IODA: an Input/Output Deep Architecture for image labeling

J. Lerouge[a,*], R. Herault[a], C. Chatelain[a,*], F. Jardin[c], R. Modzelewski[b,a]

[a] *LITIS EA 4108, INSA de Rouen, Saint Étienne du Rouvray 76800, France*
[b] *Department of Nuclear Medicine, Henri Becquerel Cancer Center*
[c] *Department of Hematology & U918 (INSERM), Henri Becquerel Cancer Center*

Abstract

In this article, we propose a deep neural network (DNN) architecture called Input Output Deep Architecture (IODA) for solving the problem of image labeling. IODA directly links a whole image to a whole label map, assigning a label to each pixel using a single neural network forward step. Instead of designing a handcrafted a priori model on labels (such as an atlas in the medical domain), we propose to automatically learn the dependencies between labels. The originality of IODA is to transpose DNN input pre-training trick to the output space, in order to learn a high level representation of labels. It allows a fast image labeling inside a fully neural network framework, without the need of any preprocessing such as feature designing or output coding.

In this article, IODA is applied on both a toy texture problem and a real-world medical image dataset, showing promising results. We provide an open source implementation of IODA[1][2].

Keywords: Deep learning architectures, deep neural network, image labeling, machine learning, medical imaging, sarcopenia

1. Introduction

When dealing with a huge amount of images, the classical computer vision problems can be either i) assigning a class to an image, known as the image classification problem; ii) partitioning an image into non-overlapping regions, known as the image segmentation problem; or iii) assigning a class or a label to each pixel of an image, known as the image labeling problem (sometimes called semantic segmentation). This last problem have received a lot of attention

*Corresponding author
Email addresses: julien.lerouge@insa-rouen.fr (J. Lerouge), romain.herault@insa-rouen.fr (R. Herault), clement.chatelain@insa-rouen.fr (C. Chatelain), romain.modzelewski@chb.unicancer.fr (R. Modzelewski)

[1] http://mloss.org/software/view/562/
[2] https://github.com/jlerouge/crino

during the last years, with important needs in the analysis of medical images, natural scenes or document images.

Depending on the application domain, an image labeling problem can be very challenging. It has to deal with a lot of variability, especially when tackling a real-world domain such as medical images or natural scenes labeling. Other difficulties may also include poor quality images or a large number of classes.

One can oppose two kinds of approaches for image labeling: dedicated approaches and learning-based approaches. Dedicated approaches often rely on a priori models of the images and/or of the labels. These models can be either handcrafted, unsupervisely learned, or statistically computed on a database. In opposition, learning-based approaches directly estimate a decision function that links pixels to their labels by exploiting a labeled image database. It makes the system more versatile, at the expense of an offline supervised learning procedure.

In many difficult application domains, dedicated methods are still the state-of-the-art methods, using strong priors on the data. It is the case in medical imaging, where 2D models (atlas) are generally fitted on the new data in order to label its pixels [1, 2, 3, 4]. However, recent advances in machine learning and computer vision make the learning-based approaches more and more accurate, and we believe that they will be able to outperform dedicated methods when they are able to efficiently handle the a priori knowledge on the data.

The Input/Output Deep Architecture (IODA) is an original learning-based approach for image labeling that relies on deep neural network architectures. It directly links a whole image to a whole label map, assigning a label to each pixel using a unique neural network forward. Instead of designing a handcrafted a priori model on labels, we propose to automatically learn the dependencies between labels. The originality of IODA is to transpose DNN pre-training input trick to outputs, in order to learn a high level representation of labels. We apply it on a medical imaging labeling problem on which we outperfom the state-of-the-art method achieved by a dedicated approach based on an a registration on an a priori model [5].

The article is organised as follows: section 2 is dedicated to a review of existing learning-based approaches for image labeling tasks. In section 3 we recall the principles of neural networks and deep architectures, and we describe our IODA approach for image segmentation and labeling. The method is evaluated on a toy problem in section 4, and on a real-world medical image segmentation problem in section 5.

2. Related works on image labeling methods

From a machine learning point of view, the image labeling process is seen as a classification process, trying to find the best function f over a labeled image dataset, that minimizes the criterion $J = \mathcal{L}(Y, f(X))$, $\mathcal{L}$ being a loss function, and the domain of f is given by

$$\begin{aligned} f : X = \{x\}^{n \times m} &\rightarrow Y = \{y\}^{n \times m} \\ \mathcal{X}^{n \times m} &\rightarrow \mathcal{Y}^{n \times m} \end{aligned} \tag{1}$$

where $n \times m$ is the image size, $x \in \mathcal{X}$ are features extracted from a pixel, and $y \in \mathcal{Y}$ is the label of the corresponding pixel. For example, one can consider the raw pixels of a greyscale image as input ($\mathcal{X} = \mathbb{R}$), the raw pixels of a color image ($\mathcal{X} = \mathbb{R}^3$), or a set of p features extracted from the neighbourhood of the current pixel ($\mathcal{X} = \mathbb{R}^p$). For this latter example, the domain of f becomes $\mathbb{R}^{p \times n \times m} \rightarrow \mathcal{Y}^{n \times m}$.

In the literature, one can oppose two kinds of approach for learning-based image labeling methods:

- performing a local, independent labeling of the pixels of an image, through the distribution $p(y|x)$
- performing a global image labeling method at the image level, through the distribution $p(Y|X)$

We now describe these two kinds of approaches.

2.1. Independent pixel labeling approaches

A first straightforward method for performing image labeling using a learning approach is to perform pixel labeling using a suitable feature set (textures, color, etc.) and a classifier [6, 7] that learns the local dependencies $p(y|x)$. Features are generally computed on the neighbourhood of the current pixel. Thus only local decisions are taken and the global function f is not sought.

Moreover, as the pixel classification stage does not output homogeneous regions, these methods are often followed by a post processing segmentation stage whose aim is to reconstruct smoothed label map based on a local decision [8, 9] Nevertheless, these sequential classification-then-segmentation approaches do not modelize the whole input distribution $p(X)$, nor the whole output distribution $p(Y)$.

As shape and label areas are strongly dependent, the pixel classification and the area segmentation should be performed together. Therefore, local pixel labeling approaches appear sub-optimal. This is related to the famous segmentation/recognition issue (also known as Sayre's paradox) saying that an object cannot be recognized before being segmented, but cannot be segmented before being recognized.

2.2. Global image labeling approaches

In the general pattern recognition domain, the segmentation/recognition issue is classically circumvented using global approaches taking a whole segmentation and recognition decision.

In this kind of approaches, the global function f is estimated. Unlike the independent pixel labeling approaches, we expect from the learning process to

model the input and output distributions, $p(X)$ and $p(Y)$. State-of-the-art learning methods for image labeling are 2D-probabilistic approaches extended from 1D method such as Hidden Markov Model (HMM) and Conditionnal Random Field (CRF). Structured output SVM approaches has also been explored for sequence labeling.

In the HMM framework, the joint probability $p(X, Y)$ is modeled, implying the (false) assumption that observations X are independent [10]. The CRF overcome this problem by modeling the conditional probability $p(Y|X)$ instead of $p(X, Y)$ [11]. Probabilistic methods have proven to be effective on 1D sequences with numerous applications such as information extraction in text, handwriting and voice recognition, or even 1D-signal segmentation. These methods have been adapted to 2D-signals through either Markov Random Field (MRF) [12, 13] or 2D-CRF [14, 15, 9], but they both suffer from a time consuming and suboptimal decoding process such as HCF or ICM [16, 17]. Indeed, one has to search for the best path among the huge number of possible paths in the observation trellis which dramatically increases with the signal and output size.

In structured output SVM approaches [18] and kernel dependency estimation [19], a kernel joint projection evaluates the co-occurrence probability of an observation X and a label Y. Although these approaches can theoretically handle complex output spaces, the inference problem of finding the best label sequence knowing the model is a hard problem. It prevents the approach from tackling problems where the dimension of the sequence is large, as it is the case for image segmentation.

In this paper, we assume that other machine learning methods such as neural network are able to perform a global image segmentation and labeling task, modeling the underlying problem of estimating $p(Y|X)$.

Estimating $p(y|X)$, even if X has a great dimension, can be achieved through Deep Neural Network (DNN) using unsupervised pre-training or regularized learning process, through the modelization of $p(X)$. The learning of $p(X)$ can be performed either independently [20, 21] or jointly [22, 23] to the learning of $p(y|X)$. These approaches have shown to be efficient on numerous problems such as natural language processing [24], speech recognition [25] or handwriting recognition [26].

In this work, we propose to address directly the image labeling problem, that is the estimation of $p(Y|X)$. Our key idea is to extend the DNN input pre-training and adapt it to the output pre-training, providing the label distribution $p(Y)$.

While input pre-training has given to neural networks the ability to deal with high dimensional input space, we assume that output pre-training allows neural networks to deal with high dimensional output space.

The next section is dedicated to a recall on neural networks, before presenting our approach.

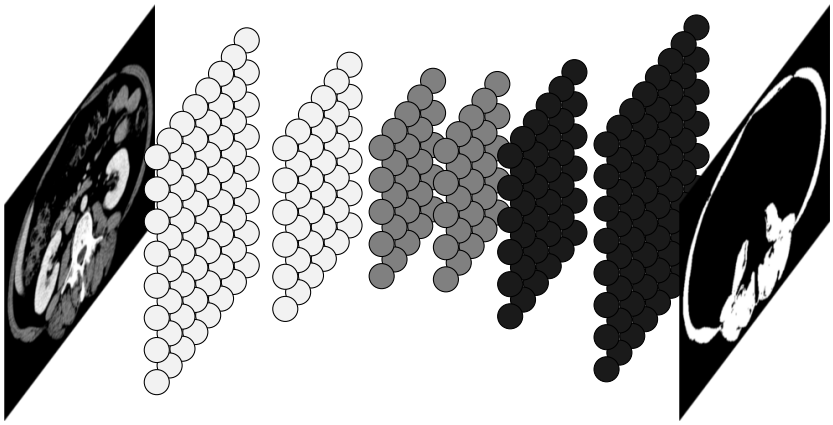


Figure 1: The IODA architecture. It directly links the pixel matrix to the label matrix. The input layers (left, light) are pre-trained to provide a high level representation of the pixels, while the output layers (right, dark) are pre-trained to learn the a priori knowledge of the problem.

3. Input/Ouput Deep Architecture (IODA)

For the image labeling task, we choose a global approach where a Deep Neural Network (DNN), a kind of Feedforward Artificial Neural Network (FANN), is used as the global decision function f.

The FANN architecture is a common artificial neural network topology used for supervised learning. In this architecture, information is processed by a sequence of computational layers. At decision step, information always flows from input to output, without any feedback. A FANN can be composed of 1, 2 or many layers [27]. In order to model all continuous functions on compact subsets of $\mathbb{R}^n$ or logical function like XOR, at least 2 computational layers must be involved. A FANN with 2 or more layers is called a Multi-Layer Perceptron (MLP). A MLP with more or far more than 2 layers can also be called a Deep Neural Network (DNN). DNN are typically used in image problems like character recognition [28]. A MLP is usually learned by an algorithm called gradient back-propagation, that cleverly performs a gradient descent through the layers.

Nevertheless, the deepest layers of a DNN are hardly trained by this technique. To help the learning of a DNN, an unsupervised pre-training is performed on deepest layers, through the use of auto-encoders (AE) which learn feature distribution [20, 21, 29, 30, 31].

In Input Output Deep Architecture (IODA), we propose to use the pre-training trick with AE not only for the input space but also for the output space, in order to learn the labels distribution as well as the features distribution. The global architecture of IODA is presented in Figure 1.

3.1. Notations and building blocks

In this section, we present the notations that will be used in this work. Then we discuss how to build a DNN using the input pre-training trick with AE, and eventually how to build a IODA with the same principle adapted to the output space.

In the preceding section, the input was a two-dimensional image matrix X and the output a label map Y. In this section, we will consider that input and output are one-dimensional flatten versions of the data, respectively an input vector $\mathbf{x}$ and a label vector $\mathbf{y}$.

3.1.1. Baseline Multi Layer Perceptron

We denote :

- a *layer*, the unit of computational operations,
- a *representation*, the unit of data.

Within this framework, the smallest MLP with universal approximation property has 2 layers (an input layer and an output layer), whereas this very same MLP has 3 representations (an input, a hidden and an output representations).

Let us consider a MLP of N layers. Each of the $N+1$ representations is denoted $\mathbf{r}_l$ with $l \in [0 \dots N]$. $\mathbf{r}_0$ is the input representation, i.e. the features $\mathbf{x}$, and $\mathbf{r}_N$ the output representation, i.e. the estimated labels $\hat{\mathbf{y}}$.

Each layer l performs the following operation at forward step,

$$\mathbf{r}_l = f_l(\mathbf{W}_l \times \mathbf{r}_{l-1} + \mathbf{b}_l) \tag{2}$$

where $\mathbf{r}_{l-1}$ and $\mathbf{r}_l$ are respectively the input and the output of the layer l, $\mathbf{W}_l$ a matrix representing a linear transformation corresponding to neuron *weights*, $\mathbf{b}_l$ a vector of offsets corresponding to neuron *biases*, and f_l a non-linear differentiable transformation corresponding to neuron *activation function*. If $\mathbf{r}_{l-1} \in \mathbb{R}^m$ and $\mathbf{r}_l \in \mathbb{R}^n$, then $\mathbf{W}_l$ is in $\mathbb{R}^{n \times m}$ and $\mathbf{b}_l$ in $\mathbb{R}^n$. The lower the index l of a representation $\mathbf{r}_l$ or of a layer $(f_l, \mathbf{W}_l, \mathbf{b}_l)$ is, the *deeper* it is. At the opposite the greater the index is, the *higher* the representation or the layer is. Figure 2 sums up the adopted notations on a 2-layer perceptron.

This is a gradient machine, i.e. the criterion (e.g. squared error, negative log likelihood, cross entropy ...) that is used to train the machine is differentiable according to its parameters. Thus the machine can be trained by gradient descent. In MLP, parameters are modified layer by layer backward from the output layer to the input layer. This is the so-called gradient back-propagation algorithm.

3.1.2. Auto-Encoder

An Auto-Encoder (AE) consists in a 2-layer MLP (see Figure 3). It tries to recover its input $\mathbf{x}$ at its output $\hat{\mathbf{x}}$ [32].

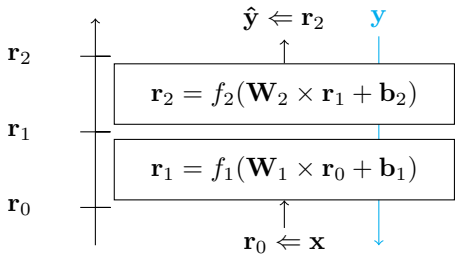


Figure 2: A 2-layer perceptron with adopted notation, input $\mathbf{x}$, output $\hat{\mathbf{y}}$ and target $\mathbf{y}$

The first layer applies a transformation from the input space $\mathbb{R}^m$ to a hidden space $\mathbb{R}^n$, the second layer inverts this transformation, i.e. does a back-projection in the original input space. When $m < n$ the first layer performs a compression of the data, and the second layer a decompression. The first layer is called *encoding layer*, the last layer is called *decoding layer*.

We denote the linear transformations ($\mathbf{U} \in \mathbb{R}^{n \times m}$, $\mathbf{a} \in \mathbb{R}^n$) and ($\mathbf{V} \in \mathbb{R}^{m \times n}$, $\mathbf{c} \in \mathbb{R}^m$), and the non-linear transformations g and h, for the encoding layer and for the decoding layer respectively. The compressed or encoded representation at the output of the encoding layer is noted $\mathbf{e}$ and conversely the decompressed or decoded output representation of the decoding layer is noted $\mathbf{d}$. Obviously, the input representation $\mathbf{x}$ and the decoded representation $\mathbf{d}$ have the same size m.

An AE is learned through the back-propagation algorithm with $\mathbf{x}$ as input and as target, and $\mathbf{V} = \mathbf{U}^\intercal$ at initialization. Noise or transformations can be applied to $\mathbf{x}$ solely at the input to increase its generalization power.

At decision, when you give an example $\mathbf{x}_k$ to an AE it estimates the example itself $\hat{\mathbf{x}}_k$. If $\mathbf{x}_k$ and $\hat{\mathbf{x}}_k$ are similar, $\mathbf{x}_k$ is likely to happen according to the training set $\mathcal{X}$; if $\mathbf{x}_k$ and $\hat{\mathbf{x}}_k$ are dissimilar, $\mathbf{x}_k$ is not likely to come from the same phenomenon that gives the training set $\mathcal{X}$: that is the modeling of $p(\mathbf{x})$.

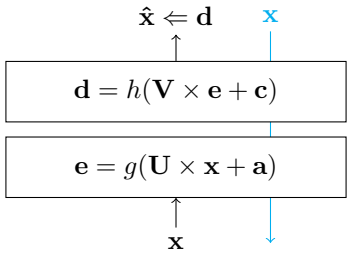


Figure 3: A 2-layer auto-encoder, input $\mathbf{x}$, output $\hat{\mathbf{x}}$ and target $\mathbf{x}$

3.2. DNN extension and Input/Ouput Deep Architecture (IODA)

In DNN, the deepest layers are used to transform the input space into a simpler yet more suitable space for the supervised task. Nevertheless, back-propagation is not efficient to train these deepest layers from random initialized parameters.

To prevent the DNN to fall into a local minimum far from a good solution, a smart initialization of the parameters is undertaken. This *pre-training* strategy consists in learning auto-encoders in an unsupervised way and stacking only their first layer to built a DNN with the desired architecture. Thereafter, the DNN is trained with a standard back-propagation supervised learning. In this way, the pre-training enables the first layers of the DNN to build a smarter representation of the input space to simplify the supervised task.

In order to address high dimensional but correlated output space, such as label map in image labeling problem, we propose to use the same AE trick as for DNN this time not only on the input space but also on the output space. AE are learned backward from the targets, in order to learn their distribution over the output space $p(\mathbf{y})$. The aim is to simplify the final supervised task by reducing the output representation. We call this architecture *Input/Ouput Deep Architecture* (IODA).

To sum up, the IODA training involves:

- an unsupervised pre-training of the input layers,
- an unsupervised pre-training of the output layers, which is specific to IODA,
- a final standard back-propagation supervised learning.

Figure 4 displays the whole process on a 5-layer MLP.

3.2.1. Pre-training of input layers

Input pre-training with AEs occurs forward from the deepest layer. At each step, the encoding part of AEs are kept aside to initialize the final IODA.

Figure 4(a) shows the pre-training of the two first (deepest) layers of a 5-layer IODA. We note m the size of the input representation of the IODA, n the size of the first hidden representation and o the size of the second one.

For the first step of the input pre-training (Figure 4(a) left), an auto-encoder is trained with back-propagation on the input representation of the IODA. The size of the input representation and of encoded representation in this AE should be respectively m and n to mimics the final IODA. Thus if $\mathbf{W}_1$ of the IODA is in $\mathbb{R}^{n\times m}$ then $\mathbf{U}_1$ of the auto-encoder must be in $\mathbb{R}^{n\times m}$. Moreover, the non linear transformation g_1 of AE should be the same as f_1 the first non-linear transformation of the IODA.

After the training of the AE, the linear transformation ($\mathbf{U}_1$,$\mathbf{a}_1$) of the encoding layer is kept aside to initialize the first layer of the IODA. Furthermore, the encoded representation $\mathbf{e}_1$ of all the training examples is also kept in order to feed the second step.

For the second step of the pre-training (Figure 4(a) right), we repeat the latter operation for the second layer of the IODA. An other auto-encoder is trained with back-propagation on $\mathbf{e}_1$ the encoded representation from the first auto-encoder. This time, the size of the input representation and of encoded representation in the AE should be respectively n and o; and the non-linear transformation g_2 be the same as f_2.

At the end of that step, the linear transformation ($\mathbf{U}_2$, $\mathbf{a}_2$) of the encoding layer is kept aside to initialize the second layer of the IODA.

Stacking more AE can be repeated if more input layers than in the given example are involved.

Eventually, input layers of desired final IODA is initialized by the weights computed on this input pre-training step (Fig 4(c)),

- $\mathbf{W}_1 \leftarrow \mathbf{U}_1$, $\mathbf{b}_1 \leftarrow \mathbf{a}_1$, for the first layer,
- $\mathbf{W}_2 \leftarrow \mathbf{U}_2$, $\mathbf{b}_2 \leftarrow \mathbf{a}_2$, for the second layer.

3.2.2. Pre-training of output layers specific to IODA

Operations are the same than in the latter input pre-training with two exceptions: they are undertaken backward from the highest layer and the parameters kept aside for the initialization of the final IODA are from the decoding part of AEs.

Figure 4(b) shows the pre-training of the two last (highest) layers of a 5-layer IODA.

A first pre-training step (Fig 4(b) left) is done with an auto-encoder on the label vector $\mathbf{y}$. The second linear transformation $\mathbf{V}_5$ of the AE should have the same shape as $\mathbf{W}_5$, the last linear transformation of the IODA. Furthermore, the second non-linear transformation h_5 of the AE should be the same as f_5 , the last non-linear transformation of the IODA.

Let's be careful, this time it is the parameters of the second layer of the AE, i.e. the decoding layer, which are kept contrary to the standard DNN pre-training. Thus, after training the AE, the linear transformation ($\mathbf{V}_5$, $\mathbf{c}_5$) is saved to initialize the last layer of the IODA. Moreover, $\mathbf{e}_5$ the encoded representation for all the training examples is kept in order to feed the next step.

A second pre-training step (Fig 4(b) right) is done with an other auto-encoder on $\mathbf{e}_5$. The second non-linear transformation h_4 of this AE should be the same as the penultimate non-linear transformation f_4 of the IODA, as well as the shape of its second linear transformation $\mathbf{V}_4$ should be the same shape as $\mathbf{W}_4$ the penultimate linear transformation of the IODA.

At the end of that step, the linear transformation ($\mathbf{V}_4$, $\mathbf{c}_4$) of the decoding layer is kept aside to initialize the penultimate layer of the DNN.

As for the input pre-training, these operations can be repeated and more AEs stacked if the architecture consists in more output layers.

Eventually output layers of the desired IODA are initialized by the weights computed on precedent pre-training steps:

- $\mathbf{W}_4 \leftarrow \mathbf{V}_4$, $\mathbf{b}_4 \leftarrow \mathbf{c}_4$, for the penultimate layer,
- $\mathbf{W}_5 \leftarrow \mathbf{V}_5$, $\mathbf{b}_5 \leftarrow \mathbf{c}_5$, for the last layer.

3.2.3. Final supervised training

After pre-trainings of input layers and output layers, a standard back-propagation is undertaken with target $\mathbf{y}$ (Fig 4(c)) on the whole MLP.

Let note that in the 5-layer architecture given as an example it exists a *link* layer, the layer number 3, between the input layers and output layers. It is not pre-trained and thus it has randomized parameters before the last back-propagation. A slightly different approach may supervisedly train this link layer before doing a last full back-propagation at a risk of over-fitting.

3.2.4. IODA training algorithm

The algorithm 1 describes the whole learning procedure for training a IODA.

We assume the existence of these two functions:

- $X' \leftarrow$MLPFORWARD($[\mathbf{W}_1, .., \mathbf{W}_K]$,$X$) that propagates X through layers $[\mathbf{W}_1, .., \mathbf{W}_K]$,
- $[\mathbf{W}'_1, .., \mathbf{W}'_K] \leftarrow$MLPTRAIN($[\mathbf{W}_1, .., \mathbf{W}_K]$,$X$,$Y$) that trains layers $[\mathbf{W}_1, .., \mathbf{W}_K]$ using back-propagation algorithm according to a labeled dataset (X,Y).

With this notation an AE is trained by $[\mathbf{U}, \mathbf{V}] \leftarrow$MLPTRAIN($[\mathbf{W}, \mathbf{W}^\intercal]$,$X$,$Y$) where $.^\intercal$ denotes the transposition. Then we can drop $\mathbf{V}$ if we want to keep the encoding part only, or drop $\mathbf{U}$ if we want to keep the decoding part.

For the sake of clarity, hyperparameters such as non-linear function of each layer does not appear in the algorithm, and all the parameters of a layer i (the linear transformation $\mathbf{W}$ and the bias b) are gathered into the generic variable $\mathbf{W}_i$.

Algorithm 1 Simplified IODA training algorithm

Input: X, a training feature set of size $Nb_{\text{examples}} \times Nb_{\text{features}}$
Input: Y, a corresponding training label set of size $Nb_{\text{examples}} \times Nb_{\text{labels}}$
Input: N_{input}, the number of input layers to be pre-trained
Input: N_{output}, the number of output layers to be pre-trained
Input: N, the number of layers in the IODA, $N_{\text{input}} + N_{\text{output}} < N$
Output: $[\mathbf{W}_1, \mathbf{W}_2, \ldots, \mathbf{W}_N]$, the parameters for all the layers

Randomly initialize $[\mathbf{W}_1, \mathbf{W}_2, \ldots, \mathbf{W}_N]$

Input pre-training

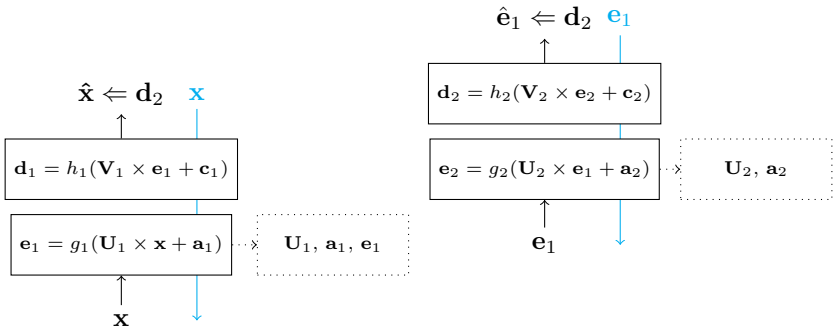
$R \leftarrow X$
for $i \leftarrow 1..N_{\text{input}}$ **do**
  {*Training an AE on R and keeps its encoding parameters*}
  $[\mathbf{W}_i, \mathbf{W}_{\text{dummy}}] \leftarrow \text{MLPTrain}([\mathbf{W}_i, \mathbf{W}_i^\intercal], R, R)$
  Drop $\mathbf{W}_{\text{dummy}}$
  $R \leftarrow \text{MLPForward}([\mathbf{W}_i], R)$
end for

Output pre-training

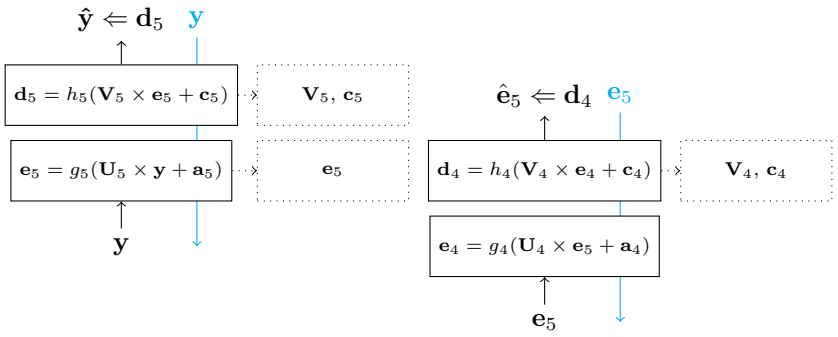
$R \leftarrow Y$
for $i \leftarrow N..N - N_{\text{output}} + 1$ **step** -1 **do**
  {*Training an AE on R and keeps its decoding parameters*}
  $[\mathbf{U}, \mathbf{W}_i] \leftarrow \text{MLPTrain}([\mathbf{W}_i^\intercal, \mathbf{W}_i], R, R)$
  $R \leftarrow \text{MLPForward}([\mathbf{U}], R)$
  Drop $\mathbf{U}$
end for

Final supervised learning

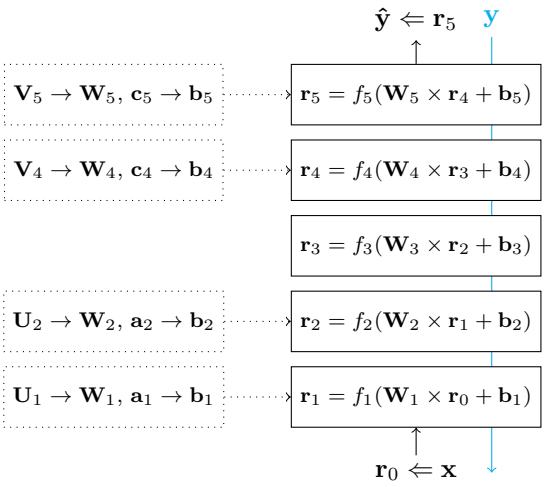
$[\mathbf{W}_1, \mathbf{W}_2, \ldots, \mathbf{W}_N] \leftarrow \text{MLPTrain}([\mathbf{W}_1, \mathbf{W}_2, \ldots, \mathbf{W}_N], X, Y)$



(a) Input pre-training. Left : Learning of the first AE, input $\mathbf{x}$, output $\hat{\mathbf{x}}$, target $\mathbf{x}$ and $g_1 \leftarrow f_1$. Right : Learning of the second AE, input $\mathbf{e}_1$ which is the encoded representation the first AE, output $\hat{\mathbf{e}}_1$, target $\mathbf{e}_1$ and $g_2 \leftarrow f_2$.



(b) Output pre-training Left : Learning of the first AE, input $\mathbf{y}$, output $\hat{\mathbf{y}}$, target $\mathbf{y}$ and $h_5 \leftarrow f_5$. Right : Learning of the second AE, input $\mathbf{e}_5$ which is encoded representation of the first AE, output $\hat{\mathbf{e}}_5$, target $\mathbf{e}_5$ and $h_4 \leftarrow f_4$.



(c) Final IODA, with pre-computed initial weights, input $\mathbf{x}$, output $\hat{\mathbf{y}}$ and target $\mathbf{y}$

Figure 4: Pre-trainings and training of a 5-layer IODA

4. Texture recognition experiments

We developed a Python library, named Crino, based on the Theano library[33]. It allows to build and train neural networks with a modular architecture, including IODA. Crino is available online[3][4] and is free to use for further research.

To demonstrate the validity of our proposition, we have performed experiments on a toy image dataset. We first describe the dataset, then the different experimental setups. We finally present and discuss the results we have obtained.

4.1. Toy dataset

We have generated a toy image dataset for a texture recognition task. The input examples are artificial images composed of two textures, taken from the Brodatz texture archive [5]. The background is taken from Texture 77, on top of which is drawn the foreground with Texture 17. The foreground consists in the portion of a disk included between two concentric circles whose center and radii are variable (randomly chosen for each sample). The labels are binary images denoting the class of the pixels, 0's are for the background pixels and 1's are for the foreground pixels. All images are 128×128 pixels, and inputs are normalized between 0 and 1. Our training and validation sets are composed of 500 images each. Two examples of the validation set are shown on Figure 5.

On this kind of images, the internal dependencies among the label structures are very high, and therefore constitutes a suitable problem for evaluating the IODA abilities. A better representation of the output space should be available and pre-training on output should improve the results of this supervised task.

4.2. Experimental setup

For this toy problem, we have built using Crino a 3-layer, 4-layer and 5-layer neural networks with a MSE criterion. For all of them, the size of the input and output representations is 128×128.

For 3-layer architectures, we have tested four hidden representation geometries: (256,256), (512,512), (1024,1024) and (2048,2048) neurons. Input pre-training has been performed from 0 to 2 layers, and output pre-training from 0 to 2 layers also. Let us emphasise that the total number of pre-trainings can not exceed 2 since at least one layer must be free of autoencoding pre-training. Finally, 4×6 setups have been trained and evaluated. Setups that share the same number of hidden neurons starts with the same initialisation weights. The results are gathered in Table 1.

For 4-layer architectures, the same procedure has been applied. Four hidden representation geometries have been evaluated: (256,128,256), (512,256,512), (1024,512,1024) and (2048,1024,2048) neurons. Input and output pre-trainings

[3] `http://mloss.org/software/view/562/`
[4] `https://github.com/jlerouge/crino`
[5] `http://www.ux.uis.no/~tranden/brodatz.html`

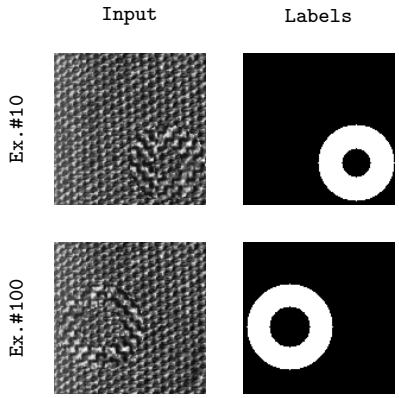


Figure 5: Two random validation examples (left) artificial image inputs (right) image labels

vary each from 0 to 3 layers, with a total number of 3 pre-trainings. The 4×10 setups results are presented in table 2.

For 5-layer architectures, four hidden representation geometries have been evaluated: (256,128,128,256), (512,256,256,512), (1024,512,512,1024) and (2048,-1024,1024,2048). Only some pre-training configurations have been tested, including the most successful strategies within 3 and 4 layers architectures (please refer to the next subsection).

If the setup contains at least one input pre-trained layer and one output pre-trained layer, it falls into our proposed IODA framework.

4.3. Parameterization

For each autoencoder, the weights are randomly initialized according to the work of [34] in order to perform a faster convergence of the gradient backpropagation algorithm. It uses a uniform distribution $\mathcal{U}(-l, l)$, where $l = \sqrt{\frac{6}{mn}}$ and where m and n are respectively the input and the output sizes of the autoencoder. The biases are initially null.

For input and output pre-trainings, auto-encoders are trained with a batch gradient descent of 100 images, controlled by a validation set in order to minimize error while avoiding overfitting. Input pre-training has therefore been stopped after 300 iterations, while only 100 iterations were enough for output pre-training since a strong overfit appeared around 200 iterations. The final supervised learning is also performed with a batch gradient descent. As we have 500 training examples, it means that the parameters are updated five times per iteration.

We have chosen an adaptive learning rate, i.e. that is varying at each iteration. Our strategy consists in increasing the learning rate after some fixed number of consecutive iterations that improves the learning criterion. In case of a degradation, we keep decreasing the learning rate until it provides a better

value of the criterion in comparison to the previous iteration. The initial value of the learning rate is 10 (same value for all the architectures).

4.4. First qualitative results

For a first qualitative result, we propose to focus on a given geometry with three different pre-training strategies. The considered geometry has 3 layers and 256 units in each hidden representations. We call these three strategies are NDA, IDA and IODA :

NDA - DNN without pre-training : No pre-training is done at all, solely a supervised learning with standard back-propagation is achieved.

IDA - DNN with input pre-training : The input layer is pre-trained using an auto-encoder on the input data. Then a supervised fine tuning is achieved.

IODA - DNN with input and output pre-training : The input and output layers are pre-trained using respectively an auto-encoder on the input data and an auto-encoder on the output data. A supervised fine tuning is then achieved.

Figure 6 shows the output of each architecture for the first example shown in Figure 5, after an increasing number of supervised iterations (pre-training has already been performed). As one can see, the best results are achieved with the input and output pre-trained architecture (IODA), while input pre-trained architecture (IDA) outperforms the non pre-trained architecture (NDA). One can also observe that the global output structure has already been learned by IODA after only 10 supervised learning iterations. It shows that the IODA strategy is much more efficient than the IDA strategy, as it speeds up the supervised learning. Finally, after a significant number of iterations, IODA is able to locate more accurately the texture change.

4.5. Quantitative results

Tables 1 and 2 shows a quantitative evaluation over the whole test dataset (500 images), using all the setups defined above. Several comments can be made out of these experiments.

First, one can observe that the results are strongly dependent from the setup. The pre-training strategies seems to have more influence on the results than the number of layer and the number of hidden units. One can notice that 3-layer or 4-layer architectures lead to very close performance. Globally, input pre-trained setups are ranked first. Among them, IODA setups, i.e. with at least one pre-trained input layer and one pre-trained output layer, achieve the best results. This demonstrates the interest of our approach. Moreover, setups with pre-trained output layers only do not guarantee good results.

The results of 5 layer architectures are globally worse than those of 3 and 4 layer architectures and for this reason not entirely reported in this article.

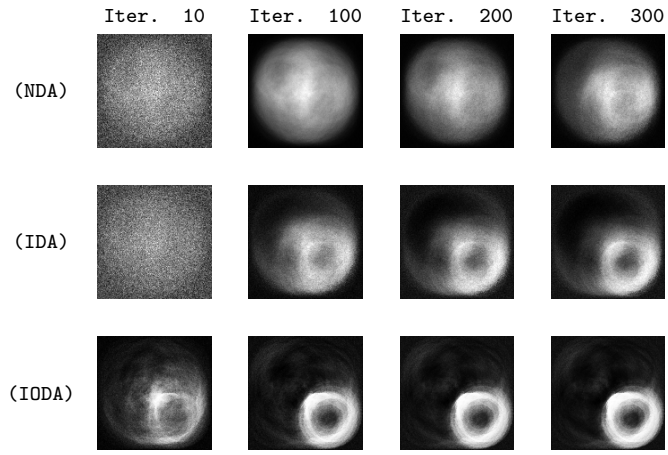


Figure 6: Evolution of the output image of the architecture according to the number of batch gradient descent iterations for the three learning strategies, using the validation example #10.

Among them, the best result is achieved using 3 pre-trained input and 1 pre-trained output, with the (2048,1024,1024,2048) geometry. It leads to a MSE of $4.42e-2$ over the test dataset (*vs.* $3.48e-02$ for the best 3-layer architecture). The behaviour with respect to the geometry and the pre-training configurations follows those of the 3 and 4 layer architecture.

In the next section, we propose the application of IODA to a real-world problem.

5. Application to a medical imaging problem

In this section, we apply the IODA architecture to a real-world problem which consists of labeling each pixel of scanner images into 2 classes. We compare the results of our approach with the state-of-the-art method for this challenging task.

5.1. Medical image segmentation task

The real-world problem addressed in this section consists in estimating the sarcopenia level, based on the scanner image labeling which is usually manually performed. Sarcopenia (loss of skeletal muscle mass) is of interest in the medical research field because this data could predict the prognosis of multiple cancers[35, 36]. Sarcopenia is assessed by manually segmenting the skeletal muscles on Computer Tomography (CT) scan slices taken at the third lumbar vertebra (L3) level.

This task is very time-consuming since the overall segmentation process take in average 4 minutes per patient for an experimented physician. Therefore, there

Table 1: Experiments on toy data set for a 3-layer architecture with different hidden sizes and pre-training setups, sorted by ascending test error. Test errors provided by IODA are in bold.

Architecture							Train error	Test error
X		$\mathbf{r}_1$		$\mathbf{r}_2$		$\hat{Y}$		
128^2	■	2048	□	2048	▨	128^2	2.64e-02	**3.48e-02**
128^2	■	1024	□	1024	▨	128^2	3.11e-02	**3.91e-02**
128^2	■	512	□	512	▨	128^2	3.26e-02	**4.10e-02**
128^2	□	2048	□	2048	▨	128^2	3.86e-02	4.59e-02
128^2	■	256	□	256	▨	128^2	4.05e-02	**4.85e-02**
128^2	□	1024	□	1024	▨	128^2	4.44e-02	5.13e-02
128^2	□	512	□	512	▨	128^2	4.81e-02	5.50e-02
128^2	■	2048	■	2048	□	128^2	5.20e-02	5.75e-02
128^2	□	256	□	256	▨	128^2	6.16e-02	6.75e-02
128^2	■	1024	■	1024	□	128^2	6.29e-02	6.77e-02
128^2	■	2048	□	2048	□	128^2	6.30e-02	6.79e-02
128^2	■	1024	□	1024	□	128^2	7.09e-02	7.55e-02
128^2	■	512	■	512	□	128^2	7.13e-02	7.60e-02
128^2	■	256	■	256	□	128^2	7.52e-02	7.98e-02
128^2	■	512	□	512	□	128^2	8.03e-02	8.48e-02
128^2	■	256	□	256	□	128^2	8.31e-02	8.75e-02
128^2	□	2048	▨	2048	▨	128^2	8.86e-02	9.37e-02
128^2	□	2048	□	2048	□	128^2	9.03e-02	9.40e-02
128^2	□	1024	▨	1024	▨	128^2	9.60e-02	1.01e-01
128^2	□	1024	□	1024	□	128^2	1.03e-01	1.06e-01
128^2	□	512	▨	512	▨	128^2	1.06e-01	1.10e-01
128^2	□	256	▨	256	▨	128^2	1.25e-01	1.28e-01
128^2	□	512	□	512	□	128^2	1.26e-01	1.28e-01
128^2	□	256	□	256	□	128^2	1.41e-01	1.41e-01

■ : input pre-training, □ : no pre-training, ▨ : output pre-training.

Table 2: Experiments on toy data set for a 4-layer architecture with different hidden sizes and pre-training setups, sorted by ascending test error. Test errors provided by IODA are in bold.

Architecture									Train criterion	Test criterion
X		$\mathbf{r}_1$		$\mathbf{r}_2$		$\mathbf{r}_3$		$\hat{Y}$		
128^2	■	2048	■	1024	□	2048	▨	128^2	2.74e-02	**3.62e-02**
128^2	■	2048	□	1024	□	2048	▨	128^2	2.95e-02	**3.75e-02**
128^2	■	1024	■	512	□	1024	▨	128^2	3.59e-02	**4.44e-02**
128^2	■	1024	□	512	□	1024	▨	128^2	3.70e-02	**4.47e-02**
128^2	■	512	□	256	▨	512	▨	128^2	3.45e-02	**4.52e-02**
128^2	■	1024	□	512	▨	1024	▨	128^2	3.52e-02	**4.53e-02**
128^2	■	2048	□	1024	▨	2048	▨	128^2	3.91e-02	**4.85e-02**
128^2	■	512	□	256	□	512	▨	128^2	4.24e-02	**5.00e-02**
128^2	■	512	■	256	□	512	▨	128^2	4.36e-02	**5.21e-02**
128^2	■	256	□	128	▨	256	▨	128^2	4.18e-02	**5.23e-02**
128^2	□	2048	□	1024	□	2048	▨	128^2	4.64e-02	5.33e-02
128^2	■	256	□	128	□	256	▨	128^2	4.59e-02	**5.36e-02**
128^2	■	256	■	128	□	256	▨	128^2	4.76e-02	**5.68e-02**
128^2	□	1024	□	512	□	1024	▨	128^2	5.07e-02	5.75e-02
128^2	□	512	□	256	□	512	▨	128^2	5.34e-02	6.02e-02
128^2	■	2048	■	1024	■	2048	□	128^2	6.01e-02	6.50e-02
128^2	■	2048	■	1024	□	2048	□	128^2	6.37e-02	6.85e-02
128^2	□	2048	□	1024	▨	2048	▨	128^2	6.23e-02	6.96e-02
128^2	■	1024	■	512	■	1024	□	128^2	6.82e-02	7.28e-02
128^2	□	512	□	256	▨	512	▨	128^2	6.60e-02	7.39e-02
128^2	□	256	□	128	□	256	▨	128^2	6.95e-02	7.50e-02
128^2	□	1024	□	512	▨	1024	▨	128^2	7.10e-02	7.74e-02
128^2	■	2048	□	1024	□	2048	□	128^2	7.72e-02	8.14e-02
128^2	■	512	■	256	■	512	□	128^2	7.84e-02	8.27e-02
128^2	■	1024	■	512	□	1024	□	128^2	7.94e-02	8.37e-02
128^2	■	256	■	128	■	256	□	128^2	8.02e-02	8.48e-02
128^2	■	512	■	256	□	512	□	128^2	8.37e-02	8.80e-02
128^2	■	256	■	128	□	256	□	128^2	8.52e-02	8.98e-02
128^2	□	2048	▨	1024	▨	2048	▨	128^2	8.70e-02	9.23e-02
128^2	■	1024	□	512	□	1024	□	128^2	9.08e-02	9.42e-02
128^2	■	512	□	256	□	512	□	128^2	9.28e-02	9.63e-02
128^2	■	256	□	128	□	256	□	128^2	1.07e-01	1.10e-01
128^2	□	1024	▨	512	▨	1024	▨	128^2	1.06e-01	1.12e-01
128^2	□	2048	□	1024	□	2048	□	128^2	1.12e-01	1.15e-01
128^2	□	256	□	128	▨	256	▨	128^2	1.11e-01	1.15e-01
128^2	□	512	▨	256	▨	512	□	128^2	1.16e-01	1.20e-01
128^2	□	256	▨	128	▨	256	▨	128^2	1.33e-01	1.37e-01
128^2	□	1024	□	512	□	1024	□	128^2	1.41e-01	1.41e-01
128^2	□	512	□	256	□	512	□	128^2	1.41e-01	1.41e-01
128^2	□	256	□	128	□	256	□	128^2	1.41e-01	1.41e-01

■ : input pre-training, □ : no pre-training, ▨ : output pre-training.

is a real need in automating the pixel labeling into two classes: skeletal muscle or not.

It is particularly challenging owing to numerous difficulties. More precisely, the method has to handle :

- The variability in the patients population:
  - the intrinsic variability in the anatomy of the patients, due to their variable genders, ages, morphologies (thin/fat) and medical states (healthy/ill) which modify significantly the shapes and the textures of the muscular, organic and fat tissues;
  - the variable organ positions : for example, kidney and liver can be present, partially or totally absent of the L3 slice;
  - the greyscale distribution overlap between muscle and internal organs.
- The variability of the images:
  - the variable quality of reconstructed CT images, due to the variable dose of radiations received by the patients during the CT acquisition (low dose / high dose);
  - the variable quantity of contrast agent that enhances the perfused tissues appearance;
  - the variable slice thickness (from submillimetric to 5mm);
  - the variable reconstruction filter used to reconstruct the images;

Figure 7 shows several images and their label. We believe that a machine learning approach could efficiently learn the intrinsic variability of this image labeling problem. For that, we dispose of a labeled dataset described thereafter.

5.2. Dataset and evaluation metrics

Our dataset consisted of 128 512×512 CT 16bit gray-level images. Each image has been manually labeled at the pixel level by a senior radiologist. Among them, 40 images come from lymphoma patients, the 88 others from breast cancer patients. As said previously, the database is composed of a wide variability of morphology, contrast and SNR between images.

We evaluated the proposed IODA automatic segmentation in comparison with the manual segmentation, and also with a referenced method[5] proposed by Chung *et al* briefly described below. In order to evaluate and compare their performance, the Jaccard index was used to measure the overlap between IODA (respectively Chung's) segmentation and the manual segmentation. We also provide the area relative difference (denoted as Diff.) metric which measures the rate of over/under-segmentation.

We now describe the Chung's dedicated method for skeletal muscle segmentation.

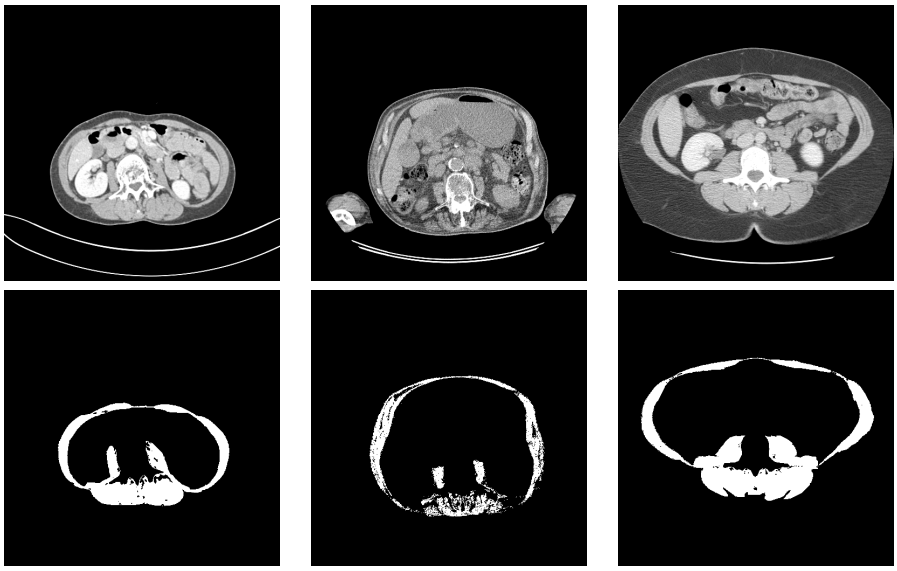


Figure 7: Examples of scanner images with their labeling below. Let us emphasize the morphology and image quality variability.

5.3. Reference automatic method

To the best of our knowledge, the only automated method for skeletal muscle segmentation at L3 was proposed by Chung *et al.* [5]. The method is based on standard shape prior coupled with an appearance model. The shape prior consists in computing the mean muscle shape on a labeled dataset, while the appearance model consists in estimating the probability distributions of both classes with a kernel density estimation method (Parzen window). In the decision process, the image is first filtered by thresholding the appearance model probability density. The final muscle segmentation is performed by an affine registration followed by a Free Form Deformation (FFD) based on a non-rigid registration. We reimplemented this method, since the original code is not available. The MATLAB toolbox MIRT[6] has been used for the non-rigid registration performed by FFD.

The next subsection describes the application of IODA to the L3 skeleton muscle image labeling.

5.4. Application and setting of IODA for CT image labeling

The IODA architecture maps the 512×512 input greyscale images into a 512×512 label matrix. We have ignored the large parts of the CT images that are non informative (black areas) for every image of the dataset. It leads to

[6]https://sites.google.com/site/myronenko/research/mirt

smaller 311×457-sized images around the patient body. This crop allows to reduce significantly the size of the architecture. Moreover, each training and test images are rigidly registered to a CT slice reference in order to reduce their size, shape and position variabilities.

For the task of learning the input and output dependencies, we have turned toward the use of 3-layer network with a MSE criterion, leading to 4 representations :

- one 311×457-sized input representation,
- two 1500-sized hidden representations,
- one 311×457-sized output representation.

The dimension of the hidden representations were empirically chosen, i.e. several geometry have been tried, and the best one have been chosen w.r.t. their performance obtained in validation. The whole resulting network is made of $145K$ hidden and output representation values, and contains $428M$ parameters. The first layer (between input and first hidden representations) and the mid-layer (between first hidden and second hidden representations) classicaly use *tanh* activation function; whereas the last layer (between second hidden and output representations) uses a *sigmoid* activation function. The first layer is pre-trained on the images, while the last layer is pre-trained on the groundtruth labels. Once pre-trained, a standard back-propagation has been performed on the whole network so as to fine tune the architecture. Since the medical imaging dataset is rather small, we have performed a gradient descent without batches, i.e. the parameters are updated at each iteration using the gradient computed on the whole training dataset. As the last layer gives a probability-like image output for the muscle tissue. This probability image must be thresholded in order to perform the final decision. This threshold has been chosen using a validation procedure in order to maximize the Jaccard index. It leads to an optimal value of 0.5. This value is the center of the output interval, but further experiments are needed to know if this value is a coincidence, or if it can be generalized.

As for the toy problem, we used our neural-network Python library, Crino, based on Theano which has not only a CPU backend, but also a GPU backend compatible with NVidia's CUDA technology. Thanks to this, we were able to run our tests on a range of different systems :

- A desktop computer featuring a NVidia Tesla C1060 GPU card with 4GB of onboard GDDR3 RAM;
- A laptop computer featuring an Intel Core i7-2760QM CPU (quad-core, 2.4GHz) and 8GB of DDR3 RAM.

Using the latter hardware setup, the overall training process of the IODA took about 35 minutes, split as follows :

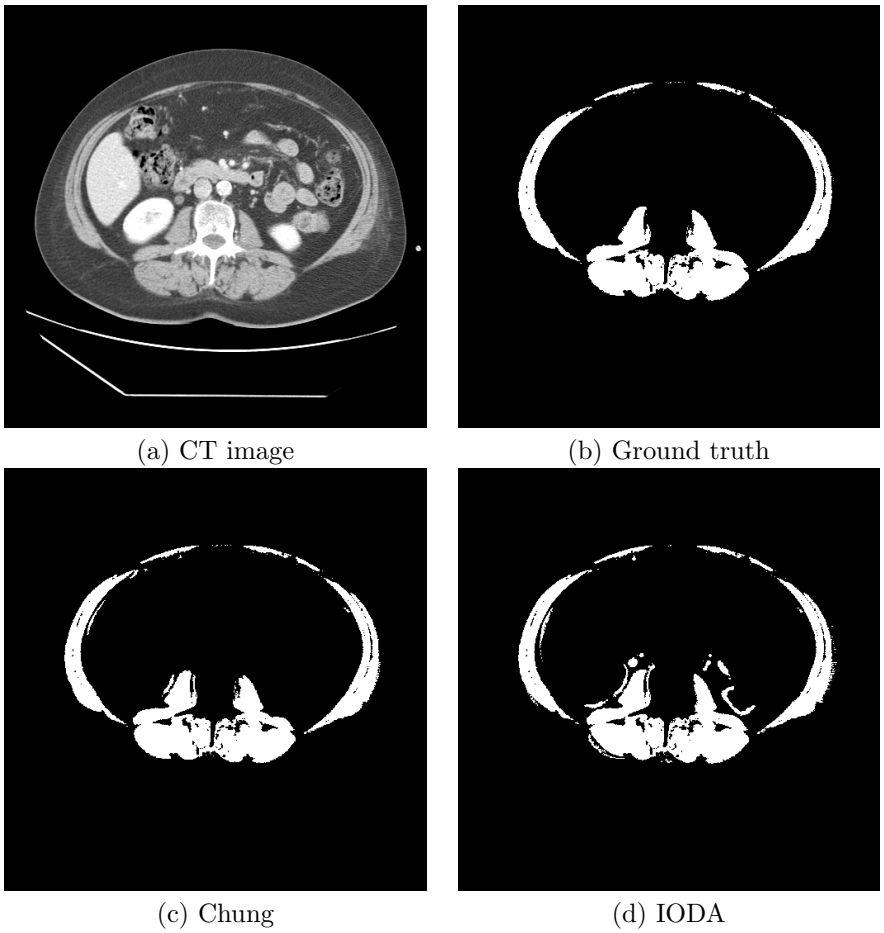
- 15 minutes for pre-training of the first layer,

- 13 minutes for pre-training of the last layer,
- 7 minutes for fine-tuning the whole network.

With the same setup, the IODA forward step of the muscle tissue segmentation process takes in average 201.2 ± 8.6 milliseconds per image, in comparison to 4 minutes (± 2 min) needed by a senior radiologist on a homemade software [36]. However, the loading in memory of the network takes approximately 10 seconds, therefore it is better to process the images in batch mode.

5.5. Results

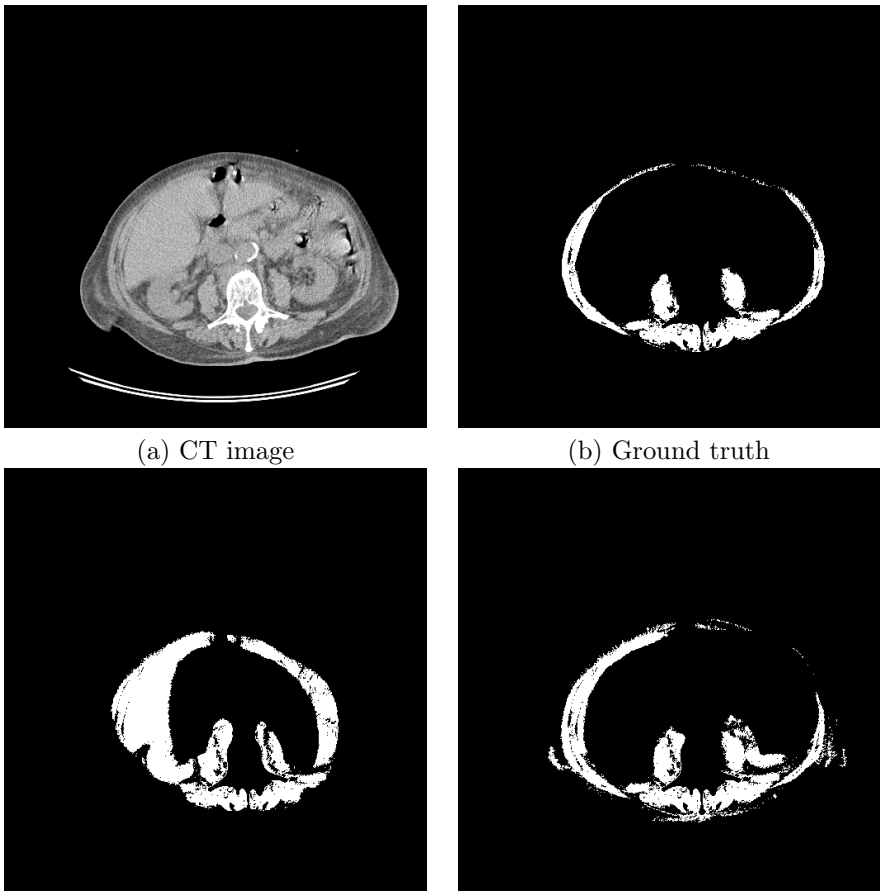
In this section we present qualitative and quantitative comparison between our neural network and state of the art approaches.



(a) CT image (b) Ground truth

(c) Chung (d) IODA

Figure 8: Non-sarcopenic patient



(a) CT image (b) Ground truth

(c) Chung (d) IODA

Figure 9: Sarcopenic patient

For a first experiment, 100 images have been used for learning the system parameters, and two images have been selected for displaying qualitative results. The first one has been considered as clean and "easy to segment" by a medical expert (See Figure 8), while the second is noisy and considered as "hard to segment" (see Figure 9). On each figure the raw image is represented in (a), ground truth labeling in (b), Chung labeling in (c) and IODA labeling in (d).

In the first case (see Figure 8), skeletal muscles, organs and fat are well delimited. Both approaches perform well : the state of the art method (Chung) achieves a Jaccard index of 97.2% whereas our proposed method (IODA) achieves 90.4%.

In the second case (see Figure 9), patient morphology is complex, and the image shows acquisition and reconstruction noise. Here, the Chung's method fails to correctly label the image with a Jaccard Index of 27.5% whereas the

IODA framework provides a much more accurate labeling leading to a Jaccard index of 64.0%.

We interpret these qualitative result by making the hypothesis that Chung method is not able to cope with strong variability of patient morphologies as it is based on a single average model, that is to say a single atlas. In opposition, our model embeds the variability of the patient morphologies through a learning process over the training set.

In order to confirm this hypothesis, we tested both methods on a large dataset, with significant variability in image contrast and in skeletal muscle shapes as suggested in Figure 7.

The regularization parameter for the FFD (λ) of Chung's method and the output muscle probability threshold of our method have been subjected to a systematic search through a 4-fold cross validation procedure, in order to maximize the Jaccard Index of these methods in validation.

We have randomly split our 128 L3 images in cross-validation and test sets as follows :

- the cross-validation set contains 100 images, itself split in 4 subsets of 25 images each,
- the test set contains the remaining 28 images.

During the cross-validation step, and for each fold, 3 subsets (= 75 images) are used for training and the remaining subset (= 25 images) is used for validation. During the test step, the entire cross-validation set is used for training and the test set is used to compute the test performance.

Method	Diff. (%)	Jaccard (%)
Chung	-10.6±40.7	60.3±32.5
NDA	0.12±9.78	85.88±5.44
IDA	0.15±9.79	85.91±5.45
IODA	3.37±9.69	88.47±4.76

Table 3: Test performance of the automatic segmentation methods. All values are reported as mean ± standard deviation.

Table 3 presents the test performance of this setup of IODA and Chung's methods. For the sake of comparison, we have also reported the results achieved by NDA and IDA strategies, as defined in 4.4, using the same setup as IODA. Chung's state of the art method gives worse results than one can expect. It confirms the hypothesis that shape and appearance prior of this method are not able to deal with too much variability. Moreover, the Diff. metric emphasizes an underestimation of the muscle tissue areas by Chung's method.

On the other hand, IODA clearly outperforms Chung's method according to both metrics. The Diff. metric suggests that IODA approach gives an average area close to the manual segmentation area. The Jaccard metric shows that IODA proposes a much better overlap of the skeletal muscles areas, and the behaviour of IODA is more stable than Chung's method since the standard

deviations are significantly lower. Let us remark that NDA and IDA approaches perform much better than Chung's method, but significantly worse than IODA. It is also of interest that NDA and IDA approaches give extremely similar results on this experiment. This is certainly due to the noisy texture of reconstructed scanner images which prevents from learning the features of the data.

6. Conclusion

In this article, we have presented a new method for image labeling that allows to learn prior knowledge on input images and output labels. The novelty lies in the automatically modelization of the output dependencies through a learning machine, whereas it usually relies on a static model like an atlas in medical applications.

As a feedforward neural network, IODA has a static architecture which implies that the input and output sizes cannot vary from an example to another. Therefore, IODA cannot be considered as a dynamic method: the processing of variable input size problems would require an image resampling preprocessing stage. Moreover, as the efficiency of IODA relies on embedding the output space, it is designed for dataset where outputs are correlated.

From a computational point of view, our approach does not require a huge amount of ressources, that makes it affordable for standard desktop computers. Nevertheless, as a lot of parameters are tuned during learning, a significant amount of memory is needed (3GB for our medical application).

Unlike other 2D-approach (CRF or HMM), our neural-based approach does not require a time consuming and suboptimal decoding process as the decision is performed using a light forward propagation through the network. Another advantage is that high order output dependencies can be modelized, while 2D approaches are generally limited to the first order dependency due to computational complexity. Indeed, IODA allows each label to depend on all other labels from the image.

From an applicative point of view, IODA could be applied on other image labeling problems. One can expect significant improvements on problems where dependencies between the output labels can be observed. This condition is often verified in medical imaging, or by instance in document image structure analysis [37, 38] or natural scene processing such as road sign detection [39, 40].

7. Acknowledgement

This work has been partly supported by the ANR-11-JS02-010 project LeMon.

References

[1] H. Wang, J. Suh, S. Das, J. Pluta, C. Craige, P. Yushkevich, Multi-atlas segmentation with joint label fusion, Pattern Analysis and Machine Intelligence, IEEE Transactions on 35 (3) (2013) 611–623.

[2] B. Zitov, J. Flusser, Image registration methods: a survey, Image and Vision Computing 21 (11) (2003) 977 – 1000.

[3] J. B. A. Maintz, M. A. Viergever, A survey of medical image registration, Medical Image Analysis 2 (1) (1998) 1–36.

[4] A. Goshtasby, Piecewise linear mapping functions for image registration, Pattern Recognition 19 (6) (1986) 459 – 466.

[5] H. Chung, D. Cobzas, L. Birdsell, J. Lieffers, V. Baracos, Automated segmentation of muscle and adipose tissue on ct images for human body composition analysis, Proc. SPIE 7261 (2009) 72610K–72610K–8.

[6] O. Chum, A. Zisserman, An Exemplar Model for Learning Object Classes, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (2007) 1–8.

[7] V. Ferrari, L. Fevrier, F. Jurie, C. Schmid, Groups of Adjacent Contour Segments for Object Detection, IEEE Transactions on Pattern Analysis and Machine Intelligence 30 (1) (2008) 36–51.

[8] G. Csurka, F. Perronnin, An Efficient Approach to Semantic Segmentation, International Journal of Computer Vision 95 (2) (2011) 198–212.

[9] P. Kohli, L. Ladický, P. H. Torr, Robust Higher Order Potentials for Enforcing Label Consistency, International Journal of Computer Vision 82 (3) (2009) 302–324.

[10] L. R. Rabiner, A Tutorial on Hidden Markov Models and Selected Aplications in Speech Recognition, in: Readings in Speech Recognition, Kaufmann, 1990, pp. 267–296.

[11] J. Lafferty, A. McCallum, F. Pereira, Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data, ICML (2001) 282–289.

[12] S. Nicolas, T. Paquet, L. Heutte, A Markovian Approach for Handwritten Document Segmentation, ICPR 3 (2006) 292–295.

[13] S. Z. Li, Markov Random Field Modeling in Image Analysis, 3rd Edition, Springer Publishing Company, Incorporated, 2009.

[14] A. Ion, J. Carreira, C. Sminchisescu, Probabilistic Joint Image Segmentation and Labeling by Figure-Ground Composition, International Journal of Computer Vision 107 (1) (2014) 40–57.

[15] P. Krähenbühl, V. Koltun, Efficient Inference in Fully Connected CRFs with Gaussian Edge Potentials, NIPS (2011) 109–117.

[16] J.Besag, On the statistical analysis of dirty pictures, Journal of the Royal Society series B, vol 48 (1986) 259–302.

[17] P. B. Chou, B. C. M., The theory and practice of Bayesian image labeling, International Journal of Computer Vision 4 (1990) 185–210.

[18] M. B. Blaschko, C. H. Lampert, Learning to Localize Objects with Structured Output Regression, ECCV (2008) 2–15.

[19] J. Weston, O. Chapelle, A. Elisseeff, B. Scholkopf, V. Vapnik, Kernel dependency estimation, NIPS (2002) 873–880.

[20] G. E. Hinton, S. Osindero, Y.-W. Teh, A Fast Learning Algorithm for Deep Belief Nets, Neural Computation 18 (7) (2006) 1527–1554.

[21] Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle, Greedy Layer-Wise Training of Deep Networks, NIPS (2007) 153–160.

[22] J. Weston, F. Ratle, R. Collobert, Deep learning via semi-supervised embedding, ICML (2008) 1168–1175.

[23] N. Srivastava, Improving Neural Networks with Dropout, Master's thesis, University of Toronto, Toronto, Canada (January 2013).

[24] R. Sarikaya, G. E. Hinton, A. Deoras, Application of deep belief networks for natural language understanding, IEEE/ACM Transactions on Audio, Speech & Language Processing 22 (4) (2014) 778–784.

[25] L. Deng, G. E. Hinton, B. Kingsbury, New types of deep neural network learning for speech recognition and related applications: an overview, ICASSP (2013) 8599–8603.

[26] A. Graves, M. Liwicki, S. Fernandez, R. Bertolami, H. Bunke, J. Schmidhuber, A novel connectionist system for unconstrained handwriting recognition, IEEE Transactions on Pattern Analysis and Machine Intelligence 31 (5) (2009) 855–868.

[27] D. E. Rumelhart, G. E. Hinton, R. J. Williams, Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1, MIT Press, Cambridge, MA, USA, 1986, Ch. Learning Internal Representations by Error Propagation, pp. 318–362.

[28] D. C. Ciresan, U. Meier, L. M. Gambardella, J. Schmidhuber, Deep big simple neural nets excel on handwritten digit recognition, Neural computation 22 (12) (2010) 3207–3220.

[29] B. Labbe, R. Herault, C. Chatelain, Learning Deep Neural Networks for High Dimensional Output Problems, in: ICMLA, Miami, USA, 2009, p. 6p.

[30] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, P.-A. Manzagol, Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion, Journal of Machine Learning Research 11 (2010) 3371–3408.

[31] Y. Bengio, A. Courville, P. Vincent, Representation Learning: A Review and New Perspectives, IEEE Transactions on Pattern Analysis and Machine Intelligence 35 (8) (2013) 1798–1828.

[32] H. Bourlard, Y. Kamp, Auto-association by multilayer perceptrons and singular value decomposition, Biological Cybernetics 59 (4-5) (1988) 291–294.

[33] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, Y. Bengio, Theano: a CPU and GPU Math Expression Compiler, Python for Scientific Computing Conference (SciPy).

[34] X. Glorot, Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, in: In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS10). Society for Artificial Intelligence and Statistics, 2010.

[35] L. Martin, L. Birdsell, N. MacDonald, T. Reiman, M. T. Clandinin, L. J. McCargar, R. Murphy, S. Ghosh, M. B. Sawyer, V. E. Baracos, Cancer Cachexia in the Age of Obesity: Skeletal Muscle Depletion Is a Powerful Prognostic Factor, Independent of Body Mass Index, Journal of Clinical Oncology 31 (12) (2013) 1539–1547.

[36] H. Lanic, J. Kraut-Tauzia, R. Modzelewski, F. Clatot, S. Mareschal, J.-M. Picquenot, A. Stamatoullas, S. Leprêtre, H. Tilly, F. Jardin, Sarcopenia is an Independent Prognostic Factor in Elderly Patients with Diffuse Large B-Cell Lymphoma Treated with Immunochemotherapy, Leukemia and Lymphoma (2013) Epub ahead of a print.

[37] V. Papavassiliou, T. Stafylakis, V. Katsouros, G. Carayannis, Handwritten document image segmentation into text lines and words, Pattern Recognition 43 (1) (2010) 369–377.

[38] P. Barlas, S. Adam, C. Chatelain, T. Paquet, A typed and handwritten text block segmentation system for heterogeneous and complex documents, Document Analysis Systems (2014) 46–50.

[39] S. M. Bascón, J. A. Rodríguez, S. L. Arroyo, A. F. Caballero, F. López-Ferreras, An optimization on pictogram identification for the road-sign recognition task using SVMs, Computer Vision and Image Understanding 114 (3) (2010) 373–383.

[40] A. Ruta, Y. Li, X. Liu, Real-time traffic sign recognition from video by class-specific discriminative features, Pattern Recognition 43 (1) (2010) 416–430.