E.T.S. de Ingeniería Industrial,
Informática y de Telecomunicación

# Study on the application of different image preprocessing algorithms in image segmentation using deep learning techniques

Grado en Ingeniería Informática

Trabajo Fin de Grado

Iñaki Velasco Rodríguez

Daniel Paternáin Dallo

Pamplona, 21/01/2022

## *ABSTRACT*

Semantic segmentation is one of the most important fields of computer vision due to its applicability. In this case, semantic segmentation will be applied to images taken from vehicles knowing the importance of autonomous driving for our future.

This project aims to achieve precise and fast results in the field of semantic segmentation with the complication of using less powerful and more affordable hardware than the one used nowadays.

Deep learning techniques will be used to solve this problem. More concretely, a specific model of convolutional neural network will be trained and in charge of making the predictions, a U-Net.

Different parameters of the U-Net will be changed to study how they affect the results. Furthermore, various image sizes, color spaces or reduction methods will be applied to study their impact on the speed and accuracy of the U-Net predictions.

Finally, all those results will be compared in order to make a final decision in which is the best combination and which fields impact the most and how.

## *KEYWORDS*

Deep learning, computer vision, semantic segmentation, convolutional neural network, reduction.

## *ACKNOWLEDGMENTS*

I am really grateful to Daniel Paternáin Dallo, director of this project, and to every other people that has contributed for guiding and helping me after so much work.

# INDEX

# 1. Introduction and objectives

Everybody has heard about artificial intelligence (AI), but not so many people know about the real importance of this field. From improving daily photos taken with a phone to cancer detection without human help, AI has strongly entered in our daily lives and will revolutionize the world we know.

Even though there is a lot to learn and improve, AI has demonstrated that machines can work better and faster than humans even in the most complex tasks. Industry is investing more and more budget each year to implement AI technologies that will improve their development. Why? Manufacturing companies have experienced high increases in their productivity by introducing robots or demand predictors, financial sector is implanting chatbots and fraud detector mechanisms, automotive sector is developing driving aid in their products or even creating self-driving vehicles…

Some of these complex tasks require, as can be assumed, complex "thinking" processes that will enable them to solve those problems. Here is where deep learning plays his role. Unlike machine learning algorithms, deep learning will attempt to mimic the human brain [1]. While machine learning needs more human intervention to learn and improve its predictions, deep learning learns by itself to recognize the important patterns that will allow it to make its decisions.

One of the most important complex tasks is computer vision. It is easy for a human being to see an incoming car and stop to avoid it or to see the cloudy sky and take an umbrella. But to make these complex decisions it is firstly needed to analyze the actual image and detect and correctly classify all the elements in it. This is what is called semantic segmentation.

One of the most prominent applications of this field is autonomous driving. Although driving is fun for some people, a necessary activity for others or even a job for certain people. Anyway, self-driving vehicles will be part of our society in a few years providing advantages like crash reduction, mobility for everyone (not only those with license such as reduced vision people) or less traffic jams (this will imply less contamination) [2]. This field will be the main application of this study as it is considered to be a very prominent and important aspect in our futures lives.

As stated before, it all starts with image segmentation. However, this simple task for a human is not that simple for a computer. Therefore, deep learning will try to imitate human brain and will use techniques such as deep neural networks with the aim of reaching better and faster decisions than humans.

This will lead to another problem: having a device in charge of executing all those operations. Currently, very powerful, expensive and scarce hardware is usually needed. Therefore, it is necessary to study and develop cheaper hardware as well as lighter models, so as to easily embed them into cars, or any other device.

One of the simplest methods to reduce model training time (very long when using not so powerful devices) is the reduction of image resolution which allows to have simpler and lighter models. This reduction can be achieved by applying algorithms such as interpolation or sliding window, but these can cause accuracy loss in the model. Therefore, image reduction will work along with more complex algorithms that use superpixel models or even feature extractors.

With all the above, the objective of this work would be to analyze, based on deep learning techniques involving convolutional neural network, different image preprocessing algorithms with the aim of reducing computational cost while maintaining accuracy. Specifically, we will deal with color spaces, superpixel obtention, feature extraction processes such as Local Binary Pattern and image reduction methods.

The structure of this project is as follows. In **2. Starting point** we describe the starting point of this work by explaining the model, dataset and hardware & software. In **3. Preprocessing and reduction techniques** all those techniques previously mentioned in the last paragraph. Then, in **4. Experimental framework**, important concepts involving experimentation such as metrics will be described. This experimentation will take place in **5. Experiments** and the document will finish with conclusions and future research lines in **6. Conclusions and future lines**.

# 2. Starting point

In this chapter we explain the starting point of the project. First, we will describe the paper which is the basis of this project as well as the deep learning based model (U-Net) used to apply semantic segmentation. We will also analyze the training set as well as hardware and software used to develop the experiments.

## 2.1. Basis

Mainstream techniques to deal with semantic segmentation is the use of convolutional neural networks with an encoder-decoder architecture like U-Net.

Some benchmarks of different procedures, methods and models applied to the same dataset used in this project (Cityscapes dataset) can be consulted in their official page [3].

One initial approach which resembles the objective of this project is the article “Superpixel-based Road Segmentation for Real-time Systems using CNN” [4] as it focuses on the issue of balancing accuracy and computational cost and its core idea is to reduce the computational complexity of the problem.

## 2.2. Model (U-Net)

Detection and classification of objects in real life images requires a method capable of determining the class of each pixel also taking into account information about surrounding pixels. Deep convolutional neural networks are the perfect approach to solve this problem.

Their strength resides in their layering which allows them to solve complex problems by detecting different patterns. Several architectures exist nowadays, but not all of them are valid in this scenario.

Some of them provide a unique classification result from a whole image, such as the LeNet-5 architecture, one of the best known ConvNet architectures, created in 1998 and commonly used for digit recognition. Its architecture is shown below and the main concepts of each layer, such as convolutions or pooling, will be explained later.

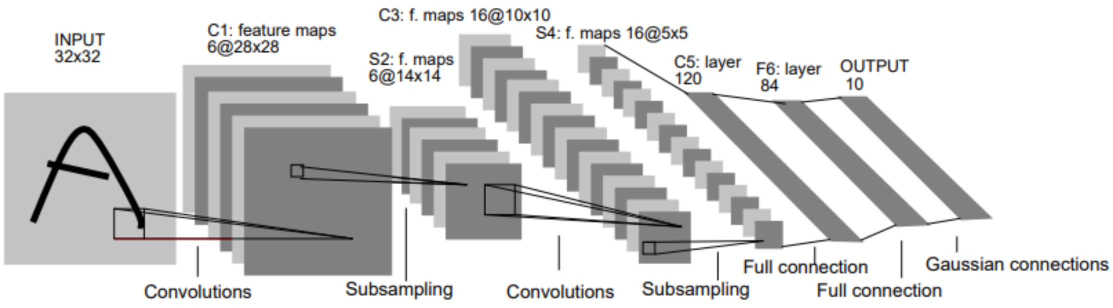


Image 1. LeNet-5 architecture [5]

However, this architecture is not useful for this application since, as discussed above, we need to classify each pixel of the image to obtain the final segmented one, and not a unique class for the whole image which is the main purpose of the architecture shown above.

In order to obtain what we need, U-Net deep convolutional network architecture will be used. This architecture will obtain an image with the same size as the input one where every pixel will have its own classification.

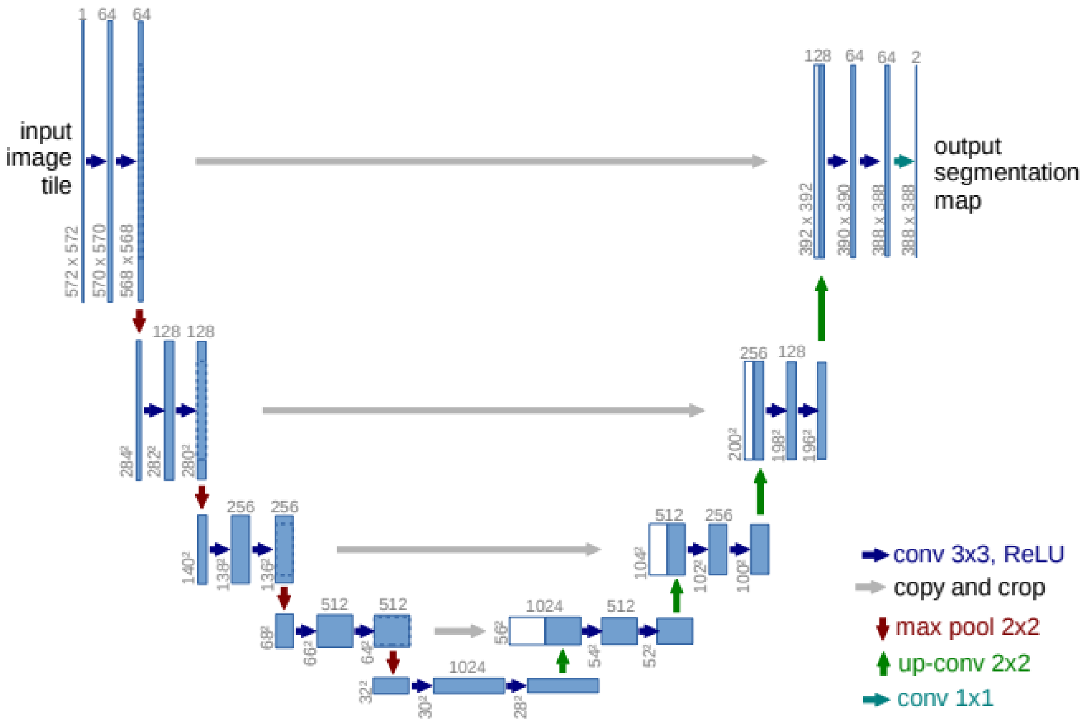


Image 2. Example of U-Net architecture [6]

As it can be seen, its name is due to its shape which is caused by the two parts that compose it: contracting path and expansive path.

## 2.2.1. Basics

First of all, some basic operations of deep convolutional neural networks will be explained in order to clarify all concepts.

As will be mentioned later in **2.4 Hardware & Software**, Python with Keras and TensorFlow have been used in this project. Therefore, all code shown below uses these technologies.

Convolution [7]:

This is the most important operation in the network. It involves the multiplication of the weights of the kernel with the input. By performing this operation several times during training, different patterns will be learnt so that objects can be correctly classified.

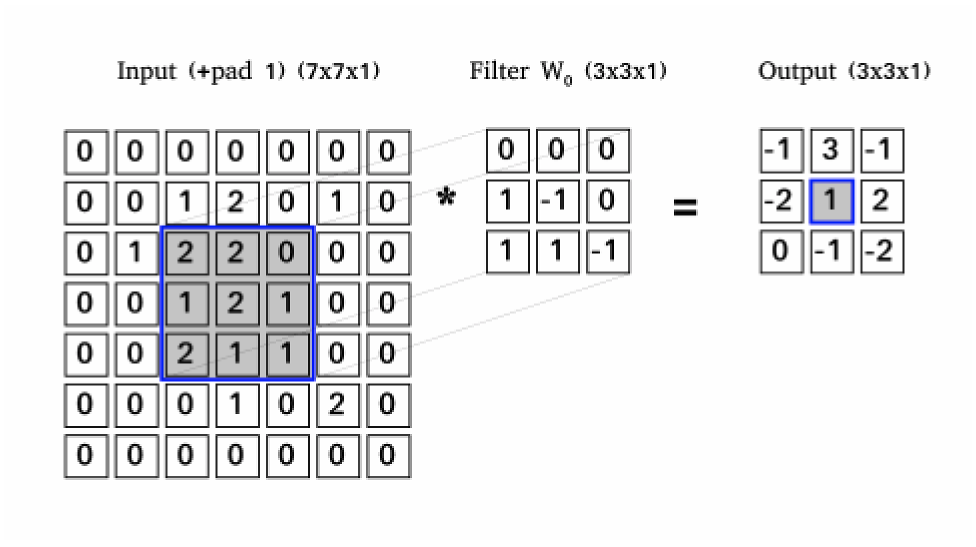
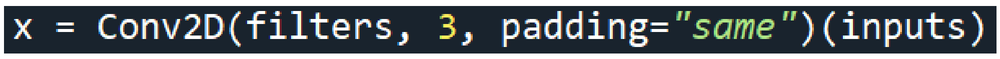


Image 3. Example of convolutional filter applied to an image [8]



```
x = Conv2D(filters, 3, padding="same")(inputs)
```

Code 1. Convolution (tensorflow.keras.layers.Conv2D)

As it can be seen, Keras offers a specific method for this operation with multiple input parameters:

filters: number of output filters of the convolution. Thus, depth of the resulting image.

3: it corresponds to kernel size (3x3 pixels) of the convolution window. Hardcoded in this case as it remains the same for the whole experiment.

padding: in this case “same” as it will allow to have the same output size as the input. “same” will imply padding with zeros.

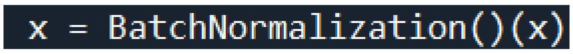
inputs: input layer for the convolution. It will be height X width X 3 for the first layer of the network if images are RGB.

There are other parameters, but they will remain with their default values.

<u>Normalization</u> [9]:

Overfitting is a problem in training. One of the methods to prevent it, obtain a better generalization and speed up training is normalizing the values.

BatchNormalization has been used in this model.



```
x = BatchNormalization()(x)
```

Code 2. Normalization (tensorflow.keras.layers.BatchNormalization)

Parameters:

All function parameters will remain default.

x: input from the previous layer.

<u>Activation</u> [10]:

Convolutions are followed by a nonlinear activation function in this model. ReLu (Rectified Linear Unit) function is used in this model, which will replace negative numbers of the filters with zeros.

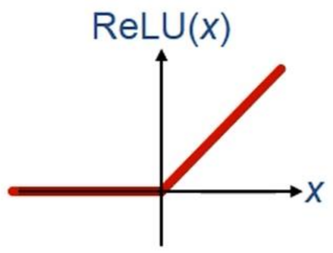
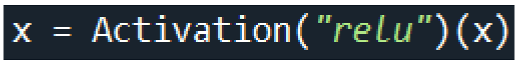


Image 4. Distribution of ReLu function.

Therefore, only the valid part of the convolution will be used.



```
x = Activation("relu")(x)
```

Code 3. Activation (tensorflow.keras.layers.Activation)

Parameters:

“relu” parameter in this function indicates the activation function to use.

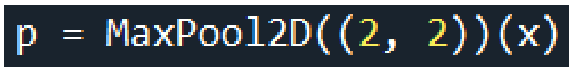
x: input from the previous layer.

There are other parameters, but they will remain with their default values.

Pooling [11]:

The pooling/downsampling layer is the layer in charge of the reduction of the spatial size of the activation maps.

Different types of pooling are available, but max pooling has been chosen for this project. This pooling will propagate the maximum activation value from each window to the next feature map.



```
p = MaxPool2D((2, 2))(x)
```

Code 4. Pooling (tensorflow.keras.layers.MaxPool2D)

Parameters:

(2, 2): it corresponds to the window size. This means that the MaxPool2D function will take the biggest value of every 2x2 window for each input channel

x: input from the previous layer.

There are other parameters, but they will remain with their default values. It is important to mention that there is a parameter called strides that determines how much the window moves between each pooling step. It will default to the window pooling size so windows will not overlap between them.

Up sampling [12]:

This layer will use a learnt kernel to map each feature vector to the corresponding window. Same as the convolution layer, it will be followed by a nonlinear activation function already explained.

This layer will increase the spatial size, contrary to the pooling operation.

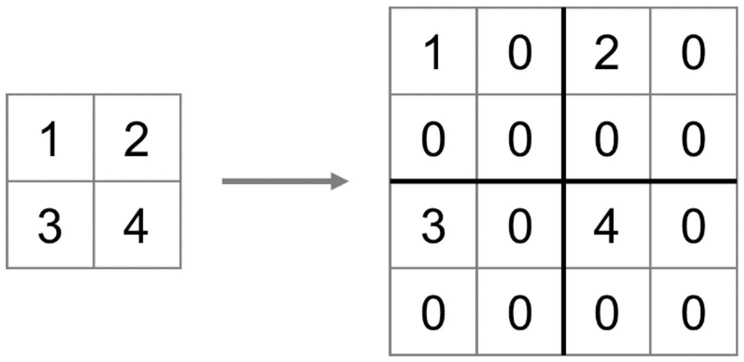
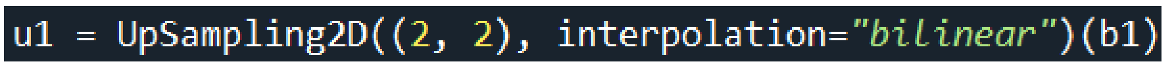


Image 5. Resulting image (right) of applying upsampling (Zeros will be calculated depending on the method chosen in the following function) [13]



```
u1 = UpSampling2D((2, 2), interpolation="bilinear")(b1)
```

Code 5. Up sampling (tensorflow.keras.layers.UpSampling2D)

Parameters:

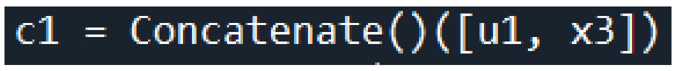
(2, 2): upsampling factors for rows and columns so one cell will be transformed into a 2x2 window

b1: input from the previous layer.

interpolation: interpolation method used to obtain the new values. In this case “bilinear” interpolation which is a way to calculate values depending on the average value of the corners.

<u>Concatenation</u> [14]:

This layer will make its operations after the upsampling layer. This will work along to constitute the expansive path and the concatenation layer will concatenate maps obtained by the upsampling layer with those of the contracting path (explained in more detail later).



```
c1 = Concatenate()([u1, x3])
```

Code 6. Concatenation (tensorflow.keras.layers.Concatenate)

Parameters:

All function parameters will remain default.

[u1, x3]: input from the previous layer, list of vectors to concatenate.

## 2.2.2. Architecture

As it is mentioned before, U-Net architecture is divided in two main parts: contracting and expansive path. This special structure allows the network to obtain great results.

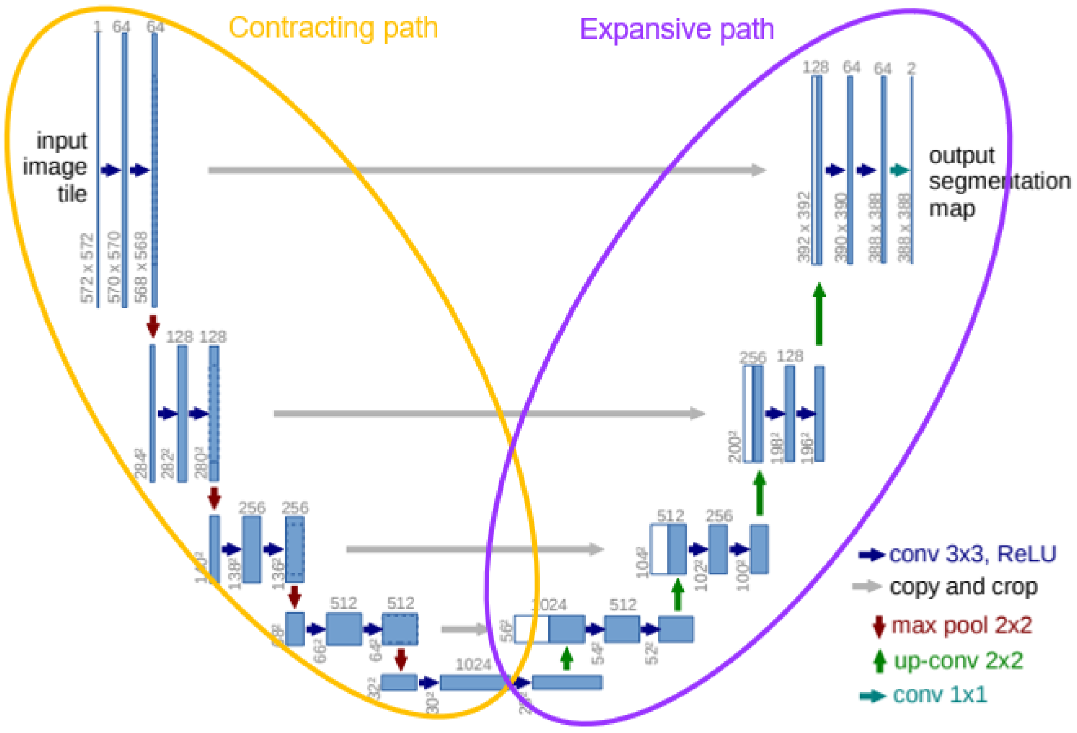


Image 6. Contracting and expansive path of the U-Net architecture

Contracting path:

As it can be seen in the previous image (rounded in orange), it is the first part of the architecture and is also called encoder.

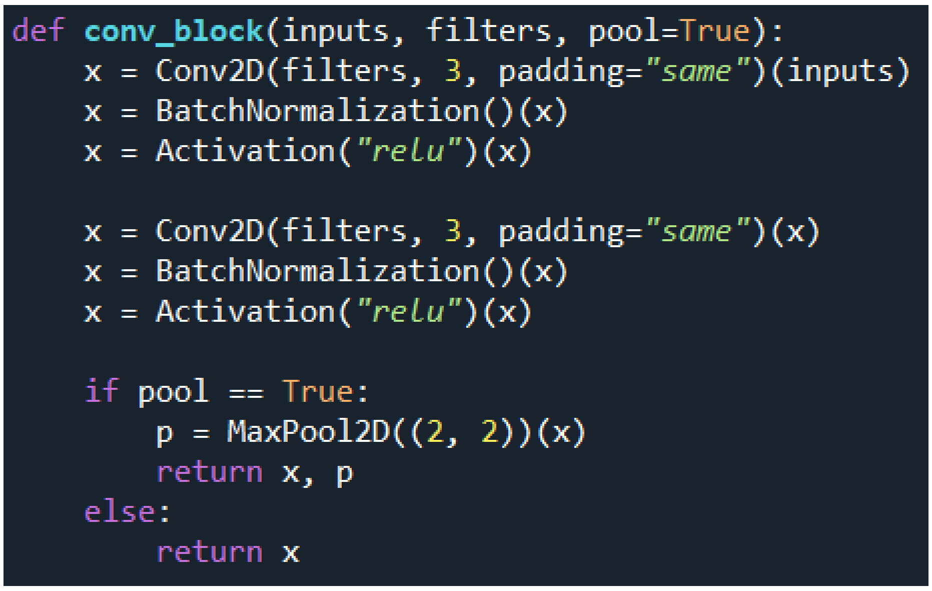
While we get down, we increase the "what" and reduce the "where" as more convolutions and pooling are being applied. This means that the spatial resolution of the feature maps is reduced by the pooling operations, while increasing number of convolutions in deeper layers produces a big number of feature maps.

In this project two U-Net models have been used with slight differences between them: one with 3 steps and one with 4 steps that will be compared afterwards.

Each contracting steps will be constituted by one convolution block following the next structure:

(All operations can be consulted in **2.2.1 Basics**)

1) Conv2D
2) BatchNormalization
3) Activation
4) Again operations 1, 2 and 3
5) MaxPool2D



```python
def conv_block(inputs, filters, pool=True):
    x = Conv2D(filters, 3, padding="same")(inputs)
    x = BatchNormalization()(x)
    x = Activation("relu")(x)

    x = Conv2D(filters, 3, padding="same")(x)
    x = BatchNormalization()(x)
    x = Activation("relu")(x)

    if pool == True:
        p = MaxPool2D((2, 2))(x)
        return x, p
    else:
        return x
```

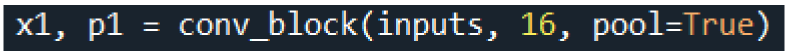
Code 7. Convolution block

As it can be observed in the conv_block function, not every call will be the same as some input parameters can be modified:

inputs: input from the previous layer. All operations mentioned before will be applied to this input.

filters: number of Conv2D to apply, this will determine output's depth.

pool: this last parameter controls whether the MaxPool2D operations is applied or not. It will default to True and will only be False during the expansive path as no pooling operations will be needed.



```python
x1, p1 = conv_block(inputs, 16, pool=True)
```

Code 8. First call of the convolution block of the 4 step U-Net model.

In code 8 the number of filters used is 16, this number will vary in each contracting step.

For the 4 step U-Net model, four calls to the conv_block function will be made with 16, 32, 64 and 128 as filters values. By doing this, the depth of the vector increases, provoking an augmentation of the "what" as more filters and pooling will be applied.

Expansive path:

Rounded in purple in image 6, we show the second part of the architecture and is also called decoder.

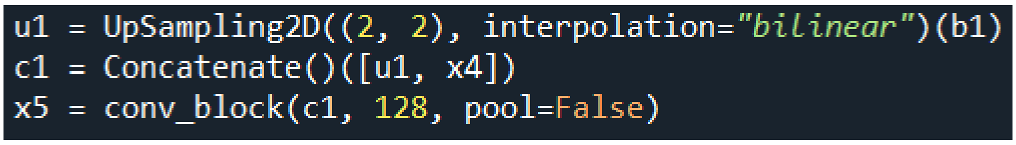
The objective of this expansive path is to create a high-resolution segmentation map providing the desired results for the experiment.

The number of expansion steps will depend on the model steps in the same way as the contracting path.

This expansion consists of a sequence of up-convolutions and concatenations with the corresponding high features of the contracting path.

(All operations can be consulted in **2.2.1 Basics**)

[1] UpSampling2D
[2] Concatenate
[3] Conv2D
[4] BatchNormalization
[5] Activation



```
u1 = UpSampling2D((2, 2), interpolation="bilinear")(b1)
c1 = Concatenate()([u1, x4])
x5 = conv_block(c1, 128, pool=False)
```

Code 9. First step of the expansive path of the 4 step U-Net model.

As it can be seen in code 9, conv_block function is called with its “pool” parameter to False so no pooling is applied.

Some variables of this last code image should be explained:

b1: corresponds to the bridge (explained afterwards) and it is the input for the first UpSampling2D as it is the output obtained after the contracting path is finished.

u1: output of the UpSampling2D that will serve as input for the Concatenate function.

x4: high feature map of the corresponding contracting step that will be concatenated to the result of the UpSampling2D operation so better results can be obtained. This concatenation will help the UpSampling2D method to better represent localization.

c1: output of the Concatenate function

x5: output of the conv_block function without pooling that will serve as input for the UpSampling2D function of the next expansion step.

The expansive path will end in the output layer (explained later).

Bridge:

This part of the architecture is in charge of “joining” both main parts of the model, the contracting and expansive path.

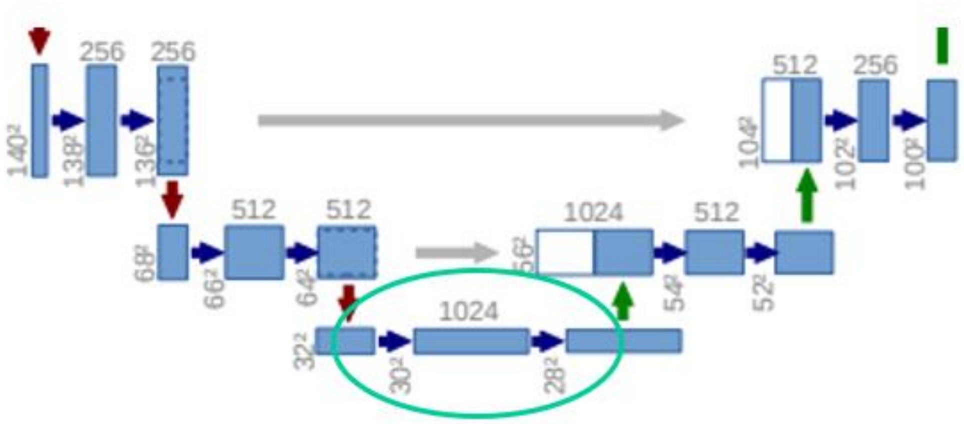
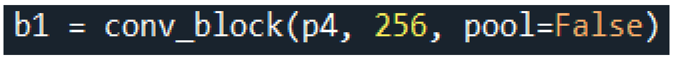


Image 7. Zoomed image showing the bridge of the U-Net

This bridge will consist of a last convolution block without pooling and it is the part of the U-Net where depth takes its maximum value.



```
b1 = conv_block(p4, 256, pool=False)
```

Code 10. Bridge.

Following depths of the contracting path, the number of filters will be 256 for the 4 step U-Net model (last convolution block of the contracting path had 128 filters).

Output layer:

The last step to obtain the results of the U-Net model where the high-resolution segmentation with all predictions is located.

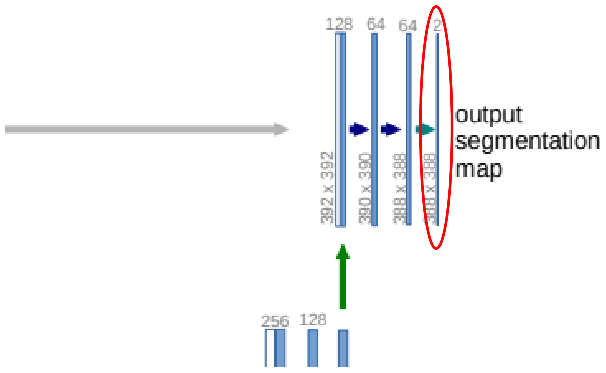
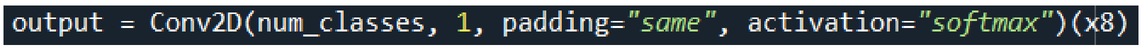


Image 8. Zoomed image showing the output layer of the U-Net

This layer's depth will be the number of output classes that the U-Net model is learning.



```
output = Conv2D(num_classes, 1, padding="same", activation="softmax")(x8)
```

Code 11. Output layer.

As it can be observed in Code 11, the output layer is just a call to the Conv2D function with some slight changes from the one of the conv_block:

num_classes: number of classes to predict, number of filters to apply.

1: kernel size.

padding: "same", equal than before.

activation: "softmax" is used as the prediction activation function which will obtain a weight for each class. With softmax, the sum of all final probabilities will be 1.

x8: input from the previous layer, it will correspond to the output from the conv_block of the last expansive step.

After having constructed the whole architecture, the model is ready to be trained with different parameters that will be explained later.

Once trained, as it is told before, it will obtain pixelwise classification for the whole image.

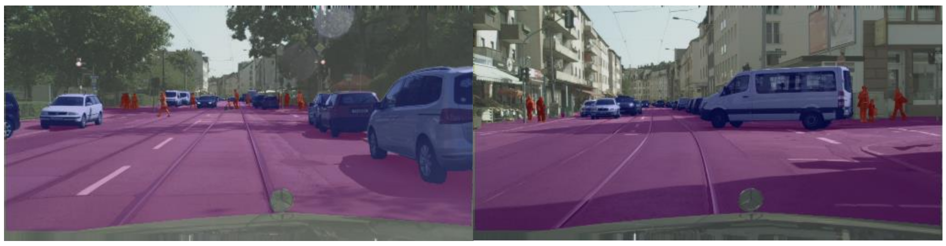


Image 9. Example of two training images with their corresponding masks overlapped (each color represents a different class)

## 2.3. Dataset

Cityscapes dataset has been used for the realization of this work. This dataset focuses on semantic understanding of urban street scenes [15].

It consists of images taken from a vehicle in 50 different cities during several months, daytimes and good weather conditions [15], as the application of this project concerns self-driving vehicles.

Each pixel of those 2048x1024 pixels images is classified as one of the 30 initial classes. These 30 initial classes are grouped into 8 different categories: void, flat, construction, object, nature, sky, human and vehicle.

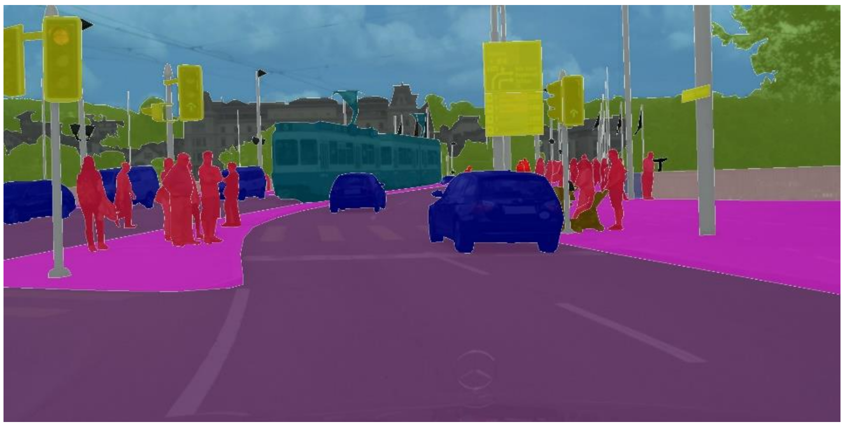


Image 10. Example of annotated image taken in Zurich (Overlayed colors correspond to the 30 semantic classes) [16]

The whole dataset consists of 5000 images divided into 3 segments:

Train: 2975 images

Val: 500 images

Test: 1525 images

All these images are labeled so different metrics can be applied to evaluate and compare different methods.

## 2.4. Hardware & Software

As previously discussed, applying different techniques in order to be able to use more accessible is one of the main aspects of this project.

The same hardware will be use along the experiments to make sure all obtained results are comparable to each other. In this case, not very powerful hardware has been used so better time results may be obtained with more powerful devices.

CPU: Intel Core i7-6700HQ

Total cores: 4

Total threads: 8

Max turbo frequency: 3.5 GHz

Processor base frequency: 2.6 GHz

GPU: NVIDIA GeForce GTX 950M

Cuda cores: 640

Memory size: 2 GB

Memory type: DDR3

RAM: 8192MB

Memory type: DDR3

Memory speed: 2133 MHz

Due to previous knowledge and all facilities provided by Python, it has been chosen as the coding language for the project. Every code file is a .py file, from the ones used for the setup of the dataset to the ones testing the models.

Spyder IDE has been used, but any other IDE can be used as long as the user feels comfortable with it. However, Spyder offers syntax highlighting, code completion and easy variable exploring (very useful when you are working with images, matrices, etc).

Keras (neural network library) and TensorFlow (open-source library for machine learning) have been used along Python to work with deep convolutional neural networks as they provide easy to use and comprehend methods and implementations.

# 3. Preprocessing & Reduction techniques

In this chapter, we will show which preprocessing and reduction techniques will be tested.

## 3.1. Color spaces

Not only the chosen model and its architecture is important for the results. Some other variables will vary along the experiments in order to improve classification. Color spaces is one of the other variables to consider. Just as color is very important to humans in terms of sight item recognition, it will also determine the convolutional neural network performance.

A color space is a specific organization of colors which is usually represented with X, Y and Z axis (3D), but more axis or less could be used. However, bigger ones will be able to hold more color combinations.

The use of one color space instead of another will be determined by the application as different results will be obtained with different ones. Colors “out of bounds” will not be captured accurately leading to worse results that will depend on how the chosen color space handles these values.

Taking into account the above considerations, several color spaces will be used in this project to compare the obtained results of using each one.

### 3.1.1. RGB

RGB (Red-Green-Blue) is one of the most popular color spaces. It is commonly used by most cameras, computer monitors and printers and it is easy to use as the color obtained is a mixture of the 3 values of the RGB scale.

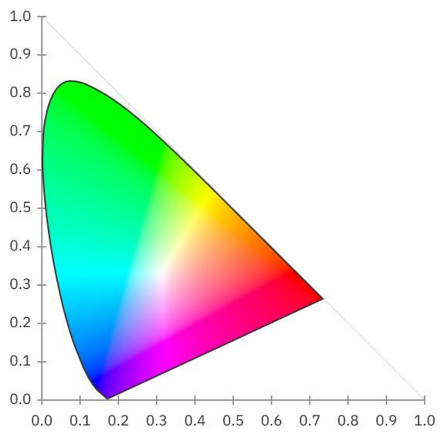


Image 11. RGB color space

### 3.1.2. HSV

HSV (Hue-Saturation-Value) or HSB (Hue-Saturation-Brightness) is another color space used in this project. It is widely used to generate high quality computer graphics and it has a different representation that RGB.

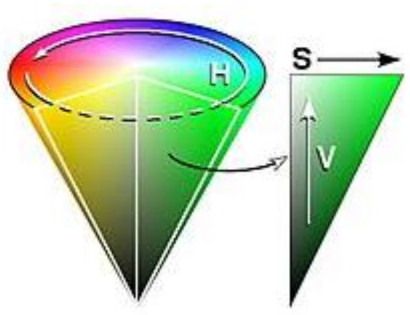
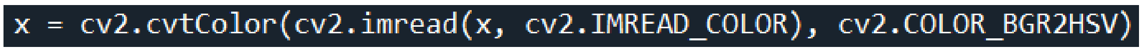


Image 12. HSV color space



```
x = cv2.cvtColor(cv2.imread(x, cv2.IMREAD_COLOR), cv2.COLOR_BGR2HSV)
```

Code 12. Image reading and HSV transformation

HSV main advantage over RGB is that it is usually more robust towards lightning changes. This is very useful when tracking colored objects in different light conditions.

### 3.1.3. Lab

The last color space used in this project is Lab, which owes its name to its 3 components: L, a and b that determine the output color. L stands for lightness, a for red-green value and b for blue-yellow value. It is used in spectrophotometers (devices that measure light intensity absorbed by a compound) or in hardware and software in charge of color measuring systems.

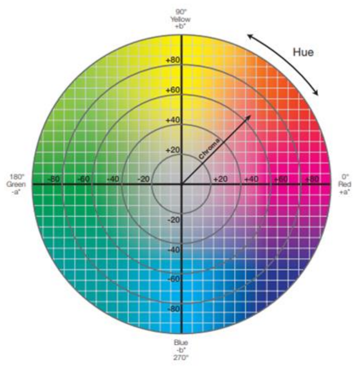


Image 13. HSV color space

```
x = cv2.cvtColor(cv2.imread(x, cv2.IMREAD_COLOR), cv2.COLOR_BGR2LAB)
```

Code 13. Image reading and Lab transformation

In theory, lab is more similar to the color perception of humans so it will be compared to compare if it obtains better results.

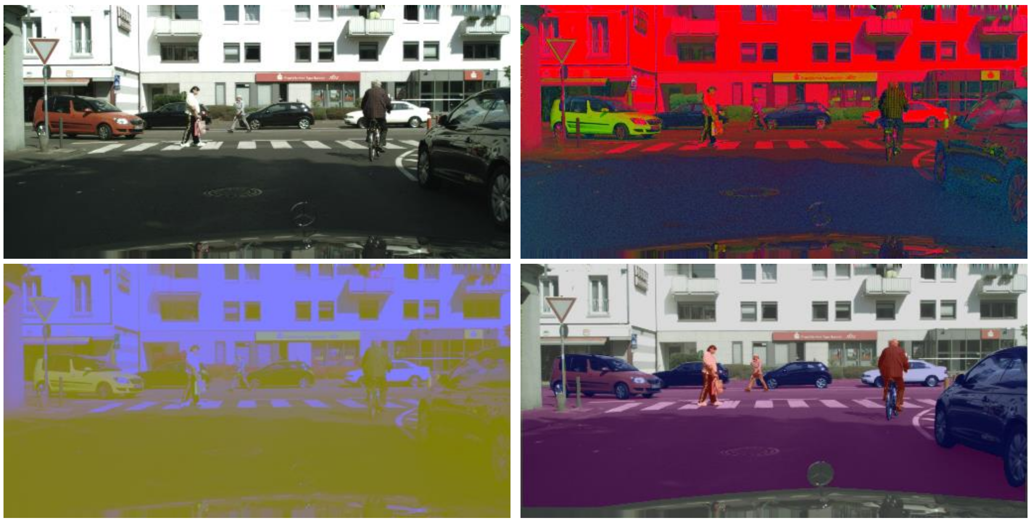


Image 14. Original dataset image in RGB (top left), image in HSV (top right), image in Lab (bottom left), RGB image masked with colors (bottom right)

As it can be observed, there are significant variations from one image to another. Model performance with each one will be evaluated afterwards.

## 3.2. Superpixels

Superpixels are the next variable that will be used to achieve the goal of this project.

Superpixels consists of a structure, not necessarily matrix, where regions with same color, texture or more characteristics are grouped into a same unity called superpíxel.

Image 15. Example of superpixel obtention in two different images [17]

Many superpixel obtention algorithms exist nowadays, but SLIC (Simple Linear Iterative Clustering), which is K-Means based, has been used in this project. SLIC labels each pixel into a superpixel based on their color similarity and proximity.

After having initialized each centroid formulas 1 and 2 are applied to classify each pixel and update each centroid.

$$D_i(p) = d_c + \frac{10}{S} d_s$$

$$d_c = \left( \sum_{j=1}^{3} \left( cc_i^j - cc_p^j \right)^2 \right)^{1/2}$$

$$d_s = \left( \left( x_i - x_p \right)^2 + \left( y_i - y_p \right)^2 \right)^{1/2}$$

Formula 1. Calculation of distance between pixel(p) and centroid($c_i$). $d_c$ corresponds to color distance and $d_s$ to spatial distance

$$c_i = \frac{1}{|C_i|} \sum_{z \in C_i} z$$

Formula 2. Centroid ($c_i$) update calculation by obtaining mean of every pixel ($C_i$) of that centroid

The aim of this method is to obtain more general shapes and simplify the image by deleting small changes allowing the U-Net model to generalize better.

However, superpixel algorithms tend to be slow and they may slow down the experiment. Anyway, performance of the U-Net model will be evaluated with this superpixel application.

## 3.3. Local Binary Pattern (LBP)

Different objects of the scene may have different textures. Textures on the road will differ a lot from those textures on a tree or a person as road tend to be flat while tree leaves have very different shapes.

Local Binary Pattern works with pixel windows to evaluate how different textures on that part of the image are.

In each window the neighborhood of the center pixel is evaluated. Pixels with a higher value than the center one will generate a 1 otherwise 0.

By doing this a binary number is generated. If the transformation of this binary number to decimal corresponds to one of the 58 uniform binary patterns, then that part of the image will be uniform.

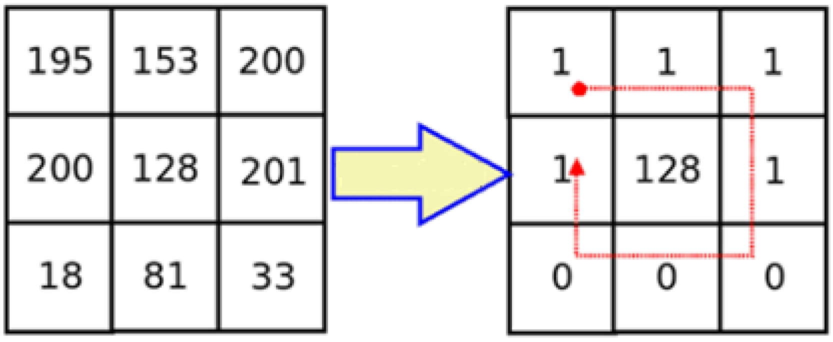


Image 16. Transformation of a windows to a binary number with Local Binary Pattern

As it can be observed in image 16, the decimal number obtained will be 1*128+1*64+1*32+1*16+0*8+0*4+0*2+1=241 that corresponds to one of the 58 uniform binary patterns. This was predictable as the top 6 pixels have much higher values than the 3 pixels below.

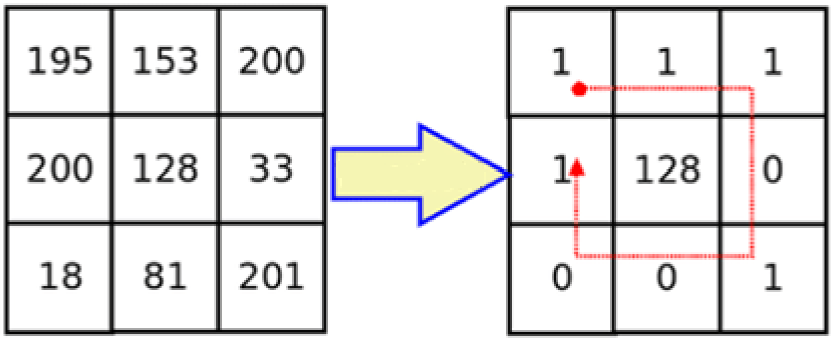


Image 17. Transformation of a windows to a binary number with Local Binary Pattern

In image 17 we obtain 1*128+1*64+1*32+0*16+1*8+0*4+0*2+1=233 that does not correspond to a uniform binary pattern. This also makes sense as the top left and the bottom right parts of the window have high values and are "separated" by 3 lower values: 33, 18 and 81.

For this experiment LBP is calculated for a fixed window size and it is concatenated as a fourth layer for RGB images providing more information than just color to the U-Net. This layer is normalized as RGB values are also normalized.

In this case, a specific parameter is used for the local_binary_pattern() function that sets the evaluation of LBP to default. This means that it is not rotation invariant which implies that the same image rotated will provide different results. This make sense as images are supposed to be always correctly oriented with floor below and sky on top.

## 3.4. Image reduction methods

As discussed before in **2.3 Dataset**, images size is 2048x1024, but some size reductions may be applied to increase speed.

Many image reduction methods exist, but two have been evaluated in this project. Each one will obtain different results and will have different image processing speed.

### 3.4.1. Sliding window (mean)

Just the name serves as the description of the method. The reduction is obtained with a window of a fixed size (4x4 in this case to reduce both axis by 4).

This window will move along the image without overlapping with the previous window. For each 4x4 matrix different operations can be made (mean, minimum, maximum, etc), but mean value has been chosen for this project.

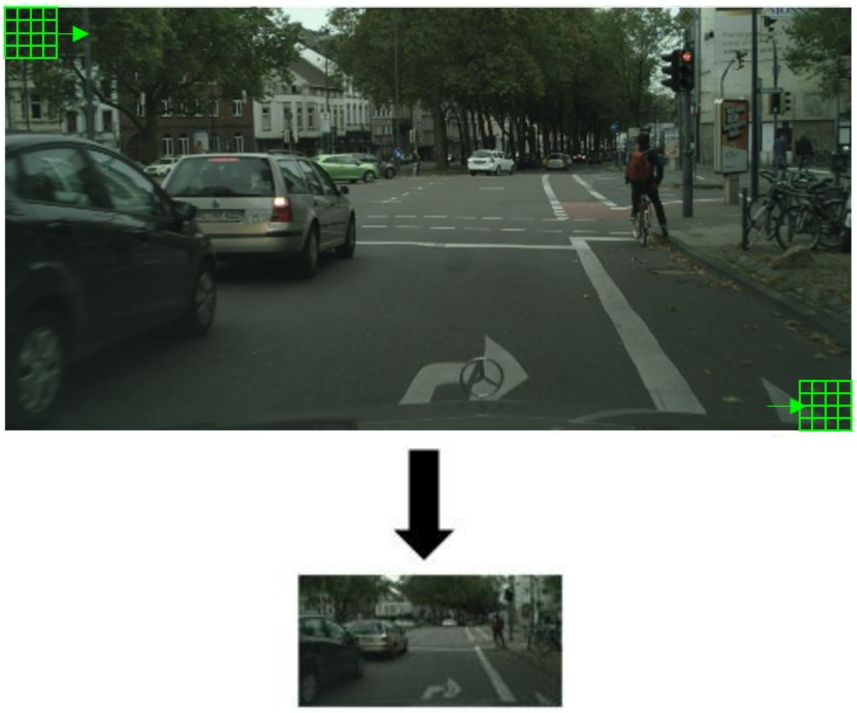


Image 18. Schematic image of how the sliding window algorithm works

### 3.4.2. Bilinear interpolation

This second method computes the weighted average of the nearest pixels.

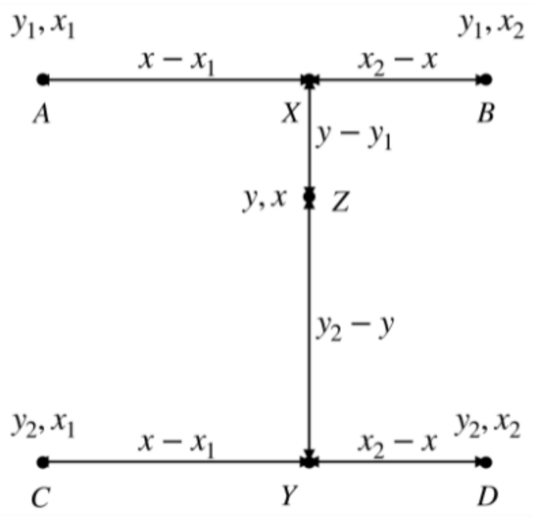


Image 19. Transformation of a windows to a binary number with Local Binary Pattern

In image 19, Z is the value to be calculated by breaking down the interpolation into linear resizing operations. To obtain it the next steps will be applied:

1) X = linear interpolation between A and B
2) Y = linear interpolation between C and D
3) Z = linear interpolation between X and Y

To apply this reduction, resize() function from OpenCV library.

Image 20. Original image from the dataset

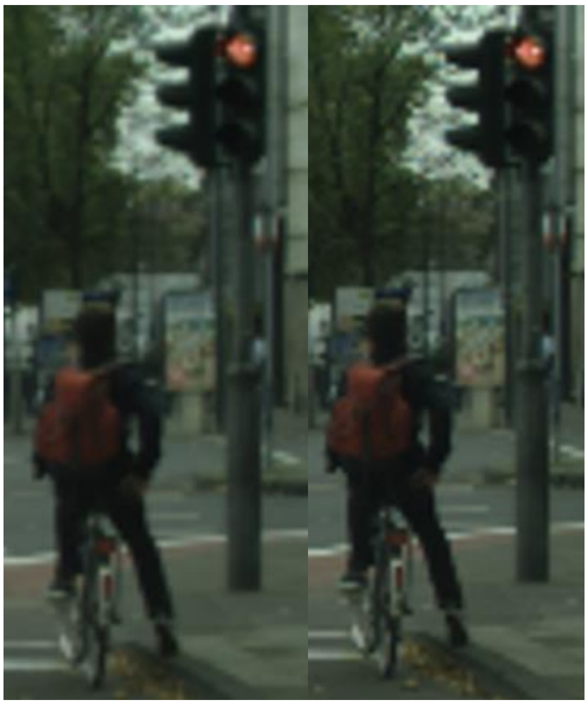


Image 21. Zoomed image reduced with mean sliding window (left) and with bilinear interpolation (right)

It is difficult to observe, but image reduced with mean sliding window seems to be blurrier than the one reduced with bilinear interpolation. Anyway, these two methods will be compared.

# 4. Experimental framework

This chapter will cover all previous concepts involving experimentation. After this explanation, experiments will take place so that a final analysis allows to decide the best model and preprocessing to apply.

## 4.1. Metrics

After having described the model to be used and the different variables that will change along the experiment, an evaluation method need to be defined.

As discussed before, finding a balance between accurate results and speed is necessary for this project in order to be able to embed these systems in different devices without the need of powerful and expensive hardware.

### 4.1.1. Accuracy

Accuracy is the most common metric and is used in almost all evaluations. It calculates how often pixel predictions equal their corresponding labels by counting how many times they are equal and dividing that result by the total of pixels.

### 4.1.2. Loss

Second metric that will be useful is loss. The loss function is used to compute the quantity that the model should seek to minimize during training [18].

In this case, cross entropy has been used, which is the most commonly used function for classification models. To be more specific, categorical cross entropy has been used as this is a multiclass classification model where the output label is a one-hot vector obtained from the argmax of the probabilities vector.

$$\text{Loss} = -\sum_{i=1}^{\substack{\text{output}\\\text{size}}} y_i \cdot \log \hat{y}_i$$

Formula 3. Categorical cross entropy

As it can be observed in formula 3, loss value will depend on the true label of a prediction and the prediction for that class. In the following lines two examples will show how this work:

Example 1:

True label: [1 0 0 0] (one-hot vector)

Model predictions: [0.1 0.5 0.2 0.3]

Loss = -1*log(0.1) – 0*log(0.5) – 0*log(0.2) – 0*log(0.3) = 2.303

Example 2:

True label: [1 0 0 0] (same vector as in example 1)

Model predictions: [0.8 0.1 0.05 0.05]

Loss = -1*log(0.8) – 0*log(0.1) – 0*log(0.05) – 0*log(0.05) = 0.223

As a conclusion from examples 1 and 2 we can see that a higher loss value is obtained if the prediction for the true class is lower. Therefore, a higher correction will take place whether a higher loss is obtained, otherwise a smaller correction will be applied.

### 4.1.3. Mean Intersection Over Union (Mean IOU)

Mean Intersection Over Union is one of the most significant evaluation methods of an image segmentation process. It first computes the IOU for each semantic class and then computes the average over classes [19].

IOU is defined as: true_positive / (true_positive + false_positive + false_negative)

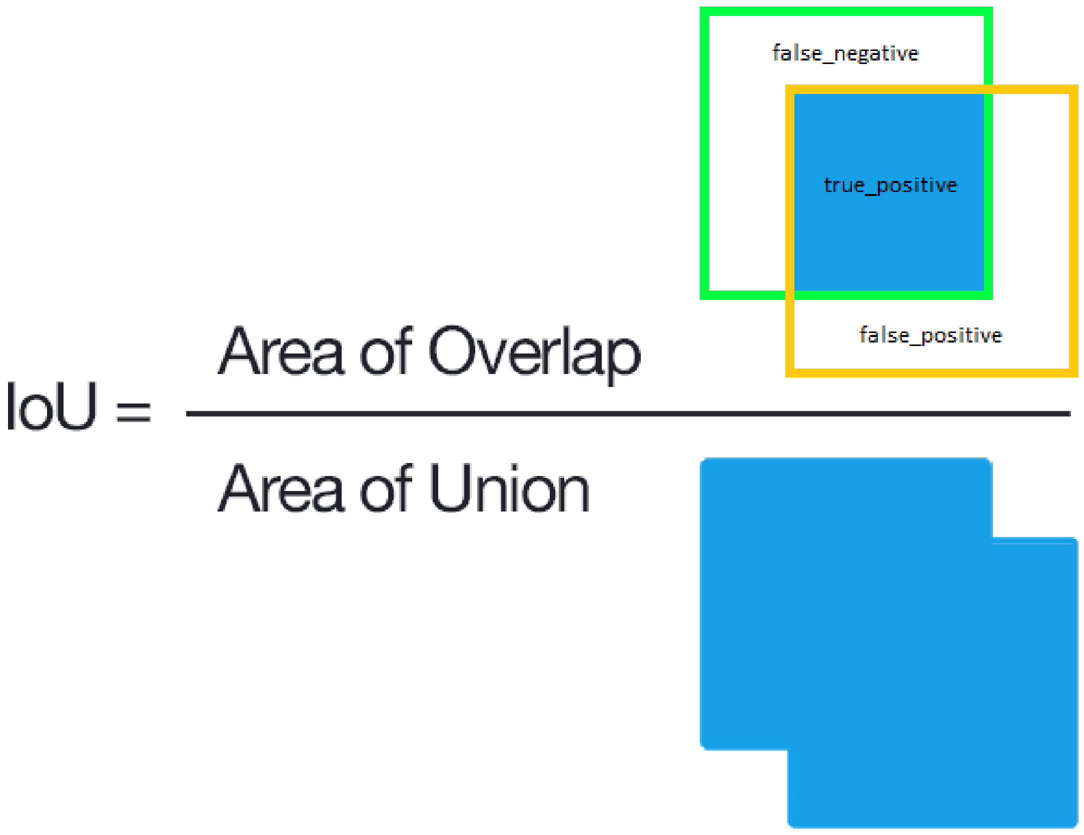


Image 22. Schema explaining IOU formula

An IOU score > 0.5 is normally considered as a “good” prediction.

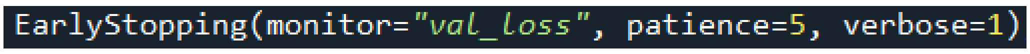
### 4.1.4. Training time

Training time should also be considered as it will probably reflect how heavy and slow some processes can be, but a high training time does not always imply slow predictions.

It will highly depend on what is applied to each image (superpixels, LBP, etc) and which architecture is being used.

Some model parameters will also be set:

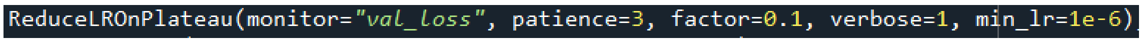
Epochs: number of times the model goes through the training set. It will be set to 20, but will stop if validation loss does not improve in several epochs in a row (patience parameter).



```
EarlyStopping(monitor="val_loss", patience=5, verbose=1)
```

Code 14. tensorflow.keras method that allows learning rate modification between epochs

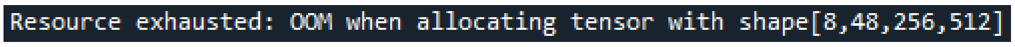
Learning rate: it controls how fast the model adapts to the problem. It will start in 1x10$^{-4}$ and will be reduced (factor parameter) if validation loss does not improve between several consecutive epochs (patience parameter).



```
ReduceLROnPlateau(monitor="val_loss", patience=3, factor=0.1, verbose=1, min_lr=1e-6)
```

Code 15. tensorflow.keras method that allows learning rate modification between epochs

Batch size: it defines the number of samples that will be propagated through the network. It will be set to 4 as higher values such as 8 are not supported with this hardware (an out of memory error is produced with 8 as batch size).



```
Resource exhausted: OOM when allocating tensor with shape[8,48,256,512]
```

Image 23. Error message when trying to train with 8 as batch size

### 4.1.5. Testing time

Testing time is the last metric that will serve as evaluation. It is one of the most important as autonomous vehicles need very fast image processing methods to make better decisions.

It will be measured after the model is trained.

Even though some slower methods may be more accurate than faster ones, a balance between speed and accuracy is needed so the faster method could be chosen before the most accurate one.

In this case, testing time will be the total time to read and predict 150 images (50 from train, 50 from val and 50 from test).

## 4.2. Number of classes

As it has been already mentioned in **2.3 Dataset**, segmented images of the original dataset are originally divided into 30 classes grouped in 8 categories.

However, having 8 principal classes has been considered unreasonable for this project.

The main application of the model is to segment images taken from a vehicle so some classes will be more important than others. Therefore, some different combinations have been evaluated:

8 classes:

Void, flat, construction, object, nature, sky, human and vehicle.

This is the original division of classes, but some of them are not strictly necessary for the segmentation and joining them into the same group will provide clearer segmentations and the model will learn better.

3 classes:

Flat, vehicle, rest of classes.

This is the simplest consideration of groups, but it is too limited as it does not predict one very important class: human.

4 classes:

Flat, vehicle, human, rest of classes.

This is a very balanced split as considers flat (road, parking, etc), vehicles (car, motorbike, bicycle, etc), human (person, rider) as separate classes from the rest.

This allow the vehicle to identify the path to follow and what to avoid/dodge, even if these decisions are not covered in this project.

5 classes:

Flat, vehicle, human, object, rest of classes.

This last division is also a very valid one as it adds one more class: object (traffic light, traffic sign, etc). However, even though it is an important group to add, it will not be used as traffic signs information will only be used for decision making systems.

Taking into account all these considerations, the final number of classes for this experiment will be 4.

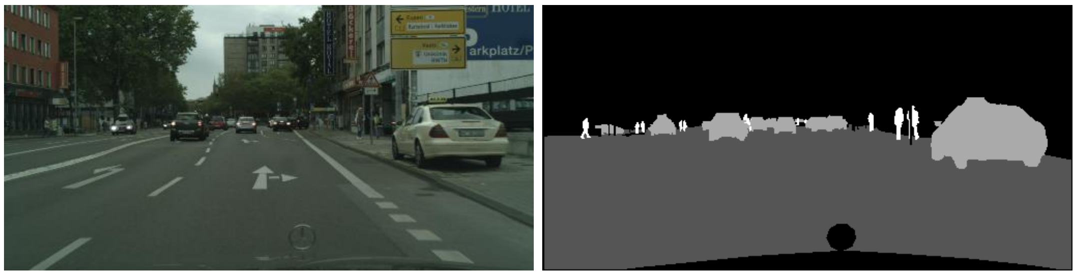
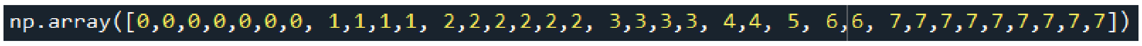


Image 24. Original image in RGB (left) and its corresponding 4 class mask (right)

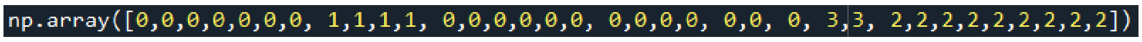
As it can be seen in image 24, the mask has only 4 different colors, each of them associated to a specific class: flat (dark grey), vehicle (light grey), human (white) and rest of classes (black).

To obtain these new masks, different from the original ones, some simple python code has been executed with just a different mask.



```
np.array([0,0,0,0,0,0,0, 1,1,1,1, 2,2,2,2,2,2, 3,3,3,3, 4,4, 5, 6,6, 7,7,7,7,7,7,7,7,7])
```

Code 16. 8 class mask



```
np.array([0,0,0,0,0,0,0, 1,1,1,1, 0,0,0,0,0,0, 0,0,0,0, 0,0, 0, 3,3, 2,2,2,2,2,2,2,2,2])
```

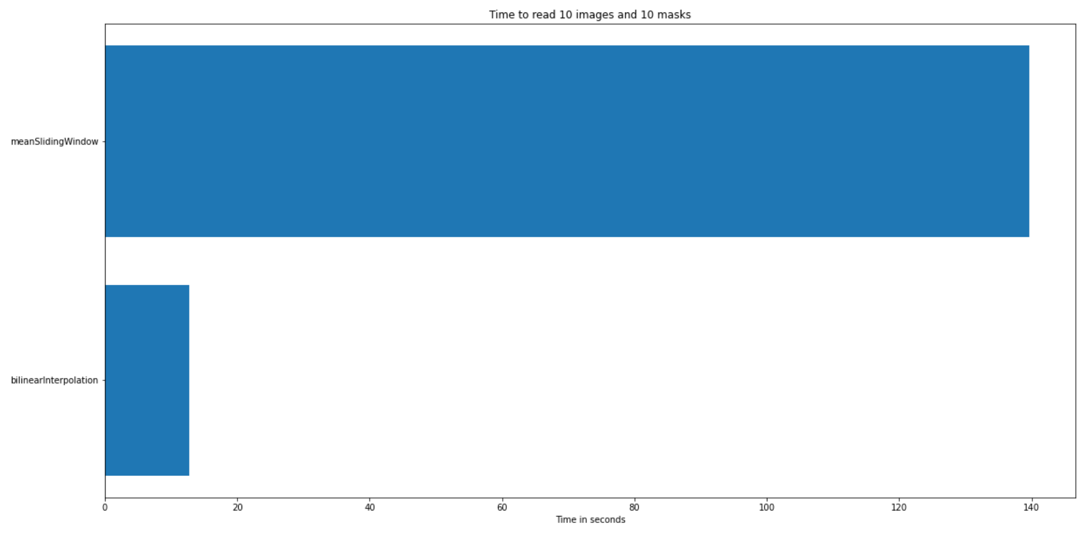
Code 17. 4 class mask

As it is shown in codes 16 and 17, some classes are merged into the same one reducing the total number of groups.

## 4.3. Image reduction method

Reduction could be applied if needed or if speed wants to be increased. Both methods mentioned in **3.4 Image reduction methods** will be tested to decide which is better.

To make this comparison 10 images from the train dataset and their corresponding masks have been read. All 10 images have been resized with sliding window (mean) and bilinear interpolation to compare resizing times.



Graph 1. Reading + resizing time for both image reduction methods (time in seconds)

Time to read and resize 10 images and masks:

Sliding window (mean): ~139.643 s (13.964 s/image&mask)

Bilinear interpolation: ~12.788 s (1.279 s/image&mask)

As it can be observed, sliding window (mean) method is much slower than bilinear interpolation (more than 10 times slower).

Considering that this project aims to obtain a balance between precision and time, sliding window (mean) is too slow. Therefore, bilinear interpolation will be used as image reduction method in case any reduction is necessary (this will be discussed in **5.1 Image size**).

# 5. Experiments & Analysis

After having made all previous considerations, experiments will be made to finally obtain the trained model.

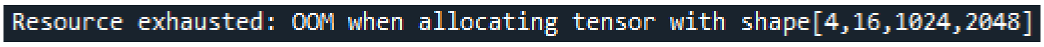
All experiments below will be made with both U-Net architectures (3 and 4 steps).

## 5.1. Image size

First training experiment will be image size as it might be fixed along the experiments.

<u>2048x1024:</u>

This is the original size. At first, model training with this image size was tried, but hardware used in this experiment runs outs of memory for such big images.



```
Resource exhausted: OOM when allocating tensor with shape[4,16,1024,2048]
```

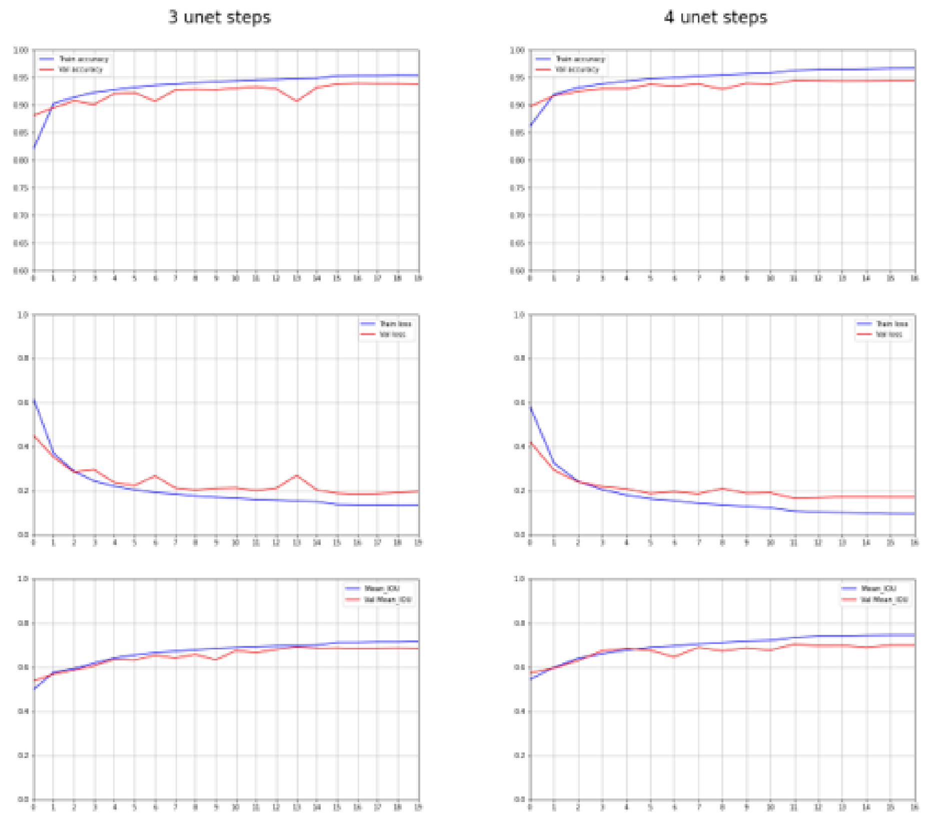
Image 25. Error message when trying to train with original size images

<u>512x256:</u>

This is the only alternative size considered for the experiments as it is previously mentioned in **3.4.1 Sliding window (mean)** (height and width reduced by 4 which means 2048/4 = 512 and 1024/4 = 256). It maintains details in the image and enables model training.

It will be the resolution to use during the project as it allows reasonable times considering the limited hardware and the method applied to obtain it will be bilinear interpolation as previously discussed in **4.1.7 Image reduction method.**

## 5.2. RGB model

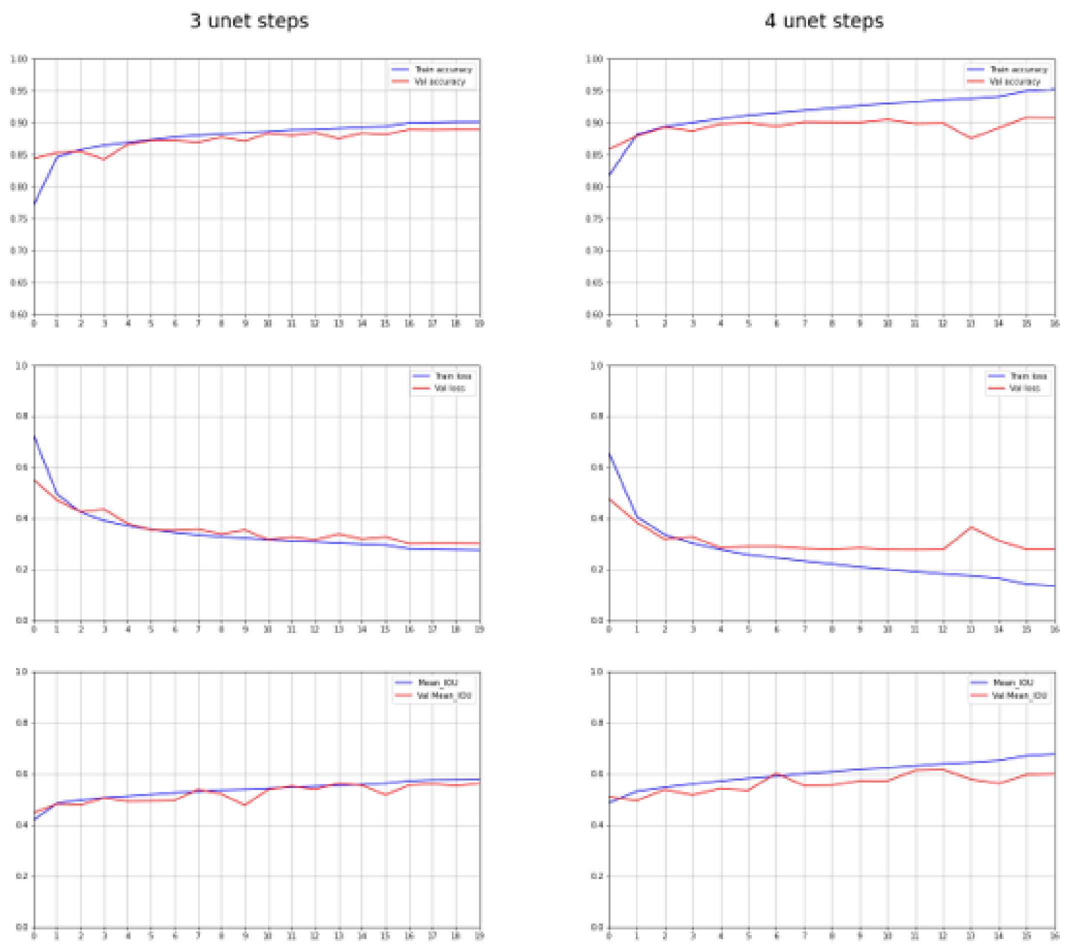


Graph 2. Accuracy, loss and mean IOU for training and validation (RGB model)

| | 3 steps | 4 steps |
|---|---|---|
| **Accuracy (train)** | 0.954 | 0.966 |
| **Accuracy (val)** | 0.939 | 0.945 |
| **Loss (train)** | 0.133 | 0.095 |
| **Loss (val)** | 0.184 | 0.166 |
| **Mean IOU (train)** | 0.716 | 0.745 |
| **Mean IOU (val)** | 0.691 | 0.704 |
| **Training time (s)** | 8890.957 | 8232.059 |
| **Testing time (s)** | 17.274 (0.115 s/image) | 19.157 (0.128 s/image) |

Table 1. Statistics for RGB model

## 5.3. HSV model

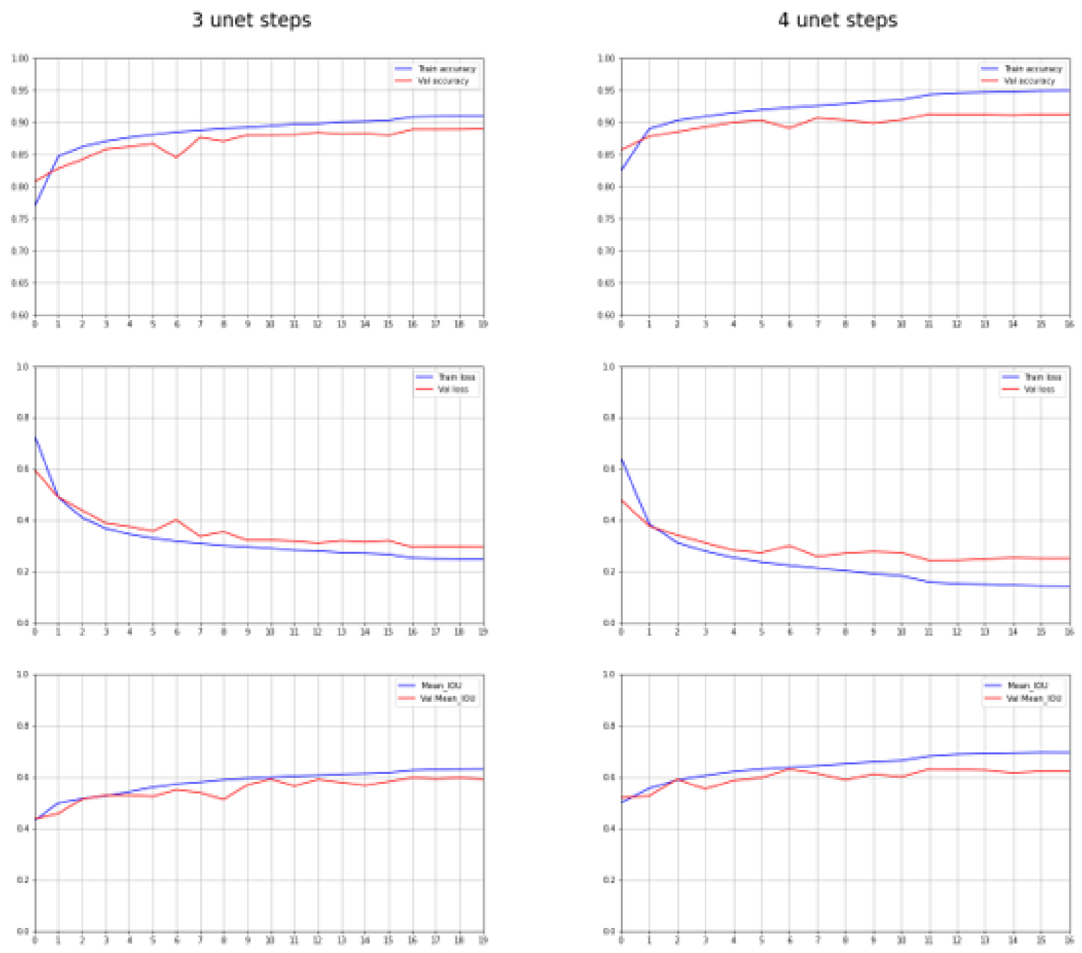


Graph 3. Accuracy, loss and mean IOU for training and validation (HSV model)

| | 3 steps | 4 steps |
|---|---|---|
| **Accuracy (train)** | 0.901 | 0.952 |
| **Accuracy (val)** | 0.889 | 0.908 |
| **Loss (train)** | 0.276 | 0.135 |
| **Loss (val)** | 0.302 | 0.278 |
| **Mean IOU (train)** | 0.578 | 0.677 |
| **Mean IOU (val)** | 0.563 | 0.616 |
| **Training time (s)** | 8863.644 | 8267.847 |
| **Testing time (s)** | 17.621 (0.117 s/image) | 19.469 (0.130 s/image) |

Table 2. Statistics for HSV model

## 5.4. Lab model

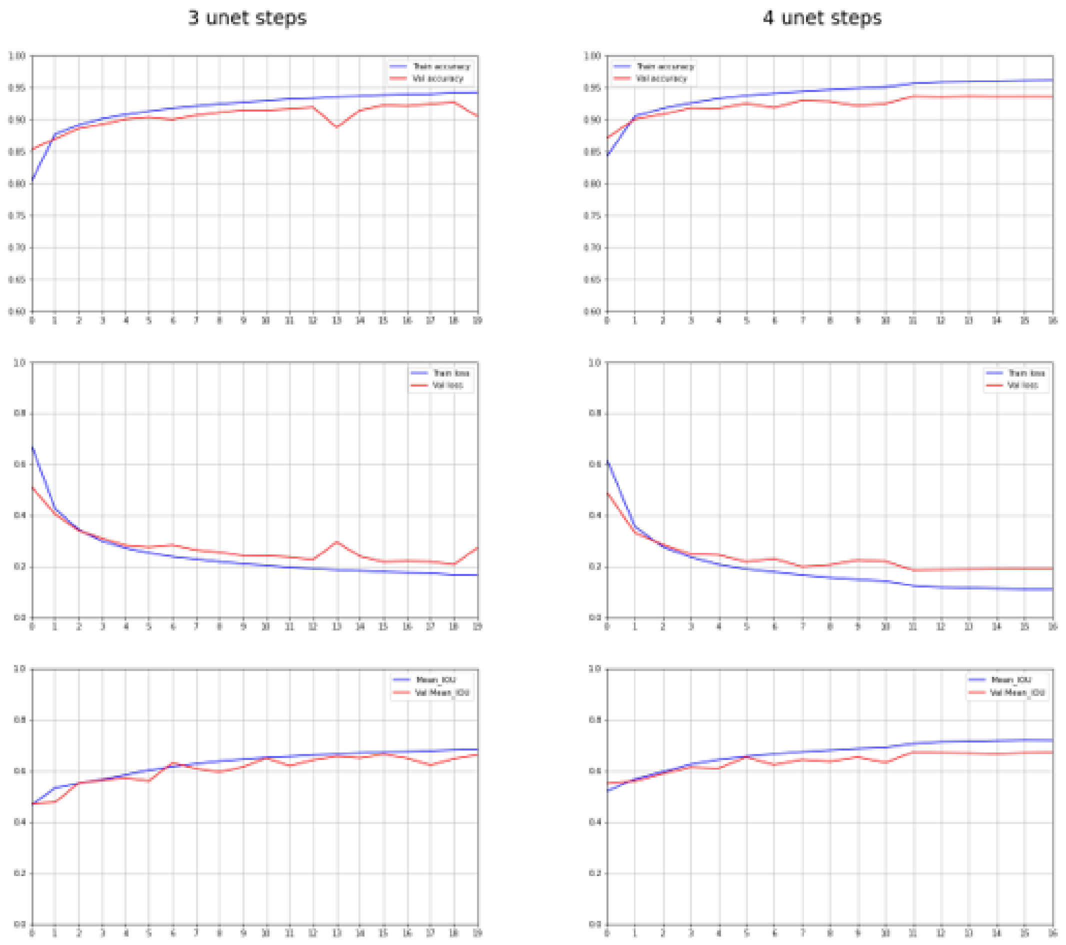


Graph 4. Accuracy, loss and mean IOU for training and validation (Lab model)

| | 3 steps | 4 steps |
|---|---|---|
| **Accuracy (train)** | 0.910 | 0.949 |
| **Accuracy (val)** | 0.890 | 0.913 |
| **Loss (train)** | 0.248 | 0.143 |
| **Loss (val)** | 0.296 | 0.243 |
| **Mean IOU (train)** | 0.631 | 0.695 |
| **Mean IOU (val)** | 0.598 | 0.630 |
| **Training time (s)** | 8832.470 | 8290.151 |
| **Testing time (s)** | 18.694 (0.125 s/image) | 19.516 (0.130 s/image) |

Table 3. Statistics for Lab model

## 5.5. RGB + HSV + Lab model

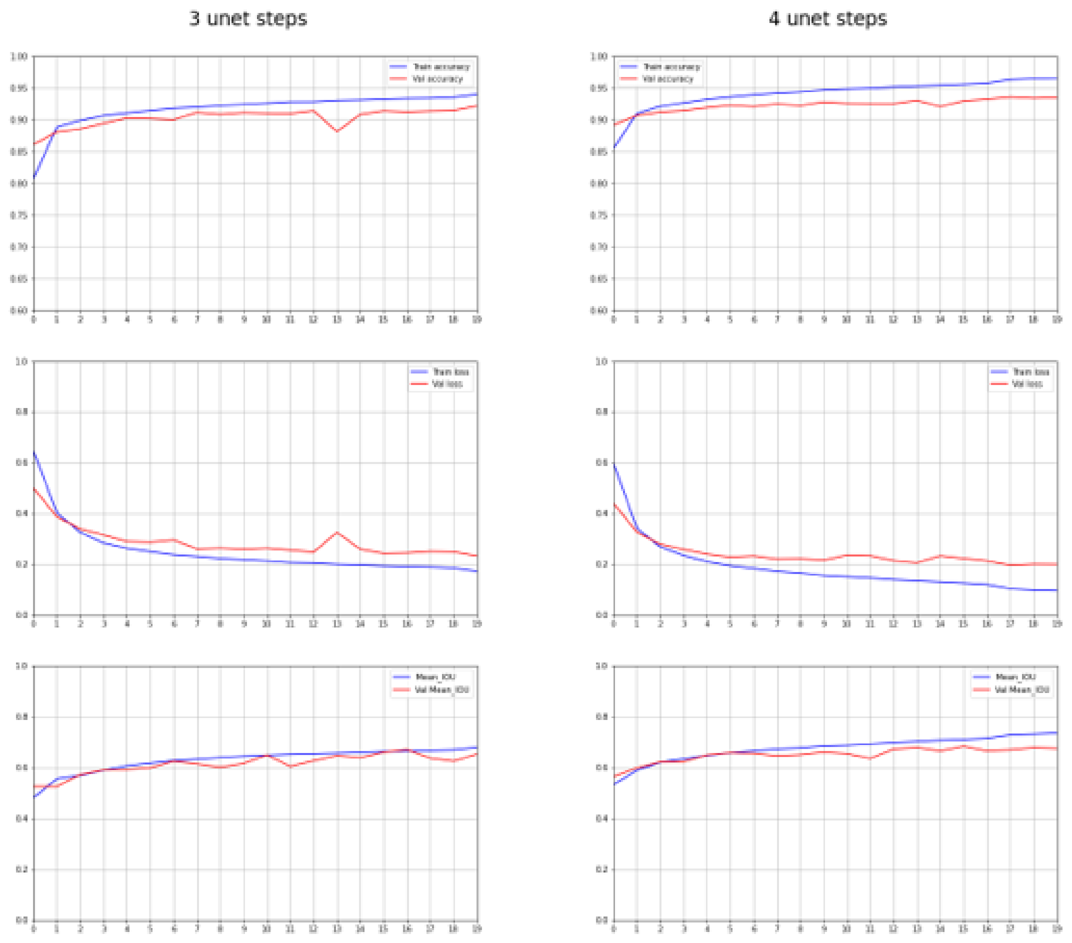


Graph 5. Accuracy, loss and mean IOU for training and validation (RGB + HSV + Lab model)

| | 3 steps | 4 steps |
|---|---|---|
| **Accuracy (train)** | 0.943 | 0.961 |
| **Accuracy (val)** | 0.927 | 0.936 |
| **Loss (train)** | 0.165 | 0.110 |
| **Loss (val)** | 0.208 | 0.186 |
| **Mean IOU (train)** | 0.686 | 0.720 |
| **Mean IOU (val)** | 0.664 | 0.674 |
| **Training time (s)** | 9076.352 | 8553.729 |
| **Testing time (s)** | 22.610 (0.151 s/image) | 23.524 (0.157 s/image) |

Table 4. Statistics for RGB + HSV + Lab model

## 5.6. RGB + Superpixels model

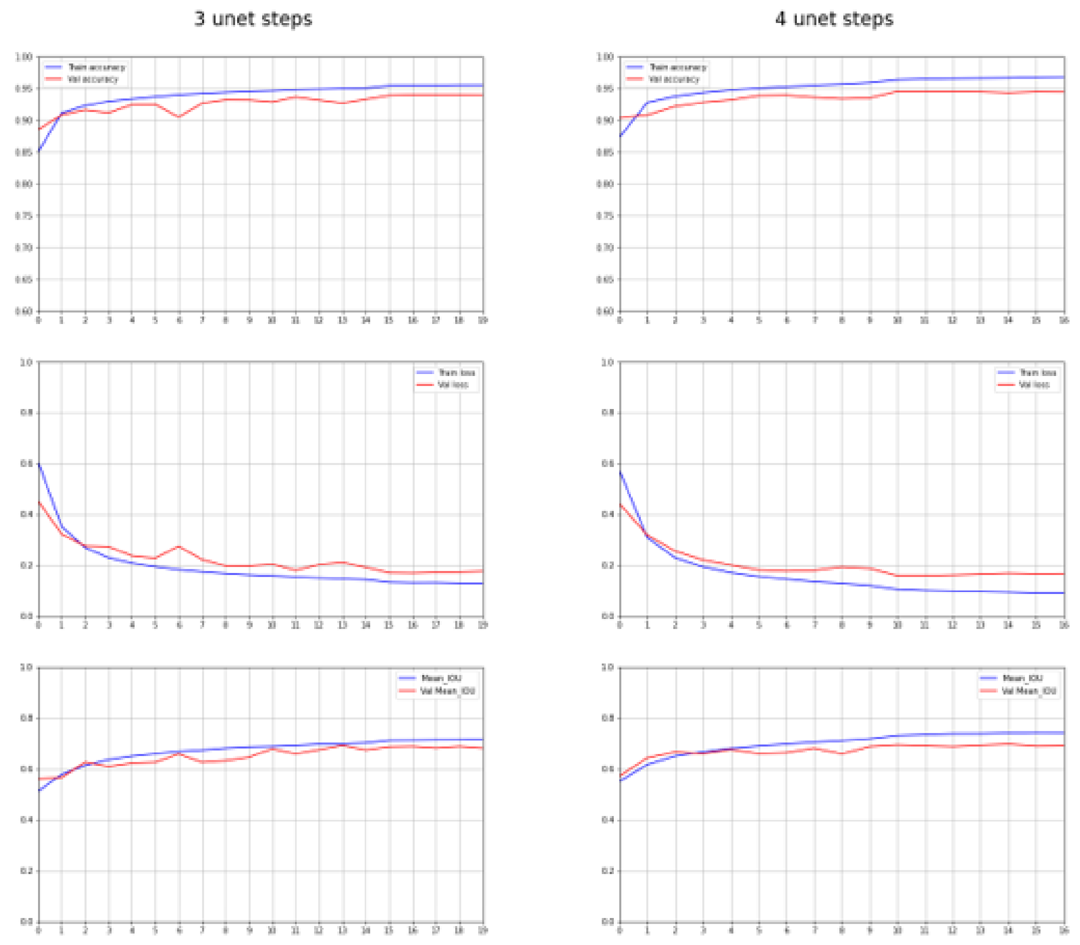


Graph 6. Accuracy, loss and mean IOU for training and validation (RGB + Superpixels model)

| | 3 steps | 4 steps |
|---|---|---|
| **Accuracy (train)** | 0.940 | 0.965 |
| **Accuracy (val)** | 0.921 | 0.936 |
| **Loss (train)** | 0.173 | 0.096 |
| **Loss (val)** | 0.233 | 0.197 |
| **Mean IOU (train)** | 0.679 | 0.735 |
| **Mean IOU (val)** | 0.670 | 0.683 |
| **Training time (s)** | 34962.019 | 35639.160 |
| **Testing time (s)** | 98.266 (0.655 s/image) | 98.294 (0.655 s/image) |

Table 5. Statistics for RGB + Superpixels model

## 5.7. RGB + Local Binary Pattern model



Graph 7. Accuracy, loss and mean IOU for training and validation (RGB + LBP model)

| | 3 steps | 4 steps |
|---|---|---|
| **Accuracy (train)** | 0.955 | 0.968 |
| **Accuracy (val)** | 0.940 | 0.945 |
| **Loss (train)** | 0.130 | 0.091 |
| **Loss (val)** | 0.170 | 0.160 |
| **Mean IOU (train)** | 0.717 | 0.743 |
| **Mean IOU (val)** | 0.692 | 0.697 |
| **Training time (s)** | 9626.007 | 8888.769 |
| **Testing time (s)** | 32.705 (0.218 s/image) | 33.643 (0.224 s/image) |

Table 6. Statistics for RGB + LBP model

## 5.8. Statistics

Some statistics will be calculated to help final conclusions.

| | | **3 steps** | **4 steps** |
|---|---|---|---|
| Accuracy (val) | Mean | 0.918 | 0.931 |
| | Median | 0.924 | 0.936 |
| Loss (val) | Mean | 0.232 | 0.205 |
| | Median | 0.221 | 0.192 |
| Mean IOU (val) | Mean | 0.646 | 0.667 |
| | Median | 0.667 | 0.679 |
| Training time | Mean | 13375.242 | 12978.619 |
| | Median | 8983.655 | 8421.940 |
| Testing time | Mean | 34.528 (0.230 s/image) | 35.601 (0.237 s/image) |
| | Median | 20.652 (0.138 s/image) | 21.52 (0.143 s/image) |

Table 7. Mean and median for validation metrics and time

| | | **3 steps** | **4 steps** |
|---|---|---|---|
| Accuracy (val) | Min | 0.889 | 0.908 |
| | Max | 0.940 | 0.945 |
| Loss (val) | Min | 0.170 | 0.160 |
| | Max | 0.302 | 0.278 |
| Mean IOU (val) | Min | 0.563 | 0.616 |
| | Max | 0.692 | 0.704 |
| Training time | Min | 8832.470 | 8232.059 |
| | Max | 34962.019 | 35639.160 |
| Testing time | Min | 17.724 (0.118 s/image) | 19.157 (0.128 s/image) |
| | Max | 98.266 (0.655 s/image) | 98.294 (0.655 s/image) |

Table 8. Min and max for validation metrics and time

## 5.9. Result analysis

Now that all models have been trained and statistics have been calculated is time to deliberate which is the best model.

First, observing both tables we can see that there is at least one experiment much slower than the rest. This can be checked by looking table 7 (training and testing time median value is much lower than the mean) and table 8 (training and testing time both have a very high difference between their corresponding maximum and minimum value)

After this consideration, looking at **5. Experiments**, we can see that this slower experiment is the one concerning the use of superpixels (comparing its testing time to the RGB model we can see that it is more than 5 times slower). This makes sense as superpixel obtention is a slow process. Therefore, **5.6 RGB + Superpixels model** is discarded.

Now there are 5 models remaining. As previously discussed, mean IOU is more representative than accuracy in image segmentation to evaluate how well is the model performing.

If we observe table 8, there is a high difference between min and max for mean IOU so there are some models much better than others. After checking again results obtained in **5. Experiments**, HSV and Lab models will be discarded due to their lower IOU compared to the rest.

After this last decision, models left are: RGB, RGB + HSV + Lab and RGB + LBP.

Once again, testing time will discard a model. RGB and RGB + HSV + Lab models testing times for 150 images are between 17.274 s (RGB with 3 U-Net steps) and 23.524 s (RGB + HSV + Lab with 4 steps) while RGB + LBP testings times are 32.705 s (3 steps) and 33.643 s (4 steps). So RGB + LBP model is discarded.

The final factor that will decide which model is better for this project will be mean IOU. While mean IOU for the RGB + HSV + Lab model is 0.664 (3 step model) or 0.674 (4 steps model), the RGB model reaches a mean IOU of 0.691 (3 steps model) and 0.704 (4 steps model). In addition, testing time is also better for the RGB model.

There is still one more decision to be made and it is to choose if a 3 steps U-Net architecture is better or worse than 4 steps one. Next table will show the final comparisons.

| | 3 steps | 4 steps |
|---|---|---|
| **Mean IOU (val)** | 0.691 | 0.704 |
| **Testing time (s)** | 17.724 (0.115 s/image) | 19.157 (0.128 s/image) |

Table 9. Mean IOU and testing time of RGB model

As it can be observed in table 9, mean IOU of the 4 steps U-Net RGB model is better than the 3 steps one (1.88% better). However, single image testing time of the 4 steps model is worse than the 3 step one (11.30% slower).

Taking into account these 2 last comparisons, the 3 steps U-Net model is considered to be better as it offers results just 1.88% worse in terms of IOU but is 11.30% faster which is very important for self-driving vehicles (main application of this study where fast decisions are necessary).

| | **3 steps** |
|---|---|
| **Accuracy (train)** | 0.954 |
| **Accuracy (val)** | 0.939 |
| **Loss (train)** | 0.133 |
| **Loss (val)** | 0.184 |
| **Mean IOU (train)** | 0.716 |
| **Mean IOU (val)** | 0.691 |
| **Training time (s)** | 8890.957 |
| **Testing time (s)** | 17.274 (0.115 s/image) |

Table 10. Metrics of the chosen model: RGB 3 steps U-Net

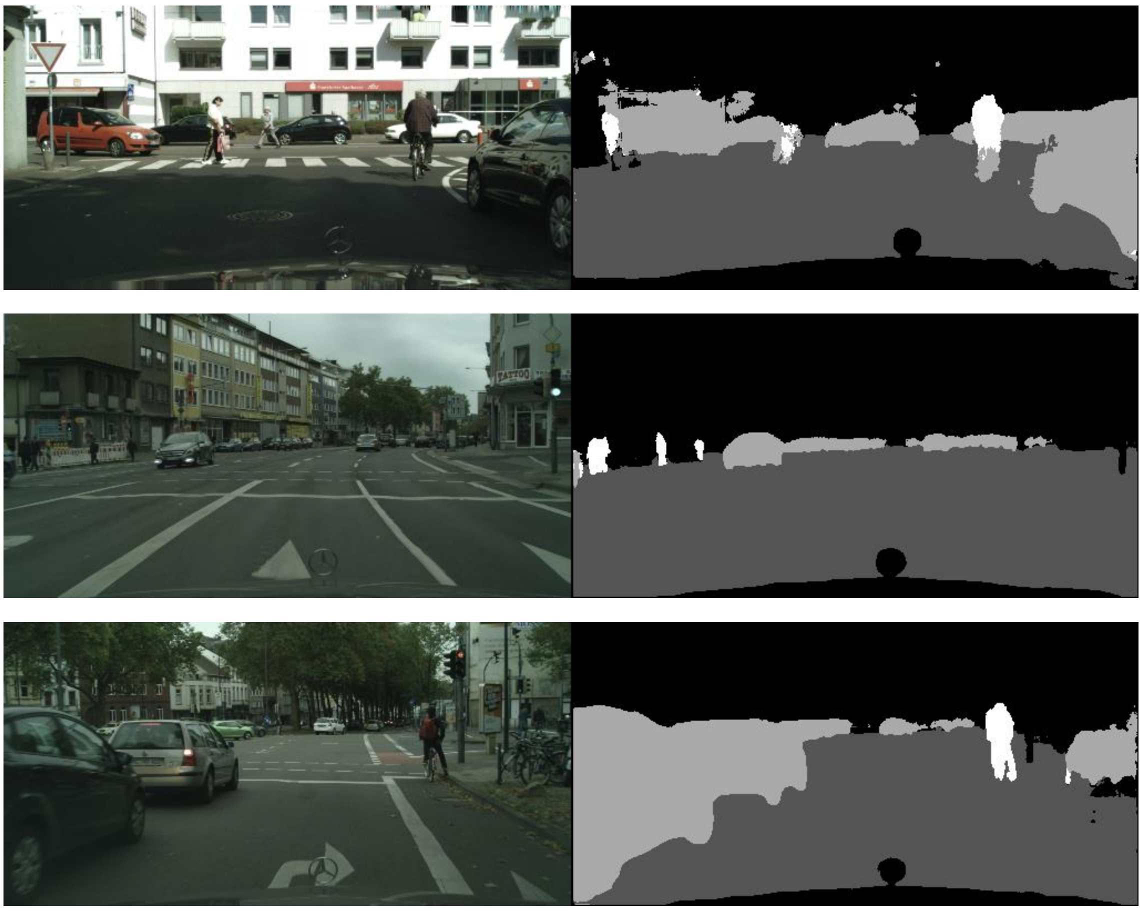


Image 26. Examples of images (left) and the predicted mask obtained by the 3 steps RGB model

# 6. Conclusions and future lines

In this project we have developed an in-depth study on different processes that allow to reduce the computational cost on deep convolutional neural network, U-Net in this case.

According to the results, not always more complex methods obtain the best outcome for every application. Even though they can obtain more accurate predictions, in some cases, like self-driving vehicles where speed is key, simpler and faster methods are better.

Taking into account this consideration, we have chosen bilinear interpolation as image reduction method as it is much faster than sliding window. We have also seen that 3 steps U-Net architectures are faster than 4 steps, but they obtain less accurate results. Anyway, speed gain is much more considerable than IOU loss (11.3% speed gain vs 1.88% mean IOU loss in RGB 3 steps U-Net).

In the future, we expect to continue analyzing more preprocessing methods that allow to maintain or increase speed without losing or even improving accuracy (mean IOU). Some of the new methods to apply could be color correction processes such as gamma correction or other image reduction techniques.

Another aspect to consider would be postprocessing. As it can be observed in the last image, some pixel predictions are far to be correct. This could be fixed by applying postprocessing methods always keeping in mind that speed is key.

One last consideration would be using better hardware. This will allow more testing and more comparisons, but remembering the importance of accessible hardware.

# 7. Bibliography and references

[1] IBM, "Deep Learning", 2020
https://www.ibm.com/cloud/learn/deep-learning

[2] Toyota, "Ventajas y desventajas de los coches autónomos"
https://www.toyota.es/world-of-toyota/articles-news-events/ventajas-desventajas-coches-autonomos

[3] Cityscapes dataset, Benchmarks Suite
https://www.cityscapes-dataset.com/benchmarks/

[4] Farnoush Zohourian, Borislav Antic, Jan Siegemund, Mirko Meuter and Josef Pauli, "Superpixel-based Road Segmentation for Real-time Systems using CNN"
https://www.is.uni-due.de/fileadmin/literatur/publikation/zohourian18visapp.pdf

[5] Yann LeCun, Léon Bottou, Yoshua Bengio and Patrick Haffner, "Gradient-Based Learning Applied to Document Recognition", 1998, page 7
http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf

[6] Olaf Ronneberger, Philipp Fischer and Thomas Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation", 2015, page 2
https://arxiv.org/pdf/1505.04597.pdf

[7] Tensorflow, Conv2D
https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv2D

[8] Antónia Vojteková, "Neural network noise reduction of astronomical images", 2019/2020, page 14
https://is.muni.cz/th/urvgk/Diploma_thesis__37_.pdf

[9] Tensorflow, BatchNormalization
https://www.tensorflow.org/api_docs/python/tf/keras/layers/BatchNormalization

[10] Tensorflow, Activation
https://www.tensorflow.org/api_docs/python/tf/keras/layers/Activation

[11] Tensorflow, MaxPool2D

https://www.tensorflow.org/api_docs/python/tf/keras/layers/MaxPool2D

[12] Tensorflow, UpSampling2D
https://www.tensorflow.org/api_docs/python/tf/keras/layers/UpSampling2D

[13] Technical Fridays, “Autoencoder: Downsampling and Upsampling”, 2019
https://kharshit.github.io/blog/2019/02/15/autoencoder-downsampling-and-upsampling

[14] Tensorflow, Concatenate
https://www.tensorflow.org/api_docs/python/tf/keras/layers/Concatenate

[15] Paperswithcode, Cityscapes
https://paperswithcode.com/dataset/cityscapes

[16] Cityscapes dataset, Examples Zurich
https://www.cityscapes-dataset.com/examples/

[17] Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurélien Lucchi, Pascal Fua and Sabine Süssstrunk, “SLIC Superpixels Compared to State-of-the-art Superpixel Methods”, 2011, page 2
https://www.researchgate.net/figure/Images-segmented-using-SLIC-into-superpixels-of-size-64-256-and-1-024-pixels_fig1_225069465

[18] Ajitesh Kumar, “Keras – Categorical Cross Entropy Loss Function”, 2020
https://vitalflux.com/keras-categorical-cross-entropy-loss-function/

[19] Keras, Image segmentation metrics (MeanIoU class)
https://keras.io/api/metrics/segmentation_metrics/#:~:text=Mean%20Intersection%2DOver%2DUnion%20is,true_positive%20%2B%20false_positive%20%2B%20false_negative