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.

To do

- rédiger paragraphe eikonal
- rédiger Unet
- implementation: quel nb d'epochs?
- hpp
  - nb of epochs
  - TV loss
- en annexe : tableaux de tous les runs
- results on dataset
- conclusion

1 Introduction

1.1 Segmentation

While looking at an image, the human brain uses a lot of knowledge to understand its content. But as for a computer, an image only is a set of integer valued pixels, making it produce an interpretation is much more complex. Segmenting an image consists in transforming the image into something that is easier to analyze, and much more meaningful. By partitioning an image into multiple sets of pixels, it

assigns a label to every pixel in it such that connected groups of pixels with the same label share certain characteristics. It allows one to locate the objects on an image, pointing 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: *A segmentation. To each segmented region of the image was affected the mean value of its pixels. Both images are from the COCO training dataset (REF?).*

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 can be disturbed by noise or textured patterns in the image, thus producing a contour where 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 corresponds to small gradient. 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 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 of an image on which it is very relevant to compute features – mean color, mean texture.

Figure 2: *A superpixel segmentation. From left to right: the original image, the original image with its calculated superpixels outlines and the resulting superpixel segmented image. Each superpixel has only one color, 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 [14] or make the objects features more understandable in object classification. Finally, computing the superpixel partition of an image can constitute a first processing step to obtain an actual segmentation for this image [8], reducing the complexity of finding the different regions of an image by grouping at first similar pixels.

1.2.1 A good superpixel segmentation

Before even starting building one, we have to define what is a "good" superpixel segmentation algorithm. As it is shown in [?], 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:

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

  $\tilde{S}$ 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 [11]:

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

  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 Neubert and Protzel in [9]:

$$\mathrm{UE}_{NP}(G, S) = \frac{1}{N} \sum_{G_i} \sum_{S_j \cap G_i \neq \emptyset} \min\{|S_j \cap G_i|, |S_j - S_j \cap G_i|\}$$

  where S_j is a superpixel and G_i is a gound truth segment. We take the minimum of the number of overlapping pixels and the number of non overlapping pixels within S_j as the leakage.

1.2.2 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].

[1] ground truth segmented or superpixel segmented?

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 [10, 4]). A classical approach for performing contour based segmentation is for instance the watershed algorithm.

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 [12] 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 [7] 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.

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 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.

1.2.3 Ambitions

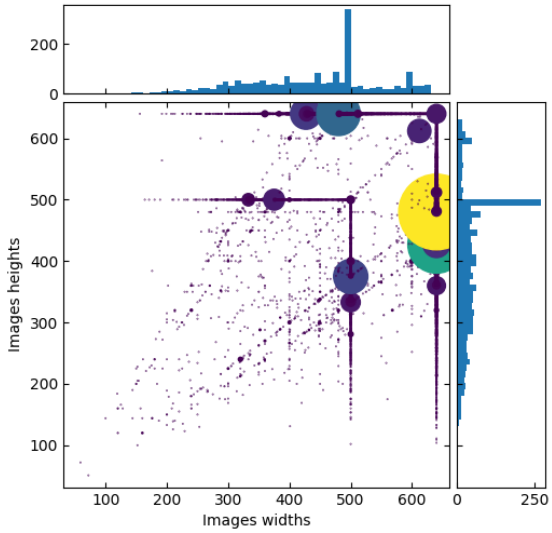
In previous work [5], we were able to notice that the quality of the initial oversegmentation, as measured by metrics including the boundary recall, the compactness and the undersegmentation error, significantly impacts the quality of the overall segmentation. The objective of this research is therefore 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.

2 Dataset generation

2.1 COCO dataset

The training of the neural network was performed on the COCO dataset [1][2]. 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 image (Figure 3). Hence the necessity to pre-process the dataset in order to standardize the input images' size.



(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 3: *Training set characterization*

The COCO dataset contains a huge number of images, but its segmentations are not very qualitative. A it has been labeled by hand in an approximative way, its segmented images lack quality and their boundaries are often imprecise (Figure 4). We subsequently introduce the Eikonal algorithm, which constitutes an important step in the process but that will not be fully detailed in this paper.

Figure 4: *Lack of precision of the COCO dataset segmentations*

[2] `http://cocodataset.org`

2.2 Eikonal

To do

2.3 Global approach

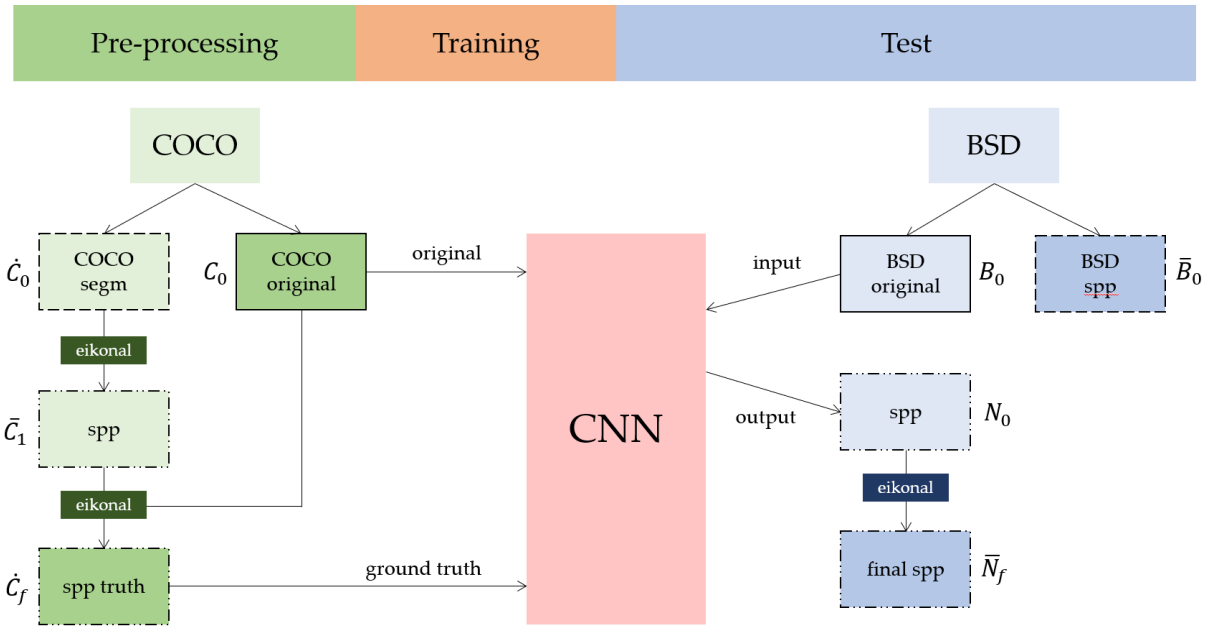


Figure 5: *Global approach. The dash type indicates if the element is an image (plain border), a segmented image (dash) or a superpixel segmented image (dash and dots).*

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* $(0, 1, 2 \dots)$.

Training 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.

Test To test the network on the BSD dataset (*cf.* Section **??**, page **??**):

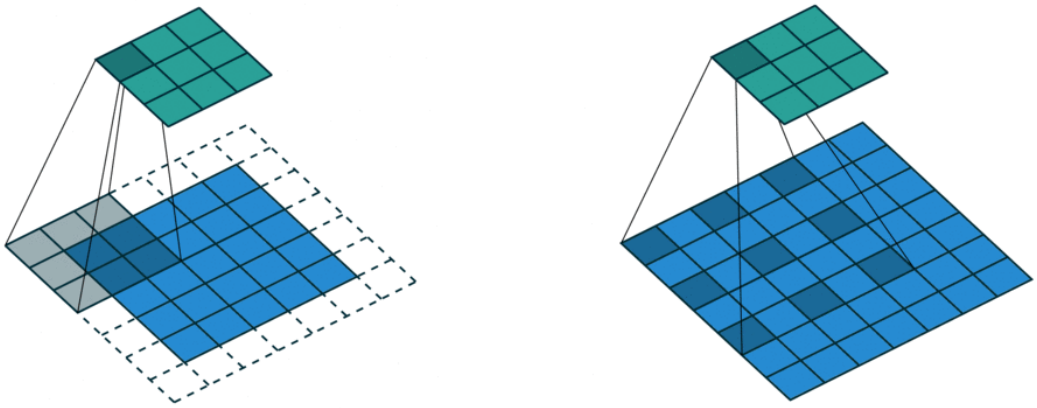
1. from the BSD dataset, extract the original image B_0 and give it to the NN;
2. get the NN output N_0, which is a nearly segmented image;
3. use the Eikonal algorithm to obtain the final superpixel segmented image $\bar{N}_f$, with superpixel labels, on which we compute the metrics by comparing it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

3 The model

3.1 Network architecture

3.1.1 Layers definitions

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 6: *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[3].

Adaptative Batch Normalization (ABN) Distinct images in the COCO dataset have very different color ranges; in order to maximize the network performances, we normalize the data. We define the *adaptative normalization function* Ψ as:

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

where BN is the classic batch normalization[4], defined as:

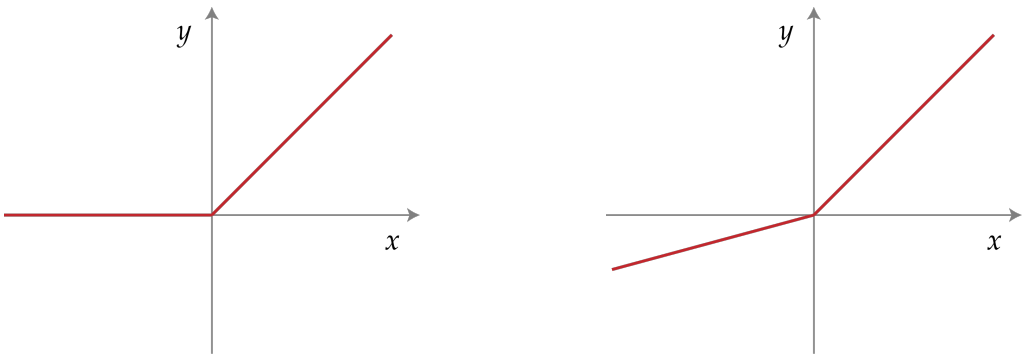
$$BN(x) = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} * \gamma + \beta.$$

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 adapt to each dataset, choosing whether or not giving a big importance to the identity term and the normalization term.

[3] ref ?
[4] reference ?

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 7a). 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 7b):

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



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

Figure 7: *Illustration of two unbounded rectifiers*

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

3.1.2 Context Aggregation Network

The primary architecture of our network is the Context Aggregation Network (CAN), introduced in 2015 [13]. It gradually aggregates contextual information without losing resolution through the use of dilated convolutions, whose parameter exponentially increases over the network layers. This exponential growth grants a global information aggregation with a compact structure [13, 6].

The input data goes through the set of layers $\{L^0, \cdots, L^d\}$, and we choose $d = 7$ (*cf.* **??**, page **??**); 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 *adaptative 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}:

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

with

$$L_j^{s-1} *_{r_s} K_{i,j}^s = \sum_{a + r_s b = x} L_j^{s-1}(a) K_{i,j}^s(b) \tag{2}$$

where Φ, Ψ^s and $K_{i,j}^s$ are defined in 3.1.1, page 7.

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*

3.1.3 UNet

To do

3.1.4 Chen + UNet

To do

3.2 Total Variation Loss

In such a search for well-segmented images, 2 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 grant an improvement in the output image smoothness, we use the Total Variation (TV) loss, defined by [**?**]:

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

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[5]. 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*

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

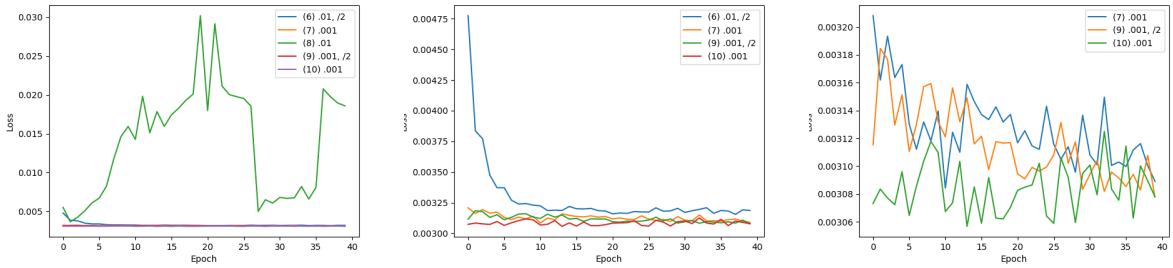
4.2 Hyperparameter tuning

4.2.1 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 8c). We thus want to change the learning rate over the epochs and we allow it to decrease until it reaches a threshold value.

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

Table 4: Runs for learning rate tuning, with $d = 7$



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

Figure 8: *Tuning of learning rate: validation loss for different runs*

After a hundred epochs, the validation loss levels while the training loss keep decreasing. It thus seems relevant to start from a learning rate of 10^{-2} and then to decrease it to 10^{-3} after a dozen epochs.

4.2.2 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.

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

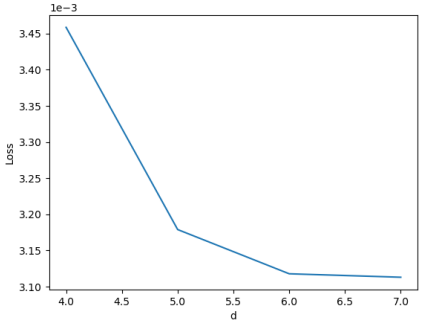


Figure 9: *Tuning of network size: loss values on the validation set*

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.

4.2.3 Number of epochs

nb epochs: on le sélectionne en prenant le minimum de la validation loss

4.2.4 TV regularization

check premiers runs attention changer valeur code PK MARCHE PAS ? - gradient fort sur outlines - comment faire ? découper images, COCO pas ouf pour ça, résolution basse

4.3 Results on dataset

cf results/images BSD dataset, BSD fait a la main et pas a l'arrache comme COCO Berkeley Segmentation Dataset 500 (BSDS500)[**?**], [3] bruno

Results Here are the previously defined metrics of some well-known superpixel segmentation algorithms.

Algorithm		BR	UE	CO
image analysis	EI [6] EIT [7] WS [8]			
neural networks	ref [9]	0.8995	0.0473	0.5409
	Ours	0.8781[10]	0.0388	0.7682

Table 5: Comparisons of metrics on the BSD dataset for different superpixel segmentation algorithms

We use the SLIC alorithm as a reference to evaluate the performances of our model.

5 Conclusion/Discussion

On a présenté un nouveau...
On a prouvé...
Il reste à faire...
relire tous les mails pour avoir toutes les infos sur performances etc

Special thanks

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

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

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

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

[9] Peer Neubert and Peter Protzel Chemnitz. Superpixel benchmark and comparison. In *Forum Bildverarbeitung 2010*, pages 205–218, 2012.

[10] 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.

[11] 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.

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

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

[14] 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

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 6: *All the runs. The batch size is 32 and the number of epochs is 40.*