Machine Learning Engineer Nanodegree

Semantic Segmentation for self driving cars

David Forino

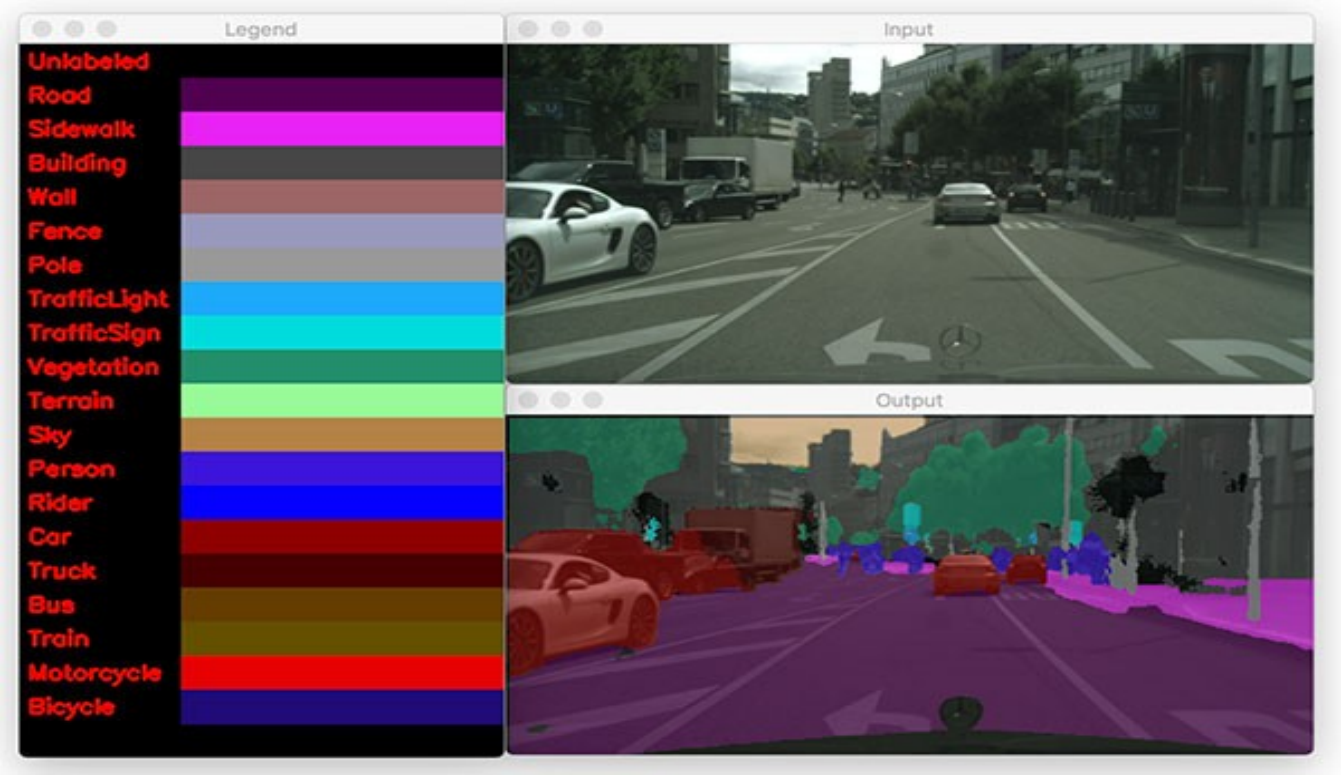
February 23th, 2019

Overview

Semantic segmentation is the task of clustering parts of images together which belong to the same object class. This type of algorithm has several use cases such as detecting road signs, cars, pedestrians, detecting tumors and much more.
Initial applications of computer vision required the identification of basic elements such as edges (lines and curves) or gradients. However understanding an image at pixel level came around only with the coining of full-pixel semantic segmentation.
Semantic segmentation is quite different and advanced compared to other image based tasks and don't have to be confused,

- Image Classification identify what is present in the image.
- Object Recognition (and Detection) identify what is present in the image and where (via a Bounding Box).
- Semantic Segmentation identify what is present in the image and where (by finding all pixels that belong it).

An example of Semantic Segmentation:

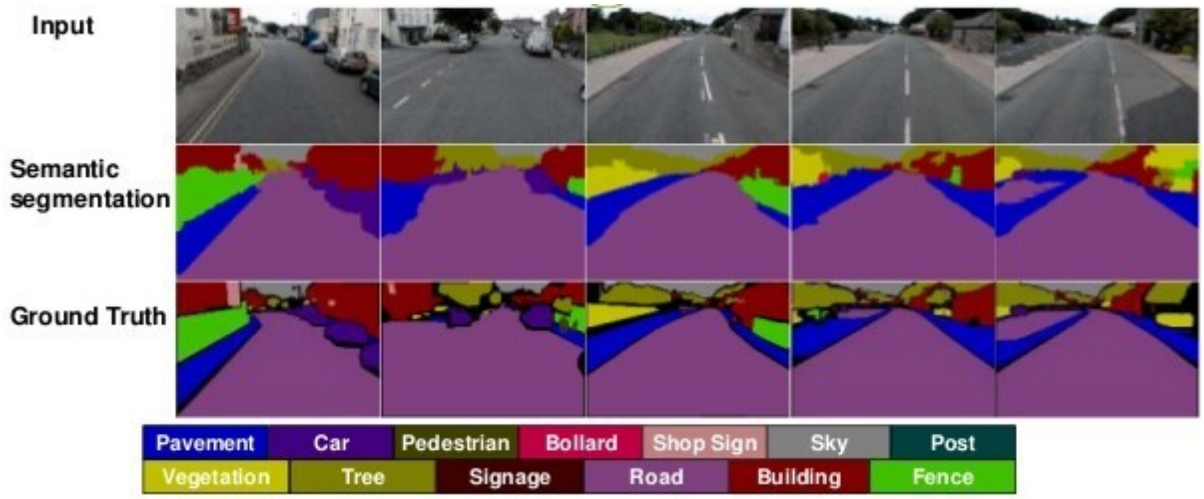


(Fig.1)

It is quite difficult to get data for semantic segmentation, luckly on Kaggle there is this dataset which suites perfectly for this purpose. This is still a field of research and in this paper you can see the progress in semantic image segmentation.

The Problem

Autonomous driving is a complex robotic task that requires perception, planning and execution within constantly evolving environments. This task also needs to be performed with utmost precision, since safety is of paramount importance. Semantic Segmentation can provide information about free space on the roads, as well as detecting lane markings and traffic signs. This needs to be done many times per second in order to keep the car constantly updated with the actual state of the road.
In this specific project I will use Semantic Segmentation to detect cars and road from a given image. In order to be able to train the model and evaluate it, is also necessary to have labels for all pixels corresponding to the related object, also called
“Ground Truth” (Figure 2).



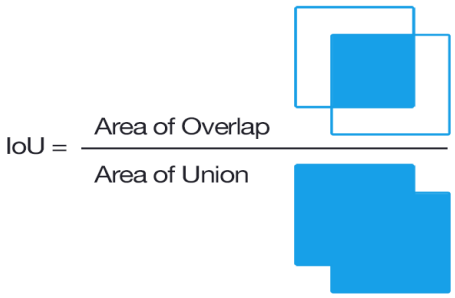
(Fig. 2)

Fast segmentation is crucial, because while having these information other actions can be taken, for instance, predict the steering angle of the car or identify whether the car needs to break and also how strong much. For this reason the model has to be able to convert the images with at least 30 frames per second speed.
Semantic Segmentation is very computational expensive, for this reason it is necessary either a very expensive hardware or a simpler model that can process images using less computing power.
For this reason I decide to choose the Enet model as a starting point which claims to be up to 18x times faster, requires 75× less FLOPs (floating point operations per second), has 79× less parameters and has similar or even better performance than other models.
I then made some modifications to it in order to increase its performance, like for example using a 4x4 pixel matrix instead of a pixel to pixel approach method, this reduce the size of the image by 4x times and results in much faster model performance compromising a bit of accuracy. It is of course not a problem to miss classify some pixels on the edge of the object if the model can make predictions more than 6x times faster.

Metrics

To measure mathematically how well the model is performing I will use a metric called “Mean IOU” which stands for “Mean Intersection Over Union”. It is defined as follow:

IOU = true_positive / (true_positive + false_positive + false_negative)

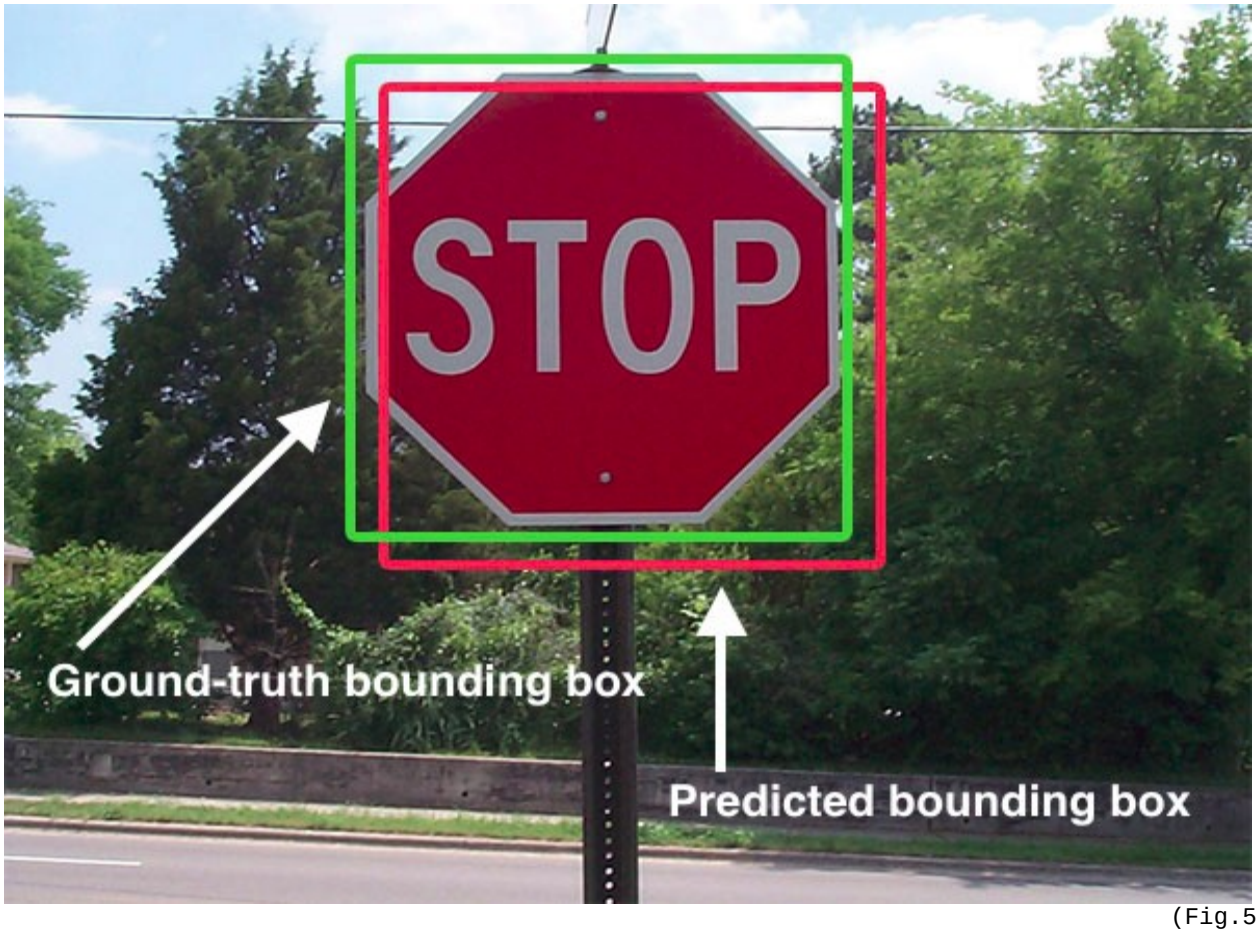


(Fig.3)

This metric is useful because it takes into count only the correct classified positive labels
instead of a general accuracy on the average classification. This is very important because
can happened that a car is very far from the camera and then takes only few pixels of the entire image. If the model predicts no cars in the image, it would score 0 accuracy using mean IOU metric instead of a higher one if we were choosing a different metric. For a better explanation, we want to be sure that the model has a high score if it correctly predicts roads and cars and their exact positions.

Example of mean IOU metric on object detection.

(Fig.4)



(Fig.5)

As we can see in the example, the model gets a high score only when its prediction and the ground truth are overlapping.

The measurement of the speed is very straight forward, I will provide several images to the model and measure the time it takes to convert them, this process will be repeated multiple times to check also the consistency.

Hardware components and more

In this project will be used a laptop with the following components:

- 16 Gb RAM
- Nvidia GTX 1050 Ti
- Intel i7 7th generation
- 250 Gb SSD

As Operative system I choose Ubuntu 18.04 but it is not mandatory.

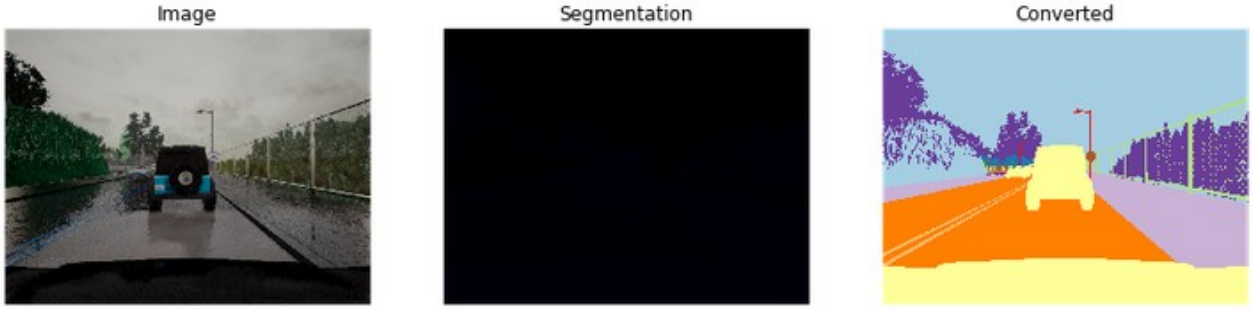
NOTE: The graphic card has to be at least the one just mentioned, older versions are not suitable for this project, 16 Gb RAM is also required.

Achieving good performance with low cost components (like the one just mentioned) is the key to build less expensive self driving cars.

Dataset

As already mentioned, in order to solve this complicated problem I used a Dataset from Kaggle. This dataset provides data images and labeled semantic segmentation captured via Carla self-driving car simulator.
The data was generated as part of the Lyft Udacity Challenge.
It has 5 sets (all 5 sets are the same) of 1000 images and corresponding labels and each set contains sets of RGB images with 600x800 size and the corresponding semantic segments, both formatted with ".png" extension. This results in a matrix of 600x800x3 for the images (3 channels because of the RGB) and 600x800x1 for the segments. To know more about how the simulator creates these images go to Sensor Camera, this simulator can provide us realistic scenarios for our model.

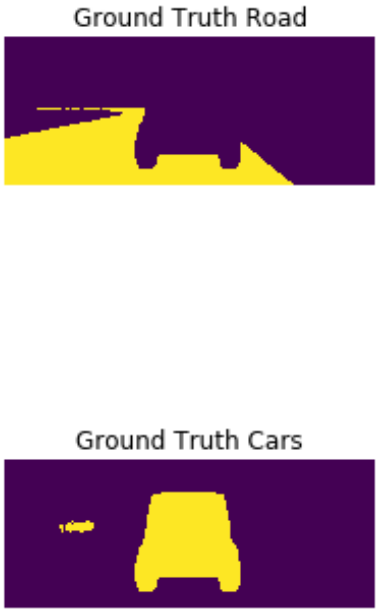
Semantic segments are pixels with low red value very difficult to distinguish, for this reason we have to convert them.



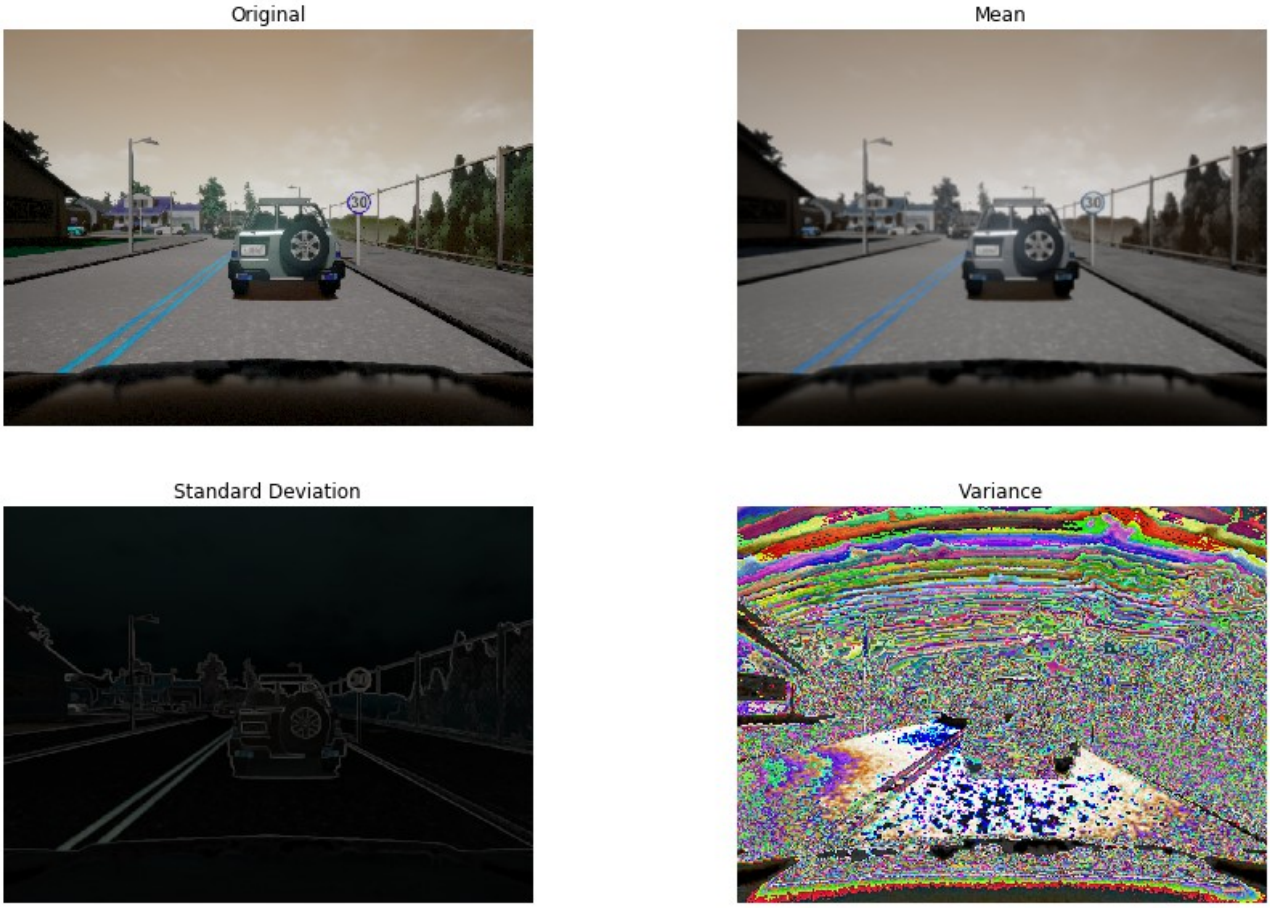
(Fig.6)

As we can see from the sample image, road and road lanes are considered 2 different objects, for simplicity we will consider lanes as part of the road. Another thing to notice is the roof of the car present in the image, which is not relevant for our task.

On the right side we can notice an example of segmentation for road and cars, like previously mentioned, this time there isn't the roof of the car, road lanes are considered part of the road and also the sky is cutted out because we won't see flying cars for now.



(Fig.7)



(Fig.8)

In this other image we can see the mean, standard deviation and variance of an image using a 4x4 kernel.

Algorithms and Techniques

In semantic segmentation the model is divided in 2 parts, the first part is used to encode the image and it does that by squeezing the image into a long sequence of matrices (i.e. from 600x800x3 pixels to 60x80x300 pixels), the second part does exactly the opposite, decode the long encoded sequence of matrices into the original image dimension (the number of channels depends on the number of classes we are trying to predict). To do so we need mainly 3 type of layers, 2D convolution, 2D Max Pooling and 2D Up Sampling respectively for encoding/decoding the image, reducing and increasing the size of the image.
For this project specifically I have used different types of layers for the Neural Network and can be summarized as follow:

- 2 Dimension Convolution
- Batch normalization
- L1 and L2 regularizers
- Dropout
- 2 Dimensions Max Pooling
- 2 Dimensions Upsampling
- RELU activation function
- PRELU activation function
- Sigmoid activation function

Convolution is an operation on two functions F and G (the filter that we can define its size), which produces a third function that can be interpreted as a modified ("filtered") version of F.
Convolution lies at the heart of any physical device or computational procedure that performs smoothing or sharpening.
2D convolution is just an extension of previous convolution by convolving both horizontal and vertical axis in 2 dimensional spatial domain.
Here you can see an example of Convolution 2D.
2D Max Pooling is an operation used to reduce the size of a matrix, it does that by choosing each maximum value of the kernel within the matrix.
2D Upsampling the opposite of Max pooling, it convert every single value of a matrix into a kernel, this is used to increase the size of a matrix.
Batch normalization is used for normalizing the value distribution before going into the next layer, this helps the model to learn faster and to use higher learning rates. Because it normalizes only one batch at the time, if the batch size is not too big (64 is the upper limit but it depends on the dataset), it adds a bit of noise into the network making it more robust and helps to generalize better.
L1 and L2 regularizer are 2 mathematical operations that help to prevent overfitting, they achieve that by penalizing high coefficients in the network. L1 penalizes using the mean absolute coefficient as a reference and L2 penalizes using the mean squared coefficient.
Dropout is also a technique used to prevent overfitting, it consists in giving a probability to each node in the layer to be deactivated for the current training process, this results in an equally training distribution on each node.
RELU (rectified linear unit) is an activation function very popular image datasets that is used to add non-linearity in the network. If the input is equal or less than 0 the output will be zero, else the output will be the same as the input. **Y = X if X > 0, else 0**
PRELU (Parametric rectified linear unit) take this idea further by making the coefficient of leakage into a parameter that is learned along with the other neural network parameters. **Y = X if X > 0, else aX**
Sigmoid is an activation function used for binary classification (like for this problem, if a pixel is part of a class or not) that has the characteristic "S"-shaped curve or sigmoid curve. **Y = e**X / (e**x + 1)**

As optimizer I have used **Adam**, an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
$m_0 \leftarrow 0$ (Initialize 1st moment vector)
$v_0 \leftarrow 0$ (Initialize 2nd moment vector)
$t \leftarrow 0$ (Initialize timestep)
while θ_t not converged **do**
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
$\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
$\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
end while
return θ_t (Resulting parameters)

Benchmark

Benchmark includes accuracy (using the previously discussed metric) and FPS (Frames per seconds) and I compared the original Enet model, my optimized version of it and another model called SEGNet.
The minimum FPS value to be achieved is 25, this value is decided due to hardware limitations, usually is at least 30 FPS in real world scenarios.

Data Preprocessing

Data preprocessing is a crucial part, even more than usual when we talk about semantic segmentation. We clearly don't want to train the model on wrong data.
The steps for data preprocessing are in order:

1. Detect if there are cars present in the picture
   (This is crucial otherwise we can't train the model to detect cars)
2. Crop the upper and lower part of the image and the masks
   (As previously mentioned, we want to remove the sky and the roof of the car)
3. Extract the masks related to cars and road including road lanes
4. Increase training examples with data augmentation

To detect if an image was suitable for training I checked in the bottom left side of the images (from pixel 200 to 480 from up going down and from pixel 0 to 600 from left going right) if at least 5% of this region contained pixels of cars.
After cropping the upper (from pixel 0 to pixel 160) and bottom (from pixel 480 to 600) part of the image, I got an image of shape 320x800x3, starting from an image of 600x800x3.

Data augmentation helped to improve the model by increasing it accuracy of over 15%. The following changes helped to maximize models' performance:

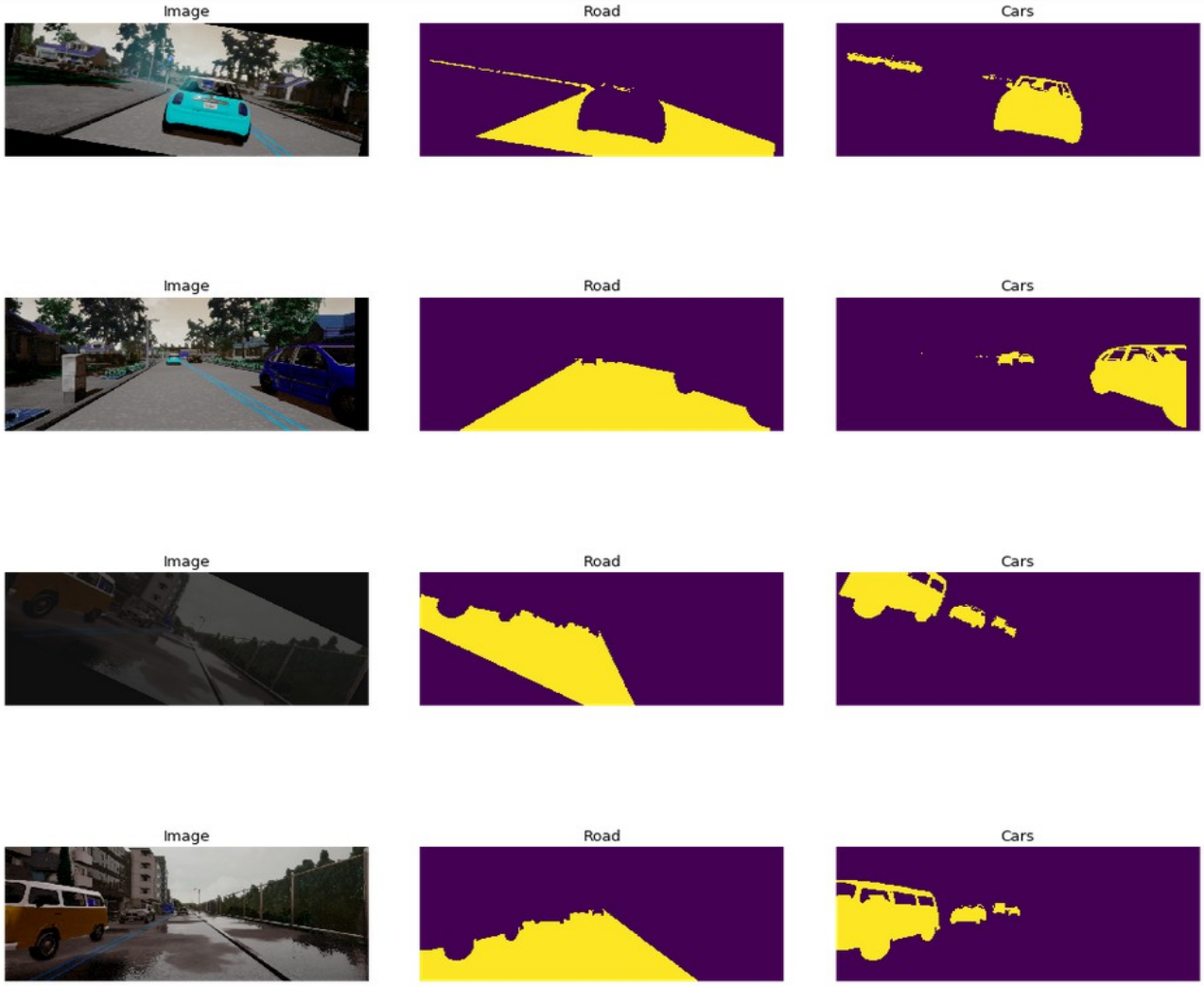
- Change contrast and brightness
  (helps to train the model in different light conditions)
- Shift images and masks
  (the shift has to be the same and also in the same direction)
- Rotate images and masks
  (the rotation has to be the same and also in the same direction)
- Flip images and masks

Final optimal parameters for data augmentation:

- **Contrast:** multiply each pixel by a range between 0.3 and 1.0
- **Brightness:** increase the value of each pixel by a range between 1 and 75
- **Shift:** shift on the X axis (horizontal) by a range between -50 and 50 pixels
- **Rotation:** range between -30 and 30 degrees

All these parameters and the flip could occur with 50% probability and they are all independent from each other.

In the following image we can see some examples



(Fig.9)

Implementation

To solve this complicated task I used a model called Enet, this model is very efficient for semantic segmentation because requires few parameters to tune, but I optimized it even more to achieve very fast performance.
Enet reduces the size of the image in the first 2 bottlenecks and then uses asymmetric and dilated layers to encode the image, afterward it increases the size of the image and decodes it.
The model apply a prediction for every single pixel in the image, which is the reason why it is very computational expensive. The modifications I implemented in this model are only 2, instead of labeling each pixel of the image I use a **matrix of 4x4 pixels.**

Name	Type	Output size
initial		$16 \times 256 \times 256$
bottleneck1.0	downsampling	$64 \times 128 \times 128$
$4\times$ bottleneck1.x		$64 \times 128 \times 128$
bottleneck2.0	downsampling	$128 \times 64 \times 64$
bottleneck2.1		$128 \times 64 \times 64$
bottleneck2.2	dilated 2	$128 \times 64 \times 64$
bottleneck2.3	asymmetric 5	$128 \times 64 \times 64$
bottleneck2.4	dilated 4	$128 \times 64 \times 64$
bottleneck2.5		$128 \times 64 \times 64$
bottleneck2.6	dilated 8	$128 \times 64 \times 64$
bottleneck2.7	asymmetric 5	$128 \times 64 \times 64$
bottleneck2.8	dilated 16	$128 \times 64 \times 64$
Repeat section 2, without bottleneck2.0		
bottleneck4.0	upsampling	$64 \times 128 \times 128$
bottleneck4.1		$64 \times 128 \times 128$
bottleneck4.2		$64 \times 128 \times 128$
bottleneck5.0	upsampling	$16 \times 256 \times 256$
bottleneck5.1		$16 \times 256 \times 256$
fullconv		$C \times 512 \times 512$

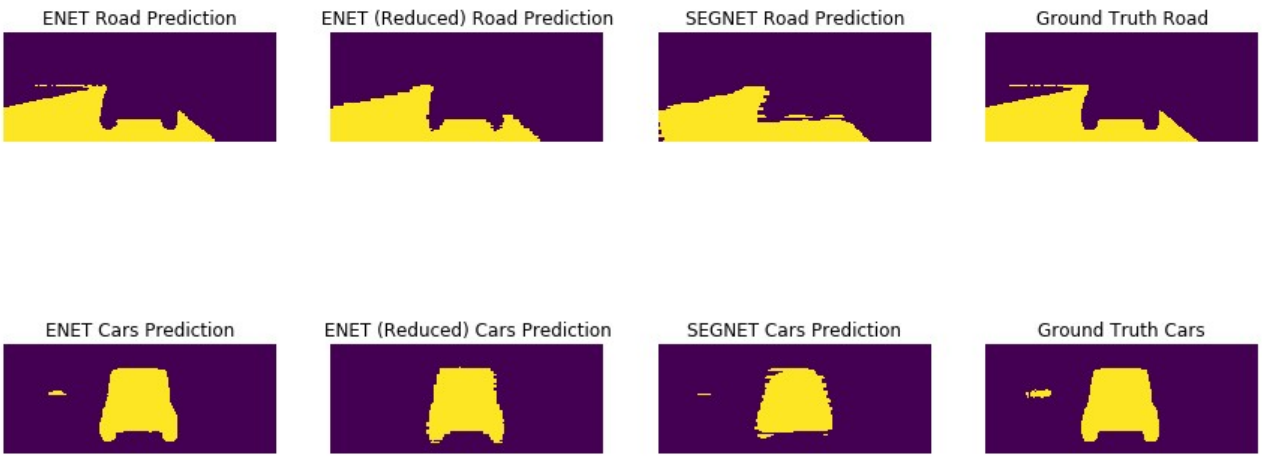
This is done by rescaling the image to 25% of its dimension using a 4x4 kernel which maintains the proportions in the image and right at the end of the model before applying the activation function, the image get resized to its original dimension using the same 4x4 kernel. At this point Instead of using 64 filters to encode and decode the image (bottleneck 1 and 4) I reduced them to 32 and the filters in the bottleneck 2 and 3 are reduced from 128 to 64.
For both ENet models L1 and L2 regularization where not necessary, both models were fitting the date very well.
SEGNet model is made out of 20 layers blocks (1 block is composed by 2D convolution layer with 3x3 kernel, batch normalization and RELU activation function), the first 10 used for encoding have 16, 16, 32, 32, 64, 64, 128, 128, 256, 256 filters respectively with a Max pooling layer of 2x2 kernel and a Dropout layer of 20% probability every second block.
The second 10 blocks used for decoding had the opposite structure, 256, 256, 128, 128, 64, 64, 32, 32, 16, 16 filters respectively, with Upsampling layer of 2x2 kernel and Dropout layer with 20% probability every second block. All models used Adam optimizer with 0.001 learning rate with was decaying during training time.

Refinement

The most time consuming part was to adjust the parameters in SEGNet model, it was badly overfitting the data and still after a lot of fine tuning I couldn’t achieve sufficient results.
This is because of hardware limitations, with this image size and my graphic card I couldn’t increase the number of filters.
After trying L2 and L1 regularizer the model was still badly overfitting, the best I could achieve was an accuracy of 73%, I could achieve the best performance by using a dropout layer with 50% probability right after every Max pooling and Upsampling layer.
The original ENet model performed very good from the beginning and there was no need for optimization on the model itself.
My ENet version was also overfitting at the very beginning, because of the few filters I was using on bottleneck 1,2,3 and 4, I restored them to the original value and the performance in term of accuracy were quite similar to the original model, was reaching 95,1% in accuracy but the speed dropped to 140 FPS (which is great but we have to remember that we are using very small images) so I decided to try a model with fewer filters but still more than the first one, I ended up by using 48 filters on the bottleneck 1 and 4 and 96 filters on the bottleneck 2 and 3, which gave me a very good compromise of both.

Evaluation

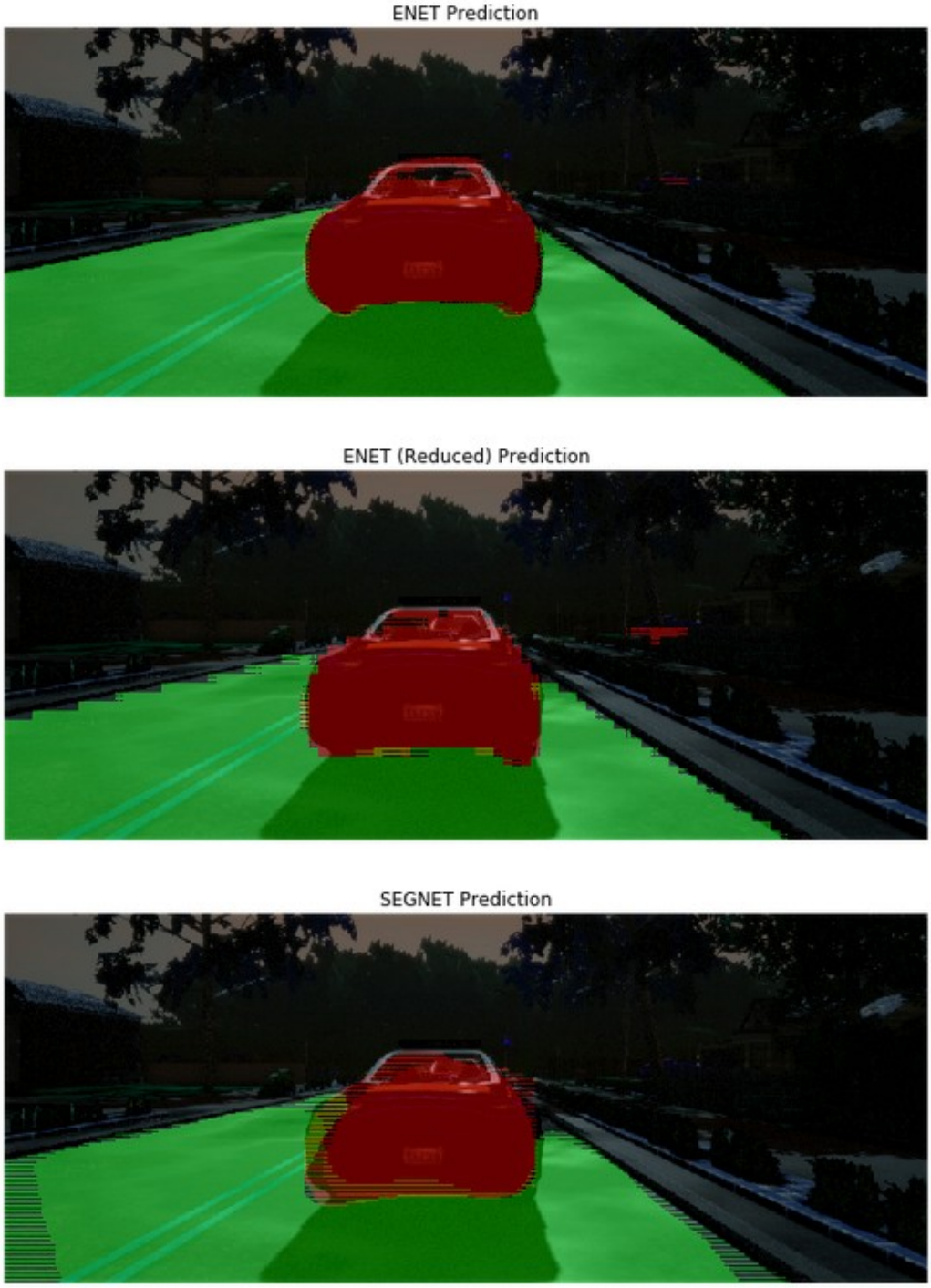
In the following images we can see the differences between the 3 models:



(Fig.11)

From a visual prospective we can see the standard Enet model performed very good, the best of the 3. The modified version of Enet performed still good even if the edges are not smooth and it couldn’t detect the cars at the end of the road. SEGNet on the other hand, could detect car and road but very poorly, the prediction is still very noisy.

Here is another visual comparison that help for a better understanding.



(Fig.12)

In this image we can confirm what previously mentioned, but I want to point out the fact that both the ENet models could detect the car on the right side which is very difficult to distinguish for a human eye with these light conditions.
Another point to consider is the image size 320x800, very small, and the fact that 1 pixel correspond to 16 pixels (4x4) to the modified Enet, this is why doesn’t have smooth edges.
For a better understanding, it is like the model was trained on a 80x200 pixel image, it surely decreases the precision.

Now the mathematical evaluation of the 3 models:

ACCURACY
ENET: 96.18%
ENET (Reduced): 93.99%
SEGNET: 85.04%

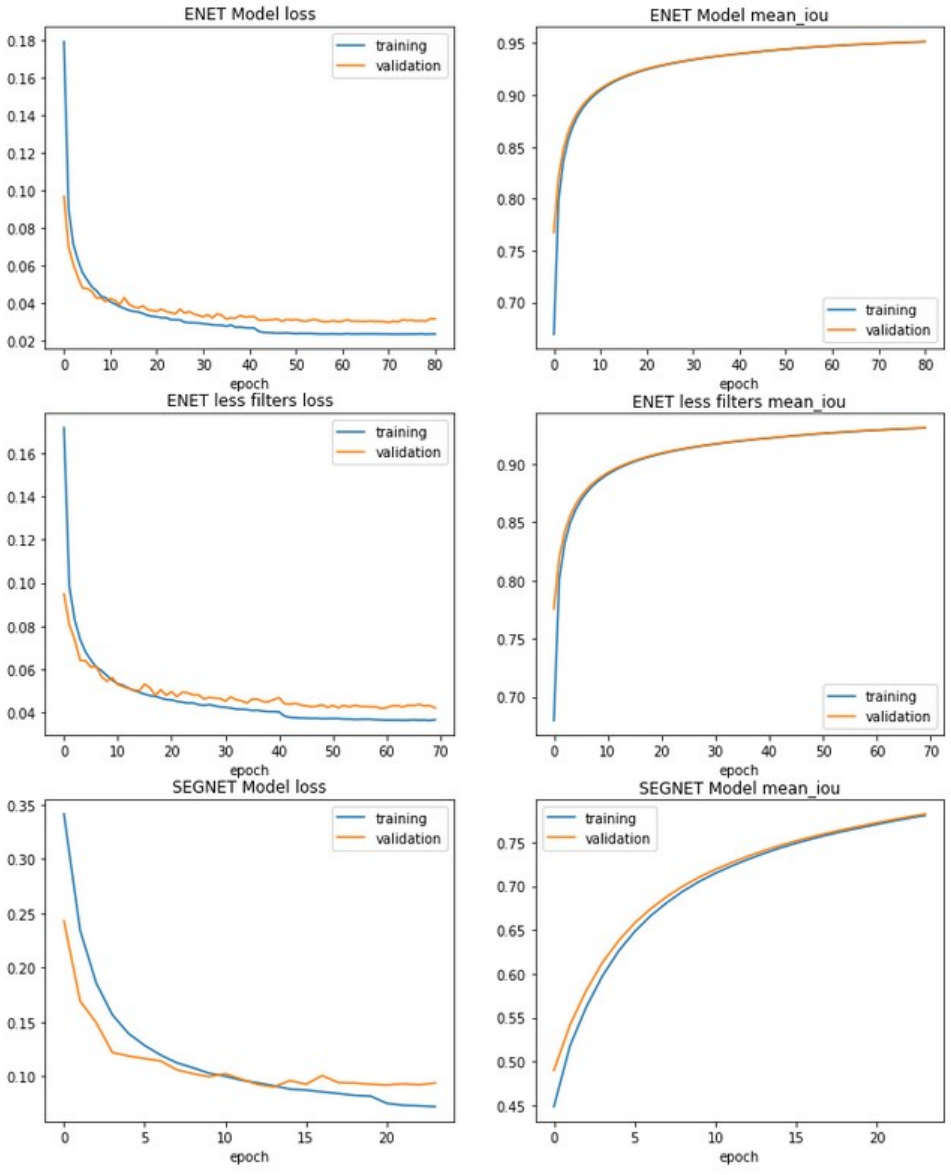
SPEED (per image)
ENET: 31.47ms
ENET (Reduced): 5.29ms
SEGNET: 31.29ms

SPEED (FPS)
ENET: 31
ENET (Reduced): 189
SEGNET: 31

As you can see the optimized version of ENet is very efficient and fast even with such an old graphic card (prize $120).

Statistics

From this graph we can see loss and mean IOU accuracy. Both ENet models could learn and increase most of the accuracy in 40 epochs. On the other hand SEGNet had a smaller training because wasn't able to generalize well after few epochs (this of course my very if we can increase the number of filters but a much more powerful GPU is needed).



Conclusion

For this specific dataset the Normal ENet result to perform better than all others (31 FPS is good), but for higher resolution images a more powerful GPU is a must.
For real world scenarios I would consider try out my version of ENet because can surely decrease the costs of self driving cars by using less expensive GPUs.

Reflection

To accomplish this project was involved a lot of research, starting from a deep understanding of semantic segmentation itself, which model to use and what metric and loss function would work well for this problem. I first started to research which models were available and well documented and afterward I was comparing them in term of efficiency.
ENet was surely the most difficult to implement compared to other models like SEGNet, which is very straight forward, but most importantly there are no examples on the internet for ENet model using Tensorflow framework. Another aspect that needed a lot of debugging was the mean intersection over union metric, unfortunately the documentation provided in Tensorflow for this metric is insufficient.
A very interesting aspect of this project is the fact that even if semantic segmentation is a well known problem, ENet is not yet used (at least in the open source community) and more specifically, my results seems to be even better than the results of the winner of the Lyft challenge, but it is difficult to compare them in term of accuracy because the datasets are different.

Improvement

This is still an area of research and new models come out quite often, but by now ENet seems the most performant one. One possible improvement is to change the number of filters in my version of ENet and test it on high resolution images to see the differences between the 2 models. It will be also good to try out new layers combination on the optimized version of ENet because its training time is still not long and this give more time to see how it behaves.