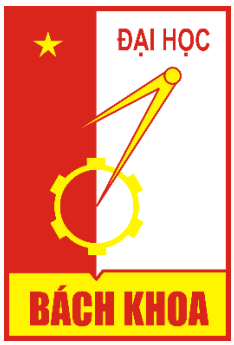
Hanoi University of Science and Technology

School of Information and Communication Technology

----- ☙ 🕮 ❧ -----



Project II

Topic: **CARLA Image Semantic Segmentation with U-Net**

Supervisor: **Ph.D. Tran Nguyen Ngoc**

Student: **Hoang Van Khanh – 20194440**

Hanoi, August 2022

CONTENT

1. Introduction

The rapid development of AI leads to many studies on autonomous robot and self-driving vehicles, in which, autonomous driving plays one of the most important roles in supporting a robot or a car to be able to observe, move, and avoid obstacles. Image Segmentation – one of the application of AI in self-driving vehicles, is the process of labelling each pixel in the image as belonging to a predefined class . In this project, I used a based network - U-Net, which was original proposed for Biomedical Image Segmentation. I modifed U-Net implementation to predict semantic segmentations of CARLA images, in which given a 600 x 800 x 3 RGB, ouput a pixel-wise segmentation of road and vehicles.

The U-Net model is implemented from scratch (using Tensorflow and Keras) and for more detail in customizing the architecture will be discussed in Section 3. The rest of the report is organized as follows : In Section 2, an overview of the chosen dataset and data augmentation, and Section 4 explains the processing pipeline. Model architectures are discussed in Section 4, and lastly proprosing the experimental result in Section 5.

The reason I carried out this project is that the dataset from CARLA is free and can be gathered much faster than in the real world and this allows me to iterate on perception pipeline quickly before fine-tuning it with real world data.

2. Dataset Overview

CARLA(**Car Learning to Act**) is an open-source simulator developed to support self-driving research and development. CARLA provides simulation-based digital resources such as buildings, vehicles, roads, etc. and can be used for free.

Here is a sample of input images (left) and the segmentation (right) produced by the simulator.

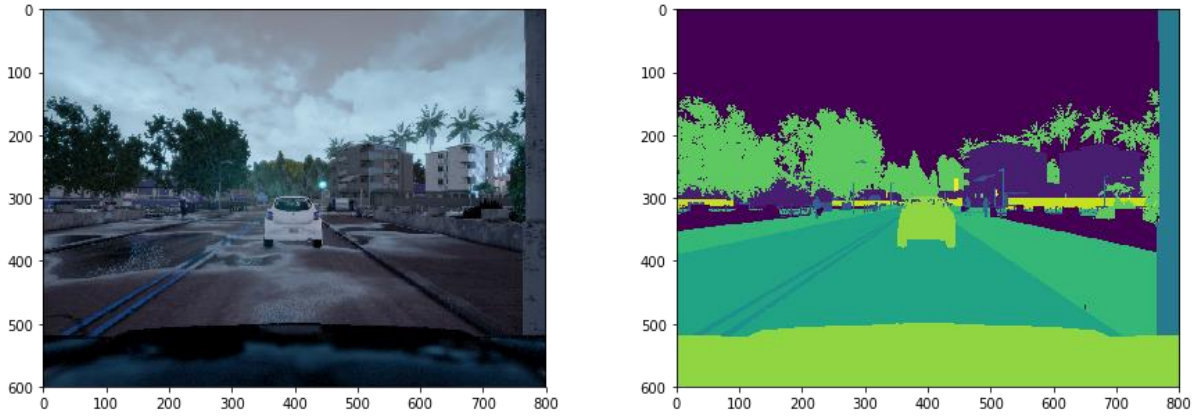


Fig 1: A sample of input and output produced by CARLA

Each pixel value in the segmentation image represents a class of object (pedestrian, car, lane, road, etc.). The dataset has 12 different classes : road, lane-marking, traffic sign, sidewalk, fence, pole, wall, building, vegetation, vehicle, pedestrian and other. However, in this project, I focus only the lane road as well as the vehicle class for my model.

The data which was generated as part of the 2018 Lyft Udacity Perceptop Challenge consists of **5000** images and **5000** their semantic segmentations. After taking advice from my teacher, I gathered **1000** more samples, which contributes the dataset to **6000** images and their labels.

Here are an sample of image and its semantic segmentations that U-Net predicted

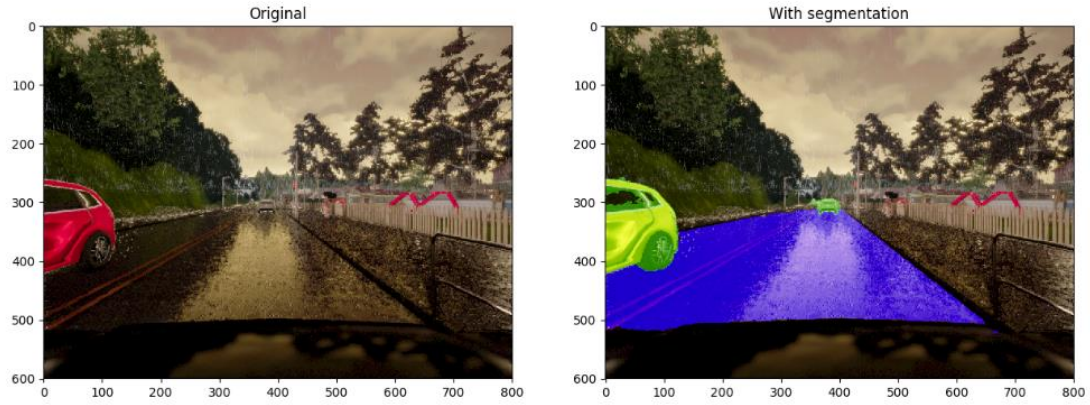


Fig 2: An sample of input and prediction from U-Net

3. Data Preprocessing and Augmentation

As I say before, in this project, I only focus on classifying cars and roads. I am also consider the lane markings as part of the road, not its own class. Lastly, the hood

of the car, which occupies the bottom 280 pixels of the image, will not need to be classified.

Here is a visualization of the dataset :

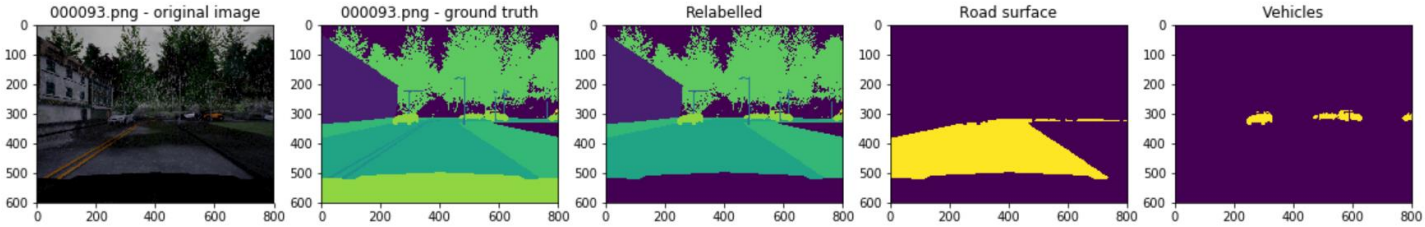


Fig 3: A sample after re-classing pixels

Augmentation

To make the U-Net model robust and capable enough to handle unknown simulator data during testing, I chose to randomly augment the input data with brightness, translations, rotations and zoom perturbations as it is being fed into the network. These augmenters are popular augmentations in a Computer Vision project.

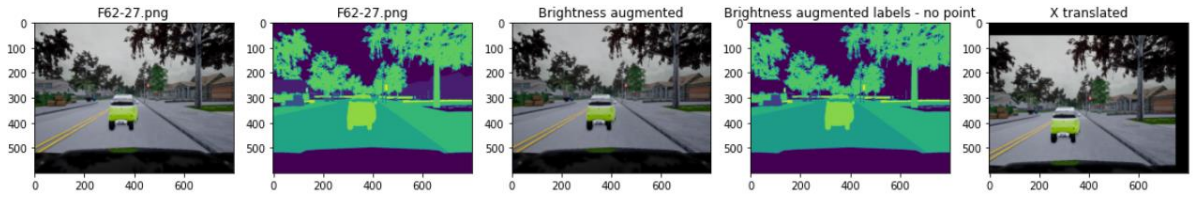


Fig 4: Sample of data augmentations (1)

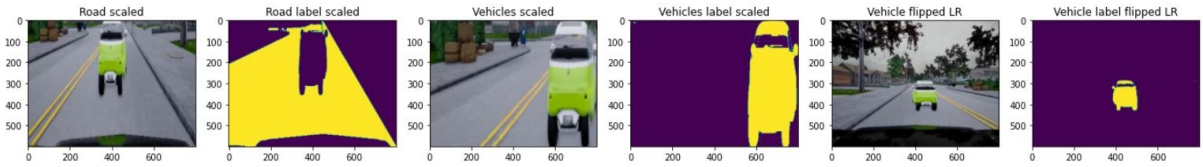


Fig 5: Sample of data augmentations (2)

Since the top of the image is the sky and the bottom is the hood of the car, I decided to crop out the top and bottom to reduce the size of the input to the network.

Here is a visualization of output from data_generator:

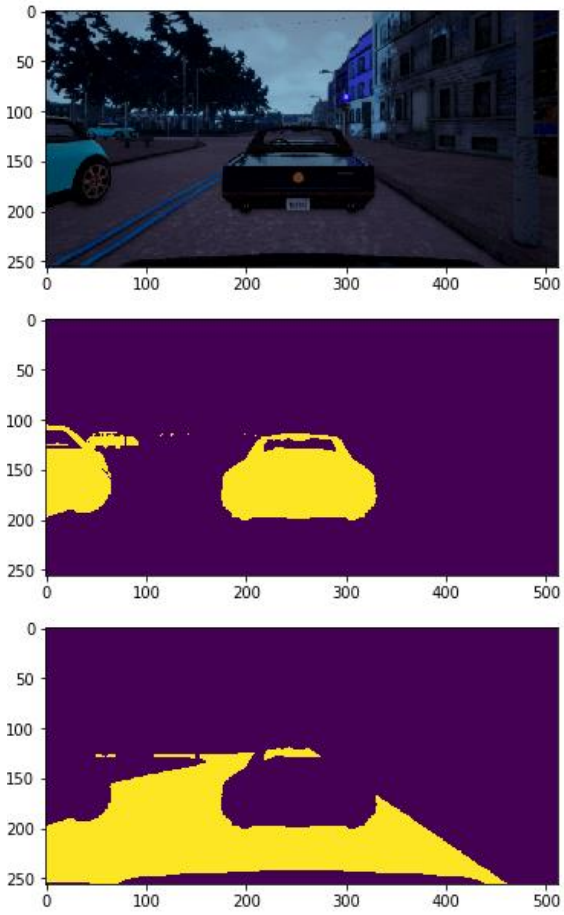


Fig 6: Sample from data_generator

4. U-Net Architectures and Approach

In general, semantic segmentation networks follow an encoder-decoder patter. The encoder portion takes the low-dimensional image into high-dimension features. The encoder can use existing networks such as ResNet or VGG because they are already well trained to recognize lines, shapes, etc.. The decoder portion takes the "encoder" features learned and converts it into pixel-level segmentations.

The architecture I chose is a very popular and effective encoder-decoder architecure – U-Net, which was original proposed for Biomedical Image Segmentation. In the U-Net original paper, the authors mentioned that U-Net can perform well even with few training images, that's the reason I chose this network (I only had 6000 images overall). Though U-Net was original designed for biomedical purposes, some of top entries on Kaggle used U-Net so I was hoping it would work in this project.

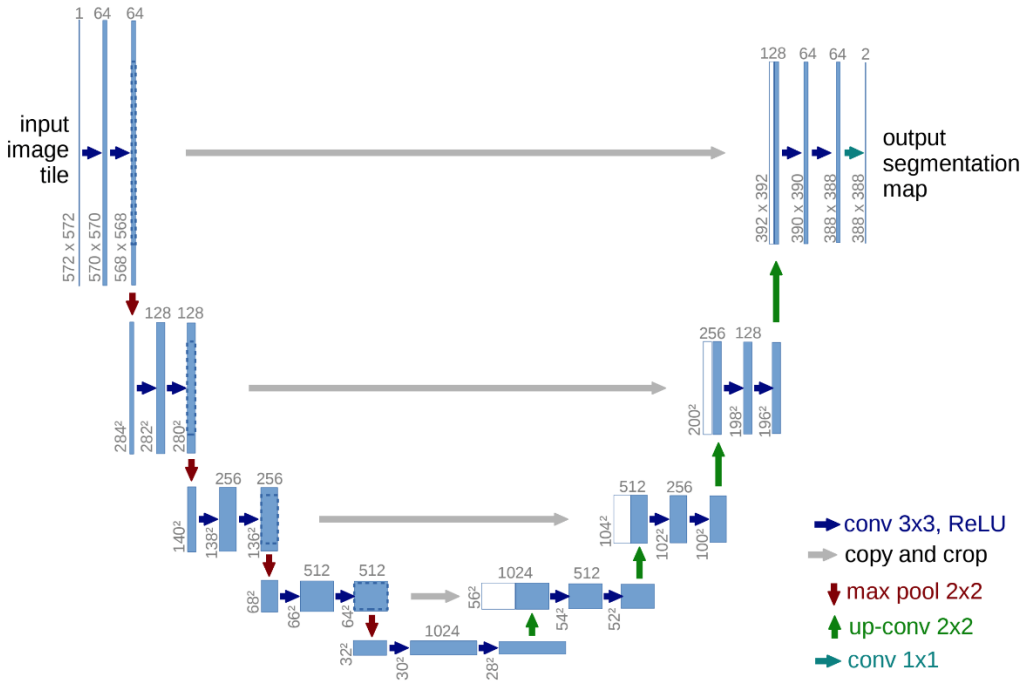


Fig 7: U-Net architecture (source: Internet)

The main ideas behind U-Net :

- Use upsampling layers in the “decoder” portion to progressively increase the image data to the desired output dimension.
- Combine the results of the “encoding” layers with the upsampling layers to help localize features.

My Approach

Since U-Net has been around for a while, there are many existing open source implementations of U-Net out there. It was for that reason and after taking my mentor - PhD. Tran Nguyen Ngoc’s advice that I re-implemented the network architecture from Peter Giannakopoulos’s Keras implementation. Instead, I focus on customizing loss function and hyperparameters. My modification to Peter’s code are :

- Remove the compile step from model generation function, this allows the inference step to load the model much faster.
- Update the weighted loss functions to allow for arbitrary image sizes.

My final model was trained using a U-Net network with 7 downsampling steps. I set the target number of epochs to 100, but I have EarlyStopping Keras callback that checks the validation loss to see if it has remained steady for at least 3 epochs. I also

added Peter's usage of reducing the learning rate by a factor of 10 on LossPlateau for 2 epochs.

Loss Function

Dice coefficient as loss function

Dice coefficient is a measure of similarity that I have used in earlier projects. Dice coefficient produces a value between 0 and 1. It is similar to IOU or Jaccard index but differs slightly in that it the denominator is simply the sum of active pixels.

d=2.|X∩Y| / (|X|+|Y|)

For each batch, intersection is calculated as:

- Intersection = sum (y_pred * y_true)
- Dice = (2 * Intersection + 1.0) / (sum(y_pred) + sum(y_true) + 1.0)

The addition of 1.0 is to prevent NaNs when denominator could otherwise be 0.

To force the network to pay more attention to cars, I developed a weighted loss function from the dice coefficient:

Loss = 3.0 - { 2 * dice(car_pred, car_truth) + dice(road_layer, road_truth) }

While training with dice coefficient, I measured other metrics: false positives, false negatives, precision and recall for each of the two classes.

It became clear that for cars, the network tended to have low precision values, which was affecting the overall F-score.

F-score as loss function

$$F_\beta = (1 + \beta^2) * \frac{\text{precision} * \text{recall}}{\beta^2 * \text{precision} + \text{recall}}$$

By setting $\beta < 1$, you weight precision more heavily than recall. And setting $\beta > 1$, you weight recall more heavily than precision.

5. Experimental result

Given the network size and input image size, I train on batches of 64 on Tesla P100 on Colab :

- Input shape : (256, 512, 3) (cropped and resized)
- Used Adam, Weighted Binary CrossEntropy + Dice loss
- Training set : 5000 images, validation set : 800, test set : 200

My final model was trained using a U-Net network with 7 downsampling steps. I set the target number of epochs to 100, but I have EarlyStopping Keras callback that checks the validation loss to see if it has remained steady for at least 3 epochs. I also added Peter's usage of reducing the learning rate by a factor of 10 on LossPlateau for 2 epochs.

F-Score from test set :

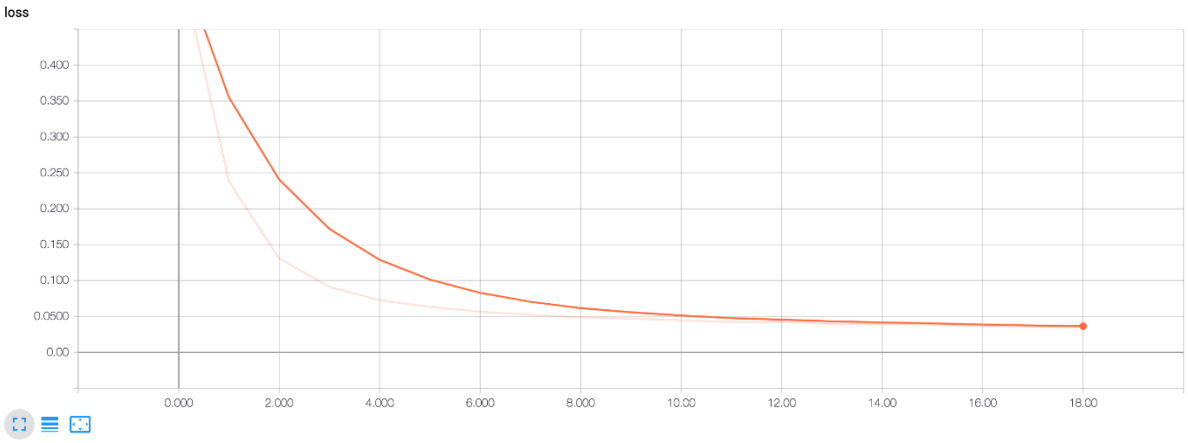
Car F-Score : 0.9587

- Precision : 0.8562
- Recall : 0.9883

Road F-Score : 0.9956

- Precision : 0.9992
- Recal : 0.9812

Tensorboard Loss Visualizations



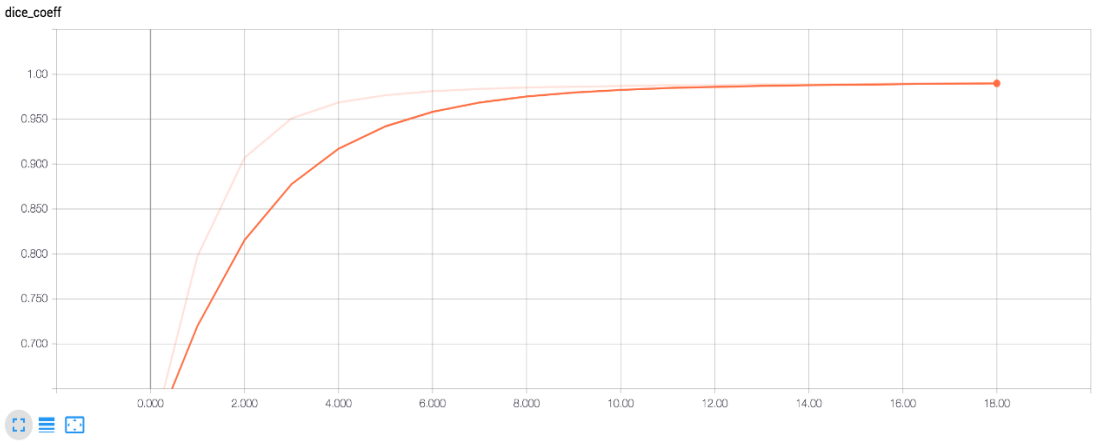


Fig 8: Loss Visualization of U-Net

Here are some segmented results from the test set

The images are laid out (from left to right): Ground Truth, Predictions, Difference

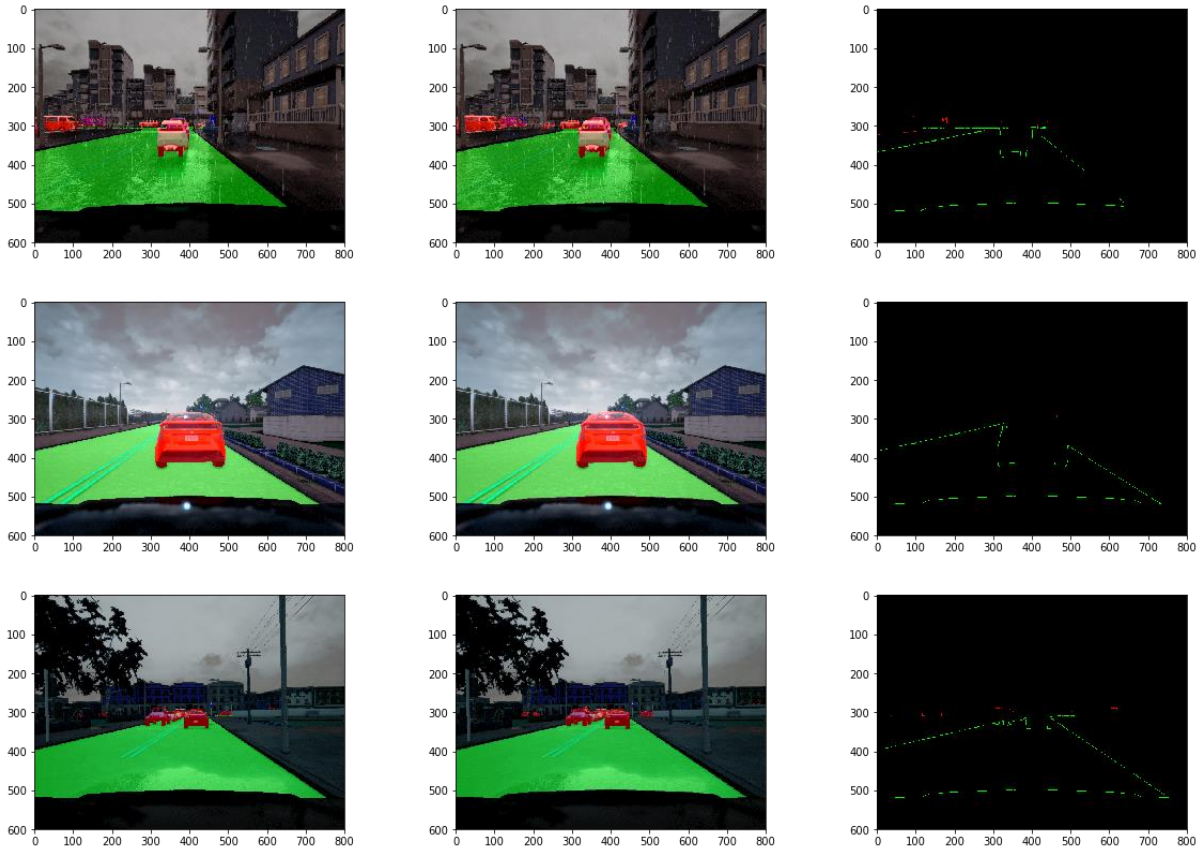


Fig 9: Some results from test set

Also, I had converted all images into a video for better visualization, the video is detached with the source code.

6. Conclusion and Further Contribution

In this project, I have experimented the U-Net model architecture on the task of CARLA image semantic segmentation. What I have achieved in this project is also an Computer Vision mini-project pipeline that I can apply for bigger one.
Beside U-Net, a few of the architecture that I would love to try them out in the future such as : IC-Net, Moblie Net, DeepLabV3,…

Transfer to the Real World

Running my trained network with real world images produced very poor results. My network does not generalize well outside of CARLA dataset.
Some of my explanations on why that is :
- The encoder portion has only ever seen simulator data. If I had used a pre-trained encoder such as ResNet, it would generalize better.
- CARLA is still a very clean world. The real world is full of debris and lighting conditions that hasn't been simulated well. I think some normalization from real world images to CARLA-style images can help with precision, but not much.

7. References

https://arxiv.org/pdf/1505.04597v1.pdf
https://viblo.asia/p/xay-dung-mang-unet-cho-bai-toan-segmentation-gDVK2QOX5Lj
https://nttuan8.com/bai-12-image-segmentation-voi-u-net/
https://github.com/petrosgk/Kaggle-Carvana-Image-Masking-Challenge
https://www.kaggle.com/c/carvana-image-masking-challenge/discussion/37208
https://www.kaggle.com/code/oluwatobiojekanmi/carla-image-semantic-segmentation-with-deeplabv3