Classification d'images de chiffres par autoencodeurs empilés

Mohamed IGADARNE
Master 2 EEA – Parcours I3A

1 Introduction

L'objectif de ce travail est d'étudier expérimentalement l'utilisation d'autoencodeurs empilés pour la classification d'images de chiffres manuscrits. Le jeu de données considéré est constitué d'images en niveaux de gris de petite taille, réparties de manière équilibrée entre dix classes (chiffres de 0 à 9) et séparées en un ensemble d'apprentissage et un ensemble de test. Les étiquettes sont encodées sous forme *one-hot*, ce qui permet d'employer directement une couche softmax en sortie du réseau.

Au-delà de la simple mise en œuvre d'un modèle de classification, ce rapport s'inscrit dans une démarche d'investigation autour de trois questions de recherche :

— **Question 1 :** dans quelle mesure le *fine-tuning* global améliore-t-il les performances d'un réseau à autoencodeurs empilés ?
— **Question 2 :** l'ajout d'un deuxième autoencodeur permet-il d'améliorer les performances de manière significative au regard du coût calculatoire supplémentaire (nombre de paramètres, complexité du réseau) ?
— **Question 3 :** jusqu'à quel point peut-on optimiser la taille des couches cachées (nombre de neurones dans chaque autoencodeur) tout en conservant une précision de classification jugée acceptable (de l'ordre de 99 %) ?

Pour répondre à ces questions, plusieurs architectures sont mises en place, en faisant varier le nombre de couches (un ou deux autoencodeurs) et le nombre de neurones cachés, avec et sans phase de *fine-tuning* supervisé. Les performances sont évaluées à l'aide de l'accuracy globale, des matrices de confusion et du nombre total de paramètres du réseau, afin d'analyser le compromis entre qualité de classification et complexité du modèle. Les sections suivantes détaillent successivement le principe des autoencodeurs et des autoencodeurs empilés, la description du jeu de données, le protocole expérimental retenu, puis les résultats obtenus pour chacune des questions de recherche.

2 Présentation théorique

2.1 Autoencodeur

Un **autoencodeur** est un réseau de neurones artificiels conçu pour apprendre une représentation compacte des données en essayant de reconstruire son entrée en sortie. Il est constitué de deux parties :

— un **encodeur** qui projette l'entrée x dans un **espace latent** h de plus faible dimension,
— un **décodeur**, qui reconstruit une approximation $\hat{x}$ de l'entrée à partir de cette représentation latente h.

Dans ce travail, nous utilisons des autoencodeurs *parcimonieux*. L'idée est d'imposer que, pour une image donnée, seule une petite fraction des neurones cachés soit fortement activée.

2.2 Autoencodeurs empilés et fine-tuning

Un seul autoencodeur permet déjà d'apprendre une nouvelle représentation des données. Pour capturer des structures plus complexes, on peut empiler plusieurs autoencodeurs : la sortie latente du premier sert d'entrée au second, et ainsi de suite. On obtient alors un *stacked autoencoder*.

L'entraînement se déroule en deux étapes :

1. **Pré-entraînement couche par couche** :
   — AE1 est entraîné sur les images d'entrée ;
   — les codes latents de AE1 sont calculés, puis utilisés pour entraîner AE2 ;
   — on obtient ainsi une hiérarchie de représentations de plus en plus abstraites dans l'espace latent.
2. **Fine-tuning global supervisé** : les autoencodeurs empilés sont reliés à une couche de classification en sortie ; l'ensemble du réseau est alors ajusté par rétropropagation à partir d'un critère supervisé (erreur de classification).

Le pré-entraînement non supervisé fournit une bonne initialisation des poids dans l'espace latent $\mathcal{Z}$, tandis que le *fine-tuning* aligne finement ces représentations sur la tâche de classification.

Dans MATLAB Les principaux paramètres d'un autoencodeur sont :
— `L2WeightRegularization` : régularisation L2 sur les poids,
— `SparsityRegularization` : poids de la contrainte de parcimonie,
— `SparsityProportion` : valeur moyenne visée pour l'activation des neurones cachés.

3 Présentation du jeu de données

3.1 Nature et organisation des données

On utilise les fonctions MATLAB `digitTrainCellArrayData` et `digitTestCellArrayData` pour charger les images de chiffres :
— `xTrainImages` et `tTrain` pour l'apprentissage,
— `xTestImages` et `tTest` pour le test.
— `les étiquettes` au format « one-hot » (`tTrain`, `tTest`).

Dans mes simulations, le jeu d'apprentissage contient **5000** images et le jeu de test **5000** images de taille 28×28 pixels, Les images sont utilisées sous forme de matrice de pixels (niveaux de gris). Pour les réseaux empilés, une étape de vectorisation est réalisée ou chaque image est représentée par un vecteur de dimension 784.

3.2 Aspect visuel et variabilité des écritures

Les chiffres manuscrits présentent une grande diversité d'écritures :
— **variabilité de forme** : certains chiffres(exemple de 9 dans la figure 1) peuvent être tracés selon des styles très différents d'une personne à l'autre ;
— **variabilité de taille et d'orientation** : les chiffres sont parfois légèrement inclinés, plus ou moins centrés, ou écrits avec une épaisseur de trait variable ;
— **présence de bruit** : on observe des irrégularités locales (pixels manquants, traits discontinus) qui rendent la reconnaissance plus difficile.

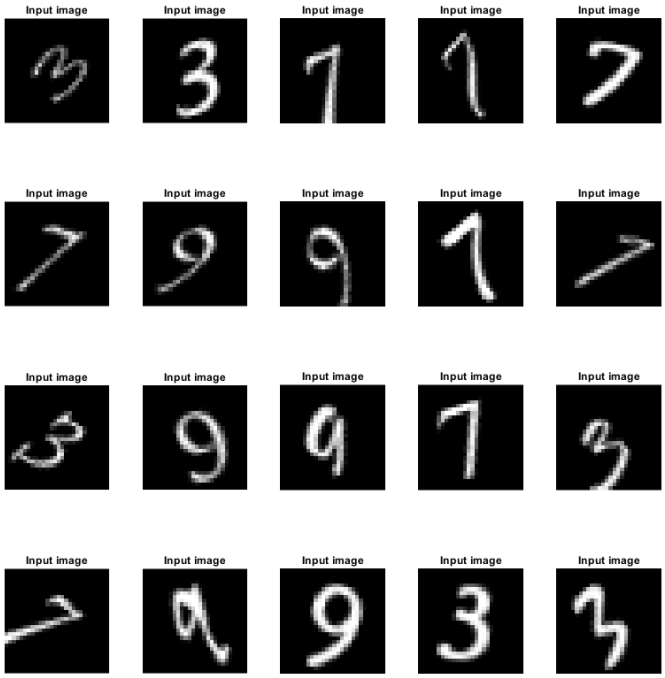

FIGURE 1 – Exemples d'images du jeu d'apprentissage .

Visuellement, certaines classes semblent plus ambiguës que d'autres. Par exemple, certains 1 peuvent être confondus avec des 7 mal écrits, et des chiffres peuvent parfois ressembler à des autres selon le style d'écriture. Cette variabilité justifie le recours à des représentations latentes plus riches, capables de capturer la structure globale du chiffre au-delà de simples caractéristiques locales.

4 Investigation de l'effet du fine-tuning et de l'ajout d'un deuxième autoencodeur

Dans cette partie, nous cherchons à répondre à deux questions principales :

— **Question 1 :** dans quelle mesure le fine-tuning global améliore-t-il les performances d'un réseau à autoencodeurs empilés ?

— **Question 2 :** quel est l'effet de l'ajout d'un deuxième autoencodeur sur les performances, à architecture comparable ?

Pour cela, nous fixons l'architecture de base suivante :

— premier autoencodeur (**AE1**) : $hiddenSize1 = 100$ neurones cachés,

— deuxième autoencodeur (**AE2**) : $hiddenSize2 = 50$ neurones cachés,

— couche de sortie : softmax à 10 neurones (chiffres 0–9).

Nous testons ensuite quatre configurations :

1. **Cas 1 :** AE1 + softmax, sans fine-tuning.
2. **Cas 2 :** AE1 + softmax, avec fine-tuning global.
3. **Cas 3 :** AE1 + AE2 + softmax, sans fine-tuning.
4. **Cas 4 :** AE1 + AE2 + softmax, avec fine-tuning global.

Le tableau 1 résume les performances globales et la complexité des quatre cas.

Configuration	Fine-tuning	Accuracy globale	Erreur	Nb paramètres
Cas 1 : AE1 + softmax	Non	80,38 %	19,62 %	79 510
Cas 2 : AE1 + softmax	Oui	99,02 %	0,98 %	79 510
Cas 3 : AE1 + AE2 + softmax	Non	70,46 %	29,54 %	84 060
Cas 4 : AE1 + AE2 + softmax	Oui	99,44 %	0,56 %	84 060

TABLE 1 – Résumé des performances globales et de la complexité pour les quatre cas étudiés.

Les matrices de confusion associées aux quatre cas sont présentées respectivement aux figures 2 à 5.

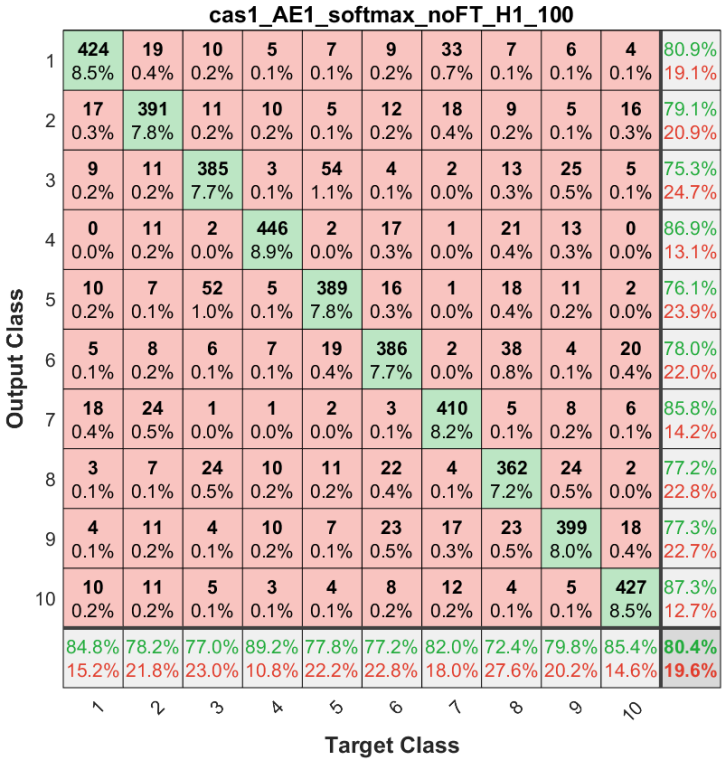

FIGURE 2 – Cas 1 – AE1 + softmax, sans fine-tuning.

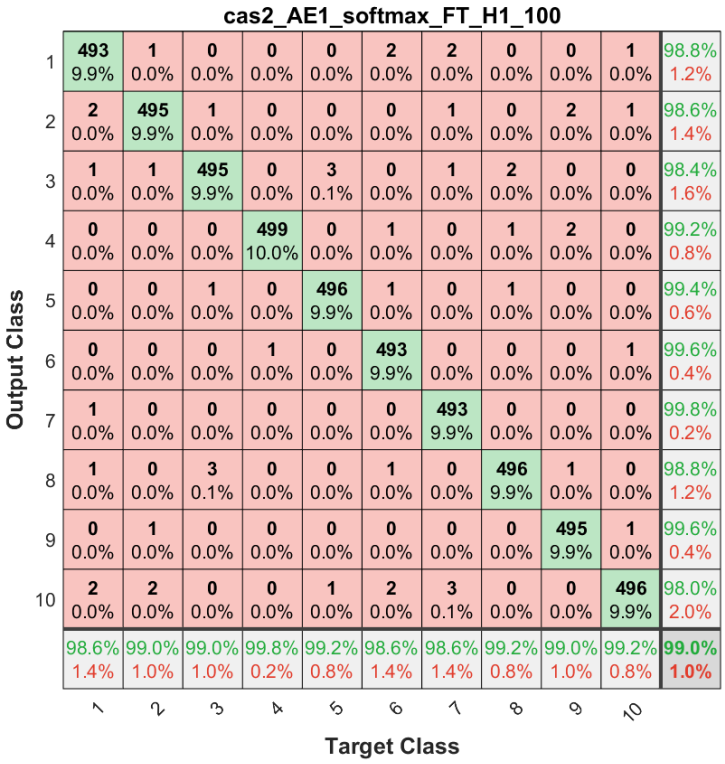

FIGURE 3 – Cas 2 – AE1 + softmax, avec fine-tuning.

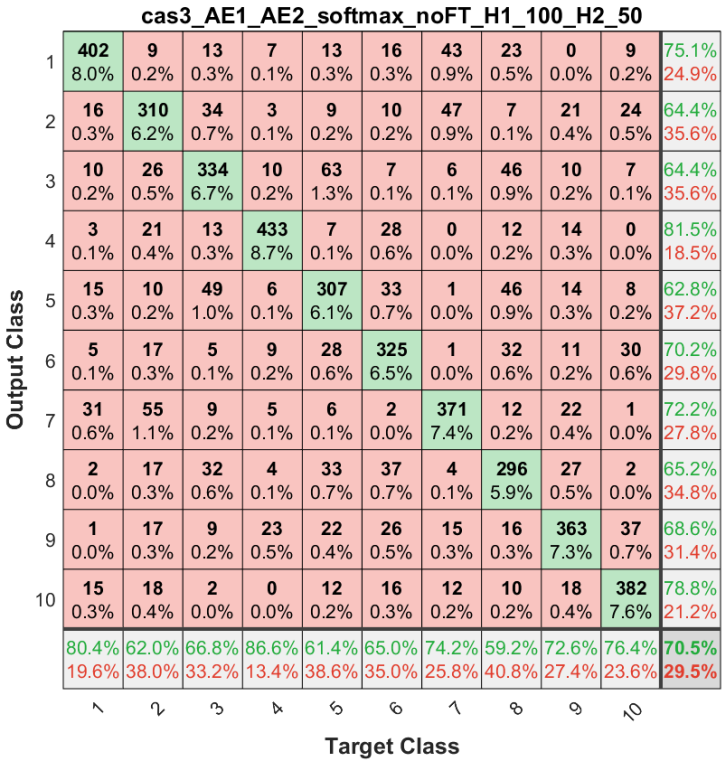

FIGURE 4 – Cas 3 – AE1 + AE2 + softmax, sans fine-tuning.

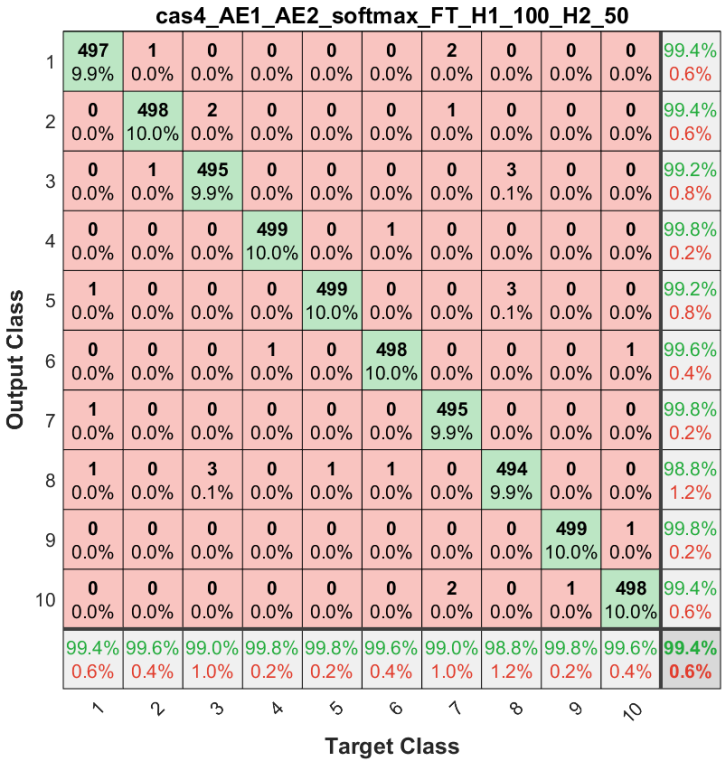

FIGURE 5 – Cas 4 – AE1 + AE2 + softmax, avec fine-tuning.

4.1 Effet du fine-tuning sur l'architecture à un autoencodeur (Cas 1 vs Cas 2)

Dans le **Cas 1** (AE1 + softmax sans fine-tuning), l'accuracy globale atteint seulement **80,4 %**, avec une erreur de 19,6 %. L'analyse de la matrice de confusion (figure 2) montre que :

- certaines classes sont relativement bien reconnues, comme le chiffre 3 (89,2 %), 0 (84,8 %) ou 9 (85,4 %) ;
- d'autres classes sont nettement plus problématiques, notamment les chiffres 7 (72,4 %) et 1–2–4 (entre 77 et 79 %), qui présentent de nombreuses confusions avec des chiffres visuellement proches.

Après fine-tuning global (**Cas 2**), l'accuracy passe à **99,00 %**, soit un gain de **18,6 points de pourcentage**. Le nombre de paramètres reste identique (79 510), seule la valeur des poids est ajustée. Toutes les classes dépassent alors 98,5 % de bonne classification (accuracy par classe comprise entre 98,6 % et 99,8 %). La matrice de confusion (figure 3) montre une diagonale quasi parfaite : chaque chiffre est presque systématiquement affecté à la bonne classe, et les confusions résiduelles sont très rares (quelques erreurs isolées par classe).

Interprétation. Ces résultats illustrent clairement l'apport du fine-tuning : l'apprentissage couche par couche fournit une bonne initialisation de l'encodeur, mais la couche softmax seule n'exploite pas pleinement le potentiel des représentations apprises. La rétropropagation supervisée sur l'ensemble du réseau permet d'ajuster conjointement l'encodeur et la softmax aux contraintes de la tâche de classification, d'où le saut très important de performance.

4.2 Effet du fine-tuning sur l'architecture à deux autoencodeurs (Cas 3 vs Cas 4)

Lorsque l'on ajoute un deuxième autoencodeur (**Cas 3** : AE1 + AE2 + softmax sans fine-tuning), les performances *avant* fine-tuning sont au contraire dégradées par rapport au Cas 1 : l'accuracy globale chute à **70,5 %** (erreur de 29,5 %), malgré un nombre de paramètres plus élevé (84 060 au lieu de 79 510). Les accuracies par classe sont très inhomogènes :

— certaines classes restent assez bien reconnues, par exemple le chiffre 3 (86,6 %) ou 0 (80,4 %),

— d'autres classes sont nettement en difficulté : chiffres 1 et 7 (respectivement 62 % et 59,2 %), 4 (61,4 %) ou 2 (66,8 %).

La matrice de confusion du Cas 3 (figure 4) présente de nombreuses confusions croisées entre ces classes, signe que les représentations extraites par le deuxième autoencodeur ne sont pas encore suffisamment alignées avec la tâche de classification.

Après fine-tuning global (**Cas 4**), la situation change complètement : l'accuracy globale atteint **99,44 %**, soit un gain de **28,9 points** par rapport au Cas 3. Toutes les classes dépassent alors 98,8 % de bonne classification, plusieurs étant très proches de 99,8–99,9 %. La matrice de confusion (figure 5) montre une diagonale pratiquement parfaite, très similaire à celle du Cas 2, avec encore un léger avantage en termes de taux global.

Interprétation. Sans fine-tuning, l'ajout d'un deuxième autoencodeur complique la tâche de la softmax, sans que les paramètres des encodeurs soient réajustés en fonction de la classification. Le réseau devient plus profond et plus complexe, mais pas nécessairement plus pertinent pour la tâche cible. Le fine-tuning global est donc encore plus crucial dans cette configuration : il permet d'exploiter la profondeur du réseau pour apprendre des représentations hiérarchiques vraiment adaptées à la discrimination des chiffres.

4.3 Effet de l'ajout d'un deuxième autoencodeur après fine-tuning (Cas 2 vs Cas 4)

Pour isoler l'effet spécifique de l'ajout du deuxième autoencodeur, il est pertinent de comparer les **Cas 2** et **Cas 4**, c'est-à-dire deux réseaux ayant tous deux bénéficié d'un fine-tuning global :

— Cas 2 : AE1 + softmax, 79 510 paramètres, 99,00 % d'accuracy.

— Cas 4 : AE1 + AE2 + softmax, 84 060 paramètres, 99,44 % d'accuracy.

L'ajout du deuxième autoencodeur augmente la complexité du réseau d'environ **5,7 %** en nombre de paramètres, et permet un gain d'accuracy global d'environ **0,44 point de pourcentage**. La différence entre les matrices de confusion des Cas 2 et 4 est très faible : les deux réseaux commettent très peu d'erreurs et les confusions résiduelles portent sur un nombre extrêmement réduit d'exemples.

Interprétation. On peut considérer que l'ajout d'un deuxième autoencodeur apporte un léger gain de performance, mais au prix d'une complexité accrue. Sur ce problème relativement simple (chiffres manuscrits de petite taille), l'architecture AE1 + softmax, correctement fine-tunée, fournit déjà des résultats quasi-parfaits. Le deuxième autoencodeur permet d'affiner encore la représentation, mais le gain reste marginal au regard de l'augmentation du nombre de paramètres. Dans un contexte contraint (ressources limitées), l'architecture avec un seul autoencodeur pourrait donc être préférée pour son meilleur compromis performance / complexité.

4.4 Synthèse

Les expériences menées montrent que :

— le **fine-tuning global** a un impact majeur sur les performances, en particulier lorsque le réseau est profond (gains de +18,6 points pour AE1 seul et +28,9 points pour AE1+AE2) ;
— l'**ajout d'un deuxième autoencodeur** sans fine-tuning dégrade fortement les performances, malgré l'augmentation de la capacité du modèle ;
— après fine-tuning, le deuxième autoencodeur permet un **gain marginal** d'accuracy (0,4 point) au prix d'une légère hausse du nombre de paramètres.

Ces résultats confirment l'importance de combiner apprentissage couche par couche et rétropropagation supervisée globale pour tirer parti de la profondeur des architectures à autoencodeurs empilés.

5 Optimisation de la taille de l'encodeur 1 à profondeur fixe

Après avoir montré l'effet du fine-tuning et de l'ajout d'un deuxième autoencodeur, nous abordons la question suivante :

« Jusqu'à quel point peut-on réduire la taille de l'architecture tout en conservant une précision de classification acceptable ? »

Pour répondre à cette question, nous considérons uniquement l'architecture **AE1 + AE2 + softmax avec fine-tuning global** (cas 4), et nous faisons varier la taille de la première couche cachée H_1 tout en gardant H_2 constante.

5.1 Protocole d'optimisation

On fixe la taille de l'encodeur 2 à

$$H_2 = 50$$

et l'on fait varier le nombre de neurones cachés de l'encodeur 1 selon

$$H_1 \in \{20,\ 40,\ 60,\ 80,\ 100\}.$$

Pour chaque paire (H_1, H_2), on suit exactement le même protocole :

1. Entraînement de **AE1** sur les images (dimension 28×28) avec un autoencodeur parcimonieux (mêmes hyperparamètres que précédemment).
2. Encodage des images d'apprentissage par AE1 pour obtenir les caractéristiques de niveau 1 (dimension H_1).
3. Entraînement de **AE2** sur ces caractéristiques, avec une taille cachée $H_2 = 50$.
4. Encodage des caractéristiques de niveau 1 par AE2 pour obtenir des caractéristiques de niveau 2 (dimension H_2).
5. Entraînement d'une **couche softmax** sur ces caractéristiques de niveau 2.
6. Construction du réseau empilé AE1+AE2+softmax, puis **fine-tuning global** sur le jeu d'apprentissage vectorisé.
7. Évaluation sur le jeu de test :
   — accuracy globale et erreur,
   — accuracy par classe (10 chiffres),
   — nombre total de paramètres (n_{params}),
   — matrice de confusion.

Le réseau avec $(H_1, H_2) = (100, 50)$, déjà étudié précédemment, sert de **réseau de référence**.

5.2 Résultats globaux

Le tableau 2 résume les performances obtenues pour chaque valeur de H_1 (test sur 5000 images).

H_1	Accuracy globale	Erreur	Nb paramètres
20	96,40 %	3,60 %	17 260
40	99,00 %	1,00 %	33 960
60	**99,42 %**	**0,58 %**	50 660
80	99,18 %	0,82 %	67 360
100	99,40 %	0,60 %	84 060

TABLE 2 – Résultats globaux pour l'architecture AE1+AE2+softmax avec fine-tuning, $H_2 = 50$ fixé et H_1 variable.

Les courbes d'accuracy, d'erreur et de nombre de paramètres en fonction de H_1 sont montrées figures 6 à 8.

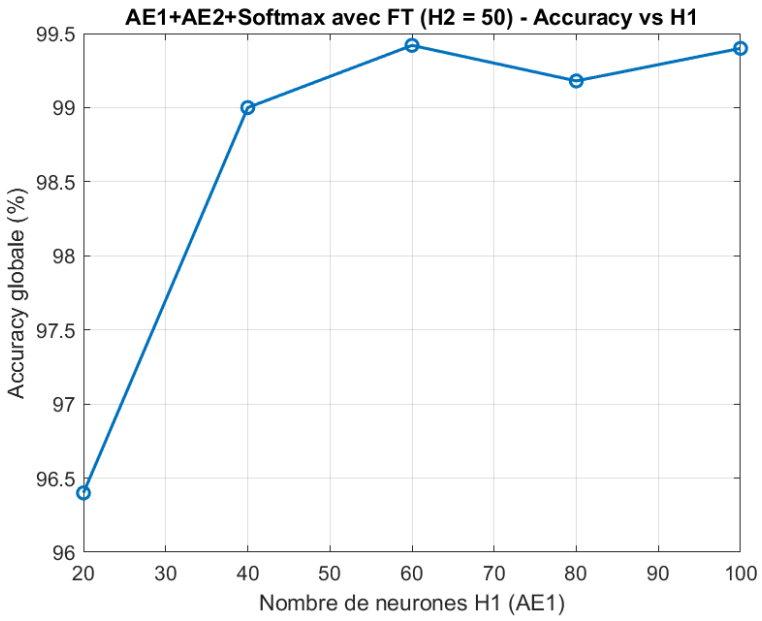

FIGURE 6 – Accuracy globale en fonction de H_1 pour AE1+AE2+softmax avec FT ($H_2 = 50$).

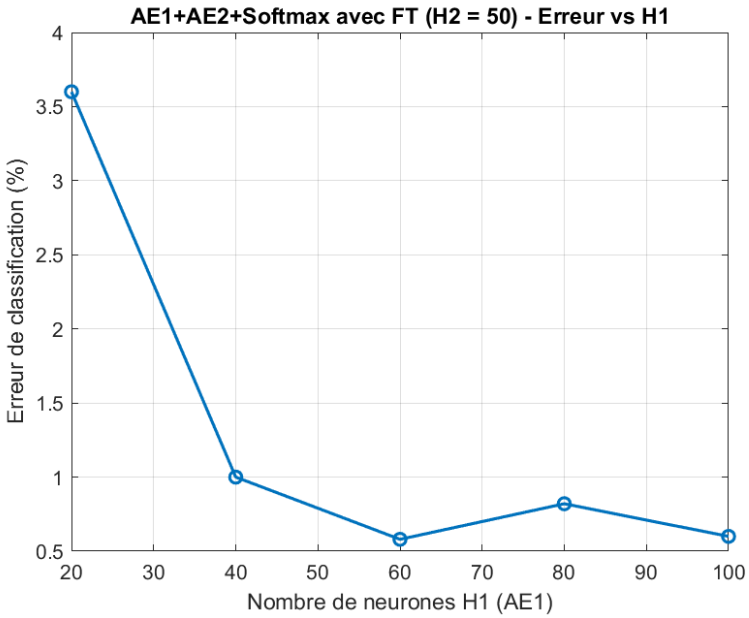

FIGURE 7 – Erreur de classification en fonction de H_1 ($H_2 = 50$).

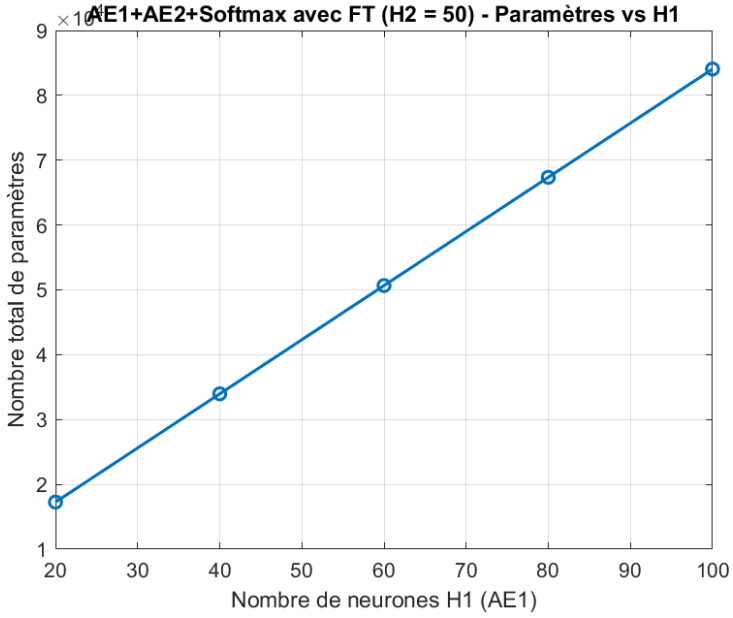

FIGURE 8 – Nombre total de paramètres du réseau en fonction de H_1 ($H_2 = 50$).

On constate que :

— l'accuracy augmente fortement entre $H_1 = 20$ (96,4 %) et $H_1 = 40$ (99,0 %), puis se stabilise pour $H_1 \geq 40$ autour de 99–99,4 % ;

— le minimum d'erreur est atteint pour $H_1 = 60$ (0,58 %), très proche du réseau de référence $H_1 = 100$ (0,60 %) ;

— le nombre de paramètres croît linéairement avec H_1 : réduire H_1 de 100 à 60 fait passer de 84 060 à 50 660 paramètres, soit une réduction d'environ **40 %**, et de 100 à 40 à seulement 33 960 paramètres (environ **60 %** de réduction).

5.3 Analyse des matrices de confusion

Les matrices de confusion correspondantes sont illustrées en figures 9 à 13.

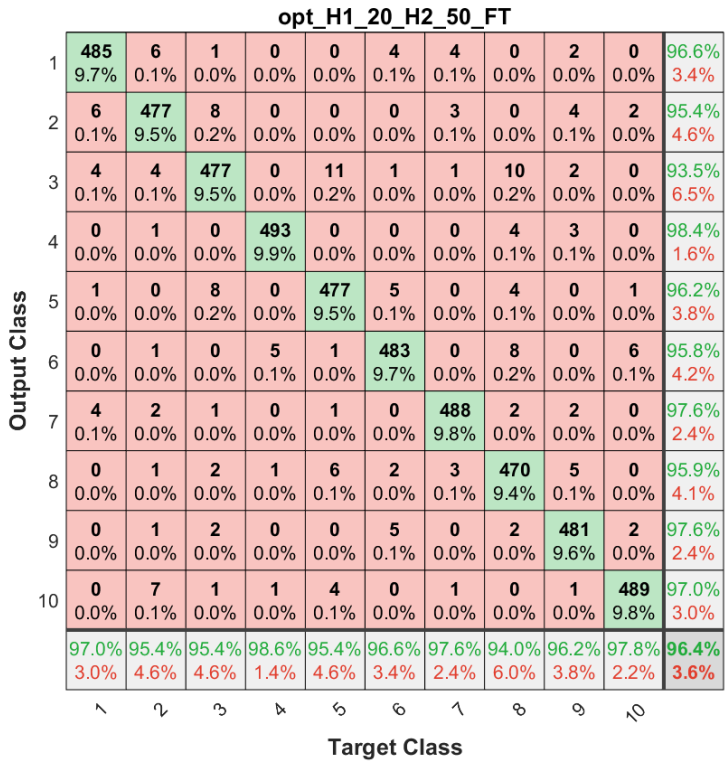

FIGURE 9 – Matrice de confusion pour $H_1 = 20$, $H_2 = 50$.

Pour $H_1 = 20$, la diagonale reste dominante mais plusieurs classes présentent encore des taux de reconnaissance nettement inférieurs à 99 % : les chiffres 1, 2, 4, 7 et 9 sont autour de 95–97 % de bonne classification. On observe davantage de confusions dispersées entre classes, signe que l'espace latent de dimension 20 est un peu trop contraint pour capturer toutes les variations d'écriture des chiffres.

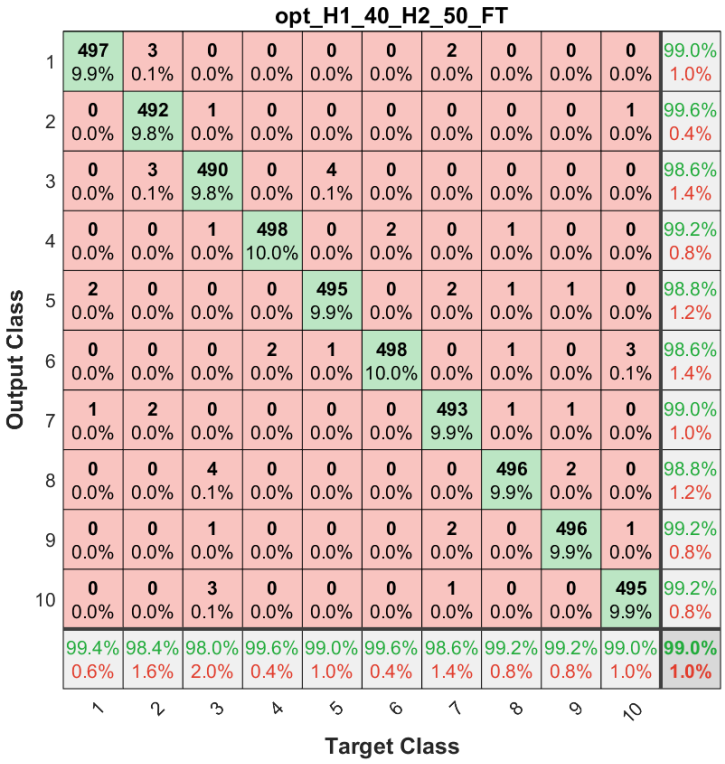

FIGURE 10 – Matrice de confusion pour $H_1 = 40$, $H_2 = 50$.

Avec $H_1 = 40$, la matrice de confusion (figure 10) est déjà très proche de la diagonale parfaite. L'accuracy globale atteint 99,0 %, et toutes les classes dépassent 98 % de bonne classification, plusieurs étant au-dessus de 99 %. Les erreurs résiduelles concernent un nombre très limité d'images.

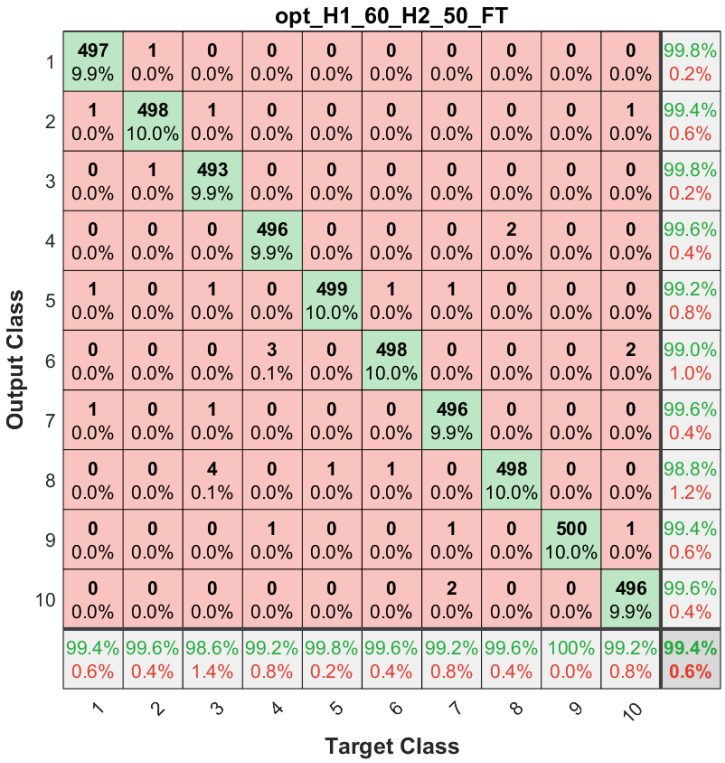

FIGURE 11 – Matrice de confusion pour $H_1 = 60$, $H_2 = 50$.

Pour $H_1 = 60$, la matrice de confusion (figure 11) est quasiment parfaite : la plupart des classes ont une accuracy par classe supérieure à 99,5 %, et certaines atteignent 100 % (aucune erreur). Les erreurs restantes sont très rares (quelques images isolées).

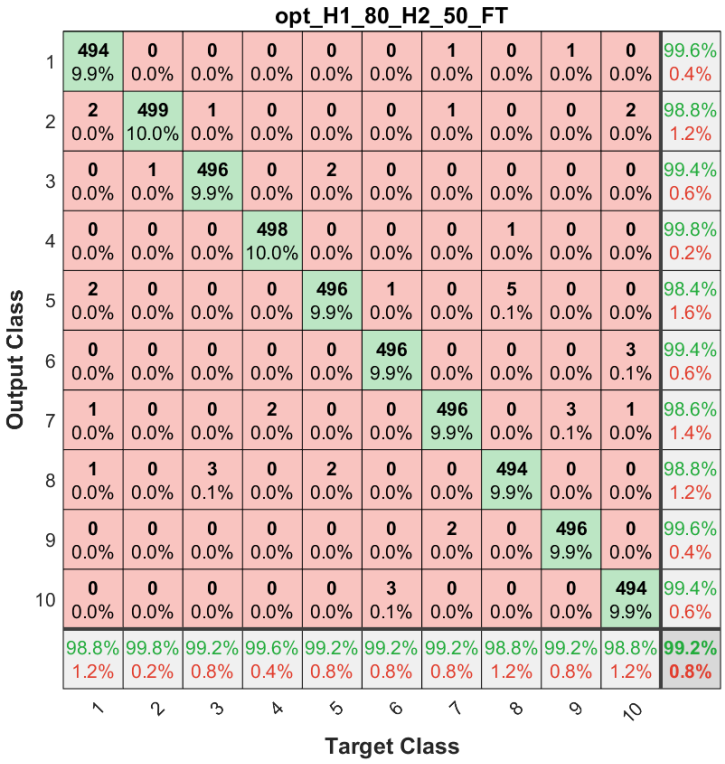

FIGURE 12 – Matrice de confusion pour $H_1 = 80$, $H_2 = 50$.

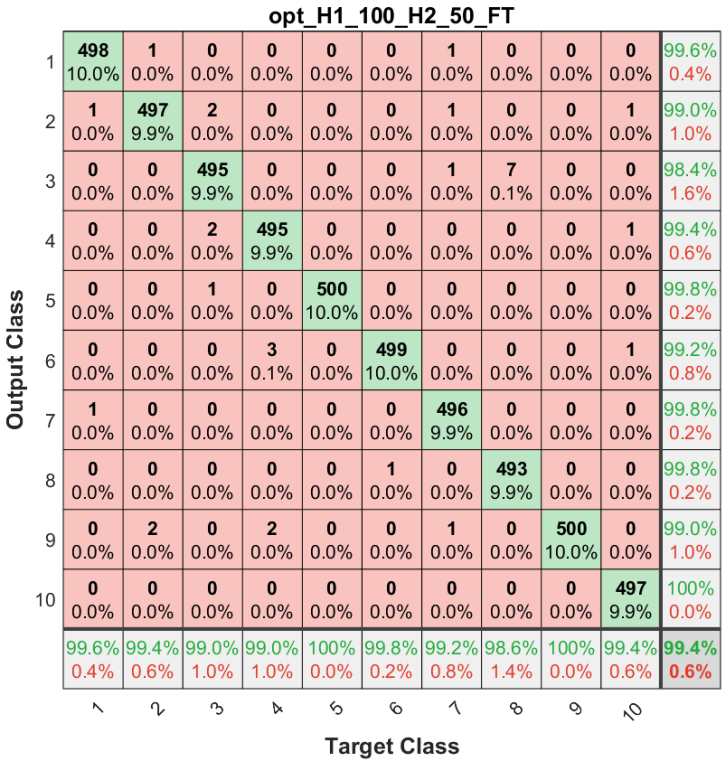

FIGURE 13 – Matrice de confusion pour $H_1 = 100$, $H_2 = 50$ (réseau de référence).

Les matrices pour $H_1 = 80$ et $H_1 = 100$ confirment cette tendance : la diagonale est presque entièrement verte, avec des accuracies par classe comprises entre 98,4 % et 99,8 %. Visuellement, les matrices pour $H_1 = 60$, 80 et 100 sont très similaires, ce qui explique le plateau observé sur la courbe d'accuracy.

5.4 Compromis performance / complexité et réponse à la question

Si l'on fixe un niveau de **précision acceptable** autour de 99 % d'accuracy globale, les résultats permettent de dégager les points suivants :

- **Sous-dimensionnement** : avec $H_1 = 20$, l'accuracy globale n'est que de 96,4 %. Même si ce taux reste correct en valeur absolue, il est nettement inférieur aux autres configurations et s'accompagne de classes à 93–96 % de reconnaissance. Pour ce problème, cette architecture apparaît sous-dimensionnée.
- **Plateau de performance** : à partir de $H_1 = 40$, l'accuracy dépasse 99 %, et les matrices de confusion deviennent très propres. Les configurations $H_1 = 40, 60, 80, 100$ donnent toutes une accuracy globale dans l'intervalle [99,0 %, 99,42 %], avec des différences très faibles entre elles.
- **Point quasi optimal** : la configuration $H_1 = 60$, $H_2 = 50$ offre la meilleure accuracy (99,42 %) tout en utilisant environ **40 % de paramètres en moins** que le réseau de référence $H_1 = 100$, $H_2 = 50$ (50 660 paramètres contre 84 060). La configuration $H_1 = 40$ reste également très intéressante : 99,0 % d'accuracy avec **60 % de paramètres en moins** par rapport au réseau de référence.

On peut donc répondre à la question posée de la manière suivante :

> *Dans le cadre de ce jeu de données (chiffres manuscrits) et pour une architecture à deux autoencodeurs avec fine-tuning, il est possible de réduire significativement la*

taille de l'encodeur 1 tout en conservant une précision de classification supérieure à 99 %. Le réseau reste très performant pour des tailles de couche cachée comprises entre 40 et 60 neurones. En dessous de 40 neurones, la perte de précision devient plus marquée.

En pratique, le choix entre $H_1 = 40$ et $H_1 = 60$ dépendra des contraintes de l'application : si l'objectif est de minimiser la complexité et la mémoire, $H_1 = 40$ constitue un bon compromis ; si l'on souhaite maximiser légèrement la précision tout en restant plus compact que le réseau de référence, $H_1 = 60$ apparaît comme un point de fonctionnement quasi optimal.

5.5 Optimisation de la taille de l'encodeur 2 à H_1 fixé

Après avoir identifié $H_1 = 60$ comme une taille quasi optimale pour le premier autoencodeur, nous cherchons à affiner la dimension de l'encodeur 2. L'objectif est toujours de réduire la complexité du réseau tout en conservant une précision de classification jugée acceptable (de l'ordre de 99 %).

5.5.1 Protocole expérimental

Dans cette expérience, la taille de l'encodeur 1 est fixée à

$$H_1 = 60,$$

et l'on fait varier la taille de l'encodeur 2 selon

$$H_2 \in \{10,\ 20,\ 30,\ 40\}.$$

Pour chaque valeur de H_2, on applique exactement le même protocole que précédemment :

1. entraînement de AE1 sur les images brutes avec $H_1 = 60$ neurones cachés ;
2. encodage des images d'apprentissage par AE1 (60 caractéristiques) ;
3. entraînement de AE2 sur ces caractéristiques, avec H_2 neurones cachés ;
4. encodage de niveau 2 (dimension H_2) ;
5. entraînement d'une couche softmax sur ces représentations ;
6. empilement AE1+AE2+softmax et **fine-tuning global** sur le jeu d'apprentissage vectorisé ;
7. évaluation sur le jeu de test : accuracy globale, erreur, accuracy par classe, nombre total de paramètres et matrice de confusion.

5.5.2 Résultats globaux

Le tableau 3 synthétise les performances obtenues pour chaque taille d'encodeur 2.

H_2	Accuracy globale	Erreur	Nb paramètres
10	98,12 %	1,88 %	47 820
20	98,48 %	1,52 %	48 530
30	**99,04 %**	**0,96 %**	49 240
40	98,74 %	1,26 %	49 950

TABLE 3 – Résultats globaux pour l'architecture AE1+AE2+softmax avec fine-tuning, $H_1 = 60$ fixé et H_2 variable.

Les courbes d'accuracy, d'erreur et de nombre de paramètres en fonction de H_2 sont présentées figures 14 à 16.

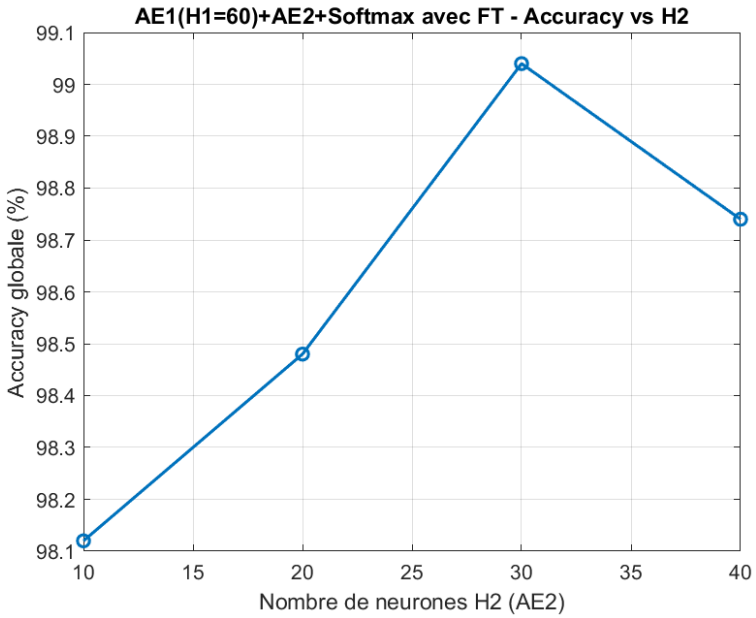

FIGURE 14 – Accuracy globale en fonction de H_2 pour AE1($H_1 = 60$)+AE2+softmax avec FT.

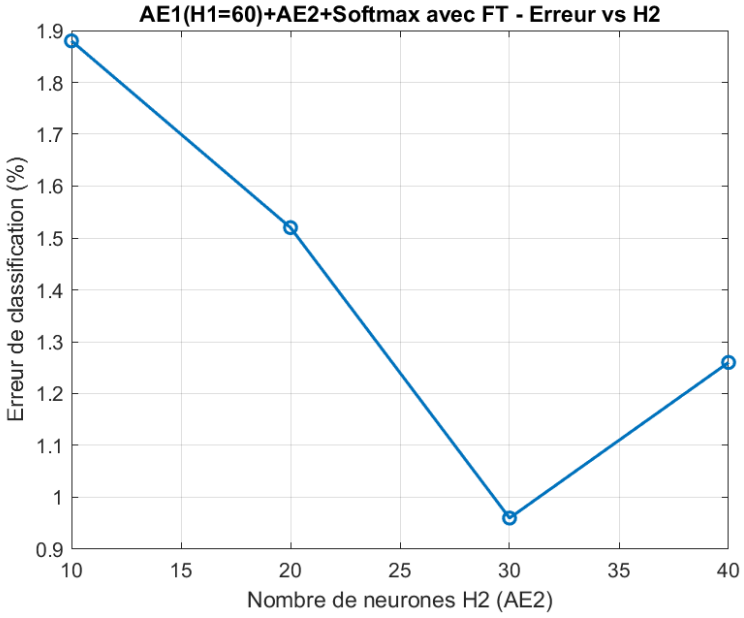

FIGURE 15 – Erreur de classification en fonction de H_2 ($H_1 = 60$).

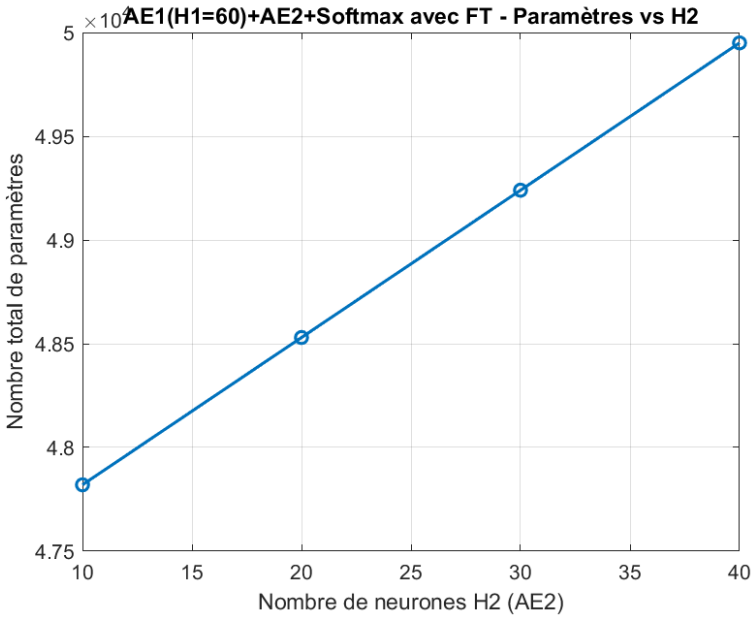

FIGURE 16 – Nombre total de paramètres en fonction de H_2 ($H_1 = 60$).

On observe que l'accuracy augmente lorsque H_2 passe de 10 à 30 neurones, puis diminue légèrement pour $H_2 = 40$. Le nombre de paramètres croît, lui, presque linéairement avec H_2, comme attendu.

5.5.3 Analyse des matrices de confusion

Les matrices de confusion correspondant à chaque valeur de H_2 sont illustrées figures 17 à 20.

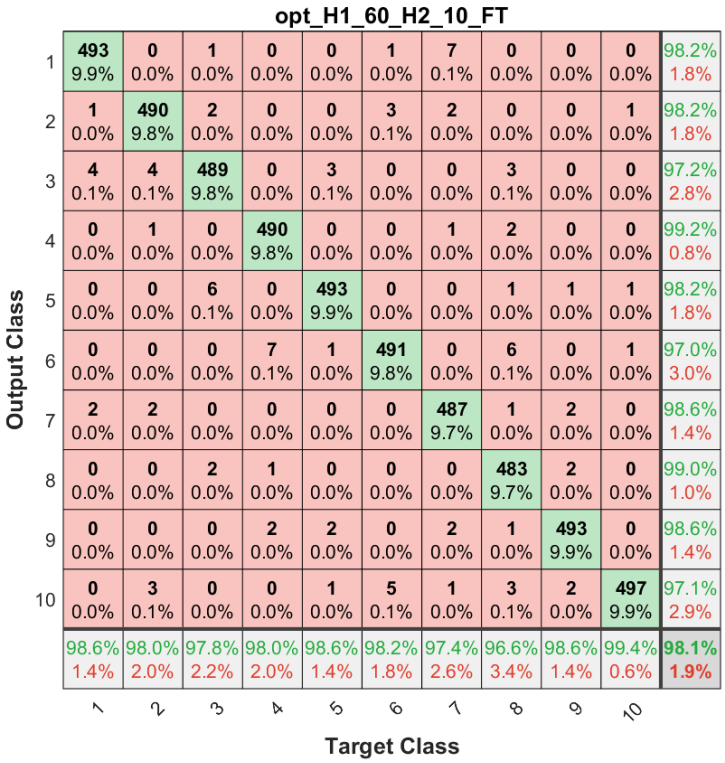

FIGURE 17 – Matrice de confusion pour $H_1 = 60$, $H_2 = 10$.

Pour $H_2 = 10$, la diagonale reste dominante et toutes les classes dépassent 97 % de bonne classification, mais l'accuracy globale est limitée à 98,12 %. Certaines classes (notamment 6 et 1–2–4) présentent encore des erreurs non négligeables, avec des accuracies autour de 96–98 %. L'espace latent de dimension 10 dans le deuxième autoencodeur semble donc trop comprimé pour capturer toutes les variations.

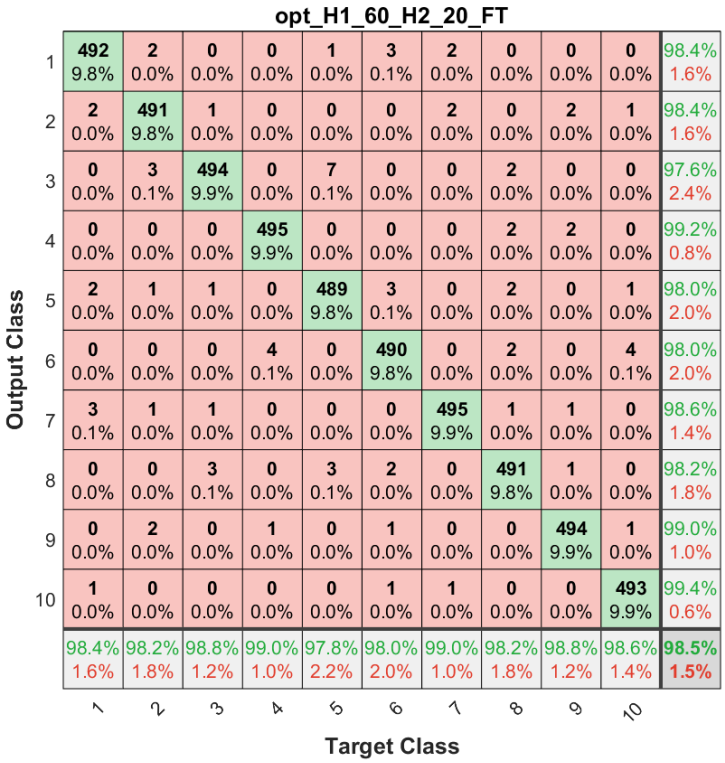

FIGURE 18 – Matrice de confusion pour $H_1 = 60$, $H_2 = 20$.

Avec $H_2 = 20$ (figure 18), l'accuracy globale monte à 98,5 %, et la plupart des classes se situent désormais entre 97,8 % et 99 %. La diagonale est plus marquée, même si quelques confusions subsistent pour certaines classes.

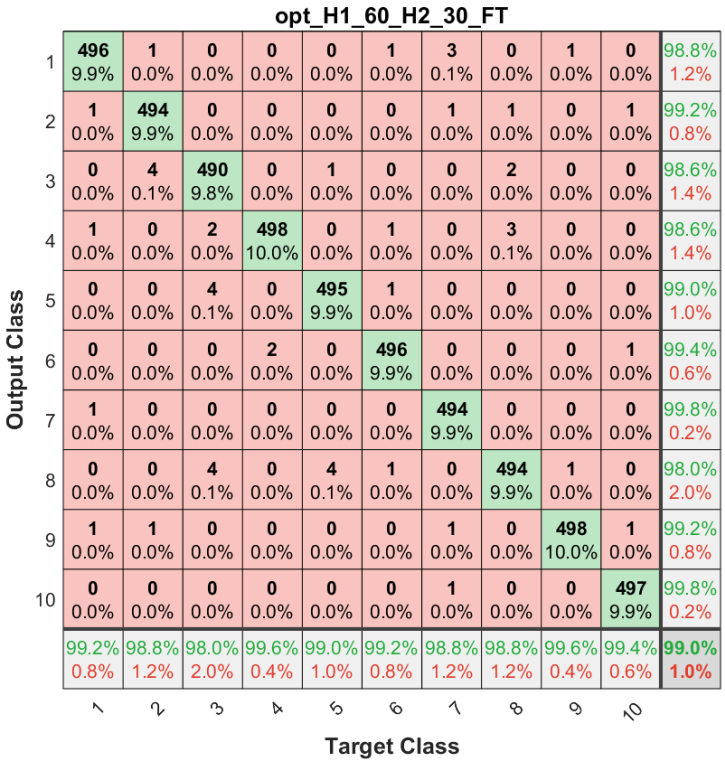

FIGURE 19 – Matrice de confusion pour $H_1 = 60$, $H_2 = 30$.

Le cas $H_2 = 30$ (figure 19) présente la meilleure performance globale : 99,0 % d'accuracy, avec des accuracies par classe comprises entre 98 % et 99,6 %. La diagonale est presque entièrement verte, et les erreurs résiduelles sont peu nombreuses et ne suivent pas de pattern de confusion particulier.

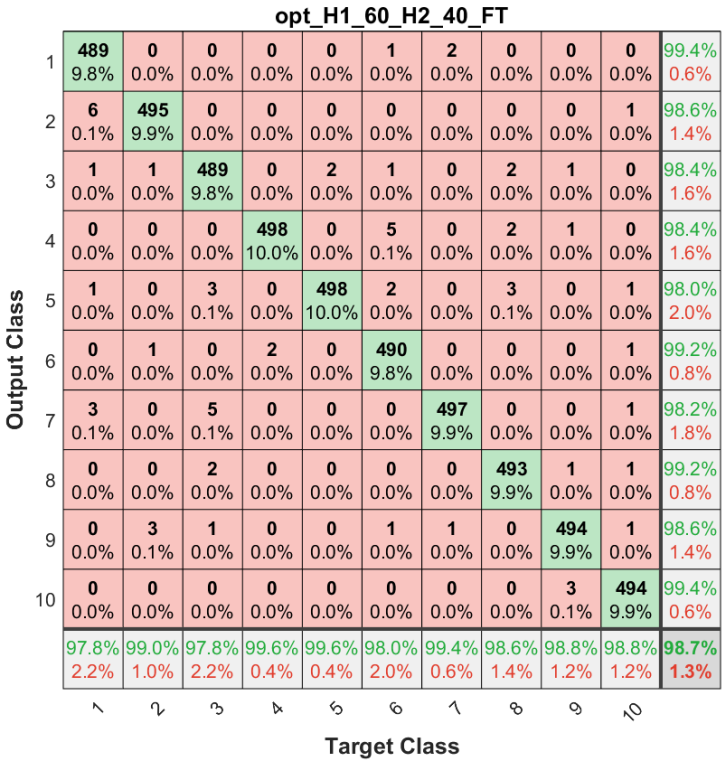

FIGURE 20 – Matrice de confusion pour $H_1 = 60$, $H_2 = 40$.

Pour $H_2 = 40$, la matrice de confusion reste de très bonne qualité (98,7 % d'accuracy globale), mais l'on constate un léger recul par rapport au cas $H_2 = 30$. Certaines classes redescendent autour de 98 %, sans gain notable sur les autres.

5.5.4 Compromis performance / complexité sur H_2

Les résultats montrent que :

— un **H trop petit** (10 neurones) limite la capacité de l'encodeur 2 : le réseau reste performant (98 %), mais n'atteint pas le niveau de précision visé (99 %), et certaines classes restent plus fragiles ;

— entre $H_2 = 20$ et $H_2 = 30$, l'accuracy progresse de 98,48 % à 99,04 % alors que le nombre de paramètres n'augmente que d'environ 700 paramètres (de 48 530 à 49 240) ;

— au-delà ($H_2 = 40$), la complexité continue d'augmenter (49 950 paramètres) sans amélioration de la précision, au contraire : l'accuracy globale redescend légèrement à 98,74 %.

Si l'on combine ces résultats avec ceux de la section précédente, on peut comparer les configurations suivantes pour $H_1 = 60$:

— $H_2 = 30$: 99,04 % d'accuracy, 49 240 paramètres ;

— $H_2 = 50$ (réseau de référence précédent) : 99,42 % d'accuracy, 50 660 paramètres.

L'augmentation de H_2 de 30 à 50 neurones permet donc un gain de l'ordre de $\approx 0{,}38$ point de pourcentage sur l'accuracy, au prix d'une hausse d'environ 1 400 paramètres (3 % de complexité supplémentaire). Selon les contraintes de l'application, on peut considérer que :

— si la priorité absolue est la **précision maximale**, la configuration $(H_1, H_2) = (60, 50)$ reste légèrement supérieure (99,42 %) ;

— si l'on cherche un **meilleur compromis performance / complexité**, la configuration $(H_1, H_2) = (60, 30)$ est très attractive : elle atteint déjà plus de 99 % d'accuracy avec un nombre de paramètres réduit par rapport au réseau de référence.

En résumé, pour ce jeu de données, on peut réduire de manière significative la taille de la deuxième couche cachée jusqu'à $H_2 \approx 30$ neurones tout en conservant une précision de classification supérieure à 99 %. En dessous de cette valeur, la perte de performance devient plus marquée, tandis qu'au-delà, l'augmentation de H_2 n'apporte plus de gain significatif au regard de la complexité supplémentaire.

6 Synthèse des résultats et réponses aux questions de recherche

Dans ce travail, nous avons étudié systématiquement l'influence de plusieurs choix d'architecture et de stratégie d'apprentissage pour un réseau à autoencodeurs empilés, appliqué à la classification d'images de chiffres manuscrits. Les expériences menées permettent de répondre aux trois questions suivantes.

1. Effet du *fine-tuning* global sur les performances.
2. Effet de l'ajout d'un deuxième autoencodeur sur le rapport performance/coût calculatoire.
3. Jusqu'à quel point peut-on optimiser l'architecture tout en conservant une précision acceptable ?

6.1 Question 1 – Effet du fine-tuning global sur les performances

Nous avons d'abord comparé quatre configurations de référence :

— **AE1 + softmax, sans fine-tuning** : accuracy $\approx 80.4\,\%$, erreur $\approx 19.6\,\%$, pour 79 510 paramètres.
— **AE1 + softmax, avec fine-tuning** : accuracy $\approx 99.0\,\%$, soit un gain de +18,6 points, pour le même nombre de paramètres.
— **AE1 + AE2 + softmax, sans fine-tuning** : accuracy $\approx 70.5\,\%$, malgré une architecture plus complexe (84 060 paramètres).
— **AE1 + AE2 + softmax, avec fine-tuning** : accuracy $\approx 99.4\,\%$, soit un gain de +29 points par rapport à la version non fine-tunée.

Les matrices de confusion montrent que :

— sans fine-tuning, les réseaux restent loin d'une diagonale parfaite, avec de nombreuses confusions entre chiffres visuellement proches ;
— après fine-tuning, les configurations atteignent des matrices quasiment diagonales : chaque chiffre est reconnu correctement dans plus de 98–99 % des cas.

Conclusion Question 1. Le fine-tuning global est absolument déterminant : il transforme un réseau pré-entraîné "brut" (70–80 % d'accuracy) en un classificateur très performant ($\approx 99\,\%$). Il est encore plus crucial lorsque l'architecture est profonde (AE1+AE2), où le pré-entraînement seul ne suffit pas à exploiter correctement la capacité du réseau.

6.2 Question 2 – Effet de l'ajout d'un deuxième autoencodeur (performance vs coût calculatoire)

La seconde question porte sur l'intérêt réel d'ajouter un deuxième autoencodeur :

> *« Est-ce que l'ajout d'un deuxième autoencodeur améliore suffisamment les performances pour justifier le surcoût en termes de complexité ? »*

Les résultats principaux sont les suivants :

— **Sans fine-tuning**, l'ajout de AE2 est défavorable :
  — AE1 + softmax : $\approx 80.4\,\%$ d'accuracy, 79 510 paramètres ;
  — AE1 + AE2 + softmax : $\approx 70.5\,\%$ d'accuracy, 84 060 paramètres.
  On augmente donc la complexité tout en dégradant les performances.

— **Avec fine-tuning**, le deuxième autoencodeur apporte un léger gain :
  — AE1 + softmax FT : $\approx 99.0\,\%$ d'accuracy, 79 510 paramètres ;
  — AE1 + AE2 + softmax FT : $\approx 99.4\,\%$ d'accuracy, 84 060 paramètres.
  On gagne $\approx 0{,}4$ point d'accuracy pour $\approx 6\,\%$ de paramètres en plus.

Les matrices de confusion des deux architectures fine-tunées sont très similaires : dans les deux cas, les erreurs sont extrêmement rares et dispersées. Le gain dû à AE2 est donc réel mais marginal à ce niveau de précision.

Conclusion Question 2.
— Sans fine-tuning, le deuxième autoencodeur a aucun utilité (plus de paramètres pour moins de précision).
— Avec fine-tuning, il apporte un **petit** gain de performance ($\approx 0{,}4$ point) au prix d'une augmentation modérée de la complexité ($\approx 6\,\%$ de paramètres en plus).

Pour ce jeu de données relativement simple, un seul autoencodeur bien fine-tuné offre déjà un excellent compromis performance / coût. Le deuxième autoencodeur n'est justifié que si l'on cherche à maximiser la précision jusqu'aux derniers dixièmes de pourcent, dans un contexte où le coût calculatoire reste acceptable.

6.3 Question 3 – Jusqu'à quel point optimiser l'architecture tout en gardant une précision acceptable ?

Enfin, nous avons cherché à optimiser plus finement la structure interne du réseau à deux autoencodeurs avec fine-tuning, en jouant sur les tailles des couches cachées H_1 (AE1) et H_2 (AE2).

1) Optimisation de H_1 (avec $H_2 = 50$ fixé)

En faisant varier $H_1 \in \{20, 40, 60, 80, 100\}$ avec $H_2 = 50$, nous obtenons :
— $H_1 = 20$: $96.4\,\%$ d'accuracy, 17 260 paramètres (réseau sous-dimensionné) ;
— $H_1 = 40$: $99.0\,\%$ d'accuracy, 33 960 paramètres ;
— $H_1 = 60$: $99.42\,\%$ **d'accuracy**, 50 660 paramètres ;
— $H_1 = 80$: $99.18\,\%$ d'accuracy, 67 360 paramètres ;
— $H_1 = 100$: $99.40\,\%$ d'accuracy, 84 060 paramètres (réseau de référence).

Les matrices de confusion montrent un **plateau de performance** dès $H_1 \approx 40$: pour $H_1 = 40, 60, 100$, la diagonale est presque parfaite. La meilleure accuracy est obtenue pour $H_1 = 60$, mais la différence avec $H_1 = 100$ est négligeable (99,42 % vs 99,40 %) alors que le nombre de paramètres est réduit d'environ 40 %.

2) Optimisation de H_2 (avec $H_1 = 60$ fixé)

En fixant $H_1 = 60$ et en faisant varier $H_2 \in \{10, 20, 30, 40\}$, on obtient :
— $H_2 = 10$: $98.12\,\%$ d'accuracy, 47 820 paramètres (couche trop comprimée) ;
— $H_2 = 20$: $98.48\,\%$ d'accuracy, 48 530 paramètres ;
— $H_2 = 30$: $99.04\,\%$ **d'accuracy**, 49 240 paramètres ;
— $H_2 = 40$: $98.74\,\%$ d'accuracy, 49 950 paramètres.

L'accuracy augmente donc jusqu'à $H_2 \approx 30$, puis diminue légèrement pour $H_2 = 40$, tandis que la complexité continue de croître.

3) Synthèse de l'optimisation

En combinant les deux études, on peut comparer :
— **Réseau de référence** : $(H_1, H_2) = (100, 50)$, 84 060 paramètres, $99.40\,\%$ d'accuracy ;
— **Réseau optimisé compact** : $(H_1, H_2) = (60, 30)$, 49 240 paramètres, $99.04\,\%$ d'accuracy.

On réduit ainsi le nombre de paramètres d'environ 40 % pour une perte de précision très faible ($\approx$ 0,36 point). Visuellement, les matrices de confusion restent quasi diagonales dans les deux cas.

Conclusion Question 3. Pour ce problème, on peut **fortement compresser l'architecture** (jusqu'à $(H_1, H_2) \approx (60, 30)$) tout en conservant une précision supérieure à 99 %. En dessous (par exemple $H_1 = 20$ ou $H_2 = 10$), la perte de performance devient significative ; au-dessus (couches plus grandes), on n'observe plus de gain mesurable malgré l'augmentation du coût calculatoire.

Conclusion générale

En résumé :

— le **fine-tuning global** est indispensable pour exploiter correctement les autoencodeurs, surtout lorsqu'ils sont empilés ;

— l'**ajout d'un deuxième autoencodeur** n'est intéressant que s'il est couplé au fine-tuning : il apporte un léger gain de précision, mais son intérêt doit être évalué au regard du coût calculatoire ;

— une étude systématique des tailles de couches montre qu'il existe une **zone optimale** ($H_1 \approx$ 40–60, $H_2 \approx 30$) où le réseau atteint déjà plus de 99 % d'accuracy avec un nombre de paramètres modéré.