

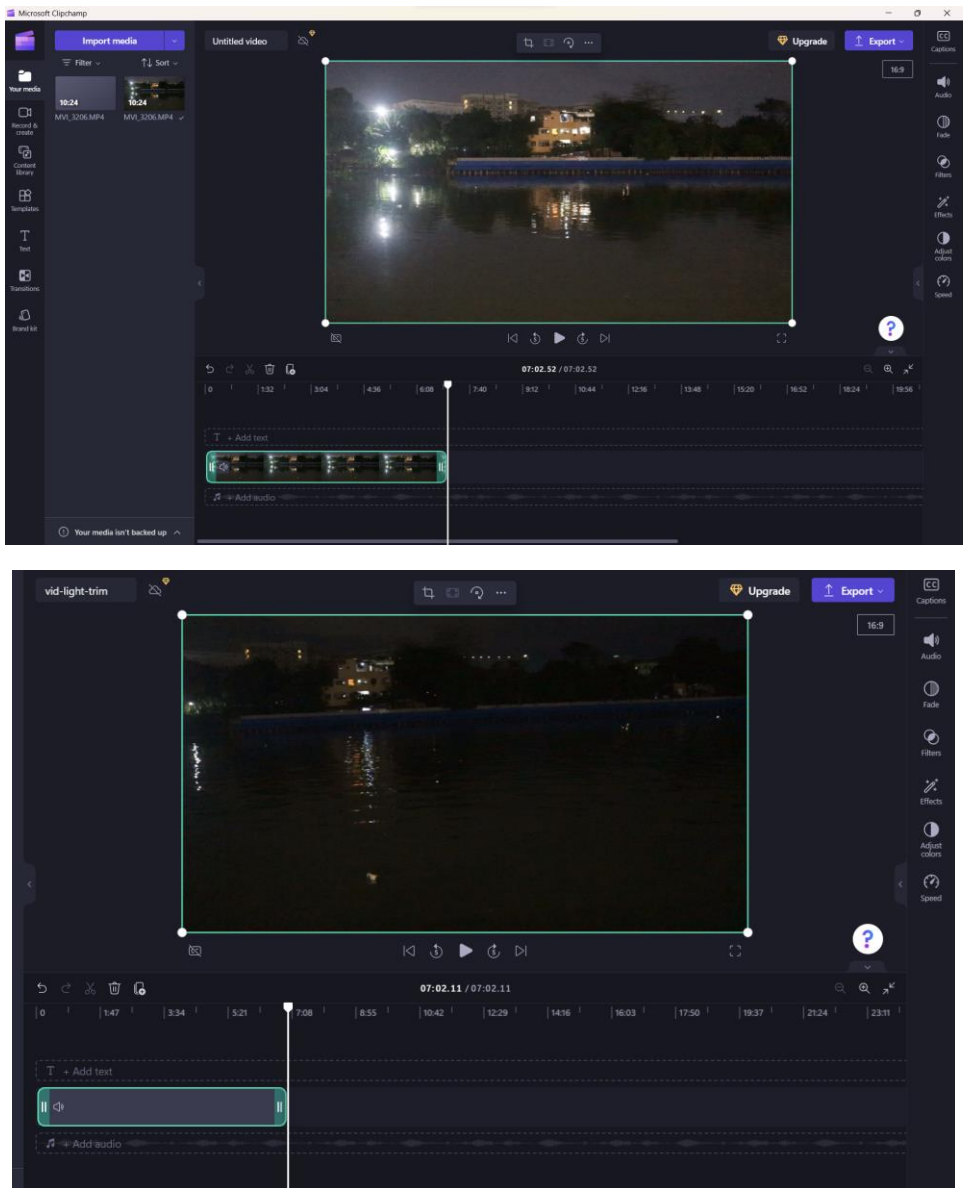
M2 PART1

A. Data Collection

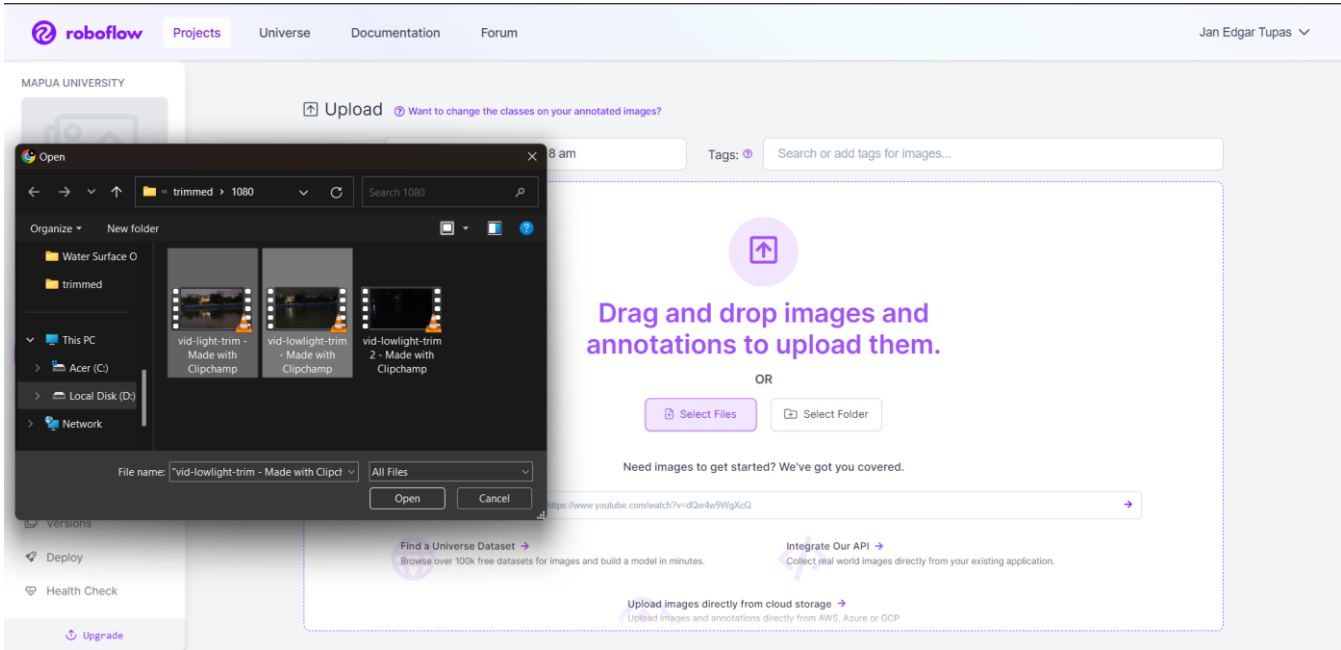
The target object for our project is surface debris floating on the water surface of the Pasig River. The data was collected on the portion of the Pasig River in front of Circuit Park. Our data was collected through video capture.

B. Data Pre-Processing

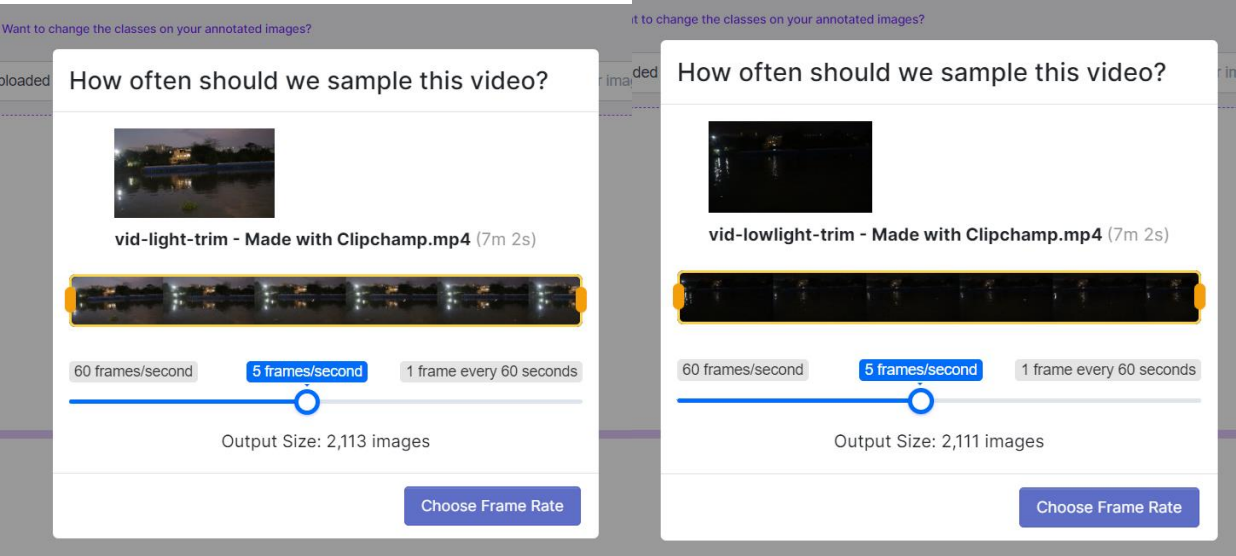
First, we need to trim our recorded videos. We used Microsoft ClipChamp to trim the video. Two videos were used where the first video had sufficient lighting and the second video had low light. The videos were trimmed to 7 minutes each.



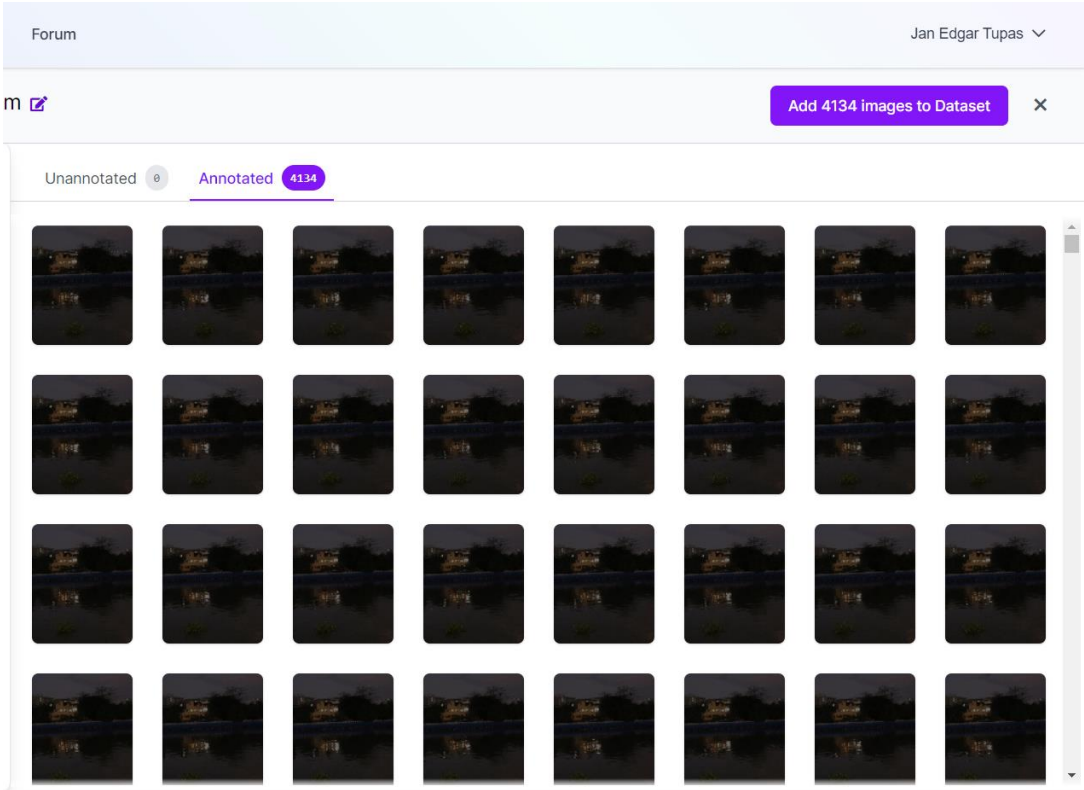
After trimming, we need to extract the images from the videos. We used Roboflow. We first uploaded the videos in preparation for image extraction.



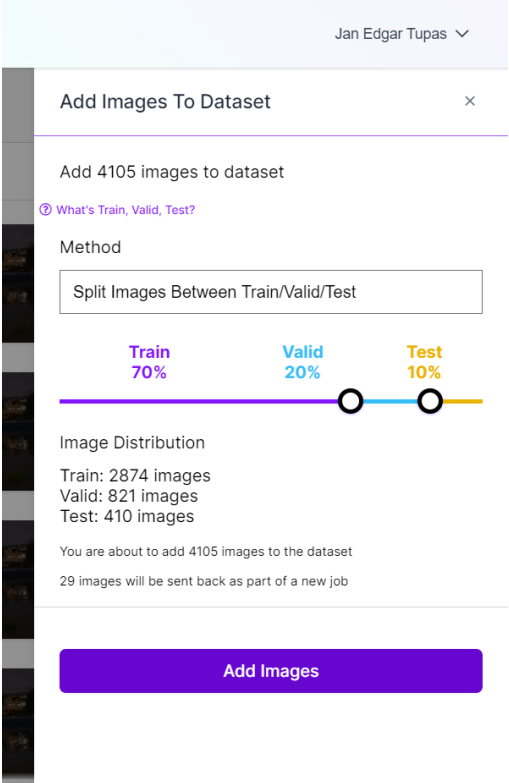
We sampled the video into 4 frames per second. This is so that we can have at least 2000 samples of data. The output is then 2113 images.



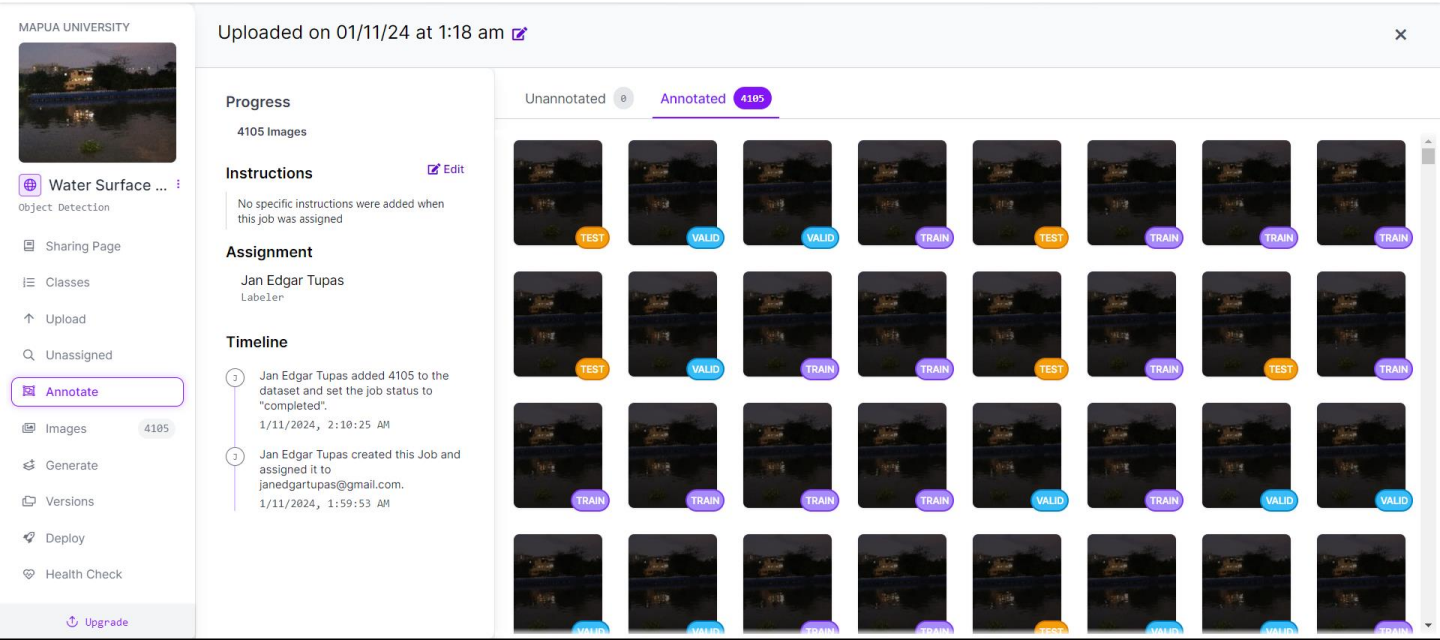
After sampling our video, we now have our images. We will add our images to our dataset. Our dataset will be used for low light image enhancement.



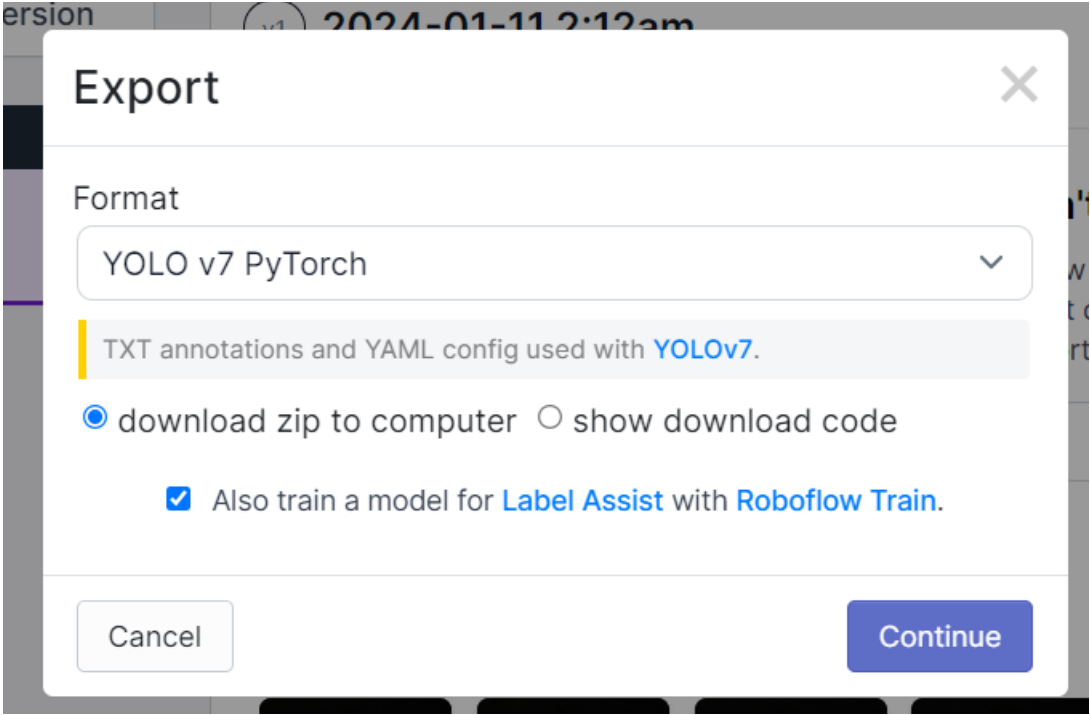
Our dataset will be split into training, testing, and validation data. We can continue on by adding the images.



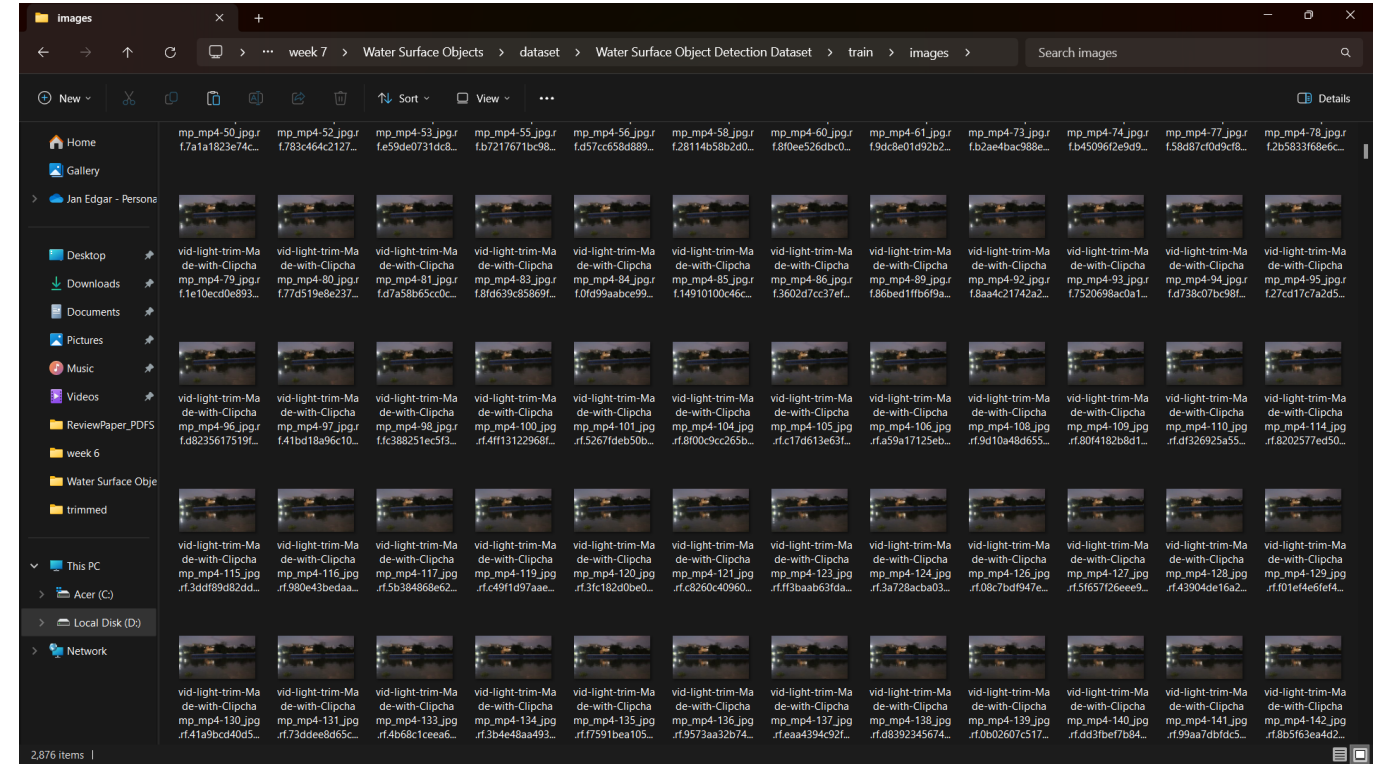
Now our dataset is complete with our images. We will now begin exporting.



We will use YOLO v7 format. The images will be downloaded in zip to the computer.

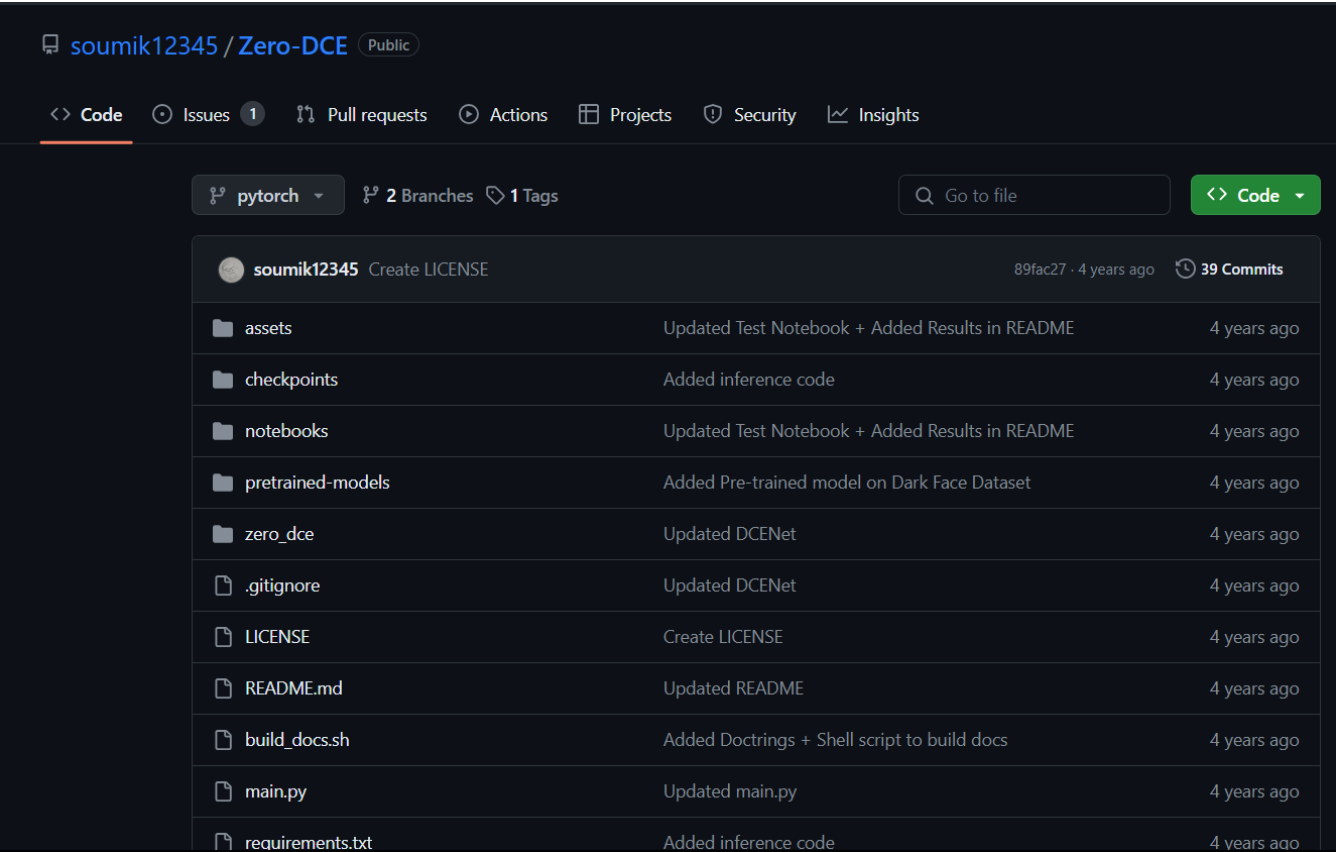


Now we are finished with data preprocessing. We have our final dataset for low light enhancement and surface debris detection on a water surface.



C. Low Light Image Enhancement using Deep Learning

We utilized the deep learning model Zero Reference Deep Curve Estimation (Zero-DCE) from Chunle Guo et al. The model can be accessed through the following link:
<https://github.com/soumik12345/Zero-DCE>



We will now perform low light image enhancement. Using the git clone command, the github repository of the deep learning model is cloned.

```
!git clone https://github.com/soumik12345/Zero-DCE

Cloning into 'Zero-DCE'...
remote: Enumerating objects: 201, done.
remote: Counting objects: 100% (201/201), done.
remote: Compressing objects: 100% (150/150), done.
remote: Total 201 (delta 98), reused 129 (delta 43), pack-reused 0
Receiving objects: 100% (201/201), 7.62 MiB | 11.11 MiB/s, done.
Resolving deltas: 100% (98/98), done.
```

Next we will install wandb. This is needed to get our dataset for training and visualize the performance of the model during training.

```
!pip install -qq wandb
```

2.1/2.1 MB 10.5 MB/s eta 0:00:00

190.6/190.6 kB 14.6 MB/s eta 0:00:00

252.8/252.8 kB 20.7 MB/s eta 0:00:00

62.7/62.7 kB 9.1 MB/s eta 0:00:00

We will import glob to manage our files. Functions from the zero_dce repository will also be imported since we will use them for training and testing.

```
from glob import glob
from zero_dce import (
    download_dataset, init_wandb,
    Trainer, plot_result
)
```

Now we will download a dataset. This will be used for training the model.

```
download_dataset(dataset_tag='dark_face')

Downloading dataset...
Downloading...
From: https://drive.google.com/uc?id=11KaOhxc0h68_NyZwacBoabE16FgPCsnQ
To: /content/Zero-DCE/DarkPair.zip
100%|██████████| 580M/580M [00:11<00:00, 49.6MB/s]
Unpacking Dataset
Done!!!
```


Now we will begin model training. The trainer function is used to initialize the model trainer. Then the image files will be initialized. Then the trained will be built and compiled with out training images.

```
trainer = Trainer()
image_files = glob('./Low/*.png')
print('Number of Images:', len(image_files))
trainer.build_data_loader(image_files)
trainer.compile()
```

Now model training will start. It will train with 200 epochs and a log frequency of 8. Then the model will be saved and accessed later for testing.

```
trainer.train(epochs=200, log_frequency=8)
trainer.save_model('model200_dark_faces.pth')
```

Epoch 1/200

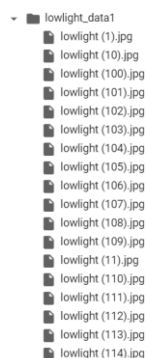
99/? [00:15<00:00, 7.88it/s]

/content/Zero-DCE/zero_dce/trainer.py:81: UserWarning: torch.nn
torch.nn.utils.clip_grad_norm(self.model.parameters(), 0.1)

Here we will save our trained model. We will utilize this model later for low light image enhancement of our own dataset.

```
[6] trainer = Trainer()
    trainer.build_model(pretrain_weights='/content/Zero-DCE/pretrained-models/model200_dark_faces.pth')
```

The lowlight images are uploaded. These will be used for enhancement.



```
lowlight_data1
├── lowlight (1).jpg
├── lowlight (10).jpg
├── lowlight (100).jpg
├── lowlight (101).jpg
├── lowlight (102).jpg
├── lowlight (103).jpg
├── lowlight (104).jpg
├── lowlight (105).jpg
├── lowlight (106).jpg
├── lowlight (107).jpg
├── lowlight (108).jpg
├── lowlight (109).jpg
├── lowlight (11).jpg
├── lowlight (110).jpg
├── lowlight (111).jpg
├── lowlight (112).jpg
├── lowlight (113).jpg
└── lowlight (114).jpg
```

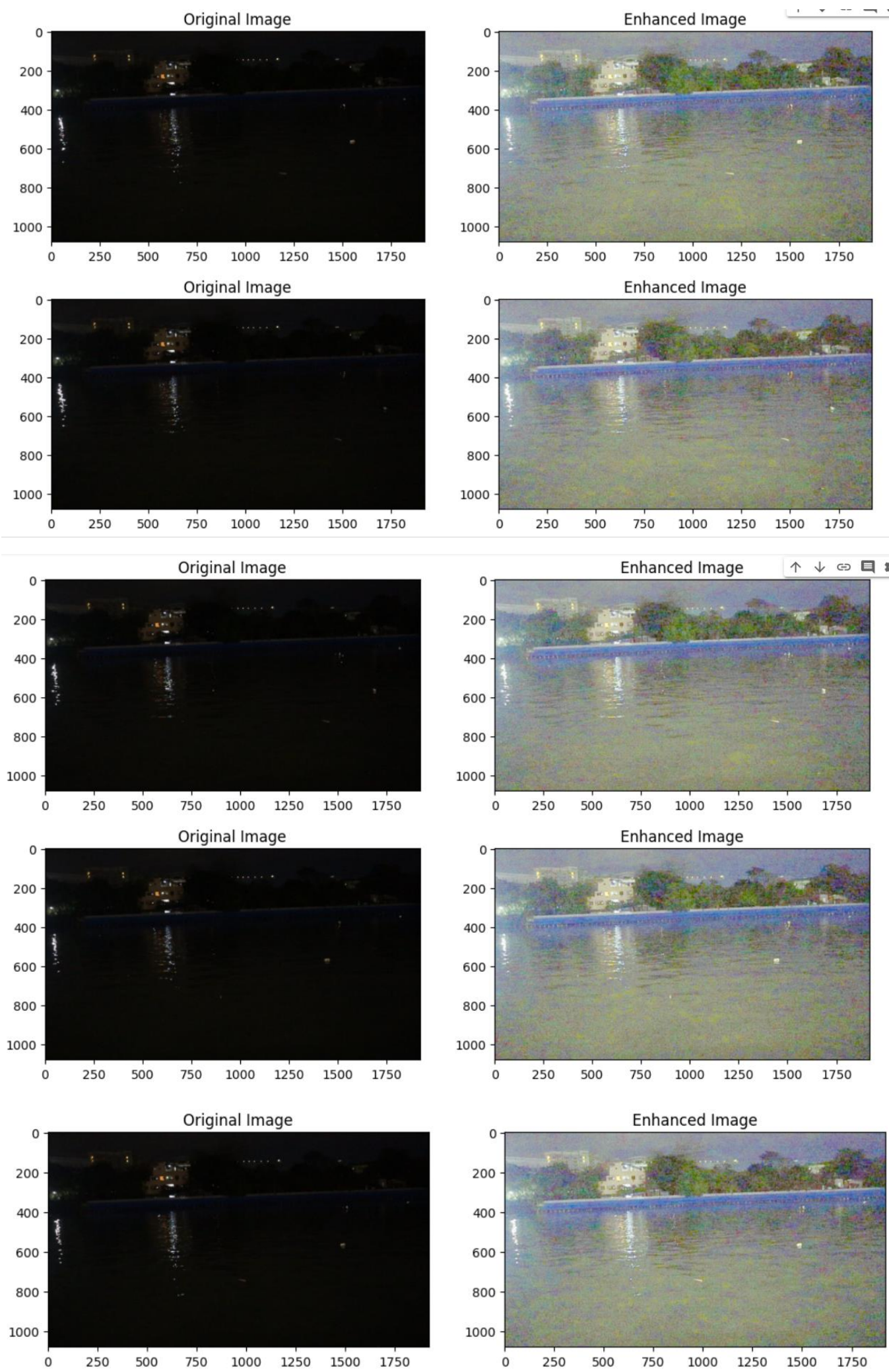
The images are stored in a list. Then they are shuffled.

```
[11] image_filesdl = glob('/content/lowlight_data1/*.jpg')
    random.shuffle(image_filesdl)
```

Here we apply lowlight enhancement on the images using the model. The results are stored in enhancedimg directory. We can visualize the results using the plot_result function and compare the outcomes.

```
output_directory = '/content/enhancedimg'
os.makedirs(output_directory, exist_ok=True)
i = 0
for image_filedl in image_filesdl:
    image, enhanced = trainer.infer_gpu(image_filedl, image_resize_factor=1)
    # Convert NumPy array to PIL Image
    enhanced_pil = Image.fromarray((enhanced * 255).astype(np.uint8))
    # Save the enhanced image locally
    enhanced_pil.save(os.path.join(output_directory, f'enhanced_image_{i}.png'))
    i += 1
```

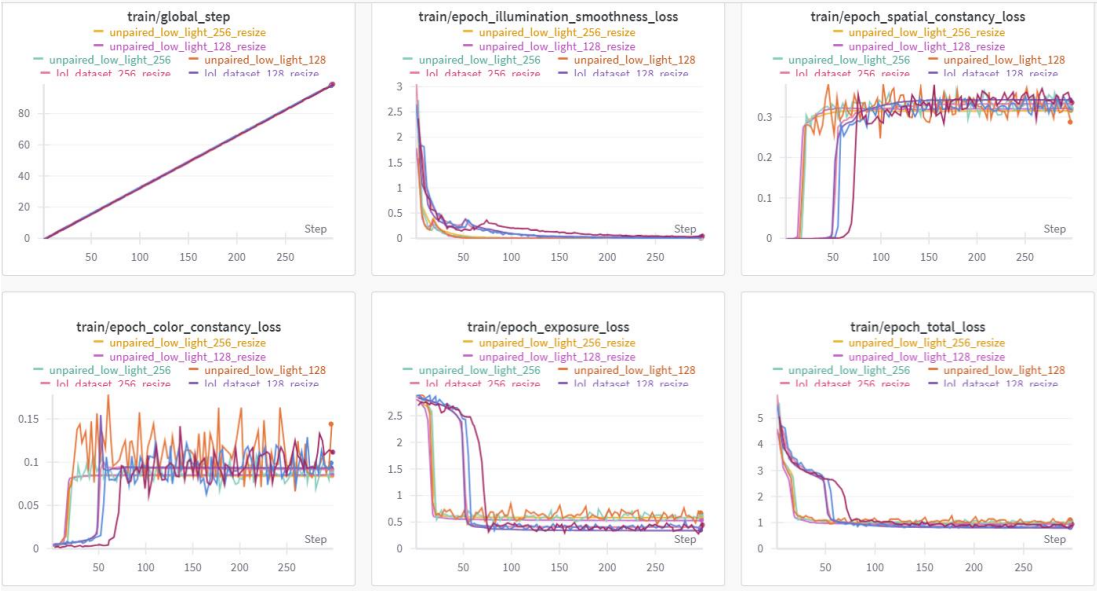
The enhanced images are now shown. When comparing, we can see that the enhanced images now have increased and the contents are visible.



We can get the luminance percentage using our luminance function. Here our average luminance percentage is 55.65%.

Average Luminance Percentage of All Photos: 55.65055615997947%

We now show the results of the model during training. These metrics show the performance of the model when training to enhance low light images.



We can get the Root-mean-square deviation as a result from the model. Here we have an RMSE of 68.53%. The RMSE provides a size of errors in the model's prediction. This result signifies that the model's error in predicting the proper lighting for the images are at 68.53%. We note that with the deep learning enhanced images, the deterioration in the quality of the images is observable compared to images with proper lighting. With this, we would suggest and prefer to use images with proper lighting.

RMSE : 68.53%