ENSTA

IP PARIS

Research Internship (PRE)

Scholar Year: 2018-2019

Convolutional Neural Networks For Road Detection

An approach based on patch classification

Confidentiality Notice

Non-confidential report and publishable on Internet

Author:
Caio SOUZA CHAVES

Promotion:
2020

Professor:
Antoine MANZANERA

Tutor:
Clément PINARD

Internship from 13/05/2019 to 02/08/2019

Host organism: U2IS - ENSTA Paris
Address: 828 Boulevard des Maréchaux
91120 Palaiseau
France

Confidentiality Notice

This present document is not confidential. It can be communicated outside in paper format or distributed in electronic format.

Abstract

This project aimed at studying different supervised learning algorithms to perform road detection in images captured from a driving context. It can be divided in three main parts: in the first one, an ordinary convolutional neural network (CNN) was trained from scratch on the KITTI dataset to classify small patches of the image as road or not-road. Some variations to this base model architecture were proposed, mainly the addition of multi-scale patches and location features, and their effect on accuracy rate was evaluated. In the second part, these CNNs models were used to perform semantic segmentation through a method of sliding window. Finally, the algorithms efficiency were studied, a minimalist and faster version introduced and all the models compared to state-of-the-art network in this field. This study revealed that patch-classification-based algorithms can achieve slightly worse, yet comparable accuracy rate when compared to what the state-of-the-art does. Besides, that algorithm is considerably lighter, which may make it a good choice in some applications for which memory and processing power are limited, such as some embedded systems.

Acknowledgment

I would like to thank professor Antoine Manzanera, Daniela Pamplona and Clément Pinard for their supervision and support throughout this project.

Contents

Part I

Introduction and Problem description

I.1 Literature Review

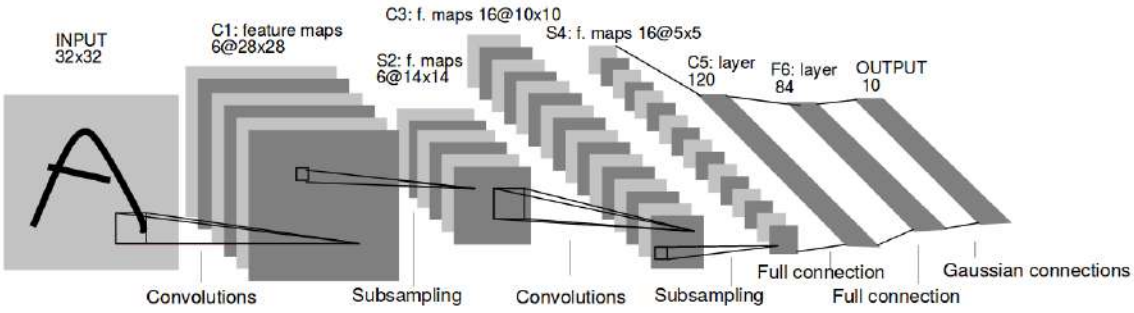
I.1.1 From Computer Vision to Convolutional Neural Networks

In the summer of 1966, a group of researchers at the Massachusetts Institute of Technology teamed up to work on "The Summer Vision Project", aiming to construct a visual system for pattern recognition in images. In this six-page-long project description document [1], some goals were set, such as identifying objects on a picture based on color and texture and doing segmentation into regions described as background, objects and "chaos". Ever since, the field of computer vision has bloomed for many reasons, including the development and popularization of digital cameras, which generated enormous amount of visual data to process and analyse, and the improvement in computer processing power, that enabled more sophisticated methods.

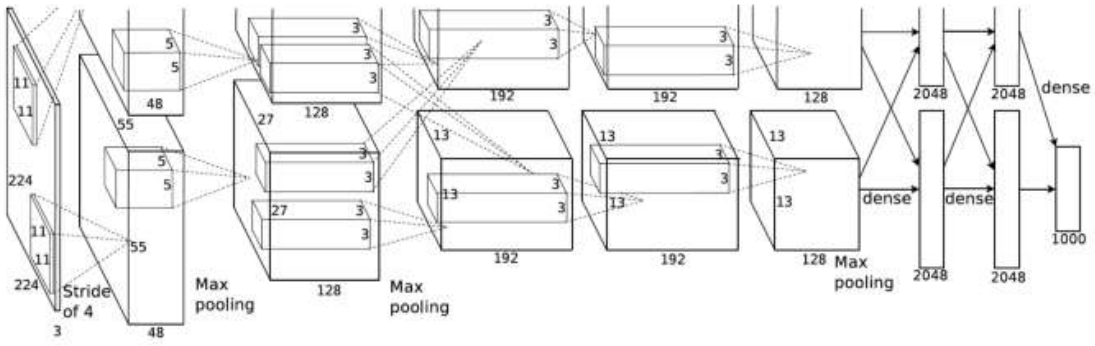
In 1989, LeCun et al. published an article about recognition of handwritten digits using networks and backpropagation in a supervised learning method [2]. Some years later, in 1998, the same researcher proposed a convolutional neural network (CNN) named LeNet-5 to perform the same task. Using gradient-based learning techniques and a 7-layer architecture, this model outperformed its competitors at that moment, showing how well CNNs can perform on image classification tasks and becoming an emblematic instance of this type of algorithm (Figure I.1).

An important event when it comes to introducing innovating, ground-breaking solutions on computer vision is the ImageNet Challenge [3]. ImageNet is a dataset of over 15 millions labeled high-resolution images with 22 thousands categories. The ImageNet Large Scale Visual Recognition Competition (ILSVRC) proposes different challenges on visual recognition. Among them, there is the classification challenge, whose goal is to design an algorithm to correctly label a subset of 1.4 million images into 1000 object classes. In the 2011, the winning algorithm used support vector machines (SVM) and achieved a 25.8% top-5 error rate, meaning that the correct class was among the five most likely the program pointed out 74.2% of the times.

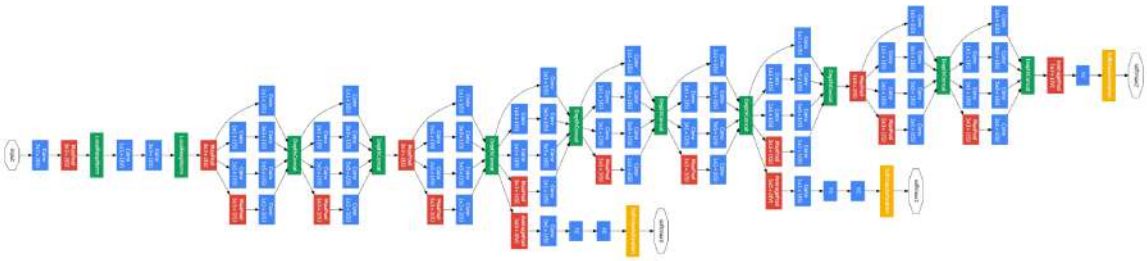
In 2012's edition, the winning algorithm made this error rate drop to 16.4%. Developed by a group of researchers from the University of Toronto, the so-called AlexNet (Krizhevsky et at. [4], figure I.1b) achieved state-of-the-art results using seven layers of convolutional and pooling operations. Ever since, CNN-inspired models have won this challenges several times are they are becoming deeper and deeper, showing more and more layers, as for example the winner of 2014, GoogLeNet, that figure I.1c depicts [5].



(a) LeNet-5. Image source: [2]



(b) AlexNet. Image source: [4]



(c) GoogLeNet. Image source: [5]

Figure I.1: Notorious and emblematic CNN models.

I.1.2 Semantic Segmentation Vs. Classification

The problem of semantic segmentation consists in assigning to each pixel in the input image a specific class in the output. It has applications in several domains, like health-care and transportation. In medical images, for instance, CNNs have been used to segment interest areas from magnetic resonance images with an accuracy comparable to that of a human specialist [6]. Another example is the one of autonomous cars: several tasks as lane and road detection use data from several sensors, like cameras and LIDAR, to offer assistance to the driver.

It is possible to perform a semantic segmentation task of an input of any size with a network developed and trained for classification. To implement this, one may use a sliding window that scans the image and assigns to each patch of pixels a class. This approach is not commonly used because it can have low efficiency in terms of time to perform this calculation, as the amount of forward steps in the CNN grows with the input's resolution.

There are also CNNs designed specifically for the task of segmentation, like SegNet [7],

U-Net [8] and FC-DenseNet [9]. They generally follow the same recipe in two stages: an encoder network followed by a decoder network. The first one learns discriminative low-resolution features, whereas the second one projects them onto the pixel space to obtain a dense segmentation.

Comparing those two options, each one presents its pros and cons. The patch-based approach usually has in general much smaller and lighter networks, that are typically faster and simpler to train. Although time-consuming, there are some trade-offs between prediction accuracy and speed that can still make this option a good choice for some applications. The classic, dense-based approach normally shows better accuracy results and it only takes one forward propagation step to output the full prediction. However, these models are generally much deeper and have much more weights to train, which make them harder to train and heavy to run and store.

I.2 Programming Tools and Dataset

Throughout the development of this project, Python was the main programming language used. More specifically the machine learning library PyTorch was used to build and train CNNs. Originally developed by Facebook's AI research group, this framework offers GPU-accelerated tensor computing and a neural networks module that allows for easy calculation of the gradient of functions through the method of automatic differentiation.

Figure I.2: PyTorch logo

In the context of supervised learning, it is fundamental to use a set of images with their respective ground truth during the learning phase of the CNN. The KITTI Vision Benchmark Suite [10] (a project of Karlsruhe Institute of Technology and Toyota Technological Institute at Chicago) provides 200 pairs of RGB images taken from cameras mounted on top of a station wagon, shown in figure I.3, in many driving environments with their respective manually annotated pixel-wise semantic segmentation in 32 different classes, like road, sidewalk, vegetation, car, sky, pedestrian etc, as the figure III.5c shows. To our task of road detection, the distinction of the classes that are not road is not relevant, thus they have been grouped in a super-class that could be named 'not road', as shown the figure I.4c. Finally, the binary semantic segmentation can be described as a function that maps every pixel *px* in the input image to a probability value P_{targ}.

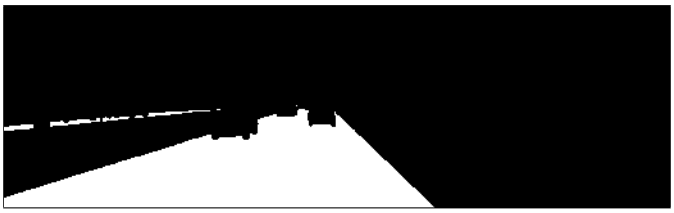
$$P_{targ}(px) = \begin{cases} 1, & \text{if pixel is road} \\ 0, & \text{if pixel is not road} \end{cases} \tag{I.1}$$

Some statistics of this dataset can be seen in the Table I.1. The images have three channels (RGB), height of 370 pixels and width of 1224. Considering the binary scenario, 77,4% of pixels across all dataset are classified as non-road. According to this number, a non-trained classification algorithm can achieve a accuracy as high as 77.4% just by always inferring non-road, which sets a baseline for the neural networks that will be later trained.

Figure I.3: KITTI automobile used for capturing images

(a) RGB image from road scene

(b) Multi-class pixel-wise semantic segmentation



(c) Binary-class semantic segmentation

Figure I.4: Example from the KITTI semantic dataset

Images resolution (channels x height x width)	3 x 370 x 1224
Total images	200 (160 training / 40 validation)
Non-road pixels (standard deviation)	77.4 % ($\sigma = 5.6\%$)
Maximum non-road pixels in any image	89.7 %
Minimum non-road pixels in any image	66.5 %

Table I.1: KITTI data set statistics

The 200 pairs of image and semantic segmentation were split in training subset (80%) and validation subset (20%). These subsets are the same for the several architectures evaluated, in order to have a fair and meaningful comparison between them.

Part II

Patch-based CNNs for classification

II.1 Base CNN model Architecture

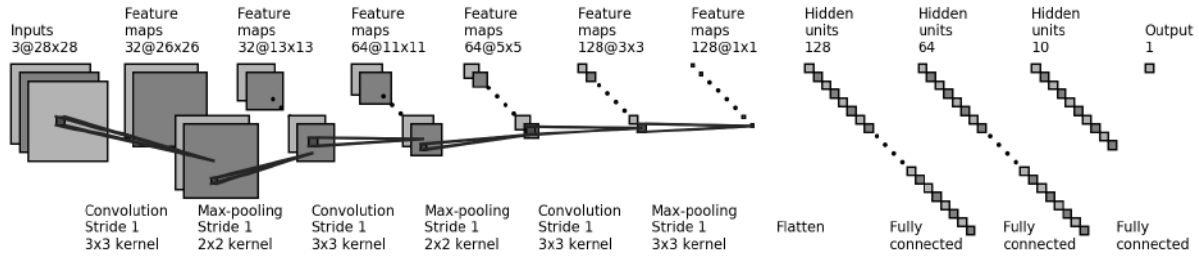


Figure II.1: Base CNN model architecture

The base architecture used in this project can be seen in the figure II.1. It was conceived based on the convolutional neural networks described in the literature, such as [2]. Given as input a patch having 3 channels of 28x28 pixels, it outputs a single scalar corresponding to the probability of that input being road.

The CNN consists of three convolutional layers that have 32, 64 and 128 filters of size 3x3, followed by rectified units (ReLU) activation functions. Between these three layers, spatial pooling operations are performed, as the figure shows, in order to reduce the resolution of the feature map and also reduce the output sensitivity to shifts and distortions [2]. After that, the vector of features is flattened and send into a sequence of three fully connected layers of size (128,64), (64,10) and (10,1). Finally, the resulting single scalar is turned into a probability value using the *sigmoid* function, seen in the equation II.1, which gives us the predicted probability of the input patch correspond to road.

$$P_{pred}(x) = \frac{1}{1 + e^{-x}} \tag{II.1}$$

II.2 Loss Function and Optimizer - Supervised Learning

Following the supervised learning approach, the predicted value (P_{pred}) must be then compared with the target value (P_{targ}) provided by the KITTI dataset using the loss function. The binary cross entropy (Eq. II.2) was chosen as the function, given that it is the binary classification problem.

$$\mathcal{L}(\vec{P}_{targ}, \vec{P}_{pred}) = -\frac{1}{N}\sum_{i=1}^{N} t_i \log p_i + (1 - t_i) \log (1 - p_i) \tag{II.2}$$

where N corresponds to the batch size, $\vec{P}_{targ} = (t_1, t_2, ..., t_N)$ and $\vec{P}_{pred} = (p_1, p_2, ..., p_N)$ are N-dimensional vectors containing, respectively, the target and prediction probabilities values.

Each weight w_i of the network is then updated based on the stochastic gradient descent (SGD) with momentum technique, according to the equation II.3. The term v_i works as moving average that smooths out the gradient step. It can be understood through a mechanical analogy of a ball rolling down a hill: the term multiplying β corresponds to the "velocity" from previous states, whereas the one multiplying $(1 - \beta)$ is the "acceleration". By taking in consideration the "momentum" term, this algorithm shows overall better performance because, it avoids local minima points while the "acceleration" still allows for changes. Finally, the parameter α is the learning rate.

$$\begin{cases} v_i \leftarrow \beta v_i + (1 - \beta)\frac{\partial \mathcal{L}}{\partial w_i} \\ w_i \leftarrow w_i - \alpha v_i \end{cases} \tag{II.3}$$

As the goal of this project was not to find the choice hyperparameters combination that maximize performance, they were chosen based on literature recommendations (as the Table II.1) and kept constant across all test that will be further presented in this report.

Learning Rate (α)	SGD Momentum (β)	Batch size (N)	Patches per images (PPI)
0.001	0.5	1	50

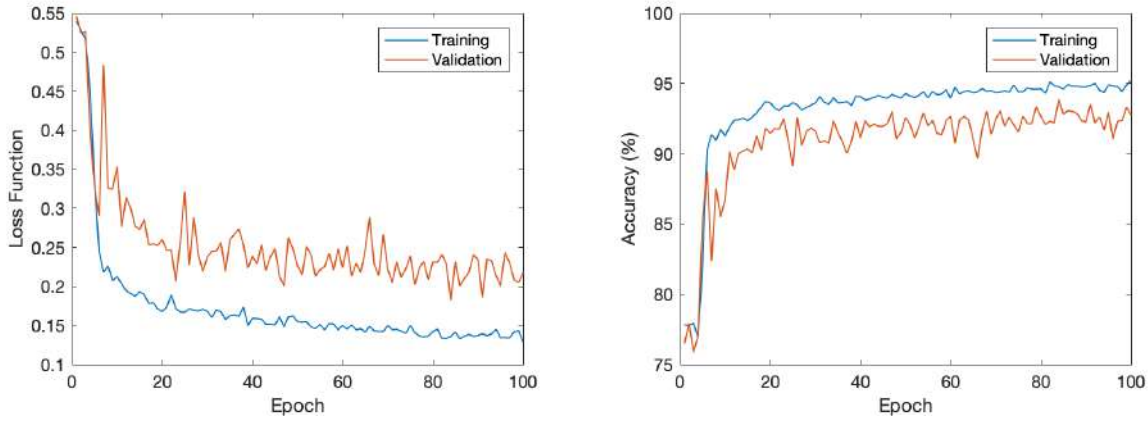
Table II.1: Learning Hyperparameters

II.3 Learning Algorithm

One by one, all the 160 images that belong to the training subset are sent to the training algorithm in which patches are taken from random selected positions across all the image and then propagated into the CNN "PPI" times (and also back-propagated). The parameter "PPI" stands for patches per images and it was fixed at 50 for all tests in this report, but its variation has major impact in the learning process: the greater it is, the longer it takes each epoch of training; also, if this parameter is too large, the network risks to overfit to the context presented in the first image input, which does not have the same variability as the whole set of images, leading to potential overall weaker performance. Obviously, it also can not be too small so the network still gets to the experience and learn from patches in diverse positions in the image.

II.4 Learning Metrics Visualization

The figures II.2 shows how evolve along one hundred epochs both the binary cross-entropy (BCE) loss and accuracy. Neither transfer learning nor weight initialization were employed, which means the CNN starts from scratch. The loss function (Fig. II.2a) falls steeply before epoch 10 and then continues to decrease in a slower rate. As expected, its value for the



(a) BCE Loss Evolution (b) Accuracy (threshold = 0.5)

Figure II.2: Learning metrics for base architecture

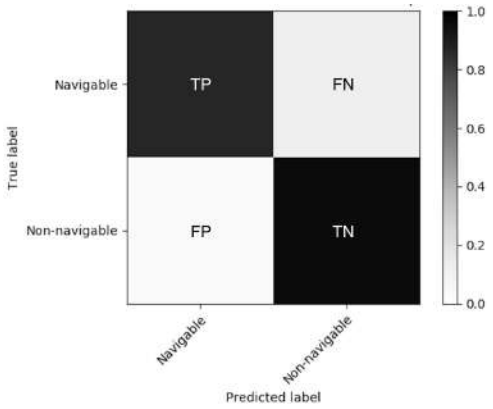


Figure II.3: Confusion Matrix

validation dataset oscillates up and down more intensively and has a greater average value, when compared to the training one.

Accuracy was calculated based on the equation II.4, where TP, TN, FP and FN represent respectively true positive, true negative, false positive and false negative, according the confusion matrix seen in figure II.3 (please note that in this context navigable is synonim for road). In order to evaluate these quantities, it was necessary to apply a threshold to the output prediction of the CNN, creating a binary prediction version: for an arbitrarily chosen threshold of 0.5, when the pixel value was in the interval $[0.0, 0.5]$ its binary value is 0 and when it was in the interval $]0.5, 1.0]$, 1. Naturally the larger accuracy may not come with a threshold of 0.5, so the optimum value will be discussed in a future session.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{II.4}$$

For the accuracy (Fig. II.2b), the starting point is between 75% and 80%, which is in agreement with the dataset characteristics shown in the table I.1. After one hundred epochs, this metric is between 90% and 95%. The same pattern of oscillations seen on the loss function graph is again present, as well as the expected gap between training and validation subsets.

II.5 Region-specific CNNs grid

Different regions in a driving scene have specific characteristics, such as context, predominant texture and color. Besides that, pixels that are closer to the image borders tend to experience greater deformations than the ones in the center due to the use of wide angle lens. As a result, forms and objects proportions are distorted and may cause problems during the learning process.

As an attempt to increase accuracy, we propose to incorporate this inherent region dependent characteristics into the training process in two phases. First, a single CNN is trained in all image (as before) until epoch 50. After that, this network is cloned in six independents ones and each one is assigned to a specific region, as illustrated in figure II.4, and they are trained for 50 epochs more. In other words, for instance the CNN allocated to region [0,0] is only trained in patches extracted from that region. In order to evaluate whether this is a good strategy, the same single CNN that first started will also continue its learning process in all image, so at the end of one hundred epochs we will be able to do a fair comparison.

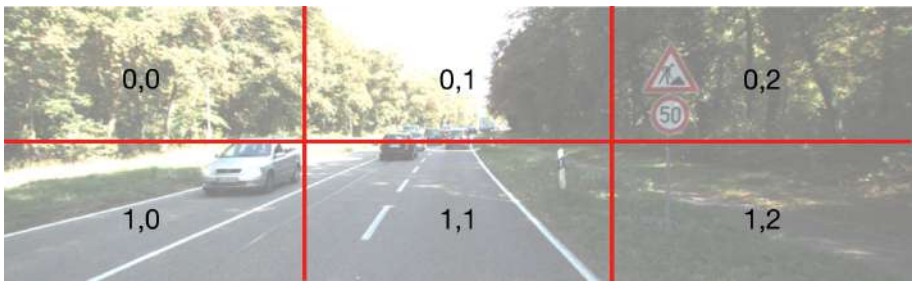
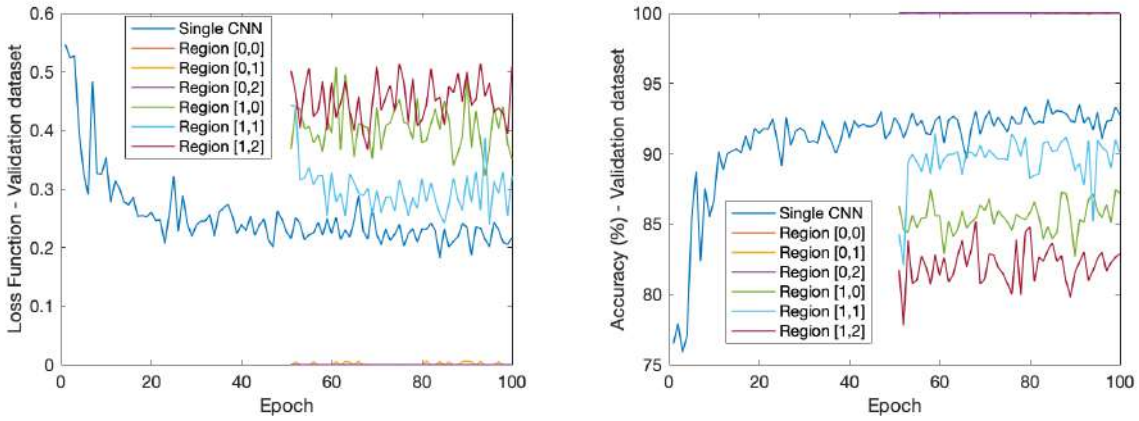


Figure II.4: Division in different regions

Figures II.5 shows the learning metrics for each region, as well as for the single CNN. First of all, it is noticeable that the loss function tendency is still downwards, showing that these changes do not cause bizarre behaviours. Second, we see that regions on top ($[0,0]$, $[0,1]$ and $[0,2]$) have much higher accuracy and smaller loss function values right after the split that occurs at epoch 50. In opposition to that, the regions on the bottom ($[1,0]$, $[1,1]$ and $[1,2]$) have lower accuracy and greater loss function. This results are as expected, since most road pixels are concentrated on the bottom, where most of driving context takes place.

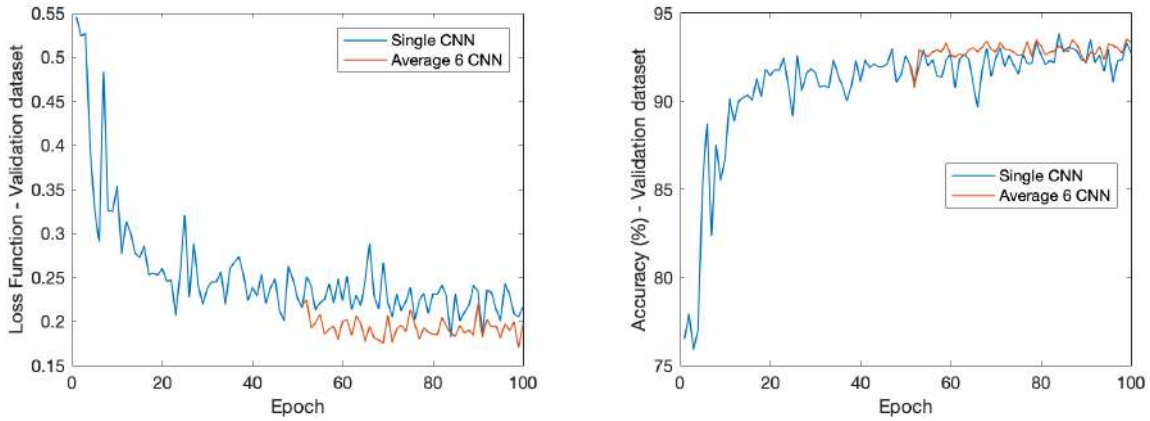


(a) BCE Loss Evolution

(b) Accuracy (threshold = 0.5)

Figure II.5: Learning metrics for base architecture

Figure II.6 shows averaged metrics of the six region-specific CNNs in comparison with the single CNN. These graphs allow us to draw some initial impressions. First, the overall performance calculated as the mean value per epoch for either loss function or accuracy is slightly better in the region-specific case. Second, the "oscillations" have a smaller amplitude for the region-specific approach.



(a) BCE Loss Evolution

(b) Accuracy (threshold = 0.5)

Figure II.6: Overall performance given the mean value of regions metrics.

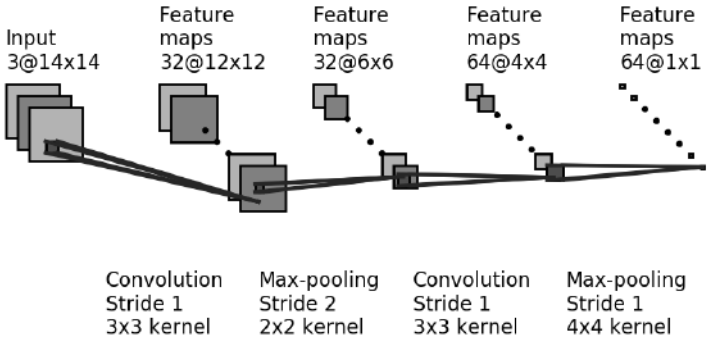
Furthermore, on the part "III - Semantic Segmentation Reconstruction From Patch Classification", the final performance of the different strategies tested (some yet to be presented) will be compared in a more methodical, standardized method.

II.6 Multi-Scale Patches

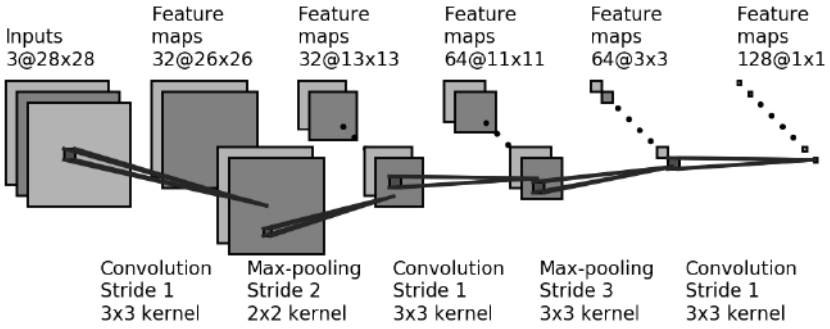
When choosing the patch size there is a trade-off between context awareness and texture recognition. A bigger selection of pixels means having access to a more meaningful sample of the image, whereas a smaller one focuses on identifying texture differences between classes such as road, sky, vegetation etc. in a driving scene.

To assess this trade-off it is proposed a multi-scale patches approach, inspired by the article [6]. In this scientific publication, the authors perform the segmentation of white matter hyperintensities (WMH) from magnetic resonance images using deep convolutional neural networks. They see remarkable improvement on their results due to (1) using multi-scale patches and (2) incorporating explicitly spatial location features.

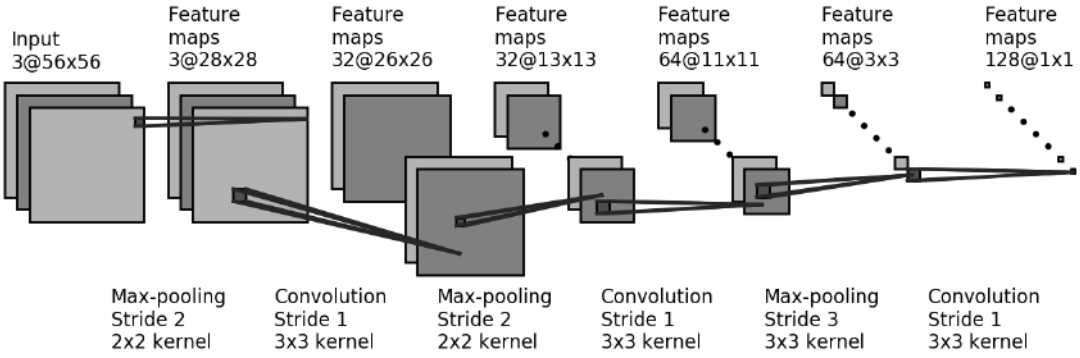
The figure II.7 shows the architecture of the designed multi-scale model. There are three sizes of patches (small: 14x14, medium: 28x28 and large: 56x56) that have independent branches *a priori*, consisting of a sequence of convolutions, ReLU activation functions and max pooling operations. Their features maps of sizes 64, 128 and 128 are then concatenated to give origin to a 320-dimensional array of features, as shown in figure II.7d. Similarly to the base model, the output is a single scalar converted to a prediction between 0 and 1 by the sigmoid function.



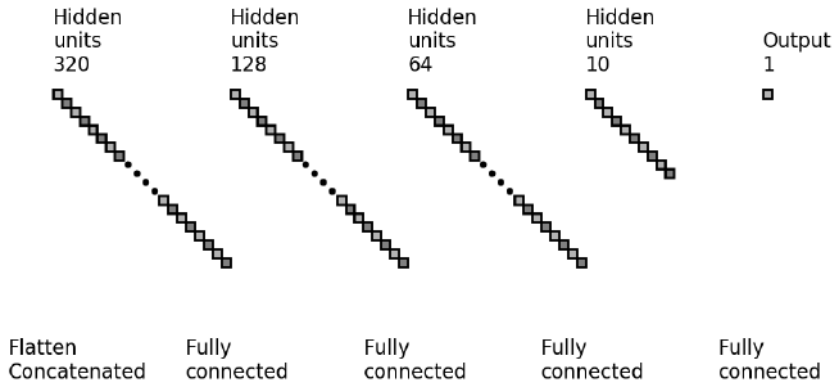
(a) Small patch branch



(b) Medium patch branch



(c) Large patch branch



(d) Fully connected layers after concatenation

Figure II.7: Multi-scale model architecture

II.7 Position Features Incorporation

The use of spatial location features is justified in [6] by the observable pattern in WMH occurrence probability map in their dataset, which is highly uniform when it comes to presenting the same kind of elements in the same location for several images.

The assumption that the same is true for the KITTI or any other driving scenes dataset raises more questions. In our problem, we have to deal with much greater variability of scenarios: First, the driving environment (city, highway, countryside). Second, the vehicle pitch angle (in a atypical situation, the road will potentially not be where it usually is), etc. Thus, we do not expect the models trained with access to these location features to perform well in extreme conditions, but they might bring nice improvement in regular ones, such as cruising down the highway. However, the probability map seen at figure II.8 still suggest that there is a pattern location, and that supports the implementation of this idea.

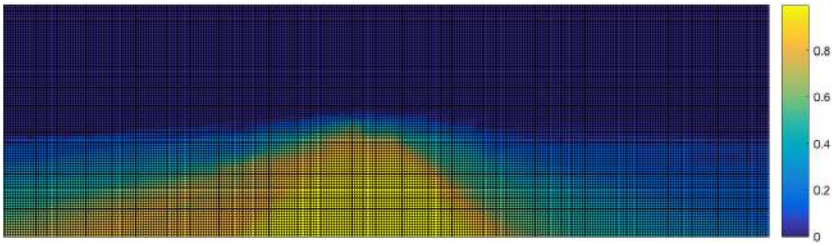


Figure II.8: Pattern in road occurrence in the probability map.

An array with two position features was concatenated to the 64-dimensional feature map seen at II.7d. The figure II.9 illustrates the way they were computed. To make them less dependable on image resolution and ratio, they were normalized by the height H of the image. The geometrical center of the rectangle $\vec{O} = [H/2, W/2]$ is considered to be the origin and reference of the coordinate system, in such a way that a point at the bottom right corner will systematically has as position features $(0.5, 0.5 * W/H)$.

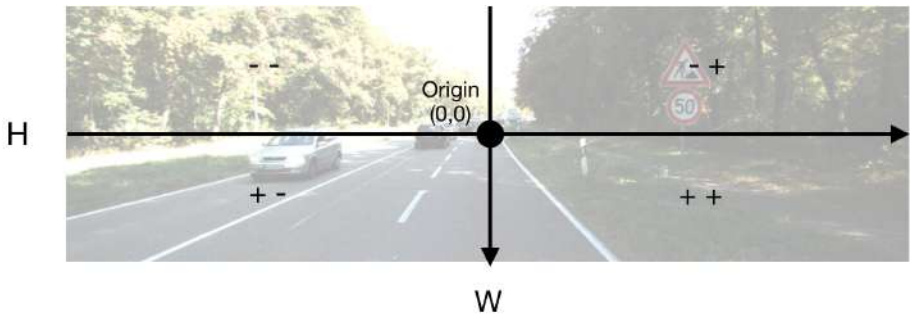


Figure II.9: Position features schema of computation

Figure II.10 shows the evolution across the 25 last epochs of the accuracy on the validation set of all proposed architectures so far. Both multi-scale patches models have good results, which shows that this strategy brings a good contribution to pattern recognition and scene understanding. Besides that, the CNN that incorporated position features showed an even higher accuracy rate than the one without, indicating the overall positive impact of this strategy in particular. Even though heavier and less practical, the region-specific grid of CNNs show the smaller improvement in comparison to the base model.

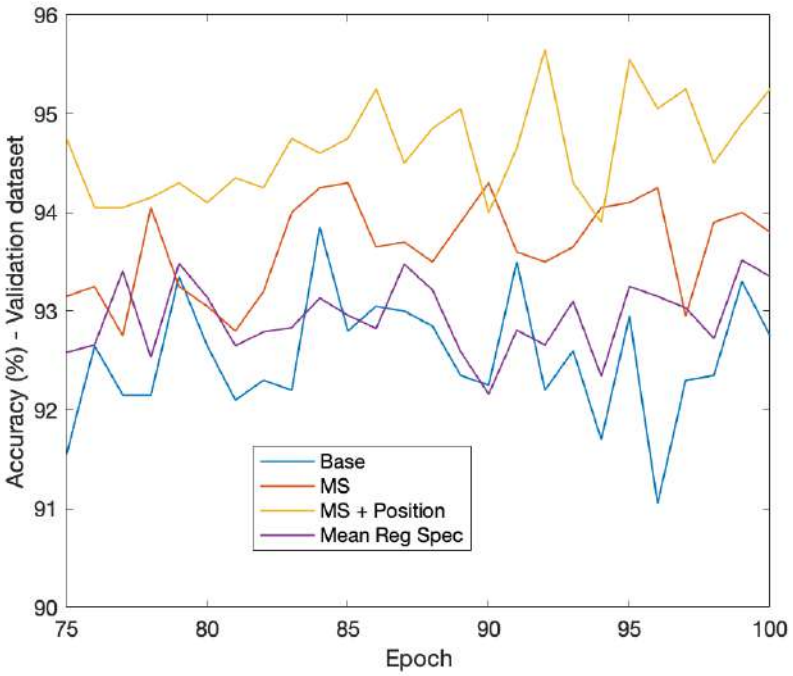


Figure II.10: Comparison between models using accuracy

Part III

Semantic Segmentation

Conventional fully convolutional neural networks most frequently used for semantic segmentation have an inherent benefit over the patch-based ones: as they take the full image at once as the input, they can explore and learn from all the context available in it at once. Also, one needs to do forward propagation just once to have the input's segmentation map with the same pixel resolution. However, these networks frequently have hundreds of layers and millions of trainable weights; as a result, learning process may be more time consuming and the model potentially inappropriate for some applications where memory or processing power are limited.

On the other hand, it is also possible to generate a semantic segmentation map using a CNN trained for classification and that will be described in the first section of this part. Nonetheless, this can be highly inefficient, since the image needs to be "scanned" by a sliding window, performing one patch-classification at a time, in order to build a semantic map with the same dimensions as the input image. We investigate if there's an amount that optimizes the trade-off between quality of the segmentation map and the time necessary to generate it in the second section, in which a parameter named stride that controls this trade-off is explained and its impact is studied.

On the following section, a state-of-the-art CNN that is among the top performing models for semantic segmentation of driving scenes is studied. It is trained for our task of road detection on the KITTI dataset in order to evaluate the performance of our models built so far with respect to the benchmark in this field. Finally, the last session contains some instances of semantic maps to illustrate the results.

III.1 Reconstruction From Patch Classification

Given the input image with size $3x370x1224$, the reconstruction algorithm should provide a segmentation with $1x370x1224$. In order to do so, the CNN trained for patch classification needs to scan the input row after row, column after column, moving forward $stride$ pixels every iteration, in this sliding window technique. After that, the pixels in the square sized $stridexstride$ contained within the patch are all assigned the same class score.

In the figure III.1 a first sample is shown. It consists of the original image overlapped with the reconstructed segmentation map, where the pixels in blue have high probability of being road and those in red, of being non-road. This result was obtained with the model with multi-scale patches and position features included and its accuracy goes as high 99.15% for a threshold equal to 0.5. The algorithm does well at contouring the car in the opposite lane, which is remarkable.

(a) Segmentation map without threshold

(b) Threshold = 0.5, accuracy = 99.15%

Figure III.1: Image 183, validation dataset; stride = 1

III.1.1 City environment

Figure III.2 shows the prediction that the "MS + Pos" model outputs for a city driving scene after applying a threshold equal to 0.5. For this image the accuracy is 85.62%. Compared to the scene shown in the figure III.1, this one has a more complex context, which introduces some sources of errors. First, the paintings on the road and the bike lane represent some variability within the class "road" that the network was not able to understand. Secondly, the sidewalk is almost completely entirely misunderstood as road, which rises the rate of false positives to 11.72%. Third, the false negative pixels (2.66%) are concentrated in an area with shadows on the asphalt, another example of class variability that was not well learned.

This example helps us understand how the CNN evolves to perform classifications and it shows some really important limitations in it. The concrete in the sidewalk, as well as in surrounding buildings, have a similar texture when compared to the road asphalt, which leads to errors. As a consequence, the model shows limitations in city environments.

Figure III.2: Image 172, validation dataset; stride = 1; threshold = 0.5; accuracy = 85.62%

III.1.2 Multi-Scale contribution visualized

The addition of multi-scale patches aimed at allowing for better context awareness without letting go from texture analysis, when compared to the base, single-scale patch model. As it can be seen in figure III.3, the accuracy increases from 93.44% to 95.15% and false positive rate drops from 4.49% to 2.62%. On the right side, for instance, the base model III.3a badly classify part of the tree trunk, whereas the MS one III.3b does not commit the same mistake. Also, the sidewalk region sees some improvement, even tough it is still partially wrong.

(a) Single-Scale, Base model, accuracy = 93.44%

(b) Multi-Scale patches, accuracy = 95.15%

Figure III.3: Image 84, validation dataset; stride = 1; threshold = 0.5

III.1.3 Position features contribution visualized

Figure III.4 illustrates the effect of explicitly incorporating two position features during the CNN training phase by comparing the multi-scale models outputs with and without this location information. For the figure on the middle (III.4b) the confusion between concrete and asphalt persists, which causes the accuracy rate to be as low as 84.42%. The false positives pixels make up for 95.31% of the badly classified pixels (FP = 14.85%, in absolute terms). The figure III.4c shows how the position features contribute to decrease this error rate: the pixels on the top region are no longer badly classified. Putting this in numbers, the accuracy increases to 87.27% and the false positive rate is now 12.11%.

(a) Base model (Accuracy = 75.99% / FP = 23.29%)

(b) Multi-scale model (Accuracy = 84.42% / FP = 14.85%)

(c) Multi-scale model with position features (Accuracy = 87.27% / FP = 12.11%)

Figure III.4: Image 165, validation dataset; stride = 1; threshold = 0.5

III.2 Stride Impact

The figure III.5 shows qualitatively what is the effect of changing the stride. Naturally, for a smaller stride, the transition between domains (road and not-road) is smoother than for a bigger one. Nonetheless, the results are still comparable, as the accuracy written in the caption shows. The rate of false negatives corresponds to the biggest source of errors in all cases, i.e. pixels that are tagged as road in the ground truth but predicted as not-road by the CNN.

The relation between stride and accuracy are shown in the figure III.6. These numbers correspond to the average accuracy across the 40 images in the validation subset with the threshold that optimized each model's performance measured for strides values of 1,3,5,10,14 and 28 (limit value, equal to the patch size). As expected, for a stride equals to one the accuracy reaches its maximum value for all the architectures and it decreases as the stride increases until 28. However, the total drop is relatively small (1.4% in average) and the results can be still considered decent for the largest stride available.

In opposition to that, the variation of this parameter has a major impact on the time required to perform the full segmentation of an image, as seen in III.7. The number of forward passes necessary to perform this task is equal to the product between image's height and width divided by the square of the stride value, which makes it a $\mathcal{O}(n) = n^2$ algorithm.

(a) Stride = 1 (Accuracy = 99.15% / FN = 0.76%)

(b) Stride = 10 (Accuracy = 98.29% / FN = 1.35%)

(c) Stride = 28 (Accuracy = 97.34% / FN = 2.19%)

Figure III.5: Image 129, validation dataset, variable stride, threshold = 0.5

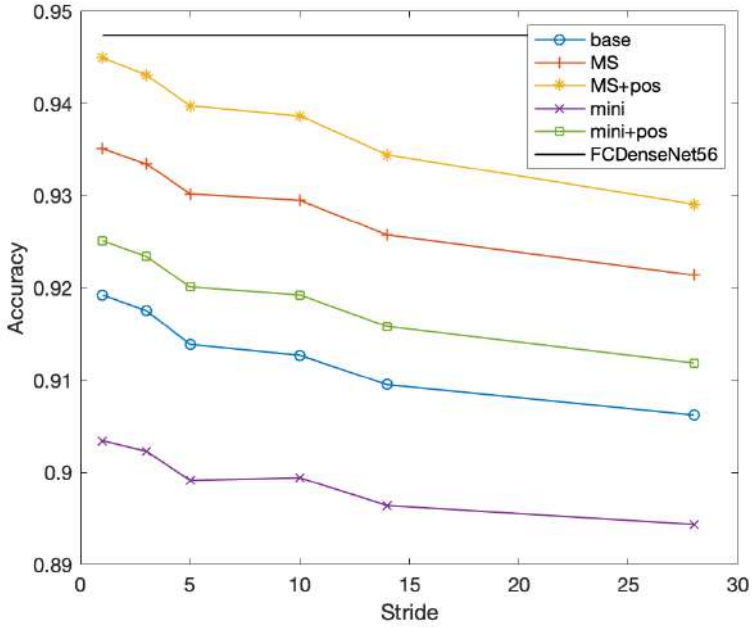


Figure III.6: Stride impact on semantic reconstruction accuracy

Evaluating an algorithm efficiency by its execution time is simple but also naive, given that this metric varies in function of both the implementation and the computer hardware. To minimize it, all tests in this report presented were run in the same computer whose specifications are: 6-core intel xeon E5-2603 1.70 GHz, 24 GB RAM, GPU GeForce GTX TITAN, Ubuntu 16.04.6 LTS, Python 3.7.3. A more sophisticated analysis in terms of energy and number of operations in one or more hardware configurations would make the discussion around this point more solid and it is left as a suggestion for future work.

III.3 Minimalist CNN for Patch Classification

One alternative to speed up our algorithm is to build a simpler CNN with less layers and less weights. In this section we propose to make a minimalist model that is still able to perform the task with satisfactory results in a reduced time.

The proposed architecture is seen on figure III.8. It has two convolutional layers that are followed by ReLU activation with filter size 4x4 and 3x3. It generates a 30-dimensional feature map that finally outputs a single output, as all the other models seen so far. It has been also tested a second variation with the position features as before that are concatenated to the feature map mentioned above, which becomes 32-dimensional in this case.

This network has only 4846 trainable weights in its simpler version and 4848 in its version with position features included. The table III.1 compares different models in terms of weighs and space required in memory to store them.

Figure III.9 shows how comparable are the prediction by the minimalist architecture and the one by the base model. For this particular sample, the three models do well at contouring the grass are at the bottom left and the pedestrian a little above. The addition of position features in the mini model causes the CNN to classify correctly the pixels next to road paintings at the center of image, which reduces the rate of false negatives without increasing the rate of false positives.

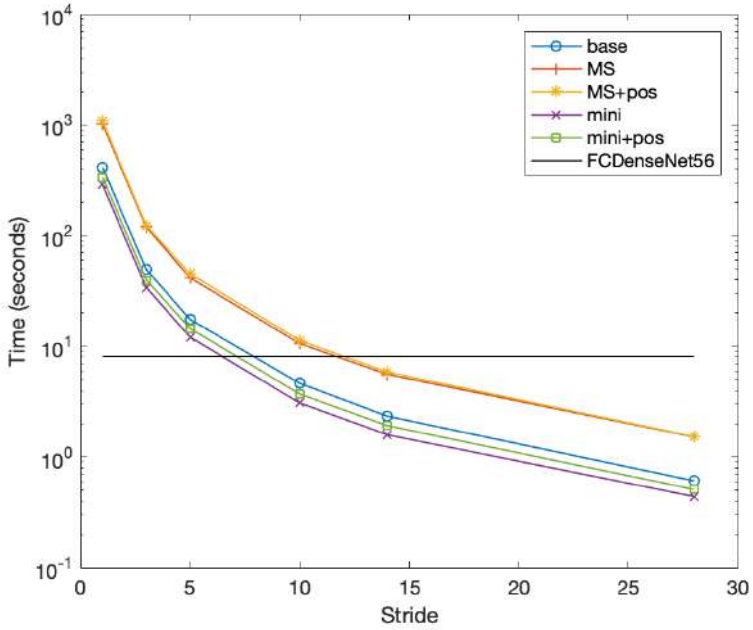


Figure III.7: Stride impact on total time

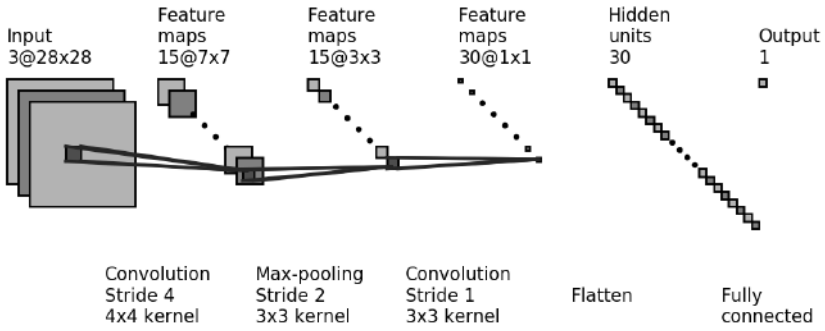


Figure III.8: Minimalist CNN model architecture

III.4 FCDenseNet56, State-of-the-art

In the paper [9], Jégou et. al. introduce a CNN model for semantic segmentation based on Densely Connected Convolutional Networks (DenseNets) that outperforms current state-of-the-art results on standard benchmarks for urban scene understanding. Their model, named FCDenseNet, achieves the best score so far in a 11-class segmentation in the CamVid test dataset [11], with a 91.5% global accuracy.

Among the three variations of architectures proposed in [9], the FCDenseNet56 was chosen because it was the lighter model. Despite the fact of being the simpler version, it has 56 layers and 1.3 million trainable weights, as the table III.1 shows. Besides that, it hits 88.9%. global accuracy on the same test described above. A qualitative sample of the task this CNN is able to perform can be seen in the figure III.10.

The PyTorch implementation done by B. Fortuner [12] available at his GitHub repository served as the starting point for this project's one. However, this model has been slightly adapted to our task of road detection: as the desired output is a probability map and no longer a multi-class classification, the last layer, which was a softmax function, has been

(a) Base model (Accuracy = 95.15% / FN = 2.55% / FP = 2.30%)

(b) Minimalist (Accuracy = 93.85% / FN = 4.01% / FP = 2.13%)

(c) Minimalist + position feats. (Accuracy = 94.80% / FN = 3.11% / FP = 2.09%)

(d) MS Patches + position feats. (Accuracy = 95.48% / FN = 3.07% / FP = 1.45%)

(e) FCDenseNet56 (Accuracy = 93.69% / FN = 3.92% / FP = 2.38%)

Figure III.9: Image 16, validation dataset, variable stride, stride = 1

Model Architecture	**Trainable weights**	**Model size**
Base	102,165	410.6 kB
MS Patches + position feats.	255,913	1.03 MB
Minimalist + position feats.	4,848	20.6 kB
FCDenseNet56	1,374,865	5.65 MB

Table III.1: Models size

Figure III.10: FCDenseNet performance on the semantic segmentation of a CamVid image. Original (left), ground truth (middle) and prediction (right). Source: [9]

replaced by a sigmoid one.

To train the FCDenseNet56, a series of techniques were employed following some suggestions presented in the base implementation. They are summarized in the list below, together with some hyperparameters.

- *HeUniform* weight initialization.
- *Rmsprop* [13] as the optimizer .
- Binary cross-entropy (BCE) as the loss function.
- Batch size: 1.
- Learning rate of 0.001 with exponential decay of 0.995 after each epoch.
- Number of epochs: 40.
- Weight decay: 0.0001.
- Dropout: 0.2
- GPU based calculation was disabled, following memory issues when CUDA was enabled (the input image probably is too large).

III.5 Performance comparison

All the models described so far were tested under the same circumstances in order to have a meaningful and fair comparison. The test procedure followed the same order:

1. Select the best epoch for the model based on the highest accuracy rate on the validation subset during the training phase.
2. Generate the semantic segmented image for all the 40 samples in the validation dataset for several values of threshold between 0.1 and 0.9 with a fixed stride of 10.
3. Accuracy is evaluated and can be seen on the fourth column of the table III.2. using the combination of epoch and threshold that maximizes this rate.

Model Architecture	**Best epoch**	**Best threshold**	**Accuracy (%)**
Base, single-scale patch	84	0.7	91.61
Multi-scale (MS) patches	85	0.5	92.95
MS patches + position feats.	92	0.5	93.87
Minimalist	95	0.3	90.50
Minimalist + position feats.	72	0.6	92.09
FCDenseNet56	36	0.6	94.74
Region specific	[50,28,46,8,9,18]	0.6	91.18
Base, single-scale patch	50	0.6	91.18

Table III.2: Performance comparison between all models in terms of accuracy

The accuracy numbers seen on table III.2 show that the winning model is in fact the FCDenseNet56 (94.74%), followed by the multi-scale with position features included (93.87%). Besides that, the minimalist model, also with position features, achieves a good accuracy (92.95%), higher than the base, single-scale one (91.61%).

The table III.3 summarizes some information already presented, but that make clearer the comparison between these three models. FCDensetNet56 has the highest accuracy but it is also the heaviest model, with 5.65 megabytes needed to store its 1.3M weights. The minimalist model is two orders of magnitude lighter and also performs the task faster, in 3.71 seconds, given a stride of 10. The "lightness" of patch-classification-based models may be interesting for applications in which memory, power and processing power constitute limiting factors of the system, as some embedded systems.

Model Architecture	**Semantic Segmentation (sec)**	**Model Size**
Minimalist + position feats.	3.71 (stride = 10)	20.6 kB
MS patches + position feats.	11.3 (stride = 10)	1.03 MB
FCDenseNet56	8.12	5.65 MB

Table III.3: Reminder of models efficiency

III.6 More examples

Image 104 from the KITTI dataset captures the moment when the car passes through a tunnel and, consequently, the light condition changes dramatically. For this kind of conditions, all the models tested fail at detecting the road (zeros true positives). This scene can be seen in figure III.11.

This shows limitations not only of our particular model, but also for all systems build on light dependant devices such as cameras. Assessing these unusual conditions (tunnel, fog etc.) is essential for a complete autonomous system for, i.e., self-driving cars.

Figure III.11: Image 104, extreme condition, validation dataset

Figure III.12 shows image 73 from the validation subset semantic segmentation map for several models. It is a emblematic example, because the accuracy results for this image follows the same order as the overall performance score, seen at table III.2. Three models (III.12a, III.12b and III.12c) have the rate of false negative higher than 3%, which is related to the parts that are further away down the road. The better performing models (III.12d and III.12e) are able to correctly detect and segment distant points of the road.

(a) Base model (Accuracy = 95.69% / FN = 3.03% / FP = 1.27%)

(b) Minimalist + position feats. (Accuracy = 94.80% / FN = 3.11% / FP = 2.09%)

(c) Multi-scale (Accuracy = 96.36% / FN = 3.33% / FP = 0.31%)

(d) Multi-scale + position (Accuracy = 97.88% / FN = 0.77% / FP = 1.35%)

(e) FCDenseNet56 (Accuracy = 98.59% / FN = 0.48% / FP = 0.93%)

Figure III.12: Image 73, validation dataset, variable stride, stride = 1

Conclusion

Several CNNs for driving scenes understanding have been proposed, evaluated and compared in the specific task of road detection and segmentation. Patch-based-classification networks showed an improvement thanks to the use of multi-patches and incorporation of location features during the learning phase. Their time efficiency has been studied, mainly in function of the stride, which allows for balancing accuracy with "speed". Also, a minimalist network model was tested, aiming at applications in which memory, energy and processing power are limiting factors. All those CNNs were then compared in terms of both time and accuracy rate to generate a semantic segmentation map to the state-of-the-art architecture in the field of driving scenes segmentation. The results showed that patch-based models can achieve slightly worse yet comparable accuracy rate in even less time than the state-of-the-art model in some scenarios.

Bibliography

[1] Seymour A Papert. The summer vision project. 1966.

[2] Yann Lecun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998.

[3] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[4] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[5] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[6] Mohsen Ghafoorian, Nico Karssemeijer, Tom Heskes, Inge WM van Uden, Clara I Sanchez, Geert Litjens, Frank-Erik de Leeuw, Bram van Ginneken, Elena Marchiori, and Bram Platel. Location sensitive deep convolutional neural networks for segmentation of white matter hyperintensities. *Scientific Reports*, 7(1):5110, 2017.

[7] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495, 2017.

[8] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[9] Simon Jégou, Michal Drozdzal, David Vazquez, Adriana Romero, and Yoshua Bengio. The one hundred layers tiramisu: Fully convolutional densenets for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 11–19, 2017.

[10] Hassan Alhaija, Siva Mustikovela, Lars Mescheder, Andreas Geiger, and Carsten Rother. Augmented reality meets computer vision: Efficient data generation for urban driving scenes. *International Journal of Computer Vision (IJCV)*, 2018.

[11] Julien Fauqueur, Gabriel Brostow, and Roberto Cipolla. Assisted video object labeling by joint tracking of regions and keypoints. In *2007 IEEE 11th International Conference on Computer Vision*, pages 1–7. IEEE, 2007.

[12] Brendan Fortuner. Fc-densenet in pytorch for semantic segmentation. https://github.com/bfortuner/pytorch_tiramisu, 2018.

[13] Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

List of Tables

List of Figures