Physique numérique
PHY-3500

Mathieu Marquis
Charles-Antoine Parent
Thomas Piuze

Étude de méthodes alternatives de régressions gaussiennes avancées

Travail présenté à : Antoine Allard, Philippe Després et Xavier Roy-Pomerleau

Travail remis le : 29 avril 2025

Département de physique, de génie physique et d'optique

Résumé

L'ajustement informatique de spectres complexes, comportant des composantes superposées ou bruitées, représente un défi important en physique expérimentale. Ce travail compare ainsi trois méthodes d'ajustement : un ajusteur déterministe basé sur la fonction `curve_fit` de SciPy, un réseau de neurones convolutif (CNN) simple et un réseau résiduel (ResNet). À partir de 7 spectres synthétiques produits à 3 niveaux de bruit différents, les performances de chaque méthode sont évaluées à l'aide du coefficient de détermination R^2 et de l'erreur quadratique moyenne (MSE). Les résultats démontrent que, pour des spectres simples, l'ajustement par minimisation déterministe des carrés offre des performances excellentes tout en étant facile à mettre en oeuvre. Cependant, pour des spectres complexes avec des composantes superposées et un bruit important, les approches d'apprentissage profond (DL) surpassent la méthode classique, en particulier grâce à leur capacité à apprendre à partir d'ensembles de données décomposées. L'ajusteur ResNet se distingue par une convergence plus rapide et des prédictions plus précises que le CNN simple, au prix d'un temps d'entraînement 50% supérieur. Des considérations pratiques telles que le choix du taux d'apprentissage, la prévention du surapprentissage et l'importance de générer des ensembles de données représentatifs sont également discutées. Enfin, des pistes sont proposées pour intégrer des estimations d'incertitudes aux prédictions, notamment par l'utilisation du Monte Carlo Drop-Out, afin de permettre la pleine application de cette méthode à la physique expérimentale.

1 Introduction

En physique expérimentale, la régression est un outil statistique essentiel pour étudier les relations entre différentes variables et pour ainsi mieux comprendre divers phénomènes physiques [1]. En particulier, l'analyse de spectres repose couramment sur des ajustements précis de fonctions théoriques à des données bruitées. Que ce soit en astrophysique, en spectroscopie ou en physique des particules, l'ajustement de spectres correspond à une étape essentielle pour extraire des informations quantitatives sur les systèmes observés et comprendre les phénomènes en jeu [2, 3, 4]. Pendant longtemps, ces ajustements étaient réalisés à l'aide de méthodes déterministes, telles que la minimisation des moindres carrés non linéaires [5]. Toutefois, avec l'augmentation de la complexité et du nombre de données de même qu'avec l'amélioration des technologies permettant d'effectuer des calculs encore plus efficacement, des méthodes alternatives, notamment basées sur l'apprentissage automatique, ont émergé comme des outils puissants pouvant grandement accélérer et améliorer le processus d'ajustement. En astrophysique notamment, les approches déterministes historiquement employées pour faire l'ajustement, telles que le logiciel d'analyse de données ORCS pour les données de SITELLE [6], un instrument installé au télescope Canada-France-Hawaï, sont de moins en moins utilisées au profit de nouvelles technologies profitant de la robustesse des méthodes d'apprentissage automatique, comme le package LUCI permettant une analyse rapide des spectres de SITELLE [7].

Dans cette optique de diversifier nos connaissances des différentes méthodes d'ajustement à notre disposition de même que de leurs avantages et inconvénients, nous proposons plusieurs méthodes d'ajustement de raies spectrales. Nous présentons ainsi la comparaison de plusieurs approches pour l'ajustement de spectres composés de plusieurs composantes gaussiennes et d'un bruit variable. Sept types de spectres à trois niveaux de bruit chacun ont été créés afin de servir de données de test et d'entraînement. D'une part, une méthode classique utilisant l'optimiseur `scipy.optimize.curve_fit` et l'estimation de paramètres initiaux avec la fonction `scipy.signal.find_peaks` a été mise en oeuvre. D'autre part, des ajusteurs fondés sur l'apprentissage profond (DL) à partir de réseaux de neurones convolutionnels (CNN) ont été conçus et entrainés sur chaque spectre. Deux architectures principales ont été explorées : un CNN simple et une version adaptée de réseaux résiduels (ResNet), inspirée de la structure des ResNets introduites par He et al. en 2015 [8]. Ces méthodes sont ensuite comparées en se basant sur différentes mesures de score, soit le coefficient de détermination R^2 et l'erreur moyenne des carrés *mean squared error* (MSE), de même qu'en comparant les temps d'exécution. Nous discutons également de comment cette approche peut être adaptée pour être utilisée en astrophysique expérimentale. Bien que l'essence du travail se concentre sur la régression en astrophysique, les conclusions établies peuvent facilement être extrapolées à d'autres applications en analyse de données. L'objectif principal du projet est ainsi d'évaluer dans quelle mesure des implémentations simples de modèles neuronaux peuvent rivaliser avec les méthodes traditionnelles en termes de précision d'ajustement, de robustesse face au bruit et de rapidité d'inférence.

Le présent rapport est organisé comme suit. La section 2 détaille les données spectrales simulées, l'architecture logicielle du projet, les différents ajusteurs implémentés, ainsi que les procédures d'entraînement et les critères d'évaluation utilisés. La section 3 présente les résultats obtenus par chaque ajusteur sur tous les spectres étudiés, les courbes d'entraînement des ajusteurs à DL et l'influence des paramètres comme le taux d'apprentissage sur la solution trouvée. La section 4 propose une analyse critique de ces résultats, discute des avantages et limites des approches explorées et propose une méthode pour appliquer les ajusteurs conçus dans un véritable contexte expérimental. Enfin, la section 5 résume les principales conclusions du projet et propose des pistes d'amélioration pour de futurs travaux. Afin d'approfondir la compréhension du lecteur, nous recommandons la consultation du code à l'adresse suivante : `https://github.com/mamar828/physique_numerique.git`.

2 Méthodes

2.1 Spectres étudiés

Dans le but de fournir un ensemble de données synthétiques représentatif de plusieurs situations expérimentales, sept types de spectres de trois niveaux de bruits sont générés et servent de données de référence. Bien que l'architecture du code discutée à la prochaine section permette facilement l'inclusion d'autres modèles, des fonctions gaussiennes sont utilisées pour l'entièreté de ce travail par simplicité et puisque celles-ci correspondent à la réponse de plusieurs instruments [9]. Cet ensemble de données permet d'atteindre une grande diversité concernant le nombre de composantes gaussiennes et leur proximité, le niveau de bruit, de même que les phénomènes physiques en jeu.

La Fig. 1 présente les sept types de spectres de référence, chacun affiché avec les trois niveaux de bruit discutés précédemment. Afin d'explorer une grande variété de configurations, ces spectres sont générés en faisant varier l'amplitude, la moyenne et l'écart-type des gaussiennes selon une loi normale centrée sur une valeur arbitraire. Les barres d'erreur illustrées dans la Fig. 1 correspondent à l'écart-type de ces distributions : elles ne délimitent pas l'intervalle complet de variation des paramètres, mais indiquent plutôt la région où environ 68% des paramètres générés ($\pm 1\sigma$) se situeront. Cette modélisation aléatoire par des lois normales est particulièrement pertinente, car de nombreux phénomènes physiques suivent naturellement des comportements gaussiens [10]. Concernant la barre d'erreur verticale grise, celle-ci représente plutôt 3σ de l'amplitude du bruit gaussien ajouté aux spectres.

Par ailleurs, un nombre arbitraire de 100 canaux a été choisi pour cette étude, constituant un compromis entre des instruments possédant un grand nombre de canaux et ceux offrant des intervalles plus petits. Ce choix permet d'obtenir des spectres présentant plusieurs composantes superposées, tout en restant suffisamment léger pour pouvoir générer et stocker efficacement un grand nombre de spectres en mémoire.

Les spectres de la Fig. 1 offrent chacun des défis différents, qui peuvent chacun être testés selon une importance de bruit variable, représentant la variété dans les résolutions d'instruments. Les sous-figures a) et b) offrent des spectres très simples à ajuster pour pouvoir comparer la performance des ajusteurs dans des conditions favorables. La très grande variabilité de l'amplitude de plusieurs gaussiennes permet d'ailleurs de générer plusieurs spectres où certaines raies ne seront pas distinguables afin d'étudier le comportement des techniques d'ajustement. Les sous-figures c) et d) présentent un défi plus intéressant avec deux gaussiennes très larges, où dans le premier cas les pics des gaussiennes seront majoritairement facilement distinguables alors que dans le deuxième cas, la forme individuelle des gaussiennes ne pourra être établie. Cet empilement de contributions est fréquent en astrophysique expérimentale, notamment dans les spectres de restes de supernovas dont la coquille en expansion provoque un décalage Doppler variable des différentes raies [11]. La sous-figure e) offre des gaussiennes extrêmement piquées, typiques d'instruments à faibles résolutions spectrales, mais à grande plage spectrale comme SITELLE [12]. La sous-figure f) permet d'observer deux gaussiennes de grandes amplitudes et dispersions, contaminées par des gaussiennes secondaires plus faibles. Cette superposition est très fréquente notamment lors de l'étude des raies [N II] et Hα, qui sont contaminées lors d'observations terrestres par des raies d'OH provenant du ciel nocturne [13]. Dans un tel cas, il est alors essentiel d'être en mesure d'ajuster chaque composante afin de pouvoir isoler la contribution de celles d'intérêt. Finalement, la sous-figure g) illustre un spectre composé de composantes gaussiennes d'amplitudes et de dispersions extrêmement différentes, mais de moyennes très rapprochées. Des spectres de cette forme indiquent que des régimes physiques entièrement distincts sont en jeu et sont souvent rencontrés lors de l'étude des composantes froides et chaudes du milieu interstellaire neutre [14]. Avec cette variété de spectres, il est ainsi possible d'étudier toute une panoplie de phénomènes physiques courants, particulièrement en astrophysique.

2.2 Ajusteurs implémentés

Afin de fournir une comparaison simple des différentes techniques d'ajustement proposées, une méthode de référence basée sur la librairie SciPy est premièrement utilisée [15]. Cette méthode fait usage de la fonction `scipy.optimize.curve_fit`, qui permet d'effectuer une régression sur une fonction quelconque en minimisant la somme des carrés entre la fonction prédite et les données. Nous cherchons ainsi à minimiser la fonction de résidu

$$S = \sum_{i=1}^{N} [y_i - f(x_i, \alpha_1, \alpha_2, \dots)]^2$$

où N correspond au nombre de données, (x_i, y_i) aux coordonnées de chaque donnée et $f(\dots)$ à la fonction à minimiser en variant ses paramètres α_j. Par simplicité, l'algorithme Levenberg-Marquardt, soit l'algorithme par défaut de la fonction `scipy.optimize.curve_fit`, est utilisé pour effectuer cette minimisation. Cet algorithme, proposé initialement par Levenberg en 1944 puis redécouvert par Marquardt en 1963, est un algorithme de minimisation de fonctions non-linéaires avec des propriétés de forte convergence [16]. Il offre l'avantage d'être plus robuste à de mauvaises estimations initiales que d'autres algorithmes, comme l'algorithme de Gauss-Newton, au détriment de la vitesse de convergence [17]. Par conséquent, cet algorithme est un choix judicieux pour l'ajustement de fonctions simples puisqu'il converge facilement vers une solution optimale, mais pas nécessairement vers le minimum global [18].

Toutefois, lorsqu'il est question de faire l'ajustement de fonctions plus complexes, telles que la superposition de plusieurs gaussiennes non résolues, la régression par minimisation des carrés est grandement limitée et nécessite une estimation initiale très précise des paramètres pour favoriser une convergence vers le minimum global. Cette méthode d'ajustement nécessite donc un algorithme tout aussi performant afin d'effectuer les estimations initiales des différents paramètres. En pratique, cet algorithme doit être ajusté selon le type de données utilisées puisque la résolution et le bruit influencent grandement les données. Cela est notamment le cas en astrophysique, où la résolution d'instruments variant entre la détection de simples pics et la mesure du profil global des raies affecte entièrement l'allure du spectre, et ainsi l'algorithme d'estimations initiales. Cela est d'autant plus valable lorsque la superposition de plusieurs raies entre en jeu, comme il peut être le cas lors de l'étude de la raie [N II], émise par les régions H II, mais dont la longueur d'onde se superpose à celle de raies d'OH présentes dans l'atmosphère, venant polluer le spectre de [N II] [13]. De plus, le bruit intrinsèque variable selon le domaine électromagnétique étudié, notamment en comparant des données radio à des données dans le visible, engendre une variation supplémentaire de la forme des pics. Ainsi, un algorithme d'estimations initiales est nécessaire pour chaque instrument, pour chaque intervalle de longueur d'onde et pour chaque superposition de raies [13]. Il est évident qu'une telle implémentation requiert des ressources considérables.

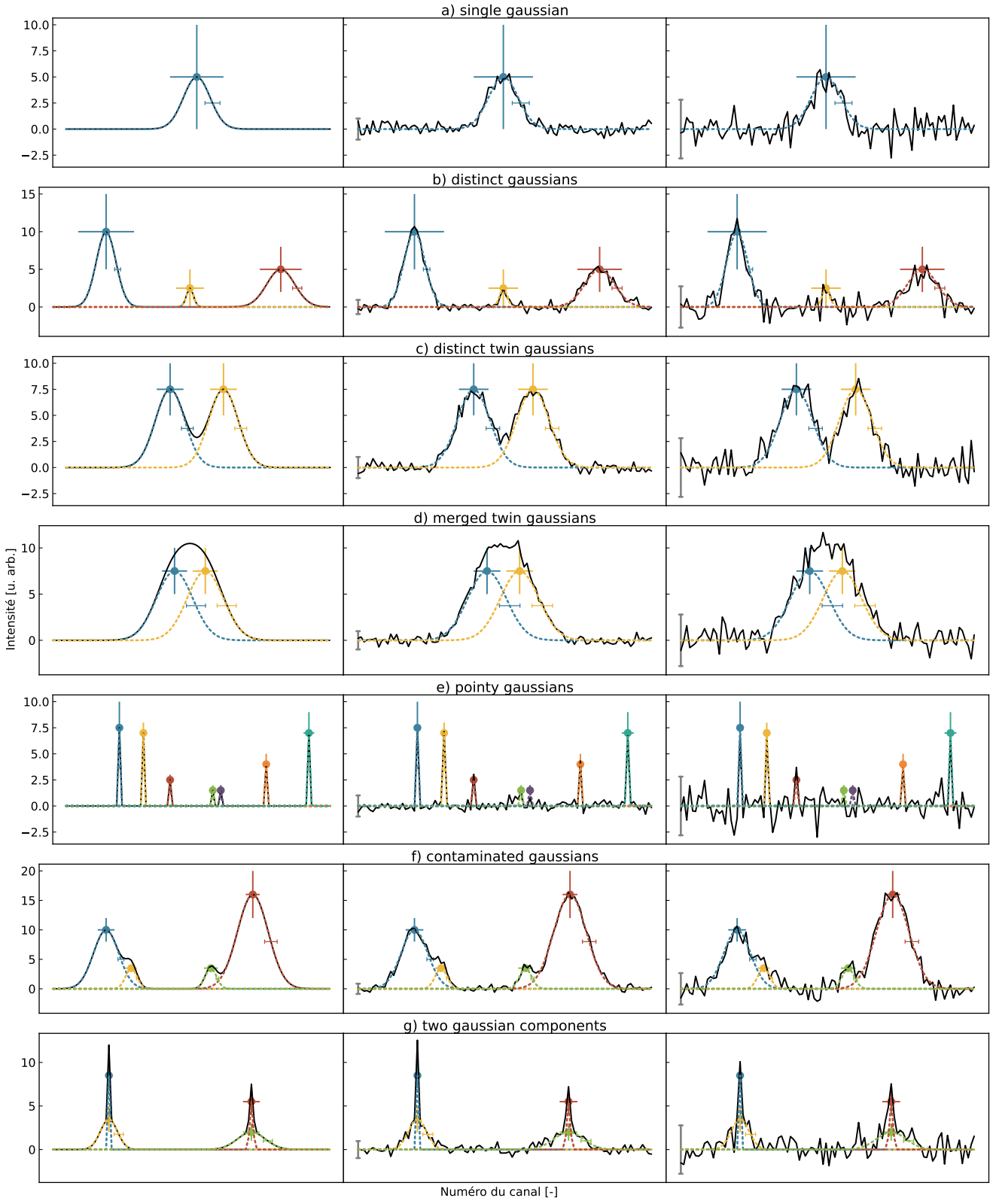

FIGURE 1 – Spectres de référence utilisés créés à partir de 100 canaux. La première colonne illustre uniquement les gaussiennes, la deuxième colonne représente l'ajout d'un bruit gaussien de $\sigma = 0.4$ et la troisième colonne possède un bruit de $\sigma = 1$. Les barres d'erreur autour du sommet de chaque gaussienne et à la mi-hauteur de chacune illustrent l'étendue de la variation des paramètres lors de la génération de l'ensemble de données. La ligne grise représente 3σ du bruit gaussien ajouté.

Dans ce travail, une implémentation très simple d'un algorithme d'estimations initiales est effectuée, et ce même algorithme est conservé pour tous les spectres utilisés. Cet algorithme effectue la détection des pics de chaque spectre grâce à la fonction `scipy.signal.find_peaks`, puis estime l'écart-type des gaussiennes en mesurant la largeur à la moitié de l'amplitude des pics détectés. Il est d'ailleurs important de noter que les arguments `prominence`, `height`, `width` et `distance` de la fonction SciPy permettent de contrôler la détection des différents pics. En ajustant la valeur de ces paramètres visuellement pour chaque spectre, les estimations initiales sont fournies à l'ajusteur SciPy pour obtenir les paramètres ajustés de chaque modèle gaussien.

Afin d'offrir une approche entièrement différente, un ajusteur CNN est également implémenté grâce à la librairie PyTorch [19]. Ce dernier, contrairement à l'ajusteur SciPy, offre l'avantage de ne pas nécessiter d'estimations initiales pour effectuer l'ajustement. Par contre, tel que discuté à la section précédente, un grand nombre de spectres synthétiques doit être créé pour entraîner le modèle, avant de l'appliquer aux données expérimentales dont la décomposition nous intéresse. Néanmoins, avec l'interface implémentée pour créer, puis générer des spectres, nous jugeons que dans le contexte d'astrophysique expérimentale discuté ici, il est bien plus rapide de créer des spectres pour chaque groupe de données plutôt que de concevoir un algorithme maison d'estimations initiales pour chaque ensemble de données.

La Fig. 2 présente le fonctionnement des CNNs bidimensionnels. Notons que les couches de *pooling* réfèrent à l'utilisation d'une méthode de sous-échantillonnage afin de réduire le nombre de dimensions. Bien qu'un très grand nombre de types de *pooling* existent, tels que la sélection de la valeur moyenne dans une fenêtre (`torch.nn.AvgPool` dans PyTorch), le *max pooling* est implémenté ici, conformément au premier CNN d'apprentissage profond, soit le « AlexNet » [20]. Cette méthode consiste ainsi à diviser les données d'entrée en cases de largeur définie et à prélever la valeur maximale de chaque case pour former le nouvel ensemble de données. Pour la première couche du réseau, les paramètres sélectionnés sont, dans l'ordre : convolution 1D avec un noyau de 5 pixels et générant 16 cartes de caractéristiques d'une longueur de 100 pixels, fonction d'activation *Rectified Linear Units* (ReLU), puis *max pooling* avec un noyau de 2 pixels. La deuxième couche est presque identique à la première : des 16 cartes de caractéristiques générées précédemment sont formées 32 nouvelles cartes. Après ces couches de convolution, les couches pleinement connectées utilisent une transformation linéaire des cartes de caractéristiques pour générer une matrice de 128 poids. La sortie est ensuite à nouveau passée dans la fonction d'activation ReLU et une dernière couche linéaire permet de réduire le nombre de caractéristiques au nombre de paramètres étudiés, soit trois fois le nombre de composantes du spectre. Le CNN implémenté offre ainsi la possibilité de convertir une série de données unidimensionnelle en entrée en les paramètres des gaussiennes présentes dans le signal, en ayant initialisé le nombre de composantes.

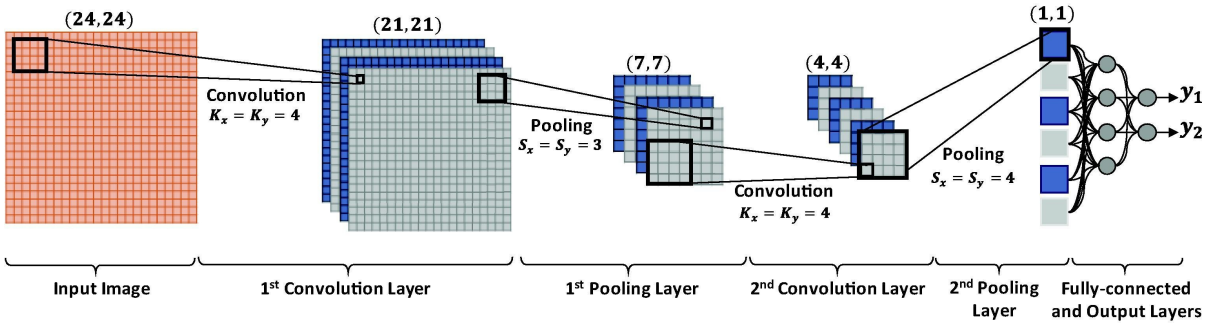

FIGURE 2 – Architecture du CNN 2D ayant inspiré celui effectué dans ce travail [21]. Bien que la séquence d'opérations ait été conservée, les paramètres de chacune ont été modifiés pour correspondre à la longueur des spectres étudiés.

Pour présenter les possibilités d'architectures plus robustes, un dernier réseau est implémenté, soit le ResNet, proposé en 2015 pour faciliter l'entraînement de réseaux substantiellement plus profonds que ceux typiquement utilisés [8]. L'architecture de ce type de réseau, bien plus complexe que celle du CNN, n'est pas expliquée en détail ici, mais nous suggérons sa lecture dans l'ouvrage de He et al. pour se renseigner sur son fonctionnement. Expliqué très sommairement, un ResNet est un réseau incorporant des connections supplémentaires entre les couches, introduisant des « raccourcis » entre l'entrée et la fin d'une couche de convolution. La sortie des blocs, que nous classons de « résiduels », correspond alors à la combinaison des données d'entrée et du résultat des couches de convolution. Ce type de réseau permet donc de réutiliser en partie les données initiales dans sa sortie. Le réseau implémenté dans cette étude utilise donc une première couche de convolution similaire à celles du CNN, suivi de deux groupes formés de deux blocs résiduels et d'un *max pooling*. Chaque bloc résiduel est formé de deux couches de convolution et sa sortie correspond à l'addition des données d'entrée et du résultat des couches de convolution. Finalement, une couche pleinement connectée très similaire à celle du CNN permet d'obtenir l'estimation des paramètres. Puisque ce réseau possède un très grand nombre de couches de convolution, des tailles de noyaux variables ont été utilisées selon les couches. La première couche utilise un noyau d'une taille de 7, les deux premiers blocs résiduels utilisent une taille de 5 et les deux derniers blocs résiduels utilisent une taille de 3. Cette diversité dans la taille des noyaux permet au réseau de capter un ensemble élargi de caractéristiques dans le

spectre : les plus petites caractéristiques sont détectées par les convolutions à petits noyaux alors que les tendances plus globales sont captées par de plus grands noyaux. Le réseau est ainsi plus robuste à diverses tailles et superpositions de raies.

2.3 Entraînement

Avant de pouvoir effectuer des prédictions et des mesures de score avec les ajusteurs DL, il est nécessaire d'effectuer un entraînement de ces réseaux pour que ceux-ci *apprennent* les paramètres optimaux. Par exemple, pour une couche de convolution de l'ajusteur CNN impliquant la convolution avec un noyau de 5 pixels pour générer 16 sorties, un total de 96 paramètres doit être appris par le modèle, puisque chaque convolution implique l'apprentissage de 5 poids et d'un biais (constante), pour un total de 6 paramètres par sortie. Le processus d'entraînement consiste ainsi à ajuster les poids de chacune des connexions internes et opérations de manière à minimiser une fonction de coût mesurant l'écart entre la prédiction du réseau et les vraies valeurs. Plus précisément, une passe avant (*forward pass*) est effectuée pour obtenir une prédiction, et l'erreur résultante est ensuite utilisée afin de rétropropager (*backpropagation*) des gradients à travers le réseau et ajuster les poids par descente du gradient. Ce processus est répété sur un grand nombre d'exemples afin que le réseau apprenne à généraliser son comportement.

Dans l'étude présentée, la fonction de coût choisie est la MSE, définie comme

$$\text{MSE} = \frac{1}{K}\sum_{i=1}^{K}[y_i - \hat{y}_i]^2$$

où y_i est la valeur réelle et $\hat{y}_i$ est la valeur prédite du paramètre i. Notons que cette sommation est effectuée sur tous les spectres et sur tous les paramètres de sortie, soit les paramètres des gaussiennes à ajuster. Cette fonction de coût a été choisie puisqu'elle est adaptée à des problèmes de régression où une prédiction précise des valeurs numériques est désirée [22].

Afin d'assurer une bonne généralisation des réseaux et d'éviter le surapprentissage, les données sont séparées en trois ensembles : un ensemble d'entraînement (*training set*), un ensemble de validation (*validation set*) et un ensemble de test (*test set*). L'ensemble d'entraînement est utilisé pour optimiser les poids des réseaux selon l'algorithme détaillé précédemment, alors que l'ensemble de validation est utile pour évaluer la performance du modèle sur des données non vues pendant l'optimisation des paramètres, permettant ainsi de mesurer la capacité du modèle à généraliser pour de nouvelles données, et ce, à chaque époque. Notons d'ailleurs que pour éviter le surapprentissage et favoriser la généralisation du modèle, l'époque offrant le meilleur score sur l'ensemble de validation est celle sélectionnée à la fin de l'entraînement. Finalement, l'ensemble de test sert à mesurer objectivement la performance finale une fois l'entraînement complété, en présentant de toutes nouvelles données. Dans ce travail, la taille des ensembles de données est répartie selon une division $0.6, 0.2, 0.2$ pour les ensemble d'entraînement, de validation et de test respectivement. De plus, chaque ajusteur est entraîné sur un spectre bien spécifique, à un niveau de bruit donné. Une telle approche permet de spécialiser les réseaux neuronaux dans l'espoir d'obtenir des algorithmes plus performants pour une tâche spécifique.

L'optimiseur utilisé pour ajuster les poids durant l'entraînement est l'algorithme Adam (`torch.optim.Adam`), soit une méthode de descente du gradient adaptative fréquemment utilisée pour son efficacité et sa robustesse [23]. De plus, un taux d'apprentissage à diminution exponentielle (`torch.optim.lr_scheduler.ExponentialLR`) est utilisé pour favoriser la convergence de l'algorithme. Les paramètres d'entraînement, choisis constants entre les réseaux pour faciliter leur comparaison directe, sont fournis au Tableau 1 et ne seront pas re-mentionnés pour chaque résultat. En effet, le but du travail présent n'étant pas d'optimiser certains modèles, mais plutôt d'effectuer un survol des avantages et inconvénients de différentes méthodes, nous n'avons pas optimisé les hyperparamètres pour chaque spectre. Il est clair qu'une telle optimisation serait nécessaire pour maximiser la performance des réseaux dans un contexte d'application réelle. Par souci de clarté, la taille totale de l'échantillon spécifiée correspond au nombre total de données utilisées ; l'ensemble d'entraînement est donc composé de 60% de ces données.

Tableau 1 – Paramètres d'entraînement des deux réseaux de DL.

Nombre d'époques	25
Taille totale de l'échantillon	10^6
Taille du mini-lot (*batch size*)	500
Taux d'apprentissage initial	0.001
Taux de diminution du taux d'apprentissage	0.99

2.4 Mesures de score

Pour être en mesure d'évaluer l'efficacité des différents ajusteurs, il est essentiel de développer une méthode pour pouvoir les comparer. Ainsi, deux métriques de score sont utilisées : le coefficient de détermination R^2 ainsi que la MSE telle qu'employée pour effectuer l'entraînement des réseaux.

Le coefficient de détermination R^2 est une mesure statistique couramment utilisée pour évaluer la qualité de l'ajustement d'un modèle par rapport aux données observées [24]. Celui-ci est défini comme

$$R^2 = 1 - \frac{\sum_{i=1}^{N} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{N} (y_i - \bar{y})^2}$$

où y_i représente les données expérimentales, $\hat{y}_i$ les valeurs prédites par le modèle et $\bar{y}$ les données expérimentales moyennes. Un score de $R^2 = 1$ indique ainsi un ajustement parfait, tandis qu'une valeur de $R^2 \approx 0$ signifie que le modèle ne représente pas mieux les données que leur moyenne. Des valeurs négatives peuvent également être obtenues et indiquent que le modèle est pire qu'une simple estimation par la moyenne. Cette métrique est particulièrement utile pour évaluer l'aptitude du modèle à capturer la variance des données. Toutefois, cette métrique est moins représentative de la qualité d'un ajustement lorsqu'un bruit important est présent dans les données, puisque celui-ci provoque nécessairement une déviation des données prédites et mesurées.

Pour offrir une métrique ne dépendant pas du bruit dans les données, la MSE est également utilisée comme métrique complémentaire afin de quantifier directement l'écart moyen entre les prédictions et les données réelles. Contrairement au R^2, une valeur de MSE plus faible indique un ajustement plus précis, tendant vers 0 lorsque les paramètres prédits sont très près des paramètres réels. Toutefois, cette métrique implique qu'il faut être en mesure de comparer directement les paramètres estimés et réels de chaque composante des spectres. Cela ne pose pas un problème pour les techniques de DL, puisque celles-ci fournissent en sortie toujours le même nombre de paramètres, mais l'ajustement par SciPy, reposant sur l'identification des pics du spectre, peut engendrer un nombre variable de composantes détectées. Dans le but de mettre toutes les méthodes sur un pied d'égalité, uniquement les spectres possédant le même nombre de composantes que le spectre original sont considérés pour le calcul de la MSE. Cette solution pour l'ajusteur SciPy, bien qu'imparfaite, offre une méthode de comparaison indépendante du bruit des données.

Pour chaque ajusteur étudié, les deux métriques sont systématiquement calculées pour chaque ajustement, permettant ainsi de comparer objectivement la performance des différentes méthodes. Le R^2 offre une mesure intuitive de la qualité globale de l'ajustement, tandis que le MSE donne une évaluation absolue de l'erreur commise. Ces deux approches combinées permettent ainsi de fournir une évaluation robuste et nuancée de la performance des ajusteurs sur les spectres synthétiques.

2.5 Architecture du projet

Dans le but de pouvoir facilement créer et générer une variété de spectres de même qu'effectuer leur ajustement efficace, une architecture orientée-objet hautement efficace est présentée. Le code entier est disponible sur GitHub, au lien fourni à la fin de la section 1.

Pour la création des spectres, l'objet `CustomGaussian` permet d'implémenter un modèle gaussien possédant des paramètres variables tel que discuté à la section 2.1. Cette classe fournit la méthode `evaluate`, qui permet d'évaluer le modèle à différentes abscisses un nombre n de fois, en permettant la variation des paramètres selon la largeur de distribution spécifiée. La fonction `get_plot` permet quant à elle d'illustrer le modèle afin de visualiser la variation de ses paramètres. Notons d'ailleurs que d'autres modèles pourraient être implémentés en autant que ceux-ci respectent le protocole établi par `CustomModel`. Afin de combiner ces modèles pour former un spectre, l'objet `Spectrum` peut être utilisé. Cette classe relie une liste de modèles aux paramètres spatiaux, soit le bruit à ajouter de même que le nombre de canaux. Tout comme la classe `CustomModel`, les objets `Spectrum` peuvent être évalués n fois avec la méthode `evaluate`, qui considère la somme de chaque modèle composant le spectre. De plus, l'ensemble des modèles présents dans le spectre ainsi que l'amplitude du bruit induit peuvent être illustrés grâce à la propriété `plot`. Finalement, les spectres peuvent facilement être sauvegardés et chargés à partir des méthodes `save` et `load` pour pouvoir être réutilisés efficacement. Ces classes sont créées dans le dossier `projet/src/models`.

Afin de relier les données d'entraînement aux paramètres réels des distributions, deux versions du protocole `SpectrumDataObject` sont implémentées dans le dossier `projet/src/data_structures`, soit `SpectrumDataArray` et `SpectrumDataset`. Ces deux classes peuvent s'instancier à partir d'un spectre et d'un nombre d'échantillonnages et conservent alors les données générées, ainsi que les valeurs des paramètres de chaque modèle ayant été utilisées pour créer chaque spectre. La première classe implémente le stockage de ses attributs à partir de matrices NumPy, faciles à manipuler pour l'ajusteur SciPy, tandis que la deuxième classe utilise plutôt des tenseurs PyTorch, qui sont nécessaires pour utiliser l'interface éponyme.

Finalement, le dossier `projet/src/fitters` contient l'ensemble des méthodes d'ajustement conçues. Pour la méthode de régression déterministe, l'ajustement par la fonction `scipy.optimize.curve_fit` est implémenté dans la classe `ScipyFitter`. Cette classe est instanciée en ne fournissant que des objets `SpectrumDataArray` et contient la méthode `fit`, qui permet, grâce à une matrice d'estimations initiales, de faire l'ajustement des modèles sur le spectre et de fournir en sortie les paramètres optimaux. Notons d'ailleurs que l'algorithme pour établir les estimations initiales est implémenté dans `projet/src/spectrums/initial_guesses.py`. Pour les méthodes de DL, celles-ci héritent de la classe de base `BaseNeuralNetworkFitter`, classe héritant elle-même de `torch.nn.Module`, qui implémente avec la méthode `training_loop` une boucle d'entraînement suivant les paramètres discutés à la section 2.3. De plus, cette classe facilite le calcul des prédictions ainsi que de la perte du modèle

avec les méthodes `predict` et `compute_loss` respectivement, et les poids entraînés peuvent être sauvegardés et chargés par les méthodes `save` et `load`. Dérivant de cette classe, l'ajusteur CNN `CNNFitter` et le réseau ResNet `ResNetFitter` implémentent chacun la méthode `forward`, qui permet d'effectuer la propagation des données dans le réseau en effectuant les opérations de chaque couche de manière séquentielle, en tirant grandement profit de l'interface PyTorch. Différentes métriques de score sont implémentées dans le fichier `score.py` afin de pouvoir comparer quantitativement les modèles étudiés, comme discuté à la section 2.4.

Somme toute, l'architecture présentée a été conçue dans l'optique de pouvoir facilement être étendue, que ce soit pour la création de nouveaux spectres, l'inclusion de nouveaux modèles et de spectres composés ou bien l'implémentation de nouvelles méthodes d'ajustement. Les protocoles comme `CustomModel`, `BaseNeuralNetworkFitter` et `SpectrumDataObject` visent ainsi à faciliter l'implémentation d'autres classes héritant de ces protocoles.

3 Résultats

Dans cette section, nous présentons les performances obtenues par chacune des méthodes d'ajustement, soit l'approche déterministe basée sur `scipy.optimize.curve_fit` ainsi que les ajusteurs par CNN et ResNet. Les résultats sont évalués selon les deux métriques introduites précédemment : le coefficient de détermination R^2 et la MSE. Les processus d'entraînement sur les 7 types de spectres et les 3 niveaux de bruit, pour un total de 21 entraînements, ont étés effectués en tirant profit de la puissance de calcul des processeurs graphiques (GPU), sur une carte NVIDIA GeForce RTX 4080. Pour l'ajusteur CNN, tous les entraînements ont étés effectués en un temps d'environ 45 minutes alors que les entraînements du ResNet ont duré environ 1 heure et 10 minutes. L'entraînement du ResNet implique ainsi une augmentation d'environ 50% par rapport à l'entraînement du CNN simple.

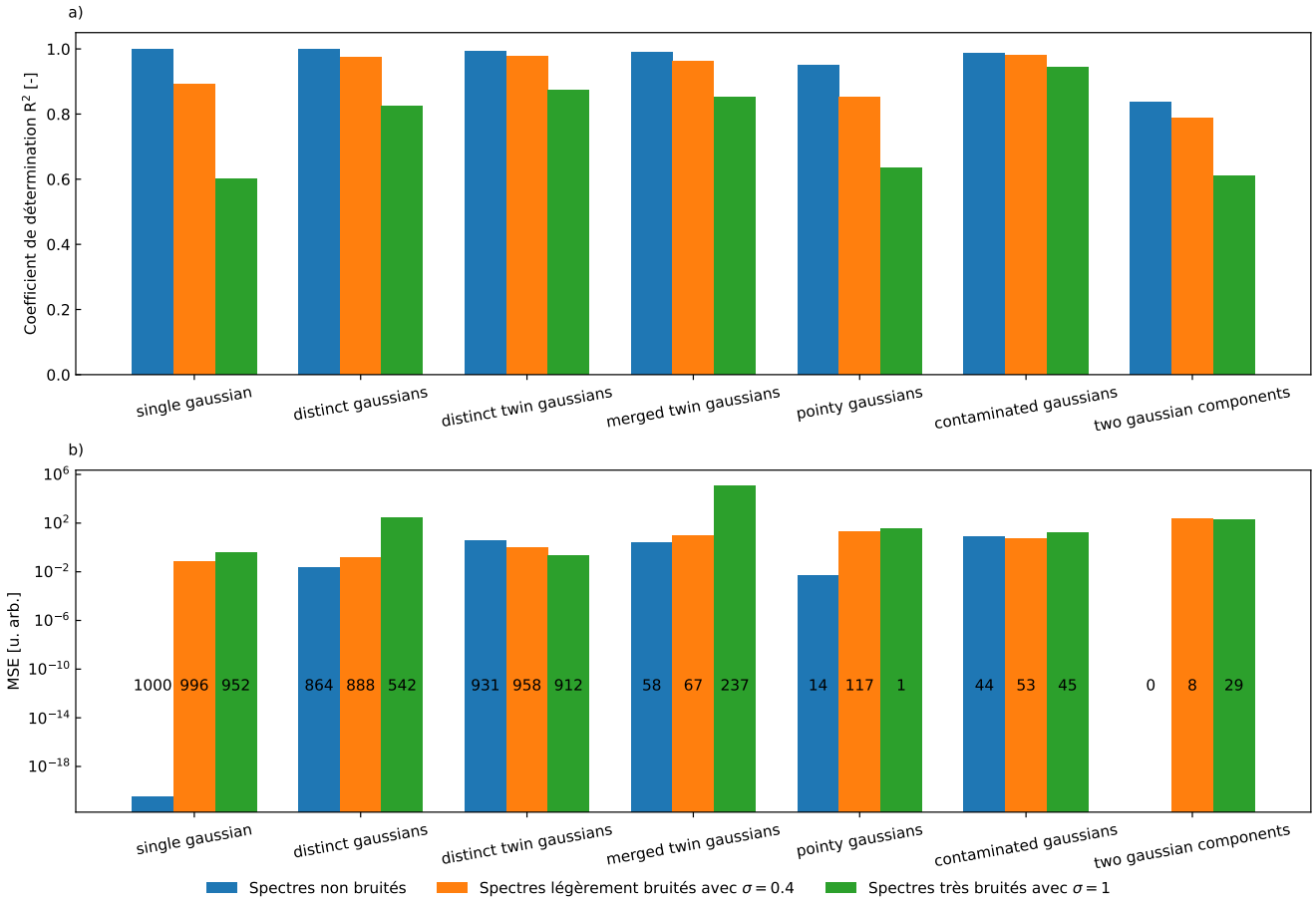

FIGURE 3 – Mesures de score pour l'ajusteur Scipy pour a) le coefficient de détermination R^2 et b) la MSE. Les résultats sont obtenus d'un échantillon de 1000 spectres basés sur les sept modèles de référence pour trois niveaux de bruit, tels que présentés à la Fig. 1. Les nombres sur les barres en b) représentent le nombre de spectres sur 1000 pour lequel l'algorithme a convergé vers un nombre de composantes égal au nombre de composantes du modèle, permettant le calcul de la MSE.

La Fig. 3 montre le coefficient de détermination R^2 et la MSE obtenus par l'ajustement de l'algorithme de Scipy sur un échantillon de 1000 spectres générés pour chacun des sept modèles de référence, pour trois niveaux de bruit. Cette figure souligne ainsi la comparaison entre la qualité des ajustements selon les différents modèles. La MSE a seulement pu être calculée pour les spectres qui ont convergé vers un nombre de raies équivalent au nombre de raies du modèle. Les nombres indiqués sur les barres de la Fig. 3 b) représentent alors la quantité de spectres pour laquelle la MSE a pu être calculée. La Fig. 4 présente le coefficient de détermination R^2 et la MSE calculés suite à l'entraînement sur chaque spectre des algorithmes de DL. Cette figure permet ainsi la comparaison directe des deux algorithmes selon les métriques de score introduites précédemment.

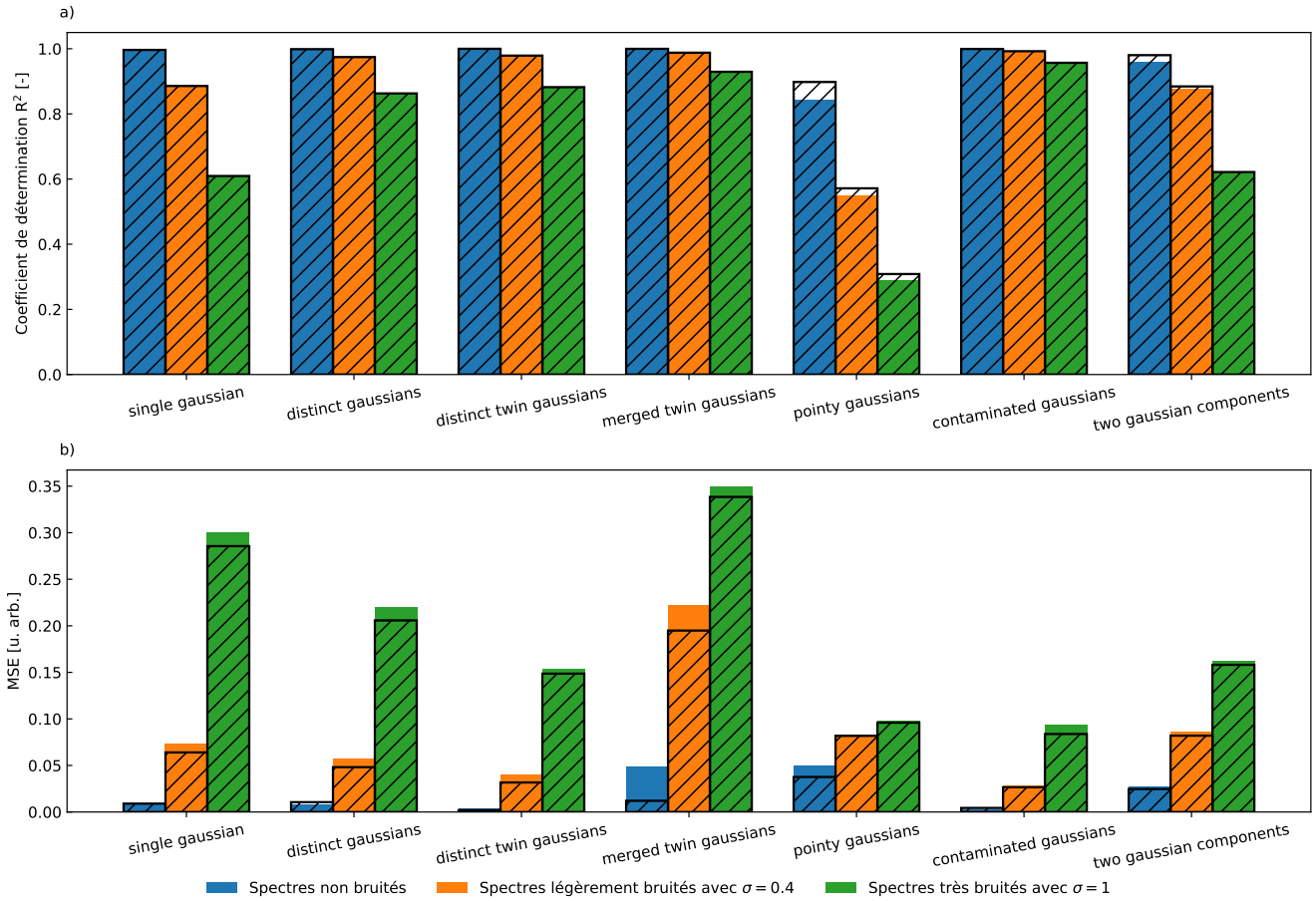

FIGURE 4 – Mesures de score sur l'ensemble de données test après l'entraînement de l'ajusteur CNN (barres pleines) et l'ajusteur ResNet (barres hachurées) pour a) le coefficient de détermination R^2 et b) la MSE. Les spectres ajustés sont ceux présentés à la Fig. 1.

La Fig. 5 illustre des exemples de décomposition effectués avec les ajusteurs ScipyFitter et ResNetFitter. La comparaison entre les deux courbes montre la différence de convergence des deux algorithmes. Les sous-figures a), c) et d) illustrent la capacité accrue de l'ajusteur ResNet à modéliser les tendances complexes comparé à l'ajusteur SciPy. Inversement, la sous-figure b) montre la meilleure performance de l'ajusteur Scipy sur des données simples, et la déviation du ResNet.

Afin de comprendre le fonctionnement des méthodes de DL, il peut être intéressant d'étudier la convergence de celles-ci, soit la rapidité avec laquelle les ajusteurs trouvent des paramètres optimaux. Pour visualiser cette convergence, des courbes d'apprentissage des différentes méthodes peuvent être réalisées, offrant la possibilité de prédire l'effet d'un nombre plus grand de données d'entraînement et permettant de faciliter l'ajustement des hyperparamètres [25]. La Fig. 6 illustre ainsi la progression de la MSE sur l'ensemble de données d'entraînement et de validation en fonction de l'époque pour les deux ajusteurs DL sur la version moyennement bruitée du spectre *distinct twin gaussians*. Notons que le choix de l'ensemble de données est arbitraire et que la tendance illustrée est cohérente pour tous les spectres.

Finalement, la Fig. 7 offre un exemple pour illustrer l'importance du choix des hyperparamètres advenant une utilisation réelle des méthodes présentées dans ce travail. Notons que ces entraînements ont été effectués seulement dans le but d'illustrer l'influence du taux d'apprentissage sur la solution trouvée ; ils ne constituent donc pas des entraînements utilisés pour la comparaison de la

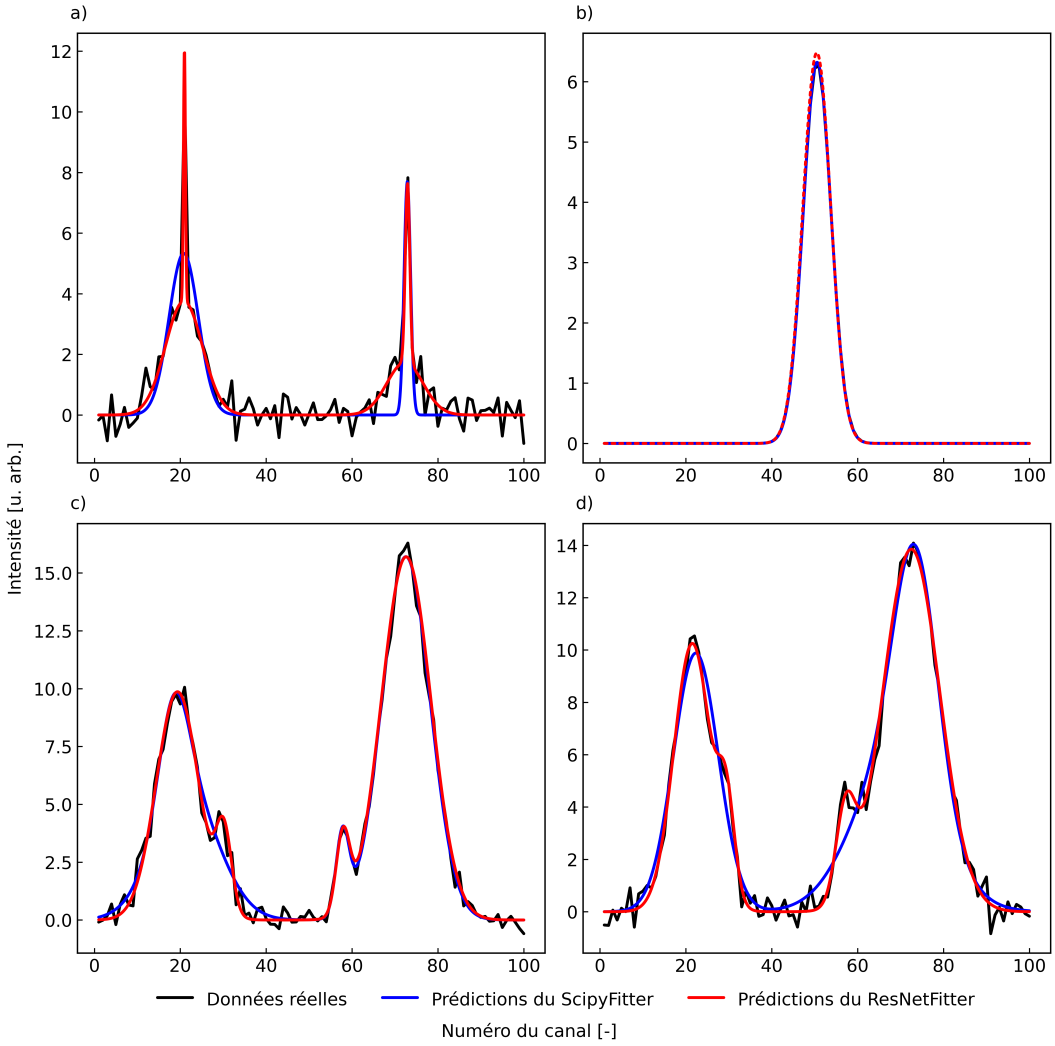

FIGURE 5 – Exemples de résultats des ajustements pour les spectres a) *two_gaussian_components* ($\sigma = 0.4$), b) *single_gaussian* ($\sigma = 0$) et deux exemples de *contaminated_gaussian* ($\sigma = 0.4$) en c) et d) pour Scipy et ResNet.

performance des ajusteurs. Contrairement aux autres entraînements, trois valeurs de taux d'apprentissage initial ont été utilisées, soit 0.01, 0.001 et 0.0001. De plus, le nombre d'époques a été exceptionnellement fixé à 50 pour permettre d'observer des tendances globales.

4 Discussion

La Fig. 3 a) montre que la méthode Scipy permet d'obtenir des valeurs de R^2 très près de 1 pour les données non-bruitées, ce qui reflète un bon ajustement. Toutefois, comme expliqué à la section 2.4, le coefficient R^2 n'est pas nécessairement représentatif de la qualité d'un ajustement, puisqu'il dépend du bruit des données. Ainsi, les valeurs de R^2 bien plus faibles pour les spectres très bruités ne signifient pas immédiatement que la méthode Scipy est incapable de traiter ces données.

La Fig. 3 b) illustre toutefois la performance limitée de la méthode Scipy. Pour chaque type de spectre et chaque niveau de bruit, le nombre d'itérations sur 1000 pour lesquelles l'algorithme a convergé vers le bon nombre de gaussiennes est inscrit sur la barre correspondante. Les spectres *single gaussian*, *distinct gaussians* et *distinct twin gaussians* sont les seuls pour lesquels ce nombre d'itérations représente une fraction significative (plus de 50 %) de l'échantillon de départ. Ces spectres sont ceux présentant la structure la plus simple, ce qui semble indiquer que l'algorithme de Scipy rencontre des difficultés de convergence pour des spectres plus complexes, comme le montre la Fig. 5. Par ailleurs, à prime abord, en s'intéressant uniquement au coefficient R^2, le spectre *contaminated gaussians* semble particulièrement bien ajusté par cette méthode, et ce, pour les trois niveaux de bruit. Or, cette méthode ne converge que rarement vers le bon nombre de gaussiennes, ce qui montre que la portée du R^2 est limitée. Il peut également sembler étrange que la méthode Scipy converge plus souvent vers le bon nombre de gaussiennes pour des spectres

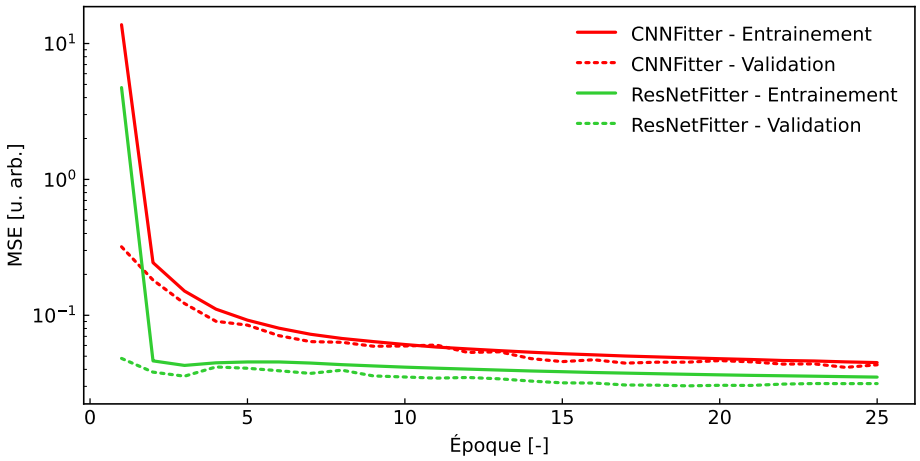

FIGURE 6 – Courbes d'apprentissage illustrant la MSE sur les ensembles de données d'entraînement et de validation en fonction de l'époque d'entraînement. L'ensemble de données provenant du spectre moyennement bruité *distinct twin gaussians* a été utilisé pour illustrer la convergence du CNN simple et du ResNet.

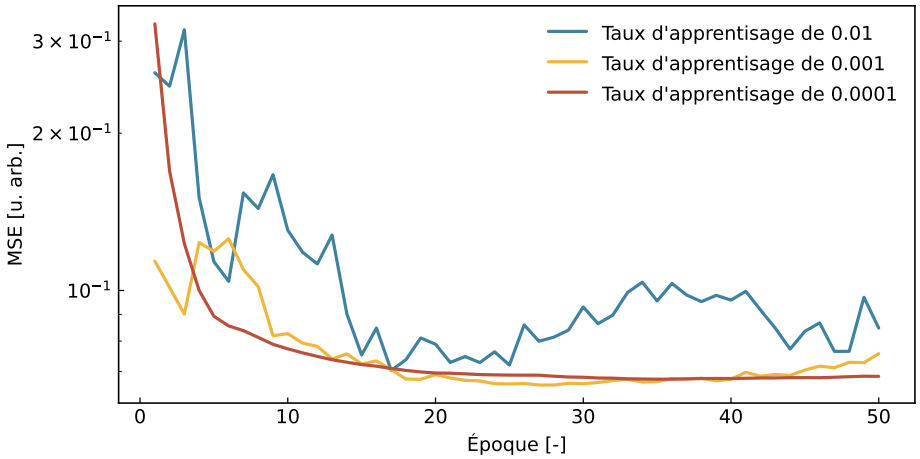

FIGURE 7 – Courbes d'apprentissage du ResNet illustrant la MSE en fonction de l'époque d'entraînement pour trois taux d'apprentissage, entraîné sur l'ensemble de données *single gaussian* avec un bruit modéré. Contrairement à tous les autres entraînements effectués pour comparer la performance des deux réseaux, celui-ci est effectué sur 50 époques pour illustrer la tendance globale de la MSE.

bruités. Cela signifie que la méthode interprète le bruit comme une gaussienne et l'ajuste en conséquence, ce qui constitue une autre limite à sa performance.

La Fig. 4 a) illustre la tendance des deux algorithmes DL à bien s'agencer aux données lisses, représentée par un coefficient R^2 très près de l'unité et ce dernier diminue avec l'ajout de bruit tel qu'attendu. De plus, il est possible de remarquer que les deux algorithmes ont des performances très similaires sur la plupart des spectres, mais que le ResNet semble performer légèrement mieux sur les données de gaussiennes très pointues (*pointy gaussians*) et les gaussiennes à composantes large et étroite (*two gaussian components*) comme le montre la Fig. 5. L'ensemble de données des gaussiennes pointues offre des composantes très étroites alors que celui des gaussiennes à deux composantes présente des tendances à plus large échelle. Il n'est donc pas étonnant que le Resnet soit en mesure de mieux modéliser ces deux spectres puisque l'architecture de ce réseau est plus profonde et combine des couches de convolution avec des noyaux de taille 7, 5 et 3. Cette architecture variée favorise ainsi la compréhension de composantes d'échelles différentes.

La Fig. 4 b) confirme également le fait que le ResNet performe mieux en moyenne que le CNN simple. On remarque cette fois que cette différence est surtout présente pour l'ensemble de données des gaussiennes fusionnées (*merged twin gaussians*), qui est formé de deux composantes très rapprochées et très larges. Ainsi, il n'est pas étonnant que le CNN simple performe moins bien

sur l'ajustement de ces gaussiennes larges, étant formé d'uniquement deux convolutions avec un noyau d'une taille de 5 pixels. Il peut cependant sembler étonnant pour l'ensemble de données des gaussiennes fusionnées (*merged twin gaussians*) qu'un très bon R^2 soit obtenu, mais que la MSE du CNN simple soit supérieure à celle obtenue pour tous les autres ensembles de données. La Fig. 8 illustre ainsi les prédictions des deux réseaux neuronaux sur des spectres sélectionnés selon l'importance de la déviation de la prédiction du CNN simple. On remarque que la somme des composantes ajustées se superpose aux données réelles, expliquant pourquoi le facteur R^2 est très près de 1. Cependant, puisque les composantes réelles ont des dispersions très similaires dans les trois spectres illustrés, la somme de ceux-ci se rapproche très près d'une gaussienne, et la contribution de chaque composante devient très difficile à distinguer. Par conséquent, le CNN simple arrive à un bon ajustement global, sans toutefois être en mesure de distinguer les composantes individuelles clairement. De son côté, par la profondeur grandement accrue de son architecture, le ResNet est bien meilleur pour ajuster les composantes individuelles. Pour les autres spectres, on remarque la non-représentativité du facteur R^2 : bien que celui-ci soit très haut, les paramètres ajustés ne sont pas nécessairement aussi bons.

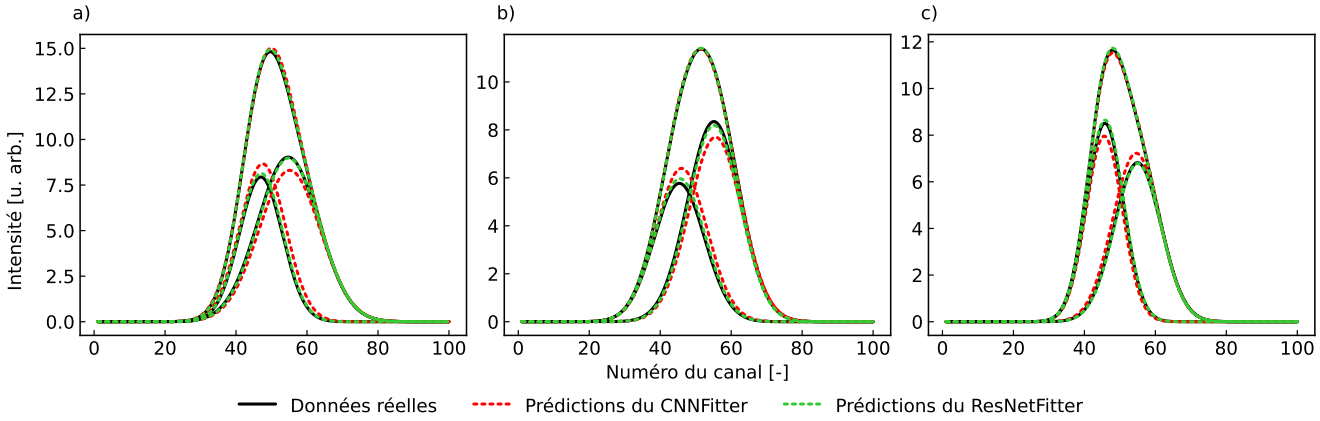

FIGURE 8 – Spectres de l'ensemble de données *merged twin gaussians* (ligne pleine noire) et prédictions du CNN simple (ligne pointillée rouge) et du ResNet (ligne pointillée verte). Les spectres présentés ont été manuellement sélectionnés pour illustrer des situations de déviation des prédictions individuelles du *CNNFitter*.

La comparaison des trois ajusteurs implémentés offre des conclusions révélatrices sur leur fonctionnement. La méthode SciPy, minimisant l'erreur quadratique, converge facilement vers la solution optimale pour les ensembles de données simples (*single gaussian*, *distinct gaussians* et *distinct twin gaussians*), contrairement aux méthodes de DL qui ne tiennent pas compte de cette erreur. Nous pouvons conclure qu'une régression déterministe, basée sur un algorithme de minimisation des carrés tel que l'algorithme de Levenberg-Marquardt, est à favoriser pour des spectres simples, de par sa facilité à être mise en place et par les hauts taux de convergence obtenus pour les ensembles de données *single gaussian*, *distinct gaussians* et *distinct twin gaussians*. Inversement, lorsqu'il est question de spectres avec des superpositions complexes, les algorithmes de DL sont bien mieux outillés pour effectuer leur ajustement convenablement puisqu'il est possible de les entraîner directement avec des solutions connues. Dans ce contexte, l'ajusteur SciPy éprouve de grandes difficultés à converger vers le minimum global, surtout lorsqu'un grand nombre de paramètres entre en jeu : la descente du gradient peut avoir tendance à converger vers des minimums locaux où l'amplitude de certaines gaussiennes est réduite à zéro et où d'autres composantes sont grandement élargies pour qu'elles ajustent plus d'une composante. Par comparaison, les algorithmes de DL sont plus robustes à un grand nombre de paramètres, et il est facile d'effectuer un entraînement plus long et sur un ensemble de données plus grand pour atteindre la convergence des ajusteurs. Enfin, la bonne détermination des estimations initiales pour la méthode SciPy est un défi très complexe et différent pour chaque spectre, mais essentiel pour permettre l'ajustement du nombre de composantes désirées. Ainsi, tel que corroboré par les meilleurs scores des méthodes de DL, nous établissons que les méthodes d'apprentissage automatique sont préférables pour l'étude de spectres à formes complexes et à composantes superposées.

D'autre part, l'étude de la Fig. 6 révèle qu'autant pour l'erreur d'entraînement que pour celle de validation, l'ajusteur ResNet converge vers une erreur minime bien plus rapidement que l'ajusteur CNN. Cette conclusion est telle qu'attendue, puisque les ResNet ont été conçus dans l'optique de permettre une convergence plus rapide que d'autres méthodes. En plus de cette convergence rapide, on remarque que l'ajusteur ResNet conduit à une solution finale meilleure que l'ajusteur CNN. Ce résultat est cohérent avec les tendances discutées précédemment. Étant donné que le temps d'entraînement du ResNet est seulement 1.5 fois plus grand que celui du CNN simple, il est clair qu'il est bénéfique de prioriser ce modèle en raison de sa convergence rapide, même si son temps d'entraînement est plus élevé.

Dans le but de présenter des méthodes pouvant être appliquées dans des contextes réels, la Fig. 7 illustre l'importance du choix du

taux d'apprentissage, soit l'un des hyperparamètres les plus importants. La courbe bleue illustre le choix d'un taux d'apprentissage bien trop grand pour le problème : l'erreur oscille fortement sans vraiment converger vers un minimum clair, ce qui indique que la descente du gradient n'arrive pas à déterminer le minimum global de la fonction de coût. La courbe jaune illustre quant à elle un bien meilleur choix de taux d'apprentissage : bien que possédant quelques oscillations, celui-ci converge rapidement vers un minimum avant de réaugmenter graduellement. Cette augmentation témoigne du phénomène de surapprentissage (*overfitting*), lorsque le modèle continue d'améliorer ses poids pour réduire l'erreur sur l'ensemble de données de test, au dépens d'être capable de généraliser ses apprentissages aux données de validation. Tel que discuté à la section 2.3 et dans une perspective de maximiser la capacité du modèle à généraliser à de nouvelles données, il est important de choisir les poids du modèle à l'époque où l'erreur de validation était minimale, ce qui n'est pas nécessairement synonyme de la dernière époque d'entraînement étant donné le surapprentissage possible. Finalement, la courbe rouge illustre un taux d'apprentissage très faible, permettant une décroissance monotone de l'erreur sans jamais être croissante. Un tel scénario est idéal pour trouver le minimum global de la fonction de coût, mais implique une convergence plus lente. Puisque le nombre d'époques choisi pour la comparaison des différentes méthodes est de 25, le choix d'un taux d'apprentissage de 0.001 semble justifié.

À la lumière de ces résultats, nous proposons donc une nouvelle méthode d'ajustement de spectres complexes tirant profit de l'architecture du ResNet présenté. Selon les performances et la généralisation supérieures de cette méthode, nous estimons qu'il s'agit de la plus adaptée parmi les trois étudiées. Cependant, pour appliquer cette méthode à de vraies données, plusieurs précautions doivent être considérées. Tout d'abord, l'ensemble de données d'entraînement doit être parfaitement représentatif des données réelles en incluant toutes les variations possibles de paramètres. Dans le cas contraire, le réseau ne sera pas en mesure de faire des prédictions convenables sur des spectres qu'il n'a jamais vus. C'est d'ailleurs pour cette raison que la génération des spectres fait varier les paramètres des modèles légèrement au-delà des bornes spécifiées : cela permet une meilleure exploration des paramètres. Ensuite, il est certain que le rapport signal sur bruit (SNR) doit être considéré pour analyser les composantes ajustées. Puisque le ResNet a de la difficulté à distinguer le bruit et des composantes faibles, il est nécessaire d'établir un critère sur l'amplitude de la composante identifiée, dépendant du bruit, pour s'assurer qu'il s'agit bien d'une détection. Similairement, il est nécessaire d'adapter les hyperparamètres ainsi que les paramètres internes du ResNet selon les spectres étudiés. Effectivement, le taux d'entraînement pourrait nécessiter un ajustement selon l'oscillation des courbes d'apprentissage, et la taille des noyaux de convolutions pourrait devoir être modifiée selon l'envergure des composantes à modéliser.

Finalement, pour une application robuste en physique expérimentale, il serait important d'inclure des incertitudes aux diverses prédictions du modèle, soit un sujet n'ayant pas été traité jusqu'à présent. Ces incertitudes sont facilement obtenues par la matrice de covariance lors d'un ajustement avec SciPy, mais sont moins naturelles avec des réseaux de neurones. Toutefois, plusieurs approches existent pour incorporer la prédiction d'incertitudes, dont le Monte-Carlo Drop-Out (MCDO) qui consiste à effectuer à répétition plusieurs prédictions en désactivant aléatoirement certains neurones, offrant ainsi une prédiction légèrement différente à chaque fois. Avec la distribution de prédictions générées, l'écart-type de cette distribution peut alors être calculé pour estimer l'incertitude sur les valeurs prédites [26]. Finalement, pour générer des ensembles de données plus représentatifs, plutôt que d'effectuer la variation des paramètres selon des lois normales indépendantes, il pourrait être bénéfique de considérer l'ensemble des paramètres lors de ce processus. Ainsi, puisque l'amplitude des raies observées en astrophysique notamment est reliée à leur dispersion, une telle considération de l'ensemble des paramètres offrirait la possibilité de fournir au modèle des données d'entraînement plus cohérentes avec la réalité expérimentale.

Pour finir cette section, il peut sembler que choisir les modèles et la décomposition à ajuster vient à l'encontre de l'objectivité désirée de l'analyse de données. Pourtant, plusieurs méthodes utilisées à ce jour reposent directement sur de tels choix ; c'est notamment le cas du logiciel d'analyse du gaz interstellaire neutre ROHSA, qui force l'ajustement de gaussiennes très larges pour simuler les composantes chaudes de ce milieu particulier [14]. Ainsi, il ne semble pas problématique d'imposer notre décomposition aux spectres étudiés, en autant que les fondements physiques sont suffisants.

5 Conclusion

Ce travail a permis de comparer l'efficacité de différentes méthodes d'ajustement de spectres complexes, soit un ajusteur basé sur l'optimisation déterministe (SciPy) et deux ajusteurs issus de l'apprentissage profond (CNN simple et ResNet). L'analyse de leur performance a révélé que, pour des spectres simples, l'approche déterministe fondée sur la minimisation de l'erreur quadratique offre des performances optimales, ce qui est attribué à sa capacité à converger itérativement vers une solution minimisant l'erreur quadratique, à condition de disposer d'estimations initiales cohérentes. Toutefois, lorsque la complexité spectrale augmente, notamment en présence de composantes superposées, de structures fines ou de bruit intense, les approches par réseaux neuronaux démontrent une supériorité marquée en étant capables d'apprendre des tendances complexes directement à partir de données d'entraînement. Parmi les ajusteurs CNN et ResNet étudiés, l'architecture ResNet s'est démarquée par une convergence plus rapide et une meilleure capacité de généralisation, avec des performances supérieures ou équivalentes pour tous les ensembles de données analysés. Ces résultats s'expliquent par la profondeur accrue du réseau et la variété de ses couches de convolution, favorisant une modélisation multi-échelle des spectres.

Le travail propose donc l'utilisation de l'architecture ResNet pour l'ajustement de spectres complexes. En vue d'applications expérimentales, plusieurs précautions doivent toutefois être prises pour garantir la robustesse du modèle, notamment confirmer la représentativité de l'ensemble de données d'entraînement, s'assurer de la cohérence des prédictions du modèle selon le bruit des spectres et optimiser les hyperparamètres selon le problème considéré. De plus, l'intégration d'une estimation des incertitudes sur les prédictions, par exemple via le MCDO, apparaît essentielle pour l'utilisation de cette méthode en physique expérimentale.

En conclusion, ce travail établit qu'un ajusteur basé sur un ResNet correctement entraîné pour un spectre spécifique constitue une méthode prometteuse pour l'analyse de spectres complexes. Des travaux futurs pourraient se pencher sur la comparaison d'autres réseaux, notamment le réseau récurrent Long Short-Term Memory (LSTM) qui offre une rétention de l'information bien supérieure à celle des réseaux présentés ici et qui est fréquemment utilisé en traitement de signal [27]. Dans un contexte astrophysique, il pourrait également être intéressant d'implémenter une version tridimensionnelle du ResNet proposé, permettant ainsi l'utilisation du contexte spatial des spectres pour augmenter la cohérence des prédictions effectuées.

Références

[1] Alan O Sykes. « An introduction to regression analysis ». In : *Université de Chicago* (1993).

[2] George B Rybicki et Alan P Lightman. *Radiative processes in astrophysics*. John Wiley & Sons, 2024.

[3] J Michael Hollas. *Modern spectroscopy*. John Wiley & Sons, 2004.

[4] Keith A Olive et al. « Review of particle physics ». In : *Chinese physics C* 38.9 (2014). Publisher : Chinese Physics C, p. 1-090001.

[5] Philip R Bevington et D Keith Robinson. « Data reduction and error analysis ». In : *McGraw-Hill, New York* (2003).

[6] Thomas Martin, Laurent Drissen et Gilles Joncas. « ORBS, ORCS, OACS, a software suite for data reduction and analysis of the hyperspectral imagers SITELLE and SpIOMM ». In : *Astronomical Data Analysis Software an Systems* 24.495 (2015), p. 327.

[7] Carter Rhea et al. « LUCI : A Python package for SITELLE spectral analysis ». In : *Research Notes of the AAS* 5.9 (2021). Publisher : The American Astronomical Society, p. 208.

[8] Kaiming He et al. « Deep residual learning for image recognition ». In : *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, p. 770-778.

[9] L Barriault et al. « Multiwavelength observations of cirrus clouds in the North Celestial Loop : a study of the OH emission ». In : *Monthly Notices of the Royal Astronomical Society* 407.4 (2010). Publisher : Blackwell Publishing Ltd Oxford, UK, p. 2645-2659.

[10] Edwin T Jaynes. *Probability theory : The logic of science*. Cambridge university press, 2003.

[11] Roger A Chevalier et Robert P Kirshner. « Abundance inhomogeneities in the Cassiopeia A supernova remnant ». In : *Astrophysical Journal, Part 1, vol. 233, Oct. 1, 1979, p. 154-162.* 233 (1979), p. 154-162.

[12] F Grandmont et al. « Final design of SITELLE : a wide-field imaging Fourier transform spectrometer for the Canada-France-Hawaii Telescope ». In : *Ground-based and Airborne Instrumentation for Astronomy IV*. T. 8446. SPIE, 2012, p. 267-278.

[13] Maxime Royer, Gilles Joncas et Mathieu Marquis. « Thermodynamics of the HII region Sh2-158 and "direct" measurement of its turbulence ». In : *The Astrophysical Journal* (in prep.).

[14] Antoine Marchal et al. « ROHSA : Regularized Optimization for Hyper-Spectral Analysis-Application to phase separation of 21 cm data ». In : *Astronomy & Astrophysics* 626 (2019). Publisher : EDP Sciences, A101.

[15] Pauli Virtanen et al. « SciPy 1.0 : Fundamental Algorithms for Scientific Computing in Python ». In : *Nature Methods* 17 (2020), p. 261-272. doi : 10.1038/s41592-019-0686-2.

[16] Jorge J Moré. « The Levenberg-Marquardt algorithm : implementation and theory ». In : *Numerical analysis : proceedings of the biennial Conference held at Dundee, June 28–July 1, 1977*. Springer, 2006, p. 105-116.

[17] Bogdan M Wilamowski et Hao Yu. « Improved computation for Levenberg–Marquardt training ». In : *IEEE transactions on neural networks* 21.6 (2010). Publisher : IEEE, p. 930-937.

[18] Mark K Transtrum et James P Sethna. « Improvements to the Levenberg-Marquardt algorithm for nonlinear least-squares minimization ». In : *arXiv preprint arXiv :1201.5885* (2012).

[19] A Paszke. « Pytorch : An imperative style, high-performance deep learning library ». In : *arXiv preprint arXiv :1912.01703* (2019).

[20] Alex Krizhevsky, Ilya Sutskever et Geoffrey E Hinton. « Imagenet classification with deep convolutional neural networks ». In : *Advances in neural information processing systems* 25 (2012).

[21] Serkan Kiranyaz et al. « 1D convolutional neural networks and applications : A survey ». In : *Mechanical systems and signal processing* 151 (2021). Publisher : Elsevier, p. 107398.

[22] Ian Goodfellow et al. *Deep learning*. T. 1. 2. MIT press Cambridge, 2016.

[23] Diederik P Kingma et Jimmy Ba. « Adam : A method for stochastic optimization ». In : *arXiv preprint arXiv :1412.6980* (2014).

[24] Rudolf J Freund et William J Wilson. *Statistical methods*. Elsevier, 2003.

[25] Tom Viering et Marco Loog. « The shape of learning curves : a review ». In : *IEEE Transactions on Pattern Analysis and Machine Intelligence* 45.6 (2022). Publisher : IEEE, p. 7799-7819.

[26] Yarin Gal et Zoubin Ghahramani. « Dropout as a bayesian approximation : Representing model uncertainty in deep learning ». In : *international conference on machine learning*. PMLR, 2016, p. 1050-1059.

[27] Greg Van Houdt, Carlos Mosquera et Gonzalo Nápoles. « A review on the long short-term memory model ». In : *Artificial Intelligence Review* 53.8 (2020). Publisher : Springer, p. 5929-5955.