# 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

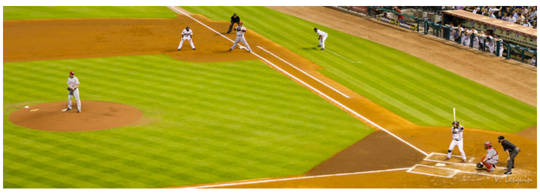
**Key-words:**
deep learning; convolutional neural networks; image segmentation

**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

The human eye can easily understand the content of an image, but making a computer interpret the disposition of its pixels is more complex. Segmenting an image is partitioning it into multiple sets of pixels. It transforms the image into something that is easier to analyze, and much more meaningful, assigning a label to every pixel in an image such that pixels with the same label share certain characteristics. It allows one to locate the objects on an image, pointing out their boundaries.



**(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?).*

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.

## 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, 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 [depth], make the objects features more understandable in object classification, and improve performances in image segmentation [img-segm], reducing the complexity of finding the different regions of an image by grouping at first similar pixels.

### 1.2.1 A good superpixel segmentation

**Metrics** Before even starting building one, we have to define what is a "good" superpixel segmentation algorithm. As it is shown in [stutz], 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 dilated outlines of the segmented image and the ones of the ground truth image. As so, it 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. Please note that boundary recall does not measure the regularity of the boundaries at all. That means an algorithm can have a very high

[1] ground truth segmented or superpixel segmented?

boundary recall while being very abrupt. 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 how close the area $A(S_j)$ of each superpixel $S_j$ is from a circle with same perimeter $P(S_j)$:

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

- **Undersegmentation Error.** Undersegmentation Error measures the "leakage" of the superpixels over the ground truth:

$$\mathrm{UE}(G,S) = \text{formula}$$

**SLIC** To do

#### 1.2.2 Ambitions

**Difficultés que l'on cherche à résoudre** To do

**Pas de vraie approche DL pour segmentation avec spps** To do
Article Bruno: Most of the aforementioned segmentation methods are based only on color information of pixels in the image. Humans use much more knowledge when performing image segmentation, but implementing this knowledge would cost considerable human engineering and computational time, and would require a huge domain knowledge database which does not currently exist. Trainable segmentation methods, such as neural network segmentation, overcome these issues by modeling the domain knowledge from a dataset of labeled pixels.

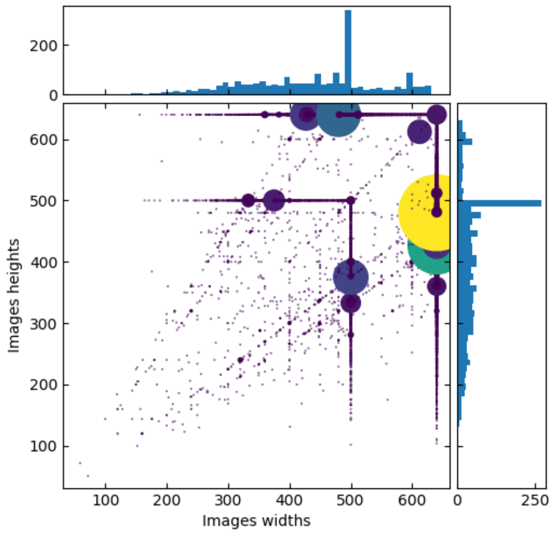
**Ambitions** To do
améliorer les métriques

# 2 Dataset generation

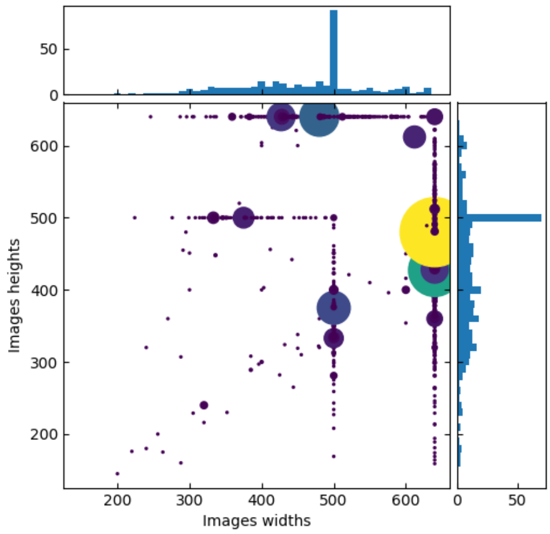
## 2.1 COCO dataset

COCO dataset[2], nb of images, examples

In order to have a better understanding of the dataset, we




(a) *plot*




(b) *plot*

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

(c) Tabular

| | Validation | Height | Width |
|---|---|---|---|
| Images | Min | 145 | 200 |
| 5000 | Max | 640 | 640 |

(d) Tabular

**Figure 3:** *Training and validation sets characterization*

Hence the necessity to pre-process the dataset in order to standardize the input images' size. Moreover, as the COCO dataset has been labeled by hand in an approximative way, its 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:** *Imprecise*

[2]site de COCO

## 2.2 Eikonal

To do

## 2.3 Global approach

**Figure 5:** *Global approach*

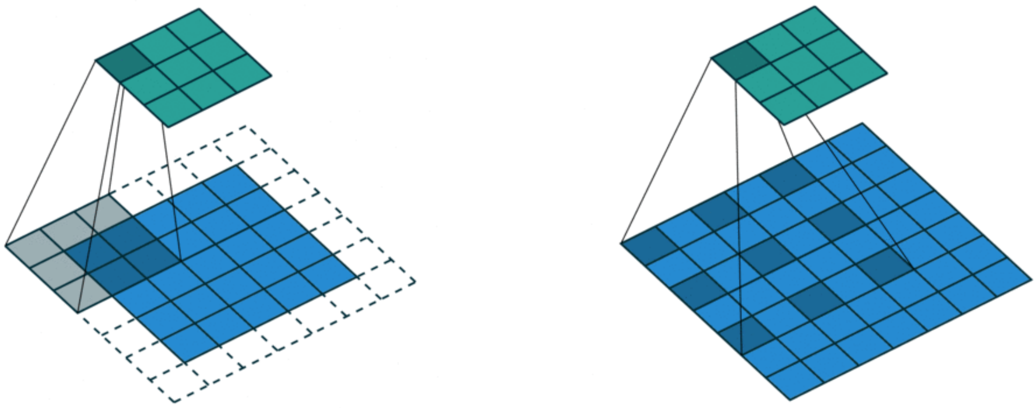
image originale -> CNN -> résultat du filtre dans eikonal -> superpixels sans couleurs + couleur moyenne pour chaque spp de l'image originale

# 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 $3 \times 3$ 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 while preserving the input resolution[3].

[3] ref ?

**Adaptative Batch Normalization (ABN)** As we have seen in (??), page **??**, we need to normalize the data. We define the *adaptative normalization function* $\Psi$ as:

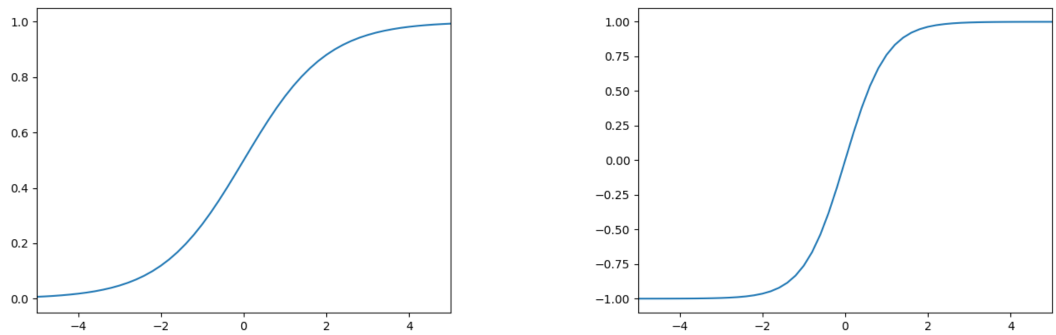
$$\Psi(x) = a\ x + b\ BN(x),$$

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, $\Psi$ combines identity mapping and batch normalization. $a$, $b$, $\gamma$ and $\beta$ are learned parameters[5] 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.

**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 (Figure 7).




**(a)** *Sigmoid* **(b)** *tanh*

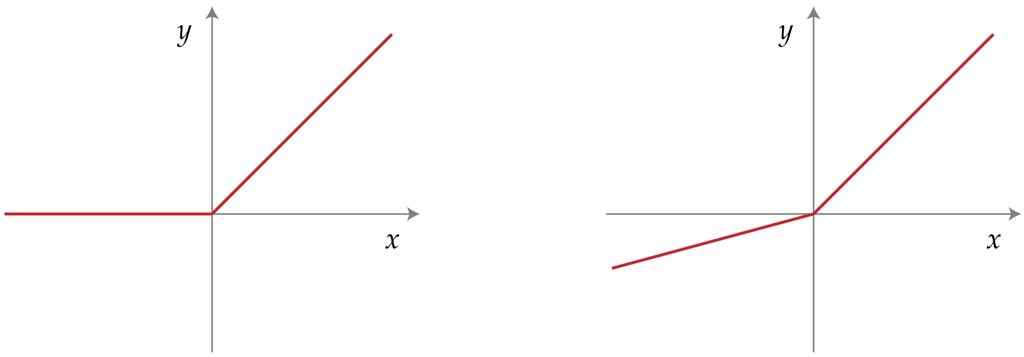
**Figure 7:** *Illustration of two bounded rectifiers*

these activation functions are often used but they 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):

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

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

[4]reference ?
[5]ref : `https://pytorch.org/docs/stable/_modules/torch/nn/modules/batchnorm.html`




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

**Figure 8:** *Illustration of two unbounded rectifiers*

### 3.1.2 Chen

**Context Aggregation Network (CAN)** [6] blabla sur le RGB en entrée, RGB en sortie I -> f(I)
To do

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| input $I$ | $\longrightarrow$ | $L^1$ | $\longrightarrow$ | $\cdots$ | $\longrightarrow$ | $L^s$ | $\longrightarrow$ | $\cdots$ | $\longrightarrow$ | output ($L^d$) |
| $m \times n \times 3$ | | $m \times n \times w_1$ | | | | $m \times n \times w_s$ | | | | $m \times n \times 3$ |

**Table 1:** *Layers*

**Architecture of a block** Each block $L_s$ is made of 3 layers:

1. A *dilated convolution*, with parameter $r_s = 2^s$,
2. An *adaptative batch normalization*,
3. A *leaky rectifier (ReLU)*,

so that the content of an intermediate layer $L^s$ can be computed from the content of the previous layer $L^{s-1}$:

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

where ... is ...
and

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

because of 3.1.1, page 5.

| Layer | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| Convolution | $3 \times 3$ | | | | | | |
| Dilation | 1 | | | | | | |
| Batch Normalization | Yes | | | | | | |
| LReLU | Yes | | | | | | |

**Table 2:** *Chen*

[6] reference

#### 3.1.3 UNet

To do

#### 3.1.4 Chen + UNet

To do

### 3.2 Total Variation (TV) Loss

#### 3.2.1 MSE

To do

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

#### 3.2.2 TV

[7] To do 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:

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

### 3.3 Implementation

The network was implemented with PyTorch[8] and we used GPU acceleration [...]
(pytorch), se renseigner (section assez courante)
GPU acceleraation

## 4 Expérience et résultats

### 4.1 Hyperparameters

petit bilan des valeurs choisies au final

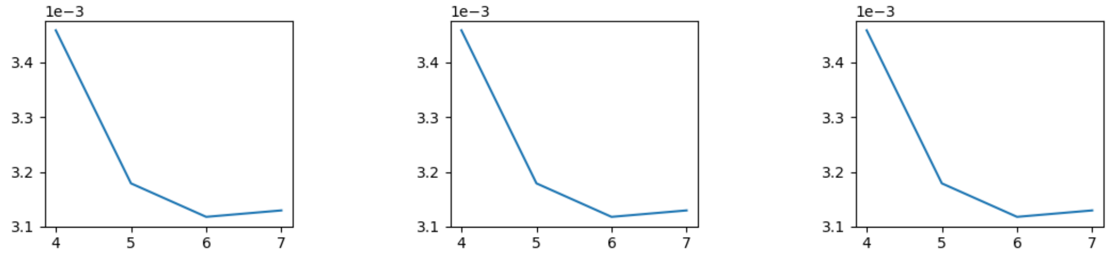
#### 4.1.1 Learning rate

| $lr_0$ | decay? | saturation? | $d$ | TV? |
|---|---|---|---|---|
| 0.001 | No | No | 7 | No |
| 0.01 | No | No | 7 | No |
| 0.01 | $\times 0.5$ every 2 epochs | $10^{-4}$ | 7 | No |
| 0.001 | $\times 0.5$ every 2 epochs | $10^{-4}$ | 7 | No |

**Table 3:** Runs for learning rate tuning

**Constant value** entrainement (lr, alpha) -> courbes de loss, et loss qui sature (cluster) d'où changement de lr au cours des epochs

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




(a) *Graph* (b) *Graph* (c) *Graph*

**Figure 9:** *Tuning of learning rate*

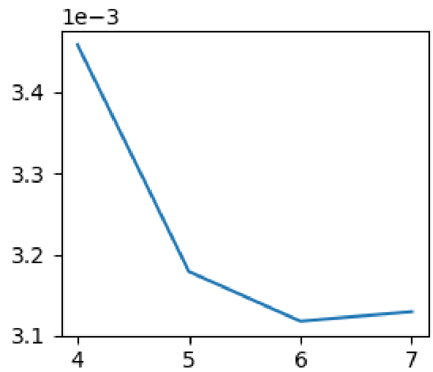
**Non constant value**

#### 4.1.2 Network size $d$

- Learning rate initialisé à 0.01, divisé par 2 toutes les 2 époques, saturation à 1e-4 - Pas de régularisation TV

| $d$ | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|
| loss$\times 10^3$ | 3.46 | 3.18 | 3.12 | 3.13 | ??? |

(a) *Loss values on validation set*




(b) *Graph*

**Figure 10:** *Tuning of network size*

CONCLUSION des run 3 à 5: Il est préférable de laisser d=7. Entre d=6 et d=7, l'amélioration semble relativement faible. bon intermédiaire entre temps de calcul et performances

#### 4.1.3 Number of epochs

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

#### 4.1.4 TV regularization

PK MARCHE PAS ? - gradient fort sur outlines - comment faire ? découper images, COCO pas ouf pour ça, résolution basse

#### 4.1.5 All the runs

a mettre en annexe, tableaux de tous les runs et graphes

### 4.2 Results on dataset

cf results/images BSD dataset, BSD fait a la main et pas a l'arrache comme COCO

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

| Algorithm | | BR | UE | CO |
|---|---|---|---|---|
| image analysis | EI [9] | | | |
| | EIT [10] | | | |
| | WS [11] | | | |
| neural networks | ref [12] | 0.8995 | 0.0473 | 0.5409 |
| | Ours | 0.8781[13] | 0.0388 | 0.7682 |

**Table 4:** 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

## Sources

[1] [1] C. Smith, J.C. Green, Titre de l'article, Titre du journal, 10 (2009) 55-72
[2] M. Truk, C. Bidul. Titre du bouquin, John Wiley and Sons, New York, 1973
[3] P. Machin, Titre de la thèse, Thèse, Université Poitiers, 1992
[4] D. Pierre, J.-P. Paul, B. Jacques, Titre communication, in: D. Editor, G. Editeur, ( éd.), Proceedings of Conference XXX , Publisher, Paris, France, 1995, pp. 3–6
[depth] 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.
[img-segm] Brian Fulkerson, Andrea Vedaldi, and Stefano Soatto. Class segmentation and object localization with superpixel neighborhoods. In Computer Vision, 2009 IEEE 12th International Conference on, pages 670–677. IEEE, 2009.
[stutz] David Stutz, Alexander Hermans, and Bastian Leibe. Superpixels: An Evaluation of the State-of-the-Art. Computer Vision and Image Understanding, April 2017.