EA RECHERCHE - *MAP 511*

Data science : son image, réseaux, apprentissage

TRAITEMENT D'IMAGES A DES FINS D'ANALYSES MEDICALES

05 Décembre 2022

Hugo Bucquet, Benjamin Lapostolle, Julien Patras
Encadrante : Stéphanie Allassonnière
Enseignant référent : Erwan Scornet

ÉCOLE POLYTECHNIQUE

IP PARIS

Remerciements

Nous tenons à remercier le Professeur Erwan Scornet, notre enseignant référent pour cet EA, pour nous avoir offert l'opportunité d'effectuer ce travail de recherche en mathématiques appliquées. Au delà des compétences techniques que nous avons pu acquérir, cette expérience nous a donné un avant goût de la recherche académique, qui nous sera utile pour notre stage de troisième année.

Nous souhaitons également exprimer notre reconnaissance à la Professeure Stéphanie Allassonnière pour nous avoir proposé ce sujet et pour nous avoir guidé tout au long de ce travail de recherche. Elle nous a présenté le monde du recalage d'images et a nous a fourni une aide précieuse sur les différentes méthodes que l'on a cherché à étudier.

Nous tenons enfin à remercier le Professeur Cristóbal Guzmán de la Pontificia Universidad Católica de Chile, avec qui nous avons pu échanger sur certains sujets de recherche actuels en Deep Learning, et qui nous a notamment introduit à la problématique de respect des données privées dans le cadre de l'entraînement de réseaux de neurones, particulièrement pertinente dans le cadre de notre étude de méthodes de recalage d'images.

TABLE DES MATIÈRES

1 INTRODUCTION

1.1 CONTEXTE

L'utilisation d'algorithmes pour traiter les **imageries médicales** est une pratique qui est apparue aux alentours des années 1980 et qui a connu un grand développement depuis. L'objectif de ces techniques est d'extraire des informations pertinentes d'imageries médicales, comme des IRM, pour faciliter le travail du médecin ou alors pour proposer des **diagnostics**. On peut citer par exemple le travail d'une équipe associée à l'université d'Oxford, *Caristo Diagnostics*, qui a créé un algorithme de machine learning permettant, à partir de CT-scan du cœur, de prédire plusieurs années à l'avance les risques de crise cardiaque.

Les méthodes de traitement d'images peuvent être regroupées en différentes catégories : segmentation d'images, recalage d'images, etc. [1] Dans notre travail, nous nous sommes concentrés sur le **recalage d'images**. Cette technique consiste à mettre en correspondance deux images afin de les comparer ou alors de combiner leurs informations respectives. Un modèle de transformation géométrique est alors choisi pour passer d'une **image source** à une **image cible**. Ces modèles peuvent être assez simples, comme une transformation rigide conservant la distance euclidienne entre les points, ou une transformation affine qui ne considère que la correspondance entre différents points de l'image. Cependant, ces transformations sont globales. Par conséquent, des modèles plus complexes utilisent des difféomorphismes afin d'autoriser des variations locales tout en préservant la **topologie de l'image** en interdisant par exemple la disparition de frontières [2]. Une fois la transformation choisie, définie par des paramètres, on définit un critère de similarité qu'on souhaite minimiser pour faire converger notre algorithme vers un jeu de paramètre donné. On peut par exemple choisir le critère *Sum of squared intensity differences* (SSD) qui consiste à calculer l'écart au carré entre les intensités des points des deux images.

Parmi les différentes méthodes de recalage d'image développées, nous nous sommes particulièrement intéressés à celle de **Large Deformation Diffeomorphic Metric Mapping** (LDDMM). Cette méthode permet, en partant d'un jeu d'images de référence, de calculer une image cible optimale ainsi qu'un ensemble de paramètres optimaux sur l'image source (nous définirons ces paramètres dans la suite de notre rapport). L'avantage de cette méthode est qu'elle est régulée par de nombreux **hyper-paramètres**, qui permettent d'adapter l'algorithme en fonction des besoins de l'opérateur. Le LDDMM peut en effet à la fois s'intéresser aux images à une échelle très **locale**, mais tout aussi bien de manière **globale**.

Tandis que des méthodes plus complexes de **Machine Learning** ont été imaginées pour améliorer le recalage d'images, les algorithmes de **réseaux de neurones** ont connu un grand développement, notamment les **CNN : Convolutional Neural Network**. Les méthodes de Deep Learning ont l'avantage de permettre d'obtenir des résultats plus rapides, et d'une qualité qui continue d'augmenter à mesure que le nombre de données augmente. En revanche, ces données doivent être assez similaires pour que le CNN apprenne à reconnaître des détails particuliers, ce qui est crucial pour des IRM de

cerveaux par exemple, mais également contraignant du fait de la variété de données et du nombre restreint de chacune. Nous expliquerons plus en détail ces aspects.

1.2 PROBLÉMATIQUE

Dans notre travail, nous avons choisi de travailler avec un algorithme de machine learning et de deep learning. L'objectif de cette étude est de mettre en exergue les **avantages** et **inconvénients** de ces techniques.

Bien que ces méthodes soient utilisées pour le recalage d'images médicales, comme des IRM de cerveau par exemple, nous avons choisi de les comparer en utilisant des jeux de données provenant du *US Postal Code Digits* dataset. Ces images sont très utiles pour faire des tests de reconnaissance d'image car elles sont peu bruitées, de basse résolution (ce qui permet de réduire le temps des calculs), et facilement classifiables.

Après avoir décrit et expliqué le fonctionnement des deux méthodes LDDMM et Deep Learning, nous discuterons des principales différences de ces algorithmes.

2
ALGORITHME DE MACHINE LEARNING

Dans notre problème, nous considérons deux images I_0 et I_1 qui correspondent à une fonction mathématique $I : \Omega \subset \mathbb{R}^2 \to [0, 1]$ renseignant l'intensité de pixel gris de l'image. Dans la suite de cette partie, on considère deux jeux de points $\mathbf{x} = (x_i)_{1 \leq i \leq n} \in (\mathbb{R}^2)^n$ et $\mathbf{y} = (y_i)_{1 \leq i \leq n} \in (\mathbb{R}^2)^n$, appelés des points de contrôle qui caractériseront nos images. Le placement de ces points jouera un rôle crucial, comme nous le verrons par la suite. Notre objectif est de trouver une **déformation** $\phi : \mathbb{R}^2 \to \mathbb{R}^2$ qui alignera ces points deux à deux, donc tel que $\phi(x_i) = y_i$. À partir de cette déformation, nous pouvons construire la nouvelle image $y \mapsto I(\phi^{-1}(y))$. Ainsi, nous recherchons des **fonctions inversibles**.

2.1 La transformation linéaire

La **transformation linéaire** est l'une des transformations les plus simples. Elle est de la forme $\phi(x) = Ax + t$ avec A une matrice et t un vecteur. Dans le cas où A est inversible, on a bien une **déformation inversible**. Pour trouver les paramètres de cette déformation — A et t — optimum, on peut choisir de minimiser le critère suivant :

$$E(\phi) = \sum_{i=1}^{n} (\|\phi(x_i) - y_i\|^2)$$

On obtient alors dans ce problème une formulation explicite des paramètres qui minimisent ce critère.

Cette transformation est donc simple à mettre en place et respecte la condition d'inversibilité. Cependant, le défaut majeur de cette transformation est qu'elle est **globale**. Par conséquent, elle ne pourra pas rendre compte des **variations locales** entre les deux images.

2.2 Cas des petites déformations

Une solution pour pallier ce problème est de créer autour des points de contrôle une zone d'influence, d'une portée plus ou moins grande, qui agira sur les points aux alentours. On peut alors proposer une déformation sous la forme $\phi(x) = x + v(x)$, avec v, le champ de déplacement, sous la forme

$$v(x) = \sum_{i=1}^{n} K(x, x_i)\alpha_i$$

avec K un noyau et les α les vecteurs différences entre les points de contrôle que l'on veut associer. Pour un **noyau gaussien** :

$$K(x, x_i) = \sum_{i=1}^{n} \exp(-\frac{\|x - x_i\|^2}{\sigma^2})$$

Suivant la valeur de σ le déplacement des points de contrôle va plus au moins influencer les points voisins. Si l'on prend un σ très faible, alors il n'y aura peu d'influence dans le voisinage des points de contrôle. En revanche, si σ est grand, alors les points voisins auront tendance à suivre **le même mouvement que celui des points de contrôle proches**.

Cette solution permet d'avoir une **déformation locale** et d'améliorer le recalage d'image. En revanche, contrairement à la solution précédente, cette déformation n'est pas inversible, en particulier lorsque les déplacements sont importants.
On a:

$$\phi(x - v(x)) = x - v(x) + v(x - v(x))) = x - \frac{\partial v}{\partial x}(x)v(x) + o(\|v\|)$$

Si la déformation ou son gradient sont grands, nous n'avons pas l'irréversibilité. Pour pallier ce problème, nous allons travailler avec des algorithmes de *Large deformation diffeomorphic metric mapping* (LDDMM) qui considère des difféomorphismes pour traduire les déformations.

2.3 Large deformation diffeomorphic metric mapping

Cette technique reprend l'idée précédente, mais au lieu de considérer un unique vecteur v qui va déplacer nos points sur potentiellement de grandes distances, on considère une **succession de champs de déplacements** qui guidera le déplacement des points à l'issue d'un grand nombre d'étapes. En utilisant des incréments infinitésimaux, on obtient l'équation différentielle suivante :

$$\begin{cases} \frac{\mathrm{d}x(t)}{\mathrm{d}t} = v_t(x(t)) \\ x(0) = x_0 \end{cases}$$

Avec une suite de difféomorphismes $(\phi_t)_{t\in[0,1]}$ qui renseigne pour un $t \in [0, 1]$ la position d'un point x à la date t, l'équation se réécrit :

$$\begin{cases} \frac{\mathrm{d}\phi_t}{\mathrm{d}t} = v_t(\phi_t) \\ \phi_0 = Id \end{cases}$$

2.3.1 • Construction de la fonction à optimiser

Dans notre problème, étant donné deux images, I_0 et I_1, notre but est de trouver un **difféomorphisme** ψ reliant les deux images tel que $I_0 \circ \psi^{-1} = I_1$. Ici, I_0 est appelé l'image source, c'est à dire l'image que l'on connaît, et I_1 est l'image cible, celle que l'on cherche à classifier.
Pour trouver ϕ, nos allons construire une famille de difféomorphismes $(\phi_{t\in[0,1]})$ avec $\phi_0 = Id$ et $\phi_1 = \psi$, qui obéissent aux équations ci-dessus. Dans notre problème d'optimisation, on cherche le champ de déplacement $(v_{t\in[0,1]}$ qui minimise le critère suivant :

$$argmin_{v_t} \int_0^1 ||v_t||_V + ||I_0 \circ \phi_1^{-1} - I_1||_{L^2}$$

$||*||_V$ correspond à une norme sur le champ de vecteurs v_t, qui permet de contrôler sa régularité. En pratique, on définit cette norme à partir d'un opérateur $L = (-\alpha\delta + \gamma)^\beta I_{n*n}$ tel que $||f||_V = ||Lf||_{L^2}$. [3]

2.3.2 • Discrétisation du problème

Pour mettre en place notre algorithme,nous procédons à une **discrétisation** du problème en paramétrant l'image I_0 par des **points de contrôle** choisis judicieusement. Pour cela, il faut choisir un noyau K (que l'on prendra Gaussien par la suite), et une famille c_k^0 de N points de contrôle et w_k^0 leur poids associé, de manière à écrire I_0 de la forme:

$$I_0(x) = \sum_{k=1}^{N} K(x, c_k^0) w_k^0$$

Dans notre approche, on construit ensuite une discrétisation de l'espace du champ vecteurs v_t:

$$v_t(x) = \sum_{k=1}^{N} K(x, c_k^1(t)) \alpha_k(t)$$

où $c_k^1(t)$ sont des points de contrôle, et $\alpha_k(t)$ les momentums associés.
Alors, tout point $x(t) = \phi_t(x_0)$ dans l'espace suit un chemin satisfaisant l'équation:

$$\dot{x}(t) = v_t(x(t)) = \sum_{k=1}^{N} K(x(t), c_k^1(t)) \alpha_k(t)$$

En 2006, **Miller** à montré que ces paramètres satisfont les équations différentielles: [4]

$$\begin{cases} \dot{c}_k^1(t) = K(c_k^1(t), c_k^1(t))\alpha_k(t) = v_t(c_k^1(t)) \\ \dot{\alpha}_{k(t)} = -\frac{1}{2} \left(d_x K(c_k^1(t), c_k^1(t))\alpha_k(t)\right)^T \alpha_k(t) \end{cases}$$

Ces équations sont importantes car elles montrent que les points de contrôle et leur momenta sont liés au cours du temps, et que leurs trajectoires ne dépendent que des **conditions initiales**. Notons $C_0 = c_0^0, c_1^0, .., c_k^0$ les points de contrôle de l'image I_0, et $C_1 = c_0^1, c_1^1, .., c_k^1$ ceux de I_1. Notre but est maintenant de travailler sur ces points, en minimisant l'erreur quadratique entre $\phi(C_0)$ et C_1, en imposant toujours une condition de difféomorphime sur la transformation.

Ainsi, il faut trouver un équilibre entre la **fidélité** de la transformation et sa **régularité/élasticité**. On introduit pour cela le paramètre γ et l'**énergie** que l'on peut ré-écrire en fonction des c_k et α_k.

$$E(v_t) = \left\| \psi^{-1}(C_0) - C_1 \right\|^2 + \gamma \int_0^1 \|v_t\|_V^2 \, dt$$

avec $\|v_t\|_V^2 = \alpha(t)^T K(C_0(t), C_0(t))\alpha(t)$

On est ainsi passé d'un problème de dimension infinie dans l'espace des difféomorphismes de I_0 à I_1 à un problème de **dimension finie**, qui consiste à optimiser les $\alpha_k(t)$ et les points $c_k^1(t)$.
En prenant $t = 0$, on trouve:

$$E(\{\alpha_i(0)_{1 \le i \le N}\}) = \|\psi^{-1}(C_0) - C_1\|^2 + \gamma \|v_0\|_V^2$$

Avec un algorithme de **descente de gradient** (notebook jupyter que l'on a utilisé pour nos résultats), on peut ainsi arriver à minimiser cette énergie.

2.4 Les données de travail

Nous avons testé ces différentes méthodes algorithmiques sur des images afin d'observer leurs performances et de comprendre comment régler l'influence des différents paramètres sur les résultats. Le fonctionnement de notre algorithme est le suivant : après avoir déterminé les points de contrôle des images, nous faisons au préalable une **transformation affine** entre les deux images afin de réduire la taille des déformations dans l'algorithme LDDMM qu'on applique ensuite.

Dans le cadre de notre étude, nous avons choisi de travailler avec une base de donnée d'images représentant des 2, issues de celle de chiffres manuscrits de taille 16×16 de la base de données de l'*U.S. Postal Service*[6]. Parmi d'autres solutions, nous avons choisi cette base, car elle présente plusieurs avantages :

- Elles sont toutes de taille 16*16. Par conséquent, nous n'avons pas à redimensionner les images et le nombre réduit de pixel permet d'accélérer la vitesse des calculs
- Le fond des images se distingue bien du nombre
- Les 2 sont assez différents et permettent de tester la robustesse des algorithmes

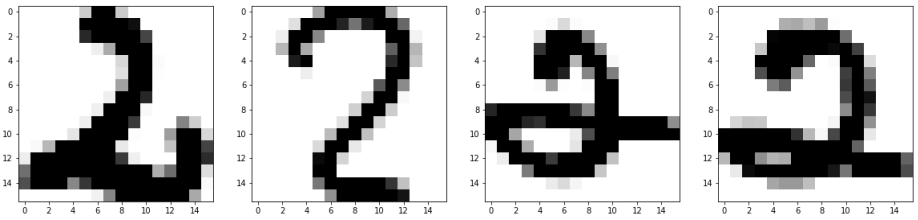

Figure 1: *Base de données*

2.5 Définir les points de contrôle

Pour faire fonctionner nos algorithmes, il faut en préalable fournir en entrée les points de contrôle de l'image source et cible. Plusieurs solutions sont alors possibles. Pour comprendre l'importance de ces paramètres, nous avons décidé de tester de placer ces points de manière **aléatoire** sur l'image, avec comme contrainte de ne pas être trop proche des bords. Ainsi, pour notre base de donnée, on obtient des points de contrôle pour l'image cible et pour toutes les autres images, comme on l'observe sur la figure 2.

En appliquant l'algorithme sur les images *target image* et *source image N°1* de la figure 2, on obtient les résultats présentés sur la figure 3. Comme on pourrait naturellement s'y attendre, les résultats que l'on trouve ne sont pas concluants. En plaçant aléatoirement les points de contrôle, nous avons créé une déformation de l'espace aléatoire.

Une autre solution est de placer **manuellement** les points de contrôle sur les images. Pour savoir où placer les points, il est important de se demander quels sont les points de nos données qui les

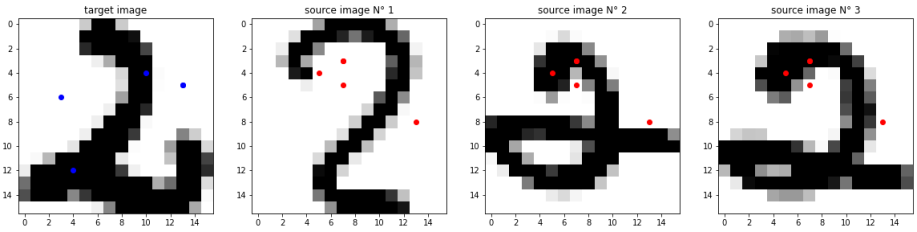

Figure 2: Placement aléatoire des points de contrôle

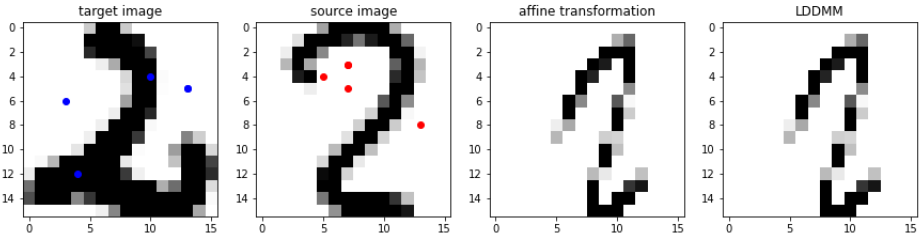

Figure 3: Résultat du placement aléatoire des points de contrôle

caractérisent le plus. Dans notre cas, comme nous travaillons avec des deux, nous avons déterminé les points caractéristiques de nos images[1], puis nous les avons placés sur deux données. On obtient alors le résultat visible sur la figure 3 ci-dessous.

Cette fois-ci, l'algorithme semble mieux fonctionner. Malgré deux deux qui ont des structures relativement différentes, on observe que l'image obtenue en résultat présente des caractéristiques de l'image cible : le bas du deux est mieux représenté avec la queue du deux qui remonte, et la boucle se réoriente. On observe donc que les points de contrôle jouent un rôle crucial, leur placement impactant grandement la qualité des résultats. Par conséquent, il est intéressant d'essayer de trouver une solution permettant de placer intelligemment ces points de manière **automatique**. Le placement manuel est une opération longue et qui requiert l'intervention d'un humain. Dans le cas de deux, nous avons pu choisir nous même nos points, mais dans le cas d'image médicale, alors c'est potentiellement un médecin qui devrait choisir où sont les zones caractéristiques de l'image.

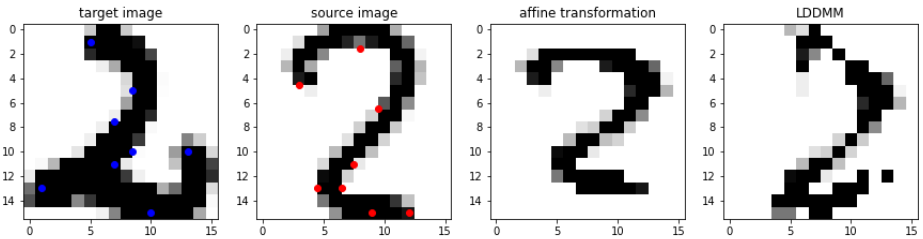

Figure 4: Résultat de l'algorithme en plaçant les points manuellement

[1] la queue du 2, milieu de la boucle, diagonale haute gauche, bout du bas, bas du 2, bout de la queue du bas, creux du 2, diagonale basse, début de la boucle

2.6 Rôle des hyper-paramètres

Avant de décrire l'automatisation du placement des points de contrôle, nous allons discuter de l'influence de σ et de γ qui correspondent respectivement à la **largeur de notre noyau**, ici gaussien, et au poids associé à la **régularisation** de notre solution dans l'énergie.

2.6.1 • Largeur du noyau

Pour rappel, notre noyau est un noyau gaussien qui s'écrit donc : $K(x, x_i) = \sum_{i=1}^{n} \exp(-\frac{\|x-x_i\|^2}{\sigma^2})$, et notre vecteur de déplacement $v_t(x(t)) = \sum_{k=1}^{N} K(x(t), c_k^1(t))\alpha_k(t)$.
La largeur du noyau, σ, détermine à quel point le voisinage des points de contrôle est influencé par leurs mouvements. Les cas extrêmes $\sigma = 0$ et $\sigma = \infty$ correspondent respectivement à un cas où le vecteur v est nul et à un cas où la déformation est en réalité globale. Plus σ augmente, plus la transformation est globale.

Par conséquent, on doit choisir la bonne valeur de σ pour que la transformation soit **locale** et qu'elle ne soit pas **minime**. On peut par exemple observer le résultat de l'algorithme avec différentes valeurs de σ sur la figure ci-dessous5. Pour ce deux précis, on observe qu'il ressemble au mieux à notre image cible pour une valeur de $\sigma = 10$. Pour des plus petites valeurs, la transformation est peu importante, et pour de grandes valeurs, on observe alors que la transformation est globale et qu'ici on a uniquement une translation du deux.

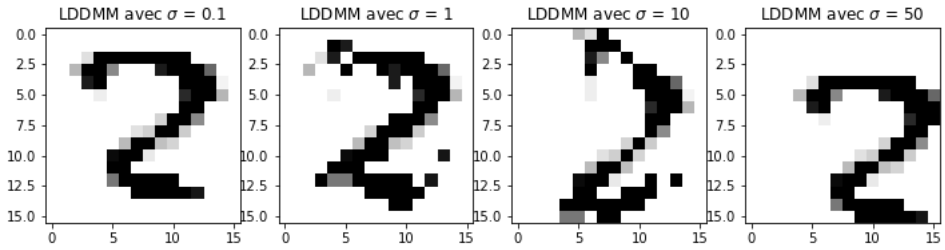

Figure 5: Influence de la largeur du noyau sur les résultats

2.6.2 • Paramètre de régularité

Le paramètre de régularité γ intervient dans l'énergie que l'on minimise. Plus il est important, plus notre déformation sera régulière et d'amplitude moindre. On étudie son influence sur notre résultat pour $\sigma = 10$. On observe sur la figure 6 que plus gamma est important, plus les déformations sont petites. On choisit donc de fixer ce paramètre à 0.1 pour s'autoriser des déformations importantes, les deux étant très différents, tout en conservant une déformation régularisée.

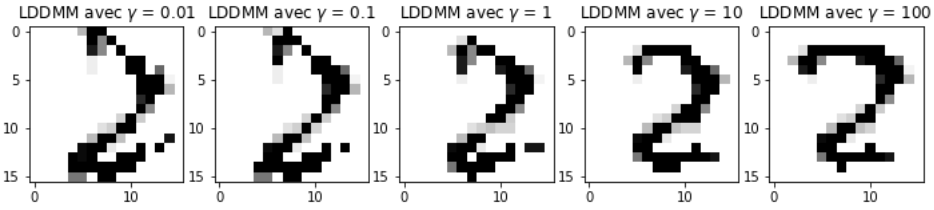

Figure 6: Influence du paramètre de la régularité sur les résultats

2.7 Automatisation du placement des points de contrôle

Nous avons réfléchi à un algorithme permettant d'optimiser le placement des points de contrôle. Il fonctionne de la manière suivante : on sélectionne les points ayant le gradient le plus important sur l'image, et on les repartit de façon à ce qu'ils couvrent une majorité de l'image.

• Description de l'algorithme

Entrée : Image I, i.e. une matrice de taille $n * n$, et N le nombre de points de contrôle que l'on souhaite mettre sur l'image.

Sortie : Liste L contenant les coordonnées des points de contrôle.

Algorithme : Pour chaque point $(i, j) \in [2, n-1]^2$, calculer un gradient discret :

$$G(i,j) = |I(i,j) - I(i,j+1)| + |I(i,j) - I(i-1,j)| + |I(i,j) - I(i,j+1)| + |I(i,j) - I(i,j-1)|$$

Ensuite, on va chercher à sélectionner les N points ayant le plus grand gradient, en s'assurant qu'ils soient assez loins les uns des autres pour être assuré de capturer la totalité de la topologie de l'image.

Pseudo-code :

Pour i allant de 1 à N:
  $a, b = argmax(G)$
  $G(a, b) = 0$
while$(dist((a, b), L) < n/N)$
$L \leftarrow L + (a.b)$
Return L

Une fois avoir obtenu les listes de points $L1$ et $L2$ pour les deux images, il faut arriver à trouver un matching des points pour permettre à l'algorithme de LDDMM de trouver le bon difféomorphisme. Pour trouver un matching satisfaisant, nous avons choisi de minimiser l'erreur quadratique :

$$\sum (L1(i) - L2(i))^2$$

en réordonnant les points de $L2$.

Il s'agit alors trouver une permutation $\pi^* \in S_N$ satisfaisante :

$$\pi^* = \underset{\pi \in S_N}{\operatorname{argmin}} \sum (L1(i) - L2(\pi(i))^2$$

Pour cela, on peut reformuler le problème sous la forme d'un problème de programmation linéaire :

Paramètres : $c_{i,j} \in [0,1] \cap \mathbb{N}$ $(c_{i,j} = 1_{\pi^*(i)=j})$

Fonction objectif : $\sum_{i,j} c_{i,j}(L1(i) - L2(j))^2$

Contraintes : $\sum_i c_{i,j} = 1$ and $\sum_j c_{i,j} = 1$

On obtient ainsi 2 listes de points de contrôle que l'on peut fournir au LDDMM.

Notre algorithme possède cependant plusieurs limitations :

- Il place les points sur le bord de l'image, mais on ne peut pas déterminer si ces points sont à l'extérieur ou à l'intérieur de celle-ci et si ils caractérisent des points caractéristiques de notre image.
- La borne inférieure de la distance entre les points de contrôle devrait être une fonction de N, n, mais aussi du nombre de points de contrôle déjà enregistrés pour garantir la **terminaison** de l'algorithme.

Cependant, pour notre cas, cet algorithme donnait des résultats satisfaisants que l'on pouvait exploiter.

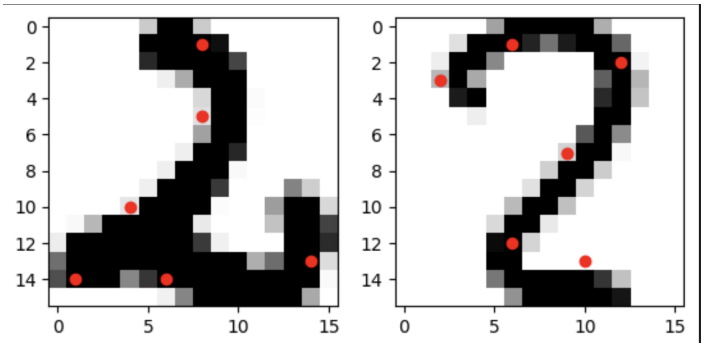

Figure 7: *Points de contrôle en sortie pour deux images de 2, avec* $N = 6$

- RÉSULTATS

Nous avons utilisé ces points de contrôles pour faire fonctionner le LDDMM. Même si à première vue, on peut estimer que ces points sont relativement bien placés, on ne parvient pas encore à obtenir des résultats concluants. En effet, on remarque par exemple sur la figure 8 que les points situés en bas du deux sont assez mal placés. Ces petits écarts par rapport à un placement idéal suffisent à induire une déformation incorrecte et à expliquer nos résultats. Ces remarques achèvent de confirmer l'importance du placement des points de contrôles.

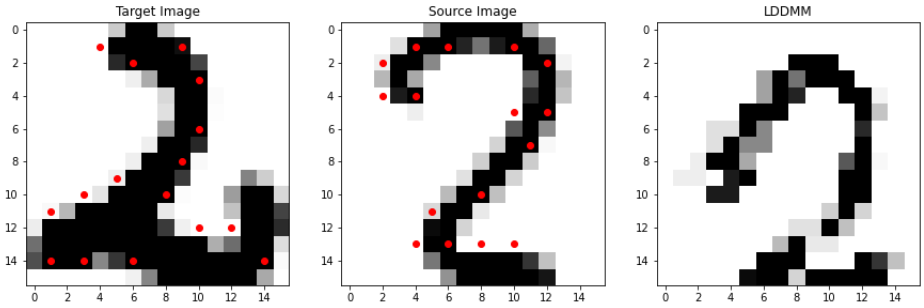

Figure 8: *LDDMM en utilisant les points de contrôles placés de manière automatique*

3
APPROCHE PAR LE DEEP LEARNING

Les méthodes de Machine Learning donnent des résultats concluants, mais demandent des **grandes capacités de calcul**. Dans le cadre de la médecine, il ne s'agit pas de comparer 2 images, qui seraient 2 coupes d'IRM de cerveaux par exemple, mais plutôt de comparer les 2 cerveaux en 3D. Il y a donc beaucoup plus de données à traiter, et les temps de calcul de méthodes de ML deviennent possiblement trop longs pour de l'exploitation. Le défi est de trouver d'autres méthodes plus efficaces, avec l'espoir de pouvoir s'en servir.

Le Deep Learning permet d'obtenir des méthodes plus élaborées se basant sur des réseaux de neurones (NN, *Neural Networks*). Dans le cas du traitement d'images, un type de NN qui s'est montré très efficace est le réseau de neurones convolutif (**CNN**, *Convolutional Neural Network*). Précisons ces termes, les méthodes et les idées qu'ils encapsulent.

3.1 Intérêt des CNN

Les CNN sont une sous-catégorie de réseaux de neurones. En l'état actuel, ils ont donné les meilleurs résultats en terme de traitement et classification d'images.

Ce sont des NN qui ont une certaine sorte de **spécialisation** leur permettant de repérer ou d'extraire des caractéristiques propres à chaque donnée, ce qui se révèle très utile dans le cas du traitement d'images. Cela se fait dans les couches dites de convolution,. Contrairement à des modèles de NN classiques MLP (*Multi Layers Perceptron*) dont toutes les couches sont dédiées à la classification, un CNN possède également ces couches convolutives en amont.

Les **couches de convolution** analysent entièrement l'image, mais en procédant par zones. Elles ont une **approche locale**, ce qui permet d'extraire les **motifs**, les **tendances d'une donnée**, comme des changements de contrastes brusques qui peuvent correspondre à une frontière. Dans le cas des chiffres ces contrastes peuvent permettre de repérer les bords du chiffre. Dans le cas d'un IRM de cerveau, ils peuvent correspondre aux bords d'une zone cérébrale.

De cette manière, la partie convolutive prend en entrée l'image de base, elle la décompose en de nouvelles images appelées **cartes de convolution**, et qui stockent des caractéristiques de l'image que les couches convolutives auront appris à mettre en évidence.

De plus, lors de l'apprentissage, le modèle repère au fur et à mesure des nouvelles caractéristiques avec des nouvelles images, et apprend cette **capacité à repérer ces caractéristiques** qui lui permettront de décrire plus précisément les images suivantes. Cette approche locale permet d'être beaucoup plus précis dans l'analyse de l'image, et permet d'effectuer une classification plus pertinente en sortie des couches de convolution.

Elle a donc 2 avantages majeurs par rapport à une approche classique, plus globale de l'image :

- elle **réduit les erreurs dans l'apprentissage**, puisque le CNN n'apprend pas à reconnaître une image mais des motifs et des caractéristiques
- **le traitement et la classification sont donc plus précis**, du fait d'une description plus fine et caractérisée des images

- Exemple

Prenons un exemple simple et pertinent pour notre sujet d'étude, dont les images sont tirées du tutoriel *Convolutional Neural Networks (CNNs) explained* de DeepLizard[7].

Plaçons-nous dans le cadre du traitement de chiffres écrits à la main en noir et blanc, et stockés comme des matrices de pixels. Le but de notre CNN est de repérer quel est le chiffre écrit qui est passé en entrée. Prenons ce 7 en entrée :

Figure 9: *Exemple d'entrée pour le CNN*

Considérons des filtres définis, qui ne seraient pas nécessairement ceux du CNN mais qui vont nous permettre de visuellement comprendre l'idée de la convolution.
La première couche convolutive pourrait avoir ces filtres, qui permettent de repérer les bords dans les 4 directions bas, droite, haut, gauche :

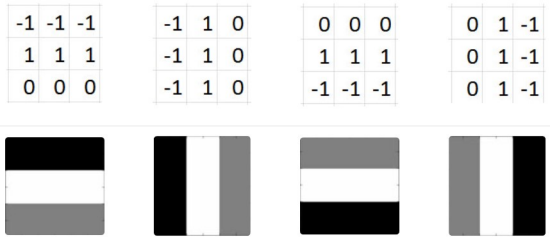

Figure 10: *Exemples de filtres CNN pour une première couche convolutive*

Figure 11: *Output de ces filtres avec le 7 ci-dessus en entrée*

Ces filtres permettant de repérer des bords, ils sont appelés *edge detectors*.
Ensuite, une fois ces informations récupérées, la deuxième couche pourrait chercher des formes plus

complexes comme des coins ou des formes rondes par exemple.

Avec la somme de toutes ces informations que le CNN aurait recueillies, il saurait ensuite beaucoup plus facilement reconnaître le chiffre en effectuant une classification en fonction des parties qu'il a repérées.

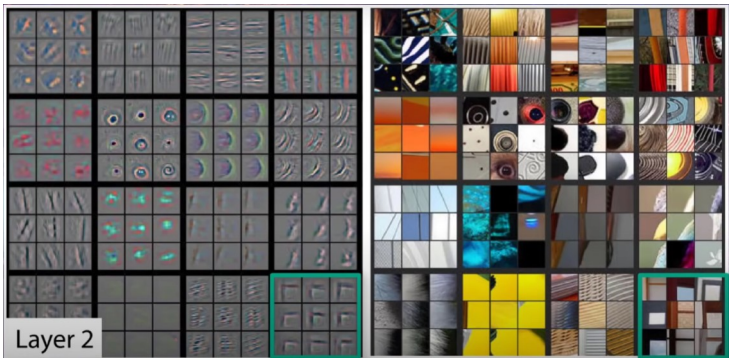

Figure 12: *Exemples de filtres CNN pour une deuxième couche convolutive*

Ces filtres sont des exemples, et lors de l'apprentissage, le CNN construit ses propres filtres, moins évidents visuellement, mais qui lui permettent d'avoir des **critères de classification pertinents**, par exemple pour décider quel était le chiffre donné en entrée, ou comment recoller deux images de chiffres de la manière la plus efficace possible. Cette notion d'efficacité est naturellement liée à la fonction de perte, déjà évoquée sous la forme d'une énergie dans le cadre du ML et qui va être précisée dans la partie à suivre.

3.2 Mise en oeuvre d'un CNN

Maintenant que ces notions sont éclaircies, mettons les à profit pour étudier des méthodes innovantes et présentées comme révolutionnaires dans le traitement d'images.

La méthode principale que nous avons étudiée et utilisée est celle proposée dans l'article *An Unsupervised Learning Model for Deformable Medical Image Registration*[5].

3.2.1 • Supervised Learning

Contrairement aux méthodes de Machine Learning, celle-ci ne nécessite pas d'avoir une valeur de référence, comme un cerveau sain de référence, d'où son nom d'*Unsupervised*. Le CNN va apprendre tout seul à partir d'un jeu de données d'entraînement à effectuer un recalage efficace (cette notion d'efficacité est expliquée en 3.2.2). Cet aspect a l'avantage de ne pas faire reposer une partie de la réussite de la méthode sur une référence ou une valeur à viser, notion qui est d'ailleurs largement discutable dans un contexte comme le traitement d'IRM de cerveaux.

Il existe également des algorithmes de recalage de Deep Learning dit *supervisés*, c'est-à-dire que le CNN doit apprendre à générer un champ de déformation de référence (ou fonction de déformation

ϕ_{gt}, *gt* pour *ground truth*). Il doit adapter ses paramètres en fonction d'un résultat contraint, comme précisé sur la figure 13

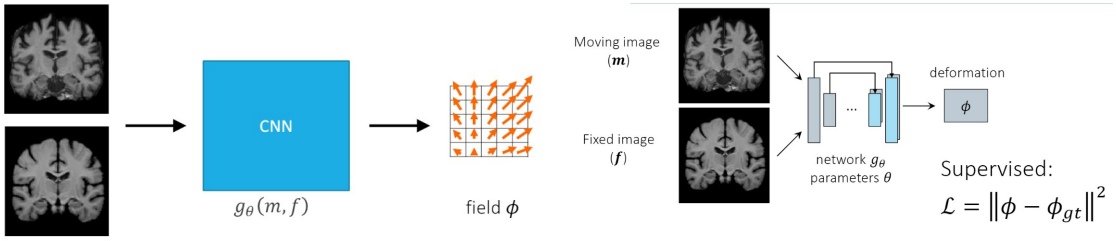

Figure 13: *Deux représentations schématiques d'un CNN supervisé*

Comme nous l'avons vu dans une approche visant à directement optimiser le champ de déformation, la fonction de perte à minimiser est

$$\mathcal{L}(F, M, \phi) = \mathcal{L}_{sim}(F, M(\phi)) + \lambda \mathcal{L}_{smooth}(\phi)$$

où l'on note $M(\phi)$ le résultat de l'application du champ ϕ à l'image à recaler M (*Moving image*) (F étant l'image de référence, i.e. l'image *fixed*), et l'on souhaite se rapprocher pour chaque paire de ce champ :

$$\hat{\phi}(F, M) = \underset{\phi}{\operatorname{argmin}}\ \mathcal{L}(F, M, \phi)$$

qui est donc le champ optimal qu'un CNN supervisé cherche à directement approcher.
Cette fonction de perte fait naturellement echo à l'énergie étudiée en partie 2.3.2.

3.2.2 • Unsupervised Learning : optimisation des paramètres

L'approche que nous avons retenue est différente, puisque les **valeurs des paramètres θ du CNN** (donc les valeurs/fonctions des neurones qui le composent) **sont apprises via des des paires de cerveau**, l'un qui est l'image fixe à atteindre et qui correspond à la moyenne des images des IRM de cerveaux, et l'autre est l'image à déformer. Plus précisément, **on entraîne un CNN à recaler chaque image du jeu de données d'entraînement sur une image fixe**, que l'on choisit comme étant la donnée moyenne de toutes ces données. Avec des images comme données, cela revient à faire la moyenne de la valeur de chaque pixel sur l'ensemble du jeu de données. On note cette image fixe F, comme dans les exemples de CNN supervisé ci-dessus.

Puisque nous ne disposons pas d'un champ de déformation optimal pour chaque paire d'image (F, M) à approcher, nous allons plutôt chercher à **minimiser une fonction de perte en optimisant les paramètres θ de notre CNN**. En termes mathématiques, voici les formulations :

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}}\ \left[\mathbb{E}_{(F,M)\sim D}[\mathcal{L}(F, M, g_\theta(F, M))]\right] \tag{1}$$

est le paramétrage que l'on souhaite approcher, où D est notre jeu de données d'entraînement.

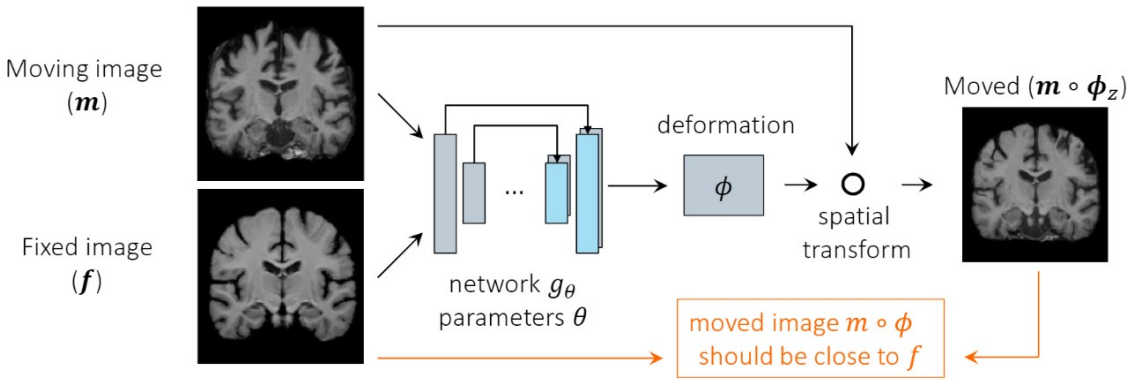

Figure 14: *Méthode d'entraînement du CNN unsupervised*

La fonction de perte est

$$\mathcal{L}(F, M, \phi) = -CC(F, M(\phi)) + \lambda \sum_{p \in \Omega} ||\nabla \phi(p)||^2$$

dont

$$\mathcal{L}_{smooth}(\phi) = \sum_{p \in \Omega} ||\nabla \phi(p)||^2$$

où Ω est l'espace dans lequel chaque donnée est, $\Omega \subset \mathbb{R}^n$ où n est la dimension de l'espace dans lequel vivent les données. Par exemple, pour des images de chiffres, Ω est l'ensemble des positions des pixels de l'image et $\Omega \subset \mathbb{R}^2$.

Il faut donc comprendre que $\mathcal{L}_{smooth}$ décrit l'ampleur des variations qu'effectue ϕ sur les pixels (ou voxels) de Ω, d'où le fait que **cette partie de la fonction de perte corresponde à la contrainte sur la régularité de ϕ, i.e. de g_θ dans** (1).
Sur ce point, précisons que cette contrainte de régularité n'assure pas que notre fonction de déformation g_θ soit un difféomorphisme. Cet aspect peut sembler être un désavantage par rapport aux méthodes de Machine Learning de type LDDMM (cf 2.3). Nous discuterons ce point plus en détail lors de l'analyse des résultats en 3.3 et 4.

D'autre part,

$$\mathcal{L}_{sim}(F, M(\phi)) = -CC(F, M(\phi)) = -\sum_{p \in \Omega} \frac{\left(\sum_{p_i}(F(p_i) - \hat{F}(p))(M(\phi(p_i)) - \hat{M}(\phi(p))\right)^2}{\left(\sum_{p_i}(F(p_i) - \hat{F}(p))\right)\left(\sum_{p_i}(M(\phi(p_i)) - \hat{M}(\phi(p_i))\right)}$$

Il s'agit de la ***cross-correlation*** **normalisée** entre les images F et $M(\phi)$, et elle permet de se rendre compte de la **proximité entre l'image fixe F à atteindre et la déformation** $M(\phi)$ de l'image mobile M.

Dans le cadre de la recherche liée à cet article, les données cibles sont des matrices de $M_n(\mathbb{R}^3)$ représentant des IRM de cerveaux en 3D, et les zones des cerveaux étudiées sont découpées en petits voxels. Donc comparer la proximité entre 2 données en les comparant voxel par voxel n'est pas pertinent, car il s'agit plus de comparer des zones, certes petites, mais décrites par plusieurs voxels.

Ainsi, $\hat{F}(p)$ et $\hat{M}(\phi(p))$ sont les intensités moyennes autour du voxel p. Pour des IRM en 3D, cette moyenne est réalisée sur un volume de n^3 avec $n = 9$, composé des pixels p_i présents dans l'expression ci-dessus. Dans notre implémentation qui vise à recaler des images 2D constituées de peu de pixels (16×16 pour les chiffres écrits présentés en 2.4, aussi utilisés avec cette méthode pour comparer ses résultats à ceux de Machine Learning), cette moyennisation est un peu moins nécessaire, puisque cette modélisation est plutôt adaptée à des données de tailles bien plus grandes.

En résumé, à chaque paire d'images, on optimise les paramètres θ de notre réseau de neurones par rapport à la fonction de perte $\mathcal{L}$. A ce stade, il s'agit d'une descente de gradient classique dans l'entraînement d'un réseau de neurones.

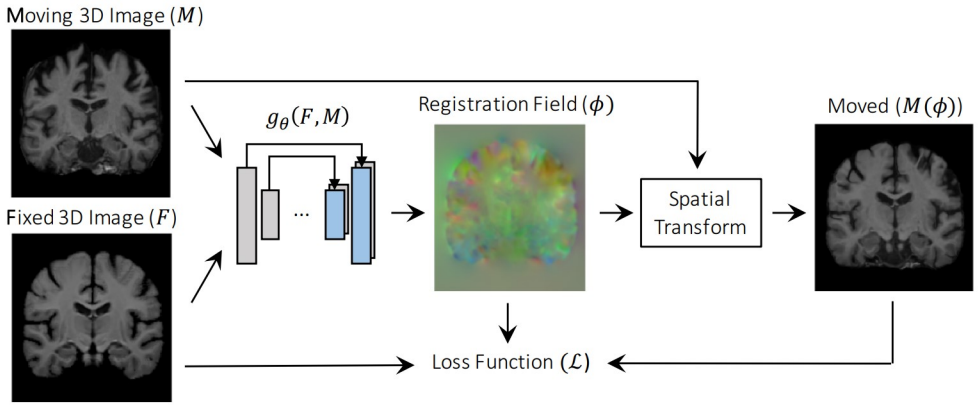

Figure 15: *Autre schématisation de la méthode d'entraînement du CNN unsupervised*

3.3 Résultats du CNN

Nous avons testé cette méthode sur la même base de données que celle utilisée pour le Machine Learning, composée de chiffres manuscrits de taille 16×16 issus de la base de données de l'*U.S. Postal Service*[6].
Le jeu de données d'entraînement étant composé de 7291 images dont 731 identifées comme étant des 2, et disposions de 2007 tests d'observation dont 198 labelisées comme des 2, nous avons entraîné notre CNN sur les 731 images pour qu'il sache effectuer des recalages entre images de 2 manuscrits.

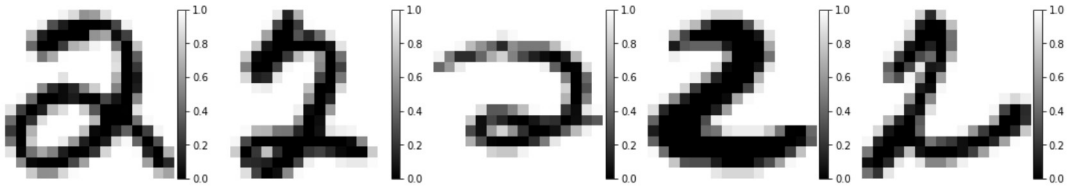

Figure 16: *Exemples de 2 issus de la base de données*

Pour entraîner un CNN comme nous l'avons décrit précedemment, nous avons utilisé les outils de la librairie *VoxelMorph* développée par l'équipe de chercheurs ayant publié l'article. Les liens vers la documentation et le repo *Git* sont dans notre bibliographie ([8]), et notre code est en annexe 6.2.

3.3.1 • Caractéristiques de notre CNN

Les paramètres de notre CNN sur lesquels nous avions une influence sont :

- λ, qui correspond au poids le $\mathcal{L}_{smooth}$ dans la fonction de perte,
- nombre d'*epochs* pour l'utilisation des données d'entraînement de notre CNN,
- la taille des batches d'entraînement

La taille de notre CNN était gérée par le librairie VoxelMorph. Nous n'avons malheureusement pas réussi à pénétrer assez profondément dans le re-paramétrage de la librairie pour personnaliser à souhait ce paramètre.

Pour les paramètres ci-dessus, nous avons testé des valeurs différentes, pour optimiser nos résultats. Pour éviter l'overfitting, et comme les optimisations à chaque epoch commençaient à converger comme on le voit sur la figure 17, nous avons choisi d'effectuer 20 *epochs*. Nous avons choisi une taille des *batch* de 58, soit le $100^{\text{ème}}$ d'un epoch.
Pour le paramètre λ, nous l'avons fait varier. Nous présenterons les résultats pour $\lambda = 0.05$ et $\lambda = 1$. Il est intéressant de regarder l'influence de λ à la fois sur la qualité du recalage mais aussi sur les valeurs du gradient.

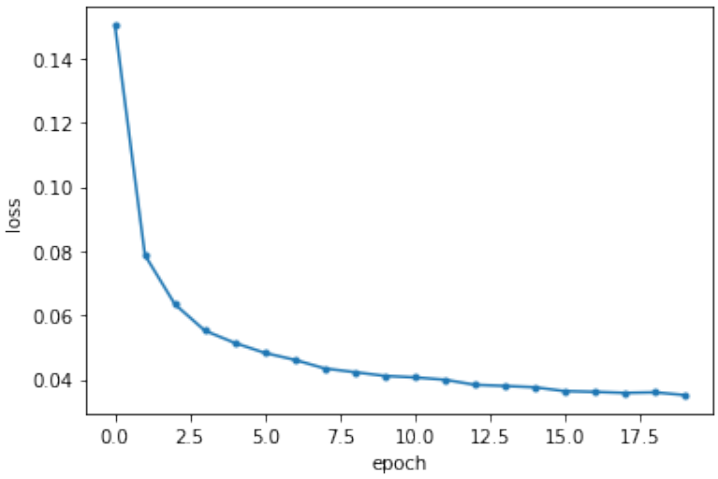

Figure 17: *Valeurs moyennes de la fonction de perte au cours de l'entraînement*

3.3.2 • Résultats

- Pour $\lambda = 0.05$:

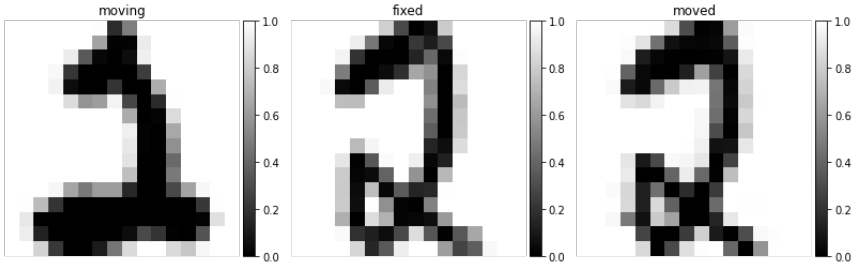

Figure 18: *Recalage pour un 2*

Pour cet exemple, le recalage a très bien marché. Regardons où la fonction de déformation a eu le plus à faire de modifications pour effectuer le recalage.

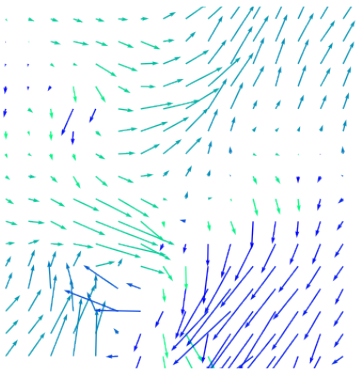

Figure 19: *Gradient pour le recalage de la figure 18*

Cette représentation du gradient est précieuse car elle permet de mettre en évidence les plus grandes différences entre les 2 images de départ. C'est un aspect particulièrement intéressant dans le cadre de l'imagerie médicale pour potentiellement mettre en avant une zone dont la forme est particulièrement différente que sur d'autres cerveaux. Cela permet de suivre l'évolution du cerveau chez un patient atteint d'une maladie cérébrale par exemple.

Voyons désormais l'adaptabilité de notre CNN à des données sensiblement différentes de celles sur lesquelles on l'a entraîné.

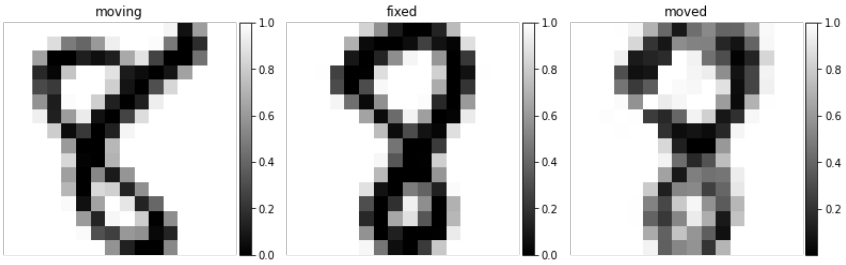

Figure 20: *Recalage avec des 8*

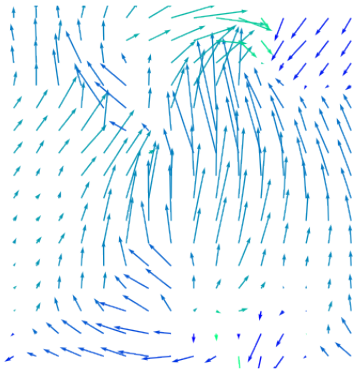

Figure 21: *Gradient pour le recalage de la figure 20*

Notre CNN sait relativement bien s'adapter aux 8 dans cet exemple. En testant sur d'autres chiffres, on remarque que le CNN s'en sort pour des chiffres ayant une forme comparable aux 2, ou du moins écrits de telles manières, mais qu'à mesure que les formes deviennent trop éloignées, comme pour un 4, le recalage devient de moins en moins bon.

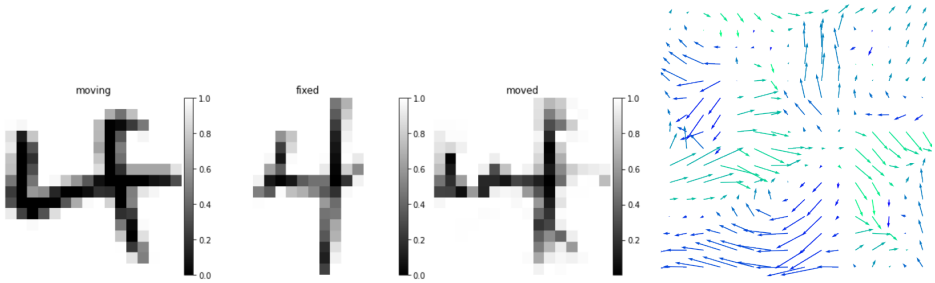

Figure 22: *Recalage et gradient avec des 4*

Faisons varier un autre aspect de nos données en entrée du CNN : l'intensité des niveaux de gris.

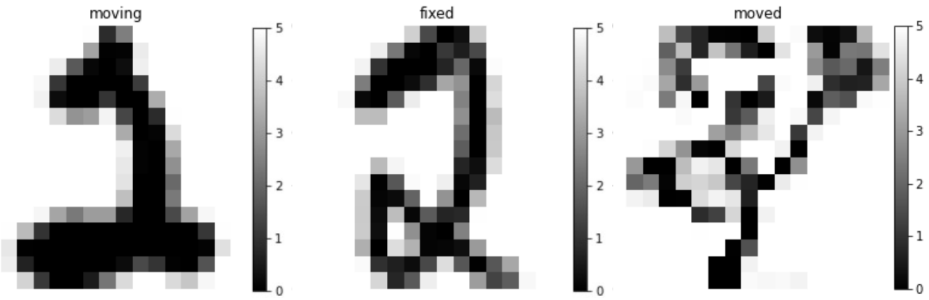

Figure 23: *Recalage avec des intensités multipliées par 5 (échelle de niveau de gris modifiée)*

Dans ce cas, le CNN n'arrive pas du tout à s'adapter. Cela peut être dû au fait que les valeurs en entrée de la fonction de déformation g_θ sont bien différentes de celles pour lesquelles elle a été entraînée, donc les valeurs de cette fonction pour de telles entrées ne sont pas bonnes face à la volonté de minimiser la fonction de perte. Nous n'avons pas contraint ses valeurs sur cet intervalle de valeur lors de l'entraînement.

- Pour $\lambda = 1$:

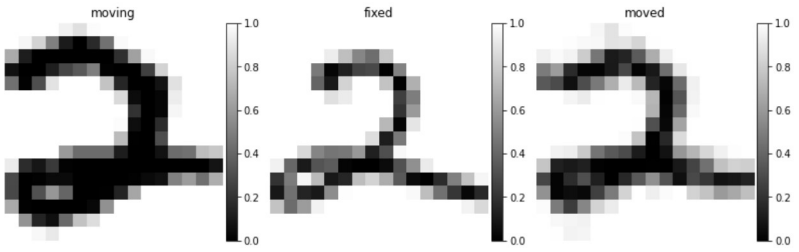

Figure 24: *Recalage pour un 2*

On remarque que l'influence d'un lambda plus grand est logiquement de contraindre les variations du gradient, et donc que le recalage est moins bon.

Et également logiquement, le CNN s'adapte moins bien à des données un peu différentes de celles sur lesquelles il a été entraîné.

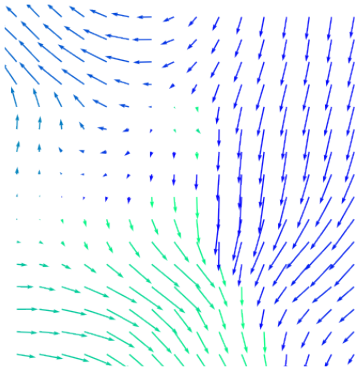

Figure 25: *Gradient pour le recalage de la figure 24*

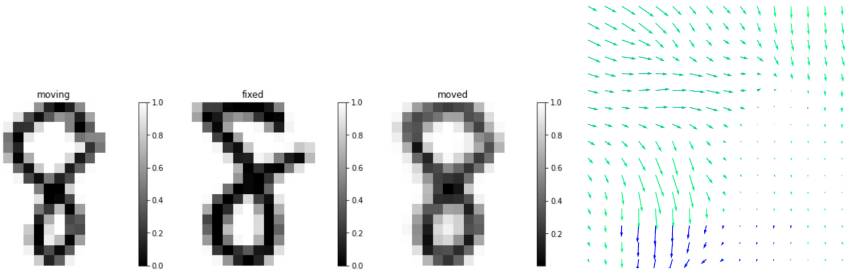

Figure 26: *Recalage et gradient avec des 8*

4
COMPARAISONS DES DIFFÉRENTES MÉTHODES

Après avoir analysé le fonctionnement des méthodes de LDDMM et de Deep Learning pour le recalage d'image, nous pouvons observer de grandes différences entre ces deux méthodes.

4.1 Régularité de la déformation

La méthode de LDDMM garantit que la transformation finale soit un difféormophisme sous réserve que le champ de déplacement v_t respecte certaines conditions de régularité et que la discrétisation du problème n'influencent pas beaucoup les résultats. Ainsi, si deux images sont si différentes qu'on ne peut pas passer d'une image à l'autre sans faire d'arrachage ou de recollement - un deux avec une boucle et un sans -, alors le recalage sera peu efficace. Cependant, **la condition de difféomorphisme sera conservée**, avec plus ou moins de régularité selon le paramètre γ. Au contraire, le CNN en Deep Learning ne garantit pas que la déformation soit un difféomorphisme : il cherche à trouver un recalage **le plus fidèle possible**, quitte à utiliser un mapping très élastique et possiblement non inversible. Par conséquent, sans un paramétrage pertinent pour garantir des déformations régulières, cet algorithme ne permettra pas de hiérarchiser la proximité entre deux images puisqu'il parviendra à toutes les recaler, comme on l'observe avec les tests effectués sur les chiffres.

4.2 Choix et nombre d'hyper-paramètres

Dans le cadre de l'algorithme de LDDMM, les résultats dépendent fortement des hyper-paramètres. À travers nos études, nous avons pu apprécier à quel point l'algorithme pouvait aboutir à des résultats différents lorsque nous changions les points de contrôles. Notre solution permettant de définir des points de contrôles automatiquement ne permet pas d'avoir des résultats convaincants, bien que les points semblent bien placés en première observation, car de petits écarts par rapport à un placement parfait perturbent grandement les résultats. Par ailleurs, nous avons également montré qu'il était nécessaire de choisir judicieusement les paramètres σ correspondant à la largeur du noyau ainsi que de γ, le paramètre de régularité. Ainsi, cette méthode requiert de bien définir les hyper-paramètres, mais offre également une certaine liberté dans son utilisation à travers ces paramètres.

Dans l'algorithme de Deep Learning, les paramètres sont moins nombreux et ne jouent pas un rôle aussi important. On peut néanmoins influencer la régularité de la fonction de déformation calculée lors de l'entraînement avec le paramètre λ. Ainsi, on peut par exemple, en autorisant de grandes déformations et en analysant le champ de déplacement des points, déterminer les zones qui ont subi d'importants changements et qui sont donc potentiellement très différentes entre les deux images.

4.3 Temps de calcul

Le temps de calcul entre ces deux méthodes est très différents. Certes, la phase d'entraînement du réseau de neurones peut être longue, mais il effectue ensuite le recalage entre deux images très rapide-

ment. Au contraire, le LDDMM ne nécessite pas de phase d'entraînement, mais calcule directement le recalage des images 2 à 2.

De plus, comme expliqué dans la partie sur le Deep Learning, les comparaisons entre 2 cerveaux se font en 3D pour mieux prendre en compte les déformations volumiques, qui sont plus pertinentes que celles surfaciques en termes d'analyse médicale. Pour des volumes, le nombre de données est naturellement très largement plus élevé que pour des images, et même si l'entraînement d'un CNN peut prendre plusieurs heures voire plusieurs jours pour de telles dimensions, il sera beaucoup plus rapide ensuite d'obtenir des résultats. La méthode de Deep Learning présente donc un avantage conséquent.

4.4 Stabilité de la méthode

Par ailleurs, l'algorithme de Deep Learning, même s'il effectue de manière très précise le recalage des images, est très **sensible aux petites variations et dépend fortement des images d'entrainement**. Par exemple, un réseau de neurones entraîné avec des images de 2 a beaucoup de mal à recaler des images de 4, comme nous l'avons vu en 3.3.2.

Pour vérifier l'instabilité des performances de l'algorithme sur des images bruitées, nous avons appliqué les deux méthodes à des images auxquelles on ajoute un bruit gaussien. On obtient alors les résultats visibles sur la figure 27 ci-dessous pour le CNN et les résultats suivants (figure 28) pour le LDDMM.

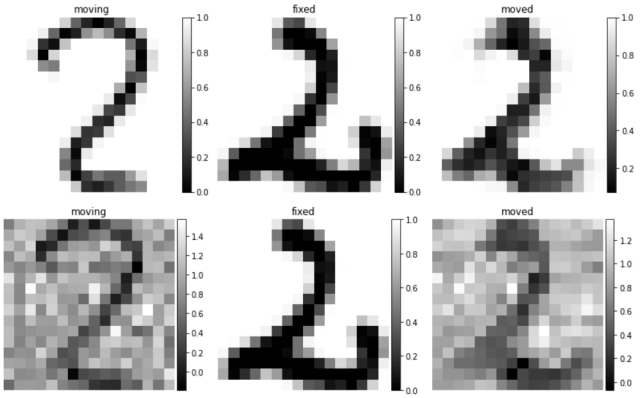

Figure 27: *Influence d'un bruit gaussien (de variance $\sigma = 0.2$) sur les résultats du CNN*

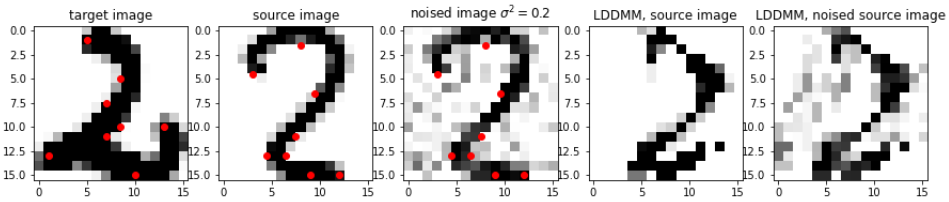

Figure 28: *Influence d'un bruit gaussien (de variance $\sigma = 0.2$) sur les résultats du LDDMM*

Sur ces figures, on observe que les performances des deux algorithmes sont conservées, l'algorithme

de Deep Learning semble stable par ajout de bruit gaussien modéré. Pour l'algorithme de LDDMM, On pouvait s'attendre à ce résultat puisque l'intensité supplémentaire du bruit gaussien n'intervient pas dans la création du difféomorphisme, le critère à minimiser ne dépendant pas de l'intensité des images.

Comme les images d'IRM dépendent à la fois de la **machine utilisée** mais aussi de **l'opérateur**, la méthode de Deep Learning est en pratique difficilement utilisable pour des IRM de cerveaux. Avant de l'utiliser, il faudrait effectuer un **pre-processing des données**, de manière à uniformiser les nuances de gris et retirer le bruit des images. De plus, les résultats seraient utilisables pour des IRM similaires, et l'entraînement d'un CNN nécessite déjà un beaucoup de données. L'utilisation effective de cette méthode n'est donc pas encore actée, et une piste de solution serait possiblement d'utiliser des méthodes de *transfer learning* pour utiliser des CNN entraînés sur des bases de données issues d'un type de machine d'IRM plus larges que celles disponibles pour les IRM d'une autre machine par exemple.

L'algorithme LDDMM ne prenant en compte que la déformation du champ de l'espace selon le critère à minimiser utilisée, on ne rencontre pas ces mêmes problèmes avec cette méthode. En revanche, le principe de l'entraînement du CNN est réplicable pour des jeux de données variés, allant de 2 manuscrits à des IRM de cerveaux, en 2D comme en 3D.

4.5 Aspect éthique des CNN

Les questions éthiques liées à l'IA concernent naturellement les réseaux de neurones entraînés qui aideraient l'humain à prendre des décisions, notamment dans le domaine médical. En effet, l'entraînement des réseaux de neurones se fait de manière autonome par la machine, une fois les hyper-paramètres et les données d'entraînement fournis par les chercheurs. Le jugement de la pertinence des résultats reste celui de l'humain, mais il est conditionné par les informations qu'il a à disposition et qui, dans le cas de l'utilisation d'un réseau de neurones pour effectuer un recalage, sont fournis par une "boîte" entraînée sur laquelle nous n'avons pas énormément de contrôle ensuite. Cet aspect est également présent dans le cas d'un difféomorphisme généré pour du LDDMM, mais dans une moindre mesure puisqu'il est généré et peut être testé facilement pour chaque paire d'images avec des points de contrôle et des hyper-paramètres différents.

Il existe également une question de respect des données privées, on parle de problématique de ***differential privacy*** pour protéger les données sensibles. Dans le cas de la médecine par exemple, un réseau de neurone peut enregistrer certaines informations du jeu de données lors de l'entraînement. Cette problématique est présente en médecine du fait du secret médical, et de l'utilisation de données personnelles. En effet, un cerveau permet d'identifier un individu. Ainsi, un important pan de la recherche en Machine Learning s'intéresse aux méthodes de *differential privacy*, qui peuvent devenir de plus en plus nécessaires dans le cadre d'études contenant des informations plus confidentielles, et qui permettraient de concilier l'utilisation des avancées techniques d'analyse et les problématiques éthiques telles que le secret médical.

5 CONCLUSION

Les méthodes de Large Deformation Diffeomorphic Metric Mapping et de Deep Learning permettent de faire efficacement du recalage d'images. Au cours de cet EA, nous avons pu tester ces deux méthodes et saisir leurs différences. Là où l'algorithme de Deep Learning est peu paramétrable, la méthode de **LDDMM** dépend fortement de nombreux hyper-paramètres. En particulier, nous avons mis en évidence l'importance des points de contrôle qui déterminent la déformation de l'image et donc le résultat. Placer manuellement les points pour s'assurer d'avoir un bon résultat est fastidieux et surtout nécessite l'avis d'un médecin dans le cadre d'imageries médicales. Ainsi, dans l'objectif d'appliquer cette méthode à de nombreuses données, nous avons créé un algorithme s'appuyant sur les différences de gradient de l'image pour placer les points de contrôles.

Les méthodes de **Convolutional Neural Networks** de Deep Learning sont quant à elles beaucoup plus rapides et effectuent un recalage très précis des images. Lorsqu'elles sont entrainées sur des images non bruitées avec des colorations similaires, on obtient des résultats très convaincants. Cependant, ces algorithmes demeurent très sensibles aux petites variations des imageries médicales et peuvent donc à l'heure actuelle difficilement être utilisées en pratique.

Enfin, plusieurs prolongements ou consolidations de notre travail peuvent être envisagés. Premièrement, nous pourrions inclure dans l'optimisation de la méthode LDDMM l'emplacement des points de contrôles afin de faire évoluer leurs positions jusqu'à l'obtention d'un résultat pertinent. Deuxièmement, un autre travail intéressant serait de calculer, à partir d'un certain nombre de données, un atlas — parfois définit arbitrairement dans le cadre d'imagerie médicale — qui représenterait l'image moyenne parmi nos données pour l'utiliser ensuite dans le recalibrage d'images. Enfin, on pourrait également changer le critère que l'on minimise dans notre algorithme, en prenant en compte par exemple la différence d'intensité entre les deux images et d'observer si cela améliore les résultats de la méthode LDDMM.

Pour les méthodes de Deep Learning, la recherche actuelle s'intéresse à la fois aux problématiques de *differential privacy*, liées au secret médical, et de *transfer learning*, pour tenter de pallier au faible nombre de données similaires et exploitables pour entraîner des réseaux de neurones.

6 ANNEXES

6.1 Codes python de nos algorithmes pour le Machine Learning

- Algorithme pour générer les points de contrôle :

```
def find_points(image, n_landmarks):
    n = len(image)
    y = n//n_landmarks
    points = np.zeros((n_landmarks,2))
    grad = np.zeros((n,n))
    for i in range(1,n-1):
        for j in range(1,n-1):
            grad[i][j] =  np.linalg.norm(image[i,j] - image[i+1,j])
            grad[i][j] += np.linalg.norm(image[i,j] - image[i,j+1])
            grad[i][j] += np.linalg.norm(image[i,j] - image[i-1,j])
            grad[i][j] += np.linalg.norm(image[i,j] - image[i,j-1])
    a,b = np.unravel_index(grad.argmax(), grad.shape)
    grad[a,b] = 0
    points[0, :] = [a,b]
    for i in range(1, n_landmarks):
        a,b = np.unravel_index(grad.argmax(), grad.shape)
        while distance(np.array([a,b]), points, i) < y + 1:
            grad[a,b] = 0
            a,b = np.unravel_index(grad.argmax(), grad.shape)
        grad[a,b] = 0
        points[i, :] = [a,b]

    return points



def assign_points(p1, p2):
    Lp_prob = p.LpProblem('Points', p.LpMinimize)
    # distance matrix
    n = len(p1)
    dist = np.zeros((n*n))
    for i in range(n):
        for j in range(n):
            dist[i*n+j] = np.linalg.norm(p1[i,:] - p2[j,:])**2
    RANGE = np.arange(n*n)
    c = p.LpVariable.dicts("c", RANGE, cat = "Integer")
```

```
    for i in RANGE:
        c[i].lowBound = 0
        c[i].upBound = 1
    Lp_prob += p.LpAffineExpression([(c[i], dist[i]) for i in range(n*n)])
    def sumi(i):
        sum = 0
        for j in range(n):
            sum += c[i*n + j]
        return sum
    def sumj(j):
        sum = 0
        for i in range(n):
            sum += c[i*n + j]
        return sum
    for i in range(n):
        Lp_prob += sumi(i) == 1
        Lp_prob += sumj(i) == 1

    status = Lp_prob.solve()
    newp2 = np.zeros((n,2))
    print(p.value(c))
    for i in range(n):
        for j in range(n):
            if p.value(c[i*n + j]):
                newp2[i,:] = p2[j,:]
    return p1, newp2
```

6.2 Codes python de nos algorithmes pour le Deep Learning

- Préambule :

```
# installer voxelmorph, qui contient aussi neurite and pystrum
!pip install voxelmorph
# imports
import os, sys
import numpy as np
import tensorflow as tf

# local imports
import voxelmorph as vxm
import neurite as ne
```

- Préparation des données :

```
train_data = np.loadtxt('zip.train')
nb = len(train_data)

resh = -(train_data[:,1:].reshape(7291,16,16) - 1)/2
num = np.array([int(train_data[:,:1][i]) for i in range(nb)])

from sklearn.model_selection import train_test_split

x_train_load, x_val_load, y_train_load, y_val_load = train_test_split(
    resh, num, test_size=0.3, random_state=10
)

digit_sel = 2

# recuperer seulement les instances ayant pour valeur 'digit_sel'
x_train = x_train_load[y_train_load==digit_sel, ...]
y_train = y_train_load[y_train_load==digit_sel]
x_val = x_val_load[y_val_load==digit_sel, ...]
y_val = y_val_load[y_val_load==digit_sel]


print('shape of x_train: {}, y_train: {}'.format(x_train.shape, y_train.shape))
print('shape of x_test: {}, y_test: {}'.format(x_val.shape, y_val.shape))
```

- Construction du modèle de CNN :

```
# configure unet input shape (concatenation of moving and fixed images)
ndim = 2
unet_input_features = 2
inshape = (*x_train.shape[1:], unet_input_features)

nb_features = [
    [16, 16, 16, 16],          # encoder features
    [16, 16, 16, 16, 16, 8]  # decoder features
]

# construction du modele en utilisant VoxelMorph
inshape = x_train.shape[1:]
vxm_model = vxm.networks.VxmDense(inshape, nb_features, int_steps=0)

# Definition de la fonction de perte/cout
losses = [vxm.losses.MSE().loss, vxm.losses.Grad('l2').loss]
```

```
# le lambda dans de L(F,M,phi) devant le terme de regularisation de phi
lambda_param = 0.05 # on peut faire varier ce choix
loss_weights = [1, lambda_param]

vxm_model.compile(optimizer='Adam', loss=losses, loss_weights=loss_weights)
```

- Entraînement du CNN :

Ce préparateur des données vient directement de la documentation de VoxelMorph.

```
def vxm_data_generator(x_data, batch_size=32):
    """
    Generator that takes in data of size [N, H, W], and yields data for
    our custom vxm model. Note that we need to provide numpy data for each
    input, and each output.

    inputs:  moving [bs, H, W, 1], fixed image [bs, H, W, 1]
    outputs: moved image [bs, H, W, 1], zero-gradient [bs, H, W, 2]
    """
    # preliminary sizing
    vol_shape = x_data.shape[1:] # extract data shape
    ndims = len(vol_shape)

    # prepare a zero array the size of the deformation
    zero_phi = np.zeros([batch_size, *vol_shape, ndims])

    while True:
        # prepare inputs:
        # images need to be of the size [batch_size, H, W, 1]
        idx1 = np.random.randint(0, x_data.shape[0], size=batch_size)
        moving_images = x_data[idx1, ..., np.newaxis]
        idx2 = np.random.randint(0, x_data.shape[0], size=batch_size)
        fixed_images = x_data[idx2, ..., np.newaxis]
        inputs = [moving_images, fixed_images]

        # prepare outputs (the 'true' moved image):
        # of course, we don't have this, but we know we want to compare
        # the resulting moved image with the fixed image.
        # we also wish to penalize the deformation field.
        outputs = [fixed_images, zero_phi]

        yield (inputs, outputs)
```

Entraînons notre modèle sur les données chargées précédemment :

```
train_generator = vxm_data_generator(x_train)
in_sample, out_sample = next(train_generator)

# visualisation
images = [img[0, :, :, 0] for img in in_sample + out_sample]
titles = ['moving', 'fixed', 'moved ground-truth (fixed)', 'zeros']
ne.plot.slices(images, titles=titles, cmaps=['gray'], do_colorbars=True);

nb_epochs = 20
steps_per_epoch = 100
hist = vxm_model.fit_generator(train_generator, epochs=nb_epochs,
steps_per_epoch=steps_per_epoch, verbose=2);
```

Visualisons l'évolution de la valeur prise par notre fonction de perte au cours de l'entraînement :

```
import matplotlib.pyplot as plt

def plot_history(hist, loss_name='loss'):
    # Simple function to plot training history.
    plt.figure()
    plt.plot(hist.epoch, hist.history[loss_name], '.-')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.show()

plot_history(hist)
```

- Test de notre CNN entraîné sur d'autres données :

```
val_generator = vxm_data_generator(x_val, batch_size = 1)
val_input, _ = next(val_generator)

val_pred = vxm_model.predict(val_input)
```

Voyons les résultats :

```
images = [img[0, :, :, 0] for img in val_input + val_pred]
titles = ['moving', 'fixed', 'moved', 'flow']
ne.plot.slices(images, titles=titles, cmaps=['gray'], do_colorbars=True);

ne.plot.flow([val_pred[1].squeeze()], width=5);
```

- Test de l'adaptabilité de notre CNN :

Testons le sur des 7 :

```
x_sevens = x_train_load[y_train_load==8, ...]

seven_generator = vxm_data_generator(x_sevens, batch_size=1)
seven_sample, _ = next(seven_generator)
seven_pred = vxm_model.predict(seven_sample)

images = [img[0, :, :, 0] for img in seven_sample + seven_pred]
titles = ['moving', 'fixed', 'moved', 'flow']
ne.plot.slices(images, titles=titles, cmaps=['gray'], do_colorbars=True);

ne.plot.flow([seven_pred[1].squeeze()], width=5);
```

Voyons comment notre CNN s'adapte à des images de 2 dont l'intensité a été modifiée :

```
factor = 5
val_input_intensity = [f * factor for f in val_input]
val_pred = vxm_model.predict(val_input_intensity)

# visualizeb
images = [img[0, :, :, 0] for img in val_input_intensity + val_pred]
titles = ['moving', 'fixed', 'moved', 'flow']
ne.plot.slices(images, titles=titles, cmaps=['gray'], do_colorbars=True);

ne.plot.flow([seven_pred[1].squeeze()], width=5);
```

BIBLIOGRAPHIE

[1] Jim Duncan, Nicholas Ayache *Medical Image Analysis: Progress over two decades and the challenges ahead*, https://hal.inria.fr/inria-00615100 (2013)

[2] Pietro Gori *Introduction to image registration*

[3] Mirza Faisal Beg, Michael I. Miller, Alain Trouvé, Laurent Younes
Computing Large Deformation Metric Mappings via Geodesic Flows of Diffeomorphisms

[4] Stanley Durrleman, Stéphanie Allassonnière, Sarang Joshi
Sparse Adaptive Parameterization of Variability in Image Ensembles

[5] Guha Balakrishnan, Amy Zhao, Mert R. Sabuncu, John V. Guttag et Adrian V. Dalca
An Unsupervised Learning Model for Deformable Medical Image Registration, CoRR, abs/1802.02604 (2018)

[6] Base de données de chiffres manuscrits de l'U.S. Postal Service

[7] Documentation et tutoriels de DeepLizard

[8] GitHub de VoxelMorph