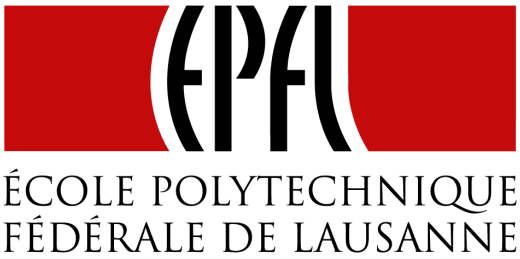
ECOLE POLYTECHNIQUE FEDERALE DE LAUSANNE
SCHOOL OF LIFE SCIENCES



Master project in Bioengineering

Using cortical models of visual perception to improve deep learning

Carried out in the laboratory of Prof. G. Francis
at Purdue University, West Lafayette, Indiana, United States
Under the supervision of Prof. G. Francis

Done by

Michael Stettler

Under the direction of Prof. M. Herzog
In the laboratory of LPSY

EPFL

External Expert Prof. S. Helie

Lausanne, EPFL 2017

Abstract

We developed a new algorithm based on the human visual cortex, namely fast Laminart. We used it as a pre-processing segmentation method and applied the results in front of deep learning methods to study how the model's segmentation capability improve image classification performance. We applied an occlusion paradigm to images and compared the performance of a deep learning system with and without pre-processing the images by the fast Laminart algorithm. We demonstrate how segmentation can benefit the performance of deep learning methods and how the algorithm can mimic human visual perception on the reconstruction of basic hidden shapes.

1 Introduction

The human brain uses a segmentation process to recognize an object with an almost infinite number of scenes, even if we have never previously seen the object in that scene. The visual system of a human certainly helps humans and other animals to understand and interact with the world. But despite the amazing evolution of our visual system, our vision can be tricked and one may perceive illusions. Many vision scientists believe that these illusions have a net benefit by helping people interpret complex scenes even though they sometimes produce wrong interpretations.

The main work presented here is the development of the so-called fast Laminart algorithm. An algorithm based on previous work of Prof. Francis [2] that aims to mimic human vision behavior. By using this algorithm, we hope to help computer vision-methods to have better recognition performance and, perhaps in the future, better scene understanding. Such algorithm development is a challenging task seeing how humans show excellent recognition skills combined with ingenuity for scene interpretation. See Figure 1.

(a) (b)

Figure 1: (a) Famous Little Prince drawing of a snake having eaten an elephant; it resembles a hat (b) Picture of a bird into the sky that can be interpreted as a rabbit making a long ski jump.

When viewing the two illustrations (Figure 1), one may recognize the famous Little Prince drawing of a boa snake eating an elephant that "grown-ups" recognize as a hat. The right image shows a simple sky picture with a bird, but one can imagine a rabbit making a long jump on skies. These two illustrations show how a human is able to use what he perceives to make jokes. In the second picture, the sky makes a strong context dependence of recognizing a bird, but the ears of the "rabbit" and the shape of a skier makes a funny interpretation, especially as it seems impossible.

By developing the fast Laminart algorithm, we hope to enhance deep learning performance, besides the two previous illustrations showing that an interpretation is not unique and could be interpreted as funny, a current deep learning object recognition algorithm would be successful if it is able to recognize a hat, an elephant or a bird. But to be successful, current deep learning methods require a

large amount of good quality data. As computers treat an image as a set of pixels, a simple shift of the picture is considered a whole new problem to solve for the algorithm. Fortunately, current deep learning methods seem to have overcome several of these issues and show outstanding results [4] in object recognition. Due to the nature of a deep learning algorithm, performance could easily deteriorate when data are either of bad quality or not sufficient enough. To demonstrate how performance could be affected, Figure 2 shows classification results when a simple occlusion paradigm (see Figure 16) is applied on a database used to train a convolutional neuronal network (CNN). The blue line shows how the accuracy performance of a CNN decreases when a simple occlusion paradigm is applied, despite the impression that the same paradigm should show only a small difference in performance when applied to humans. The poor performance in Figure 2 is rather surprising as many researchers argue that CNNs use methods similar to the human visual systems [5, 6, 8].

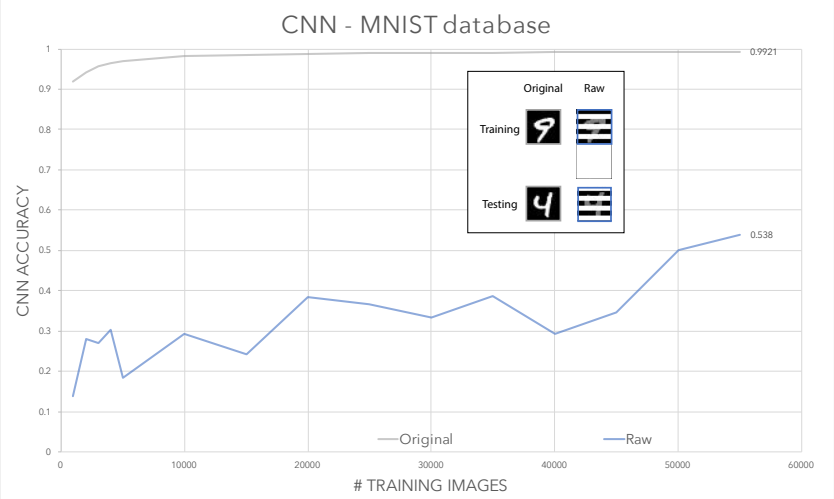


Figure 2: Graphic showing accuracy performance in number of training images for the MNIST database recognition digits using the TesnorFLow tutoral CNN. Gray line is for the original data while blue line is for the partially occluded data.

Deep learning methods create features by "averaging" across a large set of images, whereas human visual perception processes each individual image. By using the human visual characteristics explained in the Laminart model, objects contained in an image might benefit from the features created by the model. The development of the fast Laminart algorithm aims to study how pre-processing segmentation may help deep learning performance. The features implemented by the fast Laminart algorithm should partially restore the loss of performance seen in Figure 2.

2 Models

2.1 Fast Laminart Algorithm

The fast Laminart Algorithm aims to model the cortical human visual system processing in order to segment objects. The model is based on the Laminart model [2] but uses a faster way to process images while trying to keep the behavioral results of the former model. The following sections will explain the different steps performed by the algorithm. The fast Laminart algorithm receives as input a black and white image ([0; 1]) and displays or saves: the input image, k images that each contain one single object found by the algorithm, and the remaining background (Figure 3).

The main purpose of this algorithm is to explore how segmentation and perception of objects may help computer vision methods with scene understanding and object recognition. The algorithm is based of what is known about the visual cortex of the brain, but it emphasizes fast processing and behavioral performance over neurophysiological details. In the following sections, an explanation on all the main

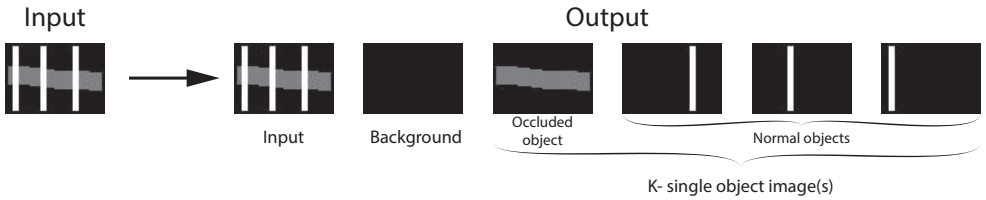


Figure 3: Example of the output images produced by the fast Laminart algorithm.

steps performed by the algorithm will be given, in to order to give a sense of how the algorithm has been implemented. As the code has been posted on GitHub, more specific details can be found by looking directly inside the code. Fast-Laminart GitHub: [link] https://github.com/michaelStettler/Fast-Laminart.

2.1.1 Image Preprocessing

As for many machine learning algorithms, fast Laminart performs some pre-processing steps before segmenting and reconstructing objects. The first step of the fast Laminart algorithm is to construct the boundaries/edges that will play a key role in later methods. Having the right boundaries is a crucial point and several issues have to be overcome in order to obtain the best boundaries possible for the majority of possible images. These steps could be interpreted as the V1 and the V2 visual cortex processing made by the human brain. The methods implemented here try to overcome several issues such as a gray scale gradient and is able to reconstruct the shapes we consider here.

Boundaries The first step performed by the algorithm is to calculate contrast boundaries. The aim is to simply get values around each pixel representing the strength of a pixel compare to its neighbours. In this sense, two similar pixels would have no boundaries between them, and a black pixel (0.0) next to a white pixel (1.0) would have the maximum strength. Figure 4 shows the four possible boundaries a pixel may have:

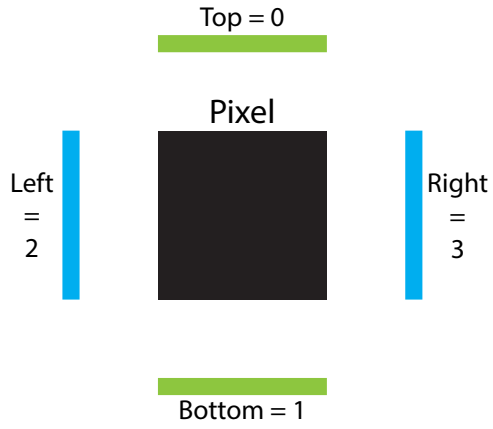


Figure 4: Description of the four boundary types for one pixel and their respective index in the boundary matrix.

The boundaries are calculated by applying convolution filters on the images. The filters used are defined as follows:

$$vertical\ left\ filter = \begin{bmatrix} 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.2 & 1.0 & -1.2 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

$$vertical\ right\ filter = \begin{bmatrix} 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & -1.2 & 1.0 & 0.2 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

Horizontal filters are made by a rotation of 90° of the vertical filters. The results are saved into matrices, and each filter is saved into its own matrix; thus all the boundaries are saved into one final tensor of size (4, m, n), with m and n representing respectively the height and width of the image and the 4 representing the four types of boundaries, in the following order:

$$top = 0,\ bottom = 1,\ left = 2\ and\ right = 3$$

By saving the boundaries as such, each pixel can access its four boundaries. This arrangement may seem counter-intuitive as one could have used only one dimension to save all the boundaries, but it allows further parts of the algorithm to perform easier grouping conditions for the segmentation of an object. The downside of this implementation resides in the indexes which force the computations to have several nested if statements that may slow down the implementation. Finally, a boundary is considered active when its value is higher than a threshold of 0.2.

Visualization tool Figure 5 shows a visualization tool used to help understand and debug properties of the model. The visualization constructs a grid where each image pixel is enlarged with gaps between them. The spaces between the pixels are used to draw any found boundaries.

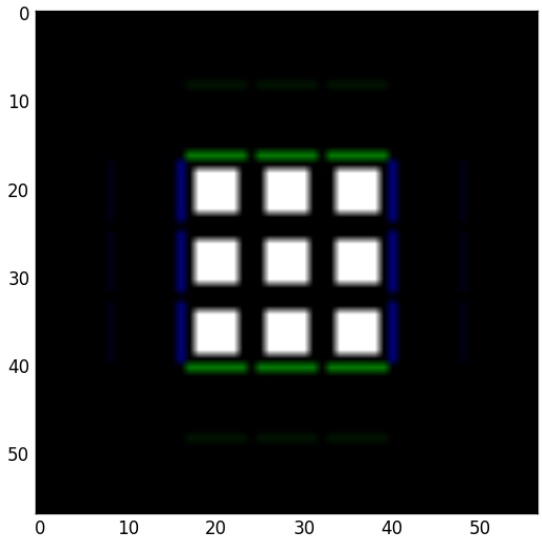


Figure 5: 3 x 3 pixels square with its boundaries rendered using the visualization tool.

One can see how the boundaries are drawn in between the pixels, top and bottom boundaries are drawn in green while left and right boundaries are drawn in blue. Furthermore, the method normalizes the boundaries, which allows for a quantitative visualization. The strength of a boundary is represented by the intensity of its color.

Pool Boundaries The pooling boundaries process aims to connect boundaries in a way that mimics how humans perceive elements in a scene. This behavior is sometimes referred to as perceptual grouping, boundary completion, spatial integration, or boundary pooling. The method uses arguments to define the size of the pooling filters as well as a strength coefficient parameter to fine tune the range and intensity of the pooling effect. The parameters were set manually to give the overall best results. Here in order to simplify the implementation, the strength of the coefficient parameters was set according to

test cases. The method adds boundaries close to the existing boundaries in the image. The boundaries added may explain some illusions [3] and may play a role in boundary segmentation [2].

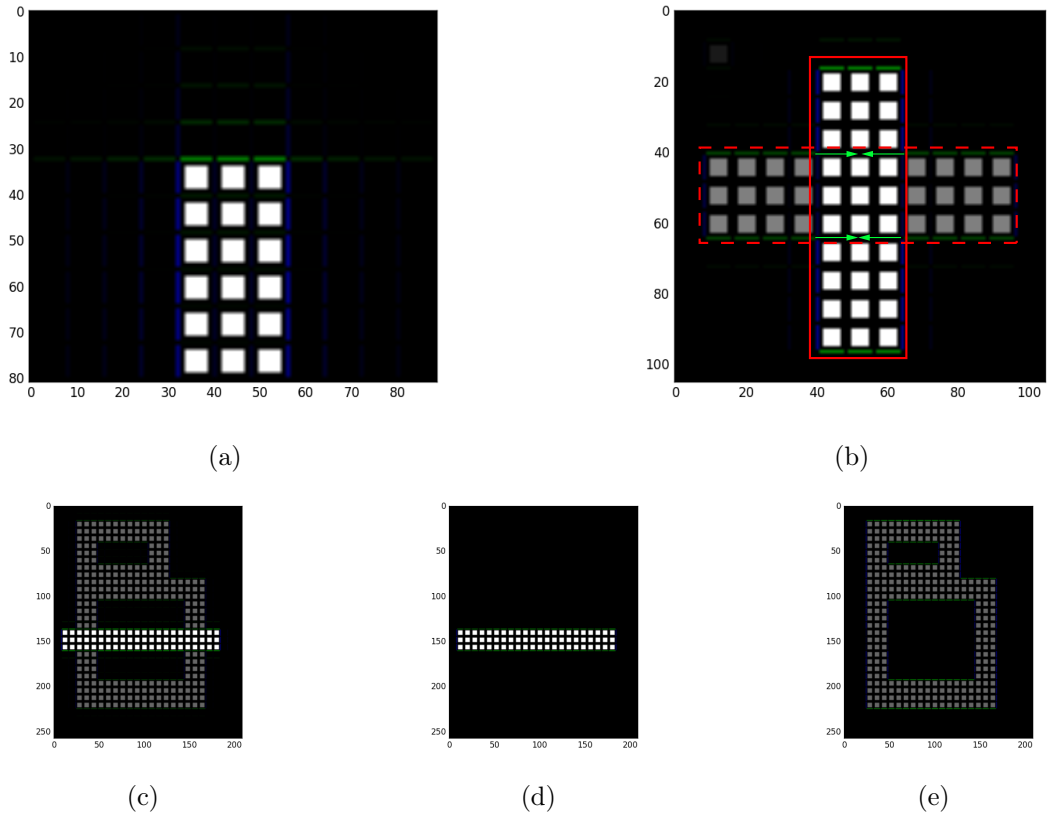


Figure 6: (a) Example of the pooling effect on a simple rectangle, (b) Representation of the pooling boundaries effect, green arrows shows the pooling effect of a light-gray rectangle (dashed line) hidden by a white rectangle (plain line), (c) Grossberg paradigm (see [3] Figure 7), (d) Segmentation of first object, (e) Reconstruction of second shape using the pooling boundaries effect.

As seen in Figure 6a, the pooling effect allows boundaries to spread outside a shape. The pooling effect is visible around the corner of the rectangle shape as the green border now spreads to neighbouring pixels. But the pooling effect is not only constrained to spread outside a shape, and may also spread within another shape, a feature that plays a role on how one can perceive a shape on top of another one [3]. To help the reader understand how the pooling effect works, Figure 6b shows a vertical white bar (plain contour) on top of a horizontal light-gray bar (dashed contour). The green arrows show how the pooling effect spreads within the white bar and creates a binding illusion, in this example, the pooling effect helps to visualize the light-gray rectangle hidden by the white rectangle. By using the pooling effect, we were able to reproduce the work of Grossberg paper [3]. The result can be seen with Figure 6c, which shows a white bar on top of a "B" shape, the algorithm segments the white bar and uses the pooling effect to reconstruct the hidden shape.

The pooling effect, used here with this Grossberg figure, shows its capabilities for simple object reconstruction (i.e. straight lines). However, for the fast Laminart algorithm, the pooling effect is meant to be used for future improvement of the algorithm by using boundary segmentation (connection of different objects together), but is not yet exploited. The pooling boundary as of the current implementation is more a historic step based on the work of the Laminart model, the reconstruction process for more elaborate shapes are yet treated more as an engineering problem. See Curve Completion and Straight Line Completion sections for more details.

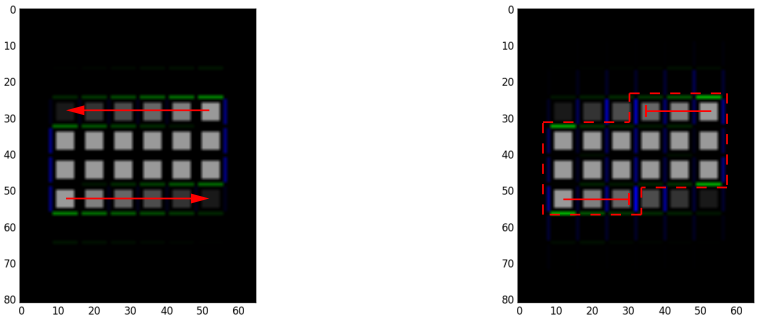
Pool Shade Boundaries When examining how pooling occurs, we identified a case that required special methods. The Pool Shade Boundaries method aims to resolve a problem found in real images. Even ostensibly black and white images may use a gradient effect that creates a smooth transition between the background and the shape. As a result, the shape is less well delimited and the filters may fail to create strong enough boundaries to let the algorithm reconstruct a shape, in other words, no closed contour can be found due to the gradients. As the algorithm uses a simple threshold to consider if a boundary is strong enough, a simple solution could be to lower the threshold, by doing so, the algorithm would be able to find a closed contour. But as a result, the algorithm may find too many small "objects" that do not correlate with the final shape. A better solution is thus to add a boundary at the right position in order to close the contour. The red arrows in Figure 7a show where the filters fail to generate active boundaries due to the gradient effect, the shape is considered "open". Figure 7b shows where boundaries could be added in order to close the shape. To add these boundaries, the method uses filters that are passed one more time to the image to detect parallel boundaries, by doing so, the method adds the extra boundaries responsible to connect parallel boundaries, and thus close the overall shape. As it may be difficult to understand the formulas to calculate the boundaries, a schematic representation is show in Figure 7c. The X boundary in the schematic representation shows the boundary responsible to for closing the shape. The general formula to calculate this boundary for a top (top = 0) type boundary is written below:

$$(m,n)_0\ (top\ boundary) = \alpha(\tag{1}$$

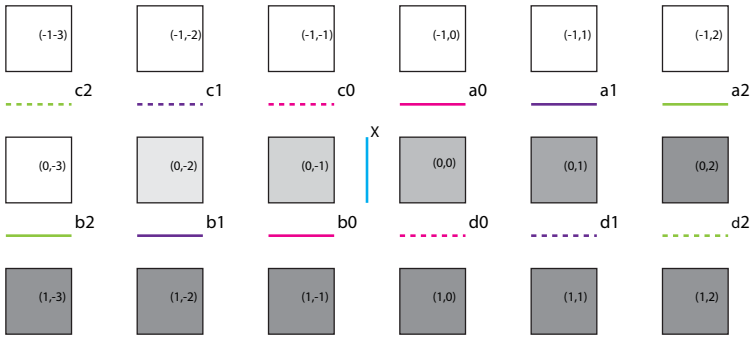
$$\sum_{i=1}^{filter\_size+1} |\frac{|(m-i+1,n-1)_2 + (m+i,n)_2|}{2} - \frac{|(m,n)_2 + (m+i,n-i)_2|}{2}| + \tag{2}$$

$$\sum_{i=1}^{filter\_size+1} |\frac{|(m-i+1,n)_3 + (m+i-1,n)_3|}{2} - \frac{|(m-i,n)_3 + (m+i-1,n-i)_3|}{2}|) \tag{3}$$

Parameters m *and* n represent the position of the pixel. The subscripts represents the kind of boundary (0 = top, 2 = left and 3 = right) while α represents the strength coefficient that may be fine tuned for each filter size. α and the filter_size are parameters given to the method. α strengthens or weakens the pooling shading effect, the filter_size plays on the spatial size of the pooling effect.



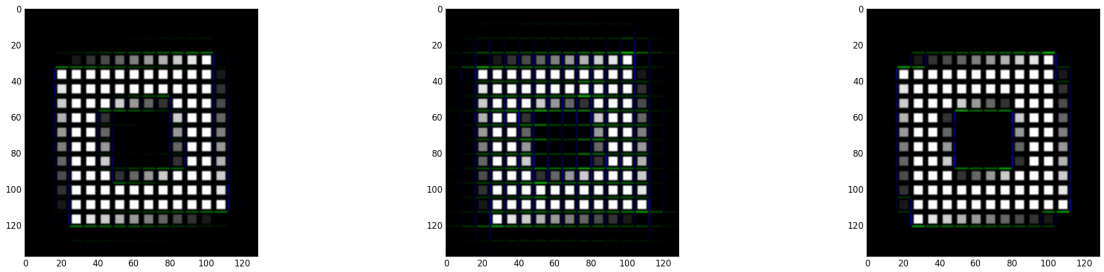
(a) (b)



(c)

Figure 7: (a) Initial boundaries displayed for a gradient example shape, red arrows shows where the contour is open (b) Boundaries displayed after the pool shade boundaries method. The dotted line shows the closed contour, the arrow shows were the newly added boundaries close the shape (c) Schematic representation of how the pooling boundary method calculates boundaries for the left boundary "X" with a filter of size 3.

Remove Inner Boundaries This method is the last pre-processing step and has to be implemented in order to counter act the two previous methods. Indeed as the pooling and the pooling-shade method only add boundaries, they also give rise to some undesired boundaries, the boundaries inside a shape. These new boundaries may cause the algorithm to fail in finding the correct shape of an object. As having the right boundaries is crucial, the aim of this method is to remove the inner boundaries that may have been added within objects. To do so, the method uses patches of pixels to calculate the mean between each pixel as well as the difference between the maximum and the minimum pixel value in the patch. The mean value could be understood as a value representing how much the patch is inside an object, a value higher that 0.5 for 4 pixels is considered inside an object. The difference gives a sense of how the patch may lie at the border of the object, a high difference corresponds to a border, while no difference means that all the pixels within the patch are the same. Thus, if the mean is higher than 0.5, and the difference is smaller than a threshold, the boundaries inside the patch are removed. A high threshold means a low sensitivity, as only the boundaries that lie in patches with small min/max value differences will be removed. Note that in this implementation, the boundaries lying in a patch with a very small difference are also removed, thus all the background is removed, it is then a matter of which method is called first that will enable to play with the pooling effect.



(a) (b) (c)

Figure 8: (a) Initial boundaries displayed on a rotating square showing all kinds of possible gradients, (b) Boundaries displayed after all the pooling boundaries preprocessing methods, (c) Boundaries display after the removal of inner boundaries (threshold difference = 0.9).

Figure 8 shows an example of the different stages of pre-processing boundaries on the same shape. As seen in 8a, without pre-processing, the boundaries fail to construct the contour of the shape. Figure 8b shows how the pooling methods add boundaries, note that the shape is yet closed but does not correlate well with the initial shape. And finally, Figure 8c displays the effect of the removal of the inner boundaries, yet the contour of the shape reassembles better to the initial shape.

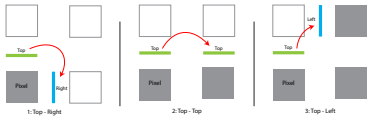
2.1.2 Finding Objects

As the boundaries are now considered to correctly represent the contour of an object, the algorithm has to find objects within the picture. In order to find an object, the implementation makes two assumptions. First the starting point of an object will be the strongest boundary value, meaning that the higher contrasted object is supposed to be in front of the others, and second, objects are considered to be in the middle of the image on a black background. These assumptions could easily be tricked and limit the application of the algorithm, but it allows an easy implementation for the application discussed below.

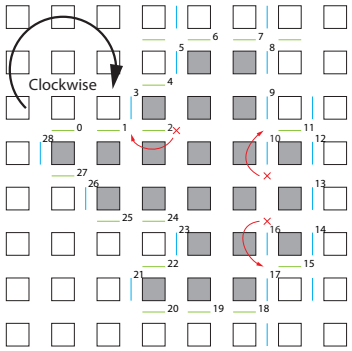
The algorithm considers two types of objects, normal and occluded objects. Normal objects have a closed contour and can be segmented out as it is, occluded objects have a hidden part behind another object. In practice, it means that the normal object doesn't need any post-processing, while the occluded object will go through a reconstruction process.

Normal Objects In order to find a normal object, the algorithm moves along its boundaries and tries to come back to its initial position in a closed loop. If it succeeds, then the object is considered as a normal object. Note that the path used by the algorithm to find the object is taken as the exact contour of the object.

Concretely, the algorithm will look for the largest boundary value in the image as a starting point. Then it will look for the three neighbouring boundaries, turning in a clockwise manner, see Figure 9a for a top boundary example. While looking for the next boundary, the algorithm checks the value, and if it is high enough, then the algorithm recurses and moves forward. If no boundaries are considered active, then the current boundary is deleted and the algorithm comes back to the previous step to look for further boundaries, see 9b for an example of how the algorithm moves on a given shape. If the algorithm comes back to its initial position, the contour is considered closed and a shape has been found.



(a)



(b)

Figure 9: (a) Representation of the three top possibilities and their order. (b) Schematic representation of how the fast Laminart algorithm recurse the boundaries to find a shape, the number represent the order of which the algorithm will move, the red cross represent when no more boundaries are found and how the algorithm comes back one step backward before looking for the next boundary.

A minimum of 7 boundaries are necessary for a shape to be considered, this value allows the algorithm to get rid of isolated single noisy pixels. The algorithm also takes care of the border of the images, if the path touches two borders, then the algorithm is considered to have found a shape, as the path is considered closed.

Finally, when a boundary is closed, the algorithm will segment out the object from the input image to a new image. To do so, the corresponding pixels of the found object in the input image are removed and marked. The marking is made by setting the value of the pixel to -1. Note that the images are treated as matrices in the algorithm, thus negative values are supported. In this way, the algorithm knows which pixels on the input image have been removed (segmented out). Then, the algorithm proceeds to the reconstruction of the segmented object in the new image, the process is explained in the reconstruction section.

Occluded Objects The second kind of object, the occluded objects, are ones that miss a border and thus cannot be reconstructed directly; in other words, the algorithm will fail to come back to its initial position, because there is no closed contour. From a human point of view, those are the objects that could be considered as partially hidden, such as the letter B in Figure 6c. In this implementation, while the algorithm is looking for a normal object but ends up in a path where no more boundaries are found, a special case is made. The algorithm will look if this non-boundary case is because an object was there before and has been previously removed. Recall that the algorithm knows where an object has been segmented out as pixels are marked, see the Reconstruction section for more details. Thus, when a contour ends up in this non-boundary case, the algorithm looks at the nature of the surrounding pixels. Given the pixels were segmented out, then the algorithm saves the position and creates a new inducer, which will be important for the Reconstruction algorithm. Next, the algorithm recurses again, but this time in a counter-clock wise manner. In the case of the algorithm finding two inducers on the same path, the algorithm considers the path as being closed and can recurse again, but this time, with the difference that the path belongs to an occluded object, meaning that the algorithm does not yet proceed to the reconstruction of the object. Indeed as an object needs a closed contour before being considered as an object, the algorithm needs first to go through an interpolation process in order to close the shape. Only in the case of a closed contour could be found after the interpolation process, then the shape is considered as an occluded object and the algorithm can proceed to its reconstruction. Figure 10 shows how the algorithm selects the boundary of an occluded object.

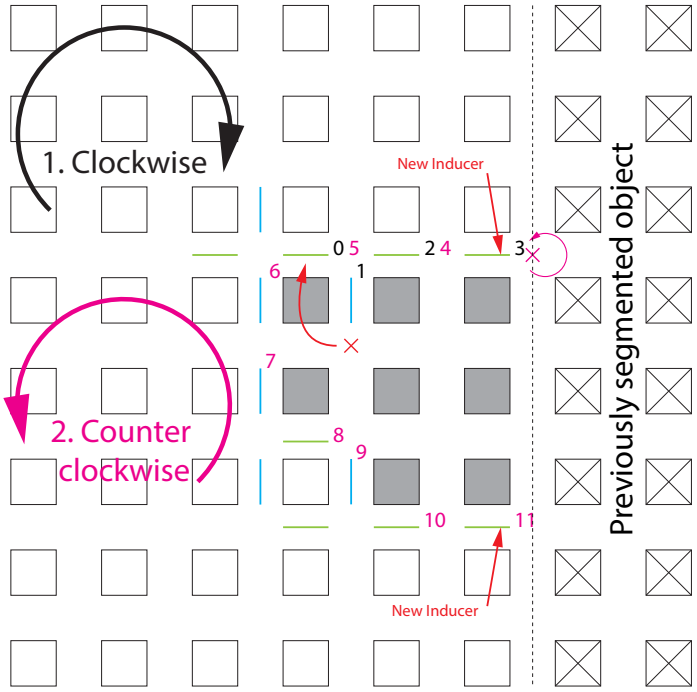


Figure 10: Schematic representation of how the boundaries of an occluded object may look. The algorithm first tries to selects the shape in a clockwise sense. When the path hits a previously segmented object (represented by the crossed squares), it changes its direction and selects the object in a counter-clockwise manner. The red cross represent where the algorithm does not find any active boundaries and makes a step backward before moving on. The purple represents when the path hit the previously segmented object. The new inducers created are shown with the red arrows

Note that the algorithm does not know how many objects are present in the picture, therefore, when an occluded path is found, the algorithm recurses in order to find all the remaining segmented paths, thus catching all the inducers present in the picture. Only then the will the algorithm proceed to group the inducers and interpolate the missing boundaries. When all the interpolations are done, then the reconstruction of objects may take place.

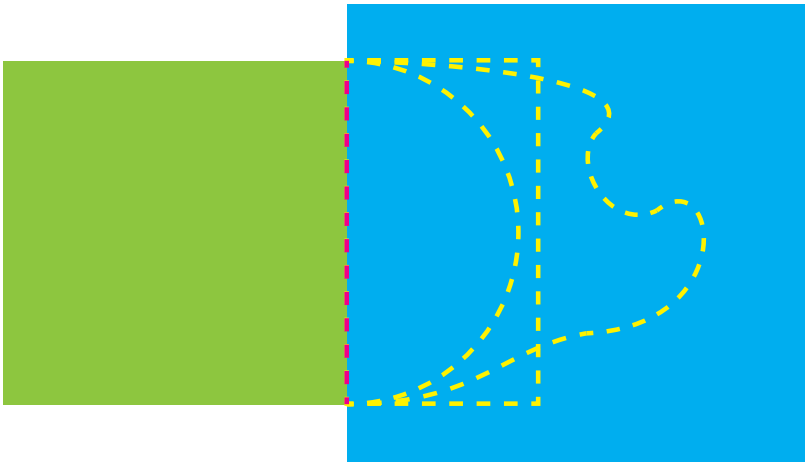


Figure 11: Representation of a ill posed problem of two squares touching each other, the red line corresponds to the case where two squares are touching each other, while the yellow lines correspond to some examples of what the green "square" could look like if it was hidden by the blue square

Inducers In the previous method, the positions of the inducers were saved, but to emulate human behavior, the sole positions of the inducers is not enough to interpolate the occluded boundaries. Indeed, as the problem is ill-posed, see Figure 11, the algorithm needs more information. Following ideas of Ben-Shahar and Ben-Yossef [1], we identified inducer properties that are used to group the inducers and interpolate the occluded boundaries. This method calculates the parameters required by the following method: Grouping, Straight Line Completion and Curve Completion.

The parameters being calculated by this method are listed below:

1. The type of boundary (top, bottom, left, right) relative to its pixel location
2. The direction of the inducer (clockwise or counter-clockwise), relative to the bounded shape. This is defined by the search process that found the inducer (see Figure 10)
3. The linearity property indicates whether the inducer is part of a straight line or a curved line of boundaries.
4. The angle (α), indicates the orientation of the inducer, relative to horizontal, right (see Figure 12)
5. The radius, indicates local curvature of the inducer. It is set to infinity when the inducer is "linear"

To calculate these parameters, the method tries to fit a linear and a quadratic model on the neighbouring path of the inducer. The smaller fitting error between the two models defines the linearity of the inducer. The angle is given by the fitting function and the radius is defined as follow:

$$r = +\infty \qquad with\ y = ax\Big|_{x=0} \tag{4}$$

$$r = \frac{(1+y'^2)^{\frac{3}{2}}}{|y''|} \qquad with\ y = ax^2 + bx\Big|_{x=0} \tag{5}$$

The parameters x and y represent the position of the inducer. The linear model is represented by the equation $y = ax$ (4) while the quadratic model is represented by the equation $y = ax^2 + bx$ (5). As the method uses the inducers as starting points, the lines pass by the origin, thus the missing

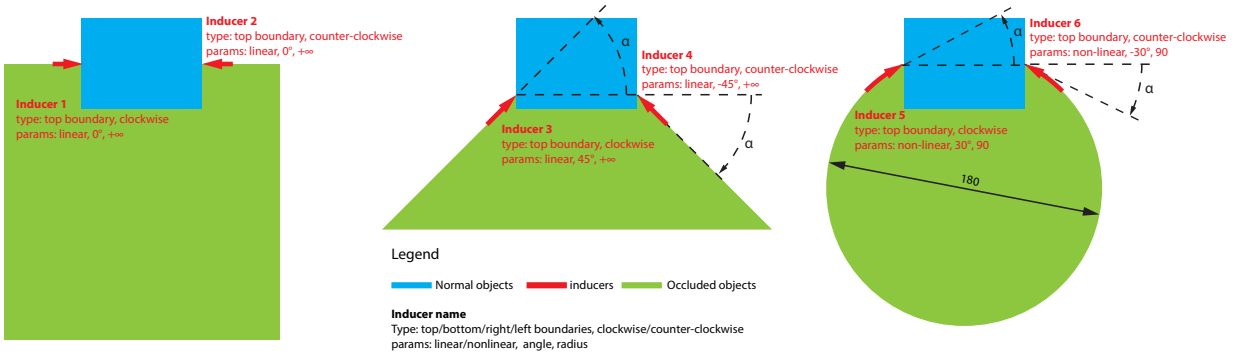


Figure 12: Example of inducers and their calculated parameters

y-intercept parameters and the evaluation of the function at $x = 0$. Note that the radius parameter is currently not used, see Curve Completion for more details.

Figure 12 shows some calculated examples of inducers. The green shapes represent the occluded objects by normal objects (blue). The red arrows represent the position of the inducers with their respective parameters. The direction of the arrows shows the type of inducer direction.

Grouping The grouping method uses a greedy algorithm approach to select the most plausible pair of inducers based on their parameters. The most plausible pair is defined by the highest score between two inducers. The score is calculated by the following formula (for inducers i and j):

$$s_{ij} = \frac{ct * cl * ca * \Delta a}{d * \Delta I} \tag{6}$$

$$with\ ct = \begin{cases} 1.0 & \text{if } t_i = t_j \\ 0.5 & \text{if } t_i = 0 \text{ and } t_j = 3 \text{ or } t_i = 1 \text{ and } t_j = 2 \\ 0.1 & \text{if } t_i = 0 \text{ and } t_j = 1 \text{ or } t_i = 2 \text{ and } t_j = 3 \\ 0.0 & \text{otherwise} \end{cases} \tag{7}$$

$$ca = \begin{cases} 1.0 & \text{if } t_i = t_j = 0 \text{ and } a_i > 0 \text{ and } y_i >= y_j \\ 1.0 & \text{if } t_i = t_j = 0 \text{ and } a_i < 0 \text{ and } y_i <= y_j \\ 1.0 & \text{if } t_i = t_j = 1 \text{ and } a_i > 0 \text{ and } y_i <= y_j \\ 1.0 & \text{if } t_i = t_j = 1 \text{ and } a_i < 0 \text{ and } y_i >= y_j \\ 1.0 & \text{if } t_i = t_j = 2 \text{ and } a_i > 0 \text{ and } x_i <= x_j \\ 1.0 & \text{if } t_i = t_j = 2 \text{ and } a_i < 0 \text{ and } x_i >= x_j \\ 1.0 & \text{if } t_i = t_j = 3 \text{ and } a_i > 0 \text{ and } x_i >= x_j \\ 1.0 & \text{if } t_i = t_j = 3 \text{ and } a_i < 0 \text{ and } x_i <= x_j \\ 0.2 & \text{otherwise} \end{cases} \tag{8}$$

$$cl = \begin{cases} 1.0 & \text{if } l_i = l_j \\ 0.7 & \text{otherwise} \end{cases} \tag{9}$$

$$and\ d = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \tag{10}$$

With, t the difference on the type of inducers, l the linearity, d the distance, I the intensity and a the angle.

All inducers are separated between the clockwise and the counter-clockwise type, which indicates border ownership relative to an object surface. This separation is possible as we want to close the shape, so a clockwise inducer can only match with a counter-clockwise inducer, recall Figure 12, by doing so, the problem is reduced by a factor of two. The formula also benefits from the use of the type of the boundaries, indeed two same types of boundary usually create more continuity, as two

same types would usually be found in straight lines. The angle also plays an important role, as we are looking for continuity, a positive angle means that the line is going up, so there will be an advantage of inducers that have a higher position. Finally the formula tries to benefit from the other parameters essentially by calculating the difference between two parameters, i.e. a smaller distance is preferred to a longer one or a similar intensity is preferred to a different one. Indeed the boundary of a light-gray shape should pair with another light-gray boundary and not with one of a white shape, if colors were present, this would correspond to paired objects having the same color, such as a blue boundary should pair with another blue boundary and not with a red boundary.

When all the scores are calculated across the inducer's list, the algorithm picks the highest one and removes the two respective inducers from the list. Then the algorithm re-calculates all the scores between the remaining inducers, picks up the new highest score, removes the new respective inducers, and so on until no more pairs of inducers are present.

Straight Line Completion A home-made function was developed to connect two grouped inducers. The aim is to interpolate the missing (occluded) boundaries before the reconstruction process. The method constructs the boundaries by solving the equation for two straight lines and interpolating the missing boundaries between the two inducers. The equation is of the kind:

$$\begin{cases} a_i x_i + b_i = y_i \\ a_j x_j + b_j = y_j \end{cases} \tag{11}$$

The slopes ($a_{i,j}$) and the positions of the two inducers are given by the previous method. Solving the equation is thus rather trivial. But the complexity of this method resides in the fact that the solution may be outside of the picture or not exist. Figure 13 tries to summarize all the possibilities for the top boundaries that may exist. The dotted lines represent the interpolation boundaries made by the method.

Curves Completion The curvature interpolation is not yet functional. We have successfully implemented the work of Ben-Shahar and Ben-Yosef [1] to reconstruct curvature from inducers, unfortunately the model needs to start with good initial parameters in order to converge. Due to this lack of robustness, we decided to try to implement the work of Zhou on Euler arc splines for curve completion work [9], we contacted Zhou and received her code but we were not yet able to run it properly.

2.1.3 Reconstruction

When every previous step has been completed, eventually a closed contour is present and a shape is considered found. The algorithm could then reconstruct the shape, in other words, the algorithm will create a new image containing the object. For normal objects, the shape and colors of the object will simply be the values of the pixels from the input image that lies within the close contour. Occluded object reconstruction is a slightly different case, the shape is also described by a closed contour, but the colors of the segmented pixels needs further processing, as part of the shape is yet missing.

To identify pixel colors, the reconstruction algorithm uses a simple recursive approach combined with a visited pixel boolean matrix. Basically, when starting with a pixel, the algorithm will go to the four direct neighbour pixels in the following order:
$1 : top, 2 : bottom, 3 : left, 4 : right$
Before moving on to the next pixel, the algorithm looks if the boundary between the two pixels is active, if not, then the algorithm recurses and sets the new pixel to visited status. The new pixel copies the value of the input images and sets the value of the input image pixel to negative (pixel value = -1) before starting a new loop. Note the negative pixel value, as the algorithm uses matrices, negative values (pixels) are supported, when displaying the image, negative values are set back to 0. The algorithm is highly recursive but has the benefit of checking every pixel, especially as objects may

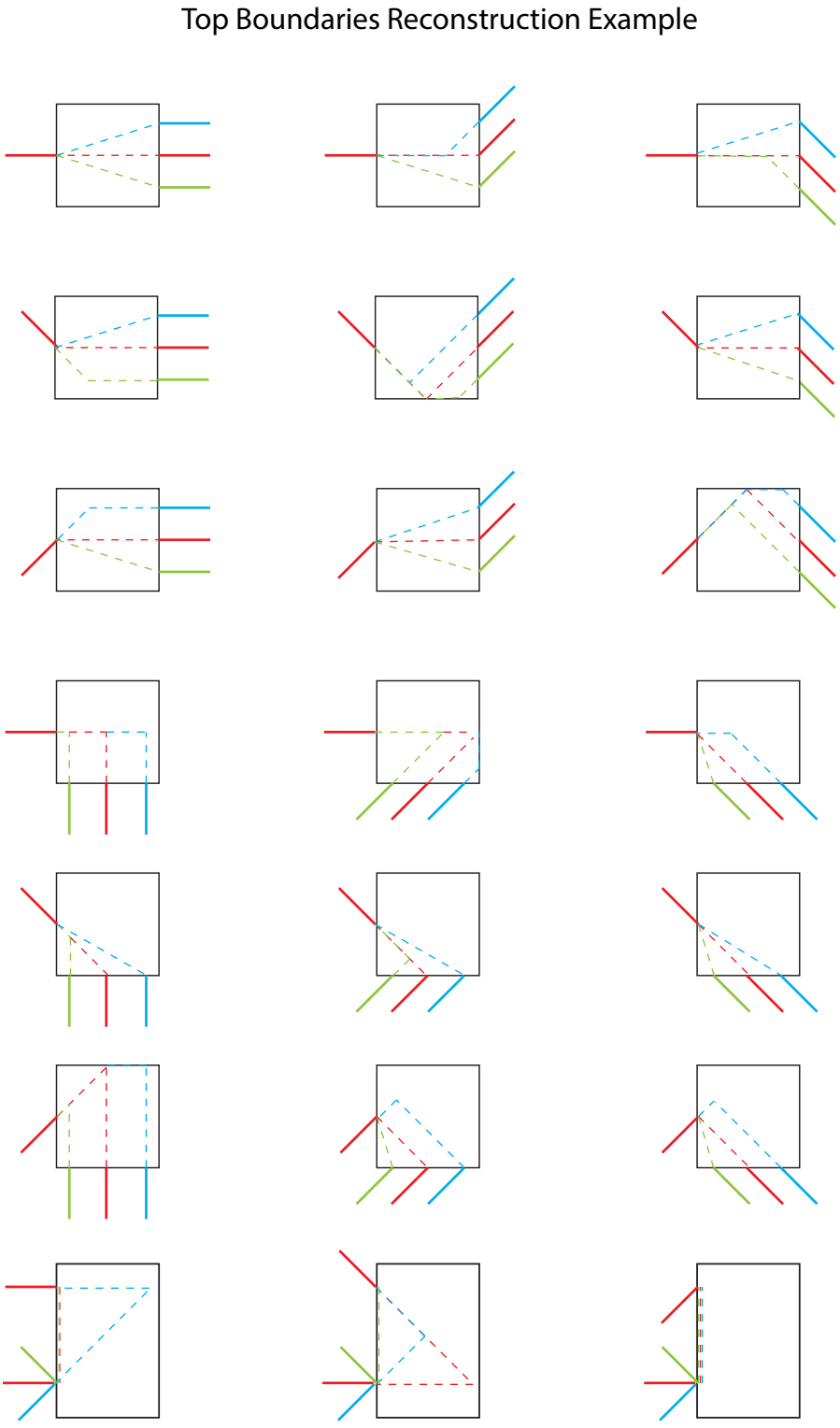


Figure 13: Example of boundaries reconstruction between a top/clockwise/linear boundary inducer to different counter-clockwise/linear inducers. The dotted line represent the interpolation boundaries between the two inducers.

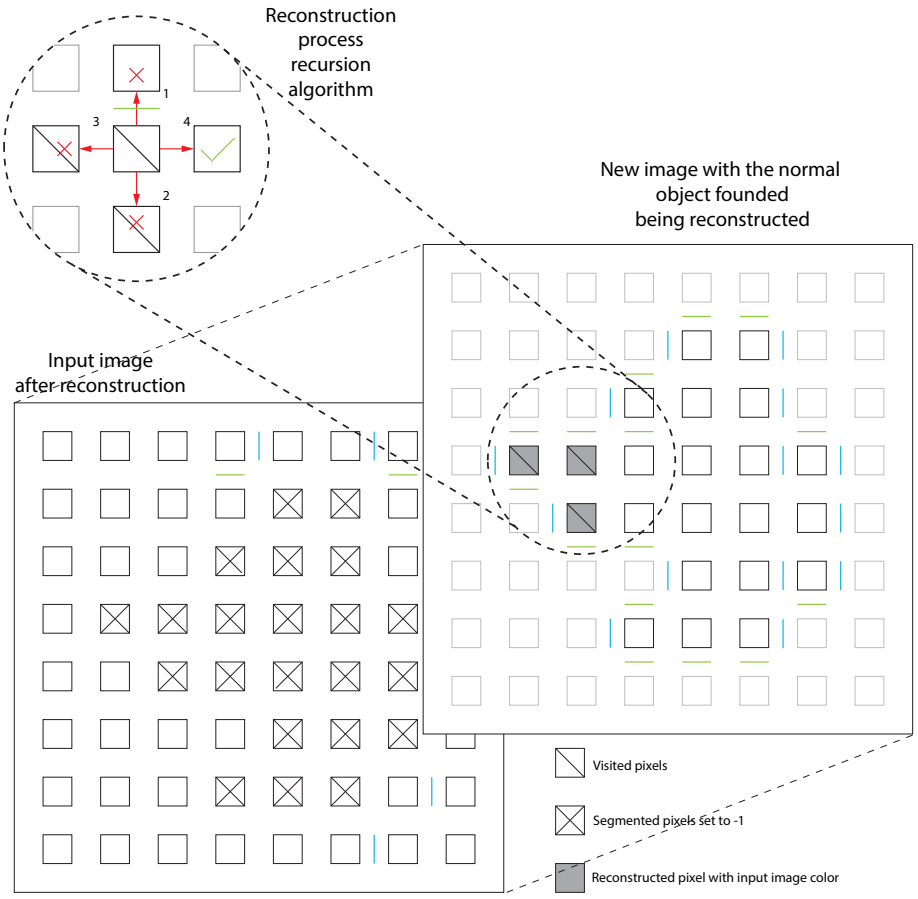


Figure 14: Schematic representation of the reconstruction of a shape. The input image is filled with negative pixels (pixel value = -1) while a new image contain the reconstructed object with its boundaries. A zoom of the pixel being reconstructed is shown in the top left border. The number represents the order in which the algorithm tries to reconstruct the neighbour. Neighbour pixel 1 blocks the recursion as a boundary is active, 2 and 3 blocked the recursion as the pixels have already been visited, the algorithm will recurse on the neighbour pixel 4.

have remaining inner boundaries around contours. See Figure 14 to observe how the algorithm would reconstruct the shape from Figure 9b

Filling Pixels In the case of an occluded object, a part of the object has been hidden and thus the reconstruction algorithm is not able to pick the value from the initial image. Therefore, when the image has a value of -1 (negative pixel), a filling process takes place to set the possible value of the pixel. As a pixel can have multiple neighbours, the algorithm will look for the 4 direct neighbours of the pixel. When a neighbour has a color, the value is added into a list. Finally, when the 4 neighbours have been identified, the pixel is filled with the mean value of the pixels contained in the list. By doing so, the filling process is able to better follow the color of a shape. Furthermore, to ensure a correct filling of the object, the algorithm makes two passes, the first one from top down and left to right, and the second pass from bottom to top and right to left. The current implementation does not fill the pixels with an average between the two passes, the purpose of the second pass is to fill pixels missed by the first pass.

3 Deep Learning model

The deep learning model used for this project is the implementation of the "Deep MNIST for Experts" TensorFlow Tutorial [7]. The model is a 2 layer CNN fully connected using a dropout of 0.5 during the training phase. For more details on the properties and implementation of the model, please refer to the tutorial's web page. To meet the requirements of the project, two slight differences were made. The first one modifies the number of training iterations. Instead of using the 20'000 total passes recommended by the tutorial, a constant number of pass per images to 20 was chosen, increasing slightly the total number of iterations to 22'000. These changes were made because we want to compare the performance of the CNN as a function of the number of images presented. The second change, was a batch configuration, that allowed us to construct pre-batch arrays for all simulations in order to remove the randomness of different image selections between simulations. By doing so, we ensure that the model always trains with the same configuration across all the conditions. The model was trained using the 55'000 images from the MNIST training set, while the accuracy is evaluated on the 10'000 images from the MNIST testing set.

Finally, as one paradigm uses images of 28x28 and the second one uses 56x28, two CNNs were implemented using the same characteristics, but one for each size of images. Characteristics of the CNN are shown in Figure 15

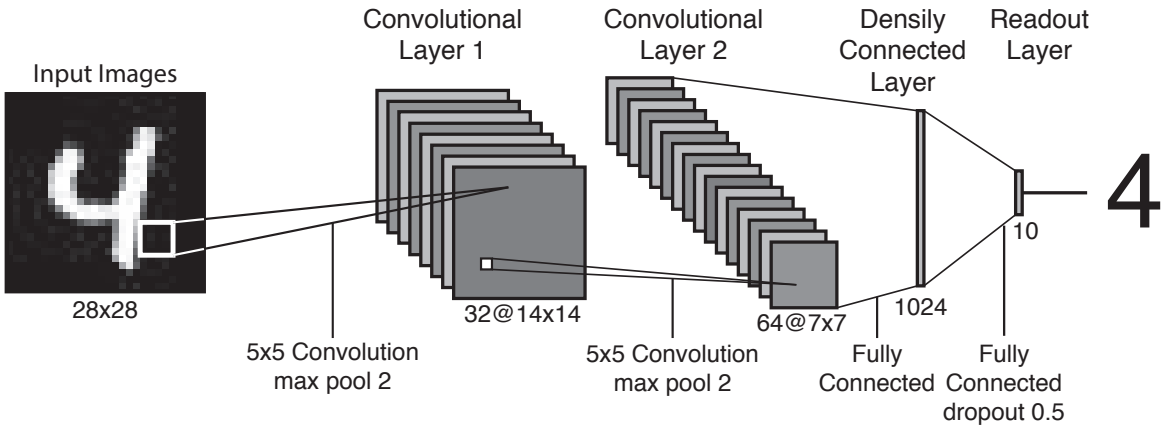
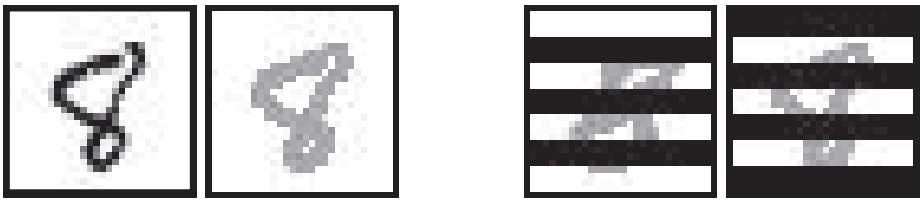


Figure 15: Convolutional neuronal network (CNN) model description of TensorFlow

4 Methods

This section will explain the paradigms used to train the CNNs. The MNIST database was chosen as it is a well known database to explore deep learning methods. It is convenient because is uses small images (28x28) that enable rather short computational time and because a tutorial in TensorFlow is available. In our case, the interesting feature is that the digits are represented in black and white images. However, a modification has been made to the database before being processed by the fast Laminart algorithm. A threshold parameter was applied to all the digits. Indeed, the algorithm tried to take care of the gradient effect, see pool shade boundary method for more information, but it often failed. A decision has been made to simply set all pixels not equal to zero to 0.4 (Figure 16b), this parameter is referred as the "Threshold". The resulting digits looks more pixelised but have the advantage of being well detected by the filters used by the algorithm.

Occlusions are created by adding horizontal bars on top of the digits. The lines are created in a way that the obstruction part made by the bars on the training digits, correspond to the visible parts of the testing digit. Thus the bars would be complementary and obstruct the whole image when merged, see Figure 16c and 16d. By doing so, we constraint the CNN information, the pixels used to train the



(a) Original (b) Threshold (c) Training (d) Testing

Figure 16: (a) - (b) Difference between an original number of the MNIST database and the same number with every pixel set to 0.4. (c) - (d) Occlusion paradigm, the training digits are occluded by 3 bars, the testing digits are occluded with 4 bars

CNN are no longer available for the recognition test. All CNN conditions were run 6 times, only the best results were kept.

Note that in terms of human behavior, this occlusion is believed by the authors to show only limited effect on human performance. In order to study the effect of the fast Laminart algorithm, several paradigms were constructed that differed in terms of the image presented to the CNN:

1. Raw: the input image used as paradigm for the fast Laminart algorithm.
2. Fast Laminart: resulting image of the input and the addition of the last occluded object created by the fast Laminart algorithm. By adding the last object to the input, the new image should keep all the original information and have the benefit from the fast Laminart algorithm.
3. Perfect Laminart: combination of the input and the threshold image, this paradigm has been created as a reference.
4. Original: the original MNIST database without any modification, it is also used as a reference.
5. Last Object: image containing only the last object created by the fast Laminart algorithm.
6. All Objects: image containing a merge of all the segmented objects found by the fast Laminart algorithm. This paradigm makes the assumption that only one object is in the picture.
7. Threshold Laminart: combination of the 0 bar for the training set and the perfect Laminart for the testing set, this paradigm is used to study how the CNN reacts to the larger image size.

The "Last Object" and the "All Objects" are used to show how the Fast Laminart algorithm will perform when using only the segmented results. Figure 17 summarizes how the paradigms are constructed.

Before describing the results, it is important to recall the aim of the project: study how preprocessing segmentation may help deep learning methods. The deep learning method here is a simple CNN applied on small images. For unoccluded images, this CNN implementation gives rather good results, with a test set score higher than 99% accuracy. In order to study the effect of the fast Laminart, the parameters of the CNN are kept unchanged for all conditions. The only change is the difference in the image size between the implementation found in the tutorial (28x28 pixels), and the one used to train the special "Laminart" paradigms (56x28 pixels). The emphasis of this point is made as the authors are not looking into the best possible results, only the difference between paradigms. A study of how well the fast Laminart algorithm could do, would require the training of CNNs with different parameters.

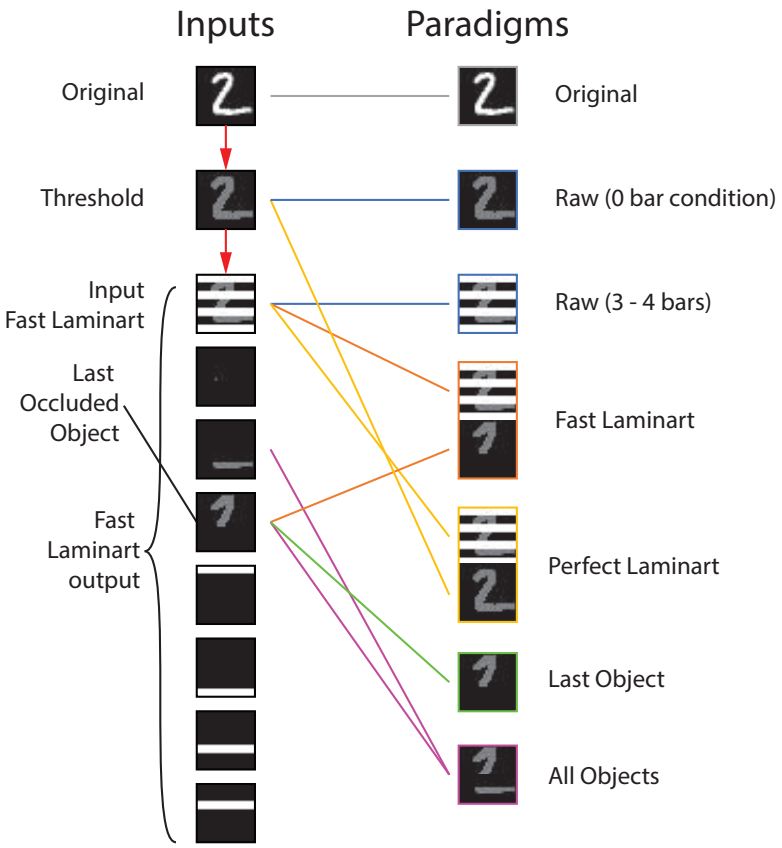


Figure 17: Paradigm construction.

5 Results

The results of the fast Laminart algorithm are treated in two sections. The first section will show the results obtain on general object segmentation and reconstruction. The second section will look more concretely on how the segmentation process may influence deep learning methods.

5.1 Object Segmentation

The first aim of the fast Laminart algorithm is to model the human visual cortex behavior. This section will show the results made by the algorithm on some home-made pictures and on a panel of good and bad results from the MNIST data base. Note that the MNIST data base is used later to train the CNNs.

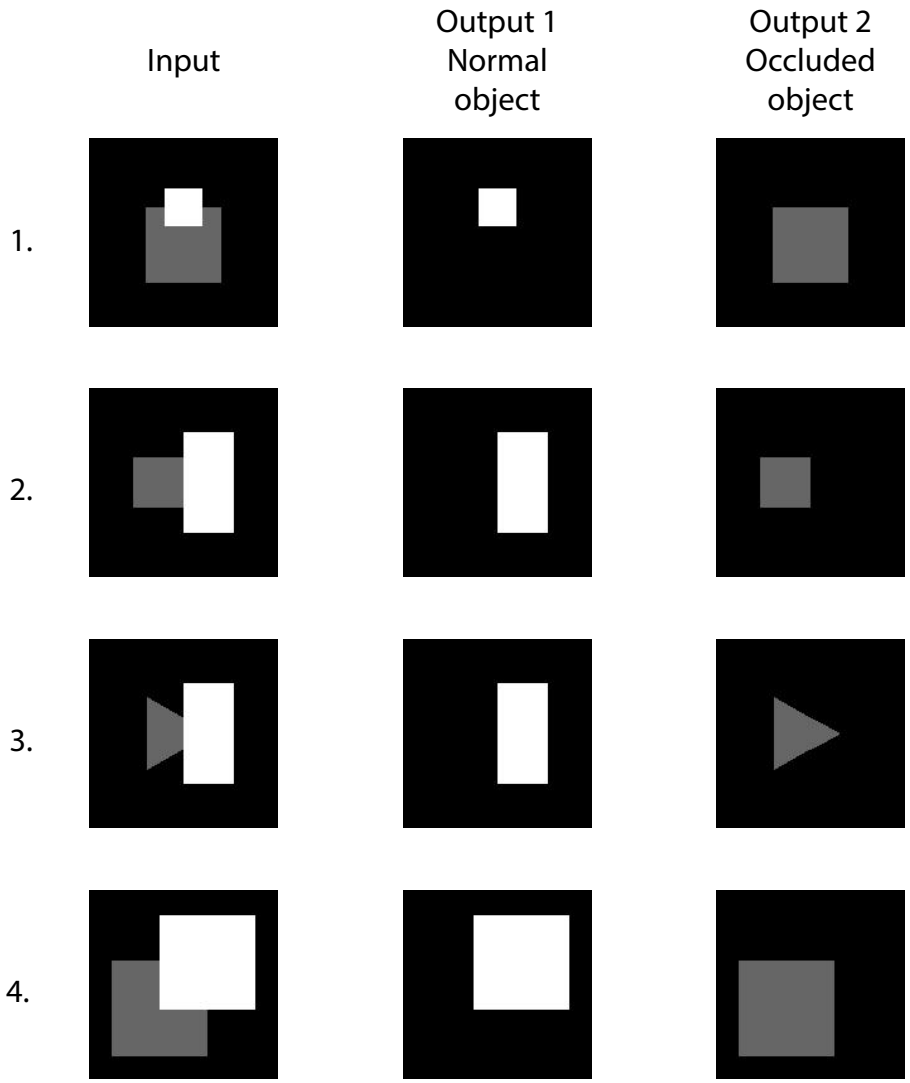


Figure 18: Example of different input feed to the fast Laminart algorithm, the Normal object is reconstructed as it is while the gray object is reconstructed.

Figure 18 shows four examples of inputs images. Each image contains two objects, a normal object (white) and an occluded object (grey) that has been reconstructed by the model. Here the grouping problem is trivial as only one pair of inducers exists. The interesting result is the output of the reconstruction process. The fast Laminart algorithm shows how the parameters of the calculated inducers help to reconstruct plausible shapes (gray objects). Case 2 describe a special case; all reconstruction cases are ill-posed problems, but due to the lack of context, case 2 shows how the algorithm chooses the simplest shape. In general, the algorithm seems to reconstruct the properties of occluded shapes in a way that is reasonable and seems similar to human behavior.

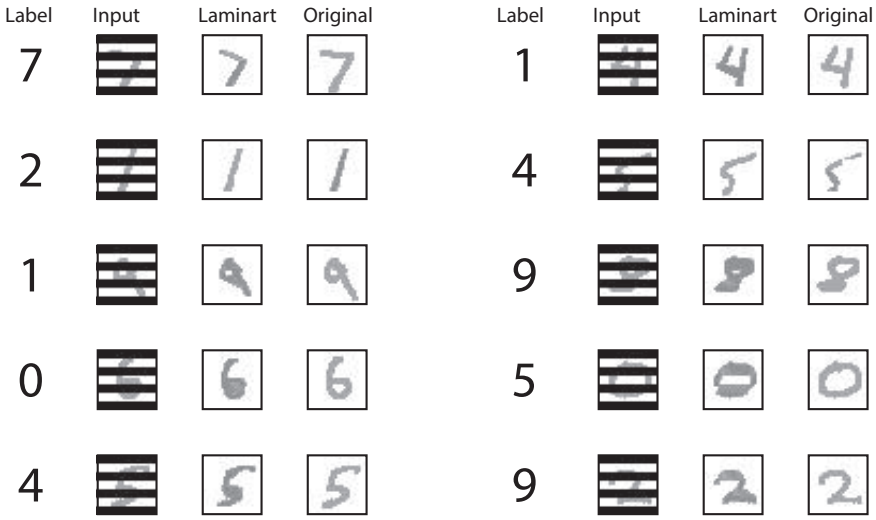


Figure 19: Example of good reconstructed numbers from the MNIST database.

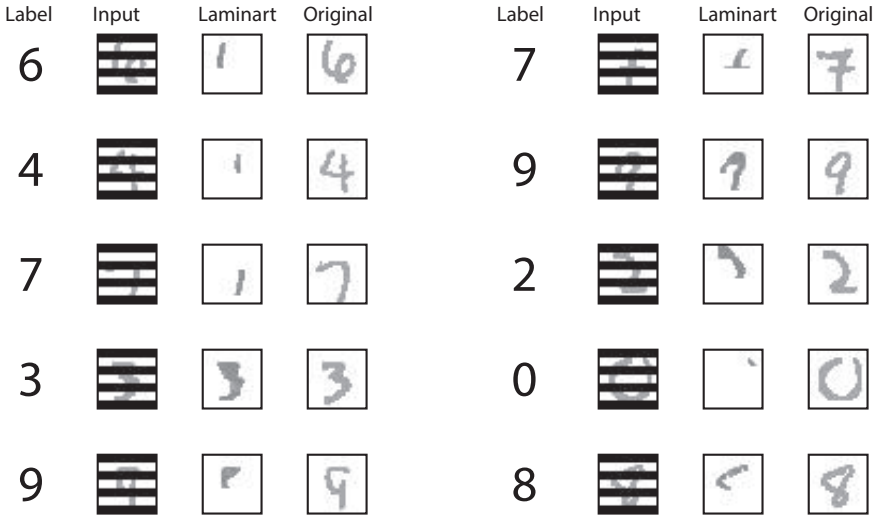


Figure 20: Example of badly reconstructed numbers from the MNIST databse.

As the MNIST database contains 70'000 handwritten digits, a selection of 10 good and 10 bad examples are shown in Figures 19 and 20. The output images shown in the panels are the first segmented object found. The grouping of inducers plays an important role in the reconstruction of objects in the MNIST database. Since many obstructing lines exist on the input images, many inducers are created and one can look at how sometimes the algorithm succeeds or fails to properly group the inducers when trying to reconstruct the digits. Another important result is the fact that the algorithm was able to process all the 65'000 digits used to train the deep learning method in less than 1 hour ($<$ 0.06 s/image).

5.2 Deep Learning Performance

We now turn to how the segmentation, grouping, and reconstruction process influences the ability of the CNN to categorize numbers from the MNIST database when the images are subjected to various types of occlusion.

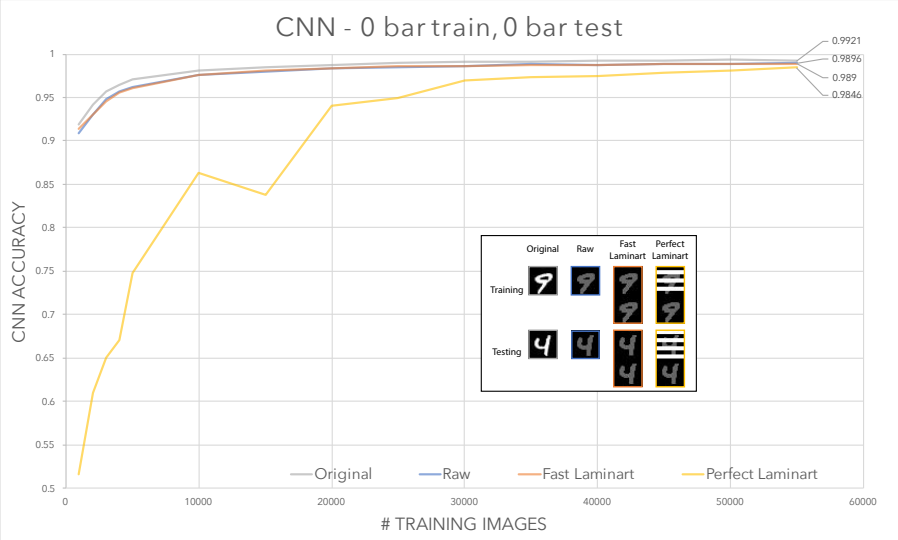


Figure 21: CNN accuracy results as a function of the number of training images used. The graph shows the paradigm of 0 bar occlusion for the training and testing set, in gray is shown as reference the original paradigm and in yellow the perfect fast Laminart.

Figure 21 displays the results of all the baseline conditions where some information is unoccluded. The gray line is the results of the unoccluded original database and shows that the tutorial has been followed correctly by reaching the final accuracy of 99.21%. The threshold condition here under the "Raw" label (blue) shows a small loss in performance due to the threshold processing (pixelised), this result can be considered as the perfect goal to reach for any further conditions. The fast Laminart (orange) shows very similar results, only .06 points behind, proving that the object recognition of the fast Laminart works properly when the boundaries are well defined, (e.g, with no occlusions). Finally, the perfect Laminart (yellow) shows that a perfect method able to reconstruct the initial digit would tend to have close results but still show a loss in performance because the CNN has to learn to mostly ignore the information in the top half of the input.

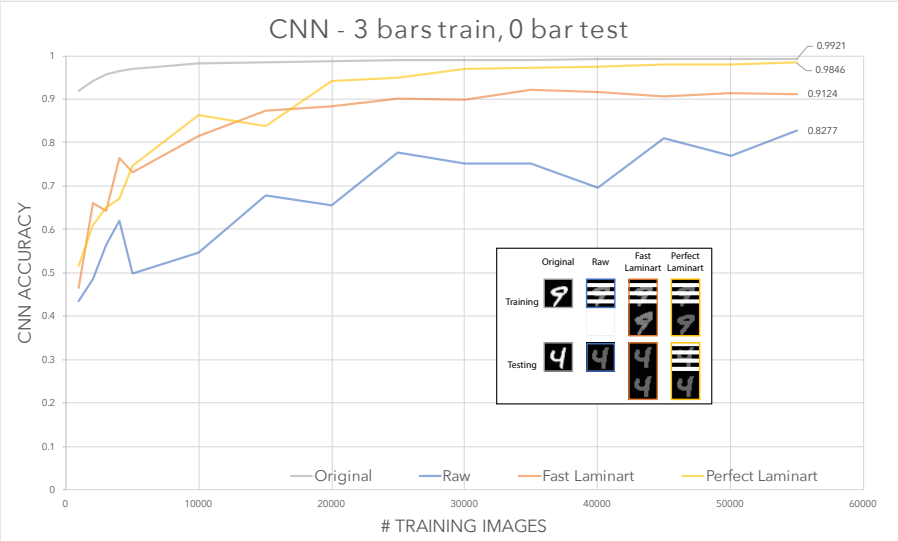


Figure 22: CNN accuracy results as a function of the number of training images used. The graph shows the paradigm of 3 bar obstruction for the training set and 0 bar for the testing set, in gray is shown as reference the original paradigm and in yellow the perfect fast Laminart.

Figure 22 could be interpreted as how the CNN would react to a low quality training database. By

using the occlusion condition only in the training phase, the graph shows how the CNN is able to use the remaining pixels to recognize the digits (blue line), the occlusion of around half of the information makes a drop in the performance of more than .16 points. The fast Laminart (orange) condition is less affected by the occlusion, showing that the addition of the reconstructed objects benefit performance. Notably, the final performance is still behind the best case by .07 points, showing how the CNN gets affected by the occlusion and its difficulty to learn using only the lower part of the image to recognize digits.

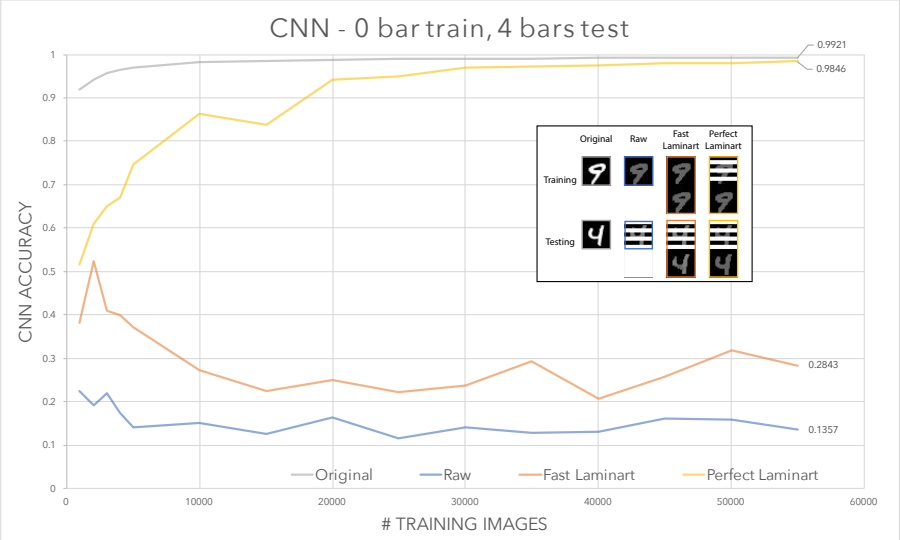


Figure 23: CNN accuracy results as a function of the number of training images used. The graph shows the paradigm of 0 bar occlusion for the training set and 4 bar for the testing set, in gray is shown as reference the original paradigm and in yellow the perfect fast Laminart.

Figure 23 could be interpreted as someone having a perfect training database but having occluded images for testing. This condition could be one of the most interesting cases for the fast Laminart algorithm. Indeed if someone with an existing classifier knows that he has bad quality test data, the use of the fast Laminart algorithm in order to augment the information of its data set should improve its results. In our case, this graph shows that the occlusion applied to the testing set strongly affects the CNN performance. When the CNN has all the information to train itself, but misses half the pixels to recognize digits (blue line) the classifier has a poor score of only 14% accuracy. These results are very similar to random guessing (10%). The fast Laminart (orange) shows here how it could help deep learning methods. By reconstructing the hidden shape behind the occlusion, the fast Laminart improves the results by a factor of two. The Perfect Laminart condition suggests that a more accurate segmentation algorithm (e.g. one that perfectly reconstructed the occluded object), could do much better still.

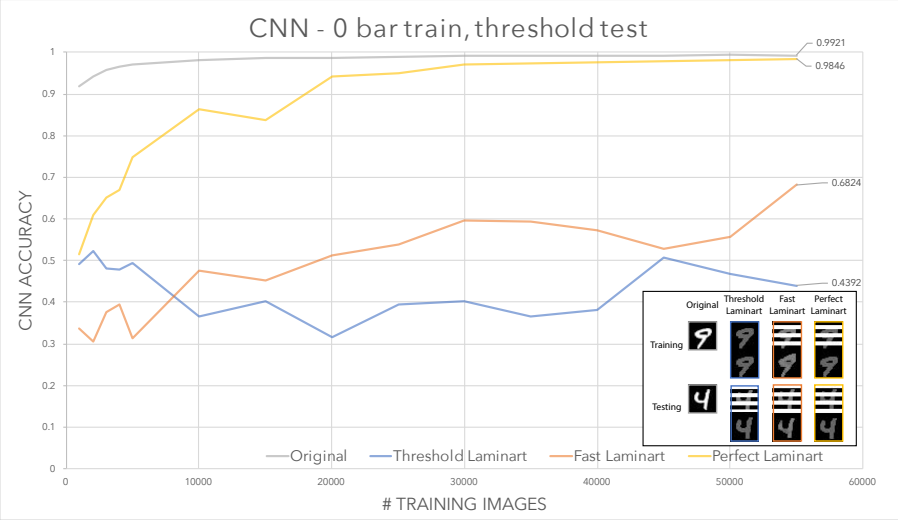


Figure 24: CNN accuracy results as a function of the number of training images used. The graph shows the threshold paradigm, consisting of 0 bar occlusion for the training set and the perfect Laminart for the testing set, in gray is shown as reference the original paradigm, in yellow the perfect fast Laminart. and in orange as reference the fast Laminart

Similarly, in Figure 24 the threshold laminart paradigm represents what would have been the results if the fast Laminart algorithm was able to perfectly reconstruct all the digits (blue). As seen in the graph, the results are still rather poor, the CNN fails to correctly discriminate more than half the digits, but shows that the current implementation of the fast Laminart algorithm scores around .15 points behind what could be achieved in principle. In other words, the poor results displayed in this graph could be interpreted as the CNN being wrongly trained/used, and they suggest that perhaps a better architecture should be applied. Indeed the algorithm here has all of the information needed to recognize the digits, yet it is hardly able to reach 50% accuracy.

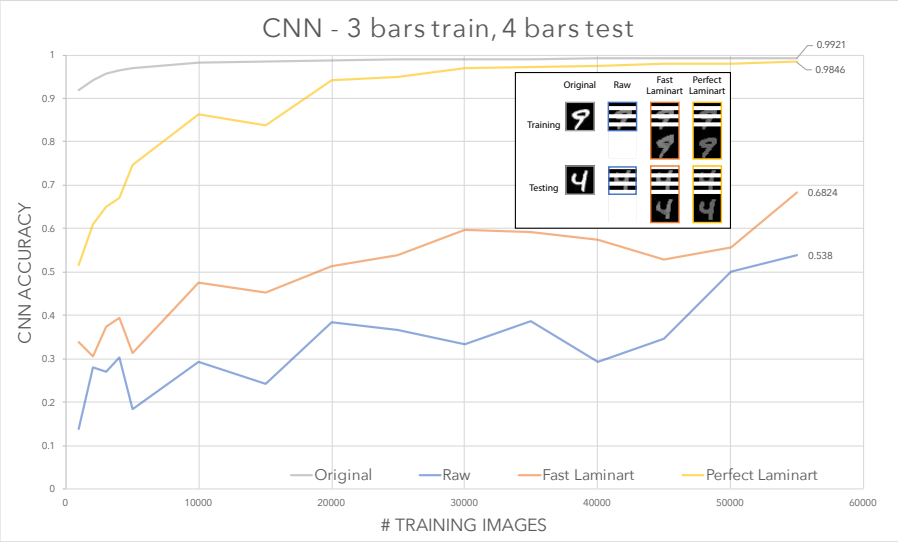


Figure 25: CNN accuracy results as a function of the number of training images used. The graph shows the paradigm of 3 bars occlusion for the training set and 4 bars for the testing set, in gray is shown as reference the original paradigm and in yellow the perfect fast Laminart.

Figure 25 shows the effect of having complementary occlusion on both the training and testing set.

As seen by the Raw paradigm (blue), using occlusion on both the training and testing set affects the performance by a factor of almost two, but the CNN does better than when using occlusion only in the testing set (Figure 23). This result shows how CNNs are meant to be trained. However, the CNN performance is still strongly affected by this occlusion paradigm. The perfect Laminart (yellow) shows the large potential that exists for pre-processing segmentation. The current fast Laminart algorithm (orange) increases performance by around .13 points compared to the raw condition (blue line). The improvement is evident. Yet it also shows that room for improvement largely exists as the current fast Laminart algorithm is more than .3 behind what would have been a perfect implementation (yellow).

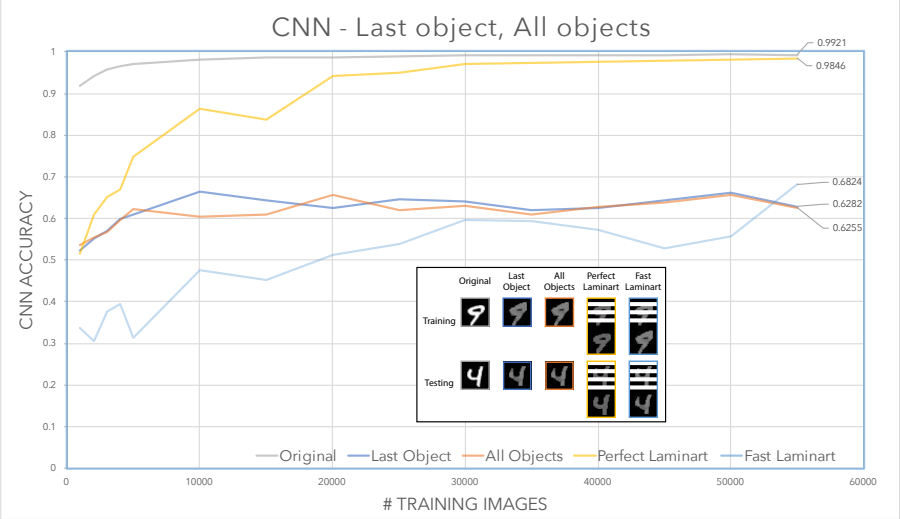


Figure 26: CNN accuracy results as a function of the number of training images used. The graph shows the paradigm of the all objects (orange) and the last object (blue) next to the Fast Laminart condition (light-blue) using 3 bars occlusion in the training set and 4 bars in the testing set as input of the fast Laminart algorithm, in gray is shown as reference the original paradigm, in yellow the perfect fast Laminart.

Finally, Figure 26 shows the results of using only the occluded objects found by the fast Laminart algorithm. The Last Object (blue) and the All Objects (orange) conditions were obtained by applying the fast Laminart algorithm to the 3 bars occlusion training set and the 4 bars occlusion testing set, but only the occluded objects are kept. In other words, the Last Object paradigm is the lower part of the Fast Laminart paradigm describe here by the light-blue line, while the All Objects has the slight difference of merging all the found occluded objects. Note that the Last Object (blue) displays the results having the fewest assumptions on the data set, while the All Objects (orange) paradigm, by merging all the reconstructed object, could be interpreted as knowing that only one object has to be found. The results show higher performance with a lower number of training data compared to the fast Laminart paradigm. As previously stipulated, the architecture of this CNN has an advantage for the smaller image size of this two paradigms. However, the Fast Laminart paradigm still obtains better performance at the end. The Last Object and the All Objects paradigm shows similar performance, yet the last Object obtains slightly better scores.

6 Discussion

The question asked by this project is: does pre-processing segmentation benefit deep learning methods? The answer is yes, see Figures 23 and graph 25. There are clear benefits when using the fast Laminart algorithm as pre-processing with the double image paradigm, created by merging in one picture the input and the reconstructed object. Figure 24 suggets that this double image paradigm decreases the performance of the CNN. Thus other CNNs might do even better.

We can argue that the size of the MNIST data base pictures are perhaps too small and the filters used by this CNN architecture may fail to overcome the occlusion effect, this could be assessed by changing the number of lines. But more especially, this decrease could probably also be explained by the small number of layers used by this CNN, indeed the size of the picture being yet twice the size using this paradigm, a larger architecture would probably deal better with the larger amount of pixels. To assess ths case, one would have to find if an architecture is able to reach the same performance as the perfect Laminart using the threshold condition. Indeed, as all the information is being present for the training and the testing set in the case of the Threshold paradigm, a good architecture should show very similar results as the Perfect Laminart. However, in the scope of this project, a single architecture was chosen to study how the fast Laminart algorithm may affect performance as we favor the difference between paradigms. Therefore, the fast Laminart increases performance. Figure 25 shows how adding the segmentation result done by the work of the current fast Laminart does give an increase of performance compared to the single image being obstructed. It is interesting to note that this first attempt of pre-segmenting algorithm already improves the performance by 126%, it is motivating as this increase could only get better. Indeed the fast Laminart algorithm could probably benefit from having better groupings, better pre-processing boundary filters and by the implementation of curve completion.

Looking at Figure 20, the current fast Laminart may induce error in the recognition performance for certain cases, as seen by the number 4, 6 and 7 that end up all by a segmented object resembling to a single small vertical bar, results that correlates with the 30% loss. Indeed, the classifier may have difficulties to discriminate between the label 1, 4, 6 and 7, thus if the classifier always outputs the label 1, three numbers out of ten are classified in the wrong category. Even surprisingly, the last object seems to have a little advantage, but this advantage could be only due to the variability of the CNN. More surprisingly, is the better performance by the single object compared to the fast Laminart paradigm while using fewer images, indeed it seems that fewer images are required to train the CNN, and the fast Laminart gets a better result only at the very end, perhaps this shows again how the architecture of the CNN may disadvantage the double image paradigms.

This project opens multiple paths for future work. Firstly, it would be interesting to run all the conditions in a bigger CNN architecture and see how this may affect the overall performance, i.e. finding a CNN able to use the double image size paradigm reaching more than 99% accuracy. Other deep learning methods would also be interesting to assess, such as simple neuronal networks or/and an unsupervised learning algorithm such as a Kohonen map method. Eventually, an updated version of the fast Laminart having better inducers/grouping methods could also be developed.

Secondly, the path to the top-down approach is yet accessible, it would be interesting to use a CNN to first segment objects on a picture, the CNN could give the position of the different objects to the fast Laminart, that could in turn use the information to see if a segmentation/reconstruction exists within the objects. The new objects would be processed by a recognition algorithm such as another CNN and the new result would be compared to the initial result. Yet this approach would require a fast Laminart able to take colors pictures and would require a more robust image processing part to construct the right boundaries.

To conclude, even if the results showed by the fast Laminart may look modest in terms of pure performance, indeed the perfect Laminart algorithm shows to be 144% better. The object perception performance done by the algorithm displayed in Figure 18, shows how the fast Laminart algorithm is able to reconstruct objects similar to human behavior. This result suggests that deep learning methods could learn with a more similar human perception. Especially if this bottom-up-top-down approach is implemented, one would probably be able to use the segmentation process to give extra information for work on a better scene understanding algorithm.

7 References

[1] Ben-Shahar, O., and Ben-Yosef, G. Tangent bundle elastica and computer vision. *IEEE Transactions on pattern analysis and machine intelligence 37*, 1 (2015).

[2] Francis, G., Manassi, M., and Herzog, M. H. Neural dynamics of grouping and segmentation explain properties of visual crowding. *Psychological Review* (2017).

[3] Kelly, F., and Grossberg, S. Neural dynamics of 3-d surface perception: Figure-ground separation and lightness perception. *Perception & Psychophysics 62*, 8 (2000), 1596–1618.

[4] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV) 115*, 3 (2015), 211–252.

[5] Simonyan, K., Vedaldi, A., and Zisserman, A. Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps. *ArXiv e-prints* (Dec. 2013).

[6] Springenberg, J. T., Dosovitskiy, A., Brox, T., and Riedmiller, M. A. Striving for simplicity: The all convolutional net. *CoRR abs/1412.6806* (2014).

[7] TensorFlow. Deep mnist for experts. `https://www.tensorflow.org/get_started/mnist/pros`.

[8] Zeiler, M. D., and Fergus, R. Visualizing and understanding convolutional networks. *CoRR abs/1311.2901* (2013).

[9] Zhou, H., Zheng, J., and Yang, X. Euler arc splines for curve completion. *Computers Graphics 36*, 6 (2012), 642 – 650. 2011 Joint Symposium on Computational Aesthetics (CAe), Non-Photorealistic Animation and Rendering (NPAR), and Sketch-Based Interfaces and Modeling (SBIM).