UNIVERSITÉ DU QUÉBEC

MÉMOIRE PRÉSENTÉ À
L'UNIVERSITÉ DU QUÉBEC À TROIS-RIVIÈRES

COMME EXIGENCE PARTIELLE
DE LA MAÎTRISE EN Maîtrise en Mathématiques et Informatique

PAR
Mamadou Sène

Titre : Utilisation du deep clustering pour la classification automatique d'images de déchets

Août 2025

Résumé

La gestion des déchets constitue un défi majeur pour les sociétés contemporaines, exigeant des solutions innovantes pour optimiser le tri et le recyclage. Ce mémoire de maîtrise se concentre sur l'utilisation du deep clustering pour la classification automatique d'images de déchets, en s'appuyant sur le jeu de données garbage classification disponible sur Kaggle [1]. Contrairement aux approches traditionnelles supervisées, telles que les réseaux de neurones convolutifs, le deep clustering offre une méthodologie non supervisée qui réduit significativement la dépendance à l'étiquetage manuel des données. Grâce à l'intégration de techniques modernes comme umap pour la réduction dimensionnelle, minibatch k-means et agglomerative clustering pour le regroupement, cette approche permet une classification efficace et innovante. Les résultats obtenus révèlent une performance compétitive, avec un silhouette score de 0.8697 pour minibatch k-means et de 0.8664 pour agglomerative clustering, ainsi qu'une précision globale de 95% en classification supervisée. Ces performances soulignent le potentiel du deep clustering pour répondre aux enjeux réels de la gestion des déchets, tout en proposant une alternative flexible et évolutive pour d'autres domaines nécessitant des solutions non supervisées.

Mamadou Sène

Nadia Ghazzali

1. `https://www.kaggle.com/datasets/sumn2u/garbage-classification-v2`

Abstract

Waste management is a critical challenge for contemporary societies, demanding innovative solutions to optimize sorting and recycling processes. This master's thesis focuses on the use of deep clustering for automatic waste image classification, leveraging the garbage classification dataset available on Kaggle [2]. Unlike traditional supervised approaches, such as convolutional neural networks, deep clustering offers an unsupervised methodology, significantly reducing the reliance on manual data labeling. By integrating modern techniques such as umap for dimensionality reduction and minibatch k-means and agglomerative clustering for grouping, this approach enables efficient and innovative classification. The results demonstrate competitive performance, with a silhouette score of 0.8697 for minibatch k-means and 0.8664 for agglomerative clustering, along with an overall classification accuracy of 95% in supervised evaluation. These findings highlight the potential of deep clustering to address real-world waste management challenges while offering a flexible and scalable alternative for other domains requiring unsupervised solutions.

2. https://www.kaggle.com/datasets/sumn2u/garbage-classification-v2

Table des matières

Liste des tableaux

Table des figures

Avant-propos

Ce mémoire est dédié à mes parents, dont l'amour et les valeurs continuent de m'inspirer chaque jour. Je tiens à remercier chaleureusement ma directrice de recherche, la professeure Nadia Ghazzali, pour son accompagnement exceptionnel et son soutien constant. Ma gratitude s'étend également aux membres du jury pour leurs contributions enrichissantes, ainsi qu'à mes collègues, camarades, amis et à ma famille pour leur appui moral et leur compréhension. Enfin, je remercie toutes les personnes, connues ou discrètes, dont le soutien moral, intellectuel ou spirituel a été essentiel à la réalisation de ce travail.

La gestion des déchets est un enjeu environnemental et économique de premier plan. Selon les estimations de la Banque mondiale, la production mondiale de déchets solides pourrait atteindre 3,4 milliards de tonnes par an d'ici 2050, contre 2,01 milliards de tonnes en 2016 [17]. Une gestion efficace des déchets nécessite des systèmes avancés de tri et de recyclage pour minimiser l'impact environnemental et maximiser la récupération des matériaux. La classification automatisée des déchets à partir d'images est une composante essentielle de ces systèmes.

Les réseaux de neurones convolutifs ont montré des résultats impressionnants dans la classification d'images, y compris pour la classification des déchets [21, 29]. Ces méthodes nécessitent toutefois des ensembles de données étiquetés de grande taille, ce qui peut être coûteux et chronophage à produire. En revanche, le deep clustering,

qui combine les approches de clustering non supervisé et l'apprentissage profond, offre une solution prometteuse en réduisant la dépendance à l'étiquetage manuel [4, 43].

L'objectif de ce mémoire est d'explorer et d'évaluer l'efficacité du deep clustering pour la classification d'images de déchets. En utilisant le dataset "garbage classification disponible sur kaggle, cette recherche vise à :

— Développer et entraîner un modèle de deep clustering pour classifier les images de déchets.
— Comparer les performances de ce modèle avec celles des méthodes supervisées existantes, telles que les réseaux de neurones convolutifs.
— Identifier les avantages et les limitations du deep clustering dans le contexte de la classification des déchets.

La méthodologie adoptée dans cette recherche comprend plusieurs étapes clés :

1. **Préparation des données** : Téléchargement et pré-traitement du dataset "garbage classification, incluant la normalisation et le redimensionnement des images. **Développement du modèle** : Conception et implémentation d'un modèle de deep clustering reposant sur l'extraction de caractéristiques via un réseau de neurones convolutifs pré-entraîné (ResNet-50), suivie de techniques de réduction de dimensionnalité et d'algorithmes de regroupement non supervisé.
2. **Entraînement et évaluation** : Entraînement du modèle sur les données d'images de déchets et évaluation de ses performances à l'aide de métriques pertinentes (précision, recall, F1-score).
3. **Comparaison avec les méthodes supervisées** : Comparer nos résultats obtenus avec ceux des modèles supervisés permet d'évaluer la pertinence et la robustesse de notre approche non supervisée, en particulier dans des contextes où l'annotation des données est coûteuse ou indisponible.

Ce mémoire est structuré comme suit :

— **Chapitre 1 : Introduction** – Présentation du contexte, des objectifs et de la méthodologie.

— **Chapitre 2 : Méthodes de classification classiques** – Analyse les méthodes de classification classiques .

— **Chapitre 3 : Méthodes neuronales** – Description détaillée des méthodes neuronales.

— **Chapitre 4 : Application sur notre jeu de données** – Présentation des résultats expérimentaux et discussion des performances comparées des modèles.

— **Chapitre 5 : Conclusion et Perspectives** – Synthèse des principaux résultats, implications pratiques et suggestions pour des recherches futures [20].

Chapitre 1

Méthodes de classification classiques

La classification est l'un des problèmes fondamentaux en apprentissage automatique et en reconnaissance de formes. Elle consiste à attribuer une étiquette ou une catégorie à une observation donnée. Les méthodes classiques de classification jouent un rôle essentiel dans divers domaines tels que la vision par ordinateur, la reconnaissance vocale, et l'analyse de texte. Ce chapitre présente un survol des méthodes de classification classiques, qui ont été largement étudiées et appliquées avant l'avènement des techniques d'apprentissage profond. Nous aborderons les concepts clés, les algorithmes couramment utilisés et leurs applications pratiques. Les méthodes de classification classiques incluent des approches supervisées et non supervisées. Les algorithmes supervisés, tels que les k-plus proches voisins (k-nn), les arbres de décision et les machines à vecteurs de support (svm), nécessitent des données étiquetées pour l'entraînement. En revanche, les méthodes non supervisées, comme le clustering hiérarchique et les k-means, ne nécessitent pas d'étiquettes et visent à découvrir des structures sous-jacentes dans les données [14, 22]. Les méthodes supervisées reposent sur des concepts de distance, de similarité et de frontière de décision pour séparer les

différentes classes [11]. Les méthodes non supervisées, quant à elles, se concentrent sur le regroupement des données similaires en clusters, facilitant ainsi l'exploration et l'interprétation des données [8]. Enfin, ce chapitre mettra en lumière les avantages et les inconvénients de chaque méthode, offrant ainsi une compréhension approfondie de leurs utilisations appropriées et de leurs limites. Les références bibliographiques à la fin du chapitre fournissent des lectures complémentaires pour approfondir les concepts discutés [2].

Définition d'une distance

La notion de distance est fondamentale en classification. Elle permet de mesurer la similitude ou la dissimilarité entre des objets. Une bonne mesure de distance est cruciale pour la performance des algorithmes de classification, en particulier pour ceux qui reposent sur des concepts de proximité, tels que le k-plus proches voisins (k-nn) et les algorithmes de clustering [11].

1. **Positivité (ou séparation)** :

$$\mathrm{d}(\mathrm{x},\mathrm{y}) \geq 0 \quad \text{et} \quad \mathrm{d}(\mathrm{x},\mathrm{y}) = 0 \iff \mathrm{x} = \mathrm{y}. \tag{1.1}$$

2. **Symétrie** :

$$\mathrm{d}(\mathrm{x},\mathrm{y}) = \mathrm{d}(\mathrm{y},\mathrm{x}). \tag{1.2}$$

3. **Inégalité triangulaire** :

$$\mathrm{d}(\mathrm{x},\mathrm{z}) \leq \mathrm{d}(\mathrm{x},\mathrm{y}) + \mathrm{d}(\mathrm{y},\mathrm{z}). \tag{1.3}$$

Parmi les distances couramment utilisées, on trouve :

— **Distance Euclidienne**

La distance euclidienne est l'une des mesures de distance les plus simples et les plus utilisées. Elle est définie comme la longueur du segment de ligne droite

reliant deux points dans un espace euclidien. Mathématiquement, la distance euclidienne entre deux points $\mathbf{x} = (\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_n})$ et $\mathbf{y} = (\mathbf{y_1}, \mathbf{y_2}, \ldots, \mathbf{y_n})$ est donnée par :

$$\mathrm{d}(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{\mathrm{i}=1}^{\mathrm{n}} (\mathrm{x_i} - \mathrm{y_i})^2}. \tag{1.4}$$

Cette formule exprime le théorème de Pythagore généralisé à des espaces de dimension supérieure. La distance euclidienne est largement utilisée dans les algorithmes de classification, notamment dans le k-nn et les méthodes de clustering telles que k-means [8].

— **Distance de Manhattan**

La distance de Manhattan, également connue sous le nom de distance L1 ou distance des blocs, est définie comme la somme des valeurs absolues des différences des coordonnées correspondantes.

Pour deux points $\mathbf{x} = (\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_n})$ et $\mathbf{y} = (\mathbf{y_1}, \mathbf{y_2}, \ldots, \mathbf{y_n})$, elle est donnée par :

$$\mathrm{d}_{\mathrm{Manhattan}}(\mathbf{x}, \mathbf{y}) = \sum_{\mathrm{i}=1}^{\mathrm{n}} |\mathrm{x_i} - \mathrm{y_i}|. \tag{1.5}$$

Cette mesure de distance est particulièrement utile dans les environnements où les déplacements sont limités à des chemins en forme de grille, comme les réseaux routiers des villes [2].

— **Distance de Minkowski**

La distance de Minkowski est une généralisation des distances euclidienne et de Manhattan. Pour un paramètre $\mathrm{p} \geq 1$, elle est définie par :

$$\mathrm{d}_{\mathrm{Minkowski}}(\mathbf{x}, \mathbf{y}) = \left(\sum_{\mathrm{i}=1}^{\mathrm{n}} |\mathrm{x_i} - \mathrm{y_i}|^{\mathrm{p}} \right)^{1/\mathrm{p}}. \tag{1.6}$$

Pour $\mathrm{p} = 2$, la distance de Minkowski correspond à la distance euclidienne, et pour $\mathrm{p} = 1$, elle correspond à la distance de Manhattan. Cette flexibilité

permet d'ajuster la mesure de distance en fonction des besoins spécifiques de l'application [11].

Plus précisément, la distance de Minkowski permet de moduler l'influence des grandes différences entre les composantes des vecteurs $\mathbf{x}$ et $\mathbf{y}$. Par exemple, lorsque p est grand, les différences les plus importantes dominent la mesure globale, ce qui peut être utile pour détecter des anomalies ou mettre en évidence des écarts significatifs. À l'inverse, pour des valeurs de p proches de 1, toutes les différences contribuent de façon plus uniforme, ce qui peut être mieux adapté à des contextes où l'on souhaite éviter que quelques valeurs extrêmes ne biaisent la mesure de similarité. Ainsi, la distance de Minkowski constitue une famille paramétrée de distances adaptée à divers contextes d'analyse ou d'apprentissage automatique.

— **Distance de Mahalanobis**

La distance de Mahalanobis prend en compte les corrélations entre les variables et est définie par :

$$\mathrm{d}_{\mathrm{Mahalanobis}}(\mathbf{x}, \mathbf{y}) = \sqrt{(\mathbf{x} - \mathbf{y})^\top \mathbf{S}^{-1} (\mathbf{x} - \mathbf{y})} \tag{1.7}$$

où ($\mathbf{S}$) est la matrice de covariance des données. Cette mesure est particulièrement utile lorsque les données présentent des corrélations, car elle normalise les distances en fonction de la dispersion des données [8].

— **Distance cosinus**

La distance cosinus mesure l'angle entre deux vecteurs dans un espace de caractéristiques, et est définie par :

$$\mathrm{d}_{\mathrm{cosinus}}(\mathbf{x}, \mathbf{y}) = 1 - \frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\| \|\mathbf{y}\|}. \tag{1.8}$$

Cette mesure est souvent utilisée dans l'analyse de texte et la reconnaissance de formes où l'orientation des vecteurs est plus importante que leur magnitude [2]. Ces différentes mesures de distance sont utilisées en fonction des

caractéristiques spécifiques des données et des besoins de l'application. Une bonne compréhension de ces mesures et de leurs propriétés permet de choisir l'algorithme de classification le plus approprié pour une tâche donnée.

Exemple :

— Soient $(\mathbf{x} = (1, 2))$ et $(\mathbf{y} = (4, 6))$.

— La distance euclidienne est :

$$\begin{aligned} \mathrm{d}(\mathbf{x}, \mathbf{y}) &= \sqrt{(1-4)^2 + (2-6)^2} \\ &= \sqrt{9+16} \\ &= 5. \end{aligned} \tag{1.9}$$

— **Saut minimum**

Le saut minimum, ou **lien simple**, entre deux clusters A et B est défini comme la distance minimale entre les points des deux clusters. Pour représenter cela sous forme matricielle, on considère les distances entre tous les points de A et B, puis on prend la valeur minimale [2] :

$$\mathrm{d}_{\min}(\mathrm{A}, \mathrm{B}) = \min \{\mathrm{d}(\mathrm{a_i}, \mathrm{b_j}) : \mathrm{a_i} \in \mathrm{A}, \mathrm{b_j} \in \mathrm{B}\}. \tag{1.10}$$

— **Sous forme matricielle**

Si D est une matrice contenant les distances entre les points des deux clusters A et B, alors :

$$\mathrm{d}_{\min}(\mathrm{A}, \mathrm{B}) = \min(\mathrm{D}). \tag{1.11}$$

— **Exemple**

Soient les clusters A et B définis comme suit :

$$\mathrm{A} = \{(1, 2),\ (3, 4)\}, \quad \mathrm{B} = \{(5, 6),\ (7, 8)\}. \tag{1.12}$$

Calculons la distance euclidienne entre chaque paire de points de A et B :

$$d((1,2),(5,6)) = \sqrt{(1-5)^2+(2-6)^2} = \sqrt{16+16} = \sqrt{32} \approx 5.66$$
$$d((1,2),(7,8)) = \sqrt{(1-7)^2+(2-8)^2} = \sqrt{36+36} = \sqrt{72} \approx 8.49$$
$$d((3,4),(5,6)) = \sqrt{(3-5)^2+(4-6)^2} = \sqrt{4+4} = \sqrt{8} \approx 2.83$$
$$d((3,4),(7,8)) = \sqrt{(3-7)^2+(4-8)^2} = \sqrt{16+16} = \sqrt{32} \approx 5.66.$$

La matrice des distances D est alors : $\mathrm{D} = \begin{pmatrix} 5.66 & 8.49 \\ 2.83 & 5.66 \end{pmatrix}$

Le saut minimum est donc : $\mathrm{d_{min}(A,B) = min(D) = 2.83}$

— **Saut maximum**

Le saut maximum, ou **lien complet**, entre deux clusters A et B est défini comme la distance maximale entre les points des deux clusters :

$$\mathrm{d_{max}(A,B) = max\left\{d(a_i, b_j) : a_i \in A, b_j \in B\right\}}$$

— **Sous forme matricielle**

$$\mathrm{d_{max}(A,B) = max(D)}$$

— **Exemple**

Reprenons les mêmes clusters A et B :

$$\mathrm{A} = \{(1,2),\ (3,4)\}, \quad \mathrm{B} = \{(5,6),\ (7,8)\}$$

La matrice des distances D reste la même : $\mathrm{D} = \begin{pmatrix} 5.66 & 8.49 \\ 2.83 & 5.66 \end{pmatrix}$

Le saut maximum est donc :$\mathrm{d_{max}(A,B) = max(D) = 8.49}$

1.1 Méthodes de classification hiérarchique

Les méthodes de classification hiérarchique (ou clustering hiérarchique) organisent les données en une hiérarchie de clusters. Ces méthodes peuvent être divisées en deux grandes catégories : ascendantes (agglomératives) et descendantes (divisives). La classification hiérarchique est largement utilisée dans de nombreux domaines, notamment la biologie, le traitement du langage naturel et la reconnaissance de formes [14].

1.1.1 Classification hiérarchique ascendante

Les méthodes de classification hiérarchique ascendante (ou agglomérative) commencent avec chaque point de données comme un cluster individuel et fusionnent les clusters les plus proches à chaque étape jusqu'à ce qu'il ne reste plus qu'un seul cluster englobant toutes les données. Ce processus est souvent représenté sous forme de *dendrogramme*, qui illustre les relations hiérarchiques entre les clusters.

Étapes de l'algorithme

1. Chaque point est initialement un cluster individuel.
2. À chaque itération, on identifie les deux clusters les plus proches en utilisant une mesure de distance appropriée (par exemple, distance euclidienne, distance de Manhattan).
3. Fusionner ces deux clusters en un nouveau cluster.
4. Répéter les étapes 2 et 3 jusqu'à ce qu'il ne reste qu'un seul cluster.

Remarque importante : influence de la mesure de distance

Le choix de la *mesure de distance* dans la classification hiérarchique ascendante

a une influence déterminante sur le résultat final. En effet, différentes distances impliquent différentes notions de similarité :

— La **distance euclidienne** est sensible aux variations d'échelle et aux valeurs extrêmes. Elle mesure la distance « à vol d'oiseau » entre les points.
— La **distance de Manhattan** est plus robuste aux variations fortes sur une seule dimension. Elle mesure la distance selon un chemin en grille.
— La **distance de Minkowski** permet de généraliser ces deux distances grâce à un paramètre p.

Ces différences influencent la manière dont les objets ou les groupes d'objets sont regroupés à chaque étape, ce qui peut aboutir à des *dendrogrammes très différents* pour un même jeu de données.

Illustration pédagogique : exemple simple avec la méthode de Ward
Pour ce faire, prenons un jeu de données simplifié constitué de cinq points dans un plan bidimensionnel :

$$\mathrm{A} = (1, 1), \quad \mathrm{B} = (1.5, 1.5), \quad \mathrm{C} = (5, 5), \quad \mathrm{D} = (3.5, 4.5), \quad \mathrm{E} = (4, 4). \tag{1.13}$$

La méthode de Ward consiste à fusionner les deux clusters dont la combinaison entraîne la plus petite augmentation de la variance intra-cluster. Elle tend à produire des clusters de taille comparable. La figure 1.1 illustre le résultat d'une classification hiérarchique ascendante appliquée sur cinq points simples en deux dimensions, en utilisant la méthode de Ward. Le dendrogramme permet de visualiser la manière dont les observations sont regroupées progressivement en clusters.

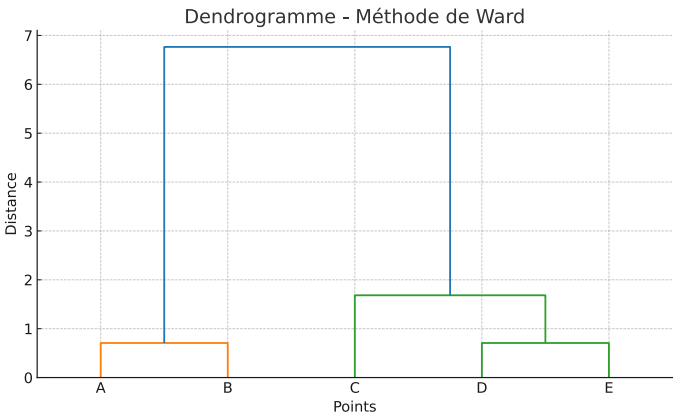

FIGURE 1.1 – Exemple de dendrogramme obtenu par la méthode de Ward.

En appliquant la classification hiérarchique ascendante avec la méthode de Ward :

- A et B seront fusionnés en premier car très proches (distance minimale).
- Ensuite, D et E seront regroupés.
- Puis C sera ajouté au cluster (D,E), formant un groupe plus grand.
- Enfin, les deux grands clusters (A,B) et (C,D,E) seront fusionnés pour former un cluster unique.

Le *dendrogramme* correspondant illustre clairement cette structure hiérarchique de fusion, et le seuil de coupe (hauteur) permet de choisir le nombre de clusters final.

Conclusion La classification hiérarchique ascendante est une méthode intuitive et visuelle pour explorer la structure des données. Elle est particulièrement utile pour des analyses exploratoires, mais nécessite une attention particulière au choix de la distance et de la méthode de linkage (Ward, moyenne, complète, etc.). L'exemple ci-dessus permet de visualiser concrètement son fonctionnement, renforçant ainsi sa compréhension.

1.1.2 Méthodes de fusion

Il existe plusieurs stratégies pour déterminer quels clusters doivent être fusionnés à chaque étape :

- **Lien simple** : La distance entre deux clusters est définie comme la distance minimale entre n'importe quel membre des deux clusters. Cette méthode peut entraîner des "chaînes" de clusters et est sensible au bruit [39].
- **Lien complet** : La distance entre deux clusters est définie comme la distance maximale entre n'importe quel membre des deux clusters. Cette méthode tend à produire des clusters compacts et sphériques [39].
- **Lien moyen** : La distance entre deux clusters est définie comme la moyenne des distances entre tous les membres des deux clusters [14].
- **Centroid linkage** : La distance entre deux clusters est définie comme la distance entre les centroïdes (moyennes) des deux clusters [14].
- **Méthode de Ward** : Cette méthode fusionne les deux clusters qui minimisent l'augmentation de la somme des carrés des distances (variance) au sein de chaque cluster [15].

Avantages :

- Ne nécessite pas de spécifier le nombre de clusters à l'avance.
- Produit une hiérarchie complète de clusters qui peut être utile pour explorer les données à différents niveaux de granularité.

Inconvénients :

- Les résultats peuvent être sensibles au choix de la mesure de distance et de la méthode de fusion.
- Les algorithmes de classification hiérarchique ascendante ont une complexité en temps quadratique, ce qui peut être prohibitif pour les grands ensembles de données [28].

Exemple : Classification hiérarchique ascendante avec dendrogramme

Un **dendrogramme** est un outil visuel pour représenter la structure hiérarchique des clusters. Chaque nœud du dendrogramme représente une fusion ou une division de clusters, et la hauteur des branches reflète la distance ou la dissimilarité entre les clusters fusionnés.

La figure 1.2 illustre un nuage de points représentant des données simulées selon deux dimensions : le revenu annuel et le score de dépenses. Ces données servent de base à l'application de la classification hiérarchique.

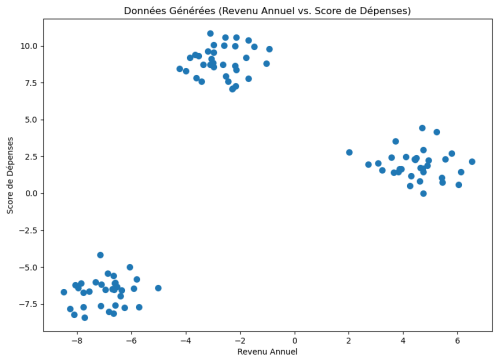

FIGURE 1.2 – Nuage de points : revenu annuel vs. score de dépenses.

La figure 1.3 montre le dendrogramme obtenu à partir de ces données. On observe comment les points sont regroupés progressivement en clusters, en fonction de leur similarité.

Explications et Résultats

1. **Génération des données :** Nous avons utilisé la fonction make_blobs de `sklearn.datasets` [18, 37, 40] pour créer un jeu de données synthétique avec 100 échantillons répartis en 3 clusters. Les données représentent des clients avec des caractéristiques "Revenu annuel" et "Score de dépenses".

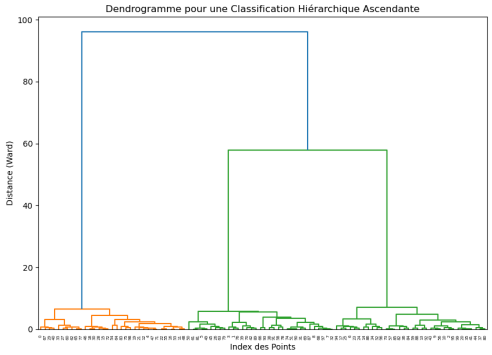

FIGURE 1.3 – Dendrogramme généré à partir des données simulées.

2. **Classification hiérarchique ascendante :** Nous avons appliqué l'algorithme de classification hiérarchique ascendante en utilisant la méthode de Ward. La méthode de Ward minimise la variance totale au sein des clusters.
3. **Visualisation du dendogramme :** Le dendogramme est une représentation visuelle de la hiérarchie des clusters. Chaque fusion de clusters est représentée par une ligne horizontale. L'axe vertical représente la distance ou dissimilarité entre les clusters fusionnés.

1.2 Méthodes de classification non hiérarchiques

1.2.1 Méthode des centres mobiles

La méthode des centres mobiles, plus communément connue sous le nom de **k-means**, est l'une des techniques de classification non hiérarchiques les plus populaires et les plus utilisées pour le partitionnement des données. L'algorithme k-means vise à partitionner n observations en k clusters où chaque observation appartient au cluster avec la moyenne (centre de cluster) la plus proche.

Étapes

Les étapes fondamentales de l'algorithme k-means sont les suivantes [2, 11, 22] :

1. **Initialisation des centres de clusters** : Choisir k centres de clusters initiaux. Cela peut se faire de manière aléatoire ou en utilisant des méthodes heuristiques comme k-means++ pour améliorer la qualité initiale des clusters [1].
2. **Assignation des points** : Assigner chaque point au centre de cluster le plus proche en utilisant une mesure de distance, généralement la distance euclidienne. Cela minimise la variance intra-cluster.
3. **Mise à jour des centres** : Mettre à jour les centres de clusters en calculant la moyenne des points assignés à chaque cluster. Le nouveau centre est le barycentre des points du cluster.
4. **Convergence** : Répéter les étapes d'assignation et de mise à jour jusqu'à ce que les centres de clusters ne changent plus significativement ou qu'un critère de convergence soit atteint (comme un nombre maximal d'itérations).

Avantages et Inconvénients

L'algorithme k-means est apprécié pour sa simplicité et sa rapidité, ce qui le rend adapté pour de grands ensembles de données. Cependant, il présente plusieurs inconvénients, notamment [12, 44] :

— **Sensibilité aux valeurs initiales** : Les résultats de k-means peuvent varier en fonction du choix initial des centres de clusters.

— **Sensibilité aux outliers** : Les outliers peuvent fortement influencer les centres des clusters.

— **Nombre de clusters k pré-défini** : L'algorithme nécessite de connaître à l'avance le nombre de clusters k, ce qui n'est pas toujours évident.

Exemple : Segmentation de clients avec `k-means`

Nous appliquons l'algorithme `k`-means à un jeu de données synthétiques représentant

des clients caractérisés par deux variables : le revenu annuel et le score de dépenses. Ces données ont été générées à l'aide de la fonction `make_blobs` de `sklearn.datasets`, qui permet de créer des groupes bien séparés. La figure 1.4 illustre la distribution initiale de ces clients dans l'espace bidimensionnel. Elle servira de base pour la segmentation à l'aide de l'algorithme k-means.

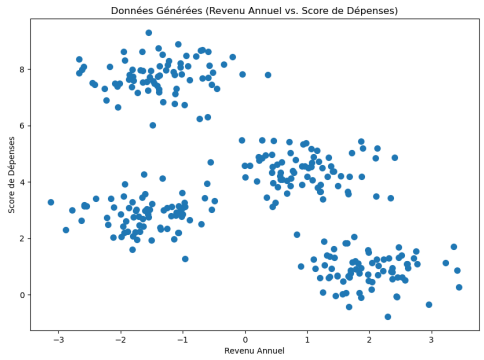

FIGURE 1.4 – Nuage de points : Revenu annuel vs. score de dépenses (clients simulés).

1.2.2 Méthode des nuées dynamiques

Les nuées dynamiques sont une méthode de classification non supervisée qui vise à partitionner un ensemble de données en un nombre fixe ou variable de classes tout en minimisant une fonction de coût. Contrairement au k-means, cette méthode repose sur une approche itérative où les centres de classes et l'affectation des individus sont mis à jour simultanément jusqu'à convergence [7]. Chaque classe est représentée par son prototype (ou centre), et l'algorithme cherche à optimiser une critère d'inertie intra-classe, permettant une meilleure compacité des groupes. Cette méthode, introduite par Diday et ses collaborateurs, est particulièrement utilisée en analyse de données exploratoire, dans des domaines comme la reconnaissance de formes, la biostatistique ou encore la segmentation de clientèle.

1.3 Réduction de dimensionnalité

1.3.1 UMAP : Uniform Manifold Approximation and Projection

`UMAP` est une méthode non linéaire de réduction de dimensionnalité fondée sur des concepts de topologie algébrique et de géométrie riemannienne. Elle permet de projeter des données à haute dimension dans un espace de plus faible dimension, tout en conservant au mieux la structure locale (voisinage) et, dans une certaine mesure, la structure globale des données [24].

L'algorithme commence par construire un graphe de voisinage approximatif dans l'espace d'origine, puis optimise une représentation dans l'espace projeté afin de préserver les relations de proximité entre les points. Comparée à d'autres techniques comme `t-SNE`, `UMAP` est généralement plus rapide, mieux adaptée à des ensembles de données volumineux (scalable), et présente de bonnes propriétés de généralisation. Elle est donc particulièrement utile pour la visualisation, le prétraitement avant un clustering, ou encore l'apprentissage semi-supervisé. La figure 1.5 présente un exemple typique de projection UMAP appliquée à un ensemble d'images de chiffres manuscrits. On y observe comment les différentes classes sont bien séparées dans l'espace réduit.

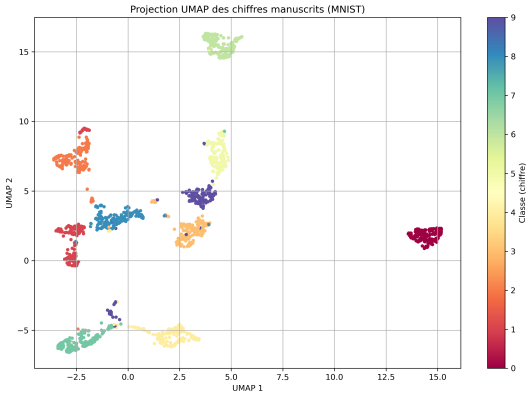

FIGURE 1.5 – Projection des chiffres manuscrits dans un espace 2D à l'aide de UMAP.

Dans cet exemple, nous appliquons l'algorithme **umap** à un jeu de données bien connu : **MNIST (digits)**, qui contient des images de chiffres manuscrits de 0 à 9, représentées sous forme de vecteurs de 64 dimensions (8x8 pixels).

Objectif

Réduire la dimensionnalité des données (64 dimensions) à 2 dimensions, tout en **préservant la structure locale** des données afin de visualiser les relations entre les différentes classes de chiffres.

Étapes clés

- **Chargement des données** :
  La fonction `load_digits()` fournit des images vectorisées des chiffres 0 à 9.
- **Standardisation** :
  Utilisation de `StandardScaler()` pour centrer et réduire les données. Cette étape est essentielle, car umap est sensible à l'échelle des variables.
- **Réduction de dimension avec umap** :
  Application de umap(n_neighbors = 15, min_dist = 0.1, metric =′ euclidean′) :
  - n_neighbors = 15 : équilibre entre structure locale et globale.
  - min_dist = 0.1 : contrôle la densité des clusters dans l'espace projeté.
  - metric =′ euclidean′ : mesure de distance utilisée entre les points.
- **Visualisation** : Chaque point est coloré selon sa classe (le chiffre correspondant). La figure générée montre comment les données sont projetées dans un espace bidimensionnel.

Résultats

- Les chiffres similaires forment des **groupes compacts**.
- Certains chiffres visuellement proches (comme 3 et 8, ou 1 et 7) peuvent être plus proches dans l'espace umap.

— UMAP permet ainsi de **visualiser les structures sous-jacentes** des données, même si elles sont initialement en haute dimension.

C'est un outil précieux pour :

— la **visualisation exploratoire** des données,

— la **préparation au clustering non supervisé**,

— ou encore la **détection d'anomalies**.

1.4 Matrice de confusion

Définition : La matrice de confusion est une représentation tabulaire des performances d'un modèle de classification, où [2] :

— Les lignes correspondent aux *classes réelles*.

— Les colonnes correspondent aux *classes prédites*.

Chaque cellule de la matrice indique le nombre d'instances ayant une classe réelle donnée et ayant été classées dans une classe particulière. Cela permet de calculer des métriques telles que :

— **Précision** : Fraction des instances correctement prédites parmi toutes les prédictions positives.

— **Rappel** : Fraction des instances correctement prédites parmi toutes les instances réelles positives.

— **F1-score** : Moyenne harmonique de la précision et du rappel.

La matrice de confusion est particulièrement utile pour évaluer les performances d'un modèle sur des ensembles de données déséquilibrés.

Chapitre 2

Méthodes neuronales

2.1 Un peu de biologie

Les réseaux de neurones artificiels tirent leur inspiration du fonctionnement du cerveau humain, et plus précisément des neurones biologiques. Pour mieux saisir cette analogie, il est essentiel de comprendre la structure et le rôle d'un neurone dans un système biologique.

Un neurone biologique est une cellule hautement spécialisée, composée de plusieurs parties distinctes : le soma (corps cellulaire), les dendrites (qui reçoivent les signaux) et l'axone (qui transmet les signaux à d'autres neurones). Ces composants interagissent pour permettre la propagation de l'influx nerveux.

La figure 2.1 illustre un schéma typique d'un neurone biologique et son intégration dans un réseau de neurones.

1. **Corps cellulaire (ou péricaryon)** : Il constitue le centre du neurone, où

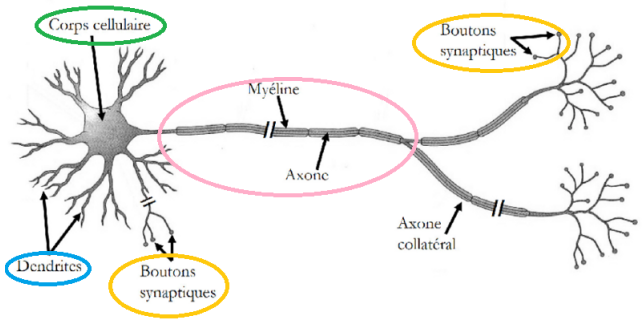

FIGURE 2.1 – Schéma d'un neurone biologique dans un réseau.

se trouve le noyau. Il est responsable de la synthèse des protéines et autres molécules essentielles au fonctionnement du neurone.

2. **Noyau** : Contenu dans le corps cellulaire, le noyau abrite le matériel génétique de la cellule et dirige ses activités.
3. **Dendrites** : Ces prolongements ramifiés émergent du corps cellulaire et servent de points d'entrée pour l'information. Les dendrites reçoivent des signaux provenant d'autres neurones et les transmettent au corps cellulaire.
4. **Axone** : Il s'agit d'un prolongement unique du corps cellulaire qui sert de chemin de sortie pour l'information. L'axone peut atteindre une longueur d'un mètre et est responsable de la transmission des signaux électriques vers d'autres neurones ou cellules.
5. **Gaine de myéline** : Enveloppant l'axone, cette gaine protectrice joue un rôle crucial dans l'augmentation de la vitesse de transmission des signaux électriques le long de l'axone.
6. **Terminaisons axonales (ou synapses)** : Ces extrémités de l'axone établissent des connexions avec les dendrites ou les corps cellulaires d'autres neurones, permettant la transmission de signaux entre les neurones.

Afin de mieux comprendre la logique qui sous-tend les réseaux de neurones artificiels, il est pertinent d'établir une analogie directe avec les neurones biologiques. Ces correspondances permettent de saisir comment les concepts biologiques ont été transposés dans un cadre mathématique et informatique.

Le tableau 2.1 met en parallèle les principaux composants d'un neurone biologique et leurs équivalents dans un neurone artificiel.

Neurone biologique	**Neurone artificiel**
Dendrites	Entrées (*input*)
Synapses	Poids (*weights*)
Axone	Sorties (*output*)
Activation	Fonction d'activation

TABLE 2.1 – Correspondance entre les composants d'un neurone biologique et ceux d'un neurone artificiel.

Cette analogie entre les neurones biologiques et les neurones artificiels est fondamentale pour comprendre comment les réseaux de neurones artificiels s'inspirent de la biologie pour traiter l'information. Dans les réseaux de neurones artificiels, les dendrites sont représentées par les entrées, qui reçoivent les données à traiter. Les synapses sont modélisées par des poids, qui ajustent l'importance des différentes entrées. L'axone correspond aux sorties du neurone artificiel, où les résultats du traitement sont envoyés. Enfin, la fonction d'activation dans un neurone artificiel remplit un rôle similaire à celui de l'activation biologique, déterminant si le neurone doit "s'activer" ou non en réponse à un ensemble donné d'entrées. Ce cadre biologique enrichit notre compréhension des réseaux de neurones artificiels, en soulignant l'importance des principes de transmission et de transformation de l'information dans les systèmes intelligents. Les **réseaux de neurones** représentent l'une des avancées les plus sophistiquées dans le domaine de l'apprentissage automatique. Ces modèles surpassent en complexité la plupart des autres techniques de Machine Learning, en raison de leur capacité à représenter des fonctions mathématiques à travers des millions de coefficients [19]. Cette capacité leur permet de modéliser des relations non linéaires complexes et d'apprendre des représentations de données à plusieurs niveaux d'abstraction.

2.2 Anatomie d'un réseau de neurones

Avant de plonger plus profondément dans les architectures et les techniques spécifiques, il est crucial de comprendre l'anatomie de base d'un réseau de neurones. Un réseau de neurones est composé de couches de neurones artificiels, chaque couche étant connectée aux suivantes par des poids ajustables. Ces couches comprennent généralement :

- **Couches d'entrée** : Où les données brutes (par exemple, pixels d'une image) sont introduites dans le réseau.
- **Couches cachées** : Plusieurs couches intermédiaires où se produisent les calculs complexes et où les représentations abstraites des données sont apprises.
- **Couches de sortie** : Où les résultats finaux sont produits, comme les classes d'objets identifiées dans une image.

Les neurones de chaque couche sont connectés aux neurones de la couche suivante par des poids, qui sont ajustés pendant l'entraînement pour minimiser l'erreur entre les prédictions du réseau et les résultats attendus.

2.2.1 Fonctionnement des réseaux de neurones

Le fonctionnement des réseaux de neurones peut être représenté à l'aide de deux types de schémas. Le premier illustre l'architecture générale d'un réseau multicouche : on y distingue la couche d'entrée, les couches cachées (ou couches intermédiaires) et la couche de sortie. Le second schéma détaille le fonctionnement interne d'un neurone artificiel : il reçoit des signaux en entrée pondérés par des poids, additionnés avec un biais, puis passés à une fonction d'activation qui produit la sortie.

La figure 2.2 montre l'architecture d'un réseau de neurones à une couche cachée. La figure 2.3 présente quant à elle le traitement effectué par un seul neurone.

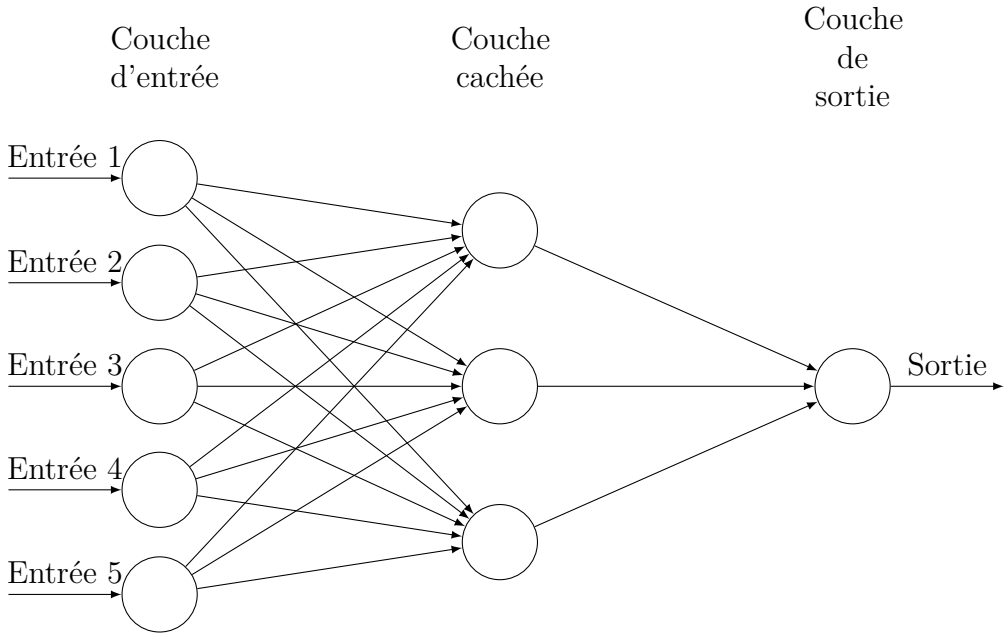

FIGURE 2.2 – Architecture d'un réseau de neurones à une couche cachée.

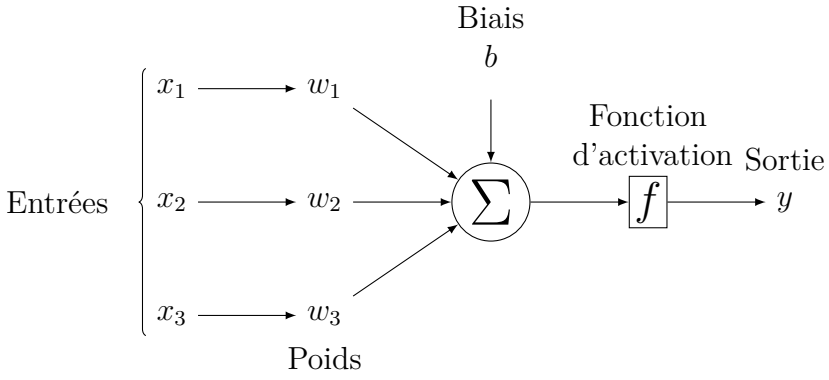

FIGURE 2.3 – Traitement d'un neurone artificiel : combinaison linéaire + fonction d'activation.

Exemple concret de fonctionnement d'un neurone artificiel

Considérons un neurone artificiel simple avec trois entrées : x_1, x_2 et x_3, associées respectivement aux poids w_1, w_2 et w_3. Le neurone est également muni d'un biais b

et utilise la fonction d'activation sigmoïde définie par :

$$f(z) = \frac{1}{1 + e^{-z}}. \tag{2.1}$$

Données de l'exemple

$$\begin{aligned} x_1 &= 0.5, \quad x_2 = -0.2, \quad x_3 = 0.1 && (2.2)\\ w_1 &= 0.4, \quad w_2 = 0.8, \quad w_3 = -0.5 && (2.3)\\ b &= 0.2. && (2.4) \end{aligned}$$

Étape : Calcul de la combinaison linéaire

$$\begin{aligned} z &= x_1 w_1 + x_2 w_2 + x_3 w_3 + b && (2.5)\\ &= (0.5)(0.4) + (-0.2)(0.8) + (0.1)(-0.5) + 0.2 && (2.6)\\ &= 0.2 - 0.16 - 0.05 + 0.2 && (2.7)\\ &= 0.19. && (2.8) \end{aligned}$$

Étape 2 : Passage par la fonction d'activation

$$\begin{aligned} f(z) &= \frac{1}{1 + e^{-0.19}} && (2.9)\\ &\approx \frac{1}{1 + 0.827} && (2.10)\\ &\approx 0.5474. && (2.11) \end{aligned}$$

Interprétation La sortie du neurone est donc d'environ 0.5474, ce qui signifie que le neurone « active » légèrement en réponse aux entrées fournies. Cela pourrait représenter, par exemple, la probabilité que l'entrée appartienne à une certaine classe dans

un problème de classification binaire.

2.3 Fonctionnement et Architecture

Dans un réseau de neurones, nous observons généralement une couche d'entrée à gauche, une couche de sortie à droite, et plusieurs couches intermédiaires dites **cachées**. Ces couches cachées contiennent des **neurones**, représentés par de petits cercles, qui exécutent des **fonctions d'activation**. Dans les réseaux de neurones de base, la **fonction logistique** est couramment utilisée comme fonction d'activation. Explorons en détail le fonctionnement d'un **neurone** [25].

2.3.1 Processus d'apprentissage

Le fonctionnement interne d'un neurone peut être divisé en deux phases principales :

1. **Calcul de la somme pondérée** : Les neurones reçoivent plusieurs **entrées** (ou **features**) notées $\mathbf{x_j}$, chacune étant associée à un poids $\mathbf{w_{ij}}$ que le modèle doit apprendre. Le neurone calcule une somme pondérée de ces entrées, additionnée d'un terme de biais $\theta_\mathbf{i}$. Cette somme pondérée, notée $\mathbf{n_i}$, est donnée par :

$$\mathbf{n_i} = \sum_{j=1}^{k}(w_{ji}x_j + \theta_i). \tag{2.12}$$

2. **Application de la fonction d'activation** : La valeur $\mathbf{n_i}$ est ensuite passée à travers une fonction d'activation, qui est souvent non-linéaire. Pour un réseau de neurones basique, la fonction d'activation utilisée est souvent la fonction

sigmoïde (logistique). La formule de cette fonction est :

$$\sigma(x) \quad = \quad \frac{1}{1+e^{-x}}. \tag{2.13}$$

La fonction sigmoïde transforme $\mathbf{n_i}$ en une valeur comprise entre 0 et 1, suivant une courbe en forme de S. Cependant, cette fonction présente l'inconvénient de comprimer les grandes valeurs positives et négatives, ce qui peut ralentir l'apprentissage et poser des problèmes de saturation.

2.3.2 Le Perceptron

Le **perceptron** est l'un des modèles les plus simples de réseau de neurones artificiels. Il est conceptuellement proche de la régression logistique, dans laquelle les entrées sont pondérées, un biais est ajouté, puis le tout est passé à une fonction d'activation (souvent de type seuil ou sigmoïde) pour produire une sortie binaire. La figure 2.4 présente une illustration schématique du perceptron. On y observe le processus de traitement d'un vecteur d'entrée à travers des poids et une fonction d'activation afin d'obtenir une prédiction.

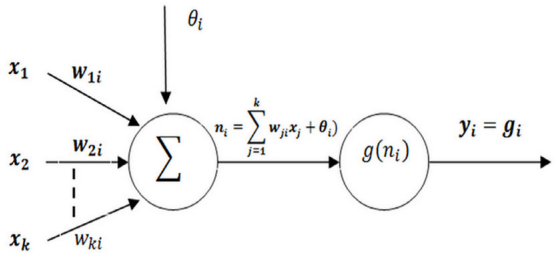

FIGURE 2.4 – Schéma du perceptron : un neurone artificiel simple avec entrées pondérées, biais, et fonction d'activation.

2.3.3 Fonctions d'activation alternatives

En pratique, d'autres fonctions d'activation sont souvent préférées à la fonction sigmoïde, principalement pour améliorer l'efficacité du calcul des gradients et accélérer les cycles d'apprentissage. Voici quelques exemples :

— **Fonction tangente hyperbolique (tanh)** : La fonction tangente hyperbolique est similaire à la fonction sigmoïde mais elle est centrée autour de zéro, ce qui aide à rendre le processus d'apprentissage plus rapide et efficace. Elle est définie par :

$$\tanh(z) \quad = \quad \frac{2}{1+e^{-2z}} - 1. \tag{2.14}$$

Cette fonction produit des sorties entre -1 et 1, ce qui facilite la gestion des gradients lors de l'apprentissage, réduisant ainsi le risque de saturation.

— **ReLU (Rectified Linear Unit)** : La fonction **ReLU** est largement utilisée en raison de sa simplicité et de son efficacité. Contrairement aux fonctions sigmoïde et tanh, qui compriment les grandes valeurs dans un intervalle limité, la fonction ReLU ne présente pas de saturation pour les grandes valeurs positives. Elle est définie par :

$$\mathrm{ReLU}(x) \quad = \quad \max(0, x). \tag{2.15}$$

Cela signifie que toutes les valeurs négatives sont mises à zéro, tandis que les valeurs positives sont conservées, ce qui permet un apprentissage plus efficace et une convergence plus rapide.

Ces fonctions d'activation sont essentielles pour l'apprentissage des réseaux de neurones, car elles introduisent des non-linéarités qui permettent au réseau de modéliser des relations complexes entre les données d'entrée et les sorties.

Remarques : L'une des principales raisons pour lesquelles des fonctions d'activation comme ReLU sont préférées est qu'elles réduisent le risque de saturation, où les neurones cessent d'apprendre efficacement parce que les gradients deviennent trop petits. Ce problème, connu sous le nom de **"vanishing gradient problem"**, est particulièrement problématique dans les réseaux profonds. Les fonctions comme ReLU et ses variantes, telles que Leaky ReLU ou Parametric ReLU, offrent des solutions pour surmonter ce défi, permettant ainsi de construire des réseaux de neurones plus profonds et plus performants.

2.4 Réseaux de neurones multicouches

La construction d'un réseau de neurones implique l'intégration de multiples **perceptrons** interconnectés selon une architecture organisée en **couches** (ou *layers*). Les neurones sont répartis en colonnes distinctes : une couche d'entrée, une ou plusieurs couches dites **cachées**, et une couche de sortie.

— **Organisation en couches** : Au sein d'une même couche, les neurones ne sont pas interconnectés. En revanche, chaque neurone d'une couche est entièrement connecté aux neurones de la couche suivante. Cette organisation permet de modéliser des relations complexes entre les entrées et les sorties.

— **Connexions inter-couches** : Toutes les **sorties** des neurones d'une couche sont transmises aux **entrées** de tous les neurones de la couche suivante. Cette connexion dense favorise le transfert d'information et l'apprentissage de représentations hiérarchiques.

La figure 2.5 illustre l'architecture d'un réseau de neurones comprenant trois couches : une couche d'entrée, une couche cachée et une couche de sortie. Cette structure constitue la base des réseaux dits **profonds**, où la multiplication des couches

permet au modèle d'apprendre des représentations de plus en plus abstraites.

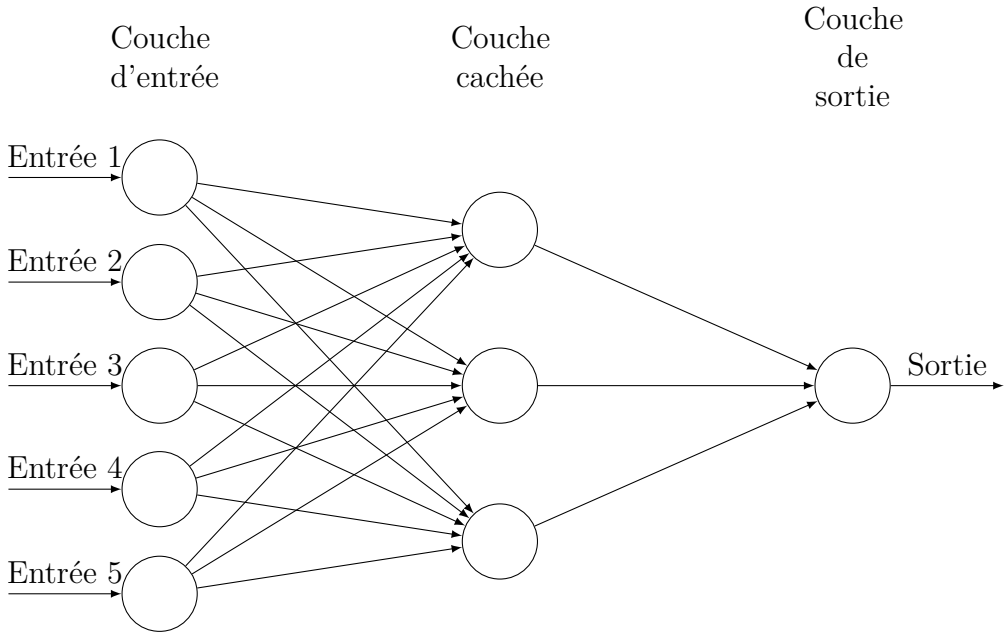

FIGURE 2.5 – Architecture d'un réseau de neurones multicouche avec une seule couche cachée.

La figure 2.6 détaille le fonctionnement d'un neurone artificiel typique : les entrées pondérées et le biais sont combinés dans une somme, puis passés à une fonction d'activation qui détermine la sortie du neurone. Ce principe, simple en apparence, constitue le fondement des réseaux profonds.

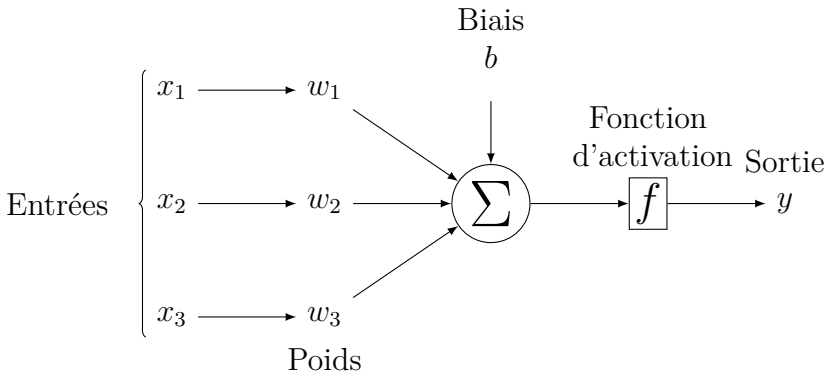

FIGURE 2.6 – Traitement d'un neurone artificiel : combinaison linéaire suivie d'une fonction d'activation.

2.5 Entraînement d'un réseau de neurones

Pour résoudre un problème **d'apprentissage supervisé**, il est essentiel de suivre un plan structuré et bien défini. Cette approche méthodique permet de s'assurer que toutes les phases de l'apprentissage sont respectées et optimisées. Les étapes fondamentales de cet entraînement incluent :

— **Jeu de données** : Le processus d'apprentissage commence par la constitution d'un ensemble de données, généralement représenté sous forme de tableau **(X, y)**. Les caractéristiques (**features**) telles que $\mathbf{x_1}, \mathbf{x_2}, \mathbf{x_3}, \ldots$ sont introduites dans le réseau à travers la première couche (input layer). Ce jeu de données sert de fondation pour l'entraînement, permettant au modèle d'apprendre les relations entre les entrées et les sorties attendues.

2.5.1 Fonction coût et calcul du gradient

Une fois le modèle défini et les données propagées vers l'avant à travers le réseau (*forward propagation*), il est nécessaire de mesurer à quel point les prédictions du réseau diffèrent des sorties attendues. C'est le rôle de la **fonction coût** (ou *fonction de perte*, en anglais *loss function*).

Définition La fonction coût $J(\theta)$ évalue la performance du réseau sur le jeu d'entraînement, en calculant l'écart entre les prédictions $\hat{y}$ et les vraies valeurs y. Son expression dépend du type de problème :

— Pour une **classification binaire**, on utilise souvent la *cross-entropy loss* :

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log(\hat{y}^{(i)}) + (1-y^{(i)})\log(1-\hat{y}^{(i)})\right]. \tag{2.16}$$

— Pour une **régression**, la *erreur quadratique moyenne* (mse) est couramment employée :

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \left(\hat{y}^{(i)} - y^{(i)} \right)^2 . \tag{2.17}$$

Rétropropagation des erreurs : Une fois le coût évalué, il est indispensable de déterminer dans quelle direction ajuster les paramètres du réseau (poids et biais) afin de diminuer cette erreur. Pour cela, on calcule le **gradient** de la fonction coût par rapport à chaque paramètre du réseau. La méthode de **rétropropagation** (ou *backpropagation*) permet de calculer efficacement ces gradients, couche par couche, en appliquant la règle de la chaîne (*chain rule*) du calcul différentiel. Ce processus descend progressivement depuis la couche de sortie jusqu'à la couche d'entrée, en ajustant les paramètres selon l'influence qu'ils ont eue sur l'erreur globale [35].

Rôle des gradients : Les gradients obtenus donnent la direction dans laquelle la fonction coût diminue le plus rapidement. Ils sont ensuite utilisés par un algorithme d'optimisation (souvent la descente de gradient, voir section suivante) pour ajuster les paramètres du réseau.

2.5.2 Algorithme de descente de gradient

Une fois la fonction coût calculée et les gradients obtenus via la rétropropagation, il faut maintenant ajuster les paramètres du réseau pour minimiser cette fonction. L'algorithme le plus utilisé à cet effet est la **descente de gradient** (*gradient descent*).

Principe de l'algorithme : L'idée principale de la descente de gradient est de mettre à jour les paramètres du réseau dans la direction opposée au gradient de la

fonction coût, car c'est dans cette direction que la fonction décroît le plus rapidement.

Si θ représente un vecteur de paramètres (poids, biais, etc.), la mise à jour se fait selon la formule suivante :

$$\theta \leftarrow \theta - \eta \cdot \nabla_\theta J(\theta) \tag{2.18}$$

où :

— η est le **taux d'apprentissage** (*learning rate*),

— $\nabla_\theta J(\theta)$ est le **gradient** de la fonction coût par rapport aux paramètres θ.

Lien avec la rétropropagation Le calcul du gradient $\nabla_\theta J(\theta)$ est obtenu par la rétropropagation. Ce mécanisme repose sur la règle de la chaîne en dérivation, appliquée de la couche de sortie jusqu'à la couche d'entrée. Les erreurs sont donc propagées *en arrière* dans le réseau pour déterminer la contribution de chaque neurone à l'erreur totale.

Pour chaque couche l, l'erreur est notée $\delta^{(l)}$, et les gradients des poids $W^{(l)}$ et biais $b^{(l)}$ sont calculés comme suit :

$$\frac{\partial J}{\partial W^{(l)}} = \delta^{(l+1)} \cdot (a^{(l)})^T, \qquad \frac{\partial J}{\partial b^{(l)}} = \delta^{(l+1)} \tag{2.19}$$

où $a^{(l)}$ est l'activation de la couche l, et $\delta^{(l+1)}$ est l'erreur calculée pour la couche suivante.

Types de descente de gradient Il existe plusieurs variantes de la descente de gradient :

- **Batch Gradient Descent** : utilise l'ensemble du jeu de données pour chaque mise à jour.
- **Stochastic Gradient Descent (SGD)** : met à jour les poids à chaque échantillon, introduisant du bruit qui peut aider à sortir des minima locaux.
- **Mini-batch Gradient Descent** : compromis entre batch et SGD, il utilise un sous-ensemble (mini-lot) des données.

Importance du taux d'apprentissage

Dans l'équation suivante de mise à jour des paramètres par descente de gradient :

$$\theta \leftarrow \theta - \eta \cdot \nabla_\theta J(\theta) \tag{2.20}$$

le terme η, appelé **taux d'apprentissage** (*learning rate*), joue un rôle fondamental dans le processus d'optimisation.

Fonction du taux d'apprentissage : Le taux d'apprentissage contrôle la **vitesse** à laquelle le modèle apprend en ajustant ses paramètres à chaque itération. Il détermine l'**amplitude du pas** effectué dans la direction opposée au gradient de la fonction coût.

- Si η est **trop petit**, la convergence vers le minimum global peut être très lente, et l'algorithme risque de stagner dans des plateaux ou des minima locaux.
- Si η est **trop grand**, les mises à jour peuvent être trop brusques, faisant osciller la fonction coût ou même provoquer une **divergence**.

Analogie intuitive On peut illustrer le rôle du taux d'apprentissage par une bille qui descend une pente :

— Un petit η fait que la bille descend lentement, pas à pas.

— Un grand η fait que la bille bondit d'un côté à l'autre sans se stabiliser au fond.

Choix judicieux du taux : Le choix du taux d'apprentissage dépend de plusieurs facteurs :

— la nature du problème (fonction convexe ou non),

— la structure du modèle,

— l'échelle des gradients calculés.

Taux d'apprentissage adaptatif : Pour surmonter les limitations d'un taux d'apprentissage fixe, plusieurs algorithmes d'optimisation adaptatifs ont été développés, tels que :

— **Adam (Adaptive Moment Estimation)**,

— **RMSprop (Root Mean Square Propagation)**,

— **Adagrad**.

Ces méthodes ajustent dynamiquement η à chaque itération et pour chaque paramètre, en se basant sur les historiques de gradients. Elles offrent ainsi une meilleure stabilité et permettent une convergence plus rapide, notamment dans les modèles profonds ou très sensibles.

Conclusion : Un bon choix du taux d'apprentissage est essentiel pour garantir la convergence rapide et fiable d'un réseau de neurones. Il constitue un des hyperparamètres les plus critiques à régler en pratique.

Convergence et erreurs de propagation Un mauvais paramétrage ou une mauvaise initialisation des poids peut conduire à des erreurs de propagation :

— **Exploding gradients** : les gradients deviennent très grands et rendent l'apprentissage instable.

— **Vanishing gradients** : les gradients deviennent trop petits, ralentissant ou bloquant l'apprentissage, surtout dans les réseaux profonds.

Des techniques comme la normalisation des poids, l'utilisation de fonctions d'activation comme Relu, ou des initialisations spécifiques (e.g. He ou Xavier) permettent de limiter ces effets [9]. En résumé, l'entraînement d'un réseau de neurones repose sur une compréhension approfondie et une implémentation rigoureuse de ces quatre composants. Chacune de ces étapes joue un rôle crucial dans la construction d'un modèle robuste et performant, capable de généraliser à de nouvelles données. Les avancées en matière de techniques d'optimisation et de gestion des données permettent d'améliorer constamment les capacités des réseaux de neurones, ouvrant la voie à des applications innovantes dans de nombreux domaines.

2.6 Apprentissage profond

2.6.1 Réseaux de neurones convolutifs

Un réseau de neurones convolutifs (ou `CNN`, pour *Convolutional Neural Network*) est une architecture spécialisée des réseaux de neurones artificiels, conçue pour traiter des données structurées en grille, telles que les images. La particularité des CNN réside dans l'utilisation de couches de **convolution**, qui permettent d'extraire automatiquement des caractéristiques locales pertinentes à partir des données d'entrée.

Ces couches appliquent des filtres (ou noyaux de convolution) qui glissent sur l'image pour produire des cartes de caractéristiques. Chaque filtre apprend à détec-

ter un type particulier de motif : contours, textures, formes, etc. Le résultat est un ensemble de cartes de caractéristiques, représentant différentes composantes visuelles de l'image à différentes échelles et orientations.

La figure 2.7 illustre un exemple typique d'architecture CNN, composée de couches de convolution, de sous-échantillonnage (pooling), puis de couches entièrement connectées, utilisées pour la classification finale.

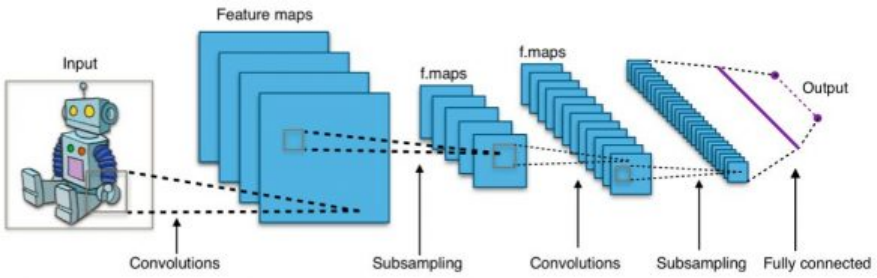

FIGURE 2.7 – Architecture typique d'un réseau de neurones convolutif. [27]

Un CNN complet comprend généralement plusieurs couches, chacune jouant un rôle spécifique :

— **Couches de convolution** : extraction automatique des caractéristiques locales.
— **Couches de pooling** : réduction de la dimension des données tout en conservant l'essentiel de l'information.
— **Couches entièrement connectées** : interprétation globale des caractéristiques extraites pour réaliser la prédiction ou la classification.

2.6.2 Couches de convolution

Avant de nous plonger dans les détails techniques de la couche de convolution, il est essentiel de comprendre le concept de "caractéristique" dans le contexte de la classification d'images. Supposons que nous souhaitions concevoir un système capable de reconnaître une image contenant un oiseau. L'objectif serait que, lorsqu'on pré-

sente une image d'oiseau à l'ordinateur, celui-ci identifie correctement la présence de l'oiseau.

Une solution intuitive pourrait consister à comparer chaque pixel de l'image d'apprentissage (une image de référence) avec chaque pixel de l'image à classifier. Si les deux images correspondent pixel par pixel, alors on pourrait conclure qu'il s'agit bien d'un oiseau. Toutefois, cette méthode présente un inconvénient majeur : elle exige que l'image à classifier soit strictement identique à l'image de référence, ce qui est rarement le cas dans les situations réelles. En effet, les images peuvent varier en termes de résolution, d'éclairage, d'orientation, d'arrière-plan, etc.

Pour surmonter ces limites, une approche plus robuste consiste à extraire des caractéristiques spécifiques de l'image source (comme des motifs ou des textures) et à rechercher ces caractéristiques dans l'image à classifier. Cette méthode repose sur le **principe de convolution**. La convolution consiste à appliquer un filtre (ou noyau) sur l'ensemble de l'image pour générer une nouvelle représentation appelée carte de caractéristiques. Le filtre agit comme une fenêtre glissante, mettant en évidence certains motifs comme les contours, les textures ou les motifs récurrents, qui sont des éléments clés pour la reconnaissance d'objets.

Les filtres utilisés dans une couche de convolution ne sont pas fixés à l'avance. Ils sont appris automatiquement durant le processus d'entraînement du réseau, ce qui permet au modèle d'identifier de manière autonome les motifs les plus discriminants pour la tâche visée [30]. Par exemple, dans le cadre de la reconnaissance d'oiseaux, le réseau pourrait apprendre à détecter des plumes, des becs ou des formes d'ailes [36].

La figure 2.8 illustre le fonctionnement d'une opération de convolution. On y voit un filtre se déplacer sur une image pour extraire des motifs visuels utiles à l'apprentissage automatique.

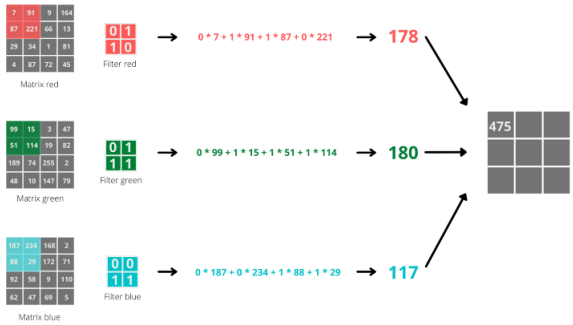

FIGURE 2.8 – Principe de la convolution appliqué à une image : un filtre génère une carte de caractéristiques.

Ce mécanisme présente également l'avantage d'être partiellement invariant aux transformations comme la translation ou la rotation : même si l'objet recherché apparaît à un endroit différent ou sous un angle nouveau, le réseau peut le détecter grâce aux motifs appris [25]. Cette propriété confère aux CNN une grande robustesse dans des environnements visuels variés [19].

En résumé, la couche de convolution joue un rôle fondamental dans l'extraction de caractéristiques visuelles pertinentes. Elle permet de transformer des images complexes en représentations plus abstraites et structurées, qui peuvent ensuite être exploitées par les couches suivantes du réseau pour effectuer la classification ou la détection.

2.6.3 Couche de pooling

Une fois que l'image a été traitée par les couches de convolution, une étape essentielle consiste à appliquer une couche de **pooling**. Ce processus permet d'extraire les informations les plus pertinentes tout en réduisant la dimensionnalité des cartes de caractéristiques. Le pooling est crucial pour réduire la complexité computationnelle, limiter le risque de surapprentissage et améliorer la généralisation du modèle.

Le pooling s'effectue via une fenêtre glissante — appelée noyau de pooling — qui se déplace sur la carte de caractéristiques. Deux méthodes principales sont couramment utilisées : le **max pooling** et l'**average pooling**.

— **Max pooling** : sélectionne la valeur maximale dans chaque sous-région définie par la fenêtre. Cette méthode met en évidence les caractéristiques les plus saillantes, telles que les bords ou les textures fortes.
— **Average pooling** : calcule la moyenne des valeurs dans chaque région. Elle est utile lorsque l'on souhaite obtenir une représentation plus lissée ou réduire l'influence du bruit.

Ces deux techniques sont illustrées dans la figure 2.9, qui montre comment la taille des cartes est réduite tout en conservant des informations significatives.

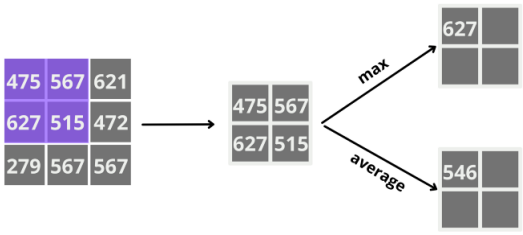

FIGURE 2.9 – Illustration du *max pooling* et de l'*average pooling*.

Le pooling contribue également à rendre le réseau plus robuste aux transformations mineures comme la translation, la rotation ou la mise à l'échelle, car il consolide les informations en une forme plus abstraite et localement invariante.

En complément, certaines architectures intègrent :

1. **Des couches de normalisation**, utilisées pour standardiser les activations et stabiliser l'entraînement du réseau.
2. **Des couches entièrement connectées**, qui prennent le relais à la fin du réseau pour effectuer la classification. Elles combinent les cartes de caractéris-

tiques réduites et génèrent la sortie finale. Le fonctionnement de ces couches est similaire à celui des neurones dans un réseau traditionnel dense.

Les réseaux de neurones convolutifs (`CNN`) sont particulièrement puissants pour des tâches telles que la classification d'images, la détection d'objets ou la segmentation sémantique. Les premières couches de convolution détectent des caractéristiques de bas niveau (bords, textures), tandis que les couches plus profondes intègrent ces signaux pour reconnaître des motifs complexes ou des objets entiers.

Parmi les avantages majeurs des CNN, on trouve : - une bonne gestion de la variance dans les données d'entrée, - une réduction du nombre de paramètres grâce au partage de poids, - et une capacité à apprendre des représentations invariantes aux transformations.

Ces propriétés font des CNN un outil de premier plan dans des domaines comme la reconnaissance faciale, la conduite autonome, ou l'analyse d'images médicales.

2.6.4 Application de l'apprentissage profond sur notre jeu de données

Dans le cadre de ce projet, nous avons utilisé un jeu de données public intitulé *garbage classification*, disponible sur la plateforme kaggle. [1] Ce jeu de données contient des images de déchets réparties en dix classes distinctes, représentant différentes catégories d'ordures telles que le plastique, le papier, le métal, le verre, le carton, les déchets organiques, les vêtements, les chaussures, les piles et les déchets résiduels. Il a été conçu pour servir de base à des projets d'apprentissage automatique et de vision par ordinateur axés sur le tri, le recyclage et la gestion durable des déchets. Ce corpus

1. `https://www.kaggle.com/datasets/mostafaabla/garbage-classification`

s'avère particulièrement adapté au développement de modèles de classification ou de détection d'objets, et à la création de solutions intelligentes pour la gestion écologique des déchets.

L'objectif était de démontrer l'efficacité de l'apprentissage profond dans le cadre d'une tâche de classification d'images de déchets. À cette fin, nous avons développé un modèle de type convolutif et l'avons entraîné sur ce jeu de données. L'entraînement supervisé a conduit à des résultats probants : le modèle a atteint une précision de **98,86 %** sur le jeu d'entraînement, avec une perte de **0,0973**, et une précision de **92,10 %** sur le jeu de validation, avec une perte de **0,2788**.

L'arrêt anticipé de l'entraînement à la **19^{e} époque** a été déclenché par un mécanisme de *early stopping*, qui surveille la performance sur le jeu de validation afin d'éviter le surapprentissage. L'écart observé entre les performances d'entraînement et de validation (environ 6,76 points de pourcentage) reste raisonnable, ce qui témoigne d'une bonne généralisation du modèle sur des données non vues. La rapidité de convergence de la fonction de perte valide la pertinence de l'architecture du réseau neuronal ainsi que l'efficacité des paramètres d'optimisation choisis.

Ces résultats soulignent le potentiel de l'apprentissage profond dans le domaine du tri automatisé des déchets. Le modèle entraîné pourrait ainsi constituer la base d'un système embarqué dans des bacs intelligents, ou être intégré dans des dispositifs industriels de tri sélectif. Cette étude expérimentale illustre concrètement comment les technologies d'intelligence artificielle peuvent contribuer à la transition vers une gestion plus durable et automatisée des déchets.

Résultats d'entraînement

Le modèle a été entraîné pendant 40 époques avec une stratégie d'arrêt anticipé (*early stopping*) afin de prévenir le surapprentissage. Les performances obtenues durant l'apprentissage sont résumées dans le tableau 2.2, qui présente les pertes (*loss*) et précisions (*accuracy*) à la fois pour les ensembles d'entraînement et de validation.

Époque	**Train Loss**	**Train Accuracy (%)**	**Validation Loss**	**Validation Accuracy (%)**
1	2,0753	29,18	1,5777	64,95
2	1,2988	67,22	1,0053	76,98
3	0,9059	78,07	0,7583	81,44
4	0,6992	83,52	0,6066	85,05
5	0,5650	86,50	0,5346	84,19
10	0,2574	94,89	0,3141	91,92
14	0,1572	97,46	0,2728	92,44
18	0,1011	98,90	0,2753	92,10

TABLE 2.2 – Pertes et précisions d'entraînement et de validation pour différentes époques.

Comme on peut le constater, la perte d'entraînement diminue régulièrement, tandis que la précision augmente de manière significative dès les premières époques. Le modèle atteint une précision de validation maximale de **92,44 %** à l'époque 14. L'arrêt anticipé déclenche la fin de l'entraînement à l'époque 19, évitant ainsi tout surajustement au jeu d'entraînement.

Les courbes d'évolution de la perte et de la précision pour les ensembles d'entraînement et de validation sont présentées à la figure 2.10. Ces courbes permettent de visualiser la dynamique d'apprentissage et la stabilité du modèle.

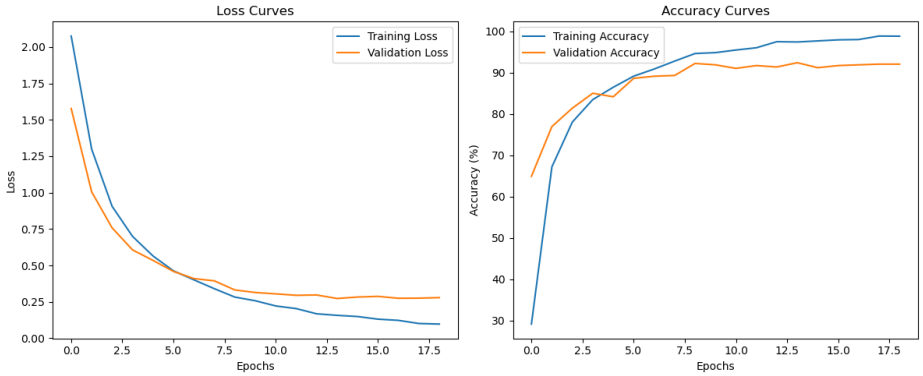

FIGURE 2.10 – Évolution des courbes de perte et de précision sur les ensembles d'entraînement et de validation.

Courbe de perte

Tendance générale :

— La perte pour l'ensemble d'entraînement décroît de manière constante, indiquant que le modèle apprend progressivement à mieux prédire les étiquettes des données.

— La perte de validation suit une trajectoire similaire au début, mais tend à se stabiliser autour d'une valeur minimale après plusieurs époques.

Convergence :

— La perte d'entraînement converge rapidement dans les premières époques, atteignant une valeur faible. Cela démontre une bonne capacité d'apprentissage initial du modèle.

— La perte de validation montre une tendance à converger sans augmentation significative, suggérant que le modèle ne surapprend pas (pas d'overfitting).

Écart entre entraînement et validation :

— L'écart entre la perte d'entraînement et celle de validation est minime après plusieurs époques, indiquant une bonne généralisation du modèle sur les données non vues.

2.6.5 Courbe d'accuracy

Tendance générale :

— L'accuracy pour l'ensemble d'entraînement augmente de manière rapide dans les premières époques, atteignant une précision proche de 98%.

— L'accuracy de validation progresse également rapidement et reste stable après quelques époques, avoisinant 92%.

Convergence et stabilité :

— Après environ 10 à 12 époques, l'accuracy de validation atteint un plateau. Cela suggère que le modèle a appris suffisamment des données d'entraînement pour bien se généraliser.

Absence de surapprentissage :

— Les courbes d'entraînement et de validation montrent des trajectoires similaires sans divergence significative. Cela indique que le modèle est bien régularisé et qu'il n'a pas surappris les données d'entraînement.

2.6.6 Interprétation des résultats

Performance globale :

— La diminution de la perte combinée avec une augmentation régulière de l'accuracy confirme que le modèle optimise efficacement les paramètres pour minimiser les erreurs.

— Les valeurs atteintes pour l'accuracy de validation (92%) sont cohérentes avec les objectifs de performance pour la classification de déchets.

Généralisation :

— Le comportement des courbes suggère que le modèle est capable de généraliser efficacement aux données non vues, ce qui est un critère essentiel dans le cadre d'un problème réel.

Points d'amélioration :

— Bien que les résultats soient satisfaisants, une légère optimisation du modèle ou des hyperparamètres (comme la réduction du learning rate ou l'augmentation des données) pourrait encore améliorer la stabilité des courbes de validation.

Chapitre 3

Deep clustering

Le *clustering*, ou regroupement non supervisé, vise à partitionner un ensemble de données en sous-groupes homogènes appelés *clusters*, de manière à ce que les éléments au sein d'un même cluster soient plus similaires entre eux qu'avec ceux des autres groupes [14, 22]. Bien que des méthodes classiques comme k-means, le clustering hiérarchique ou les modèles de mélange gaussiens aient longtemps dominé ce champ, elles se heurtent à des limites lorsqu'il s'agit de traiter des données complexes et de haute dimension.

Avec l'émergence de l'apprentissage profond, le *deep clustering* s'est imposé comme une approche hybride combinant les capacités de représentation des réseaux de neurones profonds avec les algorithmes de clustering traditionnels [26, 43]. Cette synergie permet d'apprendre des représentations latentes plus expressives, facilitant la détection de structures sous-jacentes dans les données.

L'une des approches les plus populaires repose sur l'utilisation d'un autoencodeur pour projeter les données dans un espace latent compact, suivi d'un regroupement dans cet espace [10]. Des modèles tels que le deep Embedded Clustering (DEC) illus-

trent cette intégration en optimisant simultanément la reconstruction des données et la formation de clusters cohérents [43, 45].

Le deep clustering se révèle particulièrement performant dans des domaines comme la vision par ordinateur, où il permet de capturer des relations non linéaires et des structures complexes. Grâce à sa flexibilité et sa capacité à traiter de grands volumes de données, il offre un cadre puissant pour l'analyse non supervisée moderne [26].

Dans ce mémoire, nous explorons l'application du deep clustering à la classification d'images de déchets, illustrant ainsi son potentiel dans un contexte écologique et technologique.

3.0.1 Présentation

Dans le domaine de l'analyse d'images, les réseaux de neurones profonds se sont imposés comme des outils particulièrement performants, tant pour la reconnaissance visuelle que pour la détection d'objets. Leur efficacité repose sur leur capacité à extraire automatiquement des caractéristiques discriminantes à partir des données brutes, sans nécessiter une ingénierie manuelle préalable.

Cependant, l'approche la plus répandue — l'*apprentissage supervisé* — nécessite une vérité terrain (*ground truth*) associée à chaque image du jeu d'entraînement. Cela signifie que chaque donnée d'entrée doit être accompagnée de sa classe cible, permettant au modèle de quantifier son erreur et d'ajuster ses paramètres à l'aide de la rétropropagation. Ce processus se répète jusqu'à convergence, c'est-à-dire jusqu'à ce que la performance ne s'améliore plus significativement.

L'efficacité de l'apprentissage supervisé dépend fortement de la qualité et de la quantité d'annotations disponibles. Or, dans de nombreux cas concrets, l'étiquetage

manuel est coûteux, long, voire irréalisable à grande échelle.

C'est dans ce contexte que le **deep clustering** émerge comme une alternative pertinente. Cette approche non supervisée vise à regrouper les données non étiquetées en clusters cohérents, tout en apprenant des représentations latentes robustes grâce à la puissance des réseaux de neurones profonds. En combinant apprentissage de représentations et techniques de regroupement, le deep clustering permet de structurer des ensembles d'images complexes sans recourir à une supervision explicite.

La figure 3.1 présente une vue schématique du fonctionnement des réseaux de neurones, utilisée pour illustrer les étapes successives de traitement des données d'entrée.

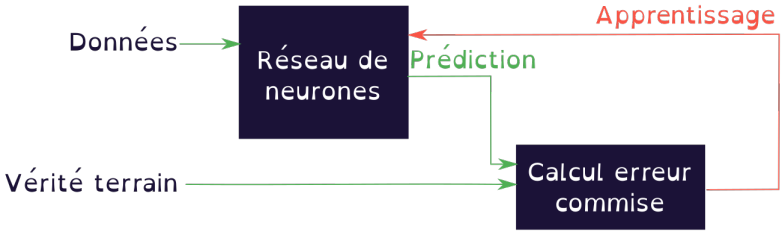

FIGURE 3.1 – Architecture schématique d'un réseau de neurones profonds. [23]

3.0.2 Principe du deep clustering

Le *clustering*, ou regroupement non supervisé, vise à organiser des données en ensembles homogènes appelés *clusters*, en se basant uniquement sur des mesures de similarité. Appliqué aux images, ce processus nécessite en amont une représentation numérique pertinente de chaque image, généralement sous forme de vecteurs de caractéristiques. Dans ce contexte, les réseaux de neurones profonds se révèlent particulièrement efficaces pour extraire automatiquement des descripteurs adaptés à la structure sous-jacente des données visuelles.

Le *deep clustering* repose sur une synergie entre l'apprentissage non supervisé et

l'expressivité des réseaux de neurones. Une méthode représentative de cette approche est **DeepCluster**, qui procède de manière itérative selon les étapes suivantes :

— Un réseau de neurones convolutifs (souvent préentraîné) est utilisé pour extraire des représentations latentes (ou descripteurs) à partir des images.
— Ces descripteurs sont ensuite regroupés en clusters à l'aide d'un algorithme tel que k-means.
— Les labels de clusters obtenus sont temporairement utilisés comme pseudo-étiquettes pour réentraîner le réseau, comme s'il s'agissait d'une vérité terrain.

Ce processus est répété sur plusieurs itérations. À mesure que le réseau apprend à produire des représentations plus discriminantes, les regroupements deviennent eux aussi plus cohérents. Il en résulte une boucle d'amélioration mutuelle entre l'extraction de caractéristiques et la formation des clusters. La figure 3.2 illustre le principe général du deep clustering, combinant représentation latente, regroupement par similarité, et réentraînement du réseau. Cette méthode permet ainsi d'exploiter efficacement de vastes

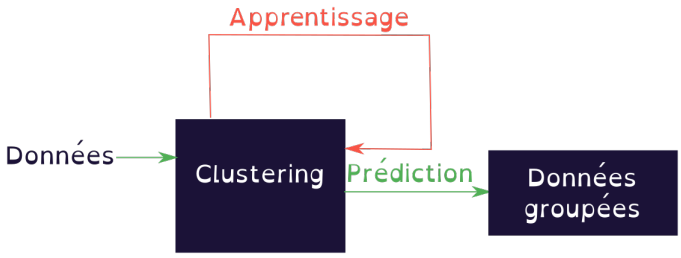

FIGURE 3.2 – Illustration du principe du deep clustering. [23]

ensembles d'images non annotées, en améliorant progressivement les performances du modèle sans supervision explicite. Le deep clustering est donc particulièrement adapté aux problématiques où les annotations manuelles sont rares, incomplètes ou coûteuses à produire, comme c'est le cas dans la classification automatique des déchets ménagers. Il existe plusieurs algorithmes qui intègrent l'apprentissage profond et le clustering. Voici quelques-uns des plus courants :

3.1 Deep embedded clustering (DEC)

Proposé par [43], DEC est une méthode innovante qui combine les avantages des autoencodeurs et des algorithmes de clustering pour améliorer la qualité du regroupement des données. Cette méthode repose sur l'idée d'apprendre une représentation compacte des données à l'aide d'un autoencodeur, puis d'appliquer l'algorithme k-means sur cet espace latent. L'autoencodeur est un réseau de neurones non supervisé qui apprend à compresser les données dans un espace latent et à les reconstruire à partir de cet espace.

3.1.1 Autoencodeur

Un **autoencodeur** est un type de réseau de neurones non supervisé conçu pour apprendre une représentation compressée des données, également appelée représentation latente. Il est composé de deux composants principaux : un *encodeur* et un *décodeur*.

— **Encodeur** : Il transforme les données d'entrée en une représentation latente de dimension réduite. Cette compression permet de capturer les caractéristiques essentielles tout en filtrant le bruit et les redondances.

— **Décodeur** : Il reconstruit les données d'entrée à partir de cette représentation latente. Le but est de minimiser la différence entre les données originales et leur reconstruction, ce qui force le réseau à apprendre une structure interne significative des données.

La figure 3.3 présente une architecture schématique d'un autoencodeur classique, mettant en évidence la symétrie entre l'encodeur et le décodeur autour du goulot d'étranglement central (la couche latente).

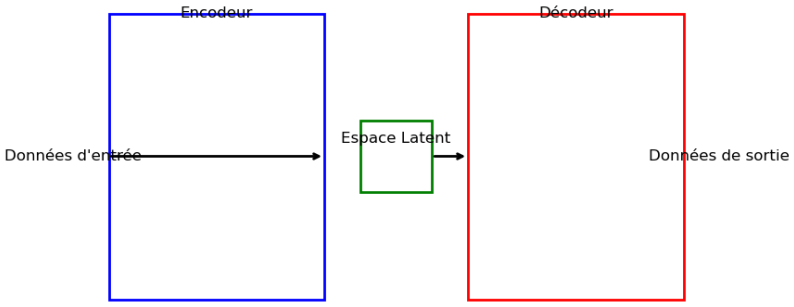

FIGURE 3.3 – Architecture typique d'un autoencodeur montrant la compression (encodeur) suivie de la reconstruction (décodeur).

3.1.2 Architecture de l'autoencodeur

L'architecture d'un autoencodeur peut varier en fonction de la complexité des données et de l'application spécifique. Dans le cas de DEC, l'architecture typique est la suivante :

3.1.3 Apprentissage de l'autoencodeur

L'apprentissage de l'autoencodeur consiste à minimiser la différence entre les données d'entrée et les données reconstruites. Cela se fait en optimisant une fonction de coût, souvent la perte de reconstruction (par exemple, l'erreur quadratique moyenne ou la perte binaire croisée).

— **Phase d'initialisation** : L'autoencodeur est pré-entraîné pour apprendre une bonne représentation latente des données. Cela peut être fait en utilisant des techniques telles que la descente de gradient stochastique (SGD) ou Adam.

— **Phase de clustering** : Une fois l'autoencodeur pré-entraîné, les représentations latentes des données sont extraites et utilisées comme entrée pour l'algo-

rithme k-means.

Itération et affinement

Le processus de clustering avec DEC (*Deep Embedded Clustering*) est de nature itérative. Après une première phase où l'algorithme k-means est appliqué sur les représentations latentes extraites par l'autoencodeur, les étiquettes de clusters ainsi obtenues sont utilisées comme pseudo-étiquettes pour guider l'affinement du modèle. L'objectif est alors de minimiser la divergence de Kullback-Leibler (KL) entre la distribution des affectations de clusters prédite par le modèle et une distribution cible artificiellement resserrée autour des centres de clusters. Cette démarche crée un mécanisme d'auto-amélioration : à mesure que l'autoencodeur affine ses représentations, la qualité des clusters s'améliore également, ce qui alimente en retour une meilleure mise à jour du modèle. La fonction de perte globale combinée peut être exprimée comme suit :

$$\mathrm{L} = \mathrm{L}_{\mathrm{rec}} + \gamma \mathrm{L}_{\mathrm{KL}} \tag{3.1}$$

où : - $\mathrm{L}_{\mathrm{rec}}$ désigne la perte de reconstruction de l'autoencodeur, - L_{KL} correspond à la divergence de Kullback-Leibler entre la distribution prédite et la cible, - γ est un hyperparamètre permettant d'équilibrer l'importance relative des deux termes dans l'objectif d'optimisation. La figure 3.4 illustre les étapes principales du processus DEC, depuis l'encodage des données jusqu'au raffinement des clusters à travers une boucle d'apprentissage itérative. La méthode DEC représente une avancée importante en matière de clustering non supervisé, en tirant parti de la capacité des réseaux de neurones profonds à apprendre des représentations discriminantes, tout en s'appuyant sur la simplicité et l'efficacité de k-means. Elle permet de découvrir des structures sous-jacentes complexes dans les données non étiquetées, ouvrant ainsi de nouvelles perspectives pour l'analyse exploratoire de données visuelles ou textuelles.

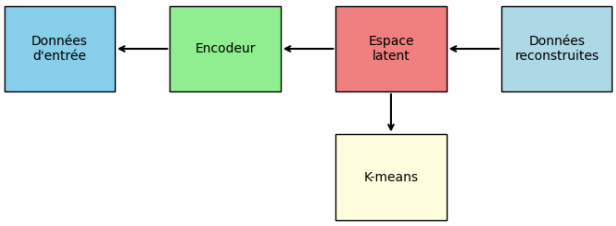

FIGURE 3.4 – Schéma du processus DEC : apprentissage conjoint de l'espace latent et des clusters.

3.1.4 Types d'autoencodeurs profonds

Les autoencodeurs sont des réseaux de neurones non supervisés conçus pour apprendre une représentation latente compressée des données en minimisant la différence entre l'entrée et sa reconstruction. Parmi leurs variantes les plus avancées, on retrouve l'**autoencodeur variationnel (VAE)** et le **Variational Deep Embedding (VaDE)**, qui intègrent des principes probabilistes pour des tâches telles que la génération de données ou le clustering.

Autoencodeur variationnel (VAE) Proposé par [20], le VAE est un modèle génératif qui encode chaque observation $\mathbf{x}$ dans une distribution latente $q_\phi(\mathbf{z}|\mathbf{x})$ supposée gaussienne. L'apprentissage repose sur la maximisation d'une borne inférieure (ELBO) de la log-vraisemblance marginale, combinant un terme de reconstruction et une régularisation de la distribution latente via la divergence de Kullback-Leibler :

$$\mathcal{L}(\mathbf{x};\theta,\phi) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] - \mathrm{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z})). \tag{3.2}$$

Le VAE permet ainsi de générer de nouvelles données tout en structurant l'espace latent, ce qui le rend utile pour la compression, l'exploration de données et le clustering.

Variational deep embedding (VaDE) Développé par [13], VaDE combine les VAE avec un modèle de mélange gaussien (GMM) pour effectuer un clustering non supervisé. Dans ce cadre, la variable latente $\mathbf{z}$ suit une distribution $p(\mathbf{z}|\mathbf{y})$ dépendant d'un cluster latent $\mathbf{y}$. Le modèle optimise la probabilité jointe $P(\mathbf{x}, \mathbf{z}, \mathbf{y})$ en intégrant simultanément les paramètres du VAE et du GMM :

$$P(\mathbf{x}, \mathbf{z}, \mathbf{y}) = P(\mathbf{y})P(\mathbf{z}|\mathbf{y})P(\mathbf{x}|\mathbf{z}). \tag{3.3}$$

VaDE surpasse des méthodes comme k-means ou les GMM traditionnels sur des ensembles tels que MNIST ou Reuters-10k, car il capture plus fidèlement les structures complexes des données non étiquetées.

Applications et limites Les autoencodeurs profonds trouvent des applications variées : génération d'images réalistes [32], analyse de données moléculaires [**?**], ou classification non supervisée. Toutefois, certains défis persistent, notamment le *posterior collapse*, où le VAE ignore l'espace latent pour privilégier une reconstruction triviale [31].

3.2 Algorithme k-means et sa variante minibatch k-means

L'algorithme k-means est l'une des méthodes de clustering les plus populaires et les plus simples à mettre en œuvre. Il permet de partitionner un ensemble de données en k

clusters en minimisant la variance intra-cluster, c'est-à-dire la distance entre les points de données et le centre (centroïde) de leur cluster respectif [22]. Le fonctionnement de k-means se déroule selon les étapes suivantes :

- **Initialisation** : Choisir aléatoirement k points dans l'espace des données comme centroïdes initiaux.
- **Assignation** : Pour chaque point de données, calculer la distance à chacun des k centroïdes et l'assigner au cluster le plus proche (souvent à l'aide de la distance euclidienne).
- **Mise à jour** : Recalculer les centroïdes de chaque cluster en prenant la moyenne des points qui leur ont été assignés.
- **Répétition** : Répéter les étapes d'assignation et de mise à jour jusqu'à ce que la convergence soit atteinte, c'est-à-dire que les centroïdes ne changent plus significativement ou que le nombre maximal d'itérations soit atteint.

Bien que k-means soit efficace sur des ensembles de données de taille modérée, son coût computationnel devient prohibitif pour des jeux de données volumineux, notamment lors de l'entraînement de modèles de deep clustering sur des millions d'images. Pour pallier ce problème, la variante *minibatch k-means* a été introduite par [38]. Cette approche conserve le principe fondamental de k-means mais utilise des sous-échantillons aléatoires (minibatchs) au lieu d'utiliser l'ensemble complet des données à chaque itération. Le fonctionnement est le suivant :

- À chaque itération, un minibatch de taille fixe (par exemple, 100 ou 1000 points) est échantillonné aléatoirement.
- Les points du minibatch sont assignés aux centroïdes les plus proches, comme dans k-means classique.
- Les centroïdes sont mis à jour de façon incrémentale à l'aide d'un schéma de moyenne pondérée (souvent une descente de gradient stochastique).

Cette approche présente plusieurs avantages :

- Réduction significative du temps de calcul, ce qui la rend adaptée aux grands ensembles de données.
- Convergence plus rapide, bien que légèrement moins précise que k-means stan-

dard.

— Facilité d'intégration dans des pipelines d'apprentissage profond impliquant des itérations fréquentes (comme dans DEC).

Dans le cadre de méthodes comme DEC, *minibatch k-means* est souvent privilégiée pour la phase de clustering sur les représentations latentes, en raison de sa rapidité et de sa capacité à gérer efficacement les grands volumes de données générés par les réseaux de neurones.

3.3 Évaluation du clustering

L'évaluation du clustering est essentielle pour mesurer la qualité des clusters formés par un algorithme, qu'il soit supervisé ou non supervisé. Dans cette section, nous détaillons les différentes métriques utilisées pour l'évaluation ainsi qu'une description de l'agglomerative clustering, un algorithme hiérarchique populaire.

3.3.1 Métriques d'évaluation

Pour évaluer la qualité des clusters, plusieurs métriques peuvent être utilisées, notamment :

3.3.2 Indice de silhouette

L'indice de silhouette, défini par Kaufman et Rousseeuw (1990) [16], est calculé comme suit [5] :

$$\text{Silhouette} = \frac{1}{n}\sum_{i=1}^{n} S(i), \quad \text{où Silhouette} \in [-1, 1]. \tag{3.4}$$

Avec :

$$S(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}} \tag{3.5}$$

— a(i) : La dissimilarité moyenne de l'objet i à tous les autres objets du même cluster C_r, donnée par :

$$a(i) = \frac{1}{n_r - 1}\sum_{j \in C_r} d_{ij} \tag{3.6}$$

où n_r est le nombre d'objets dans le cluster C_r et d_{ij} est la distance entre les objets i et j.

— b(i) : La dissimilarité moyenne minimale entre l'objet i et tous les objets d'un autre cluster C_s, donnée par :

$$b(i) = \min_{s \neq r}\left(\frac{1}{n_s}\sum_{j \in C_s} d_{ij}\right) \tag{3.7}$$

où n_s est le nombre d'objets dans le cluster C_s.

Les valeurs maximales de l'indice de silhouette sont utilisées pour déterminer le nombre optimal de clusters dans les données. Cet indice ne peut pas être calculé lorsque k = 1 (un seul cluster). Le programme pour cet indice provient de la fonction `index.S` du package **ClusterSim**, [34].

Indice de Davies-Bouldin

L'indice de Davies-Bouldin mesure la compacité et la séparation des clusters [6] :

$$DB = \frac{1}{k} \sum_{i=1}^{k} \max_{j \neq i} \frac{s_i + s_j}{d_{ij}}, \tag{3.8}$$

où s_i est la dispersion moyenne du cluster i, et d_{ij} est la distance entre les centres des clusters i et j. Un indice plus faible indique des clusters mieux séparés.

Indice de Calinski-Harabasz

Cet indice évalue le rapport entre la dispersion inter-cluster et intra-cluster [3] :

$$CH = \frac{\mathrm{Tr}(B_k)}{\mathrm{Tr}(W_k)} \cdot \frac{n-k}{k-1}, \tag{3.9}$$

où $\mathrm{Tr}(B_k)$ et $\mathrm{Tr}(W_k)$ représentent respectivement la trace des matrices de dispersion inter-cluster et intra-cluster. Un indice élevé indique des clusters bien formés.

3.3.3 Agglomerative clustering

L'agglomerative clustering, également appelé regroupement hiérarchique agglomératif, est une méthode itérative qui commence par considérer chaque point de données comme un cluster individuel. Les clusters sont ensuite fusionnés de manière hiérarchique jusqu'à ce qu'il ne reste qu'un seul cluster contenant tous les points [33].

Fonctionnement

Le processus de l'agglomerative clustering suit les étapes suivantes :

1. Chaque point est initialement considéré comme un cluster individuel.
2. La similarité ou distance entre chaque paire de clusters est calculée. Les mesures de distance courantes incluent la distance euclidienne, la distance de Manhattan et le cosinus de similarité.
3. Les deux clusters les plus similaires sont fusionnés.
4. Les étapes 2 et 3 sont répétées jusqu'à ce qu'un critère d'arrêt soit atteint, comme un nombre cible de clusters ou une distance seuil.

Avantages et Limites

— **Avantages** :
  — Capable de capturer des structures hiérarchiques dans les données.
  — Aucune hypothèse sur la forme ou la taille des clusters.
— **Limites** :
  — Complexité computationnelle élevée pour de grands ensembles de données ($\mathcal{O}(n^3)$ en version brute).
  — Sensible aux points aberrants.

3.3.4 Indice gap

L'indice Gap, défini par Tibshirani *et al.* (2001), est estimé comme suit :

$$\text{Gap}(q) = \frac{1}{B}\sum_{b=1}^{B}\log(W_q^b) - \log(W_q). \tag{3.10}$$

— B : Nombre de jeux de données de référence générés à l'aide d'une prescription uniforme.

— W_q^b : Dispersion intra-cluster définie comme dans l'indice de Hartigan.

Pour déterminer le nombre optimal de clusters q, on recherche le plus petit q tel que :

$$\text{Gap}(q) \geq \text{Gap}(q+1) - s_{q+1} \tag{3.11}$$

où :

$$s_q = \text{sd}_q \sqrt{1 + \frac{1}{B}} \tag{3.12}$$

et sd_q est l'écart-type donné par :

$$\text{sd}_q = \sqrt{\frac{1}{B} \sum_{b=1}^{B} \left(\log(W_q^b) - \bar{l} \right)^2} \tag{3.13}$$

où $\bar{l}$ est la moyenne :

$$\bar{l} = \frac{1}{B} \sum_{b=1}^{B} \log(W_q^b). \tag{3.14}$$

Dans **NbClust**, l'indice Gap est calculé uniquement si l'option `"index" = "gap"` ou `"alllong"` est activée, en raison de son coût computationnel élevé [41], [42].

Chapitre 4

Application sur notre jeu de données

Ce chapitre présente les résultats obtenus à partir de l'application de notre modèle de classification basé sur le deep clustering, ainsi que les discussions approfondies qui en découlent. L'objectif est d'évaluer la performance du modèle, d'interpréter les métriques clés et de mettre en perspective les résultats avec les travaux existants dans le domaine.

Nous commencerons par analyser les courbes de perte et de précision pour l'ensemble d'entraînement et de validation afin de mieux comprendre la convergence et la généralisation du modèle. Ensuite, nous examinerons les performances de classification à travers la matrice de confusion et le rapport de classification. Enfin, une comparaison avec d'autres approches, notamment les algorithmes supervisés de deep learning, permettra de situer notre méthode dans le paysage actuel de la recherche.

Les discussions mettront en lumière les forces et les limites du modèle proposé, tout en suggérant des pistes d'amélioration et des applications potentielles dans des

scénarios pratiques, comme la gestion des déchets. Ce chapitre vise ainsi à fournir une vision complète et critique des résultats obtenus.

4.1 Analyse détaillée des résultats

Cette section présente une analyse complète des résultats obtenus, ainsi qu'une comparaison avec l'article *Deep Clustering for Unsupervised Learning of Visual Features* de Caron et al. Cette analyse met en lumière les points forts et les limitations de notre modèle de deep clustering.

4.2 Méthodologie et analyse du deep clustering

4.2.1 Description des données

Nous disposons d'un jeu de données constitué de 19 407 images réparties en 10 classes (métal, verre, biologique, papier, batterie, déchets divers, carton, chaussures, vêtements, et plastique). Pour l'entraînement, nous utilisons 70% des données, soit environ 13 585 images, et 30% pour le test, soit environ 5 822 images. Les données utilisées dans cette étude proviennent d'un ensemble de données trouvé sur gaggle intitulé *garbage classification*. La répartition de cet ensemble est donnée ci-dessous :

— Métal : 994 images
— Verre : 3039 images
— Biologique : 983 images
— Papier : 1650 images
— Batterie : 944 images

— Déchets divers (Trash) : 772 images
— Carton : 1810 images
— Chaussures : 1977 images
— Vêtements : 5323 images
— Plastique : 1915 images

4.2.2 Pipeline général

Le pipeline expérimental mis en œuvre repose sur une chaîne de traitement complète, combinant des techniques d'extraction de caractéristiques, de réduction dimensionnelle, de regroupement non supervisé et d'évaluation des performances. L'ensemble de ces étapes est représenté schématiquement dans la figure 4.1.

— **Extraction de caractéristiques** : Les images sont encodées à l'aide du modèle préentraîné *ResNet-50*, largement reconnu pour sa capacité à générer des représentations visuelles robustes et discriminantes. Ce réseau convolutif profond est utilisé ici comme extracteur de descripteurs latents.
— **Réduction dimensionnelle** : Afin de faciliter la visualisation et d'améliorer la séparation des clusters dans un espace à faible dimension, l'algorithme UMAP(Uniform Manifold Approximation and Projection) est appliqué pour projeter les descripteurs dans un espace bidimensionnel.
— **Clustering** : Deux algorithmes complémentaires de regroupement ont été testés :
  — MiniBatch k-means : une version optimisée de k-means qui fonctionne efficacement sur de grands ensembles de données en traitant les échantillons par lot.
  — Agglomerative Clustering : un algorithme hiérarchique qui permet de capturer les relations structurelles entre groupes en construisant une arborescence de fusions successives.

— **Évaluation** : Les performances du modèle sont mesurées à l'aide de plusieurs indicateurs :

— Le *silhouette score*, utilisé pour évaluer la qualité du regroupement selon la cohésion interne et la séparation inter-cluster.

— Un *rapport de classification* basé sur des métriques standards telles que la précision, le rappel, le F1-score, ainsi que la matrice de confusion.

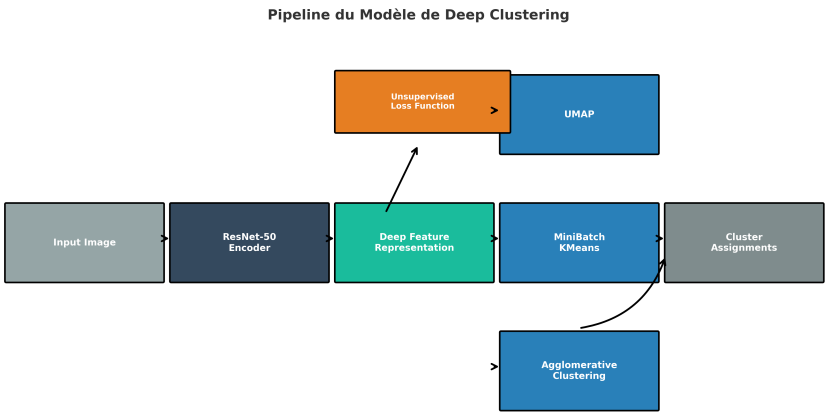

FIGURE 4.1 – Illustration du pipeline de deep clustering : de l'extraction de caractéristiques au regroupement et à l'évaluation.

4.2.3 Résultats expérimentaux

Qualité du clustering

Les résultats montrent des scores de Silhouette très satisfaisants :

— **Minibatch k-means :** 0.8697

— **Agglomerative clustering :** 0.8664

Ces scores (supérieurs à 0.8) témoignent d'une bonne séparation des clusters et d'un regroupement cohérent des images similaires. La capacité d'umap à préserver à la fois les relations locales et globales dans les données a significativement contribué à ces résultats.

Évaluation supervisée

Afin d'évaluer la qualité des représentations latentes issues du pipeline non supervisé, une validation supervisée a été réalisée en utilisant un classifieur entraîné sur les embeddings extraits. Cette étape permet de mesurer dans quelle mesure les descripteurs générés par le modèle permettent de séparer efficacement les classes.

Le tableau 4.1 présente les scores obtenus pour chaque classe, incluant la précision, le rappel, le F1-score et le nombre d'échantillons par classe. La précision globale du modèle atteint environ **92 %**, témoignant de la pertinence des représentations latentes extraites.

Les classes telles que *clothes*, *shoes* et *battery* obtiennent des scores quasi parfaits, illustrant une bonne capacité de séparation. En revanche, les classes *trash* et *plastic* présentent des performances plus modestes, probablement dues à leur similarité visuelle et à un déséquilibre du nombre d'échantillons dans l'ensemble d'entraînement.

Classe	**Précision**	**Recall**	**F1-Score**	**Support**
Battery	0,94	0,97	0,95	30
Biological	1,00	0,97	0,98	33
Cardboard	0,98	0,92	0,95	52
Clothes	0,99	0,99	0,99	167
Glass	0,92	0,92	0,92	76
Metal	0,87	0,96	0,92	28
Paper	0,89	0,94	0,91	50
Plastic	0,93	0,86	0,89	59
Shoes	0,96	0,99	0,97	67
Trash	0,90	0,82	0,86	22

TABLE 4.1 – Rapport de classification pour les différentes classes de déchets.

La figure 4.2 présente la matrice de confusion associée à cette classification. Elle permet d'identifier visuellement les confusions entre classes, notamment les erreurs de prédiction entre *trash* et *plastic*, ou encore entre *glass* et *metal*, qui peuvent s'expliquer par une forte ressemblance visuelle dans certaines images.

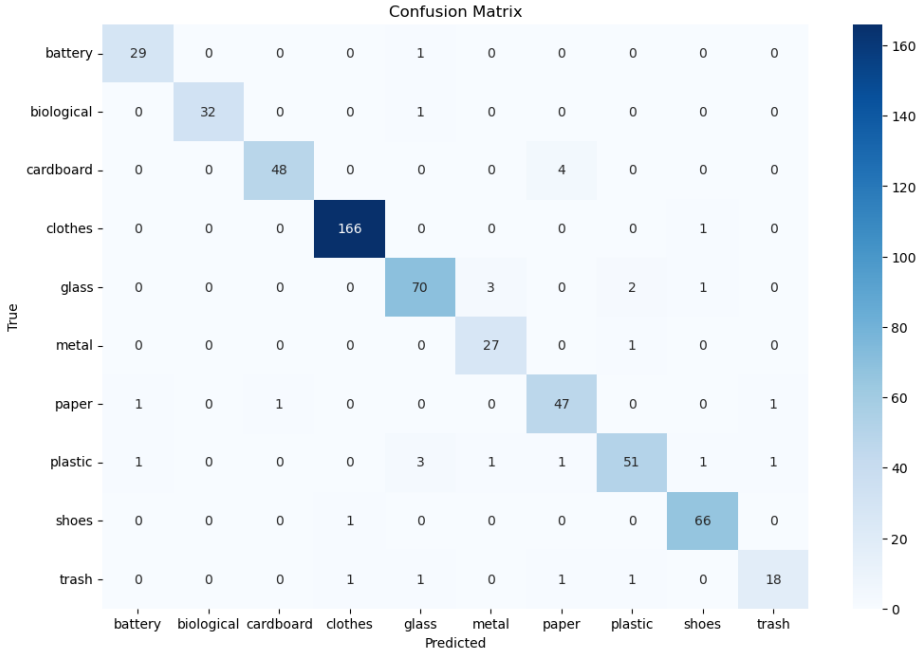

FIGURE 4.2 – Matrice de confusion illustrant les performances du classifieur sur les représentations extraites.

4.2.4 Analyse comparative avec l'approche de Caron *et al.* [4]

— **Clustering :** Contrairement à Caron *et al.* qui se limitent à k-means, notre modèle adopte minibatch k-means (pour sa faible empreinte mémoire) et agglomerative clustering, ce qui permet une meilleure adaptabilité aux grands jeux de données.

— **Réduction dimensionnelle :** umap est intégré pour capter des relations non linéaires, là où l'article de Caron ne spécifie pas explicitement de réduction dimensionnelle.

— **Évaluation :** L'utilisation du silhouette score dans notre modèle renforce l'ob-

jectivité de l'évaluation. Caron ne propose pas de métrique quantitative directe pour la séparation des clusters.

— **Domaine d'application :** Tandis que l'approche de Caron est appliquée à des données génériques (ImageNet), notre modèle est dédié à un cas concret : la classification automatique de déchets, avec des résultats directement transférables en contexte industriel.

4.2.5 Points forts et limitations par classe

Forces : Les classes *clothes*, *shoes*, et *battery* bénéficient d'une excellente séparation dans l'espace des caractéristiques, ce qui se traduit par des performances quasi parfaites.

Limites : Les catégories *trash* et *plastic* posent davantage de problèmes, en raison de similitudes visuelles ou de sous-représentation. Cela suggère la nécessité de renforcer ces classes via un rééquilibrage ou une augmentation des données.

4.2.6 Pourquoi notre algorithme classe bien les images ?

— **Modèle de base robuste :** ResNet-50 pré-entraîné fournit des descripteurs discriminants efficaces.

— **Réduction dimensionnelle adaptée :** umap facilite la capture des structures complexes de données.

— **Double approche de clustering :** La complémentarité entre minibatch k-means et Agglomerative clustering maximise la cohérence des regroupements.

— **Évaluation rigoureuse :** L'analyse croisée entre clustering et classification supervisée renforce la validité des résultats.

4.2.7 Pistes d'amélioration

— **Amélioration des classes difficiles :** Équilibrer le dataset et introduire des transformations ciblées (e.g. rotations, luminosité).
— **Diversification de l'approche :** Comparer umap à d'autres techniques (e.g. PCA), ou expérimenter des approches de type DEC ou deepcluster.
— **Stratégies hybrides :** Combiner les résultats de plusieurs algorithmes de clustering via une approche d'ensemble (fusion de clusters).

4.2.8 Comparaison avec l'apprentissage supervisé

Dans cette section, nous proposons une comparaison synthétique entre l'approche par *deep clustering* et l'apprentissage supervisé classique. Bien que l'apprentissage supervisé soit historiquement dominant pour les tâches de classification, le deep clustering présente une alternative robuste, notamment dans les contextes où les données annotées sont rares ou inexistantes. Le tableau 4.2 met en évidence les principales différences entre ces deux paradigmes selon plusieurs critères clés : nécessité d'annotations, performance, innovation méthodologique, domaines d'application et flexibilité de mise en œuvre.

Comme on peut le constater, bien que l'apprentissage supervisé reste la référence en matière de précision, le *deep clustering* constitue une alternative crédible et prometteuse, notamment dans des scénarios où l'annotation humaine est difficile, coûteuse ou imprécise. Cette approche ouvre ainsi la voie à de nouvelles perspectives dans des domaines où les données brutes sont abondantes mais non étiquetées.

Critères	**Deep clustering**	**Apprentissage supervisé**
Annotation des données	Non requise	Requise
Performance	Haute (92%) sans supervision explicite	Très haute (95–98%)
Innovation	Apprentissage non supervisé combiné à des réseaux profonds	Modèles bien établis et optimisés
Application	Cas à étiquettes rares (ex. tri de déchets)	Cas bien étiquetés (ex. vision médicale, finance)
Mise en œuvre	Plus flexible, moins dépendante des données annotées	Plus exigeante sur la qualité des annotations

TABLE 4.2 – Comparaison entre deep clustering et apprentissage supervisé

Conclusion

Le deep clustering se distingue comme une approche efficace et prometteuse pour la classification d'images non étiquetées. Notre modèle démontre des performances compétitives, avec un silhouette score supérieur à 0.86 et une précision globale avoisinant les **92%**. Comparé aux méthodes supervisées, il présente l'avantage de s'affranchir de l'annotation manuelle, ce qui le rend particulièrement pertinent pour des domaines où les ressources sont limitées. Pour aller plus loin, l'exploration d'un cadre semi-supervisé pourrait permettre de tirer parti des points forts des deux paradigmes, alliant la robustesse des représentations non supervisées à la précision offerte par l'apprentissage supervisé.

Chapitre 5

Conclusion et Perspectives

Cette étude a mis en avant le potentiel du deep clustering pour la classification automatique des déchets. En combinant les réseaux de neurones profonds avec des algorithmes de clustering non supervisés tels que minibatch k-means et agglomerative clustering, nous avons démontré que des performances élevées pouvaient être atteintes en termes de silhouette scores, avec des valeurs de 0.8697 pour minibatch k-means et de 0.8664 pour agglomerative clustering. Ces résultats témoignent de la qualité des clusters formés et de la robustesse du modèle dans la reconnaissance de différents types de déchets.

Une comparaison avec les travaux de Caron et al. dans leur article intitulé *Deep Clustering for Unsupervised Learning of Visual Features* [4] montre que notre approche adopte des techniques similaires en exploitant des représentations latentes apprises pour améliorer le clustering. Cependant, contrairement à l'approche de Caron et al., qui s'appuie principalement sur un cadre de classification d'images non supervisé appliqué à des données générales comme ImageNet, notre étude se concentre sur une application spécifique à la gestion des déchets. De plus, l'intégration de deux algorithmes de clustering différents dans notre modèle permet une évaluation comparative

approfondie, ce qui constitue une valeur ajoutée par rapport à leurs travaux.

Les recherches de Caron et al. ont également démontré l'efficacité de l'apprentissage joint entre la formation de clusters et l'extraction de caractéristiques. Notre méthodologie s'inspire de cette idée en combinant un réseau ResNet50 pré-entraîné pour extraire des caractéristiques robustes avec un processus de clustering appliqué à ces caractéristiques. Cette stratégie a permis d'atteindre une classification précise des déchets, même en l'absence d'annotations exhaustives.

En outre, une comparaison détaillée entre le deep clustering et les algorithmes d'apprentissage supervisé a été réalisée. Cette analyse a mis en évidence les avantages et les inconvénients respectifs des deux approches. Alors que les modèles supervisés, tels que ResNet ou Xception, offrent des précisions légèrement supérieures grâce à l'utilisation de données annotées, le deep clustering se distingue par sa capacité à travailler efficacement avec des données non étiquetées, une caractéristique essentielle dans des contextes où l'annotation est coûteuse ou impraticable. Cette comparaison renforce la pertinence de notre approche dans le domaine de la gestion des déchets.

En outre, notre application de techniques d'augmentation des données et d'évaluation détaillée (matrices de confusion, rapports de classification, et visualisations des clusters) a permis de confirmer que le modèle est bien adapté à des scénarios réels où la diversité et l'imprécision des données sont des défis majeurs.

5.1 Perspectives

Les résultats obtenus ouvrent de nombreuses perspectives pour l'amélioration et l'extension de cette recherche. Parmi les pistes prometteuses :

- **Amélioration de l'architecture du modèle :** L'exploration de modèles plus avancés comme Vision Transformers (ViT) ou EfficientNet pourrait améliorer encore la qualité des caractéristiques extraites et, par conséquent, des clusters formés.
- **Apprentissage semi-supervisé :** Introduire une petite quantité de données annotées pour guider le processus de clustering pourrait augmenter la précision du modèle et réduire les erreurs liées aux classes sous-représentées.
- **Applications à grande échelle :** Tester le modèle sur des ensembles de données plus larges ou provenant de différentes sources permettrait d'évaluer sa généralisabilité et de l'adapter à des contextes industriels variés.
- **Intégration de techniques basées sur la blockchain :** Garantir la traçabilité et la sécurité des données de classification pourrait être un avantage pour des systèmes déployés dans des environnements complexes.
- **Optimisation énergétique :** Réduire la complexité computationnelle du modèle serait bénéfique pour son déploiement sur des dispositifs embarqués, tels que des bacs à déchets intelligents équipés de capteurs.

Enfin, les recherches futures pourraient également s'intéresser à l'intégration de modèles de jumeaux numériques pour la surveillance en temps réel des systèmes de classification, ainsi qu'à la création de bases de données plus diversifiées pour couvrir un éventail plus large de types de déchets. En combinant ces avancées, il sera possible de maximiser l'impact de cette technologie dans le domaine de la gestion des déchets et au-delà.

Bibliographie

[1] D. Arthur and S. Vassilvitskii. K-means++ : The advantages of careful seeding. In Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pages 1027–1035, 2007.

[2] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, New York, 2006.

[3] T. Calinski and J. Harabasz. *A Dendrite Method for Cluster Analysis*. Communications in Statistics - Theory and Methods, 3(1) :1–27, 1974.

[4] M. Caron, P. Bojanowski, A. Joulin, and M. Douze. *Deep Clustering for Unsupervised Learning of Visual Features*, 2018. *arXiv preprint arXiv :1807.05520*.

[5] M. Charrad, N. Ghazzali, V. Boiteau, and A. Niknafs. Nbclust : An examination of indices for determining the number of clusters. Journal of Statistical Software, 61(6) :1–36, 2014.

[6] D. L. Davies and D. W. Bouldin. *A Cluster Separation Measure*. IEEE Transactions on Pattern Analysis and Machine Intelligence, 1(2) :224–227, 1979.

[7] E. Diday and J.-C. Simon. *Clustering Analysis*, 1976.

[8] R. O. Duda, P. E. Hart, and D. G. Stork. Pattern Classification. *Wiley, New York*, 2012.

[9] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*, 2016.

[10] X. Guo, X. Liu, E. Zhu, and J. Yin. *Deep Clustering with Convolutional Autoencoders*, 2017.

[11] T. Hastie, R. Tibshirani, and J. Friedman. The Elements of Statistical Learning. *Springer, New York*, 2009.

[12] A. K. Jain. *Data Clustering : 50 Years Beyond K-Means*. Pattern Recognition Letters, 31(8) :651–666, 2010.

[13] Z. Jiang, Y. Zheng, H. Tan, B. Tang, and H. Zhou. *Variational Deep Embedding : An Unsupervised and Generative Approach to Clustering*. arXiv preprint arXiv :1611.05148, 2017.

[14] S. C. Johnson. *Hierarchical Clustering Schemes*. Psychometrika, 32(3) :241–254, 1967.

[15] J. H. Ward Jr. *Hierarchical Grouping to Optimize an Objective Function*. Journal of the American Statistical Association, 58(301) :236–244, 1963.

[16] L. Kaufman and P. J. Rousseeuw. Finding Groups in Data : An Introduction to Cluster Analysis. Wiley, New York, 1990.

[17] S. Kaza, L. C. Yao, P. Bhada-Tata, and F. Van Woerden. What a Waste 2.0 : A Global Snapshot of Solid Waste Management to 2050. *World Bank Publications, Washington, DC*, 2018.

[18] Keras. *Keras Documentation*, 2020.

[19] M. Khichane. *Le Machine Learning avec Python*, 2021.

[20] D. P. Kingma and M. Welling. *Auto-Encoding Variational Bayes*. 2014.

[21] X. Li, J. Wang, and M. Zhao. *Convolutional Neural Networks for Garbage Image Classification*, 2017.

[22] J. MacQueen. *Some Methods for Classification and Analysis of Multivariate Observations*. In Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, volume 1, pages 281–297, *Berkeley, California*, 1967. *University of California Press*.

[23] C. Makina. Deep clustering avec réseaux de neurones : concepts et visualisations. `https://makina-corpus.com/blog/metier/ia/introduction-reseaux-de-neurones-profonds`, 2022. Consulté en mai 2025.

[24] L. McInnes, J. Healy, and J. Melville. *UMAP : Uniform Manifold Approximation and Projection for Dimension Reduction*, 2018. arXiv preprint arXiv :1802.03426.

[25] U. Michelucci. *Applied Deep Learning*, 2018.

[26] E. Min, X. Guo, Q. Liu, G. Zhang, J. Cui, and J. Long. *A Survey of Clustering with Deep Learning : From the Perspective of Network Architecture*, 2018.

[27] Mohit. *Image Processing Using CNN : A Beginner's Guide*. `https://www.analyticsvidhya.com/blog/2021/06/image-processing-using-cnn-a-beginners-guide/`, 2021. *Consulté le 19 mai 2025*.

[28] F. Murtagh and P. Contreras. *Algorithms for Hierarchical Clustering : An Overview*. WIREs Data Mining and Knowledge Discovery, 2(1) :86–97, 2012.

[29] N. Nguyen, T. Tran, and P. Le. *A Deep Learning Model for Waste Classification*, 2019.

[30] M. Parizeau. *Introduction aux Réseaux de Neurones*, 2009.

[31] A. Razavi, A. van den Oord, and O. Vinyals. *Generating Diverse High-Fidelity Images with VQ-VAE-2*. Advances in Neural Information Processing Systems, 32, 2019.

[32] D. J. Rezende, S. Mohamed, and D. Wierstra. *Stochastic Backpropagation and Approximate Inference in Deep Generative Models*. In Proceedings of the 31st International Conference on Machine Learning (ICML), pages 1278–1286, 2014.

[33] L. Rokach and O. Maimon. *Clustering Methods*. In Data Mining and Knowledge Discovery Handbook, pages 321–352. Springer, 2005.

[34] P. J. Rousseeuw. *Silhouettes : A Graphical Aid to the Interpretation and Validation of Cluster Analysis*. Journal of Computational and Applied Mathematics, 20 :53–65, 1987.

[35] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. *Learning Representations by Back-Propagating Errors*, 1986.

[36] G. Saint-Cirgue. *Apprendre le Machine Learning en une Semaine*, 2019.

[37] Scikit-learn. *Scikit-learn Documentation*, 2020.

[38] D. Sculley. *Web-Scale K-Means Clustering*. In Proceedings of the 19th International Conference on World Wide Web, pages 1177–1178, 2010.

[39] P. H. A. Sneath and R. R. Sokal. Numerical Taxonomy : The Principles and Practice of Numerical Classification. *W.H. Freeman, San Francisco*, 1973.

[40] TensorFlow. *TensorFlow Documentation*, 2016.

[41] R. Tibshirani, G. Walther, and T. Hastie. *Estimating the Number of Clusters in a Dataset via the Gap Statistic*. Journal of the Royal Statistical Society : Series B (Statistical Methodology), 63(2) :411–423, 2001.

[42] M. Walesiak and A. Dudek. *ClusterSim : Searching for Optimal Clustering Pro-*

cedure for a Given Dataset. Journal of Statistical Software, 41 :1–29, 2011.

[43] J. Xie, R. Girshick, and A. Farhadi. *Unsupervised Deep Embedding for Clustering Analysis*, 2016.

[44] R. Xu and D. Wunsch. *Survey of Clustering Algorithms*. IEEE Transactions on Neural Networks, 16(3) :645–678, 2005.

[45] B. Yang, X. Fu, N. D. Sidiropoulos, and M. Hong. *Towards K-Means-Friendly Spaces : Simultaneous Deep Learning and Clustering*, 2017.

Annexe A

Codes source en Python des diagrammes

```python
import numpy as np
from scipy.spatial.distance import cdist

# Definition des clusters
A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6], [7, 8]])

# Calcul des distances euclidiennes
distances = cdist(A, B, metric='euclidean')

# Affichage de la matrice des distances
print("Matrice des distances :")
print(distances)

# Calcul du saut minimum et maximum
saut_minimum = np.min(distances)
saut_maximum = np.max(distances)

print("\nSaut minimum :", saut_minimum)
```

```
print("Saut maximum :", saut_maximum)
```

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage
from sklearn.datasets import make_blobs

# 1. Génération des données
# Générer des données réalistes de clients
# Nous utilisons make_blobs pour créer un jeu de données avec 3 clusters
data, _ = make_blobs(n_samples=100, centers=3, cluster_std=1.0,
    random_state=42)

# Affichage des données générées
plt.figure(figsize=(10, 7))
plt.scatter(data[:, 0], data[:, 1], s=50)
plt.title('Données Générées (Revenu Annuel vs. Score de Dépenses)')
plt.xlabel('Revenu Annuel')
plt.ylabel('Score de Dépenses')
plt.savefig('data_generated.png')
plt.show()

# 2. Classification hiérarchique ascendante
Z = linkage(data, method='ward')

# 3. Visualisation du dendrogramme
plt.figure(figsize=(10, 7))
dendrogram(Z)
plt.title('Dendrogramme pour une Classification Hiérarchique Ascendante'
    )
plt.xlabel('Index des Points')
plt.ylabel('Distance (Ward)')
plt.savefig('dendrogram.png')
plt.show()

```

```
# 4. Interprétation du dendrogramme
# Découper le dendrogramme pour obtenir des clusters
from scipy.cluster.hierarchy import fcluster

# Nous choisissons une hauteur pour couper le dendrogramme qui nous donne 3 clusters
max_d = 7.5  # Cette valeur dépend de l'échelle de vos données et de vos besoins
clusters = fcluster(Z, max_d, criterion='distance')

# Affichage des données avec les clusters obtenus
plt.figure(figsize=(10, 7))
plt.scatter(data[:, 0], data[:, 1], c=clusters, cmap='prism', s=50)
plt.title('Données Segmentées en Clusters')
plt.xlabel('Revenu Annuel')
plt.ylabel('Score de Dépenses')
plt.savefig('clusters_segmented.png')
plt.show()
```

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import MiniBatchKMeans

# Génération des données
data, _ = make_blobs(n_samples=300, centers=4, cluster_std=0.60, random_state=0)

# Application de l'algorithme MiniBatch K-means
minibatch_kmeans = MiniBatchKMeans(n_clusters=4, batch_size=50, random_state=0)
minibatch_kmeans.fit(data)
y_minibatch = minibatch_kmeans.predict(data)

# Affichage des résultats
```

```
plt.figure(figsize=(10, 7))
plt.scatter(data[:, 0], data[:, 1], c=y_minibatch, s=50, cmap='viridis')
centers = minibatch_kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='red', s=200, alpha=0.75,
    marker='X')
plt.title('Clusters et leurs centres (MiniBatch K-means)')
plt.xlabel('Revenu Annuel')
plt.ylabel('Score de Dépenses')
plt.savefig('minibatch_kmeans_clusters.png')
plt.show()
```

```
import umap
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_digits
from sklearn.preprocessing import StandardScaler

# 1. Charger le jeu de données MNIST (chiffres de 0 à 9)
digits = load_digits()
X = digits.data
y = digits.target

# 2. Standardisation des données (très recommandé avec UMAP)
X_scaled = StandardScaler().fit_transform(X)

# 3. Application de UMAP pour réduire à 2 dimensions
reducer = umap.UMAP(n_neighbors=15, min_dist=0.1, metric='euclidean',
    random_state=42)
X_umap = reducer.fit_transform(X_scaled)

# 4. Affichage des données réduites en 2D + sauvegarde
plt.figure(figsize=(10, 7))
scatter = plt.scatter(X_umap[:, 0], X_umap[:, 1], c=y, cmap='Spectral',
    s=10)
plt.colorbar(scatter, label='Classe (chiffre)')
```

```
plt.title('Projection UMAP des chiffres manuscrits (MNIST)')
plt.xlabel('UMAP 1')
plt.ylabel('UMAP 2')
plt.grid(True)
plt.tight_layout()
# Sauvegarde de l'image
plt.savefig("umap_digits_projection.png", dpi=300)  # ou .pdf, .svg, etc.
plt.show()
```

```
# Structure d'un autoencodeur
import matplotlib.pyplot as plt
import matplotlib.patches as patches

fig, ax = plt.subplots(figsize=(10, 5))

# Drawing encoder
encoder = patches.Rectangle((0.1, 0.1), 0.3, 0.8, edgecolor='blue', facecolor='none', lw=2)
ax.add_patch(encoder)
ax.text(0.25, 0.9, 'Encodeur', fontsize=12, ha='center', va='center')

# Drawing latent space
latent_space = patches.Rectangle((0.45, 0.4), 0.1, 0.2, edgecolor='green', facecolor='none', lw=2)
ax.add_patch(latent_space)
ax.text(0.5, 0.55, 'Espace Latent', fontsize=12, ha='center', va='center')

# Drawing decoder
decoder = patches.Rectangle((0.6, 0.1), 0.3, 0.8, edgecolor='red', facecolor='none', lw=2)
ax.add_patch(decoder)
ax.text(0.75, 0.9, 'Décodeur', fontsize=12, ha='center', va='center')

```

```
22 # Arrows
23 ax.annotate('', xy=(0.4, 0.5), xytext=(0.1, 0.5), arrowprops=dict(
       arrowstyle="->", lw=2))
24 ax.annotate('', xy=(0.6, 0.5), xytext=(0.55, 0.5), arrowprops=dict(
       arrowstyle="->", lw=2))
25
26 # Input and output texts
27 ax.text(0.05, 0.5, 'Données d\'entrée', fontsize=12, ha='center', va='
       center')
28 ax.text(0.95, 0.5, 'Données de sortie', fontsize=12, ha='center', va='
       center')
29
30 ax.set_xlim(0, 1)
31 ax.set_ylim(0, 1)
32 ax.axis('off')
33
34 plt.savefig('autoencoder_structure.png')
35 plt.show()
```

Annexe B

Codes source en Python de nos résultats

```python
# Re-execute the pipeline drawing code after state reset

import matplotlib.pyplot as plt
from matplotlib.patches import FancyBboxPatch, FancyArrowPatch

# Create figure and axis
fig, ax = plt.subplots(figsize=(14, 7))
ax.set_xlim(0, 22)
ax.set_ylim(0, 10)
ax.axis('off')

# Define color palette
colors = {
    "input": "#95a5a6",
    "encoder": "#34495e",
    "features": "#1abc9c",
    "clustering": "#2980b9",
    "loss": "#e67e22",
    "output": "#7f8c8d"
```

```
}

# Function to draw a box
def add_box(ax, x, y, text, color, width=3.8, height=1.8, fontsize=10):
    box = FancyBboxPatch((x, y), width, height, boxstyle="round,pad=0.03",
                         linewidth=1.8, edgecolor="black", facecolor=color)
    ax.add_patch(box)
    ax.text(x + width / 2, y + height / 2, text, ha='center', va='center',
            fontsize=fontsize, color='white', weight='bold')

# Function to draw arrows
def add_arrow(ax, x1, y1, x2, y2, rad=0.0):
    arrow = FancyArrowPatch((x1, y1), (x2, y2), connectionstyle=f"arc3,rad={rad}",
                            arrowstyle='->', mutation_scale=15, linewidth=2, color='black')
    ax.add_patch(arrow)

# Draw boxes
add_box(ax, 1, 4, "Input Image", colors["input"])
add_box(ax, 5, 4, "ResNet-50\nEncoder", colors["encoder"])
add_box(ax, 9, 4, "Deep Feature\nRepresentation", colors["features"])
add_box(ax, 13, 7, "UMAP", colors["clustering"])
add_box(ax, 13, 4, "MiniBatch\nKMeans", colors["clustering"])
add_box(ax, 13, 1, "Agglomerative\nClustering", colors["clustering"])
add_box(ax, 17, 4, "Cluster\nAssignments", colors["output"])
add_box(ax, 9, 7.5, "Unsupervised\nLoss Function", colors["loss"], width=4.2, height=1.4, fontsize=9)

# Draw arrows
add_arrow(ax, 4.8, 5, 5, 5)             # Input to encoder
add_arrow(ax, 8.8, 5, 9, 5)             # Encoder to features
add_arrow(ax, 12.8, 5, 13, 5)           # Features to KMeans
```

```
add_arrow(ax, 12.8, 8, 13, 8)           # Features to UMAP
add_arrow(ax, 12.8, 2, 13, 2)           # Features to Agglomerative
add_arrow(ax, 16.8, 5, 17, 5)           # Clustering to Output
add_arrow(ax, 10.2, 5.6, 11, 7.2)       # Features to Loss
add_arrow(ax, 14.9, 2.9, 17, 4.6, rad=0.3)  # Agglomerative to Output

# Title
plt.title("Pipeline du Modèle de Deep Clustering", fontsize=15,
    fontweight='bold')
plt.tight_layout()

# Save and show
pipeline_path = "/mnt/data/pipeline_code_deep_clustering.png"
plt.savefig(pipeline_path, dpi=300)
plt.show()

pipeline_path
```

```

# Importation des bibliothèques standard
import os  # Pour interagir avec le système de fichiers
import torch  # PyTorch pour les calculs tensoriels et l'apprentissage
    automatique
import torch.nn as nn  # Modules pour construire des réseaux de neurones
import torch.optim as optim  # Algorithmes d'optimisation pour l'entraî
    nement
from torch.utils.data import DataLoader, random_split  # Gestion des
    ensembles de données
from torchvision import transforms, datasets, models  # Transformations,
     ensembles de données et modèles pré-entraînés
from sklearn.metrics import confusion_matrix, classification_report,
    silhouette_score  # Outils d'évaluation
from umap import UMAP  # Réduction dimensionnelle
from sklearn.cluster import MiniBatchKMeans, AgglomerativeClustering  #
    Algorithmes de clustering
```

```
12 import matplotlib.pyplot as plt  # Visualisation
13 import seaborn as sns  # Visualisation avancée
14 from tqdm import tqdm  # Barre de progression pour les boucles longues
15
16 # Configuration des paramètres du modèle et des données
17 data_path = r'C:\Users\Mamadou\Documents\Test\validation'  # Chemin vers
       le dossier contenant les données
18 batch_size = 64  # Taille des lots pour l'entraînement
19 epochs = 40  # Nombre maximal d'époques pour l'entraînement
20 feat_dim = 300  # Dimensions des caractéristiques latentes extraites
21 num_clusters = 10  # Nombre de clusters pour le clustering
22 learning_rate = 3e-5  # Taux d'apprentissage pour l'optimiseur
23 patience = 5  # Nombre d'époques avant déclenchement du "early stopping"
24
25 # Préparation des données avec des transformations
26 transform = transforms.Compose([
27     transforms.Resize((224, 224)),  # Redimensionne les images à 224x224
          pixels
28     transforms.RandomHorizontalFlip(),  # Applique un retournement
         horizontal aléatoire
29     transforms.RandomRotation(15),  # Applique une rotation aléatoire
30     transforms.ColorJitter(brightness=0.4, contrast=0.4, saturation=0.4,
          hue=0.2),  # Ajoute des variations de couleur
31     transforms.RandomAffine(degrees=10),  # Applique des transformations
          géométriques aléatoires
32     transforms.ToTensor(),  # Convertit les images en tenseurs PyTorch
33     transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224,
         0.225])  # Normalise les valeurs des pixels
34 ])
35
36 # Chargement des données avec ImageFolder et découpage en ensembles
37 dataset = datasets.ImageFolder(root=data_path, transform=transform)
38 train_size = int(0.7 * len(dataset))  # 70% pour l'entraînement
39 val_size = int(0.15 * len(dataset))  # 15% pour la validation
40 test_size = len(dataset) - train_size - val_size  # 15% pour le test
41 train_data, val_data, test_data = random_split(dataset, [train_size,
```

```
    val_size, test_size])  # Découpe les données
42
43 # Création des DataLoaders pour gérer les lots
44 train_loader = DataLoader(train_data, batch_size=batch_size, shuffle=
    True)
45 val_loader = DataLoader(val_data, batch_size=batch_size, shuffle=False)
46 test_loader = DataLoader(test_data, batch_size=batch_size, shuffle=False
    )
47
48 # Définition du modèle avec ResNet-50
49 class CombinedResNet(nn.Module):
50     def __init__(self, num_classes, latent_dim=feat_dim):
51         super(CombinedResNet, self).__init__()
52         self.encoder = models.resnet50(weights=models.ResNet50_Weights.
            DEFAULT)  # Modèle ResNet pré-entraîné
53         num_ftrs = self.encoder.fc.in_features  # Nombre de caracté
            ristiques de sortie de ResNet
54         self.encoder.fc = nn.Sequential(  # Remplacement de la couche FC
             pour extraire des caractéristiques latentes
55             nn.Linear(num_ftrs, latent_dim),  # Réduction dimensionnelle
56             nn.BatchNorm1d(latent_dim),  # Normalisation pour stabiliser
                 l'entraînement
57             nn.ReLU(inplace=True)  # Activation non linéaire
58         )
59         self.classifier = nn.Sequential(  # Classifieur final
60             nn.Dropout(0.5),  # Dropout pour éviter le surapprentissage
61             nn.Linear(latent_dim, num_classes)  # Couche de sortie pour
                les prédictions
62         )
63
64     def forward(self, x):
65         features = self.encoder(x)  # Extraction des caractéristiques
            avec ResNet
66         classification_output = self.classifier(features)  # Prédictions
67         return features, classification_output  # Retourne les caracté
            ristiques latentes et les prédictions
```

```
68
69 # Initialisation du modèle, de la fonction de perte et des optimisateurs
70 model = CombinedResNet(num_classes=len(dataset.classes)).cuda()  # Modèle sur GPU
71 criterion = nn.CrossEntropyLoss()  # Fonction de perte pour classification multi-classes
72 optimizer = optim.Adam(model.parameters(), lr=learning_rate, weight_decay=1e-4)  # Optimiseur Adam avec régularisation L2
73 scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, 'min', patience=patience // 2, factor=0.5)  # Réduction du LR
74
75 # Fonction d'entraînement du modèle
76 def train_model(model, train_loader, val_loader, criterion, optimizer, scheduler, num_epochs, patience):
77     train_losses, val_losses = [], []  # Suivi des pertes
78     train_accuracies, val_accuracies = [], []  # Suivi des précisions
79     best_val_loss = float('inf')  # Initialisation de la meilleure perte de validation
80     patience_counter = 0  # Compteur pour le "early stopping"
81
82     for epoch in range(num_epochs):
83         model.train()  # Passe en mode entraînement
84         running_loss, correct_train, total_train = 0.0, 0, 0
85
86         # Entraînement sur les données
87         for inputs, labels in tqdm(train_loader, desc=f'Epoch [{epoch+1}/{num_epochs}] - Training'):
88             inputs, labels = inputs.cuda(), labels.cuda()  # Déplacement sur GPU
89             optimizer.zero_grad()  # Réinitialisation des gradients
90             features, outputs = model(inputs)  # Passage avant (forward pass)
91             loss = criterion(outputs, labels)  # Calcul de la perte
92             loss.backward()  # Rétropropagation
93             optimizer.step()  # Mise à jour des poids
94
```

```
95              running_loss += loss.item() * inputs.size(0)  # Accumulation
                    de la perte
96              _, predicted = torch.max(outputs, 1)  # Prédictions
97              total_train += labels.size(0)  # Comptage total des é
                    chantillons
98              correct_train += (predicted == labels).sum().item()  #
                    Comptage des bonnes prédictions
99
100         epoch_loss = running_loss / len(train_loader.dataset)  # Perte
                moyenne
101         train_losses.append(epoch_loss)  # Enregistrement des pertes
102         train_accuracy = 100 * correct_train / total_train  # Précision
                moyenne
103         train_accuracies.append(train_accuracy)
104
105         model.eval()  # Passe en mode évaluation
106         val_running_loss, correct_val, total_val = 0.0, 0, 0
107         with torch.no_grad():  # Désactivation du calcul des gradients
108             for inputs, labels in val_loader:
109                 inputs, labels = inputs.cuda(), labels.cuda()
110                 _, outputs = model(inputs)
111                 loss = criterion(outputs, labels)
112                 val_running_loss += loss.item() * inputs.size(0)
113                 _, predicted = torch.max(outputs, 1)
114                 total_val += labels.size(0)
115                 correct_val += (predicted == labels).sum().item()
116
117         val_loss = val_running_loss / len(val_loader.dataset)  # Perte
                de validation moyenne
118         val_losses.append(val_loss)
119         val_accuracy = 100 * correct_val / total_val  # Précision
                moyenne
120         val_accuracies.append(val_accuracy)
121         scheduler.step(val_loss)  # Ajustement du taux d'apprentissage
122
123         print(f'Epoch [{epoch+1}/{num_epochs}], Train Loss: {epoch_loss
```

```
            :.4f}, Train Accuracy: {train_accuracy:.2f}%, '
124             f'Validation Loss: {val_loss:.4f}, Validation Accuracy: {
                    val_accuracy:.2f}%')
125
126         # Gestion du "early stopping"
127         if val_loss < best_val_loss:
128             best_val_loss = val_loss
129             patience_counter = 0
130         else:
131             patience_counter += 1
132             if patience_counter >= patience:
133                 print("Early stopping triggered.")
134                 break
135
136     return train_losses, val_losses, train_accuracies, val_accuracies
```