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 study different options to improve the performance of a deep learning convolutional neural network

Contents

Todo

-

1 Introduction

1.1 Segmentation

What it is piche

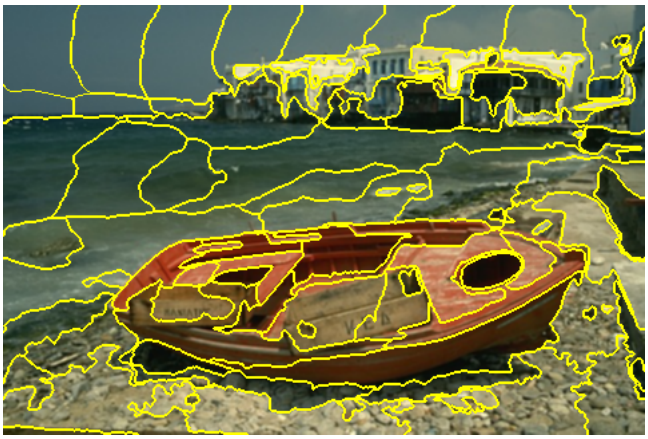


Figure 1: *An image and its segmented image*

Motivation

1.2 Superpixels

What it is piche

Figure 2: *Partition d'une image en superpixels*

Applications & Motivation démarrer une segmentation
fournir un support sur lequel faire de la classification (couleur/texture moyenne, etc)

1.3 Ce qu'est une bonne superpixelsegmentation

1.3.1 Metrics

metric1

metric1

metric1

metric1

1.3.2 SLIC

1.3.3 Autres algorithmes

1.4 Motivations/ambitions

Difficultés que l'on cherche à résoudre

Pas de vraie approche DL pour segmentation avec superpixels

Ambitions améliorer les métriques

2 Dataset generation

2.1 COCO dataset

2.1.1 The COCO dataset

COCO dataset

2.1.2 Characteristics

max, min pixels, etc graphes pour décrire les données

2.2 Eikonal

2.3 Notre utilisation de eikonal

en plus réutilisé derrière sur image qui sort du réseau
faire un petit résumé

3 The model

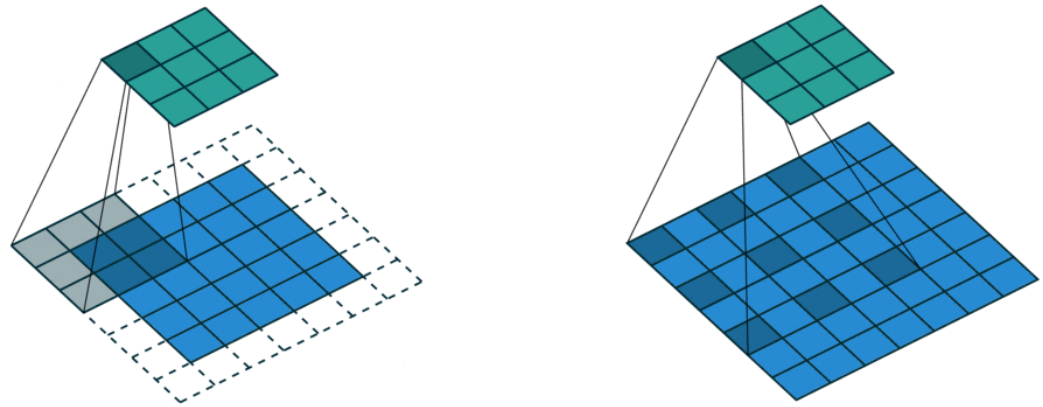
3.1 Approach

description générale de l'approche (NN puis eikonal)

3.2 Network architecture

3.2.1 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. The output $C(x)$ of a



(a) *A simple convolution* ($r = 0$) **(b)** *A dilated convolution* ($r = 1$)

Figure 3: *Illustration of two types of convolutions*

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 $r = 1$.

Adaptative Batch Normalization (ABN) As we have seen in (2.1.2), page 4, we need to 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[1], 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. a, b, γ and β are learned parameters[2] 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 4).

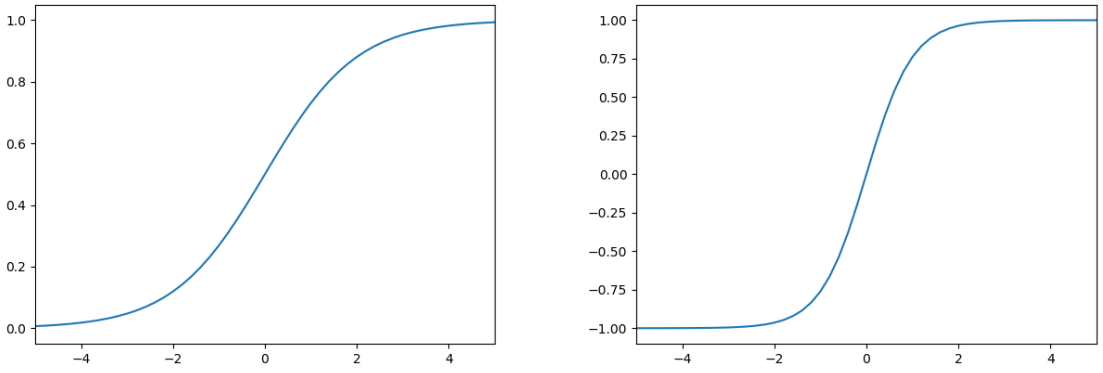
The problem with these activation functions is that they are bounded and their gradient is very low on the edges. Because we want are going to manipulate high scalar values, we have to use an unbounded activation function, such as ReLU, $\Phi(x) = \max(0, x)$ (Figure 5a). But the issue with ReLU is that 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:

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

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

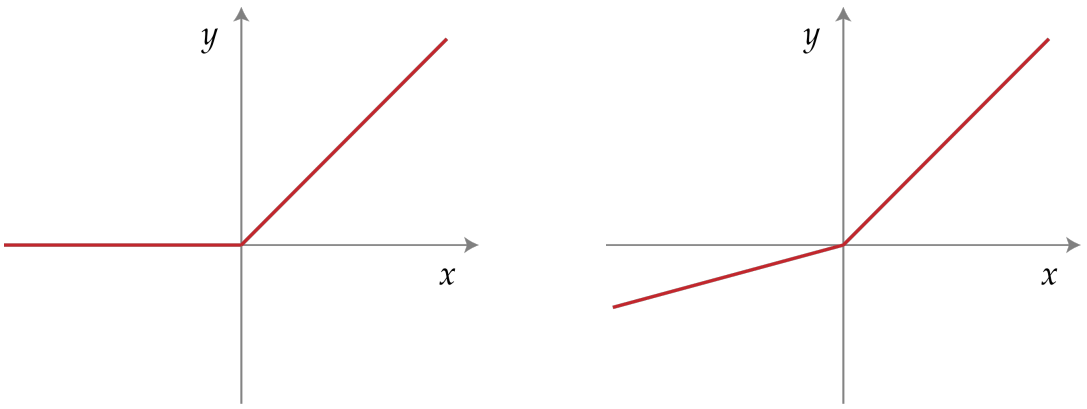
[1] reference ?

[2] ref : `https://pytorch.org/docs/stable/_modules/torch/nn/modules/batchnorm.html`



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

Figure 4: *Illustration of two bounded rectifiers*



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

Figure 5: *Illustration of two unbounded rectifiers*

3.2.2 Chen

Each block L_s is made of 3 layers:

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

3.2.3 UNet

3.2.4 Chen + UNet

3.3 Total Variation (TV) Loss

3.3.1 MSE

3.3.2 TV

Formula

Why est-ce que l'on parle de la façon dont on compute le gradient et des différentes méthodes que l'on a essayées

3.4 Implementation

The network was implemented with PyTorch[3] and we used GPU acceleration [...] (pytorch), se renseigner (section assez courante) GPU acceleraation code sur github

4 Expérience et résultats

4.1 Hyperparameters

évolution des paramètres a et b ?
entrainement (lr, alpha) -> courbes de loss, et loss qui sature (cluster) d'où changement de lr au cours des epochs

4.2 Results on dataset

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

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

[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