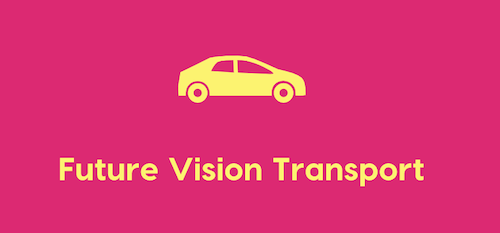
Technical Report

Designing an Image Segmentation Model for a Computer Vision System

[](https://user.oc-static.com/upload/2019/10/24/15719060749143_image2.png)

ANTONI Octave

AI Engineering Path

OpenClassrooms

November 3rd 2022

# Abstract

During this project, an Image Segmentation Model planned to be used in a Computer Vision System has been developed. The goal was to find an efficient model that could be used to process images in real time. Data augmentation techniques have been applied to improve the performance of the model, and several different models and networks have been tried.

Due to the end of Microsoft Azure Education credits, this project has been performed with limited computing power, which prevented some options from being examined.

The best

# Table of Contents

Table of Contents

[Abstract 2](#_Toc118291687)

[Table of Contents 3](#_Toc118291688)

[1 Introduction 3](#_Toc118291689)

[2 Methodology 4](#_Toc118291690)

[2.1 Image Processing 4](#_Toc118291691)

[4 Results and discussion 5](#_Toc118291692)

[5 Conclusions and recommendations 5](#_Toc118291693)

[6 Acknowledgements 5](#_Toc118291694)

[7 References 5](#_Toc118291695)

[Appendix A: Place the title of appendix here 7](#_Toc118291696)

# 1 Introduction

Future Vision Transport is a company that designs onboard computer vision systems for autonomous vehicle system. This technical note details the development of an Image Segmentation Model that will be used within a state-of-the-art Computer Vision System. This project uses the Cityscrapes dataset that provides a dataset of 5000 segmented images of European cities. As required by the assignment, the model will identify the 8 following categories:

* Flat: Roads, Sidewalks, parking, rail tracks
* Human: Individual on foot or on a bike
* Vehicle: All kind of vehicles
* Construction: Every kind of building, bridge, tunnel
* Object: Poles, traffic signs, traffic lights
* Nature: Vegetation or terrain
* Sky
* Void: Regroups the ground and unidentified static and dynamic features.

# 2 Methodology

There are 3 main parts in the implementation of an Image Segmentation Model:

* Image Preprocessing / Data Augmentation
* Encoding
* Decoding

## 2.1 Image Processing

### 2.1.1 Image Preprocessing

An analysis of the images provided in the Cityscrapes dataset (downloaded from the OpenClassrooms link because direct download requires an .edu email address) reveals that the **Test masks are invalid,** leaving me with only **3475 valid examples.** Since I was unable to obtain the source files from the Cityscrapes creators, I decided to ignore the test set and to manually recreate a test/train/validation (80% / 10% / 10%) split using the splitfolders module.

The initial dataset is composed of 4 types of mask information:

* Color masks: all the classes are assigned different colors.
* Instance Ids masks: grayscale masks labelled with Instance IDs (each different item is assigned a class and a unique object ID)
* Label Ids: grayscale mask labelled with class ID
* Mask polygons: includes detailed coordinates of each segment including instance id and all the classes.

Since the goal of this project was to perform **semantic segmentation**, I used the Label Ids masks for the project. These mask files were 2 dimensional arrays were **each pixel was labelled with the class of the object.** Since we were only interested in the categories of each pixel (as opposed to the classes), I converted the class numbers into category numbers within the data generator. I chose to **one-hot encode the categories in a 3rd dimension** instead of converting the pixel values to the category in order to prepare the masks for model training. The final preprocessed masks were **1024 x 2048 x 8 arrays** where the last dimension is the one-hot encoded category, whereas all the **source images where 1024 x 2048 x 3** arrays where the last dimension is the RGB value of each pixel.

For computation speed purposes, I started with a drastic reduction of image and mask size from 1024x2048 to 256x256, on which I trained my first models. I realized that this degraded performance too much and decided to **increase image size to preserve aspect ratio with a final value of 256x512.** A model trained withnative image resolution will have better performance, but will require much higher computational power to train.

### 2.1.2 Data Augmentation

After training my first models with the base images, I applied data augmentation to increase the performance of the model. Data augmentation techniques were not included in the model training pipeline but were used to create **additional samples.** I used the albumentations module to create 2 sets of augmented copies of each image / mask couple in the test set, **increasing the number of test samples from 2780 to 8340**. One copy was generated with data augmentation applied to the base image, while the second copy was generated with data augmentation applied to a **horizontally flipped version of the base image.** The following sequential data augmentation layers were then applied:

* Rotation: limited at 30° to simulate rolling terrain (probability = 50%)
* Random Brightness, Contrast and Gamma (probability = 70%)
* Elastic Transform or Grid Distortion (one of each, probability = 50%)

Additional data augmentation layers could be applied within the training pipeline to further improve the models. In production, I would advise to add the following augmentation layers within the pipeline : Histogram Equalization and Gaussian Smoothing.

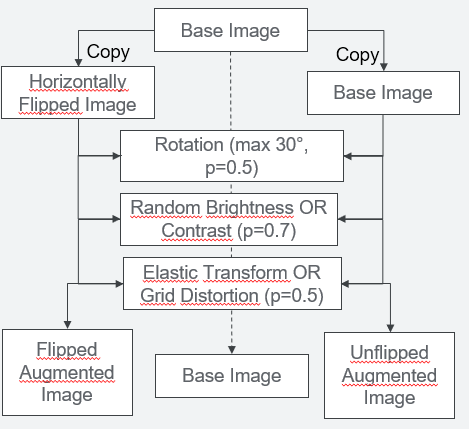


Figure Data Augmentation Pipeline

### 2.1.3 Data Generator

A custom data generator was used to generate images and masks for this project. The particularity of image segmentation is that both images and masks have to be generated before being consumed by the model during training. The generator performed the following sequential steps:

* Resizing of images to the desired height and width.
* One-Hot Encoding the mask categories by creating a 3rd dimension to the mask shapes (as discussed previously)
* Resizing of masks to the desired height and width.

## 2.2 Data Encoder

Figure : Architecture of an Image Segmentation Model

As seen in Figure 1, an Image Segmentation Model is generally composed of an Encoder and a Decoder Model, which could either be part of the same model (like UNet), or different models (Feature Extractor and Classifier). After starting training with a completely untrained model, I realized that **I could not perform efficient training of both the Encoding and Decoding pipeline with the limited computing power at my disposal.**  For reference, the untrained UNET model needed more than 2.5 hours of training per epoch on the base (not augmented) dataset, which would translate to 7.5 hours per epoch with the augmented dataset.

For this reason, I decided to use a **pre-trained feature extractor as an encoder,** and I tested both the **MobileNetV2 and ResNet-18.** MobileNetV2 was used during most of the training runs and had a good performance. I was not able to fully test ResNet-18 because its implementation with FastNet caused stability issues on my computer, although the 5 epochs of training I managed to run had good performance. Both encoders were loaded with the ImageNet weights.

## 2.3 Decoder

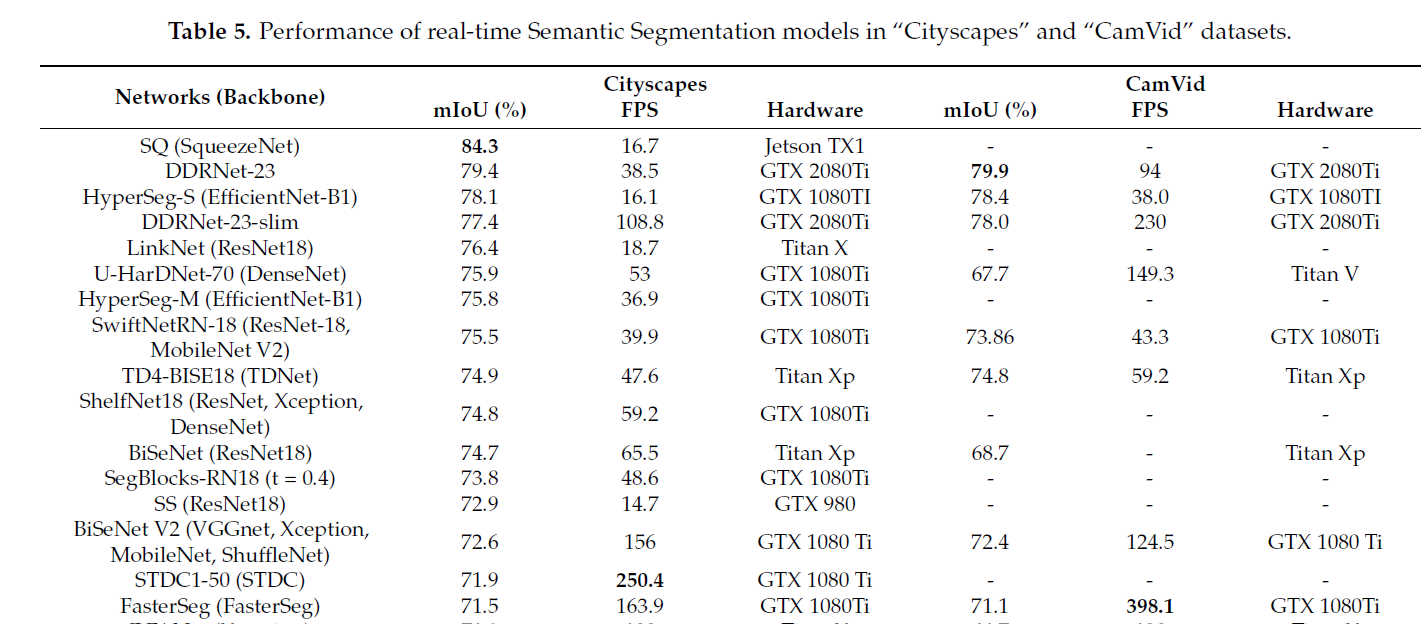
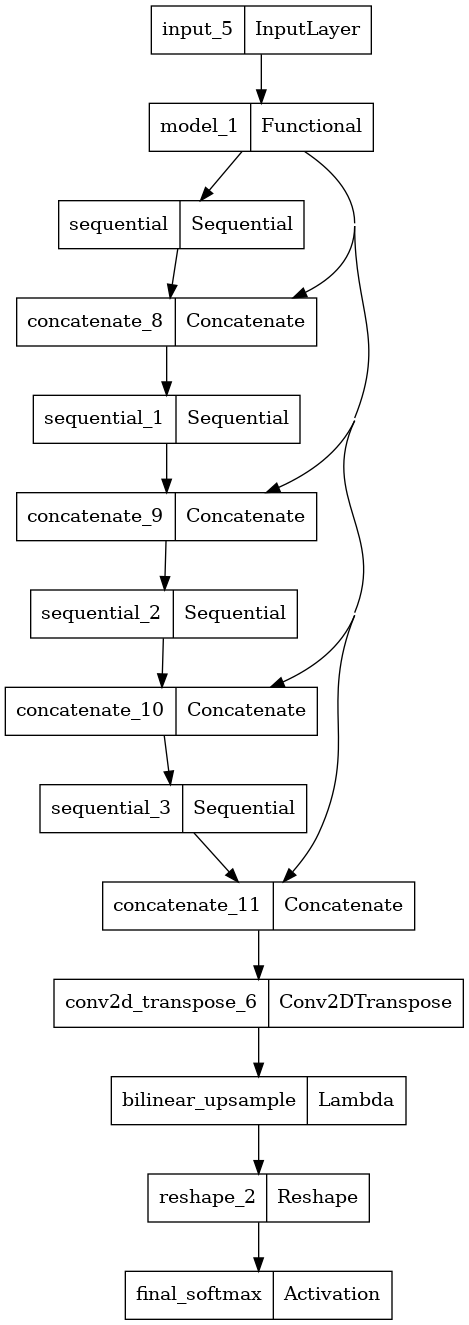
To choose a decoder, I decided to look at state of the art models and studied this research paper : Papadeas, I.; Tsochatzidis, L.; Amanatiadis, A.; Pratikakis, I. Real-Time Semantic Image Segmentation with Deep Learning for Autonomous Driving: A Survey (Reference 1). It contains a very exhaustive description of the state of the art models in Image Segmentation and provides the following summary table :

Figure : Performance of real-time Semantic Segmentation models in the “Cityscapes” and “CamVid” datasets

Even though most of the Encoders (backbones) are implemented in Keras, very few of the networks are implemented. Since Keras is a requirement for this project, I decided to use **UNET with MobileNetV2 as a backbone** for the main part of this project. Please find in Figure 3 the architecture of the model.

The “model\_1” layer is the MobileNetV2 model loaded with ImageNet weights and without top layer. Since our network is based on Unet, we need to establish **skip connections** **between our MobileNetV2 backbone and the UNET up-sampling network**. We use the activation layers of 4 Relu layers of our MobilenetV2 model (block\_1, 3, 6 and 13) for our skip connections. For our up-sampling layers, we import the U-Net layers of the pix2pix model which is already implemented in Tensorflow.

For our classification layers, we cannot use the standard softmax activation layer since we have one-hot encoded our target labels. We have to perform bilinear up-sampling after the last Conv2DTranspose layer to bring our vector back to our initial image size. We then use a Reshape layer to reduce the dimensionality of our data before using a Softmax layer as our final layer.

Figure Architecture of the MobileNet / Unet Network

The final prediction will not directly output a readable mask and we will have to convert it into a grayscale mask (or alternatively RGB mask) by using the create\_mask function that I have implemented in the notebook (see Notebook\_annex for more detail).

The implementation of the model is shown in the Notebook.ipynb file while the full testing of different models is shown in the annex file.

# 4 Results

In this part, we will only display the **test metrics.** Here are the results obtained for our MobileNetV2 – Unet model :

|  |  |  |  |
| --- | --- | --- | --- |
| **Model Settings** | **Dice Coefficient** | **Mean IOU** | **Accuracy** |
| No Data Augmentation  256x256 images | 0.835 | 0.512 | 0.883 |
| Data Augmentation  256x256 images | 0.866 | 0.556 | 0.898 |
| **Data Augmentation**  **256x512 images** | **0.884** | **0.573** | **0.911** |

It is clear that our model with Data Augmentation and 256 x 512 images (native aspect ratio) has the best performances. One issue with this model is **that it started overfitting after the 7th epoch**, which triggered the early stopping layer that I had put as a callback. The training history is displayed on Figure 5:

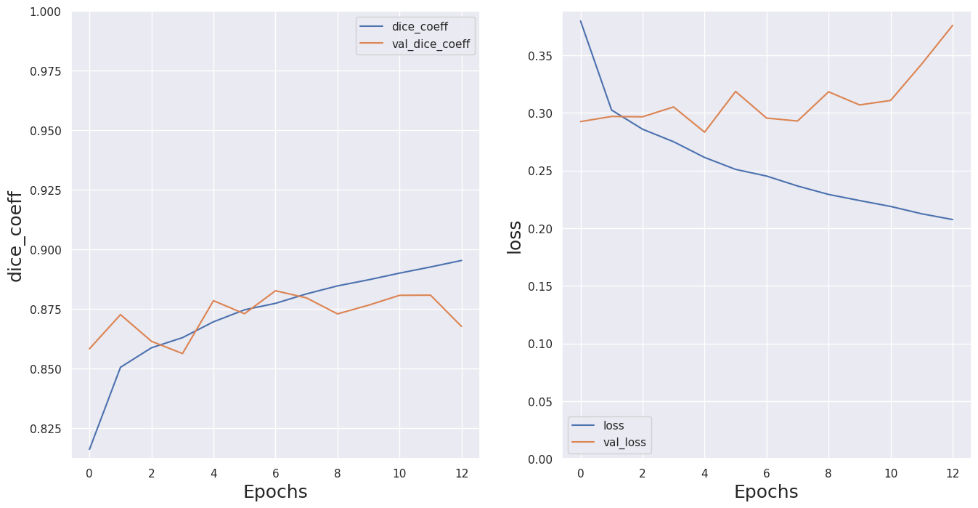


Figure Training Graph of MobileNetV2 - Unet model with Augmented 256x512 images

 We also analyze the shape of our predicted mask:

Figure Sample Prediction for our final model

It seems that our model has a good performance and is able to **adequately predict the categories of the features present in the photo.**

# 5 Conclusions and recommendations

During this project, **a Semantic Image Segmentation model** has been developed based on images extracted from the Cityscrapes dataset. The model developed so far **needs to be refined** before deployment in production

Recommendations for further improvement:

* **Use developed MobileNetV2 – Unet model as base for improvement**
* Use high performance computes 🡺 **increase the number of training epochs.**
* **Increase image size used in training** to native resolution (1024x2048)
* Verify FPS of model to **assert compatibility with Computer Vision System**

# 6 References

[1] Papadeas, I.; Tsochatzidis, L.; Amanatiadis, A.; Pratikakis, I., Real-Time Semantic Image

Segmentation with Deep Learning for Autonomous Driving: A Survey. Appl. Sci. 2021, 11, 8802. <https://doi.org/10.3390/app11198802>

[2] Exploration of Optimized Semantic Segmentation Architectures for edge-

Deployment on Drones: Vivek Parmar, Manan Suri, Narayani Bhatia, Shubham Negi <https://www.researchgate.net/publication/342733846>