



LU3MA201
PROJET, TRAVAIL D'ÉTUDE ET DE RECHERCHE
L3 MATHÉMATIQUES

Rapport final

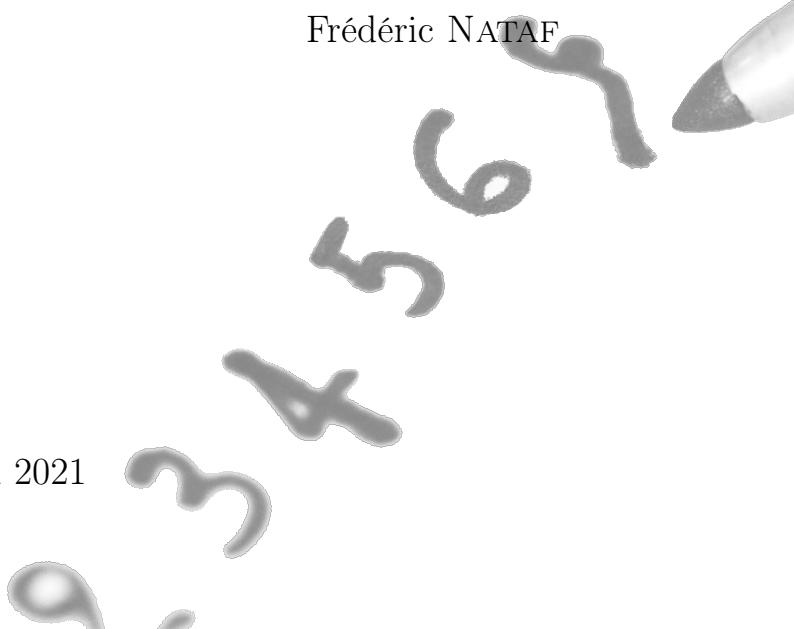
Etudiants :

Aya BOUZIDI
Amaia CARDIEL
Camille GRIMAL
Elysé RASOLOARIVONY

Encadrant :

Frédéric NATAF

9 mai 2021

A large, faint watermark or signature in the bottom right corner, consisting of several handwritten numbers (6, 5, 4, 3) and a stylized signature, with a pencil tip icon at the end.

Résumé

Ce projet de fin de licence, orienté recherche, consiste à introduire et mettre en oeuvre différents algorithmes de reconnaissance de caractères manuscrits. L'objectif de ce projet est de programmer ces algorithmes dans un langage informatique afin d'analyser leur pertinence et leurs limites respectives.

Mots clés : reconnaissance de caractères manuscrits, apprentissage automatique, distances, normes, SVD, distance tangente, algèbre linéaire, calcul différentiel



Introduction

Reconnaissance de chiffres manuscrits

La reconnaissance de caractères manuscrits est l'un des plus anciens défis du domaine de l'**apprentissage automatique**. Il consiste à chercher des algorithmes performants capables de **classifier des données** (*par exemple des images...*) en un nombre de **catégories** (*par exemple les chiffres allant de 0 à 9...*). Ce domaine de recherche a connu un vif essor ces dernières années et offre des applications très larges : *reconnaissance des codes postaux manuscrits, dématérialisation des documents administratifs par la numérisation automatique, scan d'un texte et facilitation de son exploitation par ordinateur...*

Dans ce projet, nous nous limiterons au problème de **reconnaissance de chiffres manuscrits** à partir de la base de données **MNIST** que nous présenterons dans la section suivante. Notre objectif sera de mettre en oeuvre différents algorithmes de classification présentés au chapitre 10 du livre de Lars Elden, "*Matrix Methods in Data Mining and Pattern Recognition*" [1], qui constituera notre ouvrage de référence. Nous analyserons les performances et les limites de ces algorithmes afin de familiariser le lecteur avec quelques méthodes de reconnaissance de caractères.

Base de données MNIST

Dans le cadre de ce projet, nous utiliserons la base de données **MNIST** (*Modified National Institute of Standards and Technology*), librement accessible depuis le site web de Yann LeCun [2], célèbre chercheur en Intelligence Artificielle, actuellement chargé de recherche au sein de Facebook. Il s'agit d'une base de données contenant 70 000 images carrées de 28x28 pixels en niveaux de gris allant de 0 à 255, représentant des chiffres écrits à la main allant de 0 à 9. Nous avons représenté quelques chiffres de cette base à la **Figure 1** afin de donner au lecteur un aperçu des objets étudiés.

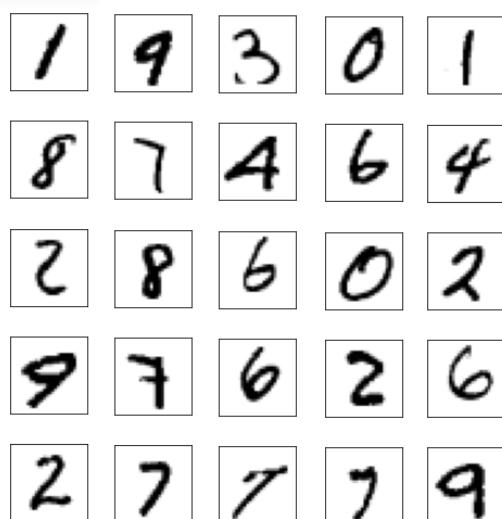


FIGURE 1 – Exemple de chiffres de la base MNIST

Cette base de données est une combinaison de deux bases de données "*bilevel*" (*en noir et blanc*) de l'agence américaine **NIST** (*National Institute of Standards and Technology*). L'une est constituée d'images de chiffres écrits par des lycéens américains et l'autre par 500 différents employés du bureau de recensement des Etats-Unis. La base **MNIST** regroupe ainsi 35 000 exemples de chacune de ces deux bases pour un total de 70 000 images. Notons que ces données ont été adaptées aux besoins de l'apprentissage automatique avant d'y être stockées : les images ont été normalisées afin de pouvoir être contenues dans un carré de 20x20 pixels, puis centrées au sein d'une image carrée de 28x28 pixels. Ces images ont également dû être lissées afin d'éviter le phénomène de crênelage que peut produire le redimensionnement d'image. Cette dernière opération a introduit des niveaux de gris à la place des données en noir et blanc d'origine [2].

Cette base a historiquement été très utilisée dans le domaine de l'apprentissage automatique. En effet, elle permet d'éviter les premières étapes du processus de reconnaissance de texte manuscrit, soit la récolte des données, leur prétraitement, la segmentation de celles-ci en caractères individuels et la sélection de caractéristiques. Ce travail ayant déjà été effectué, utiliser cette base permet de se concentrer sur la dernière étape que constitue la mise en place d'un algorithme de classification [3].

L'autre intérêt majeur est que cette base étant très utilisée, elle permet de comparer ses résultats (*performance de l'algorithme, taux d'erreur...*) avec ceux d'autres chercheurs. Tester un nouvel algorithme sur cette base est ainsi devenu un test standard dans le domaine [4]. L'ensemble des meilleurs résultats de précision, et ce pour la majorité des grandes techniques de classification (*68 algorithmes différents*), a été regroupé sur le site web de Yann LeCun [2]. Nous pouvons ainsi y voir que le plus haut taux d'erreur listé est de 12% pour une méthode de classification linéaire simple, quand les taux les plus faibles, compris entre 0,2% et 0,4%, sont atteints par les réseaux de neurones (convolutifs). C'est pourquoi on considère que cette base de données est historique et qu'elle a été "résolue".

Comme il est de pratique courante en apprentissage automatique, nous partagerons aléatoirement notre base de données MNIST en deux parties **fixées** :

- **Base d'apprentissage** : avec 80% des données qui serviront à entraîner nos programmes.
- **Base de test** : avec 20% des données qui serviront à estimer la précision de nos diverses méthodes.

Organisation du rapport

Ce rapport sera organisé comme suit : la Section I étudiera un algorithme de classification basé sur un calcul de moyenne (où nous atteindrons 82,51% de précision), la Section II se concentrera sur la méthode de classification de la SVD (près de 95,6% de précision) et la Section III sur la distance tangente (où nous atteindrons 98,18% de précision). Le code (Python) développé dans le cadre de ce projet sera regroupé en Annexe du rapport, il sera également accessible sur GitHub à [ce lien](#)¹.

1. https://github.com/drCtul/3m201_groupe8

Table des matières

I Distance aux centroïdes	6
1 Introduction	6
2 Algorithme	6
3 Analyse des résultats	7
3.1 Résultats de précision totale en fonction de chaque distance	7
3.2 Résultats de précision pour chaque distance en fonction de chaque chiffre	7
3.3 Comparaison de précision entre les distances par chiffre	10
3.4 Analyse des estimations erronées par chiffre	11
3.5 Précision et traitement de la base de données	11
3.5.1 Précision et proportion de la base d'apprentissage	11
3.5.2 Test de surapprentissage	13
3.5.3 Précision et caractère aléatoire de la répartition des données	14
3.6 Temps d'exécution	15
4 Conclusion	16
II Décomposition en valeurs singulières	17
5 Introduction	17
6 Algorithme	19
7 Analyse des résultats	22
7.1 Nombre optimal de vecteurs singuliers	22
7.2 Résultats de précision pour chaque chiffre	24
7.3 Exemples de chiffres mal classifiés	24
7.4 Analyse des estimations erronées par chiffre	25
7.5 Test de surapprentissage	27
7.6 Temps d'exécution	27
8 Conclusion	28
III Distance tangente	29
9 Introduction	29
9.1 Motivation	29
9.2 Théorie de la distance tangente	29
9.3 Transformations considérées	30
9.3.1 La translation d'axe (Ox)	30
9.3.2 La translation d'axe (Oy)	31

9.3.3	La rotation	31
9.3.4	La mise à l'échelle (<i>scaling</i>)	32
9.3.5	La transformation hyperbolique parallèle (TPH)	32
9.3.6	La transformation hyperbolique diagonale (TDH)	33
9.3.7	Epaississement (<i>thickening</i>)	34
10	Algorithme	34
11	Analyse des résultats	35
11.1	Résultats de précision (totale et par transformation)	35
11.2	Analyse des estimations erronées par chiffre	36
11.2.1	Matrices de confusion	36
11.2.2	Représentation de chiffres mal identifiés	38
11.3	Tests de l'algorithme	39
11.3.1	Test de la justesse de notre algorithme	39
11.3.2	Test de l'algorithme sur données non lissées	39
11.3.3	Test de l'algorithme sans transformation	40
11.4	Temps d'exécution	41
12	Conclusion	42
A	Code Python : Premier algorithme	44
B	Code Python : Deuxième algorithme	45
C	Code Python : Troisième algorithme	46

Première partie

Distance aux centroïdes

1 Introduction

Cette première méthode est un algorithme très simple de reconnaissance de caractères manuscrits. Il consiste à classifier une image de la base de test au chiffre dont le **centroïde** (*chiffre obtenu par calcul de moyenne sur chaque pixel*) est le plus proche.

Afin de quantifier cette "proximité" entre les données de test et les centroïdes, nous utiliserons :

1. La notion de distance

Nous utiliserons la distance euclidienne ou plus généralement la *distance de Minkowski* pour $2 \leq p \leq 10$:

$$d(u, v) = (|u_1 - v_1|^p + \dots + |u_n - v_n|^p)^{\frac{1}{p}}$$

2. La notion de similarité cosinus

Il s'agit d'une méthode mesurant le cosinus de l'angle entre deux vecteurs pour estimer leur degré de similarité. La formule en est la suivante :

$$\cos\theta(u, v) = \frac{u^T v}{\|u\|_2 \|v\|_2}$$

2 Algorithme

Notre première méthode de classification est basée sur l'algorithme suivant :

- **Etape de prétraitement :** Nous calculons le centroïde de chaque chiffre $1 \leq i \leq 9$ dans la base d'apprentissage.

Pour cela, à partir de l'ensemble des images (vecteurs) de la base d'apprentissage ayant un même label i avec $i \in [0, 9]$, nous calculons la moyenne de chaque pixel pour obtenir le centroïde associé au chiffre i . L'ensemble des centroïdes obtenu a été représenté dans la **Figure 2**.



FIGURE 2 – Centroïdes calculés à partir de notre base d'apprentissage

- **Etape de classification :** Pour chaque vecteur de la base de test, nous lui attribuons la classe du centroïde "le plus proche".

Pour associer à chaque image de la base de test une classe, nous calculons la distance ou le degré de similarité entre le vecteur représentant cette image et chacun des dix vecteurs "centroïdes". Nous comparons ensuite les dix valeurs obtenues.

Dans le cas des distances, nous associerons à chaque donnée de test la classe associée à la valeur minimale. Dans le cas de la similarité cosinus, nous associerons à chaque donnée de test la classe qui maximise la valeur.

3 Analyse des résultats

3.1 Résultats de précision totale en fonction de chaque distance

Distances	Pourcentages de précision (arrondis au centième)
Distance euclidienne	81,59%
Minkowski $p = 3$	82,51%
Minkowski $p = 4$	81,74%
Minkowski $p = 5$	80,34%
Minkowski $p = 6$	79,44%
Minkowski $p = 7$	78,49%
Minkowski $p = 8$	77,79%
Minkowski $p = 9$	77,12%
Minkowski $p = 10$	76,49%
Distance ∞	67,53%
Cosinus	82,16%

TABLE 1 – Pourcentages de précision de l’algorithme par rapport à la distance employée

Le **Tableau 1** rassemble l’ensemble des résultats de précision obtenus par méthode. Ces résultats semblent indiquer que la précision maximale pour la p-distance est obtenue pour $p = 3$ (avec 82,51%) puis que celle-ci décroît quand p croît. Pour comprendre ce phénomène, nous avons également effectué des tests de précision avec la distance infinie. Cela nous a donné un degré de précision de 67,53% (très faible). En effet, la norme infinie ne retient que la plus grande valeur d’un vecteur. Dans le cadre de notre recherche, la distance infinie ne retient donc que les aspects les plus différents entre une image test et un centroïde et attribue une distance très élevée pour chaque comparaison. Ce phénomène explique donc la décroissance de la précision de la p-distance qui, en tendant vers la distance infinie, va également tendre vers la faible précision qui lui est associée.

Enfin, notons que la précision obtenue pour la similarité cosinus (82,16%) est légèrement supérieure à la précision obtenue pour la distance euclidienne (81,59%) tout en étant très proche mais légèrement inférieure au résultat de la p-distance avec $p = 3$ (82,51%).

3.2 Résultats de précision pour chaque distance en fonction de chaque chiffre

Une analyse plus poussée des résultats obtenus pour la distance euclidienne ($p=2$) nous a permis de construire la **Figure 3**. Celle-ci représente le pourcentage de prédictions justes obtenues à l’aide de cette distance en fonction de chaque chiffre.

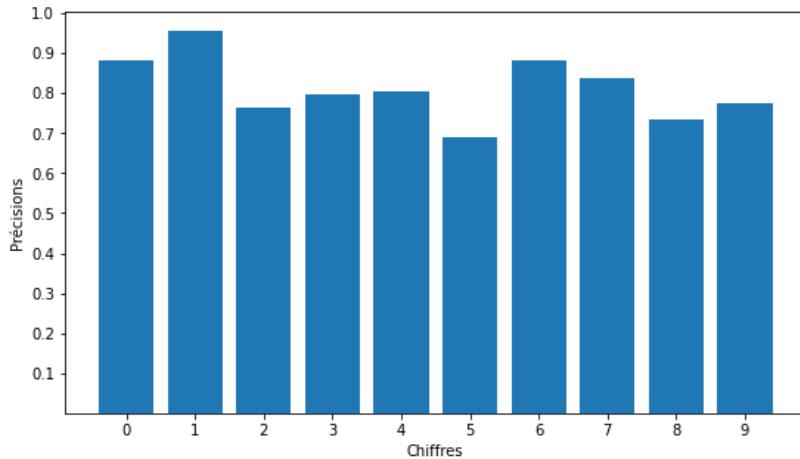


FIGURE 3 – Précision de la méthode de la distance euclidienne par chiffre

Nous remarquons que le chiffre 1 a la plus grande précision par rapport à la distance euclidienne (avec 95,62% de précision) ; le chiffre 5 est à l'inverse le moins bien estimé (avec 69,02%).

En ce qui concerne l'utilisation de la p-distance, une analyse plus fine de la précision obtenue en fonction de chaque chiffre a été représentée à la **Figure 4**. Pour plus de lisibilité, nous y avons représenté les résultats sous forme de courbes plutôt que sous forme d'histogrammes multiples.

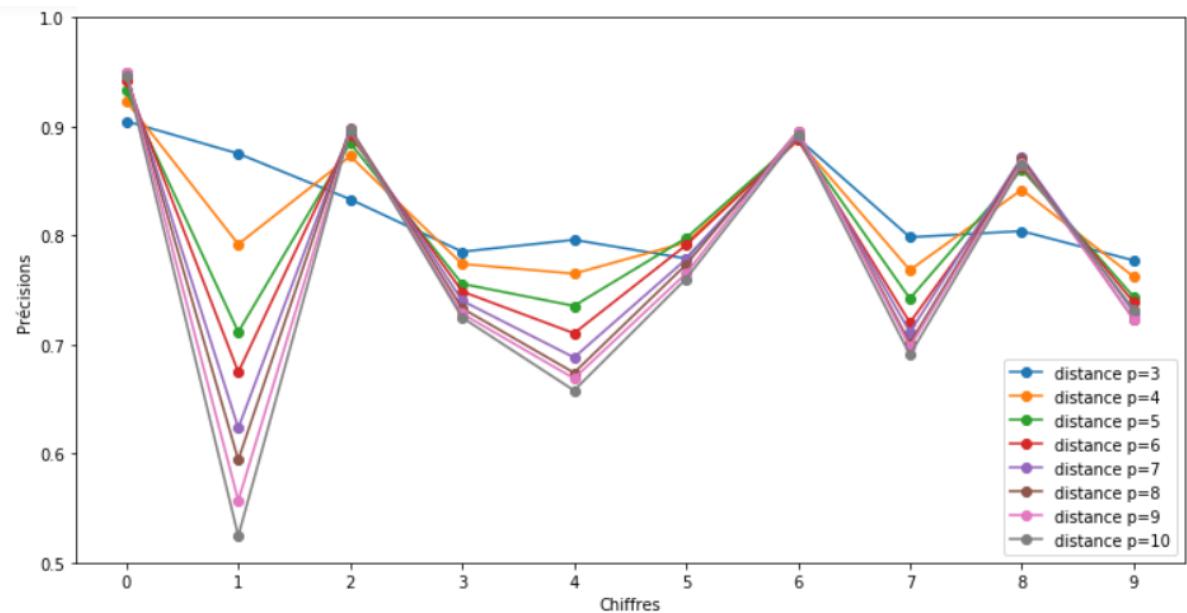


FIGURE 4 – Précision de la méthode de la p-distance par chiffre

Nous voyons sur cette figure, que pour les p-distances avec $3 \leq p \leq 10$, le chiffre 0 a la plus grande précision. Nous remarquons que plus nous augmentons p , plus les précisions de 1, 4 et 7 diminuent quand les autres précisions semblent presque constantes.

Pour la distance infinie, les résultats de précision obtenue en fonction de chaque chiffre ont été représentés à la **Figure 5**.

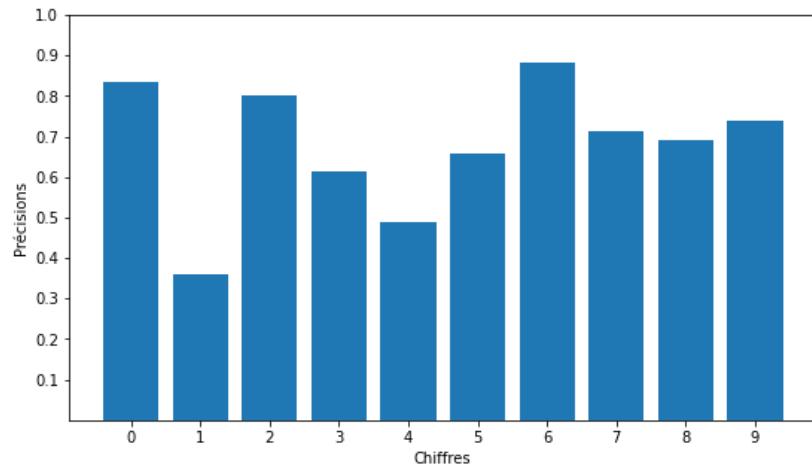


FIGURE 5 – Précision de la méthode de la distance ∞ par chiffre

Comme attendu après étude des p-distances, les degrés de précision des chiffres 1 et 4 sont particulièrement bas pour la distance ∞ (avec respectivement 36,08% et 49,00% de précision).

Enfin, pour la similarité cosinus, la **Figure 6** montre de même la précision obtenue en fonction de chaque chiffre.

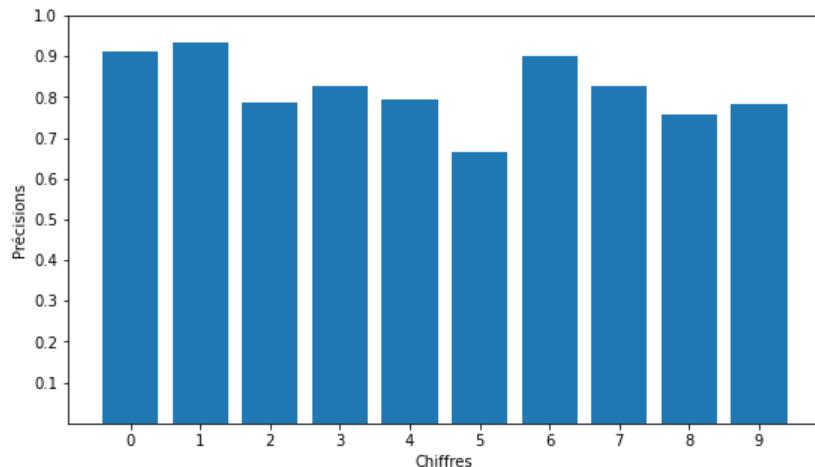


FIGURE 6 – Précision de la méthode de la similarité cosinus par chiffre

Comme pour la distance euclidienne, la similarité cosinus estime le chiffre 1 avec la plus grande précision (avec 93,28% de précision), quand le chiffre 5 est, pour sa part, le moins bien estimé (avec 66,42%).

3.3 Comparaison de précision entre les distances par chiffre

Dans cette sous-section, nous allons comparer nos distances dans leur capacité à estimer correctement chaque chiffre en traçant un graphique regroupant les résultats des quatre figures précédentes (de nouveau, pour plus de lisibilité, nous avons représenté les résultats sous forme de courbes plutôt que sous forme d'histogrammes multiples). Ainsi la **Figure 7** montre les degrés de précision en fonction de chaque chiffre, et ce pour l'ensemble des méthodes de distance ou de similarité utilisées.

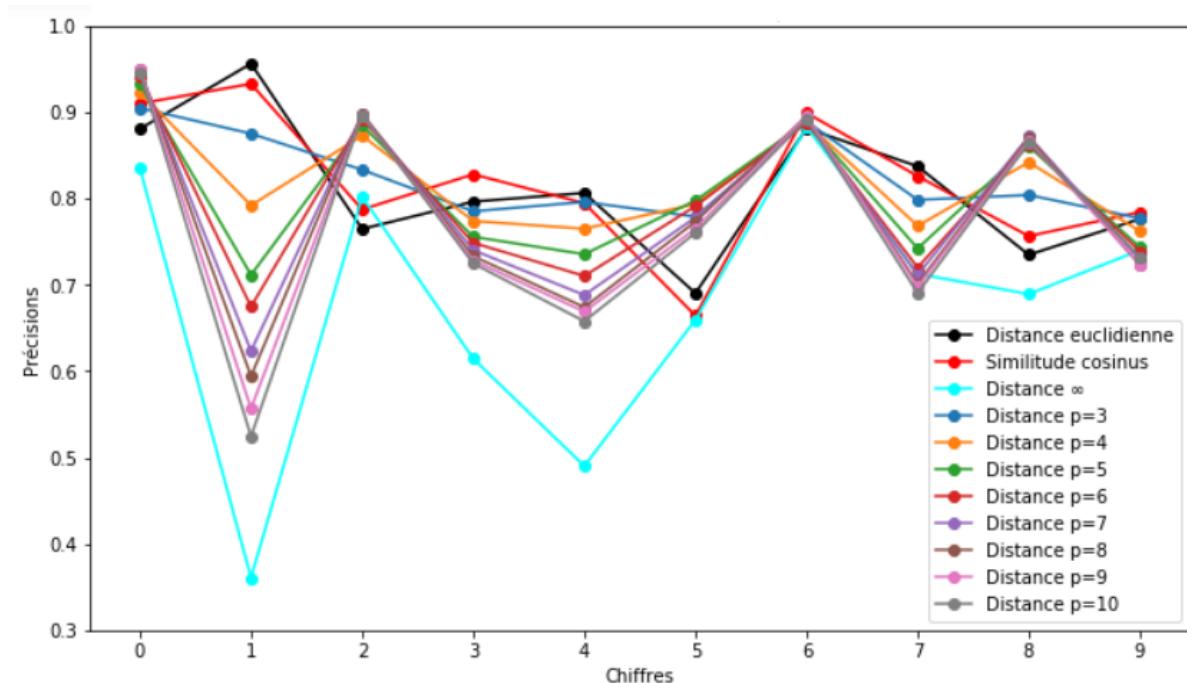


FIGURE 7 – Précision des différentes distances par chiffre

Nous observons sur cette figure croisée que certains chiffres sont identifiés avec une grande précision par toutes les distances, comme le chiffre 0 qui présente plus de 90% d'estimations justes pour les p -distances avec $3 \leq p \leq 10$, 83,52% pour la distance ∞ , 88,06% pour la distance euclidienne et 90,99% pour la similarité cosinus.

Nous observons en revanche de nettes différences dans la précision concernant l'identification de certains chiffres entre les distances. En effet, le chiffre 1 a une très grande précision avec la distance euclidienne et la similarité cosinus (*avec respectivement 95,62% et 93,28% de précision*). Ce degré de précision décroît avec la p -distance quand p croît et chute ainsi à 52,44% pour $p = 10$ et à 36,08% pour la distance ∞ . Un phénomène similaire bien que moins marqué semble se produire pour le chiffre 4 et le chiffre 7.

Ces résultats nous amènent à de premières conjectures : il semble possible que deux chiffres soient fréquemment identifiés l'un pour l'autre. Nous observons par exemple que les évolutions des précisions du chiffre 1 et du chiffre 7 semblent corrélées. Les représentations de leurs centroïdes respectifs étant assez proches, nous formons la conjecture suivante : certains chiffres semblent être régulièrement identifiés l'un à la place de l'autre, ce qui

semble être le cas du 1 et du 7. De telles conjectures nous conduisent naturellement à poursuivre notre analyse dans la sous-section suivante.

3.4 Analyse des estimations erronées par chiffre

Dans cette sous-section, nous souhaitons identifier, pour chaque chiffre de notre base de test n'ayant pas été reconnu correctement, en quel autre chiffre il a été identifié (*en proportion quant au nombre total d'exemples de ce même chiffre ayant été mal identifiés*).

En nous limitant, par souci de lisibilité, à représenter la p -distance pour $p = 3$ et $p = 9$, et à tracer des courbes plutôt que des histogrammes, nous avons représenté l'ensemble de nos résultats sous la forme de dix sous-figures, visibles à la **Figure 8**. Chacune d'entre elles représente, pour l'ensemble des données d'une même classe ayant été mal identifiées, les fréquences relatives des classes erronées qui lui ont été attribuées. Ainsi, la première sous-figure représente la fréquence relative des chiffres (de 1 à 9) qui ont été prédits de manière erronée à la place du chiffre 0 ; la deuxième sous-figure, la fréquence des chiffres prédits à la place du chiffre 1, et ainsi de suite.

Contrairement à la conjecture effectuée à la sous-section précédente, la réalité des estimations erronées semble être plus complexe qu'une simple paire de chiffres pris l'un pour l'autre. En effet, pour chaque chiffre, nous constatons qu'au moins deux autres chiffres ont été fréquemment estimés à sa place. On notera par exemple, dans le cas du chiffre 1, qui nous avait amenés à notre conjecture, que les estimations fausses ne l'identifient pas toutes à un 7, mais l'ont en revanche identifié à plusieurs autres chiffres : le 2 et le 5 (dans plus de 10% des cas d'erreur), mais aussi au 8 (dans plus de 35% des cas d'erreur). Les erreurs semblent donc plus diverses et plus complexes qu'initialement anticipées.

3.5 Précision et traitement de la base de données

Dans cette sous-section, nous avons souhaité étudier l'impact que pouvait avoir le mode de sélection et d'utilisation de la base de données d'apprentissage et de la base de données de test.

Jusqu'à présent, d'après les recommandations du livre de Lars Elden [1] et de notre encadrant, nous avons toujours utilisé une unique découpe aléatoire de notre base de données en 80% pour la base d'apprentissage et 20% pour la base de test, ces deux bases étant complètement scindées (et fixées). Dans cette section, nous avons effectué plusieurs tests afin d'estimer l'impact de la méthode de traitement de la base de données d'origine.

3.5.1 Précision et proportion de la base d'apprentissage

Nous avons tout d'abord étudié l'impact d'une découpe initiale de la base de données d'origine en testant les répartitions suivantes entre *base d'apprentissage / base de test* : 90% / 10%, 80% / 20% (*notre première répartition*), 60% / 40% et 50% / 50%.

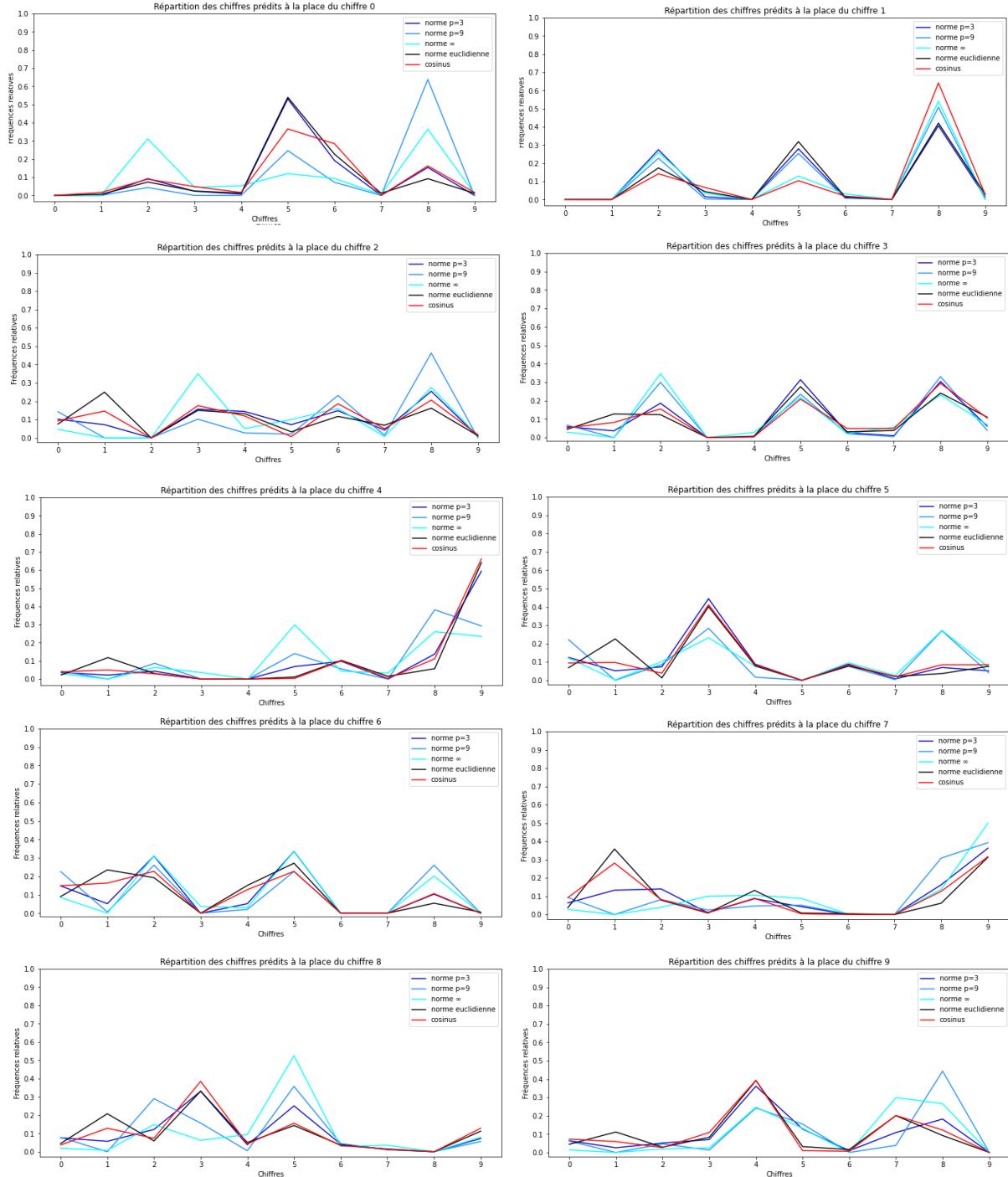


FIGURE 8 – Fréquences relatives des classes attribuées de manière erronée

Avec les nouvelles répartitions, nous avons réeffectué tous les calculs d'estimation de précision avec les distances utilisées précédemment. Nos résultats ont été regroupés au sein de la **Figure 9** sous la forme d'histogrammes qui montrent le degré de précision obtenue en fonction de chaque méthode, et ce pour nos différentes découpages.

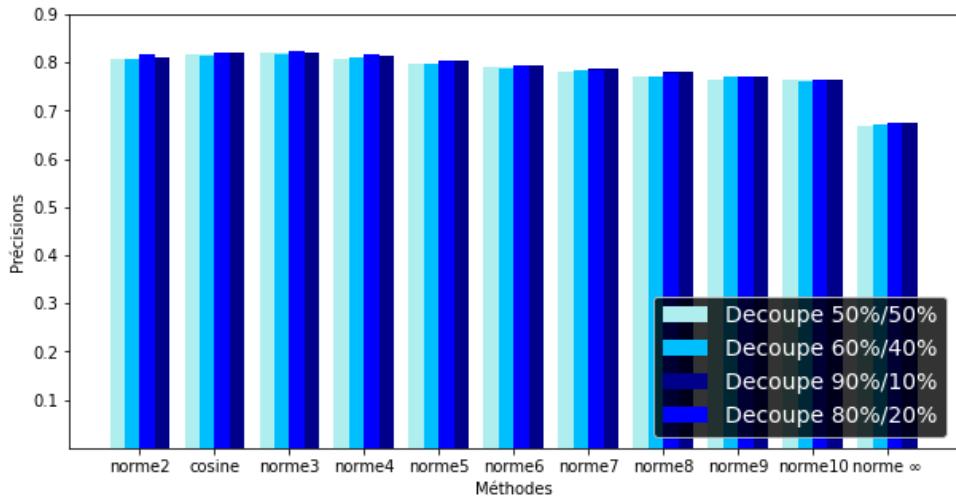


FIGURE 9 – Estimation de la précision de nos diverses méthodes pour différentes répartitions des bases de données

Cette figure semble déjà indiquer que les écarts de résultats obtenus sont très faibles. Un calcul précis nous indique que l'écart maximum (en comparant les résultats pour toutes les distances) en termes de précision entre notre répartition initiale et la répartition de :

- 50% / 50% est de 0,87%.
- 60% / 40% est de 0,99%.
- 90% / 10% est de 0,69%.

Nous pouvons conclure que les estimations de précision obtenues avec diverses répartitions nous ont toutes donné à moins de 1% près le même degré de précision que notre répartition initiale. Ce critère ne semble donc pas jouer un rôle suffisamment important.

3.5.2 Test de surapprentissage

Dans cette analyse, nous avons réalisé un test concernant le possible *surapprentissage* de nos méthodes d'apprentissage. Le surapprentissage est mis en évidence lorsqu'un modèle a de meilleurs résultats de précision quand on le teste au sein de sa propre base d'apprentissage par rapport à une base de test scindée de la base d'apprentissage.

C'est pourquoi nous avons réeffectué les calculs d'estimation de précision pour nos différentes distances en prenant cette fois notre base de test au sein même de la base d'apprentissage. Le résultat des précisions obtenues en fonction de chaque distance, et ce

pour une base de test scindée ou non, est visible à la **Figure 10**.

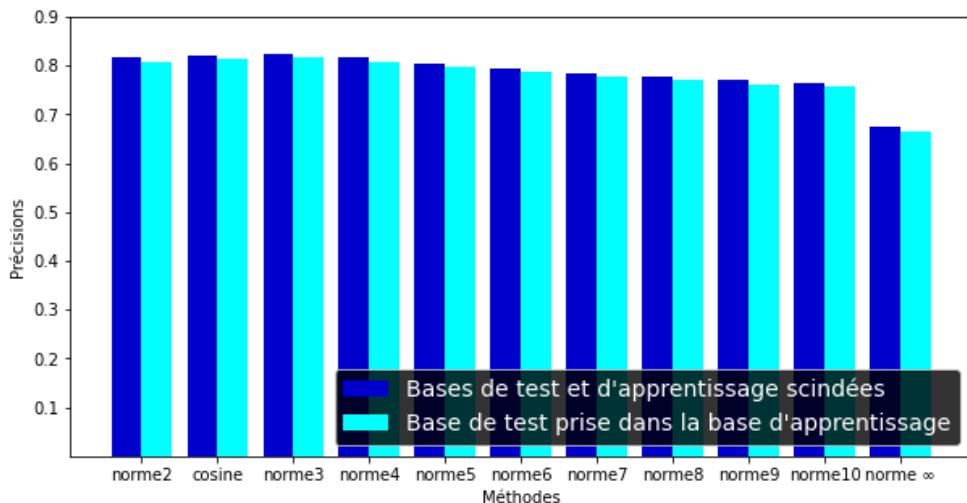


FIGURE 10 – Estimation de la précision de nos diverses méthodes, pour une base de test scindée ou non de la base d'apprentissage

Si nos estimations de précision avaient augmenté avec cette expérience, il aurait fallu conclure à un certain degré de surapprentissage de nos modèles. Néanmoins, les estimations de précision obtenues en prenant une base de test au sein de la base d'apprentissage - et non pas au sein d'une base scindée - donnent toutes à moins de 1,03% de différence près les mêmes résultats que nos premières estimations (avec deux bases complètement scindées). Cela semble indiquer que notre modèle ne souffre pas de surapprentissage.

D'un point de vue théorique, cette absence de surapprentissage peut être expliquée par le fait que notre méthode des centroïdes soit basée sur un calcul de moyenne - la moyenne étant un objet qui converge rapidement.

3.5.3 Précision et caractère aléatoire de la répartition des données

Dans cette dernière analyse, nous avons souhaité quantifier l'impact du caractère aléatoire de la sélection initiale des 80% des données de la base d'apprentissage (*et des 20% restant de la base de test*). Pour cela, nous avons effectué 100 itérations de cette découpe aléatoire initiale et avons analysé la moyenne des estimations de précision obtenues en fonction de chaque distance. La **Figure 11** compare ces résultats à nos premières estimations de précision (dans le cas d'une unique découpe initiale aléatoire).

Les estimations de précision obtenues en prenant la moyenne de ces 100 découpes donnent toutes, à moins de 0,70% de différence près, les mêmes résultats que nos premières estimations avec une seule découpe aléatoire initiale. L'impact de l'aléatoire dans la découpe initiale de notre base de données sur l'estimation de la précision de nos méthodes semble donc très limité. Nous considérerons donc comme satisfaisante la procédure qui consiste à se limiter à une unique découpe aléatoire initiale.

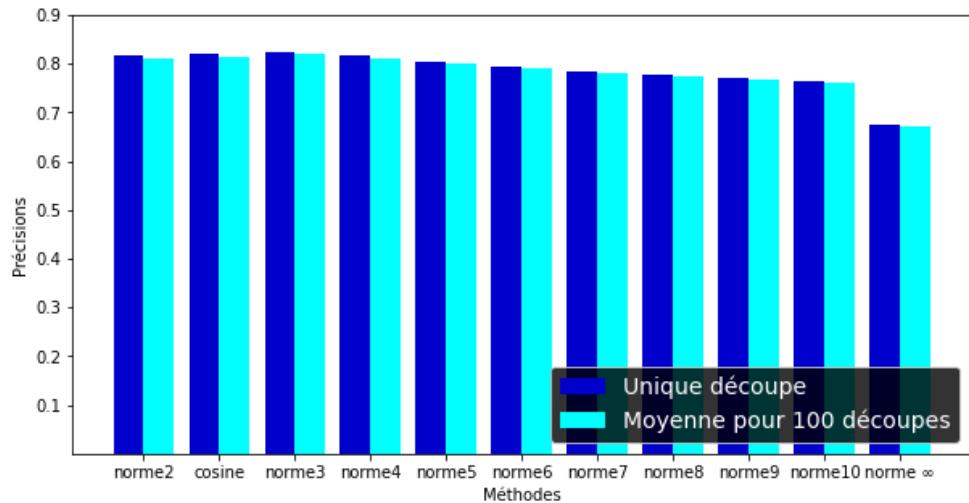


FIGURE 11 – Estimation de la précision de nos diverses méthodes pour différentes découpes initiales

3.6 Temps d'exécution

Dans cette étude, nous avons calculé, lorsque c'était possible, le temps d'exécution de chaque algorithme présenté (codé en langage Python) afin de pouvoir les comparer, non seulement sur la base de la précision mais aussi du coût de calcul. Afin que nos comparaisons aient du sens, le même ordinateur portable a été utilisé afin d'effectuer tous les calculs de ce rapport, effectués à l'aide de la fonction *timeit* de Python, depuis un *jupyter notebook*. Ce dernier élément est un facteur de ralentissement de l'exécution mais il sera de mise pour l'ensemble de nos calculs ; il ne devrait donc pas invalider la pertinence de nos comparaisons.

En ce qui concerne notre premier algorithme, nous avons obtenu les résultats suivants (avec 56 000 données d'apprentissage et 14 000 données de test à classifier) :

- **Prétraitement** (calcul des centroïdes) : 409 millisecondes ;
- **Classification** pour la distance euclidienne : 11,6 secondes ;
- **Classification** pour la p-distance $p=3$: 30,4 secondes.

L'étape de prétraitement, de l'ordre de la centaine de millisecondes, est extrêmement rapide pour cette méthode.

On observe par ailleurs qu'il y a des temps d'exécution variables pour l'étape de classification en fonction de la distance utilisée (du simple au triple entre la distance euclidienne et la p-distance $p=3$ par exemple) mais que son coût reste bas ; l'algorithme est aisément exécutable, même depuis un *jupyter notebook*, en quelques secondes.

4 Conclusion

La meilleure précision obtenue avec cette première méthode de classification sur notre base de test est de 82,51% (*pour la distance de Minkowski avec $p = 3$*), qui est une précision relativement basse. En effet, rappelons que parmi les méthodes listées au sein du site web de Yann LeCun [2], la méthode aux résultats les plus faibles atteint déjà un degré de précision de 88%.

D'un point de vue théorique, la faible performance de notre première méthode est liée au fait qu'elle soit basée sur un algorithme très simple qui ne prend pas en compte les variations de chaque classe de chiffres mais uniquement la moyenne de la classe (*le centroïde*).

Deuxième partie

Décomposition en valeurs singulières

5 Introduction

La méthode de **décomposition en valeurs singulières** ou **SVD** (*Singular Values Decomposition*) est un outil assez important en apprentissage automatique. Dans cette partie, nous allons l'exploiter pour mettre en place un deuxième algorithme de reconnaissance de chiffres manuscrits. Tout d'abord, nous présentons le théorème de la SVD :

Théorème 1 (SVD). Soit $M \in \mathcal{M}_{m,n}(\mathbb{R})$ avec $m, n \in \mathbb{N}$.

Alors il existe deux matrices orthogonales $U \in \mathcal{M}_{m,m}(\mathbb{R})$ et $V \in \mathcal{M}_{n,n}(\mathbb{R})$ telles que :

$$M = U \text{Diag}(\sigma_1, \sigma_2, \dots, \sigma_k) V^t \text{ avec } k = \min(m, n) \text{ et } 0 \leq \sigma_1 \leq \dots \leq \sigma_k$$

$\sigma_1, \sigma_2, \dots, \sigma_k$ s'appellent les **valeurs singulières** de M .

Démonstration. On a $M^t M \in \mathcal{M}_{n,n}(\mathbb{R})$ et $(M^t M)^t = M^t M$

Et puisque $\forall x \in \mathbb{R}^n : x^t M^t M x = (Mx)^t Mx = \|Mx\|_2^2$

Alors $M^t M$ est symétrique et semi-définie positive.

Donc $M^t M$ est diagonalisable de valeurs propres positives $\alpha_1, \dots, \alpha_n$.

C.à.d $\exists V \in \mathcal{M}_{n,n}(\mathbb{R})$ tq $M^t M = V \text{Diag}(\alpha_1, \dots, \alpha_n) V^t$.

On pose $\sigma_i = \sqrt{\alpha_i} \forall i \in [1, n]$

Soit v_1, v_2, \dots, v_n ses vecteurs propres, on peut les prendre comme orthonormaux.

Soit $V = (v_1, v_2, \dots, v_n) \in \mathcal{M}_{n,n}(\mathbb{R})$ donc V est orthogonale.

Pour les σ_i non nuls, on suppose sans perte de généralisation qu'il s'agit de $\sigma_1, \sigma_2, \dots, \sigma_k$.

On pose $u_i = \frac{1}{\sigma_i} M v_i$ pour $i \in [1, k]$ (*).

Pour $i \in [1, k]$, on a : $MM^t u_i = MM^t \frac{M v_i}{\sigma_i} = MM^t M v_i \frac{1}{\sigma_i} = M \sigma_i^2 v_i \frac{1}{\sigma_i} = \sigma_i^2 \frac{1}{\sigma_i} M v_i = \sigma_i^2 u_i$

Donc u_1, u_2, \dots, u_n sont des vecteurs propres de $M^t M$, ils sont donc orthogonaux.

De plus, $u_i^t u_i = (\frac{M v_i}{\sigma_i})^t \frac{M v_i}{\sigma_i} = \frac{v_i^t M^t M v_i}{\sigma_i^2} = \frac{v_i^t \sigma_i^2 v_i}{\sigma_i^2} = v_i^t v_i = 1$ car les v_i sont orthonormaux.

Donc (u_1, u_2, \dots, u_k) est une base orthonormée qu'on peut compléter en une base de \mathbb{R}^m pour obtenir U .

Soit $\Sigma = \text{Diag}(\sigma_1, \sigma_2, \dots, \sigma_k)$ alors $\Sigma^{-1} = \text{Diag}(1/\sigma_1, 1/\sigma_2, \dots, 1/\sigma_k)$

De (*), on déduit que $U = MV\Sigma^{-1}$ donc $U\Sigma = MV$ alors $U\Sigma V^t = M(VV^t) = MId = M$ car V est orthogonale.

Alors $M = U \text{Diag}(\sigma_1, \dots, \sigma_k) V^t$ avec U et V orthogonales et $k = \min(n, p)$ C.Q.F.D \square

La **Figure 12** représente les 400 premières valeurs singulières en fonction de leur indice pour la classe du chiffre 3 de la base d'apprentissage, ainsi que les images de ses trois premiers vecteurs singuliers et les 400 premières coordonnées correspondantes.

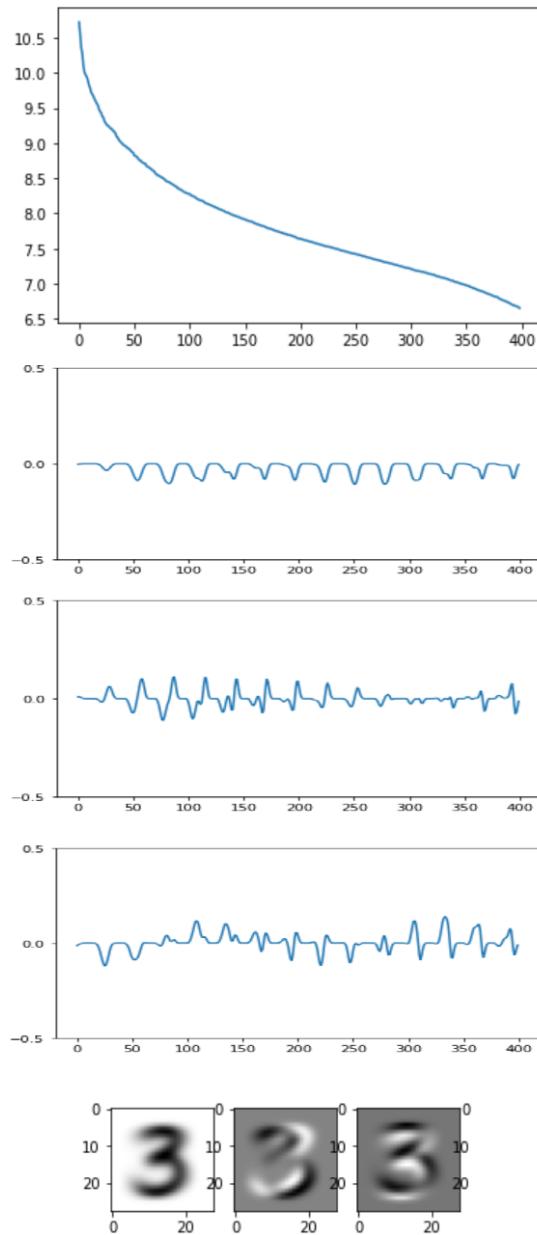


FIGURE 12 – Valeurs singulières, coordonnées des trois premiers vecteurs singuliers (du premier au troisième), images des trois premiers vecteurs singuliers

6 Algorithme

Notre deuxième méthode de classification est basée sur l'algorithme suivant :

- **Etape de prétraitement n°1 :** Pour chaque $i \in [0, 9]$, nous concaténons les vecteurs des images de classe i de la base d'apprentissage dans une grande matrice D_i .
- **Etape de prétraitement n°2 :** Soit la **SVD** pour D_i : $D_i = U_i \Sigma_i V_i^t$. Pour chaque $i \in [0, 9]$, on calcule U_i .

Soit v un vecteur de la base de test. Pour classer v , on veut calculer pour chaque $i \in [0, 9]$ le résidu relatif $r_{i,v} = \frac{1}{\|v\|_2} \min_x \|D_i x - v\|_2^2 = \frac{1}{\|v\|_2} \min_y \|U_i y - v\|_2^2$. v correspondrait au chiffre j tel que $r_j = \min_i \{r_i\}$. Cependant, ce calcul est coûteux et nous verrons dans la suite que cela ne nous permet pas d'avoir une grande précision. Il est alors intéressant de chercher à travailler avec une matrice de rang inférieur au rang de U_i . Pour cela, on introduit le *théorème d'Eckart-Young* :

Théorème 2 (Eckart-Young). Soit $M \in \mathcal{M}_{m,n}(\mathbb{R})$ de rang $r > k$ avec $m, n \in \mathbb{N}$. Soit sa **SVD** $M = U \text{Diag}(\sigma_1, \sigma_2, \dots, \sigma_k) V^t$. Alors $\min_{\text{rg}(Z)=k} \|M - Z\|_2 = \|M - M_k\|_2 = \sigma_{k+1}$ avec $M_k = \sum_{i=1}^k \sigma_i u_i v_i^t$.

D'après le **théorème d'Eckart-Young**, l'approximation de rang $k < r$ de r_i est : $r_i \approx \min_y \|U_{i,k} y - v\|_2^2$ tel que $U_{i,k} = (u_1, \dots, u_r)$. Calculer $\min_y \|U_{i,k} y - v\|_2^2$ revient alors à résoudre $U_{i,k}^t U_{i,r} y = U_{i,k}^t v$. Donc $y = U_{i,k}^t v$ et $r_{i,v} = \|U_{i,k}^t U_{i,k} v - v\|_2^2$.

Pour voir en pratique l'utilité de ce théorème, il est intéressant de voir les images construites à partir de quelques vecteurs singuliers de D_i . La **Figure 13** représente les images des dix premiers vecteurs singuliers pour chaque chiffre. Pour construire ces images (uniquement), les coordonnées des vecteurs ont été passées à la valeur absolue afin que les valeurs de niveau de gris qu'elles codent soient bien comprises entre 0 et 255 (les matrices U_i ont quelques entrées négatives).

Les premières images sont claires et il y a peu de bruit. Cependant, plus les vecteurs singuliers s'éloignent du premier vecteur, plus le bruit augmente. La **Figure 14** représente les images des vecteurs singuliers allant de 100 à 109 pour chaque chiffre. On y voit que le bruit est très élevé et qu'il augmente toujours avec l'indice du vecteur singulier considéré. Pour empêcher que cela ne perturbe nos calculs, nous choisissons donc de travailler avec les k premiers vecteurs singuliers permettant de réduire au maximum le bruit ($k_{optimal}$). Son calcul sera détaillé à la section suivante.

- **Etape de classification :** Pour un vecteur v de la base de test et pour $i \in [0, 9]$, on calcule le résidu relatif $r_{(i,k),v} = \frac{1}{\|v\|_2} \|v - U_{i,k} U_{i,k}^t v\|_2$ pour k optimal. On associe au vecteur v la classe du chiffre i qui minimise ce résidu. (La **Figure 15** représente les résidus relatifs en fonction des bases pour chaque chiffre.)



FIGURE 13 – Images des dix premiers vecteurs singuliers pour chaque chiffre

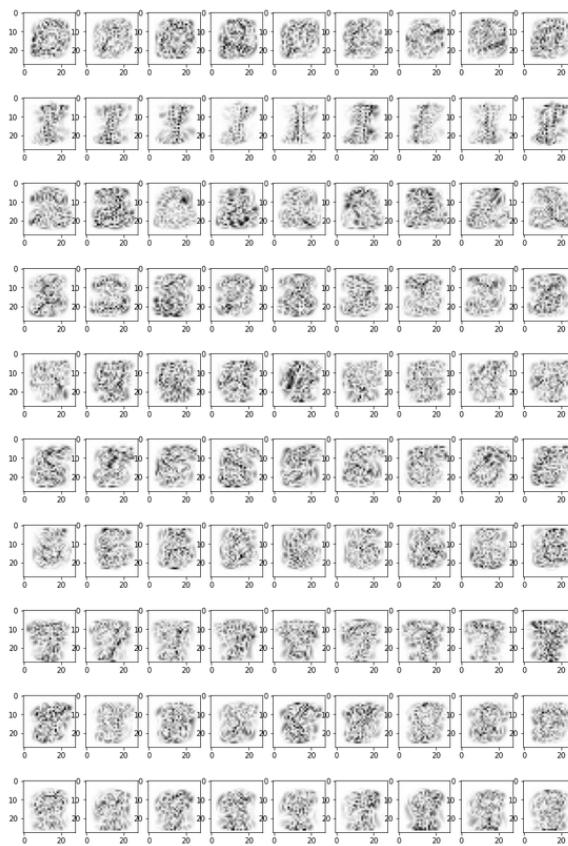


FIGURE 14 – Images des vecteurs singuliers n°100 à 109 pour chaque chiffre

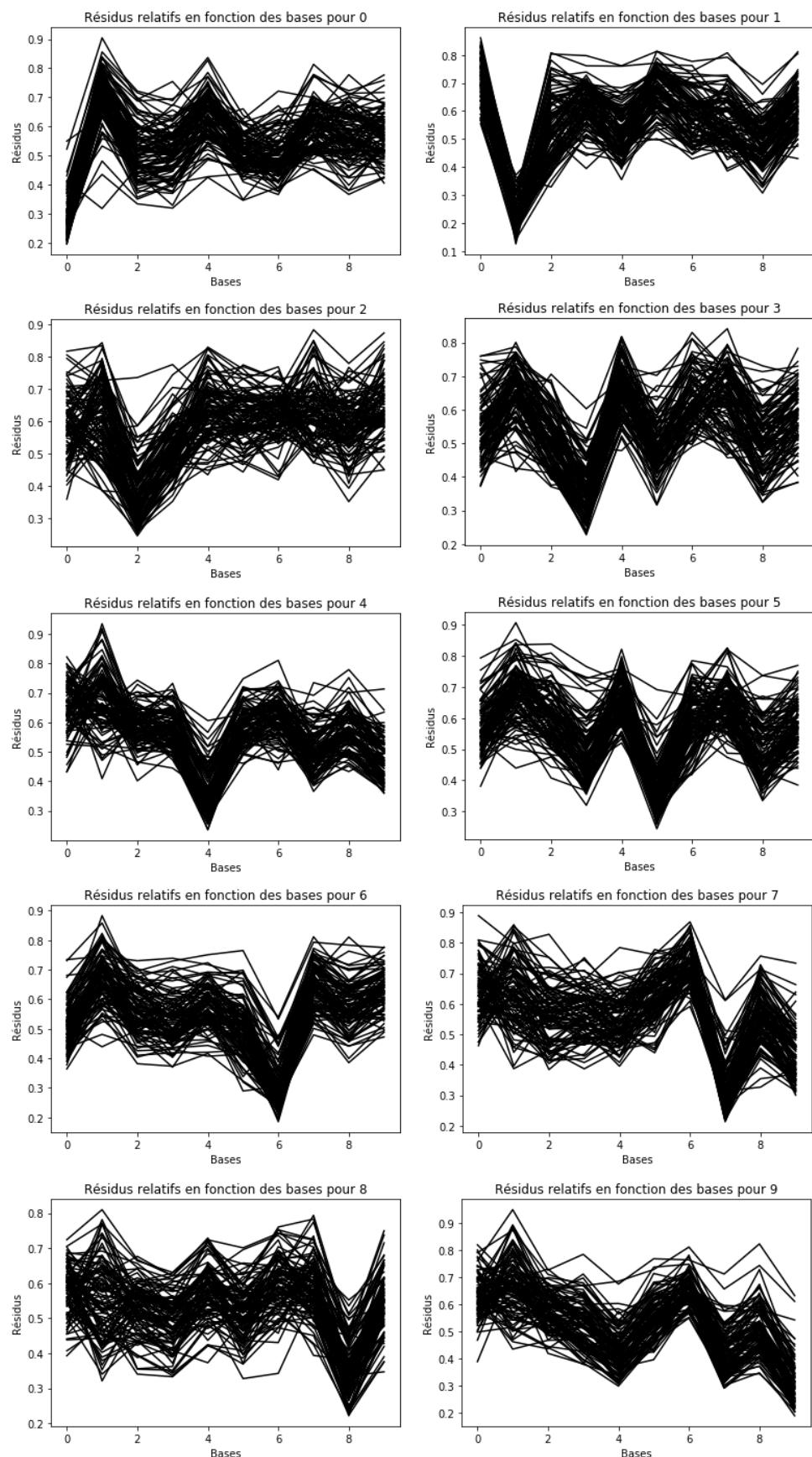


FIGURE 15 – Résidus relatifs en fonction des bases pour chaque chiffre

7 Analyse des résultats

7.1 Nombre optimal de vecteurs singuliers

Dans cette sous-section, nous cherchons à trouver le nombre de vecteurs singuliers optimal $k_{optimal}$ qui donne la meilleure précision de classification. Pour cela, nous traçons le graphe de précision en fonction de k avec $1 \leq k \leq 100$, à la **Figure 16**.

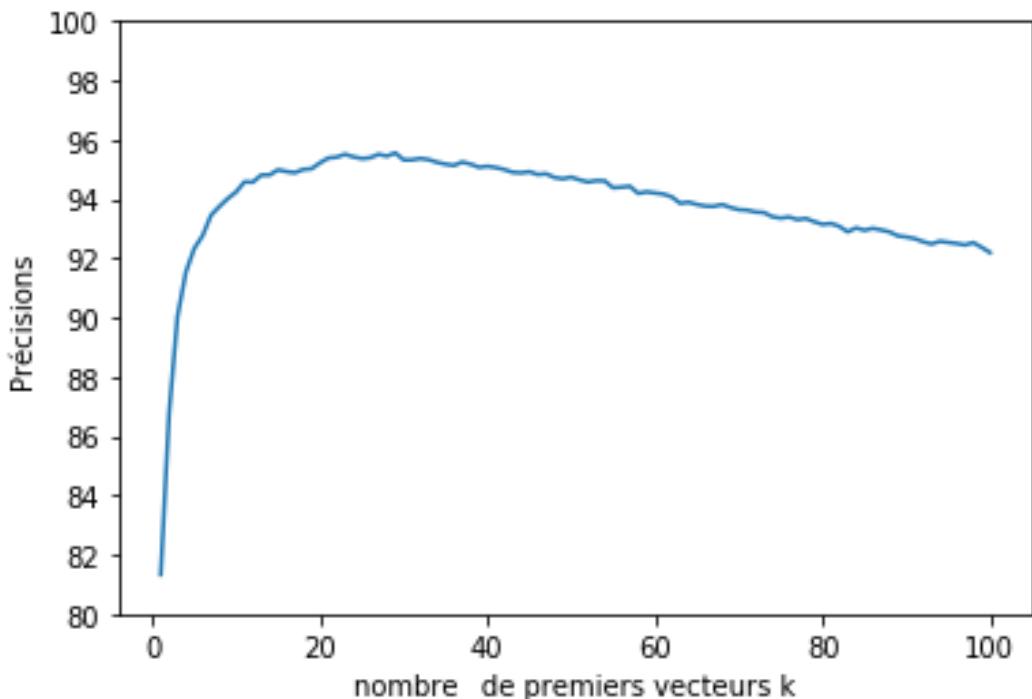


FIGURE 16 – Précision totale en fonction du nombre de premiers vecteurs singuliers k

Nous remarquons que la précision diminue quand k s'éloigne de 40 et que $20 \leq k_{optimal} \leq 40$. Nous nous intéressons alors aux précisions obtenues pour $20 \leq k \leq 40$ que nous avons représentées dans le **Tableau 2** et dans la **Figure 17**.

Nous en déduisons alors que $k_{optimal} = 28$ car ce dernier donne la plus grande précision (95,55%). Dans la suite de notre analyse, nous travaillerons donc avec $k = 28$.

Nombre de vecteurs singuliers	Pourcentage de précision
$k = 20$	95,3833%
$k = 21$	95,4129%
$k = 22$	95,5108%
$k = 23$	95,4230%
$k = 24$	95,3647%
$k = 25$	95,3921%
$k = 26$	95,5102%
$k = 27$	95,4481%
$k = 28$	95,5575%
$k = 29$	95,3194%
$k = 30$	95,3223%
$k = 31$	95,3698%
$k = 32$	95,3271%
$k = 33$	95,2386%
$k = 34$	95,1751%
$k = 35$	95,1324%
$k = 36$	95,2493%
$k = 37$	95,1749%
$k = 38$	95,0710%
$k = 39$	95,1060%
$k = 40$	95,0608%

TABLE 2 – Précisions pour $20 \leq k \leq 40$

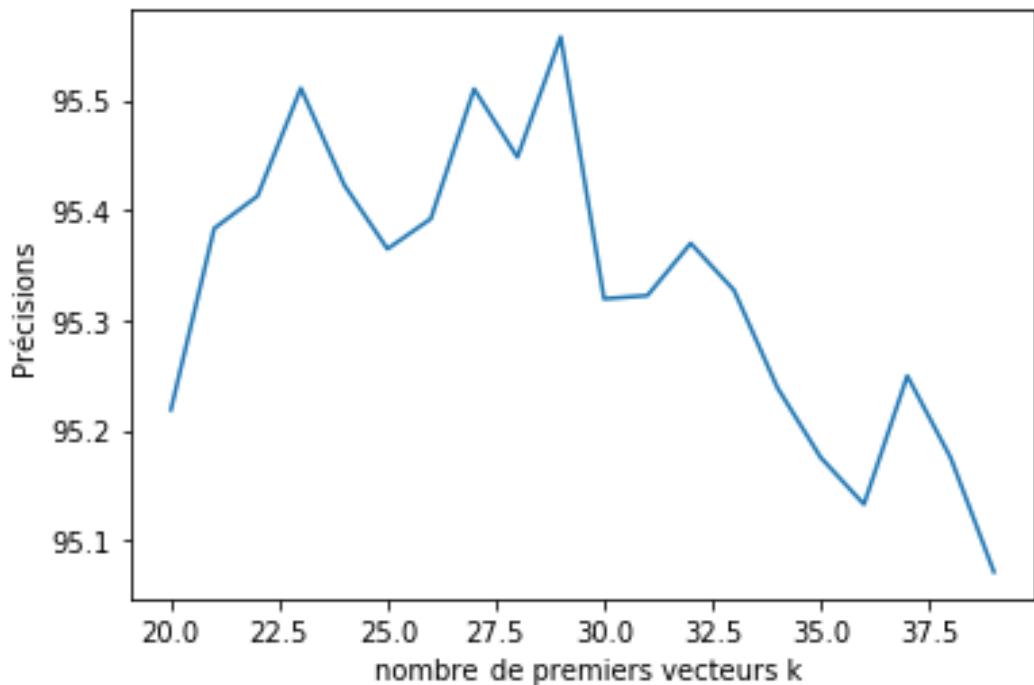


FIGURE 17 – Précision totale en fonction du nombre de vecteurs singuliers $20 \leq k \leq 40$

7.2 Résultats de précision pour chaque chiffre

Chiffre	Pourcentages de précision
0	98,46%
1	99,17%
2	94,62%
3	93,89%
4	95,41%
5	93,57%
6	97,13%
7	95,78%
8	92,32%
9	94,08%

TABLE 3 – Précision en fonction de chaque chiffre pour $k_{optimal}$

Une analyse plus fine de la précision en fonction de chaque chiffre pour $k_{optimal}$ donne le **Tableau 3** et la **Figure 18**.

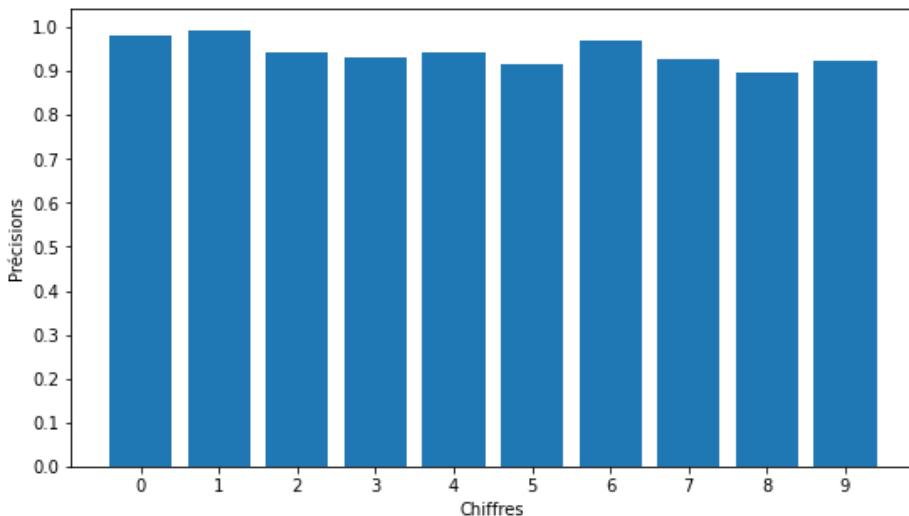


FIGURE 18 – Précision en fonction de chaque chiffre pour $k_{optimal}$

On constate que, comme pour la méthode de *Distance aux centroïdes*, le chiffre 1 est le mieux reconnu (avec 99,17% de précision) par la SVD et le chiffre 5 est de nouveau parmi les chiffres les plus mal estimés (avec une précision de 93,57%). En revanche, contrairement à notre premier algorithme, le chiffre 8 a cette fois la plus basse précision (avec tout de même plus de 92% de précision).

7.3 Exemples de chiffres mal classifiés

La **Figure 19** montre des exemples de chiffres mal classifiés par notre algorithme. On remarque que la plupart de ceux-ci ont une assez mauvaise graphie. Notons tout de même que certains d'entre eux sont bien écrits et que la majorité de ces exemples reste tout à fait identifiable par un oeil humain.

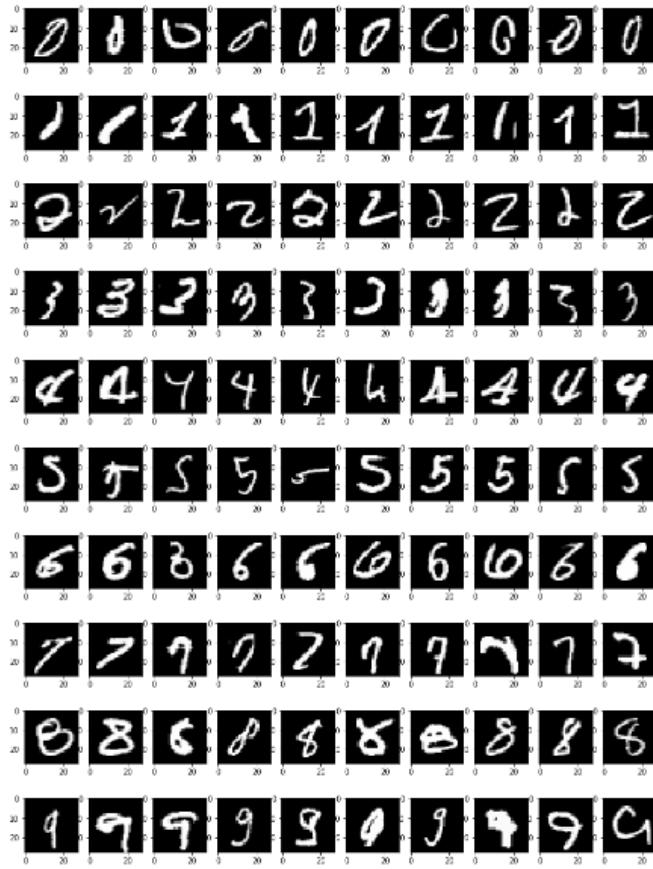


FIGURE 19 – Exemples de chiffres mal classifiés par la SVD

7.4 Analyse des estimations erronées par chiffre

Dans cette sous-section, nous souhaitons identifier, pour chaque chiffre de notre base de test n’ayant pas été reconnu correctement, en quel autre chiffre il a été identifié (*en proportion quant au nombre total d’exemples de ce même chiffre ayant été mal identifiés*). Nous obtenons alors la **Figure 20**. Chacune de ses sous-figures représente, pour l’ensemble des données d’une même classe ayant été mal identifiées, les fréquences relatives des classes erronées qui lui ont été attribuées.

Comme nous l’avions constaté pour la méthode des centroïdes, il ne semble pas que les erreurs de la SVD soient simplement dues au fait que deux chiffres soient systématiquement pris l’un pour l’autre. En effet, pour chaque chiffre, nous avons détecté au moins trois classes erronées différentes. Nous remarquerons cependant le cas à part du chiffre 1, pour lequel une classe erronée en particulier a été attribuée avec une fréquence très élevée : celle du chiffre 2 dans plus de 80% des cas d’erreur. Notons néanmoins que ce phénomène n’est pas réciproque en ce qui concerne la répartition des classes erronées attribuées au chiffre 2 : celle du chiffre 1 n’y est pas du tout majoritaire (moins de 20% des cas d’erreur quand la classe du chiffre 1 représente environ 25% des cas). Cela semble confirmer notre observation selon laquelle la SVD n’a pas simplement interverti deux mêmes chiffres de manière systématique.

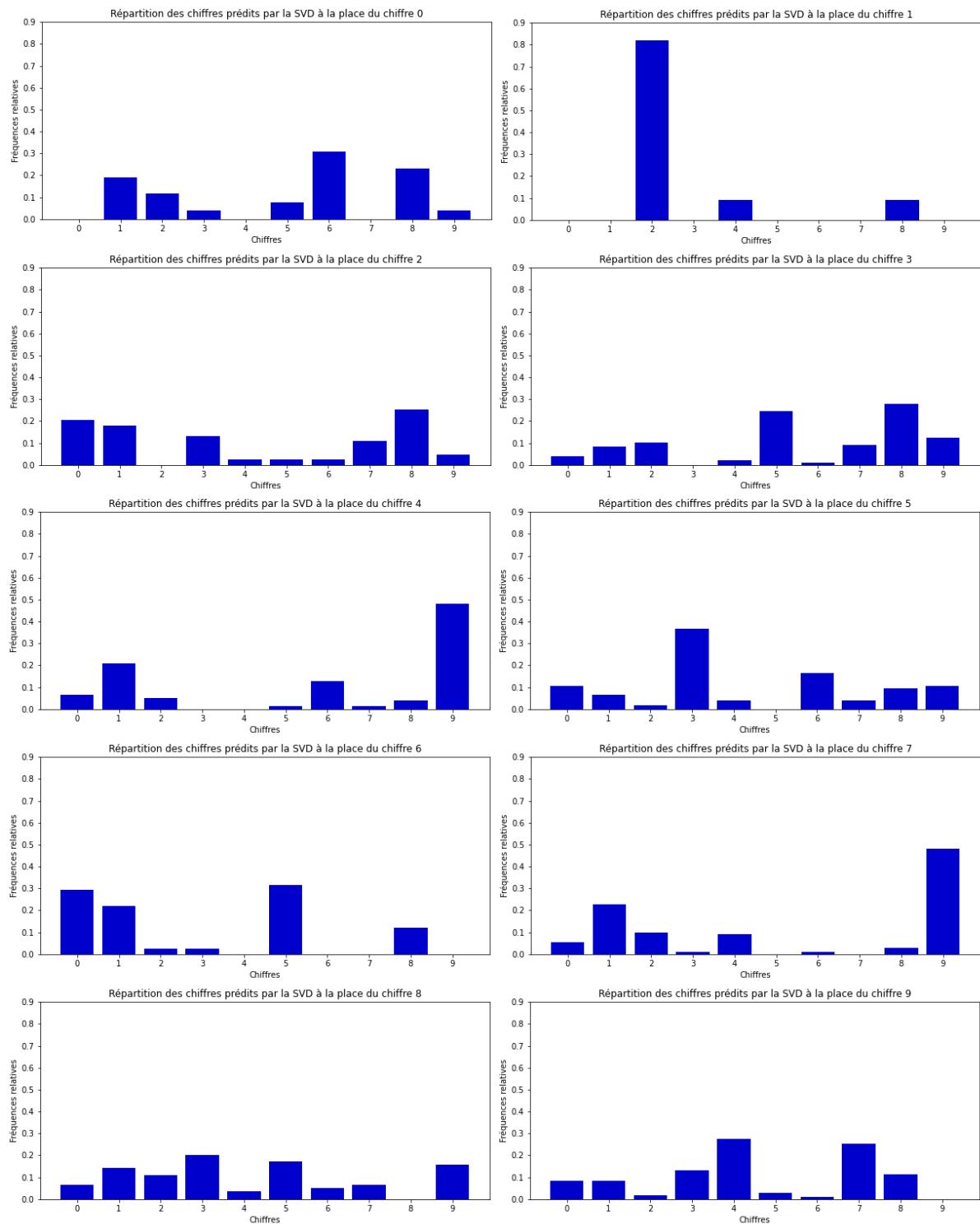


FIGURE 20 – Fréquences relatives des classes attribuées de manière erronée par la SVD

7.5 Test de surapprentissage

De même que pour notre première méthode, nous avons réalisé un test de surapprentissage vis-à-vis de la SVD. Pour ce faire, nous avons réeffectué les calculs d'estimation de précision par chiffre (pour $k_{optimal} = 28$) en prenant cette fois nos 14 000 données de test au sein même de la base d'apprentissage. Le résultat des précisions obtenues, et ce pour une base de test scindée ou non de la base d'apprentissage, est visible à la **Figure 21**.

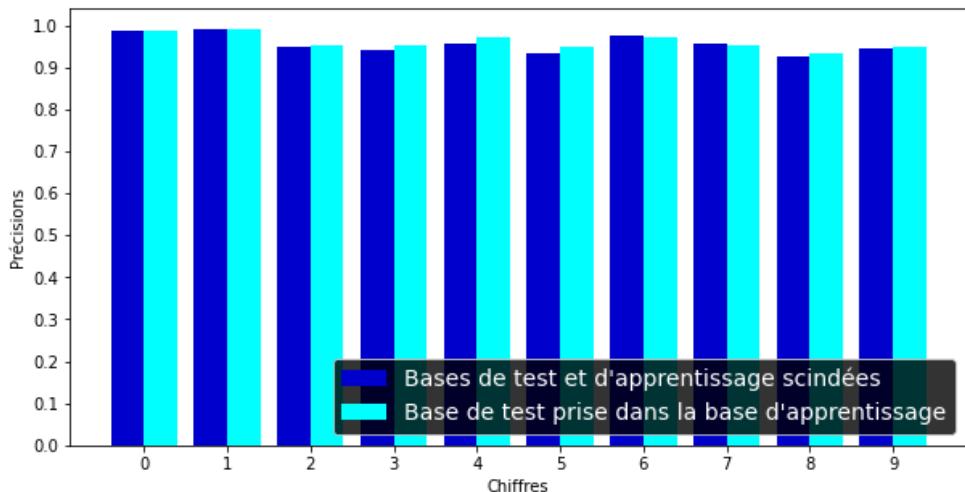


FIGURE 21 – Estimation de la précision de la SVD par chiffre (pour k_{opt}), pour une base de test scindée ou non de la base d'apprentissage

Si nos estimations de précision avaient augmenté avec cette expérience, il aurait fallu conclure à un certain degré de surapprentissage de notre algorithme de SVD. Cependant, les estimations de précision obtenues en prenant une base de test au sein de la base d'apprentissage donnent, pour chaque chiffre, à moins de 1,57% de différence près les mêmes résultats que nos premières estimations (avec deux bases complètement scindées). Cela semble indiquer que cette méthode ne souffre pas de surapprentissage.

7.6 Temps d'exécution

Comme pour notre premier algorithme, nous avons utilisé la fonction *timeit* de Python, depuis un *jupyter notebook*, afin d'estimer les temps d'exécution des étapes de la SVD. Cela nous a permis d'obtenir les résultats suivants (avec 56 000 données d'apprentissage et 14 000 données de test à classifier) :

- **Prétraitement** (calcul des D_i et U_i pour $i \in [0, 9]$) : 5 minutes 45 secondes ;
- **Classification** : 1 minute 4 secondes.

L'étape de prétraitement qui était de 0,409 secondes pour notre premier algorithme est passée à 5 minutes 45 secondes (soit 345 secondes au total) pour la SVD, ce qui est donc $\frac{345}{0,409} \approx 844$ fois plus long.

Les temps d'exécution, bien que variables pour l'étape de classification de la méthode des centroïdes en fonction de la distance utilisée, étaient de l'ordre de la dizaine de secondes. Ici, la SVD requiert 1 minute, ce qui est de 2 à 6 fois plus long.

8 Conclusion

Nous remarquons que les précisions obtenues pour la distance aux centroïdes sont très exactement les mêmes que celles obtenues dans le cas de la SVD avec $k = 1$. En effet, le premier vecteur singulier de U_i pour chaque $i \in [0, 9]$ est le centroïde du chiffre i .

Notons également que le meilleur score de précision obtenu dans notre première partie était de 82,51% (*dans le cas de la p-distance avec p=3*), quand celui-ci est de 95,56% dans le cas de la SVD (avec $k = 28$). Le défaut de précision est donc passé de 17,49% à 4,44%. Or, on a que : $\frac{17,49}{4,44} \approx 3,94$. **Nous obtenons donc une précision près de quatre fois plus grande en passant du cas optimal de la méthode des centroïdes à celui de la SVD.**

On notera tout de même un coût de calcul plus élevé pour la SVD que pour la méthode des centroïdes : si la SVD requiert seulement de 2 à 6 fois plus de temps pour l'exécution de l'étape de classification, elle requiert par ailleurs plus de 840 fois plus de temps pour son étape de prétraitement. Il est néanmoins préférable que ce coût plus élevé intervienne à l'étape de prétraitement, en amont.

Pour conclure cette section, remarquons que la méthode de la **SVD** a traduit notre problème de reconnaissance d'images manuscrites en un problème d'algèbre linéaire (nos données d'images forment alors un espace vectoriel). Malgré l'aspect très théorique de cette méthode, nous parvenons à obtenir une bonne précision de classification (et un temps d'exécution raisonnable). Néanmoins, nous allons nous intéresser par la suite à une méthode plus naturelle, qui sera basée sur le concept de **distance tangente**.

Troisième partie

Distance tangente

9 Introduction

9.1 Motivation

La **Figure 19** de notre partie dédiée à la SVD a montré que de nombreux chiffres mal classifiés par notre précédent algorithme sont tout à fait identifiables par un oeil humain. La troisième méthode que nous allons étudier, dite de la *distance tangente*, a ainsi été développée dans le but précis de résoudre ce problème en proposant un algorithme qui ne soit pas sensible aux déviations graphiques qui sont à la fois classiques et communément acceptées dans l'écriture manuscrite.

Le développement de cet algorithme, aujourd'hui jugé relativement désuet face à la performance des réseaux de neurones, constitue une étape intermédiaire importante dans l'histoire de la recherche en apprentissage automatique appliquée à la reconnaissance d'image. C'est pourquoi son étude garde toute sa pertinence. Dans cette section, nous nous baserons principalement sur le livre de Lars Elden [1] ainsi que sur l'article "*Transformation invariance in pattern recognition*" [5], dédié à la question et publié par une équipe de chercheurs, parmi lesquels Yann LeCun, en 2006.

9.2 Théorie de la distance tangente

On considère $v \in \mathbb{R}^{784}$ et $s(v, \alpha)$ une transformation de v de paramètre α . On a donc : $s(v, 0) = v$.

La **distance tangente** est définie comme une approximation de la distance minimale entre deux courbes. Dans notre cas, il s'agit de la distance minimale entre deux images p et e qu'on peut approcher par : $\min_{\alpha_p, \alpha_e} \|s(p, \alpha_p) - s(e, \alpha_e)\|_2^2$.

Par le **théorème de Taylor**, on a $s(v, \alpha) \approx s(v, 0) + \frac{ds}{d\alpha}(v, 0)\alpha$

Donc $s(v, \alpha) \approx v + t_v\alpha$ avec $t_v = \frac{ds}{d\alpha}(v, 0)$

Si $\alpha = (\alpha_i)_i$ (un vecteur ligne) alors $s(v, \alpha) \approx v + \sum_i \frac{\partial s}{\partial \alpha_i}(v, 0)\alpha_i \approx v + T_v \alpha_v^t$

Avec $T_v = (\frac{\partial s}{\partial \alpha_i}(v, 0))_i$, appelée **matrice tangente** (la concaténation en colonne des **vecteurs tangents** $\frac{\partial s}{\partial \alpha_i}(v, 0)$), alors :

$$\min_{\alpha_p, \alpha_e} \|s(p, \alpha_p) - s(e, \alpha_e)\|_2^2 = \min_{\alpha_p, \alpha_e} \|(p - e) + (T_p \alpha_p^t - T_e \alpha_e^t)\|_2^2 = \min_{\alpha_p, \alpha_e} \|(p - e) - (-T_p T_e)(\alpha_p^t \alpha_e^t)^t\|_2^2$$

On pose $A = (-T_p T_e)$, $b = p - e$ et $\lambda = (\alpha_p^t \alpha_e^t)^t$

$$\text{Alors } \min_{\alpha_p, \alpha_e} \|s(p, \alpha_p) - s(e, \alpha_e)\|_2^2 = \min_{\lambda} \|b - A\lambda\|_2^2$$

Cela revient donc à résoudre un **problème de moindre carré** !

Pour pouvoir différencier nos images, il faut d'abord les écrire comme des fonctions différentiables à deux variables. Chaque image p peut s'écrire comme une matrice P de taille 28x28. On cherche alors une fonction p différentiable telle que $p(i, j) = P_{i,j}$ pour tout $(i, j) \in [1, 28]$. ($P_{i,j}$ correspond alors à la valeur d'un pixel).

Cela est vérifié pour la fonction discontinue suivante : $p_*(x, y) = \sum_{i,j} P_{i,j} \delta(x-i)\delta(y-j)$. Pour avoir la continuité et la différentiabilité, on convolutionne p_* avec la fonction gaussienne et on obtient $p(x, y) = \sum_{i,j} P_{i,j} g_\sigma(x - i, y - j)$. (Nous travaillons avec $\sigma = 0.9$ pour notre algorithme.)

Cette convolution va ainsi **lisser** nos images et nous permettre de les différencier. La **Figure 22** représente une de nos données de test, sa version lissée, ainsi que ses dérivées partielles $p_x = \frac{\partial p}{\partial x}(x, y)$ et $p_y = \frac{\partial p}{\partial y}(x, y)$.

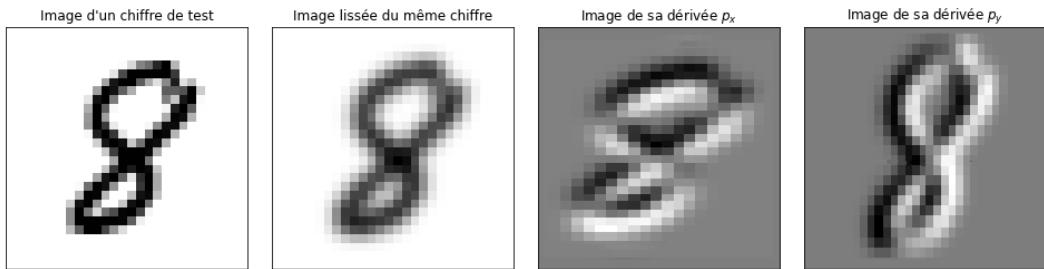


FIGURE 22 – Représentation d'une image de test, de sa version lissée et de ses dérivées

9.3 Transformations considérées

Pour notre troisième algorithme, nous avons travaillé avec sept transformations importantes. Nous présentons dans la suite la fonction correspondante à chaque transformation, sa dérivée (*vecteur tangent correspondant*) et une visualisation de l'effet de chaque transformation sur l'image lissée originale. L'objectif de leur utilisation est de rendre l'algorithme de classification invariant par rapport aux différentes transformations des neuf classes (*style d'écriture, épaisseur de l'écriture, taille des chiffres, déviation des chiffres...*). On remarquera que chacune des dérivées de nos transformations peut s'exprimer sous forme de combinaison linéaire des dérivées partielles p_x et p_y .

9.3.1 La translation d'axe (Ox)

Il s'agit de la transformation la plus simple. La translation de paramètre α_x (le long de l'axe (Ox)) qui transforme l'image $p(x, y)$ est de :

- **Fonction** : $s(p, \alpha_x)(x, y) = p(x + \alpha_x, y)$
- **Vecteur tangent** : $p_x = \frac{d}{d\alpha_x}(s(p, \alpha_x)(x, y))|_{\alpha_x=0}$

Nous avons illustré à la **Figure 23** l'impact de cette transformation sur une image de notre base de données. En effet, pour une image lissée originale p , sa translation par rapport à l'axe des abscisses de paramètre α correspond à $p + \alpha p_x$. Dans la **Figure 23**, on utilise un α positif qui translate de manière visible notre image dans le sens direct de l'axe des abscisses et un α négatif qui translate de même notre image dans le sens indirect de l'axe des abscisses. Nous avons tracé un segment vertical passant par le centre de l'image afin de permettre au lecteur de mieux visualiser la translation du chiffre.



FIGURE 23 – Image originale lissée, son vecteur tangent p_x , sa translation d'axe (Ox) de paramètre positif puis de paramètre négatif

9.3.2 La translation d'axe (Oy)

La translation de paramètre α_y (le long de l'axe (Oy)) qui transforme l'image $p(x, y)$ est de :

- **Fonction** : $s(p, \alpha_y)(x, y) = p(x, y + \alpha_y)$
- **Vecteur tangent** : $p_y = \frac{d}{d\alpha_y}(s(p, \alpha_y)(x, y))|_{\alpha_y=0}$

Nous avons illustré à la **Figure 24** l'impact de cette transformation sur une image de notre base de données. En effet, pour une image lissée originale p , sa translation par rapport à l'axe des ordonnées de paramètre α correspond à $p + \alpha p_y$. Dans la **Figure 24**, on utilise un α positif qui translate bien notre image dans le sens direct de l'axe des ordonnées et un α négatif qui translate bien notre image dans le sens indirect de l'axe des ordonnées. Nous avons tracé un segment horizontal passant par le centre de l'image afin de permettre au lecteur de mieux visualiser la translation du chiffre.

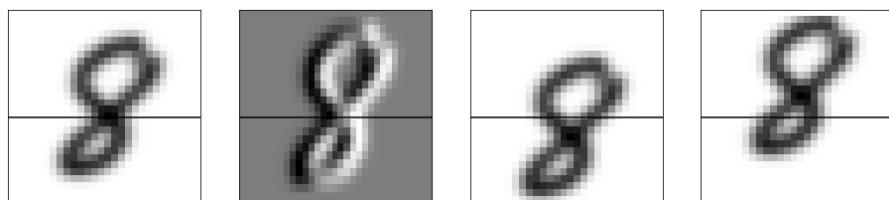


FIGURE 24 – Image originale lissée, son vecteur tangent p_y , sa translation d'axe (Oy) de paramètre positif puis de paramètre négatif

9.3.3 La rotation

La rotation de paramètre α_r qui transforme l'image $p(x, y)$ est de :

- **Fonction** : $s(p, \alpha_r)(x, y) = p(x\cos\alpha_r + y\sin\alpha_r, -x\sin\alpha_r + y\cos\alpha_r)$
- **Vecteur tangent** : $p_r = \frac{d}{d\alpha_r}(s(p, \alpha_r)(x, y))|_{\alpha_r=0} = yp_x - xp_y$

Nous avons illustré à la **Figure 25** l'impact de cette transformation sur une image de notre base de données. En effet, pour une image lissée originale p , sa rotation de paramètre α correspond à $p + \alpha p_r$. Dans la **Figure 25**, on utilise d'abord un α qui effectue une rotation de notre image de sens indirect puis un α qui fait de même dans le sens direct.

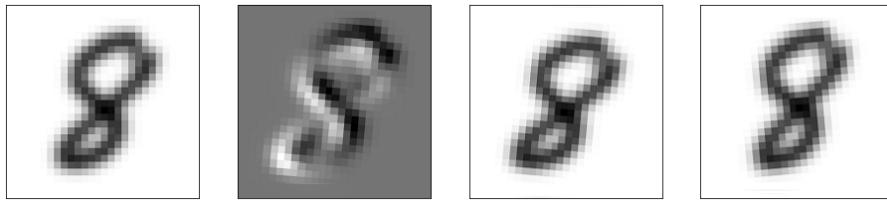


FIGURE 25 – Image originale lissée, son vecteur tangent p_r , sa rotation de sens indirect puis de sens direct

9.3.4 La mise à l'échelle (*scaling*)

Il s'agit d'une transformation qui agrandit ou rétrécit une image par un facteur d'échelle. La mise à l'échelle de facteur α_s qui transforme l'image $p(x, y)$ est de :

- **Fonction** : $s(p, \alpha_s)(x, y) = p((1 + \alpha_s)x, (1 + \alpha_s)y)$
- **Vecteur tangent** : $p_s = \frac{d}{d\alpha_s}(s(p, \alpha_s)(x, y))|_{\alpha_s=0} = xp_x + yp_y$

Nous avons illustré à la **Figure 26** l'impact de cette transformation sur une image de notre base de données. En effet, pour une image lissée originale p , sa mise à l'échelle de facteur α correspond à $p + \alpha p_s$. Dans la **Figure 26**, on utilise un α positif qui agrandit de manière visible notre image et un α négatif qui rétrécit de même notre image.

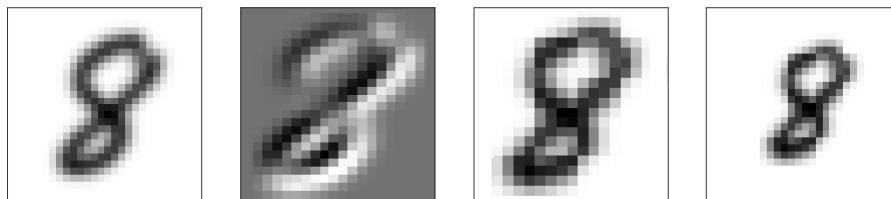


FIGURE 26 – Image originale lissée, son vecteur tangent p_s , sa mise à l'échelle de facteur positif, puis de facteur négatif

9.3.5 La transformation hyperbolique parallèle (TPH)

Il s'agit d'une transformation qui, pour un paramètre α , étire ou compresse l'image selon l'axe (Ox) ou (Oy). La transformation hyperbolique parallèle de paramètre α_p qui transforme l'image $p(x, y)$ est de :

- **Fonction** : $s(p, \alpha_p)(x, y) = p((1 + \alpha_p)x, (1 - \alpha_p)y)$
- **Vecteur tangent** : $p_{TPH} = \frac{d}{d\alpha_p}(s(p, \alpha_p)(x, y))|_{\alpha_p=0} = xp_x - yp_y$

Nous avons illustré à la **Figure 27** l'impact de cette transformation sur une image de notre base de données. En effet, pour une image lissée originale p , sa transformation hyperbolique parallèle de paramètre α correspond à $p + \alpha p_{TPH}$. Dans la **Figure 27**, on utilise un α positif qui étire notre chiffre selon l'axe (Oy) et qui le compresse selon l'axe (Ox), puis un α négatif qui étire notre chiffre selon l'axe (Ox) et qui le compresse selon l'axe (Oy).

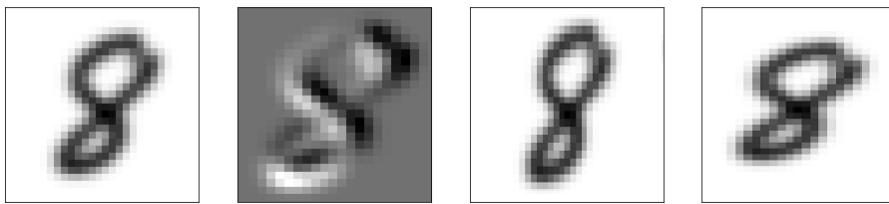


FIGURE 27 – Image originale lissée, son vecteur tangent p_{TPH} , sa transformation hyperbolique parallèle de paramètre positif, sa transformation hyperbolique parallèle de paramètre négatif

9.3.6 La transformation hyperbolique diagonale (TDH)

Il s'agit d'une transformation qui, pour un paramètre α , étire une image diagonalement. La transformation hyperbolique diagonale de paramètre α_d qui transforme l'image $p(x, y)$ est de :

- **Fonction** : $s(p, \alpha_d)(x, y) = p(x + \alpha_dy, y + \alpha_dx)$
- **Vecteur tangent** : $p_{TDH} = \frac{d}{d\alpha_d}(s(p, \alpha_d)(x, y))|_{\alpha_d=0} = yp_x + xp_y$

Nous avons illustré à la **Figure 28** l'impact de cette transformation sur une image de notre base de données. En effet, pour une image lissée originale p , sa transformation hyperbolique diagonale de paramètre α correspond à $p + \alpha p_{TDH}$. Dans la **Figure 28**, on utilise un α positif puis un α négatif qui étirent notre chiffre diagonalement.

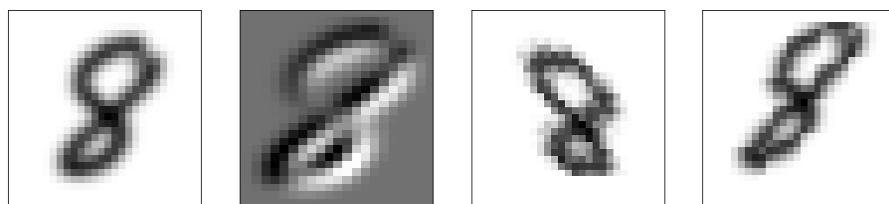


FIGURE 28 – Image originale lissée, son vecteur tangent p_{TDH} , sa transformation hyperbolique diagonale de paramètre positif, sa transformation hyperbolique diagonale de paramètre négatif

9.3.7 Epaississement (*thickening*)

Il s'agit d'une transformation qui, pour un paramètre α , épaisse ou amincit le trait d'écriture d'un chiffre. Il n'existe pas de formule simple pour la fonction de cette transformation ; c'est pourquoi on ne présentera que son vecteur tangent, p_T , de paramètre α_t :

- **Vecteur tangent** : $p_T = \frac{d}{d\alpha_t}(s(p, \alpha_t)(x, y))|_{\alpha_t=0} = (p_x)^2 + (p_y)^2$

Nous avons illustré à la **Figure 29** l'impact de cette transformation sur une image de notre base de données. En effet, pour une image lissée originale p , son épaississement de paramètre α correspond à $p + \alpha p_T$. Dans la **Figure 29**, on utilise un α positif qui épaisse visiblement le trait d'écriture de notre chiffre puis un α négatif qui l'affine.

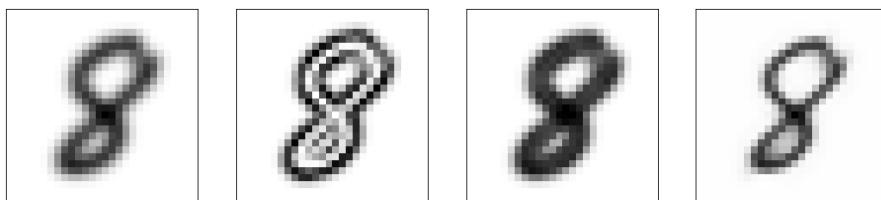


FIGURE 29 – Image originale lissée, son vecteur tangent p_T , son épaississement de paramètre positif, son épaississement de paramètre négatif

10 Algorithme

Pour cette troisième méthode de classification, notre algorithme est le suivant :

- **Etape de prétraitement** : Pour chaque chiffre (lissé) de la base d'apprentissage et de la base de test, nous calculons la matrice tangente T_p .
- **Etape de classification** : Pour chaque chiffre de la base de test, nous calculons sa distance tangente par rapport à tous les chiffres de la base d'apprentissage et le classifions selon la classe du chiffre qui minimise celle-ci.

Notons qu'il est très coûteux, dans la deuxième étape, de comparer la distance tangente entre l'image testée et l'ensemble des images de la base d'apprentissage. Ainsi, par souci pratique, nous avons choisi dans cette deuxième étape de nous contenter d'une restriction à un **échantillon** d'images. Après discussion avec notre tuteur, M. Nataf, il a été convenu qu'utiliser 10 000 données (8000 images d'apprentissage pour 2000 de test) conviendrait.

Ce choix de la taille de l'échantillon n'est pas une question subsidiaire car il revient à effectuer un arbitrage entre les précisions obtenues et le coût de l'algorithme. Pour illustrer cette remarque rapidement, un test de notre algorithme avec seulement 1000 données (200 données de test pour 800 données d'apprentissage) ne présente qu'une précision moyenne de 93,8% quand le livre de **Lars Elden [1]** mentionne une précision de 96,9% lorsque cet algorithme est appliqué à la base de données *U.S. Postal Service* (composée d'images

de résolution 16x16, inférieure à celle de notre base). Cela a confirmé notre choix de travailler avec un échantillon dix fois plus grand afin d'obtenir des résultats plus précis, comme nous allons le détailler dans la section suivante. Nous ne le préciserons donc pas systématiquement mais l'ensemble des résultats qui suivent proviennent de l'utilisation d'un même échantillon (fixé) de 10 000 données.

11 Analyse des résultats

Remarquons, avant toute analyse, que nous doutons que la prise en compte des translations et des mises à l'échelle (*scaling*) soit très significative quand nous les appliquerons à la base de données MNIST puisque, rappelons-le, cette base de données ne contient que des images préalablement normalisées centrées. En revanche, on s'attend à ce que la prise en compte d'autres transformations, notamment la rotation et les transformations hyperboliques, ait un impact sensible sur la précision. Voyons si nos résultats confirment ces hypothèses.

11.1 Résultats de précision (totale et par transformation)

Nous avons tout d'abord mis en oeuvre l'algorithme décrit ci-dessus (avec 10 000 données) afin d'obtenir des résultats de précision en utilisant l'ensemble des sept transformations considérées dans l'étape de classification.

Dans un second temps, pour approfondir notre compréhension de l'impact de chaque transformation sur les résultats, nous avons appliqué de nouveau notre algorithme mais en ne lui faisant prendre en compte à chaque fois qu'un seul type de transformation. Mathématiquement, cela revient à prendre pour matrice tangente, non pas la concaténation de l'ensemble des sept vecteurs tangents, mais de la restreindre au seul vecteur tangent de la transformation considérée.

Les résultats de précision par chiffre ainsi obtenus, à la fois pour l'ensemble des transformations d'une part et pour chaque transformation (prise individuellement) d'autre part, ont été représentés à la **Figure 30**. (Les résultats ont été représentés sous forme de courbes et non pas d'histogrammes par souci de lisibilité).

Nous remarquons que pour toutes les transformations, **les chiffres 1 et 6 ont une précision de plus de 98%**. Il s'agit des deux chiffres les mieux identifiés par notre algorithme. A l'inverse, **le chiffre 4 semble être le plus mal identifié** par notre algorithme ; il présente **moins de 95% de précision** pour l'ensemble des transformations considérées (individuellement).

Nous avons également calculé la précision de notre algorithme dans le cas où l'on ne retiendrait que le résultat de la transformation (considérée individuellement) la plus précise pour chaque classe ; nous appellerons ce procédé, la *sélection des meilleures transformations (individuelles) par classe*. Cela nous a permis de classer nos différentes méthodes - ainsi que les transformations (prises individuellement) - en fonction de leur précision dans le **Tableau 4**.

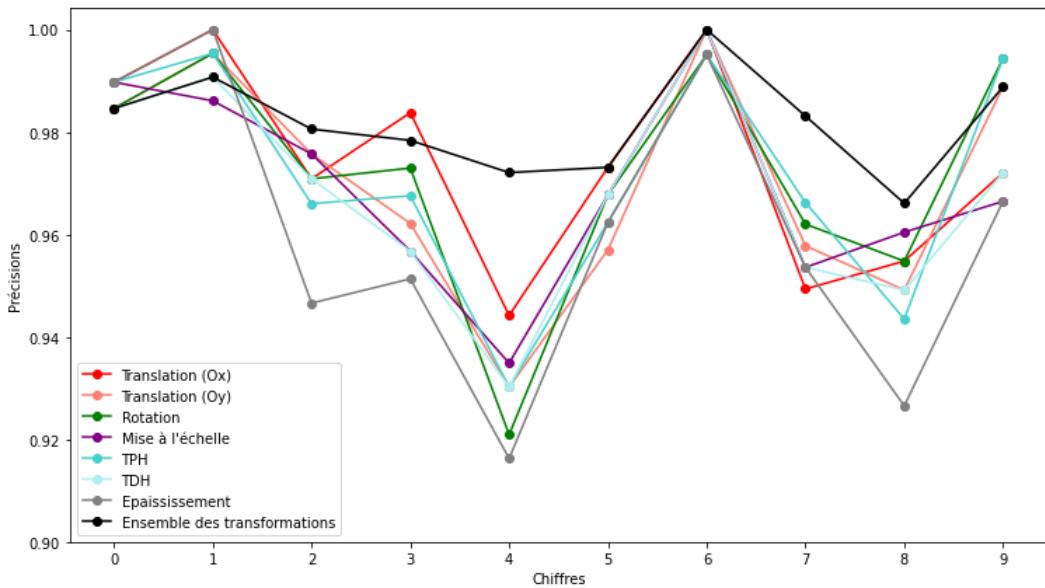


FIGURE 30 – Précision par chiffre par transformation ainsi que pour l’ensemble des transformations (avec un échantillon de 10 000 données)

Méthode ou transformation utilisée	Précision
Prise en compte de l’ensemble des transformations à la fois	98,18%
Sélection des meilleures transformations par classe	97,88%
Translation d’axe (Ox)	97,38%
Rotation	97,19%
Translation d’axe (Oy)	97,11%
Transformation parallèle hyperbolique	97,01%
Mise à l’échelle	96,91%
Transformation diagonale hyperbolique	96,76%
Epaississement	96,08%

TABLE 4 – Précision en fonction de chaque méthode ou transformation utilisée

D’après le **Tableau 4**, on remarque que contrairement à notre conjecture initiale, les translations ont des précisions plus élevées que les autres transformations, à l’exception notable de la rotation qui a, comme on l’avait supposé, une des meilleures précisions (individuelles). La méthode la plus précise, avec 98,18% de précision, est - sans surprise - celle qui consiste à prendre en compte toutes les transformations à la fois.

11.2 Analyse des estimations erronées par chiffre

11.2.1 Matrices de confusion

Dans cette sous-section, nous avons identifié, pour l’ensemble des données d’une même classe ayant été mal identifiées par une transformation individuelle donnée, les fréquences relatives des classes erronées qui lui ont été attribuées. Nous obtenons alors la **Figure 31**.

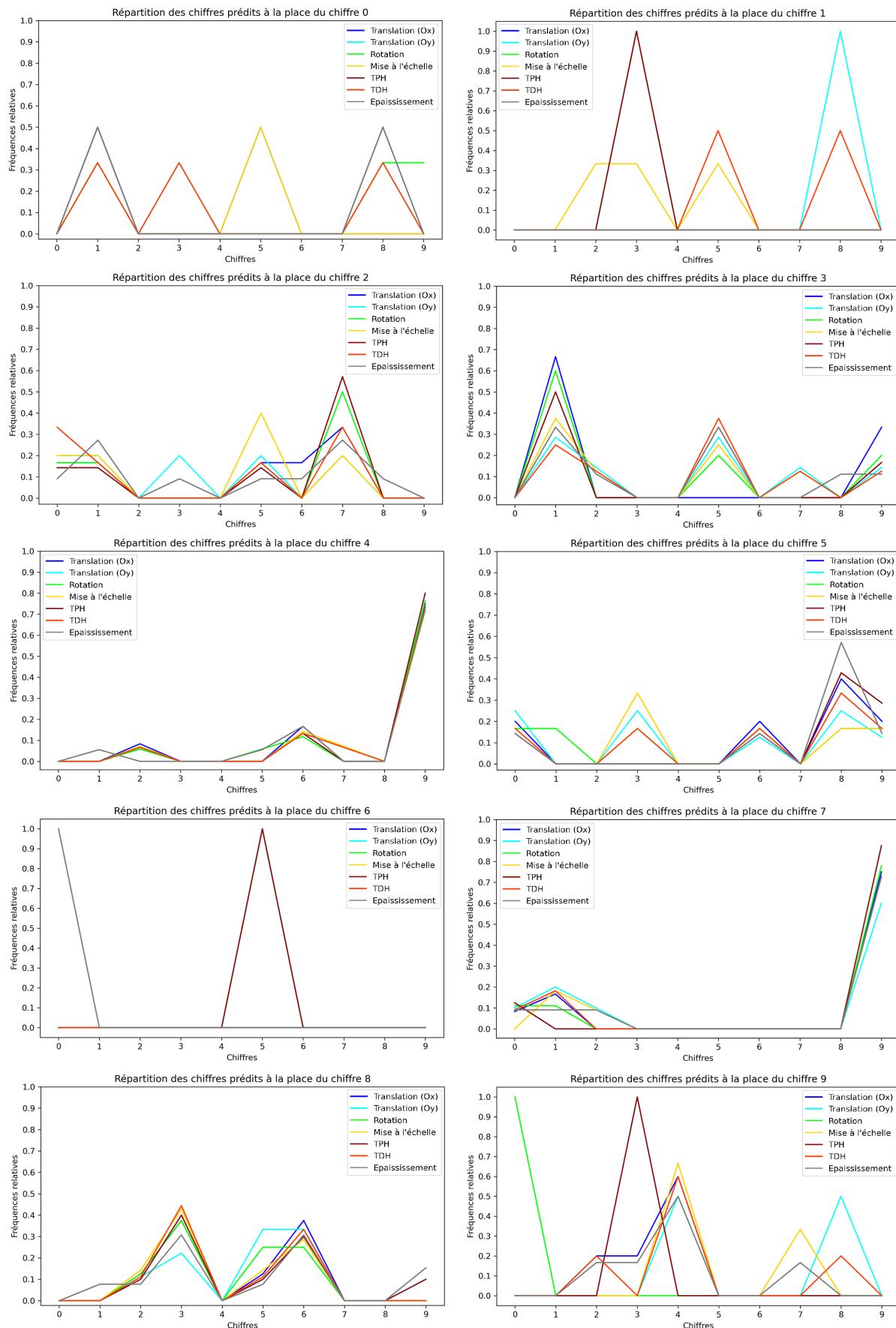


FIGURE 31 – Fréquences relatives des classes attribuées de manière erronée par la méthode de la distance tangente

Nous pouvons faire deux remarques sur ces données : tout d'abord, les chiffres 4 et 7 sont particulièrement mal identifiés, et ce pour l'ensemble des transformations, comme nous l'avons remarqué à la **Figure 30**. En examinant les matrices de confusion pour ces chiffres, en **Figure 31**, il est intéressant de noter qu'ils sont majoritairement confondus avec le chiffre 9, ce qui pourrait être dû à leur relative proximité graphique.

Par ailleurs, les chiffres souffrant le moins d'erreurs sont le 6 et le 1, mais cette fois-ci la répartition de leurs rares erreurs n'est pas la même :

- D'une part, le 6 a été faussement identifié exclusivement en 0 par l'épaississement ou en 5 par la transformation hyperbolique parallèle. (Il est donc intéressant de noter qu'un même chiffre peut être mal identifié de manière différente en fonction des transformations considérées.)
- La répartition des erreurs sur le chiffre 1 est plus variée en termes de classe, y compris parfois pour une même transformation.

La conclusion de cette analyse succincte est donc que parfois les transformations (prises individuellement) font des erreurs très similaires en nature (par exemple, en attribuant toutes la classe 9 au chiffre 7) quand parfois, au contraire, elles se trompent de manière radicalement différente.

11.2.2 Représentation de chiffres mal identifiés

Notre analyse nous a menés à trouver que trente chiffres (parmi nos 2000 données de test) ont été mal identifiés par l'ensemble des sept transformations (considérées individuellement). Nous avons représenté ces chiffres à la **Figure 32**.

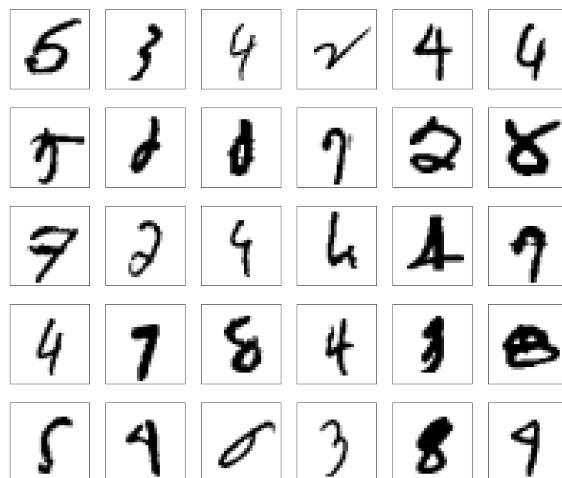


FIGURE 32 – Visualisation des trente chiffres qui ont été mal identifiés par l'ensemble de nos transformations

Pour revenir à la motivation initiale de cet algorithme, mentionnée en introduction de cette section, nous pouvons constater que les chiffres mal identifiés sont cette fois à peine reconnaissables, même pour un oeil humain. Plus particulièrement, il ne s'agit plus maintenant d'images affectées par les transformations mentionnées, mais réellement de graphies fortement différentes de la norme. Il semble donc raisonnable d'inférer que cette méthode a permis d'éliminer les erreurs dues aux déformations simples, étant donné qu'il ne reste maintenant plus que des altérations plus complexes.

11.3 Tests de l'algorithme

11.3.1 Test de la justesse de notre algorithme

Réaliser un test de surapprentissage pour ce troisième algorithme qui soit de même nature que ceux réalisés pour les deux premiers serait d'une pertinence limitée. En effet, la nature même de notre algorithme (qui classifie une donnée de test en la comparant à chaque donnée d'apprentissage) fait qu'un test de surapprentissage qui consisterait à prendre la base de test au sein de la base d'apprentissage trouvera 100% de précision. Ce résultat serait peu informatif concernant le possible degré de surapprentissage. En revanche, un tel test permet de vérifier que notre algorithme (et le code associé) fonctionne comme prévu de manière théorique (en retournant effectivement 100% de précision). Les résultats (concluants) de ce test ont été représentés au sein de la **Figure 33**.

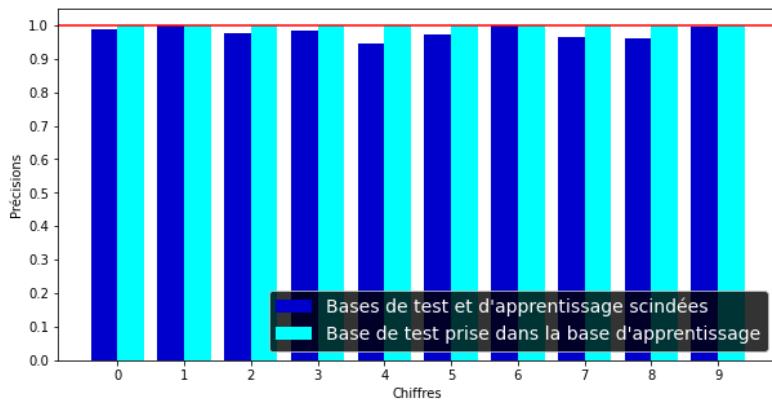


FIGURE 33 – Test de justesse de notre troisième algorithme et du code associé

11.3.2 Test de l'algorithme sur données non lissées

Nous avons établi au début de cette section l'importance d'avoir, pour l'algorithme de la distance tangente, des données "lissées" correspondant à la convolution différentiable de nos images représentées sous forme de fonctions. Il est clair d'un point de vue théorique que l'algorithme ne peut pas fonctionner correctement sans cette étape, mais il nous a paru intéressant d'effectuer une simulation en exécutant notre algorithme sans effectuer le lissage préalable des images. Les résultats de précision obtenus par chiffre (pour les transformations individuelles) sont visibles au sein de la **Figure 34**.

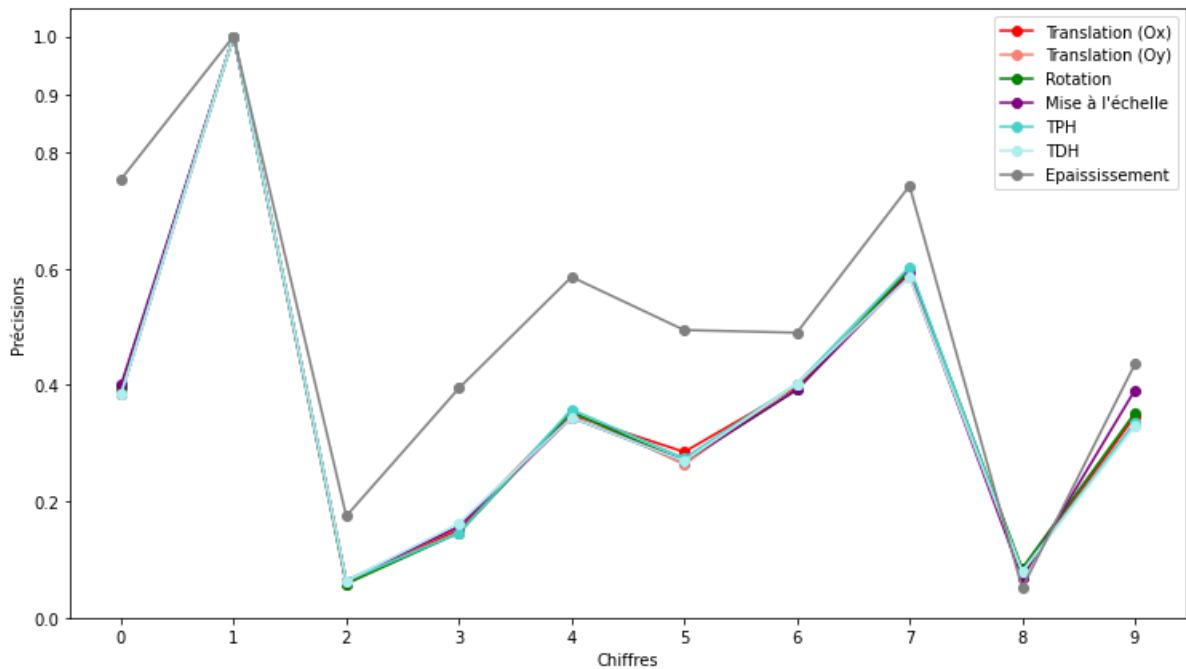


FIGURE 34 – Précision par chiffre pour chaque transformation (sans lissage)

Sans grande surprise, donc, nous avons observé des résultats de précision extrêmement bas, de l'ordre de 36% en moyenne pour la plupart des transformations avec de très fortes fluctuations entre les différents chiffres : par exemple, plus de 90% de précision pour le chiffre 1 quand le chiffre 8 a moins de 10% de précision pour l'ensemble des transformations. La disparité totale dans les données ainsi que la très basse précision moyenne semblent donc confirmer de manière claire ce à quoi nous nous attendions : l'algorithme de la distance tangente devient complètement dysfonctionnel sans le lissage et la différentiabilité artificielle de nos images considérées sous forme de fonctions.

11.3.3 Test de l'algorithme sans transformation

Remarquons que notre troisième algorithme présente en réalité deux spécificités distinctes, qu'il convient d'identifier. D'une part, il compare chaque chiffre de test à l'ensemble des chiffres d'apprentissage. D'autre part, il affine la classification en prenant en compte les sept transformations considérées.

Afin de dissocier la part de précision apportée par chacune de ces deux composantes, nous avons relancé l'algorithme mais, cette fois, sans prise en compte des transformations. Ainsi, tester l'algorithme sans transformation revient à calculer la distance euclidienne entre deux images lissées p (*de la base de test*) et e (*de la base d'apprentissage*), c'est-à-dire $\|p - e\|_2^2$ et classifier p selon l'image d'apprentissage e qui minimise cette distance. Ce test permet alors de comparer l'utilisation pour cet algorithme de la distance euclidienne et de la distance tangente (de taux d'erreur 1,82%).

La distance euclidienne nous donne un taux d'erreur de l'ordre de 3,38%. Elle a donc une précision $\frac{3,38}{1,82} \approx 1,86$ fois plus faible que celle obtenue avec la distance tangente. Leurs résultats respectifs de précision par chiffre ont été représentés dans la **Figure 35**.

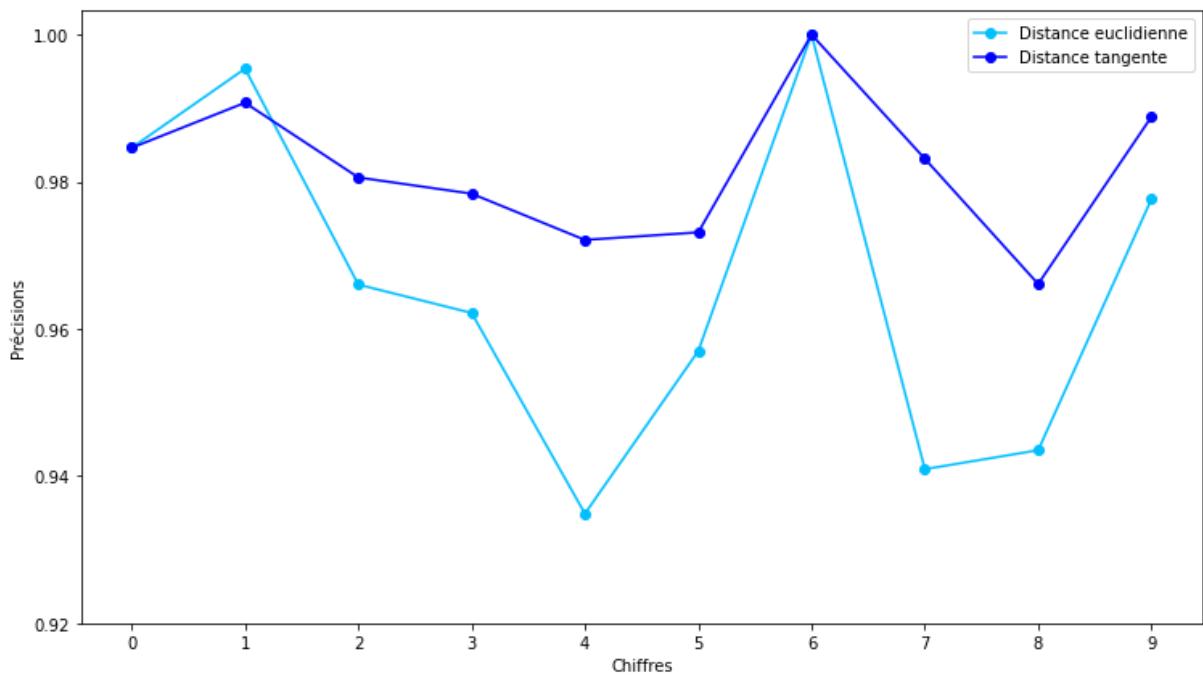


FIGURE 35 – Précision de notre algorithme associé à la distance tangente d'une part, à la distance euclidienne d'autre part

On constate sur ce graphique que la moindre précision dans le cas de la reconnaissance des chiffres 4, 7 et 8 se retrouve lorsque nous appliquons notre troisième algorithme avec distance euclidienne. Ce défaut de précision semble par ailleurs être en partie gommé par la prise en compte des transformations qu'apporte en supplément l'utilisation de la distance tangente. Il semble donc que la moins bonne classification des chiffres 4, 7 et 8 que nous avions constatée dans les sections précédentes ne soit pas une spécificité due à l'utilisation de la distance tangente mais, qu'au contraire, cette dernière contribue à gommer ce défaut.

Notons enfin que notre troisième algorithme utilisé avec la distance euclidienne - ayant une précision de 96,62% - est plus précis que la SVD qui, elle, nous donnait une précision de l'ordre de 95,56% avec un taux d'erreur de 4,44%. Notre troisième algorithme avec distance euclidienne est donc $\frac{4,44}{3,38} \approx 1,31$ fois plus précis que notre deuxième algorithme. Cependant, cette méthode de classification calcule, pour chaque chiffre de la base de test, sa distance euclidienne à l'ensemble des chiffres de la base d'apprentissage. Elle est donc très coûteuse comparée à la SVD.

11.4 Temps d'exécution

Pour ce troisième algorithme, les temps d'exécution ont constitué un enjeu bien différent de ce que nous avions rencontré avec les deux premiers algorithmes.

En effet, la compilation depuis un *jupyter notebook* sur ordinateur portable a montré ses limites, principalement lors de l'étape de classification de l'algorithme, très longue à exécuter, même pour un échantillon de 10 000 données. Nous avons ainsi dû avoir recours

à la capacité de calcul d'un ordinateur (fixe) plus puissant et avons limité notre utilisation de *jupyter notebook* afin de gagner en temps d'exécution. Il a également fallu apporter une attention particulière à l'optimisation du code (bien plus que lors de nos deux premières méthodes) et sortir de la programmation impérative à quelques reprises afin de fixer certains résultats (notamment de classification) afin de pouvoir effectuer les analyses présentées plus haut. Comme dit précédemment, cette problématique liée au coût élevé de calcul vient en grande partie du fait que cet algorithme calcule, lors de son étape de classification, pour chaque chiffre de la base de test, sa distance à l'ensemble des chiffres de la base d'apprentissage.

Ces différents éléments expliquent pourquoi nous nous sommes tenus à estimer le temps d'exécution de l'étape de prétraitement uniquement pour cet algorithme (à l'aide du même ordinateur portable que celui utilisé pour les deux premiers algorithmes, via *timeit* et *jupyter notebook* afin de garder des résultats comparables). En effet, nos tentatives d'utilisation de la fonction *timeit* lors de l'étape de classification - même lorsqu'employée depuis un ordinateur fixe puissant - se sont soldées par l'expiration de cette fonction.

Les résultats obtenus pour l'étape de prétraitement, calculés par praticité **pour un échantillon de 1000 données seulement** (200 données de test pour 800 d'apprentissage) sont les suivants :

- **Lissage des images** (étape la plus coûteuse du prétraitement) : 3 minutes et 50 secondes ;
- **Calcul des vecteurs tangents et matrices tangentes** : 1,3 secondes.

Le temps d'exécution de l'étape de prétraitement de la distance tangente est donc de presque 4 minutes, ce qui est du même ordre de grandeur que les 5 minutes 45 que nécessitait l'étape de prétraitement de la SVD mais celle-ci travaillait alors avec 70 000 données quand nous n'avons ici fait les calculs que pour 1000 données. Il semble donc que l'étape de prétraitement de notre dernier algorithme soit déjà considérablement plus coûteuse que celle de la SVD (celle-ci étant plus de 840 fois plus coûteuse que le prétraitement de la méthode des centroïdes).

12 Conclusion

Dans le cas optimal de la distance tangente (en prenant en compte l'ensemble des transformations à la fois) et avec un échantillon de 10 000 données, **la précision atteint 98,18%** ; elle est alors $\frac{3,1}{1,82} \approx 1,7$ fois plus grande que celle mentionnée par Lars Elden [1], qui est de 96,9% (obtenue lorsque cet algorithme est appliqué à la base de données *U.S. Postal Service*, composée d'images de résolution 16x16).

Cette précision de 98,18%, bien qu'obtenue à partir d'un échantillon de 10 000 données, est tout de même meilleure que celle de 95,56% obtenue dans le cas optimal de la SVD (et que celle de 82,51% obtenue dans le cas optimal de la méthode des centroïdes) avec 70 000 données. Le défaut de précision est ainsi passé de 17,49% avec le premier

algorithme à 4,44% avec le deuxième, puis à 1,82% avec notre troisième algorithme. Or, on a que : $\frac{17,49}{1,82} \approx 9,6$ et que $\frac{4,44}{1,82} \approx 2,44$. **Nous obtenons donc avec ce troisième algorithme une précision près de 9,6 fois plus grande que la méthode des centroïdes et de 2,5 fois plus grande que la SVD.**

Concernant le coût de calcul, cet algorithme est en revanche nettement plus coûteux que nos précédentes méthodes. En effet, cela est visible dès l'étape de prétraitement comme détaillé dans la section précédente et est encore renforcé lors des nombreux calculs de distance requis par l'étape de classification.

Notons que l'utilisation des transformations considérées lors de notre troisième algorithme permet, au-delà de l'intérêt propre que présente la méthode de la distance tangente, d'enrichir une base d'apprentissage. En effet, elle peut permettre d'élargir une base de données en lui ajoutant ses propres données auxquelles on a appliqué des transformations pour un certain nombre de paramètres choisis. Ainsi, la précision accrue due à la prise en compte des transformations est apportée à travers une étape de prétraitement sans avoir à l'effectuer lors de l'étape de classification. C'est pourquoi des méthodes d'enrichissement similaires constituent des outils classiques de l'apprentissage automatique. Ainsi, par exemple, la bibliothèque libre Python **Scikit-Learn** [6], dédiée à l'apprentissage automatique, propose des outils de prétraitement clé en main qui pourront effectuer des transformations proches de celles étudiées dans cette partie. Ce processus d'enrichissement des données est également très important au sein d'autres méthodes de classification. En particulier, la performance des réseaux de neurones dépend fortement de la taille, de la richesse et de la variété de la base d'apprentissage utilisée.

Enfin, il nous semble approprié de conclure cette étude par une dernière mention des réseaux de neurones qui constituent aujourd'hui les méthodes de pointe dans le domaine de la reconnaissance automatique. Une extension naturelle de ce travail de recherche serait donc de comparer les résultats de nos trois algorithmes et ceux obtenus par différents types de réseaux de neurones, des plus simples aux plus précis, tels que les réseaux convolutifs (d'après le site web de Yann LeCun [2]).

A Code Python : Premier algorithme

1. On calcule le centroïde de chaque chiffre $1 \leq i \leq 9$ dans la base d'apprentissage.

```
X,x=data_app,label_app
moy_chiff=[]
plt.figure(figsize=(15,2))
for i in range(10):
    moy=np.mean(X[x==i],axis=0)
    moy_chiff+=[moy]
    plt.subplot(1,10,i+1)
    plt.imshow(moy.reshape(28,28),cmap='gray')
```

2. Pour chaque vecteur de la base de test, on lui attribue le chiffre dont le centroïde est le plus proche par rapport à la distance choisie.

— **Distance de Minkowski** : $d(u, v) = (|u_1 - v_1|^p + \dots + |u_n - v_n|^p)^{\frac{1}{p}}$

```
def estim_minkowski(v,p):
    distances=np.array([np.linalg.norm(v-u,p) for u in moy_chiff])
    return np.argmin(distances)
```

— **Similarité cosinus** : $\cos\theta(u, v) = \frac{u^T v}{\|u\|_2 \|v\|_2}$

```
def cosine(u,v):
    return np.inner(u,v)/(np.linalg.norm(u)*np.linalg.norm(v))

def estim_cosine(v):
    distances=np.array([cosine(u,v) for u in moy_chiff])
    return np.argmax(distances)
```

B Code Python : Deuxième algorithme

1. Pour chaque chiffre i , on concatène les vecteurs représentant i dans une grande matrice D_i .

```

l_app=len(data_app)
l_test=len(data_test)
D_0=np.hstack([data_app[i].reshape(-1,1) for i in range(l_app) if
               label_app[i]==0])
D_1=np.hstack([data_app[i].reshape(-1,1) for i in range(l_app) if
               label_app[i]==1])
D_2=np.hstack([data_app[i].reshape(-1,1) for i in range(l_app) if
               label_app[i]==2])
D_3=np.hstack([data_app[i].reshape(-1,1) for i in range(l_app) if
               label_app[i]==3])
D_4=np.hstack([data_app[i].reshape(-1,1) for i in range(l_app) if
               label_app[i]==4])
D_5=np.hstack([data_app[i].reshape(-1,1) for i in range(l_app) if
               label_app[i]==5])
D_6=np.hstack([data_app[i].reshape(-1,1) for i in range(l_app) if
               label_app[i]==6])
D_7=np.hstack([data_app[i].reshape(-1,1) for i in range(l_app) if
               label_app[i]==7])
D_8=np.hstack([data_app[i].reshape(-1,1) for i in range(l_app) if
               label_app[i]==8])
D_9=np.hstack([data_app[i].reshape(-1,1) for i in range(l_app) if
               label_app[i]==9])

```

2. Soit la SVD pour D_i : $D_i = U_i \Sigma_i V_i^t$. Pour chaque $i \in [0, 9]$, on calcule U_i .

```

U_0=np.linalg.svd(D_0)[0]
U_1=np.linalg.svd(D_1)[0]
U_2=np.linalg.svd(D_2)[0]
U_3=np.linalg.svd(D_3)[0]
U_4=np.linalg.svd(D_4)[0]
U_5=np.linalg.svd(D_5)[0]
U_6=np.linalg.svd(D_6)[0]
U_7=np.linalg.svd(D_7)[0]
U_8=np.linalg.svd(D_8)[0]
U_9=np.linalg.svd(D_9)[0]

```

3. Pour un vecteur v de la base de test et pour $i \in [0, 9]$, on calcule le résidu relatif $r_{(i,k),v} = \frac{1}{\|v\|_2} \|v - U_{i,k} U_{i,k}^t v\|_2$. On classe le v au chiffre i qui minimise ce résidu.

```

def estim(v,k):
    residus=[np.linalg.norm(v-U[i][:,:k]@(np.transpose(U[i][:,:k])@v))/np.linalg.norm(v) for
             i in range(10)]
    return residus.index(min(residus))

```

C Code Python : Troisième algorithme

- On lisse les images de la base de test et de la base d'apprentissage. Pour réduire le coût du calcul, on choisit un échantillon de 8000 images pour la base d'apprentissage et un échantillon de 2000 images pour la base de test.

```
def smooth_fcl(v):
    P = np.reshape(v, (28,28))
    x, y = tuples_x, tuples_y
    return sum([P[i,j]*np.e**(-((x-i)**2+(y-j)**2)/(2*0.9**2)) for i, j in tuples])

smooth_train=[smooth_fcl(data_train[i]) for i in range(8000)]
smooth_test=[smooth_fcl(data_test[i]) for i in range(2000)]
```

- On calcule les vecteurs tangents pour chaque transformation pour tous les chiffres de nos bases d'apprentissage et de test.

- (a) Translation d'axe (Ox) :

```
def T_X(v):
    P=np.reshape(v,(28,28))
    P_=np.gradient(P)[0]
    return np.reshape(P_,(1,-1))[0]

p_x_train=[T_X(smooth_train[i]) for i in range(8000)]
p_x_test=[T_X(smooth_test[i]) for i in range(2000)]
```

- (b) Translation d'axe (Oy) :

```
def T_Y(v):
    P=np.reshape(v,(28,28))
    P_=np.gradient(P)[1]
    return np.reshape(P_,(1,-1))[0]

p_y_train=[T_Y(smooth_train[i]) for i in range(8000)]
p_y_test=[T_Y(smooth_test[i]) for i in range(2000)]
```

- (c) Rotation :

```
p_r_train=tuples_y*np.array(p_x_train)-tuples_x*np.array(
    p_y_train)
p_r_test=tuples_y*np.array(p_x_test)-tuples_x*np.array(p_y_test)
```

- (d) Mise à l'échelle :

```
p_s_train=tuples_x*np.array(p_x_train)+tuples_y*np.array(
    p_y_train)
p_s_test=tuples_x*np.array(p_x_test)+tuples_y*np.array(p_y_test)
```

- (e) Transformation parallèle hyperbolique :

```

p_TPH_train=tuples_x*np.array(p_x_train)-tuples_y*np.array(
                                         p_y_train)
p_TPH_test=tuples_x*np.array(p_x_test)-tuples_y*np.array(
                                         p_y_test)

```

(f) Transformation diagonale hyperbolique :

```

p_TDH_train=tuples_y*np.array(p_x_train)+tuples_x*np.array(
                                         p_y_train)
p_TDH_test=tuples_y*np.array(p_x_test)+tuples_x*np.array(
                                         p_y_test)

```

(g) Epaississement :

```

p_T_train=np.array(p_x_train)**2+np.array(p_y_train)**2
p_T_test=np.array(p_x_test)**2+np.array(p_y_test)**2

```

Dans le cas des transformations considérées individuellement, on prendra pour matrice tangente le simple vecteur tangent associé à la transformation considérée. Dans le cas où l'on considère l'ensemble des transformations, la matrice tangente sera la concaténation en colonne des sept vecteurs tangents.

3. Pour un chiffre de la base de test, on calcule sa distance tangente par rapport à chaque chiffre de la base d'apprentissage et on le classe selon le chiffre qui minimise cette distance.

(a) Translation d'axe (Ox) :

```

def estim_X(j): #estime l'image data_test[j]
    A=[np.hstack((np.reshape(-p_x_train[i],(-1,1)),np.reshape(
                                         p_x_test[j],(-1,1))))
        for i in range(8000)]
    b=[np.reshape(smooth_train[i]-smooth_test[j],(-1,1)) for i
        in range(8000)]
    residus=[np.linalg.lstsq(A[i], b[i], rcond=None)[1][0] for
             i in range(8000)]
    return label_train[r sidus.index(min(r sidus))]

```

(b) Translation d'axe (Oy) :

```

def estim_Y(j): #estime l'image data_test[j]
    A=[np.hstack((np.reshape(-p_y_train[i],(-1,1)),np.reshape(
                                         p_y_test[j],(-1,1))))
        for i in range(8000)]
    b=[np.reshape(smooth_train[i]-smooth_test[j],(-1,1)) for i
        in range(8000)]
    residus=[np.linalg.lstsq(A[i], b[i], rcond=None)[1][0] for
             i in range(8000)]
    return label_train[r sidus.index(min(r sidus))]

```

(c) Rotation :

```

def estim_R(j): #estime l'image data_test[j]
    A=[np.hstack((np.reshape(-p_r_train[i],(-1,1)),np.reshape(
                                         p_r_test[j],(-1,1))))
        for i in range(8000)]

```

```

b=[np.reshape(smooth_train[i]-smooth_test[j],(-1,1)) for i
    in range(8000)]
residus=[np.linalg.lstsq(A[i], b[i], rcond=None)[1][0] for
    i in range(8000)]
return label_train[r sidus.index(min(r sidus))]

```

(d) Mise à l'échelle :

```

def estim_S(j): #estime l'image data_test[j]
    A=[np.hstack((np.reshape(-p_s_train[i],(-1,1)),np.reshape(
        p_s_test[j],(-1,1)))) for i in range(8000)]
    b=[np.reshape(smooth_train[i]-smooth_test[j],(-1,1)) for i
        in range(8000)]
    residus=[np.linalg.lstsq(A[i], b[i], rcond=None)[1][0] for
        i in range(8000)]
    return label_train[r sidus.index(min(r sidus))]

```

(e) Transformation parallèle hyperbolique :

```

def estim_TPH(j): #estime l'image data_test[j]
    A=[np.hstack((np.reshape(-p_TPH_train[i],(-1,1)),np.reshape(
        p_TPH_test[j],(-1,1)))) for i in range(8000)]
    b=[np.reshape(smooth_train[i]-smooth_test[j],(-1,1)) for i
        in range(8000)]
    residus=[np.linalg.lstsq(A[i], b[i], rcond=None)[1][0] for
        i in range(8000)]
    return label_train[r sidus.index(min(r sidus))]

```

(f) Transformation diagonale hyperbolique :

```

def estim_TDH(j): #estime l'image data_test[j]
    A=[np.hstack((np.reshape(-p_TDH_train[i],(-1,1)),np.reshape(
        p_TDH_test[j],(-1,1)))) for i in range(8000)]
    b=[np.reshape(smooth_train[i]-smooth_test[j],(-1,1)) for i
        in range(8000)]
    residus=[np.linalg.lstsq(A[i], b[i], rcond=None)[1][0] for
        i in range(8000)]
    return label_train[r sidus.index(min(r sidus))]

```

(g) Epaississement :

```

def estim_T(j): #estime l'image data_test[j]
    A=[np.hstack((np.reshape(-p_T_train[i],(-1,1)),np.reshape(
        p_T_test[j],(-1,1)))) for i in range(8000)]
    b=[np.reshape(smooth_train[i]-smooth_test[j],(-1,1)) for i
        in range(8000)]
    residus=[np.linalg.lstsq(A[i], b[i], rcond=None)[1][0] for
        i in range(8000)]
    return label_train[r sidus.index(min(r sidus))]

```

(h) Pour toutes les transformations :

```
#On calcule les matrices tangentes pour les chiffres:
```

```

matrix_T_train=[np.hstack([p_x_train[i].reshape(-1,1)]+[
    p_y_train[i].reshape(-1,1)]
+[p_r_train[i].reshape(-1,1)]
]+[p_s_train[i].reshape(-1
,1)]+[p_TPH_train[i].
reshape(-1,1)]+[p_TDH_train
[i].reshape(-1,1)]+[p_T_train[i].reshape(-1,1)]
) for i in range(8000)]

matrix_T_test=[np.hstack([p_x_test[i].reshape(-1,1)]+[p_y_test [
i].reshape(-1,1)]+[p_r_test
[i].reshape(-1,1)]+[p
_s_test[i].reshape(-1,1)]+
[p_TPH_test[i].reshape(-1,1)
]+[p_TDH_test[i].reshape(-
1,1)]+[p_T_test[i].reshape(
-1,1)]) for i in range(2000
)]

```

```

def estim(j): #estime l'image data_test[j]
    A=[np.hstack((-matrix_T_train[i],matrix_T_test[j])) for i
        in range(8000)]
    b=[np.reshape(smooth_train[i]-smooth_test[j],(-1,1)) for i
        in range(8000)]
    residus=[np.linalg.lstsq(A[i], b[i], rcond=None)[1][0] for
            i in range(8000)]
    return label_train[r sidus.index(min(r sidus))]

```

Références

- [1] Lars Elden. *Matrix Methods in Data Mining and Pattern Recognition*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, 2007
- [2] Yann LeCun, Corinna Cortez, Christopher C.J. Burges. *THE MNIST DATABASE of handwritten digits*
<http://yann.lecun.com/exdb/mnist/>
- [3] Muhammad Ramzan, Hikmat Ullah Khan, Shahid Mehmood Awan, Waseem Akhtar, Mahwish Ilyas, Ahsan Mahmood and Ammara Zamir. *A Survey on using Neural Network based Algorithms for Hand Written Digit Recognition*. International Journal of Advanced Computer Science and Applications(IJACSA), 9(9), 2018
<http://dx.doi.org/10.14569/IJACSA.2018.090965>
- [4] Li Deng, *The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web]* IEEE Signal Processing Magazine, vol. 29, no. 6, pp. 141-142, Nov. 2012, doi : 10.1109/MSP.2012.2211477.
- [5] Patrice Simard, Yann Le Cun, John Denker, Bernard Victorri. *Transformation invariance in pattern recognition - tangent distance and tangent propagation*. Neural Networks : Tricks on the Trade, 1524, Springer-Verlag, pp.239-274, 1998, Lectures Notes in Computer Science
- [6] Section 6.3. *Preprocessing data* du site web officiel de la bibliothèque libre Python, Scikit-learn
<https://scikit-learn.org/stable/modules/preprocessing.html>