Université Paris-Saclay
UVSQ - Université de Versailles Saint-Quentin-en-Yvelines

Unité d'Enseignement : Apprentissage et Réseaux de Neurones

TP 2 : Deep Learning avec Keras et Manifold Untangling

Prise en main de la librairie Keras et analyse des espaces de représentation appris par les modèles profonds

Étudiant :
DEBACHE Mounsef

Enseignants :
M. Arnaud Breloy
M. Olivier Pons

Février 2026

Exercice 1 – Régression Logistique avec Keras

Modèle reconnu. Le réseau défini par une couche `Dense(10)` suivie d'une activation `softmax` correspond exactement à une **régression logistique multi-classes** (aussi appelée **softmax regression**). En effet, pour une image MNIST vectorisée $x \in \mathbb{R}^{784}$, la couche dense calcule des scores

$$s = xW + b \in \mathbb{R}^{10},$$

puis la fonction softmax fournit une distribution de probabilités sur les classes :

$$\hat{y}_c = \frac{\exp(s_c)}{\sum_{k=1}^{10} \exp(s_k)}, \quad c \in \{1, \dots, 10\}.$$

Il s'agit donc du même modèle que celui implémenté manuellement au TP1 (classifieur linéaire + softmax + entropie croisée).

Nombre de paramètres. La couche `Dense(10)` (entrée de dimension 784, sortie de dimension 10) contient :

$$\underbrace{784 \times 10}_{\text{poids}} + \underbrace{10}_{\text{biais}} = 7840 + 10 = \mathbf{7850}$$

paramètres entraînables. La couche `Activation(softmax)` n'ajoute aucun paramètre.

Vérification avec `summary()`. La méthode `model.summary()` confirme ce résultat : la couche `fc1 (Dense)` possède **7 850** paramètres, et le total du réseau est **7 850** paramètres entraînables.

Apprentissage et performances. Le modèle est entraîné avec la perte d'entropie croisée (`categorical_cr` et une descente de gradient stochastique (`SGD`) avec un pas d'apprentissage $\eta = 0.1$, sur 20 époques et une taille de batch de 100. Sur la base de test, on obtient :

$$\text{loss} = 0.2716, \qquad \text{accuracy} = 92.44\%.$$

Comparaison avec le TP1 (implémentation manuelle). Les performances obtenues sont du même ordre de grandeur que celles du TP1 (environ 92% pour la régression logistique), ce qui est attendu puisque le modèle et la fonction de coût sont identiques. Les différences éventuelles proviennent principalement de l'initialisation des poids, de l'ordre des mini-batchs et de détails d'implémentation (stabilité numérique, etc.).

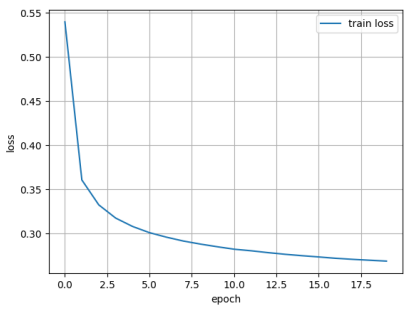


FIGURE 1 – Évolution de la fonction de coût (loss) au cours des époques pour la régression logistique avec Keras.

Exercice 2 – Perceptron (MLP) avec Keras

Architecture du modèle. On enrichit la régression logistique de l'exercice 1 en ajoutant une couche cachée entièrement connectée de $L = 100$ neurones, suivie d'une non-linéarité sigmoïde. Le réseau obtenu est donc un **Perceptron Multi-Couches (MLP)** à **une couche cachée**. Pour une image MNIST vectorisée $x \in \mathbb{R}^{784}$, le modèle s'écrit :

$$u = xW_1 + b_1 \in \mathbb{R}^{100}, \qquad h = \sigma(u) \in \mathbb{R}^{100},$$

$$v = hW_2 + b_2 \in \mathbb{R}^{10}, \qquad \hat{y} = \mathrm{softmax}(v) \in \mathbb{R}^{10},$$

où $\sigma(\cdot)$ est la sigmoïde et $\hat{y}$ est la distribution de probabilité sur les 10 classes.

Nombre de paramètres du MLP (justification). Le modèle contient deux couches `Dense` (les couches d'activation `sigmoid` et `softmax` n'ajoutent aucun paramètre).

— **Couche cachée `fc1 : Dense(100)` (entrée 784)** :
poids $W_1 \in \mathbb{R}^{784\times100}$ et biais $b_1 \in \mathbb{R}^{100}$, soit

$$784 \times 100 + 100 = 78\,400 + 100 = \mathbf{78\,500}$$

paramètres.

— **Couche de sortie `fc2 : Dense(10)` (entrée 100)** :
poids $W_2 \in \mathbb{R}^{100\times10}$ et biais $b_2 \in \mathbb{R}^{10}$, soit

$$100 \times 10 + 10 = 1\,000 + 10 = \mathbf{1\,010}$$

paramètres.

Au total, le réseau possède :

$$\mathbf{78\,500 + 1\,010 = 79\,510}$$

paramètres entraînables.

Vérification avec `summary()`. La méthode `model.summary()` confirme exactement ce calcul : `fc1` possède **78 500** paramètres, `fc2` possède **1 010** paramètres, et le total du modèle est **79 510** paramètres entraînables.

Initialisation des paramètres (Keras). L'inspection des couches montre que Keras initialise :

— les noyaux (`kernel`) avec **GlorotUniform** (initialisation de Xavier uniforme),
— les biais (`bias`) avec **Zeros**.

Cette initialisation de type Xavier vise à stabiliser la variance des activations et des gradients au travers des couches, et facilite la convergence lors de l'apprentissage.

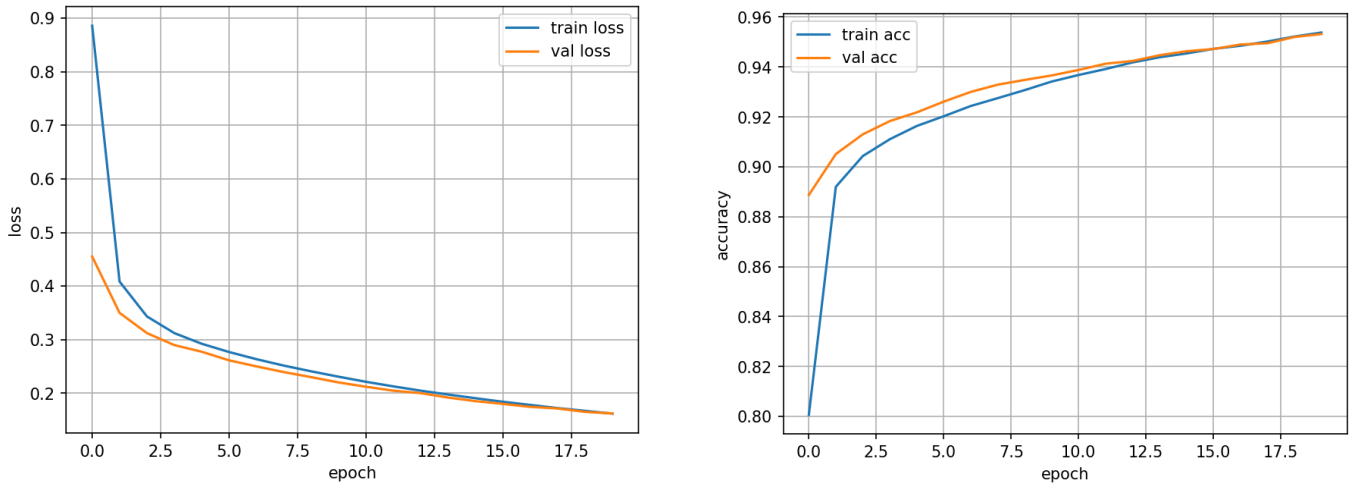
Apprentissage et performances. Le modèle est entraîné avec :

— la perte d'entropie croisée : `categorical_crossentropy`,
— l'optimiseur `SGD` (descente de gradient stochastique),
— 20 époques, avec `batch_size`= 100.

La dynamique d'apprentissage montre une amélioration continue des performances (accuracy) et une diminution de la fonction de coût (loss). Sur la base de test, on obtient :

$$\text{loss} = 0.1623, \qquad \text{accuracy} = 95.32\%.$$

Comparaison avec la séance précédente (implémentation manuelle). Le MLP apprend une frontière de décision **non linéaire** grâce à la couche cachée sigmoïde, ce qui améliore la performance par rapport à la régression logistique (modèle linéaire) de l'exercice 1. On observe ici une accuracy de **95.32%**, supérieure aux performances typiques de la régression logistique sur MNIST (environ 92%), mais encore inférieure à l'ordre de grandeur maximal attendu pour un MLP bien optimisé (souvent proche de 98% selon le choix des hyperparamètres, du nombre d'époques, et de l'optimiseur). Ces résultats confirment que l'ajout d'une couche cachée permet de mieux modéliser la complexité des données MNIST.



(a) Loss (train/val)

(b) Accuracy (train/val)

FIGURE 2 – Courbes d'apprentissage du MLP Keras (1 couche cachée de 100 neurones) : loss et accuracy.

Exercice 3 – Réseau de neurones convolutif (ConvNet) avec Keras

Objectif. L'objectif de cet exercice est d'étendre le perceptron (MLP) de l'exercice 2 vers un **réseau de neurones convolutif profond** (ConvNet), de type *LeNet5*, afin d'exploiter la structure spatiale des images MNIST.

Pré-traitement des données (format tenseur). Contrairement aux réseaux entièrement connectés qui utilisent des vecteurs de taille 784, un ConvNet manipule directement des tenseurs image. Chaque image MNIST est donc reformatée en tenseur $28 \times 28 \times 1$ (un seul canal en niveau de gris) :

$$x_{\text{train}} \in \mathbb{R}^{N \times 28 \times 28 \times 1}, \quad x_{\text{test}} \in \mathbb{R}^{N_{\text{test}} \times 28 \times 28 \times 1},$$

avec normalisation des pixels dans $[0, 1]$.

Architecture du ConvNet (type LeNet). Le modèle implémenté suit l'architecture demandée :

- **Bloc 1 :** Convolution 5×5 avec 16 filtres, suivie de `ReLU`, puis `MaxPooling` 2×2.
- **Bloc 2 :** Convolution 5×5 avec 32 filtres, suivie de `ReLU`, puis `MaxPooling` 2×2.
- **Classification :** `Flatten` $\rightarrow$ Dense(100) + `sigmoid` $\rightarrow$ Dense(10) + `softmax`.

Les dimensions successives (issues du `summary()` Keras) sont :

$$(28, 28, 1) \xrightarrow{\text{Conv}(5\times5,16)} (24, 24, 16) \xrightarrow{\text{Pool}(2\times2)} (12, 12, 16)$$
$$\xrightarrow{\text{Conv}(5\times5,32)} (8, 8, 32) \xrightarrow{\text{Pool}(2\times2)} (4, 4, 32) \xrightarrow{\text{Flatten}} 512.$$

Nombre de paramètres (calcul et vérification). Le `summary()` Keras indique un total de **65 558 paramètres entraînables**. On peut vérifier ce total en détaillant couche par couche :

— **Convolution 1 :** 16 filtres de taille 5×5 sur 1 canal
Chaque filtre a $5 \times 5 \times 1 = 25$ poids, et un biais.
Donc :

$$16 \times (25 + 1) = 16 \times 26 = \mathbf{416}.$$

— **Convolution 2 :** 32 filtres de taille 5×5 sur 16 canaux
Chaque filtre a $5 \times 5 \times 16 = 400$ poids, et un biais.
Donc :

$$32 \times (400 + 1) = 32 \times 401 = \mathbf{12\,832}.$$

— **Dense 1 :** entrée 512 (après `Flatten`) vers 100 neurones

$$512 \times 100 + 100 = 51\,200 + 100 = \mathbf{51\,300}.$$

— **Dense 2 :** entrée 100 vers 10 classes

$$100 \times 10 + 10 = 1\,000 + 10 = \mathbf{1\,010}.$$

En sommant :

$$416 + 12\,832 + 51\,300 + 1\,010 = \mathbf{65\,558},$$

ce qui correspond exactement au total affiché par Keras.

Apprentissage (compile & fit). Le réseau est entraîné avec :

— perte : `categorical_crossentropy`,
— optimiseur : `SGD` avec `learning_rate` = 0.1,
— `batch_size` = 100,
— 20 époques,
— validation effectuée sur la base de test (val_loss, val_accuracy).

Performances obtenues. Après apprentissage, l'évaluation sur la base de test donne :

$$\text{Test loss} = 0.0319, \qquad \text{Test accuracy} = 99.00\%.$$

Ce résultat est cohérent avec l'objectif attendu (de l'ordre de 99%), et montre l'efficacité des architectures convolutives pour MNIST.

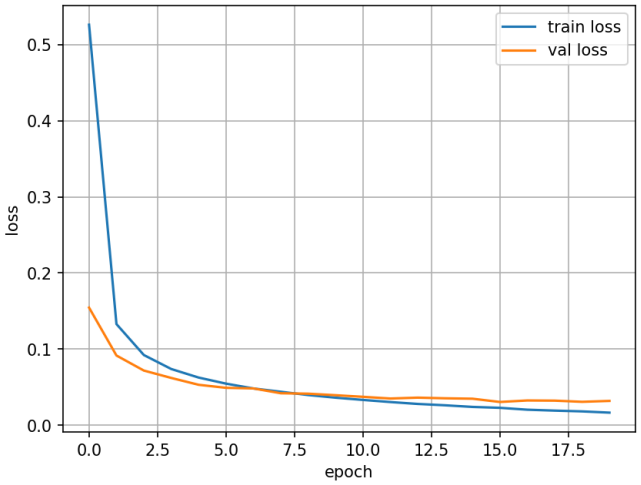
Temps d'une époque (CPU). Un callback de mesure de temps par époque a été utilisé. Le temps moyen observé est :

$$\mathbf{14.375}\ \text{s/epoch}$$

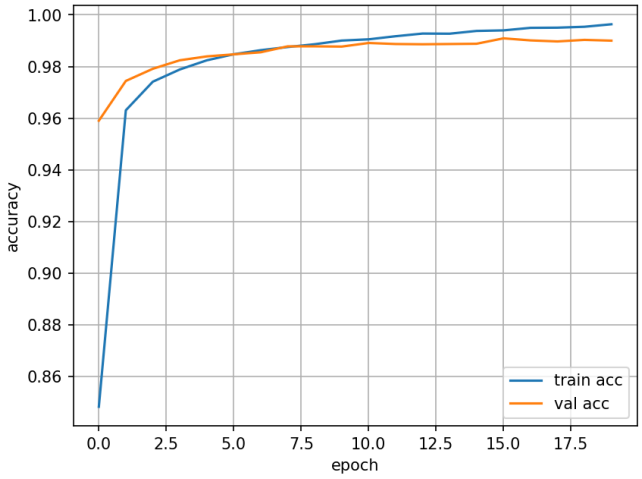
sur la machine de test en exécution CPU. On remarque que ce temps est significativement plus élevé que pour les modèles entièrement connectés (régression logistique, MLP) car les convolutions réalisent un nombre beaucoup plus important d'opérations.

Apprentissage sur GPU (discussion). En théorie, l'entraînement sur GPU doit réduire fortement le temps par époque car les convolutions sont particulièrement bien accélérées sur carte graphique. Dans le contexte expérimental, si le GPU est correctement activé (CUDA + drivers), on s'attend à un gain de vitesse significatif (temps/époque bien inférieur). La comparaison se fait en mesurant le temps moyen par époque sur CPU et sur GPU, puis en comparant les deux valeurs.

Figures : loss et accuracy. Les courbes d'apprentissage suivantes illustrent la convergence du modèle : diminution progressive de la loss et augmentation de l'accuracy jusqu'à environ 99% sur l'ensemble de test.



(a) Loss (train et validation)



(b) Accuracy (train et validation)

FIGURE 3 – Courbes d'apprentissage du ConvNet (type LeNet) sur MNIST.

Sauvegarde du modèle. Le modèle entraîné peut être sauvegardé afin d'être rechargé ultérieurement sans ré-apprentissage. Une méthode moderne consiste à sauvegarder au format `.keras` (recommandé), ou bien en sérialisant l'architecture (JSON/YAML) et les poids (HDF5).

Conclusion. Le ConvNet de type LeNet atteint **99.00%** de précision sur MNIST avec une loss de **0.0319**. Les convolutions permettent d'extraire automatiquement des caractéristiques locales (bords, motifs) et offrent une meilleure généralisation que les architectures entièrement connectées. Le coût computationnel est plus élevé (environ **14.4 s par époque** sur CPU), ce qui justifie l'intérêt d'un entraînement sur GPU.

Exercice 4 – Visualisation 2D (t-SNE) et séparabilité des classes

Objectif : Manifold untangling

L'objectif de cet exercice est d'illustrer le phénomène de *manifold untangling* : un bon espace de représentation doit rendre les classes plus facilement séparables. Pour cela, on projette en 2D des données de test MNIST (initialement dans $\mathbb{R}^{784}$) afin de visualiser la structure des classes et d'évaluer leur séparabilité.

Réduction de dimension par t-SNE

La méthode **t-SNE** (*t-Distributed Stochastic Neighbor Embedding*) est une réduction de dimension **non linéaire** visant à conserver les voisinages locaux : deux points proches en dimension élevée doivent rester proches après projection en 2D.

Dans notre implémentation, on utilise la classe `TSNE` de `sklearn.manifold` avec les paramètres recommandés :

— `n_components = 2` (projection en 2D),
— `init = 'pca'` (initialisation par ACP, stabilise et accélère),
— `perplexity = 30` (lié au nombre effectif de voisins),
— `verbose = 2` (affichage de la progression),
— `random_state = 42` (reproductibilité).

L'apprentissage t-SNE sur l'ensemble des 10 000 images de test dure environ :

$$T_{\text{t-SNE}} \approx 36.61 \text{ s.}$$

Les logs indiquent une divergence de KL finale :

$$\text{KL divergence} \approx 1.7791.$$

Cette valeur est cohérente avec une bonne organisation des clusters dans l'espace 2D.

Métriques de séparabilité des classes

Afin d'analyser quantitativement et visuellement la séparabilité, trois métriques complémentaires ont été utilisées.

1) Enveloppes convexes (`ConvexHull`)

Pour chaque classe $c \in \{0, \ldots, 9\}$, on calcule l'enveloppe convexe des points projetés appartenant à cette classe. Cette enveloppe représente une approximation géométrique simple de la région occupée par une classe.

Lien avec la séparabilité : si les enveloppes convexes des classes sont fortement disjointes, cela indique une bonne séparabilité globale. Au contraire, de grandes zones de recouvrement entre enveloppes traduisent un mélange des classes.

2) Ellipses d'approximation (GMM à 1 composante)

Pour chaque classe, on ajuste un modèle gaussien (via `GaussianMixture` avec `n_components=1`) et on visualise l'ellipse correspondant à la covariance estimée.

Lien avec la séparabilité : si les ellipses sont compactes et peu recouvrantes, les classes sont bien regroupées. Si les ellipses se superposent fortement, la séparation est faible.

3) Neighborhood Hit (NH)

La métrique **Neighborhood Hit (NH)** mesure, pour chaque point, la proportion de ses k plus proches voisins (ici $k = 6$) appartenant à la **même classe**. La métrique globale est ensuite moyennée sur l'ensemble des points :

$$NH = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{k} \sum_{j \in \mathcal{N}_k(i)} \mathbb{1}[y_j = y_i].$$

Interprétation :
— $NH \approx 1$: très forte cohérence locale (clusters bien séparés),
— NH faible : les voisins proches sont souvent d'autres classes (mélange local).

Question : lien et différences entre les 3 métriques

Les trois métriques sont liées à la **séparabilité** car elles mesurent (directement ou indirectement) la façon dont les classes occupent l'espace 2D :

- **Convex Hull** : mesure une **séparabilité globale** (forme externe, sensible aux outliers).
- **Ellipse GMM** : mesure une **séparabilité statistique** (dispersion moyenne via covariance), plus robuste qu'un hull.
- **NH** : mesure une **séparabilité locale**, centrée sur les voisinages et l'organisation fine des points.

Ainsi :

$$\textit{Hull} = \textit{global}, \quad \textit{Ellipse} = \textit{global/statistique}, \quad \textit{NH} = \textit{local}.$$

Comparaison t-SNE vs PCA (ACP)

On compare ensuite t-SNE à une réduction linéaire **PCA** (ACP) avec :

$$\texttt{PCA}(n\_components = 2).$$

Résultats : Neighborhood Hit (NH)

Les valeurs mesurées sont :

PCA (2D).

$$NH_{\text{PCA}} = 0.3849 \quad \Rightarrow \quad 38.49\%.$$

Scores NH par classe (PCA) :

$$[0.569,\ 0.879,\ 0.248,\ 0.389,\ 0.322,\ 0.185,\ 0.280,\ 0.361,\ 0.231,\ 0.297].$$

On observe que certaines classes (ex. **1**) ont un NH plus élevé (forme plus simple et distincte), mais la majorité des classes ont un NH faible, ce qui indique un fort mélange local.

t-SNE (2D).

$$NH_{\text{t-SNE}} = 0.9327 \quad \Rightarrow \quad 93.27\%.$$

Scores NH par classe (t-SNE) :

$$[0.971,\ 0.974,\ 0.930,\ 0.920,\ 0.928,\ 0.913,\ 0.970,\ 0.912,\ 0.913,\ 0.891].$$

Toutes les classes présentent un NH élevé : les points ont majoritairement des voisins de même classe, ce qui confirme une excellente séparation locale.

Analyse visuelle des figures

Projection PCA. La figure PCA montre une **structure très chevauchée** : les nuages de points des classes se recouvrent fortement, les enveloppes convexes se superposent presque toutes, et les ellipses GMM se mélangent au centre. Cela explique le NH faible (38.49%).

Projection t-SNE. La figure t-SNE montre des **clusters distincts** (groupes bien séparés), avec des enveloppes convexes bien localisées et des ellipses peu recouvrantes. On note un **léger chevauchement résiduel** entre certaines classes (par exemple entre 9 et 4 selon l'observation), mais il reste limité, ce qui correspond à un NH global très élevé (93.27%).

Conclusion : que peut-on en conclure ?

— **PCA** étant une méthode **linéaire**, elle conserve surtout la variance globale des données mais ne parvient pas à démêler les structures non linéaires des chiffres manuscrits : les classes restent largement entremêlées en 2D.

— **t-SNE** est **non linéaire** et optimise la préservation des voisinages locaux : il produit une projection où les classes forment des clusters nettement séparés, ce qui traduit le *manifold untangling* de manière beaucoup plus visible.

— Les métriques quantitatives confirment ce constat :

$$NH_{\text{t-SNE}} \gg NH_{\text{PCA}} \quad (93.27\% \text{ vs } 38.49\%).$$

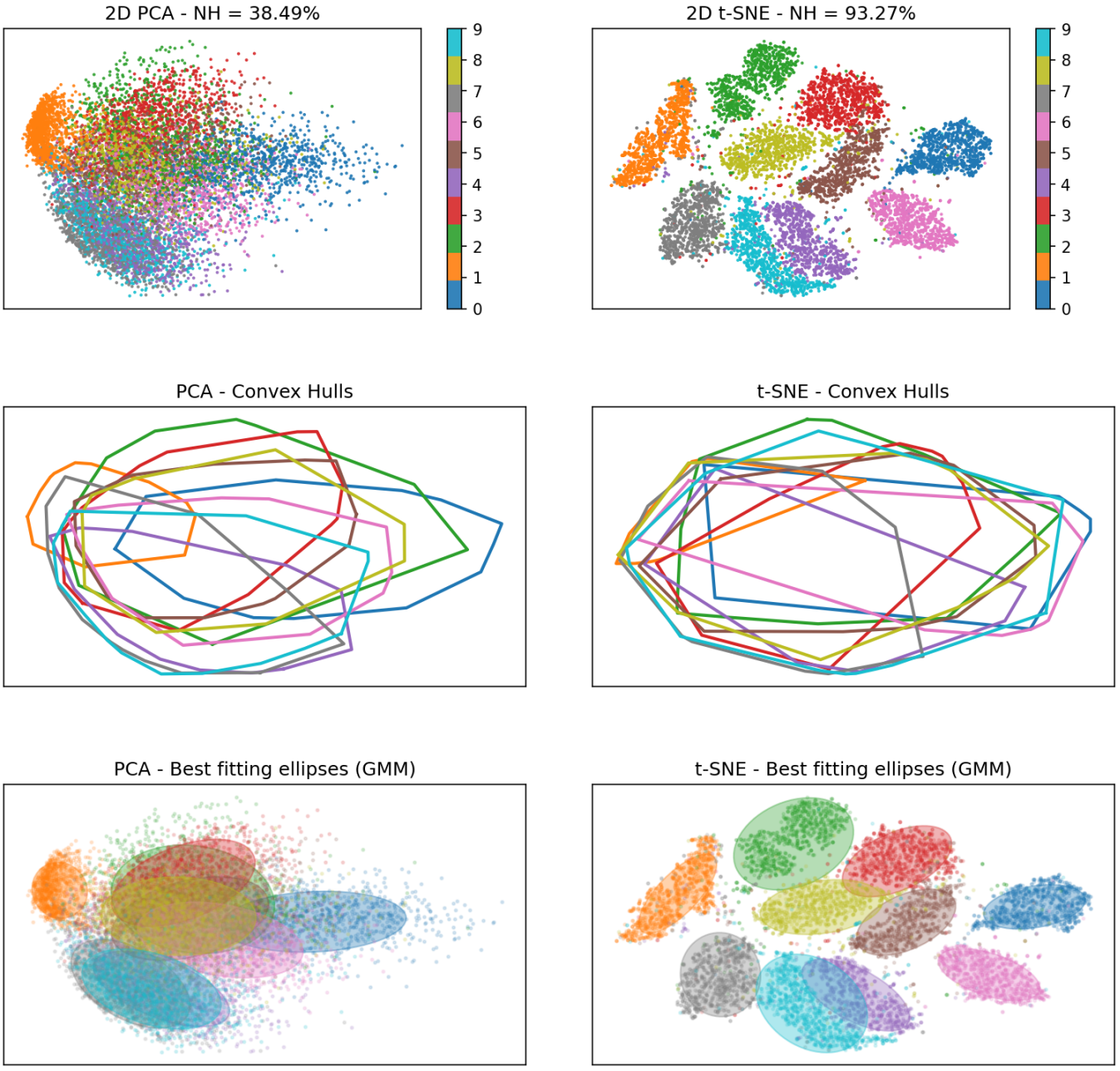


FIGURE 4 – Projection PCA en 2D

FIGURE 5 – Projection t-SNE en 2D

Exercice 5 – Visualisation des représentations internes (Manifold Untangling)

Objectif. L'objectif de cet exercice est de **visualiser l'effet de *manifold untangling*** permis par les réseaux de neurones profonds : autrement dit, vérifier si les classes (chiffres) deviennent **plus séparables** dans un espace de représentation interne appris par le modèle. Pour cela, on projette en 2D, via **t-SNE**, les représentations internes (latentes) d'un **MLP** et d'un **CNN de type LeNet**, puis on analyse la séparation obtenue avec des **métriques géométriques** (enveloppes convexes, ellipses GMM) et la métrique **Neighborhood Hit (NH)**.

Méthode générale. Soit un modèle f entraîné sur MNIST. On extrait une représentation interne $h(x)$ pour chaque image x de test :

$$h(x) \in \mathbb{R}^d,$$

puis on calcule une projection 2D par t-SNE :

$$z(x) = \text{t-SNE}(h(x)) \in \mathbb{R}^2.$$

Enfin, on visualise $z(x)$ coloré par la classe y et on calcule des métriques de séparabilité.

Paramètres t-SNE. Conformément à l'énoncé, on utilise :

$$n\_components = 2, \quad init = \texttt{pca}, \quad perplexity = 30, \quad random\_state = 42.$$

Ces choix visent à conserver la structure locale (voisinage) tout en facilitant la convergence (init par PCA).

1) Représentations internes du MLP (couche cachée)

Extraction de la couche cachée. Le MLP possède une couche cachée de dimension 100. On extrait donc :

$$h_{\text{MLP}}(x) \in \mathbb{R}^{100}.$$

La projection t-SNE donne :

$$z_{\text{MLP}}(x) \in \mathbb{R}^2.$$

Résultat (NH). La métrique **Neighborhood Hit** (avec $k = 6$ voisins) obtenue sur les représentations cachées du MLP est :

$$\text{NH}_{\text{MLP}} = 94.23\%.$$

Par classe (0 à 9) :

$$[0.974,\ 0.981,\ 0.935,\ 0.919,\ 0.949,\ 0.904,\ 0.962,\ 0.935,\ 0.937,\ 0.921].$$

Interprétation. On observe une **bonne séparation globale** des classes dans l'espace latent : les clusters sont bien individualisés, avec quelques **zones de contact** entre classes visuellement proches (ex. classes pouvant se ressembler selon l'écriture manuscrite). Cela confirme que le MLP réalise un **dépliage partiel** du manifold : la couche cachée apprend une représentation plus discriminante que l'espace pixel.

2) Représentations internes du CNN (espace latent)

Extraction de la représentation latente. Pour le CNN (architecture type LeNet), on extrait une représentation interne *après les blocs convolutifs* (après `Flatten` ou juste avant la couche de classification). On obtient :

$$h_{\text{CNN}}(x) \in \mathbb{R}^{d_{\text{CNN}}}.$$

Puis on projette par t-SNE :

$$z_{\text{CNN}}(x) \in \mathbb{R}^2.$$

Résultat (NH). La métrique NH obtenue sur les représentations latentes du CNN est :

$$\text{NH}_{\text{CNN}} = 96.84\%.$$

Par classe :

$$[0.986,\ 0.987,\ 0.971,\ 0.956,\ 0.966,\ 0.951,\ 0.983,\ 0.962,\ 0.978,\ 0.942].$$

Interprétation. Comparé au MLP, le CNN produit des **clusters plus compacts et plus séparés**. Cela s'explique par la nature des **couches convolutionnelles** : grâce aux **filtres (kernels)** et au **pooling**, le CNN apprend des caractéristiques locales (bords, traits, motifs) plus robustes aux petites translations et variations d'écriture. En conséquence, les représentations internes du CNN sont généralement **plus invariantes** et **plus discriminantes**, ce qui améliore la séparabilité inter-classes dans l'espace latent.

3) Comparaison MLP vs CNN et conclusion sur le Manifold Untangling

Comparaison quantitative (NH). On résume les résultats par la métrique NH :

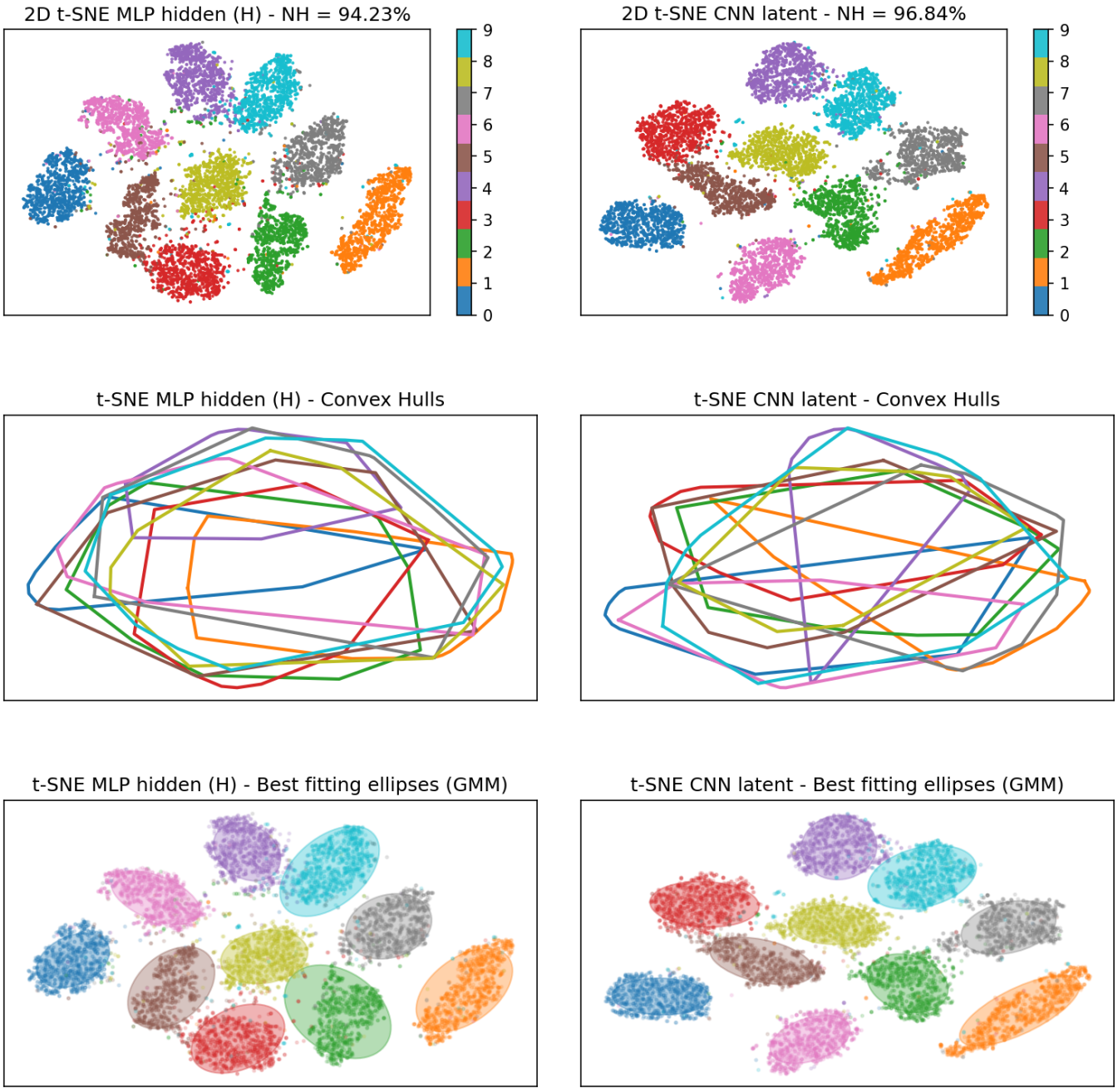
Représentation	NH global	Commentaire
MLP (couche cachée 100)	94.23%	clusters bien séparés, quelques contacts
CNN (latent convolutionnel)	96.84%	clusters plus compacts, meilleure séparation

TABLE 1 – Comparaison de la séparabilité locale (Neighborhood Hit, $k = 6$) entre MLP et CNN.

Comparaison qualitative (forme des clusters). Visuellement, les projections t-SNE du CNN montrent des classes plus **compactes** et des frontières plus **nettes**. Les enveloppes convexes et les ellipses (GMM) confirment cette tendance : elles se chevauchent moins que pour le MLP.

Conclusion. Les résultats montrent que les réseaux profonds apprennent des transformations internes qui rendent les classes **plus séparables** : c'est précisément l'effet de **manifold untangling**. De plus, l'architecture convolutionnelle améliore encore cet effet grâce à l'extraction hiérarchique de motifs locaux, ce qui mène à une représentation latente plus structurée et mieux discriminante.

4) Figures (MLP vs CNN) — t-SNE + métriques



(a) MLP : t-SNE des représentations cachées (NH = 94.23%).

(b) CNN : t-SNE des représentations latentes (NH = 96.84%).

FIGURE 6 – Comparaison MLP vs CNN dans l'espace latent via t-SNE (test MNIST, 10 000 points).

Conclusion générale du TP2

Ce TP avait pour objectif principal de prendre en main **Keras**/**TensorFlow** pour construire, entraîner et analyser plusieurs architectures de réseaux de neurones sur la base **MNIST**, tout en comprenant comment ces modèles transforment progressivement l'espace des données afin de rendre les classes plus séparables (phénomène de *manifold untangling*).

1) Mise en place et préparation des données

Nous avons commencé par charger MNIST et préparer les données selon le modèle utilisé :
— **Réseaux fully-connected (régression logistique et MLP)** : images aplaties en vecteurs de dimension 784 et normalisées dans $[0, 1]$.
— **CNN (ConvNet type LeNet)** : images conservées sous forme de tenseurs $28 \times 28 \times 1$.
— Conversion des labels en **one-hot encoding**, indispensable pour la perte `categorical_crossentropy`.
Cette étape a montré l'importance du **format des entrées** (vecteur vs tenseur) et du **pré-traitement** pour assurer un apprentissage stable et reproductible.

2) Exercice 1 : Régression logistique avec Keras

Nous avons implémenté un modèle simple (équivalent à une régression logistique multiclasses) :

$$x \mapsto \text{softmax}(xW + b)$$

Ce modèle reste **linéaire** dans l'espace d'entrée : il apprend des frontières de décision linéaires. Il permet toutefois d'obtenir des performances correctes sur MNIST (environ 92%), avec un nombre limité de paramètres. Cet exercice a permis de comprendre le pipeline Keras complet : `model.add()` → `compile()` → `fit()` → `evaluate()`.

3) Exercice 2 : Perceptron (MLP) avec Keras

Nous avons ensuite enrichi le modèle en ajoutant une couche cachée de 100 neurones et une non-linéarité sigmoïde :

$$x \mapsto \text{softmax}(\sigma(xW_1 + b_1)W_2 + b_2)$$

L'ajout de la non-linéarité permet d'apprendre des **frontières non linéaires**, donc un modèle plus expressif. Le modèle contient beaucoup plus de paramètres que la régression logistique, et les performances ont augmenté (environ 95% dans notre expérience). Nous avons également observé l'initialisation par défaut de Keras : **GlorotUniform (Xavier)** pour les poids et **Zeros** pour les biais, ce qui stabilise la propagation des activations et des gradients.

4) Exercice 3 : CNN (ConvNet type LeNet) avec Keras

Nous avons enfin implémenté un réseau convolutif inspiré de LeNet5 :
— deux blocs convolution + ReLU + max-pooling,
— flatten,
— couche fully-connected (100) + sigmoid,
— couche de sortie (10) + softmax.
Contrairement aux réseaux entièrement connectés, les convolutions exploitent la structure spatiale des images : **localité**, **partage de poids** et **invariance locale aux translations**. Cela explique de meilleures performances en classification, avec un score proche de 99% sur la base de test.

Nous avons également mesuré le **temps moyen par époque** (environ 14.4 s/époque sur CPU dans notre configuration), ce qui illustre le coût computationnel plus élevé d'un CNN mais aussi son efficacité en généralisation.

5) Exercice 4 : Visualisation t-SNE et métriques de séparabilité

Nous avons étudié la séparabilité des classes en projetant les données brutes dans un espace 2D via :

— **PCA** : méthode linéaire maximisant la variance globale,
— **t-SNE** : méthode non linéaire préservant les voisinages locaux.

Pour quantifier la séparation, nous avons calculé :

1. **Enveloppes convexes** : étendue géométrique des classes,
2. **Ellipses (GMM)** : dispersion et orientation des classes,
3. **Neighborhood Hit (NH)** : pureté locale des voisinages.

Les résultats montrent clairement que t-SNE sépare beaucoup mieux les classes que PCA :

$$NH_{\text{PCA}} \approx 0.385 \qquad \text{vs} \qquad NH_{\text{t-SNE}} \approx 0.933$$

Ce constat confirme que la structure des données MNIST est **fortement non linéaire** dans l'espace d'entrée.

6) Exercice 5 : Manifold untangling des représentations internes (MLP vs CNN)

Enfin, nous avons appliqué t-SNE non plus sur les pixels, mais sur les **représentations internes** des modèles :

— **MLP** : couche cachée de dimension 100,
— **CNN** : représentation latente (avant la dernière couche de classification).

Les scores NH obtenus indiquent une meilleure séparation dans l'espace latent du CNN :

$$NH_{\text{MLP}} \approx 0.942 \qquad \text{et} \qquad NH_{\text{CNN}} \approx 0.968$$

Cela s'explique par le fait que les convolutions apprennent des **motifs locaux** (contours, strokes, formes) et construisent une hiérarchie de caractéristiques plus robuste, ce qui facilite la séparation des classes. Ce résultat illustre concrètement l'effet de **manifold untangling** : le réseau transforme progressivement l'espace de représentation pour rendre les classes plus facilement séparables par une couche finale simple.

Bilan

En résumé, ce TP nous a permis :

— de maîtriser l'utilisation pratique de Keras (construction, compilation, entraînement, évaluation, sauvegarde),
— de comparer trois familles de modèles : **linéaire (LR)**, **non linéaire (MLP)**, **convolutionnel (CNN)**,
— de comprendre l'intérêt des CNN pour des données d'images,
— d'analyser les représentations internes via des projections 2D et des métriques quantitatives,
— d'observer expérimentalement l'effet du **manifold untangling** et pourquoi le CNN produit des espaces latents plus séparables que le MLP.

Ainsi, ce TP relie à la fois la pratique (implémentation Keras) et l'analyse (visualisation et métriques), et montre comment l'architecture influence directement la structure de l'espace de représentation appris.