



# Rapport TAV

Issam Alouane

2 Juin 2023

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Méthodes variationnelles</b>	<b>3</b>
2.1	Restauration d'images . . . . .	3
2.1.1	Débruitage . . . . .	3
2.1.2	Inpainting . . . . .	5
2.2	Contours actifs . . . . .	7
2.3	Décomposition d'une image . . . . .	11
<b>3</b>	<b>Transformations vocales</b>	<b>15</b>
3.1	Reconnaissance musicale : Shazam . . . . .	16
3.2	Séparation de sources . . . . .	18

# 1 Introduction

Ce document est le rapport des 5 tps sur 10 de ce que j'ai vu en traitement des données audio-visuelles. En général, une image se compose de 3 couleurs : rouge, vert et bleu (rgb), et on va voir dans ce rapport comment nous avons utilisé les Maths pour manipuler les données des images afin de résoudre quelques problèmes rencontrés ouachever quelques besoins, en particulier voilà les tp de traitement d'image dont je vais parler dans ce rapport : restauration des images (TP5), les contours actifs (TP6) et décomposition structure + texture des images (TP8). En addition, dans ce module, on avait la chance de manipuler des données auditaires qui touchent à la vie réelle, j'ai fait le choix de parler de TP11 : shazam (on veut savoir quelle musique on a entendu par hasard chez un ami ou sur la radio) et TP12 : séparation de sources (puisque j'avais particulièrement besoin de séparer le rythme de la musique des paroles dans ma vie). Ce rapport va vous montrer les résultats à lesquels j'ai abouti à la fin de chaque tp présenté et les résultats des parties facultatives, et aussi, comment j'ai exploité les résultats ou comment j'ai re-testé mais algorithmes.

## 2 Méthodes variationnelles

### 2.1 Restauration d'images

Dans la vie réelle, nos images ne sont pas totalement nettes pour diverses raisons, par exemple : un capteur sale, l'instabilité de la caméra, l'air qui n'est pas trop propre, ou parfois nous voulons restaurer des images endommagées. Alors pour avoir des résultats satisfaisants après un traitement donné, il faut restaurer ces images tout en conservant nos informations précieuses, ce qui n'est pas toujours achevé. Nous allons voir dans cette partie quelques méthodes de restauration tout en explorant leurs avantages et inconvénients.

#### 2.1.1 Débruitage

Notre premier exemple est le débruitage, il consiste à trouver l'image la plus proche de notre image originale (corrompue par un bruit additif) de telle sorte que la nouvelle image soit lisse. La figure suivante montre le résultat d'un débruitage en utilisant le modèle variationnel de débruitage de type "Tikhonov" qui revient à minimiser une énergie dite l'énergie de Tikhonov :



Figure 1: Debruitage avec le modele de Tikhonov avec  $\lambda = 2$

On voit bien que, même si le bruit est enlevé, le résultat de cette méthode n'est pas trop satisfaisant parce qu'une bonne partie des contours n'est pas conservée et l'image débruité est flou. Et si on augmente  $\lambda$ , l'image devient de plus en plus flou.

La science évolue, on a donc arrivé à une nouvelle méthode qui consiste en un débruitage par variation totale, cette méthode consiste à remplacer le carré du gradient de l'image débruité par la norme du gradient, pour rendre l'énergie non différentiable et on l'approxime par une fonction

différentiable. La figure suivant montre le résultat d'un débruitage sur la même image, avec deux valeurs différentes de lambda :



Figure 2: Debruitage avec le modèle de VT avec lambda = 11



Figure 3: Debruitage avec le modèle de VT avec lambda = 15

Par des tests successifs, j'ai arrivé à la valeur de lambda la plus adapté pour cette image qui est 11, et qui donne un résultat disant satisfaisant : les contours conservés et l'image débruité pas flou.

On peut aussi appliquer cette méthode à une image en rgb en séparant chaque canal de couleur pour obtenir une image moins bruitée comme le montre l'exemple suivant :

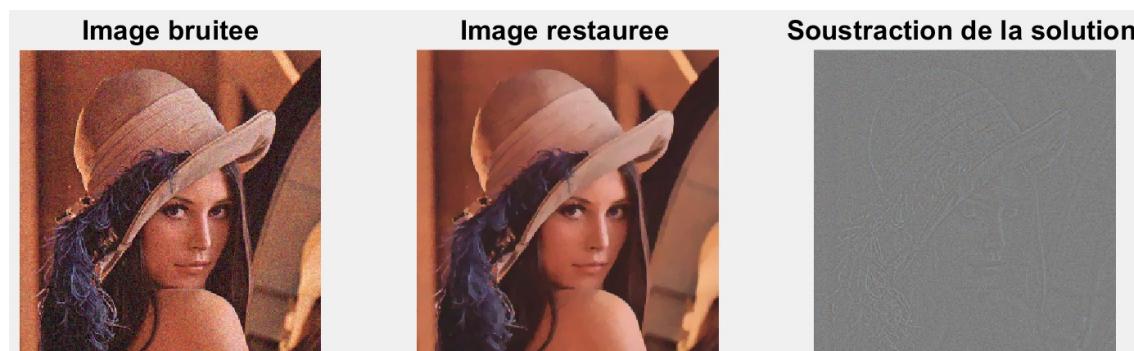


Figure 4: Debruitage avec le modele de VT sur image RGB

### 2.1.2 Inpainting

L'inpainting est une technique qui consiste à restaurer l'image dans un certain domaine. Il y a deux types d'applications : la restauration de zones endommagées, par exemple si l'image est endommagée par des tâches ou des rayures; et la réalité diminué, qui consiste à enlever un élément de l'image.

Comme première approche nous avons utiliser le modèle d'inpainting par variation totale (ou l'inpainting par diffusion), La différence entre ce modèle et le modèle de débruitage par variation totale est que le terme d'attache aux données ne doit être calculé que sur le domaine où l'image originale est fiable. Pour le masque, on a utilisé un masque prêt et on a essayé de déduire le masque à partir de l'image originale, et dans les deux cas on avait des résultats pareils (parce que dans ce cas la couleur jaune est très bien distinguée par rapport aux autres couleurs de l'image). Pour ce modèle, un lambda faible donne plus de poids à l'image d'origine, alors de plus en plus lambda est grand l'image est moins lisse. La figure suivante montre le résultat obtenu sur cette image :



Figure 5: Inpainting sur une image d'une fleur

Le résultat est satisfaisant pour la fleur avec un masque fourni, et on a aussi un bon résultat pour l'image du grenouille avec du texte comme le montre la figure suivante :



Figure 6: Inpainting sur une image d'une fleur

Par contre, si on essaie d'appliquer la même méthode pour supprimer un randonneur d'une image, le résultat n'est pas trop satisfaisant comme le montre la figure suivante :



Figure 7: Inpainting sur l'image du randonneur

On fait alors appel à la réalité diminuée, dans ce cas l'inpainting "par rapiéçage" consiste à choisir un pixel aléatoire  $p$  de la frontière entre la partie perdue et la partie fiable de notre image, et remplacer les pixels perdus dans son voisinage par des pixels dans le voisinage d'un pixel  $q$  qui est le plus proche de  $p$  au sens de dissemblance  $d(p,q)$ . Les 2 figures suivantes montrent les résultats de la suppression des 2 randonneurs de l'image :



Figure 8: Suppression de la fille par un masque prédéfinie



Figure 9: Supresion du garçon par des frontières faites à la main

On voit bien que les résultats ne sont pas très mauvais mais ne sont pas excellents en même temps. Et cette technique peut être encore amélioré en introduisant un ordre de priorité sur les pixels  $p$ , et il existe plusieurs améliorations encore poussées qui sont même intégrées dans des logiciels bien reconnus comme PaintShop Pro.

Ces techniques d'inpainting peuvent être utilisé dans diverses domaines, citons par exemple :

- La conservation du patrimoine : Les œuvres d'art, les artefacts historiques ou les photographies anciennes peuvent être endommagés par le temps, on peut alors reconstituer ce patrimoine culturel.
- Restauration photographique : Il sera possible de réparer les imperfections et de rendre les photographies anciennes proches de leur état d'origine.
- Médecine et biologie : Dans les images médicales, il est parfois nécessaire de masquer ou de supprimer certaines parties de l'image pour des raisons de confidentialité ou de clarification. On pourra alors remplir ces zones masquées en préservant la cohérence structurelle de l'image.
- Réalité virtuelle et jeux vidéo : L'inpainting par rapiècage ou par diffusion peut être utilisé pour améliorer les graphismes dans les environnements virtuels ou les jeux vidéo. En comblant les lacunes ou les imperfections dans les textures ou les modèles.

Enfin, je me suis bien amusé à essayer de restaurer le Colisée de Rome. C'est un moment le plus adapté à notre algorithme puisqu'on peut reconstruire les parties manquantes à partir des parties présentes, mais le problème était que nos fenêtres sont plutôt carrés et le Colisée a une forme circulaire (J'ai essayé de diminuer le paramètre  $t$  pour remédier à ce problème mais sans résultat). Notons bien que notre algorithme n'est pas trop robuste, le moindre des changements impliquent une grosse différence dans l'image restaurée. Voilà 2 résultats qui montrent ces aspects :

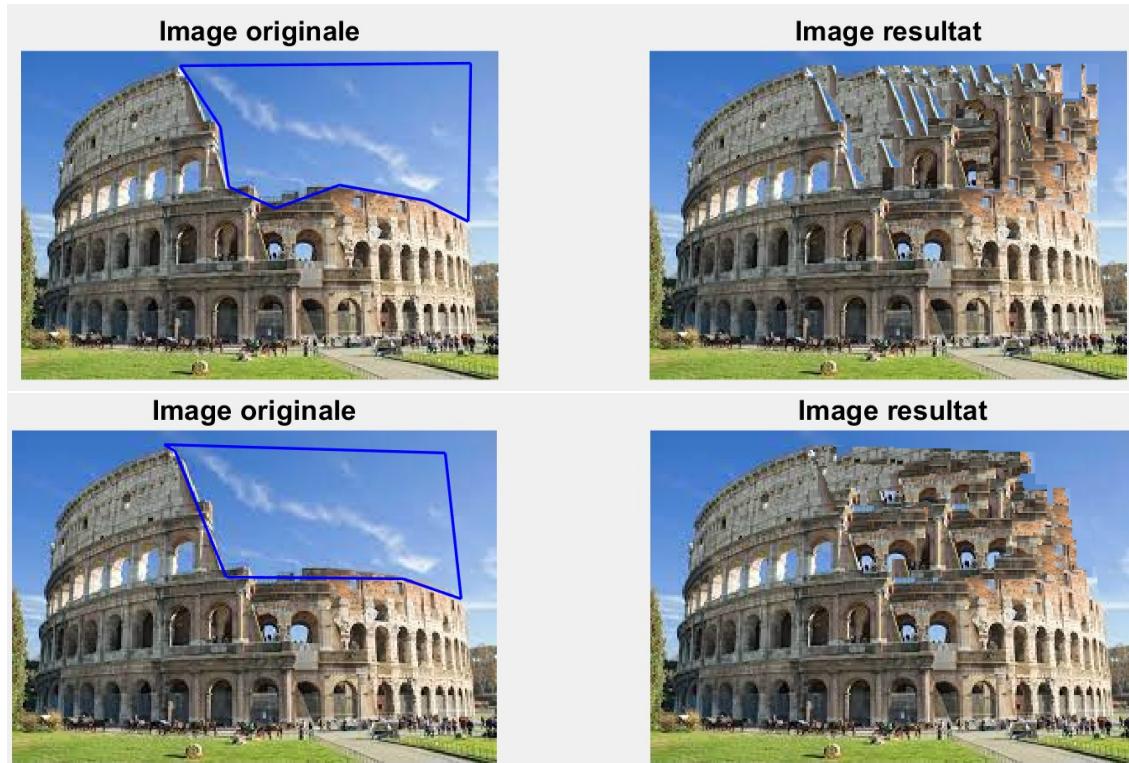


Figure 10: Restauration du Colisée de Rome par l'inpainting par rapiècage

## 2.2 Contours actifs

Un contour actif (snake) est une courbe qui se déforme au cours du temps pour se coller aux contours d'un objet. Ce qui permet de réaliser une segmentation d'objet visible qui est largement utilisé dans plusieurs domaines (Médecine et imagerie médicale, Surveillance et sécurité, Analyse

d'images satellite ...)

En premier temps, nous allons utiliser le modèle original proposé par Kass, Witkins et Terzopoulos en 1988, qui consiste en utilisation de deux paramètres alpha et beta choisis par l'utilisateur dans le calcul de l'énergie interne, ainsi que la dérivée première et seconde du contour qui est représenté par la courbe paramétrique  $P(s)$ . Pour l'adéquation aux contours, on utilise une énergie externe qui pénalise les faibles gradients du niveau de gris  $u$ . La figure suivante montre le champ de force associé à l'énergie externe obtenu par l'utilisation naïve de ce modèle.

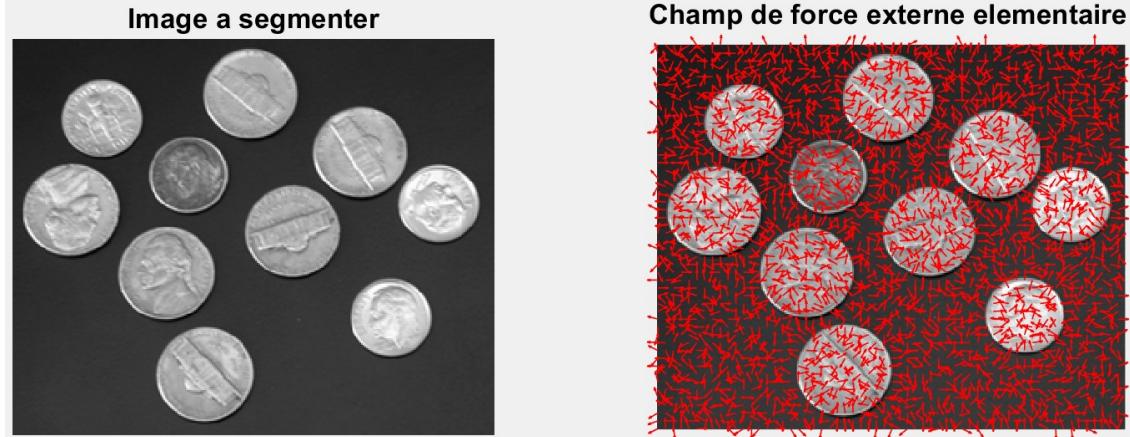


Figure 11: Le champ de force associé à l'énergie externe

Le résultat n'est pas trop bon, puisqu'on veut utiliser ces champs de force pour rediriger le contour actifs vers les contours de l'objet. Alors, il était suggéré d'utiliser un filtre de convolution gaussien à l'image avant de calculer son gradient. Et voilà le résultat obtenu qui paraît plus satisfaisant :

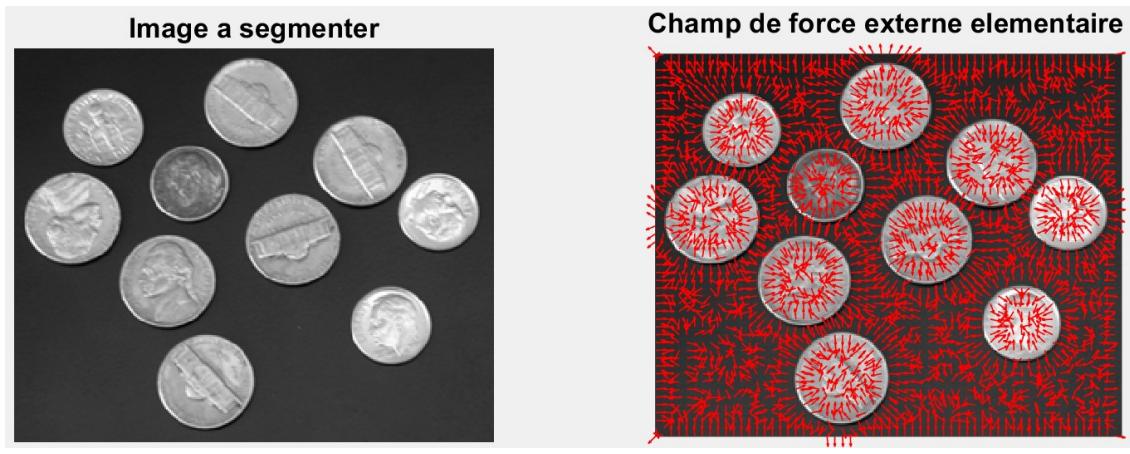


Figure 12: Le champ de force associé à l'énergie externe après application du filtre gaussien

Maintenant, d'après que le champ de force est acquis, il faut implémenter l'algorithme principal. On avait utilisé une schéma itératif qui se base sur la minimisation de la somme des deux énergies. Notons bien que le résultat de cet algorithme dépend largement de l'initialisation, une très mauvaise initialisation ne vas pas donner un bon résultat., voilà un exemple d'utilisation de cet algorithme :

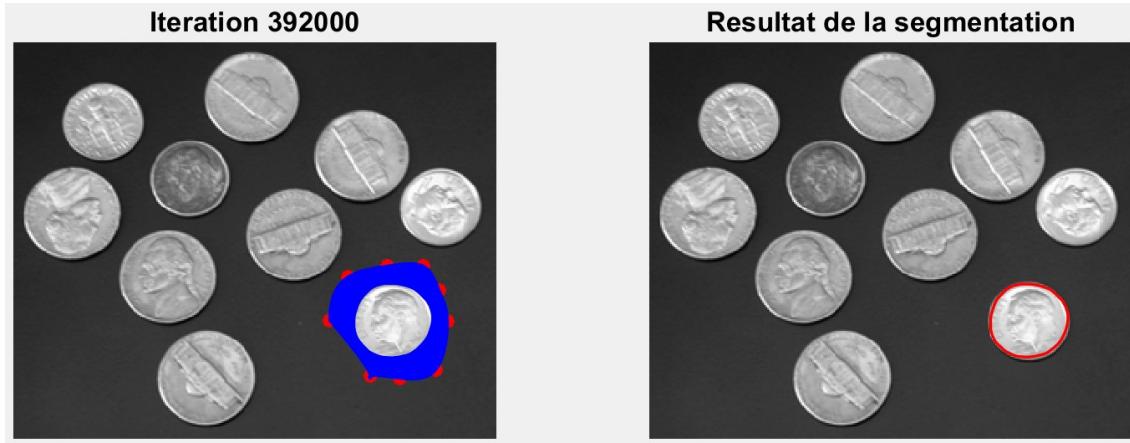


Figure 13: segmentation d'un objet dans une image

Certes, le calcul des champs de force avec la méthode précédemment évoqué n'est pas toujours précis, ce que montre l'exemple de l'image suivante où le contour actif a tendance à traverser le contour de l'objet comme le montre la figure suivante :

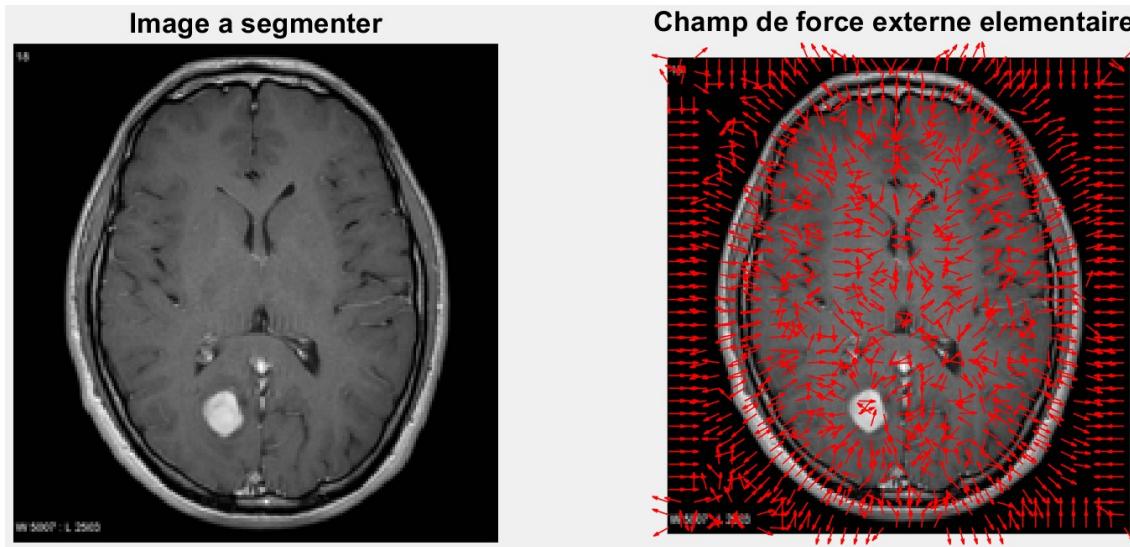


Figure 14: champ de force associé à l'énergie externe

Nous avons alors choisi de modifier la force externe en remplaçant le modèle actuel par le modèle de diffusion vers les contours (GVF, pour Gradient Vector Flow, cf. [Xu et Prince, 1998]). On a arrivé à améliorer notre champ de force :

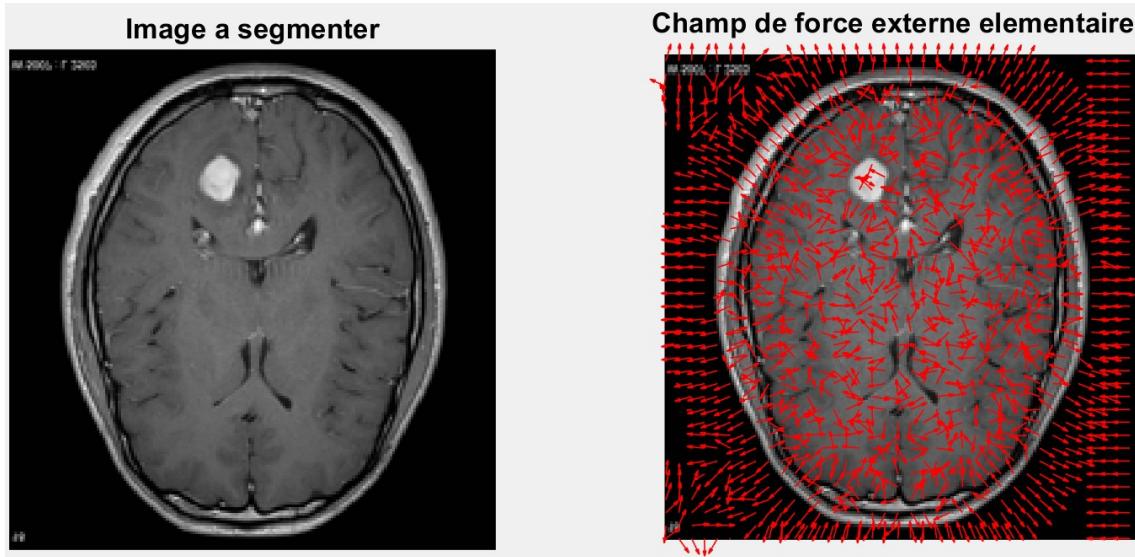


Figure 15: champ de force associé à l'énergie externe après modification

Ainsi, avec une modification du paramètre beta, on a un résultat acceptable lors de la segmentation de l'objet :

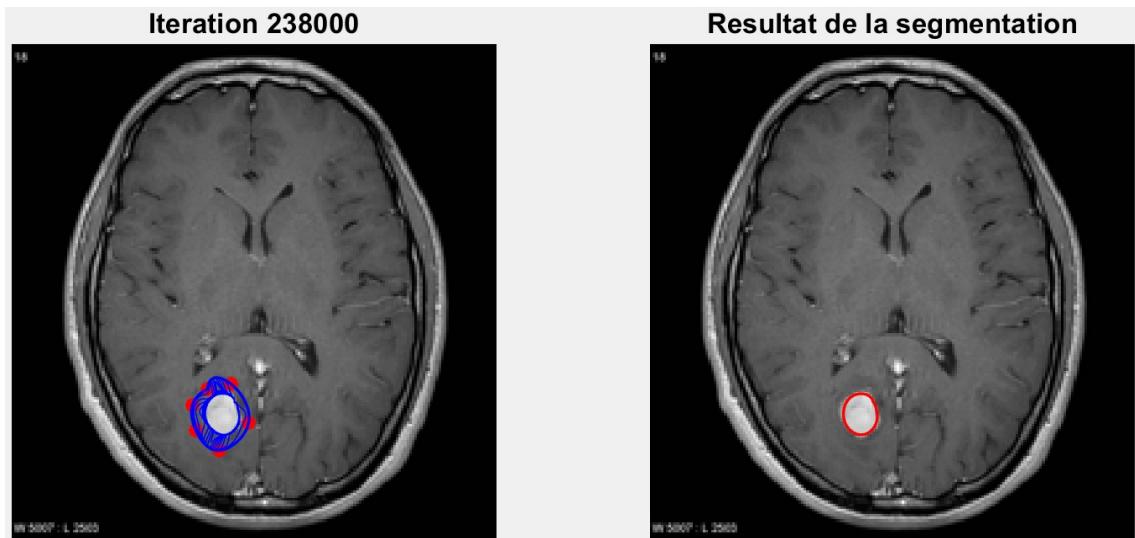


Figure 16: segmentation de l'objet

Et comme dernier test pour notre code, nous allons appliquer nos algorithmes sur une image de pomme qui est, d'après l'énoncé, particulièrement difficile à segmenter ainsi on a les champs de force et l'exemple de segmentation suivants :

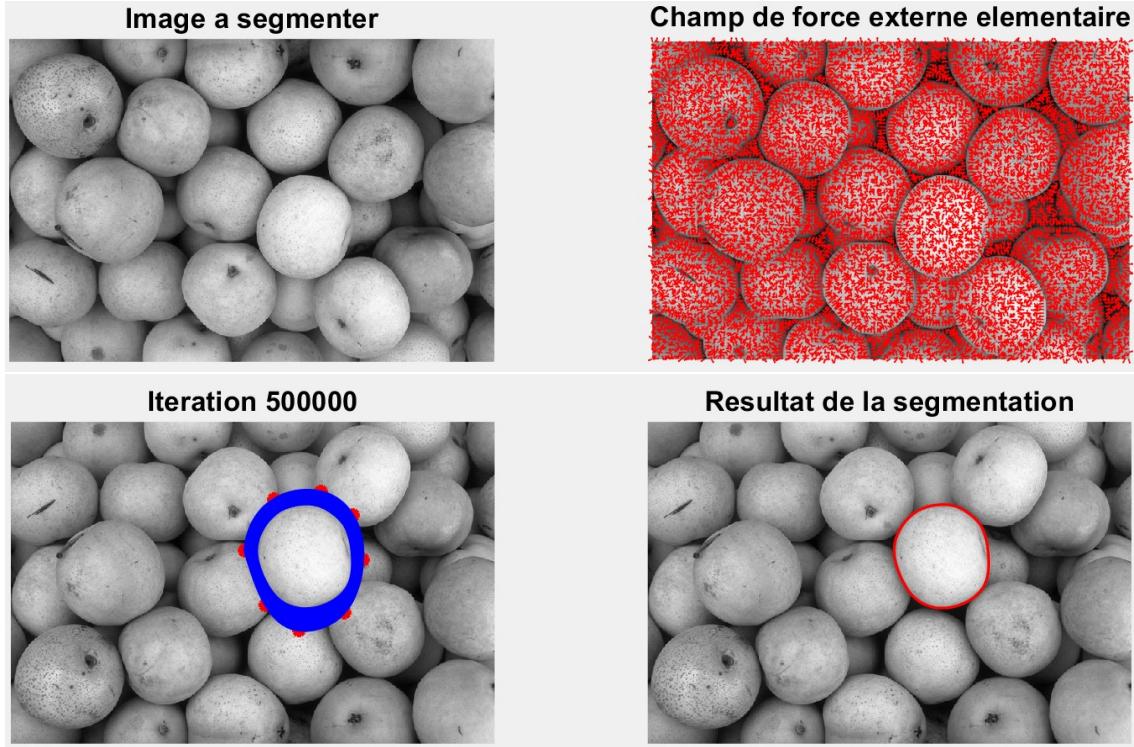


Figure 17: Application de nos algorithmes sur l'image des pommes

Les résultats obtenus sont trop satisfaisants mais plus l'image est compliquée plus notre algorithme devient lente. Ce que ouvre la porte à des aspects d'optimisation pour qu'on puisse segmenter les images compliquées (de haute qualité) en un temps raisonnable.

Ces snakes peuvent être utilisés en plusieurs domaines, citons par exemple :

- Segmentation d'images : Ils sont utilisés pour segmenter les objets d'intérêt dans une image en délimitant leurs contours précis.
- Reconnaissance de formes : Les contours actifs peuvent être utilisés pour extraire des caractéristiques de forme dans des tâches de reconnaissance de formes et de classification d'objets.
- Analyse de mouvement : Les snakes peuvent être utilisés pour suivre la localiser d'objets dans une séquence d'images, ce qui est utile dans la surveillance vidéo par exemple.

En plus, je voulais appliquer une reconnaissance de forme sur une image dans le domaine médical mais ça existe déjà dans l'énoncé, et la segmentation a été faite en Figure 16. Cette segmentation représente une bonne application des contours actifs dans la vie réelle.

### 2.3 Décomposition d'une image

Le décomposition d'une image est l'écriture de l'image sous la forme d'une somme sous la forme:  $u = \bar{u} + \bar{u}^c$  avec  $\bar{u}$  et  $\bar{u}^c$  vérifiant des conditions données (on ne fait pas une décomposition pour rien ;)). En effet, il faut juste que  $\bar{u}$  satisfasse la condition et après  $\bar{u}^c$  va automatiquement satisfaire la condition complémentaire.

La décomposition qu'on va aborder dans ce rapport est la décomposition structure + texture. Dans cette décomposition,  $\bar{u}$  correspond aux basses fréquences, et alors,  $\bar{u}^c$  aux hautes fréquences.

Or, pour mettre la main sur l'aspect fréquentiel de l'image, on doit utiliser la transformation de Fourier discrète (TFD). La figure suivante montre un exemple d'utilisation sur une image qui est une grille :

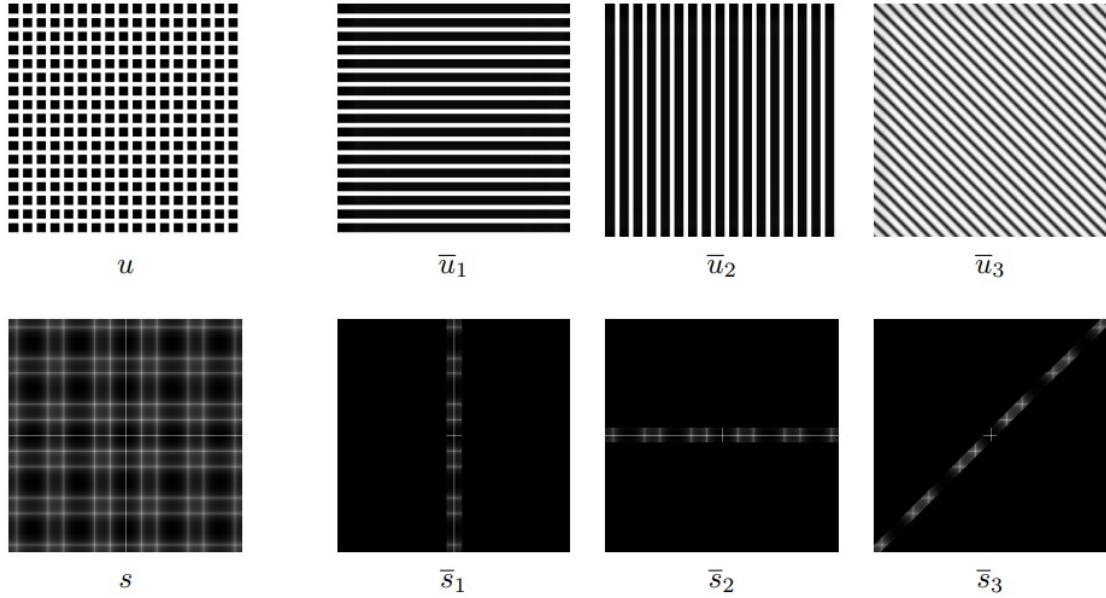


Figure 18: TFD et décomposition sur une grille

Dans l'exemple 1, régler les fréquences  $f_x$  (les fréquences dans l'axe des ordonnées) élevées sur 0 a pour effet de conserver que les rayures horizontales, et dans l'exemple 2 c'est l'inverse. Par conséquent, la direction de la bande conservatrice est Orthogonal à la direction de la partie conservée du spectre. Ceci explique le sens des rayures de l'exemple 3.

En premier temps, nous allons faire la décomposition d'une image par modification de son spectre, ceci dit, on va conserver que les fréquences qui sont situés à une distance  $\eta$  fixée de l'origine. La figure ci-après montre la décomposition faite avec  $\eta$  fixée à 0.25 :

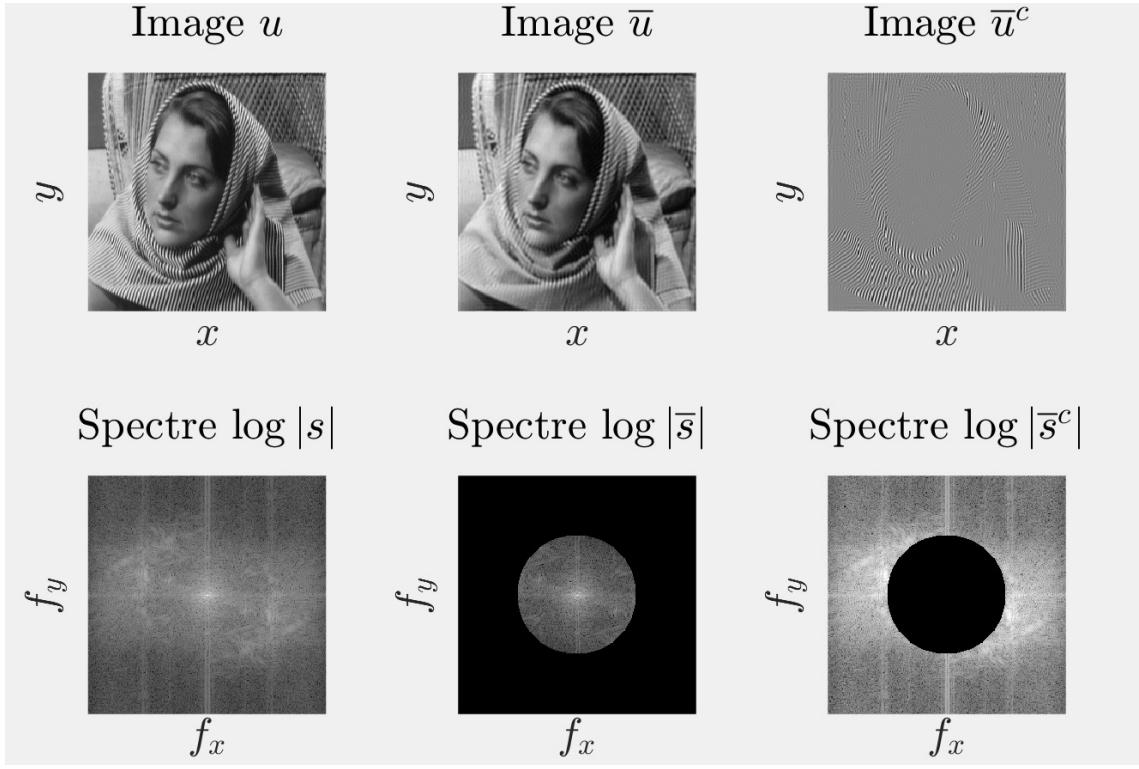


Figure 19: Décomposition Barbara

D'après les résultats expérimentaux, une valeur plus petite de  $\eta$  accordera plus d'importance aux basses fréquences et par suite donnera une structure plus nette, tandis qu'une valeur plus grande de  $\eta$  accordera de plus en plus l'importance aux hautes fréquences et par suite donnera une transition plus douce entre la structure et la texture.

Maintenant on va utiliser une autre approche : l'approche variationnelle, qui ne fait pas nécessairement recourt à la TFD. Et afin d'avoir des variations de niveaux de gris dans  $\bar{u}$  qui sont moins brutale que dans  $u$ , nous avons opté pour le modèle ROF (proposé par Rudin, Osher et Fatemi en 1992), qui fait apparaître la variation totale (la même variation totale utilisé en débruitage par variation total au début de ce rapport). La figure ci-après montre les résultats de l'exécution sur 2 images différentes :

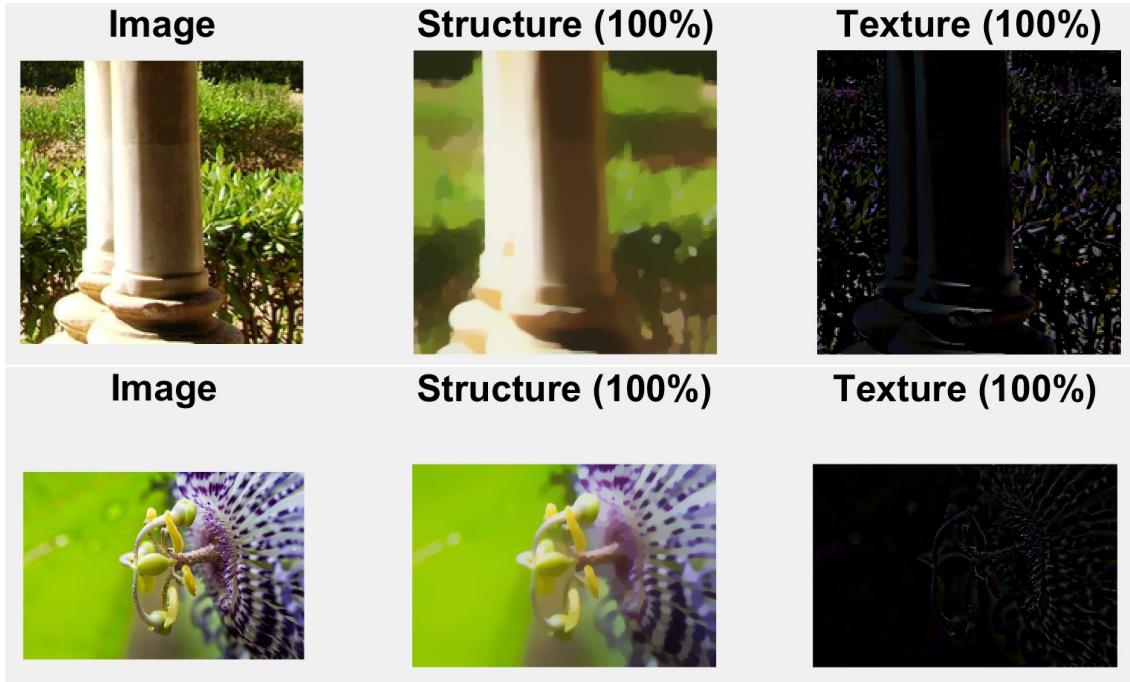


Figure 20: Décomposition ROF

On constate bien que l'exécution pour la deuxième était plus lente par rapport à la première puisque sa taille est plus grande.

Enfin, et puisqu'on a fait 2 méthodes de décomposition, ça sera trop simple d'introduire une troisième, mais non ! On ne va pas faire ça, par contre, on va jouer le chef de cuisine et mélanger les deux méthodes précédentes et les mettre au four à 180 degré, après 30 min on sortira le modèle TV-Hilbert. Ce modèle est un mixe des deux modèles précédents, L'algorithme est relativement plus lente par rapport aux 2 précédentes mais son résultat est relativement mieux ("relativement" parce qu'il faut avoir un temps d'exécution, si ça prend autant de temps avec une petite image on sera pas en mesure de l'exécuter sur une grande base de données)



Figure 21: Décomposition TV-Hilbert

Cette décomposition structure-texture peut-être utilisé en :

- Amélioration de la qualité de l'image : On peut appliquer des traitements spécifiques à chaque composante pour améliorer la qualité de l'image. Réduire le bruit par exemple

(débruitage vu précédemment).

- Compression d'image : on peut compresser efficacement les images en séparant la structure et la texture, en utilisant des méthodes de compression sans perte pour la structure, tandis que la texture peut être compressée avec des méthodes de compression avec perte qui exploitent sa redondance.
- Reconnaissance d'objets : On peut extraire des caractéristiques distinctives des objets dans une image. Ces caractéristiques peuvent ensuite être utilisées pour la reconnaissance et la classification d'objets (par la structure, on peut détecter et extraire les contours des objets).

Enfin, j'ai décidé d'améliorer la qualité d'une image à l'aide de cet algorithme, j'ai pris l'image du camera man du TP5 et j'ai appliqué l'algorithme de décomposition structure-texture suivant le modèle TV-Hilbert. J'ai ajouté à la fin du code du exercice\_3 un filtrage de la partie texture pour réduire le bruit et améliorer sa qualité (puisque c'est la partie texture qui contient le bruit) à l'aide de la fonction medfilt2 de matlab et ainsi j'ai sommé le résultat avec la partie structure, et j'ai eu le résultat suivant :

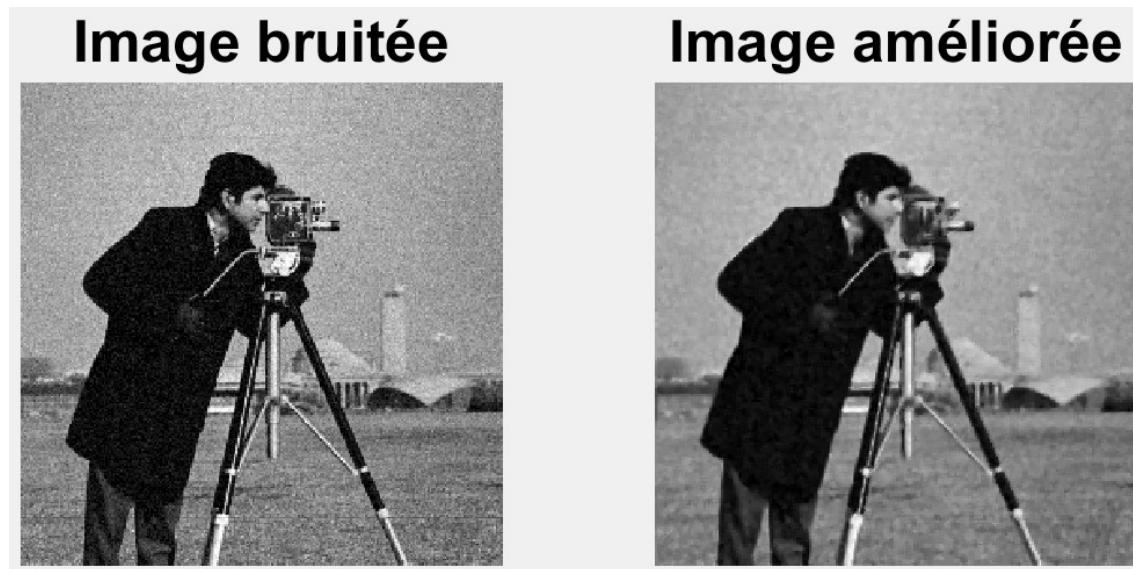


Figure 22: Amélioration de la qualité de l'image

Il y a encore du flou dans mon résultat mais avec les bonnes modifications, je pense que cette décomposition servira de méthode de débruitage de bonne calibre.

### 3 Transformations vocales

Lors de cette deuxième partie de ce rapport, nous allons parler de traitement des données auditaires. J'ai particulièrement choisi shazam et le séparation de source : le premier, je me demandais toujours comment l'application shazam (de base, loin du deep learning) marche, comment on peut reconnaître une musique à partir d'un extrait puisque je n'imaginais pas qu'elle prenne l'extrait et elle essaie de l'identifier avec tous les chansons par terre (ça sera con et trop coûteux), et le deuxième parce que dans quelques moments de ma vie, j'étais censé diviser le rythme des paroles dans quelques enregistrements musicaux, alors je vais comprendre la logique derrière.

### 3.1 Reconnaissance musicale : Shazam

De base, pour arriver à faire la reconnaissance musicale, il faut définir une "empreinte" qui pourra identifier chaque enregistrement musical. Cette empreinte doit vérifier certaines caractéristiques :

- Identifiable à partir d'un petit extrait de l'enregistrement.
- Robuste au bruit (le voix en vie réelle ne sera quasiment jamais nette).
- De taille réduite afin de faciliter le stockage.
- facile à calculer et à comparer pour faciliter la recherche dans la grande bibliothèque de tous les enregistrements musicaux.

D'abord, un sonogramme (dans le cas sonore et spectrogramme dans le cas générale) est 10 fois le log de base 10 de la transformation de Fourier à court terme du signal auditoire.

En effet, l'idée sur laquelle se fonde Shazam est d'utiliser les pics spectraux (maxima locaux du spectrogramme) pour créer une empreinte sonore propre à chaque enregistrement musical. Par expérience, il s'avère que les pics spectraux les plus robustes se situent dans les basses fréquences, alors nos échantillons nos enregistrements sont ré-échantillonnes à  $f_e = 8 \text{ kHz}$ .

Voilà l'exemple d'une empreinte d'un enregistrement musicale :

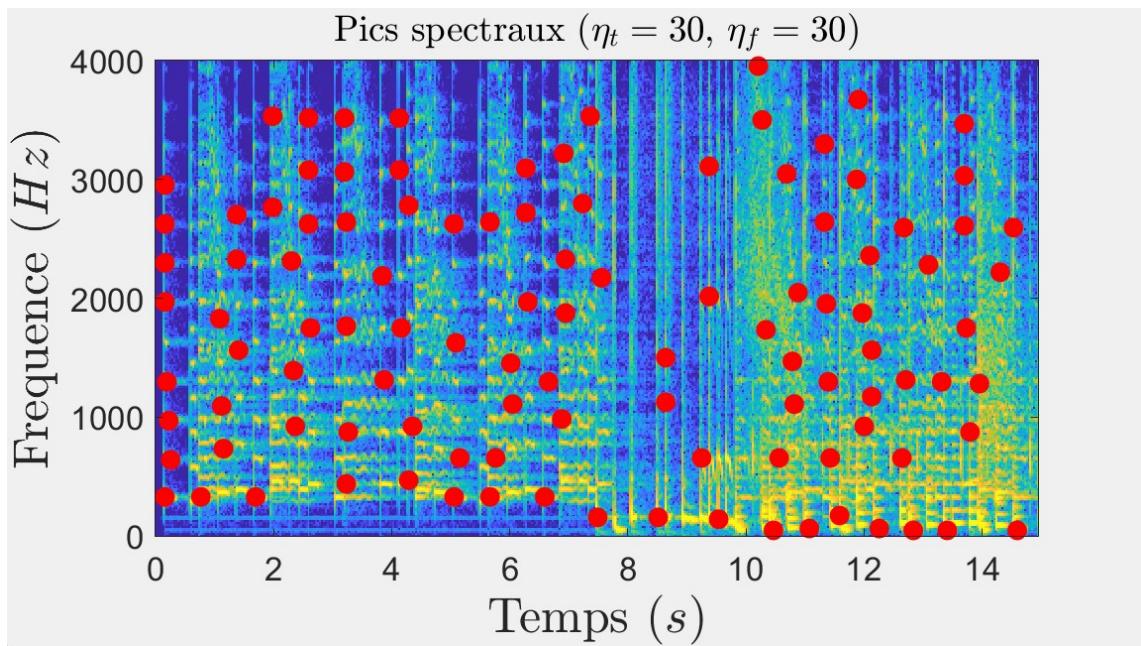


Figure 23: Pics spectraux d'un enregistrement d'une durée de 15 s

Mais, avec ces pics spectraux, l'indexation n'est pas rapide et la recherche n'est pas rapide. Pour remédier à ce problème, shazam parie chaque pics spectraux avec les  $n_v$  pics les plus proches temporellement et fréquentiellement (qui vérifient alors certaines critères). Dans ce tp, on a fixé  $n_v$  à 5, pour que ça soit raisonnable en mémoire et en temps d'exécution et bénéfique pour l'algorithme (c'est la raison principale). Et ainsi, on a le résultat suivant :

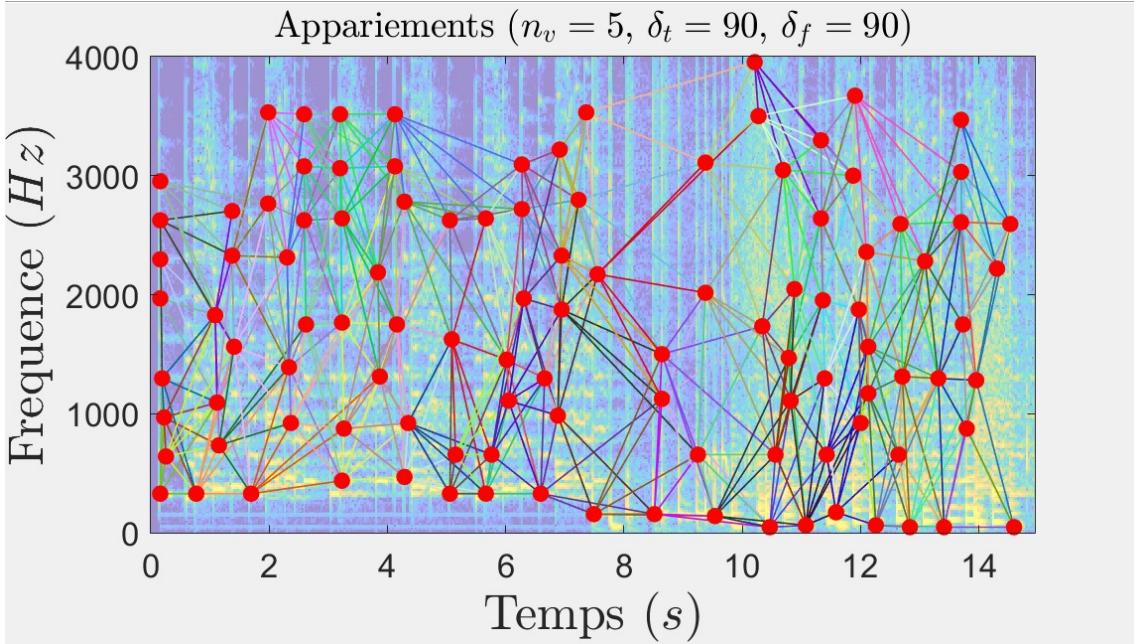


Figure 24: Appariement des pics spectraux

Maintenant qu'on a l'élément principale dans la reconnaissance musicale, on attaque le sérieux. On crée une base de donnée qui contient les empreintes de nos enregistrements musicaux dans un containers.Map et on recherche notre échantillon dans cette base de donnée. Ainsi, nous avons écrit une fonction qui reçoit en entrée la liste des paires de pics de l'extrait et renvoie, pour chaque occurrence dans la base de données de chacune de ces paires, le numéro du morceau associé et par la suite, notre morceau est celui avec plus d'occurrences.

On passe au phase de test, une exécution dans des conditions parfaites donne une précision de 92.0 %, normalement c'est bon parce que la plupart des algorithmes de reconnaissance n'arrive même pas à ce pourcentage, mais c'est décevant pour un shazam, et ce qui est encore plus décevant est qu'une exécution parfaite n'existe pas dans la vie réelle, alors quand j'ai ajouté bruit de parole le pourcentage est devenu 83 %, et quand j'ai ajouté un bruit blanc elle est devenue 67 %, et encore quand j'ai combiné les deux le pourcentage des chansons bien reconnues est devenu 49 %, ça veut dire qu'on a une chance sur 2 d'avoir notre chanson si on exécute cette algorithme en vie réelle, ce qui n'est pas tout bon.

Alors d'ici, il faut attaquer la partie facultatif, où on va améliorer notre reconnaissance musicale.

En effet, cette partie prend en considération que si un extrait débute à l'instant  $t$ , relativement au début du morceau, alors tout pic  $(t_i, f_i)$  de l'extrait doit apparaître dans le morceau à l'instant  $t_i + t$ , donc on stocke le numéro du morceau et la différence temporelle entre l'apparition de la paire dans le morceau intégral et son apparition dans l'extrait.

Maintenant la phase de test : le pourcentage des chansons bien reconnues dans des conditions parfaites est 100 %, même si on ajoute un bruit de parole ça reste à 100 %. Et à l'ajout d'un bruit gaussien et un bruit de parole, on est encore aux environs de 88 % (en moyenne, parce que le maximum que j'ai eu était 95 % et le minimum 82 %). Quel résultat qu'on a obtenu à partir de cette petite rectification !

Ainsi, je ne pense pas qu'on peut arriver à 100 % avec un bruit gaussien en terrain parce que même shazam quand on lui fait écouter un extrait qu'on peut entendre mais il y a trop de bruit

autour, il ne va pas distinguer la chanson des klaxons voitures.

Enfin, je me suis amusé à construire ma propre bibliothèque (cliquez ici pour la télécharger, bien sûr, j'ai écrit un script matlab qui ré-échantillonne les chansons à 8 khz) où j'ai mis que des chansons d'une seule artiste, et les durées de ces chansons sont différents (de 2 minutes à 7 minutes). Mais les résultats étaient encore mieux qu'auparavant, sans bruit : 100 %, avec bruit de parole : encore 100 %, avec bruit de parole et bruit blanc : entre 94 % et 99 %.

### 3.2 Séparation de sources

La séparation de sources, comme son nom l'indique, consiste à la récupération d'un instrument à partir d'un enregistrement musicale (ou même le vocal). En premier temps, nous allons attaquer la séparation harmonique/percussive, elle consiste à séparer les composantes harmoniques (voix, instruments mélodiques) des composantes percussives (batterie) tout en se basant sur le fait que le sonogramme d'une composante harmonique fait apparaître des lignes horizontales (il varie au cours du temps) tand que le sonogramme d'une composante percussive fait apparaître des lignes verticales (elle est riche en fréquence). D'abord, on appliquera un filtrage pour renforcer la composante voulue et on essaie de l'extraire avec un masque. La figure suivante montre les résultats pour  $n1=n2=17$

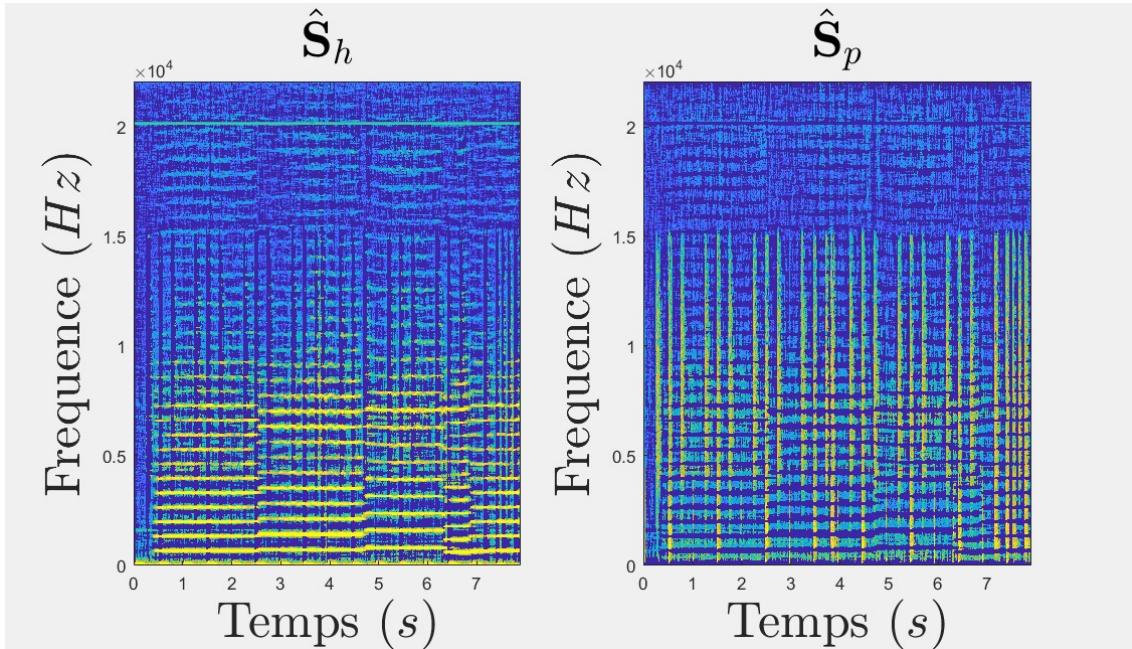


Figure 25: Sonagrammes des deux sources

Modifier  $n1$  et  $n2$  n'affecte pas trop les fréquences mais si on augmente  $n1$  ou  $n2$  considérablement ça va augmenter le bruit du fond et la couleur bleu en arrière-plan va être plus foncée et la séparation ne va pas être le plus net possible et l'inverse si on les diminue considérablement avec une séparation qui n'est pas bonne aussi (ci-joint quelques exemples des résultats qui sont nommés de la forme  $n1\_n2$ ). On pourra dire que  $n1 = n2 = 17$  est la position parfaite pour achever la bonne séparation dans notre cas. Certes, cette méthode ne marche que dans des conditions bien précise, on essaiera alors une autre approche, qui s'appelle : décomposition par NMF, c'est une méthode itératif qui sert à écrire le sonogramme sous forme d'un produit de 2 matrices dont l'une est du rang  $r$  tel que  $r$  est le nombre de sources qu'on veut séparer. La figure ci-après montre les résultats obtenus pour  $r=4$  :

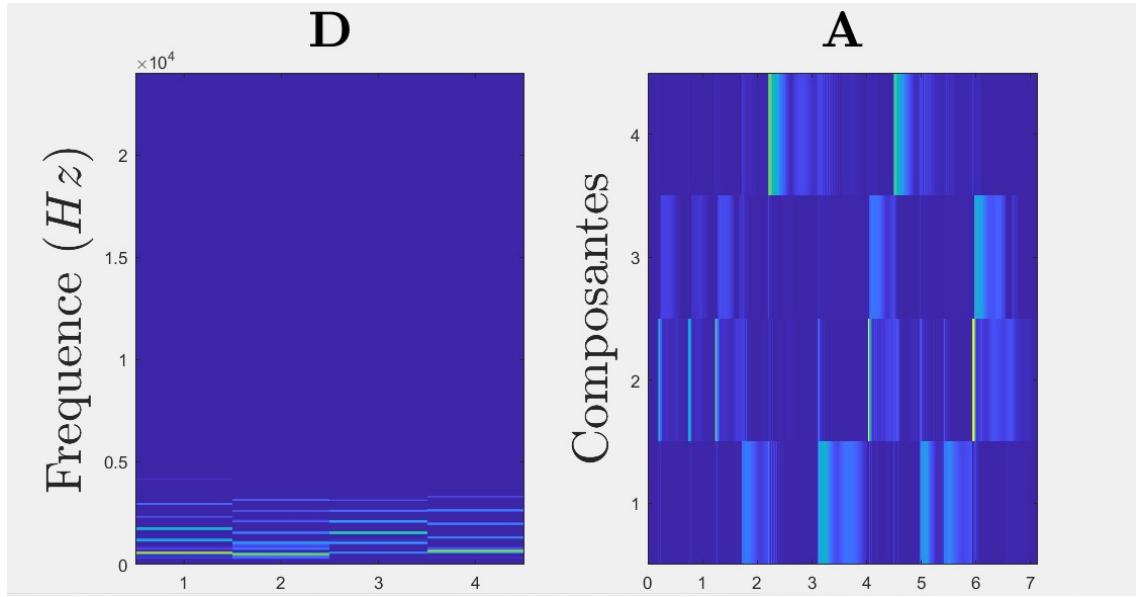


Figure 26: Dictionnaire et activations en sortie du NMF

Si on modifie  $r$  pour qu'il soit plus grand du nombre des composantes, on aura des composantes redondantes, et si on fait l'inverse on aura des composantes qui ne sont pas assez bien séparé (Vous trouvez les fichiers illustrants ces faits joint à ce rapport). Or, même cette méthode n'est pas la solution parfaite pour achever une bonne séparation de sources alors on fait appel au deep learning. Voilà le réseau qu'on a implémenté :

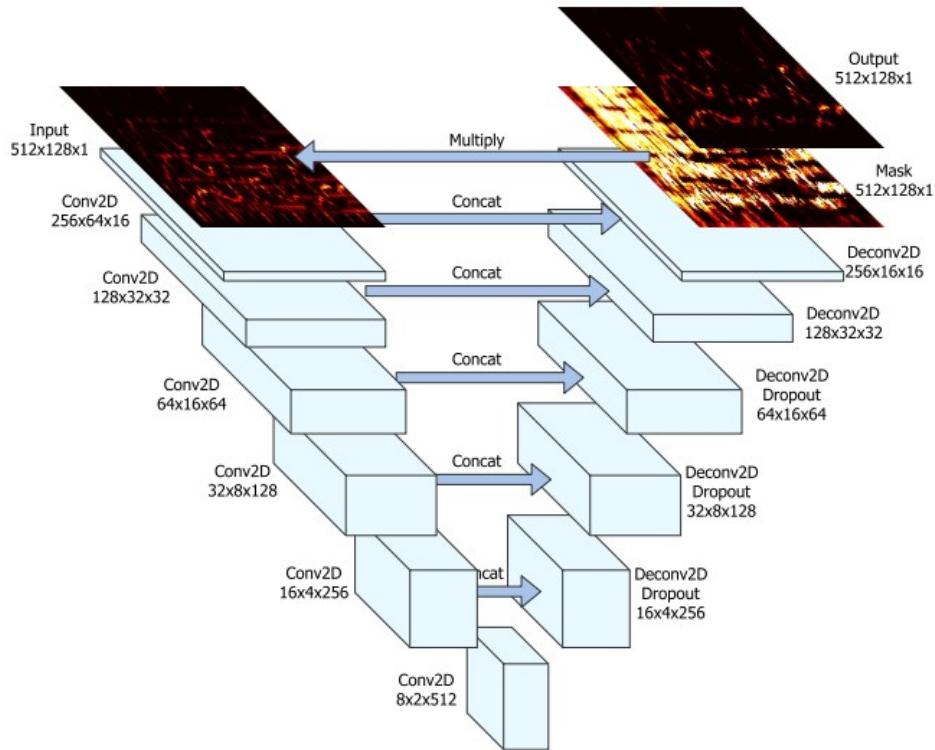


Figure 27: Architecture du réseau

En vrai, les résultats n'étaient pas trop satisfaisantes, alors j'ai opté pour quelques modifications : - Dans la fonctions getitem de DataGenerator j'ai ajouté la ligne :  
self.x = tf.keras.layers.Dropout(0.2)(self.x), alors, pendant l'apprentissage, ça fait des pauses de temps en temps pour que le réseau ne s'habitue pas sur une sorte de modèle. J'ai utilisé aussi la fonction ReduceLROnPlateau pour réduire automatiquement le taux d'apprentissage lorsque la perte sur l'ensemble de validation cesse de s'améliorer ainsi que earlyStopping qui arrête l'entraînement lorsque la performance du modèle sur un ensemble de validation ne s'améliore plus (pour des raisons liées au temps d'entraînement lors de mes tests). Aussi, j'ai modifié la fonction de perte de "mae" en "mse" (j'ai essayé aussi "binarycrossentropy" mais elle n'avait pas l'air de trop marcher). Enfin, j'ai changé le batch, de 20 à 16 (après de nombreuses expériences).

Malgré toutes ces expériences et ces modifications, je n'ai pas arrivé à avoir une bonne séparation pour toute les enregistrements musicaux, ça marche pour certaines (vous trouverez ci-joint un exemple), mais ça ne marche pas pour les autres et parfois ça donne même des trop mauvais résultats.

En gros, les performances d'un système en deep learning sont fortement liés à sa base de données et à ses hyperparamètres, et pour tomber sur les bons paramètres, ça n'a pas l'air trop évident. L'autre problème que j'ai remarqué, est que les vocals de la base de données ne sont pas parfaites et même parfois abîmés par un son de guitare bas en arrière-plan.