
Report on TwinLiteNet: An Efficient and Lightweight Model for Drivable Area and Lane Segmentation in Self-Driving Cars

Bal Narendra Sapa
University of New Haven
New Haven, CT 06511
bsapa1@unh.newhaven.edu

Ajay Kumar Jagu
University of New Haven
New Haven, CT 06511
ajagu1@unh.newhaven.edu

1 Introduction

This project focuses on enhancing the efficiency of autonomous vehicles by refining a model for drivable area detection and lane identification. The objective is to tailor the model using a dataset generated through stable diffusion. Recognizing the significance of accurate environment perception in autonomous driving, the project employs deep learning techniques, specifically semantic segmentation, to label each pixel in an image with a semantic class like road or vehicle. Common models for semantic segmentation and lane detection, such as UNet, SegNet, LaneNet, and SCNN, often demand high computational resources and are unsuitable for low-powered embedded systems in autonomous vehicles. In response to this limitation, the proposed architecture in original paper draws inspiration from ESPNet, incorporating depthwise separable convolutions and a Dual Attention Network. Distinctively, the network features two decoder blocks, akin to YOLOP, aiming for high processing speed and ease of implementation on embedded hardware.

2 Dataset

2.1 Description

This dataset contains images for Drivable Area segmentation and Lane detection. All the images are generated using Stable diffusion in Google Colaboratory. This dataset is around 90 Megabytes. The project we are working on has two label outputs for each sample. And these outputs are overlayed on the original image. Link to our project's repository. Click on the link below

<https://github.com/balnarendrasapa/road-detection>

The dataset that we annotated is pushed into GitHub with Git LFS and OneDrive. (GitHub) Link to the dataset. Click on the link below

<https://github.com/balnarendrasapa/road-detection/tree/master/datasets>

(OneDrive) Link to the dataset. Click on the link below

https://unhnewhaven-my.sharepoint.com/:u:/g/personal/bsapa1_unh_newhaven_edu/EZI6lZfbI-tKj_lwzWy92DgBo3TKj5ffd_0j2DD9jEzJgA?e=ACB0aP

Link to the Stable diffusion ipynb file. Click on the link below

https://github.com/balnarendrasapa/road-detection/blob/master/stable_diffusion/stable_diffusion.ipynb

In the given below image. We changed dataset size to 200 and height and width are provided to match the project specifications. And also the prompt is provided and it is shown in Figure 1 [2.1]

```
In [20]: dataset_size = 1
         width = 640
         height = 360
         prompt = "road in city"

In [5]: mkdir dataset

And we are ready to generate images:

In [23]: from tqdm.auto import tqdm
         progress_bar = tqdm(range(dataset_size))

         for i in range(dataset_size):
             image = pipe(prompt, height=height, width=width).images[0]
             image.save(f"dataset/road_image_{i}.png")
             progress_bar.update(1)

         image

0%|          | 0/1 [00:00<, ?it/s]
0%|          | 0/50 [00:00<, ?it/s]


Out[23]: 
```

Figure 1: Prompt used to generate the images

2.2 Data Collection

We've used stable diffusion to generate images for finetuning the model. click on the below badge to see how we worked with stable diffusion. The model we used is CompVis's stable-diffusion-v1-4 which can run on T4 GPU provided without any cost by google. links are provided in 2.1

2.3 Annotation

The images are annotated using LabelMe tool. Which is an opensource tool used to annotate image data. Each image is annotated twice one is for drivable area segmentation and another is for lane detection. label me is shown in Figure 2 [2.3].

Annotation for Drivable Area Segmentation is shown in Figure 3 [2.3].

Annotation for Lane Line Segmentation is shown in Figure 4 [2.3].

2.4 Partitioning of Dataset

The dataset is structured into three distinct partitions: Train, Test, and Validation. The Train split comprises 80% of the dataset, containing both the input images and their corresponding labels. Meanwhile, the Test and Validation splits each contain 10% of the data, with a similar structure, consisting of image data and label information. Within each of these splits, there are three folders:

- Images: This folder contains the original images, serving as the raw input data for the task at hand.
- Segments: Here, you can access the labels specifically designed for Drivable Area Segmentation, crucial for understanding road structure and drivable areas.
- Lane: This folder contains labels dedicated to Lane Detection, assisting in identifying and marking lanes on the road.

2.5 Processing

2.5.1 Random Perspective Transformation

This transformation simulates changes in the camera's perspective, including rotation, scaling, shearing, and translation. It is applied with random parameters:

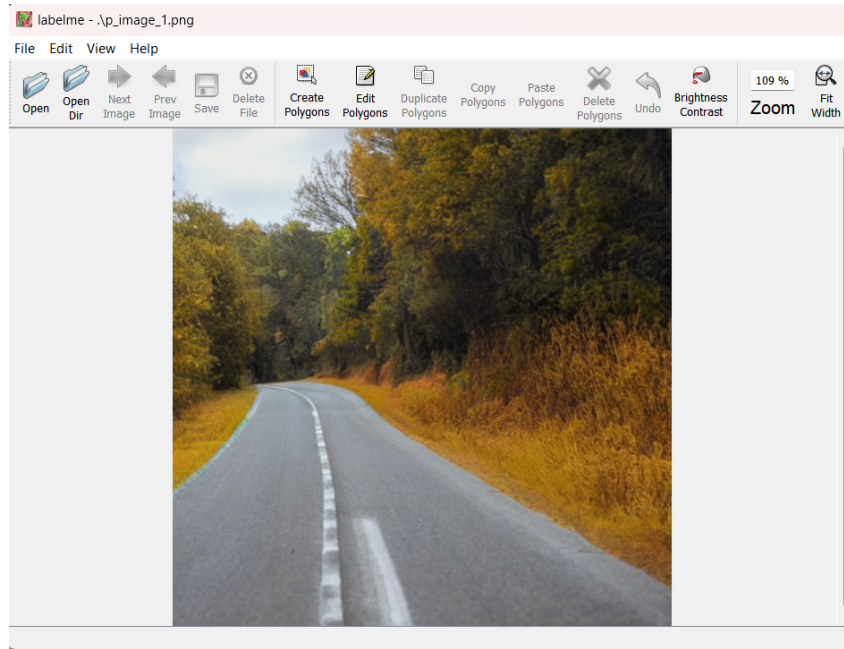


Figure 2: labelme annotation tool



Figure 3: Drivable Area Segmentation

- 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.

2.5.2 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.



Figure 4: Lane Line Segmentation

2.5.3 Image Resizing

If the Images are not in the specified size, the images are resized to a fixed size (640x360) using `cv2.resize`.

2.5.4 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.

3 Transfer Learning

3.1 Model Architecture

The TwinLiteNet architecture consists of one encoder and two decoders for two separate tasks: drivable area segmentation and lane detection. The model uses ESPNet-C as an information encoding block and incorporates Dual Attention Modules to capture global dependencies in both spatial and channel dimensions. The decoder blocks are designed to be simple, relying on ConvTranspose layers followed by batch normalization and the pRelu activation function. The input size of the model is 640x360, and the output size is $32 \times H/8 \times W/8$ for each task. Architecture is shown in Figure 5 3.1

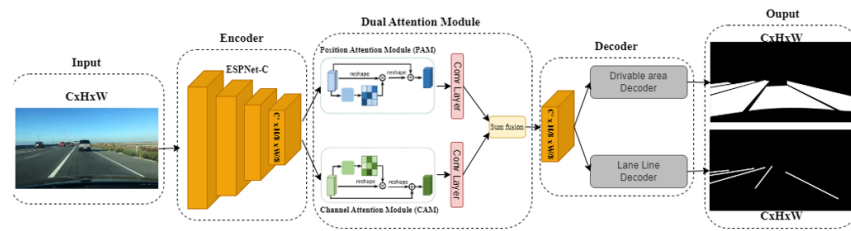


Figure 5: Architecture

3.1.1 Input and output size of each layer

- **Encoder:**
 - Convolutional layer: 3x3 kernel, 32 filters, stride 2, padding 1. Output size: $320 \times 180 \times 32$

- Depthwise separable convolutional layer: 3x3 kernel, 32 filters, stride 1, padding 1. Output size: $320 \times 180 \times 32$
- Dual Attention Module. No change in size
- Depthwise separable convolutional layer: 3x3 kernel, 64 filters, stride 2, padding 1. Output size: $160 \times 90 \times 64$
- Depthwise separable convolutional layer: 3x3 kernel, 64 filters, stride 1, padding 1. Output size: $160 \times 90 \times 64$
- Dual Attention Module. No change in size
- Depthwise separable convolutional layer: 3x3 kernel, 128 filters, stride 2, padding 1. Output size: $80 \times 45 \times 128$
- Depthwise separable convolutional layer: 3x3 kernel, 128 filters, stride 1, padding 1. Output size: $80 \times 45 \times 128$
- **Decoder 1 (Driveable Area Segmentation):**
 - ConvTranspose layer: 3x3 kernel, 128 filters, stride 2, padding 1. Output size: $160 \times 90 \times 128$
 - ConvTranspose layer: 3x3 kernel, 64 filters, stride 2, padding 1. Output size: $320 \times 180 \times 64$
 - Convolutional layer: 1x1 kernel, 1 filter, stride 1, padding 0. Output size: $320 \times 180 \times 1$
 - Sigmoid activation. No change in size
- **Decoder 2 (Lane Detection):**
 - ConvTranspose layer: 3x3 kernel, 128 filters, stride 2, padding 1. Output size: $160 \times 90 \times 128$
 - ConvTranspose layer: 3x3 kernel, 64 filters, stride 2, padding 1. Output size: $320 \times 180 \times 64$
 - Convolutional layer: 1x1 kernel, 2 filters, stride 1, padding 0. Output size: $320 \times 180 \times 2$
 - Softmax activation. No change in size

3.1.2 Objective Function

The model uses two loss functions for the proposed segmentation model: Focal Loss and Tversky Loss. Focal Loss is used to address class imbalance, while Tversky Loss is used to handle overlapping classes. Loss functions are shown in Figure 6 3.1.2

3.1.3 Configuration

Hyperparameters are shown in Table [1]

Table 1: Training Hyperparameters

Hyperparameter	value
epochs	8
learning rate	5e-4
β_1	0.9
β_2	0.999
weight decay	5e-4
ϵ	1e-8

3.2 Experiments and Results

3.2.1 Experiments

The base model is trained on the large amounts of data. We have taken those hyperparameters and experimented with them. First, we had to choose number of epochs. For this we have taken 8 epochs because taking more than 10 epochs is resulting in overfitting and it is not performing well on validation set. A learning rate of 5e-4 is chosen, because if the learning rate is small, it is resulting in smaller updates and if the learning rate is big, it is resulting in larger updates. And every other hyperparameter were taken from base model because we didn't want to change the model too much.

Focal Loss:

$$Loss_{focal} = -\frac{1}{N} \sum_{c=0}^{C-1} \sum_{i=1}^N p_i(c)(1 - \hat{p}_i(c))^\gamma \log(\hat{p}_i(c))$$

Tversky Loss:

$$Loss_{tversky} = \sum_{c=0}^C \left(1 - \frac{TP(c)}{TP(c) - \alpha FN(c) - \beta FP(c)}\right)$$

Total Loss:

$$Loss_{total} = Loss_{focal} + Loss_{tversky}$$

Figure 6: Objective Function

3.2.2 Results

Results are shown in Table [2]. Evaluation on test set is shown in Table [3]

Table 2: Training and Validation Metrics

Epoch	Learning Rate	Train Loss	Train Loss Last Batch	Val Loss	Val Loss Last Batch
1/8	0.0005	0.3725	0.2774	0.3986	0.3868
2/8	0.00038281	0.2779	0.2761	0.2781	0.2722
3/8	0.00028125	0.2262	0.1971	0.2520	0.3024
4/8	0.00019531	0.1927	0.1517	0.2323	0.2640
5/8	0.000125	0.1603	0.1538	0.1957	0.2463
6/8	7.031e-05	0.1618	0.1300	0.1957	0.1631
7/8	3.125e-05	0.1524	0.1784	0.1869	0.1533
8/8	7.81e-06	0.1541	0.1379	0.1884	0.2271

Table 3: Pixel Accuracy and Intersection over Union on Test Set

Segment	Accuracy	Intersection over Union (IOU)	Mean IOU (mIOU)
Driving Area	0.965	0.765	0.863
Lane Line	0.984	0.193	0.588

3.3 Deployment

We have built an application and deployed the model in gradio. It is show in Figure 7 3.3

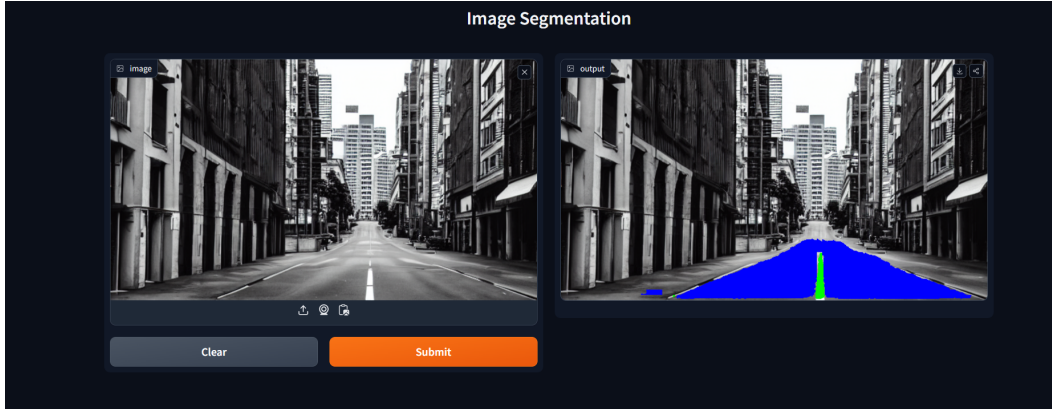


Figure 7: Deployment using Gradio

4 Mini-Network

4.1 MiniNet Architecture

The MiniNet architecture consists of several layers for image processing, including input convolution, downsampling, intermediate convolutions, upsampling, and classifiers. The model takes RGB images as input and produces two segmentation maps as output.

4.1.1 Layers, Input and output of each layers

The input to the model is a 3-channel RGB image with dimensions $3 \times H \times W$, where H is the height and W is the width. The output consists of two segmentation maps, each with dimensions $2 \times H \times W$.

- **Input Convolution Layer:**
Parameters: Conv2d(3, 16, kernel_size=3, stride=2, padding=1)
Activation: ReLU
Input Size: $3 \times 360 \times 640$
Output Size: $16 \times 180 \times 320$
- **Downsampling Layer:**
Parameters: MaxPool2d(kernel_size=3, stride=2, padding=1)
Input Size: $16 \times 180 \times 320$
Output Size: $16 \times 90 \times 160$
- **Intermediate Convolution Layers:**
Parameters:
 - Conv2d(16, 32, kernel_size=3, stride=1, padding=1)
Activation: ReLU
Input Size: $16 \times 90 \times 160$
Output Size: $32 \times 90 \times 160$
 - Conv2d(32, 64, kernel_size=3, stride=1, padding=1)
Activation: ReLU
Input Size: $32 \times 90 \times 160$
Output Size: $64 \times 90 \times 160$
- **Upsampling Layers:**
Parameters:
 - ConvTranspose2d(64, 32, kernel_size=4, stride=2, padding=1)
Activation: ReLU
Input Size: $64 \times 90 \times 160$
Output Size: $32 \times 180 \times 320$

- ConvTranspose2d(32, 16, kernel_size=4, stride=2, padding=1)
Activation: ReLU
Input Size: $32 \times 180 \times 320$
Output Size: $16 \times 360 \times 640$
- **Classifiers:**
Parameters:
 - Conv2d(16, 2, kernel_size=1)
Input Size: $16 \times 360 \times 640$
Output Size: $2 \times 360 \times 640$
 - Conv2d(16, 2, kernel_size=1)
Input Size: $16 \times 360 \times 640$
Output Size: $2 \times 360 \times 640$

4.1.2 Objective Function

The objective function of the MiniNet model involves multiple loss components, including focal loss, Tversky loss, and a total loss. The model is trained using stochastic gradient descent (SGD) with backpropagation.

4.1.3 Configuration

Hyperparameters are shown in Table [4]

Table 4: Training Hyperparameters

Hyperparameter	value
epochs	1000
learning rate	5e-4

4.2 Experiments and Results

4.2.1 Training Details

The model is trained for 1000 iterations. The model is only trained on 2 samples and overfit to 100%.

4.2.2 Model Selection

The final trained model is selected based on the best validation performance. It is then tested on a separate test dataset, and the results are reported. in Table 5

4.2.3 Results

Validation is performed every 50 iterations. The model is evaluated on both training and validation datasets. Evaluation metrics include accuracy, intersection over union (IOU), and mean IOU for driving area and lane line segmentation. the IoU is only 0.176 for Drivable Area which is significantly less than the original model's IoU values

Table 5: Test Set Results

Driving Area Acc	Lane Line Acc	Driving Area IoU	Lane Line IoU
0.867	0.978	0.176	0.023

5 Conclusion

In conclusion, we have fine-tuned the model on our custom dataset which resulted in results comparable to the original paper's results and we have also built a Mini Network based on the original paper's architecture. but the results were not as satisfying. This may be due to the mini net's architecture.