Méthodes et outils d'analyse visuelle pour la compréhension, l'optimisation et l'élaboration de modèles de réseaux de neurones profonds

Adrien Halnaut

université de BORDEAUX

LABORATOIRE BORDELAIS DE RECHERCHE EN INFORMATIQUE LaBRI

THÈSE PRÉSENTÉE

POUR OBTENIR LE GRADE DE

DOCTEUR
DE L'UNIVERSITÉ DE BORDEAUX

ÉCOLE DOCTORALE DE MATHÉMATIQUES ET D'INFORMATIQUE

Spécialité Informatique

Par **Adrien HALNAUT**

Méthodes et outils d'analyse visuelle pour la compréhension, l'optimisation et l'élaboration de modèles de réseaux de neurones profonds.

Sous la direction de : **David AUBER**
Co-directeur : **Romain GIOT**

Soutenue le 22 mars 2024

Membres du jury :

Méthodes et outils d'analyse visuelle pour la compréhension, l'optimisation et l'élaboration de modèles de réseaux de neurones profonds

M. Arnaud SALLABERRY	Maître de Conférences	Université Paul-Valéry Montpellier III	Rapporteur
M. Gilles VENTURINI	Professeur	Université de Tours	Rapporteur
M. Nicolas BONICHON	Maître de Conférences	Université de Bordeaux	Examinateur
Mme. Pascale KUNTZ	Professeure	Université de Nantes	Présidente du jury
M. David AUBER	Professeur	Université de Bordeaux	Directeur
M. Romain GIOT	Maître de Conférences	Université de Bordeaux	Co-Directeur
M. Romain BOURQUI	Maître de Conférences	Université de Bordeaux	Invité

Résumé : Les méthodes d'apprentissage profond sont grandement utilisées dans une multitude de domaines de recherche et industriels, notamment pour résoudre des tâches de classification de données. Cependant, cette technologie est souvent associée à un modèle considéré comme une «boîte noire». L'utilisateur peut comprendre les données en entrée et en sortie du réseau entraîné, mais ne connait pratiquement rien du fonctionnement interne du réseau. Cet aspect rend difficile la justification des prédictions des modèles. L'explicabilité et l'interprétabilité des réseaux de neurones est un domaine de recherche qui regroupe différentes communautés scientifiques. Il a pour objectif de faciliter la compréhension du fonctionnement des réseaux de neurones aux yeux des utilisateurs et experts. La visualisation d'information est l'une des techniques employées pour répondre à ce besoin. Elle concerne l'élaboration d'outils facilitant la compréhension et l'analyse de jeux de données, habituellement à plus de deux dimensions, au moyen de représentations visuelles et d'interactions.

Dans cette thèse, nous exploitons les informations extraites en sortie de chaque couche des réseaux pour interpréter leurs décisions via des méthodes de visualisation. Dans un premier temps, nous montrons qu'il est possible de représenter des groupes d'instances traitées similairement par un réseau au moyen d'un diagramme de Sankey. Ce processus demande le traitement de données en quantité importante, que nous avons conduit en exploitant des infrastructures de calculs à large échelle issues du domaine du BigData.

Pour étudier des scénarios plus complexes, qui impliquent des jeux de données plus grands et des architectures de réseaux plus lourdes, nous développons des méthodes de visualisation compacte. Nous utilisons deux approches : l'une concerne la représentation de la proximité des éléments en les projetant dans l'espace $\mathbb{N}$, l'autre en appliquant un post-traitement sur des projections $\mathbb{R}^N \rightarrow \mathbb{R}^2$ pour en former des grilles compactes de données. Pour évaluer l'efficacité de ces méthodes de projections, nous avons mis en place un protocole d'évaluation utilisateur. Celui-ci permet de mesurer la pertinence des méthodes de visualisations pour résoudre des tâches liées à la compréhension de jeux de données dans $\mathbb{R}^N$. Nous avons finalement conduit une évaluation suivant notre protocole pour comparer l'efficacité de la visualisation compacte avec celle de la visualisation par nuages de points. Cette évaluation est menée en utilisant deux méthodes de projection de l'état de l'art, t-SNE et Self-Sorting Maps.

Mots-clés : visualisation, réseaux de neurones, abstraction de données, données massives, visualisation compacte

---

Visual analytics for understanding, optimizing and developing deep neural network models

Abstract: Deep-learning methods are widely used in a variety of research and industrial domains, especially in the data classification task. However, this technology is often notoriously compared to a "black box". The user can understand input and output data of the network, but has little to no knowledge about its internal processing. This aspect of neural networks makes difficult to justify their predictions. Explainability and Interpretability of deep neural networks is a research domain merging with a variety of scientific communities. Its goal is to make easier the understanding of neural networks for both users and experts. Information visualization is one of the techniques used to answer this need. It consists in building tools which make easier the understanding and the analysis of usually high dimensional datasets, using visual abstractions and interactions.

In this thesis, we make use of data extracted from the output of each layer of the neural network to interpret the model decisions using visualization methods. First, we show it is possible to visualize groups of samples processed similarly by the network using a Sankey diagram. This method asks for large data processing, which we enable by using machine clusters infrastructures used in BigData operations.

In order to study more complex scenarios, involving larger datasets and heavier network architectures, we develop compact data visualization methods. We propose two approaches: the first one implies representation of data proximities using data reduction to the $\mathbb{N}$ space, the other one implies post-processing to $\mathbb{R}^N \rightarrow \mathbb{R}^2$ data projections to build compact grids of data. In order to evaluate the performances of these projection methods, we propose a user study protocol. Its goal is to measure the suitability of visualization methods in tasks related to the understanding of high-dimensional data. Finally, we carry out an evaluation following this protocol to compare the efficiency between compact data visualization and scatter plot visualization. This evaluation is conducted using state of the art methods t-SNE and Self-Sorting Maps.

Keywords: data visualization, deep neural networks, data abstraction, big data, compact visualization

---

Unité de recherche : LaBRI (UMR 5800) 351 Cours de la Libération, 33405 Bordeaux, France.

Remerciements

Je tiens avant tout à exprimer ma profonde gratitude à mon directeur de thèse, David Auber, et à mon co-directeur de thèse, Romain Giot. Leur soutien constant, leurs conseils et leur accompagnement tout au long de ce travail de recherche m'ont été d'une aide précieuse dans la réalisation de cette thèse. Les nombreux échanges, tant en personne qu'à distance, ont enrichi ma réflexion mais m'ont également soutenu moralement lors des moments difficiles. Travailler avec eux a été un véritable plaisir, du début de la thèse jusqu'à la soutenance.

Je remercie également Romain Bourqui, Frédéric Lalanne et les membres de l'équipe BKB pour leur aide précieuse et leurs contributions à la réalisation des travaux présentés dans ce manuscrit.

Mes remerciements s'adressent également aux membres du jury de thèse, Arnaud Sallaberry, Gilles Venturini, Nicolas Bonichon et Pascale Kuntz, pour avoir accepté de relire et d'évaluer mon travail, ainsi que pour leurs corrections pour l'amélioration de ce manuscrit.

Je souhaite exprimer ma reconnaissance à Géraud Senizergues, qui a encadré mon stage de fin de Licence. Ce stage a éveillé mon intérêt pour la recherche en Informatique et m'a conforté plus tard dans ma décision de poursuivre en thèse après le Master.

Un grand merci à mes collègues doctorants au LaBRI, Loann, Luc, Abel, Emile, Christophe et Marie. Nos échanges réguliers et notre entraide ont rendu cette entreprise d'autant plus agréable. Merci également à Boris et Kostia, dont les discussions, bien que nos thèses traitent de domaines complètement différents, ont toujours été enrichissantes. Je remercie également mes amis, anciens camarades de promotion ou d'ailleurs, pour leur soutien constant tout au long de cette thèse. Vos encouragements m'ont profondément touché et motivé.

Enfin, je tiens à exprimer ma gratitude la plus chaleureuse à ma famille pour leur soutien infaillible depuis le début de mes études jusqu'à l'aboutissement de cette thèse. Ce fut un long chemin, mais je peux maintenant vous dire que cela en valait le coup.

Table des matières

Chapitre 1

Introduction

Les méthodes d'apprentissage par réseaux de neurones profonds (aussi communement appelé *deep learning*) forment un sous-ensemble de techniques d'intelligence artificielle qui consiste à construire un modèle mathématique composé de nombreux paramètres. Ces paramètres ont la possibilité de se configurer de manière autonome pour répondre à un problème donné avec une intervention partielle de son concepteur. Ce paramétrage automatique intervient lors de la phase d'apprentissage du modèle. Elle consiste à alimenter le réseau de données de références indiquant les valeurs de retour attendues (aussi appelées étiquettes ou vérités terrain) en fonction de la donnée d'entrée souhaitée. L'algorithme de descente de gradient permet alors d'optimiser les nombreux paramètres du réseau afin de traiter correctement des nouvelles données d'entrées absentes des données utilisées lors de l'apprentissage.

Depuis la conception du modèle de perceptron [117], un algorithme inspiré du fonctionnement des neurones biologiques, les réseaux de neurones profonds et leur utilisation ont grandement évolué. Cette évolution est liée (1) au développement d'architectures de réseaux spécialisées pour répondre à des tâches impliquant des types de données particuliers, (2) au développement d'algorithmes plus efficaces pour optimiser le modèle mathématique du réseau, (3) au développement de méthodes efficaces d'implémentation de la descente de gradient stochastique sur les processeurs graphiques.

Cependant, l'avènement des réseaux de neurones artificiels reste récent, notamment car les recherches sur ce sujet ont été freinées par deux verrous majeurs. Le premier verrou était technologiquement lié au coût computationnel des méthodes d'apprentissage. Ces méthodes demandaient des quantités de calculs trop grandes pour être alors traitées dans un temps raisonnable par les machines de calcul. Avec l'arrivée des processeurs graphiques permettant de réduire considérablement les temps de calculs matriciels et le développement de nouvelles méthodes de calcul de descente de gradient, ces contraintes techniques ne sont maintenant plus un problème pour le développement de réseaux de neurones artificiels. Le deuxième verrou était dû au manque de données accessibles nécessaires pour entraîner et tester les réseaux. L'entraînement de ces réseaux demande en effet de grandes quantités de données de référence, ce qui nécessite le besoin à la fois de meilleures capacités de stockage et d'un travail d'étiquetage de données considérable. Ce problème est aujourd'hui résolu avec l'avènement du *Big Data*, qui a fortement augmenté le nombre de bases de données collectées et référencées sur de nombreuses thématiques, pouvant alors servir à l'apprentissage. La disponibilité de ces données a permis une démocratisation de l'apprentissage statistique, et donc des modèles de réseaux de neurones. Aujourd'hui, des bibliothèques logicielles permettent à un néophyte de construire son propre réseau de neurones profonds sur son ordinateur personnel et de lui apprendre à résoudre une tâche de classification en quelques minutes seulement.

En parallèle, un autre «problème» a progressivement fait surface pendant que les réseaux gagnaient en complexité : celui de l'interprétation des résultats de ces réseaux ainsi que leur explicabilité. Le grand nombre de paramètres définis lors de la construction du réseau ainsi que la quantité d'informations utilisées pour son apprentissage rendent l'explication du fonctionnement

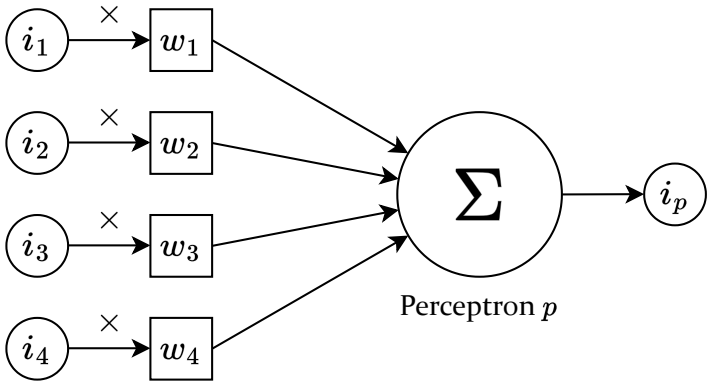


FIGURE 1.1 – Modèle du perceptron ayant 4 valeurs en entrées. Ces valeurs d'entrées $i_1, ..., i_4$ sont multipliées par des poids $w_1, ..., w_4$ qui leur sont propres, puis sont additionnées pour former la valeur i_p en sortie du perceptron. Ce modèle est la base du neurone artificiel utilisé en *deep learning*.

du réseau compliquée. En plus de cela, la capacité du réseau à résoudre les tâches pour lesquels il a été conçu n'est mesurable que sur un ensemble fini de scénarios. Ses performances sur des nouveaux scénarios ne sont pas établies, seulement estimées. L'utilisation industrielle des réseaux de neurones, notamment dans le secteur médical, pose des problèmes éthiques et juridiques. C'est pourquoi la thématique sur l'explicabilité et l'interprétabilité des réseaux de neurones gagne en importance au sein de la communauté scientifique depuis quelques années [5].

1.1 Historique des réseaux de neurones profonds

En 1958, Rosenblatt propose le modèle du perceptron [117], présenté en Figure 1.1, un système émulant le fonctionnement de neurones biologiques, capable d'apprendre par expérience. Ces neurones artificiels sont des classifieurs linéaires, c'est-à-dire que la classification est basée sur une combinaison linéaire de paramètres. Ici, les valeurs d'entrée $i_1, i_2, ..., i_k$ du perceptron sont multipliées par leurs paramètres associés $w_1, w_2, ..., w_k$ puis sommées pour retourner une valeur i_p dans $\mathbb{R}$ qui est la valeur d'activation du neurone artificiel. Les paramètres du perceptron sont appelés poids (*weights*) et sont associés aux sorties d'autres neurones afin de récupérer leur valeur d'activation. Ils peuvent aussi être associés directement à la donnée d'entrée du réseau, ce qui forme alors l'entrée du réseau de neurones. La valeur calculée par le dernier neurone du réseau est quant à elle appelée prédiction. Elle correspond à la décision du réseau pour une donnée d'entrée. Ce système de connexions entre les neurones artificiels permet de faire transiter leurs décisions à travers un ensemble d'autres neurones, ce qui explique alors l'appellation de réseau de neurones artificiels. Les poids sont quant à eux accordés en utilisant un algorithme d'apprentissage et un ensemble de données étiquetées appelé jeu de données d'apprentissage (ou d'entraînement). Ces étiquettes, souvent appelées classes, représentent la classe d'équivalence de la donnée d'entrée. Ce sont des valeurs faisant partie du même domaine des valeurs que peuvent prendre les prédictions du réseau, elles définissent donc la façon d'interpréter les prédictions du réseau. Ce processus est appelé «phase d'apprentissage» ou «phase d'entraînement», ces termes étant souvent utilisés de manière interchangeable. Cette phase consiste à modifier les poids du réseau afin que pour un élément e du jeu de données d'apprentissage, la valeur d'activation du dernier neurone du réseau soit la même valeur que l'étiquette associée à e. Par exemple, la distinction entre deux classes de données peut être exprimée par des étiquettes définies à 0 pour un groupe, et à 1 pour le second. Une fois entraîné, le réseau fournit une probabilité d'appartenance à une classe exprimée par une valeur $i \in [0; 1]$. Cette valeur indique la confiance du modèle statistique dans la prédiction.

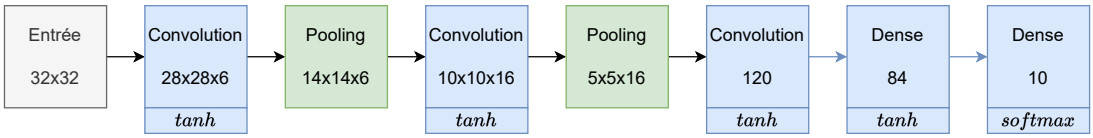


FIGURE 1.2 – Architecture du réseau LeNet5. Les couches en bleues sont celles qui «apprennent» et sont munies de fonctions d'activations (ici $tanh$ et $softmax$), les couches vertes n'ont pas de paramètres à configurer. Le réseau est composé de deux groupes de couches, le premier est composé de couches de convolution qui sont utiles pour détecter des attributs bas niveau comme des formes ou couleurs. Le deuxième groupe est composé de couches denses, capables de faire le lien entre les attributs bas niveau et des concepts de plus haut niveau.

Quelques années plus tard, deux principes viennent faire évoluer le perceptron aux réseaux de neurones profonds d'aujourd'hui : la technique de rétropropagations [27] (*backpropagation*) et les modèles multicouches [145] (*MultiLayer Perceptron*). Ces techniques sont capables de résoudre des problèmes de classification plus complexes et de manière plus efficace [71]. Les modèles multicouches sont composés de plusieurs couches (*layers*) de perceptrons liées entre elles de manière successive. Chaque perceptron d'une couche possède en entrée les sorties des perceptrons de la couche précédente. Ces couches sont appelées «couches denses» ou *fully connected layer* et l'ensemble des activations des neurones d'une couche est appelée carte d'activation. La rétropropagation consiste à modifier les paramètres des perceptrons progressivement en partant de la dernière couche du réseau vers la première.

L'évolution de l'électronique permet l'élaboration de modèles de plus en plus complexes et variés de réseaux de neurones artificiels. En 1989, LeCun et al. présentent LeNet [71], une architecture de réseau de neurones convolutif permettant de reconnaître les chiffres manuscrits. Cette architecture définie la fondation des réseaux de neurones convolutifs (*Convolutional Neural Network* ou *CNN*) qui font usage de couches de convolutions et de sous-échantillonnage. La couche de convolution est conçue spécifiquement pour la reconnaissance de données spatiales. Son fonctionnement est défini par le nombre de filtres et la taille du noyau. Ces paramètres définissent la façon dont les neurones qui composent la couche sont reliés aux sorties des neurones de la couche précédente, ce qui diffère du fonctionnement des couches denses où tous les neurones d'une couche sont connectés aux neurones de la couche précédente. En revanche, ce type de couche complexifie les données traitées en augmentant fortement leur dimensionalité, notamment si le nombre de filtres est élevé. Si un réseau mémorise trop d'informations par rapport à sa donnée d'entrée, il risque d'entrer dans une phase de surapprentissage, diminuant alors sa capacité à généraliser ses critères de classification et donc à s'adapter à des données nouvelles, absentes des données utilisées lors de son apprentissage. La couche de sous-échantillonnage (aussi appelée couche de *pooling*) réduit la taille des données traitée par le modèle en ne faisant passer à la couche suivante que les valeurs d'activations les plus importantes de la couche qui la précède. Cette couche permet entre autres d'améliorer la reconnaissance de formes sur les images en faisant abstractions des légères déformations spatiales que peut contenir la donnée d'entrée (comme le positionnement de l'objet à reconnaître par exemple). Elle permet aussi de réduire la dimensionalité des activations en sortie des couches convolutives et donc de diminuer les coûts de calculs et les risques de surapprentissage.

Cette architecture évolue en 1998 pour devenir LeNet-5 [72], présentée en Figure 1.2 et composée d'environ 6 000 paramètres. Elle devient l'architecture de référence pour les modèles convolutifs. Plus tard, des architectures encore plus complexes voient le jour, notamment grâce à l'exploitation des processeurs graphiques. En 2012, le réseau AlexNet dispose de 60 millions de paramètres [66] puis en 2014, le réseau VGG16 devient une référence de la classification d'image avec 138 millions de paramètres [124].

D'autres types d'architectures de réseaux de neurones font ensuite leur apparition. Les réseaux de neurones à mémoire (*Long Short-Term Memory* ou *LSTM*) sont présentés en 1995 par Hochrei-

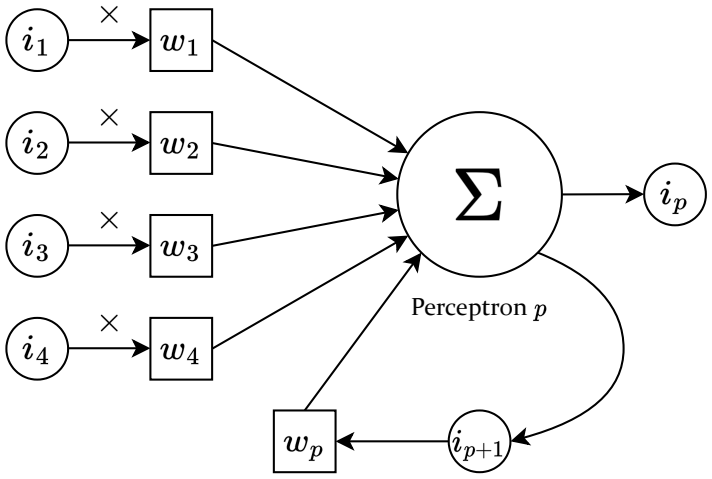


FIGURE 1.3 – Modèle simplifié d'un neurone de réseau à mémoire (dit récurent). Le neurone produit une valeur supplémentaire ayant son propre poids qui sera ajoutée au prochain calcul de sa valeur d'activation.

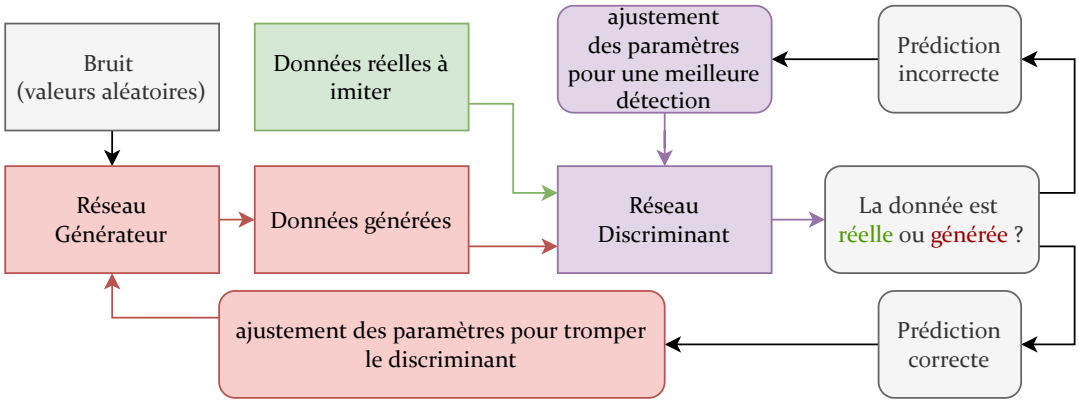


FIGURE 1.4 – Fonctionnement de l'apprentissage des *GAN*. Le réseau générateur cherche à tromper le réseau discriminant, et le réseau discriminant à discerner correctement les données réelles des données générées. La vérité terrain est connue à l'avance et permet donc d'ajuster les paramètres du discriminant en conséquence pour améliorer sa détection. Les décisions du discriminant aident le générateur à ajuster ses paramètres pour améliorer sa génération de données.

ter et al. [51] pour l'analyse de données séquentielles, telles que le traitement automatique de la parole. Ces réseaux utilisent un modèle de neurone légèrement différent du perceptron afin de conserver la valeur d'activation d'un calcul précédent, comme montré en Figure 1.3. Cette approche diffère des réseaux précédents dits à propagation avant (*feed-forward propagation*).

Les réseaux antagonistes génératifs (*Generative Adversarial Networks* ou *GAN*), sont présentés par Goodfellow et al. en 2014 [41]. Ils consistent à avoir deux réseaux s'entraînant l'un l'autre. Un de ces réseaux est le «générateur» qui génère des données suffisamment réalistes pour tromper le deuxième réseau, le «discriminant», qui lui est chargé de différencier les données réelles des générées. Le fonctionnement de ces réseaux est présenté en Figure 1.4. Cette architecture permet entre autre la génération de données de manière très réaliste [21], et sont aujourd'hui capable de tromper l'humain sur de multiples domaines d'application.

En parallèle des architectures de réseaux, de nouveaux paradigmes d'apprentissage automatique voient le jour. Nous pouvons citer notamment l'utilisation de ces réseaux en apprentissage par renforcement [131] (*Reinforcement learning*), qui consiste à laisser un réseau de neurones interagir avec un environnement. Suivant les actions du réseau, celui-ci est récompensé ou non en

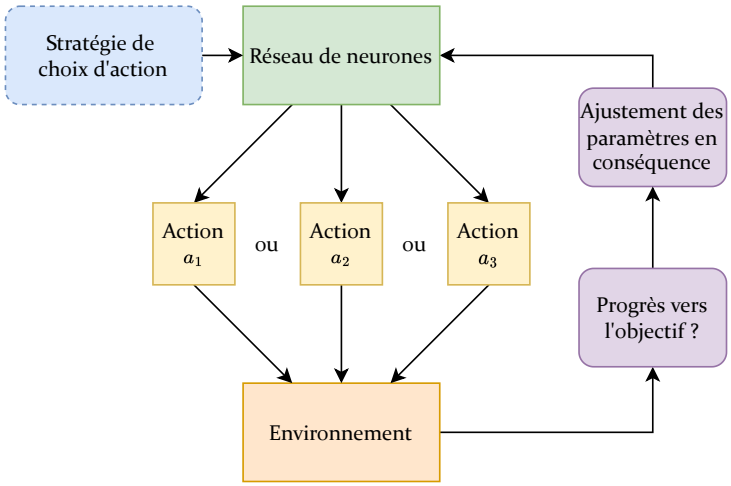


FIGURE 1.5 – Fonctionnement de l'apprentissage par renforcement. Le réseau doit accomplir un objectif et doit choisir une action parmi un ensemble proposé en se basant sur une stratégie préalablement définie par l'utilisateur. Les actions permettent d'interagir avec l'environnement et une fonction de coût permet de calculer une valeur de récompense, indiquant si l'action choisie était utile ou non à la progression vers l'objectif. Cette récompense permet ensuite au réseau d'ajuster ses paramètres en conséquence.

fonction de sa progression vers un objectif défini. Ce système de récompense permet de paramétrer le réseau afin de s'améliorer de manière itérative. La Figure 1.5 montre le déroulement d'un apprentissage par renforcement.

Enfin, au début du XXIe siècle, les progrès en matériel informatique ont grandement aidé à la recherche en intelligence artificielle en permettant notamment la construction de réseaux toujours plus complexes et le traitement de données de plus en plus grandes. Des bibliothèques logicielles [1, 19, 104], des jeux de données étiquetées [30] et des réseaux pré-entraînés (que ce soient les modèles proposés sur des plateformes communautaires comme GitHub ou Kaggle par exemple ou par des organismes de recherche) sont mis à disposition des chercheurs et néophytes et accélèrent la démocratisation de l'utilisation des réseaux de neurones. Les secteurs médicaux et industriels expérimentent aussi l'utilisation de ces techniques et la proximité de l'intelligence artificielle avec le grand public devient alors plus grande que jamais.

1.2 Les réseaux de classification

Les réseaux de classification sont les réseaux de neurones artificiels les plus utilisés dans les applications. Dans les tâches de classification, le réseau reçoit en entrée une donnée, par exemple une image, et doit la classer parmi un ensemble de concepts que le modèle a préalablement appris. Cette classification s'exprime sous la forme d'un vecteur où chaque composante correspond à un concept précis à discriminer, par exemple le nom de l'objet présenté dans l'image. La composante ayant la valeur la plus élevée du vecteur correspond à la prédiction du réseau.

Les tâches de classifications d'images sont les plus courantes. Par exemple, la reconnaissance d'écriture manuscrite [72] ou la détection de maladies biologiques à partir d'images [87], mais aussi la segmentation d'image pour la reconnaissance de route [99], la compréhension de texte [70] ou la détection de matériaux par images hyper-spectrales [74], sont des tâches de classification d'images.

L'utilisation des réseaux convolutifs est fréquente dans la résolution de ces tâches. Ces réseaux sont principalement construits en deux parties, une première partie composée de couches de convolution et de couches de sous-échantillonnage, puis une deuxième partie composée de couches denses.

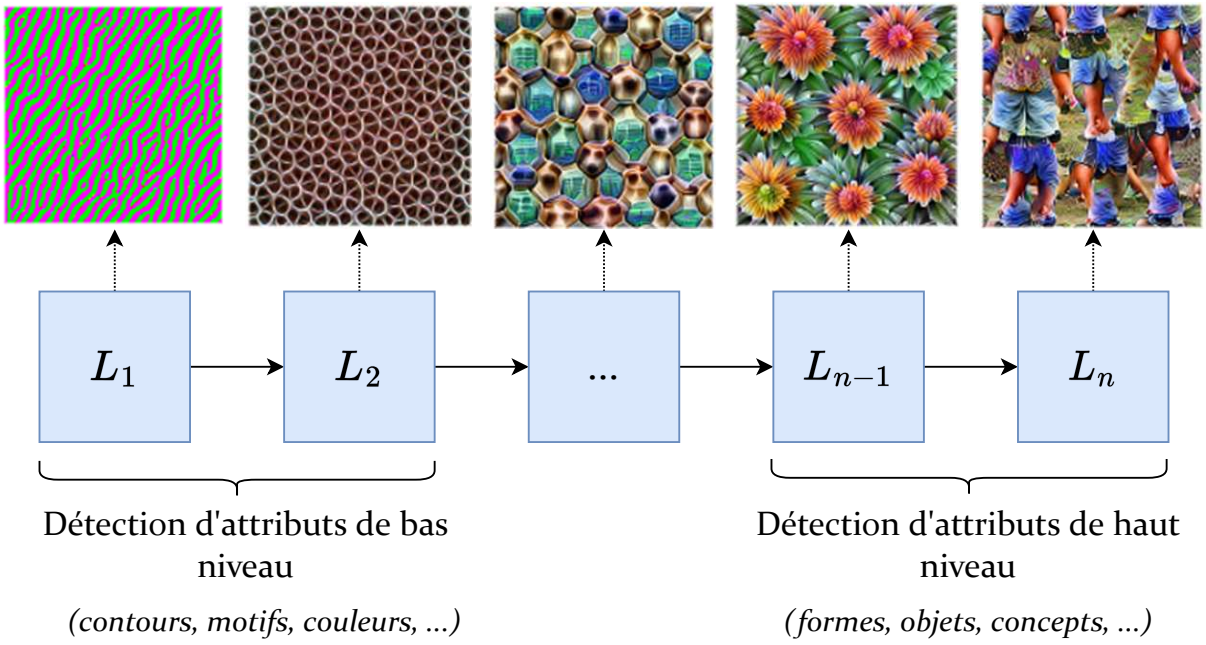


FIGURE 1.6 – Exemple, dont les images sont tirées de l'article Feature Visualization [97], d'attributs bas niveau détectés dans les premières couche (L_1, L_2) d'un réseau de classification, qui va progressivement, au fil des couches du réseau, pouvoir reconnaître des motifs et objets sur une image.

Ces deux parties, visibles sur le réseau LeNet5 [72] montré en Figure 1.2, ont chacune leur utilité propre ; dans la classification d'images, les couches de convolution servent principalement à détecter des formes et des couleurs, tandis que les couches denses servent à faire le lien entre des ensembles d'attributs «physiques» observés avec des concepts propres à la nature des données classifiées, avant de faire le lien avec l'une des étiquettes apprises. Un exemple de cette reconnaissance est montré en Figure 1.6, où les couches détectent d'abord des formes et motifs, avant de reconnaître des objets.

Afin d'évaluer la capacité de reconnaissance du réseau, il est fréquent de procéder à une phase d'évaluation après celle d'apprentissage. Cette phase d'évaluation consiste à faire classer au réseau un ensemble de données étiquetées absentes du jeu de données d'entraînement et de comparer les prédictions du réseau avec les étiquettes des dites données. En ayant un jeu de données de taille conséquente, équilibré en concepts à classer et varié en nature des données, le réseau devient capable de s'adapter à des données jamais observées durant son apprentissage. Après plusieurs sessions d'apprentissage (dites époques), si l'architecture du réseau est adaptée à la tâche et le jeu de données d'apprentissage est bien constitué, le réseau peut arriver à classer des données dans des concepts déjà appris avec une précision souvent aussi bonne voire supérieure à celle de l'humain.

La composition de jeux de données très variés comme ImageNet [24] ont permis l'élaboration de modèles aux architectures complexes, tels que VGG16 [124], dont les résultats de leur apprentissage peuvent être réutilisés dans d'autres scénarios de classification. Leur capacité à classifier des jeux de données tels qu'ImageNet leur ont permis de développer une excellente capacité de classification généralisée. Ces mêmes modèles peuvent alors être spécialisés dans la résolution de tâches plus précises en suivant une nouvelle phase d'apprentissage sur un jeu de données différent. Cependant, les poids calculés sur l'apprentissage précédent (par exemple sur ImageNet) sont conservés et permettent d'assimiler de nouveaux concepts nécessaires à la classification du nouveau jeu de donnée plus efficacement. L'emploi d'un nouvel apprentissage est appelé «apprentissage par transfert» (ou *transfert learning*). Ainsi, de nombreux utilisateurs de ces réseaux font

réapprendre une partie du réseau, notamment la partie dense en fin de réseau, sur un jeu de données d'apprentissage plus petit et spécialisé. L'apprentissage par transfert permet alors d'accélérer l'élaboration de réseaux efficaces et de faciliter leur adaptation à des scénarios plus spécifiques.

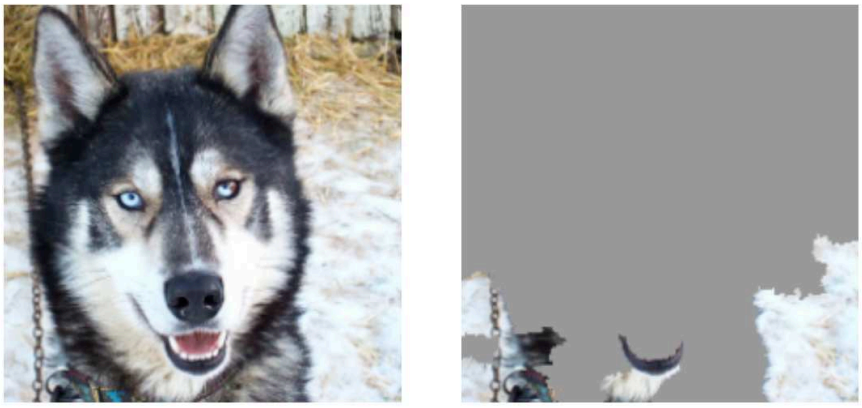
1.3 Interprétation et explicabilité des réseaux de neurones profonds

L'utilisation de plus en plus fréquente des réseaux de neurones a contribué au développement d'architectures gagnant en complexité. Des réseaux de classification comme ResNet [48] utilisent plus d'une centaine de couches de neurones, totalisant plusieurs millions de paramètres s'accordant automatiquement lors des phases d'apprentissage. Cette complexité évolue de manière empirique avec les évolutions technologiques, et a atteint aujourd'hui un niveau problématique dans l'explication des décisions prises par ces réseaux. Aujourd'hui, les décisions des réseaux de neurones modernes sont difficilement explicables que ce soit pour les experts du domaine d'application ou par les concepteurs du réseau. Ces réseaux entrainés sont alors comparés à des «boîtes noires» (*black-box*). Cette appellation indique qu'un système (ici le modèle de réseau de neurones entraîné) ne fournit aucune indication sur son fonctionnement interne pouvant justifier le résultat (ici la prédiction du modèle) de la transformation effectuée sur la donnée d'entrée. Cet aspect de l'intelligence artificielle empêche notamment d'avoir une confiance suffisante dans l'utilisation de ces techniques malgré leur efficacité. De plus, des régulations gouvernementales en Europe [1] et aux Etats-Unis [2] imposent aux exploitants d'informations d'expliquer aux individus la façon dont leurs données sont traitées. En France [3], dans le domaine médical et lorsqu'un réseau de neurones artificiels est utilisé pour apporter une aide à une décision particulière, les patients doivent être informés par des professionnels de la santé de la façon dont le modèle a traité les informations et ce qui a fait aboutir à cette conclusion. Cette loi est particulièrement importante puisqu'elle ne s'adresse pas exclusivement aux experts de l'intelligence artificielle mais aussi à ses utilisateurs qui n'ont pas forcément connaissance du fonctionnement interne du réseau.

Ces dernière années, le terme d'Intelligence Artificielle Explicable (*eXplainable Artificial Intelligence* ou *XAI*) est apparu au sein de la communauté scientifique. Cette thématique grandissante en terme de popularité [6] est principalement focalisée sur l'apprentissage machine, dont font partie entre autres les réseaux de neurones profonds, et regroupe les travaux cherchant à expliquer ou à faciliter l'interprétation des informations résultant des réseaux, pour ses concepteurs comme pour ses utilisateurs. Les contributions dans ce domaine visent différents objectifs qui peuvent être liés à l'explication d'une décision du réseau, l'interprétabilité des paramètres des couches du réseau ou la découverte de biais dans les prédictions. Les méthodes utilisées pour atteindre ces objectifs varient grandement sur leur forme et leur domaine d'application. Elles peuvent également faire intervenir d'autres thèmes de recherche comme la visualisation d'information et le traitement de données de masse, qui sont les thèmes abordés dans cette thèse.

Un exemple de méthode d'interprétabilité, LIME [114], est montré en Figure 1.7. Cette méthode est appliquée à un réseau de classification d'images qui classifie l'image d'un «Husky» en tant que «Loup». LIME permet de trouver les parties de l'image d'entrée ayant grandement contribué à cette mauvaise décision. Sur ces résultats, nous observons que c'est la présence de neige en arrière-plan de l'image qui a été principalement exploitée par le réseau et qui a contribué à obtenir cette décision, alors que l'on s'attend plutôt à ce que le réseau s'appuie sur les attributs de l'animal au premier plan pour former sa décision. Cette «explication» de la décision du réseau permet à l'utilisateur de détecter notamment un biais dans le jeu de données d'apprentissage où les images de loups auraient été toutes prises dans un environnement clair ou enneigé. À cause de cela, le réseau s'est appuyé sur cette particularité pour faire sa distinction. Cette méthode s'appuie sur une donnée d'entrée et la construction d'un masque pour expliquer la décision d'un réseau.

1. ec.europa.eu/commission/presscorner/detail/en/ip_21_1682
2. www.ai.gov
3. www.cnil.fr/fr/intelligence-artificielle/ia-comment-etre-en-conformite-avec-le-rgpd



(a) Husky classified as wolf (b) Explanation

Raw data and explanation of a bad model's prediction in the "Husky vs Wolf" task.

	Before	After
Trusted the bad model	10 out of 27	3 out of 27
Snow as a potential feature	12 out of 27	25 out of 27

"Husky vs Wolf" experiment results.

FIGURE 1.7 – Exemple de classification incorrecte expliquée par la méthode LIME [114]. Celle-ci révèle la présence d'un biais dans le jeu de données d'apprentissage, où une majorité d'images de «Loup» ont un arrière-plan ayant de la neige. Le réseau de neurones s'est servit de cette présence de neige pour distinguer les images de «Loup» d'autres animaux, notamment ici du «Husky».

D'autres techniques peuvent mettre en valeur les activations des neurones individuellement lors de la classification d'un exemple de donnée d'entrée [88]. Il est aussi possible de produire un prototype de donnée d'entrée (un exemple minimal) qui se fait classer correctement par le réseau étudié [95] permettant de mettre en évidence les attributs compris par ce réseau. Au contraire, il est aussi possible de construire un contre-exemple [76] permettant cette fois-ci de mettre en valeur les différences détectées entre deux classes discriminées par le réseau. L'explication des réseaux reste cependant très difficile puisque ces techniques ne sont pas infaillibles. Par exemple, des travaux mettent en défaut ces techniques d'explicabilité par le biais de contre-exemples [26].

Malgré les efforts des communautés travaillant sur l'explication de réseaux de neurones profonds, la tâche reste relativement complexe, et les progrès technologiques permettant de concevoir des réseaux encore plus complexes et polyvalents ne font qu'augmenter le besoin de techniques d'explicabilités. Dans cette thèse, nous proposons d'aborder la problématique par le prisme de la visualisation d'information, et plus précisément la visualisation de données en hautes dimensions. En considérant les données recueillies d'un réseau comme des données en hautes dimensions, nous nous affranchissons des limites rencontrées par les méthodes d'explications spécialisées sur des tâches et/ou des données en particulier, comme des images. Les problèmes à résoudre dans la conceptions de nos outils de visualisation sont la détection d'informations utiles à l'explicabilité des décisions des réseaux, et de les rendre compréhensibles pour les utilisateurs de nos outils.

1.4 Organisation

Dans cette thèse, nous étudions les techniques de visualisations pouvant aider à l'interprétation des décisions de réseaux de neurones profonds.

Le Chapitre 2 de ce manuscrit fait le point sur les méthodes d'explicabilité et d'interprétabilité existantes qui ont émergées des communautés d'apprentissage machine et de visualisation d'information.

Dans le Chapitre 3, nous présentons un premier outil de visualisation permettant de représenter la classification d'un réseau de neurones sous la formes de flux de données reflétant les traitements préliminaires effectués par le modèle et comprendre la contribution de chaque couche de celui-ci. Ce travail a fait l'objet d'une publication en conférence :

— **Halnaut, A.**, Giot, R., Bourqui, R., et Auber, D. (2020, Février). «Deep dive into deep neural networks with flows». Dans *Proceedings of the 15th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP 2020) : IVAPP (Vol. 3, pp. 231-239)*.

Dans le Chapitre 4, nous nous basons sur les observations faites sur ce premier outil pour nous adapter au traitement de scénarios plus complexes, variant en quantité de données classées par le réseau ainsi que son architecture. Cette nouvelle méthode a été publiée en conférence puis en version étendue :

— **Halnaut, A.**, Giot, R., Bourqui, R., & Auber, D. (2021, Janvier). «Samples Classification Analysis Across DNN Layers with Fractal Curves». Dans *ICPR 2020's Workshop Explainable Deep Learning for AI*.

— **Halnaut, A.**, Giot, R., Bourqui, R., & Auber, D. (2023, Février). «Compact Visualization of DNN Classification Performances for Interpretation and Improvement». Dans *Explainable Deep Learning AI, Methods and Challenges*.

Ces travaux nous dirigent vers la visualisation d'information compactes, et nous poursuivons cette approche de manière plus générale dans le Chapitre 5, où nous présentons un outil permettant de générer un arrangement sur une grille de pixels de n'importe quel jeu de données dans $\mathbb{R}^2$, et donc par extension de n'importe quel jeu de données dans $\mathbb{R}^N$ projeté vers $\mathbb{R}^2$ par des méthodes de réduction de dimensions (telles que PCA [105] ou t-SNE [136]). La présentation de cette méthode est accompagnée d'une évaluation quantitative contre plusieurs méthodes de l'état de l'art. Ces travaux ont fait l'objet d'une publication en conférence et d'une version étendue en cours de publication :

— **Halnaut, A.**, Giot, R., Bourqui, R., & Auber, D. (2022, Juillet). «VRGrid : Efficient Transformation of 2D Data into Pixel Grid Layout». Dans *26th International Conference Information Visualisation*.

— **Halnaut, A.**, Giot, R., Bourqui, R., & Auber, D. (2024, Avril). «Computation of Pixel-oriented Grid Layout for 2D Datasets using VRGrid». Dans *Artificial Intelligence and Visualization : Advancing Visual Knowledge Discovery*.

Enfin, dans le Chapitre 6, nous proposons de mesurer l'efficacité des méthodes de visualisations compactes par rapport aux méthodes existantes par le biais d'évaluations d'utilisateurs. Pour se faire, nous mettons en place un protocole d'évaluation et nous l'appliquons sur une évaluation entre deux méthodes de visualisation, l'une compacte et l'autre par nuage de points. Cette évaluation nous permet d'étudier également la pertinence de règles de colorations alternatives (qui sont souvent uniquement liées à la classe des éléments affichés) pour étudier les données dans leur espace d'origine $\mathbb{R}^N$. Nous discutons ensuite des perspectives des travaux présentés dans ce manuscrit avant de le conclure.

Chapitre 2

L'explicabilité des réseaux de neurones profonds

Dans ce chapitre, nous nous intéressons aux travaux existants dans le domaine de l'explicabilité des décisions et du fonctionnement des réseaux de neurones. Dans un premier temps, nous expliquons les objectifs des méthodes d'explicabilité et les deux approches principales locales et globales pour étudier le fonctionnement des réseaux. Ensuite, nous nous intéressons aux contributions apportées par la visualisation d'information dans ce domaine de recherche, et nous terminons par préciser le positionnement des travaux présentés dans cette thèse parmi les méthodes existantes.

2.1 Interprétation et Explicabilité

Les termes d'interprétabilité (*Interpretability*) et d'explicabilité (*Explainability*) sont couramment employés dans le domaine de l'Intelligence Artificielle Explicable (*eXplainable Artificial Intelligence* ou *XAI*). Ces termes sont souvent utilisés de manière interchangeable sans pour autant qu'il existe un consensus au sein de la communauté scientifique sur une définition permettant explicitement de les différencier [16]. D'après Bibal et Frénay [11], l'interprétabilité concerne les modèles dont la conception est basée sur la possibilité de comprendre son fonctionnement une fois entraîné. Le terme «transparent» est souvent employé [2], en opposition au terme «boîte noire». En ce qui concerne l'explicabilité, le terme regroupe les méthodes permettant d'indiquer comment le réseau a obtenu son résultat. Le plus souvent, celles-ci font usage de méthodes annexes au modèle, possiblement développées «pour» ce cas d'usage (on parle alors de méthodes *post-hoc*).

La différence entre les deux termes se situe donc sur la façon dont on aborde le problème de «boîte noire». Les méthodes d'interprétabilité cherchent à éviter l'aspect opaque du modèle, dès sa conception, en construisant des modèles moins complexes et interprétables [118]. Les méthodes d'explicabilité quant à elles se contentent de l'aspect opaque du modèle en proposant des outils pour déchiffrer son comportement. Ce sont ces deux définitions que nous adoptons dans cette thèse.

2.2 Méthodes locales et globales d'attribution de caractéristiques

Parmi les taxonomies accessibles dans la littérature [2], les méthodes d'explicabilité *post-hoc* se distinguent notamment en deux catégories : les méthodes locales et les méthodes globales. Les méthodes locales cherchent à expliquer le comportement du réseau pour une instance spécifique de prédiction : par exemple la classification d'une seule image parmi un jeu de données. Elles sont particulièrement efficaces pour expliquer une mauvaise classification en mettant en évidence les parties de la donnée d'entrée ayant mené à l'erreur du réseau. Les méthodes globales, quant à elles,

cherchent à expliquer le comportement du réseau dans son ensemble [119] : l'étude est portée sur un ensemble d'instances de prédiction, pouvant aller jusqu'à l'évaluation complète d'un jeu de données.

Parmi les méthodes locales souvent utilisées dans la littérature, un ensemble de méthodes s'appuie sur la construction d'un modèle simplifié de substitution à celui étudié (il est alors question de *surrogate model*). LIME [114] (*Local Interpretable Model-Agnostic Explanations*) et SHAP [79] (*SHapley Additive exPlanations*) sont deux méthodes qui servent à calculer et à mettre en évidence les parties importantes de la donnée d'entrée qui ont permis au modèle de former sa prédiction.

Dans le cas de LIME, le modèle de substitution s'appuie sur les prédictions du modèle de base sur une instance de classification. Ces prédictions sont calculées sur des données générées, qui sont des variations de la donnée d'entrées où des parties de celle-ci sont aléatoirement perturbées (par exemple, des images auront des zones de pixels grisées). Ensuite, ce modèle de substitution est capable de quantifier l'importance des diverses caractéristiques qui composent la donnée d'entrée (les données non perturbées) sur la prédiction du modèle étudié, et donc d'expliquer ses décisions sur cette instance de classification.

Au lieu de s'appuyer sur des données générées, SHAP repose sur le calcul des valeurs de Shapley. Ces valeurs permettent de quantifier la contribution de chaque caractéristique présente dans la donnée d'entrée de l'instance de classification étudiée sur la prédiction du modèle. Pour les réseaux plus complexe, la méthode repose sur l'utilisation d'un modèle de substitution pour calculer une approximation de ces valeurs.

Dans les scénarios de classification d'images, RISE [107] utilise une approche similaire à LIME en générant des données de classifications supplémentaire en créant aléatoirement des masques de pixels sur l'image d'entrée étudiée. Au lieu de reposer sur l'utilisation d'un modèle de substitution, le calcul de la carte de saillance s'effectue par une combinaison linéaire des pixels non-modifiés par le vecteur de prédiction du modèle. Avec un nombre suffisant de masques calculés, RISE obtient de meilleures performances sur ses explications que LIME. Elle permet donc de mettre en évidence les attributs bas niveaux de l'image qui affectent les décisions du réseau étudié de manière plus précise. Cette technique ne peut cependant que s'appliquer à la classification d'images, tandis que LIME n'a pas cette limitation.

Bien qu'intuitives sur leurs explications, ces méthodes reposent sur l'utilisation d'un autre modèle d'apprentissage. Suivant le degré d'explicabilité recherché, cette approche peut être problématique puisqu'elle déplace le problème d'explicabilité du modèle étudié à un autre modèle.

Grad-CAM [122] (*Gradient-weighted Class Activation Mapping*) est une autre méthode d'explicabilité pour les réseaux convolutifs. En partant de la couche de prédiction, Grad-CAM multiplie progressivement les cartes d'activations calculées par les gradients des couches précédentes jusqu'à remonter à la dernière couche convolutive du modèle. Cette technique de «développement» de la prédiction s'appelle propagation inverse (*backpropagation*) et se retrouve dans plusieurs techniques *post-hoc*. Dans le cas de Grad-CAM, cette propagation inverse se calcule jusqu'à atteindre la dernière couche de convolution du réseau, où les explications calculées sont représentées sur la donnée d'entrée sous la forme d'une carte de chaleur (*heatmap*). Cette technique est communément utilisée dans le cadre de classification d'images dont un exemple est présenté dans la Figure 2.1. Grad-CAM peut être appliquée sur chaque composante de la prédiction (donc chaque classe d'équivalence) du réseau étudié, et permet donc de mettre en évidence plusieurs ensembles de caractéristiques détectés par le réseau. La popularité de cette méthode est notamment due à l'intuition immédiate fournie à l'utilisateur sur le fonctionnement des couches convolutives, et le comparer à sa propre analyse visuelle de la donnée d'entrée. D'autres méthodes ont une approche similaire, comme Layer-wise Relevance Propagation [88] (LRP, montré en Figure 2.2), DeConvNet [154], ou encore Integrated Gradients [130] en adoptant des règles de propagation inverse différentes suivant les types de couches de neurones rencontrés. Ces méthodes limitent leur cas d'application à des réseaux possédant les couches les plus communes (les couches denses, convolutives, de sous-échantillonnage) et aux architectures linéaires de réseaux.

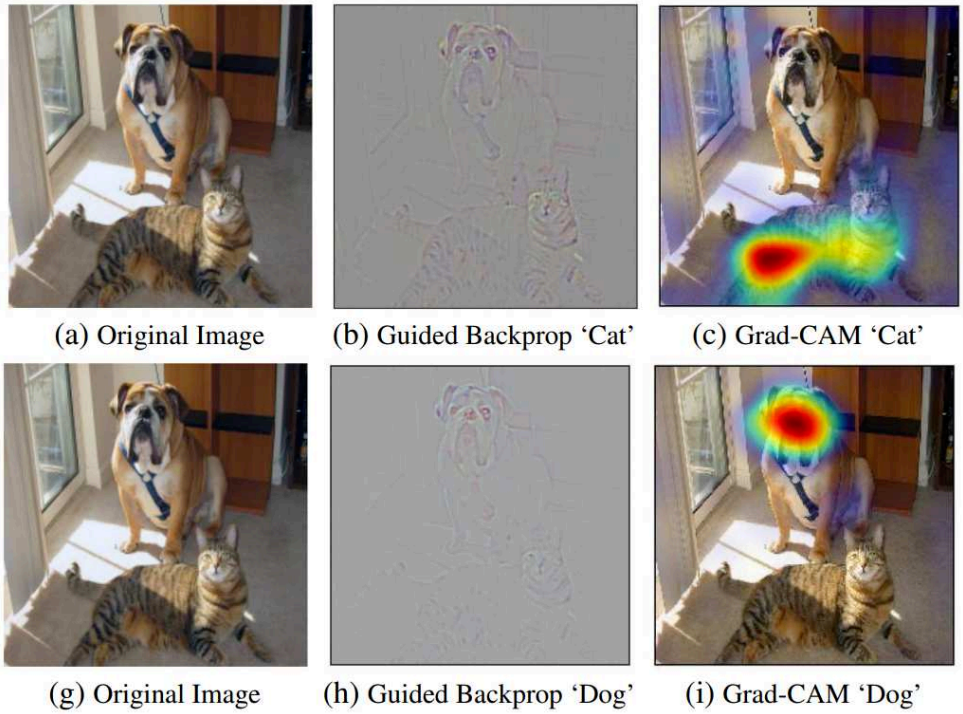
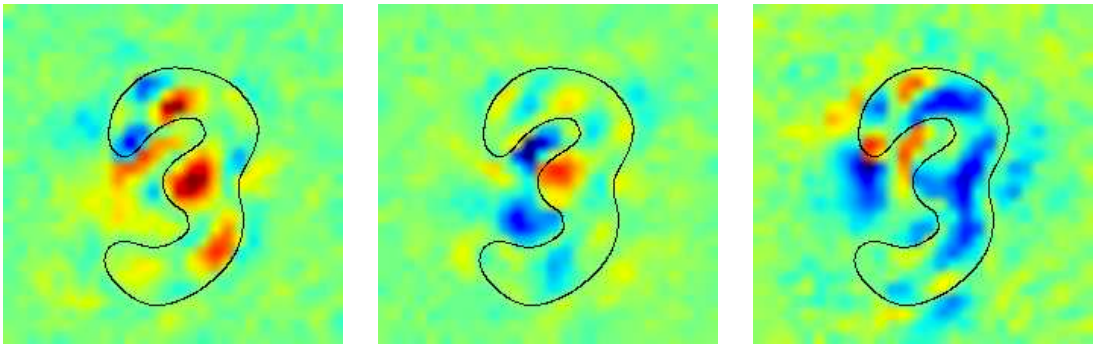


FIGURE 2.1 – Exemple d'utilisation de Grad-CAM, tiré de l'article des auteurs de l'outil [122], mettant en évidence les régions de l'image ayant contribué à la classification de «Chien» et «Chat».



(a) Carte de chaleur sur la prédiction 3

(b) Carte de chaleur sur la prédiction 8

(c) Carte de chaleur sur la prédiction 6

FIGURE 2.2 – Exemple d'application de LRP sur une tâche de classification d'image, présenté sur le site de démonstration de l'outil [2]. Les cartes de chaleur indiquent les parties de l'image ayant contribué en faveur (rouge) ou en défaveur (bleu) de la classification indiquée en légende.

Les méthodes d'explication globales ont des approches différentes pour expliquer le fonctionnement des réseaux sans s'appuyer sur une instance de prédiction en particulier. Ce type d'approche a été moins étudié par la communauté d'apprentissage profond. SAGE [20] (*Shapley Additive Global importancE*) est une méthode d'explicabilité qui fait usage des valeurs de Shapley comme SHAP, mais cette fois-ci sur un ensemble de données d'entrées pour un scénario impliquant un modèle de régression (qui n'a qu'une valeur de sortie). Elle permet ensuite de quantifier la contribution des caractéristiques détectées par le réseau étudié sur sa prédiction.

L'explication d'un réseau entraîné peut aussi se faire par la construction d'un arbre de décision servant à expliquer les décisions d'un réseau à l'aide d'un ensemble de règles conditionnelles [73]. Cette stratégie a vu le jour avant l'avènement des modèles convolutifs, et de nombreuses méthodes ont été proposées et synthétisées par Andrew et al. [4]. Ces méthodes peuvent être appliquées sur

les réseaux prenant en entrée des vecteurs à quelques composantes, mais quand la quantité d'information en entrée devient importante, la compréhension de ces règles telles quelles devient plus difficile. Dans RuleMatrix [85], un modèle de substitution est utilisé afin de générer un ensemble de règles reflétant avec une certaine fidélité le comportement du modèle étudié. Pour s'adapter aux réseaux plus complexes et donc aux règles plus nombreuses, RuleMatrix utilise une interface à vues interactives multiples pour synthétiser l'ensemble des informations à l'écran. Cependant, cette méthode se limite à l'étude des réseaux composés de couches profondes uniquement. L'idée est étendue dans CNN2DT [58] pour supporter les réseaux convolutifs, et l'ensemble de règles y est alors représenté sous la forme d'un diagramme de flux.

2.3 Approche par visualisation d'information interactive

Certaines méthodes d'explication utilisent des techniques de visualisation pour informer l'utilisateur sur le fonctionnement du réseaux. Les travaux de Olah et al. [97, 98] présentent un ensemble d'interfaces interactives en relation avec l'explicabilité des réseaux de neurones. Ces travaux montrent le potentiel de la visualisation d'information pour étudier des aspects des réseaux de neurones alors moins explorés par les méthodes présentées précédemment. Dans Feature Visualization [97], l'objectif est de maximiser l'activation de certains neurones du réseau, quelque soient leur position ou objectif. Les neurones étudiés peuvent aller d'un neurone seul dans une couche convolutive à une couche de convolution entière : il s'agit donc d'une méthode d'explicabilité globale. Il peut aussi s'agir des neurones des couches denses situées proches de la couche de prédiction. Pour obtenir l'information perçue par ces parties du réseau, des images sont générées afin de maximiser les activations des neurones étudiés. Suivant l'endroit du réseau qui est étudié, les motifs obtenus sur les images calculées varient en complexité. Des exemples de ces images calculées sont montrés dans la Figure 2.3 : Les premières couches du réseau produisent des motifs relativement simples tandis que les neurones des dernières couches du réseau produisent des images regroupant plusieurs caractéristiques que l'on peut retrouver dans une classe de données particulière. Dans les *Building Blocks of Interpretability* [98], l'approche de la méthode est similaire mais cette fois-ci locale. La réaction du réseau lors du traitement d'un exemple de classification est présentée suivant différentes techniques de visualisations. Les techniques font usage d'interactions utilisateurs pour que celui-ci puisse étudier le comportement du réseau à la fois sur un endroit précis de la donnée d'entrée, mais aussi décider de l'endroit du réseau (ie. la couche et le filtre) à explorer. En combinant ses observations sur ces aspects du réseau, l'utilisateur peut ainsi comprendre ce que le réseau a appris à reconnaître lors de la phase d'apprentissage, et comment cette connaissance a aidé le modèle à la classification d'un exemple en particulier.

Dans le récent état de l'art de La Rosa et al. [69], plus d'une soixantaine d'outils de visualisations spécialisés dans l'explication des décisions de réseaux de neurones sont recensés et catégorisés suivant : (1) leur fonctionnement, (2) le type d'architecture du réseau étudié, (3) le profil d'utilisateur ciblé, (4) la présence d'interactions et (5) leur validation par une évaluation utilisateurs. Très souvent, ces outils sont composés de plusieurs vues pouvant utiliser différentes représentations de données. Ces vues sont accompagnées d'interactions permettant aux utilisateurs d'étudier les réseaux et leurs résultats sous différentes formes et perspectives. Des outils peuvent aussi être destinés à l'explication de réseaux pour les néophytes ou bien pour les experts, comme montré par exemple en Figure 2.4 : Tensorflow Playground [125] est conçu pour expliquer le fonctionnement général des réseaux de neurones au «grand public», tandis que NNVA [47] apporte une aide à des experts d'un domaine métier spécifique de bio-informatique. Dans Tensorflow Playground, l'utilisateur peut expérimenter l'utilisation des réseaux de neurones pour une tâche de classification très simple qui consiste à classifier un ensemble de points dans $\mathbb{R}^2$. Pour cela, l'utilisateur peut modifier l'architecture du réseau à la volée, en rajoutant des couches denses et en modifiant le nombre de neurones alloués à celles-ci. Les propriétés apprises par les neurones sont affichées sous la forme de cartes de chaleur miniatures, et sont reliées aux autres en fonc-

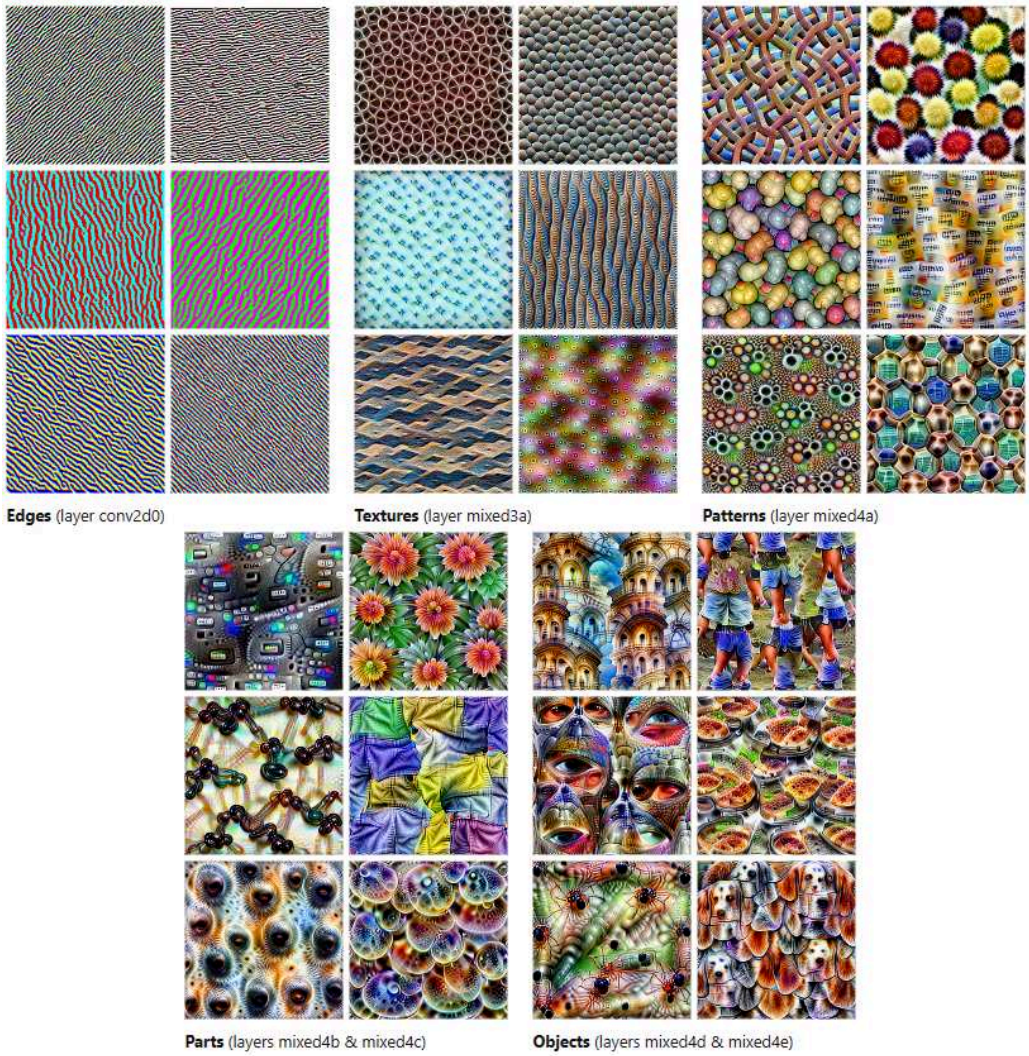
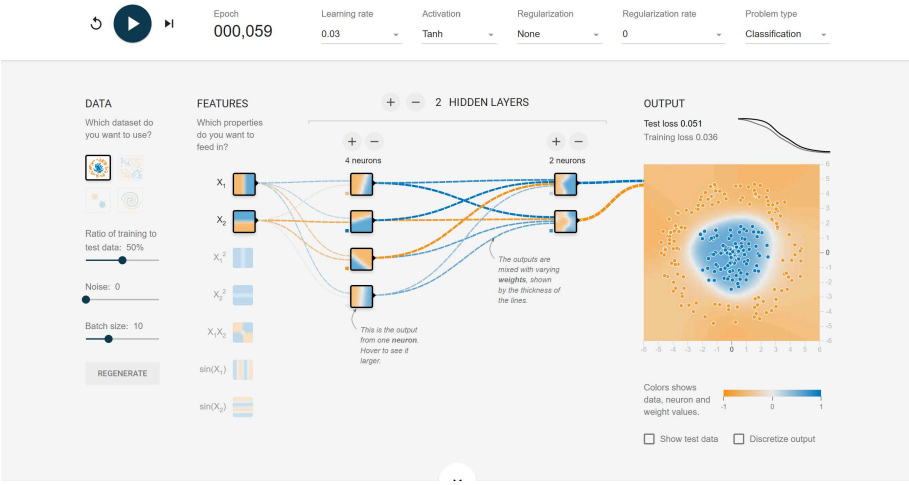


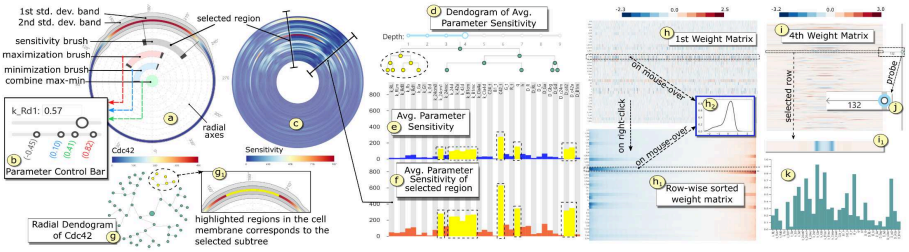
FIGURE 2.3 – Exemple d'images calculées dans Feature Visualisation de Olah et al. [97]. Cette capture d'écran est tirée de l'article publié sur *Distill*[4]. Ces images sont calculées au niveau de plusieurs couches du réseau GoogleLeNet [132] entrainé sur ImageNet, allant des premières couches convolutives en haut vers les couches plus tardives du réseau en bas. Nous pouvons observer que les motifs deviennent de plus en plus complexe au fil des couches, allant de la reconnaissance de lignes sur l'image de gauche à la reconnaissance de concepts (monument, animal, parties faciales et corporelles) plus tard.

tion de leurs poids. Pendant la phase d'apprentissage du réseau, l'utilisateur peut visualiser en temps réel l'évolution des poids de chaque neurone ainsi que les valeurs retournées par le réseau pour chaque donnée d'entrée dans $\mathbb{R}^2$. Cet outil de visualisation n'est cependant pas conçu pour visualiser le fonctionnement des réseaux plus complexes tels que LeNet. NNVA quant à lui fait usage d'une combinaison de méthodes de visualisations, telles que des cartes de chaleur circulaires, des histogrammes, des vues matricielles et de la visualisation de courbes pour représenter le comportement du réseau étudié.

Certains outils proposent des méthodes de visualisations spécialisées comme AEVis [15], VisQA [57] ou MultiRNNExplorer [123] qui utilisent des glyphes pour représenter les caractéristiques propres aux réseaux et données étudiées. Ici, nous nous intéressons à l'utilisation de méthodes de visualisations plus communes, comme l'utilisation de nuages de points, de diagrammes de flux, de matrices, et de cartes de chaleur.



(a) Tensorflow Playground [125]



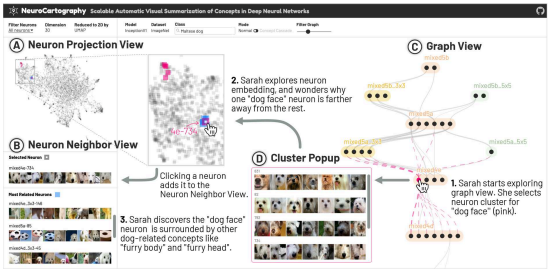
(b) NNVA [47]

FIGURE 2.4 – Les interface des outils Tensorflow Playground (capture d'écran du site internet playground.tensorflow.org), destiné aux néophytes, et de NNVA (capture de l'article des auteurs), destiné à des experts métier.

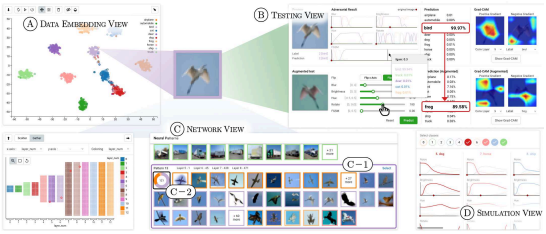
2.3.1 Représentation par nuage de points

La représentation de nuage de points est l'une des représentations que l'on retrouve souvent lorsque les réseaux sont sollicités pour des tâches de classification. Des exemples d'utilisation de nuages de points sont montrés en Figure 2.5. Cette méthode de visualisation est utilisée principalement pour représenter le résultat d'une projection de données dans $\mathbb{R}^N \rightarrow \mathbb{R}^2$ pour étudier leur topologie. Les points affichés sont très souvent étiquetés à une classe du jeu de donnée, et sont colorés de manière à les différencier d'une classe à une autre. Ces données hautement dimensionnelles peuvent être des données d'entrées encodées en des vecteurs dans $\mathbb{R}^N$, ou les cartes d'activations d'un réseau de neurones. Dans Neurocartography [103], VATUN [102], les travaux de Ma et al. [80], Summit [52] ou encore ActiVis [62], chaque point affiché représente une instance de classification d'un réseau. Dans les travaux de Rauber et al. [113] et présentés dans la Figure 2.6, les cartes d'activations au niveau de chaque couche du réseau sont projetées avec t-SNE. Les points résultants de la même donnée d'entrée sont reliés entre eux afin de représenter une forme de progression de la classification des données. La visualisation représentant les instances de classification par des points permet de détecter (1) des tendances au niveau des prédictions du réseau représentées sous la forme de groupement de points d'une ou plusieurs classes de données, et (2) les cas problématiques (*outliers*) représentés par des points d'une classe isolé dans un ensemble de points d'une autre classe de données.

Dans DeepEyes [109] et CNN2DT [58], les points représentent les neurones d'une couche du réseau plutôt qu'une instance de classification. Ces neurones sont projetés respectivement par HSNE [108] et t-SNE [136], des techniques de réduction de dimensions très utilisées dans l'étude de données hautement dimensionnelles. Sur cette projection, on peut observer une formation

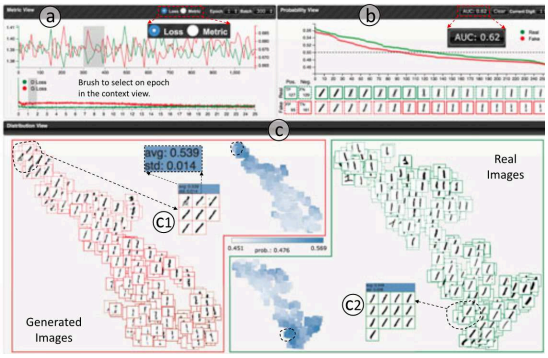


(a) Neurocartography [103]



(b) VATUN [102]

(c) DeepEyes [109]



(d) GANViz [141]

FIGURE 2.5 – Outils de visualisation utilisant la représentation par nuage de points. Ces captures sont issues des articles de leurs auteurs respectifs.

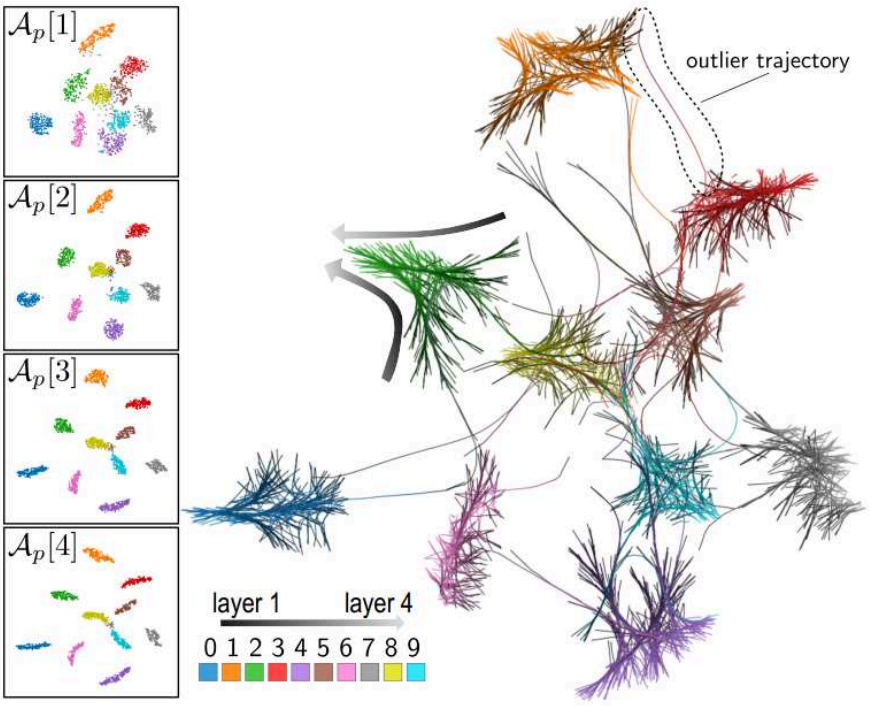
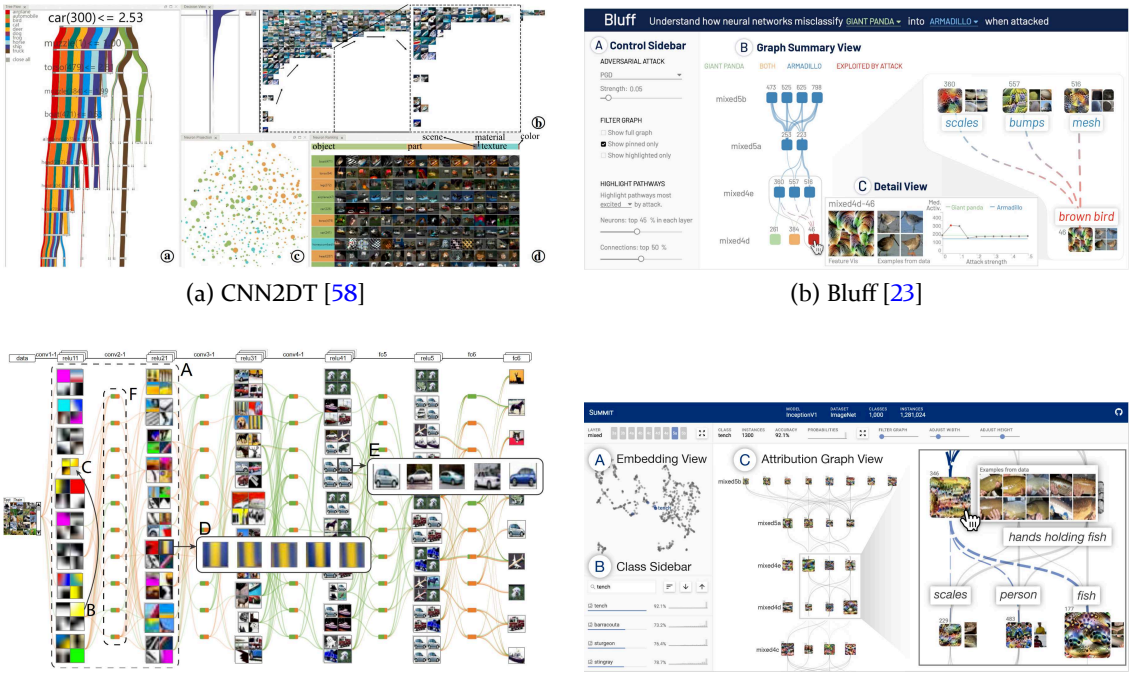


FIGURE 2.6 – Projection des cartes d'activations de quatre couches d'un réseau de neurones entraîné sur MNIST par Rauber et al. [113]. On peut y observer un regroupement des points d'une même classe et une séparation plus nette entre ces groupes à mesure que l'on progresse dans les couches du réseau.

progressive de groupes de points qui permet de distinguer les différentes classes discriminées par le réseau étudié.



(a) CNN2DT [58]

(b) Bluff [23]

(c) CNNVis [75]

(d) Summit [52]

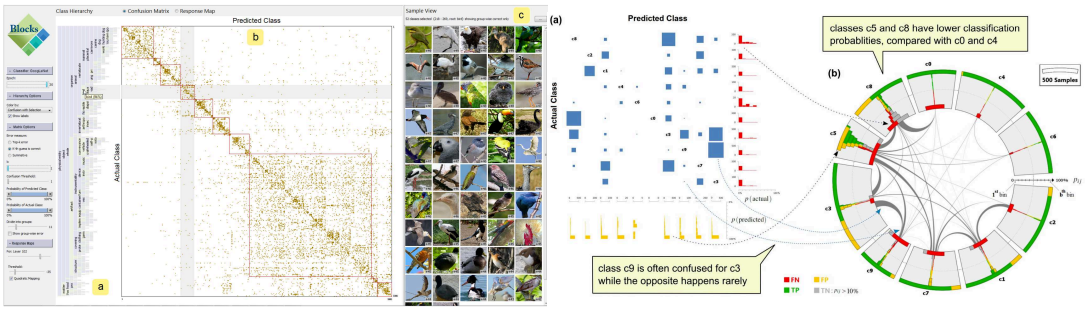
FIGURE 2.7 – Outils de visualisation utilisant des représentations par flux. Ces captures sont issues des articles de leurs auteurs respectifs.

Les positions calculées pour afficher des données sous la forme d'un nuage de points peuvent aussi être utilisées pour représenter des données encodées autrement que sous la forme d'un point. Dans GANViz [141], les images générées et les images réelles sont directement représentées à la place des points. Dans GNNLens [59], les valeurs d'analyse supplémentaires sous encodées sous la forme de glyphes positionnés à la place de chaque point du nuages.

2.3.2 Utilisation de diagrammes de flux

Contrairement à la visualisation par nuages de points, les diagrammes de flux représentent plutôt des ensembles d'éléments ayant un comportement commun plutôt que chaque élément individuellement. Dans le contexte d'explicabilité des réseaux de neurones, cette méthode de visualisation est plus souvent utilisée pour représenter des ensembles d'instance de classifications étant traitées similairement par le réseau (par exemple, les couches du réseau ont généré des cartes d'activations semblables). Des exemples d'application de cette méthode sont présentés en Figure 2.7. Dans CNN2DT [58], le diagramme de flux est employé pour représenter un arbre de décision du réseau de classification en se basant sur l'analyse des activations des neurones. Dans RNNVis [84], les flux colorés informent sur l'influence de la mémoire des réseaux récurrent sur leur prise de décision dans le domaine de compréhension de texte. Dans CNNVis [75] et Summit [52], des flux sont utilisés pour représenter une classification progressive des éléments du jeu de données au niveau de chaque couche du réseau. Bluff [23] emploie une représentation similaire mais pour étudier plus en détails les failles des réseaux de classification. InstanceFlow [111] utilise un diagramme de flux pour représenter l'évolution de la capacité du réseau à discriminer les différentes classes du jeu de données au fil des époques lors de l'apprentissage du réseau.

Les diagrammes de flux peuvent aussi être combinés avec d'autres méthodes de visualisation. Dans SANVis [101] et ProtoSteer [86], les diagrammes de flux sont utilisés conjointement avec des histogrammes et des vues matricielles afin de montrer l'attention portée par les éléments du réseau sur les données d'entrée.



(a) Blocks [12] (b) Roue de confusion [3]

FIGURE 2.8 – Exemple de matrices de confusion représentées par une vue matricielle (a) et par une roue de confusion (b). Ces captures sont issues des articles de leurs auteurs respectifs.

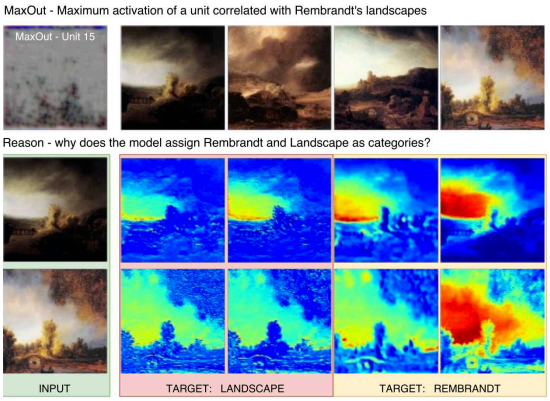
2.3.3 Matrices de confusion

Les matrices de confusion sont une représentation d'informations propres à l'étude de systèmes de classification. Elles sont souvent utilisées pour présenter les résultats de classification d'un réseau de neurones de manière synthétique, car elles permettent de révéler les performances du réseau, les classes de données les plus problématiques et les confusions les plus communes (vrais négatifs et faux positifs). Ces matrices sont arrangées de sorte que les lignes représentent les classes réelles du jeu de données testé et les colonnes les classes prédites par le réseau.

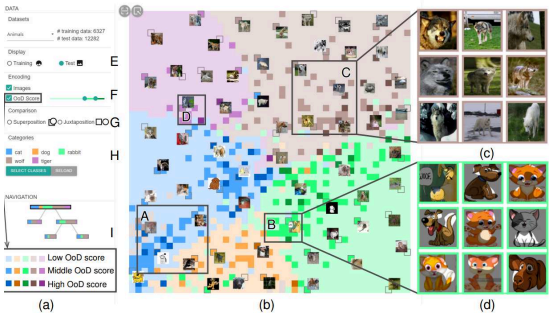
Chaque cellule (ligne=X, colonne=Y) de la matrice présente le nombre d'exemples du jeu de données de classe réelle X ayant été prédite comme étant de classe Y par le modèle. Ainsi, la diagonale de la matrice recense la totalité des scénarios où le réseau a donné une prédiction correcte, et ailleurs dans la matrice des prédictions incorrectes. Pour chacun de ces scénarios, nous pouvons connaître les classes impliquées et la quantité d'erreur similaires produites par le réseau. Il est fréquent de colorer les cellules de la matrice de confusion de manière à mettre en évidence leur fréquence dans l'ensemble des classifications testées. On peut voir l'utilisation de telles matrices dans V-Awake [36] par exemple, ou dans les scénarios impliquant un grand nombre de classes hiérarchisées dans Blocks [12]. Cette information peut aussi être visualisée sous la forme d'un cercle dont les bords sont connectés entre eux via des techniques de groupement d'arêtes [53] (*Edge Bundling*). Cette représentation est appelée «roue de confusion» (*confusion wheel*) dans la littérature [3] dont un exemple d'application est montré en Figure 2.8. La vue matricielle peut aussi être utilisée pour étudier de manière synthétique le comportement de chaque neurone d'une couche. Dans cette vue, les neurones de la couche étudiée sont listés sur les lignes de la matrice, et les scénarios sur les colonnes de la matrice. La valeur d'activation du neurone X dans un scénario Y est alors représentée en cellule (ligne=X, colonne=Y) par une valeur numérique, un symbole ou une couleur. Cette méthode appelée visualisation d'attention peut être observée dans ActiVis [62], DeepCompare [93] et VisQA [57]. Dans AttentionFlows [25] et Dodrio [144], ces informations sont représentées sous la forme d'un cercle, comme pour la roue de confusion.

2.3.4 Cartes de chaleur

Les cartes de chaleur sont principalement utilisées pour afficher les parties d'une donnée d'entrée, très souvent une image, qui ont contribué à l'activation d'un neurone ou d'une partie du réseau. Des exemples d'utilisation de cartes de chaleur sont montrés en Figure 2.9. On peut retrouver cette méthode de visualisation sur les outils implémentant les techniques citées en Section 2.2 comme VisLRPDesigner [54] qui exploite LRP ou les travaux de Strezoski et al. [127] qui exploitent Grad-CAM. Ces cartes de chaleur peuvent être présentées différemment, comme dans OoDAnalyzer [17] ou bien CNN2DT [58] où un masque de seuillage est appliqué sur les images



(a) MaxOut et Reason [127]



(b) OoDAnalyzer [17]

FIGURE 2.9 – Exemples d'utilisation des carte de chaleur. Ces captures sont issues des articles de leurs auteurs respectifs.

pour mettre en évidence les zones qui ont contribué à la décision du réseau, plutôt que le dégradé coloré présent dans les autres méthodes.

2.4 Positionnement des travaux de la thèse

Dans ce chapitre, nous avons montré différentes approches et techniques de visualisation pour aider à la compréhension des réseaux de neurones artificiels. Ceux-ci ne font que gagner en complexité et en performances, et si les méthodes proposant de construire des modèles interprétables s'avèrent être une solution [118], ils doivent encore surpasser les performances des modèles «boîtes noires» [6]. Pour expliquer le fonctionnement de ces «boîtes noires», nous avons vu qu'il existait deux approches, l'une locale qui cherche à expliquer le fonctionnement du réseau étudié sur une instance de classification particulière, et l'autre globale qui cherche à expliquer le fonctionnement du réseau à partir d'une généralisation d'instances de prédictions. Ces deux approches peuvent faire usage de méthodes de visualisation d'information pour donner une intuition supplémentaire à l'utilisateur sur le fonctionnement des réseaux de neurones. Cependant, une partie de ces travaux ne se concentre que sur des architectures de réseaux spécifiques ou des tâches particulières impliquant un type de données précis, comme des images par exemple. Ces limitations rendent l'application de ces méthodes dans d'autres scénarios difficiles voire impossibles.

Dans cette thèse, nous nous intéressons au développement de méthodes d'explicabilité *post-hoc* globale à appliquer sur des réseaux de classification déjà entraînés en utilisant un jeu de données d'évaluation. Pour cela, nous faisons usage de méthodes de visualisations d'informations pour pouvoir détecter et informer l'utilisateur des comportements de classification des réseaux étudiés. Afin de ne pas limiter les cas d'applications de nos méthodes, nous choisissons de ne pas spécialiser le traitement de l'information à un type de données en particulier. À la place, nous considérons les données d'entrées, les cartes d'activations et les prédictions du modèle comme des vecteurs dans des espaces hautement dimensionnels. En faisant usage d'un ensemble de prédictions du modèle pour générer ces vecteurs, nous cherchons une notion de proximité entre ceux-ci dans les espaces hautement dimensionnels pour détecter les comportements de classification au niveau des couches du modèle.

Chapitre 3

Utilisation de flux pour visualiser le processus de classification

Dans ce chapitre, nous présentons une première méthode de visualisation pour aider à comprendre la prise de décision des réseaux de neurones dans les tâches de classification. Nous travaillons avec des réseaux entraînés et des jeux de données d'évaluation pour discerner son comportement global. Contrairement à une grande partie des travaux existants, notre recherche n'est pas spécialisée pour un type de données d'entrée en particulier (comme des images, du texte ou de la vidéo) mais traite uniformément tout type de données. Cette approche nous permet d'étendre notre analyse jusqu'aux calculs intermédiaires réalisés par le réseau et de retrouver le rôle et la contribution de chaque couche individuellement. Ce chapitre est organisé en trois parties. Dans un premier temps, nous listons les objectifs de notre approche et les méthodes envisagées pour les remplir. Ensuite, nous présentons les techniques d'implémentation utilisées pour la réalisation de l'outil. Enfin, nous montrons l'application de l'outil sur 4 scénarios accompagnés d'une analyse des résultats obtenus.

Les contributions dans ce chapitre ont mené à :

- Une publication en conférence : **Halnaut, A.**, Giot, R., Bourqui, R., et Auber, D. (2020, Février). «Deep dive into deep neural networks with flows». Dans *Proceedings of the 15th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP 2020) : IVAPP (Vol. 3, pp. 231-239)*.
- Un prototype de la méthode en démonstration à l'adresse `https://pivert.labri.fr/flows/index.html` et dont le code est disponible à l'adresse `https://github.com/eikofee/dive-dnn-flows`.

3.1 Contexte

Nous nous intéressons à l'explicabilité des décisions de réseaux de neurones profonds pour résoudre des tâches de classification. Dans notre cas, les réseaux que nous étudions suivent une architecture à propagation avant (*feed-forward* en anglais), ce qui signifie que les données d'entrées sont transformées successivement à travers les couches du réseau. Ces couches sont composées d'un algorithme spécifique associé à des poids qui affectent les transformations qu'elles effectuent. Par exemple, une couche convolutive transforme différemment sa donnée d'entrée par rapport à une couche dense. Leur type de transformation est défini par le concepteur du réseau, mais les poids sont calculés durant la phase d'apprentissage. Cette phase d'apprentissage consiste à fournir au réseau des données d'entrée associées à des étiquettes (aussi appelée *ground truth*, voir Section 1.2). Ces étiquettes respectent la forme en sortie du réseau et représentent les prédictions attendues par le concepteur. La combinaison des données d'entrée et des étiquettes forment

le jeu de données d'entraînement. À l'aide d'un algorithme d'optimisation, ce jeu de données permet de calculer les poids des couches du réseau pour se conformer aux résultats attendus. Ainsi le fonctionnement du réseau dépend autant des données utilisées lors de l'apprentissage que des algorithmes qui le décrivent. En revanche, le concepteur n'a pas une connaissance exacte de ce que calcule le réseau.

Afin de jauger les performances du réseau après son apprentissage, le concepteur doit procéder à une phase d'évaluation. Cette phase nécessite de fournir un jeu de données d'évaluation au réseau, qui est composé de données d'entrée et des étiquettes de façon identique au jeu de données d'entraînement. Ce jeu de données doit être composé de données exclues du jeu de données d'entraînement afin d'évaluer la capacité du modèle à généraliser ce qu'il a apprit. L'évaluation consiste à calculer la précision (*accuracy* en anglais) du réseau définie par la formule suivante :

$$\text{précision} = \frac{\text{Nombre de classifications correctes}}{\text{Taille du jeu de données d'évaluation}} \tag{3.1}$$

Pendant cette phase, les paramètres des couches ne sont pas modifiés par le réseau. Plus le taux de précision est élevé, plus le réseau est performant sur la tâche définie par le concepteur.

En revanche, cette méthode d'évaluation jauge les performances du réseau au niveau global, sans fournir d'information sur la complexité de la tâche, le rapport entre le coût computationnel et les performances du modèle, ou encore la contribution de chaque couche dans la prédiction finale. Ce manque d'information empêche d'expliquer les décisions correcte et erronées du modèle.

Pour améliorer son réseau, le concepteur peut se poser plusieurs questions :

— **Q1 : Quelles sont les performances de mon réseau ?** Savoir si le réseau est performant est essentiel avant de décider si le réseau a besoin d'être amélioré ou non.

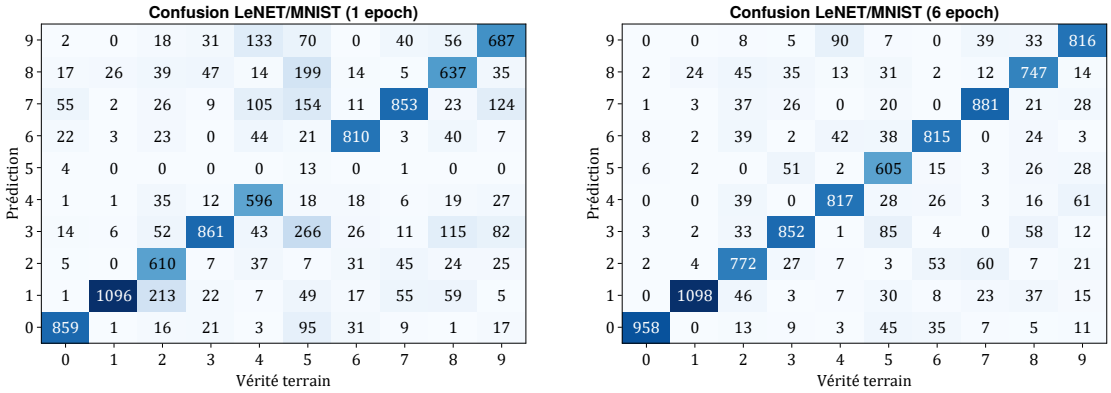
— **Q2 : Quelles ont été les données mal classées ?** En récupérant les données qui ont été mal classées par le réseau, le concepteur peut tenter de trouver une caractéristique commune qui a fait défaut lors de leur traitement.

— **Q3 : Qu'est-ce qui a induit le réseau en erreur ?** Les réseaux de classifications extraient d'abord des informations de bas niveau sur les données d'entrée puis trouvent les caractéristiques propres à la tâche de classification. Le concepteur peut être aiguillé sur la partie du réseau à améliorer s'il sait si l'erreur du réseau est due à une information de bas niveau ou de haut niveau.

— **Q4 : Quelle est la contribution de chaque couche du réseau sur la classification ?** Pour savoir quelles sont les parties du réseau à améliorer de manière générale, il faut d'abord trouver quelles sont les couches qui sont performantes et celles qui le sont moins. En mettant en évidence la capacité des couches à discriminer des sous-ensembles d'éléments du jeu de données, le concepteur peut évaluer la contribution de chaque couche sur la classification finale, et donc trouver quels sont les endroits du réseau à améliorer.

La mesure de la précision d'un réseau lors de sa phase d'évaluation permet de répondre à **Q1**, et peut être étendue pour répondre aussi à **Q2** en gardant une trace des classifications du réseau. Cette information est souvent représentée par le biais d'une matrice de confusion, dont un exemple est présenté en Figure 3.1. Celle-ci comptabilise les couples prédiction/vérités terrains émises par le réseau. Nous pouvons y observer ainsi la quantité de prédictions correctes par le réseau (prédiction = vérité terrain) mais aussi la répartition des erreurs de prédiction (prédiction ≠ vérité terrain). Parmi ces erreurs, nous pouvons aussi observer quelles classes d'éléments ont posé le plus de difficultés au réseau. Cependant, pour répondre à **Q3** et **Q4**, le réseau doit être étudié de manière plus approfondie plutôt que comme une fonction en un seul bloc, ce que la mesure de précision ne permet pas.

Les méthodes d'explication locales présentées dans le Chapitre 2 se concentrent sur le processus de prédiction du réseau sur une seule tâche de classification. Certaines se basent sur la propagation inverse des informations dans le réseau de neurones pour construire une donnée d'entrée artificielle à partir de la prédiction obtenue sur la tâche. Ces méthodes (comme Grad-CAM [122] par exemple) sont très efficaces pour répondre à **Q3**, mais ne permettent que d'étudier



(a) Confusion après 1 époque

(b) Confusion après 6 époques

FIGURE 3.1 – Exemples de matrice de confusion d'un réseau LeNet entraîné et évalué sur MNIST. Les résultats sont calculés après 1 époque (a) et 6 époques (b). Une cellule aux coordonnées (ligne = X, colonne = Y) indique le nombre de prédictions en X du modèle sur des éléments étiquetés comme étant de classe Y. Cette vue nous permet de mettre en évidence l'amélioration du réseau à mieux catégoriser les 5 après d'avantage de temps d'apprentissage, alors que les 1 on été catégorisés après seulement une seule passe d'apprentissage.

une seule classification (erronée ou non) à la fois. De plus, en reposant sur des cartes de saillance (*heatmap*), ces méthodes ne permettent que d'étudier les scénarios qui impliquent des données pouvant être visualisées par l'humain, telles que des images. Enfin, une autre limitation de ces méthodes est qu'elles limitent l'étude aux informations des premières couches du réseau uniquement, donc d'informations bas niveau (par exemple, les formes ou parties d'une image qui ont aidé à la classification) mais pas sur les concepts détectés dans les couches plus éloignées dans la classification.

D'autres méthodes d'explications plus génériques, comme *Layer-wise Releavance Propagation* [88] (LRP) permettent d'étudier les contributions individuelles des neurones et donc par extension des couches du réseau, et peuvent s'adapter à n'importe quelle nature de données classées. Elles peuvent donc aider à répondre aux questions **Q3** et **Q4** lorsqu'elles sont bien agrégées et visualisées, mais ont les mêmes limitations que les méthodes précédentes à savoir ne pouvoir traiter qu'une tâche de classification à la fois.

Dans le cadre d'évaluation des performances d'un réseau de neurones, nous pensons qu'il est plus souhaitable de pouvoir répondre à ces questions sur l'ensemble des classifications évaluées plutôt que sur des cas isolés.

Notre approche

Dans la suite de ce chapitre, nous présentons une nouvelle méthode de visualisation pour l'explicabilité des décisions de réseaux de classification. Nous voulons montrer à l'utilisateur une vue d'ensemble du réseau qui synthétise les informations produites par le réseau lors de son inférence sur un ensemble de données d'évaluations. Cette vue d'ensemble a pour objectif de combler les limitations des méthodes évoquées précédemment, notamment sur leur nécessité de traiter les tâches d'évaluation au cas par cas. Pour sa conception, nous nous appuyons sur les questions **Q1** à **Q4** et proposons des moyens pour aider à y répondre sur un ensemble de données à la fois.

3.2 Objectifs et Directives de Conception

Nous cherchons à refléter d'une part les performances du réseau (**Q1** et **Q2**), et d'une autre, à retracer de manière topologique (c'est-à-dire au niveau de chaque couche) la classification de

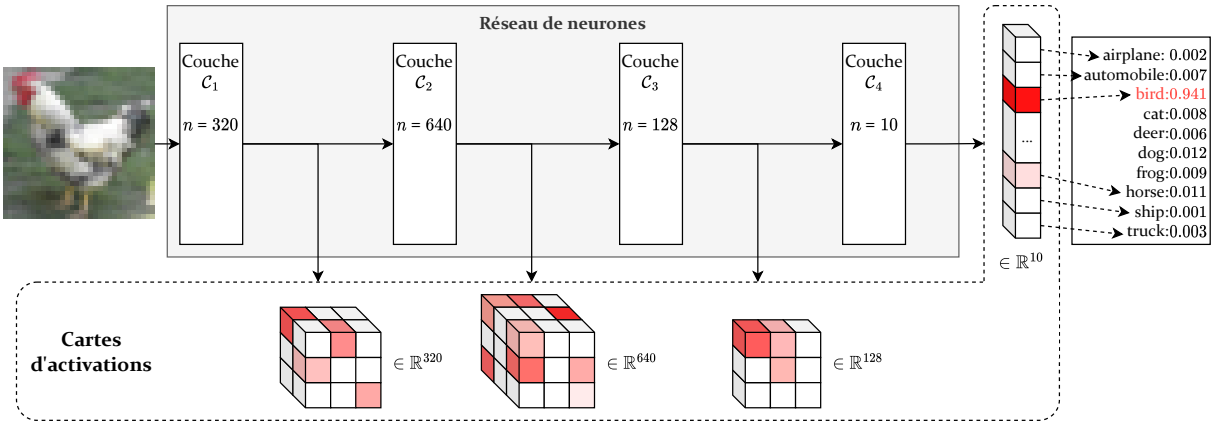


FIGURE 3.2 – Exemple de classification d'image. Les neurones s'activent plus (en rouge) ou moins (en blanc) en fonction de la donnée en entrée de chaque couche pour former des cartes d'activation dans des espaces à grandes dimensions. La sortie d'un réseau de classification est un vecteur de taille égale au nombre de classes détectable du réseau, dont la composante la plus élevée correspond à la prédiction du réseau (ici «bird»).

chaque donnée pour répondre aux questions **Q3** et **Q4**. En effet, l'utilisation de la topologie du réseau comme référence de notre outil de visualisation permet de positionner les décisions partielles du réseau, afin de montrer notamment la contribution de chaque couche, mais aussi de détecter l'endroit à partir duquel le réseau s'est trompé, que ce soit tôt dans la classification (impliquant donc des informations bas niveau) ou plus tard (impliquant donc des information haut niveau).

Pour ceci, nous étudions les cartes d'activations générées par chaque couche du réseau, obtenues en sortie d'une couche lors de la classification d'une instance. Ces cartes d'activation peuvent être vues comme des projections de la donnée d'entrée dans un espace spécifique défini lors de la construction du réseau (voir Figure 3.2). Par exemple, une couche dite «Dense» à 128 neurones projette sa donnée d'entrée dans un espace à 128 dimensions, tant dis qu'une couche de convolution à 6 filtres, ayant un noyau de taille 5×5 et prenant en entrée des matrices de taille 28×28 projette des données dans un espace à $24 \times 24 \times 6 = 3456$ dimensions. Dans ces cartes d'activation, nous nous attendons à trouver des informations montrant une classification préliminaire des données avant d'atteindre la dernière couche du réseau. Ce fonctionnement découle directement du comportement des couches du réseau, permettant une forme de classification progressive des données.

3.2.1 Classification progressive

Les premières couches d'un réseau sont paramétrées pour retrouver des attributs spécifiques aux données d'entrées (de bas niveau), tandis que les couches plus lointaines dans le réseau le sont pour retrouver des attributs spécifiques à la tâche demandée (de haut niveau). Ces attributs sont présents sous la forme de motifs dans les cartes d'activation générées par le réseau en sortie de chaque couche, c'est-à-dire des ensembles de neurones retournant des valeurs similaires sur plusieurs classifications. Nous exploitons les motifs présents dans ces cartes d'activation afin de détecter les données d'entrée partageant des motifs communs au niveau d'une couche du réseau, montrant ainsi leur classification similaire par le réseau.

La dimensionalité des cartes d'activation varie en fonction des algorithmes définis pour chaque couche du réseau, et la conception d'un encodage pour ces informations rejoint notre objectif de nous adapter à n'importe quelle nature d'information classée par le réseau. Pour cela, nous choisissons d'encoder toutes ces informations (données d'entrée, de sortie et cartes d'activation) sous la forme de vecteurs dans des espaces à hautes dimensions. La dimensionalité de ces matrices

Nature des données	Source	Encodage	Domaine
Image monochrome	MNIST [72]	Matrice 2D 28×28	$\mathbb{R}^{784}$
Image RGB	mini-imageNet [140]	Matrice 3D $84 \times 84 \times 3$	$\mathbb{R}^{21168}$
Séquence vidéo RGB	Sports-1M [63]	Matrice 4D $178 \times 178 \times 3 \times t$	$\mathbb{R}^{10^5 \times t}$
Mot	Wiki-News-300d [42]	Vecteur à 300 composantes	$\mathbb{R}^{300}$
Carte d'activation	VGG16 [124], Conv 4-2	Tenseur $8 \times 8 \times 256$	$\mathbb{R}^{16384}$

TABLE 3.1 – Exemples de données que l'on peut rencontrer dans des tâches de classification, et que l'on cherche à encoder d'une même façon. Dans le cas de la séquence vidéo, la dimensionalité des données dépend aussi de la longueur t de la trame étudiée (i.e. le nombre d'images dans la séquence).

est élevée, que ce soit pour les données d'entrées tout comme les cartes d'activation calculées par chaque couche (voir Table 3.1).

Dans ces vecteurs, nous devons trouver les informations nécessaires permettant de détecter si une couche arrive à discriminer la donnée d'entrée ou non, puis l'indiquer à l'utilisateur. Nous proposons de mesurer cette compréhension du réseau en calculant une métrique de similarité entre les cartes d'activation :

— Deux cartes d'activation similaires en sortie d'une même couche indiquent que celle-ci associe les éléments à une même catégorie (dont la sémantique nous est inconnue).
— Deux cartes d'activation non similaires en sortie d'une même couche indiquent que celle-ci associe les éléments à deux catégories distinctes.

Plus formellement, dans notre traitement de l'information, pour chaque élément $d \in D$ de notre jeu de données d'évaluation D, nous appelons $\mathcal{C}_i(d)$ la carte d'activation obtenue pour d à la sortie de la i-ème couche. Soit $Sim : \mathbb{R}^N \times \mathbb{R}^N \to \mathbb{R}$ une fonction qui mesure la similarité entre deux cartes d'activation. En s'appuyant sur le comportement progressif de la classification d'un réseau, nous nous attendons à ce que pour deux données d'entrée $d_1, d_2 \in D$ de même classe et pour $i < j$:

$$Sim(\mathcal{C}_i(d_1), \mathcal{C}_i(d_2)) \leq Sim(\mathcal{C}_j(d_1), \mathcal{C}_j(d_2)) \tag{3.2}$$

En comparant la similarité entre chaque carte d'activation d'une même couche, nous pouvons retrouver des ensembles de cartes d'activation similaires entre elles, formant ainsi la classification préliminaire recherchée. Sur un réseau parfait, ces groupes grandiraient en effectif au fil des couches traversées, jusqu'à être exclusivement composés de tous les éléments d'une même classe. Ces variations nous permettent alors d'obtenir des indices sur les performances et l'utilité de chaque couche du réseau. La Figure 3.3 montre un exemple de classification progressive que l'on pourrait étudier.

3.2.2 Méthode de visualisation

Nous cherchons à représenter la formation de groupes de cartes d'activation similaires au niveau de chaque couche du réseau étudié. Pour cela, nous avons considéré deux solutions : une visualisation par coordonnées parallèles dans un premier temps puis l'utilisation d'un diagramme de Sankey [121].

La visualisation par coordonnées parallèles est une méthode de visualisation pour étudier un ensemble de données D de dimension N. Cet espace est représenté sous la forme de N axes verticaux alignés horizontalement, et la coordonnée d_n d'un élément $d \in D$ dans la n-ième dimension est représentée par un point à l'ordonnée d_n sur le n-ième axe. Les N points découlant des coordonnées d'un même élément d sont ensuite reliés par une ligne, et ce processus est répété pour tous les éléments de D.

Cette méthode de visualisation peut être appliquée à notre cas en associant pour chaque élément $d \in D$ une valeur $V_d(x_1, x_2, \ldots, x_N)$ de dimension N, avec N le nombre de couches du réseau

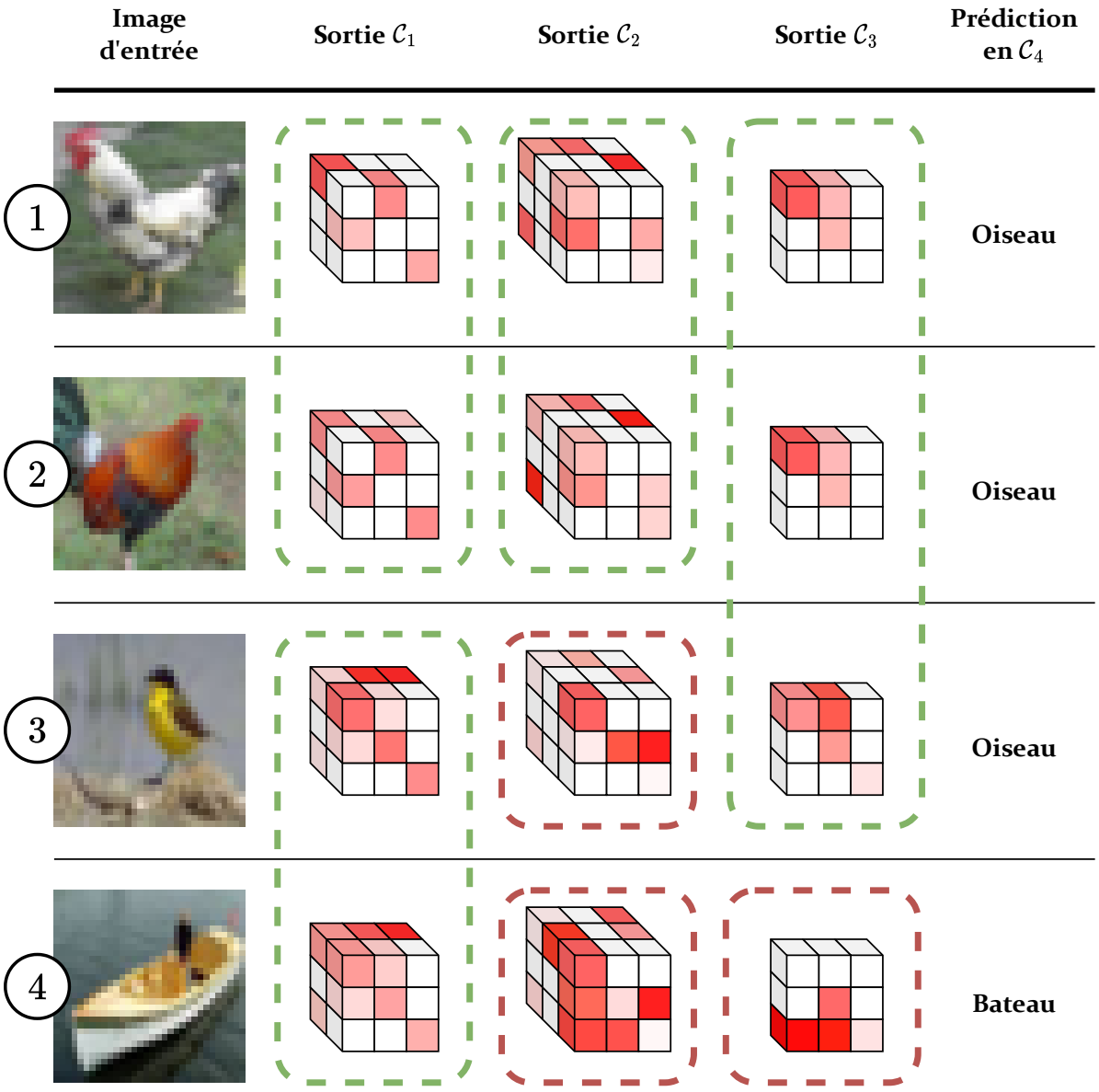


FIGURE 3.3 – Exemple de groupement par similarité de cartes d'activation. La couche $\mathcal{C}_1$ produit des cartes d'activation semblables pour les images 1 et 2 et les cartes d'activation restent similaires pendant le reste du traitement, les deux images ont donc été reconnues tôt dans le réseau à l'aide d'attributs de bas niveau. À l'inverse, les images 3 et 4 sont aussi traitées similairement par $\mathcal{C}_1$, mais ne sont plus reconnues comme similaire dès la couche suivante. La couche $\mathcal{C}_2$ les a donc discriminés dans le processus de classification. Au niveau de la couche $\mathcal{C}_3$, l'image 3 a été reconnue comme similaire aux images 1 et 2 par le réseau à l'aide d'attributs de haut niveau.

étudié. Chaque valeur $x_1, ..., x_N$ correspond à un identifiant de groupe réunissant les cartes d'activation d'éléments similaires.

Ainsi donc, cette méthode de visualisation appliqué à notre tâche consiste à construire un axe vertical pour chaque couche du réseau, et de les disposer horizontalement afin de représenter l'architecture du réseau. Sur chaque axe, nous construisons un point pour chaque donnée d'entrée d, ils représentent la carte d'activation $\mathcal{C}_i(d)$ en sortie de la couche $\mathcal{C}_i$ pour la donnée d. Ces points sont positionnés de manière à ce que les points de cartes d'activation similaires soient proches les unes des autres, et que les groupes formés par cette représentation soient espacés les uns des autres le long de chaque axe. Enfin, les points représentant une carte d'activation résultant d'une

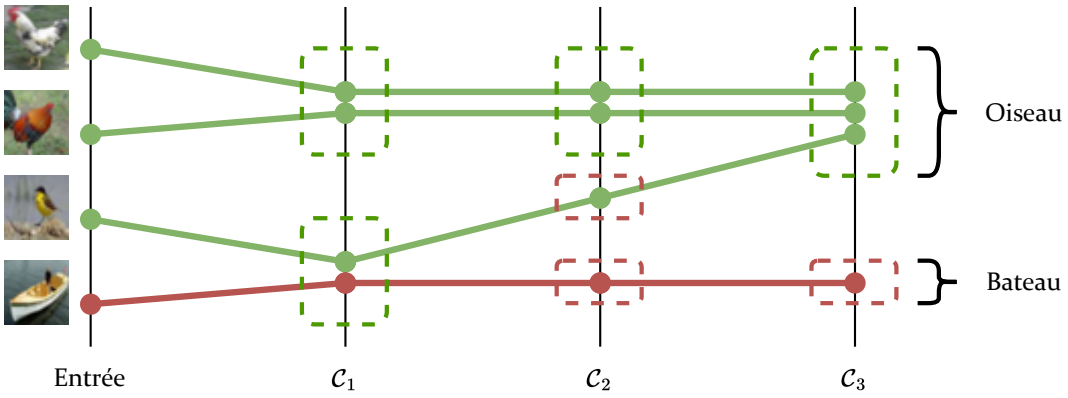


FIGURE 3.4 – Exemple de représentation de la classification progressive en utilisant la visualisation par coordonnées parallèles. Chaque axe vertical représente une couche du réseau étudié, de $\mathcal{C}_1$ la couche d'entrée à $\mathcal{C}_3$. Sur ces axes sont disposés des points représentant une carte d'activation, et les points résultant de la même donnée d'entrée sont reliés entre eux. Ces points sont colorés en fonction de la classe (vérité terrain) de la donnée d'entrée. Lorsque qu'un groupe de cartes d'activation similaires est formé en sortie d'une couche, les points correspondants à ces cartes d'activation sont rapprochés les uns des autres sur l'axe de la couche concernée.

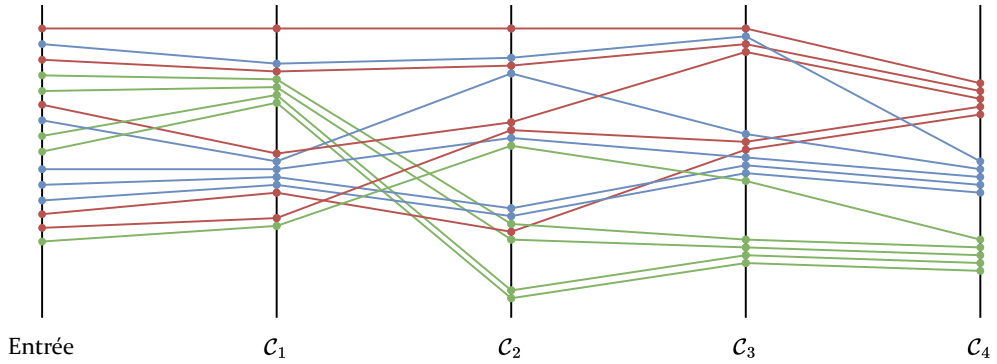


FIGURE 3.5 – Autre exemple de représentation de la classification progressive en utilisant la visualisation par coordonnées parallèles. Cet exemple implique la classification d'une quinzaine d'éléments. On peut y remarquer le début d'une complexité à la lecture, notamment marquée par les croisements des arêtes, lorsque l'effectif des groupes change fréquemment d'une couche à une autre.

même donnée d'entrée sont reliés entre eux, et sont colorés en fonction de leur vérité terrain. La Figure 3.4 montre un exemple de visualisation que nous pourrions obtenir avec cette méthode sur les mêmes données montrées en Figure 3.3. Nous pouvons y observer les quatre éléments colorés suivant leur vérité terrain (la classe `oiseau` est en verte ■ et la classe `bateau` en rouge ■). Les cartes d'activations similaires sont regroupées sur les axes correspondant aux couches sur lesquelles cette similarité a été détectée. Nous pouvons voir que les deux éléments ont été groupés dès le premier axe et forment donc des données classées tôt par le réseau, tandis que la distinction entre le 3e et 4e élément ne s'est faite que plus tard, et sont donc plus difficiles à classer pour le réseau.

Cette méthode nous permet donc d'obtenir des regroupements d'arêtes que nous pouvons assimiler à des flux de données qui se distinguent en des groupements de données similaires. Cependant, l'exemple montré ne concerne qu'un scénario de classification impliquant quatre éléments traités par le réseau. Un autre exemple impliquant une quinzaine d'éléments à classifier est montré en Figure 3.5. Nous pouvons y observer des croisements d'arêtes qui rendent plus difficiles

la lecture de la visualisation. Habituellement, Les jeux de données d'évaluations sont composés de plusieurs milliers d'éléments à classifier, et l'utilisation de cette méthode de visualisation implique donc la construction de très nombreux points rapprochés sur les axes. Cette construction favorise la multiplication des croisements d'arêtes rendant la lecture de la visualisation encore plus difficile [120].

Pour éviter ce problème de passage à l'échelle, nous choisissons d'utiliser un diagramme de Sankey [121]. Il s'agit d'un diagramme de flux utilisé pour la représentation de répartition de ressources dans un processus. Par rapport à la visualisation par coordonnées parallèles, les éléments ne sont plus représentés individuellement sur les axes et par des arêtes, mais plutôt sous la forme de flux regroupant des éléments ayant un comportement commun. Cette vue nous intéresse ici puisqu'elle nous permet d'agréger en un seul flux les arêtes de points se trouvant dans un même groupe à la couche $\mathcal{C}_n$ et à la couche $\mathcal{C}_{n+1}$. Ceci réduit le nombre d'éléments à afficher et donc la complexité de la visualisation, notamment au niveau des dernières couches du réseau où les compositions des groupes se rapprochent du résultat de la classification. Afin de pouvoir reconnaître la composition des flux, nous représentons chaque classe du jeu de données par une couleur, et les flux sont colorés en fonction de la classe des éléments qui le composent. Pour éviter d'avoir un flux représentant des données de plusieurs classes, nous séparons aussi au sein d'un même flux les données de classes différentes, même si elles se retrouvent dans le même groupe de cartes d'activation similaires en sortie de la couche suivante [120]. Dans la suite du chapitre, nous appelons «sous-groupe» un ensemble d'éléments appartenant à une classe commune au sein d'un groupe de carte d'activations similaires.

Pour résumer, cette vue nous apporte plusieurs avantages :

— Elle représente le traitement de plusieurs milliers de données par le réseau plutôt qu'un seul et unique exemple contrairement aux méthodes d'explication listées précédemment.
— Elle s'adapte à différents types de données à classifier, et donc n'est pas restreinte à la classification d'images, par exemple.
— Elle est suffisamment synthétique pour que l'utilisateur puisse interpréter des ensembles de données traitées de façon identique par le réseau sans devoir étudier chaque carte d'activation individuellement.
— L'agrégation des éléments traités similairement par le réseau en un diagramme de Sankey permet de synthétiser des scénarios de classification réalistes impliquant des milliers d'éléments en une interface utilisateur.

Pour construire ce diagramme, nous avons besoin de connaître quelles données sont traitées de manière similaire entre deux couches consécutives du réseau et de représenter cet ensemble par un flux. Formellement, la composition de l'ensemble des flux $\mathcal{F}_{n,n+1}$ entre les deux couches $\mathcal{C}_n$ et $\mathcal{C}_{n+1}$ est défini de la façon suivante :

$$\mathcal{F}_{n,n+1} = \bigcup_{t \in GT} (g_n, g_{n+1}) \quad g_n \in G(\mathcal{C}_n), g_{n+1} \in G(\mathcal{C}_{n+1}), \mathcal{GT}(g_n) = \mathcal{GT}(g_{n+1}) = t$$

avec $G(\mathcal{C}_n)$ la fonction retournant l'ensemble des groupes de cartes d'activation similaires en sortie de la couche $\mathcal{C}_n$, GT l'ensemble des classes du jeu de données, et $\mathcal{GT}(e)$ une fonction qui retourne la classe $t \in GT$ de l'élément e.

Pour calculer la similarité entre les cartes d'activation, nous nous orientons vers l'utilisation de méthodes de partitionnements (*clustering*) non-supervisées qui sont souvent employées lors du traitement de données de masse, impliquant fréquemment des données hautement dimensionnelles. Ces méthodes de partitionnements font partie de l'ensemble des algorithmes d'apprentissage. On peut distinguer deux sous-ensembles de méthodes : les méthodes dites supervisées qui nécessitent d'avoir au préalable une connaissance sur la structure des données traitées afin de l'apprendre à l'algorithme et les méthodes non-supervisées qui ne nécessitent pas d'avoir cette connaissance a priori. Ces méthodes non-supervisées sont donc bien adaptées à notre cas d'usage, où nous avons uniquement connaissance de l'encodage des données en entrée du réseau et la classe à laquelle elle appartient ; nous n'avons donc pas d'information sur la façon dont elles sont

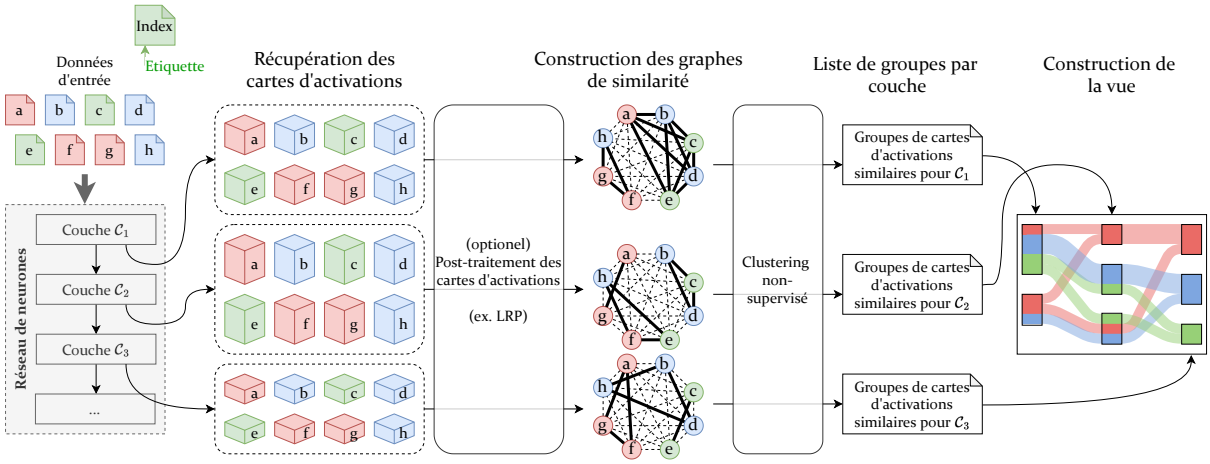


FIGURE 3.6 – Fonctionnement de la méthode proposée. À partir d'un modèle entraîné et d'un jeu de données d'évaluation, nous produisons une visualisation sous la forme d'un diagramme de Sankey montrant la classification progressive du réseau sur les données.

transformées par le réseau lors de leur classification, et ne pouvons donc pas informer l'algorithme de partitionnement sur la structure des données qu'il doit traiter. Ces méthodes nous permettent d'obtenir la liste des groupes de cartes d'activation similaires $G(\mathcal{C}_n)$ pour chaque couche n du réseau.

3.3 Présentation de la méthode

La Figure 3.6 récapitule le fonctionnement de notre méthode. Dans un premier temps, nous extrayons les données produites par chaque couche du réseau, à savoir un ensemble de cartes d'activation pour chacun des éléments du jeu de données d'évaluation. Nous avons mentionné en Section 3.1 la méthode d'explicabilité LRP [88] qui permet de mesurer la contribution individuelle de chaque neurone sur la classification finale dans le réseau. Nous pouvons appliquer ici cette méthode sur les cartes d'activation extraites pour limiter le bruit provoqué par les activations de neurones n'ayant que trop peu contribué à la classification des éléments, afin d'améliorer la qualité du partitionnement des données. Nous traitons ensuite chaque couche du réseau séparément en calculant des groupes d'éléments traités de façon similaire. Pour cela, nous employons une méthode de partitionnement non-supervisée pour détecter les éléments traités similairement par chaque couche du réseau. Dans ce chapitre, nous utilisons le *Markov Clustering Algorithm* [138, 139] (MCL), une technique de partitionnement qui nécessite la modélisation d'un graphe où chaque sommet correspond à un élément à regrouper, et les arêtes entre ces sommets sont pondérées par un indice de similarité. Nous construisons donc pour chaque couche du réseau un graphe, où chaque sommet correspond à une carte d'activation produite par cette couche. Chaque sommet est relié aux autres par une arête que nous pondérons par la proximité des deux cartes d'activation dans $\mathbb{R}^N$.

Les partitions ayant été calculées pour chaque couche du réseau, nous les représentons ensuite par des flux à la manière d'un diagramme de Sankey : chaque axe vertical représente une couche du réseau, et chaque classe classifiée possède une couleur distincte. Les flux sont colorés en fonction de leur composition, et sont organisés de manière à ce que les données traitées similairement soient rapprochées sur l'axe vertical correspondant à la couche où cette similarité a été calculée.

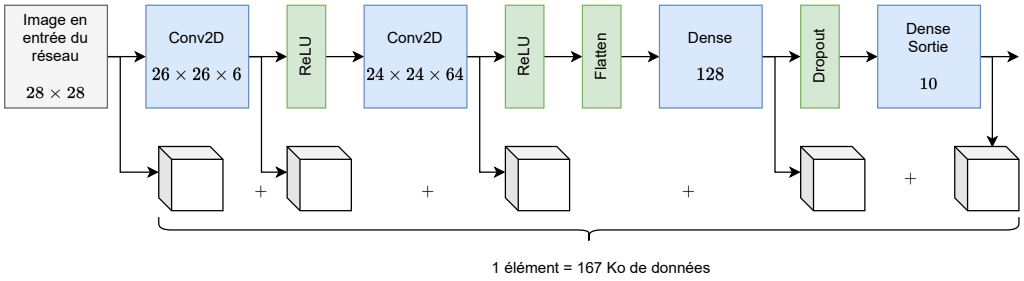


FIGURE 3.7 – Taille des données collectées de notre réseau inspiré de LeNet pour un élément. Le jeu de données regroupant 10 000 éléments, nous nous retrouvons avec 1.7Gio de données à traiter pour un réseau et un jeu de données pourtant très simples.

3.4 Technique et Implémentation

Cette section présente les aspects techniques et les choix d'implémentation pour cette méthode, allant du traitement de la donnée à la conception de la visualisation.

Afin de récupérer les cartes d'activation du réseau étudié, il nous faut modifier son architecture afin de paramétrer chaque couche intermédiaire du réseau comme étant une couche de sortie (faisable sur *TensorFlow* [1], utilisé dans notre implémentation, et *PyTorch* [104], une autre bibliothèque populaire pour la conception de réseaux de neurones artificiels). Cette modification de l'architecture du réseau nous permet de changer la fonction de prédiction du réseau de $f(d) \rightarrow \mathcal{D}(\mathcal{C}_N)$ en $f'(d) \rightarrow \bigcup_{n=1}^{N} \mathcal{D}(\mathcal{C}_n)$ pour un réseau à N couches avec d un élément du jeu de données d'évaluation, et $\mathcal{D}(\mathcal{C}_n)$ le domaine de définition de la n-ème couche du réseau.

3.4.1 Extraction et traitement des données du réseau

Extraction. Notre méthode s'appuie sur les données calculées par les couches du réseau avant qu'elles soient traitées par la couche suivante. Nous extrayons les cartes d'activation du réseau de classification pour former un ensemble de vecteurs à dimensions variables. Si le temps de traitement reste pratiquement inchangé puisque les calculs de prédictions restent les mêmes pour le réseau, l'espace mémoire nécessaire pour stocker ces informations est significativement plus grand, puisque nous obtenons plus de données en sortie d'une seule prédiction. Cette augmentation de la donnée générée est à considérer pour la totalité du jeu de données d'évaluation, puisque notre étude du réseau se base sur celui-ci. De nos jours, les machines de calculs sont composées d'une multitude de cœurs pouvant traiter des données de manière parallèle et efficacement, mais les données que nous traitons ici sont suffisamment grandes pour que l'espace de stockage ainsi que la mémoire vive de la machine devienne une limitation sur les jeux de données et/ou architectures plus grandes. Sur un jeu de données et une architecture simples, tels que LeNet5 et MNIST, la collecte des cartes d'activation demande environ 1.7 Gio de mémoire. Lors de l'utilisation de réseaux plus complexes tel que VGG16 [124], les informations collectées pèsent environ 13 Gio. Si l'on étudie le jeu de données ImageNet, ce pour quoi VGG16 a été conçu, le traitement demande alors environ 152 Gio de mémoire. Il faut aussi considérer le stockage des paramètres du modèles ou encore la mémoire *overhead* nécessaire à la bibliothèque utilisée pour calculer la classification.

Afin de pouvoir adapter notre méthode sur des cas d'études moins triviaux, nous avons choisi d'utiliser une infrastructure de calcul distribué, sur laquelle il est plus facile d'augmenter la mémoire de calcul ou ses performances distribuées que sur une machine traditionnelle. Ces infrastructures matérielles sont très souvent utilisées lorsqu'il est question de traitement *Big Data* et ont l'avantage de pouvoir être étendues «horizontalement» (multiplier le nombre de nœuds de calcul et de stockage) plutôt que «verticalement» (remplacer le processeur par un autre plus puissant, par du stockage plus grand) réduisant de manière significative les coûts de traitement de la donnée. Cette approche distribuée demande cependant à repenser la conception des algorithmes

de traitement, et à tenir compte de nouvelles limitations. Par exemple, par rapport à une machine traditionnelle où les lectures et écritures en mémoire vive ont des temps d'exécution négligeables, les infrastructures distribuées demandent à ce que la données soit équitablement distribuée sur les nœuds de calcul. Cette répartition a un coût en temps non négligeable, et l'organisation même des tâches demande un coût mémoire important supplémentaire à prendre en compte. De manière analogue, la collecte des résultats de calcul demande une gestion réfléchie pour ne pas saturer la mémoire du nœud directeur. Pour faciliter la gestion de la mémoire, il est possible pour les nœuds de calcul de lire et d'écrire sur un espace de stockage commun à tous les nœuds. Cette fonctionnalité est utile pour pouvoir sauvegarder les résultats de calculs intermédiaires sans occuper la mémoire vive des machines. Nous pouvons ainsi appliquer une multitude de nouvelles opérations sur ces résultats sans devoir reprendre l'exécution des algorithmes de zéro. Ceci économise du temps et des ressources de calculs à moindre coût, qui est celui de lecture et d'écriture dans cet espace de stockage. La complexité supplémentaire dans la conception des algorithmes de traitement se justifie donc par la capacité à supporter l'étude de scénarios significativement plus complexes. Ici, nous utilisons l'infrastructure *Labo's in the Sky with Data* (LSD) du LaBRI qui est composée de 51 machines de calcul totalisant 2.7Tio de mémoire vive, permettant alors le traitement et le stockage de données massives. Ces opérations sont pilotées par les couches logicielles *Apache Spark* [153] et *Hadoop* [14] qui nous permettent d'exploiter l'infrastructure efficacement. Pour cette étape d'extraction des données, chaque nœud de calcul est chargé de collecter et indexer les cartes d'activation d'un ensemble d'éléments du jeu de données.

Post-traitement. Une fois les données extraites, nous pouvons, si besoin, appliquer une méthode de post-traitement sur les cartes d'activation. Notre processus n'impose pas de contrainte particulière aux méthodes de post-traitement utilisées, si ce n'est que chaque carte d'activation extraite d'une couche doit rester un vecteur dans un espace dimensionnel commun avec les autres cartes d'activation de la même couche. Dans notre prototype, nous avons appliqué la méthode LRP. Comme cette méthode s'applique à la classification d'un seul élément à la fois, nous pouvons appliquer la méthode directement depuis les nœuds de calcul sans avoir besoin de collecter toutes les cartes d'activation une première fois, ce qui nous évite une redistribution (*shuffle*), opération très coûteuse sur les architectures distribuées.

Partitionnement. Le partitionnement s'applique au niveau de chaque couche individuellement, il n'est donc pas nécessaire de conserver en mémoire les cartes d'activation en sortie des autres couches du réseau, et celles-ci peuvent être conservées dans l'espace de stockage de l'infrastructure distribuée. Nous avons étudié plusieurs options pour le calcul du partitionnement : *K-Means*, l'*Analyse en Composantes Principales* [147] (PCA) et le *Markov Clustering Algorithm* [139, 138] (MCL).

K-Means est un algorithme de partitionnement qui peut s'utiliser sur des grands jeux de données multidimensionnels, mais qui repose sur le calcul de distance euclidienne pour calculer les groupes. L'utilisation de cette métrique est problématique dans les espaces hautement dimensionnels, très en lien avec le phénomène du fléau de la dimensionalité [9]. En effet, l'utilisation de la distance euclidienne dans de tels espaces perd en significativité à mesure que le nombre de dimension croît, puisque le calcul de la somme des carrés de chaque composante hausse les distances entre éléments de manière exponentielle.

PCA [147] est une autre méthode permettant de projeter des données hautement dimensionnelles dans des espaces plus petits et donc plus simples à étudier en se basant sur le calcul de vecteurs propres. Nous pourrions utiliser conjointement PCA et K-Means pour projeter les cartes d'activation dans un espace plus simple dans un premier temps, puis calculer les partitions avec K-Means ensuite puisque les distances entre éléments projetés feront alors meilleur sens. Néanmoins, cette méthode possède ses limitations, notamment au niveau de la complexité du calcul qui est de $O(C^2.D + C^3)$ avec D la taille du jeu de données et C le nombre de composantes de ces données, donc leur nombre de dimensions. Bien qu'une implémentation distribuée du calcul de PCA existe dans *Apache Spark*, la complexité de nos données ici est trop importante pour que PCA soit utilisée.

La solution optée dans ce chapitre est l'utilisation de MCL [139], qui calcule un partitionnement de données dans un graphe en simulant une marche aléatoire. Cet algorithme utilise une mesure de dissimilarité à spécifier en amont pour calculer les groupes, et a une complexité théorique de $O(V^3)$ où V est le nombre de sommets dans le graphe. L'implémentation *mcl* disponible sur le site de son auteur [138] est une optimisation de l'algorithme initial afin de réduire considérablement sa complexité en dessous de $O(V^2)$. Cette implémentation n'est cependant pas prévue pour être distribuée sur les infrastructures telles que LSD, et doit donc être exécutée sur une machine locale.

Pour utiliser cet algorithme, nous devons donc transformer notre ensemble de carte d'activation en un graphe pondéré $\mathcal{G}_{\mathcal{C}}(V, E)$ où chaque carte d'activation $\mathcal{C}(d)$ est encodée par un sommet $v_d \in V$ et la dissimilarité entre deux cartes d'activation $\mathcal{C}(a)$ et $\mathcal{C}(b)$ est la pondération de l'arête $e_{ab} \in E$ reliant les sommets v_a et v_b. La mesure de dissimilarité utilisée dans ce chapitre est la distance cosinus, qui calcul la différence angulaire entre deux vecteurs plutôt que celle de leur position comme le fait la distance euclidienne, et donc s'adapte mieux aux espaces à hautes dimensions. Le calcul de la dissimilarité entre les éléments est similaire au calcul d'une matrice de distance en termes de complexité, mais l'utilisation de LSD nous permet d'accélérer cette étape en répartissant la tâche sur les nœuds de calculs. Ceci fait, la matrice est alors reconstruite sur le nœud directeur avant d'être transférée sur une machine performante locale. Cette étape est essentiellement la plus coûteuse en espace mémoire, puisque le nœud directeur doit disposer de suffisamment de mémoire, en plus des ressources nécessaires au fonctionnement d'Apache Spark et Hadoop, pour reconstruire la matrice de taille $D \times D$.

La matrice étant reconstruite, elle peut maintenant être traitée par *mcl*. L'auteur de *mcl* précise que pour que l'algorithme fonctionne efficacement sur un graphe dense, ce qui est le cas ici puisque la matrice correspond à un graphe entièrement connecté, il est recommandé d'élaguer au préalable ce graphe en appliquant un filtre *k-NN* ou de seuillage. Le filtre *k-NN* consiste à conserver uniquement pour chaque sommet les k arêtes les reliant aux voisins les plus similaires. Le seuillage consiste à conserver uniquement les arêtes dont la pondération est inférieure à une valeur spécifiée, donc à conserver les voisins dont la valeur de dissimilarité est suffisamment faible. Dans les deux cas, l'objectif est de conserver l'information que l'on considère comme importante (la similarité entre les éléments) et de retirer la moins importante (on ne cherche pas à grouper deux éléments ayant une dissimilarité trop grande). Dans ce chapitre, nous avons choisi d'utiliser le filtre *k-NN* avec $k = 10$, qui correspond au nombre de classes d'éléments dans les jeux de données étudiés dans nos tests. Il faut noter que ce paramètre k a une fonction différente de celui de *K-means*, et n'indique pas un nombre de groupes à détecter par *mcl*.

Données finales. Une fois l'exécution de *mcl* terminée, nous disposons, pour chaque couche du réseau, d'un ensemble de groupes d'identifiants qui correspondent aux cartes d'activation ayant été traitées de manière similaire par la couche en question. Le traitement des données permet donc de réduire un jeu de données de potentiellement plusieurs dizaines de Gio en des listes d'identifiants de quelques Kio au maximum.

3.4.2 Conception de la méthode de visualisation

La conception de la représentation se fait en deux étapes. La première consiste à construire les composants essentiels à la représentation graphique, et la seconde consiste à organiser ces composants de manière à faciliter la lecture et l'interprétation de la visualisation. Nous pouvons représenter les composants de la visualisation par un dessin de graphe $\mathcal{D}$ $(V_{\mathcal{D}}, E_{\mathcal{D}})$ en deux dimensions : les sommets $V_{\mathcal{D}}$ représentent des groupes d'éléments similaires sur une couche donnée et les arêtes $E_{\mathcal{D}}$ représentent les flux entre deux groupes de couches différentes. Conformément à nos choix de conception en Section 3.2, nous créons un sommet par sous-groupe, c'est-à-dire par groupe d'éléments calculé et par classe d'éléments présente dans le dit groupe. Ces sous-groupes sont ordonnés verticalement suivant la couche à laquelle ils appartiennent, et sont espacés de manière à pouvoir différencier les éléments d'un groupe d'éléments similaires à un autre.

Ensuite, nous relions les sommets de couches successives qui possèdent au moins un identifiant d'élément en commun. Les arêtes construites de la sortes sont pondérées par le nombre d'identifiants en communs qu'elles représentent. Cette pondération nous sert à organiser la disposition des composants du graphes et à mesurer la taille des flux que l'on doit dessiner. Nous organisons les composants de manière à limiter les croisements d'arêtes à trois niveaux :

— Le premier concerne la couche dans laquelle les cartes d'activation ont été récupérées. Les sommets appartenant au même graphe sont ainsi alignés sur un même axe vertical sur notre visualisation.
— Le second concerne l'organisation des groupes calculés par MCL, les ensembles de groupes de cartes d'activation similaires sont déplacés verticalement pour limiter les croisements de flux.
— Le dernier concerne l'organisation des groupes d'éléments d'une classe commune au sein de ceux calculés par MCL. Ces sommets sont réorganisés localement pour améliorer la lecture de chaque groupe sans perturber l'intégrité de la visualisation.

L'organisation de ces composants de graphes de manière à limiter les croisements font partie du problème de minimisation du croisement entre couches (*2-layer crossing minimization* [61]), un problème *NP*-difficile. Ici, nous avons utilisé l'algorithme de Sugiyama [129] qui réarrange les sommets par rapport aux barycentre de leur position avec celle de leurs voisins. Cet algorithme a été choisi par sa simplicité d'implémentation, mais n'importe quelle méthode réduisant le nombre de croisement d'arêtes d'un graphe multi-couche peut être utilisée ici. Afin d'éviter les croisements des flux les plus gros, nous considérons le poids des arêtes dans l'application de l'algorithme. Cet algorithme est appliqué dans un premier temps sur l'organisation des groupes le long des axes, puis au niveau des sous-groupes. Après l'application de l'algorithme de réduction de croisements, nous dessinons le diagramme de Sankey en nous appuyant sur les coordonnées 2D de chaque sommet du graphe pour positionner les composants de la visualisation.

Couches du réseau. Les couches du réseau sont représentées par des axes verticaux alignés horizontalement suivant l'ordre défini dans l'architecture du réseau. Ces axes sont accompagnés d'une étiquette pour pouvoir les identifier. Pour faire le lien avec les performances du réseau analysé, nous avons rajouté deux axes *Prediction* et *Ground Truth* qui reflètent respectivement les prédictions du réseau et les classes réelles des éléments, et qui simulent une matrice de confusion.

Groupes et sous-groupes. Les groupes calculés par MCL sont représentés par des rectangles alignés sur l'axe vertical correspondant à la couche où ils ont été calculés. Leur taille varie de manière à représenter la proportion des données qui en fait partie. Ces groupes respectent le positionnement des sommets du graphe calculé à l'étape précédente. En ce qui concerne les sous-groupes, ils sont représentés comme étant une fraction du rectangle préalablement dessiné. Ces fractions sont colorées en fonction de la classe qu'ils représentent, et positionnés toujours en respectant l'ordre calculé par l'algorithme de minimisation de croisements. Enfin, ils sont dimensionnés de façon à représenter leur proportion dans le groupe dont ils font partie.

Flux. Les flux sont dessinés en respectant les connexions des arêtes du graphe préalablement calculé. Nous dessinons ces flux en utilisant des courbes de Bézier pour faciliter la lecture des flux qui se croisent. L'épaisseur de ces flux est proportionnelle à la quantité d'éléments présente dans un flux (c'est-à-dire la proportion d'éléments d'un même groupe et d'une même étiquette). Enfin, ces flux sont colorés de la même couleur que l'étiquette qu'ils représentent. Nous choisissons une teinte légèrement transparente pour rendre visibles les flux qui se croisent dans la visualisation.

Légende. La dernière chose à dessiner est la légende, qui indique à quelle classe correspond chaque couleur utilisée. Celle-ci est décrite horizontalement sous le diagramme.

Tout le processus de construction de la visualisation est résumé en Figure 3.8.

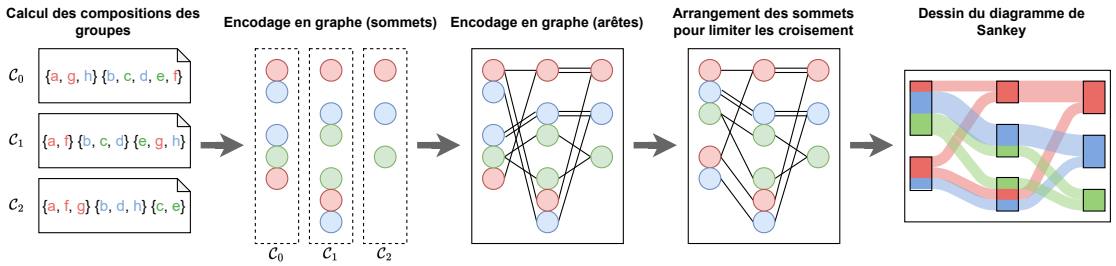


FIGURE 3.8 – Construction de la visualisation par diagramme de Sankey. Les groupes calculés en Section 3.4.1 sont encodés en un graphe 2D où chaque sommet représente un sous-groupe d'identifiants dont les éléments appartenant à la fois au même groupe calculé par *mcl* et à une même classe. Ces sommets sont alignés verticalement selon la couche concernée par le partitionnement de manière à représenter l'architecture du réseau étudié. Ils sont ensuite reliés entre eux si ils font partie de deux couches successives, et s'ils possèdent tous les deux au moins un identifiant d'élément en commun. Ces arêtes sont pondérées par le nombre d'identifiants en communs. Les sommets du graphe sont ensuite réarrangés de manière à limiter les croisements de leurs arêtes et faciliter la lecture de la visualisation. Le diagramme de Sankey est ensuite construit à partir des positions des sommets du graphes, et l'épaisseur des flux en fonction de la pondération de chaque arête.

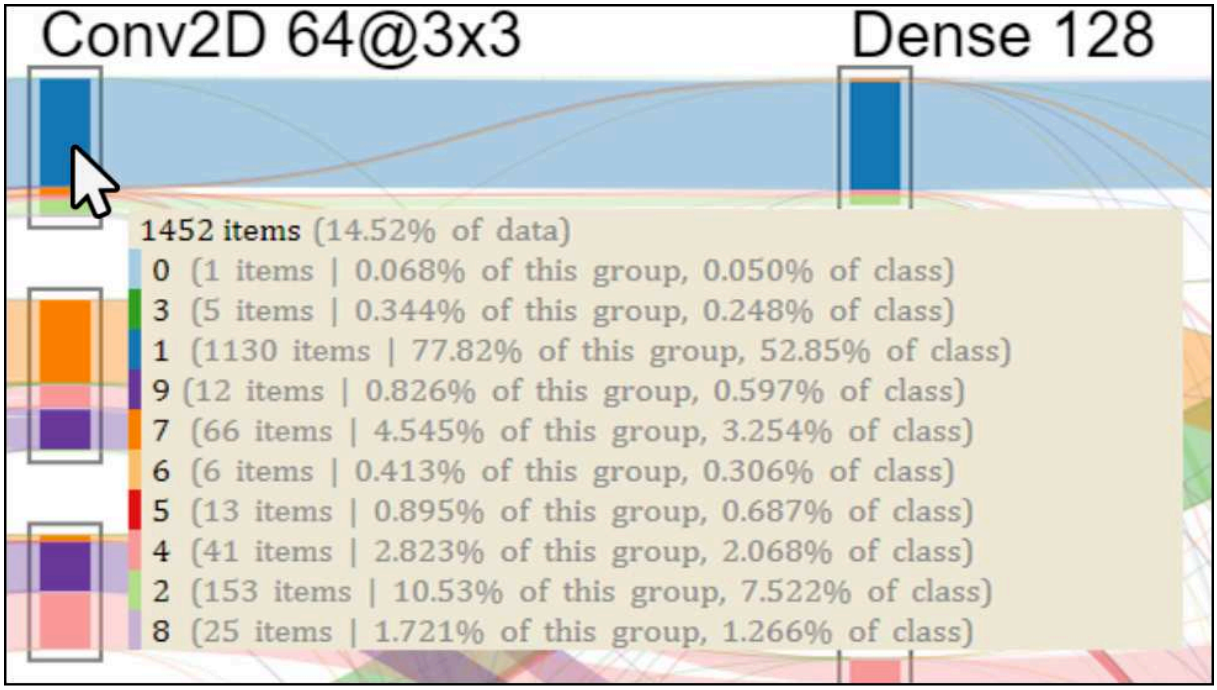
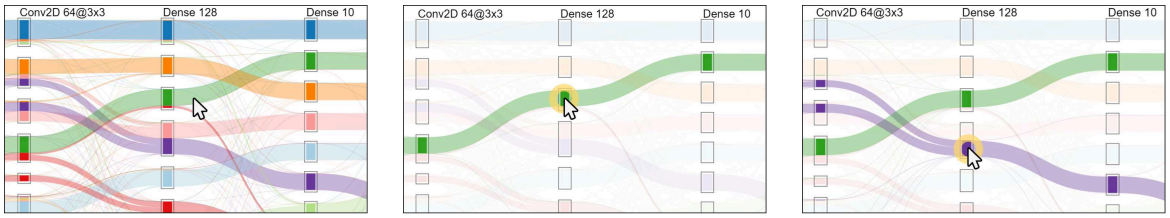


FIGURE 3.9 – Une fenêtre détaillant la composition de ce groupe apparaît lorsque le curseur de la souris pointe sur un groupe. On y obtient des informations sur la quantité d'éléments présents dans le groupe et leur classe, ainsi que la proportion de la totalité des éléments de chaque classe présente dans le groupe pointé. Ces informations nous permettent de quantifier les discriminations faites par la couche et de potentiellement détecter des *outliers* si ceux-ci ne sont pas visibles sur les groupes dessinés.

3.4.3 Interactions utilisateur

Afin d'afficher plus d'informations sur les composants et de faciliter la lecture des flux affichés à l'écran, notre outil de visualisation dispose de diverses interactions pour l'utilisateur.

Composition des groupes. Le survol de la souris sur les groupes dessinés permet d'obtenir



(a) Vue de base (b) Clic sur un premier sous-groupe (c) Clic sur un second sous-groupe

FIGURE 3.10 – Un clic sur un sous-groupe permet de mettre en évidence son flux de données, indiquant la provenance de son effectif, et comment celui-ci évolue plus tard dans le réseau. Plusieurs flux peuvent ainsi être mis en évidence, et cette vue filtre peut être désactivée en cliquant n'importe où ailleurs dans l'outil.

le nombre d'éléments du groupe et de chaque classe qui le compose comme montré en Figure 3.9. La proportion des éléments d'une classe par rapport à la totalité des éléments de cette même classe est indiquée sous la forme d'un pourcentage, permettant de quantifier plus précisément les ensembles discriminés par la couche. Une proportion occupée élevée d'une classe en particulier permet de détecter une «classification» tôt dans le réseau. Nous pouvons aussi avoir plus de précisions concernant les éventuels autres classes présentes dans le groupe, mais pas assez nombreuses pour être dessinées correctement (on parle alors souvent d'intrus ou d'*outliers*).

Filtrage des flux de données. Il est possible de filtrer les informations affichées sur l'outil en cliquant sur le sous-groupe souhaité pour pouvoir mieux étudier les mouvements des éléments d'un sous-groupe en particulier, comme montré en Figure 3.10. Ainsi, tous les sous-groupes de l'outil de visualisation contenant un ou plusieurs éléments du sous-groupe sélectionné sont mis en évidence, et les autres sont mis en retrait en diminuant leur opacité. Les flux reliant ces sous-groupes sont eux-aussi mis en évidence. Cette interaction permet notamment d'améliorer la lisibilité des divergences et convergences des données composant un sous-groupe en particulier, dans le cas d'étude d'architectures ou de tâches de classifications plus complexes. Ce filtrage d'élément peut être étendu à plus d'un sous-groupe en particulier en cliquant sur d'autres sous-groupes dans la visualisation. Il est également possible d'afficher uniquement les sous-groupes et flux contenant des éléments d'une classe de données en particulier (ou plusieurs si besoin) en cliquant sur la classe voulue en légende. Pour effacer tous les filtres et retrouver la vue initiale, il suffit de cliquer dans un espace vide de la visualisation.

Implémentation des interactions. Notre méthode de visualisation implique potentiellement d'afficher de nombreux composants (axes, groupes, flux, etc.). Dans le pire des cas, nous pouvons avoir chaque sous-groupe d'une couche $\mathcal{C}_n$ (potentiellement égal au nombre de classes dans le jeu de données) connecté à tous les groupes de la couche $\mathcal{C}_{n+1}$, et ce entre chaque couche consécutive du réseau. Si MCL calcule un nombre élevé de groupes, et/ou si le nombre de classes du jeu de données est grand, alors le nombre de composants à dessiner pourrait drastiquement augmenter. Afin d'éviter de devoir itérer sur chaque composant pour détecter celui avec lequel l'utilisateur cherche à interagir, nous décidons d'utiliser une table de correspondance, communément appelée *LookUp Table* (*LUT*). En utilisant un espace de dessin $\mathbf{C}_{alt}$ parallèle à celui que l'utilisateur observe $\mathbf{C}$, nous attribuons une couleur unique à chaque composant interactif reproduit sur cet espace de dessin. Lors de l'utilisation de la visualisation, nous récupérons la position du curseur dans $\mathbf{C}$ et la reportons sur $\mathbf{C}_{alt}$, nous pouvons en obtenir la couleur du pixel «survolé» dans cet espace de dessin alternatif. Nous récupérons ensuite le composant sélectionné par l'utilisateur via la *LUT* associant la couleur du pixel survolé à un composant. Cette utilisation d'un espace de dessin alternatif nous permet de produire une grande quantité de composants interactifs tout en gardant une complexité de calcul linéaire pour détecter les interactions utilisateurs avec eux. En revanche, il est nécessaire de désactiver toute technique d'anticrénelage lors du dessin sur $\mathbf{C}_{alt}$ car il en résulterait des couleurs différentes pour un même composant. Malgré sa

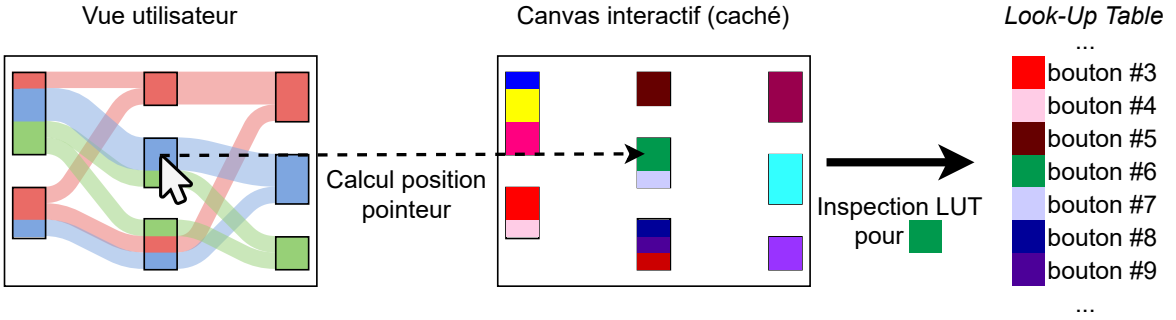
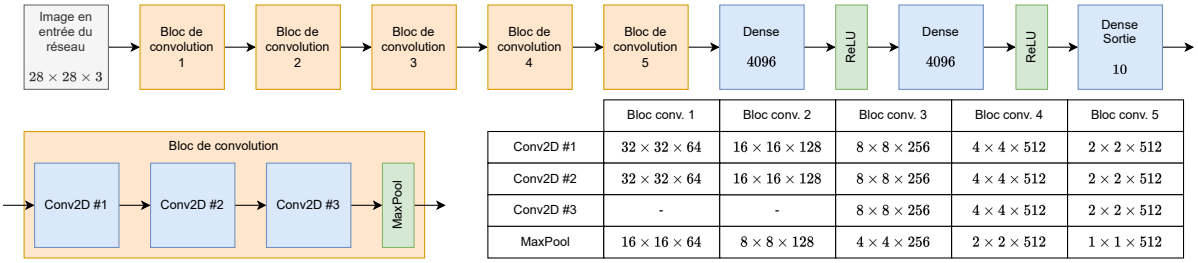


FIGURE 3.11 – Implémentation des interactions utilisant une LUT.



	Bloc conv. 1	Bloc conv. 2	Bloc conv. 3	Bloc conv. 4	Bloc conv. 5
Conv2D #1	$32 \times 32 \times 64$	$16 \times 16 \times 128$	$8 \times 8 \times 256$	$4 \times 4 \times 512$	$2 \times 2 \times 512$
Conv2D #2	$32 \times 32 \times 64$	$16 \times 16 \times 128$	$8 \times 8 \times 256$	$4 \times 4 \times 512$	$2 \times 2 \times 512$
Conv2D #3	-	-	$8 \times 8 \times 256$	$4 \times 4 \times 512$	$2 \times 2 \times 512$
MaxPool	$16 \times 16 \times 64$	$8 \times 8 \times 128$	$4 \times 4 \times 256$	$2 \times 2 \times 512$	$1 \times 1 \times 512$

FIGURE 3.12 – Architecture du modèle VGG16 adapté et utilisé dans notre scénario 4. Par comparaison, le traitement d'un élément par le réseau génère 1.3 Mo de données. La classification complète du jeu de données génère donc 13Go de données à traiter.

vitesse de traitement, cette méthode présente cependant un désavantage qui est l'impossibilité de gérer des interactions avec des composants qui se superposent, qui dans notre visualisation, correspond aux flux qui se croisent entre les couches. L'interprétation de telles interactions seraient ambigües (sur quel flux l'utilisateur a voulu cliquer ?), nous avons donc choisi de ne pas attribuer d'interactions sur les flux dessinés.

3.5 Cas d'étude

Nous avons appliqué cette méthode de visualisation sur plusieurs problèmes de classification. Les visualisations présentées dans cette sections sont disponibles sur le site internet du projet : `https://pivert.labri.fr/flows/index.html`. Ces problèmes impliquent deux modèles :

— Un premier modèle inspiré par LeNet5 [72], qui est très populaire pour les tâches de reconnaissances simples, est présenté en Figure 3.7. Ce modèle a été conçu pour reconnaître des chiffres manuscrits avec un bon taux de réussite de ∼ 98% de prédictions correctes. L'architecture du modèle est relativement simple comparée aux réseaux utilisés en milieu industriel, mais il sert à illustrer notre méthode.

— Un second modèle basé sur VGG16 pour traiter des images de taille 32 × 32 pixels, présenté en Figure 3.12. VGG16 a été conçu pour la classification du jeu de données ImageNet, composés de 14 millions d'images réparties dans des milliers de classes différentes. Il s'agit d'un modèle bien plus complexe que le premier, et ses performances sur ImageNet lui permettent de s'adapter à différentes tâches de classification en milieu industriel. Ce modèle est utilisé pour montrer la scalabilité de la méthode sur des réseaux plus grands et complexes. Il est notamment composé d'une partie convolutive, et d'une partie de couches denses. Dans notre scénario, nous gardons les paramètres des couches convolutives optimisés sur ImageNet, et nous faisons apprendre les couches denses pour adapter le modèle à MNIST.

Nous étudions l'efficacité de la méthode de visualisation sur 4 scénarios :

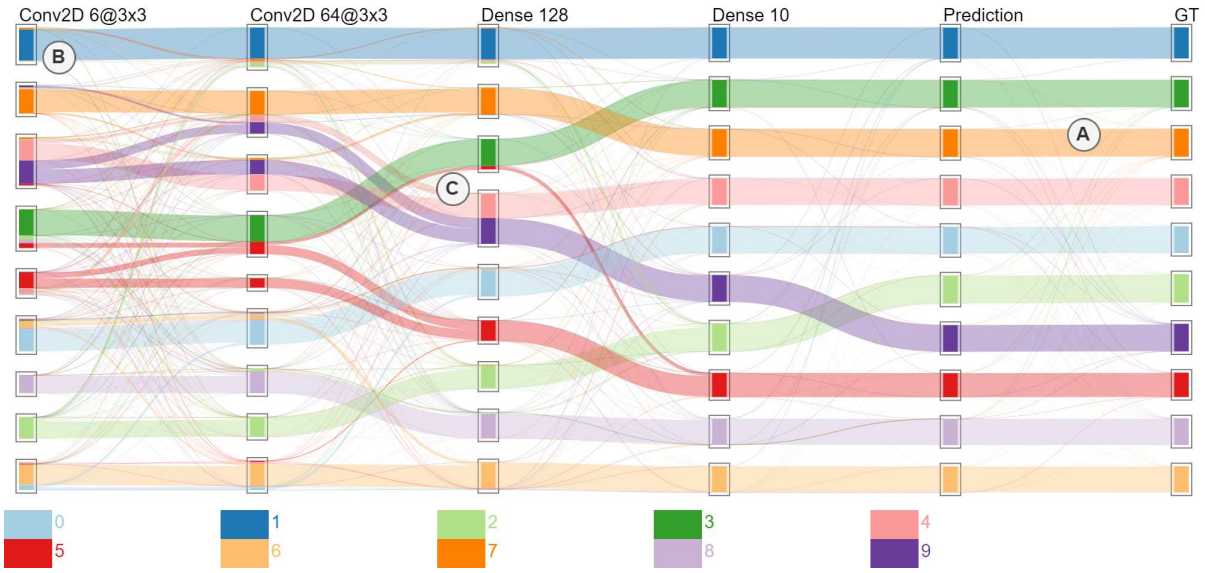


FIGURE 3.13 – Visualisation de la classification du jeu de données MNIST par LeNet5 avec 98, 5% des prédictions correctes. Les bonnes performances du réseau sont visibles par le faible nombre de divergences entre les deux derniers axes en (A), certaines classes sont reconnues similaires dès la première couche du réseau en (B) tandis que certaines sont discriminées que tardivement en (C).

— Scénario 1 : LeNet5 entraîné et évalué sur MNIST [72]. Ce scénario sert essentiellement à montrer comment la méthode permet d'aider à comprendre le fonctionnement interne d'un réseau entraîné, ainsi que de détecter quelles classes lui sont plus faciles à traiter que d'autres. Le modèle a été entraîné sur un jeu de données de 60 000 images et évalué sur 10 000 autres, et atteint un taux de 98, 5% de classifications correctes.
— Scénario 2 : LeNet5 entraîné et évalué sur Fashion-MNIST [72]. Ce jeu de données est une alternative à MNIST, qui au lieu de proposer des chiffres manuscrits, est composé de vêtements à classifier selon leur type. Ces données sont un peu plus difficiles à classer pour LeNet5, où il n'obtient que 87, 9% de précision. Ce scénario sert d'exemple de cas concret, où un réseau arrive à classifier la majorité d'un jeu de données, mais rencontre encore des difficultés sur une partie non négligeable ce celui-ci. Nous nous attendons ici à ce que la méthode de visualisation offre des indications sur quoi et où ces difficultés ont été rencontrées par le réseau.
— Scénario 3 : LeNet5 entraîné avec MNIST mais évalué sur Fashion-MNIST. Ce scénario sert de représentation d'un cas où le réseau a mal appris durant la phase d'entraînement, et que les faibles performances qui en découlent, ici 5, 6% de précision, sont dues à la phase d'entraînement et non à l'architecture du réseau.
— Scénario 4 : VGG16 pré-entraîné avec ImageNet, spécialisé et évalué sur MNIST. À la base conçu pour des tâches de classification plus difficiles, nous employons ici VGG16 pour traiter un cas de classification où le réseau est bien plus complexe que ce qu'il ne devrait l'être pour résoudre une tâche simple. Ici, le modèle atteint une précision de 97%, qui est donc plus basse que celle de LeNet5 malgré l'entraînement plus étendu (paramétrage d'ImageNet et de MNIST) et l'architecture plus complexe du réseau. Nous nous attendons donc ici à ce que la méthode de visualisation donne des indications sur une des portions trop complexes du modèle qui le font régresser dans la classification de ce jeu de données.

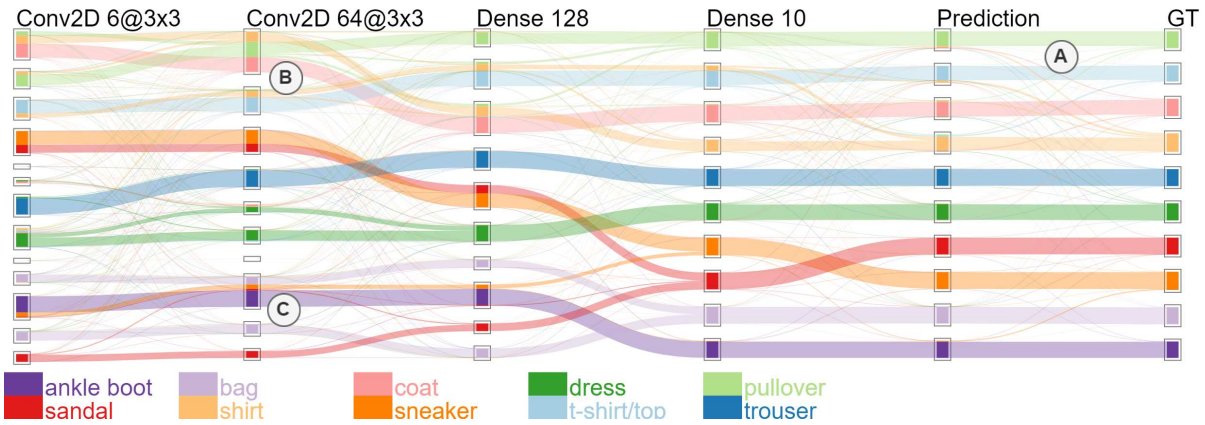


FIGURE 3.14 – Visualisation de la classification de Fashion-MNIST par LeNet5 avec 87,5% de précision. Contrairement au premier scénario, il y a bien plus de fluctuation en (A) entre les axes `Prediction` et `GT`, indiquant des performances globalement plus faibles. La séparation des groupes est bien moins uniforme que dans le scénario précédent, avec les couches de convolution qui n'arrivent pas à séparer certaines classes en (B) et en (C).

3.5.1 Scénario 1 : LeNet5 entraîné et évalué avec MNIST

Le résultat de notre méthode de visualisation est montré en Figure 3.13. On remarque quelques flux divergents entre les axes `Prediction` et `GT` ((A)), ils représentent le faible taux d'erreur du réseau sur ce jeu de données. La classification progressive est facilement repérable à travers les couches du réseau : certaines classes d'éléments visuellement similaires sont détectées dès les premières couches du réseau, par exemple les `1` ■ ((B)). On remarque aussi que la majorité des classes d'éléments sont distinguées les uns des autres à partir de la première couche dense du réseau, à part quelques `5` ■ mélangés dans des `3` ■, et quelques `9` ■ dans des `4` ■ ((C)). En regardant la couche précédente, on remarque que celle-ci n'a pas suffisamment séparé les classes concernées pour être aussi reconnues que les autres par la couche `Dense128`.

3.5.2 Scénario 2 : LeNet5 entraîné et évalué avec Fashion-MNIST

Le résultat de la méthode de visualisation est montré en Figure 3.14. On remarque dans un premier temps que les flux sont bien plus mélangés entre `Prediction` et `GT` ((A)), notamment entre les classes `shirt` ■ et `t-shirt/top` ■, que dans le premier scénario (en Figure 3.13). On remarque aussi que les couches de convolution ont bien plus de difficultés à séparer une grande partie des éléments à elles seules ((B) et (C)). La plus grande partie de la classification semble avoir lieu sur les couches denses du réseau plutôt que celles de convolutions. On peut supposer donc que le réseau pourrait être plus performant en améliorant les couches de convolution et/ou en élargissant la partie dense du réseau.

3.5.3 Scénario 3 : LeNet5 entraîné avec MNIST mais évalué avec Fashion-MNIST

Le résultat de notre méthode de visualisation est montré en Figure 3.15. On remarque directement les faibles performances du réseau, avec une multitude de groupes calculés qui n'ont qu'un faible effectif, et de nombreux croisements de flux au fil des couches. Ce comportement n'est pas représentatif du jeu de données d'entraînement pourtant bien équilibré sur l'effectif des différentes classes. On peut conclure que les couches n'ont pas retenu d'information utile à la séparation d'éléments. On peut remarquer tout de même que la classe `trouser` ■ a été plutôt bien reconnue en dépit des performances globales du réseau ((B)). On peut associer cet aspect particulier à deux observations : la première est que la classe `trouser` ■ et le `1` manuscrit ont tous les deux une forme globalement verticale, et donc les propriétés détectées par les couches

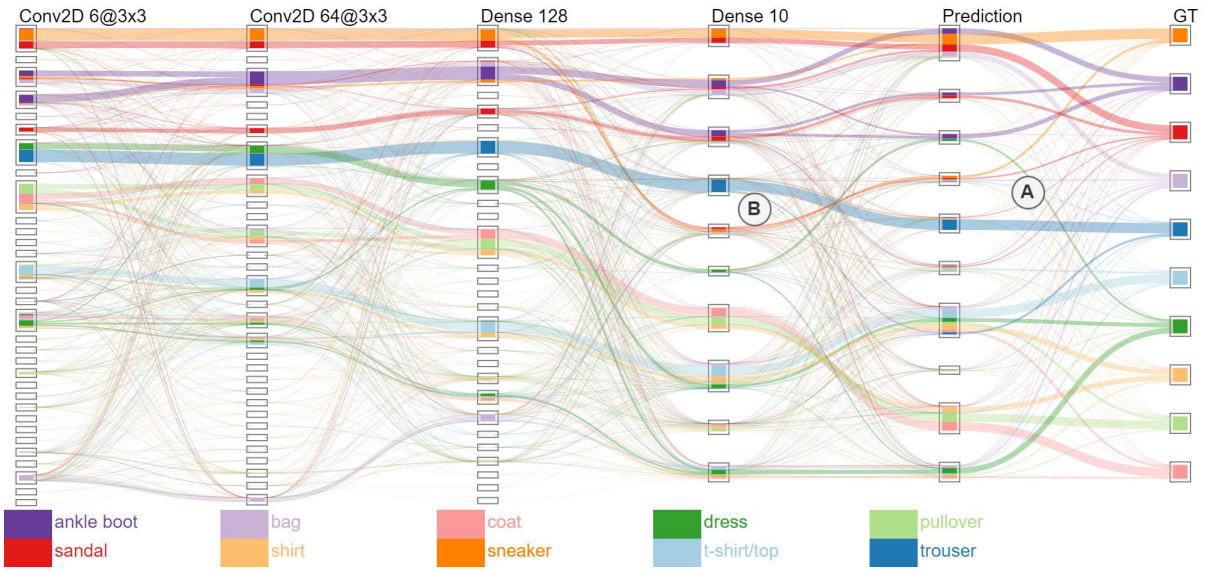


FIGURE 3.15 – Visualisation de la classification de Fashion-MNIST avec LeNet5 ayant appris avec MNIST avec une faible précision de 5,6%. Celle-ci est bien représentée en (A) avec des groupes loin d'être uniformes. Sur les dernières couches du réseau, seule la classe `trouser` ■ *semble* avoir été comprise en (B).

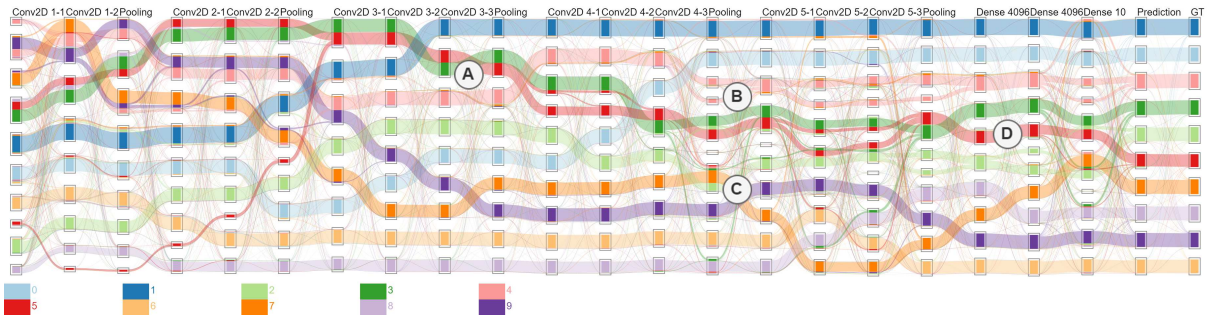


FIGURE 3.16 – Visualisation de la classification de MNIST avec VGG16 pré-entrainé. La classification semble être pratiquement faite dès le 3e bloc convolutif en (A), mais les convolutions suivantes rend les séparations plus confuses, notamment en (B) et en (C). Les couches denses en (D) semblent néanmoins corriger en partie cette confusion.

convolutives ont une certaine similarité entre les deux éléments même s'ils sont dans des jeux de données différents. La seconde observation est que le `1` manuscrit et `trouser` sont tous les deux la «deuxième» classe de leurs jeux de données respectifs, et donc qu'avec la première observation, le modèle a reconnu dans les éléments `trouser` des caractéristiques du `1` et les a donc classés comme tel en maximisant la valeur de la deuxième composante de son vecteur de sortie.

3.5.4 Scénario 4 : VGG16 avec paramètres ImageNet, entraîné et évalué avec MNIST

Le résultat de la méthode de visualisation est montré en Figure 3.16. Celle-ci est sensiblement plus large que les visualisations vues jusqu'à présent à cause de la complexité du réseau qui est bien plus grande, avec 21 couches contre 5 dans le modèle précédent. Dans ce scénario, nous observons une sorte de stagnation de la classification, c'est-à-dire que les couches ne semblent pas améliorer la discrimination des classes. En effet, à partir de la couche `Conv2D 3-3`, mis à part la séparation des classes `3` ■ et `5` ■ ((A)), la plupart des éléments sont déjà bien distingués les uns des autres. Passé cette couche dans le diagramme, les groupes d'éléments semblent moins bien définis, il y

a même une «dégradation» de la classification vers la couche de `Pooling` suivant `Conv2D 4-3` car des éléments d'une même classe, auparavant groupés entre eux, sont séparés (notamment les `4` en (B)) ou bien mélangés avec les éléments d'une autre classe (des `2` avec des `7` en (C)). Cependant, ces erreurs ont l'air pour la plupart d'être corrigées par les couches denses ((D)), mais celles-ci arrivent bien après la dégradation observée. On peut donc en conclure que modifier l'architecture du réseau entre la couche où a eu lieu la dégradation et les couches Denses pourrait mener à une meilleure performance du réseau, tout en réduisant sa complexité.

3.6 Discussion

Nous avons réussi à représenter le comportement progressif de la classification sur des modèles simples et complexes : toutes les données intermédiaires, quel que soit leur forme dans $\mathbb{R}^N$, sont représentées de la même façon sur l'outil de visualisation et peuvent être interprétées comme une représentation de la donnée d'entrée à travers le réseau. Pour comparer la nature des données entre elles, nous exploitons une métrique de similarité dans l'espace $\mathbb{R}^N$, qui permet de déterminer des ensembles de données traitées (et donc reconnues) similairement par le réseau. Le calcul de ces ensembles est appliqué de manière non supervisée et peut donc s'adapter à tous les scénarios sans modification du processus. En plus de cela, des méthodes de post-traitement permettant de mieux exploiter les cartes d'activation, ici LRP, peuvent aussi être employées dans le but d'améliorer l'interprétation des résultats affichés.

Dans notre étude de cas, des informations importantes ont pu être relevées grâce à cette méthode de visualisation. Des classes sont plus reconnaissables par le réseau et demandent peu d'opérations (ou de couches) afin d'être discriminées et classées.

Au contraire, d'autres classes ne sont détectées que tardivement dans le réseau et indiquent l'importance de la détection des attributs de haut niveau par rapport à ceux de bas niveau. La contribution, positive ou négative, de chaque couche est facilement remarquable en observant les convergences et divergences des flux entre chaque couche.

Si ce prototype d'outil d'explicabilité des réseaux de neurones remplit la plupart de nos objectifs initiaux, certains points restent à améliorer :

— Le calcul de dissimilarité entre chaque paire de cartes d'activation revient à calculer $|D|^2$ valeurs, ce qui est couteux en termes de ressources. De plus, une très grande partie de ces valeurs n'est pas utile à l'algorithme de clustering (ici *mcl*) puisqu'un filtre *k-NN* est appliqué en amont afin de réduire la densité du graphe de similarité. Enfin, ce filtre *k-NN* demande un argument k, correspondant au nombre de voisins à conserver pour chaque sommet du graphe, qui peut varier suivant le scénario étudié. Dans notre étude, la valeur $k = 10$ a donné des résultats satisfaisants en termes de construction de groupes, mais d'autres cas d'usage pourraient demander plusieurs tentatives avant d'avoir une valeur k donnant des résultats satisfaisants.

— Un autre point à améliorer concerne la méthode de visualisation utilisée. Un nombre important de groupes calculés (comme dans le troisième scénario) ou un jeu de données très grand peuvent rendre la lisibilité des groupes plus difficile, car ceux-ci seront écrasés verticalement sur les axes. Malgré la nature synthétique de notre visualisation, il se peut que cette vue ne soit pas adaptée aux scénarios les plus extrêmes. Il nous faut donc travailler sur une utilisation plus efficace de l'espace de dessin pour fournir des informations complémentaires sur la classification, comme permettre le suivi d'un élément individuellement plutôt que nous limiter à des sous-groupes.

— La technique actuelle ne permet que de traiter les réseaux aux architectures linéaires, alors que certaines architectures, comme celle de ResNet, contiennent des connections en branches qui *peuvent* être représentées par un diagramme de Sankey, mais qui impliquent de superposer des flux d'origine ou de destination identiques.

Ensuite, il faut préciser que la méthode apporte une vue d'ensemble aidant à l'explicabilité des prises de décisions d'un réseau de neurones, mais ne remplace pas les informations apportées par les méthodes d'explicabilité traitant au cas par cas qui apportent en conséquent une granularité d'information plus petite. Certaines méthodes d'explicabilité calculant des partitions de données similaires, comme CNNVis [75], utilisent des représentants parmi les éléments des groupes formés. Ces représentants sont indiqués sous leur forme dans le jeu de données d'évaluation (des images par exemple) et servent à induire l'utilisateur sur les attributs que les éléments ont en commun dans ces partitions calculées. Cette approche se heurte à l'une des limites évoquées en début de chapitre, à savoir la difficulté à s'adapter à des scénarios impliquant des données de nature différente. De plus, passées les couches initiales d'un réseau, l'utilisation de la donnée d'entrée comme représentant d'informations dans un espace hautement dimensionnel n'est plus aussi intuitive pour l'utilisateur à cause des nombreuses transformations qu'a subi la donnée lors de sa classification. En effet, à ce stade de la classification, le réseau ne travaille plus avec des attributs bas niveau proches de la donnée d'entrée, mais plutôt avec des attributs de haut niveau, proches de la compréhension globale. L'exploration et la compréhension des données dans $\mathbb{R}^N$ au-delà de la mesure de similarité reste donc un point à approfondir.

Pendant le développement de ce prototype, InstanceFlow [111], un autre outil a été présenté à la communauté pour mesurer les performances globales d'un réseau via l'étude de la phase d'apprentissage du réseau. Cette méthode ne se limite pas à une nature de données en particulier (comme une image) et fait également usage d'un diagramme de Sankey pour représenter la classification d'une manière temporelle. En revanche, au lieu de s'appuyer sur l'architecture du réseau comme référence topologique de la visualisation, les auteurs ont décidé d'utiliser les époques lors de l'apprentissage du réseau pour indiquer l'amélioration du réseau au cours de son apprentissage. InstanceFlow utilise les classes et prédictions comme critères de groupement de données, différents des cartes activations utilisées dans notre méthode. Ces travaux parallèles montrent qu'une représentation basée sur un élément temporel et que comparer le traitement d'un élément par rapport aux autres sans tenir compte de sa nature, donne une intuition à l'utilisateur sur ce que comprend le réseau lors de son entraînement. L'aspect synthétique de la visualisation est aussi mis en valeur puisqu'elle est complétée par d'autres formes de visualisations pour communiquer des informations complémentaires sur le scénario étudié.

3.7 Conclusion

Dans ce chapitre, nous avons présenté un outil de visualisation afin d'expliquer le processus de classification d'un réseau de neurones. Il se démarque d'autres méthodes d'explicabilité parce qu'il s'appuie sur le traitement d'un jeu de données d'évaluation complet plutôt que de traiter les classifications au cas par cas. De plus, contrairement à beaucoup d'outils de visualisation pour l'explicabilité, cet outil ne se limite pas à un type de données en particulier (comme des images) et traite les informations comme des données dans $\mathbb{R}^N$ à tous les niveaux de l'étude du réseau. Le traitement et l'encodage des données d'entrée et des cartes d'activation est identique, et peut donc s'adapter à de nombreux scénario de classification. Si nécessaire, nous pouvons intégrer une autre méthode d'explicabilité (dans ce chapitre, LRP) dans le traitement des données pour rendre la notion de similarité plus précise. Nous avons montré quatre scénarios d'application pour cet outil, et y avons relevé la classification progressive et la contribution de chaque couche des réseaux. Notre méthode s'applique sur les tâches de classification complexes, mais à mesure que les nombres de couches et de groupes calculés augmentent, la lisibilité de l'information diminue. Nous retenons que la représentation temporelle de la classification des données (ici par les couches du réseau) révèle des informations supplémentaires sur le fonctionnement d'un réseau entraîné, mais cet outil doit être re-étudié afin de supporter des cas d'applications plus complexes impliquant des architectures de réseaux non-linéaires, un nombre plus élevé d'éléments à classifier, et permettre une étude plus précise des sous-groupes observés. Ces aspects sont étudiés dans le prochain chapitre,

où nous présentons une évolution de notre approche en employant une méthode de représentation de l'information plus compacte permettant de s'adapter à des architectures de réseaux plus complexes ainsi qu'à l'affichage de jeux de données plus grands.

Chapitre 4

Visualisation compacte par courbes fractales

Dans ce chapitre, nous cherchons à améliorer les travaux présentés dans le chapitre précédent en essayant de résoudre le problème de passage à l'échelle relevé lors de l'application de notre méthode. Son application sur des modèles aux architectures de réseaux plus complexes ou l'étude de jeux de données plus grands peut rendre la lecture de l'outil de visualisation plus difficile et donc moins efficace. Les problèmes de passage à l'échelle relevés concernent notamment :

— la lisibilité de l'information en cas de détection de nombreux groupes de données similaires,
— l'application à des architectures «avec des branches» plus fréquemment utilisées dans les scénarios plus complexes,
— le paramétrage de la méthode de calcul de groupes similaires qui peut demander plusieurs essais avant d'obtenir des résultats satisfaisants.

Pour cela, dans ce chapitre, nous utilisons des méthodes de visualisation dites «orientées pixel», c'est-à-dire qui associe à chaque valeur des données visualisées un pixel sur l'écran. L'objectif de ces méthodes est de maximiser le rapport information affichée par espace de dessin utilisé. L'outil présenté dans ce chapitre fait usage de telles méthodes au moyen de courbes fractales. La courbe fractale utilisée ici permet de projeter des données sur une grille de pixels. Ces grilles nous servent ensuite à afficher de grands jeux de données où chaque élément y sera représenté par un pixel, maximisant donc la quantité d'informations que l'on peut afficher à l'écran. Par rapport au chapitre précédent, nous représentons ici la totalité des traitements d'une couche du réseau par une grille plutôt qu'un axe vertical qui nous empêche de pouvoir représenter les architectures de réseaux «avec des branches».

Ces travaux sont présentés en quatre parties. En premier lieu, nous rappelons les problématiques soulevées dans le chapitre précédent et ce qu'apporte la visualisation orientée pixel dans notre cas. Ensuite, nous détaillons la façon dont nous transformons les données collectée lors de la classification du réseau vers une grille de pixels. La troisième partie présente l'implémentation de notre méthode. Enfin, nous clôturons le chapitre sur des exemples d'applications de la méthode et discutons des différences avec la visualisation par flux du chapitre précédent, et des points à améliorer pour une future itération.

Les contributions dans ce chapitre ont mené à :

— Une publication en workshop : **Halnaut, A.**, Giot, R., Bourqui, R., & Auber, D. (2021, Janvier). «Samples Classification Analysis Across DNN Layers with Fractal Curves». Dans *ICPR 2020's Workshop Explainable Deep Learning for AI*.

— Une version étendue en chapitre de livre : **Halnaut, A.**, Giot, R., Bourqui, R., & Auber, D. (2023, Janvier). «Compact visualization of DNN classification performances for interpretation and improvement». Dans *Explainable Deep Learning AI (pp. 35-54). Academic Press.*.

— Un prototype de la méthode en démonstration à l'adresse https://pivert.labri.fr/frac/index.html et dont le code est disponible à l'adresse https://github.com/eikofee/viz-fractal-curve.

4.1 Contexte

L'outil de visualisation par flux présenté au chapitre 3 est capable de représenter la classification progressive d'un réseau au niveau de chaque couche, mais demande un espace de dessin à l'écran considérable lors du traitement de réseaux complexes. Par exemple, le scénario VGG16/MNIST implique l'affichage d'une vingtaine d'axes verticaux disposés horizontalement.

Lorsque la méthode est appliquée sur des scénarios où le réseau a des difficultés à classer un jeu de données, tels que le scénario où le réseau LeNet entraîné sur MNIST est évalué sur Fashion-MNIST, les nombreux groupes d'éléments similaires calculés sont affiché le long des axes verticaux mais de manière écrasée, rendant leur lecture plus compliquée pour l'utilisateur. La Figure 4.1 montre un exemple de soucis de lisibilité lié à une quantité importante d'informations affichée dans un environnement trop restreint. La zone de dessin de la visualisation mesure 800 pixels de haut. Sur cet exemple, le groupe (a) est composé de 392 éléments (~4% des données totales) qui semblent être essentiellement de la classe rouge. En réalité, celui-ci est bien composé à 90% d'éléments de la classe rouge, et les 10% restants de 7 autres classes. Celles-ci ne sont pratiquement pas visibles, car ces informations sont affichées sur 9 pixels seulement. Le groupe (b) est lui composé de 1545 éléments (~15% des données totales) qui semblent être répartis entre les classes bleu foncée et verte. Ces deux classes forment en fait 91% de la composition du groupe, les 9% restants rassemblent entre 1 et 38 éléments de toutes les classes restantes. Ces informations sont affichées dans un espace de 37 pixels de haut. L'espace d'affichage est donc fortement sous-exploité par rapport à la quantité d'informations que l'on souhaite afficher. Cette limitation sur l'affichage des groupes calculés peut s'étendre aux scénarios impliquant la détection de nom-

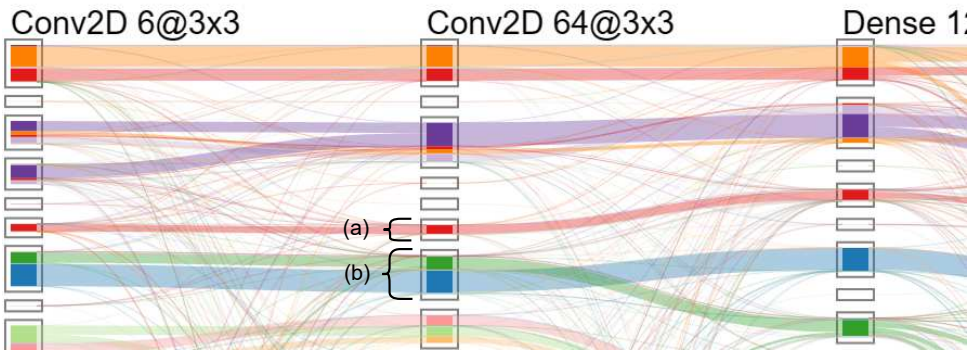


FIGURE 4.1 – Portion de la visualisation obtenue lors de l'application de la méthode du Chapitre 3 sur le modèle entraîné sur MNIST mais évalué sur Fashion-MNIST. Le nombre important de groupes positionnés le long d'un axe réduit l'espace disponible pour afficher leur composition.

breuses classes différentes, comme par exemple le jeu de données ImageNet [24] qui regroupe un millier de classes d'images.

Le *data-ink ratio* [134] est une mesure de l'efficacité d'une méthode de visualisation sur son utilisation de l'espace de dessin. Il s'agit du rapport entre la taille de l'espace de dessin utilisé (souvent mesuré en pixels) et la quantité d'information affichée. Lorsque la quantité d'informations est supérieure à la taille de l'espace de dessin utilisé, il y a une occlusion d'informations (comme montré en Figure 4.1). Lorsque la quantité d'informations est inférieure à la taille de l'espace de dessin utilisé, il y a alors de l'espace de dessin sous-exploité. Afin d'obtenir un rapport optimal, les méthodes de visualisation orientées pixel associent une information numérique à un pixel de l'écran d'affichage, et évitent de laisser de l'espace non-utilisé.

4.2 Choix de visualisation

Nous souhaitons représenter la classification progressive du réseau en ayant une utilisation de l'espace de dessin plus efficace que celle de la méthode du Chapitre 3. Nous conservons notre approche consistant à collecter les cartes d'activations de chaque couche pour chaque classification du réseau, et à étudier la similarité entre ces cartes d'activations pour trouver des groupements de données.

Cette méthode de visualisation doit aussi pouvoir s'adapter aux architectures de réseaux non-linéaires. Nous faisons le choix d'utiliser des méthodes de visualisation orientées pixel pour représenter l'état de la classification en sortie de chaque couche du réseau. Chaque couche du réseau est représentée par une grille de pixels, où chaque pixel représente un exemple de classification. La proximité de ces pixels avec les autres donne une indication sur leur similarité dans l'espace $\mathbb{R}^N$ des cartes d'activations. De cette façon, si des pixels d'une même classe se retrouvent proches les uns des autres dans la grille de pixels, alors nous pouvons considérer que la couche correspondante du réseau a reconnu l'appartenance de ces éléments à une même classe. Ces grilles de pixels sont ensuite agencées sur l'espace de dessin de manière à représenter l'architecture du réseau.

L'utilisation de ces grilles de pixels nous permet notamment de nous affranchir de l'utilisation d'axes verticaux pour représenter les couches dans la méthode de visualisation proposée dans le Chapitre 3, ce qui était la principale limitation pour l'utilisation de la méthode sur des architectures de réseaux non-linéaires. Dans ce chapitre, nous cherchons une méthode pour construire ces grilles de pixels qui ne demande pas de paramétrage particulier pour permettre son application dans n'importe quel espace dimensionnel. L'utilisation de grilles de pixels implique aussi une augmentation de la densité d'informations à l'écran. Il nous faut donc prévoir des moyens pour faciliter leur analyse, notamment par le biais d'interactions utilisateurs.

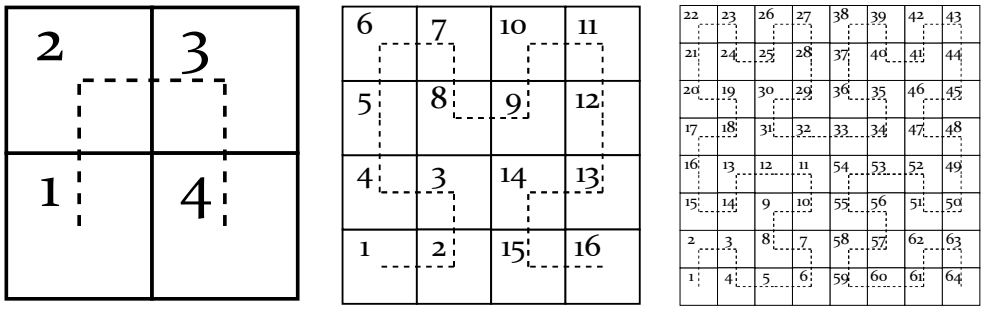
4.3 Travaux existants sur la visualisation compacte

L'utilisation de méthodes de visualisations compactes est motivée par le fait que ces techniques sont efficaces pour donner un aperçu d'un ensemble d'informations de données de dimensions quelconques.

Nous nous inspirons des travaux de Keim et al. [64, 65] pour concevoir notre outil. Ils proposent notamment d'utiliser les courbes fractales pour ordonner des éléments par ordre de proximité, afin de les présenter sous une forme compacte et agencée de façon que les éléments proches dans leur espace d'origine le soient aussi sur l'espace de visualisation.

D'abord utilisées telles quelles puis combinées avec des heuristiques pour améliorer la préservation des distances originales, comme la disposition en spirale et le découpage de dessin, les courbes fractales sont utilisées pour visualiser des grands jeux de données à plusieurs dizaines de milliers d'éléments. On peut retrouver l'utilisation de courbes fractales en visualisation d'information dans GreenCurve [148], GosperMap [8] ou Jasper [135] pour compacter la visualisation de graphes. Il existe de nombreuses courbes fractales, mais la courbe de Hilbert [50], présentée en

Figure 4.2, est dite la plus efficace à représenter les groupes d'éléments proches dans $\mathbb{R}^N$ une fois qu'ils sont projetés dessus [89].



(a) Courbe à 4 éléments (b) Courbe à 16 éléments (c) Courbe à 64 éléments

FIGURE 4.2 – Construction de la courbe de Hilbert.

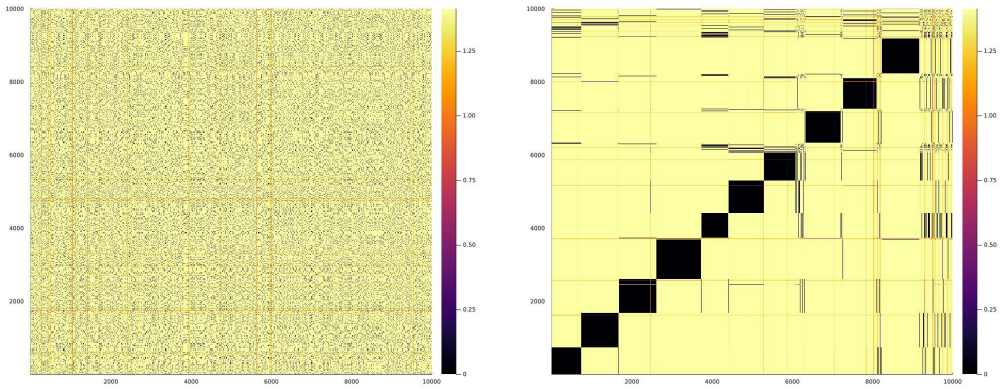
L'utilisation de méthodes de visualisations compactes est encore marginale dans le domaine de l'explication de réseaux de neurones. Parmi les travaux qui en font usage, nous pouvons citer OoD Analyzer [17] qui représente la classification d'un ensemble d'éléments par un réseau sous la forme d'une grille compacte, où chaque cellule représente la classification d'un élément du jeu de données, et où les éléments classés de manière particulière sont mis en évidence en utilisant une couleur de cellule plus prononcée que le reste de la grille.

En nous inspirant de ces travaux, nous proposons d'exploiter la courbe de Hilbert [50] pour représenter les cartes d'activation des couches d'un réseau de neurones pour former une cartographie des données dans $\mathbb{R}^N$, de manière que les données proches dans $\mathbb{R}^N$ le soient tout aussi sur la grille construite par la courbe fractale. Nous faisons en sorte que les données soient d'abord projetées en une seule dimension tout en conservant une notion de proximité entre les éléments. Ensuite, nous les disposons dans l'ordre en suivant la courbe de Hilbert.

4.4 Traitement des cartes d'activation

4.4.1 Transformation des données $\mathbb{R}^N$ vers $\mathbb{N}$

L'utilisation de cette courbe demande au préalable de projeter les données d'entrée dans $\mathbb{R}^N$ (les cartes d'activation) dans un ordre linéaire (l'ordre de la courbe), donc vers $\mathbb{N}$, où la valeur dans $\mathbb{N}$ représente la position de l'élément original sur la courbe. Cette projection rejoint le problème d'arrangement linéaire minimum [106] (*Minimum Linear Arrangement Problem*) qui consiste à trouver un ordre dans lequel parcourir les sommets d'un graphes de manière à minimiser la somme des poids des arêtes reliant les sommets adjacents. Il s'agit notamment d'un problème d'optimisation combinatoire qui est NP-difficile [37]. Pour calculer cet ordre, nous transformons les données dans $\mathbb{R}^N$ en une matrice de distance, basée sur la distance cosinus qui est plus adaptée que la distance Euclidienne dans les espaces à grandes dimensions. Nous réarrangeons ensuite les lignes et colonnes de cette matrice pour former un ordre linéaire dans lequel les colonnes ou lignes consécutives représentent deux éléments similaires dans $\mathbb{R}^N$, cette similarité pouvant être identifiée par une faible distance entre ces deux éléments dans la matrice de distance. Pour calculer cet ordre, nous faisons usage de l'algorithme VAT [10] (*Visual Assessment of clustering Tendency*). Cet algorithme est une méthode pour détecter visuellement les partitions dans un jeu de données à partir d'une matrice de dissimilarité. Il réarrange les lignes et colonnes d'une matrice de distance calculée sur le jeu de données de manière à pouvoir détecter les partitions des données lorsque la matrice est visualisée sous la forme d'une carte de chaleur. La Figure 4.3 montre exemple d'application de l'algorithme avec une matrice de distance de taille $10\,000 \times 10\,000$. La nouvelle position de chaque ligne et colonne dans la matrice réarrangée par VAT nous donne l'ordre linéaire dans lequel disposer les éléments du jeu de données dans $\mathbb{R}^N$.



(a) matrice de distance dans $\mathbb{R}^N$ (b) Matrice réordonnée avec VAT.

FIGURE 4.3 – Exemple d'utilisation de l'algorithme VAT sur une matrice de distance 10 000 × 10 000. VAT forme visuellement des partitions représentant des données proches dans $\mathbb{R}^N$ et permet d'ordonner les 10 000 éléments suivant leur proximité relative.

Un tel processus est cependant coûteux ; le calcul d'une matrice de distances est de complexité de temps $O(N^2)$ et l'est tout autant en occupation d'espace mémoire. L'exécution de l'algorithme VAT est aussi de complexité $O(N^2)$. L'approche distribuée expliquée dans le Chapitre 3 nous est utile ici pour calculer la matrice de distance. Ce calcul distribué est efficace car il ne demande pas de redistribution des informations (*shuffle*) à travers les nœuds de l'infrastructure, cette opération étant coûteuse en temps si celle-ci est trop souvent demandée. Cependant, une version distribuée de l'algorithme VAT est moins pertinente pour notre cas d'usage. En effet, l'algorithme VAT demande de calculer successivement N fois un minimum dans la matrice, en excluant progressivement des cellules de la matrice à chaque fois qu'un minimum est trouvé. Afin d'adapter ce comportement de manière distribuée, nous devons mettre à jour les valeurs à exclure sur chacun des nœuds de calcul de l'infrastructure, ce qui demande une redistribution à chaque mise à jour de la matrice. Cette implémentation résulte en un nombre trop important de redistributions au point où le temps d'exécution de l'algorithme est majoritairement dû aux opérations de redistributions plutôt qu'aux opérations sur la matrice. L'approche distribuée n'est donc pas une bonne solution pour compenser la complexité de l'algorithme VAT.

Pour résoudre le problème de complexité de VAT, nous avons adopté une approche parallèle de l'algorithme en faisant usage des processeurs graphiques d'une machine de calculs. Ce matériel offre la possibilité de traiter les opérations matricielles de manière bien plus efficace que sur processeurs classiques montés en infrastructure parallèle. En utilisant *ArrayFire* [152], une bibliothèque qui exploite les outils CUDA [96] et OpenCL [90] permettant de faire usage des processeurs graphiques pour réaliser des calculs de manière plus performante, nous avons conçu une implémentation de VAT sur processeur graphique tournant sur bien plus d'unités de calculs et avec des temps de transferts fortement réduits par rapport à une version distribuée. Cette implémentation est présentée en Algorithme 1, mais demande en revanche une importante quantité de mémoire pour le stockage des matrices de distance (de taille N^2 avec N la taille du jeu de données étudié). Contrairement à notre approche distribuée, la mémoire ici ne peut pas être étendue aussi facilement, et nous devons pour l'instant utiliser l'implémentation distribuée pour traiter les jeux de données plus grands. Lors de la réalisation de ces travaux, le calcul de matrices de distance pour 10 000 éléments a pu être réalisé sur un processeur graphique nVidia Quadro T1000 équipé de 4Go de mémoire vidéo.

Une synthèse des étapes de calculs nécessaires est présentée en Figure 4.4.

Algorithme 1 : Algorithme VAT [10] adapté pour processeur graphique. L'utilisation de matrices comme des masques booléens (suivant si les cellules valent 0 ou 1) peut être parallélisée sur ces processeurs et d'obtenir des temps de calcul bien amoindris par rapport à une implémentation classique ou distribuée.

Entrée : matrice de distance D de taille N
Résultat : Position des indices de D pour obtenir la matrice ordonnée
$I \leftarrow \{0, 0, ..., 0\}_N$;
$P \leftarrow I$;
$J \leftarrow \{1, 1, ..., 1\}_N$;
$maxValCol \leftarrow$ maxForEachLine(D) ; ▶ $maxValCol$ est une colonne unique
$maxLine \leftarrow$ argmaxForEachColumn($maxValCol$) ;
$D \leftarrow D + \mathbb{1}_N \cdot 2$; ▶ on ignore les valeurs diagonales pour la recherche du minimum
$I_{maxLine} \leftarrow 1$;
$J_{maxLine} \leftarrow 0$;
$P_1 \leftarrow maxLine$;
pour $i \in \{2 \dots N\}$ **faire**
  $subD \leftarrow D[I == 1, J == 1]$; ▶ filtre en utilisant I et J
  $orig \leftarrow \{1 \dots N\}[J == 1]$; ▶ indices originaux des lignes filtrées
  $minValCol \leftarrow$ minForEachLine($subD$) ;
  $locMinLine \leftarrow$ argminForEachColumn($minValCol$) ;
  $minLine \leftarrow orig[locMinLine]$; ▶ conversion indice à $subD$ en indice à D
  $I_{minLine} \leftarrow 1$;
  $J_{minLine} \leftarrow 0$;
  $P_i \leftarrow minLine$;

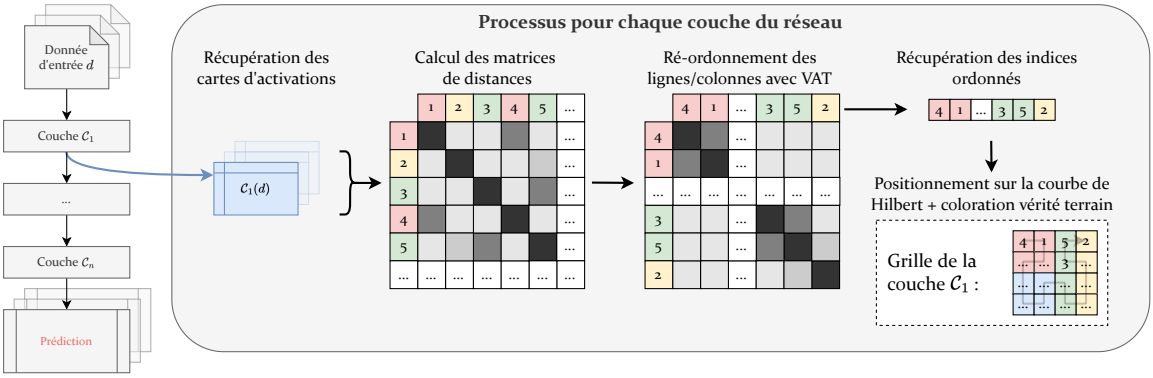
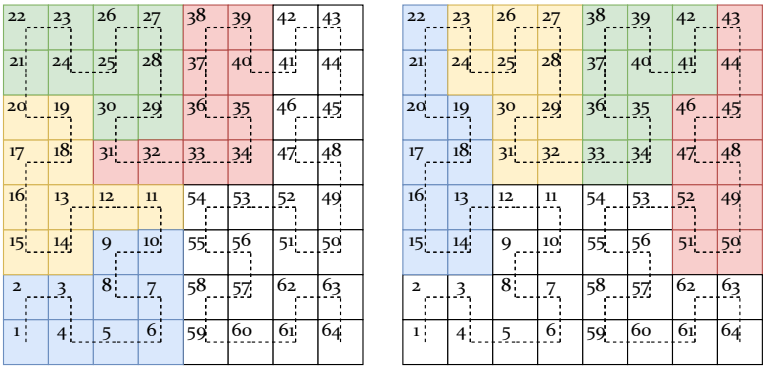


FIGURE 4.4 – Processus de construction de la grille de pixels pour chaque couche du réseau étudié.

4.4.2 Conception de l'outil de visualisation

Construction des grilles de pixels. Nous construisons les grilles de pixels en positionnant chaque élément du jeu de données sur la courbe de Hilbert en suivant la séquence d'indices calculée par VAT. Chaque pixel est coloré par la couleur associée à la classe d'équivalence de l'élément. Afin d'obtenir une image carrée, le nombre d'éléments à positionner sur la courbe fractale doit être d'une taille égale à 4^x. Puisque notre jeu de données ne contient que $4^6 < 10000 < 4^7$ éléments, une partie de la courbe fractale n'est pas utilisée. Nous utilisons donc des pixels noirs, couleur qui n'est attribuée à aucune vérité terrain, pour combler cet espace non utilisé. Aussi, afin d'utiliser l'espace de dessin de manière symétrique, nous centrons les positions des éléments vers le milieu de la courbe fractale de taille 4^7 en décalant les indices de $(N - 4^x)/2$ positions sur la courbe de Hilbert, avec N la taille du jeu de données affiché et x entier tel que $4^{x-1} < N \leq 4^x$.



(a) Projection à partir du début de la courbe

(b) Projection centrée sur la courbe

FIGURE 4.5 – Exemple de projection de 40 éléments répartis dans 4 classes différentes d'effectifs égaux sur la courbe de Hilbert. En (a), les éléments sont positionnés à partir de la première case de la courbe, en bas à gauche. En (b), les éléments sont centrés sur le milieu de la courbe (32e et 33e cases). Lorsque les éléments sont centrés, les groupes sont positionnés de manière symétrique, ce qui nous semble plus facile à interpréter que la grille obtenue en (a).

La Figure 4.5 donne un exemple de ce centrage pour un jeu de données de taille $N = 40$ sur une courbe de Hilbert de $4^{x=4} = 64$ cellules.

Interactions pour la compréhension. Nous rendons notre outil de visualisation interactif pour afficher l'état de la classification de toutes les classes au niveau de chaque couche, ou bien celle d'une classe en particulier. Pour cela, nous générons donc deux autres types de coloration des pixels montrés en Figure 4.6. La coloration montrée en (a) est la coloration habituelle des éléments par rapport à leur classe d'équivalence.

(a) Grille de pixels de base

(b) Classe filtrée (ici la verte)

(c) Coloration montrant la qualité de la classification

FIGURE 4.6 – Variation de la colorations d'une grille de pixels (a) pour filtrer les classes (b) et pour évaluer la qualité de la classification (c).

La coloration (b) permet de mettre en évidence une classe d'éléments en particulier et de masquer les autres dans la grille. Il nous suffit de produire plusieurs variations de la grille de pixels originale (a) mais en colorant en noir les pixels des classes filtrées. Lors de l'utilisation de l'outil de visualisation, il suffit de remplacer la coloration originale par une de ces variations en fonction du choix de l'utilisateur.

La dernière coloration (c) permet de montrer la qualité de la classification. Lorsqu'une couche regroupe tous les éléments d'une même classe dans $\mathbb{R}^N$, cette classification efficace est représentée par une zone de couleur unie dans la grille. A l'inverse, une couche qui ne produit pratiquement aucune différenciation entre les classes produit une grille de pixels aux couleurs très mélangées

sans structure apparente. On peut donc détecter visuellement l'efficacité d'une classification en vérifiant si les pixels contigus sont de même couleur ou non. Afin de montrer l'efficacité de la classification sur toutes les classes, nous produisons une coloration (c) où chaque pixel correspondant à un élément à la position i sur la courbe de Hilbert est coloré en noir si ses voisins aux positions $i-1$ et $i+1$ sont de la même classe, gris si l'un des deux est d'une classe différente, et blanc si les deux voisins sont tous les deux d'une classe différente de l'élément à la position i. Avec cette coloration, la quantité de pixels blancs et gris ou noirs sur la grille donne une indication sur les performances de la couche à discriminer les différentes classes du jeu de données. Vu que cette coloration fait usage de la couleur noire, nous changeons la couleur des pixels des positions non utilisées de la courbe de Hilbert (ici en bleu).

Construction de l'outil de visualisation. Le graphe encodant l'architecture du réseau étudié est dessiné en utilisant l'algorithme de Sugiyama [129]. Les grilles de pixels sont dessinés au niveau de chaque sommet et sont reliés par des lignes en pointillés animés en forme de flux pour représenter l'aspect séquentiel du processus de classification du réseau. Le nom des couches dessinées est écrit au-dessus de chaque grille pour pouvoir les identifier. La légende indiquant la couleur associée à chaque classe du jeu de données est indiquée sous le dessin final.

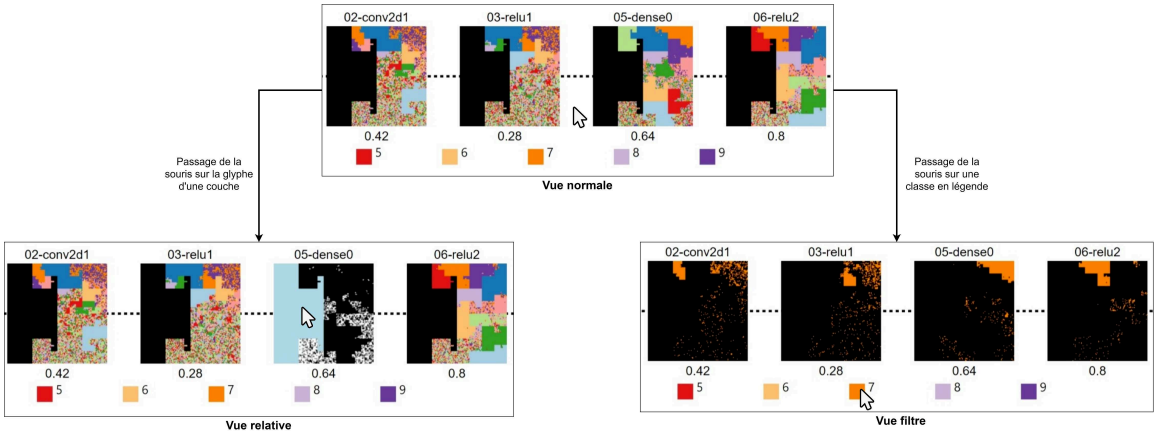


FIGURE 4.7 – Récapitulatif des interactions de l'outil de visualisation. Le passage du pointeur de la souris sur les différents éléments résulte en des vues différentes.

Nous intégrons aussi deux interactions pour alterner entre les colorations des grilles de pixels, présentées en Figure 4.7. L'utilisateur peut filtrer une classe d'éléments en particulier en faisant survoler le pointeur de la souris sur une classe dans la légende. La coloration des grilles de pixels change alors en celle mettant en évidence la classe choisie. Si l'utilisateur souhaite obtenir la coloration montrant la qualité de la classification d'une couche en particulier, il peut faire survoler le pointeur de la souris sur la grille de pixels correspondant à la couche choisie.

4.5 Résultats et comparaison avec la visualisation par flux

Dans cette section, nous montrons les résultats de cette méthode de visualisation sur les mêmes scénarios que dans le Chapitre 3, et nous comparons ces résultats avec la méthode de visualisation par flux. Nous étudions un scenario supplémentaire faisant usage d'un réseau de neurones ayant une architecture à branchements afin d'avoir un cas d'usage que la précédente visualisation ne pouvait pas supporter. Ce réseau est constitué de deux architectures de LeNet5 en parallèle, dont l'une qui alterne l'image d'entrée avec une rotation à 90° dans le sens horaire. L'architecture du modèle de LeNet a été modifiée par rapport au chapitre précédent pour mieux respecter la partie convolutive du modèle original (la deuxième couche de convolution passe de 64 filtres à 16 filtres). En revanche, nous ne changeons pas la taille des couches denses (128 neu-

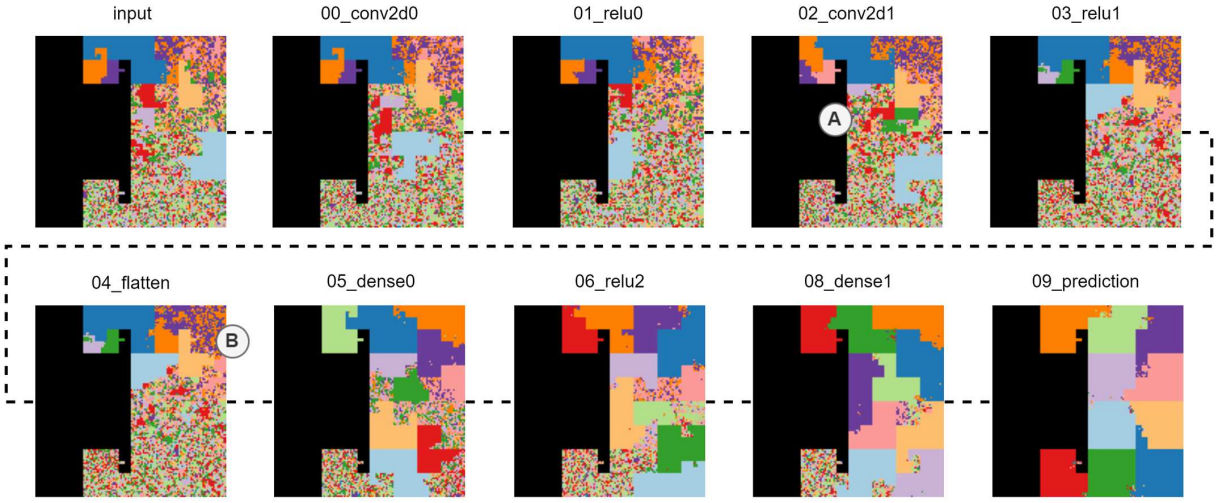


FIGURE 4.8 – Visualisation de la classification du jeu de données MNIST par LeNet5 avec 98, 5% des prédictions correctes.

rones ici par rapport à 84 sur le modèle original) car nous avons observé, grâce à la visualisation précédente, qu'ils apportent une contribution très importante dans la classification finale.

Les résultats présentés dans ce chapitre ont été édités pour pouvoir être lisibles convenablement en format portrait dans ce manuscrit. Ils sont aussi présentés sur le site internet du projet : https://pivert.labri.fr/frac/index.html.

4.5.1 Scénario 1 : LeNet5 entraîné et évalué avec MNIST

La visualisation du scénario est montrée en Figure 4.8. De manière analogue à la méthode du chapitre précédent, on remarque la formation de groupes de couleurs unies sur la grille de prédiction, mettant en valeur les 98, 5% de précision du modèle. On remarque aussi la formation de blocs de couleurs unies sur les premières couches du réseau, signifiant une discrimination efficace, notamment des classes `0` et `1`, que nous avions observé précédemment. Le «mélange» des classes `3` et `5` est aussi observable en couche `02_conv2d1` (A) par leur proximité sur la grille de pixels. La distinction entre les classes `7` et `9` ne s'est faite que bien plus tard (B) contrairement au scénario du chapitre précédent. On peut supposer que la simplification de la deuxième couche de convolution a rendu cette différenciation plus compliquée pour le modèle. On retrouve cependant la distinction tardive des classes `4` et `9` par leur mélange partiel dans les grilles de pixels.

4.5.2 Scénario 2 : LeNet5 entraîné et évalué avec Fashion-MNIST

La visualisation obtenue sur ce scénario est présentée en Figure 4.9. Le changement du nombre de filtres pour la deuxième couche de convolution cause une diminution des performances du réseau par rapport à celui du chapitre précédent. Ce changement est reflété sur la grille de la couche de prédictions où les mélanges au sein des blocs de couleurs sont bien plus nombreux. On retrouve cependant des similitudes dans le processus de classification : des petits groupes d'éléments des classes `trouser` et `bag` sont détectés sur la première couche de convolutions, et restent ainsi jusqu'à la fin de la classification. Les classes `coat`, `shirt` et `pullover` sont toujours aussi compliquées à discriminer tout au long de la classification, tout comme les classes `sneaker`, `sandal` et `ankle boot`. Cependant, ces mélanges forment des groupes qui sont distincts les uns des autres jusqu'à la dernière couche (A et B).

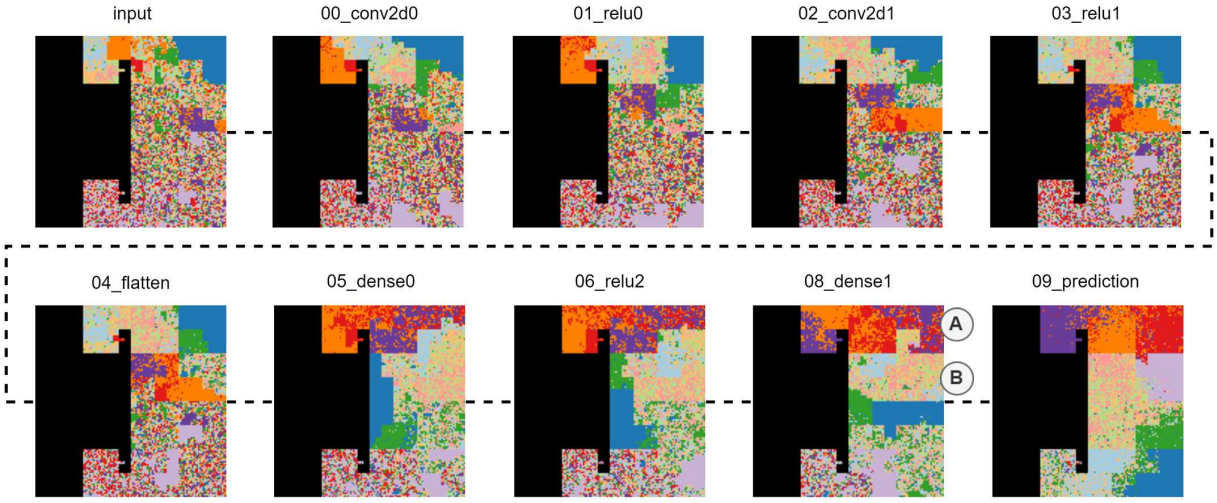


FIGURE 4.9 – Visualisation de la classification du jeu de données Fashion-MNIST par LeNet5 avec 73, 4% des prédictions correctes.

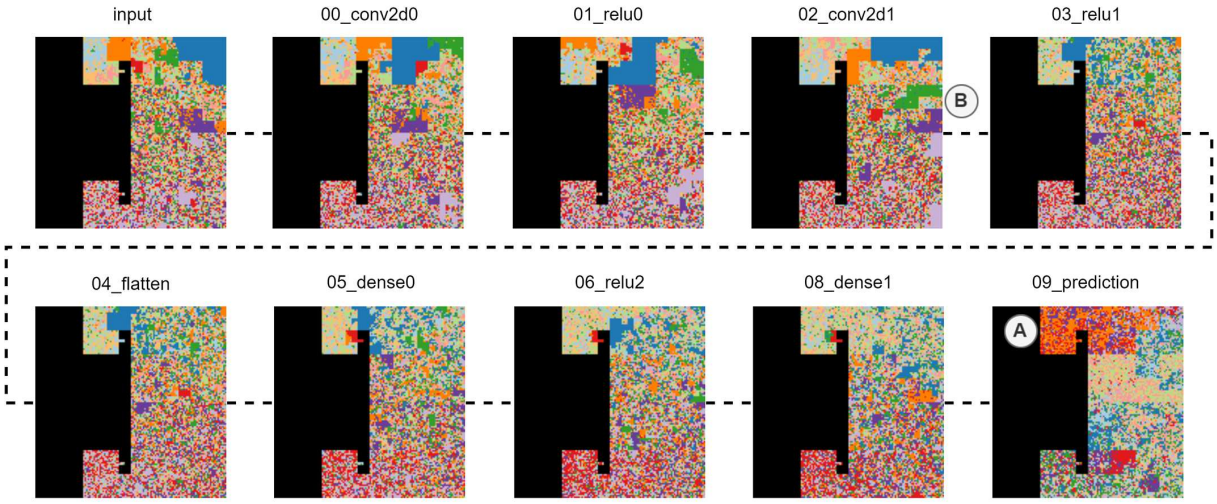


FIGURE 4.10 – Visualisation de la classification du jeu de données Fashion-MNIST par LeNet5, mais qui a été entraîné avec le jeu de données MNIST. Il obtient 11, 3% de prédictions correctes.

4.5.3 Scénario 3 : LeNet5 entraîné avec MNIST mais évalué avec Fashion-MNIST

La visualisation du scénario utilisant un réseau qui n'a pas appris sur un bon jeu de données est montrée en Figure 4.10. Là encore, la précision du modèle est reflétée sur la grille de pixels de la couche de prédictions avec un mélange important des éléments. On retrouve aussi la détection partielle de quelques groupes de données dans les couches de convolution (B). Ces groupes sont cependant «perdus» dans les couches qui suivent. On remarque sur la grille de la couche de prédictions que des ensembles de classes suivent une tendance similaire dans la classification : les classes `sneaker` et une partie des éléments de la classe `ankle boot` sont proches dans la grille (A) et ne sont pas mélangées avec des classes comme `trouser` ou `dress`.

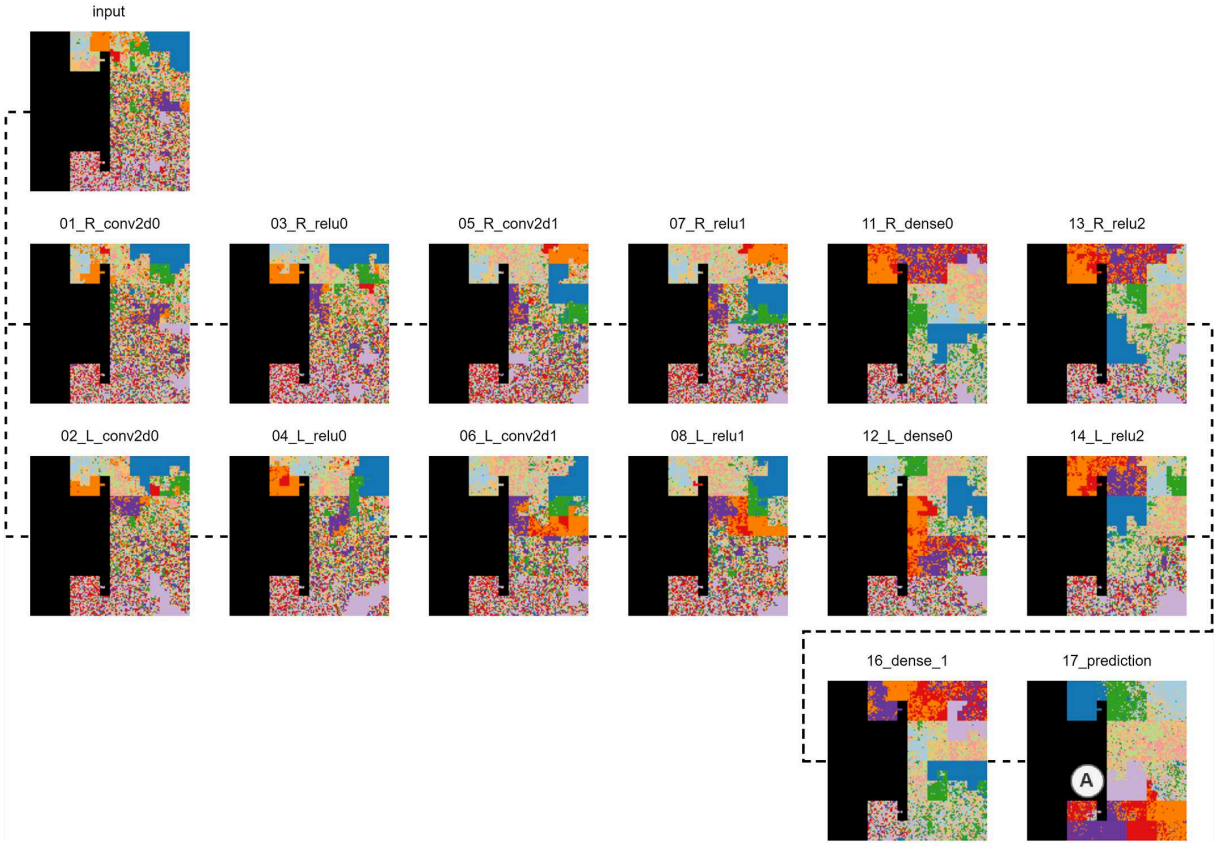


FIGURE 4.11 – Visualisation de la classification du jeu de données MNIST par le réseau à branchements DoubleLeNet5 avec 73, 8% des prédictions correctes.

4.5.4 Scénario 4 : DoubleLeNet5 entraîné et évalué avec Fashion-MNIST

La visualisation du scénario utilisant notre réseau à branches est montrée en Figure 4.11. Le gain de précision sur cette tâche par rapport au réseau LeNet5 est marginal, et la grille en couche de prédictions présente peu de différences avec celui du scénario 2. Une différence notable cependant concerne les éléments de la classe `sandal` séparés en deux groupes sur la grille (Ⓐ). En inspectant les valeurs des éléments en sortie du réseau (avant le choix de la valeur maximale), on remarque que les éléments du groupe le plus à droite présentent des valeurs gravitant autour de 0, 82 pour la composante de la classe `sandal` et 0, 03 pour la composante de la classe `ankle boot`. En revanche, les valeurs des éléments du groupe à gauche gravitent autour de 0, 55 pour `sandal` et des valeurs plus étendues entre 0, 05 et 0, 30 pour `ankle boot`. Cette nette différence de valeurs entre les deux groupes, et donc des distances dans $\mathbb{R}^N$ avec le reste des éléments de la classe `ankle boot` explique la séparation de la classe `sandal` en deux. La proximité de ces clusters dans $\mathbb{R}^N$ peut ainsi expliquer la représentation de la classe `ankle boot` entre les groupes `sandal`. Néanmoins, cette perturbation ne comptabilise pas d'erreurs de classification pour autant.

En ce qui concerne les branches parallèles du réseau, nous ne remarquons pas de comportement très différent entre les deux «sous-modèles», ce qui peut expliquer le faible gain de performance de cette architecture en particulier.

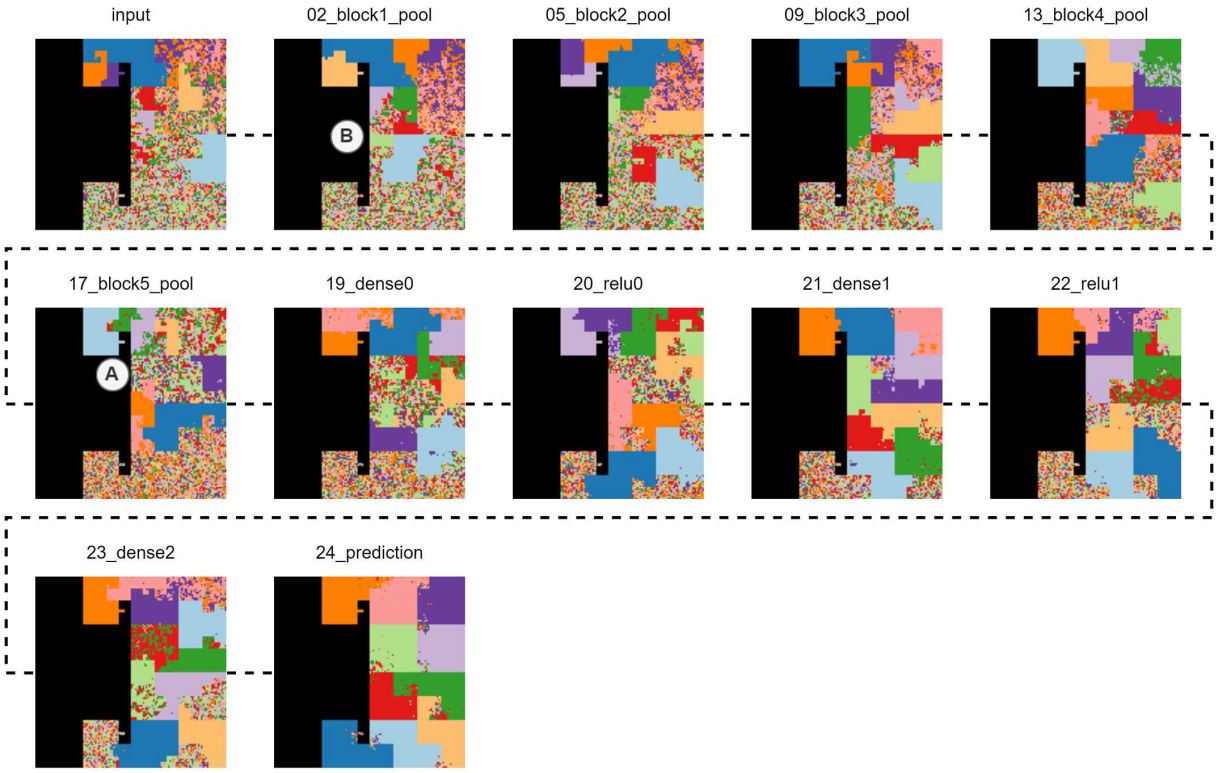


FIGURE 4.12 – Visualisation de la classification du jeu de données MNIST par VGG16 avec 97,0% des prédictions correctes.

4.5.5 Scénario 5 : VGG16 spécialisé et évalué sur MNIST

La visualisation de ce scénario est présentée en Figure 4.12. La composition détaillée des blocs convolutifs du réseau est masquée ici par soucis d'espace, seules les dernières couches de chaque bloc ont été dessinées. Les calculs intermédiaires de ces blocs sont visibles sur l'outil de démonstration sur le site internet du projet.

Le modèle obtient 97,0% de précision sur cette tâche, ce qui est très similaire aux performances obtenues dans le Chapitre 3. On s'attend donc à retrouver dans les grilles de pixels une bonne partie des observations faites sur la visualisation par flux. C'est le cas des blocs convolutifs qui discriminent rapidement les différentes classes du jeu de données, et on retrouve la régression à partir du 4e bloc convolutif. En revanche, ici, la régression semble bien plus importante sur la grille issue du 5e bloc convolutif avec le groupe des 3 complètement éparpillé (A). On peut voir aussi une autre différence entre les deux visualisations : Dans la grille du premier bloc convolutif, les groupes des 1 et des 2 sont bien séparés (B), là où MCL avait détecté un cluster regroupant les deux classes dans le chapitre précédent. Cette visualisation nous montre donc que les informations permettant de discerner les classes 2, 3 et 4 ont été perdues par ce 5e bloc convolutif, et que ces éléments sont confondus dans $\mathbb{R}^N$. En plus de cela, nous pouvons remarquer la présence de clusters plus ou moins bien formés d'autres classes telles que les 9 et les 6 dans ce mélange, montrant donc que ce dernier occupe un grand espace dans $\mathbb{R}^N$.

Optimisation du réseau. On propose d'optimiser le réseau pour obtenir de meilleures performances et de réduire sa complexité en se basant sur la visualisation obtenue. D'après notre première étude du réseau sur la visualisation en Figure 4.12, le 5e bloc convolutif semble interférer avec la classification de l'information. Nous pensons que cette perte de performances est due à une surinterprétation des attributs bas niveaux, et donc à un surapprentissage. Nous proposons deux solutions (schématisés en Figure 4.13) pour résoudre ce problème :

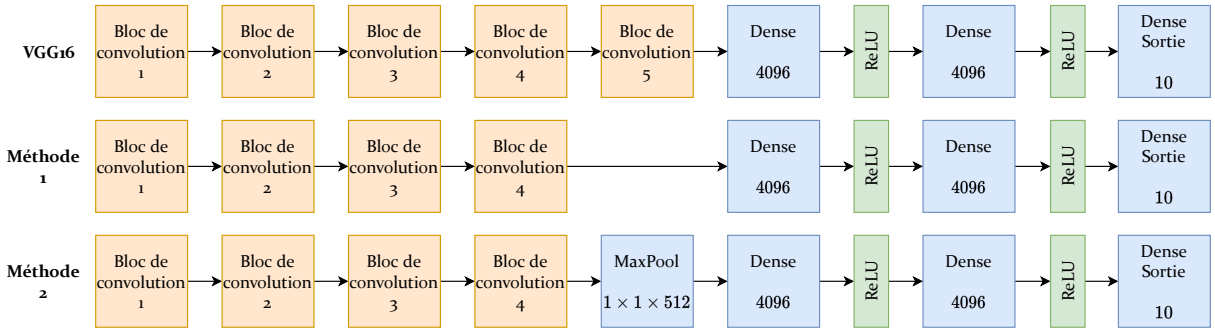


FIGURE 4.13 – Pistes d'améliorations envisagées pour optimiser les performances du réseau VGG16 dans le scénario 5. La première consiste à retirer le 5e bloc convolutif du réseau et de relier directement la sortie du 4e bloc à la couche dense. La seconde consiste à uniquement conserver la couche de *pooling* du 5e bloc pour réduire les opérations appliquées sur la donnée d'entrée sans changer le nombre de poids en entrée de la couche dense.

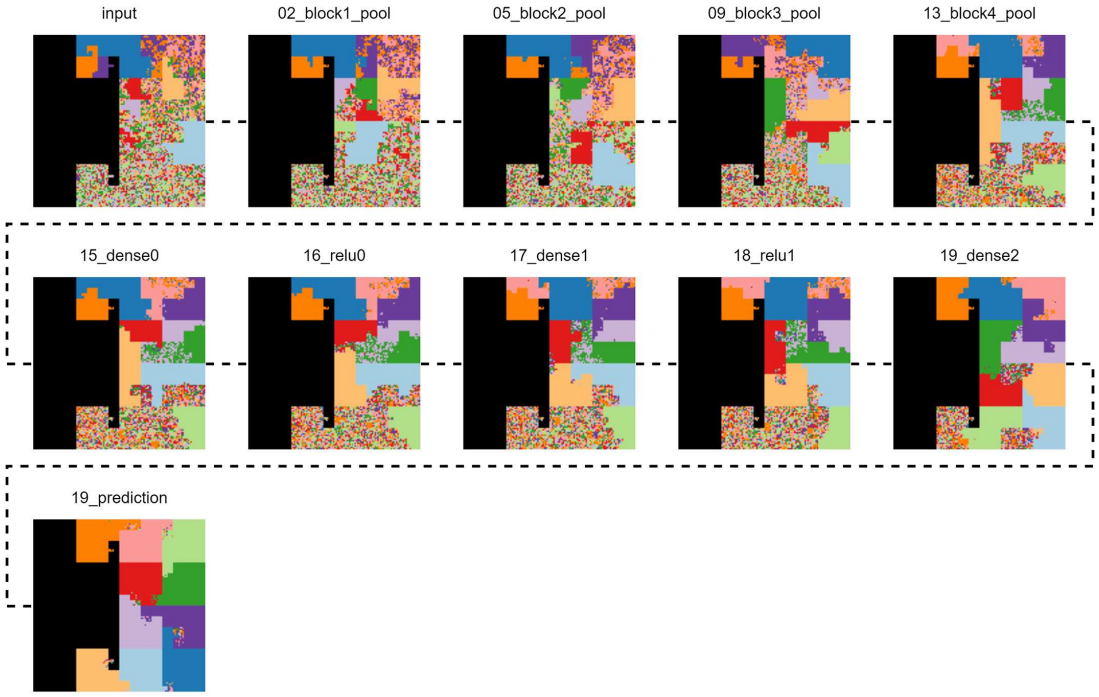
TABLE 4.1 – Performances et complexité des variations de VGG16. La méthode 1 consiste à retirer entièrement le 5e bloc convolutif du réseau, tandis que la méthode 2 consiste à retirer les couches convolutives du même bloc mais en conservant la dernière couche de *pooling*. Les performances obtenues sont similaires entre les deux méthodes, mais la méthode 2 permet de réduire la complexité du réseau original de près de 20%.

Modèle	Précision	Nombre de paramètres (relatif)
VGG16	97, 00%	33 638 218 (100%)
Méthode 1	98, 65%	32 850 250 (97%)
Méthode 2	98, 35%	26 558 794 (79%)

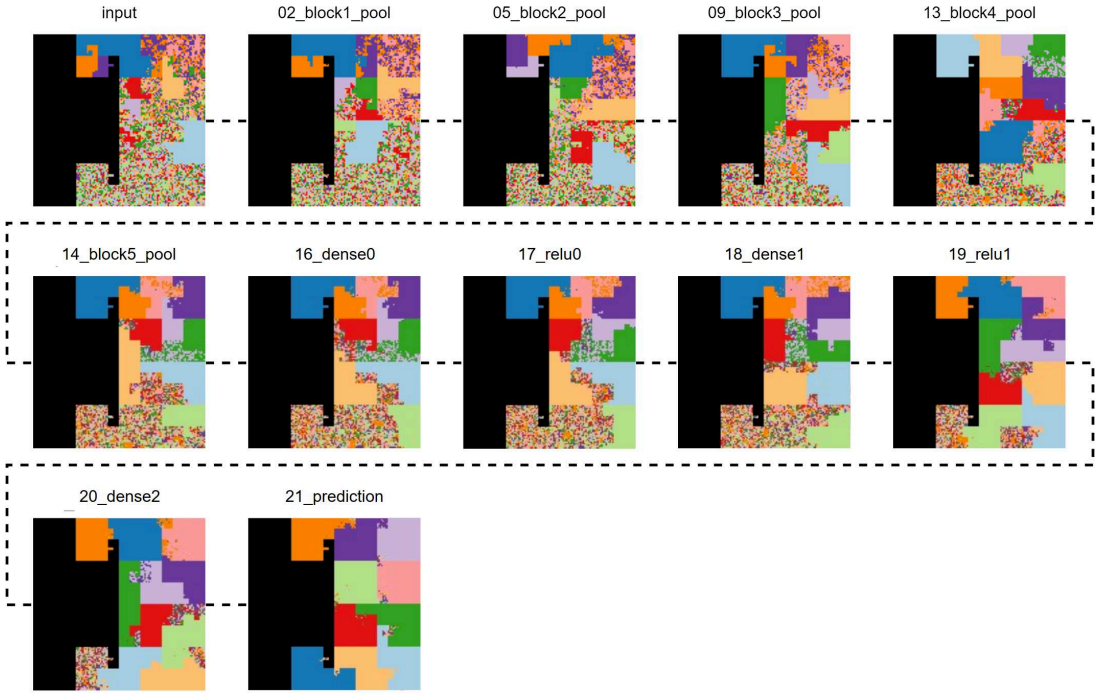
— La première consiste à retirer le 5e bloc et de connecter directement la sortie du 4e bloc à la première couche dense. Le nombre de poids de la couche dense augmente, mais la réduction du nombre d'opérations sur la donnée d'entrée par les couches de convolution devrait mener à un gain de précision.
— La deuxième consiste à enlever la partie convolutive du bloc mais à garder la couche de *pooling* pour conserver la taille des données en entrée de la couche dense. Cette méthode diminue considérablement la complexité du modèle car elle n'augmente pas le nombre de poids de la couche dense. On s'attend à un gain de performances, mais possiblement plus faible que la première option.

Nous avons construit les deux réseaux en suivant ces options, conservé les paramètres issus de l'entraînement sur ImageNet et ré-entrainé la partie dense des réseaux, sans conserver les paramètres déjà appris. Ceci nous permet de baser les potentielles améliorations obtenues uniquement sur nos observations et non sur de l'apprentissage par transfert.

La performance de ces nouveaux réseaux est présentée en Table 4.1. Les modifications apportées à l'architecture permettent un gain d'au moins 1, 35% et une réduction du nombre de paramètres dans le réseau. La conservation de la couche de *pooling* du bloc convolutif permet de diminuer le nombre de paramètres de 21%, ce qui est loin d'être négligeable sur un réseau aussi complexe que VGG16. Les visualisations de ces nouveaux réseaux sont présentées dans la Figure 4.14.



(a) VGG16 modifié (5e bloc retiré)



(b) VGG16 modifié (5e bloc retiré mais pooling conservé)

FIGURE 4.14 – Visualisations obtenues des classifications du jeu de données MNIST par les deux architectures modifiées de VGG16. La première option est affichée en (a) et la deuxième option en (b). (a) obtient 98, 65% de précision avec 2, 34% de paramètres en moins et (b) obtient 98, 35% de précision avec 21% de paramètres en moins.

4.6 Discussion

La méthode ne demande pas d'intervention de l'utilisateur pour fournir des paramètres en particulier afin générer la visualisation, en comparaison avec la méthode de visualisation par flux

où MCL nécessitait un paramètre pour pouvoir calculer les groupements de données. En revanche, les calculs ne sont pas plus rapides, notamment à cause du calcul d'une matrice de distance (utilisation de N^2 en espace de stockage) risquant de poser des problèmes de passage à l'échelle pour les jeux de données plus grands. De plus, si VAT permet de calculer des groupes de données similaires dans $\mathbb{R}^N$ par leurs distances par paire, il présente une complexité de $O(N^2)$ en nombre d'opérations. Dans nos implémentations, nous avons exploité les architectures distribuées et les processeurs graphiques pour palier en partie cette complexité élevée en temps et en mémoire, mais cette complexité ne permet pas l'utilisation de ces méthodes à plus large échelle. L'algorithme VAT a été cependant retravaillé dans la littérature et présente plusieurs variations, mais qui ne correspondent pas à notre cas d'usage de la méthode :

— co-VAT [10] est une variation proposée par l'auteur pour supporter des matrices de dissimilarité rectangulaires (dont les lignes et colonnes ne représentent pas les éléments d'un même ensemble). Cette variation n'est pas applicable à notre cas d'usage, où la matrice de dissimilarité (ou de distances) est calculée sur un même ensemble d'éléments.

— iVAT [142, 46] est une variante qui change la façon dont les longues distances sont perçues sur la matrice réarrangée, mais le calcul de l'ordre des lignes et colonnes reste globalement le même. Néanmoins, cette variante ne nous est pas utile puisque nous ne visualisons pas la matrice de distance réarrangée.

— reVAT [56] et son extension bigVAT [55] sont d'autres variantes de l'auteur de la méthode originale pour s'adapter aux jeux de données plus grands en réduisant sa complexité. reVAT calcule un tri approximatif des éléments et délimite les partitions dans $\mathbb{R}^N$. Cette méthode n'est donc pas compatible avec notre utilisation de la courbe de Hilbert car nous cherchons à trier précisément les éléments dans $\mathbb{R}^N$ pour les placer sur la courbe. bigVAT propose la même approche mais en ne traitant qu'un échantillon du jeu de données plutôt que l'ensemble.

— sVAT [45] est une autre variante conçue pour s'adapter aux grands jeux de données, mais demande à l'utilisateur une estimation du nombre de groupe à détecter, ce qui n'est pas souhaitable dans notre cas d'usage.

Nos expérimentations avec les réseaux de neurones font usage de TensorFlow [1], qui peut fonctionner sur processeur et sur carte graphique (GPU) en exploitant notamment la technologie CUDA [96]. Dans nos expérimentations, nous avons remarqué que pour un même réseau entraîné, ayant donc les mêmes paramètres sur chaque couche, la classification d'une donnée sur un CPU ou sur un GPU ne donne pas exactement les mêmes cartes d'activation entre chaque couche. Les différences sont minimes et n'ont pas d'influence (du moins, dans nos scénarios) sur la classification finale. En revanche, elles influent sur le calcul des distances dans $\mathbb{R}^N$ dans notre processus pour générer les grilles de pixels. Ainsi, lors de l'exécution de l'algorithme VAT, les valeurs choisies pour ordonner les lignes et colonnes ne sont pas forcément les mêmes dans la matrice de distance, et la disposition des éléments sur la courbe fractale n'est donc pas la même d'une évaluation à une autre.

Les différents scénarios montrent que l'on obtient les mêmes observations que sur la visualisation précédente, mais de manière différente. La représentation de chaque élément individuellement sous la forme d'un pixel est envisageable pour montrer la classification de l'information au niveau de chaque couche du réseau. Dans le scénario utilisant le réseau VGG16 pour classer le jeu de données MNIST, nous avons proposé une manière de simplifier le modèle d'après les observations fournies par cette méthode de visualisation. Cette proposition a mené à de meilleures performances et à une réduction du nombre de paramètres du réseau.

Malgré le gain d'espace et les observations que l'on peut faire sur cette méthode de visualisation par rapport à celle du Chapitre 3, nous pouvons relever deux problèmes sur l'utilisation des courbes fractales :

— La courbe de Hilbert demande de placer 4^n éléments afin d'être exploitée entièrement, sans espace vide. Dans les scénarios étudiés, nous avons travaillé avec des ensembles d'évaluation composés de 10 000 éléments qui ne peuvent donc pas recouvrir entièrement la

courbe fractale. Les grilles produites ont donc de grands espaces non-utilisés ($4^7 - 10\,000 = 6\,384$ pixels, à peu près 39% de la grille). Cet aspect est plutôt problématique dans notre cas puisque nous essayons de maximiser la quantité d'information affichée par espace utilisé sur l'écran.

— La transformation de la donnée dans $\mathbb{R}^N$ en une suite de positions sur la courbe fractale revient à effectuer une projection $\mathbb{R}^N \rightarrow \mathbb{N}$, avant de la visualiser dans un espace $\mathbb{R}^2$. Si la courbe de Hilbert permet de conserver la proximité des éléments de $\mathbb{R}^N$ une fois qu'ils sont positionnés sur la courbe, la réduction à une seule dimension nous fait perdre de l'information concernant les données dans $\mathbb{R}^N$ et l'espace $\mathbb{N}$ représenté sur la grille de pixels devient très déformé.

4.7 Conclusion

Dans ce chapitre, nous avons amélioré la méthode de visualisation précédente en représentant la classification progressive de réseaux de neurones sous la forme de grilles de pixels. L'utilisation de ces grilles de pixels nous permet de mieux utiliser l'espace de dessin pour fournir des informations sur les données étudiées, et donc à nous adapter à des scénarios plus complexes. Cette méthode nous permet aussi de nous adapter aux architectures de réseaux non-linéaires. Ces grilles de pixels sont générées à partir d'une courbe fractale et les distances entre éléments dans $\mathbb{R}^N$. Sur les scénarios étudiés et malgré les conditions sous-optimales d'utilisation de la courbe de Hilbert concernant le nombre d'éléments à traiter, nous avons pu retrouver les mêmes observations faites lors du chapitre précédent. Contrairement à la méthode du Chapitre 3 où l'utilisation de l'algorithme MCL demandait un paramétrage pouvant varier suivant le scénario d'application, la méthode de ce chapitre ne demande aucun paramétrage particulier de la récupération des cartes d'activations à la construction des grilles de pixels. Dans le chapitre suivant, nous approfondissons le travail sur la construction des grilles de pixels en représentant au mieux les données dans $\mathbb{R}^N$ et en utilisant des méthodes minimisant la perte d'espace de dessin quelle que soit la taille du jeu de données.

Chapitre 5

Transformation d'une projection 2D vers une grille

Dans ce chapitre, nous poursuivons l'amélioration de nos travaux sur une vue compacte pour représenter la classification au niveau de chaque couche d'un réseau. Dans le chapitre précédent, nous avons montré qu'utiliser des grilles de pixels pour afficher la classification du réseau met en évidence les proximités des éléments dans $\mathbb{R}^N$. En revanche, nous avons relevé des problèmes au niveau de l'utilisation de l'espace de dessin. Bien que l'approche cherche à minimiser le nombre de pixels utilisés, il reste une perte d'espace non négligeable sur la grille construite. Cette perte peut s'étendre à $3 \times 4^N - 1$ cellules si le jeu de données affiché est de taille $4^N + 1$ (soit un élément de plus que la taille d'une courbe de Hilbert de taille 4^N qui doit donc être de taille 4^{N+1} pour contenir le jeu de données), ce qui représente donc près de 75% de la grille de pixels. De plus, les données à l'origine dans $\mathbb{R}^N$ ont subi deux projections : la première de $\mathbb{R}^N$ vers $\mathbb{N}$ pour construire une suite ordonnée d'indices, puis de $\mathbb{N}$ vers $\mathbb{N}^2$ lors de leur positionnement sur la courbe fractale. Cette succession de projections provoque une perte d'information significative puisqu'on ne conserve que le voisinage direct des éléments. Des alternatives existent pour projeter les éléments de $\mathbb{R}^N$ vers $\mathbb{R}^2$, comme par exemple MDS [67]. Celle-ci cherche à minimiser une fonction de stress et reflète à la fois les distances courtes et longues entre les éléments de $\mathbb{R}^N$. Pour la réalisation des grilles de pixels, nous avons besoin d'une disposition compacte de nos éléments dans $\mathbb{N}^2$, où chaque élément est adjacent aux autres, sans espace vide entre. Les méthodes de projection dans $\mathbb{R}^2$ ne sont donc pas utilisables telles quelles, puisqu'elles encodent les distances plus grandes entre les éléments par des espaces vides.

Dans ce chapitre, nous proposons une méthode permettant de calculer un arrangement d'éléments en grille (un ensemble de coordonnées dans $\mathbb{N}^2$) pour n'importe quel jeu de données dans $\mathbb{R}^2$, à la seule condition que des points ne partagent pas la même position dans $\mathbb{R}^2$. Nous nous focalisons sur des données dans $\mathbb{R}^2$ afin d'exploiter les avantages des projections classiques. Cet arrangement permet entre autres d'utiliser n'importe quelle méthode de projection $\mathbb{R}^N$ vers $\mathbb{R}^2$ (comme l'algorithme t-SNE [136] souvent utilisé pour la visualisation d'information) dans notre outil de visualisation de classification des réseaux de neurones. L'utilisation de ces méthodes de projection nous permettent par la suite d'étudier les espaces $\mathbb{R}^N$ des cartes d'activation comme on étudierai n'importe quel jeu de données $\mathbb{R}^N$. Nous évaluons l'efficacité de notre méthode face aux techniques d'arrangements compacts existantes en utilisant des métriques reflétant la préservation de l'information lors du passage de $\mathbb{R}^2$ vers $\mathbb{N}^2$.

En premier lieu, nous présentons notre démarche pour mesurer ce qu'est un bon arrangement $\mathbb{R}^2 \to \mathbb{N}^2$ et les travaux existants. Ensuite, nous expliquons le fonctionnement de notre méthode d'arrangement et détaillons son implémentation. Après cela, nous montrons les résultats de l'évaluation de notre méthode face à celles existantes, et clôturons sur une discussion des améliorations potentielles. Enfin, nous présentons une application de la méthode pour l'étude des réseaux de neurones de manière analogue aux méthodes des chapitres précédents.

Les contributions dans ce chapitre ont mené à :

— Une publication en conférence : **Halnaut, A.**, Giot, R., Bourqui, R., & Auber, D. (2022, Juillet). «VRGrid : Efficient Transformation of 2D Data into Pixel Grid Layout». Dans *26th International Conference Information Visualisation*.

— Une version étendue en chapitre de livre : **Halnaut, A.**, Giot, R., Bourqui, R., & Auber, D. (2024, Avril). «Computation of Pixel-oriented Grid Layout for 2D Datasets using VRGrid». Dans *Artificial Intelligence and Visualization : Advancing Visual Knowledge Discovery*.

— Un outil dont le code source est disponible à l'adresse https://github.com/eikofee/vrgrid.

5.1 Travaux existants

5.1.1 Méthodes d'arrangement polyvalentes

Nous présentons des méthodes ne dépendant pas du type de données ou d'un domaine d'application en particulier.

Self-Sorting Maps [128], ou SSM, est une méthode de projection de $\mathbb{R}^N$ vers une grille orthogonale. L'algorithme place les éléments aléatoirement dans la grille dans un premier temps, puis effectue des permutations entre des groupes de cellules afin de maximiser le score de corrélation croisée (*Cross-Correlation Score*, ou CCS), servant à comparer les distances d'origine (dans $\mathbb{R}^N$) avec celles sur la vue finale (en grille). La complexité algorithmique de SSM est de $O(L.n.\log(n))$, avec L le nombre d'itérations de l'algorithme à chaque étape pour converger vers une configuration stable. Elle peut être réduite à $O\left(L.\log(n)^2\right)$ dans sa version parallélisée. Ce facteur peut varier d'un jeu de données à un autre, mais SSM peut tout de même être utilisé pour les très grands jeux de données dans un temps de calcul raisonnable.

IsoMatch [35] est une méthode de projection en deux temps prenant en entrée des données dans $\mathbb{R}^N$ et les arrangeant dans un ensemble de positions cibles, pouvant être les coordonnées $\mathbb{N}^2$ des cellules d'une grille, mais également un arrangement quelconque imposé par l'utilisateur. IsoMatch utilise dans un premier temps IsoMap [133] pour projeter les données d'entrée dans $\mathbb{R}^2$, puis utilise l'algorithme Hongrois [68] pour attribuer les positions calculées de $\mathbb{R}^2$ vers les positions cibles. Dans l'article des auteurs, IsoMatch est comparée à SSM et obtient des résultats «légèrement, mais statistiquement meilleurs» en CCS : donc en préservation des distances. Les auteurs d'IsoMatch proposent aussi de comparer les deux méthodes suivant une «fonction d'énergie» qui calcule une déformation de la structure des données entre son espace d'origine et celui de l'arrangement. L'objectif des méthodes est donc d'obtenir les valeurs les plus faibles possibles sur cette fonction, et IsoMatch est statistiquement meilleure que SSM à cette tâche. En revanche, l'algorithme souffre d'une complexité assez élevée de $O\left(n^4\right)$, notamment à cause de l'utilisation de l'algorithme Hongrois étant de complexité $O\left(n^3\right)$, ce qui rend son utilisation difficile sur les grands jeux de données tels que ceux que l'on utilise dans nos outils de visualisations.

Distance-preserving Grid [49, 81], ou DGrid, est une méthode d'arrangement en grille en deux temps comme IsoMatch. D'abord, les données dans $\mathbb{R}^N$ sont projetées dans $\mathbb{R}^2$ via une méthode de projection (t-SNE est recommandé pour cette étape), puis les données sont réarrangées sous la forme de partitions dans $\mathbb{N}^2$ de façon à former une grille. Avec une complexité de calcul de seulement $O(n.\log(n))$, cette méthode est la plus rapide des trois et ne dépend principalement que de la méthode de projection $\mathbb{R}^N \to \mathbb{R}^2$ employée. Elle peut fonctionner sur n'importe quel jeu de données. Pour son évaluation, les auteurs de DGrid l'ont comparée en employant une métrique mesurant la conservation du voisinage des points entre la disposition initiale des données dans $\mathbb{R}^N$ et celle en grille. Ils obtiennent des résultats sensiblement meilleurs que SSM et IsoMatch.

Hagrid [22] est une méthode d'arrangement sur une grille d'éléments préalablement disposés sur $\mathbb{R}^2$. La méthode exploite les courbes fractales, comme celle de Hilbert utilisée dans le chapitre

précédent, pour affecter une nouvelle position aux éléments proches des coordonnées constituant la courbe fractale. La méthode peut être perçue comme une insertion successive d'éléments sur une grille dont le nombre de cellules est supérieur ou égal au nombre d'éléments à positionner. En cas de «collision» d'un élément avec un autre déjà positionné, l'algorithme recherche une nouvelle position libre proche sur la courbe fractale afin d'y insérer l'élément problématique. Contrairement aux méthodes présentées auparavant, Hagrid ne cherche pas à produire un arrangement compact d'une projection de points en $\mathbb{R}^2$.

Nmap [31] (*Neighborhood Treemap*) est une technique de projection ayant pour but de remplir complètement un espace de dessin en affichant les éléments d'un jeu de données sous la forme de rectangles positionnés les uns à côté des autres. La taille de ces rectangles est représentative de la valeur de poids associée à chaque élément. Les Nmaps diffèrent des *treemaps* [60] car elles ont la particularité de positionner les éléments de façon à représenter le voisinage des éléments, là où les *treemaps* se limitent à la représentation des structures hiérarchiques. La méthode consiste à partitionner l'espace de dessin en deux parties égales dans un premier temps et de déplacer la frontière entre ces deux parties en fonction du poids des éléments de part et d'autre de cette frontière. Cette approche est envisageable pour produire un arrangement compact, mais l'utilisation de valeurs de poids faisant varier les tailles des éléments donne un rendu différent de celui d'un affichage en grille que l'on recherche ici.

OoDAnalyzer [17] propose une approche adaptée à l'étude de performances de réseaux de neurones pour détecter les mauvaises prédictions. La méthode consiste à projeter dans un premier temps les informations dans $\mathbb{R}^2$, puis à associer chaque élément ou groupe d'éléments dans la cellule d'une grille, en fonction du voisinage de ceux-ci. Cette approche a une complexité de $O(kN^2)$ avec k le nombre de voisins à considérer mais nécessite un environnement dynamique pour pouvoir explorer les données à différents niveaux de grossissement, ce qui empêche d'avoir tous les éléments d'un jeu de donnée visibles à la fois.

5.1.2 Évaluation des méthodes d'arrangement

Les méthodes d'arrangement polyvalentes présentées ont utilisé des métriques d'évaluation différentes pour mesurer la qualité de leurs arrangements entre elles.

— SSM utilise le CCS comme fonction à optimiser pour calculer son arrangement et ses évaluations sont basées sur cette même métrique.
— IsoMatch introduit la notion de déformation des données avec une fonction d'énergie.
— DGrid propose de mesurer la préservation du voisinage de chaque élément pour évaluer la qualité d'un arrangement.

Ces métriques permettent d'évaluer la qualité de l'arrangement en grille par rapport aux données dans $\mathbb{R}^N$. Nous nous intéressons dans ce chapitre uniquement à la qualité du plongement de $\mathbb{R}^2 \rightarrow \mathbb{N}^2$, puisqu'une évaluation prenant en compte la forme initiale des données dans $\mathbb{R}^N$ dépendrait alors des performances de la projection de $\mathbb{R}^N \rightarrow \mathbb{R}^2$ utilisée qui peut varier selon les objectifs de visualisation fixés (par exemple, t-SNE sert à obtenir un partitionnement des données, tandis que PCA sert à trouver une corrélation entre les éléments). Nous nous concentrons donc sur le meilleur moyen de représenter la totalité des résultats de ces projections en une grille compacte.

Pour définir ce qu'est un bon plongement d'un jeu de données de $\mathbb{R}^2$ dans $\mathbb{N}^2$, nous considérons qu'il est important d'avoir des bons résultats dans deux domaines :

— La préservation d'informations sur le jeu de données représenté, à savoir la relation qu'a un élément avec tous les autres. Les métriques CCS (préservation des distances entre les éléments) et préservation des voisinages en font partie.
— La préservation d'informations sur la représentation graphique de la projection, il ne faudrait pas que l'arrangement en grille diffère trop du positionnement obtenu par la projection originale (qui n'existe pas avec SSM) afin de ne pas perdre son sens et de faciliter son interprétation. Par exemple, si nous projetons un jeu de données dans $\mathbb{R}^2$ en se ser-

vant de t-SNE et que la représentation par nuage de points de cette projection positionne un ensemble d'éléments sur la partie supérieure gauche du dessin, nous voudrions que ce même ensemble d'éléments soit positionné dans la partie supérieure gauche de la grille compacte. Ce comportement n'est pas mesuré par les métriques CSS et de préservation du voisinage, qui ne considèrent que les relations entre éléments.

Les métriques CCS et de préservation des voisinages (que l'on nommera dans la suite du chapitre KNP pour *k-Neighborhood Preservation*) nous semblent suffisantes pour le domaine de préservation d'information sur les données. Pour celui reposant sur les informations du dessin de la projection, nous proposons de nous appuyer sur des mesures d'évaluations utilisées dans le domaine de recherche sur les dessins de graphes.

Une des problématiques du dessin de graphe est celui de la suppression de superposition des sommets (*overlap removal*) [40]. Cette problématique concerne les graphes ayant des sommets de tailles variables pouvant se chevaucher si ces derniers sont dessinés trop proches les uns des autres, résultant en une interprétation du graphe plus difficile. Les méthodes de suppression de superpositions cherchent à déformer les positions de ces sommets afin de n'avoir aucune superposition parmi eux, tout en altérant le moins possible le dessin du graphe d'origine, afin d'en améliorer la lecture. Chen et al. [18] recensent plusieurs méthodes de ce domaine de recherche et proposent de les comparer en utilisant des fonctions permettant de mesurer les déformations produites par leur application. Parmi ces métriques, nous retenons dans ce chapitre celles concernant l'ordonnancement orthogonal des sommets (*Orthogonal Ordering Preservation*) et le mouvement des nœuds entre leur position d'origine et celle après suppression des superpositions (*Node Movements*) pour les raisons suivantes :

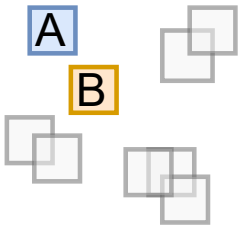
— L'ordonnancement orthogonal consiste à mesurer le nombre de permutations qu'il y a entre deux sommets sur les axes X et Y. Nous souhaitons qu'un sommet A se trouvant en haut à gauche d'un autre sommet B ne se retrouve pas en bas à droite de ce dernier une fois la transformation effectuée. Cette mesure est plutôt pertinente dans notre cas d'usage puisque les grilles ont une disposition orthogonale et nous cherchons à ce que cette disposition corresponde le plus possible aux données originales. Dans la suite de ce chapitre, on nommera cette mesure la préservation du positionnement relatif des éléments, ou RPP (*Relative Positioning Preservation*).

— Le mouvement des nœuds consiste à mesurer la distance cumulée qu'ont parcouru tous les nœuds entre leur disposition initiale et celle après transformation. Cette mesure est similaire à l'objectif de la fonction de calcul d'énergie proposée par les auteurs d'IsoMatch, puisque qu'elle permet de mesurer directement les transformations appliquées au graphe original. Dans notre cas d'usage, le déplacement des éléments lors de l'arrangement en grille est inévitable, mais on cherchera à minimiser les déplacements des éléments pour ressembler au mieux à la configuration d'origine. Dans la suite de ce chapitre, on nommera cette mesure la préservation du positionnement global des éléments, ou GPP (*Global Positioning Preservation*).

La combinaison des deux mesures permet d'évaluer correctement plusieurs cas de figures où l'une seule d'elles n'est pas suffisante, dont des exemples sont montrés en Figure 5.1.

5.2 Définition des métriques d'évaluation

Dans cette section, nous travaillons avec un jeu de données $\mathcal{D} \subset \mathbb{R}^2$, et nous nommons $\mathcal{G}(u, v)$ la fonction retournant les distances entre les éléments $u \in \mathcal{D}$ et $v \in \mathcal{D}$ dans la grille et respectivement $\delta(u, v)$ la fonction retournant les distances dans le positionnement initial dans $\mathbb{R}^2$. Pour les calculs impliquant les coordonnées des points sur les axes x et y, nous utilisons les notations $x_{\mathcal{G}}(u)$ et $y_{\mathcal{G}}(u)$ pour obtenir les coordonnées de u dans la grille et $x_{\delta}(u)$ et $y_{\delta}(u)$ pour celles dans le positionnement initial dans $\mathbb{R}^2$.



(a) Données d'entrées dans $\mathbb{R}^2$

Arrangement						
RPP	Bonne	Mauvaise	Bonne	Mauvaise	Bonne	Moyenne
GPP	Bonne	Mauvaise	Moyenne	Moyenne	Moyenne	Moyenne

(b) Evaluations de GPP/RPP en fonction de l'arrangement produit dans $\mathbb{N}^2$

FIGURE 5.1 – Exemples de mesures RPP et GPP sur un arrangement dans $\mathbb{N}^2$. RPP retourne des valeurs moyennes lorsque les éléments A et B sont permutés sur un axe, et mauvaises lorsqu'il y a permutation sur les deux axes. L'évaluation est bonne lorsqu'il n'y a pas de permutation. GPP retourne des bonnes valeurs lorsque les éléments A et B sont proches de leur position de départ, et se dégrade empiriquement lorsque ces éléments s'éloignent de leur position initiale. La combinaison des deux mesures permet de s'assurer que les éléments soient bien positionnés dans la grille.

Préservation des distances (DP) [128]

Le score de corrélation croisée est défini par la formule suivante :

$$DP = \frac{1}{|\mathcal{D}|^2} \sum_{(s,t)\in\mathcal{D}^2} \frac{\left(\mathcal{G}(s,t) - \overline{\mathcal{G}}\right)\left(\delta(s,t) - \overline{\delta}\right)}{\sigma_{\mathcal{G}}\sigma_{\delta}} \tag{5.1}$$

où $\overline{\mathcal{G}}$ et $\overline{\delta}$ sont les moyennes des distances entre les éléments respectivement dans la grille et dans $\mathbb{R}^2$. $\sigma_{\mathcal{G}}$ et σ_{δ} sont les écarts types des distances entre les éléments respectivement dans la grille et dans $\mathbb{R}^2$. Le score de corrélation croisée retourne des valeurs décroissants de manière exponentielle à mesure que les écarts de distance entre la configuration d'origine et celle de l'arrangement grandissent. Des valeurs élevées indiquent de bonnes performances de la méthode d'arrangement puisque les distances d'origine sont bien représentées une fois dans que les éléments sont positionnés dans la grille.

Préservation des voisinages (KNP) [49]

Nous calculons la préservation des voisinages avec la formule suivante :

$$KNP_r = \frac{1}{2 - 2^{1-r}} \sum_{1\leq i\leq r} \sum_{\forall s\in\mathcal{D}} \frac{|knn_{\delta}(s, f_i(s)) \cap knn_{\mathcal{G}}(s, f_i(s))|}{|\mathcal{D}| \times f_i(s) \times 2^{i-1}} \tag{5.2}$$

avec r la distance de Tchebychev maximale sur laquelle calculer la préservation du voisinage, $\mathcal{D}$ l'ensembles des éléments du jeu de données mesuré, $knn_{\delta}(s,k)$ la fonction retournant les k plus proches voisins de s dans $\mathbb{R}^2$ et $knn_{\mathcal{G}}(s,k)$ les k plus proches voisins de s dans la grille en utilisant la distance de Tchebychev. La formule du calcul de la distance de Tchebychev (aussi connue sous le

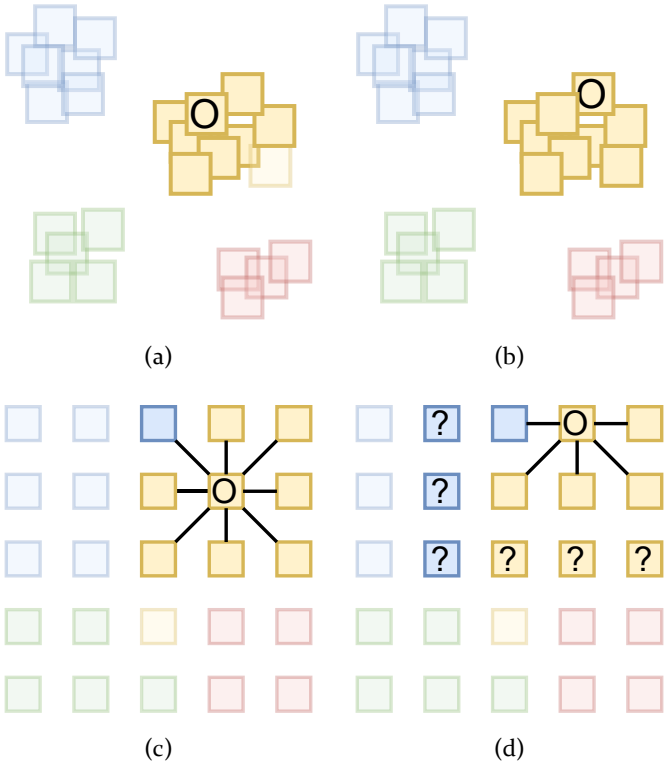


FIGURE 5.2 – Exemples d'arrangements de points dans $\mathbb{R}^2$ ((a) et (b)) et de leur arrangement en grille dans $\mathbb{N}^2$. Les voisins de «O» sont mis en évidence et peuvent être calculés d'une même façon dans les exemples (a) et (c). La composition du voisinage peut donc être comparée entre les deux dispositions. Pour l'arrangement de (b) en grille (d), le point étudié se trouve en bordure de la grille, donnant plusieurs choix possibles (marqués par un « ?») pour décider des 8 voisins de ce point dans la grille. Le résultat de la mesure de calcul de la préservation du voisinage peut drastiquement changer suivant les voisins choisis dans ce cas de figure. Cette préservation peut être bonne si le voisinage est alors complété avec des points jaunes, ou bien au contraire médiocre si le voisinage est complété avec les points bleus.

nom de *chessboard distance*) entre deux points a et b dans un espace $\mathbb{R}^2$ est définie de la manière suivante :

$$d_{\text{Tchebychev}}(a,b) = max(|x_a - x_b|, |y_a - y_b|) \tag{5.3}$$

où x_a et x_b sont les coordonnées sur l'axe x des points a et b et y_a et y_b sur l'axe y. La fonction $f_i(s)$ retourne le nombre de voisins à calculer pour l'élément $s \in \mathcal{D}$ en fonction de son positionnement dans l'arrangement en grille, définie par :

$$f_0(s) = 0, \quad f_i(s) = \begin{cases} f_{i-1}(s) + 1 + 2i & \text{si } s \text{ est dans un coin de la grille} \\ f_{i-1}(s) + 1 + 4i & \text{si } s \text{ est en bordure de la grille} \\ f_{i-1}(s) + 8i & \text{sinon} \end{cases} \tag{5.4}$$

Habituellement, pour mesurer la préservation du voisinage entre deux espaces, nous calculons le ratio entre le nombre de voisins commun d'un point entre les deux espaces, ce calcul repose sur un nombre de voisins fixes. En revanche, nous considérons ici des dispositions de points en grille, où la taille du voisinage de chaque point varie en fonction de son positionnement dans la grille. Sur les exemples montrés en Figure 5.2, les 8 plus proches voisins des éléments qui ne se trouvent pas en bordure de la grille correspondent aux éléments adjacents dans les 8 directions autour de ceux-ci. En revanche, les 8 plus proches voisins des éléments en bordure ou dans les coins de la grille ne sont pas les éléments adjacents à ceux-ci mais plutôt ceux vers le centre de la grille (donc

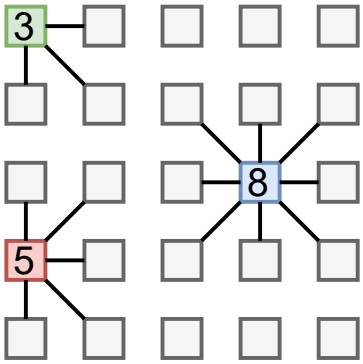


FIGURE 5.3 – Taille du voisinage à considérer suivant notre approche en fonction du positionnement du point à étudier, qu'il soit dans un coin (3), en bordure de grille (5) ou bien à l'intérieur (8).

à une distance de Tchebychev supérieure à 1). La disposition en grille limite considérablement le positionnement des éléments par rapport à d'autres, et chaque élément a au minimum deux voisins à la même distance de lui. Si nous calculons le ratio sur les 8 plus proches voisins, nous considérons que la préservation se mesure dans un rayon de 1 cellule pour les éléments positionnés dans la grille et dans un rayon plus grand pour les cellules en bordure et dans les coins. Afin d'obtenir une mesure cohérente pour tous les éléments de la grille, nous proposons deux modifications sur la formule de la mesure : nous changeons la façon dont les voisins sont sélectionnés, et nous calculons la préservation du voisinage au-delà de l'adjacence des sommets étudiés.

Dans notre mesure, nous considérons le voisinage d'un élément e comme l'ensemble des éléments $v \in V$ tels que $d_{\text{Tchebychev}}(e, v) \leq r$. En considérant que le voisinage d'un élément e dans la grille est l'ensemble des points à une distance de Tchebychev définie de e, le calcul du voisinage se fait de manière uniforme sur tous les éléments de la grille et ne devient pas déséquilibré suivant le positionnement de e sur la grille, comme sur la Figure 5.3.

Ensuite, nous observons la préservation du voisinage dans une région autour de chaque élément plutôt qu'à son adjacence. Avec la formule actuelle, qu'un élément se trouve à une cellule d'écart avec son voisin supposé ou qu'il s'en trouve à cinquante, le ratio obtenu reste le même. Afin de privilégier les configurations proches de l'idéal (quelques cellules d'écart), nous proposons d'évaluer le voisinage à plusieurs niveaux de proximité. Le dénominateur 2^{i-1} dans la formule 5.2 permet d'appliquer cette préférence, en réduisant l'importance de la préservation du voisinage sur le score final pour un élément à mesure que la distance utilisée pour calculer le voisinage $1 \leq i \leq r$ augmente. Nous appliquons cette réduction pour privilégier la préservation du voisinage de e lorsqu'il est composé des éléments adjacents à e par rapport au voisinage composé d'éléments plus éloignés de e. Le facteur $(2 - 2^{1-r})^{-1}$ permet quant à lui de normaliser le score obtenu sur $[0; 1]$.

Préservation du positionnement relatif (RPP)

Le score d'ordonnancement orthogonal est calculé par la formule suivante :

$$RPP = \frac{ooni_x + ooni_y}{|\mathcal{D}|(|\mathcal{D}| - 1)} \tag{5.5}$$

avec $|\mathcal{D}|$ la taille du jeu de données, et $ooni_x$ et $ooni_y$ les nombres d'inversions sur les axes, définis de la façon suivante :

$$ooni_x = \sum_{\substack{(u,v)\in\mathcal{D}^2 \\ x_\delta(u)>x_\delta(v)}} \begin{cases} 1 & \text{si } x_\mathcal{G}(u) < x_\mathcal{G}(v) \\ 0 & \text{sinon} \end{cases} \tag{5.6}$$

$$ooni_y = \sum_{\substack{(u,v)\in\mathcal{D}^2 \\ y_\delta(u)>y_\delta(v)}} \begin{cases} 1 & \text{si } y_\mathcal{G}(u) < y_\mathcal{G}(v) \\ 0 & \text{sinon} \end{cases} \tag{5.7}$$

avec $x_\mathcal{G}(s)$ et $y_\mathcal{G}(s)$ deux fonctions qui retournent respectivement la coordonnée sur l'axe x et l'axe y de l'élément s dans la grille. Les fonctions $x_\delta(s)$ et $y_\delta(s)$ font de même sur le jeu de données original dans $\mathbb{R}^2$, et nous servent pour chaque couple d'éléments à effectuer une seule comparaison entre eux.

La fonction retourne une valeur normalisée indiquant le nombre de permutations sur les axes que la configuration en grille obtient par rapport aux données d'origine. Des valeurs plus faibles signifient que l'ordonnancement a été mieux conservé lors de l'arrangement des données en grille.

Préservation du positionnement global (GPP)

Dans le domaine du dessin de graphes, le calcul du mouvement des nœuds combine un calcul de redimensionnement du dessin ainsi qu'un calcul des déplacements des sommets du graphe entre son état original et après la résolution des superpositions des sommets. Dans ce chapitre, nous travaillons avec des méthodes qui calculent des arrangements en grille. La taille de ces grilles dépend du nombre d'éléments à arranger, il n'y a donc pas de redimensionnement différent d'une méthode à une autre. En revanche, nous nous intéressons au calcul du déplacement des éléments qui représente une déformation appliquée sur l'ensemble des données. Le calcul du mouvement des nœuds est défini par la fonction suivante :

$$GPP = \frac{1}{|\mathcal{D}|} \sum_{s\in\mathcal{D}} ||\mathcal{G}(s) - shift(s)||^2 \tag{5.8}$$

et dont la fonction $shift(s)$ renvoie le déplacement du sommet s en prenant en compte les délimitations de l'enveloppe convexe des données dans $\mathbb{R}^2$. Elle permet notamment de recentrer les deux enveloppes convexes pour mieux calculer le déplacement de chaque noeud. Cette fonction est définie par la formule suivante :

$$shift(s) = [x_\delta(s) + xbb_\mathcal{G} - xbb_\delta, y_\delta(s) + ybb_\mathcal{G} - ybb_\delta] \tag{5.9}$$

où $xbb_\mathcal{G}$ et $ybb_\mathcal{G}$ sont les coordonnées sur l'axe x et y du centre de l'enveloppe convexe de l'arrangement en grille, et xbb_δ et ybb_δ sont de même pour les données d'origine dans $\mathbb{R}^2$. GPP retourne des valeurs plus élevées si les déplacements entre la configuration d'origine et celle en grille sont grands et nombreux, et donc une valeur plus faible signifie que l'arrangement est efficace.

5.3 Conception de VRGrid

VRGrid (Voronoi Relaxation Grid) est une méthode cherchant à produire un arrangement en grille avec comme objectifs :

— Déplacer le moins possible les données d'entrée dans $\mathbb{R}^2$ vers un arrangement en grille (RPP & GPP).
— Préserver au mieux les distances entre éléments (CCS).
— Conserver une cohérence dans le voisinage des éléments (KNP).

L'approche que nous proposons pour VRGrid diffère de SSM sur la construction de la grille. Au lieu de préparer les données dans une grille et d'effectuer des permutations entre les cellules, VRGrid construit itérativement la grille dans le même espace que celui des données d'entrée $\mathbb{R}^2$. Les données d'entrée sont ensuite déplacées progressivement vers les coordonnées des cellules de la grille. Ces mouvements cherchent à respecter les objectifs mentionnés ci-dessus. Ils se basent

sur une méthode itérative utilisant les diagrammes de Voronoi et des centroïdes de leurs cellules, et le remplissage de bordures réduisant le nombre d'éléments à traiter au fil de l'algorithme.

Dans cette section, nous présentons en premier lieu la partie itérative de VRGrid sur un petit jeu de données. Ensuite, nous évoquons les défis à relever pour un passage à l'échelle sur la taille des jeux de données et nos solutions. Après cela, nous décrivons l'algorithme étape par étape de manière plus formelle, en détaillant les outils algorithmiques et techniques utilisés pour son implémentation et le calcul de sa complexité.

5.3.1 Répartition uniforme des points (algorithme de Lloyd)

VRGrid repose essentiellement sur l'application d'un algorithme de relaxation permettant de répartir un ensemble de points $\mathcal{D}$ dans un espace fini et convexe dans $\mathbb{R}^2$. L'objectif de l'application de cette méthode est de répartir les points de manière à obtenir une disposition uniforme se rapprochant de celle d'une grille. Ici, nous avons choisi d'utiliser l'algorithme de Lloyd [78] comme algorithme de relaxation.

Cet algorithme permet de calculer une configuration de tessellation de Voronoi centroïdale [29] (*Centroidal Voronoi Tesselation*, ou CVT) qui consiste à ce que chaque site de cellule du pavage soit aussi le centroïde de celle-ci. Ce calcul n'est possible que si tous les points ont des positions différentes dans $\mathbb{R}^2$, ce qui définie la seule contrainte de notre méthode sur le jeu de donnée d'entrée. La configuration de tessellation de Voronoi centroïdale est relativement similaire à celle d'une disposition en grille, et la convergence de l'algorithme vers une CVT dans $\mathbb{R}^2$ a été prouvée [28] du moment que l'espace d'application est fini et convexe, ce qui est le cas pour notre usage.

Il consiste en la répétition de trois étapes jusqu'à stabilisation des données :

— Le calcul du diagramme de Voronoi $V(\mathcal{D})$ en se basant sur $\mathcal{D}$: chaque point $d \in \mathcal{D}$ se voit attribué la cellule $v_d \in V(\mathcal{D})$.
— Le calcul du centroïde $C(v_d)$ de chaque cellule $v_d \in V(\mathcal{D})$.
— Le déplacement des points $d \in \mathcal{D}$ vers le centroïde correspondant $C(v_d)$.

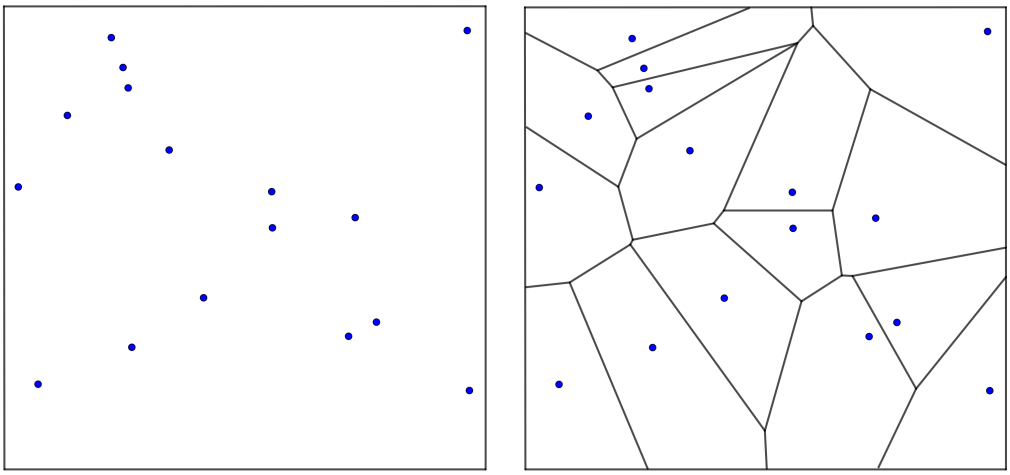
La stabilisation est atteinte lorsque la somme des distances entre les points avant leur déplacement et les centroïdes est en dessous d'un seuil ϵ :

$$\sum_{d \in \mathcal{D}} ||d - C(v_d)|| \leq \epsilon \quad \epsilon \sim 0 \tag{5.10}$$

Une itération de l'algorithme de Lloyd est présentée sur la Figure 5.4. Puisque cet algorithme consiste à calculer un diagramme de Voronoi, il faut donc que les points dans $\mathcal{D}$ aient tous des coordonnées différentes dans $\mathbb{R}^2$. Dans notre cas d'usage, si le jeu de données en entrée de VRGrid n'est pas garanti de respecter cette condition, nous pouvons contourner ce problème en déplaçant légèrement les coordonnées des points dans $\mathbb{R}^2$ dans une direction aléatoire et sur une distance très courte. Néanmoins, l'interprétation de points partageant une même position dans $\mathbb{R}^2$ (pouvant être le résultat d'une projection $\mathbb{R}^N \rightarrow \mathbb{R}^2$ effectuée en amont par exemple) n'est pas du ressort de VRGrid, puisque l'objectif de ce dernier est de produire un arrangement en grille ne contenant aucune superposition de point.

5.3.2 Arrangement de la disposition uniforme en grille

La Figure 5.5 montre la disposition des points sur l'exemple précédent lorsque l'algorithme de Lloyd est stabilisé. Cette configuration est proche de celle de la grille, mais la convergence vers une configuration en grille n'est pas garantie par l'algorithme. Dans l'article de Q. Du et al. [29], l'algorithme est appliqué à un ensemble de points disposés *presque* en grille orthogonale. Après quelques itérations, les points s'étaient éloignés de leur position initiale et la CVT ne correspond pas à une grille orthogonale mais plutôt à une grille aux cellules plus ou moins hexagonales. En plus de cela, le nombre d'itérations nécessaires avant d'atteindre une configuration stable dépend de la densité des données initiales ; plus les points sont proches les uns des autres, plus le nombre



(a) Sommets d'entrée dans $\mathbb{R}^2$

(b) Calcul du diagramme de Voronoi

(c) Calcul des centroïdes de chaque cellule

(d) Déplacement des sommets vers les centroïdes

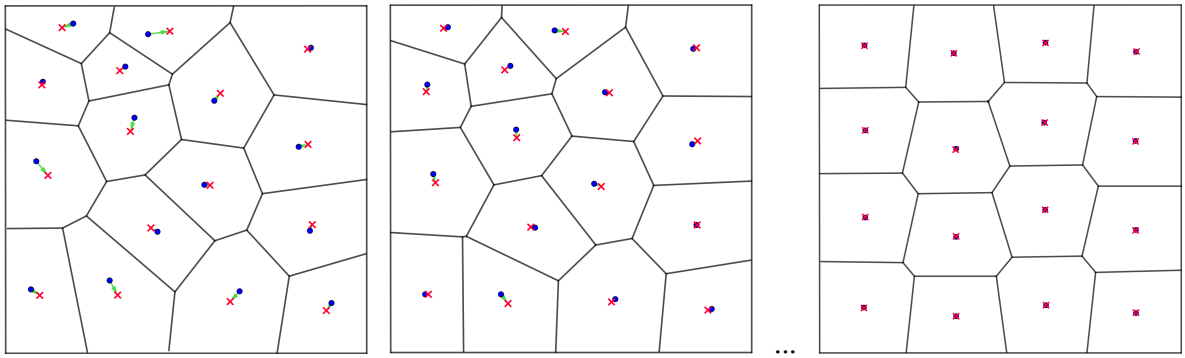
FIGURE 5.4 – Itération de l'algorithme de Lloyd sur 16 points. Les 3 étapes (b), (c) et (d) sont répétées jusqu'à stabilisation des données, c'est-à-dire que l'ensemble des déplacements calculés en étape (d) soient minimes.

d'itérations pour les répartir dans les espaces vide sera important. L'exemple en Figure 5.6 montre les différences entre un jeu de données projeté aléatoirement dans $\mathbb{R}^2$ et par t-SNE et le nombre d'itérations nécessaires avant d'atteindre une configuration stable. On y remarque aussi que passées les premières itérations, les déplacements sont bien moins conséquents alors même que la configuration stable est loin d'être atteinte.

Pour résoudre ces problèmes, nous apportons les changements suivants à l'algorithme.

Réservation de coordonnées cibles

Notre idée pour calculer un arrangement en grille et d'exploiter l'algorithme de Lloyd pour obtenir une répartition des données le plus proche d'une configuration en grille. Pour cela, nous préparons une liste de coordonnées «cibles» qui correspondent aux emplacements des cellules de la grille finale dans $\mathbb{R}^2$. Ces coordonnées sont calculées de sorte à obtenir une grille orthogonale de $(\lceil\sqrt{N}\rceil + 2) \times (\lceil\sqrt{N}\rceil + 2)$ cellules pouvant (1) accueillir les N éléments du jeu de données d'entrée, (2) un ensemble d'éléments «factices» (pour combler les cellules vides au cas où il y a plus de cellules que d'éléments à arranger), et enfin (3) un ensemble de cellules vides autour de celles mentionnées précédemment, qui serviront à délimiter notre espace de calcul.



(a) Après 2 itérations (b) Après 3 itérations ... (d) Après 19 itérations, une configuration CVT est obtenue, proche d'une grille

FIGURE 5.5 – Application répétée de l'algorithme de Lloyd sur le même exemple qu'en Figure 5.4. Les déplacements sont de plus en plus faibles au fil des itérations et résultent après stabilité en une configuration CVT qui est proche d'une configuration en grille également.

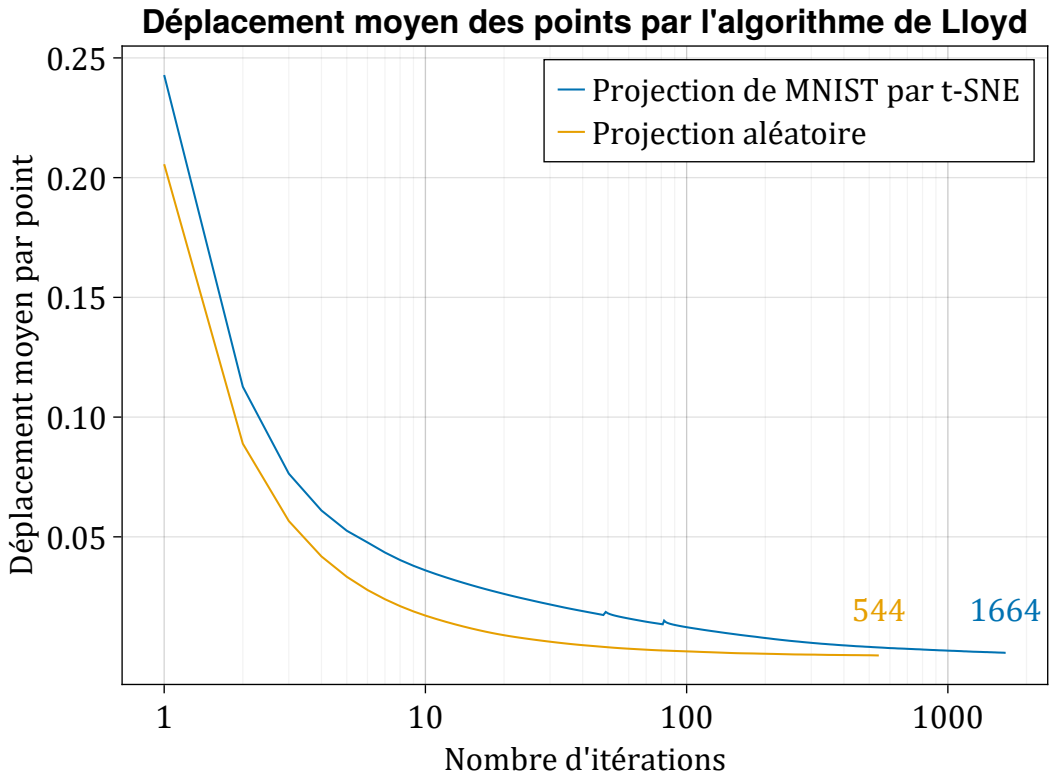


FIGURE 5.6 – Déplacement moyen des éléments dans deux jeux de données de 10 000 éléments sur chaque itération de l'algorithme de Lloyd jusqu'à stabilisation. Ces jeux de données varient en densité : le jeu de données t-SNE/MNIST (en bleu) possède des zones denses en points (groupes), tandis que les positions aléatoires (en orange) ont une densité uniforme dans $\mathbb{R}^2$. L'algorithme demande 1664 itérations pour obtenir une configuration stable sur t-SNE/MNIST alors qu'il ne demande que 544 itérations sur le jeu de données aléatoire pourtant de même taille. On peut aussi noter que les plus gros déplacements sont dans les deux cas effectués lors de la première dizaine d'itérations alors que les suivants sont bien moins significatifs.

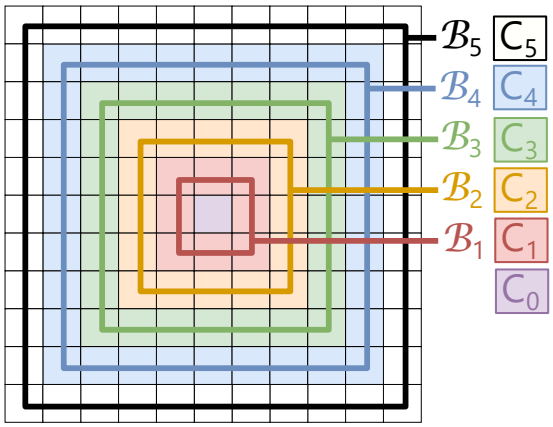


FIGURE 5.7 – Réservation des positions de la grille finale à l'aide d'enveloppes convexes.

Positionnement de points après stabilisation partielle et bordure

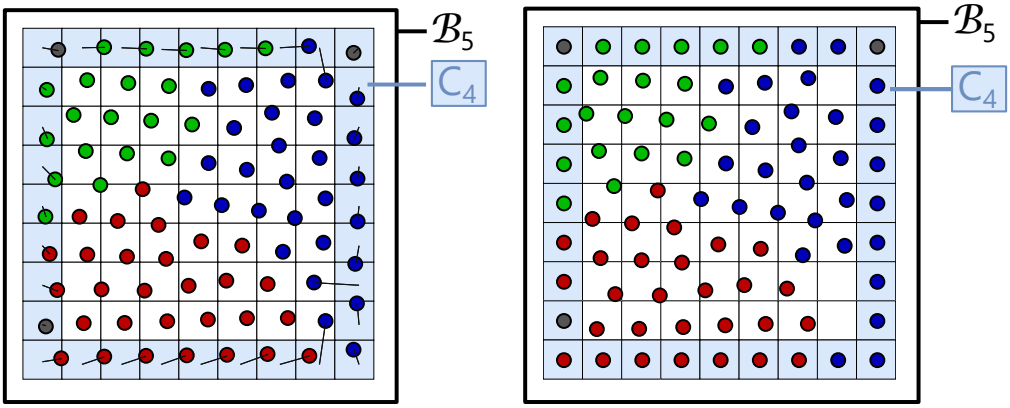
Ce changement à l'algorithme de Lloyd permet de réduire le nombre d'itérations de l'algorithme tout en forçant l'obtention d'une structure en grille dès que possible. Nous définissons une condition d'arrêt qui est soit un facteur de stabilité, soit un nombre fixe d'itérations à effectuer. Une fois cette condition atteinte, les points les plus écartés du centre de l'espace de calcul sont fixés sur les coordonnées des cellules les plus proches calculées précédemment. Tous les points ne sont pas fixés en une seule fois cependant, cette étape est réalisée progressivement en commençant par les cellules les plus à l'extérieur de la grille (en considérant leur distance de Tchebychev), pour se rapprocher progressivement de son centre. Ces cellules forment une sorte de cadre, respectant une structure en grille délimitant l'espace de calcul pour les prochaines applications de l'algorithme de Lloyd. Une fois que les cellules les plus à l'extérieur ont leurs points affectés, ceux-ci deviennent fixes et ne sont plus déplacés dans la suite de l'algorithme. Les autres points non-fixés sont à nouveau traités par l'algorithme de Lloyd jusqu'à obtenir une nouvelle configuration stable, avant d'être affectés aux cellules vides les plus à l'extérieur, et ce ainsi de suite jusqu'à que toutes les cellules soient remplies.

Concrètement, nous définissons en début d'algorithme un ensemble d'enveloppes convexes englobant n^2 coordonnées des cellules de la grille (avec $n^2 \leq |\mathcal{D}|$) en partant du centre de l'espace de calcul, comme montré en Figure 5.7. Nous nommons ces enveloppes convexes $\mathcal{B}_0, \mathcal{B}_1, \cdots, \mathcal{B}_k$ avec $k = \left\lceil \frac{\sqrt{N}}{2} \right\rceil + 1$. Chacune de ces enveloppes $\mathcal{B}_x$ consiste en une zone carrée recouvrant $w_x \times w_x$ coordonnées de cellules avec

$$w_x = \begin{cases} 2(x+1) & \text{si } \left\lceil \frac{\sqrt{N}}{2} \right\rceil + 1 \text{ est pair} \\ 2x - 1 & \text{sinon} \end{cases} \tag{5.11}$$

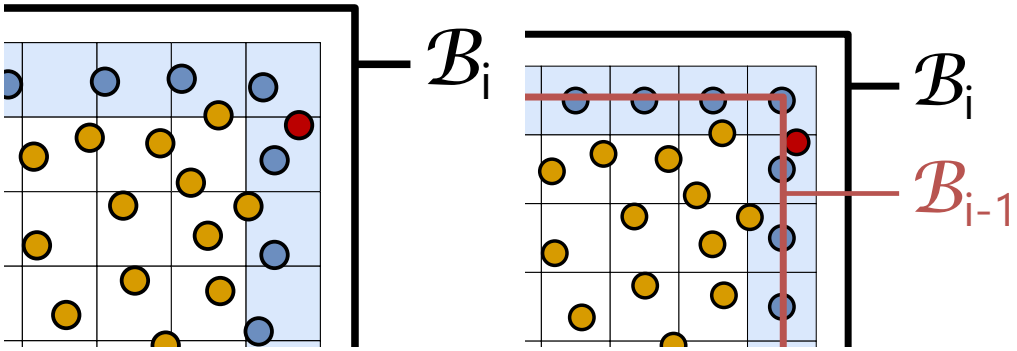
On nomme «bordure» $\mathcal{C}_x$ les cellules délimitant l'enveloppe $\mathcal{B}_x$.

A la x-ième itération de VRGrid, les points non-fixés sont relaxés par l'algorithme de Lloyd dans l'enveloppe convexe $\mathcal{B}_{k-x}$. Une fois la stabilité atteinte, les points les plus proches des coordonnées de $\mathcal{C}_{k-x}$ y sont affectés et fixés, comme illustré en Figure 5.8. Nous appliquons ensuite un changement d'échelle de centre O sur le reste des points non-fixés afin qu'ils soient contenus dans l'enveloppe convexe suivante $\mathcal{B}_{k-x-1}$, avec O étant le centre de $\mathcal{B}_k$. Ce calcul permet d'éviter que des points trop proches les uns des autres ne se retrouvent en dehors de l'enveloppe convexe $\mathcal{B}_{k-x-1}$ après affectation des positions. La Figure 5.9 illustre un de ces cas, où le point rouge se trouve à l'extérieur de l'enveloppe convexe de la prochaine itération de l'algorithme. Si tous les points ne sont pas fixés, l'algorithme passe alors à la $(x+1)$-ième itération.



(a) Configuration stable (b) Affectation des points

FIGURE 5.8 – Affectation des points à la bordure après stabilisation des points à la première itération de VRGrid.



(a) (b)

FIGURE 5.9 – Exemple de problème d'affectation aux bordures. Le point rouge se trouve trop loin des cases de $\mathcal{C}_{i-1}$ et se retrouve en dehors de $\mathcal{B}_{i-1}$ après affectation des points au début de la prochaine itération de VRGrid. Pour résoudre ce problème, nous appliquons un changement d'échelle de centre O (O étant ici le centre de $\mathcal{B}_i$) sur les points non-fixés après affectation aux bordures de manière à garantir que les points sont dans l'enveloppe $\mathcal{B}_{i-1}$ avant de démarrer l'itération de VRGrid suivante.

5.3.3 Implémentation et analyse de complexité

Le fonctionnement de VRGrid est présenté en Figure 5.10 et son algorithme en Algorithme 2. Nous étudions ici sa complexité en reprenant chaque grande étape de calcul. Cette complexité dépend notamment de la taille du jeu de données en entrée que l'on nomme N par soucis de lisibilité.

Initialisation

La phase d'initialisation de VRGrid consiste à calculer les coordonnées cibles des cellules de la grille, à préparer les enveloppes convexes à utiliser, définir les points de la bordure initiale et à appliquer un changement d'échelle sur les données d'entrées. Ces opérations n'impliquent pas de calculs complexes et ont une complexité algorithmique de $O(N)$.

Algorithme de relaxation (Lloyd)

L'algorithme de Lloyd consiste en trois étapes, (1) le calcul du diagramme de Voronoi, (2) le calcul des centroïdes des cellules résultantes, et (3) le déplacement des points vers ces nouvelles

Algorithme 2 : Implémentation détaillée de VRGrid. L'algorithme dépend d'une fonction $diagrammeVoronoi$ qui retourne pour chaque point $u \in D$ un ensemble de points V_d formant la cellule de Voronoi de u.

Entrée : Ensemble de points $\mathcal{D} \subset \mathbb{R}^2$, sans superposition, de taille N ;
Facteur de stabilité ϵ
Résultat : Ensemble de points $\mathcal{D} \subset \mathbb{R}^2$ arrangés en grille
▶ Initialisation
$k \leftarrow \lceil \sqrt{N}/2 \rceil + 1$;
si $\lceil \sqrt{N} \rceil + 1 \mod 2 = 0$ **alors**
  $\mathcal{C}_0 \leftarrow \{(-1,1),(1,1),(1,-1),(-1,-1)\}$;
sinon
  $\mathcal{C}_0 \leftarrow \{(0,0)\}$;
▶ Définition des positions des cellules
pour $i \in \{1,\dots,k\}$ **faire**
  $\mathcal{C}_i \leftarrow \{(-i,i),(i,i),(i,-i),(-i,-i)\}$;
  pour $x \leftarrow \{-i+1,\dots,i-1\}$ **faire**
    $\mathcal{C}_i \leftarrow \mathcal{C}_i \cup \{(-i,x),(i,x),(x,i),(x,-i)\}$;
$\mathcal{F} \leftarrow \mathcal{C}_k$;
$\mathcal{D} \leftarrow replacer(\mathcal{D}, k)$;
▶ Boucle principale
$iter \leftarrow 1$;
tant que $|\mathcal{D} \backslash \mathcal{F}| > 0$ **faire**
  tant que $\neg stable$ **faire**
    $d \leftarrow 0$;
    $V \leftarrow diagrammeVoronoi(\mathcal{D})$;
    pour $u \in \mathcal{D}$ **faire**
      $c_u \leftarrow calculCentroid(V_u)$;
      ▶ d vérifie si la configuration est stable, donc peut être calculé différement si besoin.
      $d \leftarrow d + \|u - c_u\|$;
      $u \leftarrow c_u$;
      $eltDehors \leftarrow eltDehors \vee \neg estDans(\mathcal{C}_{iter}, u)$;
    si $eltDehors$ **alors**
      $\mathcal{D} \leftarrow \mathcal{F} \cup replacer(\mathcal{D} \backslash \mathcal{F}, k)$;
    $stable \leftarrow d \leq \epsilon$;
  ▶ Atribution des coordonnées de la grille
  $qt \leftarrow quadtree(\mathcal{D} \backslash \mathcal{F})$;
  pour $c \in \mathcal{C}_{k-iter}$ **faire**
    $tgt \leftarrow plusProche(qt, c)$;
    $\mathcal{D}[tgt] \leftarrow c$;
    $\mathcal{F} \leftarrow \mathcal{F} \cup \{\mathcal{D}[tgt]\}$;
  $iter \leftarrow iter + 1$;

coordonnées. Ces opérations sont répétées un certain nombre de fois jusqu'à stabilisation. Ce nombre d'itérations est difficilement estimable car il dépend de la taille de l'ensemble de points, de leur disposition dans $\mathbb{R}^2$ (et notamment de leur densité) et enfin du facteur de stabilisation choisi pour l'algorithme.

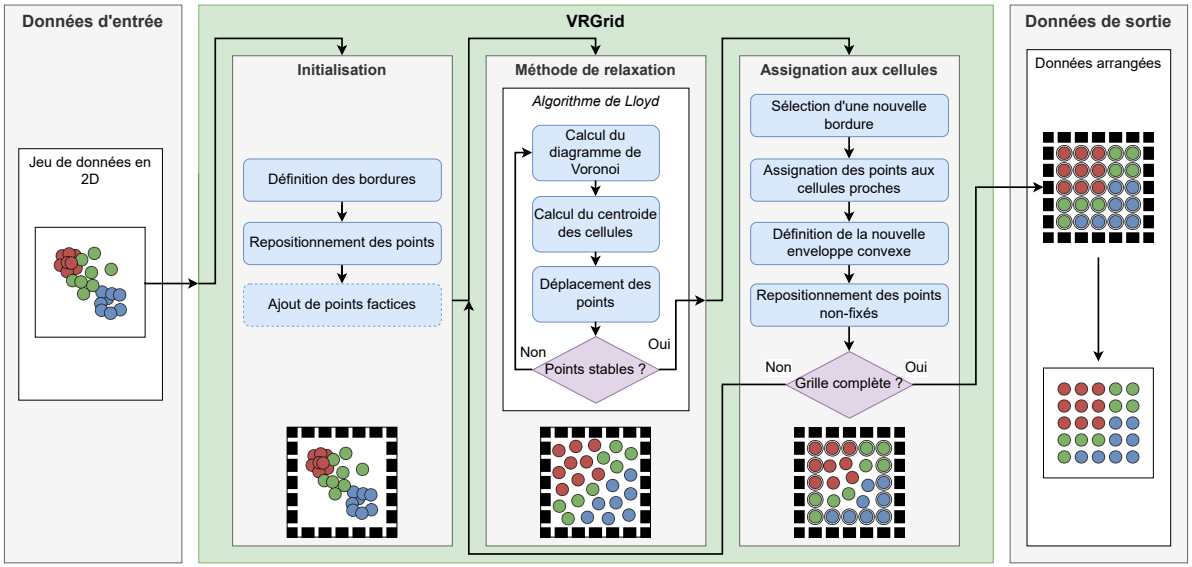


FIGURE 5.10 – Fonctionnement de VRGrid par étape.

Afin d'être efficace, notre implémentation de l'algorithme de Lloyd repose sur la complémentarité du diagramme de Voronoi avec la triangulation de Delaunay : les centres des cercles circonscrits de deux triangles voisins (partageant un côté en commun) issus de la triangulation de Delaunay peuvent être reliés pour former les contours des cellules du diagramme de Voronoi [34], comme montré en Figure 5.11.

(1) Calcul du diagramme de Voronoi. Plusieurs algorithmes existent pour calculer ce diagramme. On peut notamment citer l'algorithme de Fortune [33] qui a une complexité algorithmique de $O(N.\log(N))$. Cependant, afin de rendre moins complexes les étapes suivantes du calcul, nous utilisons à la place la construction de la triangulation de Delaunay par insertion successive de points [34] afin de calculer le diagramme de Voronoi étant donné que les deux constructions sont complémentaires. Cette construction par insertion successive de points étant de complexité $O(N.\log(N))$, la complexité de cette étape l'est tout autant.

(2) Calcul des centroïdes des cellules du diagramme. Le calcul des centroïdes se fait en complexité linéaire en parcourant les arêtes du graphe formé par la construction du diagramme de Voronoi. Ces arêtes sont parcourues au maximum deux fois pour calculer le centroïde des cellules qu'elles délimitent. Il y a donc au maximum un parcours de $2E_V$ arêtes pour calculer les centroïdes, avec E_V le nombre d'arêtes dans le diagramme de Voronoi. Par la complémentarité du diagramme de Voronoi avec la triangulation de Delaunay, nous savons que E_V est égal au nombre d'arêtes E_D dans la triangulation de Delaunay. Nous savons aussi que la triangulation de Delaunay forme un graphe planaire, ce qui limite le nombre d'arêtes E_D à $3V_D - 6$ avec V_D le nombre de sommets de ce graphe. Comme les sommets de ce graphe correspondent aux N points utilisés pour calculer le diagramme de Voronoi, nous pouvons donc limiter le nombre d'arêtes à parcourir à $2(3N - 6)$, ce qui nous donne une complexité de $O(N)$.

(3) Déplacement des points vers les centroïdes. L'étape (3) est de complexité $O(N)$ puisqu'il s'agit uniquement d'affecter une nouvelle position aux N points du jeu de données.

Nombre d'itérations de l'algorithme de Lloyd. Le nombre itérations nécessaire afin d'atteindre une configuration stable dépend de multiples facteurs concernant le jeu de données. Le positionnement initial des points, la densité des groupes de points et la façon de calculer la stabilité de la configuration jouent un rôle dans le nombre d'itérations nécessaire à l'algorithme de Lloyd. Nous pouvons définir un nombre maximum d'itérations i avant d'assigner les points pour simplifier le calcul de la complexité. Cette étape de calcul de VRGrid est donc de complexité $O(i.N.\log(N))$.

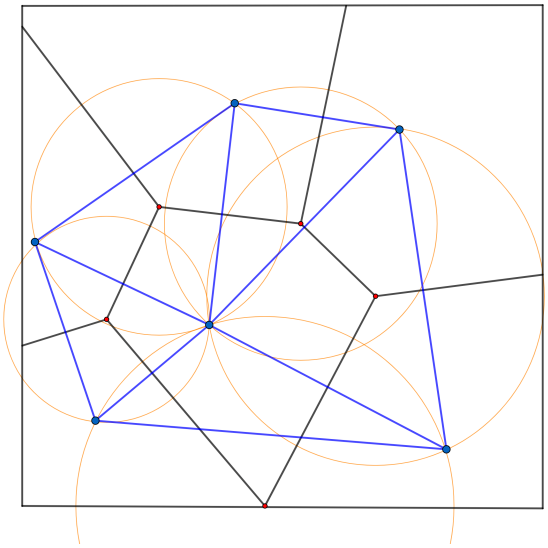


FIGURE 5.11 – Relation de dualité entre le diagramme de Voronoi (en noir) constitué à partir de sommets (en bleu), dont les points aux intersections des cellules (en rouge) correspondent aux centres des cercles circonscrits (en orange) des triangles issus de la triangulation de Delaunay des mêmes sommets.

Assignation des points aux cellules préparées

L'assignation des points aux cellules en bordures consiste à trouver les éléments non-fixés qui sont les plus proches de chacune des coordonnées en bordure préparées pour l'itération actuelle. Nous employons une structure de *quad-tree* pour trouver efficacement ces éléments avec une complexité de $O(\log(N))$ par requête, donc de $O(N.\log(N))$ pour l'ensemble des coordonnées de la bordure actuelle. Pour ce qui est de l'affectation des points aux coordonnées prévues et du calcul du changement d'échelle sur les points restants, la complexité reste linéaire.

Nombre d'itérations de l'algorithme

VRGrid doit appliquer l'algorithme de Lloyd et l'affectation des points aux coordonnées de la bordure pour chacune des enveloppes convexes préparées lors de l'étape d'initialisation. Le nombre d'enveloppes convexes étant défini à $\lceil\sqrt{N}/2\rceil + 1$, la complexité totale de VRGrid est donc de $O(\sqrt{N}.i.N.\log(N))$, soit $O(i.N^{1.5}.\log(N))$.

5.4 Performances de VRGrid et des méthodes existantes

VRGrid a été implémentée en C++ sur une base logicielle optimisée pour le dessin de graphes complexes inspirée par Tulip [7]. Cette base logicielle nous permet de faciliter les constructions de la triangulation de Delaunay ainsi que la recherche de sommets nécessaires à l'exécution de VRGrid. Cette implémentation a été utilisée pour comparer VRGrid avec les autres méthodes de l'état de l'art.

La complexité de notre méthode varie suivant le nombre d'itérations de l'algorithme de Lloyd ; nous évaluons VRGrid sous deux configurations que l'on nomme VRGrid_P et VRGrid_F :

— VRGrid_P est la configuration privilégiant la qualité de l'arrangement à la vitesse d'exécution. La condition d'arrêt est basée sur les déplacements des points entre deux itérations, et les points sont affectés aux bordures dès lors que ces déplacements d sont très faibles ($d \leq 0.01$ lorsqu'une cellule de la grille est de taille 1×1).

— VRGrid_F est la configuration privilégiant la vitesse d'exécution à la qualité de l'arrangement. En se basant sur l'observation relevée en Section 5.3.2 où les déplacements de points

sont significativement plus petits après une dizaine d'itérations, nous limitons le nombre d'itérations à 10 avant d'affecter les points aux bordures.

5.4.1 Jeux de données testés

Afin d'évaluer les méthodes d'arrangements, nous avons besoin de jeux de données variant en nature, en taille et en topologie. Cette diversité dans les jeux de données nous permet de mesurer la capacité des méthodes à supporter des cas plus ou moins communs de projections de données. Nous utilisons donc des jeux de données provenant de plusieurs sources :

— Un ensemble de jeux de données tirés du *Matlab Toolbox for Dimensionality Reduction* [137] qui sont de petites tailles (1064 éléments par jeu de données) et dans $\mathbb{R}^3$, mais qui ont des topologies uniques et interprétables par l'humain. L'utilisation de méthodes de projections $\mathbb{R}^N \rightarrow \mathbb{R}^2$ sur ces jeux de données permet de comparer notamment l'interprétation de la visualisation entre leur topologie et la forme de leur projection. Dans ce chapitre, nous utiliserons les jeux de données Swissroll, Broken Swissroll, Helix et Twinpeaks, qui sont présentés en Figure 5.12. Ces jeux de données ont notamment été utilisés par les auteurs de SSM pour mesurer les performances de leur méthode [128].

— Deux «grands» jeux de données dans $mathbbR^{28\times28}$ et fréquemment utilisés dans le domaine de l'apprentissage machine (*machine learning*), qui sont MNIST [72] et Fashion-MNIST [150], déjà présentés dans les chapitres précédents. Ces jeux de données sont souvent utilisés par des méthodes de projections de données, telles que t-SNE [136], UMAP [82] ou IsoMap [133], pour démontrer leur capacité à projeter des jeux de données complexes. Nous pouvons donc les inclure dans notre évaluation pour étudier des scénarios d'applications d'arrangement en grilles de données complexes déjà projetées dans $\mathbb{R}^2$. L'utilisation de ces jeux de données a pour objectif de créer des scénarios des cas d'usages proches de ce qui peut se faire en matière de visualisation d'information en termes de taille et de données à arranger.

Ces jeux de données sont projetés dans $\mathbb{R}^2$ avant d'être arrangés en grilles. Les méthodes utilisées sont t-SNE [136] et UMAP [82], particulièrement efficaces pour produire un partitionnement de données (*clusters*), ainsi que MDS [67] et IsoMap [133] permettant d'avoir des projections plus éparses.

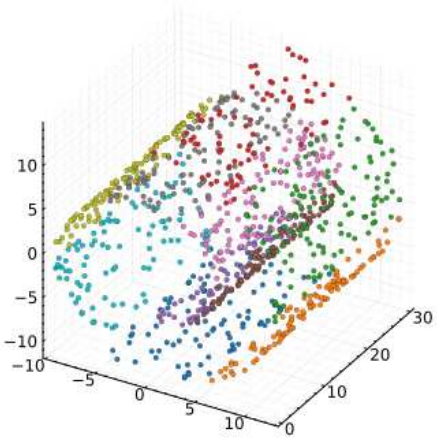
Pour étudier la stabilité des méthodes, nous employons une troisième catégorie de données :

— 800 jeux de données générés aléatoirement dans $\mathbb{R}^2$. Ces jeux de données sont répartis équitablement en différentes combinaisons faisant varier leur taille de 64, 100, 1 000 ou 10 000 éléments. Pour chacune de ces tailles, nous générons 100 jeux de données en utilisant une méthode de distribution avec une probabilité égale en utilisant *Permuted Congruential Generator* [100] (PCG), et 100 autres en suivant une loi de distribution normale pour avoir des données un peu plus denses. Ces différents paramètres nous permettent de tester le comportement des méthodes sur des jeux de données plus génériques, mais variant en termes de densité. Comme ces derniers jeux de données sont générés aléatoirement en $\mathbb{R}^2$, ils n'ont pas besoin d'être projetés comme les précédents et ont des topologies bien plus simplifiées.

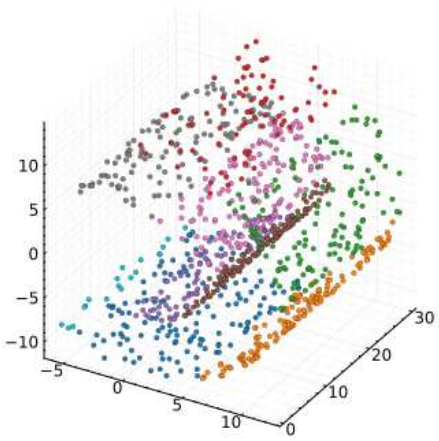
Dans notre étude, nous faisons la distinction entre les jeux de données projetés qu'on appellera **données réelles** et les jeux de données générés qu'on appellera **données aléatoires**.

5.4.2 Métriques d'évaluation

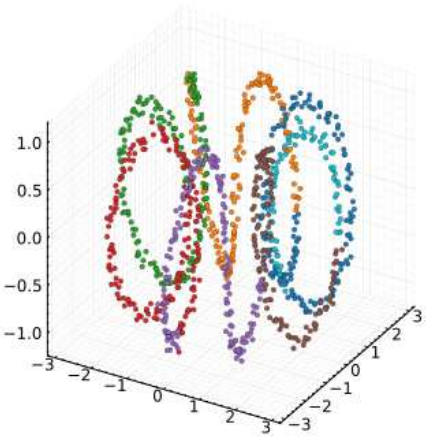
Les formules d'évaluation sont les mêmes que celles présentées en Section 5.2. Nous faisons la distinction entre les données réelles et celles aléatoires, mais l'uniformité des données aléatoires nous permet de procéder en plus à des tests statistiques afin de déterminer si les différences de performances entre les méthodes sont significatives ou non. Ces tests statistiques sont réalisés avec la méthode de Wilcoxon-Mann-Whitney [146] vérifiant l'hypothèse que la distribution des



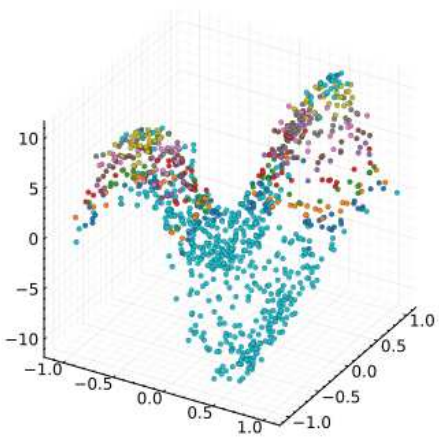
(a) Swissroll



(b) Broken Swissroll



(c) Helix



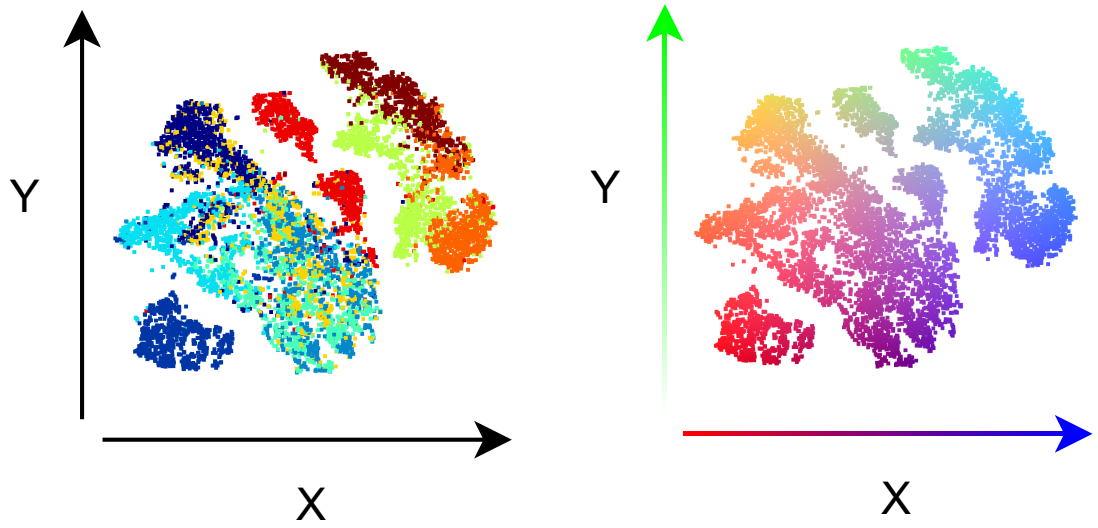
(d) Twinpeaks

FIGURE 5.12 – Projection des jeux de données tirés du *Matlab Toolbox for Dimensionality Reduction*. Les points sont colorés suivant leur classe d'équivalence définie par l'algorithme de génération.

valeurs dans deux échantillons est identique. Ici, nous vérifions cette hypothèse sur les performances individuelles de chaque configurations de VRGrid contre SSM et DGrid. Si les tests effectués par cette étude relèvent des valeurs inférieures à 5×10^{-2}, alors les performances sont significativement différentes.

5.4.3 Comparaison des arrangements calculés

Les données réelles sont accompagnées d'étiquettes pour chaque élément : ils appartiennent à une classe d'équivalence parmi dix. La méthode de coloration habituelle en visualisation d'information consiste à associer une couleur à chaque classe, et de colorer les éléments à l'écran en fonction de leur classe. Ici, nous nous intéressons aussi aux déplacements potentiellement non-uniformes des données entre leur positionnement initial dans $\mathbb{R}^2$ et leur position une fois dans la grille. La coloration par étiquette ne nous permet pas d'identifier les déplacements des éléments



(a) Coloration par étiquette

(b) Coloration par position dans $\mathbb{R}^2$

FIGURE 5.13 – Coloration des éléments par leur étiquette (a) et par leur positionnement dans $\mathbb{R}^2$ (b). La coloration par positionnement permet notamment de différentier les éléments au sein d'un même groupe d'équivalence (par exemple, les deux ensembles de points rouges en (a) sont colorés différemment en (b)).

individuellement, seulement si ceux-ci se trouvent dans un groupe d'éléments appartenant à une autre classe d'équivalence. Nous proposons une coloration des éléments complémentaire à la première pour observer les déplacements de chaque élément individuellement. Celle-ci consiste à colorer les éléments par rapport à leur position initiale dans $\mathbb{R}^2$: les éléments sont colorés avec une teinte de rouge ou de bleu s'ils sont placés respectivement sur la gauche ou la droite de l'espace de visualisation, et une teinte de vert est ajoutée en fonction de leur position verticale. Un exemple de cette coloration est donné en Figure 5.13. Formellement, la coloration Rouge-Vert-Bleu d'un élément est déterminée par la fonction suivante :

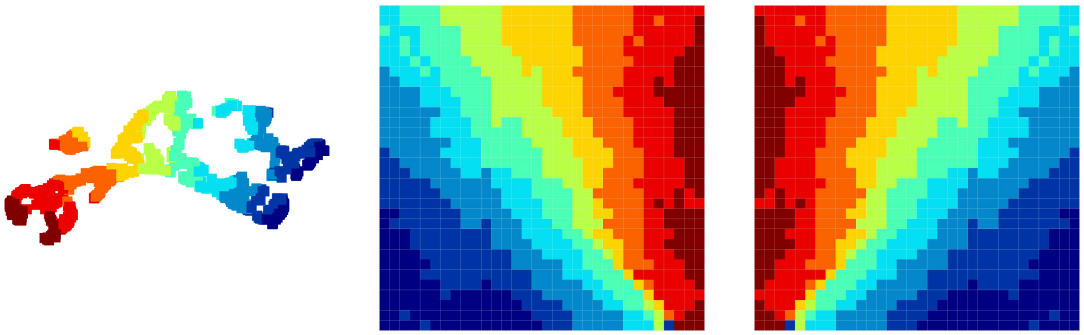
$$couleur_{R,V,B}(x, y) = (1 - x, y, x) \tag{5.12}$$

avec x et y les positions d'origine sur respectivement l'axe horizontal et l'axe vertical normalisées entre 0 et 1. Cette formule applique donc un dégradé de couleurs sur les éléments à proximité les uns des autres sur leur positionnement initial dans $\mathbb{R}^2$. Cette coloration permet d'avoir un aperçu rapide de la déformation des données apportée par les méthodes d'arrangement en grille : une petite déformation n'aura que peu de discontinuités dans les couleurs du dégradé affiché sur la grille, tandis qu'une grande déformation aura un plus grand impact visuel sur le dégradé. Cette coloration permet aussi de mettre en évidence des incohérences qui sont détectées difficilement dans la coloration habituelle et par les métriques d'évaluation : par exemple un élément mal positionné au sein de son groupe de classe d'équivalence aura une couleur très différente de ses voisins.

5.4.4 Implémentation des méthodes de l'état de l'art

Lors de la rédaction de ce manuscrit, les implémentations de SSM et de DGrid n'étaient pas disponibles auprès des auteurs. Néanmoins, l'algorithme SSM est présenté dans l'article de Strong et al. [128], et celui de DGrid dans la thèse de Mamami [81], ce qui nous a permis de les implémenter.

Il faut noter que l'utilisation de SSM dans ce chapitre n'est pas exactement ce pourquoi il a été conçu. En effet, celui-ci est conçu à l'origine pour arranger des éléments de $\mathbb{R}^N$ vers une structure



(a) Donnée d'entrée (b) Arrangement de SSM (c) Arrangement de SSM retourné sur l'axe X

FIGURE 5.14 – Exemple de correction apportée aux résultats de SSM, pour la combinaison UMAP/Swissroll (a). SSM produit un arrangement (b) qui n'est pas aligné avec la projection initiale, comme on le voit avec les éléments rouges sur la droite de la grille alors qu'ils sont sur la gauche de l'image en entrée. Pour cet arrangement, la valeur GPP est de 0.38. Si l'on retourne la grille le long de l'axe vertical comme montré en (c), les éléments sont mieux alignés avec les données d'entrée, et cette configuration obtient une meilleure valeur GPP de 0.08.

dans $\mathbb{R}^2$ (par exemple, une grille) en s'appuyant uniquement sur une mesure de distance dans $\mathbb{R}^N$ entre les éléments. Le positionnement initial des données dans $\mathbb{R}^N$ n'est quant à lui pas conservé lors du calcul de l'arrangement. Ce comportement permet d'éloigner ou rapprocher les éléments correctement les uns des autres dans la grille, mais dans le cas où $\mathbb{R}^{N=2}$, il ne permet pas de garantir que le positionnement initial des éléments dans $\mathbb{R}^2$ soit reflété dans la grille. Ainsi, plusieurs exécutions de l'algorithme sur un même jeu de données peut produire des arrangements orientés différemment. Ce comportement est très pénalisé par les métriques d'évaluation sur le positionnement relatif et global, malgré le fait que le souci ne relève que de l'orientation de l'arrangement. Afin d'obtenir une évaluation équilibrée entre les méthodes, nous réorientons les arrangements proposés par SSM, comme montré en Figure 5.14. Cette réorientation consiste à pivoter les arrangements produits de 90°, 180° ou 270°, et d'inverser les positions des cellules horizontalement ou verticalement, de façon à obtenir la configuration ayant la meilleure valeur sur la préservation de positionnement global. Le meilleur arrangement réorienté est ensuite évalué contre les autres méthodes.

Malgré sa complexité algorithmique élevée, nous avons aussi procédé à des tests avec IsoMatch dont l'implémentation est disponible à l'adresse `github.com/ohadf/isomatch`. Cependant, le programme a cessé de fonctionner après plus d'une dizaine d'heures de traitement sur les jeux de données MNIST et Fashion-MNIST : nous ne l'avons donc pas inclus dans notre évaluation.

5.5 Résultats

Un échantillon de résultats des arrangements est présenté en Figure 5.15 avec la coloration par étiquette, et en Figure 5.16 avec la coloration par position. La Table 5.1 donne les résultats des tests statistiques portant sur la significativité des comparaisons de performances entre les méthodes. Enfin, les résultats de l'évaluations sont séparées par métrique ainsi qu'entre les données réelles et les données aléatoires. Ces résultats sont discutés dans chacunes des sous-sections qui suivent.

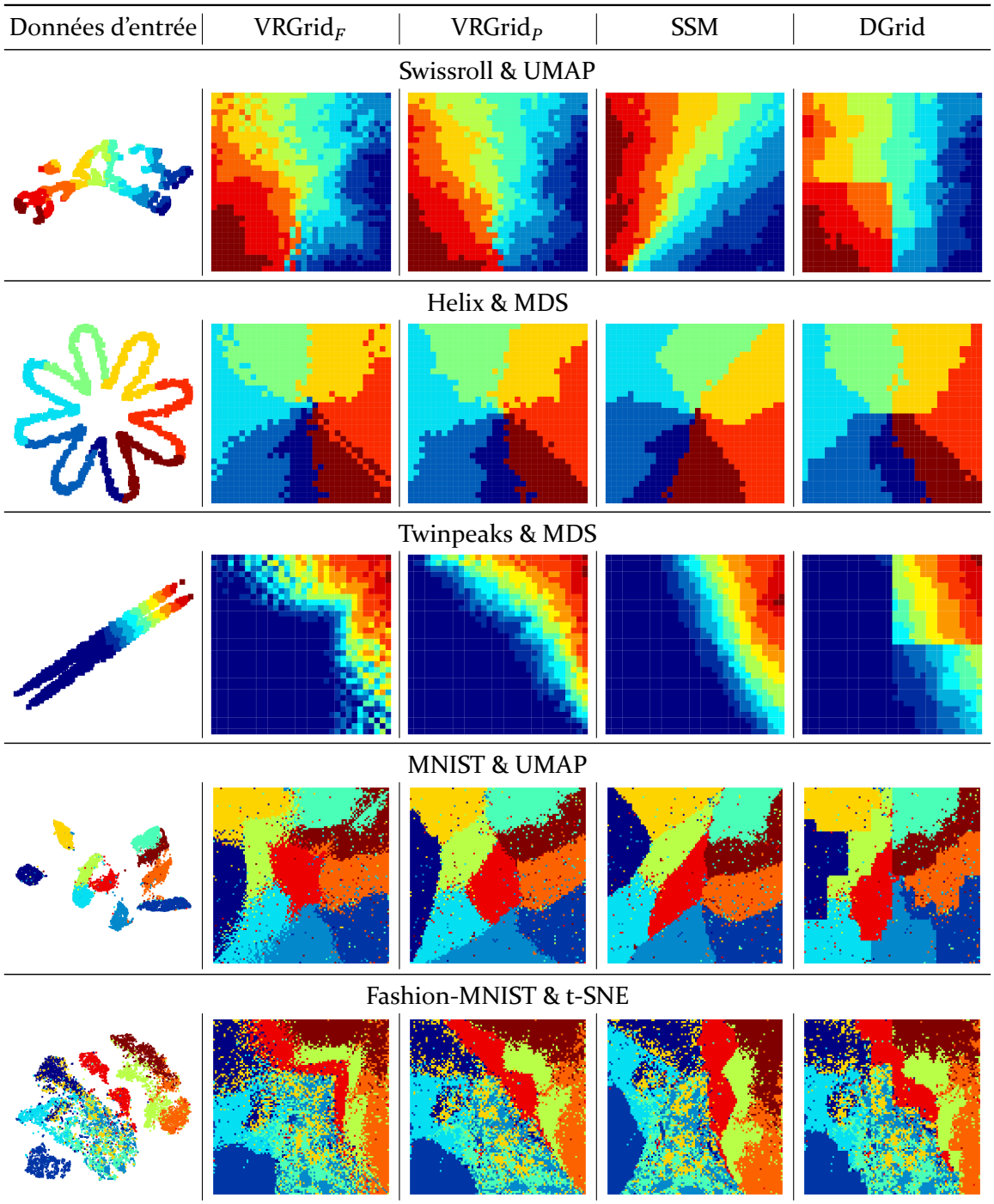


FIGURE 5.15 – Extrait des résultats obtenus sur différentes méthodes testées sur Swissroll, Helix, Twinpeaks, MNIST ou Fashion-MNIST, projetés en $\mathbb{R}^2$ (colonne de gauche) avec UMAP, MDS et t-SNE. Dans cette figure, nous utilisons la coloration par étiquette. Les jeux de données d'entrée sont reconnaissables sur les arrangements produits, mais on peut remarquer des particularités sur chaque méthode : VRGrid$_F$ a tendance à avoir des contours de classes bruités par rapport aux autres méthodes, voire provoquer des distorsions (4e et 5e lignes). VRGrid$_P$ obtient des résultats bien plus cohérents, mais dont les bordures entre classes sont légèrement moins nettes que celles des arrangements produits par SSM. Cependant, ce dernier a tendance à déformer globalement le positionnement initial des points, mais la donnée d'entrée reste amplement reconnaissable. Enfin, les arrangements de DGrid ont de grandes distorsions sur certains scénarios (1ère et 3e lignes), provoquant un rendu qui semble discontinu et fragmenté.

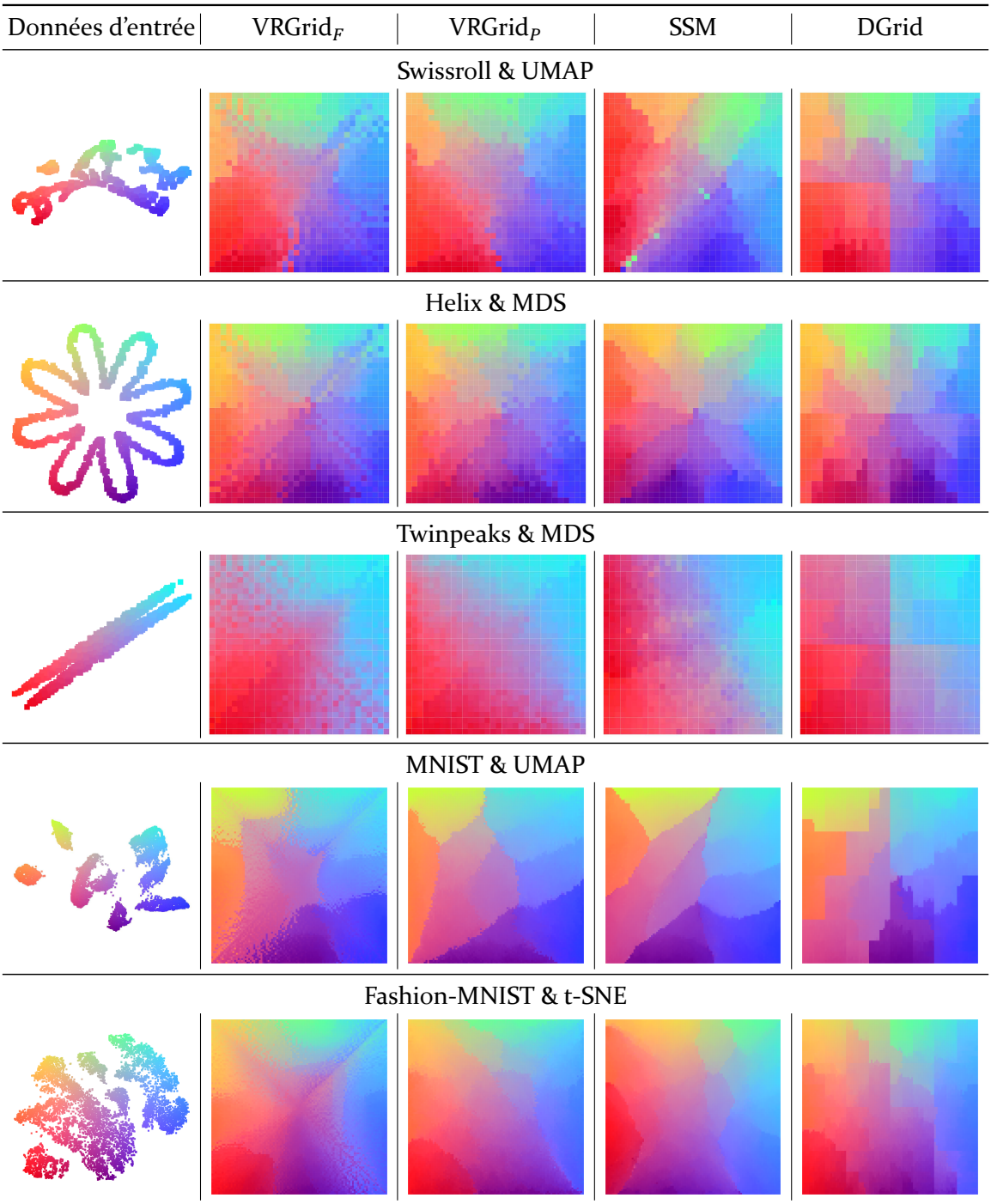
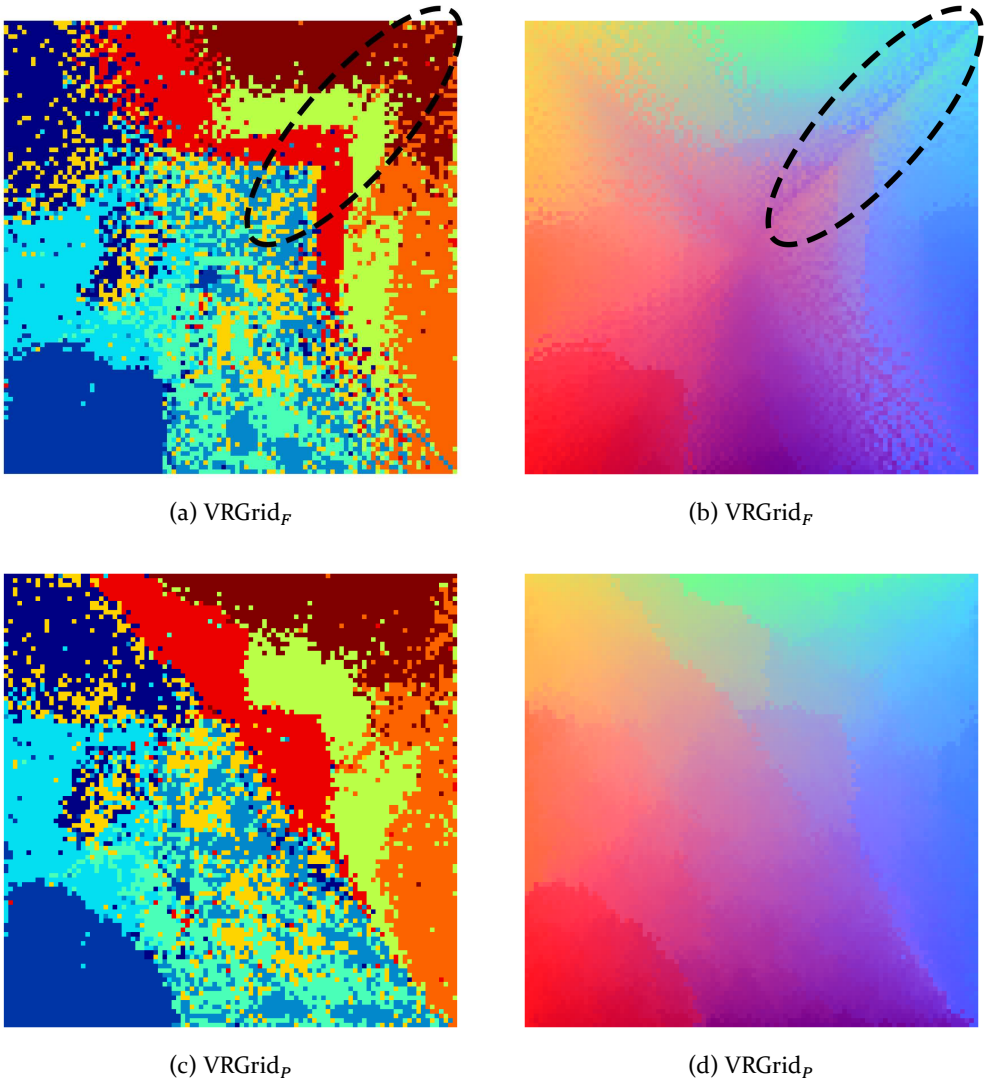


FIGURE 5.16 – Même résultats qu'en Figure 5.15 mais en utilisant la coloration par position initiale. Nous pouvons retrouver les observations faites précédemment, mais nous pouvons en plus remarquer quelques incohérences dans les arrangements proposés par SSM, notamment sur la première ligne où des points (en vert clair) sont positionnés loin du reste des éléments de leur groupe (en haut à droite). On remarque aussi que malgré les bordures moins nettes de VRGrid$_P$avec la coloration précédente, les éléments au sein d'un même groupe restent positionnés de manière cohérente.

5.5.1 Étude Visuelle



(a) VRGrid$_F$ (b) VRGrid$_F$

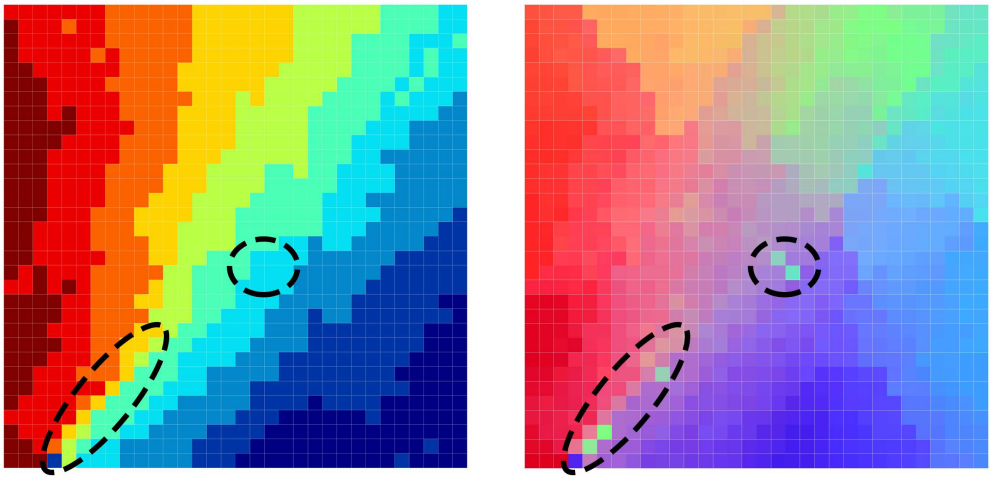
(c) VRGrid$_P$ (d) VRGrid$_P$

FIGURE 5.17 – Résultats des méthodes VRGrid$_F$ et VRGrid$_P$ sur la projection t-SNE de Fashion-MNIST. VRGrid$_F$ produit une déformation au niveau des diagonales de la grille, comportement qui n'est pas présent chez VRGrid$_P$.

La topologie initiale des données est reconnaissable sur les arrangements produits par VRGrid$_F$ et VRGrid$_P$. On peut cependant remarquer une distorsion sur les rendus de VRGrid$_F$ au niveau des diagonales, mis en évidence sur la Figure 5.17, qui n'est pas présente sur les rendus de VRGrid$_P$. Une autre différence notoire entre les deux rendus est la netteté des séparations des groupes, VRGrid$_F$ présente un mélange entre les éléments qui est bien moins présent dans les rendus de VRGrid$_P$.

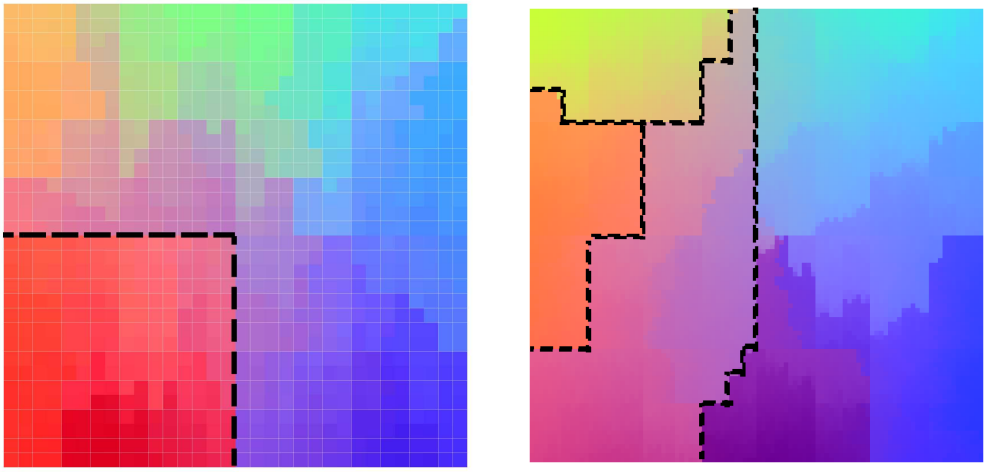
Sur les arrangements de SSM, on peut remarquer une légère déformation de la forme des données d'entrée, mais elle reste globalement très reconnaissable. À certains endroits, indiqués en Figure 5.18 sur les rendus colorés par position, on peut apercevoir des points de couleur très différente de leurs voisins, il s'agit d'éléments très éloignés de leur position de départ. Ces points sont difficilement détectables dans la coloration par étiquette. Ce comportement n'est pas présent à chaque application de SSM sur ces mêmes jeux de données, on peut supposer que la nature aléatoire de l'initialisation de l'algorithme y joue un rôle.

Sur les arrangements de DGrid, on peut voir la formation d'un quadrillage très prononcé et une forte discontinuité des données sur certains scénario, relevés en Figure 5.19.



(a) (b)

FIGURE 5.18 – Résultats de la méthode SSM sur la projection UMAP de Swissroll. On observe dans la vue utilisant la coloration par position des points mal placés, qui sont pratiquement indétectables sur la vue utilisant la coloration par étiquette.



(a) Swissroll (b) MNIST

FIGURE 5.19 – Résultats de la méthode DGrid sur les projections t-SNE de Swissroll (à gauche) et UMAP de MNIST (à droite). On remarque une discontinuité très prononcée des données et donne l'impression d'avoir une image fragmentée. Cette discontinuité est très visible sur les deux colorations, mais la topologie initiale des données est reconnaissable sur la majorité des scénarios testés.

5.5.2 Significativité des différences mesurées

Toutes les valeurs testées statistiquement par le test de Wilcoxon-Mann-Whitney [146] sont inférieures à 5×10^{-2} et indiquent donc que toutes les différences de performances entre VRGrid et les autres méthodes sont significatives. Ce test ne permet pas de comparer quelle performance est meilleure qu'une autre, nous devons donc les comparer séparément.

5.5.3 Préservation des distances (DP)

Sur les jeux de données réels, VRGrid_P obtient les meilleurs résultats avec des valeurs en moyenne 2.57% supérieures à SSM et 2.61% supérieures à DGrid. VRGrid_P produit des résultats

Méthode	Métrique d'évaluation	contre SSM	contre DGrid
VRGrid_P	DP	3.52×10^{-56}	6.51×10^{-67}
	NP	1.03×10^{-83}	7.20×10^{-34}
	RPP	6.14×10^{-125}	1.37×10^{-27}
	GPP	2.55×10^{-125}	4.20×10^{-43}
VRGrid_F	DP	2.00×10^{-4}	1.66×10^{-28}
	NP	7.71×10^{-114}	1.93×10^{-6}
	RPP	1.70×10^{-89}	6.17×10^{-19}
	GPP	2.01×10^{-104}	3.92×10^{-7}

TABLE 5.1 – p-valeurs du test statistique de Wilcoxon sur VRGrid_P et VRGrid_F contre SSM et DGrid sur les scénarios concernant les jeux de données aléatoires. Les valeurs inférieures à 5×10^{-2} indiquent que les mesures effectuées sont significativement différentes et peuvent donc être comparées.

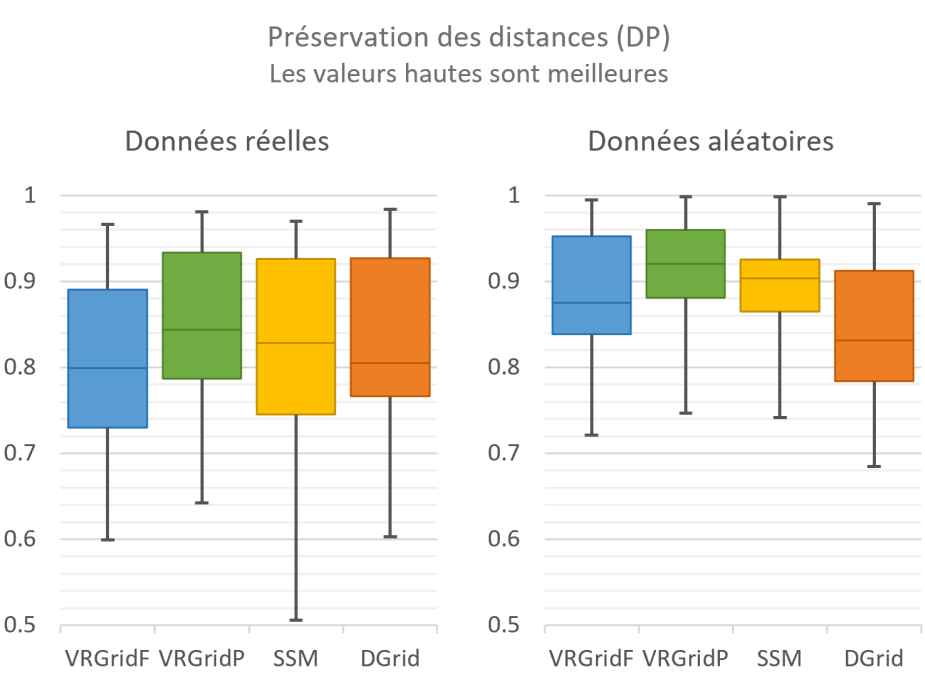


FIGURE 5.20 – Résultats agrégés des méthodes concernant la préservation des distances sur les jeux de données réels (à gauche) et aléatoires (à droite). Les valeurs hautes sont meilleures.

bien meilleurs que ceux de VRGrid_F avec un écart de 5.41%. VRGrid_P obtient aussi les meilleurs résultats sur les jeux de données aléatoires, en étant 1.77% meilleur que SSM, 6.73% que DGrid et 2.63% que VRGrid_F. Les quatre méthodes parviennent plutôt bien à conserver les distances entre les données d'entrée et de sortie, avec à leur tête VRGrid_P.

5.5.4 Préservation du voisinage (KNP)

Avec des données réelles, DGrid produit les meilleurs arrangements avec un écart de 8.94% avec ceux de SSM, 19.44% avec VRGrid_P et 37.52% avec VRGrid_F. Les méthodes sont ordonnées différemment sur les données aléatoires, avec SSM en tête avec des résultats meilleurs de 4.76% face à VRGrid_P, 12.16% avec DGrid et 16.19% avec VRGrid_F.

Sur ces résultats, VRGrid_P est devancé par SSM sur les deux familles de données, et VRGrid_F obtient les moins bons résultats. En ce qui concerne les performances de DGrid sur les jeux de

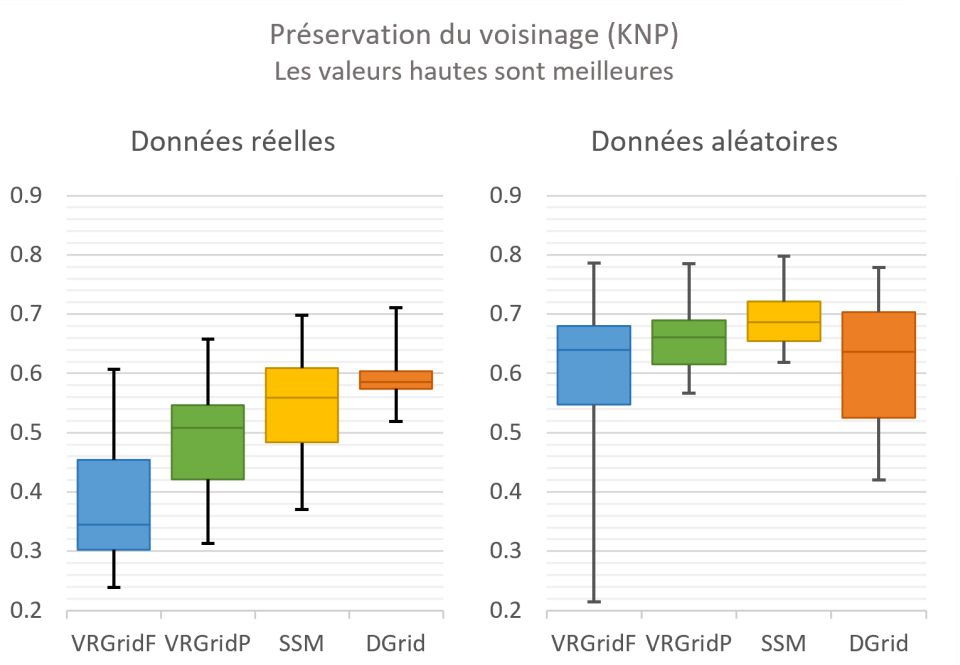


FIGURE 5.21 – Résultats agrégés des méthodes concernant la préservation du voisinage sur les jeux de données réels (à gauche) et aléatoires (à droite). Les valeurs hautes sont meilleures.

données aléatoires, nous avons remarqué que la méthode obtient de moins bons résultats sur les jeux de données dont la taille de la grille résultante n'est pas une puissance de 2. Cependant, la méthode obtient des résultats corrects sur les jeux de données MNIST et Fashion-MNIST (grilles de tailles 100×100), ce comportement demande donc une étude plus approfondie pour en trouver la cause exacte.

Pour ce qui est des performances en général, nous pouvons attribuer les moins bons résultats de VRGrid frontières moins nettes relevées lors de l'étude visuelle. En revanche, la discontinuité des données sur les rendus de DGrid ne semble pas impacter les performances mesurées sur la méthode ici, ce qui peut amener à reconsidérer la pertinence de cette métrique dans notre évaluation.

5.5.5 Préservation du positionnement relatif (RPP)

Sur les jeux de données réels, VRGrid_P obtient les meilleurs arrangements avec des performances en moyenne 44.22% meilleures que SSM, 6.53% meilleures que DGrid et 29.04% meilleures que VRGrid_F. Les écarts s'agrandissent sur les jeux de données aléatoires, où les rendus de VRGrid_P sont 62.04% meilleurs que SSM, 15.01% que DGrid et 34.04% que VRGrid_F.

Les mauvais résultats de SSM étaient plus ou moins attendus sur cette métrique puisque la méthode n'a pas de rétention du positionnement initial des données, malgré le correctif appliqué expliqué en Section 5.4.4. On peut ainsi observer que les rendus de SSM sont légèrement «tournés» par rapport aux positionnements initiaux et arrangements de VRGrid, ce qui est pénalisé par cette métrique d'évaluation. Pour ce qui est des performances de VRGrid_P et VRGrid_F, on peut voir que l'utilisation d'une méthode de relaxation de Voronoi est plutôt efficace à préserver l'ordonnancement des éléments, mais aussi que le facteur de stabilité est important pour obtenir de bons résultats.

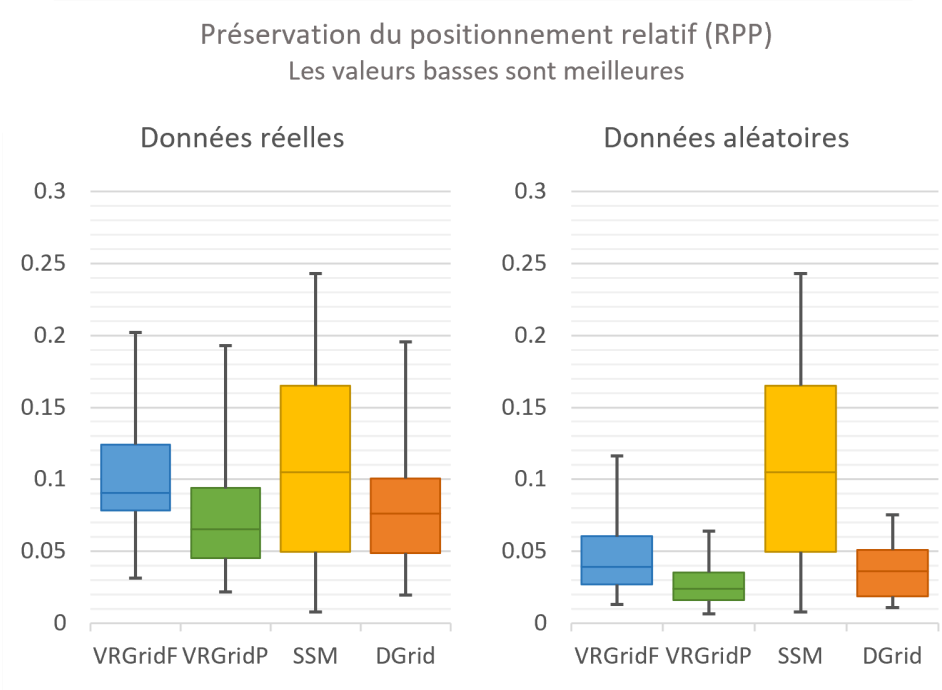


FIGURE 5.22 – Résultats agrégés des méthodes concernant la préservation du positionnement relatif sur les jeux de données réels (à gauche) et aléatoires (à droite). Les valeurs basses sont meilleures.

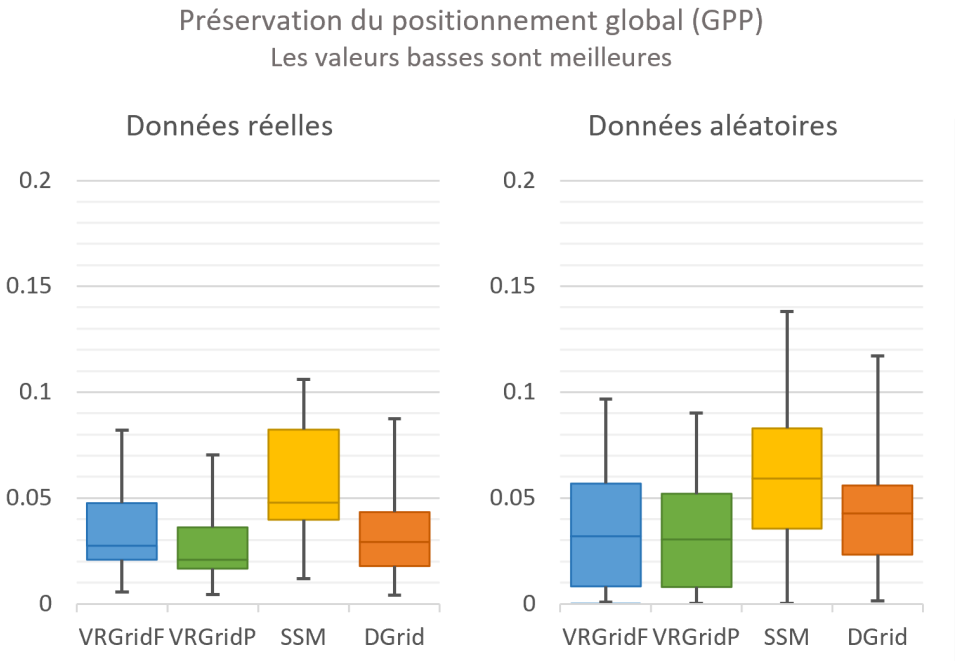


FIGURE 5.23 – Résultats agrégés des méthodes concernant la préservation du positionnement global sur les jeux de données réels (à gauche) et aléatoires (à droite). Les valeurs basses sont meilleures.

5.5.6 Préservation du positionnement global (GPP)

Sur des données réelles, VRGrid$_P$ obtient les meilleurs résultats avec un écart de 46.59% par rapport à SSM, 14.83% face à DGrid et 21.60% avec VRGrid$_F$. Les résultats sont similaires avec

Type de données	Taille	VRGrid$_F$	VRGrid$_P$	SSM	DGrid
Données réelles	1 024	1,65	66,94	0,93	<0,01
	10 000	46,17	5 594,37	16,13	0,01
Données aléatoires denses	64	0,04	0,46	0,02	<0,01
	100	0,07	1,01	0,03	<0,01
	1 000	1,65	68,53	0,52	<0,01
	10 000	46,94	5 628,95	15,92	0,01
Données aléatoire éparpillées	64	0,05	0,40	0,03	<0,01
	100	0,07	0,98	0,04	<0,01
	1 000	1,64	63,10	0,55	<0,01
	10 000	46,57	4 999,92	9,46	0,01

TABLE 5.2 – Temps de calcul moyen en secondes sur les jeux de données organisés par densité et taille.

les données aléatoires ; VRGrid$_P$ produit des arrangements 46.47% meilleurs que ceux de SSM, 23.55% meilleurs que ceux de DGrid et 17.66% meilleurs que ceux de VRGrid$_F$.

De la même manière que sur la préservation du positionnement relatif, les résultats de SSM sont fortement pénalisés malgré la préservation plus ou moins globale de la forme des données. On remarque aussi que les distorsions et les discontinuités des arrangements de VRGrid$_F$ et DGrid sont moins pénalisées que le décalage global des données dans les arrangements de SSM.

L'utilisation d'une méthode de relaxation est encore une fois mise en valeur sur cette métrique d'évaluation, ainsi que l'importance du facteur de stabilité.

5.5.7 Temps de calcul

Dans tous les scénarios testés, DGrid obtient des temps de calculs drastiquement plus rapides que les autres méthodes, avec des temps inférieurs à 0.02 secondes quel que soit le jeu de données arrangé. SSM obtient ensuite des résultats plus rapidement que VRGrid$_P$ et VRGrid$_F$, dont l'écart avec cette dernière se creuse sur des jeux de données plus grands. Les temps de calcul obtenus par VRGrid$_P$ sont considérablement élevés par rapport aux autres méthodes (autour de 270 fois plus lent que SSM), et la densité des données n'a qu'un impact moyen sur le temps de calcul total, avec un écart de 11% entre les données aléatoires denses et éparts. La limitation de VRGrid$_F$ à 10 itérations par bordure à remplir permet d'obtenir un temps de calcul jusqu'à 100 fois plus rapide que VRGrid$_P$, mais cela ne suffit pas à être plus rapide que SSM.

Si VRGrid$_P$ a pu obtenir des résultats très compétitifs sur les métriques d'évaluation précédentes, la méthode ne peut pas se mesurer à SSM et DGrid en ce qui concerne le temps de calcul. Même en réduisant considérablement le nombre d'itérations de l'algorithme et donc en sacrifiant grandement la qualité des rendus comme observé sur les résultats de VRGrid$_F$, la méthode ne parvient pas à dépasser SSM en termes de rapidité.

5.6 Discussion

Dans cette évaluation, nous avons pu montrer que VRGrid, plus précisément VRGrid$_P$, est très compétitive pour obtenir des arrangements de qualité en se basant sur des métriques proposées par les méthodes de l'état de l'art et utilisées dans le domaine du dessin de graphe. En effet, VRGrid$_P$ obtient les meilleurs résultats sur la préservation des distances (DP), la préservation du positionnement relatif (RPP) et sur la préservation du positionnement global (GPP). En revanche, la préservation du voisinage est un point faible de la méthode, et ne parvient pas à surpasser les

méthodes de l'état de l'art sur les jeux de données réels, même avec un temps de calcul bien plus étendu.

Nous avons montré que SSM rencontre des difficultés à conserver la forme des données originales, et peut provoquer des mauvais placements de points qui peuvent être sujets à de mauvaise interprétations comme en Figure 5.18.

En ce qui concerne DGrid, bien que la méthode obtienne des résultats moins bons que VRGrid$_P$ sur RPP et GPP, la discontinuité des données dans les arrangements ne semble pas affecter de manière significative les métriques d'évaluation, là où les résultats de SSM semblent plus appropriés pour une analyse visuelle comme dans le scénario UMAP sur Swissroll. Cette observation indique que les métriques employées dans cette étude ne sont peut-être pas suffisantes pour couvrir tous les cas d'analyse visuelle d'information.

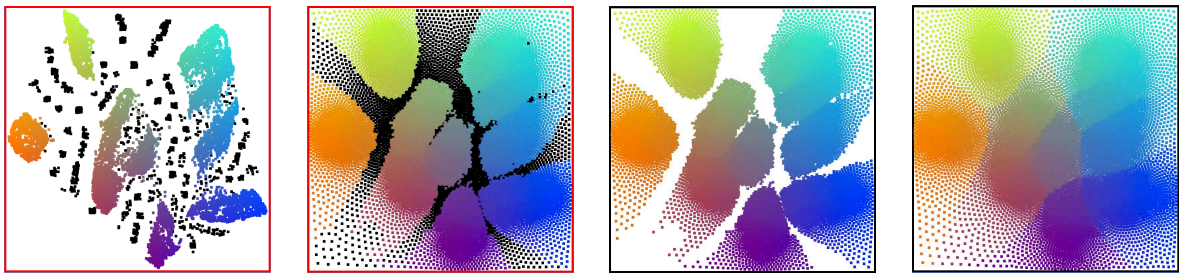
Avec cette évaluation, nous pouvons affirmer que toutes les méthodes sont utiles dans des cas d'usage différents. Si une projection compacte d'informations seule est nécessaire, sans accorder d'importance au positionnement initial des informations, SSM est particulièrement efficace puisqu'elle conserve bien les distances et le voisinage des éléments, avec un temps de calcul raisonnable. Si l'arrangement compact sert d'outil de visualisation supplémentaire à la projection classique, VRGrid et DGrid se montrent particulièrement efficaces ; la première propose des arrangements impliquant un minimum de déplacements de points mais des temps de calculs plus grands, tandis que la seconde obtient des arrangement pratiquement instantanément mais avec une discontinuité plus prononcée dans la disposition des points. Parmi les scénarios qui pourraient faire usage de telles représentations d'informations, on peut mentionner les outils de visualisation à vues multiples (impliquant des vues synthétiques avant de choisir celle à explorer en particulier), ce qui peut concorder avec notre objectif initial de vouloir représenter l'espace $\mathbb{R}^N$ au niveau de chaque couche d'un réseau de neurones.

Enfin, nous montrons dans notre évaluation que VRGrid ne peut pas se mesurer aux méthodes de l'état de l'art lorsque le temps de calcul est un critère important de choix de méthode. Le nombre d'itérations nécessaires à VRGrid$_P$ pour produire de bons arrangement avoisine, voire dépasse, la taille du jeu de données traité, ce qui implique une complexité de $O(N^{2.5}.log(N))$. Cette complexité reste inférieure à celle d'IsoMatch, mais rejoint l'ensemble des méthodes à complexité quadratique, ce qui est un problème lors du traitement des données de masses, chose fréquente dans le domaine de l'intelligence artificielle. Nous avons proposé une version de VRGrid diminuant considérablement le temps de calcul de la méthode, mais la perte en qualité est trop importante pour que ce gain de temps soit justifié.

En l'état, l'implémentation de l'algorithme de Lloyd est la partie de la méthode à améliorer pour gagner en temps de calculs. Plusieurs pistes d'améliorations sont présentées dans la littérature, notamment au niveau algorithmique impliquant une surrelaxation [151] (*over-relaxation*) permettant de réduire le nombre d'itérations à calculer, ainsi qu'au niveau de l'optimisation matériel en faisant usage de processeurs graphiques [116, 155]. Ces optimisations ont le potentiel, d'après leurs auteurs, d'accélérer la convergence de l'algorithme vers une CVT au moins 10 fois plus rapidement que l'algorithme traditionnel s'exécutant sur processeur classique. D'autres méthodes permettent aussi de calculer une configuration CVT en moins d'itérations qu'avec l'algorithme de Lloyd, comme la méthode de Broyden-Fletcher-Goldfarb-Shanno [77] (*BFGS*). Ce genre d'optimisation pourrait considérablement aider à mitiger le problème de scalabilité de la méthode pour pouvoir s'adapter à des données plus grandes en des temps raisonnables, sans pertes de qualité d'arrangements.

5.6.1 Pistes supplémentaires envisagées

L'un des soucis relevés avec les arrangements produits par VRGrid$_P$ (et de manière plus significative avec VRGrid$_F$) est le positionnement des points aux frontières entre les groupes. Nous supposons que ce comportement découle des déplacements de points trop grands lors des pre-



(a) Remplissage initial de la zone de calcul avec des points fictifs (en noir)

(b) Itération de l'algorithme de Lloyd jusqu'à une première stabilisation

(c) Suppression des points fictifs

(d) Poursuite de l'algorithme VRGrid classique

FIGURE 5.24 – Modification de VRGrid pour améliorer les placements des points en bordure des groupes. En rajoutant des points fictifs dans les espaces vides de la zone de calcul jusqu'à la première stabilisation de l'algorithme de Lloyd, les points sont moins enclins à se mélanger avec les autres groupes au début de l'application de VRGrid.

mières itérations de l'algorithme de Lloyd et il est envisageable d'améliorer cela en apportant des modifications à VRGrid.

Contrainte des distances parcourues par les points (*clamping*)

Dans un premier temps, nous avons ajouté un paramètre supplémentaire à VRGrid permettant de définir une distance de déplacement maximum pour les points non-fixés. Avec une distance maximale suffisamment petite, nous pouvons ainsi empêcher les points de s'éloigner trop rapidement de leur position de départ et donc des points des groupes voisins. Cependant, par rapport à VRGrid$_P$, ce paramétrage produit un arrangement très légèrement moins bon sur les quatre mesures de qualité (différence inférieure à 1%).

Remplissage des espaces vides

Un autre moyen d'empêcher les points de rejoindre trop rapidement les points des autres groupes voisins est de combler l'espace vide les séparant par des points factices, comme montré en Figure 5.24. Dans notre implémentation, nous remplissons les espaces vides en fonction du nombre d'éléments entourant ces espaces vides ; plus ils sont nombreux, plus l'espace vide sera rempli. Pour cela, nous avons utilisé la triangulation de Delaunay calculée dans l'algorithme pour positionner ces points fictifs au milieu des arêtes dépassant une certaine taille. Ces points sont aussi considérés par l'algorithme de Lloyd comme les points du jeu de données d'entrée, et restent jusqu'à que qu'une première configuration stable soit atteinte, après quoi ils sont retirés du calcul. Avec cette méthode, les points des différents groupes restent séparés par une frontière définie par ces points factices, qui pourra s'adapter à la suite des itérations de l'algorithme de Lloyd. Lorsqu'une première configuration stable est atteinte, ces points factices sont retirés pour permettre de combler les derniers espaces vides de la zone de calcul, avant d'assigner les points aux bordures comme le déroulement normal de VRGrid. La qualité de l'arrangement produit par cette méthode est très légèrement meilleure que celle produite par VRGrid$_P$ mais augmente le temps de calcul de l'algorithme (le temps d'obtenir la première stabilisation), ce qui est peu souhaitable compte tenu de l'apport en qualité obtenu par cette méthode.

Mesure	VRGrid$_P$	Contrainte	Remplissage	SSM	DGrid
DP	0,8422	0,8398	**0,8428**	0,8355	0,8109
KNP	0,4653	0,4641	0,4761	0,5559	**0,5745**
RPP	0,0613	0,0622	**0,0612**	0,1025	0,0693
GPP	**0,0178**	0,0179	**0,0178**	0,0297	0,0214

TABLE 5.3 – Valeurs mesurées sur les arrangements produits par VRGrid$_P$et les versions modifiées de VRGrid sur la projection UMAP de MNIST. L'arrangement produit par remplissage au préalable des espaces vides avec des points fictifs temporaires permet d'obtenir un arrangement légèrement de meilleure qualité par rapport à VRGrid$_P$.

5.6.2 Mesures et nombre d'itérations obtenus sur la projection UMAP/MNIST

La qualité des arrangements produits par ces versions modifiées de VRGrid pour le scénario MNIST projeté avec UMAP sont présentés dans le Tableau 5.3 et les déplacements moyens par itérations en Figure 5.25.

Comme indiqué précédemment, les méthodes ont produit un arrangement de qualité similaire, mais la version de VRGrid remplissant les zones vides de l'espace de calcul en premier lieu obtient les meilleures valeurs. Pour ce qui est du temps de calcul, les deux versions modifiées de VRGrid demandent plus d'itérations afin de calculer l'arrangement (+1.6% pour la version avec remplissage et +2.4% pour la version limitant les déplacements). Cependant, ce temps de calcul supplémentaire ne permet pas de rattraper les méthodes d'état de l'art SSM et DGrid sur la mesure de préservation du voisinage.

5.7 Application à l'étude des réseaux de neurones

Nous avons appliqué la méthode VRGrid$_P$sur les cartes d'activation produites par LeNet5 lors de la classification de MNIST, projetées par t-SNE. Les résultats obtenus sont présentés en Figure 5.26, aux côtés des visualisations obtenues sur ces mêmes données (bien que les méthodes de projection soient différentes) dans les chapitres précédents.

Sur ces résultats, nous pouvons remarquer des points communs entre les différentes visualisations ; le groupe bleu foncé ■ est toujours identifiable dès les premières couches et reste plus ou moins homogène pendant toute la classification. De même, les groupes rose ■ et violet ■ sont mélangés dans les deux couches de convolution, avant de se séparer sur les couches denses. On remarque, à ce niveau-là, une différence entre l'arrangement produit par VRGrid$_P$par rapport aux deux autres méthodes : ces deux groupes semblent être séparés correctement dès la couche Dense 1 alors que la visualisation par flux et celle basée sur les courbes fractales ne montrent pas de réelle séparation entre ces groupes. Ces différences peuvent être dues à la méthode de projection employée (ici t-SNE), même si les données étudiées sont les mêmes. Cette différence nous oriente vers la nécessité d'avoir plus d'information disponible concernant les relations entre éléments dans l'espace $\mathbb{R}^N$ des cartes d'activation, afin de pouvoir comparer les distances séparant les groupes, et donc la capacité du réseau à discriminer les différentes catégories d'éléments. Une autre différence importante entre les arrangements par VRGrid$_P$et les grilles de pixels construites en utilisant la courbe de Hilbert est le positionnement des points sur la grille. Là où les points forment des groupes plus ou moins homogènes sur l'arrangement VRGrid$_P$, les points sur la courbe de Hilbert sont très dispersés et certaines classes d'éléments ne forment même pas de groupe. Il peut s'agir là d'une limitation de l'utilisation de la courbe de Hilbert et/ou une limitation de la projection vers $\mathbb{N}$ lors du calcul des matrices de distance et le calcul de l'ordre des lignes et colonnes qui suit. En revanche, nous pouvons observer une nette amélioration de la classification entre les couches Conv 2 et Dense 1, ce qui nous encourage à poursuivre les travaux sur la représentation de cette classification progressive des éléments par les réseaux de neurones.

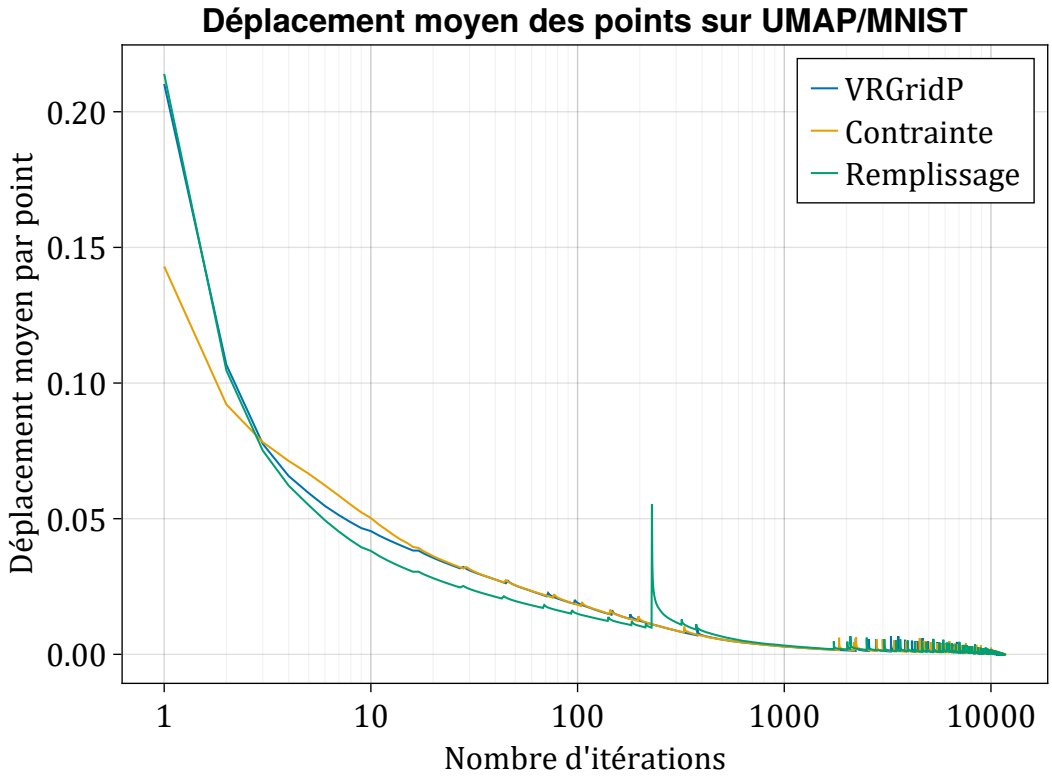


FIGURE 5.25 – Déplacement moyen des points par itération et par méthode. VRGrid$_P$(en bleu) a eu besoin de 11 504 itérations pour calculer l'arrangement, la version ajoutant temporairement des points fictifs (Remplissage, en vert) 11 688 itérations et la version appliquant des contraintes (Contrainte, en orange) sur les déplacement a eu besoin de 11 776 itérations. On peut remarquer le pic de mouvements de la version avec remplissage après ~ 110 itérations, où les points fictifs sont retirés du calcul, et les autres points remplissent l'espace dégagé par ceux-ci.

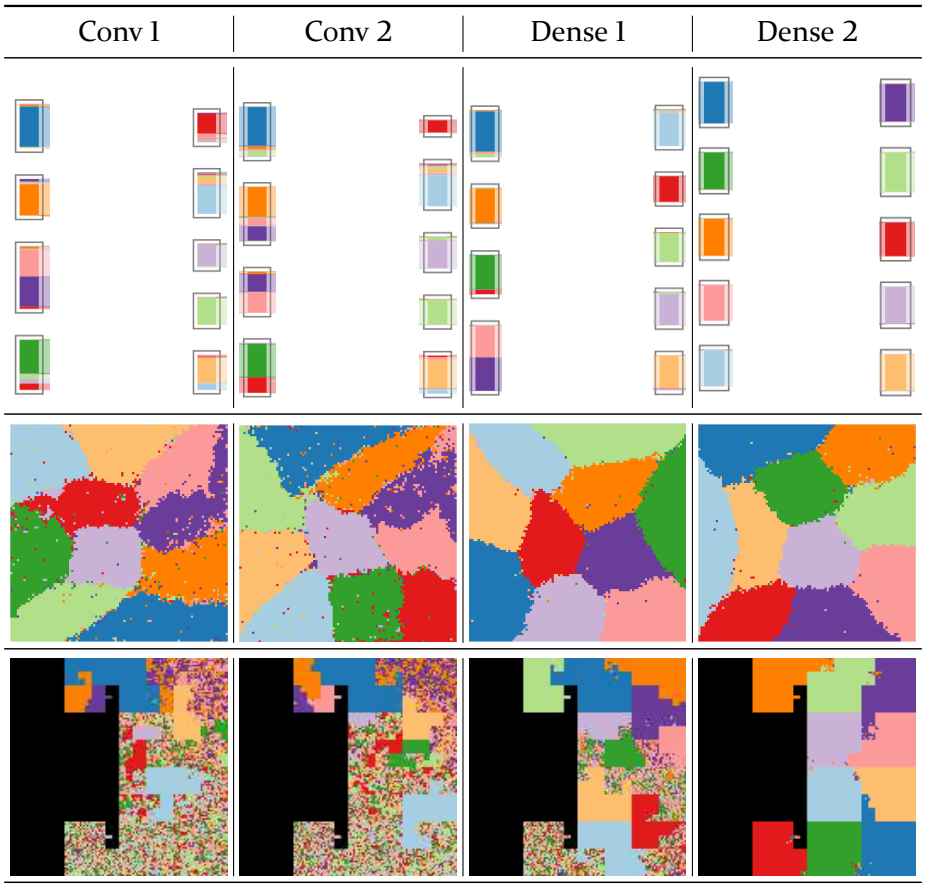


FIGURE 5.26 – Comparaison des visualisations proposées dans ces trois derniers chapitres, sur un scénario commun, celui de la classification de MNIST par LeNet5. La première ligne contient les groupes obtenus sur la visualisation par flux (Chapitre 3, scindés en deux colonnes). La deuxième ligne montre les arrangements obtenus par VRGrid$_P$ sur une projection t-SNE des cartes d'activation calculées par la classification. Enfin la troisième ligne contient les grilles de pixels construites sur la base de la courbe de Hilbert (Chapitre 4). Les données affichées sont toutes colorées de la même palette.

5.8 Conclusion

Dans ce chapitre, nous avons montré une méthode de post-traitement permettant de générer un arrangement compact en grille de n'importe quel jeu de données dans $\mathbb{R}^2$. Cette méthode est prévue pour être appliquée à la suite d'une projection de données en $\mathbb{R}^N$ vers $\mathbb{R}^2$, afin de générer une grille de pixels déformant le moins possible la projection initiale. VRGrid a été conçue notamment pour faire face aux problèmes rencontrés dans le Chapitre 4 : obtenir une meilleure exploitation de l'espace de dessin en produisant des représentations de données compactes, et pouvoir s'adapter à différentes techniques de réductions de dimensions.

Via un protocole d'évaluation basé sur des métriques empruntées aux méthodes d'état de l'art sur la représentation compacte ainsi que sur le dessin de graphe, nous avons montré que VRGrid permet d'obtenir de bonnes représentations compactes par rapport aux méthodes de l'état de l'art, à condition d'allouer un temps de traitement suffisamment important. Cette méthode, et plus particulièrement l'algorithme de Lloyd, dispose cependant de plusieurs pistes d'améliorations pouvant alléger cette limitation.

En contrepartie, l'algorithme SSM peut être utilisé pour traiter des jeux de données aussi grands que ceux traités dans le chapitre précédent. Dans l'optique où une projection intermédiaire d'un jeu de données $\mathbb{R}^N$ n'est pas nécessaire mais qu'une fonction de similarité est suffisante

pour comparer les données (comme ce fût le cas dans le chapitre précédent), SSM est à considérer comme approche pour générer des grilles de pixels plutôt que de passer par des matrices de distance suivi d'un ordonnancement sur une courbe fractale.

Dans les deux cas, nous avons maintenant à notre disposition des méthodes pouvant projeter des informations de $\mathbb{R}^N$, l'espace de calcul des couches de réseaux de neurones, vers un espace compact sur $\mathbb{N}^2$, permettant d'explorer plusieurs grands jeux de données sur le même écran.

Chapitre 6

Méthodologie d'évaluation qualitative de méthodes de visualisation

Dans ce chapitre, nous cherchons à compléter notre étude dans la visualisation compacte d'informations initiée dans le chapitre précédent. Lorsque nous avons présenté et évalué VRGrid contre des méthodes d'état de l'art, nous nous sommes concentrés sur une évaluation quantitative basée sur la préservation d'information et la minimisation de déformations entre la visualisation compacte et la visualisation sous la forme de nuages de points. Nous avons montré les forces et faiblesses des méthodes testées, mais cette évaluation s'est focalisée uniquement entre des méthodes de visualisation compacte. L'efficacité même de la visualisation compacte par rapport à la visualisation en nuage de points, qu'on appellera classique, reste à montrer.

Ici, nous choisissons de conduire une évaluation qualitative, à l'aide de participants, pour comparer l'efficacité à résoudre des tâches communes lors de la visualisation d'information. Cette comparaison est conduite entre la visualisation par nuages de points et la visualisation compacte. Cependant, cette évaluation nécessite d'avoir une méthodologie de comparaison de méthodes de visualisation efficace. Dans ce chapitre, nous présentons la méthodologie que nous avons mis en place pour faire de telles comparaisons. Nous appliquons ensuite cette méthodologie dans une étude utilisateurs pour comparer l'efficacité des méthodes de visualisation Self-Sorting Maps et t-SNE pour la visualisation de données $\mathbb{R}^N$ dans $\mathbb{R}^2$. Par la suite, nous analysons et discutons des résultats obtenus et du ressenti des participants sur cette évaluation.

Ce chapitre est découpé en quatre parties. Dans un premier temps, nous présentons un état de l'art des études utilisateurs existantes portant sur la visualisation d'information. Ensuite, nous présentons la méthodologie que nous suivons pour pouvoir mener une étude utilisateur comparative entre méthodes de visualisations. Cette méthodologie demande au préalable de préparer des jeux de données configurables, ce que nous présentons en troisième partie. Enfin, nous présentons une application de cette méthodologie en comparant deux méthodes de visualisations, en y incluant une discussion sur les résultats obtenus, avant de conclure le chapitre.

Les contributions dans ce chapitre ont mené à :

- L'élaboration d'une méthodologie pour comparer l'efficacité de méthodes de visualisations.
- Une méthode de coloration contextuelle pour étudier la topologie des données dans $\mathbb{R}^N$.
- Un exemple d'application, proposé aux participants à l'adresse `https://pivert.labri.fr/eval-\{fr,en\}/index.html` et dont le code est disponible à l'adresse `https://github.com/eikofee/gridviz-eval`.
- Un outil de génération de données dans $\mathbb{R}^N$ respectant des contraintes topologiques, disponible à l'adresse `https://github.com/eikofee/grid-eval-data-generator`.

6.1 Évaluations utilisateur sur les méthodes de visualisations d'informations

Les propositions de méthodes de visualisation d'information sont nombreuses dans la littérature. Munzner a proposé la méthodologie des blocs imbriqués [91, 92] (*nested model*) pour la conception d'outils de visualisations. Cette méthodologie a pour but d'aligner les fonctionnalités de l'outil conçu avec les besoins des experts, en guidant le processus de développement à différents niveaux (modélisés par des blocs dans la méthodologie). Les performances de ces méthodes peuvent ensuite être mesurées dans bien des cas de manière quantitative, mais une évaluation avec des participants permet de s'assurer de la praticabilité des méthodes et d'avoir des retours pertinents des experts (ou non) vers les concepteurs de la méthode. La conduite de ces évaluations peut varier d'un domaine de recherche à un autre. Par exemple, Spinner et al. proposent explAiner [126], un *framework* pour la conduite d'évaluations sur des outils d'explications de méthodes d'intelligence artificielle. De manière plus générale, les évaluations basées sur des utilisateurs peuvent prendre des formes différentes. Dans M2Lens [143] de Wang et al., les utilisateurs sont des experts qui utilisent l'outil dans des scénarios préparés par les auteurs. Ces utilisateurs sont observés sur leur utilisation de l'outils, et leurs retours sont pris en compte pour évaluer la pertinence de l'outil. Les ressentis de ces utilisateurs sont souvent collectés sous la forme de questionnaires de Likert [94], comme par exemple dans l'évaluation utilisateur menée par Jin et al. pour leur outil GNNLens [59].

Dans ce chapitre, nous nous penchons plutôt vers les évaluations de tâches spécifiques, c'est-à-dire que les utilisateurs sont menés à répondre à un ensemble de tâches succinctes sur un nombre relativement important de scénarios. Dans l'étude menée par Woodburn et al. [149], les participants sont menés à manipuler trois méthodes de visualisations interactives de données hiérarchiques (les *treemaps*, *icicle plots* et *sunburst plots*). Ces manipulations sont réparties dans 4 tâches interactives (navigation dans les éléments, compréhension d'attributs des éléments et compréhension de la hiérarchie représentée). Le taux de réussite des participants ainsi que leur temps de réponse sont étudiés pour évaluer les performances des méthodes de visualisation pour répondre à ces tâches. Ces participants sont ensuite menés à donner leur préférence quant à la méthode qu'ils préfèrent via un questionnaire à la fin de l'évaluation. L'étude menée par Pommé et al. [110] cherche à comparer trois méthodes de colorations pour un même format de visualisation d'information (la matrice de confusion). Celle-ci est menée posant des questions de compréhension des données représentées par ces matrices à travers 4 tâches. Ces évaluations peuvent aussi porter sur des concepts basique de la visualisation d'information, comme les formes ou les couleurs. L'évaluation utilisateur de Giovannangeli et al. [38] sur la détection d'intrus cherche à mettre en évidence les attributs visuels (forme et/ou couleur) rendant difficile la détection d'intrus en visualisation. Contrairement aux évaluations mentionnées précédemment, cette évaluation est elle composée d'une seule tâche qui est celle de trouver un intrus dans une grille de 8×8 symboles.

Ces évaluations utilisateurs peuvent être menées notamment avec l'aide de plateformes en ligne dites de *crowdsourcing*, afin d'évaluer des systèmes avec un grand nombre de participants. Ces plateformes sont de plus en plus sollicitées par la communauté scientifique afin d'obtenir des évaluations plus complètes de leurs systèmes [13]. Cependant, bien que les évaluations portant sur un grand nombre de participant sont généralement préférées car ils offrent une meilleure confiance dans les résultat obtenus, le nombre de participants ne définie pas la qualité d'une évaluation. Ellis et Dix [32] relèvent des points pouvant améliorer la pertinence d'une évaluation utilisateur (par exemple, ne pas se limiter à une population en particulier). Merino et al. [83] présentent une synthèse étendue des méthodes d'évaluation utilisateurs conduites dans plus d'une centaine d'outils de visualisation et proposent plusieurs lignes de conduites afin de rendre les évaluations utilisateurs plus pertinentes.

6.2 Méthodologie d'évaluation

Notre comparaison de méthodes de visualisation cherche à mesurer leur capacité et leur facilité d'utilisation pour répondre à des objectifs propres à l'étude de données dans $\mathbb{R}^N$.

6.2.1 Types de données

Notre évaluation reflète l'observation de similarités sur des données dans $\mathbb{R}^N$ projetées dans $\mathbb{R}^2$. Cette similarité se traduit par la proximité des éléments dans $\mathbb{R}^N$, qui doit donc être visible sur la visualisation produite une fois ces données projetées dans $\mathbb{R}^2$. En plus de cela, des étiquettes sont associées à chaque élément pour indiquer leur appartenance à une classe d'équivalence. Les éléments ayant une même classe d'équivalence appartiennent à un même domaine continu de valeurs dans $\mathbb{R}^N$. Cependant, certains éléments étiquetés d'une classe peuvent se trouver éloignés des autres éléments de cette même classe, qu'on appelle «intrus» (*outliers*).

6.2.2 Objectifs et tâches

Nous définissons des objectifs que nous considérons comme importants lors de l'analyse d'une projection de données $\mathbb{R}^N$, lorsqu'elle est visualisée dans un espace en deux dimensions :

- **G1 : Détecter la répartition des données**. Quantifier la proportion d'une classe d'éléments est une tâche commune lors d'analyses faisant usage de méthodes de visualisation. Compter individuellement chaque élément du jeu de données n'est pas nécessaire, mais l'utilisateur doit pouvoir estimer et comparer les tailles des groupes de données facilement.
- **G2 : Comparer les similarités et distances entre les groupes de données**. Avoir une indication sur comment deux groupes d'éléments sont différents l'un de l'autre dans $\mathbb{R}^N$ est important dans plusieurs domaines d'application. Cette différentiation peut se calculer notamment par une mesure de dissimilarité, souvent synonyme de distance dans l'espace $\mathbb{R}^N$.
- **G3 : Comprendre la disposition générale des données dans** $\mathbb{R}^N$. Des jeux de données dans $\mathbb{R}^N$ peuvent prendre de nombreuses formes dans leur espace d'origine. Par exemple, les éléments peuvent être alignés le long d'un axe, des groupes d'éléments peuvent partager un même domaine de valeurs, deux groupes différents peuvent avoir une proximité bien plus prononcée entre eux mais moins avec les autres. Lorsque les données sont projetées, la réduction du nombre de dimensions implique des changements dans leur disposition. La disposition d'origine n'est donc pas forcément détectable facilement, voire pas du tout, par l'utilisateur.
- **G4 : Détecter les intrus**. Il est très fréquent, lors de la projection de données à priori groupées selon un étiquetage, de vouloir détecter facilement les données dites «mal classées» (*outliers*). La détection de ce genre de données se fait principalement en colorant tous les points en fonction de leur étiquette, puis en trouvant des points isolés d'une couleur différente de celle de leur voisinage.

À partir de ces objectifs, nous pouvons définir une liste de tâches permettant d'évaluer si un utilisateur est capable de remplir ces objectifs et avec quelle facilité.

- **T1 : Trouver la classe de données la plus représentée**. Dans cette tâche, nous demandons à l'utilisateur de trouver l'étiquette du groupe ayant la plus grande population à l'écran (G1).
- **T2 : Trouver le groupe le plus proche d'un autre donné dans** $\mathbb{R}^N$. Dans cette tâche, nous désignons un groupe sur la projection affichée à l'écran, et nous demandons à l'utilisateur de trouver le groupe le plus proche dans $\mathbb{R}^N$ de celui désigné (G2).
- **T3 : Retrouver la structure des données dans** $\mathbb{R}^N$. Dans cette tâche, l'utilisateur doit interpréter le résultat de la projection affichée pour deviner la disposition (la topologie) d'origine des données dans $\mathbb{R}^N$ (G3).

— **T4 : Trouver le nombre d'intrus dans un groupe donné**. Dans cette tâche, nous demandons à l'utilisateur de compter le nombre d'intrus au sein d'un groupe donné (G4).

Les questions composant l'évaluation appartiennent chacune à une des tâches décrites. Afin de faire varier la difficulté de chaque question et/ou couvrir des configurations précises, il est nécessaire d'avoir des jeux de données ayant des caractéristiques particulières :

— Pour T1 : une taille différente de chaque groupe étiqueté (N1). Une question peut être difficile si la différence entre deux tailles de groupes est petite.
— Pour T2 : des distances de groupe à groupe plus ou moins grandes mais connues avant la projection (N2).
— Pour T3 : une topologie précise connue avant la projection (N3).
— Pour T4 : un nombre d'intrus précis et réparti comme souhaité avant la projection (N4).
— Pour toutes les tâches, un nombre de groupes et un nombre de dimensions dans $\mathbb{R}^N$ définis (N5).

6.2.3 Caractéristiques des données

Étant donné la quantité de paramètres à ajuster suivant les questions, nous proposons de générer des jeux de données pouvant respecter cet ensemble de paramètres plutôt que de chercher et adapter des jeux de données existants à nos questions. En ce qui concerne la variété de topologies à étudier, nous avons choisi de nous concentrer sur des types structurels faisant varier la distance entre les groupes en les éloignant ou en les rapprochant les uns des autres. Nous voulons aussi avoir la possibilité d'inclure des groupes dans d'autres groupes. Ainsi, nous proposons de nous focaliser sur 4 types structurels que nous pouvons rencontrer lors d'étude de données dans $\mathbb{R}^N$:

— Structure «**séparée**» : tous les groupes du jeu de données sont séparés les uns des autres sans proximité apparente entre eux.
— Structure «**mélangée**» : il y a au moins deux groupes ayant des éléments partageant un même domaine de valeurs.
— Structure «**bulle**» : il y a au moins un groupe dont toutes les données se trouvent dans l'enveloppe convexe formée par les données d'un second groupe, sans que les éléments des deux groupes partagent un même domaine de valeurs.
— Structure «**proche**» : il y a au moins deux groupes étant significativement proches par rapport aux autres groupes du jeu de données.

Ces structures ne sont pas représentatives de toutes celles que l'on peut rencontrer en étudiant des données dans $\mathbb{R}^N$ (comme les structures volontairement particulières Swissroll, etc... du *Matlab Toolbox for Dimensionality Reduction* [137] étudiées dans le Chapitre 5), mais sont des structures génériques pouvant être présentes dans des jeux de données comme MNIST par exemple (proximité des «5» et «8», mélange des «4» et «9», ...).

La Figure 6.1 montre des exemples de jeux de données générés dans $\mathbb{R}^2$ représentant chacune des structures. Ces données sont montrées à titre d'exemple, puisque nous voulons générer les données dans $\mathbb{R}^N$ demandant une projection au préalable pour notre évaluation.

6.3 Génération des données d'évaluation

Nous proposons une méthode de génération de jeux de données dans $\mathbb{R}^N$ qui permet de répondre aux besoins N1 à N5. Cette génération repose sur la disposition aléatoire de points pivots $p \in \mathcal{P}$ dans $\mathbb{R}^N$, ayant chacun leur hypervolume $\mathcal{D}_p$ où les données sont générées. Ces hypervolumes servent à ce que n'importe quel point dans un hypervolume soit plus proche des autres points de ce même hypervolume que d'un point situé dans un autre hypervolume. Plus formelle-

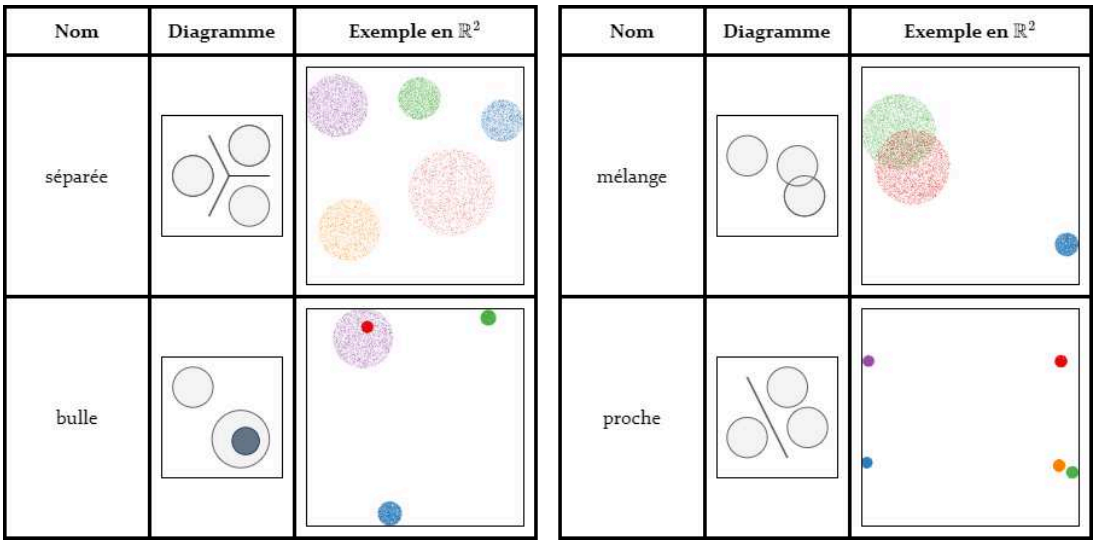


FIGURE 6.1 – Exemple de jeux de données générés en $\mathbb{R}^2$ respectant chacune des quatre structures prédéfinies. Ces types de structures sont associés à un diagramme pour aider à leur mémorisation lors de l'évaluation. La disposition des groupes structure «partagée» est plus ou moins uniforme. Dans la structure «bulle», le groupe rouge est inclus dans le groupe violet sans que des éléments de ce dernier s'y trouvent mêlés (leur union est vide). Dans la structure «mélange», les éléments des groupes vert et rouge partagent un même domaine de valeurs. Enfin dans la structure «proche», les groupes orange et vert sont significativement plus proches entre eux que le reste des groupes du jeu de données.

ment, cette propriété se décrit par l'équation suivante :

$$\forall x, y \in \mathcal{D}_p \implies \forall z \in \bigcup_{p' \in \mathcal{P} p \neq p'}, ||x-y|| < ||x-z|| \wedge ||x-y|| < ||y-z|| \tag{6.1}$$

Les hypervolumes sont définis comme étant l'ensemble des valeurs dans $\mathbb{R}^N$ se trouvant à une distance inférieure ou égale à un quart de la distance minimale entre deux points pivots. Ils peuvent donc être considérés comme des volumes délimités par une n-sphère de rayon égal au quart de la distance minimale entre deux points pivots :

$$\forall p, q \in \mathcal{P}, \mathcal{D}_p = \bigcup_{x \in \mathbb{R}^N} \{x\} \text{ tel que } ||x-p|| < \frac{1}{4} min\left(\bigcup_{p', q' \in \mathcal{P}} ||p'-q'|| \right) \tag{6.2}$$

Nous définissons ces points pivots de manière hiérarchique afin d'obtenir différents niveaux de distances entre les pivots, comme montré par exemple dans la Figure 6.2. Nous appelons ces relations «Liens pivots» dans notre algorithme. Ce système de pivots s'adapte à N dimensions, car il repose uniquement sur le calcul de distances : il permet donc de générer des jeux de données dans $\mathbb{R}^N$ (N1, N5). Une fois la hiérarchie établie, nous pouvons générer les points composant le jeu de données aléatoirement dans les hypervolumes centrés sur les points pivots situés au dernier niveau de la hiérarchie. Par construction, nous pouvons maintenant générer un jeu de données dont les proximités (donc les similarités) des éléments entre eux sont contrôlées.

Nous pouvons aussi forcer un rapprochement ou un éloignement des points pivots pour respectivement faire superposer ou éloigner les hypervolumes où les éléments du jeu de données sont générés. Cette manipulation nous permet d'avoir un contrôle sur les distances entre les groupes de données ainsi que de produire la structure «mélange» (N2). En revanche, le rapprochement ou l'éloignement de ces points peut modifier la structure des données établie par la hiérarchie des

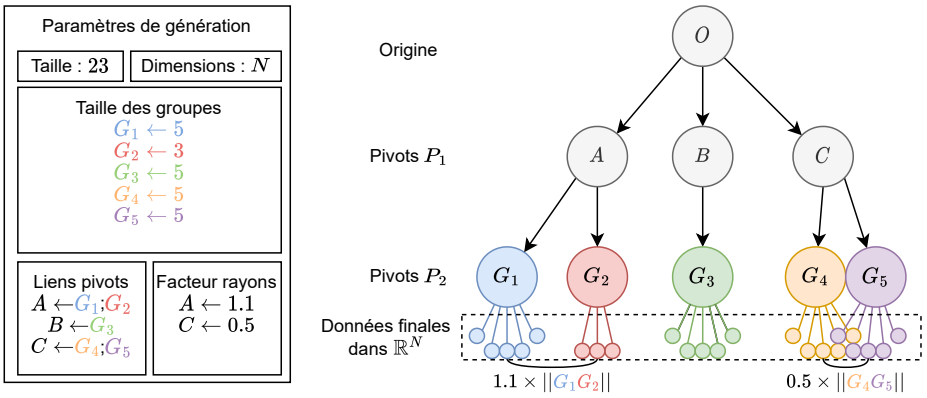


FIGURE 6.2 – Présentation du système de génération de jeux de données. Avec les paramètres définis en (a), l'algorithme génère une structure de points dans $\mathbb{R}^N$ à partir d'une structure arborescente de pivots en (b). À l'aide de liens pivots, nous générons deux groupes G_1 et G_2 proches l'un de l'autre (dans le pivot A), deux groupes G_4 et G_5 (dans le pivot C) partageant un domaine de valeurs (par réduction des distances entre les pivots de C) et enfin un groupe G_3 isolé par rapport aux autres (dans le pivot B).

points pivots, il faut donc vérifier que leur positionnement dans $\mathbb{R}^N$ reste cohérent par rapport à la structure que l'on souhaite obtenir.

Enfin, nous permettons aussi d'indiquer des relations d'exclusion entre des points pivots ayant le même parent. Ces exclusions sont prises en compte lors de la génération des points dans les zones de calculs ; lorsqu'un point est généré, sa distance avec chaque pivot exclu est vérifiée. Si ce point est inclus dans l'une de ces zones de calcul exclues, le tirage aléatoire de sa position est relancé.

Avec ces paramètres, il devient possible de générer des jeux de données correspondant aux 4 structures mentionnées précédemment (N3) :

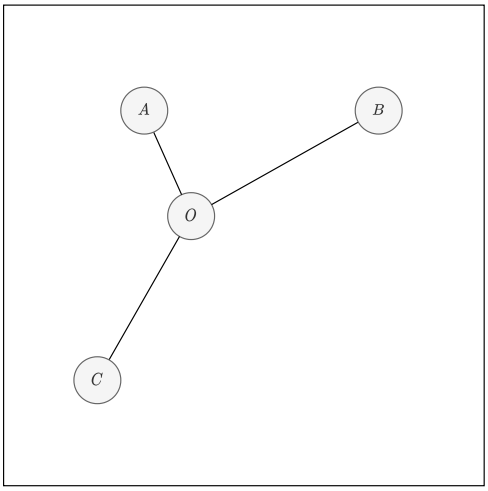
— Pour la structure «séparée», il suffit que tous les pivots générateurs possèdent le même point pivot parent sans spécifier de «facteur rayon».
— Pour la structure «mélangée», il suffit de réduire la distance entre deux points pivots ayant un même parent pour que leurs zones de calculs se superposent.
— Pour la structure «bulle», il faut que la distance entre deux groupes pA et pB soit considérablement réduite (voire nulle), que le rayon de $\mathcal{D}_{pA}$ soit inférieur à $\mathcal{D}_{pB}$, et enfin d'exclure $\mathcal{D}_{pA}$ de $\mathcal{D}_{pB}$.
— Pour la structure «proche», il suffit d'ajouter un niveau de profondeur dans la hiérarchie avec un pivot parent ayant au moins deux pivots enfants.

Concernant les intrus, nous nous contentons de changer aléatoirement l'étiquette d'un élément pour une autre **après** leur projection. Cette altération est donc propre à chaque résultat de projection plutôt que sur le jeu de données initial, car l'objectif G4 concerne la perception des intrus (au niveau de la visualisation) plutôt que leur prévention (au niveau de la projection) (N4).

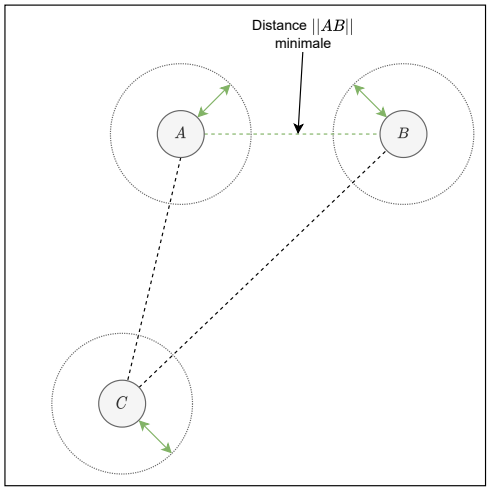
Afin d'éviter des biais de visualisation sur les jeux de données produits, l'ordre des étiquettes est mélangé afin que les couleurs utilisées ne soient pas les mêmes entre deux générations de configurations identiques. De même, l'ordre des éléments du jeu de données est mélangé afin d'éviter des biais au niveau de la projection et du dessin, notamment sur les superpositions d'éléments.

6.3.1 Implémentation

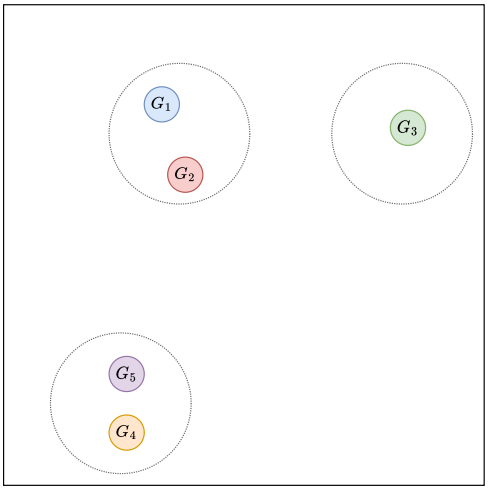
L'algorithme de génération est présenté en Figure 6.3, et son implémentation en C++ est disponible à l'adresse https://github.com/eikofee/grid-eval-data-generator. Les résultats des générations sont composés d'une part d'un fichier listant les coordonnées de chaque point



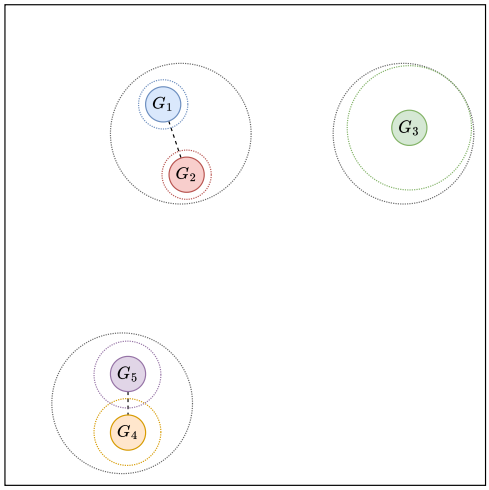
(a) Génération des pivots P_1



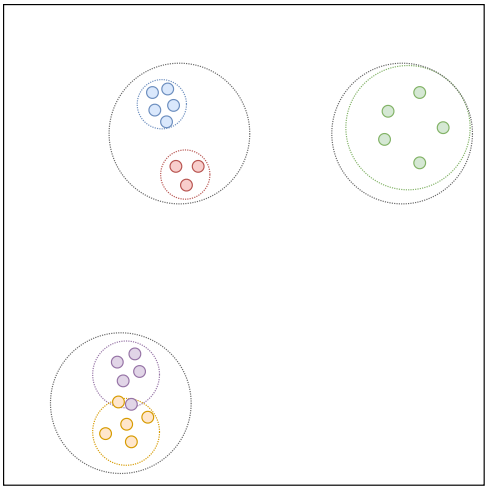
(b) Calcul des rayons des pivots P_1



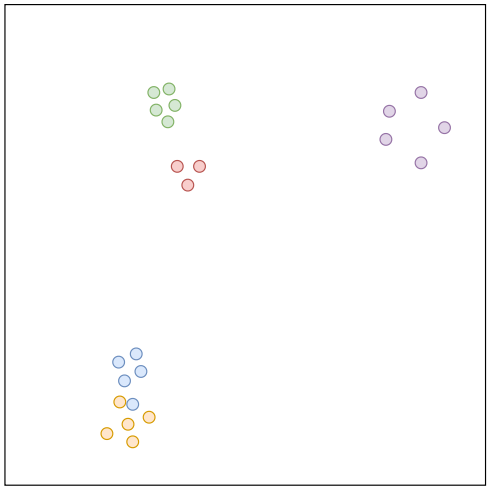
(c) Génération des pivots P_2



(d) Calcul des rayons des pivots P_2 (avec paramètres)



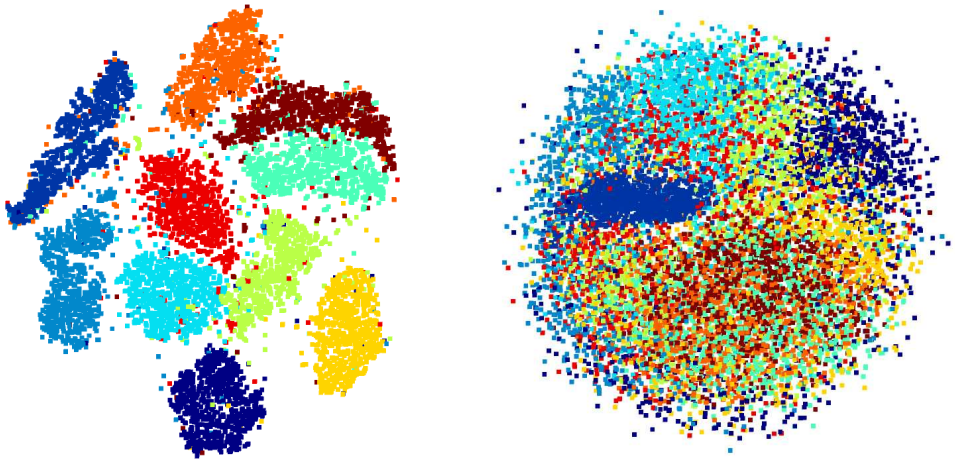
(e) Génération des points des groupes



(f) mélange des étiquettes et résultat final

FIGURE 6.3 – Processus de génération de données avec les mêmes paramètres donnés en Figure 6.2.

du jeu de données accompagnés de leur étiquette respective, d'une autre d'un fichier listant les



(a) Projection t-SNE de MNIST (b) Projection MDS de MNIST

FIGURE 6.4 – Comparaison de résultats de projections t-SNE et MDS du jeu de données MNIST, colorié par classe. La projection t-SNE permet de discerner les différentes classes d'éléments, mais la projection MDS est difficilement exploitable pour discerner les groupes.

coordonnées des points pivots choisis et leur distance par rapport aux autres points pivots, afin notamment de vérifier que la structure obtenue concorde bien avec celle souhaitée.

6.4 Méthodes de visualisation testées

Nous avons discuté dans les deux sections précédentes des tâches d'analyses que nous souhaitons évaluer, ainsi que notre méthode pour générer des jeux de données dans $\mathbb{R}^N$ pouvant respecter des topologies spécifiques. Nous pouvons maintenant conduire une évaluation entre une visualisation compacte et par nuage de points «classique».

6.4.1 Choix des méthodes de visualisation

Dans cette évaluation, nous avons choisi d'utiliser Self-Sorting Maps [128] (SSM) pour générer nos vues compactes, pour sa capacité à projeter des données de $\mathbb{R}^N$ directement en un environnement compact sans passer par une projection intermédiaire vers $\mathbb{R}^2$, et pour ses bonnes performances évaluées dans le chapitre précédent concernant la préservation des distances et du voisinage.

Pour ce qui est de la visualisation par nuages de points classique, nous proposons d'utiliser t-SNE [136] pour plusieurs raisons. La première concerne la nature des données visualisées dans l'évaluation, celles-ci concernent des ensembles de données répartis sous forme de groupes dans $\mathbb{R}^N$. Pour ce type de données, les méthodes de projections non uniformes, comme t-SNE, UMAP [82], etc. sont très utilisées dans la littérature par rapport aux méthodes de projections uniformes, comme PCA ou MDS, notamment pour leur capacité à représenter les structures complexes et non-linéaires [136].

La Figure 6.4 montre une comparaison de visualisation entre t-SNE et MDS pour le jeu de données MNIST [72]. On y voit que la visualisation non uniforme de t-SNE permet de discerner plus facilement les différents groupes de données, identifiables par leur couleur, par rapport à MDS. De plus, la nature des groupes qui sont très denses dans $\mathbb{R}^N$ implique un nombre très important de superpositions d'éléments (*overlaps*), qui ne sont pas souhaitables pour cette évaluation. Ensuite, malgré l'apparition d'alternatives non uniformes à t-SNE comme UMAP, l'utilisation de t-SNE occupe encore une majorité de travaux de visualisations.

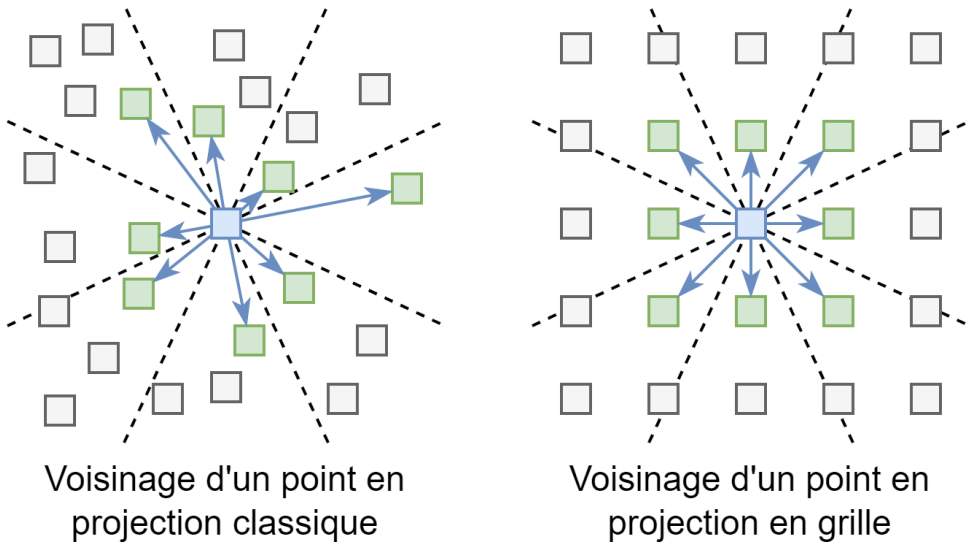


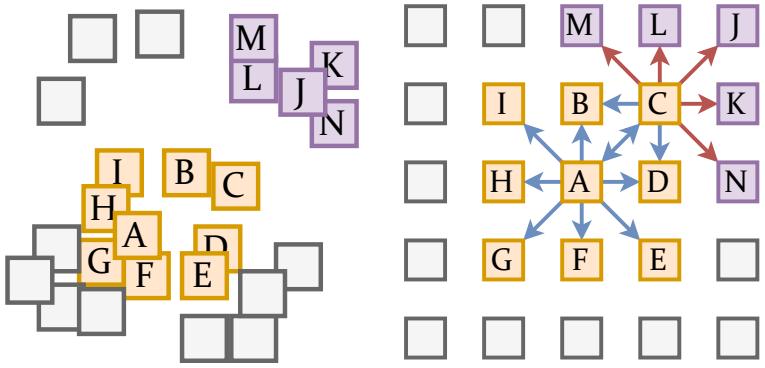
FIGURE 6.5 – Calcul pour un point donné (en bleu) de son voisinage dans $\mathbb{R}^2$ (en vert) à partir de secteurs angulaires.

6.4.2 Encodage des distances

t-SNE comme SSM sont des projections non uniformes, l'espace d'origine est fortement déformé afin de répondre aux besoins de la visualisation. Ainsi, deux distances entre deux paires d'éléments sur la visualisation ne peuvent pas être comparées (même si cette comparaison est généralement faite) car une distance sur la visualisation n'a pas la même signification que dans l'espace d'origine $\mathbb{R}^N$. Nous pensons qu'il est difficile pour l'utilisateur de répondre aux tâches T2 et T3 sans indication supplémentaire sur la nature des données.

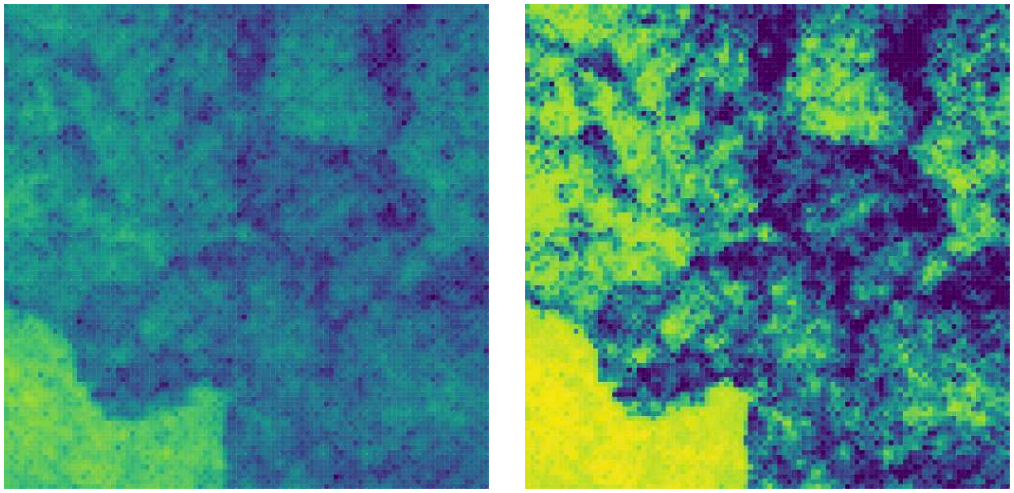
Pour combler ce manque d'information, nous proposons un encodage couleur supplémentaire à l'utilisateur informant sur les distances dans $\mathbb{R}^N$ entre un point et son voisinage. Nous calculons le voisinage $V(p)$ de chaque point p sur la projection colorée. Ce voisinage est composé des points les plus proches (s'ils existent) de p dans les 8 directions autour de p dans $\mathbb{R}^2$, comme indiqué en Figure 6.5 pour les deux méthodes de visualisations. Ensuite, nous calculons la moyenne des distances entre p et les points de $V(p)$ dans $\mathbb{R}^N$. Cette façon de calculer le voisinage permet entre autres d'obtenir des valeurs relativement faibles pour les points situés au milieu de groupe d'une même classe, et des valeurs plus grandes sur les points en bordure de ces mêmes groupes, comme montré en Figure 6.6.

Dans un premier temps, nous avons essayé de colorer les visualisations en calculant une échelle de couleurs linéaire entre les valeurs minimale et maximale obtenues, mais ce coloriage est difficilement exploitable car les valeurs calculées ne sont pas équitablement distribuées entre les deux extrêmes. Cette méthode donne une impression que l'échelle de couleur est fortement sous-exploitée, ce qui correspond à un problème connu de coloration de données dont la distribution n'est pas linéaire [115, 43]. Pour améliorer la lisibilité de notre coloration, nous basons notre échelle de couleur sur une quantification uniforme des distances moyennes calculées en calculant une dizaine de quantiles de l'ensemble de ces valeurs, en plus des extrêmes. La couleur appliquée à un élément dont la distance moyenne calculée est v est donc le résultat de l'interpolation linéaire entre les couleurs des quantiles a et b avec $a \leq v \leq b$. L'échelle de couleurs obtenue de cette façon n'est pas linéaire, mais permet de mieux exploiter le spectre de couleurs pour représenter les distances, tout en s'assurant que les valeurs proches des extrêmes aient leurs couleurs respectives. Un exemple d'application de cette méthode est présenté en Figure 6.7, où l'on peut voir une variation des distances plus prononcée sur cette méthode plutôt que la coloration en suivant une échelle linéaire.



(a) Exemple de données dans $\mathbb{R}^N$ (b) Visualisation en grille

FIGURE 6.6 – Exemple d'encodage de distances d'un jeu de données (a) dans $\mathbb{R}^N$ (avec $N = 2$ dans le cas présent) sur (b) une représentation en grille. Le voisinage est déterminé comme montré dans la Figure 6.5 et représenté par des flèches. Ici, le voisinage de l'élément A est composé d'éléments proches dans $\mathbb{R}^N$, donc la moyenne des distances entre A et ses voisins est relativement petite (représentées par les flèches bleues). En revanche, le voisinage de l'élément C est composé d'éléments en commun avec A, mais aussi avec des éléments d'un autre groupe dans $\mathbb{R}^N$. La moyenne des distances dans $\mathbb{R}^N$ entre C et ses voisins est donc plus grande que celle calculée pour A, et les deux éléments seront donc colorés différemment pour refléter ceci.



(a) Coloration des distances en utilisant une échelle de couleurs linéaire

(b) Coloration des distances en utilisant une échelle basée sur les quantiles

FIGURE 6.7 – Exemple de coloration des distances sur la projection du jeu de données MNIST dans une grille par SSM. Dans les deux cas, les couleurs claires représentent des distances moyennes courtes avec le voisinage de l'élément coloré, et les couleurs foncées les distances moyennes longues. En (a), la coloration est appliquée par interpolation linéaire sur l'échelle de couleur entre les distances minimum et maximum calculées. En (b), la coloration est appliquée en basant l'échelle de couleurs sur des quantiles des distances calculées plutôt que les deux extrêmes uniquement. Nous pouvons observer sur cette deuxième coloration une utilisation plus large de la palette de couleurs par rapport à la première, ce qui permet de mieux distinguer les distances.

6.4.3 Choix des questions pour les utilisateurs

Nous suivons le protocole présenté précédemment et générons un jeu de données par question. Nous souhaitons limiter le temps de l'évaluation à une heure maximum. Au-delà, les parti-

cipants pourraient ressentir une fatigue visuelle qui ferait défaut à la qualité de leurs réponses. Si l'on considère que notre évaluation compare 2 méthodes de visualisation en étant composée de 4 tâches et qui fait intervenir 2 méthodes de coloration, le tout impliquant 4 structures de données possibles, nous atteignons déjà $2 \times 4 \times 2 \times 4 = 64$ questions. Ce nombre élevé de questions est déjà problématique avant même de faire varier les paramètres de génération des jeux de données. Si l'on considère que chaque question demande entre 40 et 60 secondes au participant pour y répondre, nous dépassons rapidement la limite d'une heure fixée. Nous choisissons donc de limiter le nombre de combinaisons en fonction de leur pertinence par rapport aux tâches concernées. Par exemple, pour la tâche T2 (trouver le groupe le plus proche d'un autre), nous pensons qu'une question portant sur un jeu de données à la structure «séparée» n'est pas pertinente dans notre évaluation, car cette structure n'implique justement pas de proximité particulière entre les groupes de données. Aussi, nous définissons certains scénarios comme étant faciles où l'on s'attend à ce que le participant réponde rapidement (moins de 20 secondes) afin d'ajuster le nombre de combinaisons de paramètres incluses dans notre évaluation :

— T1 : La difficulté de la question varie en fonction de la différence de taille qu'il y a entre les deux plus grands groupes du jeu de données. Une grande différence résulte en une question facile et une petite différence résulte en une question difficile.
— T2 : La structure «séparée» n'est pas utile ici puisque la nature de la structure ne montre pas de proximité significative entre les groupes. Les autres structures nous semblent équivalentes en termes de difficulté lorsque les deux colorations sont montrées à l'écran.
— T3 : La différence entre les structures «séparées» et «proches» étant subtile, nous souhaitons prioriser l'évaluation sur ces structures plutôt que sur les structures «mélange» et «bulle», où les réponses nous semblent évidentes.
— T4 : Puisque nous comparons les méthodes de visualisation sur les mêmes nombres d'intrus, nous incluons dans cette tâche quelques questions de «bruit». Ces questions ne seront pas considérées dans l'étude des résultats et n'ont pour but que d'empêcher les participants de répondre par élimination en remarquant une redondance dans les questions. Nous n'avons pas non plus besoin de vérifier les résultats pour toutes les structures possibles.

En considérant ces ajustements, la Table 6.1 présente les combinaisons que nous avons utilisées pour générer les jeux de données, séparées par ensembles faciles et plus difficiles. La Figure 6.8 quant à elle présente des exemples de jeux de données générés en trois dimensions pour chacune des topologies étudiées, accompagnés des projections calculées par SSM et t-SNE. Ces projections montrent ainsi les différences que les participants peuvent observer lors de l'évaluation.

6.4.4 Hypothèses

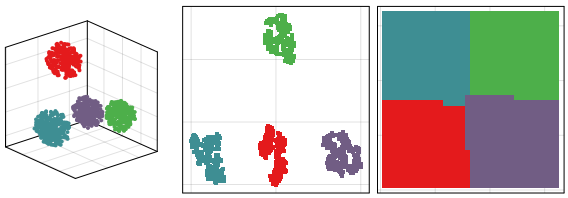
Nous formulons maintenant des hypothèses sur les capacités des visualisations testées pour remplir les quatre objectifs :

— **H1 : La comparaison des tailles des groupes est plus facile avec SSM qu'avec t-SNE.** t-SNE forme habituellement des partitions de points (*clusters*) variants en densité sur sa visualisation, mais leur taille relative aux autres partitions peut être facilement évaluée en observant leur superficie sans tenir compte de la densité. En revanche, si les tailles des groupes sont proches, nous pensons qu'il est plus difficile pour l'utilisateur de trouver lequel des groupes est le plus grand ou plus petit, notamment à cause de la superposition des points qui peut donc fausser la comparaison.
La visualisation obtenue de SSM quant à elle peut prendre plusieurs formes, mais la totalité des points est visible à l'écran. La probabilité d'erreur est donc plus faible. Pour appuyer notre hypothèse, nous pouvons compter informatiquement la quantité de pixels sur chaque vue our déterminer l'étiquette la plus présente à l'écran. Puisque la vue t-SNE possède des superpositions d'éléments, certaines informations sont cachées. Ce comptage de

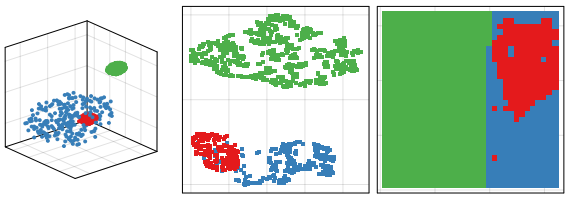
Ensemble		Paramètres					Total
Tâche	Set	# groupes	Diff. tailles	Structures [1]	# intrus	Colorations [2]	
T1	Facile	{3, 8}	Grande	Aléatoire	0	{}	12
	Difficile	{3, 5, 5, 6}	Faible	Aléatoire	0	{}	
T2		{5, 7}	Faible	{M, B, P}	0	{, +}	24
T3	Facile	{5}	Faible	{M, B}	0	{, +}	24
	Difficile	{5, 7}	Faible	{S, P}	0	{, +}	
T4	Comparable	Aléatoire	Faible	{S}	{3, 8, 12}	{}	10
	Factice	Aléatoire	Faible	{S}	{1, 5, 5, 14}	{}	

[1] S = «séparée», M = «mélangé», B = «bulle», P = «proche»
[2] = coloration par étiquettes, = coloration par distances

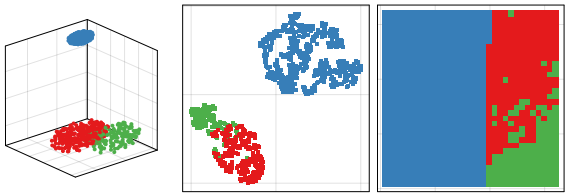
TABLE 6.1 – Paramètres utilisés pour la génération des jeux de données pour notre évaluation. Les colonnes «Ensemble» indiquent quelle tâche est concernée par les ensembles de questions, les colonnes «Paramètres» regroupent les paramètres modifiés pour chaque ensemble de questions, et enfin la colonne «Total» donne le nombre de questions pour cet ensemble. Le nombre de questions de chaque tâche est le résultat du produit du nombre d'arguments différents dans chacune des colonnes. Seul l'ensemble factice de la tâche 4 ne concerne pas les deux méthodes de projections évaluées.



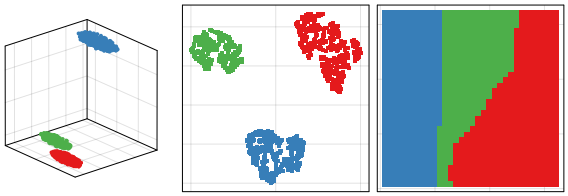
(a) Topologie «séparée»



(b) Topologie «bulle»



(c) Topologie «mélange»



(d) Topologie «proche»

FIGURE 6.8 – Exemples de générations de jeux de données pour chacune des topologies étudiées en 3 dimensions, avec leurs projections t-SNE et SSM.

pixels serait donc erroné pour la vue t-SNE, mais pas pour la vue SSM qui elle est dispensée de toute superposition d'éléments.

— **H2 : SSM est plus efficace que t-SNE pour étudier les distances entre groupes, et la coloration par distance rend la tâche plus facile.** En utilisant les résultats de t-SNE, il est difficile d'estimer une distance entre deux partitions de données, puisque la méthode est une projection non uniforme, ce qui signifie que les distances dans $\mathbb{R}^2$ au-delà du voisinage des points n'ont pas de signification. Nous supposons donc qu'à elle seule, ou même avec la coloration par distance proposée précédemment, cette méthode de visualisation n'est pas efficace pour remplir cet objectif.

SSM arrange les cellules de la grille en fonction des relations entre éléments. En observant les frontières communes entre les groupes d'éléments, il est possible de deviner la distance qui sépare ces groupes dans $\mathbb{R}^N$. Ainsi, les groupes ayant des frontières communes plus grandes montrent que ces groupes sont plus proches l'un de l'autre que deux groupes ayant des frontières communes plus petites ou même inexistantes. Pourtant, la méthode maximise l'utilisation de l'espace de dessin et n'affiche donc aucun espacement entre les groupes, qui est pourtant un attribut intuitif à l'utilisateur pour évaluer des distances. Il faut donc un temps d'adaptation à l'utilisateur pour pouvoir analyser ces frontières efficacement. En revanche, nous supposons que la coloration par distance facilite l'interprétation des distances en visualisation compacte.

— **H3 : Comme H2, la vue SSM est plus adaptée à l'étude des distances et donc par extension à la détection de topologies que t-SNE.** G2 et G3 sont des objectifs plus ou moins liés, puisque connaître les relations entre chaque groupe (donc les distances) permet de comprendre la structure globale des données, même si elle n'est pas explicitement représentée sur la projection.
Puisque nous supposons que t-SNE ne peut pas remplir G2 efficacement, nous supposons que la méthode ne peut pas remplir G3 efficacement non plus pour détecter les structures de données où tous les groupes sont séparés les uns des autres. Cependant, pour les structures où plusieurs groupes partagent un même domaine de valeurs (structures «mélange» et potentiellement «bulle»), nous pensons que t-SNE est capable de les représenter correctement par le biais de nuages de points se mélangeant partiellement. Utilisée conjointement avec la coloration par distances, nous pensons aussi que t-SNE est capable d'informer sur la densité des groupes affichés.
Comme SSM est déjà capable de représenter les distances entre groupes, nous supposons que cette méthode de visualisation a des capacités supérieures à celles de t-SNE sur cet objectif, et donc peut permettre de détecter un plus grand nombre de types de structures dans $\mathbb{R}^N$. Comme pour H2, nous supposons aussi que la coloration par distance permet de faciliter la compréhension des relations entre les points, et donc de faciliter aussi la détection de structures particulières.

— **H4 : Les vues compactes sont bien plus efficaces pour détecter les intrus que la vue t-SNE qui souffre de nombreuses superpositions.** La visualisation par nuage de points souffre fréquemment de superpositions d'éléments affichés, et donc d'occlusion d'informations. Parmi ces informations peuvent se trouver notamment des intrus, et nous supposons donc que cette méthode de visualisation n'est pas adaptée à remplir cette tâche efficacement.
En revanche, la vue compacte permet d'éviter toute superposition de points. Nous supposons donc que cette méthode de visualisation remplit parfaitement cet objectif.

6.4.5 Organisation de l'évaluation

Nous avons produit un questionnaire sous la forme d'un site internet afin que les évaluations puissent être conduites dans les navigateurs des participants, et donc dans un environnement familier.

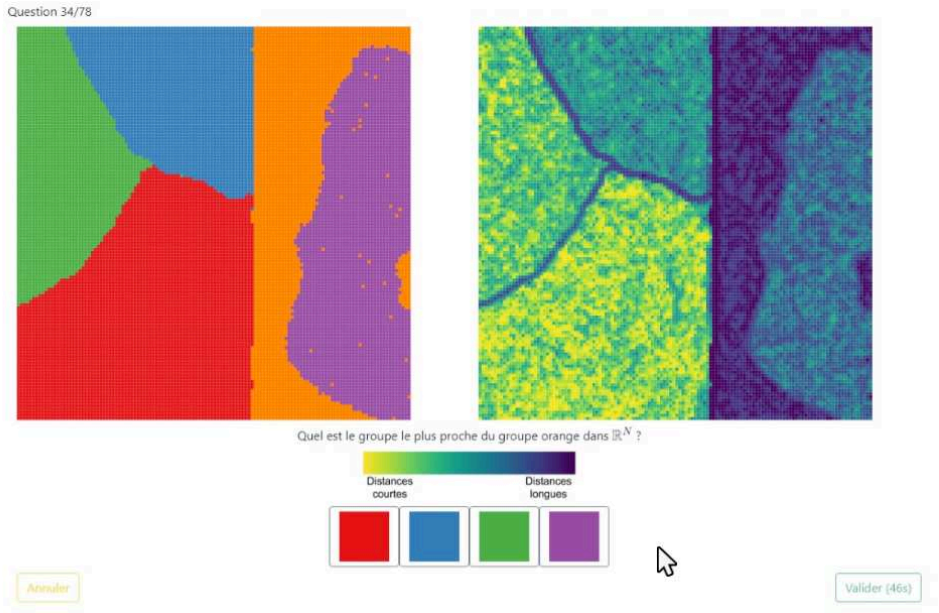
Concernant la coloration des données, nous utilisons deux colorations des points affichés à l'écran :

— La première est représentative des étiquettes des données projetées. Nous utilisons ici une échelle de 8 couleurs recommandée par l'outil ColorBrewer[1] [44] où chacune d'elles est attribuée à une étiquette, présentée en Figure 6.9.

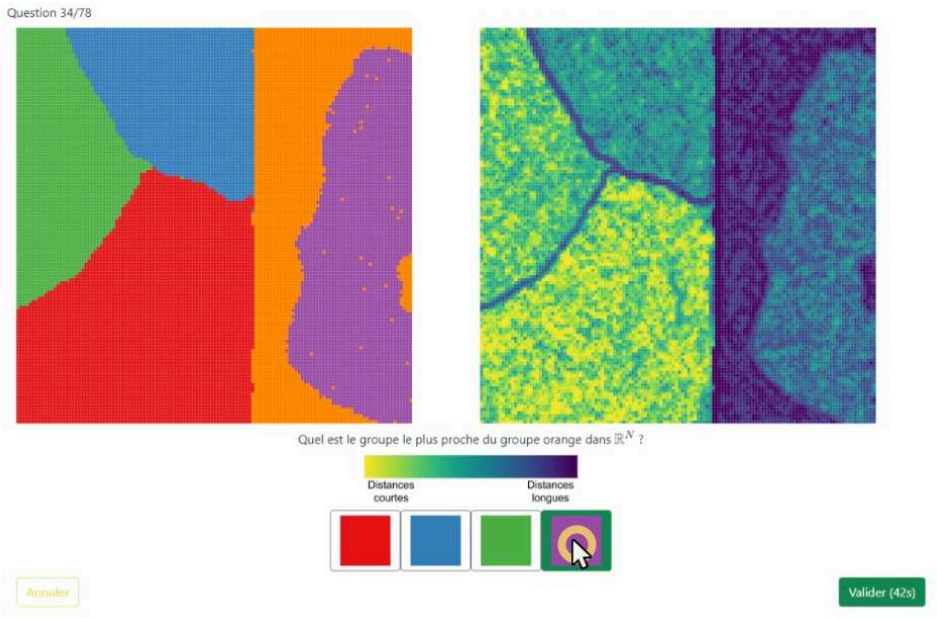
— La seconde est la coloration par distances et donc destinée à informer sur les distances dans $\mathbb{R}^N$ du jeu de données projeté. Nous proposons entre autres cette coloration pour assister l'utilisateur sur les objectifs G2 et G3, si nécessaires.

1. colorbrewer2.org/

FIGURE 6.9 – Échelle de couleurs «8-class Set1» proposée sur l'outil ColorBrewer [44].



(a) Exemple de question, le bouton «Valider» est bloqué.



(b) Lorsque la réponse est sélectionnée, le bouton «Valider» s'active.

FIGURE 6.10 – Exemple d'écran pour une question de l'évaluation.

Afin de répondre aux questions, les participants disposent d'un clavier visuel. Ils doivent valider leur réponse en deux temps comme montré en Figure 6.10 : un premier clic sur le bouton correspondant à la réponse voulue, puis un second sur un bouton «Valider». Ceci permet entre autre d'éviter les erreurs de manipulation. Toutes les visualisations sont présentées sous la forme d'images de taille 512×512 pixels. Nous avons choisi cette dimension d'image plutôt qu'un redimensionnement d'images s'adaptant à l'écran afin d'éviter que les navigateurs internet fassent du

Tâche	Ensemble		Tâche	Ensemble	
T1	Facile 1.00×10^{0}	Difficile 2.19×10^{-4}	T3	Coloration 5.03×10^{-1}	Coloration + 4.31×10^{-2}
T2	Coloration 1.52×10^{-10}	Coloration + 4.20×10^{-4}	T4	- 4.00×10^{-9}	

TABLE 6.2 – p-valeurs obtenues lors des tests statistiques de Wilcoxon-Mann-Whitney sur les résultats collectés sur les méthodes de visualisation t-SNE et SSM dans les différents sous-ensembles de questions. Les valeurs inférieures à 5×10^{-2} indiquent que les différences sont significatives. Inversement, les valeurs supérieures à 5×10^{-2} indiquent que les performances des méthodes n'ont pas de différence significative.

post-traitement pouvant dégrader la visualisation (par exemple, l'anti-crénelage qui provoque du flou sur l'image). Si besoin, les utilisateurs peuvent effectuer des grossissements sur les images, mais celles-ci conservent leur définition initiale.

À la fin de l'évaluation, nous demandons aux participants de répondre à un questionnaire [2]. Ce questionnaire est séparé en plusieurs parties, portant chacune sur une tâche de l'évaluation. Pour chacune de ces tâches, le participant doit indiquer s'il a trouvé facile ou difficile l'utilisation de chacune des méthodes de visualisation (avec ou sans coloration supplémentaire pour les tâches T2 et T3) pour répondre aux questions. Lorsque la coloration par distance est proposée, le participant doit aussi indiquer s'il a trouvé cette vue annexe utile ou non. Ces ressentis sont exprimés sur une échelle allant de 1 (difficile) à 4 (facile). Cette même échelle est aussi utilisée pour évaluer l'utilité de la coloration par distance. Après avoir évalué la facilité d'utilisation des méthodes, les participants doivent décrire au mieux la stratégie qu'ils ont employée avec les visualisations proposées pour répondre aux questions. Pour terminer, le questionnaire dispose d'un champ de texte afin que les participants puissent s'exprimer plus précisément sur leurs ressentis ou faire part de leurs observations. Enfin, pendant l'évaluation même, les participants sont observés pour relever des comportements passés comme anodins par l'utilisateur (utilisation de grossissement ou rapprochement de l'écran par exemple).

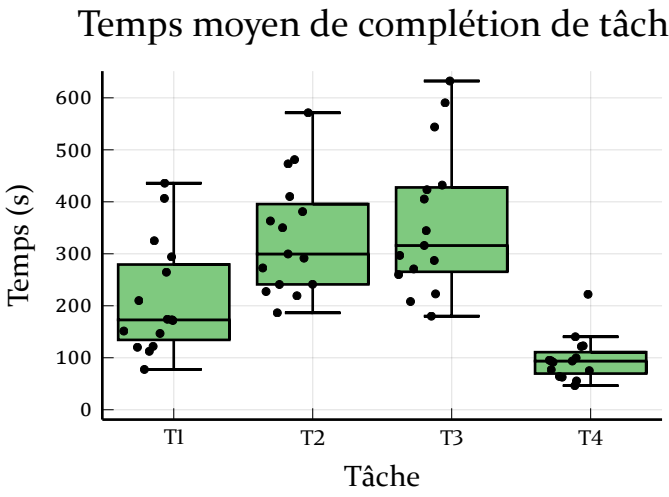
6.5 Résultats et discussion

6.5.1 Résultats globaux

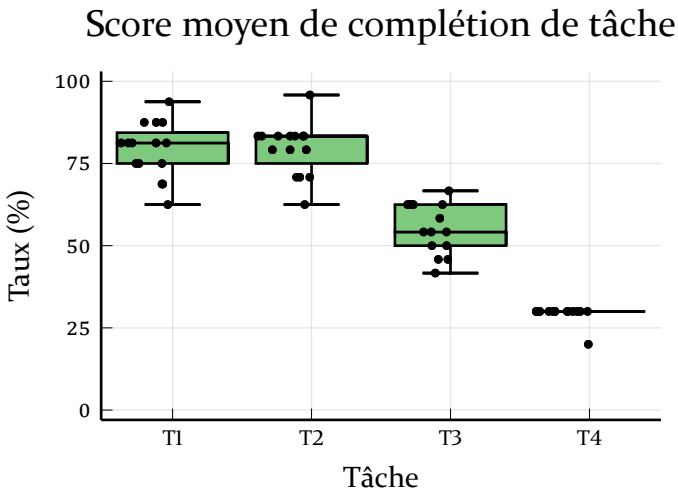
L'évaluation a été conduite sur 15 participants, ayant une expérience variable sur la visualisation d'information et/ou les données à hautes dimensions. Les temps de complétion moyens et le taux de réussite moyen pour chacune des tâches sont montrés en Figure 6.11. En incluant le temps de lecture de l'introduction et des exercices précédant l'évaluation, les participants étaient sollicités pendant environ 1 heure. On remarque que le taux de réussite sur les tâches T3 (recherche de structure) et T4 (recherche des intrus) sont bien moindres par rapport aux tâches T1 (comparaison de tailles) et T2 (comparaison de distances). En revanche, on remarque aussi une faible variation des résultats sur la tâche T4 par rapport aux autres tâches.

Pour ce qui est de la significativité des différences entre les techniques t-SNE et SSM, nous avons conduit une vérification statistique basée sur le test signé de Wilcoxon-Mann-Whitney [146] comme dans le Chapitre 5. Si les valeurs obtenues sont inférieures à 5×10^{-2}, alors les différences entre les deux méthodes sont significatives. Ces valeurs sont présentées en Table 6.2. Nous avons deux valeurs supérieures à 5×10^{-2} pour les comparaisons de performances entre t-SNE et SSM, la première sur la tâche T1 concernant les questions que nous avons considérées comme faciles et

2. Disponible à l'adresse https://docs.google.com/forms/d/1FBKr265tTysj-AqpUm6c4dQiatKFrDPaGAdyVPeNHHE



(a) Temps moyen consacré à la réponse aux questions par tâche des participants.



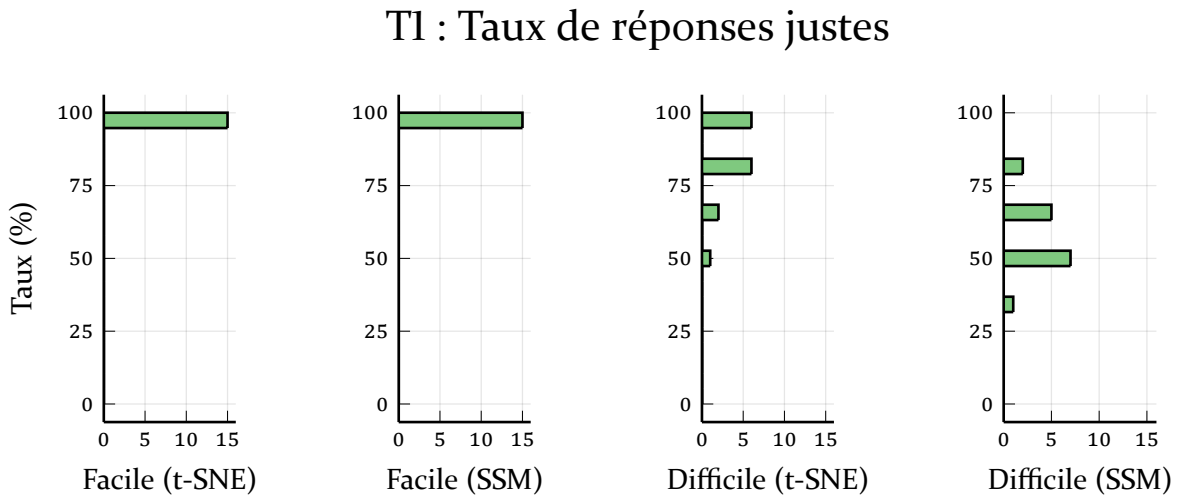
(b) Taux de réussite moyen aux questions par tâche des participants.

FIGURE 6.11 – Temps moyen de réponse aux questions et taux de réussites sur chaque tâche par les participants. Les tâches T1 et T2 ont globalement été mieux réussies que les tâches T3 et T4, mais on remarque aussi que la variance des résultats en T4 est faible.

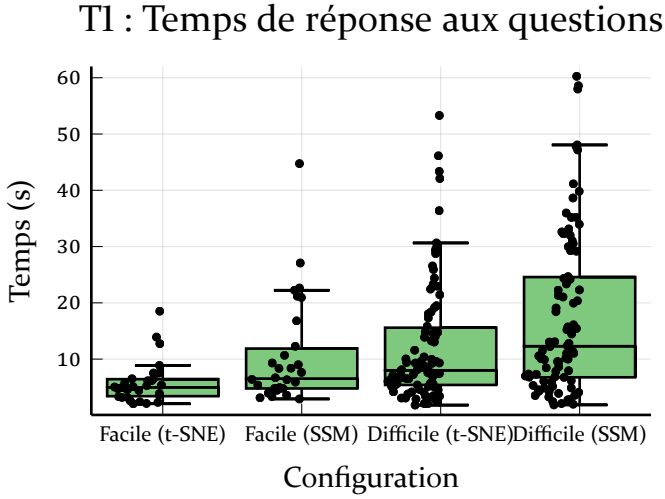
la seconde sur la tâche T3 concernant l'étude de structures dans $\mathbb{R}^N$ en disposant uniquement de la coloration par étiquettes.

6.5.2 Tâche 1 : Trouver la classe de données la plus représentée

La Figure 6.12 récapitule les bonnes réponses données par les participants, ainsi que le temps de réponse moyen des participants, en fonction de la méthode de visualisation utilisée et de la difficulté de la question. Comme indiqué précédemment, il n'y a pas de différence significative entre les performances de SSM par rapport à t-SNE. En revanche, nous constatons aussi que tous les participants ont répondu correctement aux questions faciles, quelle que soit la méthode de visualisation employée. Cependant, sur les questions difficiles, t-SNE s'est montré plus efficace que SSM sur cette tâche. Les participants ont passé plus de temps à étudier les visualisations pour les questions difficiles par rapport aux plus faciles. On remarque aussi que sur les questions



(a) Taux de réussite moyen classé par méthode et par difficulté.



(b) Temps consacré à chaque question.

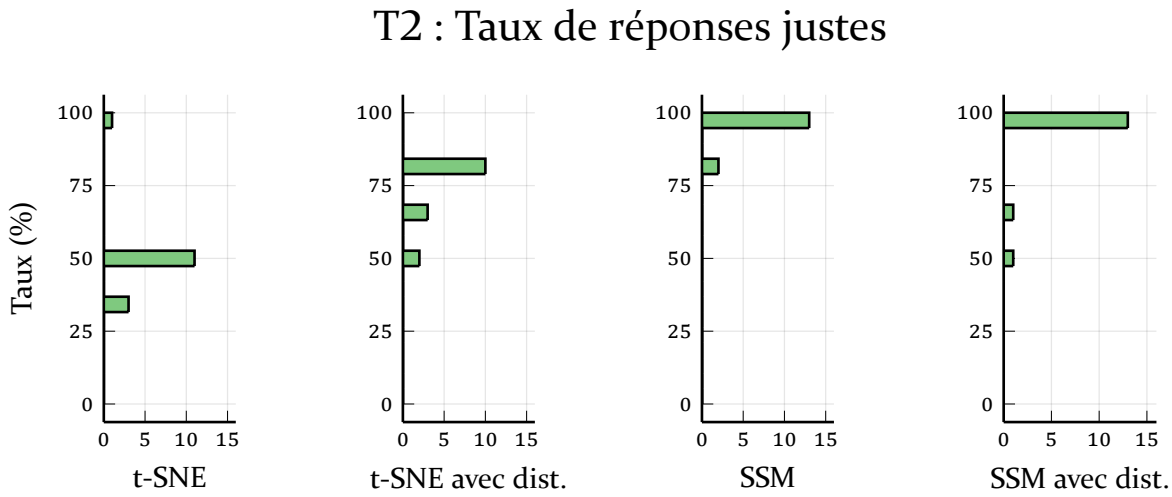
FIGURE 6.12 – Résultats de l'évaluation utilisateurs sur la tâche T1.

difficiles, les participants ont légèrement passé plus de temps à étudier les visualisations SSM que celles de t-SNE.

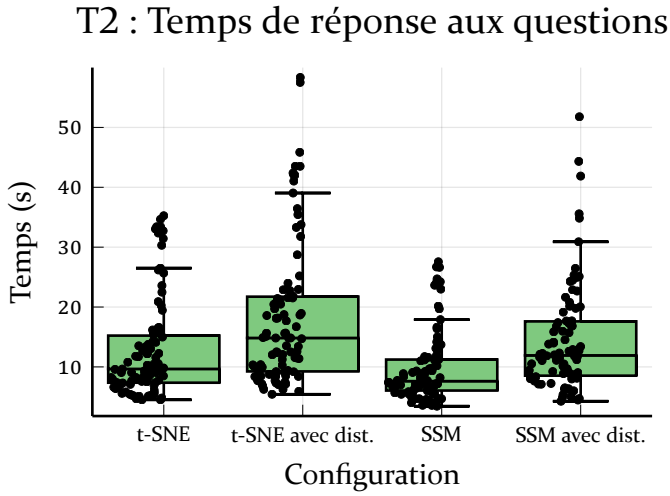
L'efficacité des deux méthodes ne peuvent donc pas être comparées sur les questions faciles, et les résultats obtenus par SSM sur les questions plus difficiles ne s'alignent pas avec nos hypothèses de départ. Après étude des réponses individuelles, nous pensons que la forme des vues produites par SSM a un rôle à jouer sur la facilité de comparaison des groupes, mais il faudrait plus de questions sur cette tâche pour que cette hypothèse soit vérifiée. Nous observons aussi que la différence paramétrique entre nos questions faciles et difficiles est trop abrupte ; nous nous attendions à obtenir une légère différence de performances obtenues sur les questions facile, amplifiée sur les questions difficiles, mais ce n'est pas le cas ici.

6.5.3 Tâche 2 : Trouver le groupe le plus proche d'un autre dans $\mathbb{R}^N$

Les résultats obtenus sur cette tâche sont statistiquement comparables d'après les valeurs obtenues en Table 6.2. La Figure 6.13 montre les résultats obtenus par les participants, ainsi que leur temps de réponse moyen, séparés par méthode de visualisation et colorations fournies. Les visualisations produites par SSM ont été les plus efficaces pour répondre aux questions posées



(a) Taux de réussite moyen classé par méthode et par coloration.



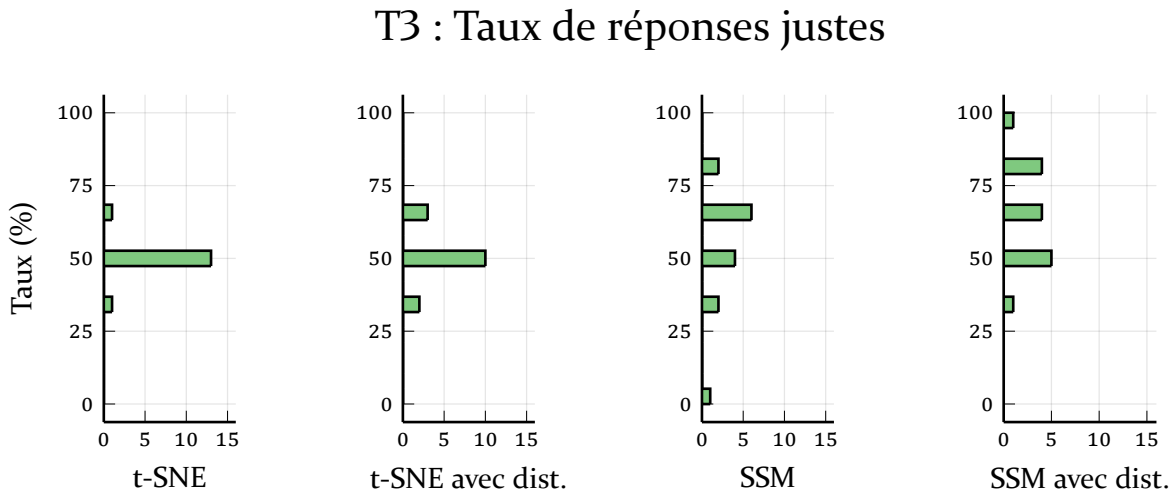
(b) Temps consacré à chaque question.

FIGURE 6.13 – Résultats de l'évaluation utilisateurs sur la tâche T2.

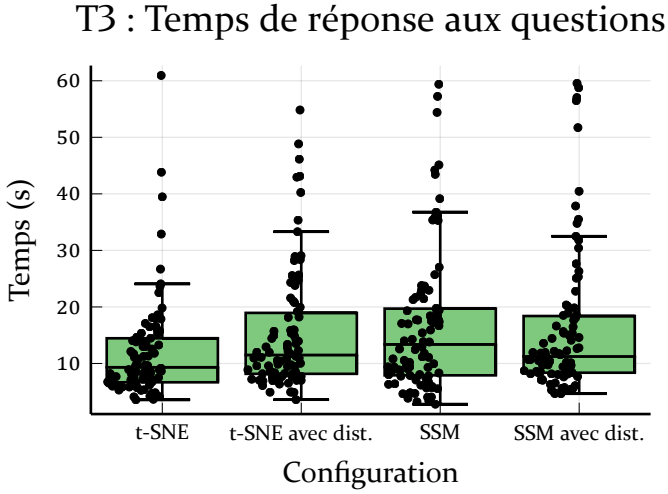
aux participants, avec une majorité d'entre eux ayant obtenu un sans-faute, que ce soit avec ou sans la coloration par distance. Nous remarquons cependant que le taux de réussite de certains utilisateurs est plus faible lorsque la coloration par distance était proposée que sans, ce qui n'est pas en accord avec nos hypothèses de départ. Pour ce qui est de t-SNE, les résultats montrent que la coloration par distance a aidé les participants à répondre correctement à la question, bien que, comme spécifié en hypothèse H2, t-SNE n'a pas de rétention des distances longues des éléments entre eux. Ainsi, si le placement des groupes respecte le positionnement des données dans $\mathbb{R}^N$, la coloration par distance s'avère être utile, mais n'a pas permis aux participants d'obtenir de sans-faute. Les utilisateurs ont passé moins de temps à étudier les visualisations sans la vue par distance qu'avec, mais dans leurs cas respectifs, les utilisateurs ont passé moins de temps à étudier les visualisations obtenues à partir de SSM que celles à partir de t-SNE.

6.5.4 Tâche 3 : Retrouver la structure des données dans $\mathbb{R}^N$

La Figure 6.14 récapitule les résultats obtenus auprès des participants, ainsi que leur temps de réponse moyen, regroupés par méthode de visualisation et de colorations fournies. La Figure 6.15 quant à elle montre plus en détail les réponses données par les participants. Les scénarios de visua-



(a) Taux de réussite moyen classé par méthode et par coloration.



(b) Temps consacré à chaque question.

FIGURE 6.14 – Résultats de l'évaluation utilisateurs sur la tâche T3.

lisation étudiés sont indiqués en abscisse avec le type de visualisation utilisé ainsi que la topologie réelle des données visualisées. La proportion des cas où les participants ont détecté correctement la topologie des données est indiquée en vert. Les proportions des cas où les réponses ont été incorrectes sont indiquées par des couleurs en fonction des mauvaises réponses données ; en bleu foncé pour la structure «bulle», jaune pour «proche», orange pour «mélange» et violet pour «séparée». Nos tests statistiques ont montré que les performances des méthodes de visualisation ne sont pas comparables dans le cas où seule la coloration par étiquette était fournie. On peut remarquer dans ces résultats que les deux méthodes ont leurs faiblesses pour la détection de structures en particulier : Pratiquement aucun participant n'a détecté les structures «proche» (confondue avec «séparée») et «bulle» (équitablement confondue avec les trois autres structures), tandis que «mélange» et «séparée» ont été détectées pratiquement à tous les coups. Concernant SSM, les structures «bulle», «proche» et «mélange» ont été globalement bien détectés par les participants (autour de 70% de réussite), cependant la structure «séparée» a été très difficile à détecter pour les participants (confondue majoritairement avec «proche»).

Lorsque la coloration par distance est fournie, les résultats sont meilleurs pour les deux méthodes de visualisations, sans obtenir de sans-faute pour autant. Concernant t-SNE, la structure «bulle» n'a pas été détectée une seule fois (encore une fois confondue équitablement avec les

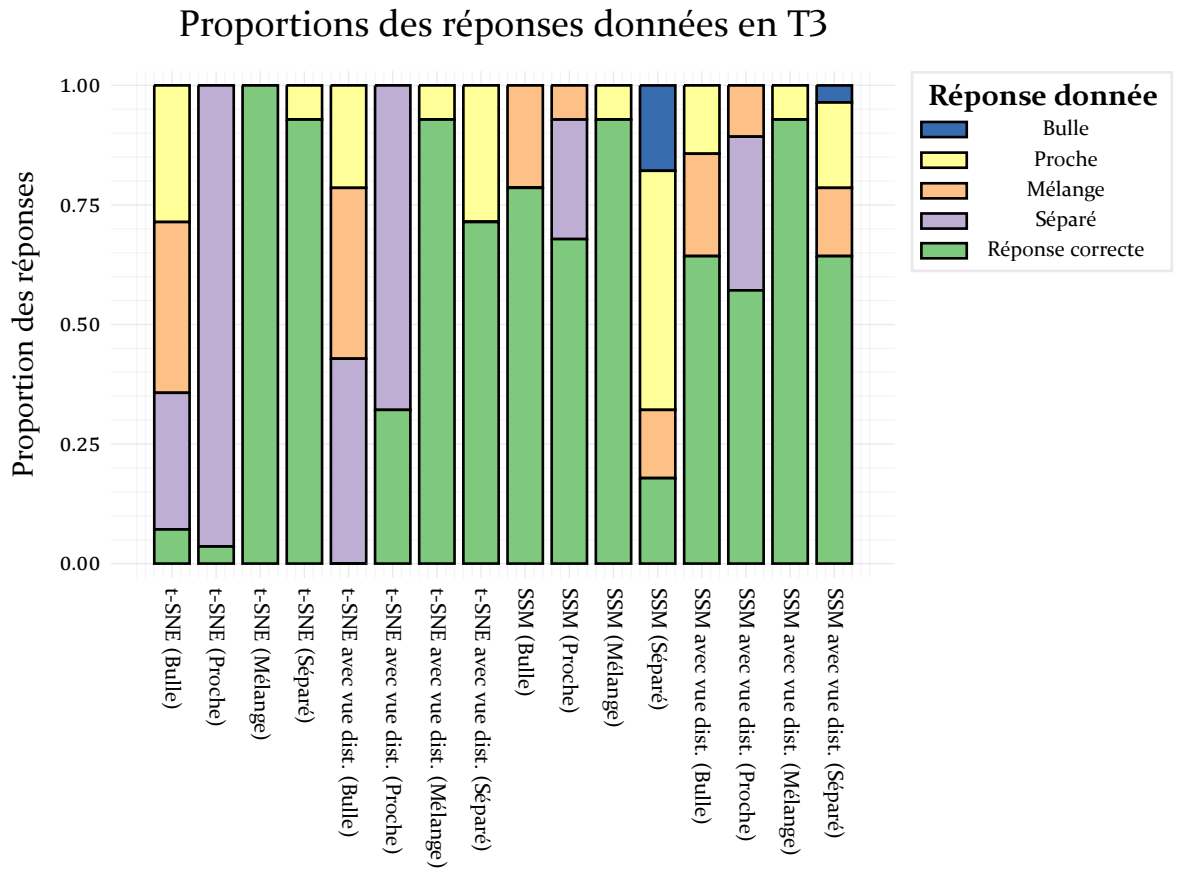


FIGURE 6.15 – Réponses données par les participants aux questions de la tâche T3, triées par type de topologie projetée.

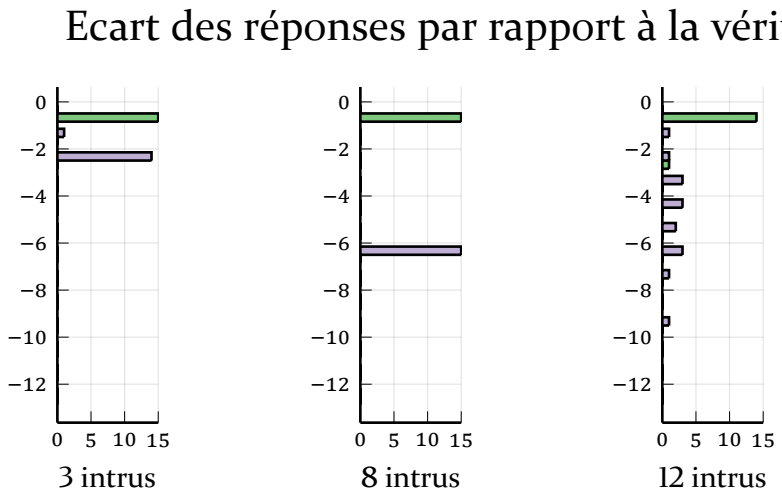
trois autres structures), mais plus de participants ont trouvé la structure «proche» par rapport à lorsqu'ils ne disposaient que de la coloration par étiquette (les autres l'ayant toujours confondue avec «séparée»). Concernant SSM, la distinction entre les structures «proche» et «séparée» est meilleure que sans la coloration par distance, mais la détection des autres structures n'a pas eu d'amélioration significative.

Pour ce qui est du temps de réponse aux questions, celui-ci reste à peu près équivalent entre toutes les configurations, mais on remarque que dans quelques cas, des participants ont passé presque 60 secondes à étudier les visualisations, que ce soit pour t-SNE ou pour SSM.

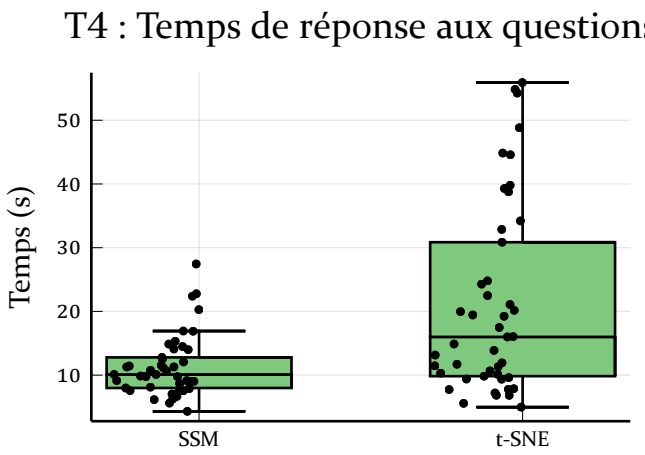
Notre hypothèse de départ n'est donc que partiellement vérifiée ; seule, la coloration par étiquette ne permet pas à SSM de se distinguer de t-SNE. La visualisation d'information par SSM seulement accompagnée d'une coloration par distance ne permet pas de faire efficacement la distinction entre des structures «proche» et «séparée». Accompagnée de la coloration par distance, cette confusion disparaît partiellement, mais la détection des autres structures n'est pas spécialement améliorée.

6.5.5 Tâche 4 : Trouver le nombre d'intrus dans un groupe

Les résultats et temps moyen de réponses des participants sur la Tâche T4 sont présentés en Figure 6.16, triés par nombre d'intrus prédéfinis dans chaque visualisation. La comparaison entre les deux méthodes est possible puisque les résultats sont statistiquement différents.



(a) Écarts des réponses des participants à la vérité, classées par nombre d'intrus à trouver à la tâche T4, utilisant la visualisation SSM en vert, et t-SNE en mauve.



(b) Temps consacré à chaque question en fonction de la méthode employée.

FIGURE 6.16 – Résultats de l'évaluation utilisateurs sur la tâche T4.

Sur les visualisations compactes de SSM, seul un participant a commis une erreur, tous les autres ont obtenu un sans-faute. En revanche, sur la visualisation par nuage de points de t-SNE, au moins un intrus a été omis à pratiquement chaque question.

Pour ce qui est du temps de réponse moyen, on voit clairement que les participants ont passé moins de temps pour détecter correctement les intrus sur la vue compacte que sur la vue par nuage de points.

Notre hypothèse est donc ici vérifiée, mais il faut noter que les résultats obtenus n'ont pas spécialement de variance : sur les visualisations t-SNE, la majorité des participants ont compté le même nombre d'intrus, on peut donc imaginer que les questions posées étaient particulièrement problématiques, et qu'il aurait fallu proposer plus de questions afin de confirmer les performances de t-SNE. En revanche, nous pensons aussi que ces résultats sont parfaitement attendus, puisque SSM n'a aucune superposition d'éléments alors que t-SNE en a bien plus.

6.5.6 Ressenti global

Comme indiqué en Section 6.4.5, nous avons demandé aux participants leur ressenti sur la facilité d'utilisation des différentes méthodes de visualisation et de coloration après avoir com-

Visualisation	T1	T2	T3	T4
t-SNE	**3.07**	1.64	2.71	2.21
t-SNE avec coloration par distances	-	3.07	3.14	-
SSM	2.42	2.21	2.21	**3.85**
SSM avec coloration par distances	-	**3.42**	**3.28**	-

TABLE 6.3 – Indice de facilité moyen donné par les participants sur les différentes méthodes et colorations en fonction des tâches. L'indice varie entre 1 pour «Difficile» à 4 pour «Facile». Les valeurs en gras indiquent que la méthode est, d'après les participants, la plus facile pour répondre à la question.

plété les quatre tâches. La moyenne de ces résultats est présentée en Table 6.3. Les participants pouvaient noter les méthodes pour chaque tâche avec un chiffre entre 1 et 4 inclus, respectivement entre difficile et facile d'utilisation. Les participants n'ayant pas connaissance de la justesse de leurs réponses, il s'agit donc d'un indice décrivant l'intuitivité de la méthode pour la tâche donnée.

Sur ce sondage, nous pouvons voir que la vue t-SNE est ressentie comme plus facile à étudier pour comparer les tailles de groupes (T1) et détecter des topologies de données (T3). En revanche, sur la comparaison de distances (T2) et la détection d'intrus (T4), SSM a été plus appréciée. Pour les deux méthodes, la présence de la coloration par distances a facilité l'étude des données pour répondre aux questions des tâches T2 et T3, avec une préférence pour la visualisation compacte de SSM.

Sur les différentes tâches, la plupart des participants ont employé la même stratégie :

T1 : Trouver la classe de données la plus représentée. Les participants ont évalué la taille des groupes principalement au premier coup d'œil sur le dessin. Pour la visualisation compacte, des participants ont essayé de calculer les aires ou de réarranger mentalement les zones de couleurs pour être facilement comparables. Quelques participants ont indiqué préférer les séparations des groupes sur la visualisation par t-SNE, mais ont eu des doutes sur certaines de leurs réponses notamment à cause de la densité des points formant les groupes. En revanche, ils ont aussi indiqué que lorsque les formes n'étaient pas découpées orthogonalement, la tâche était très difficile sur la vue compacte.

T2 : Trouver le groupe le plus proche d'un autre dans $\mathbb{R}^N$. Des participants ont indiqué n'avoir aucune certitude dans leurs réponses lorsque la vue t-SNE sans coloration par distance était proposé. Ils cherchaient notamment des potentiels points se mélangeant dans d'autres groupes pour déterminer leur proximité. D'autres ont comparé les distances séparant les groupes, mais cette stratégie n'est pas correcte puisque t-SNE ne place pas les groupes en fonction de cette information. Lorsque la coloration par distance était jointe à la visualisation t-SNE, les participants ont majoritairement comparé la couleur des frontières délimitant les groupes. Pour ce qui est de la visualisation proposée par SSM, les participants se sont beaucoup appuyés sur les frontières communes des groupes pour déterminer la proximité. En cas de point présent dans un autre groupe, de manière similaire à un intrus, des participants ont considéré qu'il s'agissait d'un indice sur la proximité des groupes prévalent sur les autres observations, même s'il s'agit d'un artéfact de l'algorithme. Pour une partie des participants, la coloration par étiquette était suffisante pour répondre aux questions, et s'est servie de la coloration par distance pour confirmer leurs hypothèses. Pour une autre partie en revanche, elle s'est avérée être indispensable.

T3 : Retrouver la structure des données dans $\mathbb{R}^N$. Les stratégies utilisées par les participants sont assez différentes pour cette tâche. La majorité s'accorde en revanche sur la recherche des structures de type «mélange» avant de chercher d'autres indices. Pour les deux méthodes de visualisation, la coloration par distance a été principalement sollicitée pour trouver d'éventuels

indices sur les proximités entre les groupes. Les participants ont trouvé qu'il était difficile de discerner les topologies «proches» et «séparé» dans les deux méthodes de visualisation.

T4 : Trouver le nombre d'intrus dans un groupe. Tous les participants se sont rapprochés de l'écran ou ont utilisé le grossissement de leur navigateur pour étudier les visualisations de t-SNE, et beaucoup ont hésité à compter certaines taches de couleurs qu'ils confondaient avec une erreur de crénelage sur l'image. Pour ce qui est de la visualisation compacte, les participants ont compté les intrus en faisant un balayage dans une direction de l'image, et ont pu vérifier leurs comptes une fois ou deux de plus avant de valider. Les participants ont relevé la difficulté de répondre aux questions avec la visualisation t-SNE, et une partie d'entre eux ont également indiqué que les couleurs utilisées complexifiaient la tâche lors de certains mélanges, comme des intrus violets dans un groupe bleu.

6.6 Discussion

Dans cette section, nous faisons le point sur ce qui a été relevé dans cette évaluation, et nous discutons des éléments à revoir dans notre protocole d'évaluation.

6.6.1 Résultats d'évaluation SSM/t-SNE

Notre évaluation a pu confirmer plusieurs des hypothèses énoncées en Section 6.4.4, notamment sur les meilleures capacités de SSM pour étudier les distances dans $\mathbb{R}^N$ ainsi que la détection d'intrus. Cependant, cette méthode reste moins efficace que t-SNE pour l'étude de la taille des groupes, par manque d'habitude ou d'intuitivité, et la détection de structures en tâche T3 n'est pas statistiquement différente que celle de t-SNE si la coloration par distance n'est pas fournie.

Les retours des participants démontrent que les stratégies employées diffèrent sur les tâches concernant l'étude de distances et de topologie sur la visualisation SSM, mais restent similaires sur la visualisation t-SNE. De plus, malgré le meilleur taux de réussites des participants sur ces tâches, ceux-ci n'ont pas trouvé la visualisation proposée par SSM particulièrement plus faciles d'utilisation. On peut en conclure que les utilisateurs d'une visualisation compacte de la sorte doivent au préalable apprendre à l'utiliser avant de pouvoir l'exploiter efficacement car ces résultats ne correspondent pas à nos hypothèses.

Pour ce qui est de la coloration par distances, les retours varient entre les participants. Certains l'ont trouvée indispensable pour répondre aux questions tandis que d'autres y trouvaient que peu d'utilité. En plus de cela, lorsque proposée en supplément de la visualisation compacte, nous avons pu constater une légère dégradation des performances des participants sur les structures «proche» de données, où l'on peut supposer que la proximité n'était pas suffisamment mise en évidence sur la coloration par distances. La technique de coloration doit donc être approfondie afin de se montrer plus efficace.

Les résultats obtenus nous orientent vers un nouvel ensemble d'hypothèses à vérifier :

— **La forme des données projetées porte une influence sur l'évaluation des tailles des groupes**. Les résultats obtenus en T1 montrent que la visualisation t-SNE est plus efficace que SSM pour comparer la taille des groupes de données. Pourtant, la visualisation t-SNE possède des éléments superposés qui empêchent de distinguer l'intégralité des éléments d'un même groupe pour pouvoir les comparer à un autre, ce que SSM n'a pas. Il y a donc un facteur supplémentaire favorisant la comparaison des tailles des groupes sur t-SNE qui n'est pas présent sur SSM. Il faut donc approfondir cette question avec des paramètres de génération de jeux de données supplémentaires.

— **Une coloration par distance plus robuste permet de faciliter la différentiation des structures «séparé» et «proche» avec SSM**. Les résultats obtenus en tâche T3 sont encourageants pour l'étude de structures de jeux de données dans $\mathbb{R}^N$, mais le nombre de questions proposées dans cette évaluation est trop faible pour observer une nette amélioration : les jeux de données utilisés sont différents entre les questions avec et sans la

coloration par distance, il faudrait donc revoir notre protocole d'évaluation pour confirmer cette hypothèse.

— **Pour un même jeu de données, SSM est plus efficace que t-SNE pour détecter les intrus dans des groupes**. Cette hypothèse est similaire à H4, mais la vérification de celle-ci devrait être proposée sur des mêmes jeux de données projeté, mais avec un nombre bien plus grand de questions pour avoir une comparaison plus significative et moins prône à des cas difficiles comme nous avons possiblement eu dans cette évaluation. Seulement, un trop grand nombre de questions portant sur cette tâche peut impliquer des facteurs indésirables chez les participants, comme de la fatigue oculaire. Il nous faut donc revoir notre protocole d'évaluation pour mieux vérifier cette hypothèse.

6.6.2 Protocole d'évaluation

Les résultats de l'évaluation ont montré peu de variance sur certaines tâches, comme l'étude des distances (T2) ou la détection d'intrus (T4), et n'ont pas permis de montrer des différences significatives entre les méthodes SSM et t-SNE, et ce pourtant malgré le nombre conséquent de questions au total, et le temps nécessaire pour la complétion de l'évaluation proche d'une heure. Afin d'obtenir des résultats plus exploitables, il faudrait augmenter le nombre de participants de manière significative, ou bien proposer un questionnaire «inversé» à un même nombre de participants, où les jeux de données générés sont les mêmes mais en générant la visualisation avec l'autre technique de projection (*Between-participant experiment*) [112]. De la même manière, nous pourrions séparer les participants entre ceux qui disposent de la coloration par distances et ceux qui ne l'ont pas, mais cette séparation nécessite un nombre plus important de participants. Une autre solution serait de concentrer les participants sur une ou deux tâches en particulier, plutôt que les quatre. Ce processus pourrait permettre de couvrir plus de cas de visualisations à la fois, et ainsi donc obtenir des résultats plus significatifs. Ceux-ci seront cependant moins «entraînés» au préalable avant de rencontrer les tâches difficiles (par exemple, la tâche T3 sur l'étude de topologie est plus difficile que la comparaison de taille (T1) ou de d'étude des distances (T2), mais les participants ont pu se familiariser avec les visualisations sur une quarantaine de questions avant d'y être confrontés).

6.7 Vers une nouvelle évaluation utilisateur

Cette première évaluation nous a permis de comparer deux méthodes de visualisations sur de multiples tâches, mais nous avons pu observer qu'il était difficilement possible d'avoir des résultats fiables en un temps de test raisonnable si l'on essaye de couvrir toutes les possibilités de structures de données proposées en Section 6.3. De plus, la dernière tâche utilisateur concernant la détection d'intrus demande bien plus de tests avant de pouvoir être comparés avant d'obtenir un résultat qui nous semble pourtant évident. Dans cette section, nous développons nos idées d'amélioration à apporter au protocole de comparaison de méthodes de visualisation.

6.7.1 Automatisation de la tâche T4

La faible variabilité des résultats obtenus sur la tâche T4 (recherche d'intrus) est notamment due au faible nombre de questions posées aux participants. Cependant, simplement augmenter le nombre de questions aux participants risquerait de nuire à la qualité des résultats obtenus ; en effet, compte tenu de la durée déjà longue de l'évaluation, et prenant en compte l'accumulation de fatigue des participants pour répondre aux questions, nous risquerions d'avoir plus d'erreurs dues à la baisse d'attention des participants qu'à cause de la méthode d'évaluation en elle-même.

À la place, nous proposons de faire passer le test à un programme informatique automatiquement plutôt qu'à un humain. Ce programme aura pour tâche de détecter les intrus à l'écran en utilisant les mêmes images générées pour les humains. De ce fait, le nombre d'intrus retourné

par le programme concernera seulement les superpositions ambiguës et complètes des intrus, et donc reflèteront les performances des méthodes de visualisations que nous cherchons à mesurer. Il a été de plus montré que conduire des études automatisées sur la visualisation d'information, et notamment sur la détection d'intrus [39, 38], donne des résultats corrélés avec ceux de participants humains. Une fois la tâche automatisée, il sera possible de faire passer un nombre bien plus important de tests et obtenir donc des résultats plus fiables.

6.7.2 Réorganisation des tâches pour une évaluation inter-groupe

Le principal problème rencontré dans notre évaluation est le manque important de variétés de scénarios dans nos questions afin de pouvoir couvrir tous les scénarios envisagés dans notre protocole par chaque participant. Comme mentionné plusieurs fois dans cette section, augmenter le nombre de questions pour chaque participant est difficilement concevable puisque l'évaluation est déjà longue et le risque de fatigue des participants pourrait fausser les résultats de l'évaluation.

Nous devons donc nous orienter vers un modèle d'évaluation inter-groupe, où la totalité des scénarios n'est pas explorée par chaque participant, mais leurs résultats se complémentent afin que l'on ait la totalité des scénarios envisagés par deux participants ou plus. Cette approche demande un nombre plus important de participants, et il faut donc peut-être revoir notre façon de proposer l'évaluation pour par exemple l'adapter aux plateformes de *crowdsourcing*, comme la plateforme Mechanical Turk d'Amazon [3] ou Prolific [4].

En ce qui concerne le protocole même, nous pouvons d'ores et déjà enlever la tâche T4 de l'évaluation puisqu'elle peut être faite par une machine pour obtenir des résultats plus significatifs. Ensuite, des structures se sont révélées être peu pertinentes sur certaines tâches, comme la structure «mélange» sur la tâche T2 (détection des distances), puisque les participants trouvaient instantanément la proximité des deux groupes mélangés, alors que nous n'avons pas assez exploré les cas potentiellement plus intéressants, comme les structures «proches» sur la tâche T2 ou T3, qui sont des scénarios plus difficiles à traiter dans les vrais cas d'usage. Enfin, les résultats obtenus sur la tâche T1 (comparaison de taille) nous surprennent sans que nous trouvions une réelle explication derrière les résultats. Nous pourrions essayer de creuser le problème en proposant plus de variété dans les questions, par exemple avec plus de granularité entre les tailles des groupes ou bien en ayant des jeux de données plus grands ou plus petits et faisant donc varier la taille des points à l'écran.

6.8 Conclusion

Dans ce chapitre, nous avons proposé un protocole d'évaluation visant à comparer des techniques de visualisation sur quatre points que nous avons considérés comme importants lors de la visualisation d'information à hautes dimensions. Ces quatre points couvrent la comparaison des effectifs de groupes de données, leurs distances les uns aux autres, la structure globale des données dans $\mathbb{R}^N$ et pour finir la détection d'intrus. Afin de couvrir un grand nombre de cas d'usages, nous avons conçu un outil permettant de générer un jeu de données respectant certaines caractéristiques, notamment la taille des groupes, leurs distances avec les autres groupes, ainsi que leur structure globale dans $\mathbb{R}^N$. Cette génération de données permet de connaître à l'avance la nature d'un jeu de données et donc d'évaluer les capacités des méthodes de projection et/ou de visualisation à représenter cette nature.

Nous avons conduit une évaluation entre la méthode de visualisation t-SNE et une méthode de visualisation compacte SSM. Nous avons fourni une méthode de coloration supplémentaire afin de faire ressortir les distances $\mathbb{R}^N$ sur la visualisation finale dans l'optique d'aider les utilisateurs à approfondir leur étude des jeux de données projetés. Sur une base de 74 questions, nous

3. www.mturk.com
4. www.prolific.co

avons conduit l'évaluation en suivant le protocole présenté à 15 participants. En considérant l'introduction à la problématique, les participants ont pris entre 40 et 60 minutes pour compléter l'évaluation, ce qui la rend particulièrement longue. Malgré l'exhaustivité des cas rencontrés, les résultats obtenus manquent de variance et peuvent donc difficilement généraliser les comportements observés. Néanmoins, nous avons pu confirmer la majorité des hypothèses posées avant l'évaluation. Ces confirmations permettent notamment de cibler les limites des techniques évaluées grâce aux résultats obtenus ainsi que les ressentis des participants.

Nous avons enfin discuté des limites de notre protocole d'évaluation. Si celui-ci couvre une grande partie de cas d'usages et plusieurs aspects de la visualisation de données, il demande un nombre conséquent de participants et doit possiblement se limiter à deux techniques de visualisations ou projections pour être efficace. Ici, nous avons évalué les capacités de t-SNE et SSM, mais en même temps la pertinence de la coloration par distances, ce qui a limité le nombre de questions associées à un cas d'usage en particulier, et donc finalement la variance des résultats relevés. Cette évaluation a donné lieu à de nouvelles hypothèses traitant de différences de comportements plus précis entre les méthodes de visualisation. Afin d'améliorer le protocole, nous devrions envisager de répartir les cas d'usages sur différents groupes de participants afin d'obtenir un ensemble de résultats complémentaires (*Between-participant experiment*), et en ciblant les scénarios les plus prompts à avoir de la variance dans les résultats plutôt que d'avoir des scénarios trop simples qui n'en ont pas.

Chapitre 7

Conclusion et travaux futurs

7.1 Récapitulatif des contributions

Les travaux présentés dans ce manuscrit ont permis d'explorer le domaine de l'explication de réseaux de neurones en s'appuyant sur l'analyse de masses de données hautement dimensionnelles au moyen de méthodes de visualisations dédiées :

7.1.1 Visualisation de la classification par des flux

Notre première contribution propose de faire un pont entre le réseau de neurones profonds pendant sa phase d'évaluation, et la visualisation d'information par des flux. Dans un premier temps, nous nous connectons en sortie de chaque couche du réseau afin de collecter toutes les informations calculées par le réseau pour obtenir un ensemble de cartes d'activation pour chaque prédiction du réseau. Ces cartes d'activation peuvent être ensuite soumises à des traitements supplémentaires basés sur des techniques d'explicabilités déjà existantes, comme *Layerwise Relevance Propagation*. Nous considérons ces données comme étant des points dans un espace $\mathbb{R}^N$ afin de calculer des distances entre toutes les cartes d'activation obtenues en sortie d'une même couche. Ce calcul nous permet de construire un partitionnement des données en une multitude de groupes composés uniquement de données considérées similaires par le réseau. Ces relations de similarités sont synthétisées dans un diagramme de Sankey afin de mettre en évidence les flux de données traités similairement par le réseau et leur évolution au fil des couches.

Cet outil est présenté en démonstration sur le site internet `https://pivert.labri.fr/flows/index.html` accompagné de plusieurs cas d'études, où des comportements de classification ont pu être observés. Des perspectives d'amélioration des réseaux ont aussi été proposés afin de réduire leur complexité en se basant sur l'évolution de la forme des flux. En plus de cela, cette contribution a montré l'efficacité des plateformes de calcul distribués afin d'étudier des scénarios de classification complexes comme l'utilisation de VGG16. Ces plateformes de calcul ont l'avantage d'être plus facilement mises à niveau que les machines de calcul conventionnelles pour traiter des cas plus complexes.

7.1.2 Visualisation compacte de la classification avec des courbes fractales

L'une des faiblesses de la méthode précédente est notamment sa lisibilité lors de l'étude de cas complexes impliquant plus de groupes à afficher. Les flux deviennent moins faciles à lire et la méthode ne permet pas de s'adapter aux architectures de réseaux non-linéaires.

Afin de remédier à ce manque de lisibilité, nous nous sommes orientés vers la visualisation orientée pixels, où chaque pixel représente un élément traité par le réseau. Nous abandonnons ainsi le calcul arbitraire de groupes de données similaires en amont et nous cherchons plutôt à positionner les pixels de manière représentative. Ainsi, les pixels des éléments du jeu de données traitées similairement sont placés côte à côte.

À l'aide d'un réarrangement calculée par la méthode VAT [10], nous parvenons à décider d'un ordre d'affichage des pixels sur le glyphe dont le placement suit la courbe fractale de Hilbert. Cette courbe a la particularité de respecter la proximité des éléments dans $\mathbb{N}^2$ par rapport à leur position dans $\mathbb{N}$ sur la courbe. Avec cette technique, nous parvenons à retrouver les comportements observés avec la première méthode de visualisation, et nous pouvons nous adapter aux architectures de réseaux non-linéaires. Nous proposons une démonstration de cet outil à l'adresse https://pivert.labri.fr/frac/index.html accompagnée des cas d'usages étudiés dans ce manuscrit.

Néanmoins, ce type de projection effectuée sur les grilles de pixels nécessite une projection vers un espace en une dimension afin de contraindre l'ordre des exemples dans la courbe fractale. Bien que la visualisation soit elle-même en deux dimensions, nous avons donc une perte d'informations importante par rapport aux données initialement dans $\mathbb{R}^N$. En plus de cela, l'utilisation de courbe fractale pour générer nos grilles de pixels est très limitée puisqu'elle nous impose un nombre n^4 d'éléments afin de ne pas impliquer de perte d'espace de dessin. Dans nos cas d'études où nous travaillions avec 10 000 éléments, près de 40% de l'espace de dessin était ainsi inutilisé, et cette perte peut monter jusqu'à 75% de l'espace de dessin dans le pire des cas.

7.1.3 Arrangement compacte pour une projection $\mathbb{R}^2 \to \mathbb{N}^2$

Les méthodes de visualisation compactes sont donc viables : le traitement des données intermédiaires d'un réseau de neurones profonds implique de travailler avec une très grande quantité d'informations. Ces informations varient en dimensions et chaque représentation d'une couche implique de synthétiser les milliers d'éléments traités par le réseau en une représentation graphique uniforme. La visualisation orientée pixels est suffisamment compacte pour nous permettre de nous adapter à ces très grandes quantités de données en occultant le moins d'informations possible. En revanche, l'utilisation de courbes fractales impose trop de contraintes afin d'être exploitables pour l'étude des données collectées lors des classifications des réseaux de neurones. Ces contraintes se posent notamment sur la nécessité de projeter les informations vers $\mathbb{N}$ dans un premier temps, impliquant la perte d'une partie de l'information originale, ainsi que sur la taille des jeux de données.

Nous nous concentrons donc sur le développement d'une méthode permettant d'arranger des données $\mathbb{R}^2$ en une grille de pixels. Nous appelons cette méthode VRGrid, un algorithme conçu pour arranger des données $\mathbb{R}^2$ en une grille compacte en déformant le moins possible les données initiales. Cette méthode permet d'utiliser au mieux l'espace de dessin et corrige donc l'une des limitations de la méthode précédente. L'application de VRGrid sur les données extraites des réseaux de neurones demande au préalable de les projeter vers $\mathbb{R}^2$. Il existe pour cela de nombreuses méthodes de projections «classiques», telles que t-SNE ou MDS, afin de représenter les données hautement dimensionnelles dans un espace en 2 dimensions. Ces méthodes sont souvent utilisées en visualisation d'information grâce à leur accessibilité d'analyse et capacité à s'adapter à de nombreuses natures de données. À l'aide de métriques d'évaluations empruntées aux domaines de recherche de réduction de dimensions et de dessin de graphes, nous avons pu évaluer VRGrid contre deux autres techniques de l'état de l'art, Self-Sorting Maps [128] et DGrid [49, 81]. Cette évaluation a montré que VRGrid déforme moins la projection initiale et conserve mieux les distances pair à pair des données dans $\mathbb{R}^2$.

Cependant, notre méthode s'avère être très coûteuse en temps de calcul et son utilisation devient difficile à justifier pour de très grands jeux de données. Dans les travaux autour de la méthode de relaxation utilisée dans VRGrid, des optimisations algorithmiques et matérielles sont proposées afin de considérablement accélérer son calcul. La poursuite de travaux sur VRGrid est donc envisageable afin de réduire ce temps de calcul, tout en conservant la bonne qualité des arrangements produits.

7.1.4 Comparaison de méthodes de visualisations

Nous avons pu mesurer l'efficacité de VRGrid sur des métriques d'évaluation de manière quantitative contre d'autres méthodes d'arrangements en grille, mais nous n'avons pas évalué l'efficacité de la méthode de visualisation compacte elle-même par rapport aux méthodes de visualisation par nuage de point. Cette efficacité doit être mesurée avec l'aide d'utilisateurs dans un cadre d'étude de données en hautes dimensions. Il est cependant difficile de comparer la visualisation compacte avec la visualisation par nuages de points classiques, puisque celle-ci diffère grandement en techniques de projections exploitables (par exemple les différences entre les méthodes uniformes comme MDS et les méthodes non-uniformes comme t-SNE).

Nous avons donc mis en place un protocole d'évaluation de méthodes de visualisation basée sur quatre tâches que nous avons considérées comme étant importantes lors de l'étude de données à grandes dimensions par méthode de visualisation. Ce protocole repose sur la génération de jeux de données suivant des caractéristiques et structures au préalable définies et sûres, avant d'être projetés par les méthodes testées. Le code permettant la génération de ces jeux de données est disponible en ligne à l'adresse `https://github.com/eikofee/grid-eval-data-generator`, et la plateforme d'évaluation à l'adresse `https://pivert.labri.fr/eval-{fr/en}`. Les utilisateurs participant à l'évaluation, par l'intermédiaire des tâches définies, doivent retrouver ces caractéristiques pour identifier la nature du jeu de données original.

Nous avons appliqué notre protocole sur une comparaisons des méthodes SSM et t-SNE avec 15 participants et avons pu observer des comportements qui corrélaient avec certaines de nos hypothèses de départ, notamment sur la capacité pour SSM de relever des informations sur les distances entre éléments que t-SNE n'a pas. Cependant, nous avons aussi remarqué une variance assez faible dans les résultats obtenus, ce qui encourage donc à revoir la façon d'appliquer le protocole d'évaluation. Nous proposons notamment de se focaliser sur certains cas d'usage en particulier et/ou en augmentant le nombre de participants, ainsi que l'automatisation des tâches dont le résultat nous semble évident telles que la détection d'intrus.

7.2 Travaux futurs

7.2.1 Extensions de la vue compacte

Les contributions présentées dans ce manuscrit ont toutes des pistes d'améliorations possibles, qui convergent vers la viabilité des méthodes compactes pour explorer des jeux de données dans $\mathbb{R}^N$, quelle que soit leur topologie. Les vues compactes générées grâce aux courbes fractales dans le Chapitre 4 nous ont permis de proposer une vue synthétique de la classification. VRGrid, présentée dans le Chapitre 5, nous permet d'améliorer encore l'utilisation de l'espace et de minimiser les déformations lors du passage de $\mathbb{R}^2$ à $\mathbb{N}^2$. Puis, dans le Chapitre 6, en exploitant davantage les capacités de SSM, nous avons proposé une coloration alternative de l'information afin d'explorer plus en détail la disposition des éléments dans leur espace d'origine. Cette coloration est un outil supplémentaire d'exploration de données hautement dimensionnelles, et notre évaluation utilisateur (bien qu'incomplète) a montré une meilleure efficacité d'utilisation sur la vue compacte par rapport à la vue par nuages de points classiques. On peut donc dire que la visualisation compacte permet une analyse plus profonde que l'étude des groupes d'éléments similaires que l'on avait sur nos deux premiers prototypes.

Il nous faudrait maintenant proposer un outil d'étude de réseau de neurones complet qui, partant du réseau entraîné et d'un jeu de données d'évaluation, peut «cartographier» les cartes d'activation calculées dans leurs espaces respectifs. Nous avons un exemple d'outil expérimental disponible à l'adresse `pivert.labri.fr/proto/index.html` affichant les cartes d'activation projetées avec t-SNE et arrangées par VRGrid$_F$. Cet outil permet d'afficher les 10 plus proches voisins d'un élément cliqué au niveau de chaque couche du réseau. Nous y pouvons donc observer, comme montré en exemple en Figure 7.1 la discrimination progressive des différentes classes du réseau

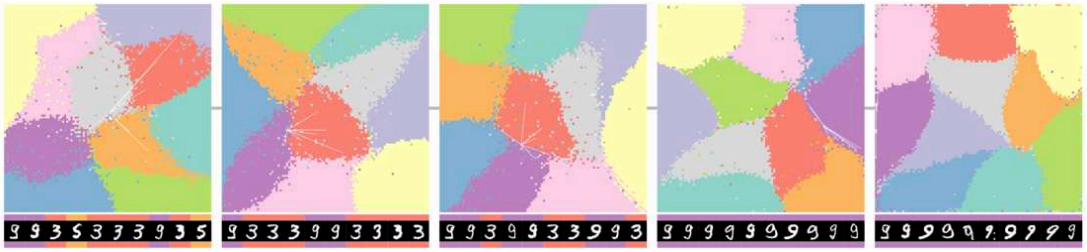


FIGURE 7.1 – Prototype d'outil de visualisation pour l'étude de la classification du jeu de données MNIST par un modèle LeNet. Ces données sont dans un premier temps projetées avec t-SNE puis arrangées avec VRGrid. Ces grilles sont ensuite présentées de manière successives pour représenter l'architecture du réseau étudié, comme les travaux des chapitres 3 et 4. En cliquant sur un élément dans la grille, les 9 plus proches éléments de l'élément choisi dans $\mathbb{R}^N$ sont affichés, afin de représenter la classification progressive mentionnée en chapitre 3. Sur cet exemple, nous remarquons que le 9 choisi est proche de 3 et de 5 , mais que celui-ci est rapproché d'autres 9 au fur et à mesure du traitement de la donnée par le modèle. On remarque cependant que le positionnement de ces éléments, représenté par les segments blancs sur les grilles de pixels, ne reflète pas vraiment cette proximité calculée.

(dans notre exemple, entre les 9 et les 3). Les proximités de ces cartes d'activation peuvent ensuite être comparées afin de retrouver les contributions individuelles de chaque couche et pour détecter une logique derrière les prédictions du réseau. Un autre avantage de l'utilisation de la visualisation compacte est qu'elle permet plus facilement l'affichage d'informations contextuelles étant donné le gain de place éventuel et l'absence d'occlusion d'information (superposition des éléments). Par exemple, la coloration des distances dans $\mathbb{R}^N$ proposée dans le Chapitre 6 peut assister l'utilisateur à comprendre la formation des cartes d'activations dans leur espace hautement dimensionnel.

Enfin, si nous avons montré que SSM est efficace pour représenter des données $\mathbb{R}^N$ de manière compacte, son utilisation dans le contexte des réseaux de neurones demande d'adapter l'algorithme pour avoir une certaine rétention du positionnement des éléments projetés. En effet, dans le chapitre 3, nous avons adapté l'ordre du placement des groupes calculés sur l'axe vertical de chaque couche de manière à limiter les croisements des flux et donc de faciliter la lecture de l'outil. Ce système de flux facilite la lecture des «mouvements» de données d'une couche à une autre, mais n'est pas applicable tel quel lorsque l'on travaille avec une projection vers $\mathbb{N}^2$, telles que les grilles de pixels compactes, comme dans les chapitres suivants. Il faut donc éviter que les éléments ne se déplacent trop d'un arrangement à un autre, car ceux-ci ne feront que perturber la lecture de la visualisation. Dans le Chapitre 5, nous avons montré que SSM ne prenait pas en compte le positionnement initial des points dans $\mathbb{R}^2$. Ce comportement implique que l'utilisation de la méthode sur les cartes d'activation de couches successives d'un réseau ne garantit pas une consistance dans le placement des groupes d'éléments similaires. Nous avons plusieurs pistes pour améliorer cet aspect pour notre cas d'application :

— Pivoter et/ou retourner les arrangements calculés pour limiter la déformation avec l'arrangement précédent (cette déformation peut être calculée de manière analogue avec les formules RPP et GPP du Chapitre 5). Cette piste a le désavantage de s'appliquer sur l'entièreté de l'arrangement et donc peut s'avérer inutile si certains groupes sont déjà «bien placés» mais pas d'autres.
— Définir une configuration initiale de la grille de SSM basé sur l'arrangement de la couche précédente, plutôt que de placer les points aléatoirement pour influencer l'arrangement final. Si les distances entre les points de deux groupes différents n'ont pas changé drastiquement d'une couche à une autre, nous pouvons nous attendre à ce que SSM ne les permute pas lors de son calcul.

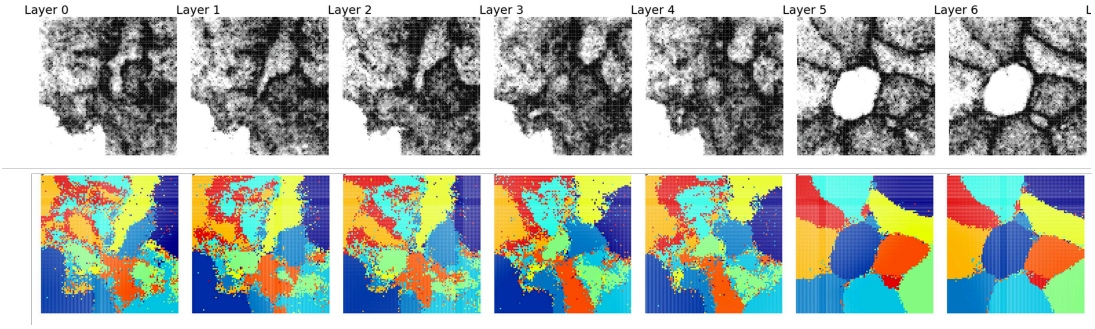


FIGURE 7.2 – Projections par SSM des cartes d'activation d'un modèle LeNet classifiant le jeu de données MNIST. La coloration par étiquette (ligne du bas) ainsi que la coloration par distances (ligne du haut, les couleurs claires indiquent les distances faibles et inversement pour les couleurs sombres) sont fournies. Ces colorations permettent d'étudier la formation des groupes de données similaires calculés par chaque couche du réseau, nous pouvons ainsi attribuer une «densité» à chaque groupe d'éléments. Un groupe ayant une forte densité (courtes distances) indique que ses éléments sont très similaires entre eux, et à l'inverse un groupe ayant une faible densité (grandes distances) indique que ses éléments sont moins similaires entre eux. Sur cet exemple, on aperçoit notamment les 1 manuscrits, représentés en bleu foncé, qui forment un groupe très dense dès l'entrée du réseau. Ils sont donc très bien reconnus par le réseau dès les premières couches. On y observe aussi la formation de groupes distincts au fil des couches du réseau, bien délimités par des frontières sombres sur la coloration par distances.

— Combiner la piste précédente avec l'utilisation d'une projection $\mathbb{R}^N \to \mathbb{R}^2$ comme t-SNE au préalable pour «uniformiser» les domaines de calculs de SSM. Comme t-SNE produit des nuages de points où les éléments de deux groupes différents sont espacés par la même distance dans $\mathbb{R}^2$, cela peut faciliter le comportement de SSM que l'on recherche en définissant une configuration initiale pour la grille. Nous avons appliqué cette piste sur notre scénario récurent du jeu de données MNIST classé par LeNet et obtenons la visualisation montrée en Figure 7.2. Nous y remarquons que les arrangements produits pour les premières couches convolutives (Layer 1 à Layer 4) du réseau sont similaires, mais avec un changement important une fois arrivé dans les couches denses (Layer 5 et 6).

Cette notion de temporalité, ainsi que les informations contextuelles avec les interactions utilisateur, sont des points majeurs à explorer lors du développement des outils.

7.2.2 Applications sur des scénarios variés

Les scénarios d'application testés pour nos méthodes de visualisations se sont principalement limités à des cas de classification d'image (MNIST et Fashion-MNIST) sur des réseaux de neurones convolutifs (LeNet et VGG16). Bien que nos méthodes aient été conçues afin de ne se pas limiter à un type de données en particulier, nous n'avons pas conduit d'essais sur des données autres que des images, ou bien sur des réseaux aux architectures complexes comme les réseaux récurrents (RNNs) ou les réseaux génératifs (GANs). Les premiers sont prévus pour traiter des données séquentielles, comme la reconnaissance de texte, tandis que les seconds fonctionnent à la fois sur la génération (Générateur) et la classification (Discriminateur) de données. Ces scénarios impliquent le traitement de données dans $\mathbb{R}^N$, mais il se pourrait qu'il faille adapter le processus de traitement des données (sur l'extraction des cartes d'activation pour les RNNs par exemple) ainsi que la visualisation elle-même (projection combinée ou séparée des cartes d'activation du Générateur et du Discriminateur pour les GANs par exemple) pour pouvoir étudier ces cas.

Enfin, un autre point non abordé dans ce manuscrit est l'application de ces méthodes de visualisation *pendant* l'entraînement du réseau étudié. La comparaison de la classification d'un jeu

de données entre deux époques différentes peut permettre d'observer les progrès du réseau sur sa capacité à discriminer les différentes classes de données au niveau de chaque couche.

L'exploration des données pendant les entraînements des réseaux nous orientent vers l'étude de nouveaux types d'information, tels que les les poids individuels des neurones d'un réseau qui évoluent d'une époque à une autre. La visualisation de ces poids est bien différente que celle des cartes d'activations, car les valeurs des activations individuelles des neurones ne correspondent pas à un même espace dans $\mathbb{R}^N$. Les caractéristiques de ces données sont en opposition avec la visualisation des projections des activations, où toutes les cartes d'activations appartiennent à un même espace dans $\mathbb{R}^N$.

Bibliographie

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. {TensorFlow} : a system for {Large-Scale} machine learning. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, pages 265–283, 2016.

[2] Amina Adadi and Mohammed Berrada. Peeking inside the black-box : a survey on explainable artificial intelligence (xai). *IEEE access*, 6 :52138–52160, 2018.

[3] Bilal Alsallakh, Allan Hanbury, Helwig Hauser, Silvia Miksch, and Andreas Rauber. Visual methods for analyzing probabilistic classification data. *IEEE transactions on visualization and computer graphics*, 20(12) :1703–1712, 2014.

[4] Robert Andrews, Joachim Diederich, and Alan B Tickle. Survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowledge-based systems*, 8(6) :373–389, 1995.

[5] Plamen P Angelov, Eduardo A Soares, Richard Jiang, Nicholas I Arnold, and Peter M Atkinson. Explainable artificial intelligence : an analytical review. *Wiley Interdisciplinary Reviews : Data Mining and Knowledge Discovery*, 11(5) :e1424, 2021.

[6] Alejandro Barredo Arrieta, Natalia Díaz-Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador García, Sergio Gil-López, Daniel Molina, Richard Benjamins, et al. Explainable artificial intelligence (xai) : Concepts, taxonomies, opportunities and challenges toward responsible ai. *Information fusion*, 58 :82–115, 2020.

[7] David Auber. Tulip—a huge graph visualization framework. *Graph drawing software*, pages 105–126, 2004.

[8] David Auber, Charles Huet, Antoine Lambert, Benjamin Renoust, Arnaud Sallaberry, and Agnes Saulnier. Gospermap : Using a gosper curve for laying out hierarchical data. *IEEE transactions on visualization and computer graphics*, 19(11) :1820–1832, 2013.

[9] Richard E Bellman. *Dynamic programming*. Princeton university press, 2010.

[10] James C Bezdek, Richard J Hathaway, and Jacalyn M Huband. Visual assessment of clustering tendency for rectangular dissimilarity matrices. *IEEE Transactions on fuzzy systems*, 15(5) :890–903, 2007.

[11] Adrien Bibal and Benoît Frénay. Interpretability of machine learning models and representations : an introduction. In *ESANN*, 2016.

[12] Alsallakh Bilal, Amin Jourabloo, Mao Ye, Xiaoming Liu, and Liu Ren. Do convolutional neural networks learn class hierarchy ? *IEEE transactions on visualization and computer graphics*, 24(1) :152–162, 2017.

[13] Rita Borgo, Luana Micallef, Benjamin Bach, Fintan McGee, and Bongshin Lee. Information visualization evaluation using crowdsourcing. In *Computer Graphics Forum*, volume 37, pages 573–595. Wiley Online Library, 2018.

[14] Dhruba Borthakur. The hadoop distributed file system : Architecture and design. *Hadoop Project Website*, 11(2007) :21, 2007.

[15] Kelei Cao, Mengchen Liu, Hang Su, Jing Wu, Jun Zhu, and Shixia Liu. Analyzing the noise robustness of deep neural networks. *IEEE transactions on visualization and computer graphics*, 27(7) :3289–3304, 2020.

[16] Diogo V Carvalho, Eduardo M Pereira, and Jaime S Cardoso. Machine learning interpretability : A survey on methods and metrics. *Electronics*, 8(8) :832, 2019.

[17] Changjian Chen, Jun Yuan, Yafeng Lu, Yang Liu, Hang Su, Songtao Yuan, and Shixia Liu. Oodanalyzer : Interactive analysis of out-of-distribution samples. *IEEE transactions on visualization and computer graphics*, 27(7) :3335–3349, 2020.

[18] Fati Chen, Laurent Piccinini, Pascal Poncelet, and Arnaud Sallaberry. Node overlap removal algorithms : An extended comparative study. *Journal of Graph Algorithms and Applications*, 2020.

[19] François Chollet et al. Keras. https://keras.io, 2015.

[20] Ian Covert, Scott M Lundberg, and Su-In Lee. Understanding global feature contributions with additive importance measures. *Advances in Neural Information Processing Systems*, 33 :17212–17223, 2020.

[21] Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A Bharath. Generative adversarial networks : An overview. *IEEE signal processing magazine*, 35(1) :53–65, 2018.

[22] Rene Cutura, Cristina Morariu, Zhanglin Cheng, Yunhai Wang, Daniel Weiskopf, and Michael Sedlmair. Hagrid : using hilbert and gosper curves to gridify scatterplots. *Journal of Visualization*, 25(6) :1291–1307, 2022.

[23] Nilaksh Das, Haekyu Park, Zijie J Wang, Fred Hohman, Robert Firstman, Emily Rogers, and Duen Horng Polo Chau. Bluff : Interactively deciphering adversarial attacks on deep neural networks. In *2020 IEEE Visualization Conference (VIS)*, pages 271–275. IEEE, 2020.

[24] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet : A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[25] Joseph F DeRose, Jiayao Wang, and Matthew Berger. Attention flows : Analyzing and comparing attention mechanisms in language models. *IEEE Transactions on Visualization and Computer Graphics*, 27(2) :1160–1170, 2020.

[26] Ann-Kathrin Dombrowski, Maximillian Alber, Christopher Anders, Marcel Ackermann, Klaus-Robert Müller, and Pan Kessel. Explanations can be manipulated and geometry is to blame. *Advances in neural information processing systems*, 32, 2019.

[27] Stuart Dreyfus. The computational solution of optimal control problems with time lag. *IEEE Transactions on Automatic Control*, 18(4) :383–385, 1973.

[28] Qiang Du, Maria Emelianenko, and Lili Ju. Convergence of the lloyd algorithm for computing centroidal voronoi tessellations. *SIAM journal on numerical analysis*, 44(1) :102–119, 2006.

[29] Qiang Du, Vance Faber, and Max Gunzburger. Centroidal voronoi tessellations : Applications and algorithms. *SIAM review*, 41(4) :637–676, 1999.

[30] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[31] Felipe SLG Duarte, Fabio Sikansi, Francisco M Fatore, Samuel G Fadel, and Fernando V Paulovich. Nmap : A novel neighborhood preservation space-filling algorithm. *IEEE transactions on visualization and computer graphics*, 20(12) :2063–2071, 2014.

[32] Geoffrey Ellis and Alan Dix. An explorative analysis of user evaluation studies in information visualisation. In *Proceedings of the 2006 AVI workshop on BEyond time and errors : novel evaluation methods for information visualization*, pages 1–7, 2006.

[33] Steven Fortune. A sweepline algorithm for voronoi diagrams. In *Proceedings of the second annual symposium on Computational geometry*, pages 313–322, 1986.

[34] Steven Fortune. Voronoi diagrams and delaunay triangulations. In *Handbook of discrete and computational geometry*, pages 705–721. Chapman and Hall/CRC, 2017.

[35] Ohad Fried, Stephen DiVerdi, Maciej Halber, Elena Sizikova, and Adam Finkelstein. Isomatch : Creating informative grid layouts. In *Computer graphics forum*, volume 34, pages 155–166. Wiley Online Library, 2015.

[36] Humberto S Garcia Caballero, Michel A Westenberg, Binyam Gebre, and Jarke J van Wijk. V-awake : A visual analytics approach for correcting sleep predictions from deep learning models. In *Computer Graphics Forum*, volume 38, pages 1–12. Wiley Online Library, 2019.

[37] Michael R Garey, David S Johnson, and Larry Stockmeyer. Some simplified np-complete problems. In *Proceedings of the sixth annual ACM symposium on Theory of computing*, pages 47–63, 1974.

[38] Loann Giovannangeli, Romain Bourqui, Romain Giot, and David Auber. Color and shape efficiency for outlier detection from automated to user evaluation. *Visual Informatics*, 6(2) :25–40, 2022.

[39] Loann Giovannangeli, Romain Giot, David Auber, Jenny Benois-Pineau, and Romain Bourqui. Analysis of deep neural networks correlations with human subjects on a perception task. In *2021 25th International Conference Information Visualisation (IV)*, pages 129–136. IEEE, 2021.

[40] Loann Giovannangeli, Frederic Lalanne, Romain Giot, and Romain Bourqui. Forbid : Fast overlap removal by stochastic gradient descent for graph drawing. In *Graph Drawing and Network Visualization : 30th International Symposium, GD 2022, Tokyo, Japan, September 13–16, 2022, Revised Selected Papers*, pages 61–76. Springer, 2023.

[41] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11) :139–144, 2020.

[42] Edouard Grave, Piotr Bojanowski, Prakhar Gupta, Armand Joulin, and Tomas Mikolov. Learning word vectors for 157 languages. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*, 2018.

[43] Donna L Gresh. Self-corrected perceptual colormaps. *IBM, Armonk, NY, Tech. Rep. RC24542 (W0804–104)*, 2008.

[44] Mark Harrower and Cynthia A Brewer. Colorbrewer. org : an online tool for selecting colour schemes for maps. *The Cartographic Journal*, 40(1) :27–37, 2003.

[45] Richard J Hathaway, James C Bezdek, and Jacalyn M Huband. Scalable visual assessment of cluster tendency for large data sets. *Pattern Recognition*, 39(7) :1315–1324, 2006.

[46] Timothy C Havens and James C Bezdek. An efficient formulation of the improved visual assessment of cluster tendency (ivat) algorithm. *IEEE Transactions on Knowledge and Data Engineering*, 24(5) :813–822, 2011.

[47] Subhashis Hazarika, Haoyu Li, Ko-Chih Wang, Han-Wei Shen, and Ching-Shan Chou. Nnva : Neural network assisted visual analysis of yeast cell polarization simulation. *IEEE Transactions on Visualization and Computer Graphics*, 26(1) :34–44, 2019.

[48] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[49] Gladys M Hilasaca and Fernando V Paulovich. A visual approach for user-guided feature fusion. In *Anais Estendidos da XXXII Conference on Graphics, Patterns and Images*, pages 133–139. SBC, 2019.

[50] David Hilbert and David Hilbert. Über die stetige abbildung einer linie auf ein flächenstück. *Dritter Band : Analysis· Grundlagen der Mathematik· Physik Verschiedenes : Nebst Einer Lebensgeschichte*, pages 1–2, 1935.

[51] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8) :1735–1780, 1997.

[52] Fred Hohman, Haekyu Park, Caleb Robinson, and Duen Horng Polo Chau. Summit : Scaling deep learning interpretability by visualizing activation and attribution summarizations. *IEEE transactions on visualization and computer graphics*, 26(1) :1096–1106, 2019.

[53] Danny Holten. Hierarchical edge bundles : Visualization of adjacency relations in hierarchical data. *IEEE Transactions on visualization and computer graphics*, 12(5) :741–748, 2006.

[54] Xinyi Huang, Suphanut Jamonnak, Ye Zhao, Tsung Heng Wu, and Wei Xu. A visual designer of layer-wise relevance propagation models. In *Computer Graphics Forum*, volume 40, pages 227–238. Wiley Online Library, 2021.

[55] Jacalyn M Huband, James C Bezdek, and Richard J Hathaway. bigvat : Visual assessment of cluster tendency for large data sets. *Pattern Recognition*, 38(11) :1875–1886, 2005.

[56] JM Huband, JC Bezdek, and RJ Hathaway. Revised visual assessment of (cluster) tendency (revat). In *IEEE Annual Meeting of the Fuzzy Information, 2004. Processing NAFIPS'04.*, volume 1, pages 101–104. IEEE, 2004.

[57] Théo Jaunet, Corentin Kervadec, Romain Vuillemot, Grigory Antipov, Moez Baccouche, and Christian Wolf. Visqa : X-raying vision and language reasoning in transformers. *IEEE Transactions on Visualization and Computer Graphics*, 28(1) :976–986, 2021.

[58] Shichao Jia, Peiwen Lin, Zeyu Li, Jiawan Zhang, and Shixia Liu. Visualizing surrogate decision trees of convolutional neural networks. *Journal of Visualization*, 23(1) :141–156, 2020.

[59] Zhihua Jin, Yong Wang, Qianwen Wang, Yao Ming, Tengfei Ma, and Huamin Qu. Gnnlens : A visual analytics approach for prediction error diagnosis of graph neural networks. *IEEE Transactions on Visualization and Computer Graphics*, 2022.

[60] Brian Johnson and Ben Shneiderman. Tree-maps : A space filling approach to the visualization of hierarchical information structures. Technical report, 1998.

[61] Michael Jünger and Petra Mutzel. 2-layer straightline crossing minimization : Performance of exact and heuristic algorithms. In *Graph Algorithms and Applications I*, pages 3–27. World Scientific, 2002.

[62] Minsuk Kahng, Pierre Y Andrews, Aditya Kalro, and Duen Horng Chau. A cti v is : Visual exploration of industry-scale deep neural network models. *IEEE transactions on visualization and computer graphics*, 24(1) :88–97, 2017.

[63] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *CVPR*, 2014.

[64] Daniel A Keim. Pixel-oriented visualization techniques for exploring very large data bases. *Journal of Computational and Graphical Statistics*, 5(1) :58–77, 1996.

[65] Daniel A Keim. Designing pixel-oriented visualization techniques : Theory and applications. *IEEE Transactions on visualization and computer graphics*, 6(1) :59–78, 2000.

[66] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6) :84–90, 2017.

[67] Joseph B Kruskal and Myron Wish. *Multidimensional scaling*, volume 11. Sage, 1978.

[68] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2) :83–97, 1955.

[69] Biagio La Rosa, Graziano Blasilli, R Bourqui, D Auber, Giuseppe Santucci, Roberto Capobianco, Enrico Bertini, Romain Giot, and Marco Angelini. State of the art of visual analytics for explainable deep learning. In *Computer Graphics Forum*. Wiley Online Library, 2023.

[70] Siwei Lai, Liheng Xu, Kang Liu, and Jun Zhao. Recurrent convolutional neural networks for text classification. In *Twenty-ninth AAAI conference on artificial intelligence*, 2015.

[71] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4) :541–551, 1989.

[72] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11) :2278–2324, 1998.

[73] Benjamin Letham, Cynthia Rudin, Tyler H McCormick, and David Madigan. Interpretable classifiers using rules and bayesian analysis : Building a better stroke prediction model. 2015.

[74] Ying Li, Haokui Zhang, and Qiang Shen. Spectral–spatial classification of hyperspectral imagery with 3d convolutional neural network. *Remote Sensing*, 9(1) :67, 2017.

[75] Mengchen Liu, Jiaxin Shi, Zhen Li, Chongxuan Li, Jun Zhu, and Shixia Liu. Towards better analysis of deep convolutional neural networks. *IEEE transactions on visualization and computer graphics*, 23(1) :91–100, 2016.

[76] Shusen Liu, Bhavya Kailkhura, Donald Loveland, and Yong Han. Generative counterfactual introspection for explainable deep learning. In *2019 IEEE global conference on signal and information processing (GlobalSIP)*, pages 1–5. IEEE, 2019.

[77] Yang Liu, Wenping Wang, Bruno Lévy, Feng Sun, Dong-Ming Yan, Lin Lu, and Chenglei Yang. On centroidal voronoi tessellation—energy smoothness and fast computation. *ACM Transactions on Graphics (ToG)*, 28(4) :1–17, 2009.

[78] Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2) :129–137, 1982.

[79] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. *Advances in neural information processing systems*, 30, 2017.

[80] Yuxin Ma, Arlen Fan, Jingrui He, Arun Reddy Nelakurthi, and Ross Maciejewski. A visual analytics framework for explaining and diagnosing transfer learning processes. *IEEE Transactions on Visualization and Computer Graphics*, 27(2) :1385–1395, 2020.

[81] Gladys Marleny Hilasaca Mamani. *A visual approach for user-guided feature fusion*. PhD thesis, Universidade de São Paulo.

[82] Leland McInnes, John Healy, and James Melville. Umap : Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv :1802.03426*, 2018.

[83] Leonel Merino, Mohammad Ghafari, Craig Anslow, and Oscar Nierstrasz. A systematic literature review of software visualization evaluation. *Journal of systems and software*, 144 :165–180, 2018.

[84] Yao Ming, Shaozu Cao, Ruixiang Zhang, Zhen Li, Yuanzhe Chen, Yangqiu Song, and Huamin Qu. Understanding hidden memories of recurrent neural networks. In *2017 IEEE Conference on Visual Analytics Science and Technology (VAST)*, pages 13–24. IEEE, 2017.

[85] Yao Ming, Huamin Qu, and Enrico Bertini. Rulematrix : Visualizing and understanding classifiers with rules. *IEEE transactions on visualization and computer graphics*, 25(1) :342–352, 2018.

[86] Yao Ming, Panpan Xu, Furui Cheng, Huamin Qu, and Liu Ren. Protosteer : Steering deep sequence model with prototypes. *IEEE transactions on visualization and computer graphics*, 26(1) :238–248, 2019.

[87] Sharada P Mohanty, David P Hughes, and Marcel Salathé. Using deep learning for image-based plant disease detection. *Frontiers in plant science*, 7 :1419, 2016.

[88] Grégoire Montavon, Wojciech Samek, and Klaus-Robert Müller. Methods for interpreting and understanding deep neural networks. *Digital Signal Processing*, 73 :1–15, 2018.

[89] Bongki Moon, Hosagrahar V Jagadish, Christos Faloutsos, and Joel H. Saltz. Analysis of the clustering properties of the hilbert space-filling curve. *IEEE Transactions on knowledge and data engineering*, 13(1) :124–141, 2001.

[90] Aaftab Munshi. The opencl specification. In *2009 IEEE Hot Chips 21 Symposium (HCS)*, pages 1–314. IEEE, 2009.

[91] Tamara Munzner. A nested model for visualization design and validation. *IEEE transactions on visualization and computer graphics*, 15(6) :921–928, 2009.

[92] Tamara Munzner. *Visualization analysis and design*. CRC press, 2014.

[93] Sugeerth Murugesan, Sana Malik, Fan Du, Eunyee Koh, and Tuan Manh Lai. Deepcompare : Visual and interactive comparison of deep learning model performance. *IEEE computer graphics and applications*, 39(5) :47–59, 2019.

[94] Tomoko Nemoto and David Beglar. Likert-scale questionnaires. In *JALT 2013 conference proceedings*, pages 1–8, 2014.

[95] Anh Nguyen, Alexey Dosovitskiy, Jason Yosinski, Thomas Brox, and Jeff Clune. Synthesizing the preferred inputs for neurons in neural networks via deep generator networks. *Advances in neural information processing systems*, 29, 2016.

[96] John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. Scalable parallel programming with cuda : Is cuda the parallel programming model that application developers have been waiting for ? *Queue*, 6(2) :40–53, 2008.

[97] Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. Feature visualization. *Distill*, 2(11) :e7, 2017.

[98] Chris Olah, Arvind Satyanarayan, Ian Johnson, Shan Carter, Ludwig Schubert, Katherine Ye, and Alexander Mordvintsev. The building blocks of interpretability. *Distill*, 3(3) :e10, 2018.

[99] Gabriel L Oliveira, Wolfram Burgard, and Thomas Brox. Efficient deep models for monocular road segmentation. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4885–4891. IEEE, 2016.

[100] Melissa E O'neill. Pcg : A family of simple fast space-efficient statistically good algorithms for random number generation. *ACM Transactions on Mathematical Software*, 2014.

[101] Cheonbok Park, Inyoup Na, Yongjang Jo, Sungbok Shin, Jaehyo Yoo, Bum Chul Kwon, Jian Zhao, Hyungjong Noh, Yeonsoo Lee, and Jaegul Choo. Sanvis : Visual analytics for understanding self-attention networks. In *2019 IEEE Visualization Conference (VIS)*, pages 146–150. IEEE, 2019.

[102] Cheonbok Park, Soyoung Yang, Inyoup Na, Sunghyo Chung, Sungbok Shin, Bum Chul Kwon, Deokgun Park, and Jaegul Choo. Vatun : Visual analytics for testing and understanding convolutional neural networks. In *Eurographics Conference on Visualization (EuroVis)-Short Papers. The Eurographics Association*, 2021.

[103] Haekyu Park, Nilaksh Das, Rahul Duggal, Austin P Wright, Omar Shaikh, Fred Hohman, and Duen Horng Polo Chau. Neurocartography : Scalable automatic visual summarization of concepts in deep neural networks. *IEEE Transactions on Visualization and Computer Graphics*, 28(1) :813–823, 2021.

[104] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch : An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[105] Karl Pearson. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin philosophical magazine and journal of science*, 2(11) :559–572, 1901.

[106] Jordi Petit. Experiments on the minimum linear arrangement problem. *Journal of Experimental Algorithmics (JEA)*, 8, 2003.

[107] Vitali Petsiuk, Abir Das, and Kate Saenko. Rise : Randomized input sampling for explanation of black-box models. *arXiv preprint arXiv :1806.07421*, 2018.

[108] Nicola Pezzotti, Thomas Höllt, B Lelieveldt, Elmar Eisemann, and Anna Vilanova. Hierarchical stochastic neighbor embedding. In *Computer Graphics Forum*, volume 35, pages 21–30. Wiley Online Library, 2016.

[109] Nicola Pezzotti, Thomas Höllt, Jan Van Gemert, Boudewijn PF Lelieveldt, Elmar Eisemann, and Anna Vilanova. Deepeyes : Progressive visual analytics for designing deep neural networks. *IEEE transactions on visualization and computer graphics*, 24(1) :98–108, 2017.

[110] Luc-Etienne Pommé, Romain Bourqui, Romain Giot, and David Auber. Relative confusion matrix : Efficient comparison of decision models. In *2022 26th International Conference Information Visualisation (IV)*, pages 98–103. IEEE, 2022.

[111] Michael Pühringer, Andreas Hinterreiter, and Marc Streit. Instanceflow : Visualizing the evolution of classifier confusion at the instance level. In *2020 IEEE Visualization Conference (VIS)*, pages 291–295. IEEE, 2020.

[112] Helen C Purchase. *Experimental human-computer interaction : a practical guide with visual examples*. Cambridge University Press, 2012.

[113] Paulo E Rauber, Samuel G Fadel, Alexandre X Falcao, and Alexandru C Telea. Visualizing the hidden activity of artificial neural networks. *IEEE transactions on visualization and computer graphics*, 23(1) :101–110, 2016.

[114] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. " why should i trust you ?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144, 2016.

[115] Bernice E Rogowitz and Lloyd A Treinish. Data visualization : the end of the rainbow. *IEEE spectrum*, 35(12) :52–59, 1998.

[116] Guodong Rong, Yang Liu, Wenping Wang, Xiaotian Yin, David Gu, and Xiaohu Guo. Gpu-assisted computation of centroidal voronoi tessellation. *IEEE transactions on visualization and computer graphics*, 17(3) :345–356, 2010.

[117] Frank Rosenblatt. The perceptron : a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6) :386, 1958.

[118] Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1(5) :206–215, 2019.

[119] Rabia Saleem, Bo Yuan, Fatih Kurugollu, Ashiq Anjum, and Lu Liu. Explaining deep neural networks : A survey on the global interpretation methods. *Neurocomputing*, 2022.

[120] Joris Sansen, Gaëlle Richer, Timothée Jourde, Frédéric Lalanne, David Auber, and Romain Bourqui. Visual exploration of large multidimensional data using parallel coordinates on big data infrastructure. In *Informatics*, volume 4, page 21. MDPI, 2017.

[121] Mario Schmidt. The sankey diagram in energy and material flow management : part ii : methodology and current applications. *Journal of industrial ecology*, 12(2) :173–185, 2008.

[122] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam : Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision*, pages 618–626, 2017.

[123] Qiaomu Shen, Yanhong Wu, Yuzhe Jiang, Wei Zeng, KH Alexis, Anna Vianova, and Huamin Qu. Visual interpretation of recurrent neural network on multi-dimensional time-series forecast. In *2020 IEEE Pacific Visualization Symposium (PacificVis)*, pages 61–70. IEEE, 2020.

[124] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv :1409.1556*, 2014.

[125] Daniel Smilkov, Shan Carter, D Sculley, Fernanda B Viégas, and Martin Wattenberg. Direct-manipulation visualization of deep networks. *arXiv preprint arXiv :1708.03788*, 2017.

[126] Thilo Spinner, Udo Schlegel, Hanna Schäfer, and Mennatallah El-Assady. explainer : A visual analytics framework for interactive and explainable machine learning. *IEEE transactions on visualization and computer graphics*, 26(1) :1064–1074, 2019.

[127] Gjorgji Strezoski and Marcel Worring. Plug-and-play interactive deep network visualization. *VADL : Visual Analytics for Deep Learning*, pages 0100–0106, 2017.

[128] Grant Strong and Minglun Gong. Self-sorting map : An efficient algorithm for presenting multimedia data in structured layouts. *IEEE Transactions on Multimedia*, 16(4) :1045–1058, 2014.

[129] Kozo Sugiyama, Shojiro Tagawa, and Mitsuhiko Toda. Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(2) :109–125, 1981.

[130] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *International conference on machine learning*, pages 3319–3328. PMLR, 2017.

[131] Richard S Sutton and Andrew G Barto. *Reinforcement learning : An introduction*. MIT press, 2018.

[132] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[133] Joshua B Tenenbaum, Vin De Silva, and John C Langford. A global geometric framework for nonlinear dimensionality reduction. *science*, 290(5500) :2319–2323, 2000.

[134] Edward R Tufte. The visual display of quantitative information. *The Journal for Healthcare Quality (JHQ)*, 7(3) :15, 1985.

[135] Jason Vallet, Guy Melançon, and Bruno Pinaud. Jasper : Just a new space-filling and pixel-oriented layout for large graph overview. In *Conference on Visualization and Data Analysis (VDA 2016)*, volume 2016, pages 1–10, 2016.

[136] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

[137] Laurens Van Der Maaten, Eric Postma, and Jaap Van den Herik. Dimensionality reduction : a comparative. *J Mach Learn Res*, 10(66-71) :13, 2009.

[138] Stijn Marinus Van Dongen. MCL - a cluster algorithm for graphs.

[139] Stijn Marinus Van Dongen. *Graph clustering by flow simulation*. PhD thesis, 2000.

[140] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. *Advances in neural information processing systems*, 29, 2016.

[141] Junpeng Wang, Liang Gou, Hao Yang, and Han-Wei Shen. Ganviz : A visual analytics approach to understand the adversarial game. *IEEE transactions on visualization and computer graphics*, 24(6) :1905–1917, 2018.

[142] Liang Wang, Uyen TV Nguyen, James C Bezdek, Christopher A Leckie, and Kotagiri Ramamohanarao. ivat and avat : enhanced visual analysis for cluster tendency assessment. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 16–27. Springer, 2010.

[143] Xingbo Wang, Jianben He, Zhihua Jin, Muqiao Yang, Yong Wang, and Huamin Qu. M2lens : Visualizing and explaining multimodal models for sentiment analysis. *IEEE Transactions on Visualization and Computer Graphics*, 28(1) :802–812, 2021.

[144] Zijie J Wang, Robert Turko, and Duen Horng Chau. Dodrio : Exploring transformer models with interactive visualization. *arXiv preprint arXiv :2103.14625*, 2021.

[145] Paul John Werbos. *The roots of backpropagation : from ordered derivatives to neural networks and political forecasting*, volume 1. John Wiley & Sons, 1994.

[146] Frank Wilcoxon. Individual comparisons by ranking methods. In *Breakthroughs in statistics*, pages 196–202. Springer, 1992.

[147] Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3) :37–52, 1987.

[148] Pak Chung Wong, Harlan Foote, Patrick Mackey, George Chin, Zhenyu Huang, and Jim Thomas. A space-filling visualization technique for multivariate small-world graphs. *IEEE Transactions on Visualization and Computer Graphics*, 18(5) :797–809, 2011.

[149] Linda Woodburn, Yalong Yang, and Kim Marriott. Interactive visualisation of hierarchical quantitative data : an evaluation. In *2019 IEEE Visualization Conference (VIS)*, pages 96–100. IEEE, 2019.

[150] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist : a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv :1708.07747*, 2017.

[151] Xiao Xiao. *Over-relaxation Lloyd method for computing centroidal Voronoi tessellations*. University of South Carolina, 2010.

[152] Pavan Yalamanchili, Umar Arshad, Zakiuddin Mohammed, Pradeep Garigipati, Peter Entschev, Brian Kloppenborg, James Malcolm, and John Melonakos. ArrayFire - A high performance software library for parallel computing with an easy-to-use API, 2015.

[153] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark : Cluster computing with working sets. In *2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 10)*, 2010.

[154] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

[155] Jiaqi Zheng and Tiow-Seng Tan. Computing centroidal voronoi tessellation using the gpu. In *Symposium on Interactive 3D Graphics and Games*, pages 1–9, 2020.