CentraleSupélec

2 ème année – Cursus ingénieur

PROJET S8 :

Construction de base de données de spectométrie de masse et débruitage de spectres par Deep Learning

Étudiants:
Hugo GINOUX
Aymane HANINE
Abdessamad LACHGAR
Tiémoko DEMBELE

Encadrants:
Emilie CHOUZENOUX
Mouna GHARBI

June 24, 2023

Contents

1 Abstract

La spectrométrie de masse est une technique de physique permettant d'identifier des molécules, grâce à des spectres caractéristiques de certaines structures chimiques. Toutefois, les spectres obtenus durant les expériences sont souvent très bruités, ce qui rend l'identification des composants chimiques difficile. D'autre part, plusieurs applications basées sur l'apprentissage automatique (classification de spectres, identification de composants...) nécessitent de grandes bases de données réalistes de spectres pour réaliser l'apprentissage et donner des résultats efficaces. Ce travail a donc deux objectifs principaux : proposer des méthodes de création de bases de données réalistes et débruiter des spectres grâce à une technique de deep learning.
Nous avons tout d'abord obtenu des spectres par 3 techniques différentes (banque de données du projet européen MassBank, bibliothèque Pyisopach et génération aléatoire). Après la création de trois bases de données réalistes, nous avons proposé une architecture de réseau de neurones appelée autoencodeur que nous avons entraînée et optimisée pour débruiter des signaux. Enfin, nous avons construit un site web dans le but que ce travail soit accessible à tous.
L'ensemble de nos codes sont disponibles sur ce Gitlab : `https://gitlab.com/Dembelet/spectrodl-jupyternotebooks`.

2 Introduction

2.1 Problème Initial

2.1.1 Fonctionnement de la spectrométrie de masse

La spectrométrie de masse permet d'obtenir, à partir d'un composé chimique inconnu, un spectre qui comporte des pics caractéristiques de certaines structures basiques. À partir de ce spectrogramme, on peut en déduire la formule du composé chimique initial [4].

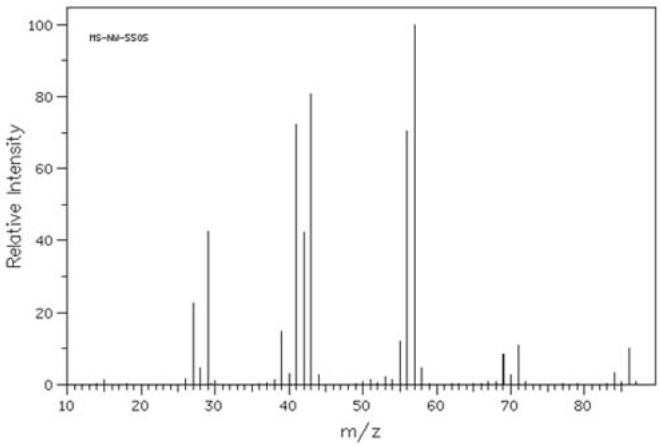


Figure 1: Exemple de spectre de masse

Le fonctionnement d'un spectromètre est le suivant :

- Une source d'ionisation : le composé initial est ionisé, c'est à dire vaporisé en plus petits composants chargés électriquement
- Un analyseur : il sépare les ions obtenus selon leur masse réduite m/Z
- Un détecteur : il transforme les ions en courant électrique, ce qui permet d'obtenir un spectrogramme après traitement

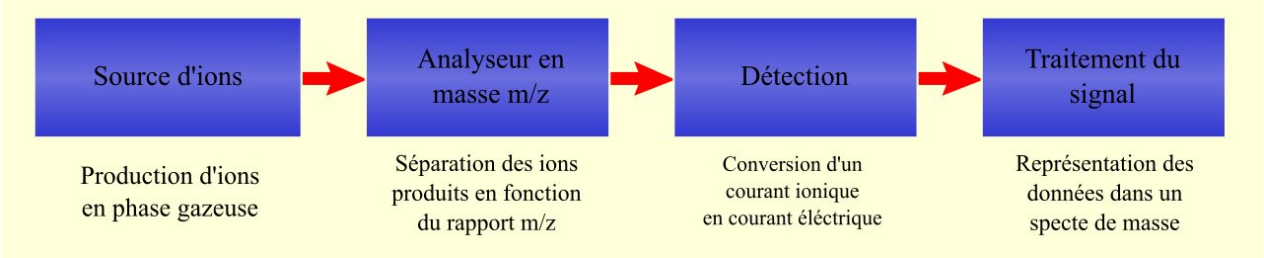
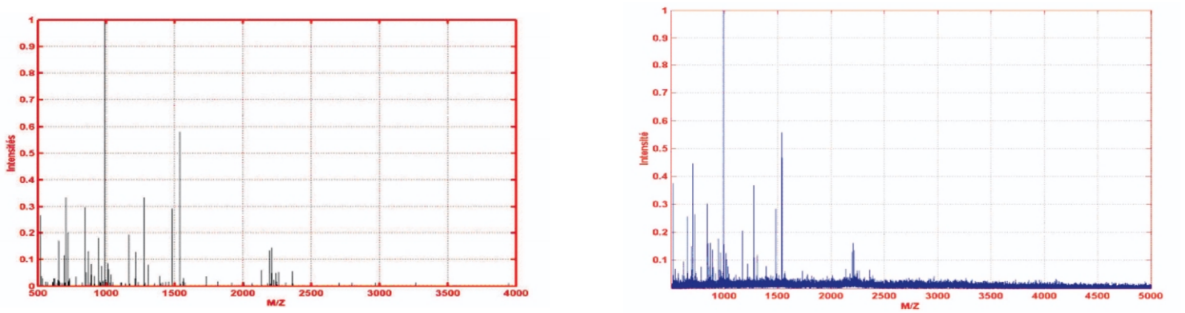


Figure 2: Strucure d'un spectromètre, d'après Wikipédia [5]

2.1.2 Le problème du bruit

Dans la pratique, les spectres obtenus en sortie du spectromètre sont souvent assez bruités, ce qui rend difficile l'identification des molécules. En effet, il est très difficile de dire si un pic est dû à la présence d'un groupe caractéristique, ou bien à du bruit de mesure. Pour illustrer ce phénomène, voici les spectres réel et mesuré de la même molécule :



(a) Réel

(b) Mesuré

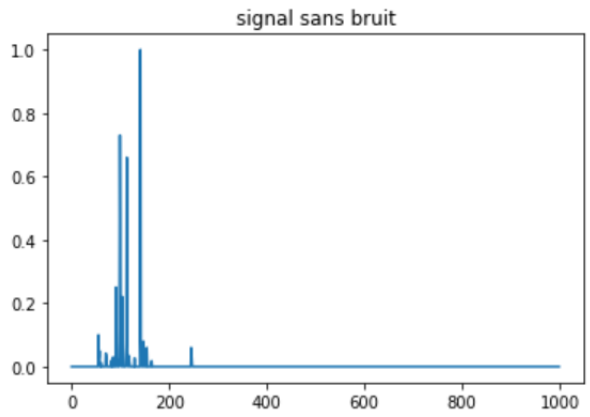
Figure 3: Spectres de la protéine Acyl-CoA

Il existe diverses méthodes pour débruiter numériquement ces spectres. Nous allons survoler les techniques classiques principales, à base de seuillages et d'inversion de matrices. Puis nous développerons notre propre stratégie de débruitage, **fondée sur le principe du deep learning**.

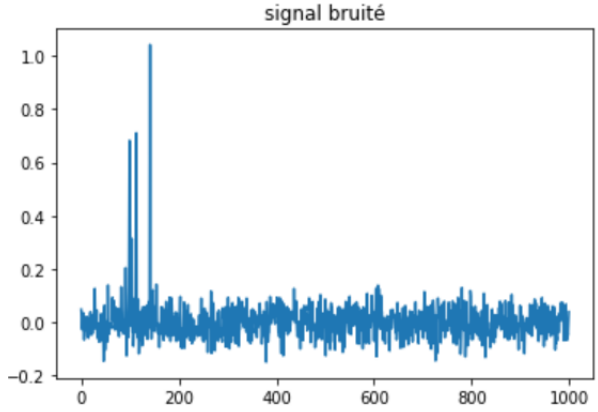
2.2 Etat de l'art

2.2.1 Seuillage naïf

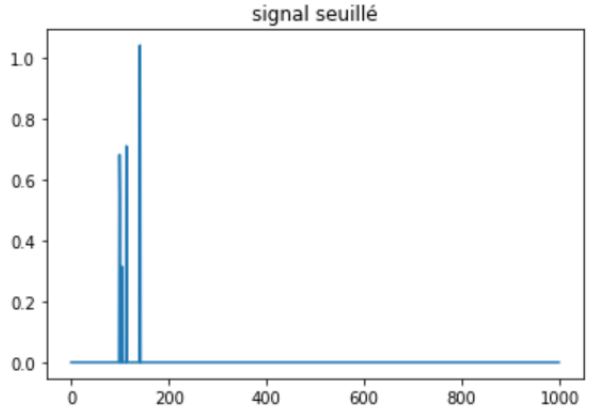
Pour débruiter un signal, l'idée la plus naturelle serait tout simplement d'appliquer un seuillage : les points en dessous d'un certain seuil seraient tout simplement mis à la valeur 0.



(a) Signal sans bruit



(b) Signal bruité



(c) Signal seuillé

Cette méthode a l'avantage d'être simple, rapide et facilement interprétable. Néanmoins, elle présente plusieurs inconvénients :

- Les pics inférieurs au seuil sont forcément effacés
- Le seuil peut être difficile à fixer : trop élevé on perd trop d'informations, trop faible le signal n'est pas assez débruité

2.2.2 Débruitage par ondelettes

Un spectre de masse typique est constitué de l'enregistrement séquentiel du nombre d'ions qui arrivent sur le détecteur avec les valeurs correspondantes de leurs valeurs m/z.

Des pics (maximum locaux du spectre observé) d'intensité d'amplitude variable correspondent aux diverses protéines

Le principe sous-jacent pour ce type de débruitage est d'tiliser la transformée en ondelettes invariante par translation pour isoler la composante du bruit:

- transformer le signal du domaine temporel dans le domaine des coefficients d'ondelettes par la transformation
- éliminer les coefficients en dessous d'un certain seuil
- faire ensuite la transformation inverse.

Le bruit affecte de manière égale tous les coefficients des résolutions grossières, alors que le vrai signal des pics est caractérisé par peu de grand coefficients aux résolutions fines.

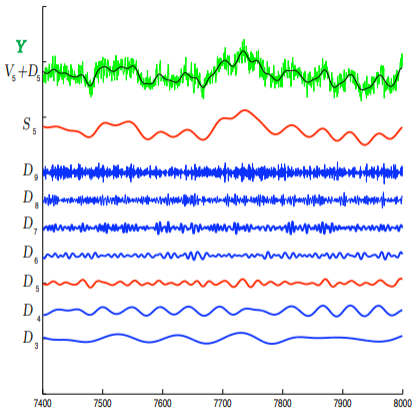


Figure 5: Décompositions à base d'ondelettes

2.2.3 Seuillage flou multi-échelle

Dans le cadre de la caractérisation de protéines pour la génomique, une équipe de chercheurs a développé en 2005 une technique innovante pour distinguer les pics "parasites" des pics "utiles", et ainsi permettre un seuillage plus efficace [3]. Cette technique s'effectue en trois étapes :

- Séparation du spectre en deux : les basses fréquences (que du contenu intéressant) et les hautes fréquences (des pics "parasites" et des pics "utiles") , grâce à un filtrage par ondelettes.
- Pour les basses fréquences, on détermine quels sont les pics informatifs en optimisant l'entropie de Shannon. On amplifie les pics jugés "utiles".
- On reconstruit le spectre en sommant les hautes fréquences et les basses fréquences, puis en repassant dans le domaine temporel.

L'approche est intéressante constitue une amélioration notable de la technique de seuillage, et a permis d'identifier la protéine recherchée chez d'autres espèces que l'homme.

2.2.4 Approche par dictionnaire

Une autre technique plus mathématique, développée par Afef Cherni, Émilie Chouzenoux et Marc-André Delsuc consiste à inverser une matrice de grande taille [1].
En effet, sous certaines hypothèses, le signal mesuré $\mathbf{y}$ et le signal $\bar{\mathbf{x}}$ contenant des informations sur les isotopes présents dans le mélange sont reliés par l'équation

$$\mathbf{y} = \mathbf{D}\bar{\mathbf{x}} + \mathbf{n}$$

Où $\mathbf{D}$ est une matrice contenant des informations sur $d(m, z)$ la distribution isotopique d'une molécule de masse m et de charge z, calculée à partir de modèle "averagine" [2]. L'inversion de $\mathbf{D}$ n'est pas possible directement, car elle est de taille $M \times T$ où M est le nombre de valeurs que la masse m peut prendre - rapidement plusieurs milliers pour une grosse molécule- et $T = MZ$ où Z est le nombre de valeurs possibles pour la charge. Il faut donc passer par des méthodes numériques pour approcher une valeur $\mathbf{x}$ proche de $\bar{\mathbf{x}}$, ce qui est fait dans l'article par optimisation convexe.

2.3 Notre démarche : le Deep Learning

Notre méthode de débruitage par deep learning se base sur un modèle historiquement inspiré du fonctionnement d'un cerveau humain appelé "réseau de neurones". Il est constitué d'un certain nombre de variables appelées "neurones", organisées en couches connectées les unes aux autres. Le fonctionnement se fait en plusieurs étapes :

- Le signal d'entrée, bruité, est échantillonné pour pouvoir être considéré comme un vecteur x. Pour toute la suite, nous considérerons un échantillonnage de 1001 points, entre $m_{min} = 0$ et $m_{max} = 1000$.
- Chaque point x_i active les neurones d'entrée. Les neurones de la couche suivante sont connectés à la couche d'entrée via des poids qui sont initialisés aléatoirement. Pour rendre le fonctionnement global non linéaire, on compose par une *fonction d'activation* (par ex : *relu*, *tanh*, *sigmoid*...)
- De proche en proche, chaque couche est activée par la précédente en fonction de l'état de ses poids et de sa fonction d'activation. Finalement, la couche de sortie (qui contient autant de neurones que la couche d'entrée) s'active à son tour, formant un signal de sortie $\hat{y}$.
- Le signal de sortie est comparé au signal réel sans bruit y, et les poids sont actualisés via une méthode dite de "backpropagration" du gradient afin de minimiser une certaine quantité $L(\hat{y}, y)$ caractéristique de la "distance" entre $\hat{y}$ et y.

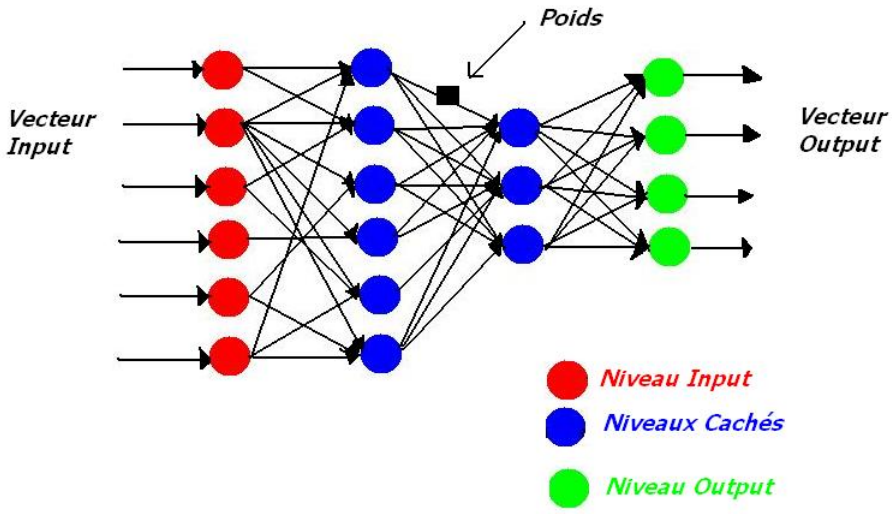


Figure 6: Schéma d'un réseau de neurones générique

L'objectif est qu'après avoir vu beaucoup de spectres bruités et débruités, le réseau de neurones ait "appris" la configuration de poids qui lui permette de retirer le bruit par lui-même.

3 Obtention des données

Les algorithmes de deep learning sont souvent très performants, mais très gourmands en données d'entraînement. La première étape du projet est donc de trouver des sources de spectres de masse et de proposer des méthodes de construction de base de donnée réalistes pour pouvoir entraîner correctement notre algorithme.

3.1 MassBank

Les premières données exploitables que nous avons trouvées proviennent de ce site web : `https://massbank.eu/MassBank/Search`. Il s'agit d'un écosystème open source, développé par différents groupes de recherches en Allemagne et au Luxembourg. La base de données contient 29098 spectres de molécules connues.

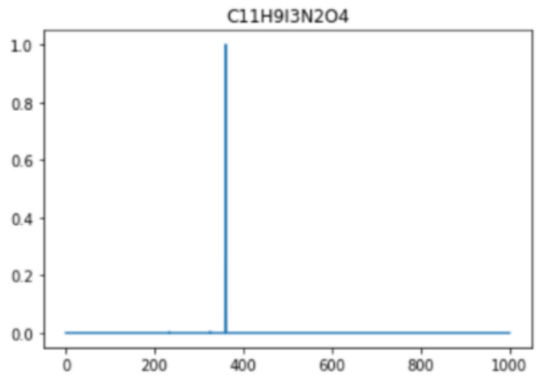
Nous avons téléchargé le dataset complet sous la forme d'un fichier csv qui contient 4 colonnes : la formule chimique de la molécule, les abscisses des pics, l'intensité brute de chaque pic et l'intensité relative (le pic le plus faible étant mis à la valeur 1).

```
MSdata = pd.read_csv('MSdata.csv', encoding='mac_roman')
MSdata.head()
```

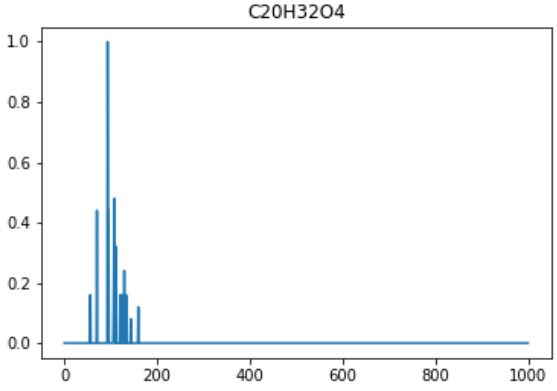
	Formule chimique	peak	intensitÈ	intensitÈ relative
0	C27H39NO2	['84.1', '105.1', '107.1', '114.1', '115.1', '...	['12461', '2208', '2394', '40390', '2816', '31...	['32', '6', '6', '105', '7', '8', '6', '82', '...
1	C27H41NO2	['67.1', '81.1', '84.1', '85.1', '93.1', '96.1...	['996', '1626', '3012', '2434', '1034', '1039'...	['29', '47', '87', '70', '30', '30', '67', '52...
2	C33H49NO7	['84.1', '85.1', '87.0', '97.0', '99.0', '102....	['8514', '1990', '1086', '907', '621', '1378',...	['161', '38', '21', '17', '12', '26', '20', '2...
3	C33H51NO7	['67.1', '81.1', '84.1', '85.0', '87.0', '96.1...	['450', '620', '2636', '1396', '607', '476', '...	['21', '29', '123', '65', '28', '22', '34', '2...
4	C20H32O3	['123.120', '135.200', '139.200', '145.180', '...	['12500.0', '29166.7', '29166.7', '16666.7', '...	['1', '3', '3', '2', '1', '39', '1', '2', '1',...

Figure 7: Lecture du fichier sous python

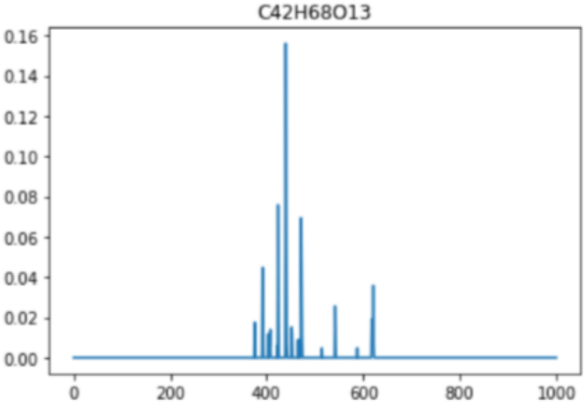
Après un léger traitement des données (str to float, normalisation des pics, troncature des spectres) nous pouvons en visualiser quelques spectres issus de cette base de données :



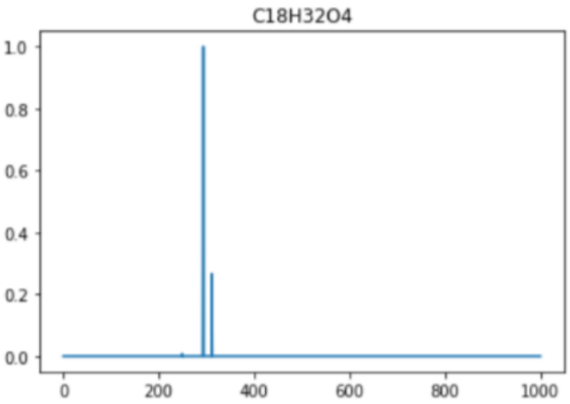
(a) $C_{11}H_9I_3N_2O_4$



(b) $C_{20}H_{32}O_4$



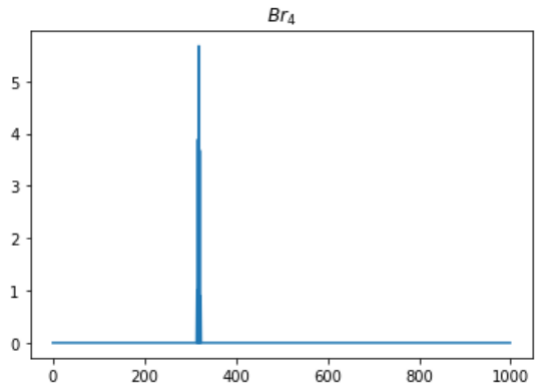
(c) $C_{42}H_{68}O_{13}$



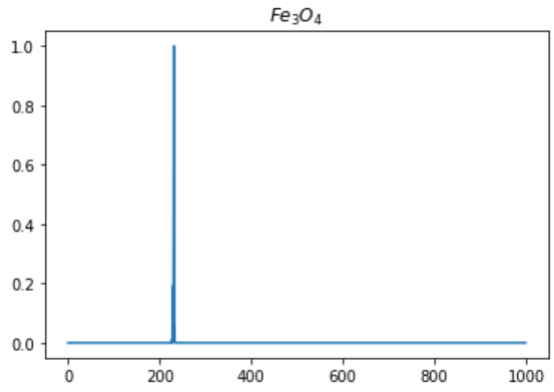
(d) $C_{18}H_{32}O_4$

3.2 Pyisopach

Notre deuxième source de données est le module *pyisopach* de python, dont la documentation se trouve ici : `https://pypi.org/project/pyisopach/`
Ce module permet d'obtenir la masse et la position des pics dans le spectre de masse pour nimporte quelle molécule ! Voici quelques exemples :



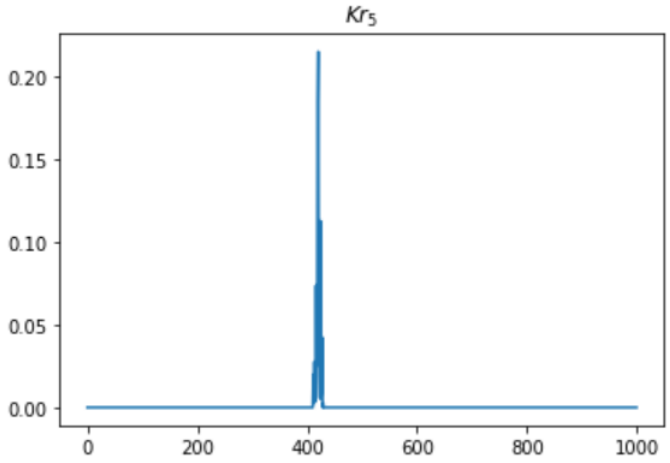
(a) Br_4



(b) Fe_3O_4

Nous pouvons donc obtenir en théorie **une infinité de spectres** représentant des molécules différentes. Toutefois, ce module semble étrange et suspect pour deux raisons :

- Il ne donne pas les mêmes spectres que la source MassBank pour les molécules répertoriées par celle-ci (peut-être à cause des différents réglages et tolérances des appareils de mesure)
- Il est capable de donner des spectres pour nimporte quelle combinaison d'atomes, même si celle-ci n'a aucune existence physique. Par exemple, voici le spectre pour le Kr_5 (qui ne peut pas exister dans la nature puisque le Krypton est un gaz rare) :



Par conséquent, cette source de spectres ne semble pas très fiable. Toutefois, ce n'est pas grave pour notre projet. En effet, pour entraîner notre IA nous n'avons pas besoin d'avoir des spectre **qui existent réellement**, mais il suffit d'avoir des spectres **qui pouraient exister** - ie qui ressemblent à des spectres réels. C'est pourquoi, pour compléter notre base de données, nous pouvons utiliser ce module pour générer des spectres vraisemblables aléatoirement grâce à ce code :

```
import pyisopach
n_c=np.random.randint(1, 20)
n_o=np.random.randint(1, 20)
n_h=np.random.randint(1, 40)

mol = pyisopach.Molecule("C"+str(n_c)+"H"+str(n_h)+"O"+str(n_o))
locs, sizes = mol.isotopic_distribution()
```

3.3 Génération aléatoire

Dans cette partie, nous essayons de générer aléatoirement des graphes qui ressemblent à des spectres de masse de molécules réelles. Pour déterminer quelle fonction produit les spectres les plus proches de la réalité, nous utilisons un petit réseau de neurones classifieur qui essaye de déterminer quels sont les spectres réels. Tout le code pour cette partie se trouve dans le fichier `comparaisons_simulateurs.ipynb`.

3.3.1 Description des fonctions

Nous avons écrit des fonctions python qui génèrent des spectres de masse aléatoirement. Ces fonctions fonctionnent toutes de la cette manière :

- Choix aléatoire du nombre de pics principaux
- Pour chaque pic principal, choix du nombre du nombre de pics annexes et positionnement de ceux-ci

Voici des exemples de spectres générés par ces fonctions :

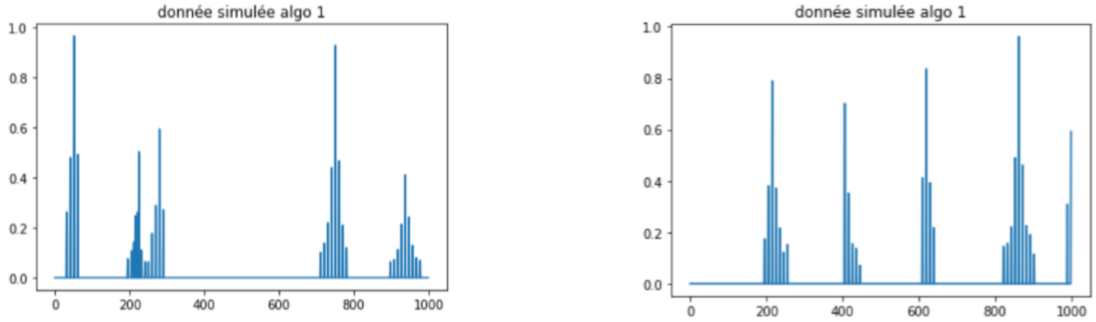


Figure 10: Fonction 1

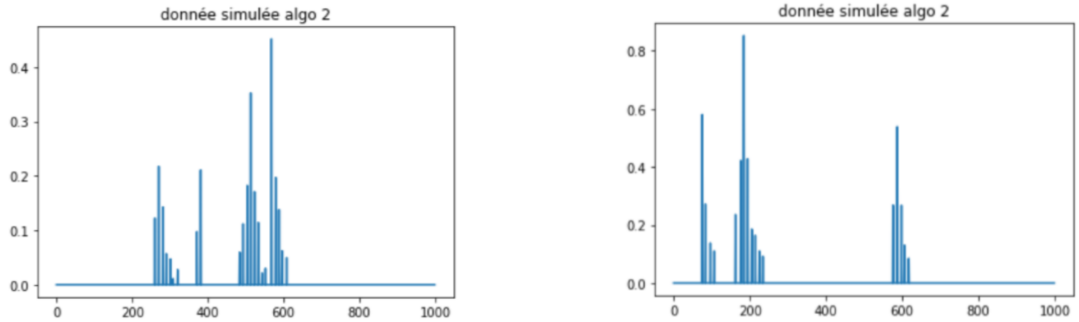


Figure 11: Fonction 2

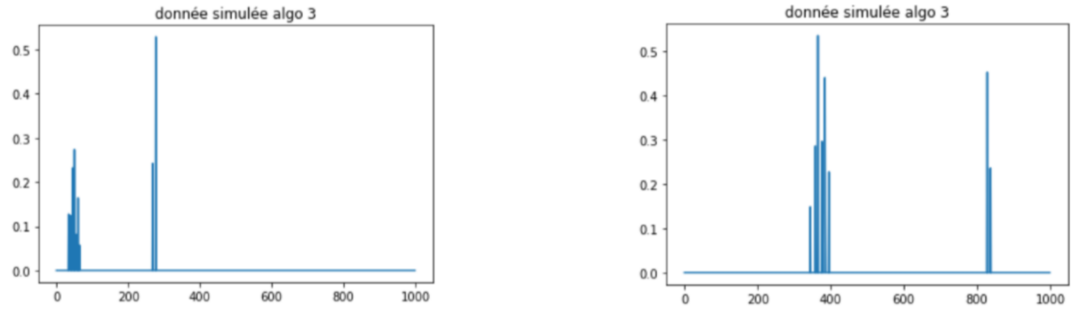


Figure 12: Fonction 3

Nos fonctions diffèrent par les lois de probabilité utilisées pour le nombre et le placement des pics :

	Fonction 1	Fonction 2	Fonction 3
Nombre de pics principaux	$\mathcal{U}([2, 8])$	$\mathcal{U}([1, 6])$	$\mathcal{P}(\lambda = 2)$
Nombre de pics secondaires	$\mathcal{U}([0, 12])$	$\mathcal{U}([0, 10])$	$\mathcal{P}(\lambda = 1)$
Localisation des pics principaux	$\mathcal{U}([0, 1000])$	$\mathcal{E}(\lambda = 500)$	$\mathcal{E}(\lambda = 500)$

3.3.2 Choix de la meilleure fonction

Nous voulons choisir la fonction qui produit des spectres les plus semblables possible à des vrais spectres de masse. Pour ce faire, nous mélangeons des données simulées à des données réelles (issues de MassBank 3.1) et nous entraînons un réseau de neurones classifieur basique (3 couches *Dense*) à distinguer les deux types sur 5 époques. La fonction qui parvient le mieux à "tromper" le réseau de neurones sur des données de test sera la plus efficace !

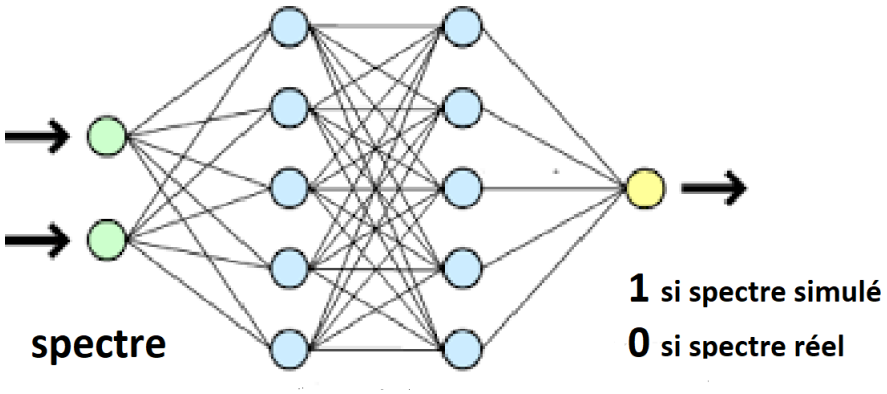


Figure 13: Réseau de neurones pour distinguer les spectres simulés

Voici les résultats pour nos trois fonctions. Comme notre but est de tromper le modèle, on retiendra la fonction qui donne l'accuracy **la plus faible** (idéalement 50%).

```
[28] scores_1 = model_1.evaluate(X_1_test, y_1_test, verbose=0)
     print('Accuracy on test data for algo 1 : {:.4f}'.format(scores_1[1]))

     scores_2 = model_2.evaluate(X_2_test, y_2_test, verbose=0)
     print('Accuracy on test data for algo 2 : {:.4f}'.format(scores_2[1]))

     scores_3 = model_3.evaluate(X_3_test, y_3_test, verbose=0)
     print('Accuracy on test data for algo 3 : {:.4f}'.format(scores_3[1]))
```

```
Accuracy on test data for algo 1 : 0.9972
Accuracy on test data for algo 2 : 0.9966
Accuracy on test data for algo 3 : 0.9962
```

Figure 14: Accuracy de test

C'est très serré, mais la **fonction 3** est celle qui génère les spectres les plus proches de la réalité. C'est donc celle qu'on retiendra par la suite. Les résultats sont toutefois à prendre avec des pincettes, car les écarts sont peu significatifs.

Remarque On aurait pu pousser la recherche plus loin et tester plus de fonctions différentes. Mais la suite du projet montrera que cette fonction est suffisante pour produire des résultats corrects, nous n'avons donc pas jugé utile de poursuivre dans cette voie.

4 Création du réseau de neurones

Une fois armés de ces trois méthodes pour obtenir des données, nous allons créer puis entraîner trois réseaux de neurones débruiteurs, un pour chaque source de données. Nous avons tout fait dans le langage python, en utilisant la bibliothèque *Keras* avec *Tensorflow* en backend. Vous pouvez retrouver le code dans le fichier `projet_mlp.ipynb`.

4.1 Architecture globale

4.1.1 Structure

L'algorithme est un réseau de neurones, comme présenté Figure 22.

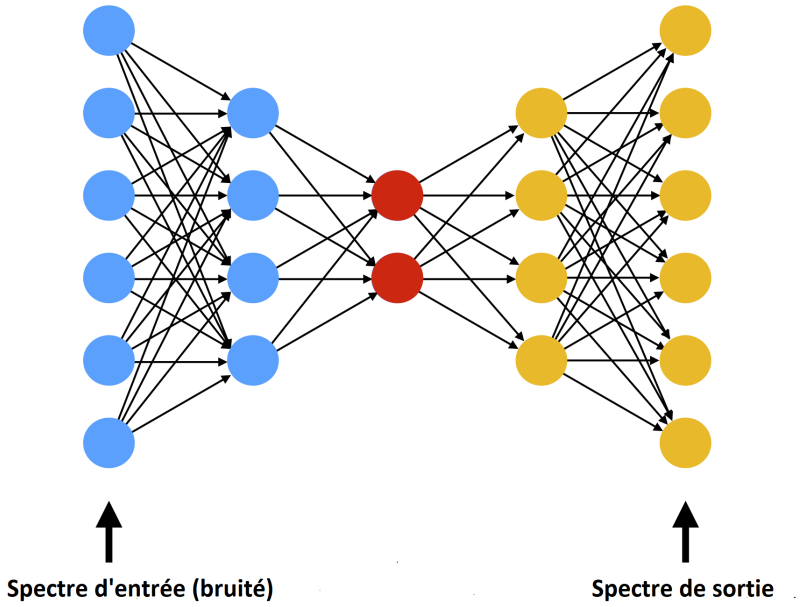


Figure 15: Autoencodeur

Nous avons choisi un **autoencodeur**, c'est-à-dire que les couches intérieures sont plus petites que celles d'entrée et de sortie. L'intérêt est que l'information est "contractée" en plus petite dimension, pour que seuls les motifs importants soient conservés alors que les détails (le bruit) sont ignorés. La dernière couche et de même taille que la première, pour reconstruire le signal débruité avec autant de points que le signal d'entrée.

4.1.2 Couches

Les **couches** que nous avons utilisées sont des couches *Dense*, c'est à dire complètement connectées. Nous avons également tenté des couches à convolution 1D (*Conv1D*) et récurrentes (*LSTM*, *GRU*) réputées être meilleures dans le traitement des séries temporelles, ce qui est un problème qui peut ressembler au nôtre, mais leurs performances étant inférieures à celles des couches *Dense*, nous avons laissé tomber.
Voici ce que cela peut donner sur *Keras*, avec une seule couche intérieure de 500 neurones:

```
model=models.Sequential()
model.add(layers.Dense(500, activation='relu', input_shape=input_shape))
model.add(layers.Dropout(0.2))
model.add(layers.Dense(1))
model.add(layers.Dropout(0.2))
model.add(layers.Flatten())
```

Notez que les *Dropout* retirent les connexions peu utiles et permettent d'éviter l'overfitting.

4.2 Paramètres expérimentaux

4.2.1 Fonction de perte

Pour que le réseau parvienne à optimiser ses poids, il lui faut une quantité à minimiser appelée **loss function**. Nous aurions pu prendre la quantité classique dans ces situations appelée *Mean Squared Error* définie par

$$mse(y, \hat{y}) = \sum_{i=0}^{n} (y_i - \hat{y}_i)^2$$

qui correspond à la somme des écarts au carré entre les points du spectre prédit $\hat{y}$ et les points du spectre réel y.

Cette fonction de perte pourrait tout à fait marcher. Mais notre situation est un petit peu particulière, puisque nous étudions des spectres où 99% des valeurs sont nulles. Pour prendre cela en compte, nous avons décidé d'ajouter à la *mse* un terme qui va pénaliser les spectres avec trop de valeurs non nulles. Avec λ un hyperparamètre, l'expression de la loss devient alors :

$$customloss(y, \hat{y}) = \sum_{i=0}^{n} (y_i - \hat{y}_i)^2 + \lambda \sum_{i=0}^{n} \hat{y}_i^2$$

L'implémentation sur *Keras* donne ceci :

```
import keras.backend as K

def customloss(y_true, y_pred) :
  penalty_coef=1
  mse = K.sum(K.square(y_pred-y_true))
  penalty = K.mean(K.square(y_pred))
  return mse+penalty_coef*penalty
```

4.2.2 Tuning des paramètres

Pour chacun des trois réseaux autoencodeurs (entraînés avec des données différentes), il est possible de modifier 4 hyperparamètres pour améliorer les performances :

- Le nombre de couches intérieures
- Le nombre de neurones dans chaque couche intérieure
- L'optimiseur et les paramètres associés
- Le nombre d'époques

Pour évaluer chaque combinaison d'hyperparamètres, on génère un ensemble de validation issu de la **même méthode que pour produire les données d'entraînement** et regarde la *mse* obtenue avec. Les modèles ont tous été entraînés sur 8000 spectres et évalués sur 2000 autres, issus du même dataset, avec un *batch_size* = 32. Les codes utilisés pour tester ces hyperparamètres se trouvent dans les notebooks `projet_architecture_massbank`, `projet_architecture_pyiso` et `projet_architecture_generate`. Nous avons essayé diverses combinaisons pour ces paramètres, et voici un tableau récapitulatif des meilleurs résultats :

Dataset d'entraînement	Couches	Optimiseur	Nombre d'époques	meilleure *mse*
MassBank	[300, 300, 300, 300]	RMSprop	3	1.3×10^{-4}
Pyisopach	[450, 150, 100]	Adam	3	3.3×10^{-5}
Génération aléatoire	[500, 500, 500]	Adam	5	5.8×10^{-8}

Avec un seuillage raisonnable (*seuil* = 0.3), sur notre ensemble de spectres MassBank (voir section 3.1), on obtient une $\mathbf{mse = 1.7 \times 10^{-4}}$. À première vue, il semblerait donc que ces modèles soient plus efficaces que le seuillage simple. Néanmoins il y a un risque d'overfitting (le modèle s'adapte trop à la méthode de génération des spectres), il faut donc évaluer les modèles sur des spectres réels.

4.3 Evaluation des modèles et résultats expérimentaux

4.3.1 Résultats sans renormalisation des signaux

Pour évaluer les performances des 3 modèles obtenus sur des vrais spectres, nous générons un ensemble de test (X_{test}, Y_{test}) provenant de la base de données MassBank (pour avoir des spectres réels). Nous regardons ensuite la *mse* obtenue grâce à cette ligne de commande :

```
loss, mse = model.evaluate(X_test.reshape(X_test, Y_test, batch_size=16)
```

Le tableau suivant récapitule les résultats :

Dataset d'entraînement	*mse* sur des vrais spectres
MassBank	1.5×10^{-4}
Pyisopach	1.7×10^{-4}
Génération aléatoire	1.7×10^{-4}

Le modèle le plus performant est celui entraîné sur les données de MassBank, ce qui est logique puisque l'ensemble de test provient de la même source.
On remarque que pour tous nos modèles, la *mse* est très proche de celle obtenue par un simple seuillage (1.7×10^{-4}). Pour cette métrique, nos modèles sont donc aussi performants que la méthode naïve, ce qui est un peu décevant.
Toutefois, lorqu'on regarde les spectres débruités par nos modèles, on s'aperçoit que l'allure générale est proche du spectre original, mais l'échelle n'est parfois pas la bonne (le pic maximal se retrouve à 0.1 au lieu de 1 par exemple). Cela a pour effet de pénaliser violemment la *mse*, alors qu'il suffit de renormaliser le spectre pour obtenir un débruitage efficace.

4.3.2 Résultats avec renormalisation des signaux

On ajoute donc, après la prédiction, un couche de renormalisaton des spectres. Cela s'implémente tout simplement avec une boucle for :

```
Y_pred = model.predict(X_test)
for i in range(n):
  Y_pred[i] = Y_pred[i]/max(Y_pred[i])
print("mse : ", np.mean(mean_squared_error(Y_test, Y_pred)))
```

Calculons la *mse* obtenue :

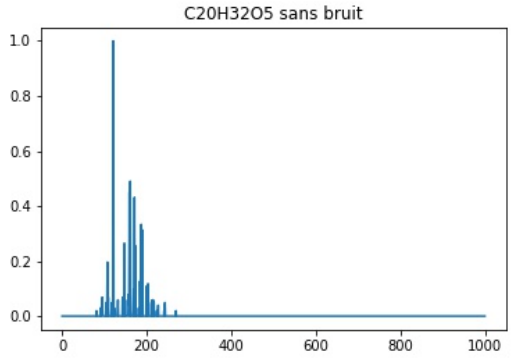
Dataset d'entraînement	*mse* sur des vrais spectres
MassBank + renormalisation	1.9×10^{-4}
Pyisopach + renormalisation	3.9×10^{-4}
Génération aléatoire + renormalisation	5.1×10^{-4}

Pour les trois modèles, la renormalisation a augmenté la *mse*. On peut abondonner cette technique, qui ne semble pas porter ses fruits.

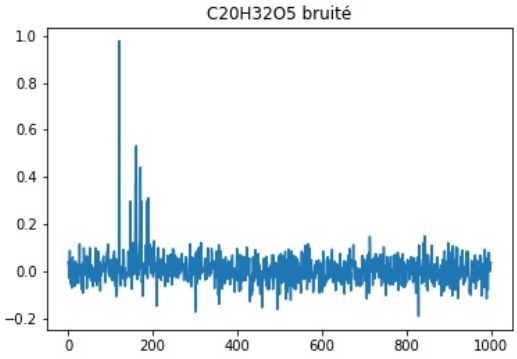
4.4 Observation de deux exemples

4.4.1 Molécule de $C_{20}H_{32}O_5$

Observons les résultats de nos modèles sur le spectre de la molécule de $C_{20}H_{32}O_5$. Les deux graphes suivants présentent le spectre réel et le spectre bruité de cette molécule.



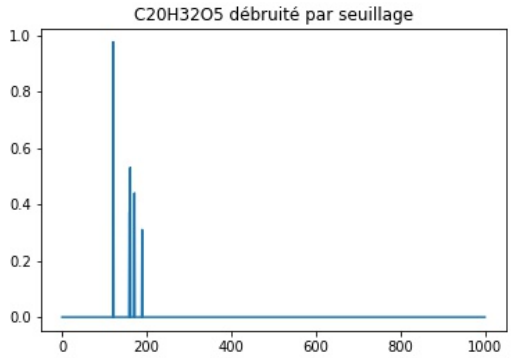
(a) Spectre réel



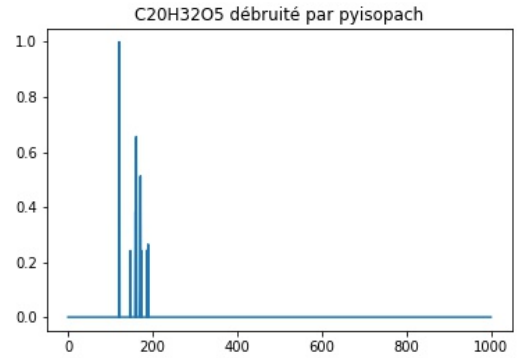
(b) Spectre bruité

Figure 16: Molécule de $C_{20}H_{32}O_5$

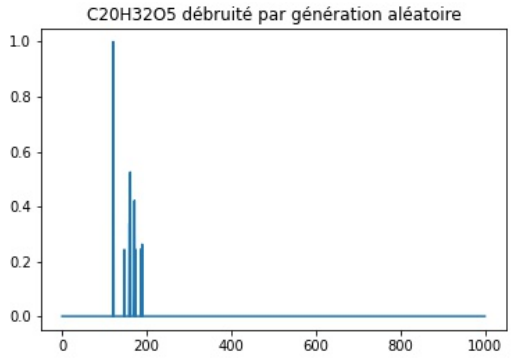
On utilise maintenant chacune de nos 4 techniques de débruitage (seuillage naïf et les trois méthodes d'entraînement du réseau de neurones) sur le spectre 16b. Voici les résultats obtenus :



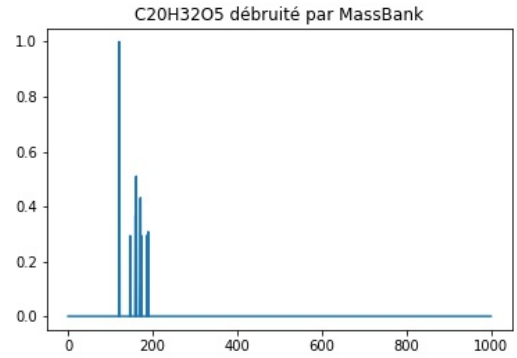
(a) Seuillage, $mse = 5.9 \times 10^{-4}$



(b) Pyisopach, $mse = 2.4 \times 10^{-4}$



(c) Génération aléatoire, $mse = 2.3 \times 10^{-4}$

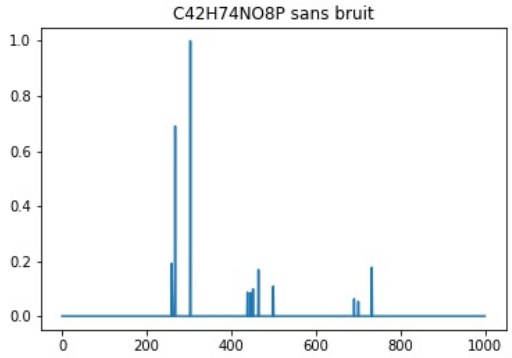


(d) MassBank, $mse = 1.9 \times 10^{-4}$

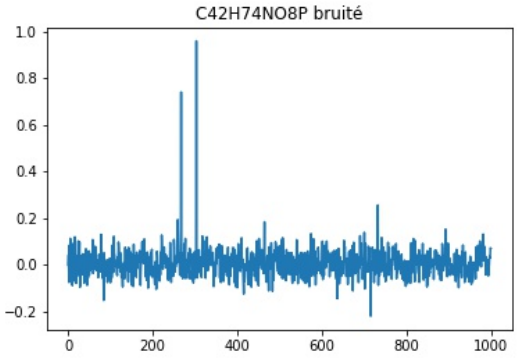
Nos méthodes de deep learning semblent meilleures que le seuillage naïf, puisque les spectres reconstruits paraissent plus fidèles à l'original. En particulier, le pic moyen à environ $m = 150$ est conservé par nos méthodes, mais pas par le seuillage.

4.4.2 Molécule de $C_{42}H_{74}NO_8P$

Observons les résultats de nos modèles sur le spectre de la molécule de $C_{42}H_{74}NO_8P$. Les deux graphes suivants présentent le spectre réel et le spectre bruité de cette molécule.



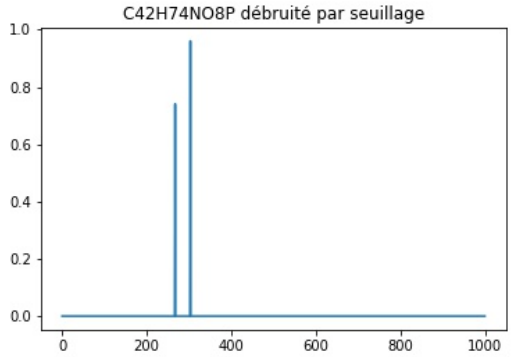
(a) Spectre réel



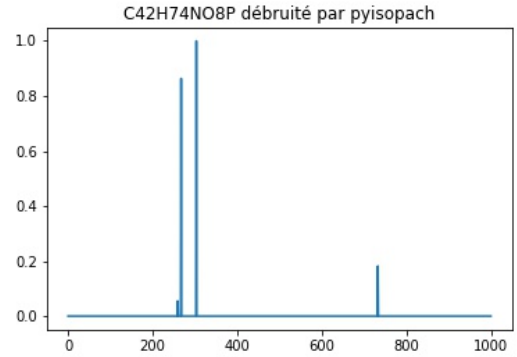
(b) Spectre bruité

Figure 18: Molécule de $C_{42}H_{74}NO_8P$

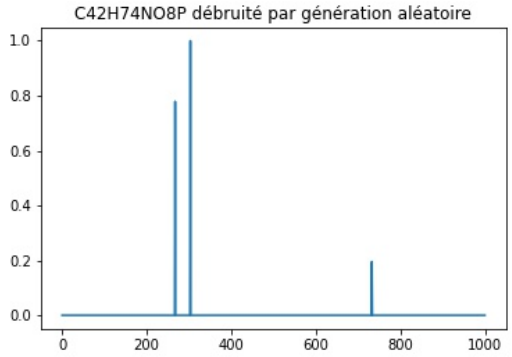
On utilise maintenant chacune de nos 4 techniques de débruitage (seuillage naïf et les trois méthodes d'entraînement du réseau de neurones) sur le spectre 18b. Voici les résultats obtenus :



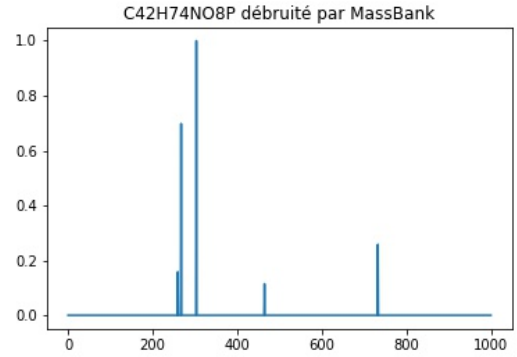
(a) Seuillage, $mse = 1.5 \times 10^{-4}$



(b) Pyisopach, $mse = 1.2 \times 10^{-4}$



(c) Génération aléatoire, $mse = 1.2 \times 10^{-4}$



(d) MassBank, $mse = 6.1 \times 10^{-5}$

Comme précédemment, les trois techniques de deep learning semblent donner un meilleur débruitage que le seuillage naïf. En effet, certains pics secondaires sont maintenus, notamment avec la technique MassBank qui donne un débruitage très proche de l'original.

4.5 Test d'autres types de couches

À l'image des couches Denses, nous avons également essayé des couches à convolution 1D (*Conv1D*), Ces couche créent un filtre qui est convolué avec l'entrée de la couche sur une seule dimension pour produire une sortie auquel on applique une fonction d'activation Relu utilisée précédemment. Et des couches (*conv1Dtranspose*) qui représentent la transposition de (*Conv1D*) plutôt qu'une déconvolution réelle. La composition du réseau est représentée sur la figure suivante :

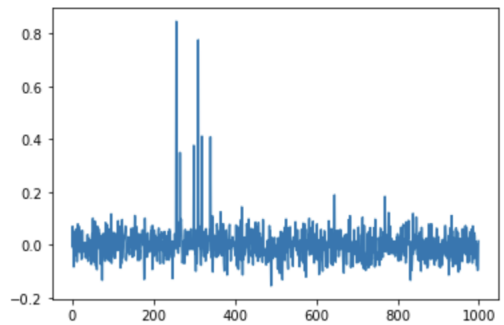
```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_6 (Conv1D)            (None, 998, 128)          512
_________________________________________________________________
conv1d_7 (Conv1D)            (None, 996, 32)           12320
_________________________________________________________________
conv1d_transpose_4 (Conv1DTr (None, 998, 32)           3104
_________________________________________________________________
conv1d_transpose_5 (Conv1DTr (None, 1000, 128)         12416
_________________________________________________________________
conv1d_8 (Conv1D)            (None, 1000, 1)           385
=================================================================
Total params: 28,737
Trainable params: 28,737
Non-trainable params: 0
_________________________________________________________________
```

Figure 20: Composition du réseau de neurones

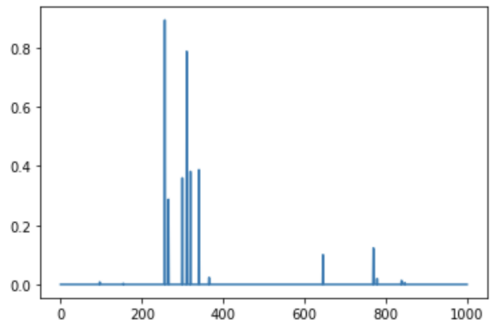
```
input_shape = (1000,1)
max_norm_value = 2.0
model = Sequential()
model.add(Conv1D(128, kernel_size=3, kernel_constraint=max_norm(max_norm_value),
                 activation='relu', kernel_initializer='he_uniform',
                 input_shape=input_shape))
model.add(Conv1D(32, kernel_size=3, kernel_constraint=max_norm(max_norm_value),
                 activation='relu', kernel_initializer='he_uniform'))
model.add(Conv1DTranspose(32, kernel_size=3, kernel_constraint=max_norm(max_norm_value),
                          activation='relu', kernel_initializer='he_uniform'))
model.add(Conv1DTranspose(128, kernel_size=3, kernel_constraint=max_norm(max_norm_value),
                          activation='relu', kernel_initializer='he_uniform'))
model.add(Conv1D(1, kernel_size=3, kernel_constraint=max_norm(max_norm_value),
                 activation='relu', padding='same'))
```

Pour que le réseaux optimise ses poids nous avons choisi la fonction de perte : *Mean Squared Error (MSE)* ainsi que l'optimiseur *Adam*.

Nous avons également évalué le modèle sur des vrais spectres, nous avons alors obtenu : $mse = 2.5 \times 10^{-3}$. Ce qui confirme que cette architecture est moins performante que la précédente basée sur les couches Denses. La figure suivante, montre un exemple de débruitage effectué en utilisant l'architecture (*Conv1D*) :



(a) Spectre bruité



(b) Spectre débruité

5 Plateforme web

Dans l'objectif de faciliter l'utilisation de notre solution, c'est à dire débruiter le spectre bruité issu du spectromètre, nous avons developpé une application web qui utilise en frontend du **React JS** et en backend un serveur **Flask (microframework de développement web en Python)**.

L'application web est disponible en cliquant ici.

5.1 Fonctionnement

Comme évoqué ci-dessus, notre plateforme repose sur du **React JS** côté frontend et sur **Flask** côté backend.

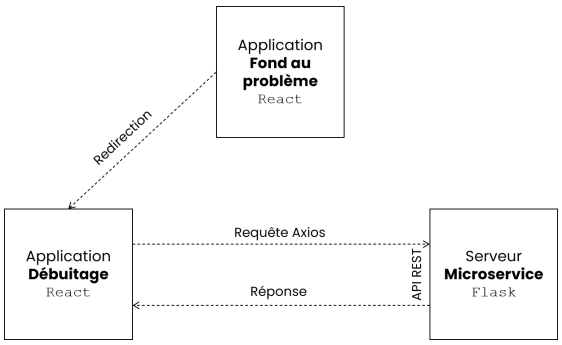


Figure 22: Structure de l'application web

5.1.1 Front-end

Le front-end est composé de deux applications web react dont la prémière pour présenter essentiellement le fond du problème et des exemples de résultats obtenus via notre solution, et la seconde pour donner la possibilité à l'utilisateur d'exploiter notre solution.

5.1.2 Back-end

Le serveur est codé en python avec Flask sous forme de microservices utilisant la specification REST (Representational state transfer).

5.2 Fonctionnalités

Le site est consitué de deux parties principales : une pour débruiter un signal fourni par l'utilisateur en utilisant les techniques de deep learning présentées dans les parties précédentes, et une pour visualiser le spectre d'une molécule à partir de sa formule chimique.

5.2.1 Débruitage de spectres

L'utilisateur doit uploader son signal bruité sous la forme d'un fichier *csv* à deux colonnes. Lorsqu'il soumet le fichier, un programme vérifie d'abord son format et affiche le signal. L'utilisateur dispose ensuite de 3 boutons pour débruiter : un pour chaque méthode d'entraînement (*MassBank*, *Pyiso* ou *Génération aléatoire*). Ainsi, il est possible de comparer les trois méthodes.
Enfin, l'utilisateur peut télécharger le signal débruité.

5.2.2 Visionnage de spectres

Nous avons également ajouté une partie qui permet juste à l'utilisateur de visionner des spectres en fonction de la formule chimique, dans le but qu'il puisse tout simplement comparer son signal aux signaux existants et chercher manuellement une correspondance.

Remarque Une idée intéressante aurait été d'écrire un algorithme qui repère directement les molécules dont le spectre est le plus proche (en terme de *mse* par exemple) du signal uploadé par l'utilisateur. Mais ce n'est pas une méthode de deep learning.

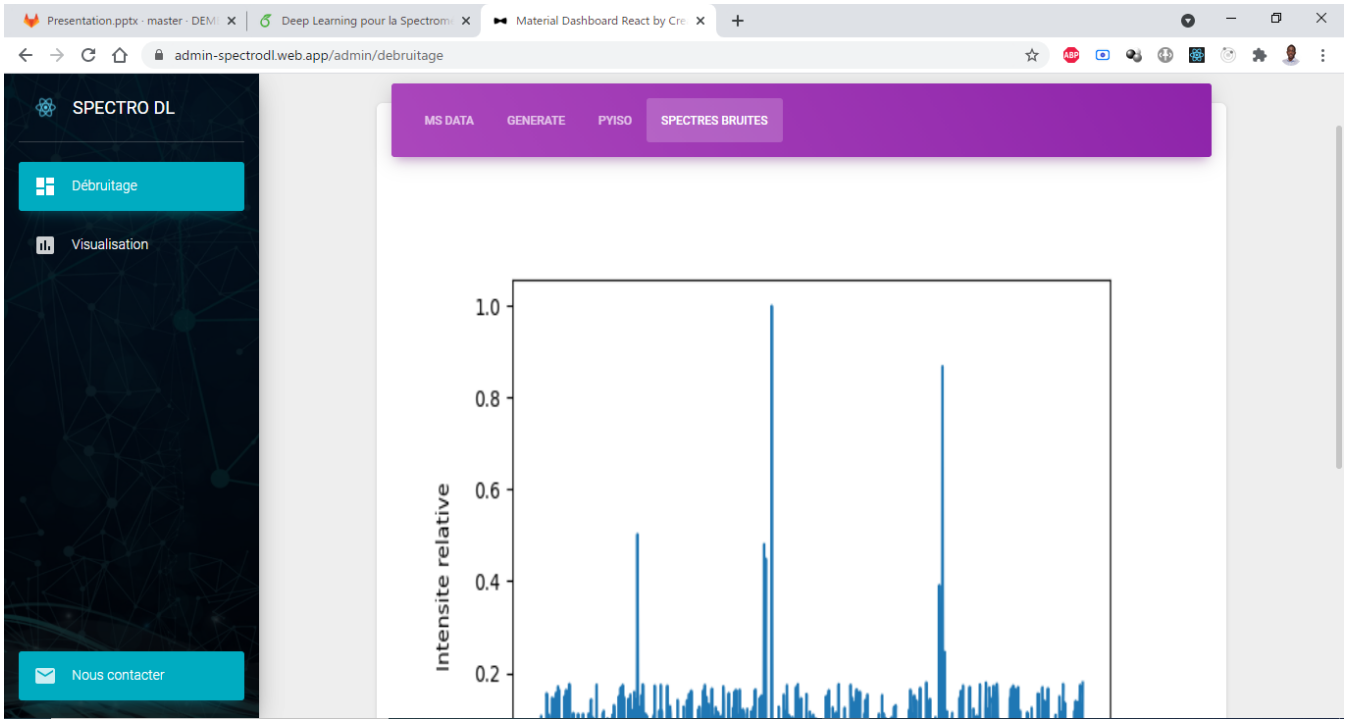


Figure 23: Capture d'écran de la partie débruitage

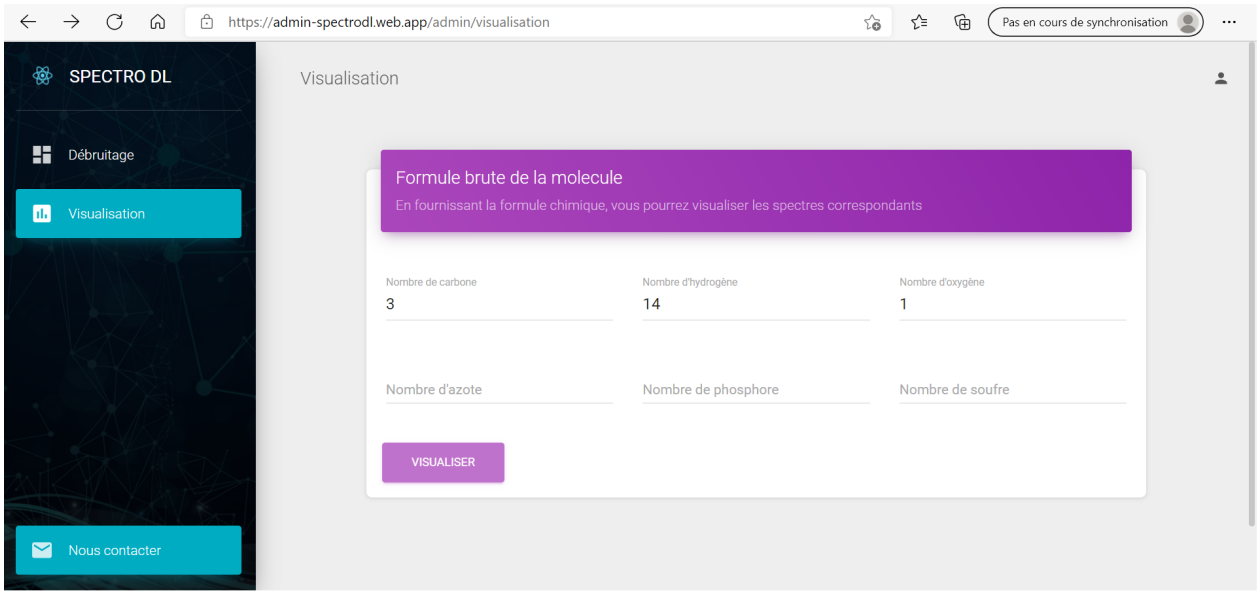


Figure 24: Capture d'écran de la partie visionnage

6 Conclusion et perspectives

Durant ce projet, nous avons élaboré une méthode pour construire une base de données de spectres arbitrairement grande. Nous nous sommes basés sur une base de données préexistante (*MassBank*) pour élaborer une fonction générative de signaux la plus vraisemblable possible. Nous avons également utilisé la bibliothèque *Pyiso* de python qui peut générer une quantité infinie de spectres.
Par la suite, nous avons utilisé ces sources de données pour entraîner un autoencodeur à débruiter des signaux. Nous avons également comparé les performances de ces modèles à la méthode basique de seuillage.
Enfin, dans l'optique de rendre ce travail accessible à tous, nous avons mis en ligne une plateforme de débruitage utilisant cette technique.

Ce sujet est vaste et beaucoup de possibilités s'offrent aux étudiants, il est possible que nous ayons pris des mauvaises directions ou négligé certaines approches. Néanmoins nous pouvons déjà suggérer un certain nombre de perspectives et améliorations pour le groupe qui pourrait reprendre ce projet l'an prochain.

- Étant limités par la quantité de RAM disponibles sur nos ordinateurs et sur Google Colab, nous avons dû nous limiter à entraîner nos modèles sur 10000 spectres. Il pourrait être intéressant d'évaluer le modèle entraîné avec toutes les données *MassBank*, voire même avec des données générées en plus, en utilisant un supercalculateur ou bien en ne stockant pas toutes les données (en utilisant un *generator* python, avec le mot-clé *yield* par ex).
- La partie génération de spectre 3.3.2 peut être améliorée. Nous avons utilisé une technique astucieuse pour comparer 3 fonctions de génération de spectres entre elles, mais cette technique peut être très largement développée pour obtenir des spectres bien plus crédibles. Il suffirait de faire une *gridsearch* avec un certain nombre d'hyperparamètres représentant les lois de probabilités et les paramètres associés.
- Notre site de débruitage est assez contraignant pour l'utilisateur, puisqu'il faut que celui-ci échantillonne lui-même son spectre mesuré en 1001 points, sur l'intervalle $m/Z \in [0, 1000]$. Une amélioration simple serait que cet échantillonnage soit fait automatiquement lorsque l'utilisateur upload son fichier.

Malgré les difficultés engendrées par le fait de devoir travailler en distanciel, nous sommes assez fiers du travail réalisé. Nous avons également beaucoup appris en réalisant ce projet, que ce soit en termes de spectrométrie, deep learning ou développement web.

References

[1] Delsuc Marc-André3 Cherni Afef, Chouzenoux Emilie. Fast dictionary-based approach for mass spectrometry data analysis.

[2] J.-P Speir M.-W. Senko and F.-W. McLafferty. Collisional activation of large multiply charged ions using fourier trans-form mass spectrometry. *Analytical Chemistry*, 1994.

[3] B. Rossi et M. Samson N. M. Nafati, J.-M. Guigonis. Technique objective de traitement des spectres de masse protéomiques basée sur le seuillage flou multi-échelle. 2005.

[4] Nicole Morin Nicole Sellier. Qu'est-ce que la spectrométrie de masse ? *CultureSciences Chimie*, 2002.

[5] Wikipédia. Spectrométrie de masse.