Projet 9 report

Open Class Room
Project 9
Computer Vision
Cityscape

2022

Build a computer vision system for an autonomous vehicle

Axel Favreul

Under the direction of
Richard Ball
Open ClassRoom Mentor

Table of Contents

Introduction ... *3*

1. Image segmentation ... *4*

1.1. Overview ... 4

1.2. Metrics ... 4

1.2.1. Pixel accuracy ... 4

1.2.2. Dice coefficient ... 4

1.3. Architecture ... 4

1.3.1. Target ... 4

1.3.2. Down and up-sampling ... 5

1.3.3. U-net ... 6

1.3.4. Deeplab ... 7

2. Training ... *9*

2.1. Pipeline structure ... 9

2.2. Data augmentation ... 9

2.3. U-net and Deeplab baseline ... 9

2.3.1. U-net ... 9

2.3.2. Deeplab baseline ... 11

2.4. Hyper parameters tuning ... 11

2.4.1. Unet with batch size of 16 and learning rate of 0.001. ... 11

2.4.2. Unet with batch size of 32 and learning rate of 0.01. ... 12

2.5. Data augmentation ... 12

2.6. Results ... 13

Conclusion ... *15*

Introduction

Future vision transport is a company that designs onboard computer vision systems for autonomous vehicles. Computer vision system have four main components:

- Real-time image acquisition
- Image processing
- Image segmentation
- Decision making system

The scope of this project focuses on the third component: image segmentation. This component receives an image from the processing component as input, and provides as an output a mask containing a label for each object detected in the input image. This mask will then be pass on to the decision-making system.

Additionally, image segmentation provides not only a label for each object detected in the picture, but also the contour of said object.

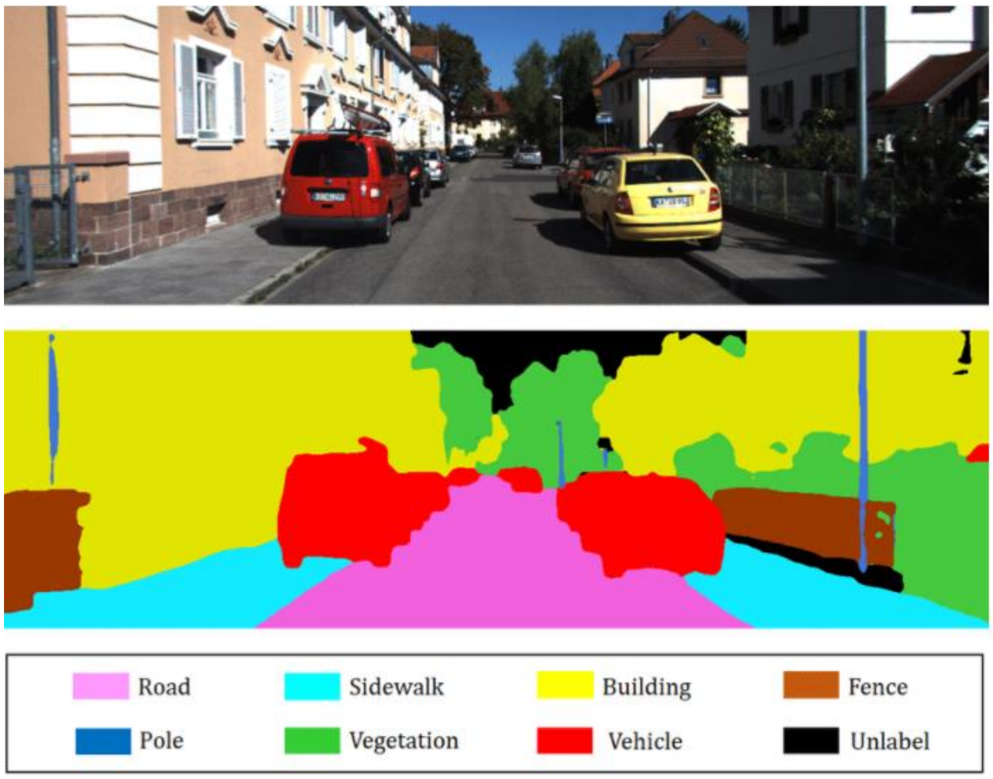


Figure 1: illustration of image segmentation

We trained the model on a dataset containing 1500 pictures of urban traffic. These pictures were taken in various European cities and with the point of view of a car. Four different masks are provided for each picture.

Due to the limitation of available compute power, we tested only a few value for hyper-parameters and only on the u-net model. The project also tested the impact on metrics due to different batch and dataset sizes.

1. Image segmentation

1.1. Overview

In image classification task the model predict a label for each image given as input. In image segmentation, we want to know the shape of that object which is identifying which pixel belongs to the form. The purpose of a model in image segmentation is to predict a label for each pixel in the picture.

Additionally, instance segmentation takes the concept a step further by identifying different type within the same class. However, the scope of the project will be limited to regular image segmentation with eight classes.

The eights classes are defined as follow:

- Void
- Flat
- Construction
- Object
- Nature
- Sky
- Human
- Vehicle

1.2. Metrics

1.2.1. Pixel accuracy

Pixel accuracy is the percentage of pixel in the input image that are correctly predicted by the model. When using pixel accuracy we face the same issue as using accuracy in an imbalanced set. If the amount of pixel to identify as a specific form is proportionally low compare to the amount of background's pixel, accuracy might be high even though some forms are not identified correctly.

1.2.2. Dice coefficient

The dice coefficient is 2 * the area of Overlap divided by the total number of pixels in both images:

$$\frac{2 * |X \cap Y|}{|X| + |Y|}$$

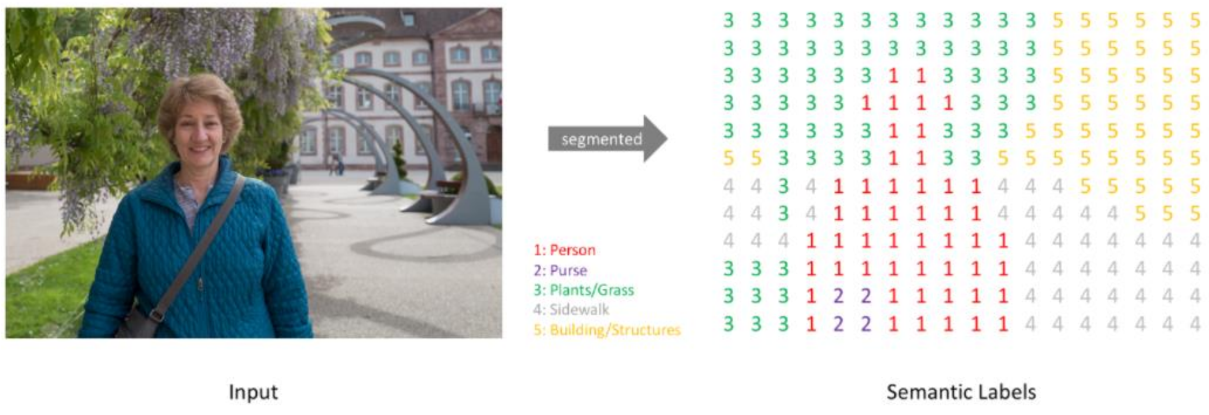
Where X is the predicted set of pixels and Y is the ground truth.

Area of Overlap represent the number of pixel correctly predicted for each class. The dice coefficient is the equivalent of F1 score it gives a better understanding of performance model with imbalanced dataset, which is often the case in image segmentation.

1.3. Architecture

1.3.1. Target

As stated in the overview the aim is to predict the value of a pixel, each value corresponding to a class. To avoid imposing ordinal order and similar to how categorical values are treated our target will be one hot encoded. Instead of having a target of dimension height*width*1 with value for each pixel ranging from 1 to n (n being the number of classes) we will create a target of dimension height*width*N. Every one of the N masks of dimension height*width corresponding to a class for which every pixel have a value of 0 or 1.



```
3 3 3 3 3 3 3 3 3 3 3 3 5 5 5 5 5 5
3 3 3 3 3 3 3 3 3 3 3 3 5 5 5 5 5 5
3 3 3 3 3 3 1 1 3 3 3 3 5 5 5 5 5 5
3 3 3 3 3 1 1 1 1 3 3 3 5 5 5 5 5 5
3 3 3 3 3 3 1 1 3 3 3 5 5 5 5 5 5 5
5 5 3 3 3 3 1 1 3 3 5 5 5 5 5 5 5 5
4 4 3 4 1 1 1 1 1 1 4 4 4 5 5 5 5 5
4 4 3 4 1 1 1 1 1 1 4 4 4 4 4 5 5 5
4 4 4 1 1 1 1 1 1 1 1 4 4 4 4 4 4 4
3 3 3 1 1 1 1 1 1 1 1 4 4 4 4 4 4 4
3 3 3 1 2 2 1 1 1 1 1 4 4 4 4 4 4 4
3 3 3 1 2 2 1 1 1 1 1 4 4 4 4 4 4 4
```

Figure 2 target semantic label

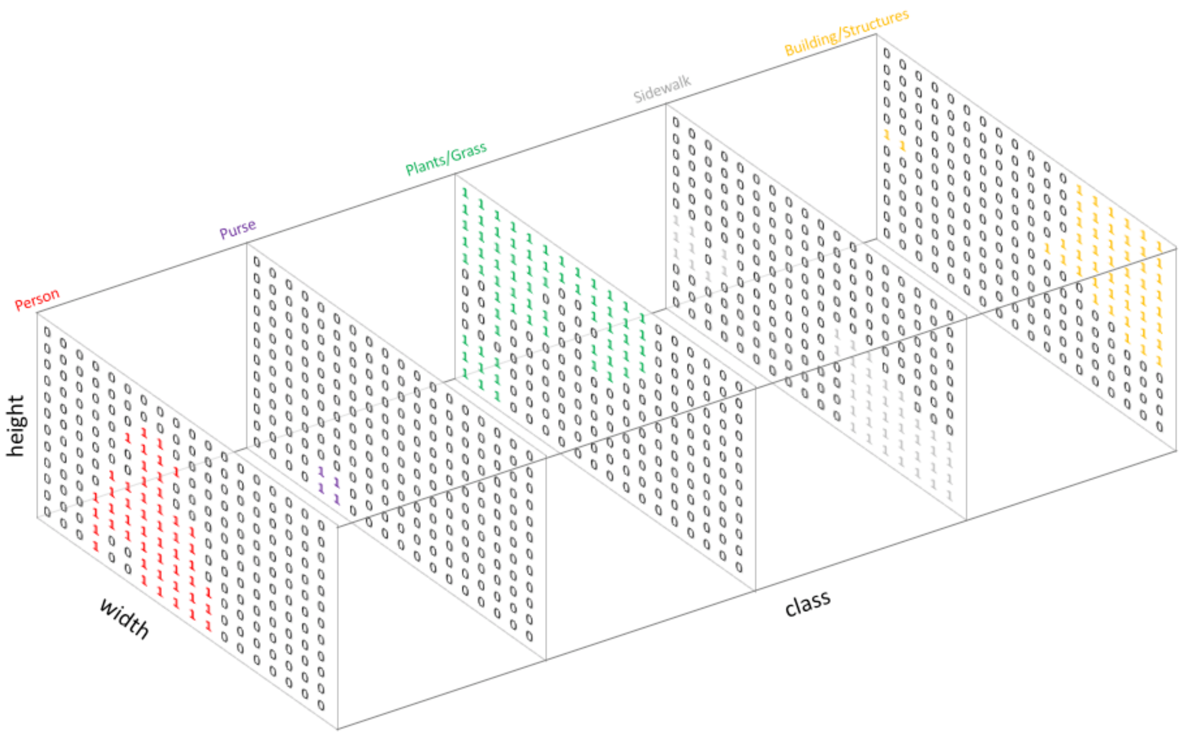


Figure 3: target one hot encoding

To visualize the prediction we use argmax function along the z axis and assign a color to each label.

1.3.2. Down and up-sampling

The architecture used generally for image classification consist of a Convolutional Neural Network of a few convolutional and pooling layers followed by a few fully connected layers at the end. The output is then a feature map representing a heatmap of the required class. The heatmap of the object we want to detect is still valid information for segmentation we want to keep it. This feature map obtained is down sampled due to the set of convolutions performed.

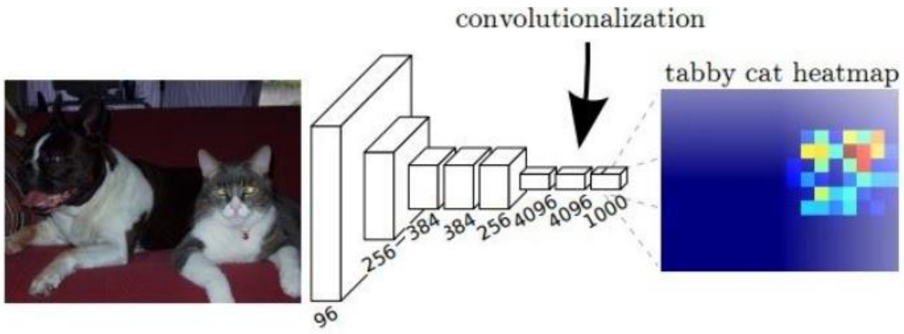


Figure 4: convolutionalization

Since the feature map obtained at the output layer is a down sample, we want to up- sample it using interpolation. Bilinear up sampling or learned up sampling can be used for that process. The down sampling part of the network is called an encoder and the up sampling part is called a decoder. Pooling layer in the down sampling process is necessary to reduce the amount of parameters.

Initial model none as FCN-32 is based on this architecture but the output observed lack information. The loss is due to down sampling 32 times with convolution layers and then up-sampling 32 times. Improvement can be achieved by using FCN-16 and FCN-8 where information from the previous pooling layer is used along with the final feature map.

1.3.3. U-net

U-net consists of an encoder which down-samples the input image to a feature map and the decoder which up samples the feature map to input image size using learned deconvolution layers.

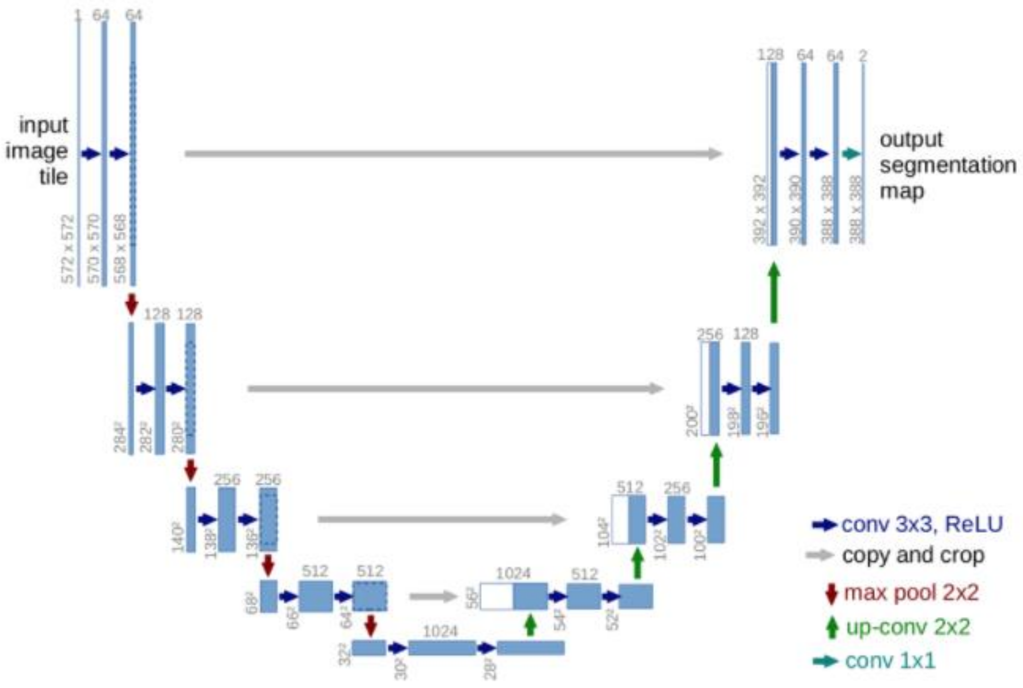


Figure 5: U-net architecture

To solve the information loss problem U-net send information to every up-sampling layer in the decoder from the corresponding down-sampling layer in the encoder.

The encoder is made of three convolutional blocs containing two *conv2D* layers followed by a *MaxPooling2D* layer followed by a *Dropout* layer (which is optional).

The decoder is a succession of three deconvolutional blocs containing one *Conv2DTranspose* layer followed by a *concatenate* layer that concatenate the previous layer with the corresponding layer in the encoder, followed by a *dropout* layer followed by two *Conv2D* layers.

Conv2DTranspose layer perform the up-sampling same as a convolution but backward and by preserving the same connectivity. The weight in the transposed convolution are learnable.

Concatenate layer is used to give information during the up-sampling step as it perform the operation with the corresponding layer encoder.

1.3.4. Deeplab

Deeplab has been developed by a google research team. They offer several techniques to improve the existing results and decrease computational costs. The three main improvements suggested as part of the research are as followed:

- Atrous convolutions
- Atrous spatial pyramidal pooling
- Conditional random fields usage for improving final output.

Atrous convolution aimed at solving the excessive downsizing due to consecutive pooling operations. Atrous convolution is also known as hole convolution or dilated convolution and help keeping information in larger context for the same amount of parameters. Dilated convolution works by increasing the size of the filter by adding zeros to fill the gap between parameters. Let consider a 3x3 regular convolution filter, with a dilation rate of 2, one zero is inserted between every other parameter making the filter look like a 5x5 convolution. Now the filter get the context of a 5x5 convolution while having 3x3 convolution parameters.

In Deeplab the last pooling layers are modified to have a stride of 1 instead of 2, which keep the down sampling rate to only 8x (instead of 32). Then it applied a series of atrous convolutions instead of regular convolution to capture the larger context.

Spatial pyramidal pooling was introduced to capture multi-scale information from a feature map. Before SPP the methods was to process an image at different resolution. The input image is convolved with 3x3 filters of dilation rates 6, 12, 18 and 24 and the outputs are concatenated together.

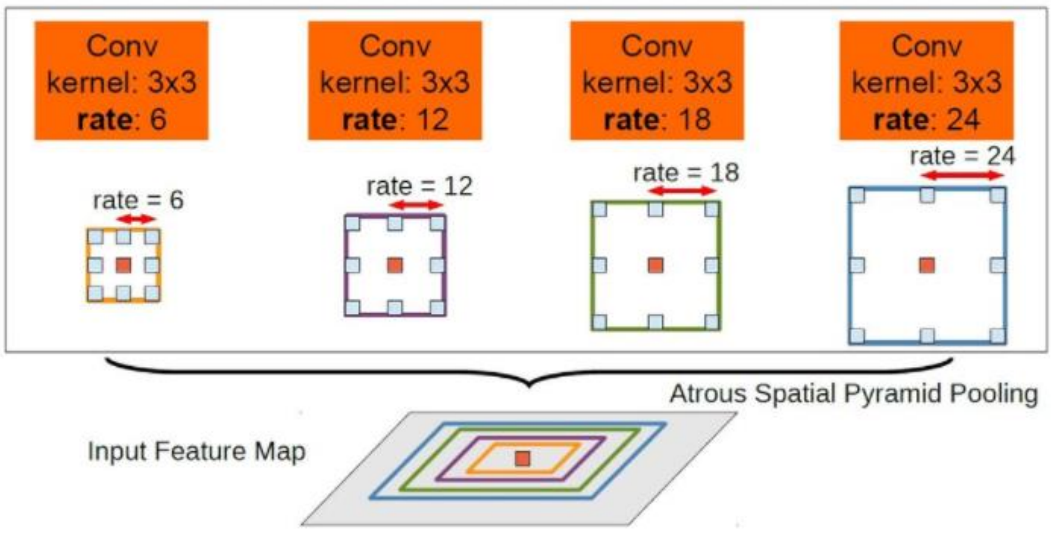


Figure 6: Atrous Spatial Pyramid Pooling

Conditional Random field is a post-processing step that tries to improve the result by defining sharper boundaries. The classification process for pixels take into account the surrounding pixels

2. Training

2.1. Pipeline structure

Pipeline used for the project has been build using Azure Machine learning Studio. The first step for the pipeline is dedicated to data augmentation. Second step is training and model registering. Third step is inference.

2.2. Data augmentation

We used the *Augmentor* package, which presents the main advantages of being easy to implement. It also allows the user to modify the original image with the mask as a pair. The augmentation was built with the following features with different magnitude:

- Rotation (radom rotation of the picture)
- Horizontal flip (reversing the entire rows and columns of an image pixels)
- random zoom
- Perspective skewing left, right, top, bottom and corner (transform the image so that it appears that you are looking at the image from a different angle)
- Elastic distortion (random distortions while maintaining the image's aspect ratio)

However, *Augmentor* presents a few important downsides worth mentioning:

- *Augmentor* automatically save the augmented images and masks in the same output folder. There is no option to save them in different folders. The output name can be used to sort them out prior to training the model.

Due to limited capacity for storing pictures and compute time, the number of augmented picture has only been fixed to 2000. Which means the baseline will be tested on 1500 and we will test the effect of data augmentation on a set of 3500 pictures.

See dedicated section for results.

2.3. U-net and Deeplab baseline

2.3.1. U-net

The baseline U-net model has been trained using *Adam* as the optimizer, we used the basic value for learning rate of 0.001. Adam is known to be a good starting point among adaptive optimizers especially with sparse data. Image segmentation does not properly fall into that category, however it could be considered as close to that when the background takes the majority of the picture. (The mask for classes like pedestrian of panel signalization contain a lot of 0 for only a few 1). Initially the dataset was not augmented and the dataset was divided into train, validation and test set as follow: 70% for training, 20% for validation and 10% for test. Batch size was fixed to 32 for the baseline. The model was trained on 150 epoch using a GPU. Training lasted 8 hours and 20 minutes and the model contained 135,294,656 parameters from which 26,368 are non trainable. See figure below for performance.

Validation Dice coefficient increased to 0.85 after 50 epoch and we observed only small increased in the following epoch (0.865 after 150 epoch). See table below.

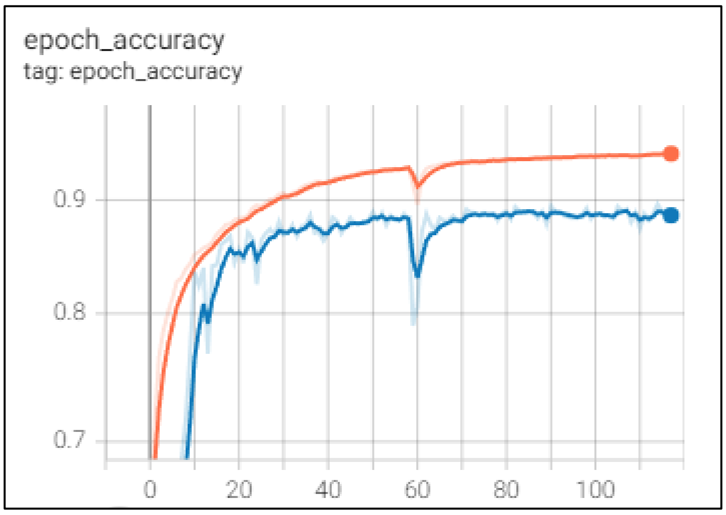


Figure 7: accuracy vs epoch baseline

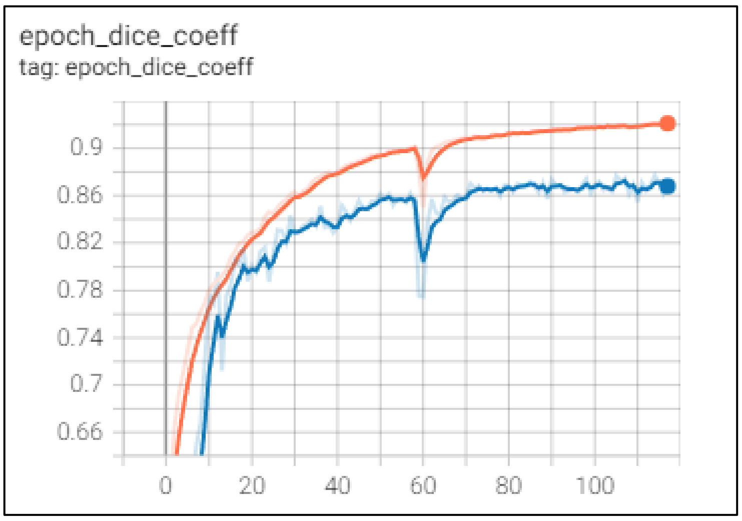


Figure 8: dice coefficient vs epoch. Baseline.

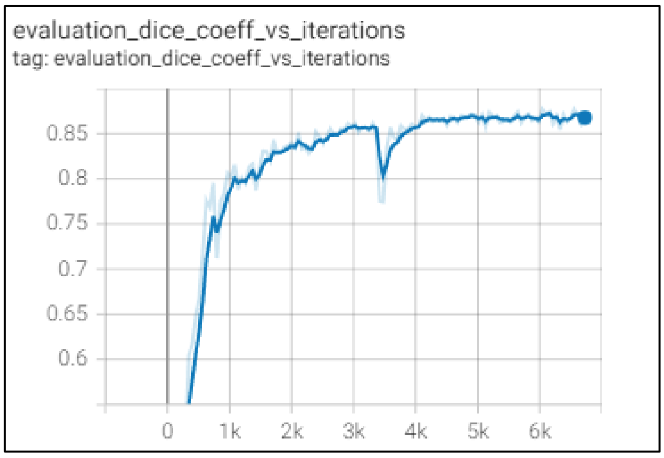


Figure 9: dice coefficient vs iteration. Baseline.

Looking at the loss evolution, we can clearly see the moment were the validation loss and the training loss diverge. This can be interpreted as a warning that he model is overfitting after 30/50 epochs. See table below.

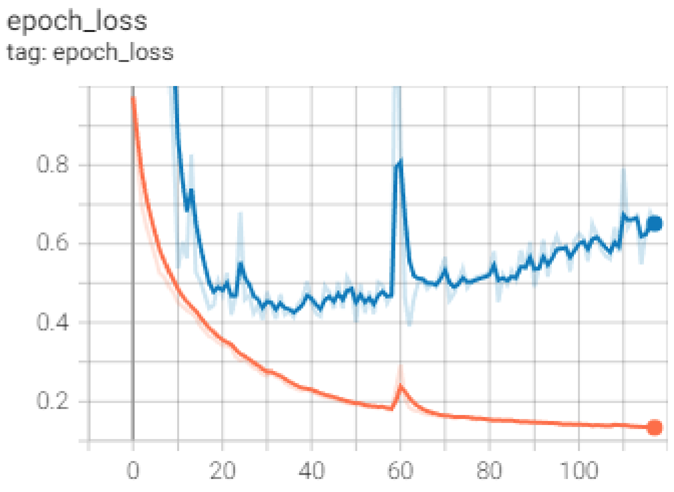


Figure 10: loss vs epoch. Baseline.

Given the previous observation regarding the evolution of accuracy and dice coefficient after 50 epochs, we will limit the number of epochs during the hyper-parameters tuning phase.

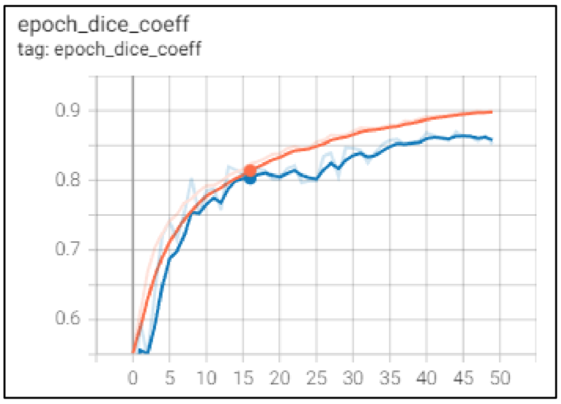
2.3.2. Deeplab baseline

The baseline U-net model has been trained using *adam* as the optimizer, we used a learning rate of 0.001. The dataset was not augmented and was divided into train, validation and test set as follow: 70% for training, 20% for validation and 10% for test. Batch size was fixed to 32 due to a lack of compute power. The model was trained on 150 epoch but did not reach a maximum value. As suspected the model performed better than unet despite long training time and the validation dice_coeff reached 0.885.

2.4. Hyper parameters tuning

2.4.1. Unet with batch size of 16 and learning rate of 0.001.

Training lasted 3h45 for 50 epoch, validation dice coefficient reached 0.856 after 35 epochs.



. *Figure 9: dice coefficient vs epoch. batch size 16.*

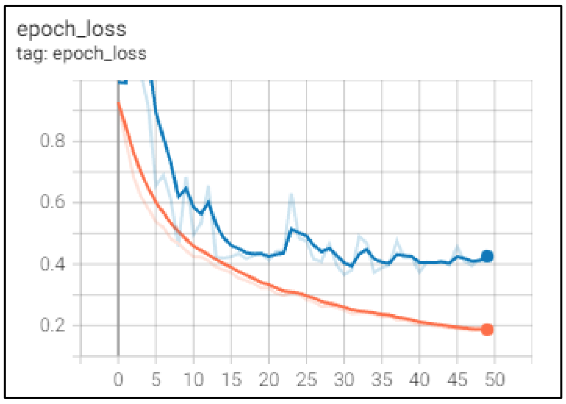


Figure 10: loss vs epoch. batch size 16

2.4.2. Unet with batch size of 32 and learning rate of 0.01.

Training lasted 3h42 for 50 epochs, validation dice coefficient reached 0.8335 at epochs 50 but the curve, unlike with the previous parameters, did not seems to settle and was still increasing. The loss' behavior indicates a convergence toward the same values (0.2 and 0.4). These graphics indicates that training could be extended although the loss indicates that we are close to overfit the model.

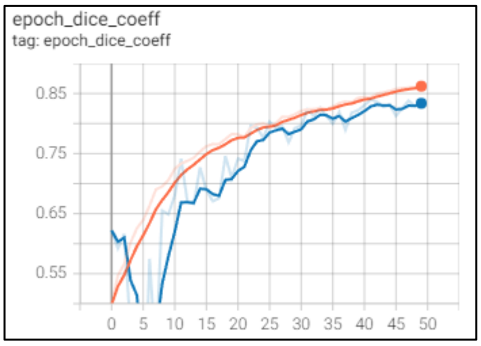


Figure 11: dice coefficient vs epoch. lr = 0.01

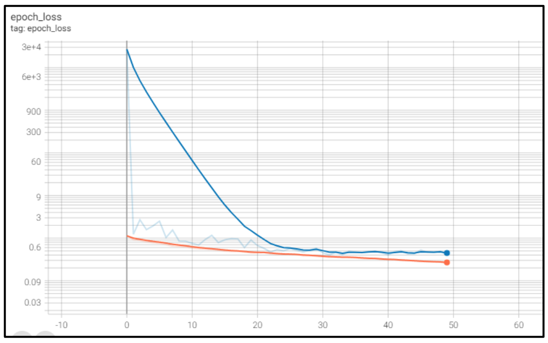


Figure 12 : loss vs epoch. Lr = 0.01

2.5. Data augmentation

As stated, we doubled the number of photo and looked at the impact on the baseline indicator. The number of photo for validation remained the same.

- Timing for training 50 epochs went from 3h30 to 7h30.
- Accuracy and dice coefficient kept slowly increasing until epoch 50, indicating that the model did not converge to a max.
- Performance on validation set were better than on the training set (both for accuracy and dice coefficient). Confirming the conclusion from previous point.
- Validation and training loss converged to roughly the same value with slightly better result after the first 50 epochs. On average, both curves crossed twice without indicating a tendency for divergence. Another point to increase training time in order to increase performance.

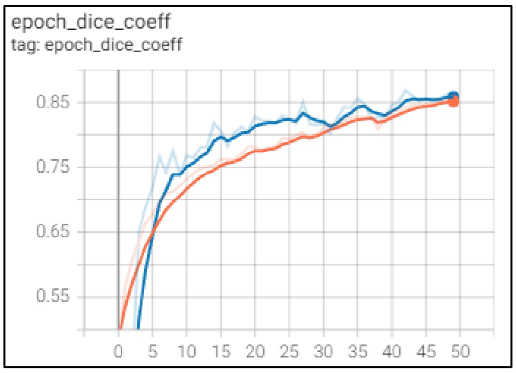


Figure 13: loss vs epoch. With augmentation.

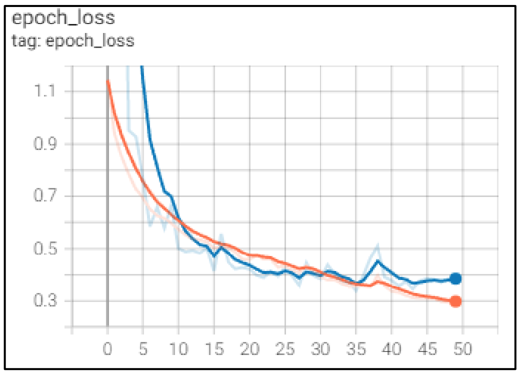


Figure 14: dice coefficient vs epoch. With augmentation.

An additional 20 epochs confirmed that dice coefficient kept increasing almost reaching 0.88. Roughly doubling the amount of data for training allowed the model an increase of the dice coefficient of 0.04 alone compared to baseline model. Although the training time more than doubled from 3h30 to almost 10 hours without indication of overfitting nor having reached maximum value. Impact of data augmentation on the model's indicator is undeniable and more consequent than hyper-parameters tuning. It is likely that training for more could improve performance.

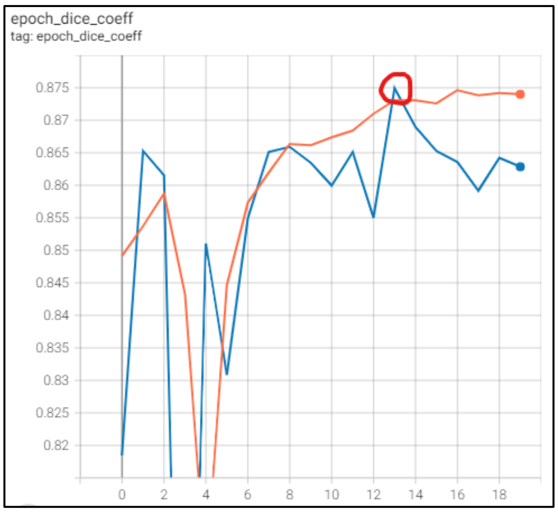


Figure 15 : dice coeficient vs epoch. With augmentation. Epoch 50 to 70.

2.6. Results

As can be observed on the below example the definition in the background is slightly better with the model trained with data augmentation. Those details reflect the increase of the dice coefficient. For example the detection of the signalization panel and pedestrians (respectively red and black box).

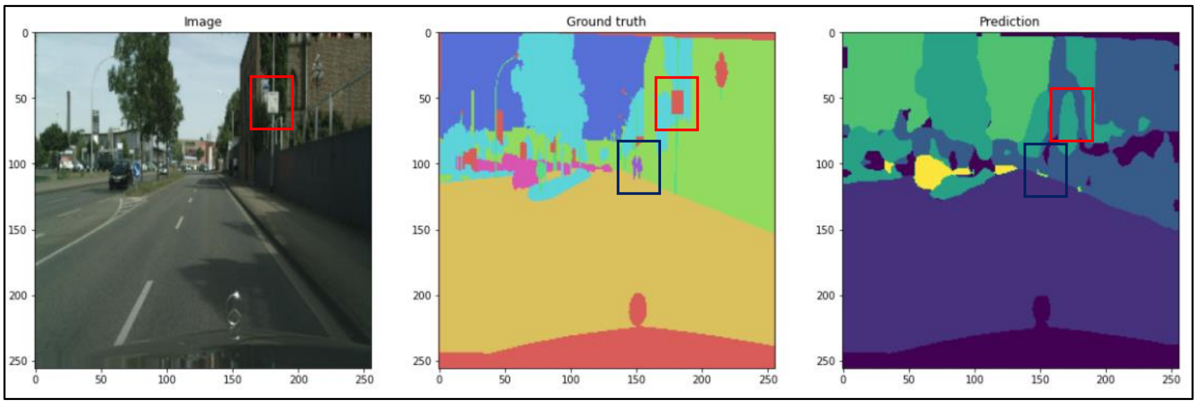


Figure 16 : prediction with baseline model.

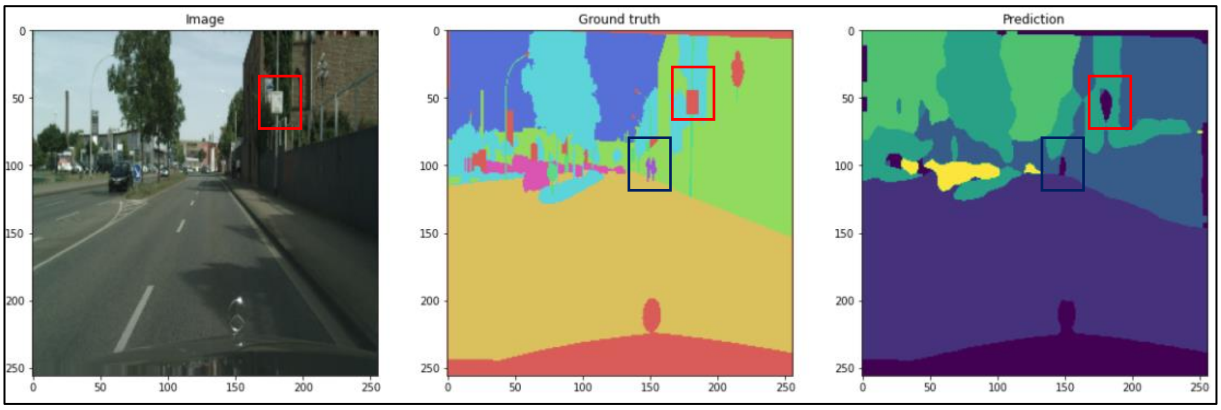


Figure 17: prediction with model trained with data augmentation.

Conclusion

The majority of the training was done using the U-net model although DeepLab is a more recent model. We focused on studying the impact of hyper-parameters and data augmentation on the metrics. The baseline model with no data augmentation needed 35 to 45 epoch to converge toward max performance. Overfitting of the model was clearly identified on the baseline after 60 epoch by looking at the loss evolution. We also observed that the number of epoch required for convergence slowed as learning rate increase (35 epoch for 0.001 vs <50 for 0.01).

We performed data augmentation prior to training to roughly double the number of picture. We did not study the impact on data augmentation when this step is added as an initial layer in the model. If changes in hyper-parameters had an impact on convergence, we observed small variation on metrics. However, the first run with double the amount of picture increased the performance by a significant amount at the expense of training time. Due to limited compute power and computer memory, we could not study the results of an even bigger increase in number of picture. Same limitation applied to bigger batch size.

As a way ahead, it could be interesting to increase further more data augmentation and to try a bigger batch size. It is also worth noticing that the resulting model was quite heavy and exceed most free solution for deployment including the azure subscription supplied by OpenClassRoom.