IMAGE SEGMENTATION BY SUPERPIXELS

T.Dumont[a], **B.Figliuzzi**[b]

a. MINES ParisTech, theo.dumont@mines-paristech.fr
b. MINES ParisTech CMM, bruno.figliuzzi@mines-paristech.fr

Key-words:
deep learning; convolutional neural networks; image segmentation

Abstract:
In this paper, we present an algorithm based upon convolutional neural networks for generating superpixel partitions of images.

By combining an algorithm that generates superpixel partitions through the resolution of the Eikonal equation and ground truth segmentations from the Microsoft Common Objects in Context (COCO) dataset, we were able to generate training examples of superpixel partitions of the images of the dataset. These training examples arise in the form of RGB image where the color is averaged over each superpixel. A convolutional network architecture is then trained on these images. A superpixel algorithm is finally applied to the output of the network to construct the seeked partition.

The algorithm is evaluated on the Berkeley Segmentation Dataset 500. It yields results in terms of boundary adherence that are comparable to the ones obtained with state of the art algorithms including SLIC, while significantly improving on these algorithms in terms of compactness and undersegmentation.

1 Introduction

1.1 Segmentation

While looking at an image, the human brain uses a lot of knowledge to understand its content. However, from the perspective of a computer, an image is only a set of integer valued pixels. Hence, making it produce an interpretation is much more complex. Segmenting an image consists in transforming the image in a representation that is easier to analyze, and much more meaningful. A segmentation partitions an image into multiple connected sets of pixels sharing certain characteristics including similar colors or texture patterns. It allows one to locate the objects on an image or to point out their boundaries.
The applications of such a process are numerous: control of the an object outlines on a production line, face detection, medical imaging, pedestrian detection, video surveillance... They justify our search for higher segmentation performances.

(a) *The original image* **(b)** *The segmented image*

Figure 1: *Segmentation examples. To each segmented region of the image was affected the mean value of its pixels. Both images are from the COCO training dataset [1].*

Image segmentation is a challenging task and there is currently no comprehensive theory in this field, not least because a given segmentation is often aimed at a specific application. The methods based on

gradient [19, 4] can be disturbed by noise or textured patterns in the image, thus producing a contour where there is actually not. In addition, it is difficult to find a contour if two regions share similar colors or gray levels, or when a contour is present in a region with small gradient values. Moreover, the result of a segmentation depends on the applications and on the scale of the objects of interest. Even human based segmentations are characterized by a significant variability in terms of result.

1.2 Literature

A large amount of research has been conducted on image segmentation, based upon a variety of techniques including active contours, clustering, or region splitting or merging. These methods can be roughly classified into contour based methods and region based methods [4].

Contour based methods For contour based methods, contour detection algorithms are first used to obtain candidate contours, and one has to find a way to transform these into closed contours (see [13, 4]). A classical approach for performing contour based segmentation is for instance the watershed algorithm [19].

Region based methods For region based methods, the image is oversegmented into small regions which are then merged to obtain an actual segmentation. To that end, a classical approach consists in representing the oversegmented image by a region adjacency graph connecting nearby regions. In this case, we transform the image segmentation problem into a graph clustering problem. A vast amount of literature in the field of image processing is dedicated to algorithms based upon graph clustering. In 2000, Shi and Malik [17] proposed a novel graph-theoretic criterion to measure the effectiveness of an image partition, the normalized cut, and an efficient technique for minimizing this criterion based on a generalized eigenvalue problem. This algorithm extracts global impression and obtains good results on static images and motion sequences. In 2004, Felzenszwalb and Huttenlocher [9] introduced a graph-based image segmentation method based on pairwise region comparison. In their approach, pixels are merged according to their intensity differences across boundaries and between neighboring pixels within each region. The segmentation criteria are adaptively adjusted to take into account the variability in neighbor regions. These algorithms constitute the main approaches to perform graph-based segmentation.

Most of the time, the starting point of a region based methods consists in finding a way to build an over-segmentation of the image.

1.3 Ambitions

In previous work [6], we were able to notice that the quality of the initial oversegmentation, as measured by metrics including the boundary recall, the compactness and the undersegmentation error, significantly impacts the quality of the overall segmentation. Deep learning algorithms have recently been applied to segmentation tasks and currently represent the state-of-the art solution in this field [14, 12]. Nevertheless, the question of generating deep learning based over-segmentations has not been extensively studied yet, mostly due to the lack of proper training datasets for this particular task. The objective of this research is therefore to develop a deep learning based algorithm to generate a superpixel partition with improved metrics.

Most of the segmentation methods are based on color information of pixels in the image, not taking into account all the human knowledge that one uses to understand what he sees. Implementing this knowledge would require a lot of computational time and a huge database, which does not currently exist. A neural network overcomes this issue, modeling this knowledge from a dataset of labeled images.

[?] petit résumé de quelques lignes du contenu de l'article et annonce le plan.

2 Background on superpixels segmentation

2.1 Superpixels

Superpixel algorithms are a class of techniques that partition an image into several small groups of pixels that share the same properties. Such a process highly reduces the number of characteristics of an image, as each superpixel is composed of hundreds of pixels, which leads to the amount of calculation being reduced for a further processing. In addition, as the pixels of a superpixel share similar attributes, they constitute regions on which it is relevant to compute features – mean color, mean texture.

Figure 2: *Example of a superpixel segmentation. From left to right: original image, original image with its calculated superpixels outlines and resulting superpixel segmentation. Each superpixel is affected the mean color of the original image over the superpixel region.*

Thus, superpixel algorithms are often considered as a pre-processing step in a number of applications. They can be used for depth estimation [21] or make the objects features more understandable in object classification. In addition, computing the superpixel partition of an image can constitute a first processing step to obtain an actual segmentation for this image [10], reducing the complexity of finding the different regions of an image by grouping at first similar pixels.

2.2 A good superpixel segmentation

Before even starting building one, we have to define what is a "good" superpixel segmentation algorithm. As shown in [18], it is a very ambiguous task as many relevant criteria exist. Several metrics, though, are commonly chosen in the literature to evaluate whether a superpixel algorithm gives good results or not.

Let $G = \{G_i\}_i$ and $S = \{S_j\}_j$ be partitions of the same image $I : x_n \mapsto I(x_n)$, $1 \leq n \leq N$. G is the ground truth segmented image[1] and S is the segmented image obtained from a superpixel algorithm.

- **Boundary Recall.** The first criterion we want the algorithm to respect is quite logically to detect most of the ground truth's outlines. Boundary recall measures the intersection between the outlines of the segmented image and the ones of the ground truth image. Before the computation, the segmented image outlines are dilated by a square of side 5 pixels. As such, boundary recall indicates the proportion of real boundaries being detected, with a tolerance margin of a few pixels. If $\mathrm{D}(G, \tilde{S})$ is the number of detected boundary pixels and $\mathrm{UD}(G, \tilde{S})$ the number of undetected boundary pixels in the segmented image S, then the *boundary recall* is:

$$\mathrm{Rec}(G, S) = \frac{\mathrm{D}(G, \tilde{S})}{\mathrm{D}(G, \tilde{S}) + \mathrm{UD}(G, \tilde{S})} \in [0, 1]$$

[1] ground truth segmented or superpixel segmented? [**?**]

$\tilde{S}$ being the segmented image with its boundaries dilated by a square of side 5 pixels. Please note that boundary recall does not measure the regularity of the boundaries at all. That means an algorithm can have a very high boundary recall while being very tortuous. This nourishes the need of a metric that quantifies the regularity of the boundaries.

- **Compactness.** In order to simplify the superpixel segmented image as much as possible, its superpixels need to be smooth and regular. We thus want to build a criterion that computes the ratio of the region area $A(S_j)$ with respect to a circle with the same perimeter as the superpixel S_j, weighted by the ratio of pixel numbers inside the region [16]:

$$\mathrm{Co}(G,S) = \frac{1}{N}\sum_{S_j} |S_j| \frac{4\pi A(S_j)}{P(S_j)^2}$$

  As such, a high compactness tends to indicate regular and little tortuous contours.

- **Undersegmentation Error.** Undersegmentation Error measures the "leakage" of the superpixels over the ground truth. We adopt the formulation proposed by Achanta et al. in [2]:

$$\mathrm{UE}(G,S) = \frac{1}{N}\left[\sum_{G_i}\left(\sum_{S_j|S_j\cap G_i>B}|S_j|\right) - N\right]$$

  Given a region G_i from the ground truth segmentation, $\{S_j|S_j \cap G_i > B\}$ is the set of superpixels S_j leaking across the boundaries of G_i, B being a tolerance margin — here, the minimum number of pixels is 5 percent of $|S_j|$. Superpixels that do not fit the ground truth result in a high value of UE.

2.3 Algorithms

2.3.1 SLIC

The Simple Linear Iterative Clustering (SLIC) algorithm, introduced by Achanta et al. [2, 3], is a region based method and is ranked among the most used algorithms for superpixel segmentation. It constructs a superpixel partition by applying a local K-means clustering algorithm on the image. During initialization, K cluster centers locations are selected on the image. Each pixel is then associated to the closest cluster center in the image according to a distance involving the color proximity and the physical distance between the pixel and the seed. The distance between the previous and the new locations is used to compute a residual error E, and the procedure is iterated until the error E converges.

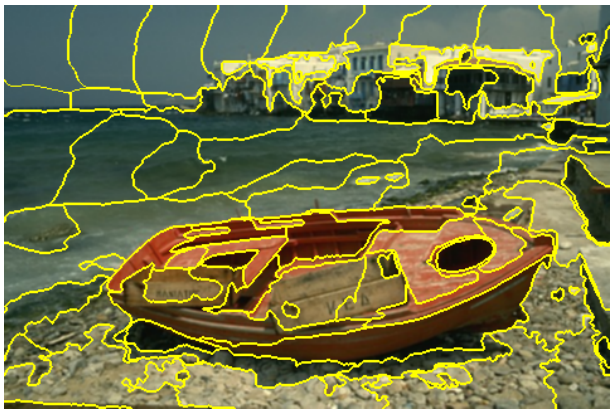


Figure 3: *Illustration of the SLIC superpixel segmentation of an image of the BSDS500.*

2.3.2 Eikonal

An algorithm based upon the Eikonal equation has recently been proposed for generating superpixels. The algorithm draws an analogy between waves propagating in an heterogeneous medium and regions growing on an image at a rate depending on the local color and texture. The Eikonal equation is a non-linear partial differential equation describing the propagation of waves in a domain $\Omega \subset \mathbb{R}^2$. At each point x of the domain, the local velocity is denoted $F(x)$. For all x in Ω, Eikonal equation reads:

$$\begin{cases} ||\nabla U(x)|| = \dfrac{1}{F(x)} & \forall x \in \Omega \\ U(x) = 0 & \forall x \in \partial\Omega, \end{cases} \tag{1}$$

where $\partial\Omega$ denotes the boundary of the domaine Ω. The solution $U(x)$ to Eikonal equation can conveniently be interpreted as the minimal traveling time required for the wave to travel from $\partial\Omega$ to point x.

In what follows, a pixel in image $\mathscr{I}$ is denoted by p and its coordinates by (x, y). $\mathbf{C}(p)$ denotes the color at pixel p in the CIELAB color space. The Eikonal algorithm is initialized by selecting N *seeds* or *cluster centers* locations $\{s_1, s_2, ..., s_N\}$. Then, a velocity field $F_i(p)$ is associated to each seed s_i depending on the color and the texture of both the seed and the pixel location $p := (x, y)$. The labels of the seeds are then gradually propagated from the labeled pixels to the unlabeled pixels according to the local velocities $\{F_i(x)\}_{i=1,..,N}$. On the domain defined by the image, Eikonal equation therefore reads

$$\begin{cases} ||\nabla U(p)|| = \dfrac{1}{F(p)} & \forall p \in I \\ U(p) = 0 & \forall p \in \Gamma. \end{cases} \tag{2}$$

Computing the propagation associated to (2) enables to obtain a superpixel partition of $\mathscr{I}$. Starting from several pixel seeds on the image, we associate a wave to each one. The speed of the wave associated to a pixel is high if it arrives on a pixel that is similar to the original seed, and low otherwise. Wen two wavefronts meet, a contour is generated.

3 Proposed approach and architecture

For our model to train, we build a superpixel segmentation dataset using the COCO dataset and a version of the Eikonal-based superpixel algorithm [7], applied on an image of the dataset (three color channels) and a segmentation mask (one channel). By considering a velocity function in (2) putting a significant weight on the segmentation mask, it is possible to construct a superpixel of the image which respects the boundaries of the segmentation mask.

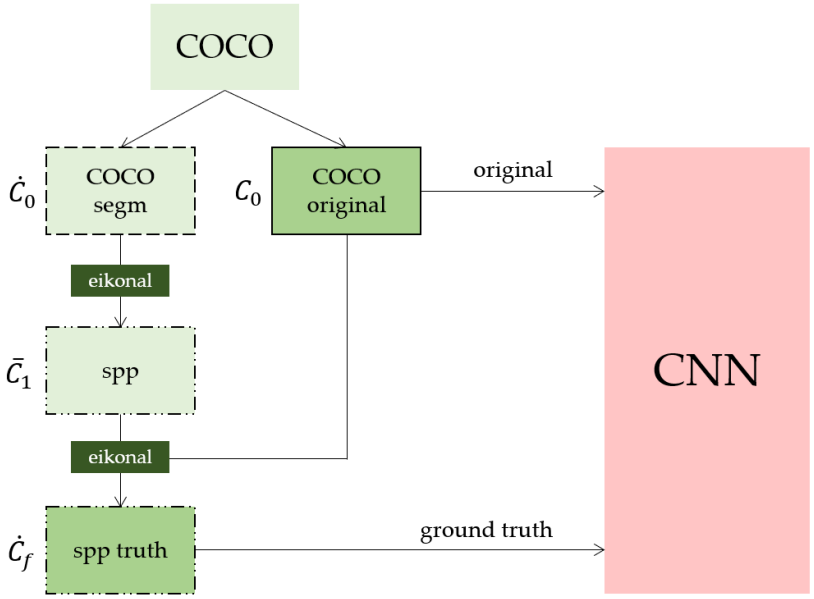


Figure 4: *Training method. The dash type indicates if the element is an image (plain border), a segmented image (dash) or a superpixel segmented image (dash and dots).*

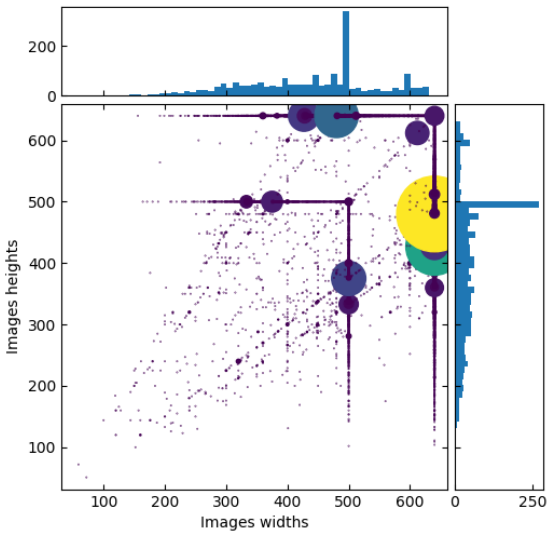
We denote as $\dot{S}$ a segmented image with regions of *the same color* $(0, 255, 255)$ and as $\bar{S}$ a segmented image with regions of *the same label* (0, 1, 2 ...). The training of the convolutional neural network runs as follows:

1. from the COCO dataset, extract the semantic segmented image $\dot{C}_0$ and transform it into a superpixel segmented image $\bar{C}_1$ with the Eikonal algorithm;
2. from $\bar{C}_1$, compute the $\dot{C}_f$ image by assigning to each superpixel the mean color of its pixel on the original image C_0. The idea is that such a transformed image, relatively close to the original image, seems easier to learn by a network than a labeled image such as $\bar{C}_1$;
3. use $(G, S) = (C_0, \dot{C}_f)$ as an training input to the neural network.

3.1 Dataset generation

The training of the neural network was performed on a modified version of the COCO dataset [1]. This dataset gathers images of complex everyday scenes containing common objects in their natural context and can be used for object detection, segmentation and captioning. In order to have a better understanding of this dataset, we can analyze its images (Figure 5). Hence the necessity to pre-process the dataset in order to standardize the input images' size.

The COCO dataset contains a huge number of images, but its segmentations are not very qualitative. As it has been labeled by hand in an approximative way, the boundaries of its segmented images are often imprecise (Figure 6). We subsequently use the Eikonal algorithm[**?**] to process the images.



(a) *COCO dataset images: widths, heights, and number of images for each dimension*

Training		Height	Width
Images	Min	51	59
118 287	Max	640	640

(b) *Characteristics of the COCO dataset*

Figure 5: *Training set characterization*

Figure 6: *Lack of precision of the COCO dataset segmentations*

3.2 Model

3.2.1 Network architecture

Context Aggregation Network The primary architecture of our network is the Context Aggregation Network (CAN), introduced in 2015 [20]. It gradually aggregates contextual information without losing resolution through the use of dilated convolutions, whose field of view increases exponentially over the network layers. This exponential growth grants a global information aggregation with a compact structure [20, 8].

The input data goes through the set of layers $\{L^0, \cdots, L^d\}$, and we choose $d = 7$ (*cf.* 4.2, page 10); the input and the output are images, so they have 3 feature maps.

	L^1		L^2		L^6		$L^d = L^7$	
$m \times n \times 3$	$\longrightarrow$	$m \times n \times 24$	$\longrightarrow$	$\ldots$	$\longrightarrow$	$m \times n \times 24$	$\longrightarrow$	$m \times n \times 3$

Table 1: *Layers of the network*

Each block L^s, $s \in [\![2, d-2]\!]$ is made of a *dilated convolution*, with parameter $r_s = 2^s$, an *adaptive batch normalization*, and a *leaky rectifier (ReLU)*, so that the content of an intermediate layer L^s can be

computed from the content of the previous layer L^{s-1}:

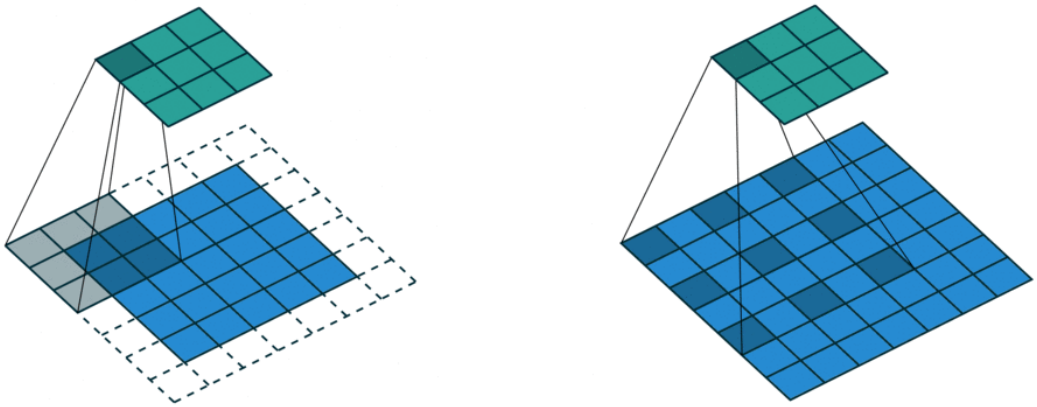
$$L_i^s = \Phi\left(\Psi^s\left(b_i^s + \sum_j L_j^{s-1} *_{r_s} K_{i,j}^s\right)\right). \tag{3}$$

with

$$L_j^{s-1} *_{r_s} K_{i,j}^s = \sum_{a+r_s b=x} L_j^{s-1}(a) K_{i,j}^s(b) \tag{4}$$

where Φ, Ψ^s, $K_{i,j}^s$ and the evoked layers are defined below.

Dilated convolution We consider a layer $L = (L_j)_{j\in[\![1,w]\!]}$, w being the number of feature maps L_j of L. We also consider $K = (K_{i,j})_{i,j}$, each $K_{i,j}$ being a 3×3 convolutional kernel. The dilated convolution operation of $K_{i,j}$ on L_j is denoted by $L_j *_r K_{i,j}$, r being the dilation parameter.



(a) *A simple convolution* ($r = 1$) **(b)** *A dilated convolution* ($r = 2$)

Figure 7: *Illustration of two types of convolutions*

The output $C(x)$ of a pixel x is:

$$\begin{aligned} C(x) :&= (L_j *_r K_{i,j})(x) \\ &= \sum_{a+rb=x} L_j(a) K_{i,j}(b) \\ &= \sum_b L_j(x - rb) K_{i,j}(b) \end{aligned}$$

and we recognize the simple convolution when $r = 1$. A dilated convolution enables the network getting larger receptive fields — which allows the network to aggregate information at increasing scales on the image — while preserving the input resolution [20].

Adaptive Batch Normalization (ABN) Distinct images in the COCO dataset have very different color ranges; in order to maximize the network performances, we normalize each batch at each layer of the network, so that the images of the batch have similar color ranges. The network can thereby consider a batch as a whole. We define the *adaptive normalization function* Ψ as [8]:

$$\Psi(x) = a\ x + b\ BN(x),$$

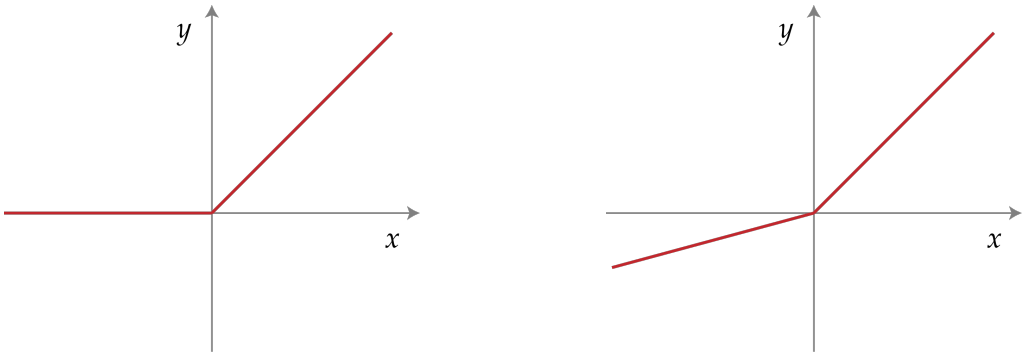
where BN is the classic PyTorch batch normalization, defined as:

$$BN(x) = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} * \gamma + \beta.$$

As such, Ψ combines identity mapping and batch normalization. The a, b, γ and β parameters are learned by backpropagation. It allows the model to choose to either give importance to the identity term or to the normalization term. Because it keeps the weights close to 0, batch normalization is also a way to regularize the network [11] such that it is easier to generalize, and it is thus unnecessary to use dropout to mitigate overfitting.

Leaky rectifier (LReLU) In order to let our neural network model complex patterns in the data, we have to add a non-linear property to the model. It often is an activation function, such as a sigmoid or a tanh, but these activation functions are bounded and their gradient is very low on the edges. Because we are going to manipulate high scalar values, we have to use an unbounded activation function, such as ReLU, $\Phi(x) = \max(0, x)$ (Figure 8a). But the issue with ReLU is that all the negative values become zero immediately, which decreases the ability of our model to train from the data. Hence the implementation of a *leaky rectifier*, LReLU (Figure 8b):

$$\Phi(x) = \max(\alpha x, x), \text{ with } 0 < \alpha < 1.$$



(a) *ReLU* **(b)** *LReLU* ($\alpha = 0.2$)

Figure 8: *Illustration of two unbounded rectifiers*

By implementing a Leaky Rectifier, we are able to take into account the negative valued pixels.

Layer L_s		1	2	3	4	5	6	7
Conv	Input w_s	3	24	24	24	24	24	24
	Output w_{s+1}	24	24	24	24	24	24	3
	Receptive field	3×3	3×3	3×3	3×3	3×3	3×3	1×1
	Dilation r_s	1	2	4	8	16	1	1
	Padding	1	2	4	8	16	1	0
ABN		Yes	Yes	Yes	Yes	Yes	Yes	Yes
LReLU		0.2	0.2	0.2	0.2	0.2	0.2	No

Table 2: *Our Context Aggregation Network*

3.2.2 Total Variation Loss

In such a search for well-segmented images, two criteria have to be fulfilled. The output needs to be as close as possible to the ground truth image; but we also need the segmented image to present a lot of zones where the color gradient $\nabla f(I)$ is equal to 0. Thus, we want to implement a train loss function that could help us satisfy these two criteria. In order to grant an improvement in the output image

smoothness, we use the Total Variation (TV) loss, often used in noise removal algorithms [15, 5]:

$$L_{TV} = \frac{1}{N}\sum_{i=1}^{N} |\hat{f}(I)_i - f(I)_i|^2 + \alpha_{TV}\frac{1}{N}\sum_{i=1}^{N} |(\nabla f(I))_i|^2$$

where α_{TV} is a hyperparameter that is going to be tuned and that allow us to give more or less importance to the gradient term.

4 Experiments and results

4.1 Implementation

The network was implemented with PyTorch[2]. With a batch size of 32, the training of a model was quite long on a GPU, mainly due to the size of the training dataset and the complexity of our model. In further sections we discuss how the hyperparameters impact the model's performances — metrics and temporal efficiency — and we conduct experiments to find a good-performing architecture. We found that the following values worked well on the BSD dataset:

batch_size	epochs	d	lr_0	decay for lr_0	α_{TV}
32	80	7	10^{-2}	10^{-3} after 10 ep.	0

Table 3: *Hyperparameter values*

4.2 Hyperparameter tuning

Learning rate At first, we tested several values for the learning rate lr. But either it was too low, and the loss would stagnate over the epochs, either it was too high and the loss wouldn't decrease (Figure 9b). We thus want to change the learning rate over the epochs and we allow it to decrease until it reaches a threshold value.

Introducing a decay and a threshold broadly gives the same results as switching the learning rate only once, after a dozen epochs. It thus seems relevant to start from a learning rate of 10^{-2}, which allows the network to adapt quickly at first, and then to decrease it to 10^{-3} after a dozen epochs. Decreasing the learning rate below this value gives negligible results and essentially only slows down the model.

Network size *d* In order to tune our network size, we fix the learning rate at 10^{-2}, we divide it by 2 every 2 epochs and we threshold at 10^{-4}. We do not use the TV-regularization.

As expected, the higher d is, the lower the loss function becomes. We indeed add a convolution layer to our model each time d is incremented. However, this addition has a cost; while improving the precision results of our network, we increase its computation time. As between $d = 6$ and $d = 7$, the improvement seems relatively low, we choose to keep 7 as a value for d, which seems a good compromise between computation time and precision performances.

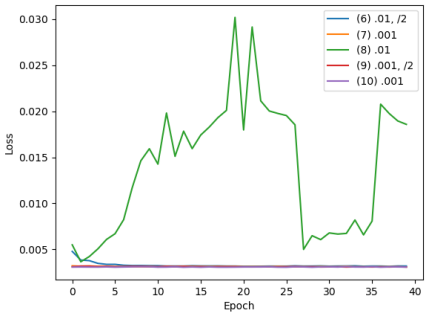
Number of epochs After a hundred epochs of training, the validation loss levels while the training loss keeps decreasing. We thus trained our models over 80 epochs. After each training, we chose the network weights that were providing the best validation loss results.

TV regularization experience [?]
interpretation [?]

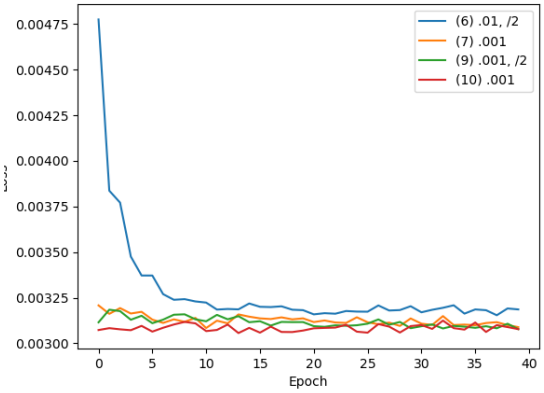
[2]Repository can be found at `https://github.com/theodumont/superpixels-segmentation`.

run id	lr_0	decay	threshold	started from
6	10^{-2}	$\times.5$ every 2 ep.	10^{-4}	run 0 ep.5
7	10^{-3}	No		run 0 ep.40
8	10^{-2}	No		run 0 ep.40
9	10^{-3}	$\times.5$ every 2 ep.	10^{-4}	run 0 ep. 40
10	10^{-3}	No		run 9 ep.40

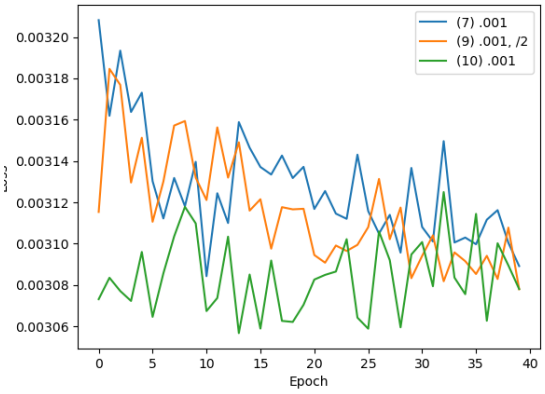
(a) Runs for learning rate tuning, with $d = 7$



(b) *Runs 6, 7, 8, 9 and 10*



(c) *Runs 6, 7, 9 and 10*



(d) *Runs 7, 9 and 10*

Figure 9: *Tuning of learning rate: validation loss for different runs*

d	4	5	6	7	8
loss$\times 10^3$	3.46	3.18	3.12	3.11	???

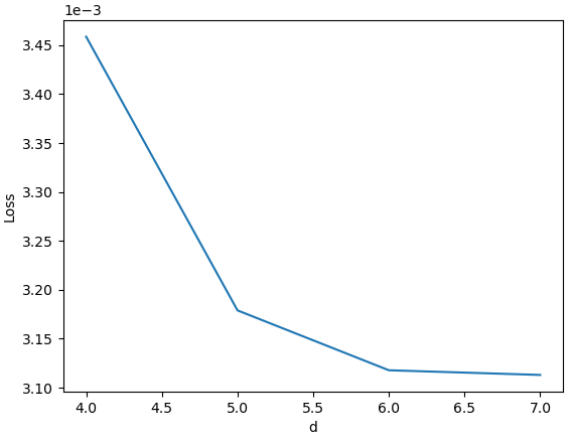


Figure 10: *Tuning of network size: loss values on the validation set*

α_{TV}					
loss$\times 10^3$					

Figure 11: *Tuning of α_{TV}: loss values on the validation set*

4.3 Results on dataset

To assess its performances, we evaluated our model on the Berkeley Segmentation Dataset 500 (BSDS500)[4]. It only contains 500 images but provides very qualitative ground truth manual segmentations for each image. We examine the three metrics that we previously defined (2.2): boundary recall, undersegmentation and compactness.

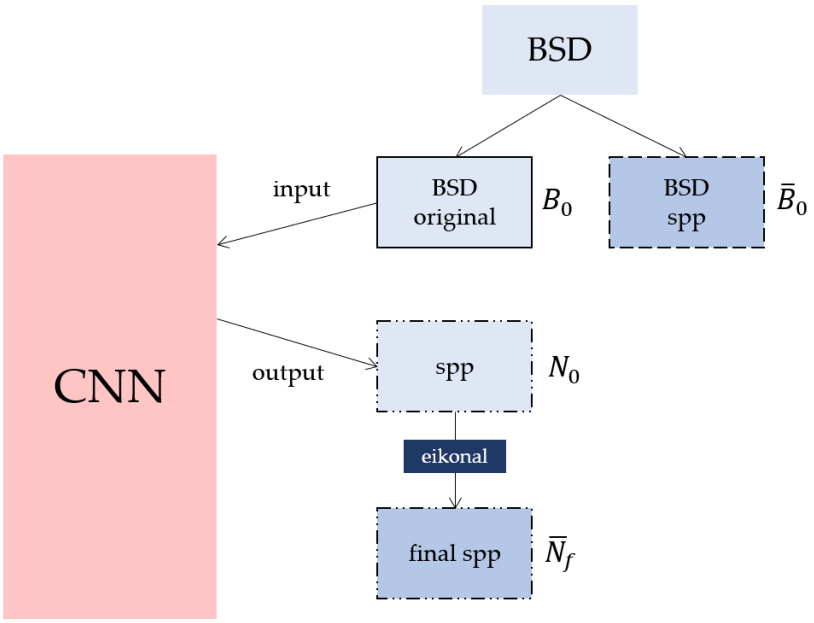


Figure 12: *Testing method. The dash type indicates if the element is an image (plain border), a segmented image (dash) or a superpixel segmented image (dash and dots).*

To test the network on the BSD dataset:

1. from the BSD dataset, extract the original image B_0 and give it to the neural network;
2. get the neural network output N_0, which is a nearly segmented image;
3. use the Eikonal algorithm to obtain the final superpixel segmented image $\bar{N}_f$, with superpixel labels, on which we compute the metrics by comparing it to the ground truth $\bar{B}_0$.

[?] Here are evaluated the metrics for some superpixel segmentation algorithms: the watershed algorithm (WS) [19], eikonal (EI) and eikonal with textures (EIT) [7]. [?]

Algorithm		BR	UE	CO
image analysis	EI			
	EIT			
	WS			
neural networks	SLIC	0.90	0.05	0.54
	Ours	0.88	0.04	0.77

Table 4: Comparisons of metrics on the BSDS500 dataset for different superpixel segmentation algorithms

We use the SLIC algorithm as a reference to evaluate the performances of our model. It yields very good results: the undersegmentation sees a 0.01 improvement, and the compactness is way better

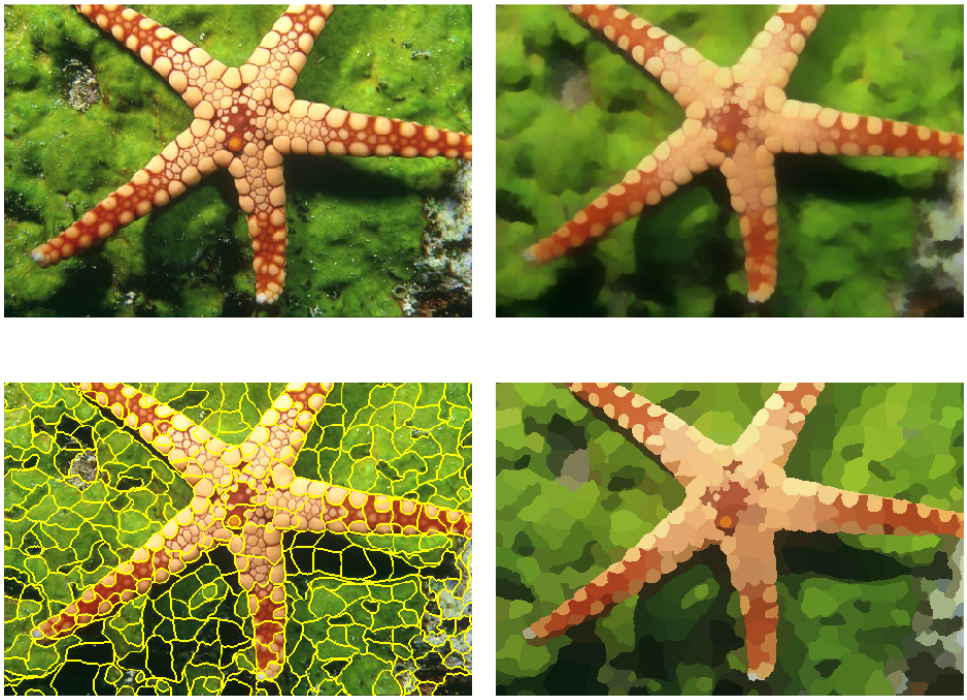


Figure 13: *Application of the model to an image of the BSD500. From left to right, up to down: original image, output image of the neural network, superpixel segmentation of the output image overlaying on original image, and superpixel segmented image with each superpixel being displayed with the average color of the pixels belonging to it.*

(improvement of 0.23). The boundary recall is slightly smaller for our model than for the SLIC algorithm, but this is not a problem as the SLIC compactness is very low. The contours oscillate and thus intersect more with the ground truth image outlines.

5 Discussion and conclusion

In this article, we presented an algorithm based upon convolutional neural networks for generating superpixel partitions of images. This work indicates that convolutional neural networks can achieve good performance on superpixel segmentation. Because of the complexity of both the model and the problem we are trying to resolve, we were able to conduct little experiments for the tuning of our hyperparameters. It would thus be interesting to push the study further, notably by conducting additional tests for a better choice of our parameters. We ran some experiments on the implementation of a U-Net [14] network, but only have a few results yet. We would like to continue testing this network, and try to mix it with the CAN, for instance adding only one U-Net layer on the CAN. Furthermore, testing our model on other datasets would show how well it generalizes.

Whereas at first glance implementing a TV-loss regularization could totally have yielded better results, it does not improve the model. We tried to analyze why such an initiative didn't pay off. Actually, adding the gradient term in the loss function is enjoining the network to reduce the overall gradient function of the image. Nevertheless, while a superpixel segmented image has a lot a regions where its gradient is 0, it also has a very high gradient on the objects contours. Although the results obtained are satisfying without a TV-regularization, we could pre-process each image by dividing it into several smaller ones to reduce the amount of outlines in each image the neural network is going to process. However, the COCO dataset is not suitable for this task as its images have a very low resolution. Cutting them would result in losing a lot of global information and thus making the use of a CAN meaningless.

Special thanks

This work would not have been possible without the precious help of Mr. Bruno Figliuzzi, who guided me throughout this project and provided me with judicious advice. He helped me to understand the different challenges in image analysis and in research in general better, and I would like to thank him very much for that.

References

[1] Microsoft coco: Common objects in context. 2014.

[2] Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurélien Lucchi, Pascal Fua, and Sabine Susstrunk. Slic superpixels compared to state-of-the-art superpixel methods. *IEEE transactions on pattern analysis and machine intelligence*, 34, 05 2012.

[3] Radhakrishna Achanta and Sabine Susstrunk. Superpixels and polygons using simple non-iterative clustering. pages 4895–4904, 07 2017.

[4] P. Arbeláez, M. Maire, C. Fowlkes, and J. Malik. Contour detection and hierarchical image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(5):898–916, may 2011.

[5] Antonin Chambolle. An algorithm for total variation minimization and applications. *J. Math. Imaging Vis.*, 20(1–2):89–97, 2004.

[6] Kaïwen Chang. *Machine learning based image segmentation.* PhD thesis, Université de recherche Paris Sciences et Lettres, 2019.

[7] Kaiwen Chang and Bruno Figliuzzi. Fast Marching Based Superpixels Generation. pages 350–361, May 2019.

[8] Qifeng Chen, Jia Xu, and Vladlen Koltun. Fast image processing with fully-convolutional networks. 2017.

[9] P. F. Felzenszwalb and D. P. Huttenlocher. Efficient graph-based image segmentation. *International journal of computer vision*, 59(2):167–181, 2004.

[10] Brian Fulkerson, Andrea Vedaldi, and Stefano Soatto. Class segmentation and object localization with superpixel neighborhoods. *In Computer Vision*, pages 670—-677, 2009.

[11] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[12] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3431–3440, June 2015.

[13] X. Ren, C. C. Fowlkes, and J. Malik. Scale-invariant contour completion using conditional random fields. In *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*, volume 2, pages 1214–1221, Oct 2005.

[14] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. 2015.

[15] L I Rudin, S Osher, and E Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D: Nonlinear Phenomena*, 60(1-4):259–268, 1992.

[16] Alexander Schick, Mika Fischer, and Rainer Stiefelhagen. Measuring and evaluating the compactness of superpixels. *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*, pages 930–934, 2012.

[17] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on pattern analysis and machine intelligence*, 22(8):888–905, 2000.

[18] David Stutz, Alexander Hermans, and Bastian Leibe. Superpixels: An evaluation of the state-of-the-art. *Computer Vision and Image Understanding*, April 2017.

[19] L. Vincent and P. Soille. Watersheds in digital spaces: an efficient algorithm based on immersion simulations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(6):583–598, June 1991.

[20] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. 2015.

[21] C. Lawrence Zitnick and Sing Bing Kang. Stereo for image-based rendering using image over-segmentation. *International Journal of Computer Vision*, 75(1):49–65, 2007.

Appendices

Experiments

[?]

id	d	lr_0	decay?	thresh.?	α_{TV}	comments	conclusion
3	4	10^{-2}	$\times.5$ every 2 ep.	10^{-4}	0		
4	5	10^{-2}	$\times.5$ every 2 ep.	10^{-4}	0		
5	6	10^{-2}	$\times.5$ every 2 ep.	10^{-4}	0		$d = 7$
0	7	10^{-2}	$\times.5$ every 2 ep.	10^{-4}	0		
11	8	10^{-2}	10^{-3} after 10 ep.	10^{-4}	0		
6	7	10^{-2}	$\times.5$ every 2 ep.	10^{-4}	0	from run 0 ep.5	
7	7	10^{-3}	No		0	from run 0 ep.40	$lr_0 = 10^{-2}$,
8	7	10^{-2}	No		0	from run 0 ep.40	then 10^{-3}
9	7	10^{-3}	$\times.5$ every 2 ep.	10^{-4}	0	from run 0 ep. 40	after 10 ep.
10	7	10^{-3}	No		0	from run 9 ep.40	
12							
13							
14							

Table 5: *All the runs. The batch size is 32.*