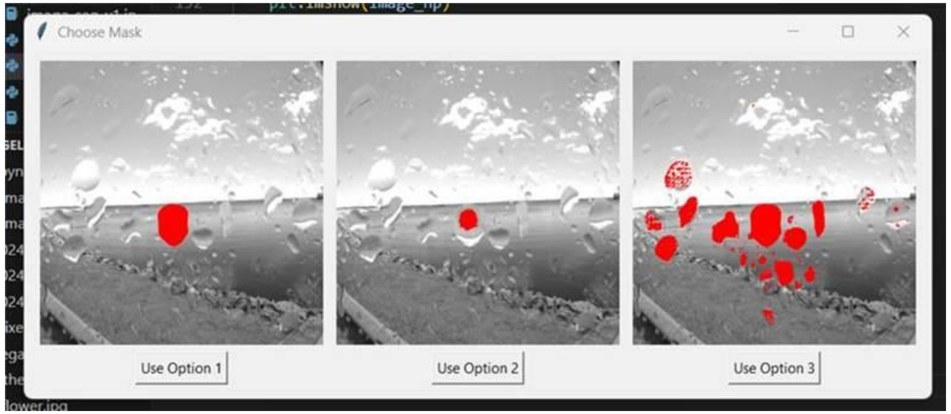
Updated (again) Data Generation

This led to the best results I ever had, but again it took a long time to get the data
Average time: 4 minutes per mask



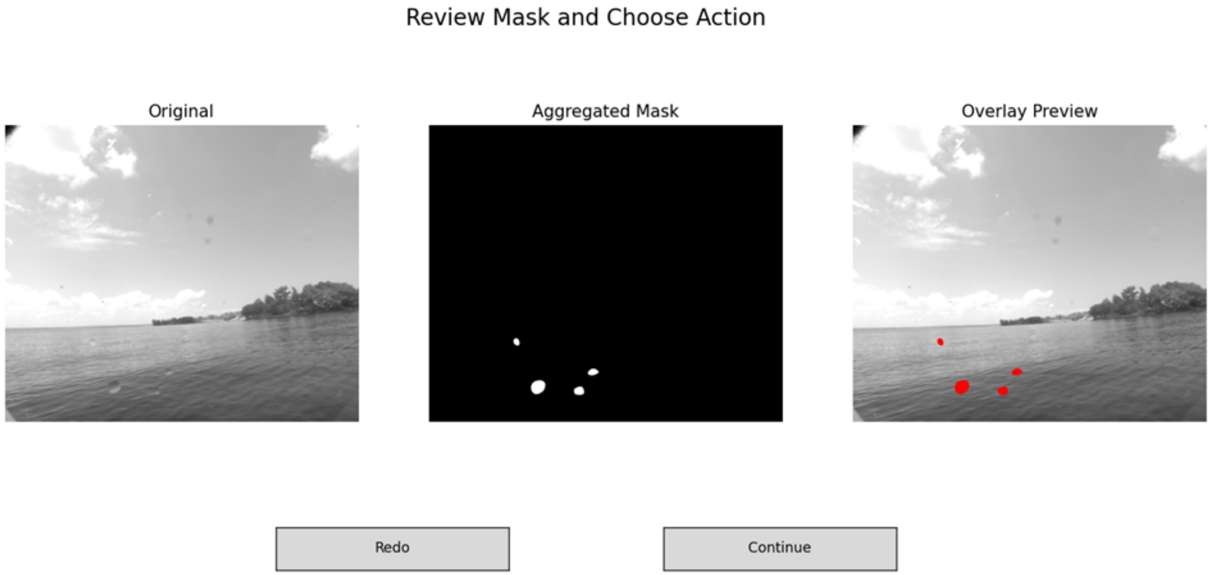
Finalized Data Generation

- **I realized the best mask was ‘Option 1’ 90% of the time.**



- **So I made the program automatically pick ‘Option 1’ for every single raindrop.**

- **At the end I added a redo button after the mask to fix the raindrops that it messed up**



Finalized Data Generation

- **This automated program, with manual reviewing led to the most efficient way of gathering good data**
- **Average time per mask: 30 seconds**

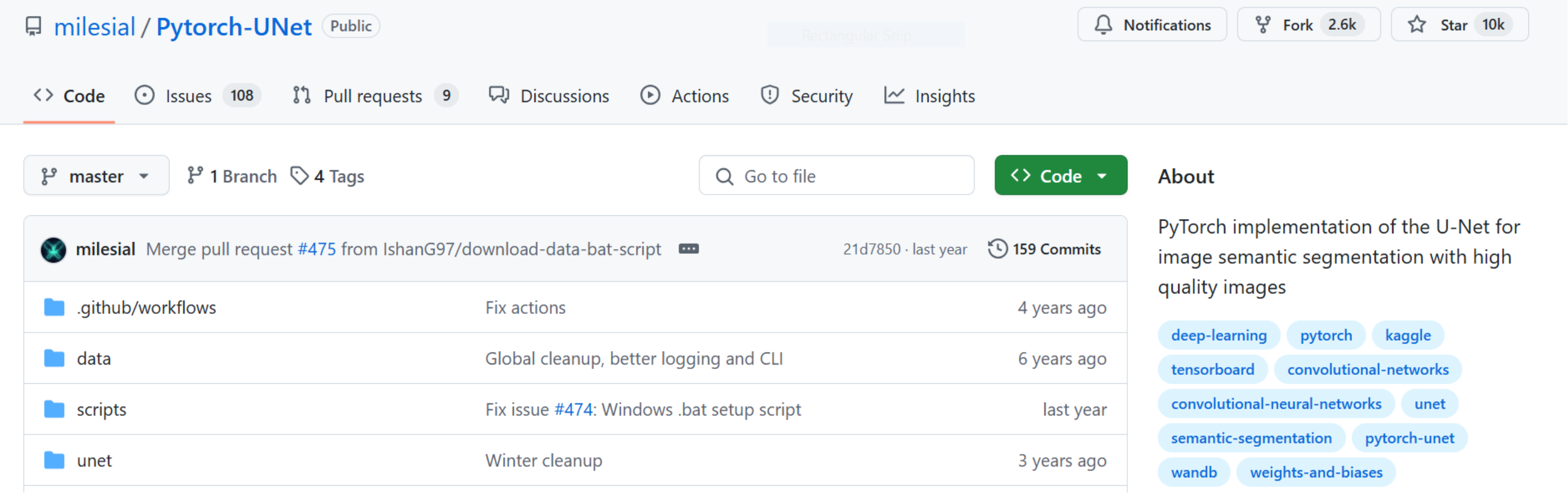
April: Small Updates

- To make the Image segmentation app better, I also included various small changes:
- To make the datapoints less rigid, I utilized gaussian blur: Softer edges, smooth transitions, noise reduced
- Larger and simplified gui, better for manually reviewing data

Training My Own Model:

- **After developing a fast and reliable way to generate raindrop masks, Dr. Tong tasked me with training my own segmentation model**

- **I researched several architectures and decided to implement U-Net, a well- established model for semantic segmentation**



Upcoming:

- I trained the model using the ground truth masks I had previously created for Xeerak
- The model trained over multiple epochs – each one being a full pass over the dataset
- During training I monitored the loss, which tells me how accurate the model's training predictions are
- I trained the model in batches, because training 1 image at a time would be too slow, and training all images at once uses too much memory

```
(venv) PS C:\Users\justi\OneDrive\Desktop\ImageLabelingProject> python train.py
✅ Loaded dataset.
Number of training samples: 59
✅ Model moved to device: cpu

🔁 Starting epoch 1/5
  Batch 1, Loss: 0.7075
  Batch 2, Loss: 0.6798
  Batch 3, Loss: 0.6569
  Batch 4, Loss: 0.6614
  Batch 5, Loss: 0.6496
  Batch 6, Loss: 0.6479
  Batch 7, Loss: 0.6053
  Batch 8, Loss: 0.6138
  Batch 9, Loss: 0.6082
  Batch 10, Loss: 0.6091
  Batch 11, Loss: 0.6003
  Batch 12, Loss: 0.5747
```

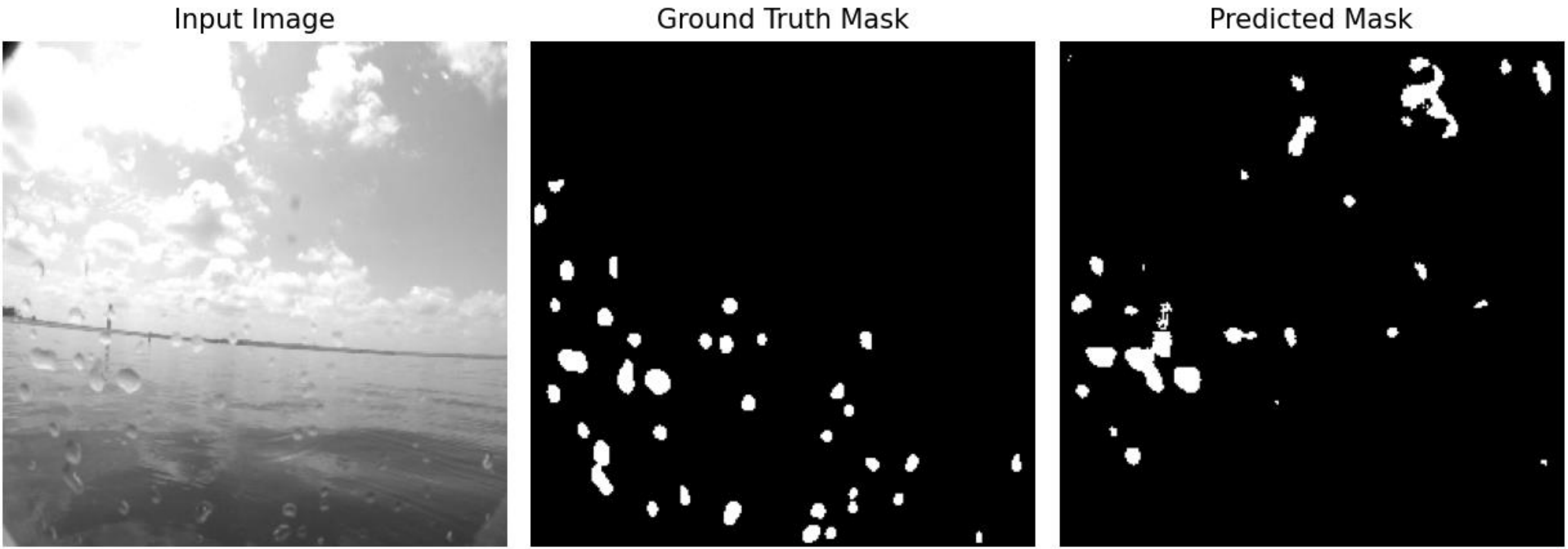
```
  Batch 13, Loss: 0.5727
  Batch 14, Loss: 0.5774
  Batch 15, Loss: 0.5605
  Batch 16, Loss: 0.5492
  Batch 17, Loss: 0.5537
  Batch 18, Loss: 0.5229
  Batch 19, Loss: 0.5331
  Batch 20, Loss: 0.5597
  Batch 21, Loss: 0.5114
  Batch 22, Loss: 0.4989
  Batch 23, Loss: 0.5182
  Batch 24, Loss: 0.5130
  Batch 25, Loss: 0.5112
  Batch 26, Loss: 0.4950
  Batch 27, Loss: 0.4752
  Batch 28, Loss: 0.5239
  Batch 29, Loss: 0.4605
  Batch 30, Loss: 0.4594
✅ Epoch 1 complete – Avg Loss: 0.5670
```

First Model Prediction

```
Batch 22, Loss: 0.3379
Batch 23, Loss: 0.3111
Batch 24, Loss: 0.3467
Batch 25, Loss: 0.3445
Batch 26, Loss: 0.3241
Batch 27, Loss: 0.3107
Batch 28, Loss: 0.3878
Batch 29, Loss: 0.3467
Batch 30, Loss: 0.3478
✅ Epoch 5 complete — Avg Loss: 0.3514
```

I'm using Binary Cross-Entropy Loss with Logits (BCEWithLogitsLoss), which is ideal for pixel-wise binary classification — in this case, identifying each pixel as raindrop (1) or not (0) It also handles the raw outputs directly for stability and efficiency.”

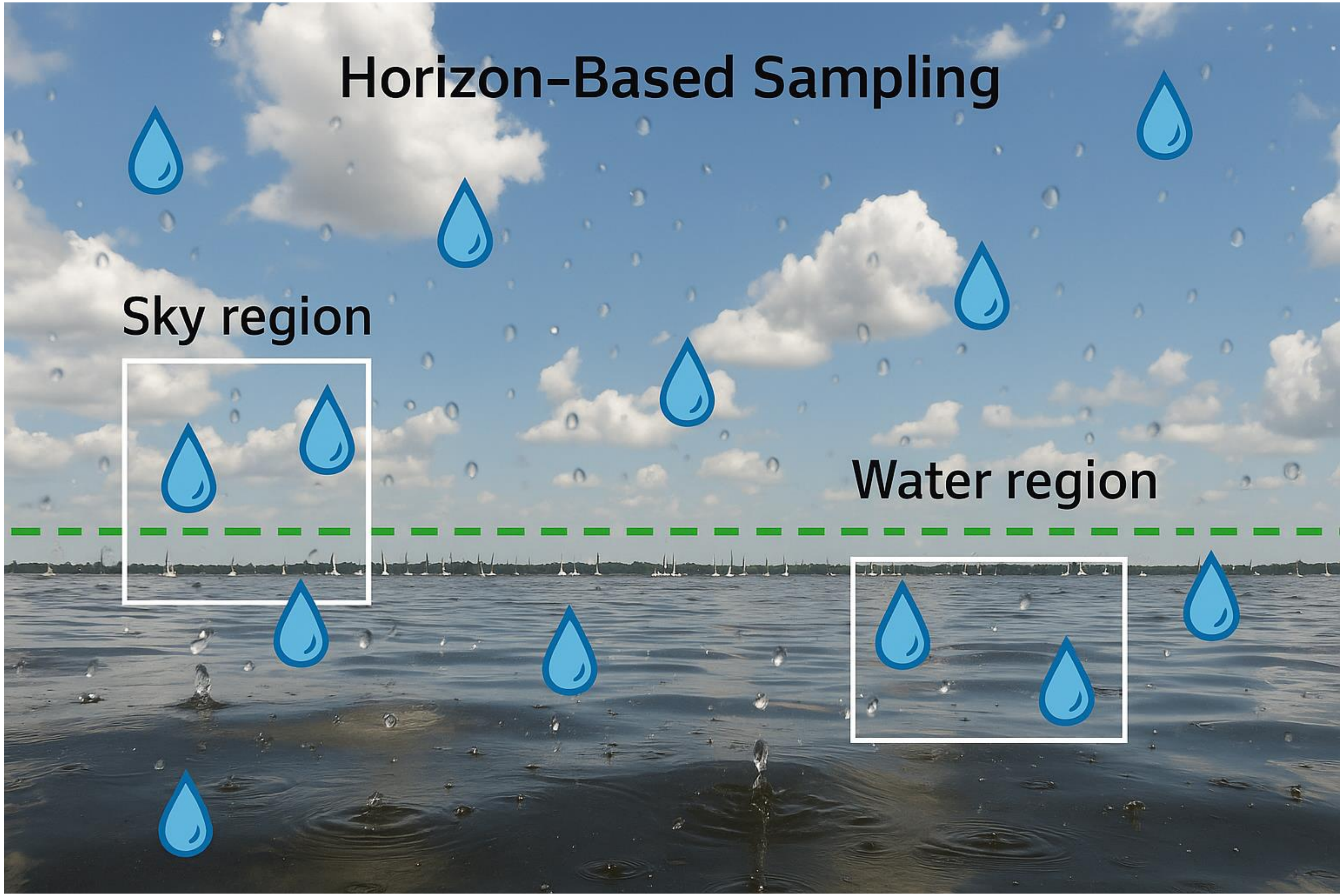
First Generated Mask



Upcoming

- I'm estimating a rough horizon line (around y = 200 pixels down the image) to separate the sky (above) and water (below) in each image. This helps me pull training examples from both regions, which look very different and affect how raindrops appear.

- From these regions, I’m extracting small image crops, or “samples,” that are 256×256 pixels in size. These are the pieces of the image I feed into my model for training. I’m aiming for 200–300 total samples to give the model a variety of lighting, textures, and drop shapes.


Horizon-Based Sampling
Sky region
Water region

Upcoming

- To measure how well my model is learning, I’m setting aside 10 complete images and their corresponding masks as a validation set. These aren't used during training — they help me check if the model can generalize to new, unseen data.
- After preparing this new data, I plan to retrain my segmentation model and then compare the results to the original version. This will help me see if the new approach leads to more accurate raindrop detection, especially across different parts of the scene.