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

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. 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 muscles 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 20 ms.

Cela nous amène donc à la principale motivation de ce stage, à savoir de réaliser le détourage de la langue à partir de nos 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.

—Annonce breve du plan et explication du deroulement— (approche simpliste des réseaux de neurones)

4 Environnement

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 entoure 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.

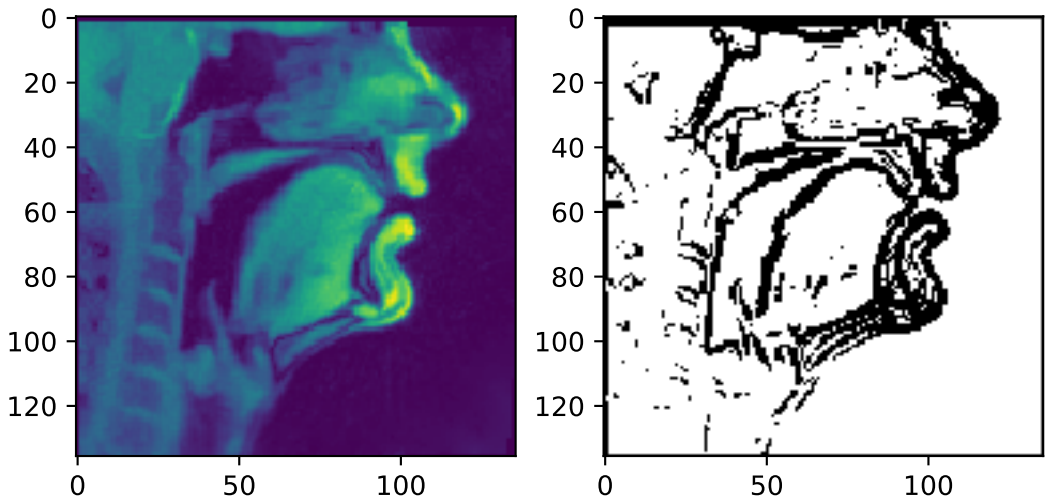
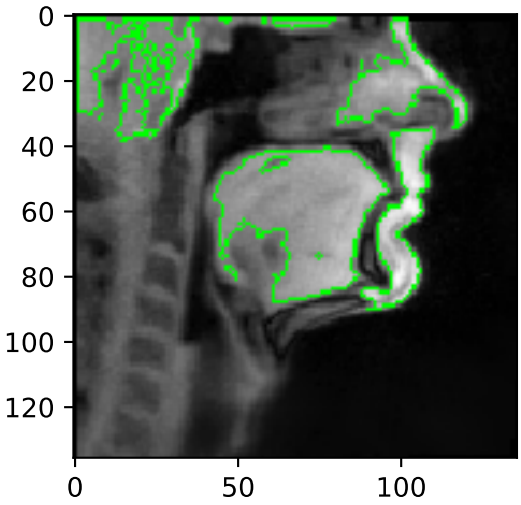


Figure 1

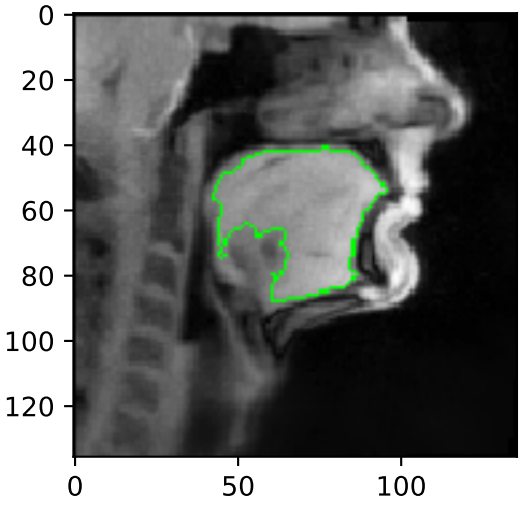
Observations: Bien que rudimentaire, la méthode implémentée dessine des contours qui semblent être les bons. Néanmoins, remarquons que ceux ci ont une épaisseur de plusieurs pixels, ce qui implique de grosses imprécisions sur la forme des tissus. De plus, cela ne nous permet pas de restreindre le dessin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

6.1.2 Première utilisation d' OpenCV

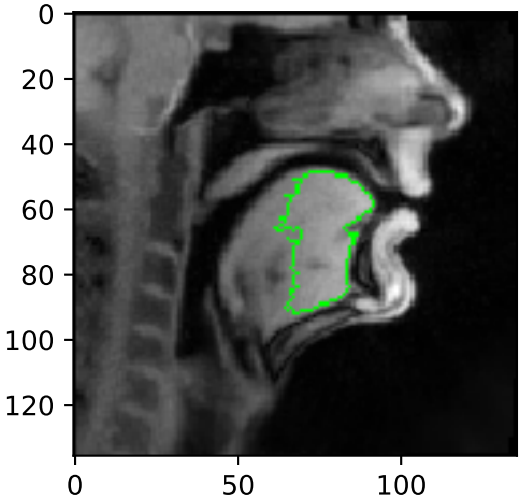
Nous nous proposons à présent de travailler avec des outils d'extraction de contours existants sous Python. La plus simple des méthodes proposées par l'extension OpenCV fournit un résultat(ci-dessous) a priori satisfaisant pour la première image dont on dispose.



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.

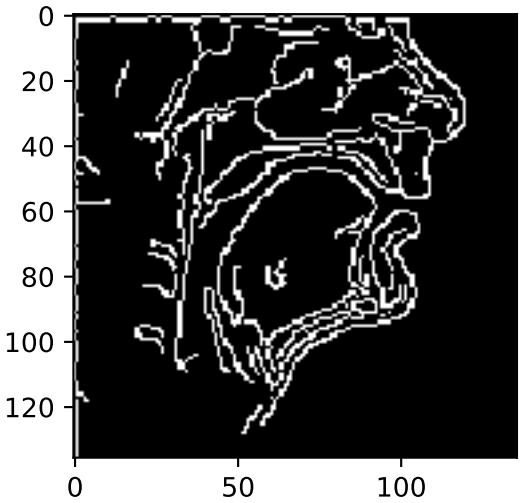


Voyons néanmoins que la détection du contour de la langue n'est pas aussi satisfaisant sur toutes les images IRM disponibles. Par exemple, le dessin du contour recherché sur l'image 500 donne le résultat non exploitable ci-dessous. En effet, le dessin du contour est perturbé par la présence régulière 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.



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)



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 contourées 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 couches de neurones, 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.

L'objectif est tout d'abord, pour des raisons de simplicité, 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 réfléchis 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).

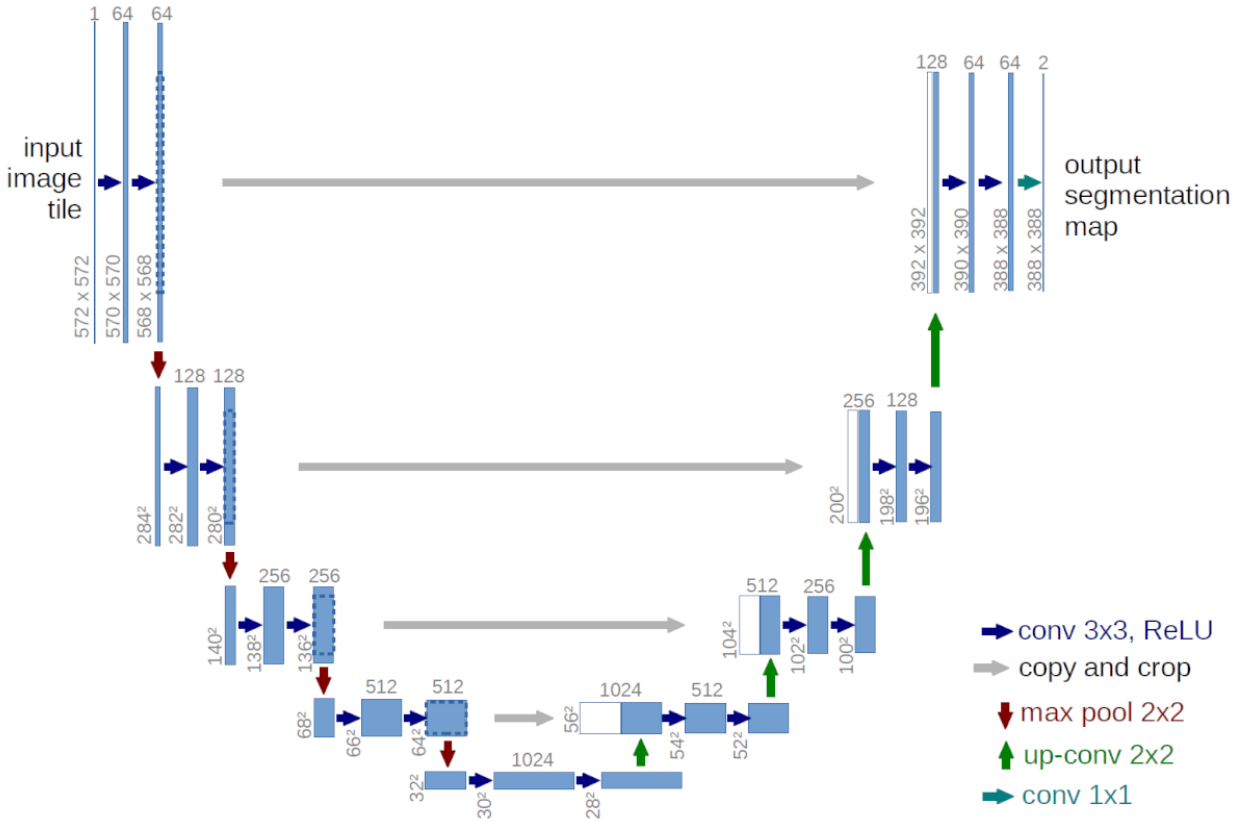


Figure 2

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).

– Explication de l'architecture du réseau (notamment pourquoi on effectue les étapes de copy/crop) –

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) et de reconstruire une image dont la taille est la moitié de celle d'entrée selon le schéma ci-dessous.

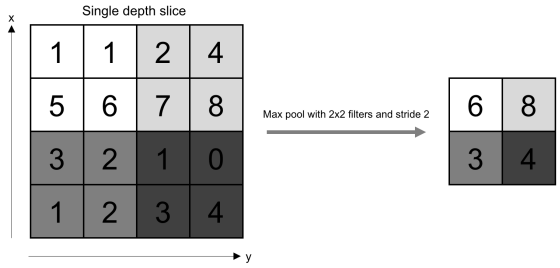


Figure 3

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.

—Influence taille de l'image—

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

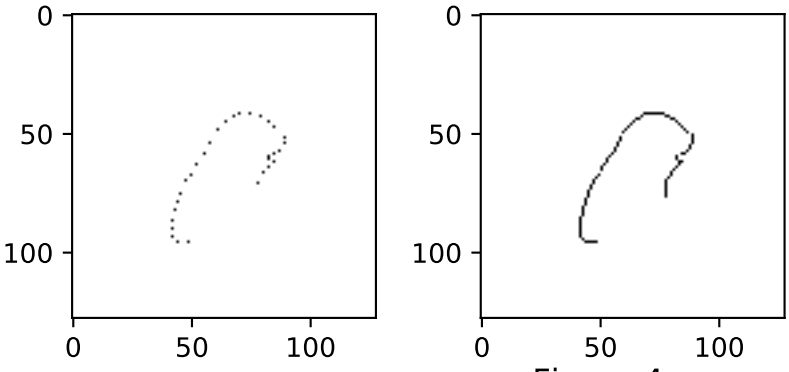


Figure 4

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.

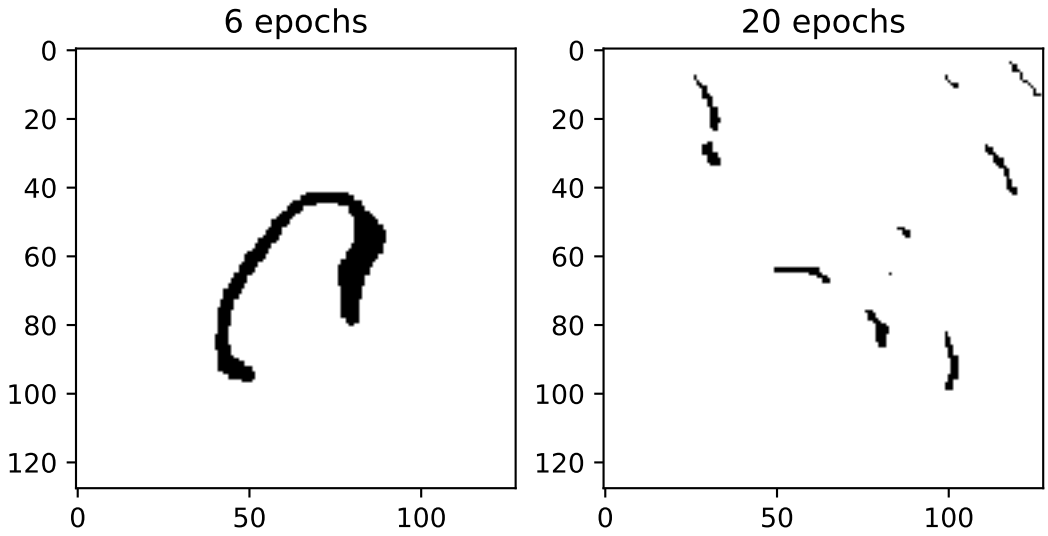


Figure 4

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 de par l'objet représenté sur l'image de sortie. En claire, 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 supplante 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 n'est pas issu de nos résultats mais illustre clairement le propos).

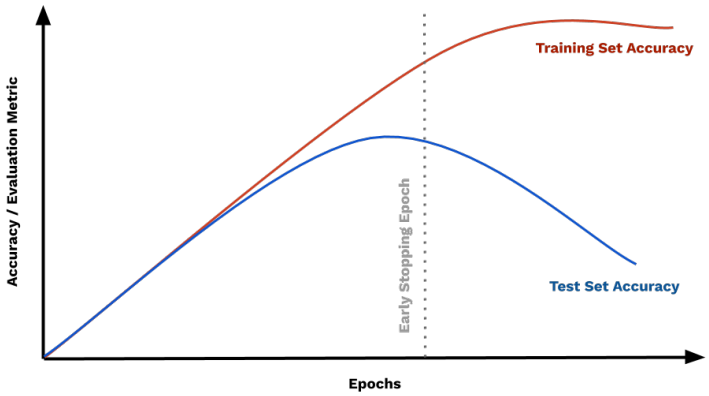


Figure 5

7.3.2 Taille d'un batch

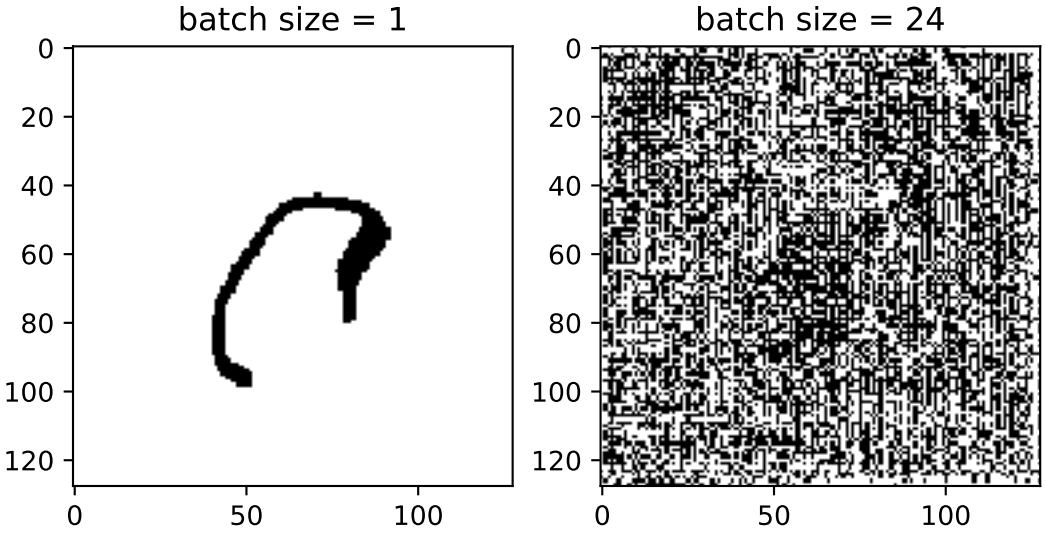


Figure 6

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.

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 comme on peut le voir ci-après. Ce défaut constitue une raison supplémentaire d'utiliser la méthode d'early stopping.

– ILLUSTRATION DU PB —

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 kernel 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.

(d'apres Tensorflow: 9239714)

Couche	Formule	Paramètres
Conv1	((3x3x1+1)x32)x2	640
Conv2	((3x3x32+1)x64)x2	36992
Conv3	((3x3x64+1)x128)x2	147712
Conv4	((3x3x128+1)x256)x2	590336
Conv5	((3x3x256+1)x512)x2	2360320
Deconv1	((2x2x512+1)x512)	1049088
Conv6	((3x3x512+1)x256)x2	1179904
Deconv2	((2x2x256+1)x256)	262400
Conv7	((3x3x256+1)x128)x2	295040
Deconv3	((2x2x128+1)x128)	65664
Conv8	((3x3x128+1)x64)x2	147584
Deconv4	((2x2x64+1)x64)	16448
Conv9	((3x3x64+1)x32)x2	36928
Conv1x1	((1x1x32+1)x2)	66
Total		7664066

REVOIR CALCULS!

D'après le calcul explicité précèdemment, il faudrait au minimum environ $\frac{7664066}{128*128} = 467$ 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 juge cela inintéressant puisqu'il ne sera 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.

—Loss function et Dice coefficient—

pourquoi coefficient dice si haut ?

On peut fair de la validation "naive" en comparant le nombre de pixels ou en faisant un produit convolutionnel

8 Résultats

8.1 Tentative de Transfert Learning

8.2 Premiers résultats de l'Ultrasound model:

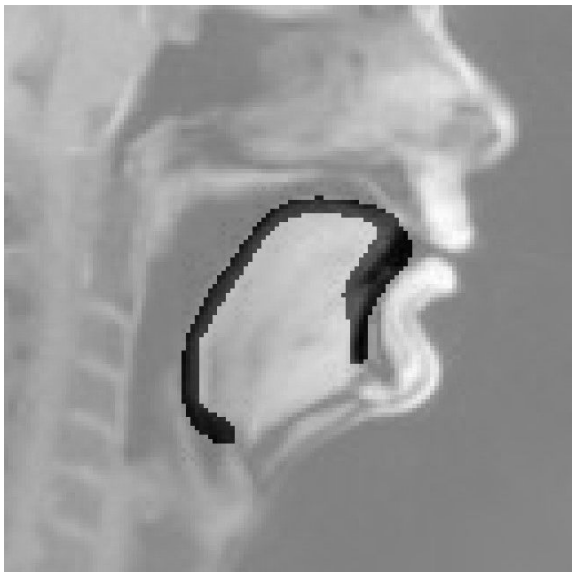


Figure 7

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 cela 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 kernel 2x2, les pixels sont traités 4 par 4 et non individuellement, ce qui pourrait propager de grandes valeurs au voisinage du contour désiré. Enfin, la loss

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.

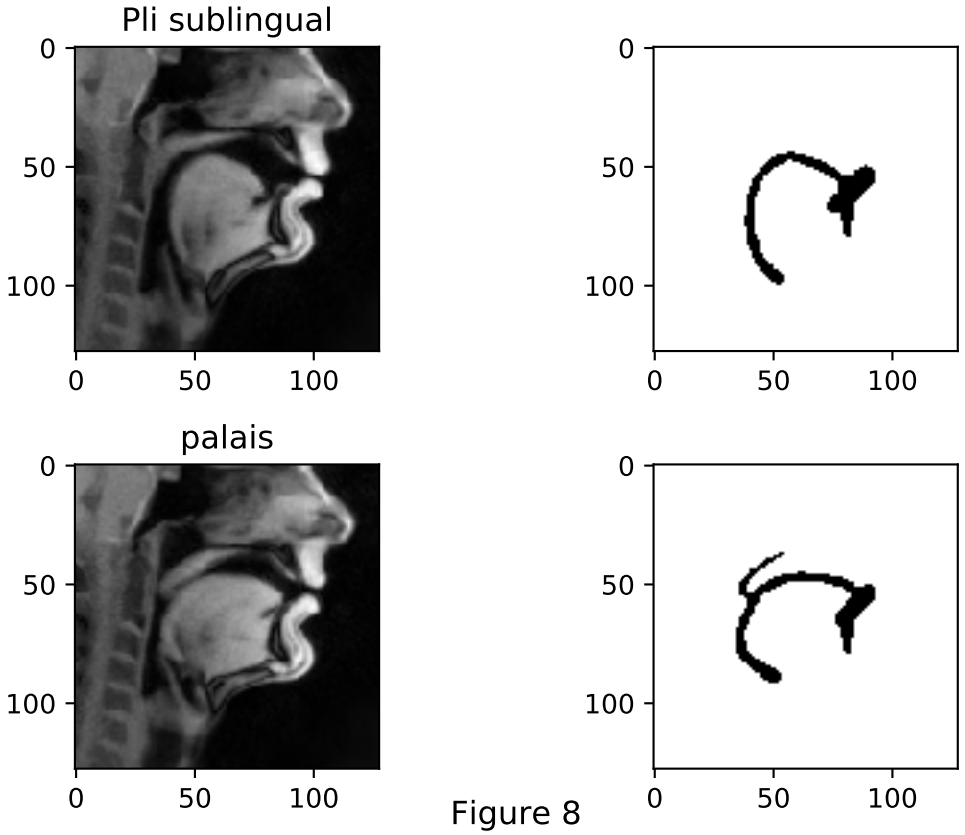


Figure 8

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 nombre d'images.

Autre réflexion: Comment gérer les images en niveaux de gris? (sorte de halo et doute sur le contour réel de la langue)

8.3 Augmentation de la taille des données

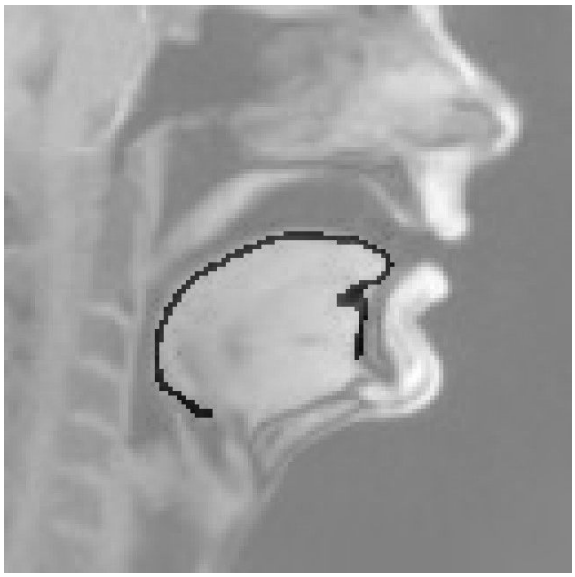


Figure 9

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 des 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:

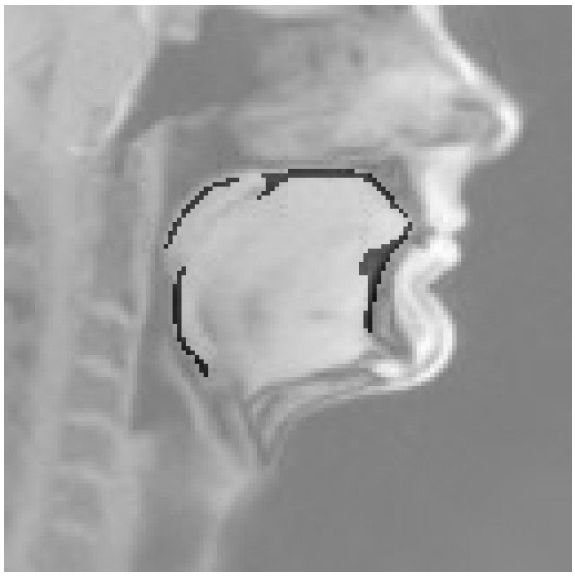


Figure 10

–Passage sur différence entre 474 images et 576 (montrer que l'on a probablement un optimum d'images à mettre entre les deux).

Pour le meilleur nombre d'images : Problème récurrent –> les images où le bout de la langue est en contact avec les levres (impossible de trouver de contour)

passage sur les limites

9 Conclusion

10 Annexes

Codes

Références: Google glossary