Introduction:

This project is about detecting the Drivable area and detecting lanes on the road. This project is mainly about finetuning the model with the dataset we generated using stable diffusion.

Data Preprocessing:

1. Random Perspective Transformation:
   This transformation simulates changes in the camera's perspective, including rotation, scaling, shearing, and translation. It is applied with random parameters:
   - **degrees:** Random rotation in the range of -10 to 10 degrees.
   - **translate:** Random translation in the range of -0.1 to 0.1 times the image dimensions.
   - **scale:** Random scaling in the range of 0.9 to 1.1 times the original size.
   - **shear:** Random shearing in the range of -10 to 10 degrees.
   - **perspective:** A slight random perspective distortion.
2. HSV Color Augmentation:
   - This changes the hue, saturation, and value of the image.
   - Random gains for hue, saturation, and value are applied.
   - The hue is modified by rotating the color wheel.
   - The saturation and value are adjusted by multiplying with random factors.
   - This helps to make the model invariant to changes in lighting and color variations.
3. Image Resizing:
   - If the Images are not in the specified size, the images are resized to a fixed size (640x360) using cv2.resize.
4. Label Preprocessing:
   - The labels (segmentation masks) are thresholded to create binary masks. This means that pixel values are set to 0 or 255 based on a threshold (usually 1 in this case).
   - The binary masks are also inverted to create a binary mask for the background.
   - These binary masks are converted to PyTorch tensors for use in training the semantic segmentation model.

Entire dataset is not loaded at the same time since this may cause memory overload error.

Loading Pretrained Parameters:

The pretrained model is loaded using pytorch. The entire network is loaded with its pretrained weights.

```
import TwinLite as net


model = net.TwinLiteNet()
model = torch.nn.DataParallel(model)
model = model.cuda()
model.load_state_dict(torch.load('best.pth'))

```

```
<All keys matched successfully>
```

Model Fine-Tuning:

We have Fine-Tuned the pretrained model. We did not add any new layers to the model. We trained the whole model on our dataset.

Hyperparameters:

Learning rate = 5e-4

Epochs = 8

Weight decay = 5e-4

```
from tqdm import tqdm
from loss import TotalLoss

lr = 5e-4
optimizer = torch.optim.Adam(model.parameters(), lr, (0.9, 0.999), eps=1e-08, weight_decay=5e-4)

criteria = TotalLoss()
```

A polynomial Learning rate Scheduler was used to dynamically decrease the learning rate.

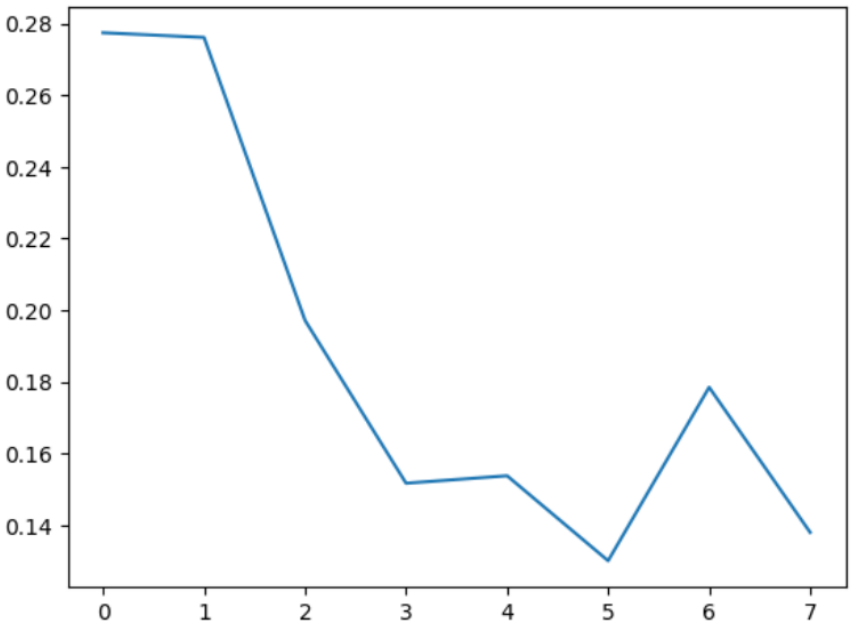
```
def poly_lr_scheduler(args, optimizer, epoch, power=2):
    lr = round(args["lr"] * (1 - epoch / args["max_epochs"]) ** power, 8)
    for param_group in optimizer.param_groups:
        param_group['lr'] = lr

    return lr
```

Results:

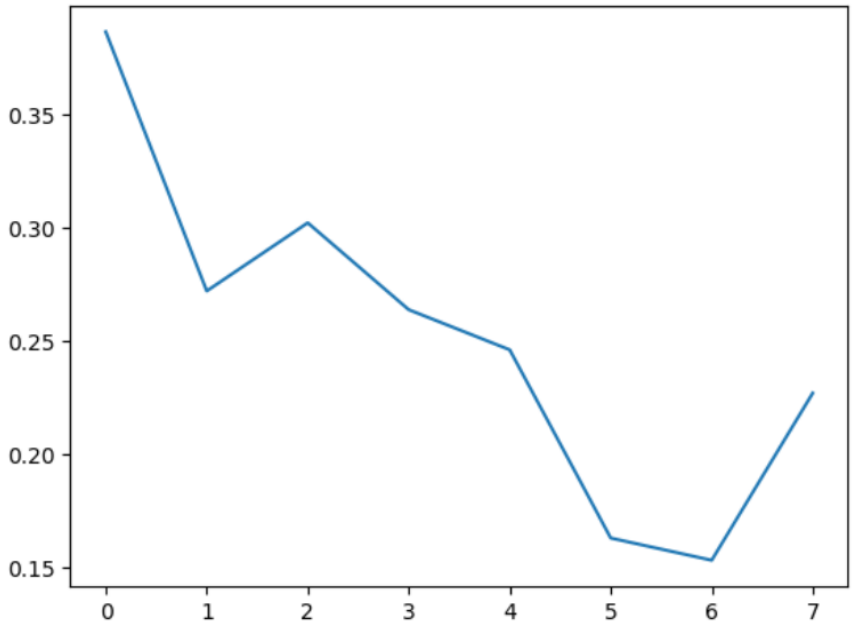
Training Loss:

Training Loss decreases over 8 epochs



Validation Loss:

Validation loss decreases over 8 epochs.



- Evaluation metrics are pixel accuracy and IoU(Intersection over Union).
- We have achieved an accuracy of 95.9% for the Driving area segment.
- We have achieved an accuracy of 98.4% for the Lane Line segment.
- An average of 97.15%-pixel accuracy is achieved which is comparable to the original model's accuracy.

Links:

1. Click here to go the repository that contains the code we have worked on.

2. Click here to go the Google Colab where we worked on.