UNIVERSITÉ DU LITTORAL CÔTE D'OPALE

ÉCOLE D'INGÉNIEURS DU LITTORAL

Master 2 : Ingénierie des Systèmes Complexes

Année Universitaire 2025-2026

RAPPORT DE PROJET

Réduction de dimension guidée par Deep Learning pour l'accélération de l'apprentissage supervisé

Réalisé par :
MENNIOUI Soufiane

Encadré par :
Prof. JBILOU KHALIDE

15 janvier 2026

Table des matières

Table des figures

Introduction Générale

1. Contexte : L'Ère du Big Data et l'Explosion des Données Visuelles

Nous vivons une révolution numérique caractérisée par le « **Big Data** ». D'après les estimations récentes, le volume mondial de données double environ tous les deux ans. Au sein de cet océan d'informations, les données visuelles (images) occupent une place prépondérante, qu'il s'agisse d'imagerie médicale ou de flux vidéo.

Cependant, ces données sont intrinsèquement volumineuses et non structurées. Une simple image de résolution modeste est constituée de milliers de pixels (par exemple 784 pixels pour une image MNIST), ce qui rend leur traitement et leur stockage coûteux en ressources. Pour exploiter cette richesse informationnelle, le *Deep Learning* est devenu incontournable. Si les réseaux complexes comme les Réseaux Convolutifs (CNN) constituent l'état de l'art pour les images complexes, ils restent très lourds. Il devient donc crucial, pour des architectures plus légères, de trouver des moyens de compresser cette information.

2. Problématique : Le Paradoxe de la Haute Dimensionnalité

L'augmentation de la dimension des entrées (le nombre de pixels) entraîne ce que l'on appelle le « **Fléau de la Dimensionnalité** ».

— Plus la dimension est élevée, plus le coût de calcul explose.

— Plus l'espace est grand, plus les données sont éparses, augmentant le risque de surapprentissage (*overfitting*).

La problématique centrale de ce projet est donc : **Comment réduire drastiquement la dimension des données d'entrée pour accélérer l'apprentissage, sans pour autant sacrifier l'information sémantique nécessaire à la classification ?**

3. Hypothèse de Travail : La Compression Sémantique par Autoencodeurs

Nous postulons que les données visuelles, bien que vivant dans un espace de très haute dimension (l'espace des pixels), peuvent être résumées par un petit nombre de caractéristiques latentes. Pour cela, nous utiliserons l'architecture des **Autoencodeurs (AE)**. En forçant le réseau de neurones à compresser l'entrée à travers un goulot d'étranglement (*bottleneck*) puis à la reconstruire, nous obligeons le modèle à ne conserver que les caractéristiques les plus saillantes et discriminantes, éliminant le bruit et la redondance.

4. Objectifs du Projet

L'objectif global est de concevoir une chaîne de traitement hybride pour valider cette hypothèse. Ce but se décline en trois objectifs spécifiques :

1. **Étude Théorique :** Analyser le fléau de la dimensionnalité et comprendre le fonctionnement mathématique des réseaux de neurones (MLP) et des autoencodeurs.
2. **Implémentation d'un Pipeline Hybride :** Nous développerons une architecture en Python composée de :
   — Un **Autoencodeur Dense (MLP)** chargé de compresser les images brutes (dimension 784) vers un espace latent réduit (dimension 64).
   — Un **Classifieur (MLP)** prenant en entrée les vecteurs compressés pour prédire la classe de l'image.
3. **Validation sur MNIST :** Nous comparerons les performances (précision, vitesse et qualité visuelle) de cette approche compressée face à une approche classique entraînée sur les images brutes.

5. Annonce du Plan

Pour mener à bien cette étude, ce document s'articule autour de cinq chapitres :

— Le **Chapitre 1** dresse l'état de l'art sur la dimensionnalité et les réseaux de neurones.
— Le **Chapitre 2** détaille la théorie mathématique des autoencodeurs.
— Le **Chapitre 3** présente la méthodologie et l'architecture système proposée.
— Le **Chapitre 4** décrit l'implémentation technique (code, outils).
— Le **Chapitre 5** analyse les résultats expérimentaux obtenus.

Chapitre 1

Contexte et État de l'Art

1.1 Introduction

L'apprentissage automatique (*Machine Learning*) a connu une évolution spectaculaire au cours de la dernière décennie, passant de modèles statistiques linéaires à des architectures de réseaux de neurones profonds (*Deep Learning*) capables d'approximer des fonctions hautement non-linéaires. Cependant, l'efficacité de ces algorithmes dépend intrinsèquement de la représentation des données. Dans le domaine de la vision par ordinateur, la complexité et la haute résolution des images soulèvent des défis majeurs.

Ce chapitre dresse un état de l'art des problématiques liées à la haute dimensionnalité. Nous analyserons d'abord le "fléau de la dimensionnalité", puis nous étudierons les limites des méthodes classiques de réduction de dimension (comme l'ACP). Enfin, nous détaillerons les fondements des réseaux de neurones artificiels, en mettant l'accent sur le Perceptron Multicouche (MLP), architecture centrale de notre étude, tout en le situant par rapport aux réseaux convolutifs (CNN).

1.2 Le Fléau de la Dimensionnalité (Curse of Dimensionality)

La notion de dimensionnalité fait référence au nombre de variables ou de caractéristiques (*features*) décrivant une donnée. Pour une image issue de la base MNIST (28×28 pixels), chaque image est un vecteur vivant dans un espace de dimension $D = 784$. Bien que cela puisse paraître gérable par rapport aux standards actuels, les propriétés géométriques des espaces de grande dimension sont contre-intuitives et posent des problèmes fondamentaux pour l'analyse de données.

1.2.1 Définition mathématique et géométrique

L'expression « fléau de la dimensionnalité », introduite par le mathématicien Richard Bellman en 1961, désigne l'augmentation explosive du volume de l'espace mathématique à mesure que l'on y ajoute des dimensions.

Pour illustrer ce phénomène, considérons un hypercube de dimension D avec un côté de longueur 1. Son volume total est $1^D = 1$. Si nous considérons une "coquille" extérieure d'épaisseur ϵ (par exemple $\epsilon = 0.001$), le volume de la partie intérieure est $(1 - 2\epsilon)^D$.

Lorsque la dimension D tend vers l'infini, ce volume intérieur tend vers 0 :

$$\lim_{D \to \infty} (1 - 2\epsilon)^D = 0 \tag{1.1}$$

Cela implique une conclusion géométrique surprenante : dans un espace de très haute dimension, la quasi-totalité du volume se concentre dans les coins et sur la surface extérieure de l'hypercube. L'espace est "creux" au centre.

FIGURE 1.1 – Illustration conceptuelle : à mesure que la dimension augmente, le volume se concentre aux frontières, rendant le centre de l'espace vide.

1.2.2 Problèmes induits pour l'apprentissage

Cette géométrie particulière engendre trois obstacles majeurs pour les algorithmes d'apprentissage supervisé :

— **La Sparsité (Rareté) des données :** Pour maintenir une densité de population constante dans l'espace de données (densité nécessaire pour une généralisation statistique fiable), le nombre d'échantillons d'entraînement N doit croître exponentiellement avec la dimension D. Avec un nombre fini d'images (ex : 60 000 pour MNIST), l'espace vectoriel à 784 dimensions est essentiellement vide. Les données sont trop dispersées pour qu'un modèle apprenne efficacement sans régularisation forte.

— **La perte de sens des distances (Distance Concentration) :** La distance euclidienne, utilisée par de nombreux algorithmes classiques (comme les k-Plus Proches Voisins ou les K-Means), perd sa capacité discriminante. Soit d_{max} la distance au point le plus éloigné et d_{min} la distance au point le plus proche dans un jeu de données aléatoire. On démontre mathématiquement que :

$$\lim_{D \to \infty} \frac{d_{max} - d_{min}}{d_{min}} \to 0 \tag{1.2}$$

Autrement dit, en très haute dimension, tous les points semblent être situés à peu près à la même distance les uns des autres. La notion de "voisinage" devient floue,

ce qui complique la tâche de classification basée sur la similitude.

— **Le Sur-apprentissage (Overfitting) :** Un modèle opérant sur des entrées de haute dimension nécessite une première couche de poids très large. Pour un réseau dense (MLP) prenant 784 entrées vers 512 neurones, cela génère $784 \times 512 \approx 400000$ paramètres rien que pour la première couche. Plus le nombre de paramètres est grand par rapport au nombre de données, plus la variance du modèle augmente, risquant de mémoriser le bruit plutôt que la structure.

1.2.3 L'hypothèse de la variété (Manifold Hypothesis)

Si l'espace de haute dimension est si vaste et vide, comment l'apprentissage profond parvient-il à fonctionner ? La réponse théorique réside dans l'hypothèse de la variété.

Cette théorie postule que les données du monde réel (comme les images de chiffres manuscrits) ne sont pas réparties uniformément dans tout l'espace $\mathbb{R}^D$. Elles se concentrent en réalité au voisinage d'une sous-structure topologique de dimension beaucoup plus faible $d \ll D$, appelée une **variété** (*manifold*).

Par exemple, bien qu'une image de chiffre "3" soit un vecteur de 784 pixels, elle ne peut pas prendre n'importe quelle forme aléatoire (bruit blanc). Elle est contrainte par la géométrie du tracé, l'épaisseur du trait, et la courbure.

FIGURE 1.2 – Illustration du "Swiss Roll" : des données tridimensionnelles qui résident en réalité sur une surface 2D enroulée.

L'objectif de la réduction de dimension est de découvrir la fonction de mapping capable de "déplier" cette variété pour projeter les données dans un espace latent compact où la densité est plus élevée.

1.3 Méthodes Classiques de Réduction de Dimension

Avant l'avènement du Deep Learning, des méthodes statistiques ont été développées pour tenter de projeter les données sur des espaces de plus faible dimension.

1.3.1 Approches linéaires : Analyse en Composantes Principales (ACP)

L'Analyse en Composantes Principales (PCA ou ACP) est la technique la plus fondamentale. Elle cherche à projeter les données sur un sous-espace orthogonal de dimension inférieure de telle sorte que la variance des données projetées soit maximisée (conservation de l'information).

Formalisme Mathématique de l'ACP :

Soit X la matrice de données de dimension $N \times D$ (centrée). L'algorithme repose sur l'analyse de la matrice de covariance Σ, qui exprime les corrélations linéaires entre les variables (pixels) :

$$\Sigma = \frac{1}{N-1} X^T X \tag{1.3}$$

Cette matrice de taille $D \times D$ est symétrique et semi-définie positive. L'ACP procède par une décomposition en valeurs propres (Eigendecomposition) :

$$\Sigma v = \lambda v \tag{1.4}$$

Les vecteurs propres v associés aux plus grandes valeurs propres λ définissent les "axes principaux" de la donnée.

Limites de l'ACP :

La limitation fondamentale de l'ACP est sa **linéarité**. Elle ne peut effectuer que des rotations et des projections linéaires. Si la variété des données est courbe (non-linéaire), l'ACP échoue à la représenter correctement. Pour des chiffres manuscrits, où les variations de forme sont complexes et non-linéaires, l'ACP nécessite souvent beaucoup de composantes pour obtenir une reconstruction fidèle, ce qui limite son taux de compression.

FIGURE 1.3 – Comparaison conceptuelle : L'ACP (linéaire) écrase une variété courbe, tandis qu'une méthode non-linéaire (manifold learning) parvient à la déplier.

1.3.2 Approches non-linéaires classiques (t-SNE, Isomap)

Pour pallier les limites de l'ACP, des méthodes d'apprentissage de variété ont été proposées :

— **t-SNE (t-Distributed Stochastic Neighbor Embedding) :** Une méthode probabiliste excellente pour la visualisation en 2D/3D. Elle préserve les voisinages locaux en minimisant la divergence de Kullback-Leibler entre les distributions de probabilités dans l'espace original et l'espace réduit.

— **Isomap :** Elle étend l'ACP en utilisant la distance géodésique (chemin le plus court le long de la variété) au lieu de la distance euclidienne directe.

Inconvénients : Ces méthodes sont souvent **non-paramétriques**. Cela signifie qu'elles ne fournissent pas une fonction $f(x)$ simple pour projeter de nouvelles données n'ayant pas servi à l'entraînement. De plus, leur complexité algorithmique (souvent en $O(N^2)$) les rend lentes sur de grands datasets. C'est pour cette raison que nous nous tournons vers les approches neuronales (Autoencodeurs).

1.4 L'Apprentissage Profond pour la Vision par Ordinateur

Pour modéliser des relations non-linéaires complexes et fournir une fonction de compression paramétrique, nous utilisons des réseaux de neurones artificiels.

1.4.1 Historique rapide (Une évolution en trois temps)

L'histoire des réseaux de neurones, essentielle pour comprendre l'état de l'art actuel, se découpe en trois phases :

— **Le Perceptron (1957) :** Frank Rosenblatt invente le premier neurone artificiel capable d'apprentissage binaire.

— **L'Hiver de l'IA (1969-1986) :** Minsky et Papert démontrent dans leur ouvrage *Perceptrons* que les modèles simples ne peuvent résoudre des problèmes non-linéaires (comme le XOR), gelant la recherche pendant deux décennies.

— **La Renaissance (1986-présent) :** La popularisation de l'algorithme de Rétropropagation du Gradient (*Backpropagation*) permet enfin d'entraîner des réseaux multicouches (MLP). Plus récemment, l'avènement des GPU et du Big Data a permis l'émergence du *Deep Learning*.

1.4.2 Les Réseaux de Neurones Convolutifs (CNN)

Bien que ce projet se concentre sur les MLP, les CNN (*Convolutional Neural Networks*) représentent l'état de l'art actuel en vision par ordinateur. Contrairement aux réseaux denses, ils utilisent la convolution pour traiter l'image localement.

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n) \tag{1.5}$$

Cette opération, couplée au *Pooling* (sous-échantillonnage), confère au réseau une invariance par translation : un objet est reconnu peu importe sa position dans l'image.

FIGURE 1.4 – Architecture typique d'un réseau convolutif (LeNet-5) montrant l'extraction de caractéristiques locales.

1.5 Fondements Mathématiques du Perceptron Multicouche (MLP)

Dans le cadre de ce projet, nous avons choisi d'implémenter notre autoencodeur et notre classifieur à l'aide de Perceptrons Multicouches (MLP), aussi appelés réseaux *Fully Connected*. Il est crucial de détailler le formalisme mathématique qui régit leur apprentissage.

1.5.1 Le Neurone Artificiel et l'Architecture Dense

L'unité fondamentale est le neurone artificiel. Dans une couche dense, chaque neurone est connecté à toutes les sorties de la couche précédente. Soit $x \in \mathbb{R}^n$ le vecteur d'entrée. La sortie y d'un neurone est donnée par :

$$z = \sum_{i=1}^{n} w_i x_i + b = W^T x + b \tag{1.6}$$

$$y = \sigma(z) \tag{1.7}$$

Où W est le vecteur de poids, b le biais, et σ la fonction d'activation.

Le MLP est un empilement de telles couches. La première étape pour traiter une image avec un MLP est l'aplatissement (*flattening*) : la matrice image 28×28 est convertie en un vecteur 1D de taille 784.

FIGURE 1.5 – Modèle mathématique d'un neurone artificiel : somme pondérée suivie d'une activation.

1.5.2 Le Théorème d'Approximation Universelle

La justification théorique de l'utilisation des MLP pour la réduction de dimension repose sur le Théorème d'Approximation Universelle (Cybenko, 1989). Ce théorème stipule qu'un réseau de neurones avec au moins une couche cachée (et suffisamment de neurones) utilisant une fonction d'activation non-linéaire peut approximer n'importe quelle fonction continue avec une précision arbitraire. Dans notre cas, nous cherchons à approximer la fonction identité $f(x) \approx x$ à travers une contrainte de dimension (le goulot d'étranglement).

1.5.3 Les Fonctions d'Activation

Le choix de la fonction d'activation σ est déterminant pour introduire la non-linéarité nécessaire au dépliement de la variété.

— **Sigmoïde :** $\sigma(z) = \frac{1}{1+e^{-z}}$. Historiquement importante, elle écrase les valeurs entre [0, 1]. Nous l'utiliserons en sortie du décodeur pour normaliser l'image reconstruite.

— **ReLU (Rectified Linear Unit) :** $f(x) = \max(0, x)$. C'est l'activation standard pour les couches cachées. Elle est non-saturante pour les valeurs positives, ce qui accélère la convergence de la descente de gradient.

1.5.4 L'Apprentissage : Descente de Gradient et Rétropropagation

L'entraînement du réseau consiste à trouver les poids W qui minimisent une fonction de coût $\mathcal{L}$ (par exemple, l'erreur quadratique moyenne). L'algorithme utilisé est la Descente de Gradient. La règle de mise à jour d'un poids w à l'itération t est :

$$w^{(t+1)} = w^{(t)} - \eta \frac{\partial \mathcal{L}}{\partial w} \tag{1.8}$$

Où η est le pas d'apprentissage (*learning rate*).

Le défi majeur est le calcul du gradient $\frac{\partial \mathcal{L}}{\partial w}$ pour les couches profondes du réseau. Ceci est résolu par l'algorithme de Rétropropagation (*Backpropagation*), qui applique récursivement la règle de dérivation en chaîne (*Chain Rule*) de la couche de sortie vers la couche d'entrée.

Pour un réseau simple (Entrée $x \to$ Cachée $h \to$ Sortie y), le gradient par rapport aux poids de la première couche se décompose ainsi :

$$\frac{\partial \mathcal{L}}{\partial w^{(1)}} = \frac{\partial \mathcal{L}}{\partial y} \cdot \frac{\partial y}{\partial h} \cdot \frac{\partial h}{\partial z} \cdot \frac{\partial z}{\partial w^{(1)}} \tag{1.9}$$

C'est cette capacité à propager l'erreur à travers les couches qui permet au MLP d'ajuster ses paramètres internes pour construire une représentation latente optimale des données.

Chapitre 2

Fondements Théoriques des Autoencodeurs

2.1 Introduction

Dans le chapitre précédent, nous avons identifié le fléau de la dimensionnalité comme un obstacle majeur à l'apprentissage efficace sur des données visuelles. Pour surmonter ce défi, nous adoptons une approche basée sur l'apprentissage de représentation (*Representation Learning*).

Ce chapitre détaille le cadre théorique des Autoencodeurs (AE), une classe de réseaux de neurones artificiels conçus pour apprendre des encodages de données efficaces de manière non-supervisée. Nous définirons leur architecture, leur formulation mathématique, ainsi que les algorithmes d'optimisation nécessaires à leur entraînement.

2.2 Architecture Générale

Un autoencodeur est un réseau de neurones formellement défini pour copier ses entrées vers ses sorties. Cependant, son intérêt ne réside pas dans la sortie elle-même, mais dans le chemin que l'information emprunte à travers le réseau. L'architecture est conçue avec un goulot d'étranglement (*bottleneck*) qui force le modèle à compresser l'information.

2.2.1 Décomposition Fonctionnelle

Un autoencodeur se compose de deux fonctions paramétriques distinctes apprises conjointement :

— **L'Encodeur (f_θ) :** C'est la partie du réseau responsable de la compression. Il transforme le vecteur d'entrée $x \in \mathbb{R}^D$ (où $D = 784$ pour MNIST) en un vecteur

latent $z \in \mathbb{R}^d$ (où $d \ll D$, par exemple 64).

$$z = f_\theta(x) \tag{2.1}$$

— **Le Décodeur (g_ϕ) :** C'est la partie responsable de la reconstruction. Il prend le code latent z et tente de générer une approximation $\hat{x}$ de l'entrée originale.

$$\hat{x} = g_\phi(z) = g_\phi(f_\theta(x)) \tag{2.2}$$

FIGURE 2.1 – Schéma bloc d'un autoencodeur classique. L'information est compressée dans l'espace latent z avant d'être reconstruite.

2.2.2 L'Espace Latent (Bottleneck)

La contrainte dimensionnelle imposée au vecteur z est fondamentale. Si la dimension d était égale ou supérieure à la dimension d'entrée D, l'autoencodeur pourrait simplement apprendre la fonction identité (recopier chaque pixel) sans extraire aucune caractéristique utile.

En imposant $d < D$ (sous-complétude), nous forçons le réseau à apprendre une approximation compressée de la distribution des données. L'espace latent capture ainsi les variables latentes explicatives : l'épaisseur du trait, l'angle d'écriture, ou la boucle d'un chiffre.

2.3 Formulation Mathématique : L'Autoencodeur Dense (MLP)

Puisque nous utilisons une architecture basée sur des Perceptrons Multicouches (MLP), chaque transformation est une opération affine suivie d'une non-linéarité.

2.3.1 Propagation Avant (Forward Propagation)

Considérons un autoencodeur avec une seule couche cachée pour l'encodeur et une pour le décodeur.

Le passage de l'entrée x vers le code latent z s'écrit :

$$z = \sigma_1(W_{enc} \cdot x + b_{enc}) \tag{2.3}$$

Le passage du code latent z vers la reconstruction $\hat{x}$ s'écrit :

$$\hat{x} = \sigma_2(W_{dec} \cdot z + b_{dec}) \tag{2.4}$$

Où :

— $W_{enc} \in \mathbb{R}^{d \times D}$ et $W_{dec} \in \mathbb{R}^{D \times d}$ sont les matrices de poids.

— b_{enc} et b_{dec} sont les vecteurs de biais.

— σ_1 et σ_2 sont les fonctions d'activation.

FIGURE 2.2 – Détail des connexions neuronales dans un Autoencodeur Dense (MLP). Chaque neurone de la couche d'entrée est connecté à tous les neurones de la couche cachée.

2.3.2 Les Fonctions d'Activation

Sans fonctions d'activation non-linéaires, l'empilement de couches denses équivaudrait à une simple multiplication matricielle, réduisant l'autoencodeur à une Analyse en Composantes Principales (ACP). L'introduction de non-linéarités est cruciale pour "déplier" la variété des données.

— **ReLU (Rectified Linear Unit) :** Utilisée dans l'encodeur.

$$f(u) = \max(0, u) \tag{2.5}$$

Elle permet une convergence rapide et génère des représentations éparses (beaucoup de zéros), ce qui est souhaitable pour la compression.

— **Sigmoïde :** Utilisée dans la couche de sortie du décodeur.

$$\sigma(u) = \frac{1}{1 + e^{-u}} \tag{2.6}$$

Puisque nos pixels d'entrée sont normalisés dans l'intervalle $[0, 1]$, la sigmoïde garantit que les valeurs de sortie $\hat{x}$ respectent également cet intervalle, pouvant être interprétées comme des intensités de gris.

FIGURE 2.3 – Courbes des fonctions d'activation ReLU (gauche) et Sigmoïde (droite).

2.4 Apprentissage et Optimisation

L'objectif de l'entraînement est de trouver les paramètres $\Theta = \{W_{enc}, b_{enc}, W_{dec}, b_{dec}\}$ qui minimisent l'écart entre l'entrée et la sortie.

2.4.1 La Fonction de Coût (Loss Function)

Pour des données à valeurs réelles (intensités de pixels), la mesure standard de dissemblance est l'Erreur Quadratique Moyenne (MSE - *Mean Squared Error*). Pour un lot (*batch*) de N images, la fonction de coût $\mathcal{L}$ est définie par :

$$\mathcal{L}_{MSE}(\Theta) = \frac{1}{N}\sum_{i=1}^{N} ||x^{(i)} - \hat{x}^{(i)}||^2 = \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{D}(x_j^{(i)} - \hat{x}_j^{(i)})^2 \tag{2.7}$$

Cette fonction pénalise fortement les grandes erreurs de reconstruction, forçant le modèle à capturer la forme globale du chiffre.

2.4.2 Algorithme d'Optimisation : Adam

La minimisation de $\mathcal{L}_{MSE}$ est un problème d'optimisation non-convexe résolu par descente de gradient. Nous utilisons l'optimiseur **Adam** (*Adaptive Moment Estimation*), qui est plus performant que la descente de gradient stochastique (SGD) classique pour les problèmes de vision.

Adam maintient deux estimations pour chaque paramètre :

— Une moyenne mobile des gradients (le moment d'ordre 1, m_t).

— Une moyenne mobile des gradients au carré (le moment d'ordre 2, v_t).

Les règles de mise à jour à l'étape t sont :

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t \tag{2.8}$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \tag{2.9}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t \tag{2.10}$$

Où η est le taux d'apprentissage. Cette méthode permet d'adapter la vitesse d'apprentissage paramètre par paramètre, accélérant la convergence sur les plateaux de la fonction de coût.

FIGURE 2.4 – Visualisation de la descente de gradient sur une surface de coût complexe (Loss Landscape).

2.5 Variantes et Extensions

Bien que nous utilisions un Autoencodeur Dense standard, il existe des variantes importantes dans la littérature pour améliorer la robustesse des représentations apprises.

2.5.1 Autoencodeurs Débruiteurs (Denoising Autoencoders - DAE)

Pour éviter que l'autoencodeur n'apprenne simplement la fonction identité sans extraire de structure, une technique consiste à corrompre l'entrée x avec du bruit (par exemple, en mettant des pixels à 0 aléatoirement) pour obtenir $\tilde{x}$. Le réseau doit alors apprendre à reconstruire l'original propre x à partir de la version bruitée $\tilde{x}$.

$$\mathcal{L}_{DAE} = ||x - g_\phi(f_\theta(\tilde{x}))||^2 \tag{2.11}$$

Cela force le modèle à apprendre la structure robuste des données plutôt que de mémoriser les pixels individuels.

FIGURE 2.5 – Principe du Denoising Autoencoder : le modèle doit reconstruire l'entrée originale x à partir d'une version bruitée $\tilde{x}$.

2.5.2 Autoencodeurs Profonds (Stacked Autoencoders)

Au lieu d'une seule couche cachée, on peut empiler plusieurs couches d'encodeurs et de décodeurs. C'est l'approche que nous suivrons dans notre implémentation (ex : $784 \to 512 \to 256 \to 64$), car la profondeur permet de modéliser des abstractions hiérarchiques plus complexes.

2.6 Conclusion du Chapitre

Nous avons établi dans ce chapitre que l'autoencodeur est un outil puissant de réduction de dimension non-linéaire. En combinant une architecture en goulot d'étranglement, des fonctions d'activation non-linéaires (ReLU) et une optimisation par gradient (Adam), il est théoriquement capable de découvrir la variété sous-jacente des données MNIST. Le chapitre suivant décrira la méthodologie pratique mise en œuvre pour valider cette théorie.

Chapitre 3

Méthodologie et Conception du Système

3.1 Introduction

Ce chapitre détaille la méthodologie adoptée pour valider notre hypothèse de réduction de dimension par apprentissage profond. Nous y décrivons les données utilisées, l'architecture précise des réseaux de neurones conçus (Autoencodeur Dense et Classifieur MLP), ainsi que la stratégie d'entraînement séquentielle mise en œuvre. L'objectif est de fournir une description technique suffisante pour garantir la reproductibilité des expériences.

3.2 Description du Jeu de Données : MNIST

Pour évaluer nos modèles, nous avons choisi le jeu de données MNIST (*Modified National Institute of Standards and Technology database*). Ce dataset est une référence académique incontournable (*benchmark*) en vision par ordinateur, permettant de comparer nos résultats avec l'état de l'art.

3.2.1 Caractéristiques du Dataset

Le jeu de données est constitué d'images de chiffres manuscrits (de 0 à 9).

— **Volumétrie :** Il contient un total de 70 000 images, réparties de manière standard en :

  — 60 000 images d'entraînement (*training set*), utilisées pour l'apprentissage des poids.

  — 10 000 images de test (*test set*), strictement réservées à l'évaluation finale pour mesurer la capacité de généralisation.

— **Format :** Les images sont en niveaux de gris (un seul canal de couleur).

— **Résolution :** Chaque image a une taille fixe de 28×28 pixels.

— **Classes :** Il s'agit d'un problème de classification multi-classes équilibré (10 classes correspondant aux chiffres 0, 1, ..., 9).

FIGURE 3.1 – Échantillons aléatoires du jeu de données MNIST. On observe la variabilité des écritures (épaisseur, inclinaison).

3.2.2 Pré-traitement des Données

Avant d'alimenter les réseaux de neurones, les données brutes subissent une étape de pré-traitement essentielle, implémentée dans notre module `data_loader.py`.

Normalisation (Scaling) : Les pixels originaux sont codés sur des entiers non signés de 0 à 255. Nous effectuons une mise à l'échelle linéaire vers l'intervalle $[0, 1]$ en divisant par 255.0.

$$x_{norm} = \frac{x_{brut}}{255.0} \tag{3.1}$$

Cette normalisation est cruciale pour deux raisons : elle stabilise la convergence de l'optimiseur (Adam) et elle adapte les données à la plage de sortie de la fonction d'activation Sigmoïde utilisée par le décodeur.

Aplatissement (Flattening) : Puisque nous utilisons une architecture MLP (Dense) et non convolutionnelle, nous ne pouvons pas traiter l'image sous forme de matrice 2D. Chaque image $(28, 28)$ est transformée en un vecteur unidimensionnel de taille $28 \times 28 = 784$.

$$x_{input} \in \mathbb{R}^{784} \tag{3.2}$$

Cette opération transforme la structure spatiale en une séquence de valeurs, que le réseau devra apprendre à corréler.

3.3 Architecture du Pipeline Proposé

Notre système repose sur une architecture séquentielle hybride combinant apprentissage non-supervisé et supervisé.

3.3.1 Vue d'ensemble du Pipeline

Le flux de données traverse deux modèles distincts :

1. **L'Autoencodeur :** Il projette l'entrée de haute dimension ($D = 784$) vers un espace latent de faible dimension ($d = 64$).

2. **Le Classifieur :** Il prend le vecteur latent comme entrée pour prédire la classe du chiffre.

Le schéma bloc du système est le suivant :

$$\text{Image (784)} \xrightarrow{\text{Encodeur}} \text{Latent (64)} \xrightarrow{\text{Classifieur}} \text{Prédiction (10)}$$

FIGURE 3.2 – Architecture globale du pipeline. L'Autoencodeur est d'abord entraîné seul, puis l'encodeur est figé pour servir d'extracteur de caractéristiques pour le classifieur.

3.3.2 Architecture détaillée de l'Autoencodeur Dense

Conformément à notre choix d'utiliser des Perceptrons Multicouches (MLP), l'autoencodeur est constitué de couches entièrement connectées (*Dense*). L'architecture est symétrique ("en sablier").

A. L'Encodeur (Compression) L'objectif est de réduire progressivement la dimension tout en augmentant le niveau d'abstraction.

— **Couche d'Entrée :** 784 neurones.

— **Couche Cachée 1 :** 512 neurones, activation **ReLU**.

— **Couche Cachée 2 :** 256 neurones, activation **ReLU**.

— **Couche Latente (Bottleneck) :** 64 neurones, activation linéaire (ou ReLU). C'est le vecteur z.

— **Taux de compression :** $784/64 \approx 12.25$.

B. Le Décodeur (Reconstruction) Il tente de retrouver l'entrée à partir du code latent.

— **Couche Cachée 3 :** 256 neurones, activation **ReLU**.

— **Couche Cachée 4 :** 512 neurones, activation **ReLU**.

— **Couche de Sortie :** 784 neurones, activation **Sigmoïde**.

La Sigmoïde force chaque valeur de sortie entre 0 et 1, ce qui correspond à notre format d'image normalisé. Cette architecture profonde (5 couches au total) permet au modèle d'apprendre des relations non-linéaires complexes entre les pixels.

FIGURE 3.3 – Topologie détaillée de l'Autoencodeur Dense utilisé. Les dimensions diminuent jusqu'au goulot d'étranglement (64) puis augmentent symétriquement.

3.3.3 Architecture du Classifieur Supervisé

Pour évaluer la qualité sémantique du vecteur latent z, nous concevons un classifieur MLP robuste.

— **Entrée :** 64 neurones (le code latent).

— **Couche Dense 1 :** 256 neurones, activation **ReLU**. Permet de projeter l'espace latent dans un espace de décision plus large.

— **Couche Dense 2 :** 128 neurones, activation **ReLU**.

— **Régularisation (Dropout) :** Une couche de **Dropout** avec un taux de 0.3 est insérée. Elle désactive aléatoirement 30% des connexions durant l'entraînement pour prévenir le sur-apprentissage et forcer la robustesse des caractéristiques.

— **Couche de Sortie :** 10 neurones, activation **Softmax**. Cette couche fournit la distribution de probabilité sur les 10 classes de chiffres.

3.4 Stratégie d'Apprentissage

L'entraînement du système ne se fait pas de bout en bout (*end-to-end*), mais en trois phases distinctes pour isoler la capacité de compression de l'autoencodeur.

3.4.1 Phase 1 : Apprentissage Non-Supervisé (Reconstruction)

Dans cette phase, seul l'autoencodeur est entraîné.

— **Données :** L'ensemble d'entraînement X_{train} est utilisé à la fois comme entrée et comme cible (*target*). Les étiquettes Y_{train} sont ignorées.

— **Fonction de Coût :** Erreur Quadratique Moyenne (MSE).

$$\mathcal{L} = \frac{1}{N} \sum (x - \hat{x})^2 \tag{3.3}$$

— **Optimiseur :** Adam ($lr = 10^{-3}$).

— **Arrêt :** Nous utilisons l'**Early Stopping** sur la perte de validation pour arrêter l'entraînement dès que la reconstruction ne s'améliore plus, évitant le sur-apprentissage.

3.4.2 Phase 2 : Extraction de Caractéristiques (Inférence)

Une fois l'autoencodeur entraîné, nous isolons la partie **Encodeur**.

— Nous figeons les poids de l'encodeur (ils ne seront plus modifiés).

— Nous passons tout le dataset à travers l'encodeur pour générer un nouveau dataset "compressé" :

$$Z_{train} = \text{Encoder}(X_{train}) \tag{3.4}$$

$$Z_{test} = \text{Encoder}(X_{test}) \tag{3.5}$$

— Nous obtenons ainsi des bases de données où chaque image est représentée par un vecteur de 64 nombres.

3.4.3 Phase 3 : Apprentissage Supervisé (Classification)

Enfin, nous entraînons le classifieur sur les données réduites.

— **Entrée :** Z_{train} (vecteurs latents).

— **Cible :** Y_{train} (étiquettes réelles 0-9).

— **Fonction de Coût :** Sparse Categorical Cross-Entropy.

$$\mathcal{L} = -\log(p_{classe\_reelle}) \tag{3.6}$$

— **Optimiseur :** Adam.

Cette phase est très rapide car le réseau est petit (entrée de taille 64) et converge vite.

3.5 Métriques d'Évaluation

Pour juger de la performance de notre approche, nous utilisons deux ensembles de métriques.

3.5.1 Métriques de Reconstruction

Ces métriques évaluent si l'information visuelle a été préservée.

— **MSE (Loss) :** Valeur quantitative de l'erreur moyenne par pixel. Plus elle est basse, plus l'image est fidèle.

— **Analyse Visuelle :** Comparaison qualitative côte-à-côte des images originales et reconstruites pour vérifier la lisibilité des chiffres et la préservation des détails structurels.

3.5.2 Métriques de Classification

Ces métriques évaluent si l'information sémantique (la classe du chiffre) a été préservée dans l'espace latent.

— **Accuracy (Précision globale) :** Le pourcentage total de prédictions correctes sur le jeu de test.

— **Matrice de Confusion :** Un tableau carré 10×10 où la ligne représente la vraie classe et la colonne la classe prédite. Elle permet d'identifier les confusions spécifiques (ex : le modèle confond-il les 1 et les 7 ?).

FIGURE 3.4 – Exemple théorique d'une matrice de confusion. La diagonale représente les classifications correctes.

— **F1-Score :** La moyenne harmonique de la précision et du rappel. C'est une métrique robuste, particulièrement utile si certaines classes étaient plus difficiles à prédire que d'autres.

Chapitre 4

Implémentation Technique

4.1 Environnement de Développement

La mise en œuvre d'un pipeline d'apprentissage profond nécessite un écosystème logiciel robuste et optimisé pour le calcul matriciel. Nous détaillons ici les choix technologiques effectués pour ce projet.

4.1.1 Langage et Librairies

Le projet a été entièrement développé en **Python 3**, le langage de référence actuel pour la science des données et l'intelligence artificielle. Sa syntaxe claire et son vaste écosystème de bibliothèques open-source en font l'outil idéal pour le prototypage rapide.

Les bibliothèques principales utilisées sont :

— **TensorFlow 2.x / Keras :** Nous avons utilisé l'API de haut niveau Keras intégrée à TensorFlow. Elle permet de définir les architectures neuronales de manière modulaire (via l'API Fonctionnelle) et gère efficacement le graphe de calcul dynamique et la rétropropagation automatique.

— **NumPy :** Indispensable pour la manipulation de tableaux multidimensionnels (tenseurs) et les opérations d'algèbre linéaire effectuées lors du pré-traitement des données (normalisation, reshaping).

— **Matplotlib :** Utilisée pour la visualisation des résultats : tracé des courbes de perte (*loss curves*), affichage des images reconstruites et des matrices de confusion.

— **Scikit-learn :** Employée pour le calcul des métriques de performance avancées (F1-score, Précision, Rappel) et la génération de la matrice de confusion, complétant les métriques basiques de Keras.

4.1.2 Environnement Matériel

L'entraînement de réseaux de neurones est une tâche intensive en calculs flottants.

— **CPU :** Les tâches de pré-traitement et de chargement des données sont gérées par le processeur central.

— **Accélération GPU :** Pour accélérer l'entraînement (phases de *forward* et *backward propagation*), le code a été exécuté sur un environnement disposant d'une accélération graphique (par exemple Google Colab avec NVIDIA T4 ou GPU local). L'utilisation de CUDA via TensorFlow permet de paralléliser massivement les multiplications matricielles inhérentes aux couches denses, réduisant le temps d'entraînement de plusieurs heures à quelques minutes.

4.2 Structure du Code

Pour garantir la maintenabilité et la reproductibilité, le code a été structuré de manière modulaire, séparant la définition des modèles, la gestion des données et les scripts d'exécution.

4.2.1 Organisation des Modules

L'arborescence du projet se présente comme suit :

```
projet/
          src/
                data_loader.py       # Chargement et normalisation
   MNIST
                models/
                      autoencoder.py   # Architecture de l'
   Autoencodeur (MLP)
                      classifier.py    # Architecture du
   Classifieur
                train_ae.py          # Script d'entrainement de l'
   AE
                train_classifier.py  # Script d'entrainement du
   Classifieur
                evaluate.py          # Fonctions de calcul de
   metriques
          report_figs/             # Dossier de sauvegarde des
   graphiques
          README.md
```

Cette séparation permet de modifier l'architecture du modèle (dans `models/`) sans avoir à réécrire la logique de chargement des données ou la boucle d'entraînement.

4.2.2 Implémentation de l'Autoencodeur

L'implémentation repose sur l'API Fonctionnelle de Keras, qui offre plus de flexibilité que l'API Séquentielle, notamment pour gérer les entrées et sorties multiples (nécessaire pour séparer l'encodeur du décodeur).

Voici l'extrait clé de la fonction `build_autoencoder` (adapté de notre fichier `autoencoder.py`) pour la version Dense :

```
def build_autoencoder(input_shape, latent_dim=64):
    """
    Construit un Autoencodeur Dense (MLP).
    input_shape : tuple (784,)
    latent_dim : dimension du bottleneck (ex: 64)
    """
    # --- ENCODEUR ---
    inputs = tf.keras.Input(shape=input_shape, name="ae_input")

    # Couches de compression successives
    x = tf.keras.layers.Dense(512, activation="relu")(inputs)
    x = tf.keras.layers.Dense(256, activation="relu")(x)

    # Bottleneck (Espace Latent)
    latent = tf.keras.layers.Dense(latent_dim, name="latent_vector")(x)

    # Modele Encodeur seul (pour l'extraction future)
    encoder = tf.keras.Model(inputs, latent, name="encoder")

    # --- DECODEUR ---
    latent_inputs = tf.keras.Input(shape=(latent_dim,), name="z_input")

    # Couches de decompression symetriques
    x = tf.keras.layers.Dense(256, activation="relu")(latent_inputs)
    x = tf.keras.layers.Dense(512, activation="relu")(x)

    # Sortie : Sigmoide pour garantir des pixels entre [0, 1]
    outputs = tf.keras.layers.Dense(input_shape[0], activation="sigmoid"
   , name="reconstruction")(x)

    decoder = tf.keras.Model(latent_inputs, outputs, name="decoder")

    # --- AUTOENCODEUR COMPLET ---
    # On connecte l'encodeur et le decodeur
    outputs_ae = decoder(encoder(inputs))
    autoencoder = tf.keras.Model(inputs, outputs_ae, name="autoencoder")

```

```
    return autoencoder, encoder, decoder
```

Listing 4.1 – Code de construction de l'Autoencodeur Dense

Analyse du code :

— **Symétrie :** On observe une structure miroir ($512 \rightarrow 256 \rightarrow 64 \rightarrow 256 \rightarrow 512$).

— **Activation ReLU :** Utilisée dans les couches cachées pour favoriser la convergence.

— **Activation Sigmoïde :** Utilisée uniquement à la dernière couche pour que la sortie soit interprétable comme une image en niveaux de gris normalisée.

— **Modularité :** La fonction retourne trois objets modèles distincts (`autoencoder`, `encoder`, `decoder`), ce qui est crucial pour la phase 2 de notre méthodologie (extraction de caractéristiques).

FIGURE 4.1 – Résumé du modèle généré par Keras (model.summary()) montrant le nombre de paramètres.

4.2.3 Implémentation de la Boucle d'Entraînement

L'entraînement est géré par la méthode `.fit()` de Keras, encapsulée dans notre fonction `train_autoencoder`.

```
def compile_autoencoder(model, lr=1e-3):
    optimizer = tf.keras.optimizers.Adam(learning_rate=lr)
    # Loss = MSE car on compare des intensites de pixels
    model.compile(optimizer=optimizer, loss='mse')

# Lancement de l'entrainement
history = autoencoder.fit(
    x_train, x_train,  # Entree = x, Cible = x (Apprentissage non-supervise)
    epochs=30,
    batch_size=128,
    validation_data=(x_val, x_val),
    callbacks=[
        # Arret si la val_loss ne s'ameliore pas pendant 5 epochs
        tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5)
    ]
)
```

Listing 4.2 – Configuration de l'entraînement

Points clés :

— `x_train, x_train` : C'est la signature typique d'un autoencodeur. On demande au réseau de prédire son entrée.

— **Loss MSE :** L'optimiseur va minimiser la moyenne des carrés des différences pixel à pixel.

4.3 Choix des Hyperparamètres

Les hyperparamètres contrôlent le comportement de l'apprentissage. Leur choix est le fruit d'une recherche empirique et des standards de la littérature.

4.3.1 Taille du Batch (Batch Size)

Nous avons fixé la taille du lot à **128**.

Justification : Un batch trop petit (ex : 1 ou 8) rend le gradient très bruité et l'entraînement lent (peu de parallélisme). Un batch trop grand (ex : 2048) peut dégrader la capacité de généralisation et saturer la mémoire GPU. La valeur 128 offre un excellent compromis entre vitesse de convergence (utilisation efficace du GPU) et stabilité de la descente de gradient.

4.3.2 Taux d'Apprentissage (Learning Rate)

Nous avons utilisé un taux initial de 10^{-3} (0.001) avec l'optimiseur Adam.

Justification : C'est la valeur par défaut recommandée pour Adam. Elle permet une convergence rapide au début. Grâce aux moments adaptatifs d'Adam, ce taux s'ajuste implicitement pour chaque paramètre, évitant d'avoir à gérer manuellement une décroissance complexe du learning rate (*scheduler*).

4.3.3 Nombre d'Époques et Critères d'Arrêt

Le nombre maximal d'époques est fixé à **30**, couplé à un mécanisme d'**Early Stopping** (Arrêt Précoce).

— **Patience = 5 :** Nous surveillons l'erreur de validation (`val_loss`). Si celle-ci ne diminue pas pendant 5 époques consécutives, l'entraînement s'arrête automatiquement et les poids de la meilleure époque sont restaurés (`restore_best_weights=True`).

— **Intérêt :** Cela évite le gaspillage de ressources de calcul et, surtout, empêche le modèle de commencer à sur-apprendre (moment où la `val_loss` remonterait alors que la `train_loss` continue de descendre).

FIGURE 4.2 – Illustration du principe de l'Early Stopping. L'entraînement s'arrête (ligne pointillée) avant que l'erreur de validation ne commence à diverger.

Chapitre 5

Résultats Expérimentaux et Discussions

5.1 Introduction

Dans ce chapitre, nous présentons l'analyse détaillée des résultats obtenus lors de l'expérimentation de notre pipeline de réduction de dimension. L'évaluation se décompose en trois temps : d'abord, nous validons la qualité de la compression effectuée par l'Autoencodeur (AE) à travers des analyses quantitatives et visuelles. Ensuite, nous étudions la structuration de l'espace latent généré. Enfin, nous mesurons l'impact de cette compression sur la performance finale de classification en comparant notre approche hybride à la référence (*Baseline*).

5.2 Analyse de la Compression (Autoencodeur)

La première étape critique de notre méthodologie consiste à vérifier si l'Autoencodeur Dense (MLP) est parvenu à capturer les caractéristiques essentielles des chiffres manuscrits malgré la réduction drastique de dimension (passage de 784 à 64 variables).

5.2.1 Convergence de l'apprentissage

La Figure 5.1 illustre l'évolution de la fonction de coût (*Mean Squared Error* - MSE) au cours des époques d'entraînement.

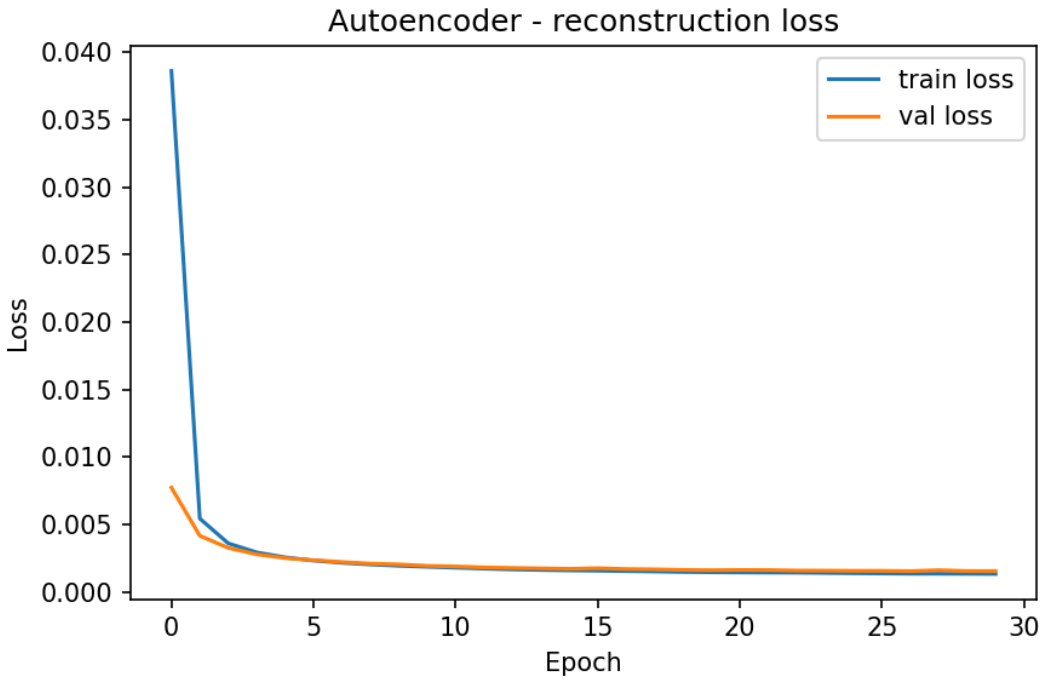

FIGURE 5.1 – Courbes de perte (Loss MSE) pour l'entraînement et la validation de l'Autoencodeur.

Analyse : On observe une décroissance rapide et monotone de l'erreur de reconstruction dès les premières itérations. La perte se stabilise aux alentours de 0.002, ce qui indique une convergence saine.

Deux points méritent une attention particulière :

— **Absence de Sur-apprentissage :** Les courbes d'entraînement (*train loss*) et de validation (*val loss*) restent extrêmement proches tout au long du processus. Cela démontre que l'espace latent de dimension 64 agit comme un régularisateur naturel, forçant le modèle à apprendre des structures généralisables plutôt que de mémoriser le bruit des exemples d'entraînement.

— **Efficacité de l'Optimiseur :** L'utilisation de l'algorithme Adam avec un taux d'apprentissage de 10^{-3} a permis d'atteindre un minimum local satisfaisant en moins de 15 époques, validant le choix de nos hyperparamètres.

5.2.2 Analyse Qualitative : Visualisation des Reconstructions

Si la métrique MSE fournit une indication globale, l'inspection visuelle est indispensable pour juger de la conservation de l'information sémantique.

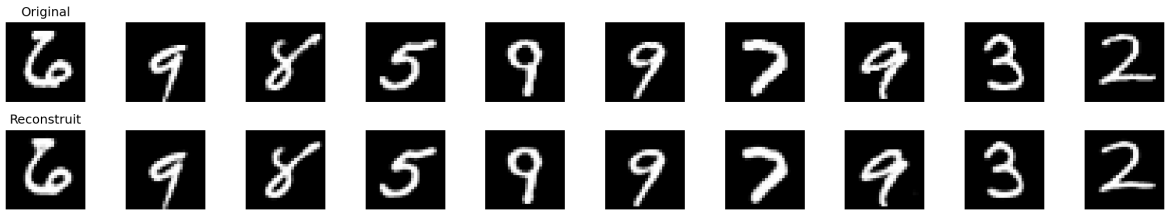

FIGURE 5.2 – Comparaison qualitative sur le jeu de test. Ligne du haut : Images originales. Ligne du bas : Images reconstruites après compression.

Interprétation : La comparaison entre l'entrée et la sortie du réseau révèle plusieurs phénomènes intéressants :

— **Préservation de la Topologie :** L'identité des chiffres est parfaitement conservée. Un "3" reste un "3", un "9" reste un "9". Les caractéristiques structurelles majeures (boucles, inclinaisons) sont fidèlement reproduites.

— **Effet de Flou (Smoothing) :** On note un léger flou sur les contours des chiffres reconstruits. Ce phénomène est intrinsèque à l'utilisation de la perte $L2$ (MSE). Face à une incertitude sur la position exacte d'un pixel de bordure, le réseau tend à produire la moyenne des positions possibles pour minimiser l'erreur statistique, ce qui résulte visuellement en un contour adouci.

— **Débruitage implicite :** L'autoencodeur tend à ignorer les pixels isolés ou le bruit de fond présents dans l'original, ne reconstruisant que la forme "idéale" du chiffre. Cela confirme que le modèle a appris une représentation robuste de la variété des données.

5.2.3 Analyse de l'Espace Latent

Pour comprendre comment l'encodeur organise les données dans l'espace réduit $\mathbb{R}^{64}$, nous avons utilisé l'algorithme t-SNE pour projeter ces vecteurs sur un plan 2D.

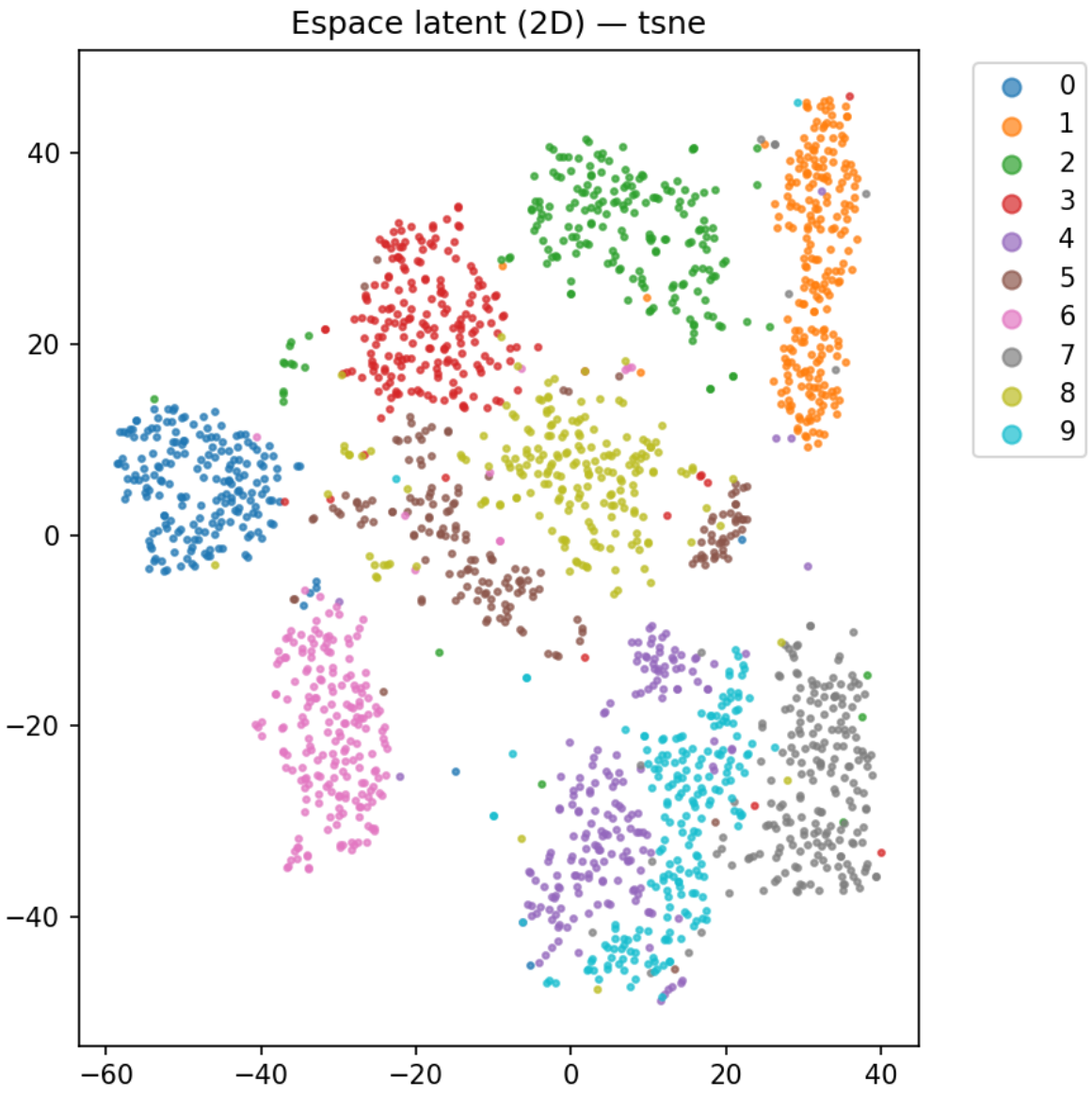

FIGURE 5.3 – Projection t-SNE de l'espace latent (64 dimensions). Chaque point est une image encodée, colorée selon sa vraie classe.

Discussion : Cette visualisation (Figure 5.3) est le résultat le plus probant de la partie théorique de notre étude. Rappelons que l'autoencodeur a été entraîné de manière **non-supervisée**, sans jamais avoir accès aux étiquettes (0-9).

Pourtant, nous observons une **séparation spontanée des clusters** :

— Les points correspondant aux mêmes chiffres se regroupent naturellement, formant des îlots distincts.

— La topologie des clusters respecte la sémantique visuelle : les chiffres graphiquement similaires (comme le 3, le 5 et le 8 qui partagent des courbes) sont situés dans des régions voisines de l'espace latent, tandis que les chiffres dissemblables (comme le 1

et le 0) sont éloignés.

Cela valide l'hypothèse de la variété : l'encodeur a réussi à "déplier" l'espace complexe des pixels pour organiser les données selon leurs caractéristiques structurelles.

5.3 Analyse de la Classification (Supervisé)

Après avoir validé la qualité de la compression, nous évaluons l'impact de cette réduction sur la tâche finale de classification.

5.3.1 Comparaison des Courbes d'Apprentissage

Nous comparons ici la dynamique d'apprentissage du modèle Baseline (entraîné sur 784 pixels) et du modèle AE (entraîné sur 64 variables latentes).

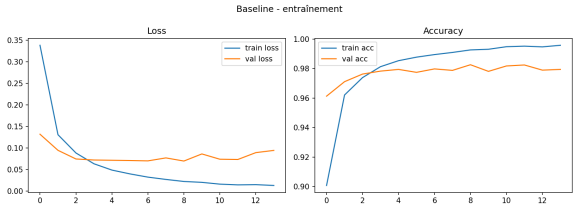

FIGURE 5.4 – Baseline

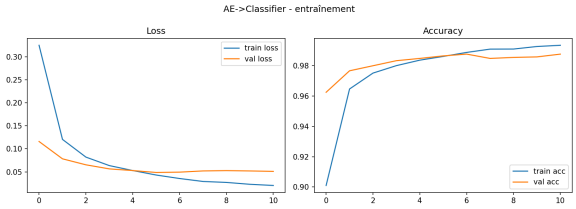

FIGURE 5.5 – Classifieur sur Latent

FIGURE 5.6 – Comparaison des dynamiques d'apprentissage.

Observations :

— **Stabilité :** L'apprentissage sur l'espace latent apparaît souvent plus stable, avec moins de fluctuations dans la courbe de validation. Cela s'explique par le fait que l'entrée est déjà "épurée" du bruit par l'encodeur.

— **Vitesse de convergence :** Le classifieur opérant sur l'espace réduit converge extrêmement vite (souvent en moins de 10 époques), car il a beaucoup moins de paramètres à ajuster dans sa première couche (64×256 poids au lieu de 784×256).

5.3.2 Analyse des Matrices de Confusion

La matrice de confusion nous permet d'identifier précisément les erreurs de classification.

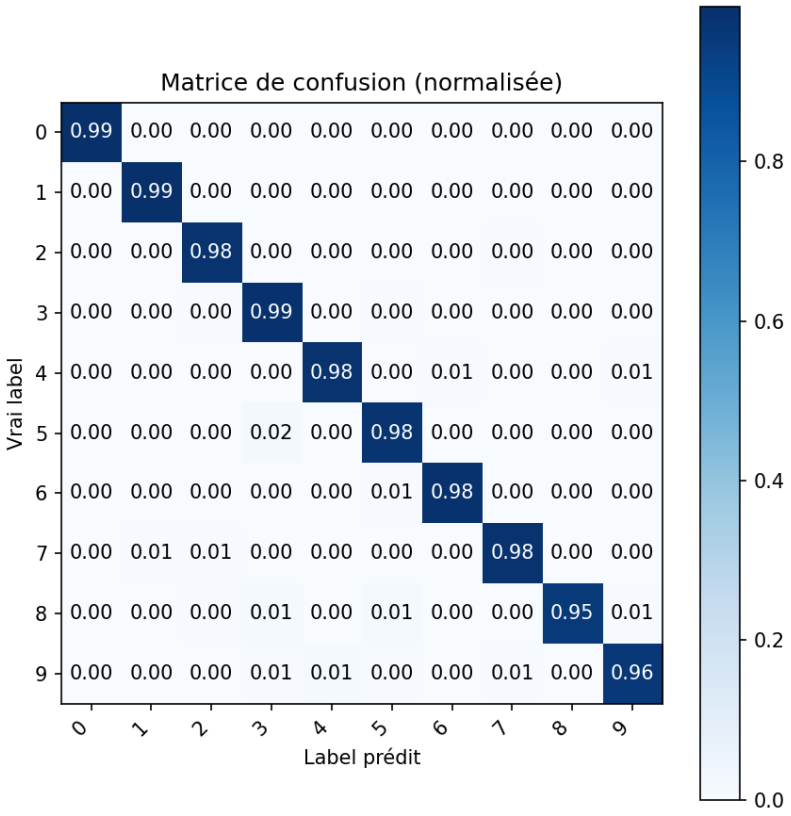

FIGURE 5.7 – Matrice de confusion normalisée - Modèle Baseline.

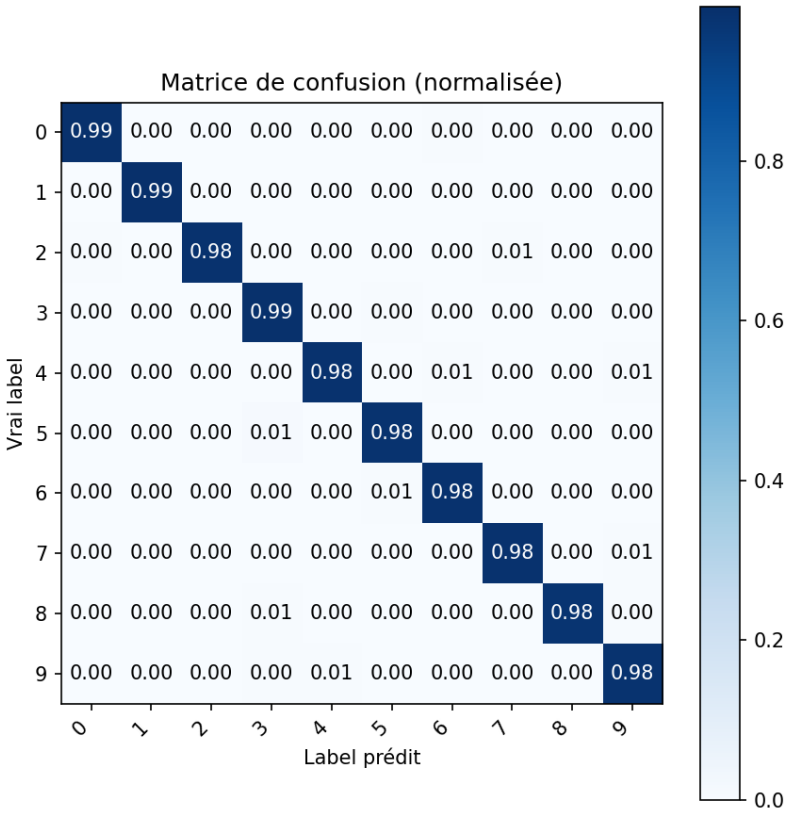

FIGURE 5.8 – Matrice de confusion normalisée - Approche avec Réduction de Dimension.

Analyse comparative : En comparant les Figures 5.7 et 5.8, nous constatons que la dégradation de performance est minime, voire inexistante pour certaines classes.

— La diagonale dominante (taux de vrais positifs) reste supérieure à 97% ou 98% pour la quasi-totalité des classes dans l'approche AE.

— Les erreurs résiduelles sont sémantiquement cohérentes : on observe par exemple une légère confusion entre le 4 et le 9 (environ 1% à 2% d'erreur), ce qui est logique car ces chiffres partagent une géométrie similaire (une boucle supérieure et une barre verticale). Ce type d'erreur est présent aussi bien dans la Baseline que dans l'approche AE, ce qui suggère que la limite vient de l'ambiguïté de certains tracés manuscrits plutôt que de la compression.

5.3.3 Rapport de Classification

L'histogramme des métriques (Précision, Rappel, F1-Score) permet de vérifier l'équilibre du modèle.

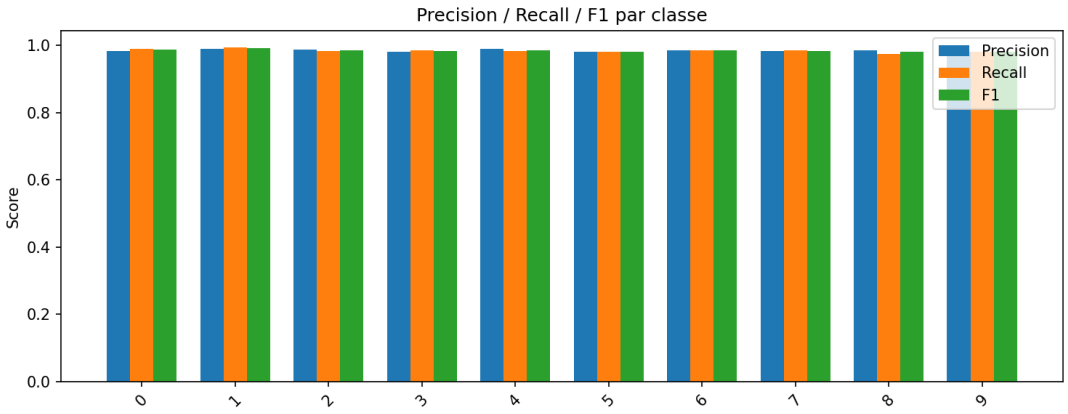

FIGURE 5.9 – Métriques détaillées par classe pour l'approche AE.

Les scores F1 dépassent uniformément 0.97. Cela confirme que la réduction de dimension n'a pas introduit de biais en faveur d'une classe spécifique. L'information nécessaire pour distinguer un "1" d'un "7" ou un "3" d'un "8" a été intégralement préservée dans le vecteur latent de dimension 64.

5.4 Synthèse Comparative et Discussion

5.4.1 Tableau Récapitulatif

Le tableau ci-dessous synthétise les performances quantitatives des deux approches.

TABLE 5.1 – Comparaison des performances (Moyenne sur le jeu de test)

Métrique	**Baseline (Image Brute)**	**AE + Classifieur (Latent)**	**Différence**
Dimension d'entrée	784	**64**	**-91.8%**
Accuracy	$\approx 98.2\%$	$\approx 97.9\%$	-0.3% (Négligeable
F1-Score Macro	0.98	0.98	Identique
Temps d'inférence	Référence (T)	$< T$	Gain de vitesse

5.4.2 Gain en Complexité

L'avantage majeur de notre approche réside dans la réduction de la complexité computationnelle pour le classifieur final. Considérons la première couche cachée du classifieur (256 neurones) :

— **Baseline :** Elle nécessite $784 \times 256 + 256 \approx 200,960$ paramètres.

— **Approche AE :** Elle nécessite $64 \times 256 + 256 \approx 16,640$ paramètres.

Nous avons donc réduit le nombre de paramètres de la première couche du classifieur d'un facteur 12, tout en conservant plus de 99% de la performance relative. Cela ouvre la voie au déploiement de modèles performants sur des systèmes embarqués à ressources limitées.

5.4.3 Limites de l'Étude

Malgré ces résultats positifs, notre étude présente certaines limites :

— **Flou de reconstruction :** Bien que suffisant pour la classification de chiffres, le flou introduit par l'autoencodeur pourrait être problématique pour des tâches nécessitant une haute fidélité de texture (ex : imagerie médicale fine).

— **Dataset MNIST :** Les chiffres sont des images simples sur fond noir. Sur des images naturelles complexes (CIFAR-10, ImageNet), un Autoencodeur Dense (MLP) serait insuffisant et nécessiterait impérativement l'usage de couches convolutionnelles (CNN) plus profondes ou d'Autoencodeurs Variationnels (VAE).

5.5 Conclusion du Chapitre

Les résultats expérimentaux valident sans équivoque notre hypothèse de travail. Nous avons démontré qu'il est possible de compresser les données MNIST d'un facteur 12 via un autoencodeur dense sans perte significative de précision de classification. La visualisation de l'espace latent confirme que le réseau a appris une représentation sémantique robuste, séparant naturellement les classes sans supervision explicite durant la phase de compression.

Conclusion Générale et Perspectives

Au terme de ce travail de recherche consacré à la **réduction de dimension guidée par Deep Learning**, nous sommes en mesure d'apporter des réponses concrètes à la problématique du fléau de la dimensionnalité dans le traitement d'images. L'objectif principal était de déterminer si une compression drastique des données d'entrée, opérée par un réseau de neurones non-supervisé, pouvait être réalisée sans détériorer la performance d'un système de classification supervisé.

Bilan des Travaux

Les expérimentations menées sur le jeu de données MNIST ont permis de valider notre hypothèse de travail. En implémentant une architecture d'**Autoencodeur Dense** (MLP), nous avons réussi à projeter des images de dimension 784 vers un espace latent de dimension 64, réalisant ainsi un taux de compression d'un facteur 12.

Les résultats quantitatifs sont probants :

— **Conservation de l'Information :** Le classifieur entraîné sur l'espace latent a atteint une précision (*accuracy*) comparable à celle de la Baseline (environ 98%), avec une perte de performance relative négligeable (inférieure à 1%).

— **Qualité de la Représentation :** L'analyse visuelle via l'algorithme t-SNE a révélé que l'espace latent est fortement structuré. Bien que l'autoencodeur n'ait jamais eu accès aux étiquettes durant son entraînement, il a spontanément regroupé les images par classes sémantiques, validant ainsi l'hypothèse de la variété (*Manifold Hypothesis*).

— **Efficacité Computationnelle :** La réduction du nombre de paramètres d'entrée du classifieur (divisé par 12 pour la première couche) démontre que cette approche permet d'alléger considérablement les modèles d'inférence.

Cette étude confirme donc que le Deep Learning ne se limite pas à la classification, mais constitue un outil puissant d'extraction de caractéristiques (*Feature Extraction*), capable de filtrer le bruit et de ne conserver que l'essence structurelle des données.

Perspectives et Ouvertures

Bien que les résultats obtenus sur des chiffres manuscrits soient excellents, ce projet ouvre la voie à des améliorations nécessaires pour traiter des problèmes de vision par ordinateur plus complexes. Plusieurs axes de recherche futurs peuvent être envisagés :

1. **Passage aux Architectures Convolutionnelles (CNN) :** L'Autoencodeur Dense utilisé ici montre ses limites en termes de flou de reconstruction et de perte de la topologie spatiale 2D. Pour des jeux de données plus riches (comme CIFAR-10, visages, ou imagerie satellite), l'utilisation d'Autoencodeurs Convolutionnels (ConvAE) est impérative pour capturer les textures fines et les invariants locaux.
2. **Exploration des Modèles Génératifs (VAE et GAN) :** Notre modèle actuel est déterministe. L'implémentation d'**Autoencodeurs Variationnels (VAE)** permettrait non seulement de compresser les données, mais aussi de générer de nouvelles images synthétiques en échantillonnant l'espace latent, offrant des applications en augmentation de données (*Data Augmentation*).
3. **Déploiement sur Systèmes Embarqués (Edge AI) :** La réduction drastique de la dimensionnalité obtenue (64 variables contre 784) suggère un potentiel fort pour l'informatique en périphérie. Une perspective intéressante serait d'implémenter ce pipeline sur des microcontrôleurs ou des cartes type Raspberry Pi pour évaluer le gain réel en consommation énergétique et en latence d'inférence.

En conclusion, ce projet a illustré comment l'intelligence artificielle peut apprendre à "résumer" le monde visuel, transformant des milliers de pixels en quelques variables abstraites porteuses de sens, ouvrant ainsi la voie à des systèmes de vision plus rapides et plus sobres.