Rapport de stage 1A

Alexis Houssard

Année 2018-2019

Contents

1 Informations relatives au stage

Titre du stage :

Suivi de la langue dans une séquence d'images IRM par apprentissage profond

Établissement d'acceuil :

Stage CNRS au sein du **LORIA**
(Laboratoire Lorrain de Recherche en Informatique et ses Applications)

Encadrant :

Yves Laprie

Dates de début et fin de stage :

Du 04 Juillet 2019 au 28 août 2019

2 Préambule

2.1 Résumé et mots-clés

2.1.1 Mots-clés:

Machine Learning, Sequence of images, Convolutional neural networks, Object tracking

2.2 Remerciements

Je tenais, par le biais de cette section, à remercier tout particulièrement le Centre National de la Recherche Scientifique pour la proposition de stage qui m'ait été faite et surtout Yves Laprie pour m'avoir contacté et avoir fait preuve d'une grande bienveillance. Mes remerciements vont également aux membres de l'équipe MultiSpeech qui ont su m'intégrer très rapidement dont Nicolas F et Nicolas grâce auxquels j'ai pu profiter d'une aide précieuse et dans la plus grande des sympathies. C'est sans doute grâce à eux que j'ai découvert le plus dans le domaine du machine learning et même au-delà. Je ne pourrais oublier Michel O qui m'a également beaucoup apporté, tant en matière de fonctions python qu'en matière de culture mexicaine. Enfin je remercie le LORIA pour son acceuil et ses services qui ont rendu ce stage le plus agréable possible.

3 Introduction

L'imagerie médicale et ce qu'elle peut nous apprendre occupent bon nombre de chercheurs provenant d'une multitude de domaines différents. Le projet qui va être décrit dans ce document s'inscrit dans une approche physique de la synthèse de la parole qui repose essentiellement sur l'exploitation d'images médicale du conduit vocal. Dans notre cas, nous traiterons des images acquises par résonnance magnétique dynamique mais les travaux pourront être généralisés à d'autres types de données visuelles. L'objectif primaire du projet est de pouvoir synthétiser la parole à partir de l'étude du conduit vocal et plus précisément de sa forme géométrique. Pour cela, il est nécessaire de réaliser du détourage d'images afin de distinguer avec précision la surface des différents articulateurs. Les données dont nous disposons sont des séquences de 4500 images issues d'un film correspondant à une heure de parole dont le nombre total d'images s'élève à 200 000. Etant donné le mouvement constant des articulateurs, l'estimation des formes doit être effectuée à une fréquence d'environ 50 Hz.

Cela nous amène donc à la principale motivation de ce stage, à savoir de réaliser le détourage de la langue à partir de données IRM. Toutefois, le nombre d'images à étudier étant conséquent, il est inenvisageable de labelliser toutes les images manuellement, c'est pourquoi notre but est d'approcher le problème de manière automatisée en exploitant l'apprentissage automatique grâce aux réseaux de neurones.

Dans ce rapport, nous nous proposerons d'abord d'exposer certains éléments théoriques, qui pourront éclairer certains points du projet réalisé, à travers une approche simpliste des réseaux de neurones à convolutions. Par la suite, il s'agira de présenter le déroulement des travaux à travers les méthodes utilisées ainsi que les choix qui ont été faits. Nous terminerons notre développement par une description des résultats obtenus et une analyse des améliorations qui pourraient être apportées.

4 Environnement

Le stage s'est déroulé au bâtiment du LORIA (Laboratoire Lorrain de la Recherche en Informatique et ses Applications) sur le campus de la Faculté des Sciences de Villers-lès-Nancy. Je fus amené à travailler au sein d'une équipe de recherche spécialisée dans le traitement automatique des langues et des connaissances et plus précisément chargée d'analyse, de perception et de reconnaissance automatique de la parole. J'ai de ce fait rencontré des chercheurs, des ingénieurs ainsi que des doctorants provenant d'une multitude de pays différents, ce qui fut très enrichissant d'un point de vue culturel. Par ailleurs, cela m'a permis de découvrir l'environnement du laboratoire de recherche informatique ainsi que le cadre et le rythme de travail d'un chercheur.

5 Problématique

Donner les objectifs: faire de l'inférence (apprentissage supervisé)

6 Outils et Méthodes

6.1 Première approche: Détourage de la langue sur Python

La première étape du projet consistait à explorer diverses méthodes de détourage d'images disponibles sur Python. Bien qu'aucune de ces méthodes directes n'était destinée à être utilisée par l'équipe Multispeech, cela permet de comprendre l'intérêt porté aux réseaux de neurones.

6.1.1 Approche naïve

J'ai dabord essayé d'implémenter mon propre algorithme de détourage en m'inspirant d'une méthode classique par seuil de norme. En d'autres termes, on s'impose un seuil et une norme qui va nous servir à évaluer le contraste au voisinage de chaque pixel de l'image traitée. A chaque étape du parcours de l'image, on calcule la norme grâce aux coordonnées des quatre pixels qui entourent le pixel courant. Ainsi, si la norme calculée dépasse le seuil choisi on considère que le pixel courant se situe sur une bordure de l'image, l'agorithme dessine alors un pixel noir sur l'image de sortie.

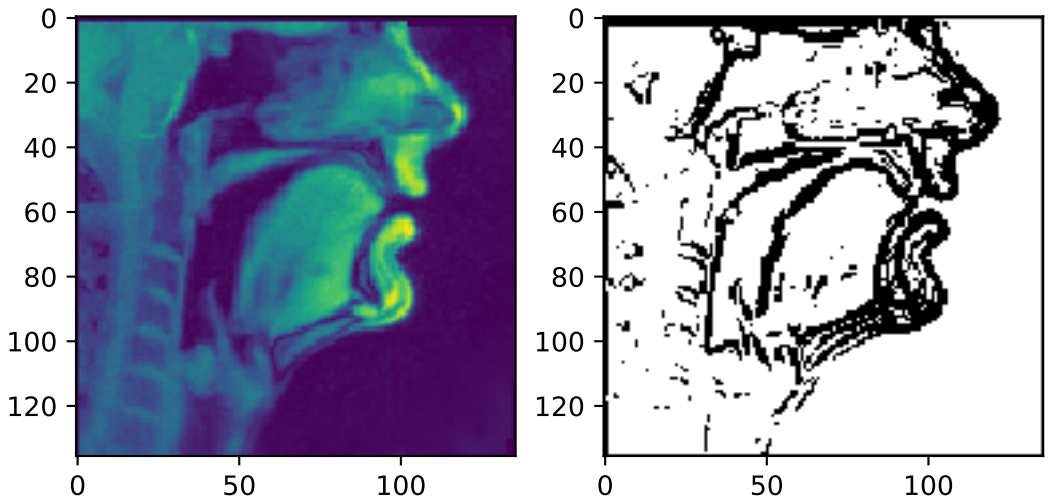


Figure 1

Observations: Bien que rudimentaire, la méthode implémentée dessine des contours qui semblent être les bons. Néanmoins, remarquons que ceux-ci ont une épaisseur de plusieurs pixels, ce qui implique de grosses imprécisions sur la forme des tissus. De plus, cela ne nous permet pas de restreindre le dessin des contours à celui de la langue uniquement.

6.1.2 Première utilisation d'OpenCV

Nous nous proposons à présent de travailler avec des outils d'extraction de contours existant sous Python. La plus simple des méthodes proposées par l'extension OpenCV fournit un résultat(ci-dessous) a priori satisfaisant pour la première image dont on dispose.

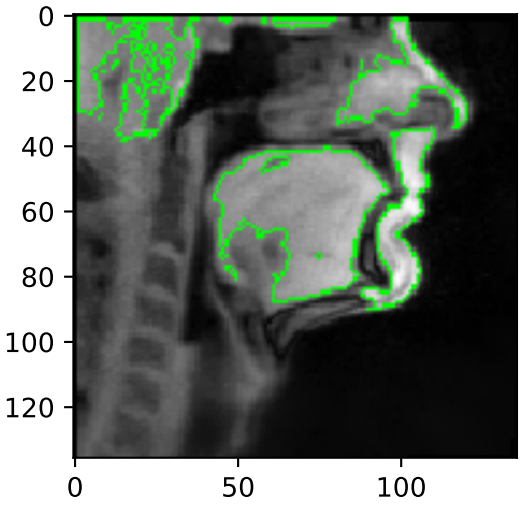


Figure 2

Le gros avantage de cette méthode est qu'elle permet aisément de superposer notre image avec un contour individuel (voir ci-dessous), celui de la langue par exemple.

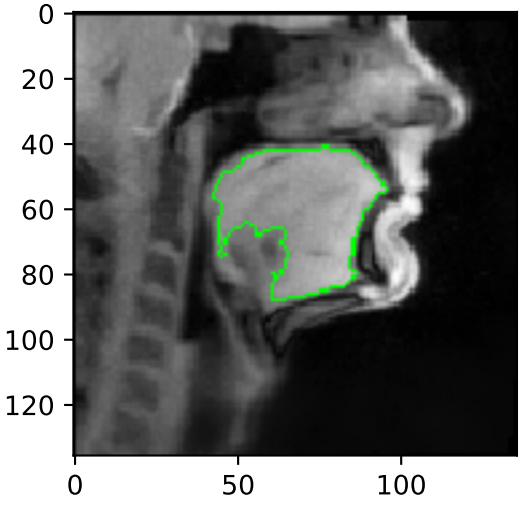


Figure 3

On voit néanmoins que la détection du contour de la langue n'est pas aussi satisfaisant sur toutes les images IRM disponibles. Par exemple, le dessin du contour recherché sur l'image 500 donne le résultat non exploitable ci-dessous. En effet, le dessin du contour est perturbé par la présence régulière d'ombres sur les images IRM. On déduit de cela que la précision de cette méthode est parfois trop fine pour détourer au mieux la langue, compte tenu de la qualité de l'imagerie IRM.

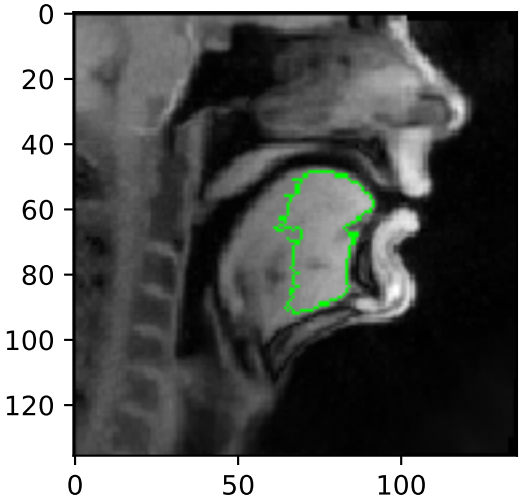


Figure 4

6.1.3 Utilisation du filtre de Canny

Il existe également un algorithme dit de Canny déjà implémenté dans le module OpenCV qui est en théorie plus performant, mais ici aussi il n'est pas possible d'extraire unqiuement le contour de la langue car la détection est faite automatiquement sur l'ensemble de l'image.(Voir le résultat ci-dessous)

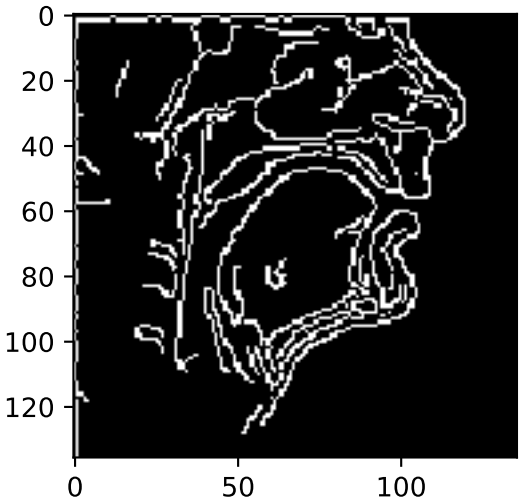


Figure 5

Tout cela nous permet de comprendre la complexité des méthodes de détourage d'images ainsi que les nombreux défauts qu'elles comportent. L'objectif du projet est donc d'éviter le détourage manuel ou algorithmique en utilisant la puissance des réseaux de neurones afin de faire apprendre spécifiquement au réseau la détection du contour de la langue à partir d'une base d'images préalablement détourées à la main.

6.2 Les réseaux de neurones

L'apprentissage automatique est en plein essor et offre de nombreuses applications dans une multitude de domaines et particulièrement dans le traitement des images. Parmi les méthodes d'apprentissage profond on dégage notamment les réseaux de neurones concolutionnels ou CNN (de l'anglais Convolutional Neural Network) très utilisés pour la reconnaissance de formes, la classification d'images ou la détection de bordures. Ils reposent sur des structures de neurones organisés en couches, appellées couches convolutionnelles, qui sont en fait des matrices de poids et qui jouent le rôle de filtres. Selon la matrice appliquée à notre image (qui est également une matrice dans le cas d'images en niveaux de gris), l'image résultante contiendra certaines carctéristiques de l'image de départ. A titre d'exemple, un filtre peut ne conserver que les bordures verticales tandis qu'un autre retiendra les bordures horizontales de la même image d'entrée.

Pour des raisons de simplicité, l'objectif est tout d'abord de réaliser ce que l'on appelle du **Transfert learning**. Concrètement, il s'agit d'exploiter un réseau de neurones déjà entraîné sur un échantillon d'images à des fins spécifiques et de l'entraîner de nouveau en ajoutant nos propres données afin de détourner la fonction initiale du CNN à notre avantage. Pour faire simple, il est possible d'utiliser un réseau de neurones entraîné à reconnaître la forme d'une main et de le réentraîner dans le but de reconnaître la forme de la langue. Dans la pratique, les poids du réseau résultant du préentraînement sont conservés dans des fichiers sous forme d'Array par exemple et sont ensuite réutilisés lors de l'étape de transfert learning comme points de départ, tandis que la méthode classique est d'initialiser les poids du réseau aléatoirement. Les résultats de cette méthode sont variables selon l'application et le réseau choisi mais restent généralement bons pour des applications similaires, voire parfois meilleurs lorsque que peu de données sont disponibles (ce qui est notre cas puisque nous n'avons que 272 images dont on "connait" le masque). Voir le papier: Convolutional Neural Networks for Medical ImageAnalysis: Full Training or Fine Tuning? .

Je propose en premier lieu de revenir sur la nature d'un réseau de neurones et son fonctionnement, ou du moins sur ce que j'ai pu comprendre et retenir, car j'ai moi même dû avant toute chose apprendre les fondements du machine learning grâce aux membres de l'équipe et de nombreux sites et articles.

7 Choix

7.1 Architecture du réseau: le réseau U-Net

Le fonctionnement d'un réseau de neurones repose donc essentiellement sur son architecture, à savoir la manière dont on dispose les différentes couches de neurones. Le choix de cette structure est crucial puisque les résultats du modèle dépendent de la nature des couches et de l'ordre selon lequel elles se succèdent. Savoir organiser un réseau de neurones convolutionnel n'est pas chose aisée, c'est pourquoi de nombreux modèles éprouvés sont librement disponibles sur le net. Parmi eux, le réseau U-Net et toutes ses dérivées suivent une architecture précise (décrite sur la Figure 2) dont l'apprentissage est spécialisé dans le domaine de l'imagerie médicale (images IRM ou à Ultrasons).

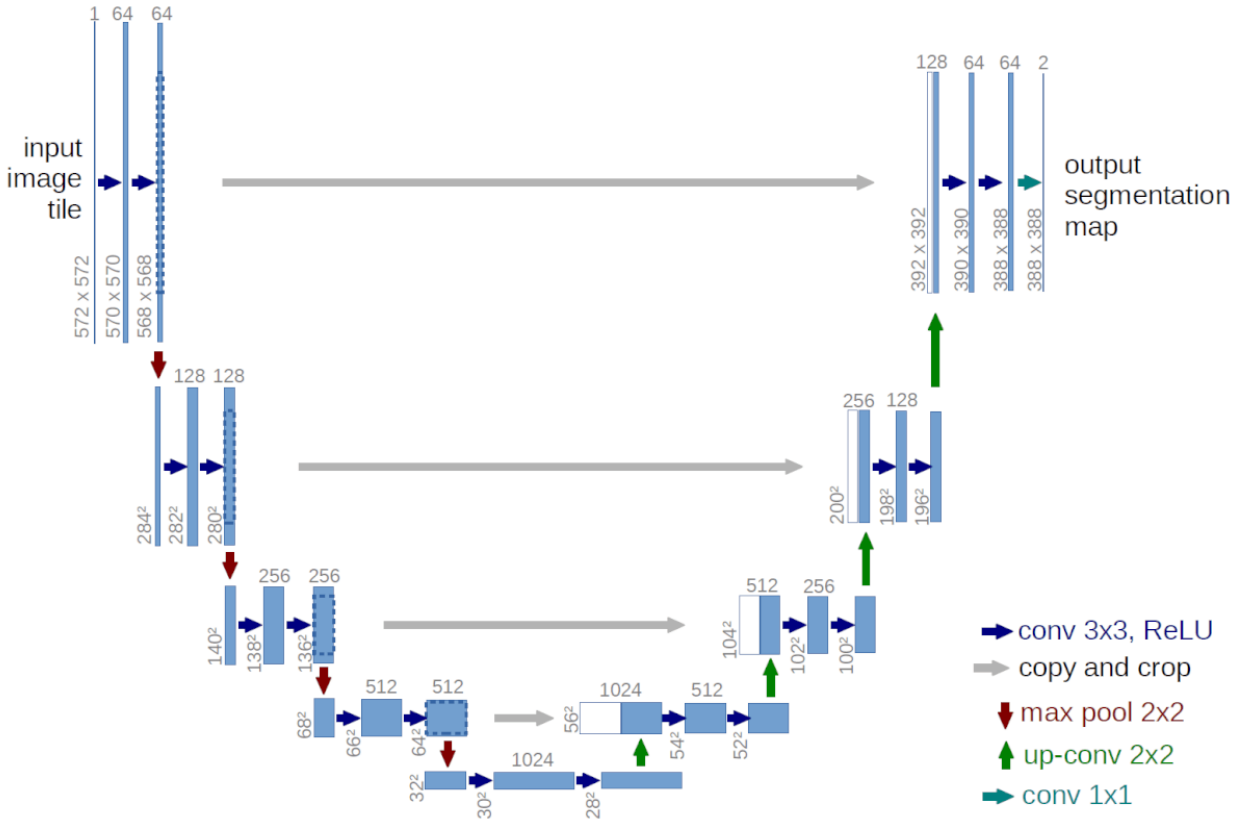


Figure 6

Le réseau U-Net possède une structure d'autoencodeur constituée de deux parties distinctes que sont l'encodeur et le décodeur. On distingue sur la première moitié de la figure 2 l'encodeur qui consite en la contraction de la taille de l'image et en l'augmentation du nombre de canaux (c'est à dire le nombre de matrices). Lors de cette étape, on réduit les informations spatiales tout en extrayant de plus en plus de caractéristiques (features en anglais). Le décodeur, quant à lui, constitue la partie expansive du réseau durant laquelle l'image de sortie est reconsituée grâce à une succession de "blocs d'expansion" symétrique à la structure de la partie contractante qui confère au réseau cette forme de U dont il tire son nom.

Sur le réseau U-Net classique présenté sur cette dernière figure, les dimensions d'entrée (572x572) ne sont pas choisies au hasard et suivent certaines règles imposées par l'architecture du réseau mais nous verrons plus tard comment s'acquitter de certaines de ces contraintes pour traiter nos images.

On se propose tout d'abord d'expliquer brièvement le principe des étapes de convolution. La convolution est un calcul matriciel effectué à partir d'une image et d'un filtre. Il consiste en le calcul d'une nouvelle matrice

dont chaque élément résulte d'une combinaison linéaire. Pour chacun de ces éléments, on sélectionne une région de l'image d'entrée dont la taille est la même que celle du filtre, il s'agit la plupart du temps d'un carré. Ensuite on multiplie chaque élément de la sélection avec l'élément correspondant du filtre, c'est à dire l'élément qui possède les mêmes indices. Enfin on somme toutes les multiplications effectuées pour obtenir l'élément de la matrice de sortie.

Autrement dit, si l'on dispose d'une matrice $(a_{i,j})$ et d'un filtre $(b_{i,j})$ de taille 3x3, on sélectionne les 3x3 premiers pixels de l'image et le pixel p qui en résulte vaut:

$$p = \sum_{i=0}^{2} \sum_{j=0}^{2} a_{i,j} * b_{i,j}$$

. Pour calculer le prochain pixel de l'image de sortie, on va décaler la sélection sur l'image d'entrée d'un certain nombre de pixels, ce nombre est appelé le **stride**. On se doute alors assez aisément que si le stride est mal choisi, la sélection dépassera de l'image. On introduit alors ce que l'on nomme le **padding** et qui n'est en fait que le nombre de zéros que l'on ajoute autour de l'image d'entrée et qui permet de dépasser de celle-ci pour appliquer le filtre. On se propose de schématiser chaque étape afin d'éclaircir au maximum le propos.

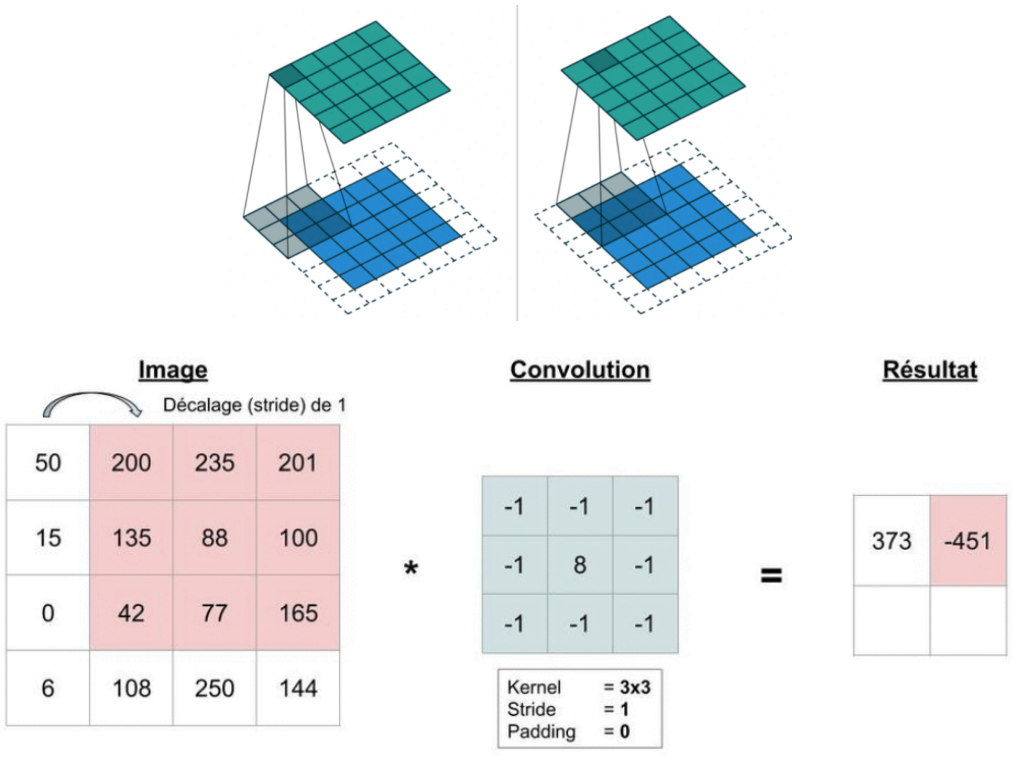


Figure 7: Convolution

Vient ensuite la phase expansive lors de laquelle interviennent deux étapes cruciales que nous tenterons de décrire. Premièrement, les étapes de **déconvolution** (*transposed convolution* en anglais) sont à l'origine de l'augmentation de la taille des images en cours de traitement. A l'instar d'une convolution classique, la déconvolution s'effectue à partir d'un filtre, d'un stride et d'un padding mais la taille de l'image de sortie est plus importante que celle de l'image d'entrée. Ceci s'explique par le choix d'un padding plus important qui induit un plus grand nombre d'étapes de calcul et donc plus d'informations à la sortie de la déconvolution. On explicitera dans la section suivante la formule mathématique à l'origine de ce phénomène.

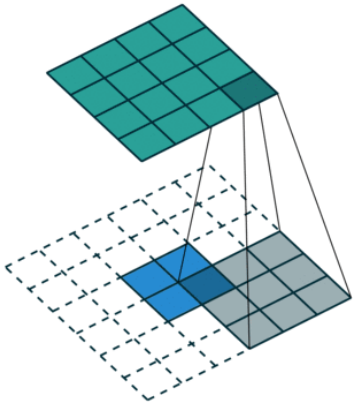


Figure 8: Deconvolution

Deuxièmement, chaque étape de déconvolution est accompagnée par une étape de **concaténation** (dite de *copy and crop* sur la figure 2) qui joue un rôle non négligeable dans la précision du résultat. Une étape de concaténation consiste à conserver la moitié des pixels de l'image de sortie d'un bloc d'expansion et de la concaténer avec l'image obtenue en sortie du bloc symétrique de la phase contractante. Le but de cette opération est la reconstruction d'une image de taille augmentée qui conserve les informations extraites lors de la phase de contraction. En effet le fait d'ajouter un certain nombre de zéros durant l'"up-sampling" peut entraîner une perte des informations issues de l'encodeur. L'avantage de l'organisation du réseau est que la partie contractante renseigne la nature de l'information à extraire tandis que la partie expansive renseigne la localisation dans l'image de ces informations.

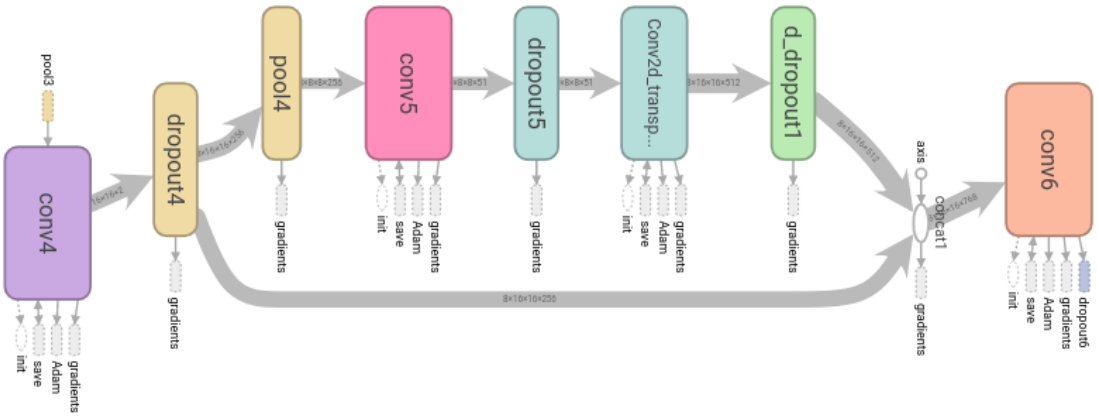


Figure 9: Concatenation

Sur ce graphique généré à partir de l'outil Tensorboard, on peut observer la première étape de concaténation réalisée au cours de l'inférence. La couche de convolution numérotée 4 prend une entrée des données de dimensions 8x16x16x128, ce qui signifie que le batch size vaut 8 et qu'on dispose à cet instant de 128 images de taille 16x16, et renvoit en sortie des données de dimensions 8x16x16x256. Viennent ensuite plusieurs étapes intermédiaires puis, avant de rentrer dans le bloc de déconvolution symmétrique, les données ont une dimension de 8x16x16x512. Intervient alors la concaténation entre la copie des données en sortie de la convolution 4 et la moitié des données en sortie de de la déconvolution nommée Conv2dTranspose sur la figure ci-dessus. Ainsi, les données reconstruites en entrée de la convolution 6 sont de dimension 8x16x16x512 et sont composées de 256 images issues de l'encodeur et de 256 images issues du décodeur.

Voir la vidéo de l'Université de fribourg pour une explication schématique en anglais: https://lmb.informatik.uni-freiburg.de/people/ronneber/u-net/.

7.2 Traitement des données d'entrée

7.2.1 Taille de l'image

Le réseau U-net est initialement conçu pour une entrée de taille 512. Le choix de la dimension de l'entrée n'est pas arbitraire puisqu'elle dépend des couches présentes dans l'encodeur. Dans le cas traité, les images vont subir quatre étapes de **max-pooling** de taille 2x2 (flèches rouges sur la figure 2). Qu'est ce que le max-pooling? C'est une étape de réduction des images qui consiste à extraire les informations d'une région et d'en conserver le maximum. Le principe est de balayer les pixels de l'image quatre par quatre grâce à un carré de taille 2x2 (appelé kernel ou noyau en francais) et de reconstruire une image dont la taille est la moitié de celle d'entrée selon le schéma ci-dessous.

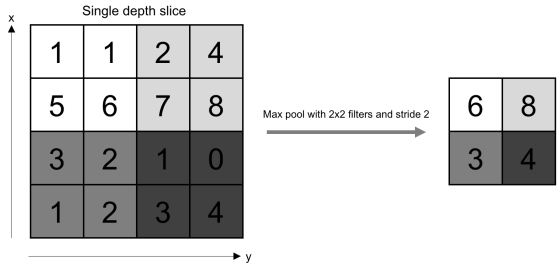


Figure 10

C'est pour cette raison que les images doivent être divisibles par 2 à chaque nouvelle étape de pooling. En ce qui nous concerne, les IRM traitées par le réseau doivent impérativement être de dimension divisible par $2^4 = 16$. Etant donné que les images sont de taille 136x136, nous les réduirons à une taille de 128x128.

S'ajoute à cela les nombreux redimensionnements dus aux étapes de convolution. En effet, si on porte attention aux dimensions des images annotées sur la figure 2, la convolution réduit chaque côté de l'image de 2 pixels. En réalité cette réduction dépend des paramètres de la convolution que sont la taille du filtre, le stride et le padding.

Il est alors possible de calculer la taille de l'image de sortie grâce à l'expression:

$$O = \frac{I - K + 2.P}{S} + 1$$

où O est la dimension de la sortie, I celle de l'entrée, K celle du noyau et où P et S représentent le padding et le stride.

Pour ne pas ajouter de contraintes sur la taille de l'image que l'on doit fournir en entrée du réseau, on peut choisir les paramètres des convolutions de sorte qu'il n'y ait pas de perte sur la dimension des images dans les blocs de convolution. Par exemple, nous travaillerons dans la suite avec un noyau de taille 3x3 ainsi qu'un padding et un stride de 1. Pour une image de taille 128x128, on obtient une image de sortie de dimension: $O = \frac{128-3+2*1}{1} + 1 = 128$

Pour finir, le modèle étant compatible avec un certain nombre de tailles d'image, il n'en reste pas moins que ce sont les pixels qui contiennent l'information et nous portons donc peu d'intérêt à réduire davantage la taille de nos images, bien au contraire.

7.2.2 Format des masques

Notre objectif étant d'obtenir, à partir du contour de la langue, un nombre de points suffisant pour déterminer la forme géométrique du conduit vocal, nous avons procéder dans un premier temps à l'entraînement du modèle avec de simples points situés sur la langue. Le but est de comparer les résultats du CNN avec les différents formats d'entrée que sont les contours définis sous la forme d'une suite de points, les contours

filiformes et les régions sachant que le réseau a été pensé pour la segmentation d'images et particulièrement l'extraction de régions.

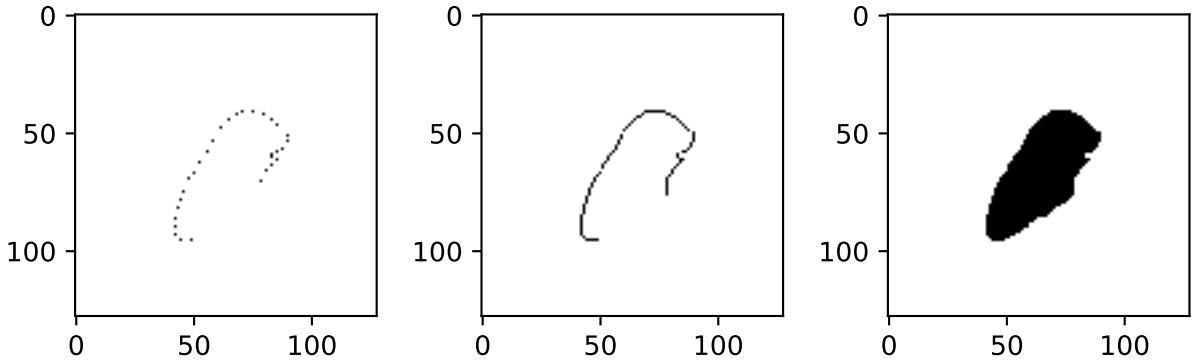


Figure 11

Le réseau même sur-entraîné ne semble pas parvenir à reconstituer des contours corrects à partir de masques définis comme suite de points, voici ce que l'on obtient en pratique:

Figure 12

Comme les contours importés en entrée sont discrets et qu'il provienne d'un détourage manuel approximatif, l'apprentissage du réseau sur l'entière partie du lot d'entraînement n'en est que plus complexe et il faudrait un certain nombre d'époques pour reconstruire des masques valides. Il n'en resterait pas moins que le modèle ne pourrait s'adapter à de nouvelles images, c'est pourquoi on décide de ne pas travailler avec ce type de masque.

Enfin, bien que le réseau soit conçu pour de l'extraction de régions, seule le bord de la langue nous intéresse et il est donc plus raisonnable de spécialiser le réseau de neurones à détecter un contour filiforme. Tous les résultats présentés par la suite seront donc générés à partir de ce second type de masque.

Avant d'entraîner le réseau, tout un prétraitement des images est réalisé afin de les redimensionner ou encore de restreindre les pixels à des valeurs comprises entre 0 et 1, mais plus encore, les images subissent dans le code original une rotation aléatoire. En d'autres termes, un vecteur de deux nombres compris entre 0 et 1 est généré aléatoirement pour chaque image traitée. Ces deux nombres sont alors comparés à 0.5 et selon le résultat logique de la comparaison, l'image en question subit ou non une inversion. Le premier nombre décide de l'inversion horizontale et le second de l'inversion verticale. On retrouve ainsi quatre types d'images lors de l'entraînemnet du modèle. Etant donné la taille du lot d'apprentissage, certains entraînement conduisent à des masques retournés ou tournés dans le sens opposé à celui de l'image d'origine, sûrement parce que le

réseau a appris un plus grand nombre d'images tournés dans ce même sens que dans les autres configurations. On peut donc observer le type de résultats ci-dessous calculés à différentes étapes de l'apprentissage durant lequel le masque peut subir une rotation:

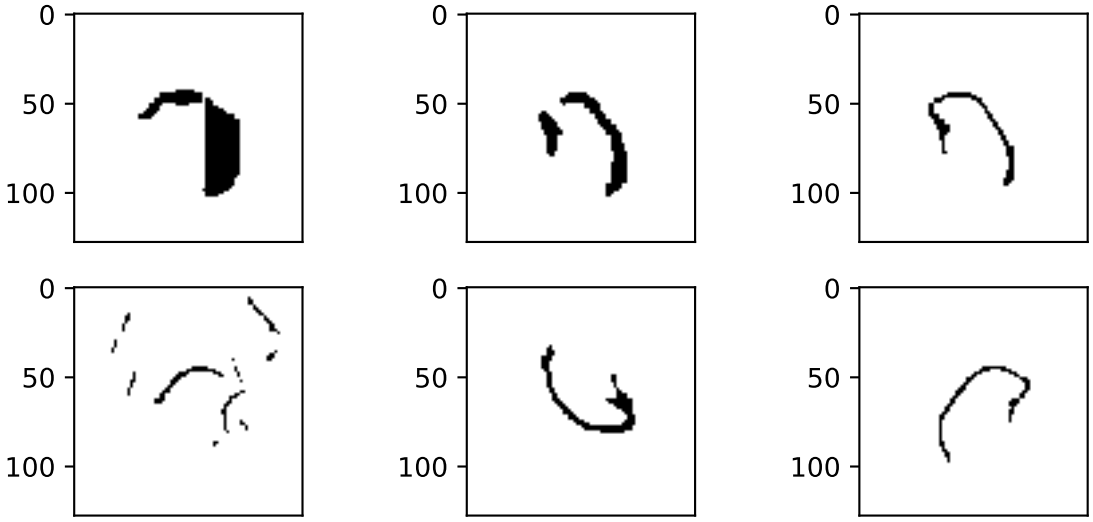


Figure 13

Sachant que ces modifications surviennent de manière aléatoire, l'aspect des masques de sortie du réseau est imprévisible et varie à chaque nouvel apprentissage. Ces observations m'ont donc permis de me focaliser davantage sur les codes de prétraitement des images et de finalement décider d'abandonner les inversions, considérant qu'elles n'apportent aucune utilité au projet. Ceci reste anecdotique, mais il est bon de remarquer que nos images IRM représentent toutes une personne dirigeant son regard vers la droite, alors qu'il est coutume dans le domaine médical de travailler avec des images orientées dans l'autre sens. Le choix d'écarter les inversions du traitement des images s'explique par le fait que l'on cherche avant tout à avoir un réseau avec une précision optimale pour notre type d'images et qu'une fois les résultats satisfaisant, il serait toujours possible d'inverser les images pour les mettre dans l'orientation reconnue par le CNN. Cette idée se révéla particulièrement utile à la progression du projet puisque les résultats en était bien meilleurs et surtout prévisibles.

7.3 Paramètres d'apprentissage

7.3.1 Nombre d'époques

Une époque est un cycle complet d'apprentissage du modèle sur l'ensemble des données mises à disposition. Le nombre d'époques est donc le nombre de fois que le réseau va balayer l'ensemble des images de training. Le choix de ce paramètre est très important car il détermine la précision des résultats.

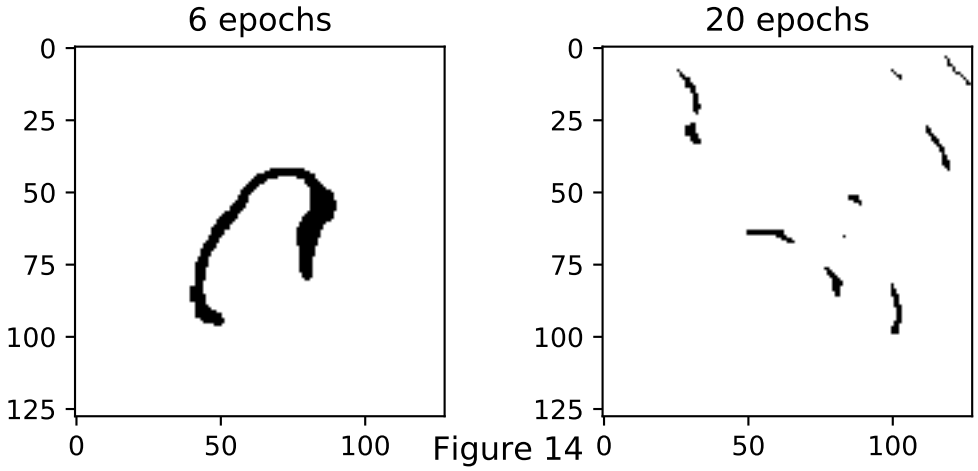


Figure 14

Il faut savoir, et on le remarque sur nos échantillons d'images, que plus l'apprentissage du modèle est long, plus celui-ci va se spécialiser. Ici, cela se remarque sur l'objet représenté sur l'image de sortie. En clair, si l'apprentissage du modèle est trop long, celui ci va apprendre à distinguer de fins détails qui ne nous intéressent nullement et va délaisser les objets de premier plan tels que la langue.

Ce phénomène de surentraînement (ou outfitting en anglais) est en fait très courant en Deep Learning et constitue le premier grand risque d'erreur auquel nous sommes exposés ici. En effet, si le nombre de paramètres d'apprentissage domine largement le nombre de paramètres à apprendre, le réseau va être capable de retenir tous les détails des données sur lesquelles il s'est entraîné et sera donc incapable de reconnaitre quoi que ce soit sur une image nouvelle si celle ci diffère légèrement des images apprises. On perd en fait en capacité d'adaptation et cela peut très vite conduire à un réseau inefficient.

Le choix concernant le nombre d'itérations a d'abord été fait par tâtonnement puisque ce paramètre dépend grandement du modèle, d'un apprentissage et des données. C'est pourquoi le grand défaut de notre CNN est le fait que le nombre d'époques optimal va changer à chaque nouvel entraînement. On ne sait donc pas à l'avance comment réaliser au mieux notre apprentissage.

Une méthode courante pour palier cette incertitude est le fait d'arrêter l'entraînement de manière automatique lorsque la précision du modèle sur l'échantillon de test est la meilleure, c'est ce que l'on appelle l'**early stopping**. Cela repose sur le comportement des fonctions de perte calculées à partir des lots d'entraînement et de test. Tandis que la première est sensée décroître tout au long de l'entraînement, la seconde a tendance à être localement convexe de sorte qu'elle décroît au début de l'entraînement pour atteindre un minimum local et augmenter par la suite. L'objectif de la méthode est donc de sauvegarder l'état du modèle à une fréquence donnée et d'arrêter l'entraînement lorsque la précision n'augmente plus depuis un certain nombre d'étapes. Les prédictions seront alors faites à partir de la sauvegarde aboutissant à la meilleure précision. On peut représenter le procédé simplement comme ci dessous (ce schéma est emprunté à [1] mais illustre clairement le propos).

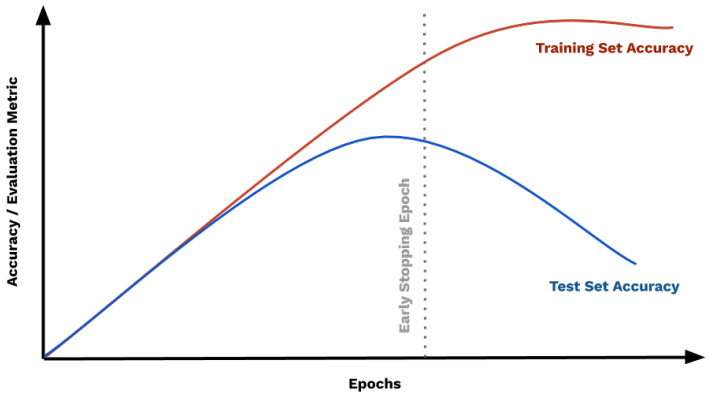


Figure 15

L'implémentation de l'arrêt prématuré sous Tensorflow nécessite certains prérequis dont je ne dispose pas, contrairement à Keras ou Pytorch. N'ayant pas assez de temps pour transposer le code sous un autre module, j'ai choisi de procéder de manière simple afin de trouver le meilleur nombre d'itérations à effectuer en terme de précision. J'ai donc implémenté un code de comparaison des images qui diffère légèrement de celui du coefficient de Dice afin de donner une idée du pourcentage de similitude entre un masque attendu et un contour prédit. Cette fonction renvoit le rapport entre le cardinal de l'intersection des deux contours et le cardinal du contour prédit (on considère ici un contour comme un ensemble de pixels) qui est plus important. La méthode est alors de lancer un long entraînement durant lequel on enregistre les checkpoints toutes les 250 itérations et prédit le contour de certaines images pour chacun de ces checkpoints. Une fois l'apprentissage achevé, on conserve dans une matrice les coefficients de similitude calculés pour chacune des images choisies à toutes les étapes de l'entraînement enregistrées. On compare enfin les moyennes de ces coefficients pour chaque checkpoint et on en extrait le maximum afin d'obtenir le nombre d'itérations conduisant à la meilleur précision en moyenne.

Cela va nous permettre de nous donner une idée du nombre d'itérations à effectuer au minimum, mais cela reste à titre indicatif puisque les comparaisons ne sont faites que sur des images labellisées et qui ont donc servis à l'entraînement du modèle. Le résultat obtenu est de 11 750 itérations pour un batch size de 8, bien que visuellement les résultats sont mauvais. Cela est du au fait qu'un long apprentissage réduit le nombre de pixels du contour qui est bien localisé, ainsi le ratio calculé est élevé car on réduit le dénominateur. Cependant le coefficient n'augmente plus tellement à partir de 6000 itérations, on va donc choisr un nombre proche de 6000 selon les observations.

7.3.2 Taille d'un batch

Un batch est un sous lot d'images issu de l'échantillon d'entraînement sur lequel le modèle effectue une itération d'entraînement. La décision concernant la taille du batch va donc déterminer le nombre d'images que le réseau traite simultanément et ainsi impacter la précision de ce dernier. En effet, à chaque itération, le modèle estime l'erreur commise entre les valeurs attendue et obtenue et ajuste les poids du réseau par la méthode du gradient. La batch size est alors la taille des échantillons à partir desquels le réseau renouvelle ses poids.

On comprend que selon la valeur de cette taille, l'ajustement des poids se fait différemment et la convergence de la précision du modèle est plus ou moins rapide. Intuitivement, la vitesse de convergence de la précision du modèle est fonction décroissante de la taille du batch. En réalité, cela est dû au fait que plus on réduit le nombre d'exemples lors de l'étape de correction, plus la variance de la précision va augmenter.

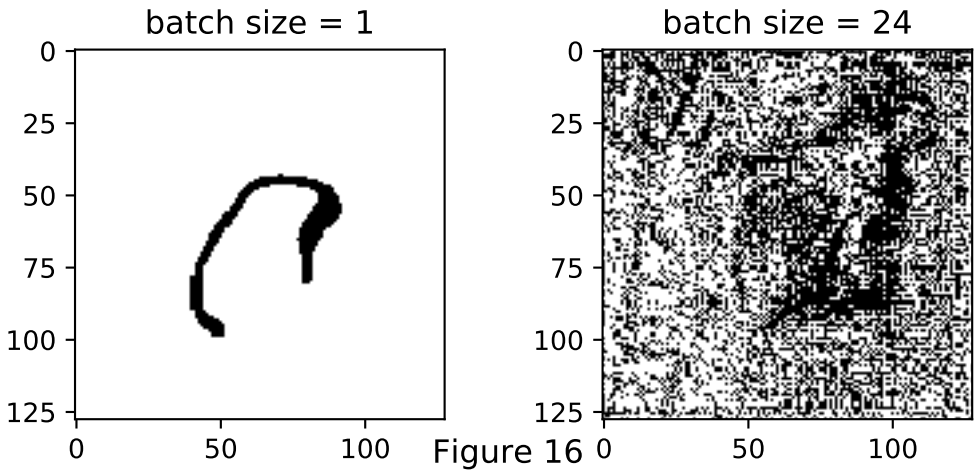
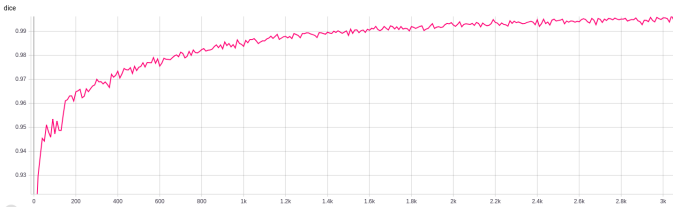
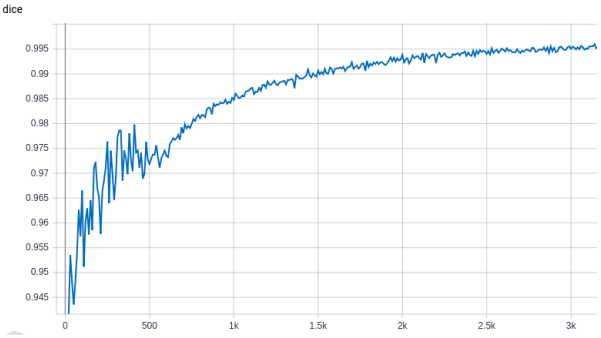


Figure 16

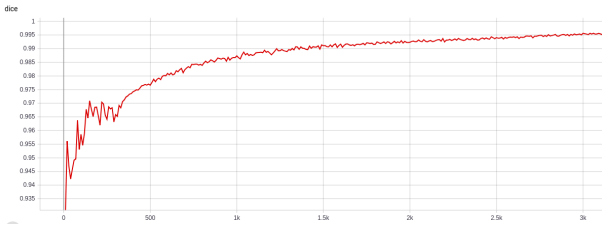
La taille du batch est souvent choisie égale à 24 ou plus, ce qui conduisit à des absurdités lors des premiers essais de notre étude. J'étais donc contraint de choisir un batch size de 1 mais, comme expliqué précèdemment, la variance sur la précision de notre modèle va être conséquente. Cela contribue à l'indécision sur le nombre d'époques à choisir puisque qu'un batch size de 1 (appelé aussi mini-batch) entraîne des oscillations de la précision des prédictions (mesurée grâce au coefficient de Dice que l'on explicitera en 7.3.4) comme on peut le voir ci-après. Ce défaut constitue une raison supplémentaire d'utiliser la méthode d'early stopping.



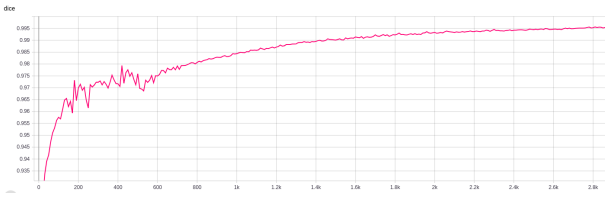
batch size = 1



batch size = 4



batch size = 8



batch size = 24

On peut déduire de ces représentations graphiques la tendance que suit la précision de nos résultats en fonction de la taille de batch choisie pour l'entraînement considéré. La convergence du coefficient de Dice vers son asymptote horizontale semble être plus rapide pour de faibles tailles de batch mais l'entraînement nécessite un plus grand nombre d'itérations pour se stabiliser comme en témoigne les variations du coefficient de Dice dans le cas d'un batch size de 1. Pour visualiser ce phénomène, il convient de se focaliser sur les variations du coefficient de Dice à partir de 500 itérations environ car le début de l'apprentissage n'est jamais le même à cause de l'initialisation aléatoire. De plus, la valeur vers laquelle le coefficient converge augmente avec la taille du batch, même si dans notre exemple les différences restent anecdotiques. Ces observations s'expliquent par le fait que le réseau s'améliore en calculant ses poids à partir d'un plus large échantillon d'images puisque la minimisation de la fonction de perte est plus fine. Bien que la précision soit meilleure en augmentant la valeur du paramètre en question, la généralisation du modèle est toutefois moins bonne que

pour de plus petites tailles de batch. C'est un choix que l'on doit faire en fonction des ressources informatiques dont on dispose et des autres paramètres d'apprentissage. (Voir [2])

Il s'avèra dans la pratique que les contours obtenus en optant pour un batch size de 24 sont nettement plus fins et proches de ce que l'on peut obtenir en réalisant du détourage manuel.

Après un examen plus approfondi du fonctionnemnt de l'algorithme d'entraînement, on comprend d'une part que l'origine de l'instabilité du modèle provient de **l'initialisation de ses poids** et d'autre part que l'inaboutissemnent des entraînements à batch size élevés est dû à la **fréquence d'enregistrement des poids**.

En ce qui concerne l'initialisation des poids avant chaque phase d'entraînement, il est commun de procéder de manière aléatoire en utilisant une distribution des poids normale ou uniforme. Dans notre exemple, les poids sont intialisés suivant une loi uniforme sur $[0, 1]$. Ainsi, selon les itérations de la descente de gradient, l'état final du réseau ne sera pas nécessairemnt le même après deux initialisations différentes, ce qui explique en partie la variabilité des résultats d'une même expérience comme on peut le voir ci-dessus.

Par ailleurs, lors d'un entraînement, l'algorithme réalise un certain nombre d'étapes d'apprentissage qui correspond en réalité au nombre d'itérations de la méthode de descente de gradient effectuées. Ce nombre d'étapes s'exprime de la manière suivante: $steps = \left\lfloor \frac{trainlength}{batchsize} \right\rfloor * epochs$. Mathématiquement, le nombre d'étapes par époque est une fonction décroissante de la taille du batch. Lorsque l'on dispose de peu de données, il peut être problèmatique de choisir une taille de batch importante puisque le nombre d'étapes diminue en conséquence de sorte que l'on doit choisir un plus grand nombre d'époques, ce qui augmente considérablement le temps d'entraînement du modèle. De plus, le code Python responsable de l'apprentissage du réseau est écrit de telle sorte que l'enregistrement des poids suit une fréquence fixée en nombre d'étapes réalisées. Si cette fréquence est trop importante par rapport au nombre total d'étapes alors le réseau va se renouveller peu de fois et risque même de s'entraîner pour rien. Eclaircissons cela avec un exemple concret. Prenons le cas de la figure 5 et les paramètres qui ont conduit à ce genre de résultats qui étaient un nombre d'époques égale à 5, un batch size de 24 et un nombre de données de 272. L'algorithme enregistrait un fichier dit de "checkpoint" comportant les poids du réseau toutes les 1000 étapes d'entraînement. Si on calcule le nombre d'étapes réalisées lors de l'entraînement dans son intégralité, on obtient environ : $\lfloor \frac{272}{24} \rfloor * 5 = 55$ étapes. Le fichier de poids utilisé pour faire la prédiction des masques était alors celui de l'initialisation (issu d'un tirage aléatoire) qui ne s'est jamais mis à jour.

7.3.3 Taille de l'échantillon d'apprentissage

Avant de changer de modèle, il est toujours préférable de revenir à l'origine du problème, à savoir les données que l'on exploite. On a d'abord entraîné notre modèle sur 272 images labelisées, mais sans se préoccuper de la relation entre la complexité du réseau et la taille du training set. On en revient donc au principe de l'overfitting et nous allons à présent vérifier par le calcul si overfitting il y a et ainsi estimer le nombre d'images que l'on doit labelliser.

Afin de se faire une idée du nombre d'images à ajouter dans notre lot de données d'entraînement, il faut reprendre l'architecture de notre réseau et calculer le nombre de paramètres d'apprentissage. Il s'agit en fait du nombre total de variables qui vont évoluer au cours de l'entraînement et qui sont stockés dans l'ensemble de nos filtres (qui sont, pour rappel, des matrices). Le calcul est simple et s'effectue couche par couche en suivant la formule suivante:

$$\sum_{j=1}^{N} (K_j * I_j + 1) * O_j$$

Où N est le nombre de couches du réseau, K_j la taille du noyau de la couche j, I_j le nombre de filtres de l'entrée et O_j le nombre de filtres en sortie. L'ajout d'un 1 supplémentaire provient du fait que chaque nouveau filtre possède un biais.

Enfin, sachant que chaque pixel d'une image en entrée du réseau va compter pour un paramètre, il suffit, pour estimer le nombre d'images à prendre en compte, de diviser le nombre de paramètres d'apprentissage obtenu par la dimension des images.

Couche	Formule	Paramètres
Conv1	((3x3x1+1)x32) + (3x3x32+1)x32	9568
Conv2	((3x3x32+1)x64) + (3x3x64+1)x64	55424
Conv3	((3x3x64+1)x128) + (3x3x128+1)x128	221440
Conv4	((3x3x128+1)x256) + (3x3x256+1)x256	885248
Conv5	((3x3x256+1)x512) + (3x3x512+1)x512	3539968
Deconv1	((2x2x512+1)x512)	1049088
Conv6	((3x3x512+1)x256) + (3x3x256+1)x256	1769984
Deconv2	((2x2x256+1)x256)	262400
Conv7	((3x3x256+1)x128) + (3x3x128+1)x128	442624
Deconv3	((2x2x128+1)x128)	65664
Conv8	((3x3x128+1)x64) + (3x3x64+1)x64	110720
Deconv4	((2x2x64+1)x64)	16448
Conv9	((3x3x64+1)x32) + (3x3x32+1)x32	27712
Conv1x1	((1x1x32+1)x2)	66
Total		8456354

D'après le calcul explicité précèdemment, il faudrait au minimum environ $\frac{8456354}{128*128} = 516$ images pour entraîner au mieux notre modèle. Ce résultat est utile à titre informatif mais ne constitue nullement un nombre à respecter avec précision. Pour de faibles lots de données qui varient peu, l'overfitting se révèle parfois satisfaisant et peut même dépasser en terme de précision un modèle non surentraîné.

Remarque sur la Data Augmentation :

Une méthode classique et simple pour augmenter la taille de notre échantillon sans pour autant avoir plus de données est de procéder à quelques modifications qui ne perturberont pas le réseau en ce sens qu'il n'apprendra pas de nouvelles choses mais il adaptera sa manière de reconnaître ce qui est déjà appris. Concrètement, il s'agit de modifier nos images en leur appliquant une légère rotation ou un filtre de couleur et de les ajouter à l'échantillon d'entraînement. Ici il est malheureusement peu utile de réaliser ce genre d'augmentation de données puisque le support IRM est normalisé. Il pourrait être envisageable de translater légèrement les photos et leurs masques mais on a d'abord jugé cela inintéressant puisqu'il ne sera, a priori, jamais demandé au réseau de reconnaître la forme de la langue sur une image translatée.

Cela nous oblige donc à détourer manuellement de nouvelles images afin d'atteindre la taille optimale du lot d'apprentissage. En s'adonnant à l'ajout de nouvelles images labellisées, nous avons dû importer des données d'une autre séquence d'images provenant du même enregistrement vidéo. Nous avons alors remarqué que selon la séquence d'images, la position du locuteur peut être amenée à changer. Si les résultats obtenus aux tests de détourage de plusieurs images issues des deux séquences exploitées semblent corrects, il reste possible que le réseau conduise à plus d'erreurs si l'on effectue ces tests sur des images issues d'une séquence qui diffère de celles déjà vues. L'idée d'une augmentation de données en réalisant des translations s'impose alors comme une probable nécessité.

7.4 Fonction de perte et coefficient de Dice

La précision du modèle est mesurée de deux manières différentes lors de l'apprentissage. L'une étant nécessaire à la progression de l'entraînement et l'autre est utilisée à titre indicatif. La première est ce que l'on nomme couramment la fonction de perte (*Loss function* en anglais). Son rôle est de mesurer l'erreur moyenne commise par le modèle sur le lot d'entraînement afin d'affiner les résultats, c'est cette fonction qu'il convient de minimiser par la méthode du gradient lors d'une itération de l'entraînement. Quant à la seconde méthode,

il s'agit du calcul d'un indicateur statistique appelé coefficient de Dice qui va nous permettre de mesurer la similarité de deux images binaires.

La fonction de perte choisie pour ce type de classification est l'**entropie croisée binaire**. Admettons que l'on veuille classer chaque pixel d'une image dans l'une des deux catégories suivantes: noir ou blanc. Les pixels labellisés en noir prendrons la valeur 1 et les pixels labellisés en blanc prendrons la valeur 0. Si les pixels noirs suivent une distribution théorique dont la loi de probabilité est donnée par Q et que la probabilité prédite par le modèle est P, alors l'entropie $H(Q)$ de Q est définie par l'expression:

$$H(Q) = \mathbb{E}_Q[-\log(Q)]$$

Et on nomme **entropie croisée** de Q par rapport à P l'entité:

$$H_Q(P) = \mathbb{E}_Q[-\log(P)] = -\sum_{c=1}^{N} Q(y_c)\ \log(P(y_c))$$

Avec deux distributions discrètes, où $c \in C$ désigne une classe et N le nombre de pixels à prédire.

Dans le cas d'une classification binaire, il est courant de prendre Q telle que $Q(y_i = 1) = y_i$ et $Q(y_i = 0) = 1 - y_i$. On note $P(y_i = 1) = \hat{p_i}$ la probabilité prédite par le modèle concernant le ième pixel. Il vient que l'entropie croisée pour un pixel y_i vaut:

$$H_i(P, Q) = -\sum_{c=0}^{1} Q(y_i = c)\ \log(P(y_i = c)) = -[\ y_i\ \log(\hat{p_i})\ +\ (1 - y_i)\ \log(1 - \hat{p_i})\ ]$$

A partir de cette quantité, on calcule la moyenne empirique des entropies croisées sur l'ensemble de nos N pixels qui est la fonction de perte L ci-après:

$$\begin{aligned} L &= \frac{1}{N}\ \sum_{i=1}^{N} H_i(P, Q) \\ &= -\frac{1}{N} \sum_{i=1}^{N}\ y_i \log \hat{p_i} + (1 - y_i) \log\left(1 - \hat{p_i}\right) \end{aligned}$$

Le cas idéal serait celui où les lois théoriques et prédites sont en adéquation pour l'ensemble des points étudiés. Dans ce cas de figure la fonction de perte serait nulle, or le modèle n'est pas parfait donc la fonction de perte pour notre modèle est toujours supérieure à 0. Pour comprendre cela, il faut se focaliser sur la différence $D_{KL} = H(P, Q) - H(Q)$ (où $H(Q) = \mathbb{E}_Q[-\log(Q)]$ l'entropie de Q) pour un point donné. D_{KL} est la différence de Kullback-Leibler dont l'expression est la suivante: $D_{KL} = -\sum_{c \in \mathcal{C}} Q(c) \log\left(\frac{Q(c)}{P(c)}\right)$. La différence de Kullback-leibler est toujours supérieure à 0 puisque la fonction logarithme néperien est concave sur $\mathbb{R}^+*$. Cela traduit le fait que plus les prédictions du modèle comportent d'imprécisions, plus la valeur de la perte est importante et c'est pour cela que l'on cherche à la minimiser.

Le **coefficient de Dice** est un outil de comparaison de deux échantillons. Ici, les échantillons seront le masque que l'on fournit en entrée du réseau dans le lot d'entraînement et le masque prédit en sortie du réseau. On note alors $\mathcal{X}$ l'ensemble formé par les pixels noirs du masque d'entrée et $\mathcal{Y}$ celui formé des pixels noirs du masque prédit. Le coefficient de Dice $\mathcal{D}$ pour ces deux échantillons vaut alors:

$$\mathcal{D} = \frac{2\ |\mathcal{X} \cap \mathcal{Y}|}{|\mathcal{X}| + |\mathcal{Y}|}$$

où |A| est le cardinal de l'ensemble A.

8 Résultats

8.1 Premiers résultats de l'Ultrasound model:

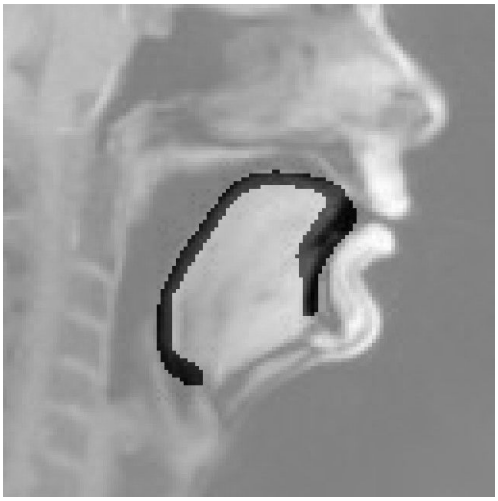


Figure 7

Ces premières images de sortie semblent être en juxtaposition avec le contour de la langue comme le montre la figure ci-dessus. Les paramètres pris en compte dans l'apprentissage qui a abouti à ce résultat sont un batch size de 1, un fichier d'entraînement contenat les 272 images que l'on a à dispostion et un nombre d'époques égal à 5. En revanche, les contours produits demeurent trop épais compte tenu des attentes que l'on s'impose. La première raison qui expliquerait cela est le fait que notre modèle est surentraîné et produit un contour proche de ce que l'on veut mais avec de grosses imprécisions. La deuxième raison pourrait probablement provenir du fait que notre réseau applique à l'image du masque une succession de couches convolutionnelles et de Pooling. Ces méthodes pourraient conduire à un élargissement de la zone de contour due à la taille du filtre. En effet, lors d'une étape de convolution de noyau 2x2, les pixels sont traités 4 par 4 et non individuellement, ce qui pourrait propager de grandes valeurs au voisinage du contour désiré.

Erreurs notables du modèle :

De surcroît, on dégage plusieurs particularités posant problèmes en balayant les masques en sortie du CNN. En voici (ci-dessous) deux exemples typiques.

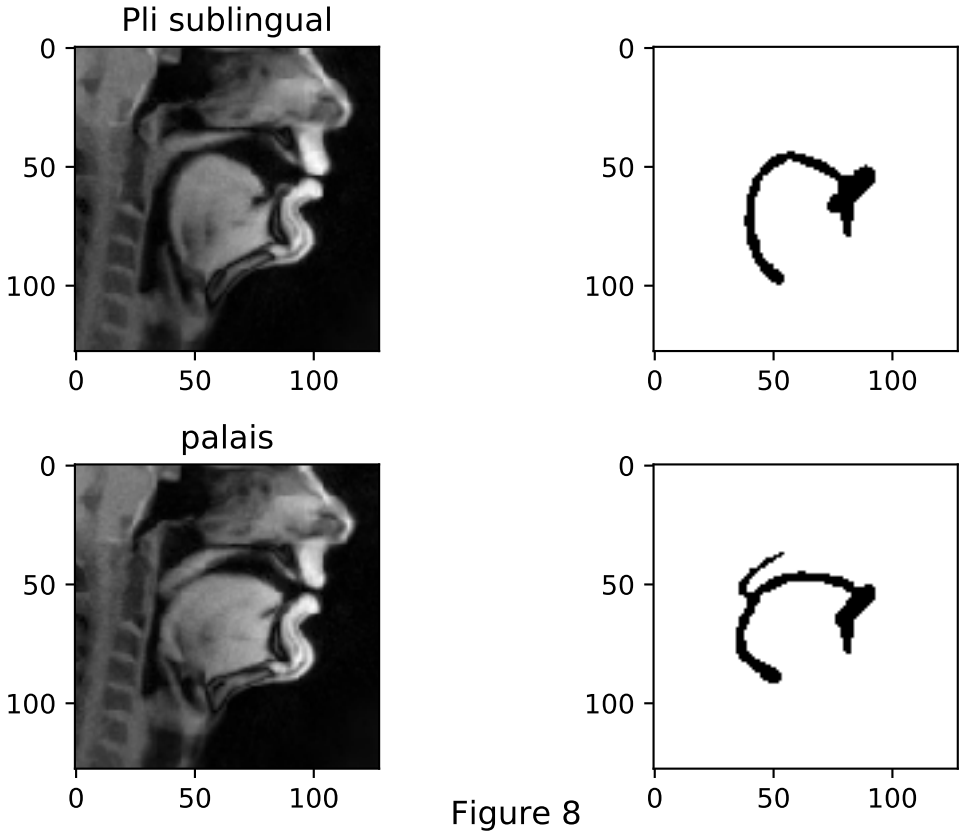


Figure 8

Observations : Si certaines images sont raisonnalblement contourées, il reste néanmoins de grosses imprécisions sur le traitement des images représentant un pli sublingual relativement marqué. De plus, les masques générés à partir des images sur lesquelles la langue est en contact avec le palais présentent un contour qui semble être la délimitation de ce dernier. Cela provient très sûrement du fait que l'on fait du surapprentissage et que les poids ont été modifiés de sorte à détecter le contour du palais, étant donné de sa proximité avec la langue dans bon nombre d'images.

8.2 Augmentation de la taille des données

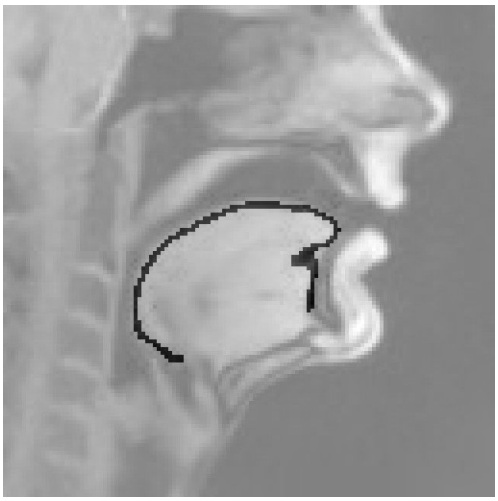


Figure 9

Le souci majeur du modèle étant de détenir trop peu d'images sur lesquelles entraîner notre réseau de neurones profond, j'ai d'abord doubler la taille du training set (dont la taille était alors de 474 images) afin d'observer les changements sur les masques. Pour constater une réelle différence, j'ai ciblé des images sur lesquelles la langue était particulièrement pliée. Il s'avèra que la première amélioration que l'on visait était atteinte. En effet, les contours générés en sortie sont plus fins et permettent désormais de visualiser la forme de la langue avec plus de précision. On dégage également le fait que le pli sublingual qui posait autrefois problème est reconnu par le réseau, bien que cela reste grossier sur de rares images. Cela n'est, en revanche, pas inquiétant en ce sens que l'on ne s'attend pas à ce que le réseau fasse beaucoup mieux que le détourage manuel dont il s'inspire.

Enfin, la dernière observation que l'on peut faire à partir des récents résultats, et qui n'est pas des moindres, est que le modèle semble s'être amélioré sur les images précèdemment imprécises, mais fait moins bien pour les contours plats. Tout se déroule alors comme si on avait spécialisé notre réseau pour la détection des plis de la langue. Cette spécialisation n'est pas surprenante puisque l'augmentation des données d'apprentissage nécessite d'augmenter également le nombre d'époques afin d'obtenir des résultats exploitables.

Le problème est à présent la détection des contours lorsqu'il y a contact entre la langue et d'autres tissus. La séparation étant visuellement difficile, le détourage manuel va conduire à de nombreuses imprécisions qui risquent de se retrouver en sortie de l'autoencodeur. L'archétype de cette problèmatique est le contact entre la langue et le palais lorsque le sujet ne parle pas car la seule délimitation visible correspond à la surface supérieure du palais. Ainsi la portion de la langue collée au palais n'est pas détectée et le réseau se sert de l'unique contour qu'il détient (le palais), comme on peut le voir sur l'image suivante:

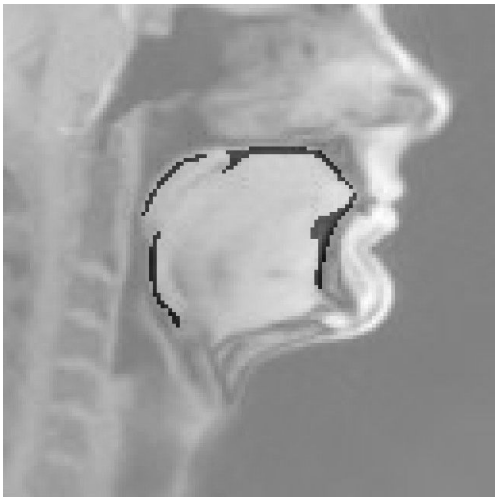


Figure 10

8.3 Augmentation du batch size

On a déjà vu en 7.3.2 la corrélation existant entre la précision du modèle et une taille de batch élevée. On a donc fait le choix après plusieurs essais d'utiliser une taille de batch de 8 compte tenu de l'adéquation entre les masques prédits et la presque totalité des images IRM qui ont été testées.

Notons en premier lieu que cette modification des paramètres a permis de réduire considérablement l'épaisseur du contour prédit, que ce soit pour les images du lot d'entraînement comme pour des images de test non apprises. Le pli sublingual se distingue alors plus clairement et il est facile de reconnaître avec exactitude le contour de la langue étant donné que le contour prédit est par endroit large d'un seul pixel. Voyons ce à quoi cela peut ressembler pour trois images différentes: —IMAGES—

En second lieu, il est particulèrement intéressant de visualiser la spécialisation du réseau pour les cas spéciaux dont on a parlé précèdemment. On peut remarquer par exemple sur des images non apprises où la langue est en contact avec le palais que le réseau passe par diverses phases d'apprentissage. Tandis qu'il détoure d'abord une partie du palais, il apprend ensuite à supprimer la zone du contour qui est fausse et rectifie ainsi ses erreurs. Voici le type d'évolution que l'on peut observer au cours d'un apprentissage:

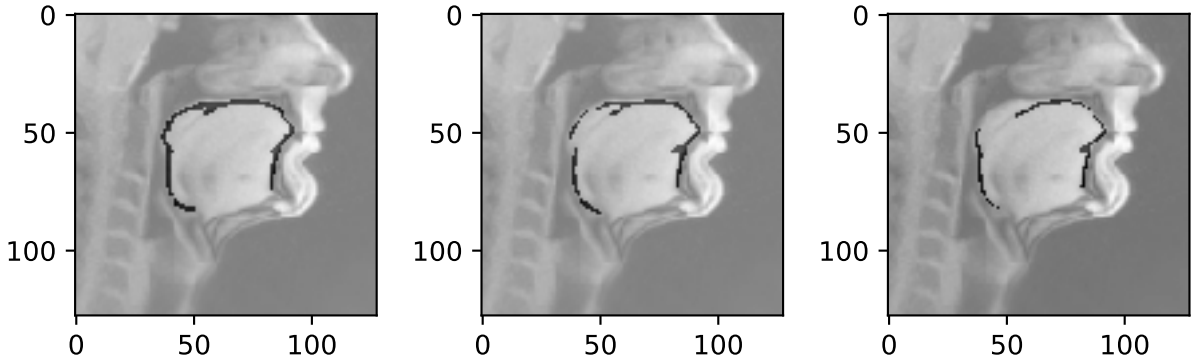


Figure 11

Le réseau “apprend” à sélectionner uniquement le contour de la langue et va donc au-delà de la simple détection de contours, on comprend alors l'utilité des réseaux de neurones.

8.4 Deuxième augmentation des données

On a finalement augmenter à nouveau la taille du lot d'entraînement en ajoutant une vingtaine de contours d'images où langue et palais étaient presque confondus puis un bon nombre d'images auquelles on a fait subir une légère translation. Le lot d'entraînement comporte alors 609 images et il s'agit, comme attendu, de la meilleure manière d'entraîner le modèle qui ait été expérimentée ici. L'effet recherché, à savoir la détection d'un contour peu visible dans le cas d'un contact avec le palais, est à présent réalisable et la précision globale du réseau n'en est que meilleure. Si l'on reprend l'exemple de l'image présentée ci-dessus avec le nouvel entraînement, voici ce que l'on obtient en sortie du réseau:

Figure

Remarque Pour le meilleur entraînement réalisé, il subsiste un problème récurrent qui concerne les images où le bout de la langue est en contact avec les levres. Le contour est alors impossible à trouver pour le réseau car il n'a jamais appris de tels contours. On pourrait donc répéter ce qui a été fait pour le contact entre la langue et le palais mais on ne juge pas nécessaire de connaître le contour de la langue dans cette configuration puisque nous nous intéressons à ce qu'il se passe dans le conduit vocal uniquement. Il est de ce fait possible d'obtenir des résultats semblables à l'image ci-dessous.

Figure

Résumé des paramètres choisis

Images d'entraînement	Nombre d'images de test	Nombre d'itérations	Taille du batch
609	101	6250	8

9 Limites

L'examen des travaux réalisés nous a permis de dégager certaines limites du modèle. Pour commencer, le réseau exploité ici est classique et la qualité de ses résultats a été prouvée à de nombreuses reprises pour ce qui est de la segmentaion d'images biomédicales, mais il n'est pas spécialisé dans la réalisation de détection de contours. L'objet du projet était donc de ré-entraîner ce modèle sans en changer l'architecture, il est donc possible et fort probable qu'un réseau autrement constitué aboutisse à de meilleurs résultats. Vient ensuite le fait que nous traitons des images IRM en niveaux de gris et que le point de départ de notre travail est un détourage manuel. Or les images en niveaux de gris ne présentent pas toujours de contours très contrastés mais plutôt une sorte de halo qui rend le détourage manuel, et donc automatique, approximatif, c'est pourquoi on peut distinguer un léger décalage entre le contour produit et l'image d'origine. On comprend alors pourquoi il est essentiel de détourer les images d'entraînement de la même manière dans les zones de fortes indécisions, au risque d'y trouver en sortie du réseau des contours épais. S'ajoute à cela le fait que la taille de l'échantillon d'images sur lequel on se base pour entraîner le modèle reste faible malgré les diverses augmentations dont elle a pu bénéficier. Il est donc envisageable que l'imprécision augmente en réalisant des tests sur des séquences d'images qui s'écartent des données exploitées. Pour finir, la contrainte concernant le choix du nombre d'itérations à effectuer lors de l'apprentissage ne conduit pas à de meilleurs résultats pour chaque image individuellement mais sur l'ensemble du lot. Notons que ce choix repose sur un algorithme de comparaison simpliste et que le choix des autres paramètres est avant tout empirique.

10 Conclusion

Pour conclure ce document, le peu d'éléments théoriques apportés au sujet des réseaux de neurones artificiels nous a permis de comprendre les tenants et aboutissants du réseau U-Net et de le spécialiser dans la détection du contour de la langue. La majeure partie du travail fut consacrée à une recherche à la fois empirique et théorique des différents paramètres à choisir afin de mener au mieux l'apprentissage du réseau de neurones et d'en affiner les résultats. Il a été également nécessaire d'implémenter quelques fonctions python de pré-traitement des données dans le but d'en extraire au mieux ce qui suscitait notre intérêt. Au vue des contours obtenus à la fin des expérimentations, il n'a finalement pas semblé utile de faire du Transfert learning. En effet, la méthode par inférence a conduit à une détection satisfaisante du contour de la langue en ce sens qu'elle est relativement précise spatialement et égalemnent en terme d'épaisseur du contour. Néanmoins, il serait possible d'envisager le problème de manière plus approfondie en réfléchissant notamment à une méthode qui prendrait en compte les images qui précèdent l'image traitée afin d'en déduire les positions possibles de la langue (Voir [4]).

Figure :Rendu final

11 Annexes

11.1 Dépôt des codes

Un dépôt Github contenant les fichiers de code ainsi qu'une partie des données est disponible à cette adresse: https://github.com/AlexisHoussard/UltrasoundImageSegmentation

11.2 Références:

[1] : [https://deeplearning4j.org/docs/latest/deeplearning4j-nn-early-stopping]: "What is early-stopping"

[2] : [https://www.degruyter.com/downloadpdf/j/itms.2017.20.issue-1/itms-2017-0003/itms-2017-0003.pdf]: "Impact of Training Set Batch Size on the Performance of Convolutional Neural Networks for Diverse Datasets"

[] : [https://developers.google.com/machine-learning/glossary/?hl=fr]: Google glossary for machine learning

[] : [https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a]: "Understand the binary cross-entropy"

[] : [Aurore Jaumard-Hakoun, Kele Xu, Pierre Roussel-Ragot, Gérard Dreyfus, Bruce Denby. Mai 2016]: "Tongue contour extraction from ultrasound images based on deep neural network"