INF8225 - Projet
Adaptation - Landmark Detection and Tracking in Ultrasound using a CNN-RNN Framework

Denis CORBIN – 1794521
Tal MEZHERITSKY – 1788524
Polytechnique Montréal

Resumé

Nous présentons une adaptation du travail intitulé – « *Landmark Detection and Tracking in Ultrasound using a CNN-RNN Framework* » dans la cadre du concours *MICCAI CLUST 2015 challenge dataset* (De Luca et al. 2015). Dans ce travail, nous effectuons d'abord l'entraînement d'un réseau de neurones complètement convolutif (CNN) de type auto-encodeur afin d'identifier l'emplacement de vaisseaux sanguins dans une image ultrason. Nous effectuons ensuite l'entraînement d'un réseau de neurones récurent (RNN) de type LSTM afin de peaufiner nos résultats par suite de l'encodage afin d'introduire une notion de séquence. Ces entraînements sont effectués en faisant varier plusieurs paramètres clés tels que le *learning rate*, la taille des *minibatch*, l'ajout de *dropout*, l'ajout de *features maps* et le nombre *d'epochs*. Les réseaux sont ensuite combinés afin d'obtenir des résultats qui diffère de l'article de référence, mais qui sont tout de même concluants.

1 Introduction

L'imagerie par ultrasons (US) est une modalité d'imagerie très utilisée dans le domaine médical. C'est une méthode non invasive, peu couteuse et qui possède une bonne résolution temporelle. Sa résolution temporelle élevée rend l'US une modalité idéale pour des applications de suivi de mouvement d'organes ainsi que les interventions guidées par imagerie.

Dans le but d'uniformiser la recherche sur les algorithmes de suivi d'organes par US, la société MICCAI a mis en ligne une base de données annotée d'acquisitions du foie par US. Sur ces acquisitions, il est possible de voir certains vaisseaux principaux du foie. Ceux-ci apparaissent comme des ellipses noires dans les images (Figure 1). Cette base de données a été rendue publique dans le cadre d'un concours nommé CLUST15 (De Luca et al. 2015). Le but du concours est de concevoir un algorithme qui peut, en temps réel, suivre de manière automatique la position des vaisseaux visibles sur les acquisitions d'US.

Figure 1 : Exemple d'acquisition du foie par US fournie dans le concours CLUST15. Deux vaisseaux principaux sont identifiés par des points jaunes.

Le site web du concours CLUST15 fournit une liste des implémentations les plus performantes avec des liens vers les articles décrivant ces approches. Puisque la majorité n'utilisent pas des approches d'intelligence artificielle discutées dans le cadre du cours INF8225, nous ne les décrierons pas.

Une des approches utilisant un réseau CNN a été publiée par (Nouri et al. 2015). Le réseau CNN est entraîné à générer des représentations de basse résolution de l'image d'entrée. Le but est de réduire la dimensionalité de telle sorte que des segments de l'image originale qui contiennent la cible se retrouvent près l'une de l'autre une fois rendue à la représentation de basse résolution.

Parmi les articles proposant des solutions à ce concours, nous avons trouvé l'article de (Rangamani et al. 2016) intitulé «*Landmark Detection and Tracking in Ultrasound using a CNN-RNN Framework* ». Dans cet article les auteurs proposent une architecture d'un réseaux neuronal convolutif (CNN) qui sert d'encodeur-décodeur auquel s'ajoute un réseau récurrent (RNN) de type LSTM. Cette architecture sera décrite plus en détail à la section 2 de ce rapport. Vue les excellents résultats réclamés par les auteurs, et notre intérêt envers l'imagerie médicale, nous avons décidé de tenter de répliquer le réseau décrit dans l'article et comparer nos résultats.

Dans le cadre de ce rapport, nous allons débuter par une courte revue littéraire des autres approches qui ont été utilisées dans CLUST15. Nous allons ensuite décrire en détail l'architecture que nous avons conçue ainsi que les

expériences que nous avons effectuées pour l'optimiser. À la section 3, nous présenteront les résultats de nos expériences et nous les discuteront. Finalement, nous présenteront une discussion critique de notre approche ainsi qu'une description des difficultés auxquelles nous avons fait face durant le projet.

2 Méthodes

Dans cette section, nous allons détailler l'architecture que nous avons développée. L'architecture est en majorité basée sur celle proposée par (Rangamani et al. 2016), mais comporte quelques différences. Cette architecture est composée d'une part d'un CNN qui permet d'identifier l'emplacement de la cible dans l'image d'US. La deuxième composante de l'architecture est un réseau RNN de type LSTM. Ce réseau utilise la relation temporelle entre les versions encodées des images d'US pour optimiser la position obtenue par le réseau CNN. Les informations de ces deux réseaux sont combinées pour obtenir la position de la cible.

Nous allons commencer par expliquer le réseau CNN, ensuite le réseau LSTM. Puis, nous expliquerons comment nous combinons leurs sorties et finalement, nous présenteront les paramètres d'entraînement et les expériences que nous avons effectués dans le but d'optimiser notre architecture. Toutes les implémentations ont été faites avec la libraire PyTorch de Python.

2.1 Encodeur-Décodeur CNN

La composante CNN de l'architecture sert à obtenir la position de la cible en prenant comme entrée une seule image brute d'US. Le réseau CNN est complètement convolutif, c'est-à-dire qu'en entrée et à la sortie nous obtenons une image de la même taille (640x480 pixels). La figure 2 montre un schéma du réseau CNN implémentée.

Le réseau CNN peut être divisé en un encodeur et un décodeur. L'encodeur prend en entrée une image d'US et génère une représentation de cette image dans un espace latent de dimension réduite (20x15 pixels). L'encodeur est composé de 6 étapes de convolution (taille 3x3, stride 1, padding 1) suivie d'une normalisation de batch et une couche d'activation ReLu. Après chacune de ces étapes, une réduction de la dimension est appliquée avec une couche de max-pooling (taille 2x2, stride 2

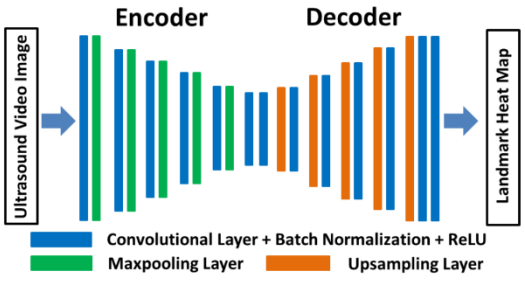

Figure 2 : Représentation schématique du réseau CNN encodeur-décodeur implémenté (Rangamani et al. 2016)

Le décodeur prend en entrée la représentation latente de l'image originale et donne en sortie une image ayant la taille originale qui permet de déterminer la position de la cible avec une opération de moyenne pondérée sur les intensités des pixels. Cette opération sera détaillée à la section 2.4. De manière similaire à l'encodeur, le décodeur est composé de 6 étapes de convolution (taille 3x3, stride 1, padding 1), suivie d'une normalisation de batch et une activation ReLu. Pour augmenter la dimension de l'image, le décodeur utilise des couches de suréchantillonnage avec un facteur d'agrandissement de 2. Une dernière convolution de la même taille est appliquée à l'image avant d'obtenir la sortie.

2.2 LSTM

Bien que le réseau CNN soit capable d'identifier l'emplacement de la cible dans une image "statique" dans le temps, il ne permet pas de capturer les variations temporelles dans les séquences d'images d'US. C'est donc dans ce but que la composante LSTM est ajoutée à l'architecture globale.

Le réseau LSTM va opérer sur les représentations latentes produites par l'encodeur. Il prend en entrée une séquence de 3 images US encodées correspondantes aux temps $t-3$ à $t-1$. Le réseau produit à la sortie une prédiction de l'image encodée au temps présent, c'est-à-dire le temps t. Pour ce faire, le réseau est composé de 3 couches ou 3 unités LSTM. La séquence est composée des 3 images encodées qui sont « aplaties » pour former 3 vecteurs d'une longueur de 300 chiffres. Chaque unité LSTM reçoit une seule image encodée en entrée, la variable cachée h est mise à jour à chaque couche pour obtenir finalement la sortie y finale qui est un seul vecteur de taille 300 qui est ensuite réorganisé comme un tenseur de dimensions (1,1,15,20) pour les étapes subséquentes.

Le réseau LSTM apprend donc à modéliser les variations temporelles dans l'espace latent des séquences d'images d'US. Cette prédiction peut être ensuite combinée avec celle produite par le réseau CNN.

2.3 Combinaison CNN-LSTM

En ayant 2 approches pour la détection de la cible anatomique, (Rangamani et al. 2016) ont proposé de combiner les prédictions du réseau CNN et du réseau LSTM.

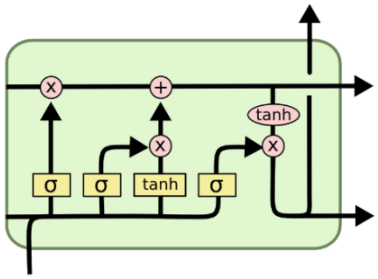

Figure 3 : Schéma de l'architecture d'une cellule LSTM

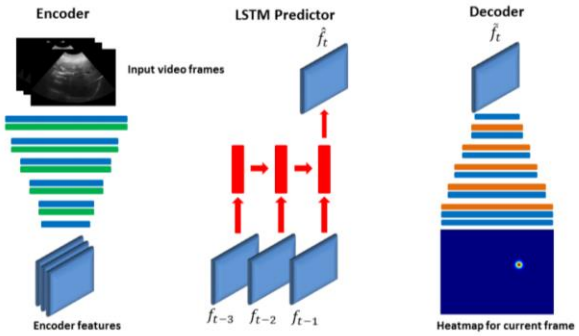

Figure 4 : Schéma décrivant le fonctionnement de l'architecture complète. La sortie du réseau LSTM est combinée à celle du CNN dans l'espace latent (Rangamani et al. 2016)

L'hypothèse est que combiner les forces de chacune des approches, soit l'analyse d'image du CNN et la capture de relations temporelles du LSTM, permettrait d'obtenir de meilleurs résultats. La figure 4 montre schématiquement la combinaison des deux approches.
La manière proposée pour combiner les réseaux est d'intervenir au niveau de la représentation latente à la suite de l'encodage. Une fois que l'image du temps t est encodée, elle est ajoutée à la prédiction de l'espace latent produite par le réseau LSTM. Le réseau LSTM ayant accès aux représentations latentes des images précédentes. La combinaison se fait selon cette expression :

$$f = \lambda f_t + (1 - \lambda)\hat{f}_t \quad (1)$$

Où $\hat{f}_t$est la prédiction de la variable latente au temps t du LSTM et f_t est l'image au temps t encodée par le CNN. C'est donc une simple combinaison proportionnelle des 2 résultats qui varie selon le paramètre λ. Cette nouvelle variable latente f peut ensuite être traitée par le décodeur du réseau CNN pour finalement obtenir les coordonnées de la cible.

2.4 Entraînement et expériences

Dans cette sous-section, nous allons décrire plus en détail comment nous avons entraîné l'architecture décrite ci-haut

Entraînement de l'encodeur-décodeur
Pour le réseau encodeur-décodeur CNN, nous avons initialement suivi l'entraînement suggéré par les auteurs. C'est-à-dire qu'en entrée nous donnons l'image US brute et à la sortie nous voulons obtenir une image contenant une distribution gaussienne à l'endroit où la cible se trouve (figure 5). L'ensemble de données CIL-01 du concours CLUST15 contient 144 images US annotées. Nous avons séparé ces images en 100 images d'entrainement, 22 images de validation et 22 images de test. Ensuite, pour chacune des images annotées nous avons générée le *heatmap* correspondant. Ce résultat est une image noire avec une distribution gaussienne avec un rayon de 10 centrée à l'endroit où se trouve la cible. Pour le taux d'apprentissage, nous commençons avec 0.01 et nous diminuons à 0.001 après 75% des *epochs*.

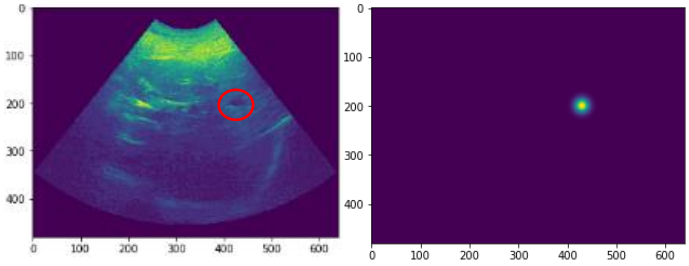

Figure 5 : Exemple d'une paire image/*heatmap* utilisé pour l'entraînement du CNN

No. de couche	Canaux d'entrée	Canaux de sortie
Conv1	1	2
Conv2	2	2
Conv3	2	4
Conv4	4	4
Conv5	4	8
Conv6	8	8

Tableau 1: Nombre de canaux d'entrée et de sortie pour chaque couche convolutive de l'encodeur

Puisque (Rangamani et al. 2016) n'ont pas mentionné la fonction de perte utilisée pour l'entraînement, nous avons commencé avec une métrique de similarité d'erreur au carrée (MSE) ou fonction de perte L2. À cause de résultats non concluants, nous avons ajouté une nouvelle métrique à la perte L2. Celle-ci a été tiré de (Xu et al. 2018) et elle pose une contrainte sur la distance entre la position prédite par le réseau et la position réelle de la cible dans le but de régulariser les résultats du réseau. La métrique s'appelle *Landmark location error* ou LLE. La LLE se calcule de la manière suivante :

$$\hat{s} = \frac{\sum H_i S_i}{\sum H_i}, \quad \hat{t} = \frac{\sum H_i T_i}{\sum H_i} \quad (2)$$

$$LLE = \sqrt{(s - \hat{s})^2 + (t - \hat{t})^2} \quad (3)$$

Où $\hat{s}$ et $\hat{t}$ sont les coordonnées prédites par le réseau selon les 2 axes de l'image. Elles sont calculées à partir des intensités de l'image générée H et les positions des intensités dans l'image (S, T). Finalement la LLE est la distance euclidienne entre la position prédite $(\hat{s}, \hat{t})$ et la vraie position (s, t). C'est essentiellement un calcul de la moyenne des intensités, pondérée selon la position dans l'image.

La LLE a également servie à la définition de notre critère de précision pour évaluer le réseau aux étapes de validation et de test. Puisque notre tâche ne se rapporte pas à la classification, nous avons décidé de mettre un seuil sur la distance entre la position prédite et celle attendue pour décider si un exemple a été réussi ou non. Ce seuil a été fixé à 15 pixels. Si

la position prédite par le réseau se trouve à une distance inférieure à 15, l'exemple est considéré comme réussi et va faire augmenter le score de précision du réseau pour l'*epoch* courante.

Une autre information manquante dans l'article de (Rangamani et al. 2016) est le nombre de *feature maps* générées par chaque couche de convolution. Dans nos expériences, nous avons débuter avec 1 seule *feature map* pour chaque couche. Toutes les convolutions s'appliquaient sur l'image originale et on se fiait sur l'opération de *maxpooling* pour encoder les informations. Nous avons également essayé de générer plus de *feature maps*. Le tableau 1 montre l'évolution du nombre de *feature maps* dans l'encodeur. Le décoder fait l'opération inverse et réduit le nombre de *feature maps* de 8 à 1. Nous nous sommes limités à 8 *feature maps* pour des raisons de limitations de puissance de calcul de nos ordinateurs.

Finalement, pour déterminer les autres hyperparamètres optimaux pour l'entraînement, nous avons effectués différentes expériences. Nous avons fait varier le nombre d'*epochs* d'entraînement et la taille des *minibatch*. Nous avons évalué aussi la performance du réseau avec l'utilisation de couches *Dropout*. Lorsque des couches *Dropout* ont été utilisées, ces couches étaient placées à la suite de chaque couche de convolution avec une probabilité de 0.1 et 0.5 pour la dernière couche (Park et al. 2017).

Entraînement du réseau LSTM
Comme expliqué à la section 2.2, le réseau LSTM prend en entrée une séquence de 3 images US encodées par l'encodeur du CNN et donne en sortie une prédiction de l'image encodée du pas de temps suivant. Il fallait donc obtenir un modèle CNN satisfaisant avant de procéder à la préparation des données pour le réseau LSTM. Une fois le meilleur réseau CNN identifié, nous avons encodés toutes les images disponible (annotées ou non) dans l'ensemble de données CIL-1. Ensuite, les données obtenues ont été séparées en 3 ensembles : entraînement, validation et test. Il est important de noter que les exemples ont été séparés de la même manière que pour l'entraînement du réseau CNN. C'est-à-dire qu'une image qui a servi à entraîner le réseau CNN ne se retrouve pas dans l'ensemble test du réseau LSTM. Cette remarque est importante à l'étape où l'architecture complète est évaluée.

Chaque exemple de la base de données est composé d'une séquence de 3 images encodées qui précède l'image qui a été annotée dans la base de données. Par exemple, si l'image annotées est la numéro 20, les images qui sont données en entrée au réseau LSTM sont les images 17,18 et 19. Donc, en entrée le réseau prends un tenseur de dimensions (3,1,300) et en sortie il redonne un tenseur de dimensions (3,1,300) également. Puisqu'on s'intéresse seulement à la dernière sortie du réseau nous retournons seulement la dernière composante du tenseur qui est de taille (1,1,300).

La fonction de perte pour l'entraînement du réseau LSTM n'a pas été spécifiée par (Rangamani et al. 2016) non plus. Nous avons décidé d'utiliser MSE une fois de plus puisqu'on veut que le réseau apprenne à prédire l'ensemble de la représentation latente de l'image. Pour le score de précision, nous avons mis un seuil de MSE de 4%. C'est-à-dire qu'un exemple est considéré réussi si l'espace latent obtenue ressemble à 96% à l'espace latent voulu. Les expériences d'entrainement ont fait varier le nombre d'*epochs* et la taille des *minibatch*. Le taux d'apprentissage était le même que pour l'entraînement du réseau CNN.

Test de l'architecture complète
Une fois que le réseau CNN et le réseau LSTM sont entraînés, il est possible de tester l'architecture complète et comparer sa performance à celle du réseau CNN seul. Pour ce faire, nous avons utilisés l'ensemble test des 2 réseaux composé de 22 exemples. Ces exemples sont les même que ceux utilisés pour la phase de test du réseau CNN seul. Pour chaque exemple, l'image courante (temps = t) est donnée en entrée au réseau CNN, et les images encodées précédente (temps = $t-3$ à $t-1$) sont données en entrée au réseau LSTM. Ensuite, la représentation latente courante (temps = t) du CNN et du LSTM sont combinés selon l'équation (1). Finalement la représentation latente combinée est donnée au décodeur. Le résultat est évalué avec la perte LLE+MSE. À titre d'expérience, nous avons fait varier le paramètre de combinaison λ afin d'identifier le ratio ideal pour combiner la sortie du CNN et du LSTM.

3 Résultats

3.1 Encodeur-Décodeur CNN

L'auto-encodeur complètement convolutif a d'abord été implémenté et plusieurs variations de l'architecture globale présentée dans les sections précédentes ont été testées. Nous avons d'abord fait varier le nombre d'*epochs* entre 100, 300, 500 et 1000. Nous avons constaté que le retour sur investissement diminuait rapidement plus on augmentait le nombre d'*epochs*. En effet, les *epochs* supérieurs à 200 conduisaient généralement au surentraînement de notre CNN, ce qui faisait en sorte qu'un nouveau meilleur modèle n'était jamais retrouvé puisque la précision sur l'ensemble de validation diminuait ou stagnait. Les résultats pour 500 et 1000 epochs sont disponibles à la figure 6 tandis que ceux pour 300 epochs seront présentés à travers les prochains résultats (Figure 7 – E et F).

Le nombre d'*epochs* qui semblait le plus intéressant et qui a été conservé pour les prochaines expériences était celui de 300. Ce nombre nous assurait d'entraîner le réseau le plus possible tout en limitant le nombre d'*epochs* inutiles liés au surentrainement et le temps de calcul global. La précision sur l'ensemble test était plus élevé que sur celui de validation, une observation qui sera discutée dans la section suivante.

La seconde avenue qui a été explorée est celle qui faisait varier la taille des *minibatchs*. Pour ce faire, considérant que nous avons seulement utilisé 100 images pour l'entraînement, nous avons testé des *minibatchs* de taille égale à 1, 5 et 20 avec 300 *epochs*. Le but de cette approche était de trouver le

meilleur compromis entre le temps d'entraînement et la précision obtenue sur l'ensemble de validation. On constate sur la figure 7 qui regroupe les 3 courbes de précision (A-C-E) que l'essai effectué avec les *minibatchs* de 1 est de loin celui qui permet d'obtenir de meilleurs résultats.

Il faut toutefois noter que plus la taille des *minibatchs* est faible, plus le temps d'entraînement augmente et cela de manière significative. Puisque nous avons défini notre propre fonction de perte (LLE → Équation 2 et 3), elle n'est pas autant optimisée que celles qui sont déjà implémentées sous *PyTorch* telles que la MSE, un aspect qui sera discuté à la section 4. Afin de tester d'autres architectures, une taille de *minibatch* de 20 a été utilisée, mais une taille de *minibatch* de 1 a tout de même été utilisée pour l'architecture finale considérant qu'elle donne la meilleure précision au détriment d'un coût plus élevé en calcul.

Nous avons ensuite tenté d'ajouter des couches *dropout* dans notre architecture afin de régulariser notre entrainement. Des couches *dropout* avec un probabilité de 0.1 ont été utilisés à chaque couche, sauf à la dernière où la probabilité était de 0.5. Un nombre d'*epochs* égale à 300 ainsi qu'une taille de *minibatch* de 20 ont été utilisés à des fins de rapidité. Les couches *dropout*, qui sont censées ignorer certains poids du réseau lors de l'entraînement et ainsi palier au problème de surentraînement, n'ont pas produit les résultats escomptés.

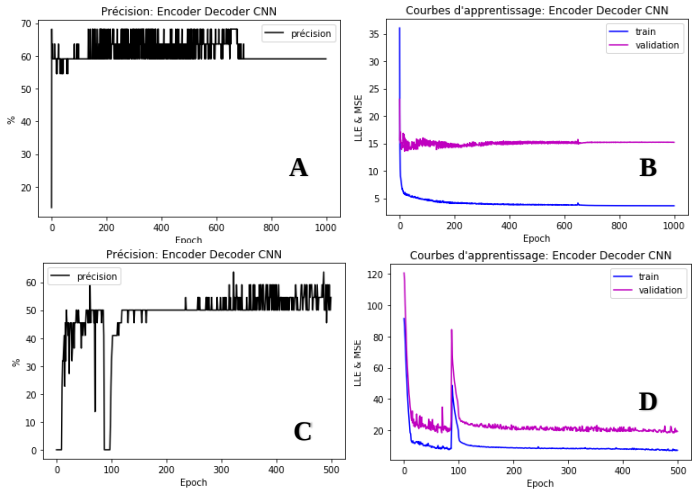

Figure 6 : Effet de la variation du nombre d'*epochs* sur l'entrainement avec un taux d'apprentissage variable et une taille de *minibatch* de 20. A (Précision sur l'ensemble validation) et B (Perte) avec 1000 *epochs* (77% sur l'ensemble test avec une perte moyenne de 12.0304). C (Précision sur l'ensemble validation) et D (Perte) avec 500 *epochs* (67% sur l'ensemble test avec une perte moyenne de 13.5634).

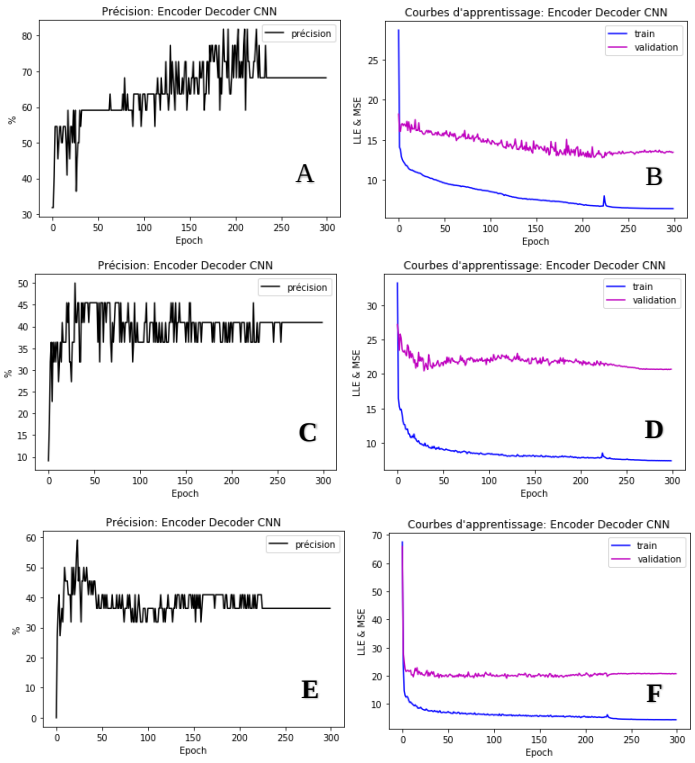

Figure 7 : Effet de la variation de la taille des *minibatchs* sur l'entrainement avec un taux d'apprentissage variable pour 300 *epochs*. A (Précision sur l'ensemble validation) et B (Perte) avec *minibatch* size = 1 (77% sur l'ensemble test avec une perte moyenne de 17.6428). C (Précision sur l'ensemble validation) et D (Perte) avec *minibatch* size = 5 (36% sur l'ensemble test avec une perte moyenne de 20.4307). E (Précision sur l'ensemble validation) et F (Perte) avec *minibatch* size = 20 (50% sur l'ensemble test avec une perte moyenne de 24.2521).

En effet, comme on peut le voir sur la figure 8, la précision optimale (~25%) sur l'ensemble de validation était significativement plus faible à comparer des essais précédents qui n'incluaient pas de Dropout dans leur architecture. Cette faible précision sur l'ensemble de validation s'est inévitablement traduite en une plus faible précision sur l'ensemble de test (0%). Ce résultat est indéniablement plus faible que le 77% obtenu sans *dropout* avec des *minibatchs* de 1 et 300 *epochs* (Figure 7 – A et B) Il va donc sans dire que les *dropouts* ne feront pas parties de l'architecture finale du CNN en raison des résultats obtenus ci-haut.

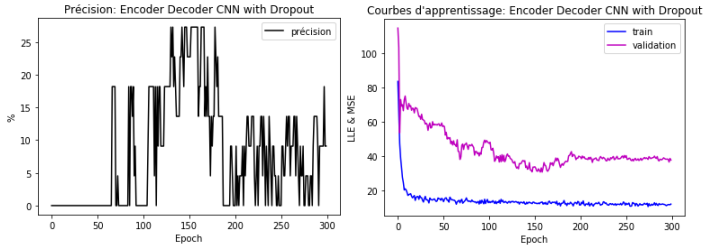

Figure 8 : Effet de l'ajout de Dropout dans l'architecture de notre réseau CNN avec un taux d'apprentissage variable pour 300 epochs et une taille de minibatch de 20 sur la précision sur l'ensemble validation et la perte. (0% sur l'ensemble test avec une perte moyenne de 35.3564).

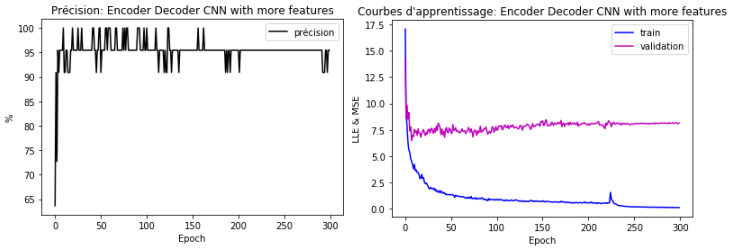

Figure 9 : Effet de l'ajout de features maps supplémentaire dans l'architecture de notre réseau CNN avec un taux d'apprentissage variable pour 300 epochs et une taille de minibatch de 1 sur la précision sur l'ensemble validation et la perte. (91% sur l'ensemble test avec une perte moyenne de 9.0386).

Finalement, la dernière avenue qui a été explorée est celle qui utilisait plusieurs *features maps*. Cette dernière avenue est celle qui nous as permis d'obtenir les meilleurs résultats jusqu'à maintenant comme on peut le voir dans la Figure 9.

L'ajout de plus de *features maps* dans notre architecture est donc concluante par le très haut taux de précision qu'elle nous procure sur l'ensemble de test. C'est donc l'architecture présentée dans la figure 9 (Architecture composée de plusieurs *features maps* sans *dropout* entraînée par *minibatch* de 1 sur 300 *epochs* avec un taux d'apprentissage variable) qui sera utilisée dans notre architecture finale ainsi que pour entraîner notre LSTM qui sera le sujet de la prochaine sous-section.

3.2 LSTM

Les expériences sur notre LSTM étaient plutôt limitées puisque nous voulions conserver le plus d'homogénéité avec l'entraînement de notre CNN afin qu'un entraînement commun soit une possibilité dans le futur. Ainsi, le nombre d'*epochs* a été fixé à 300 et les couches *dropout* n'ont pas été employées considérant qu'elles n'ont pas donné de bons résultats dans la section précédente. De plus, nous avons seulement effectué les tests avec l'architecture qui incluait plusieurs *features maps* puisque cette dernière était évidemment

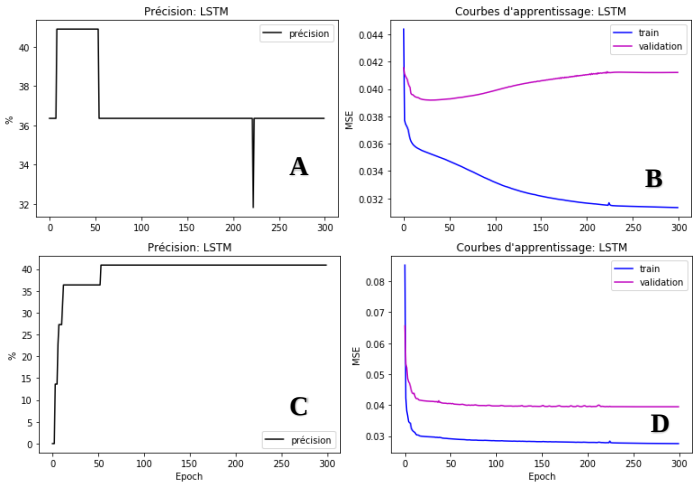

Figure 10 : Effet de la variation de la taille des *minibatchs* sur l'entrainement avec un taux d'apprentissage variable pour 300 *epochs*. A (Précision sur l'ensemble validation) et B (Perte) avec minibatch size = 1 (32% sur l'ensemble test avec une perte moyenne de 4.256%). C (Précision sur l'ensemble validation) et D (Perte) avec minibatch size = 20 (64% sur l'ensemble test avec une perte moyenne de 4.0712%).

la plus performante. Ainsi, l'unique exploration consistait à faire varier la taille des *minibatchs* afin d'en observer l'effet. Ces résultats sont rapportés ci-dessus, soit à la Figure 10.

On constate que la taille de *minibatch* de 20 est celle qui semble la plus appropriée dans le cas du LSTM. On remarque également le surentraînement de manière significative sur cette figure où l'on constate qu'après 25 epochs pour les *minibatchs* de 1, l'erreur sur l'ensemble de validation (Figure 10 - B) commence à augmenter, ce qui rend les *epochs* subséquentes inutiles. Le meilleur LSTM qui sera utilisé dans notre architecture finale est donc celui qui ingurgitera plusieurs *features maps* sans *dropout* entraînée par *minibatch* de 20 sur 300 *epochs* avec un taux d'apprentissage variable. La combinaison de notre CNN et de notre LSTM sera présentée à la sous-section suivante.

3.3 Combinaison CNN/LSTM

Ayant entraînés notre CNN et notre LSTM avec des paramètres d'entraînement optimaux, nous sommes maintenant prêts à les combiner afin d'observer si l'ajout d'un LSTM permet de mieux prédire la position de la cible. En utilisant seulement le CNN, on peut se rappeler qu'une identification réussite du vaisseau sanguin (distance inférieure à 15 pixels par rapport au « ground truth ») était observable 91% du temps sur l'ensemble test avec une perte moyenne de 9.0386. En combinant les deux approches et en faisant varier le facteur combinatoire λ, on peut observer les résultats présentés à la Figure 11.

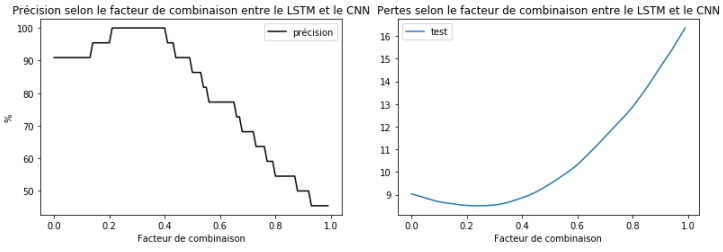

Figure 11 : Effet de la variation du facteur combinatoire $\boldsymbol{\lambda}$ sur la précision sur l'ensemble de test. Le meilleur facteur combinatoire avec un précision de ± 0.05 est de $\boldsymbol{\lambda}$ = 0.25 avec une précision de 100% sur l'ensemble test et une perte de 8.5138.

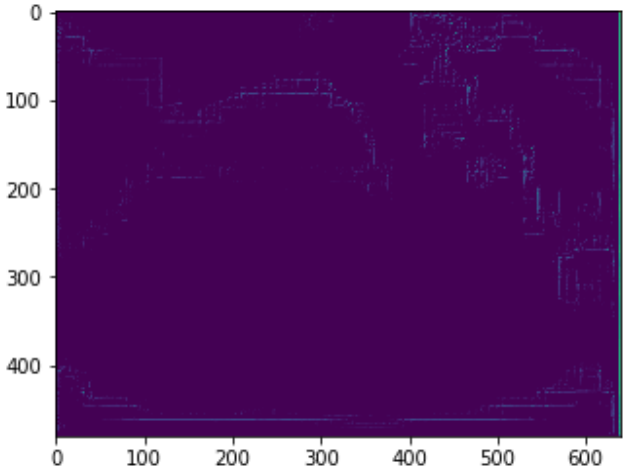

Figure 12 : Sortie obtenue à la suite du passage d'une image d'entrée dans le réseau CNN + LSTM.

On constate que notre facteur $\boldsymbol{\lambda}$(0.25) est très proche de celui obtenu par (Rangamani et al. 2016) qui était de 0.2.

Finalement, les images obtenues à la sortie notre réseau de neurones ressemblaient à l'image présentée à la Figure 12, donc rien de visuellement concret.

Toutefois, en calculant la moyenne pondérée des intensités de cette image comme il a été calculé pour la LLE, on retrouve la position prédite du centre de notre cible qu'on peut facilement reconstruire et comparer visuellement et conclure sur la validité de notre algorithme. Le résultat reconstruit selon cette méthode nous donne la Figure 13 – A que l'on peut comparer avec la Figure 13 – B qui est le *ground truth*. On remarque rapidement que la version reconstruire qui découle de notre réseau est très près de celle attendue.

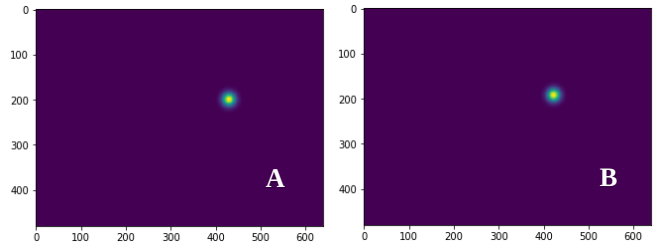

Figure 13 : Sortie reconstruite selon la moyenne pondérée des intensités de l'image obtenu avec la LLE (A) et image attendue (B)

En bref, bien que notre algorithme ne permette pas de reconstruire directement l'image en raison de la fonction de perte choisie, un sujet qui sera plus abordé dans notre discussion critique, nous avons tout de même réussi à reconstruire les images originales avec une distance inférieure de 15 pixels pour l'ensemble de test complet.

4 Discussion critique

Le but de ce projet était de se familiariser avec les réseaux complètement convolutifs et avec les réseaux RNN, particulièrement les réseaux utilisant des cellules LSTM. Vue notre intérêt envers l'imagerie médicale, le concours CLUST15 était un défi idéal pour appliquer les notions apprises dans le cadre du cours INF8225 au domaine biomédical. L'article de (Rangamani et al. 2016) présentait une manière concrète d'appliquer les CNN et le réseau LSTM à ce défi. Pour cette raison nous avons décidé de répliquer leur étude.

Au terme de ce projet nous pouvons jeter un regard critique sur notre méthodologie et ce que nous avons accompli. Nous allons faire un bref retour sur nos résultats et discuterons de ce qu'on aurait pu faire différemment afin de potentiellement améliorer l'issue de notre projet.

Comme présenté à la section 3, notre meilleur réseau combinant le réseau CNN et le réseau LSTM a atteint une perte moyenne de 8.51 sur notre ensemble test composé de 22 images. Puisque la perte est composée en majorité de la LLE, on peut traduire cette perte en une distance de notre prédiction à la cible. Sachant que la résolution des images dans la base de données CIL-1 est de 0.3mm, on peut estimer que notre réseau a une erreur moyenne de 2.5mm. Une erreur 5 fois plus élevée que celle réclamée par (Rangamani et al. 2016). De plus, la figure 12 montre que nous n'avons pas réussi à répliquer la sortie de leur réseau qui est supposée ressembler à la figure 13B. Il est donc difficile de dire que nous avons réussi à répliquer les résultats de cet article bien que nous réussissions à trouver la position de la cible par le biais de la moyenne pondérée sur les intensités.

Comment expliquer une telle disparité dans les résultats? Selon nous, 2 facteurs importants ont mener à notre incapacité à obtenir des résultats similaires. Le problème se résume au manque de détail dans la conception du réseau CNN. Les auteurs n'ont pas spécifié 1) Le nombre de *feature maps* utilisés

après chaque couche convolutive et 2) La fonction de perte utilisée lors de l'entraînement CNN. Entre ces 2 facteurs, la fonction de perte est probablement celle qui nous a empêché d'obtenir une distribution gaussienne à la sortie du CNN car on ne savait pas comment bien indiquer au réseau s'il performait bien ou non. Bien que la LLE lui a permis de bien identifier l'emplacement, nous ne croyons pas que le réseau a vraiment appris à détecter la structure du vaisseau sanguin. Le réseau a plutôt appris à trouver la région où ce vaisseau se trouve. Nous aurions préféré utiliser directement une fonction de perte qui permettait d'identifier le maximum d'intensité dans notre image, c'est-à-dire le centre du vaisseau d'intérêt ou encore là où la gaussienne est à sa valeur maximale de 1. Toutefois, une simple fonction « max () » n'est pas différentiable ce qui par conséquent ne nous permettait pas d'effectuer la rétropropagation du gradient. Ceci nous a donc contraint à utiliser la LLE qui ne donnait pas les résultats visuellement escomptés. De plus, plus d'information quant au nombre de *feature maps* à utiliser aurait probablement permis au réseau de vraiment capturer la forme du vaisseau. Comme mentionné plus haut, les limitations en puissance de calcul nous ont également empêché de vraiment explorer l'avenue d'augmentation du nombre de *feature maps*.

En rétrospective, on s'aperçoit que même si un article montre des résultats prometteurs et une méthodologie claire, celle-ci n'est pas toujours complète, surtout si on consulte un article de conférence qui n'a pas nécessairement passé par le processus de revue par les pairs. Nous avons également réalisé l'importance des études de réplication pour valider certaines méthodes qui sont publiées dans le domaine grandissant de l'IA. Si on avait à refaire ce projet, on se fierait moins sur un seul article d'application mais plutôt sur une collection d'articles présentant différentes approches pour résoudre la problématique du concours CLUST15.

5 Conclusion

En conclusion, nous avons réussi à concevoir un modèle de détection de cible avec une architecture CNN-LSTM basée sur l'article de (Rangamani et al. 2015). Toutefois, nous n'avons pas réussi à reproduire les images de sortie de leur réseau et nous avons obtenu une erreur moyenne de 2.5mm comparativement à leur erreur moyenne de 0.51mm obtenue sur la même base de données CIL-01. Malgré ces manquements, nous avons exploré les effets de différents paramètres d'apprentissage lors de l'entraînement d'un réseau complètement convolutif et un réseau LSTM. Étant deux réseaux importants dans l'application de l'intelligence artificielle dans le domaine médical, nous sommes satisfaits de nos expériences et nos apprentissages.

Remerciements

Nous voulons remercier le professeur Samuel Kadoury PhD pour l'idée initiale du projet et l'accès à la base de données du concours CLUST15.

Lien Github

https://github.com/talmezh/INF8225

Références

De Luca, V., Benz, T., Kondo, S., König, L., Lübke, D., Rothlübbers, S., Somphone, O., Allaire, S., Bell, M.L., Chung, D.Y.F. and Cifor, A., The 2014 liver ultrasound tracking benchmark. Physics in medicine and biology, 60(14), p. 5571 (2015)

Nouri, D. and Rothberg, A., Liver Ultrasound Tracking using a Learned Distance Metric. In Proc. MICCAI workshop: Challenge on Liver Ultrasound Tracking, pp. 5-12 (2015)

Park S., Kwak N. Analysis on the Dropout Effect in Convolutional Neural Networks. In: Lai SH., Lepetit V., Nishino K., Sato Y. (eds) Computer Vision – ACCV 2016. ACCV 2016. Lecture Notes in Computer Science, vol 10112. Springer, Cham (2017)

Rangamani, A., Xiong, T., Nair, A., D Tran, T., Chin, S. Landmark Detection and Tracking in Ultrasound using a CNN-RNN Framework. (2016).

Xu, Z., Huo, Y., Park, J., Landman, B., Milkowski, A., Grbic, S., Zhou, S., Less is More: Simultaneous View Classification and Landmark Detection for Abdominal Ultrasound Images. arXiv e-prints, arXiv:1805.10376v2 (2018)