Implementation of Sobel edge detector and U-Net

Lorenzo Veronese

NECSTCamp NL1, Politecnico di Milano

Introduction

This project was developed during the NL1 path of NECSTCamp and was focused on two important fields of image processing: contour detection and segmentation. During the first part, a Sobel algorithm, which includes a Gaussian filter and a contour detector, was programmed in C. The second part was about implementing a U-Net on Python using the Tensorflow library.

Contour detector

Sobel algorithm includes two steps. The first one is to blur the image: this means that it is deprived of some details so that they will not be detected as contours during the next step. The second part consists of the computing of the gradient in every zone of the picture. This will give the picture's contours as output. In this case, BMP images representing some organs were used as a case of study.

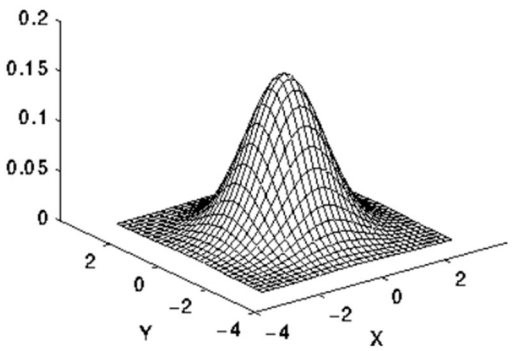
Gaussian blurring

Before looking for its contours, preparation of the image is needed. Generally, we don't want to know the contours of each small element of the picture. Instead, we want to locate its main subject. So, it's a good practice to process the image using a gaussian filter.

The Gaussian filter is one of the most used blurring filters and consists of creating an odd dimensioned squared matrix, filled with numbers corresponding to values of a discretized gaussian slope, centered in the center of the matrix. A continuous gaussian is represented by the following equation:

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

An example of this is shown in *Figure [1]* on the right. Note that all values are divided by 273, which corresponds to the sum of all matrix's values, for normalization. This matrix is called kernel. A consideration about the parameter of the gaussian is needed: while the mean must be 0, the standard deviation σ can



$\frac{1}{273}$

1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1

Figure [1]. 2-D Gaussian distribution with mean (0,0) and sigma=1 and its discretization

acquire any value: a low value will return an elongated curve, while a high value will result in a squashed one. As it will be clear in the following part, a high σ value implies a high smoothing of the image. Generally, it is a good practice to choose a σ equals to half of the kernel size.

Given the kernel, we can perform a convolution product. This means that the matrix slides all over the picture, putting its center on each pixel. At each step (each positioning on a pixel), each matrix value is multiplied by the corresponding pixel value: all these results are then summed together: this is the new pixel value. If we didn't normalize during the building of the kernel, we here would have to divide the resulting pixel by the sum of the matrix's values. Note that for the structure of the convolution, the pixel under the matrix's center has a high weight: this is reasonable since it shouldn't be overhung by the surrounding pixels.

The convolution on the border pixels needs further considerations: in those cases in fact the kernel goes over the picture. To solve this special case, we can choose to mirror the picture, or to create a black border: in this case, the last technique was chosen.

Sobel algorithm

The Sobel algorithm follows the same concepts of the gaussian filter. Another time, it uses a convolution product of some kernels on the image, modifying its values to underline some characteristics, in this case the gradient in two directions.

-1	0	+1
-2	0	+2
-1	0	+1

Gx

+1	+2	+1
0	0	0
-1	-2	-1

Gy

Figure [2]. Convolution kernels for gaussian algorithm.

As shown in *Figure [2]*, we can create two 3x3 matrices: the left one (Gx) will look for the horizontal contours, the right one (Gy) for the vertical ones. Consider for example the first kernel and a black and white image (one channel). A contour is characterized by a high variation in pixels intensity; when we have a vertical contour, this will be "amplified" by the multiplication of its values by the couples of opposite values of the kernel Gx. Thanks to this process, the higher will be the pixel's values variation, the higher will be the resulting value.

After computing convolutions of Gx and Gy over the image, we will obtain Gix (horizontal contours) and Giy (vertical contours). We can now calculate the gradient on each pixel:

$$|G| = \sqrt{Gix^2 + Giy^2}$$

Finally, we apply a threshold to eliminate all pixels under a certain value. The resulting matrix is the image we were looking for: it represents the contours of the starting picture.

Results

Some applications of the algorithm are displayed in *Figure 3*. In this case, biomedical images of CT scans, memorized in BMP format, were used as a case of study.

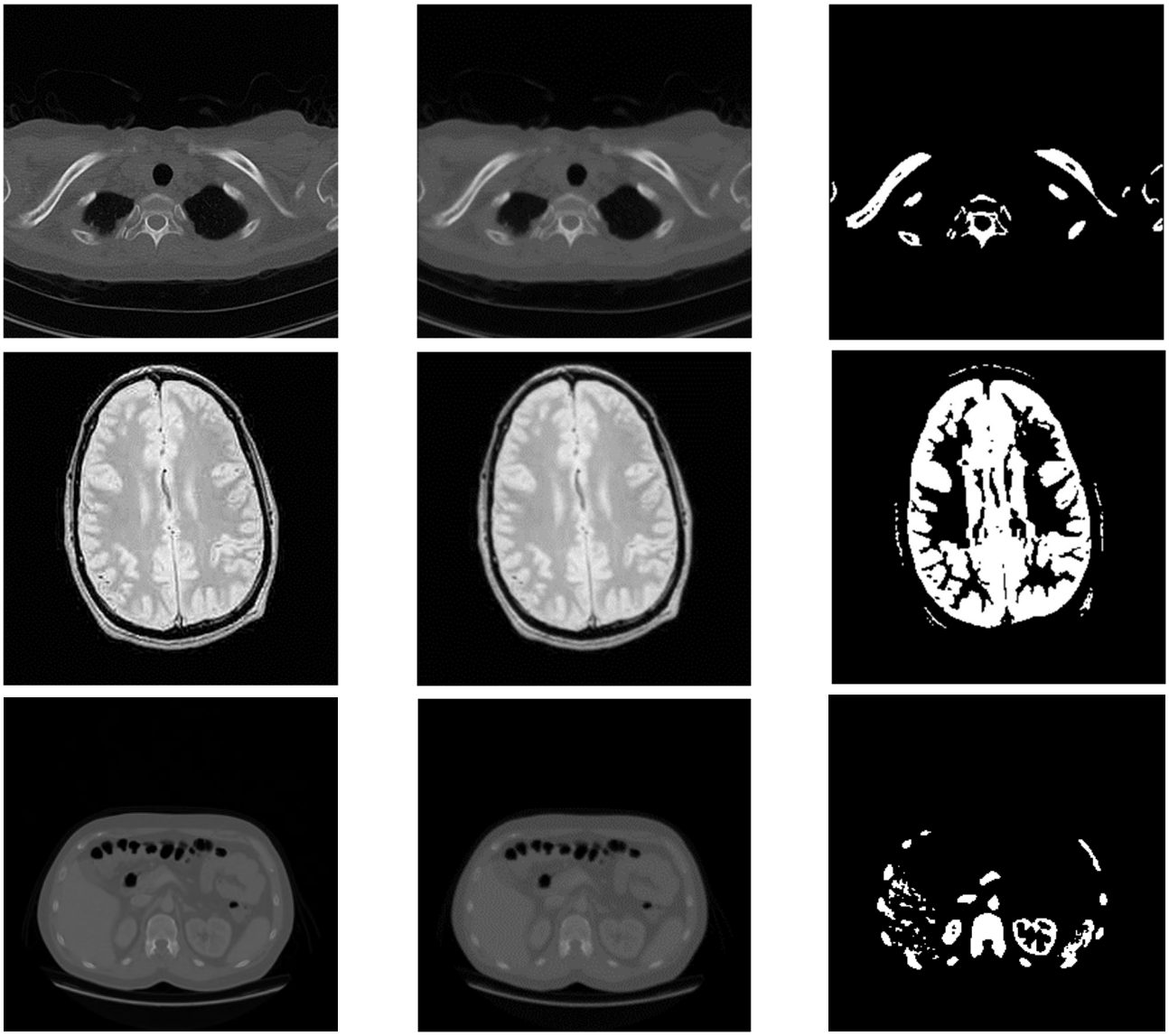


Figure 3. Some results of the gaussian filter (centre. σ=1, kernel of length 3) and Sobel operator (right. Various sensibilities) applied on biomedical images (left). From the top to the bottom there are: lungs, head, abdomen.

U-Net

U-Net is a convolutional neural network proposed by Olaf Ronneberger, Philipp Fischer, Thomas Brox in 2015. It is particularly effective on biomedical images.

Data reading

Dataset used is taken from CTORG and it's a series of 140 images in Nifti-1 format. Each of these files is made of a variable number of 512x512 pictures called slices which together form a scan of the body from head to pelvis. It's like cutting the body with horizontal planes (this is why we talk about slices) and looking at the zone of cutting. Unfortunately, due to hardware limits, not all these slices were used. Furthermore, slices were rescaled to 256x256.

The first 100 files were used as the training set, the remaining ones as the testing set.

U-Net structure

U-Net it's a type of convolutional neural network created to work on biomedical images without having lots of data involved. It consists in applying the following operations:

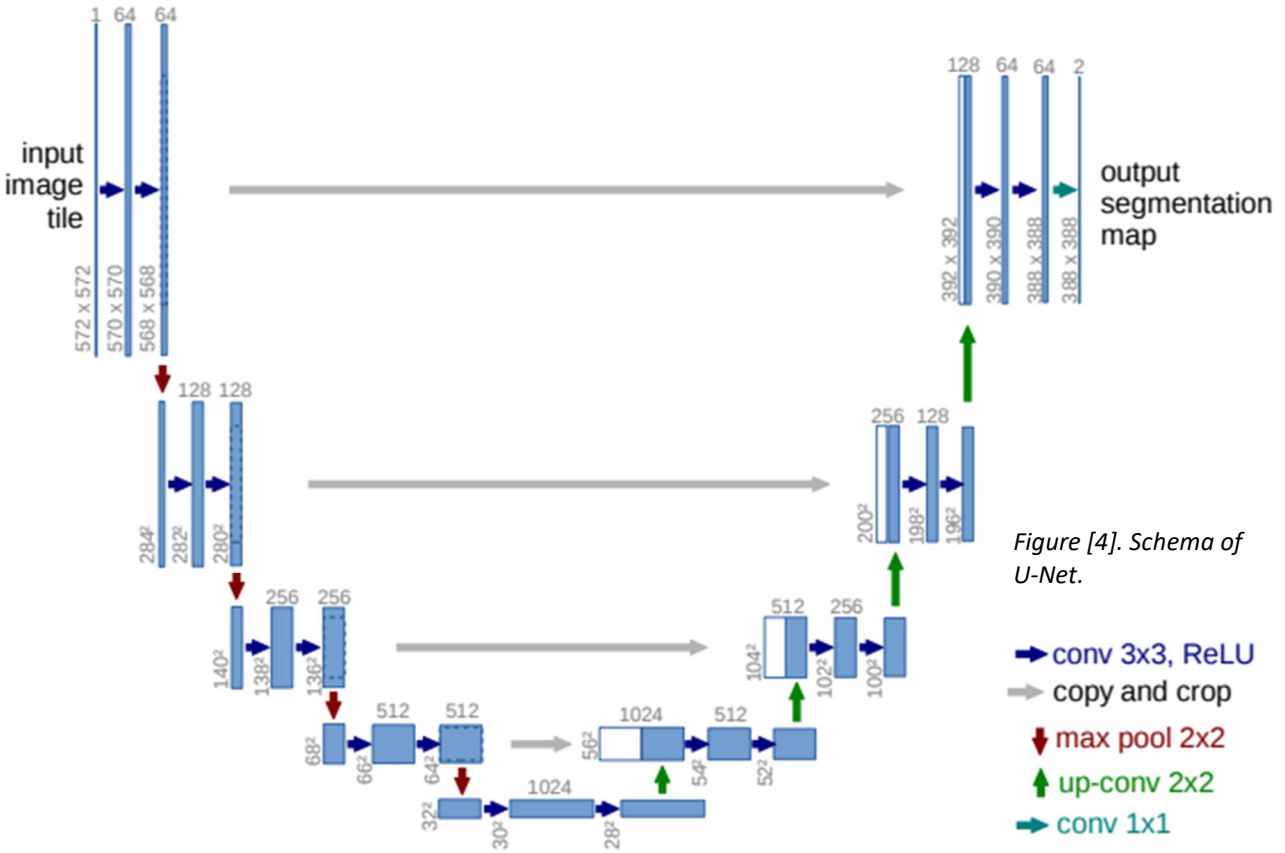


Figure [4]. Schema of U-Net.

- 2D convolution: a kernel slides the image in the same way we have seen during the description of the Sobel edge detector. Using learnable parameters, it finds patterns in the picture. Note that in this case the border is simply eliminated, so that, for example, after the first step a 572x572 image becomes a 570x570 one.
- Max pooling: it takes the maximum value over a set of neighboring pixels. This reduces the complexity of the image, drastically reducing its number of pixels but without losing information.
- Transpose convolution: it acts in reverse with respect to max pooling, increasing the number of pixels.

The resulting schema has a form of "U", which gives the algorithm its name.

Results

In this case, U-Net has been used to find bones. The following pictures (*Figure 5*) show some examples of prediction by the neural network.

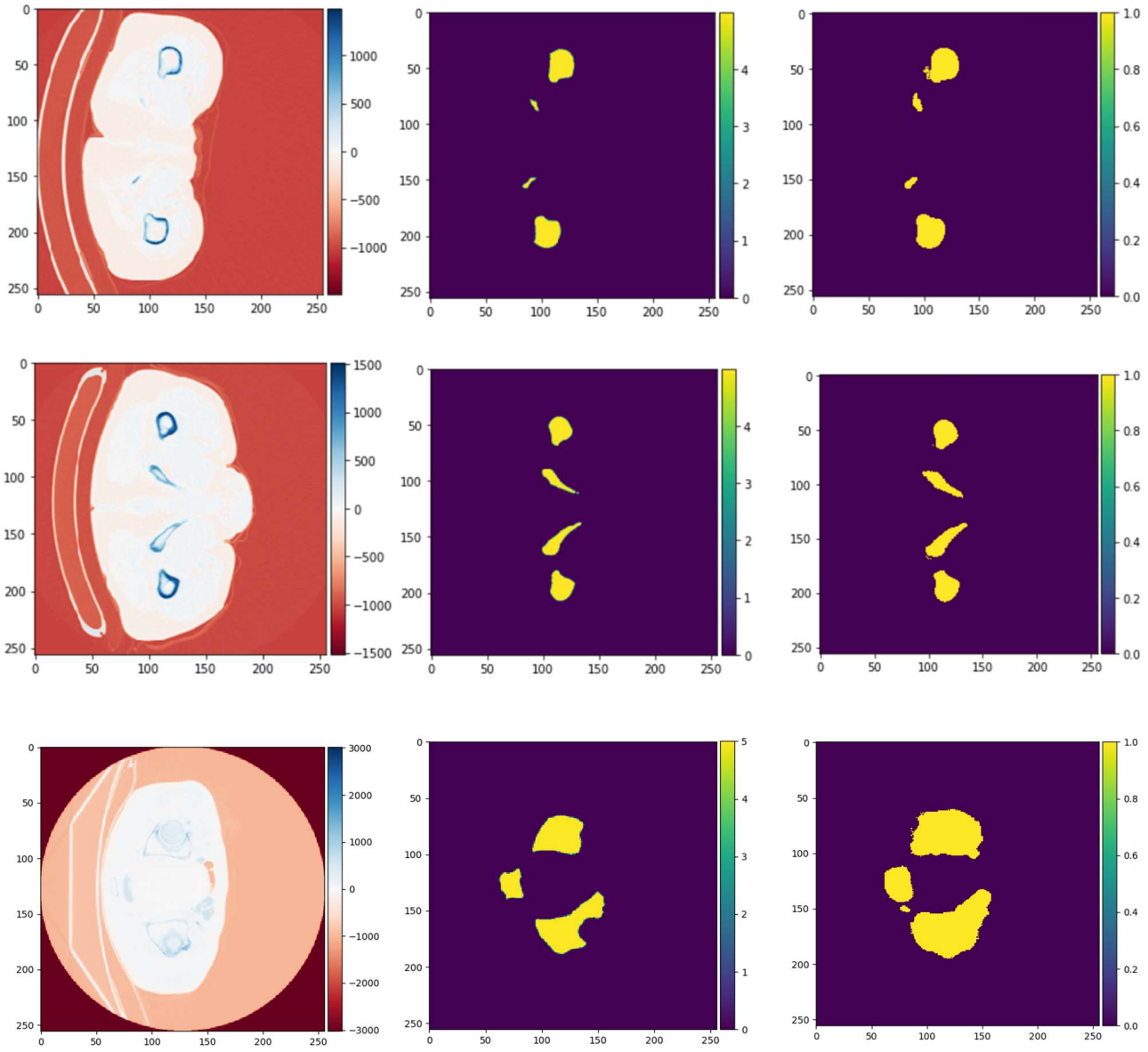


Figure 5. The first column contains the original image, the second one the labelled image, the last one the predicted image.

References

Source code can be found at: https://github.com/LorenzoVeronese/necst

Sobel Algorithm

- Videos from Computerphile.
  Blurs: https://www.youtube.com/watch?v=C_zFhWdM4ic
  Sobel algorithm: https://www.youtube.com/watch?v=uihBwtPIBxM
  Canny edge detector: https://www.youtube.com/watch?v=sRFM5IEqR2w
- [1] Gaussian smoothing: https://homepages.inf.ed.ac.uk/rbf/HIPR2/gsmooth.htm
- [2] Sobel edge detector: https://homepages.inf.ed.ac.uk/rbf/HIPR2/sobel.htm

U-Net

- NiBabel, a Python library to read Nifti-1 files: https://wiki.cancerimagingarchive.net/download/attachments/61080890/README.txt?version=1&modificationDate=1578071801521&api=v2
- An online tool to visualize Nifti-1 files: https://socr.umich.edu/HTML5/BrainViewer/
- Database for U-Net: https://wiki.cancerimagingarchive.net/display/Public/CTORG:+CT+volumes+with+multiple+organ+segmentations
- Videos from Computerphile.
  Deep Learning: https://www.youtube.com/watch?v=TJlAxW-2nmI&t=43s
  Neural Networks: https://www.youtube.com/watch?v=BFdMrDOx_CM
  Convolutional Neural Networks: https://www.youtube.com/watch?v=py5byOOHZM8
- [4] U-Net paper: https://arxiv.org/abs/1505.04597
- Videos from 1 to 6 (the following is the first video's link) about how to build a U-Net using Tensorflow.: https://www.youtube.com/watch?v=azM57JuQpQI
  The resulting code is the following:
  https://github.com/bnsreenu/python_for_microscopists
- Tensorflow Python library: https://www.tensorflow.org/tutorials/images/cnn