# REPORT

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

**Padmaja Phadke**
**Pijush Pal**

## 1. INTRODUCTION

This project aims to optimize the performance of self-driving cars by refining a model for detecting drivable areas and identifying lanes. The main goal is to customize the model using a dataset created through stable diffusion. Understanding the critical role of accurate environmental perception in autonomous driving, this project utilizes deep learning techniques, particularly semantic segmentation, to assign semantic classes like road or vehicle to each pixel in an image. Traditional models for semantic segmentation and lane detection, such as UNet, SegNet, LaneNet, and SCNN, often require significant computational resources and are not suitable for low-powered embedded systems in self-driving vehicles. To address this challenge, the proposed architecture in the original paper draws inspiration from ESPNet, incorporating depth wise separable convolutions and a Dual Attention Network. Notably, the network includes two decoder blocks, similar to YOLOP, with the aim of achieving high processing speed and easy implementation on embedded hardware.

## 2. DATASET

**Dataset Compilation:**

- Aggregated a custom dataset by gathering images from diverse university groups.
- Prioritized scenes depicting roads for inclusion in the dataset.

**Focused Data Collection:**

- Emphasized the collection of data specifically on lane markings and drivable areas within road scenes.
- This targeted approach aimed to enhance the dataset's specificity for the intended task.

**Dataset Partitioning:**

- Divided the dataset into three distinct sets for training, validation, and testing purposes.
- Training Set: 80% of the dataset (160 samples).
- Validation Set: 10% of the dataset (20 samples).
- Testing Set: 10% of the dataset (20 samples).

**Subset Composition:**

- The dataset is structured into three subsets, each serving a specific purpose:

- Images: Includes the original images depicting road scenes.
- Drivable Area: Contains data related to drivable areas within the road scenes.
- Lane Markings: Focuses on collecting information specifically about lane markings within the road scenes.

**GitHub Link of the Project**

https://github.com/pijush2022/Lane_detection.git

**GitHub Link of the Dataset**

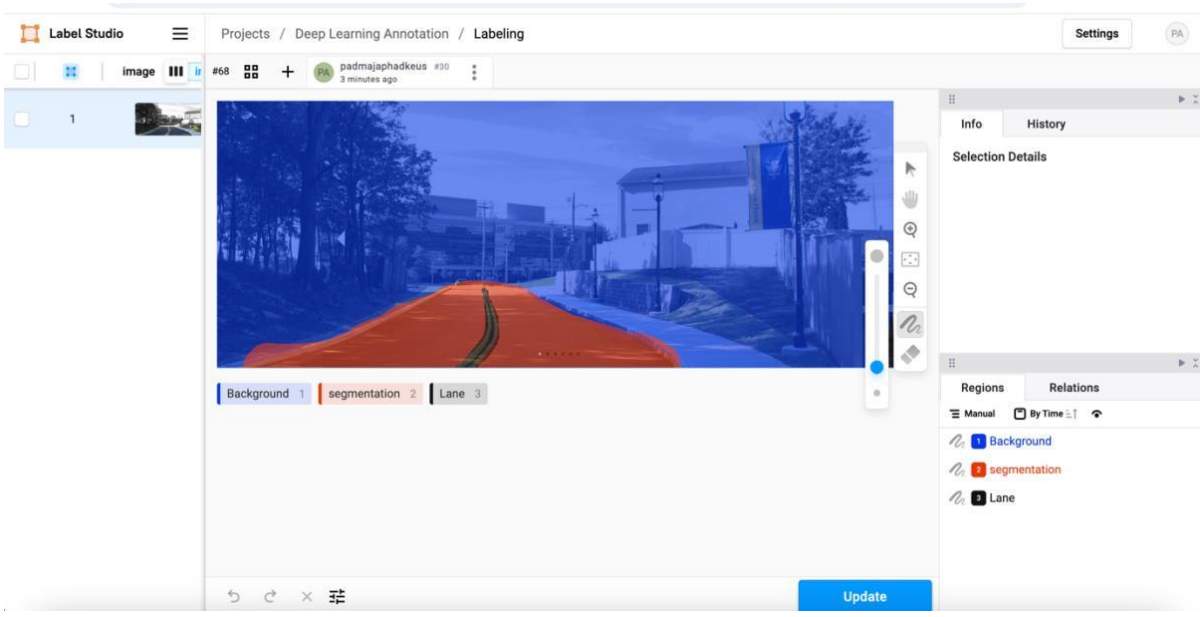
The dataset that we annotated is pushed into GitHub with Git LFS and OneDrive. (GitHub) Link to the dataset. Click on the link below

Dataset Link: -

https://drive.usercontent.google.com/download?id=1BI8xGxoWWjDZYkTLBt4qOki1vfcG29lP&export=download&authuser=0

**Annotation**

- Our project leverages Label Studio, an image annotation tool, to meticulously label road lines and drivable areas within our image dataset.
- Link: https://labelstud.io/guide




**Original Image:**


**Annotation for Drivable area segmentation:**

**Annotation for Lane detection:**

**PROCESSING**

**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.

**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.

**Image Resizing**

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

**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

**Model Architecture**

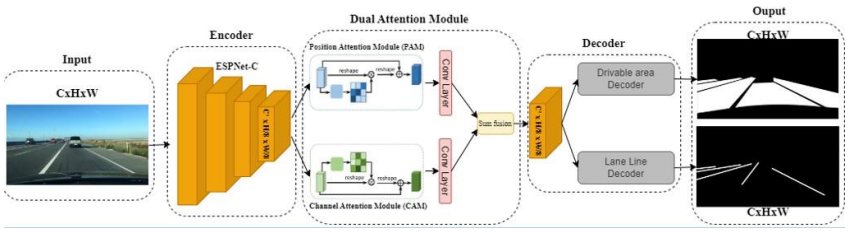
**Architecture Overview:** TwinLiteNet consists of one encoder and two decoders. It's designed for driveable area segmentation and lane detection tasks.

**Encoder**: Utilizes ESPNet-C as an information encoding block.

**Dual Attention Modules:** Incorporated to capture global dependencies in both spatial and channel dimensions.

**Decoder Blocks:** Designed to be simple. Relies on ConvTranspose layers followed by batch normalization and the pRelu activation function.

**Input and Output Sizes:** Input size: 640x360.
**Output size:** 32xH/8xW/8 for each task.




Architecture

**Input and output size of each layer:**

**Encoder:**

Convolutional layer:3x3 kernel, 32 filters, stride 2, padding 1. Output size: 320 × 180 × 32.

Depth wise separable convolutional layer: 3x3 kernel, 32 filters, stride 1, padding 1.
Output size: 320 × 180 × 32.
Dual Attention Module. No change in size
Depth wise separable convolutional layer: 3x3 kernel, 64 filters, stride 2, padding 1.
Output size: 160 × 90 × 64.
Depth wise separable convolutional layer: 3x3 kernel, 64 filters, stride 1, padding 1.
Output size: 160 × 90 × 64.
Dual Attention Module. No change in size
Depth wise separable convolutional layer: 3x3 kernel, 128 filters, stride 2, padding 1.
Output size: 80 × 45 × 128.

Depth wise separable convolutional layer: 3x3 kernel, 128 filters, stride 1, padding 1.
Output size: 80 × 45 × 128.

**Decoder 1 (Drivable Area Segmentation):**

ConvTranspose layer: 3x3 kernel, 128 filters, stride 2, padding 1.

Output size: 160 ×90 × 128.
ConvTranspose layer: 3x3 kernel, 64 filters, stride 2, padding 1.
Output size: 320 × 180 × 64.
Convolutional layer: 1x1 kernel, 1 filter, stride 1, padding 0.
Output size: 320 ×180 ×1.
Sigmoid activation. No change in size

**Decoder 2 (Lane Detection):**

ConvTranspose layer: 3x3 kernel, 128 filters, stride 2, padding 1.

Output size: 160 × 90 × 128.
ConvTranspose layer: 3x3 kernel, 64 filters, stride 2, padding 1.
Output size: 320 × 180 × 64.
Convolutional layer: 1x1 kernel, 2 filters, stride 1, padding 0.
Output size: 320×180×2.
Softmax activation. No change in size

**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.

Configuration

Hyperparameters are shown in Table

| Hyperparameter | value |
|---|---|
| epochs | 8 |
| learing rate | 5e-4 |
| $\beta_1$ | 0.9 |
| $\beta_2$ | 0.999 |
| weight decay | 5e-4 |
| $\epsilon$ | 1e-8 |

## EXPERIMENTS

**Training Data and Base Model:**

- The base model was trained on extensive datasets.
- We utilized the hyperparameters from this base model for our experiments.

**Epoch Selection:**

- Chose 30 epochs for training.
- Overfitting was observed beyond 30 epochs, leading to decreased performance on the validation set.

**Learning Rate Choice:**

- Selected a learning rate of 5e-4.
- Aimed to strike a balance; smaller rates result in minor updates, while larger rates lead to significant updates.

**Consistency with Base Model:**

- Maintained other hyperparameters consistent with those of the base model.
- Goal was to preserve the model's fundamental characteristics while avoiding excessive alterations.

```
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()
```

```
[ ] args = dict()

    args["lr"] = lr
    args["max_epochs"] = 30
    args["onGPU"] = True
```

**Loss**

```
Training loss for last batch: 0.11452271789312363
Validation loss for last batch: 0.26914864778518677
-----------------------------------------
Epoch: 30/30
Learning rate: 1e-08

        29/29      0.06289      0.04731       0.1102: 100%|██████████| 20/20 [00:05<00:00,  3.64it/s]

Average Training Loss: 0.13972755856812
Average Validation Loss: 0.18667576710383096

Training loss for last batch: 0.11020036041736603
Validation loss for last batch: 0.25874409079551697
-----------------------------------------
```

**Objective Function**

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:

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

Total Loss:

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

# RESULT

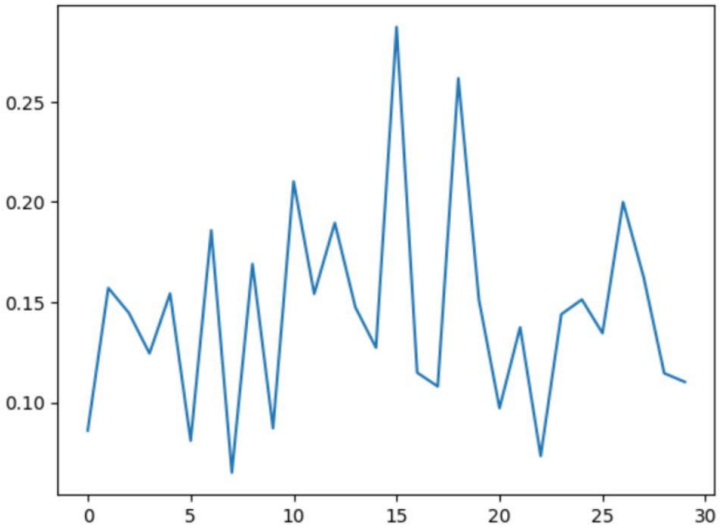
**Training Loss and Validation Loss:**

```
Training loss for last batch: 0.11020036041736603
Validation loss for last batch: 0.25874409079551697
-----------------------------------------
```

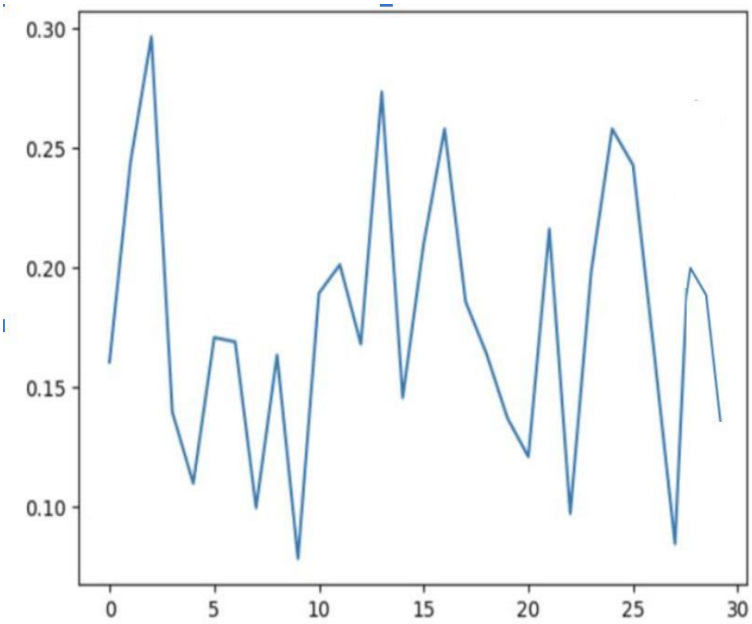
**Accuracy and Intersection over Union:**

```
Driving area Segment: Acc(0.963)    IOU (0.778)    mIOU(0.868)
Lane line Segment: Acc(0.984)    IOU (0.209)  mIOU(0.597)
```

**Training Loss:**

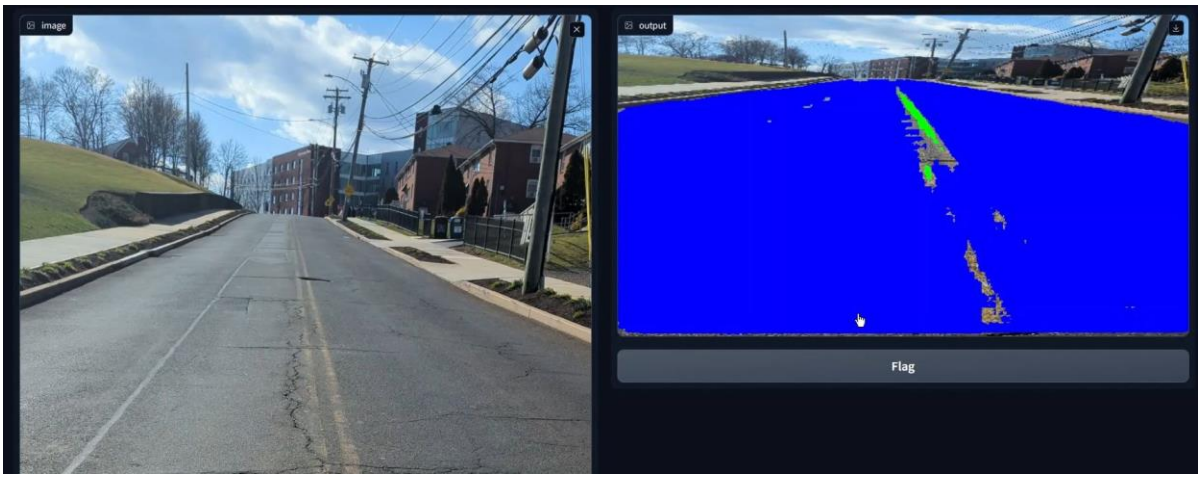



**Validation Loss:**



## DEPLOYMENT

We have built an application and deployed the model in gradio.




Deployment using Gradio

## 4. MINI-NETWORK

### 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.

### Layers, Input and output of each layer

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 × 90 × 160.

**Upsampling Layers:**
*Parameters:*

ConvTranspose2d (64, 32, kernel size=4, stride=2, padding=1)
*Activation:* ReLU
*Input Size:* 64 × 90 × 160.
*Output Size:* 32 × 180 × 320.
ConvTranspose2d (32, 16, kernel size=4, stride=2, padding=1)
*Activation:* ReLU
*Input Size:* 32 × 180 × 320.
*Output Size:* 16 × 360 × 640.

**Classifiers:**
*Parameters:*

Conv2d (16, 2, kernel_size=1)
*Input Size:* 16 × 360 × 640.
*Output Size:* 2 × 360 × 640.
Conv2d (16, 2, kernel_size=1)
*Input Size:* 16 × 360 × 640.
*Output Size:* 2 × 360 × 640.

### 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.

### Configuration

Hyperparameters are shown in Table.

| Hyperparameter | value |
|---|---|
| epochs | 1000 |
| learing rate | 5e-4 |

### Experiments and Results

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

**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.

**Results**: Validation occurs every 50 iterations, encompassing assessment on both training and validation datasets. Evaluation criteria encompass accuracy, intersection over union (IOU), and mean IOU for both driving area and lane line segmentation. Notably, the IOU for Drivable Area stands at a mere 0.176, a notable deviation from the IOU values reported in the original model.

# 5.CONCLUSION

In conclusion, through fine-tuning the model on our custom dataset, we have achieved results that are comparable to those reported in the original paper. Our efforts have yielded commendable outcomes, demonstrating the effectiveness of our approach.