

Reconnaissance de caractères

Marin Costes
Jonathan Mboko
Manuel Mockus
Octave Mestoudjian

5 mai 2019

Table des matières

1	Introduction	3
2	Pré-traitement des données	4
3	Première méthode : calcul des écarts à la moyenne	5
3.1	Théorie	5
3.2	Pratique	6
4	Deuxième méthode : utilisation de la décomposition en valeurs singulière	8
4.0.1	Théorie	8
4.0.2	Pratique	9
4.0.3	Paramètres	11
5	Distance tangente	13
5.0.1	Théorie	13
5.0.2	Pratique	15
5.0.3	Algorithme	18
5.0.4	Tests	18
6	Test globaux	20
6.0.1	Écarts a la moyenne	20
6.0.2	Décomposition en valeurs singulières	20
6.0.3	Distance tangente	21
7	Conclusion	22

Chapitre 1

Introduction

Le cœur du projet est d'écrire un code permettant de reconnaître automatiquement des caractères manuscrits, le but étant de se familiariser avec des méthodes usuelles de reconnaissance de forme en les implémentant pour un cas particulier et en comprenant la théorie. Nous utiliserons comme référence le travail de Lars Elden "Matrix Methods in Data Mining and Pattern Recognition [1].

Les caractères à analyser seront ici des chiffres de 1 à 9 sous forme de vecteurs de taille $(28 \times 28, 1)$ à valeurs entières comprise entre 0 et 255 : chaque vecteur représente une image de taille 28×28 en nuance de gris (0 = noir, blanc = 255). Nous disposons d'une base de données de 70000 chiffres manuscrits labellisés stockés sous le format décrit précédemment.

Nous avons choisi comme langage de programmation pour notre projet le Python.

Chapitre 2

Pré-traitement des données

La base de donnée étant un fichier matlab nous avons dans un premier temps dû la traduire dans un format utilisable sous python. Nous avons choisi comme format de stockage une liste de liste de vecteurs : le k^{eme} élément de cette liste est une liste des vecteurs correspondants au chiffre k . Nous avons ensuite divisé la base de donnée en quatre afin que chacun des membre du groupe puisse effectuer des tests indépendants.

Les algorithmes étudiés relèvent du domaine de l'apprentissage automatique : après une phase d'apprentissage où l'on donne à l'algorithme des images et leurs chiffres respectifs comme références, on a une phase de traitement où l'algorithme doit reconnaître le chiffre à partir de l'image seule. Pour avoir des résultats pertinents, il faut donc distinguer clairement les données utilisées pour l'apprentissage et celles utilisées pour le traitement. Le choix que nous avons fait est donc pour chaque base de donnée individuelle de diviser les données en 20% fixe de données utilisées exclusivement pour les tests et de prendre pour donnée d'apprentissage aléatoirement 50% des données restantes.

Chapitre 3

Première méthode : calcul des écarts à la moyenne

3.1 Théorie

La première méthode étudiée est une approche assez intuitive du problème. Dans un premier temps, à partir du jeu de données destiné à l'entraînement, pour chaque chiffre on crée une image moyenne de toutes les image du chiffre : le centre de gravité (centre de masse, isobarycentre) de tous les vecteurs de ce chiffre.

Une image inconnue est alors classée en cherchant l'image moyenne telle que l'écart de ressemblance entre l'image à tester et cette image soit minimum.

On définit l'écart de ressemblance entre deux images comme étant la distance dans une certaine norme choisie entre les deux vecteurs x et y de \mathbf{R}^{784} correspondant aux images :

$$d(x, y) = \|x - y\|$$

Le choix de la norme et son influence sur les résultats est détaillé dans la partie pratique.



FIGURE 3.1 – Images moyennes de chaque chiffre

3.2 Pratique

Pour analyser l'efficacité de notre programme nous avons choisi d'observer deux types de données, le taux de réussite global de notre programme pour différentes normes, et le taux de réussite de notre programme pour chaque chiffre avec une norme fixe.

Nous avons comparé les taux de réussite de l'algorithme -pour chaque chiffre et en moyenne- pour les normes de Minkowski ($\|v\|_n = \sqrt[n]{\sum_k |v_k|^n}$) entre 1 et 10 et pour la norme infinie ($\|v\|_\infty = \max_k |v_k|$) (table 3.1).

TABLE 3.1 – Pourcentage de chiffres correctement identifiés par norme de Minkowsky

Norme	0	1	2	3	4	5	6	7	8	9	Total
1	84	99	41	66	70	35	80	77	35	75	67
2	89	97	74	82	85	72	87	83	74	77	82
3	90	90	82	81	84	81	88	81	84	77	84

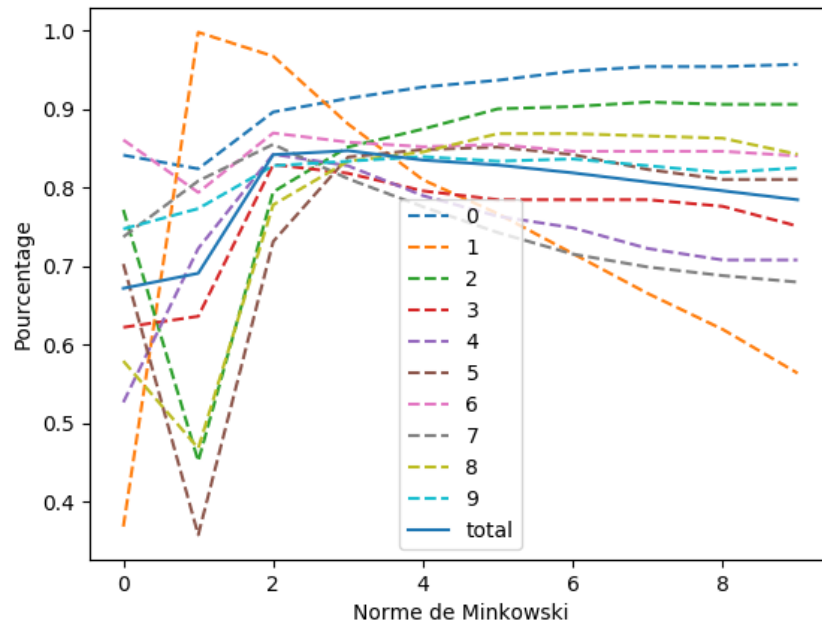


FIGURE 3.2 – Performance de l'algorithme pour chaque chiffre et en moyenne en fonction de la norme de Minkowsky (la valeur 0 en abscisse représente la norme infinie).

On remarque qu'en moyenne les normes 2 et 3 sont les plus performantes

et aussi celles pour lesquelles les résultats obtenus pour les différents chiffres sont les plus proches.

La distance associée à la norme infinie ne prenant en compte que l'écart maximal composante par composante entre deux vecteurs, appliquée à la comparaison d'image elle est inefficace car l'écart maximal n'est pas représentatif de l'ensemble de l'image. De plus, quand k augmente, le comportement de la norme k de Minkowski tend vers celui de la norme infinie, ce qui se vérifie dans les résultats obtenus.

Pour la norme 1, on observe de mauvais résultats moyens avec beaucoup de disparités entre les différents chiffres (Figure 3.1). On a donc cherché à savoir lorsqu'un chiffre est mal reconnu avec quel chiffre il a été le plus confondu (Table 3.2).

On remarque pour la norme 1 que les 1 sont toujours correctement identifiés, mais que les autres chiffres sont souvent confondus avec des 1.

TABLE 3.2 – Représente pour les normes 1,2 et 3 et pour un chiffre i donné avec quel autre chiffre le chiffre i est le plus souvent confondu par l'algorithme

Norme	0	1	2	3	4	5	6	7	8	9
1	1	1	1	1	1	1	1	1	1	1
2	5	8	1	8	9	3	1	9	1	4
3	5	8	8	8	9	3	2	9	3	4

Chapitre 4

Deuxième méthode : utilisation de la décomposition en valeurs singulière

4.0.1 Théorie

Étant donné les résultats décevants de notre première méthode, il est nécessaire d'utiliser une autre technique qui prenne en compte les variations entre les différentes images représentant un même chiffre, cette technique repose sur un outil mathématique très puissant : la décomposition en valeurs singulières (SVD par ses initiales en anglais *Singular Value Decomposition*). Il s'agit ici dans un premier temps, de créer pour chaque chiffre (on parlera ici uniquement des trois pour alléger l'écriture) une matrice A de 784 lignes, comportant autant de colonnes que le nombre de 3 dont on dispose dans notre base d'entraînement. Chaque colonne de cette matrice représente une image de 3. On dispose alors d'un sous espace vectoriel de \mathbf{R}^{784} , engendré par les colonnes de A (on appellera cet espace vectoriel "l'espace des trois"). Or le théorème de décomposition en valeurs singulières, nous permet de décomposer cette matrice ainsi :

$$A = \sum_{i=1}^{784} \sigma_i u_i v_i^T$$

Les u_i sont alors une base orthonormée de l'espace des trois. De plus lors d'une SVD, la base est ordonnée de manière à ce que les σ_i forment une suite décroissante. Les premiers vecteurs u de la SVD sont alors représentatifs de l'espace vectoriel des 3, étant donné que toutes les matrices de la somme sont de norme 1. On peut donc construire un espace des trois réduit ayant pour base (orthonormée) les premiers vecteurs u de la SVD. Ce sous-espace

est de faible dimension par rapport à l'espace de dimension 784 dans le quel il vit et il est donc très peu probable qu'il intersecte le sous-espace relatif à un autre chiffre, ce qui garantit que les différents espaces réduits seront bien distincts dans le calcul à venir.

Maintenant qu'on dispose d'un espace vectoriel des 3 réduit (doté d'une base orthonormée), on peut facilement calculer la distance, pour la norme deux, du chiffre non identifié (c'est à dire un vecteur de \mathbf{R}^{784}) à cet espace (cela revient à un calcul des moindres carrés). Si on effectue ce calcul pour l'espace correspondant à chaque chiffre on peut alors labelliser l'image, en constatant de quel espace vectoriel elle est le plus proche.

4.0.2 Pratique

Pour cette méthode, nous avons effectué une classification (clustering) plus fine des résultats : si lors de l'analyse d'une image, on ne peut l'identifier de façon décisive à un chiffre, celle-ci est rejetée, ce qui permet d'améliorer le taux d'identification correct parmi les images non rejetées.

Plus précisément, une image z sera rejetée si le minimum $m1$ de l'ensemble des $\|(I - U_k U_k^T)z\|_2$ (les moindres carrés pour chaque chiffre) n'est pas "significativement" plus petit que le deuxième plus petit élément de cet ensemble $m2$ (deux bases de chiffres pourraient alors correspondre à l'image). Pour définir ce "significativement" nous avons introduit un seuil compris entre 0.5 et 1 : une image sera rejetée si après le calcul de ses moindres carrés $m1 > seuil * m2$. Plus le seuil est bas plus l'on est exigeant : avec un seuil de 1 aucune image n'est rejetée et avec un seuil de 0.5 l'évaluation d'une image est jugée concluante seulement si $m1 \leq 0.5m2$.

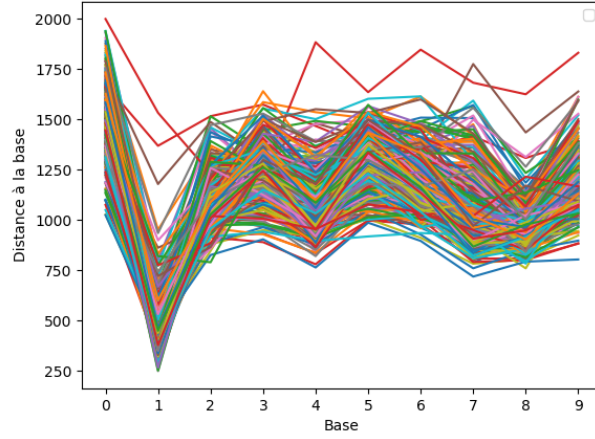


FIGURE 4.1 – Distance pour chaque image de 1 aux espaces vectoriels (de dimension 20) représentant les différents chiffres. On remarque que les images sont bien plus proches de l'espace des uns que des autres espaces et il y a donc très peu d'erreur.

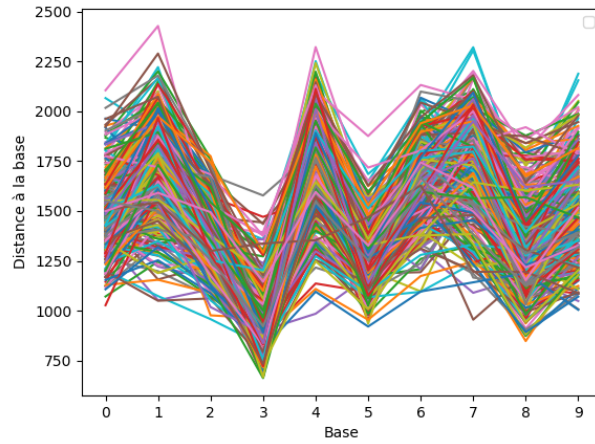


FIGURE 4.2 – Distance pour chaque image de 3 aux espaces vectoriels (de dimension 20) représentant les différents chiffres. On peut remarquer des pics secondaires pour les espaces des 5 et des 8 ce qui signifie que les 3 sont plus difficiles à identifier clairement que les 1 et seront généralement pris pour des 5 et des 8 par l'algorithme lorsqu'il se trompera.

4.0.3 Paramètres

Nous avons cherché à optimiser cette deuxième méthode en jouant sur deux paramètres : le nombre de vecteurs de base de U choisis k et le seuil. Le premier jeu de test a été effectué pour des valeurs du seuil entre 0.9 et 1 (des valeurs inférieures du seuil donnant un taux de rejet trop élevées env. 20%) et une valeur de k (nombre de vecteurs de base) entre 1 et 7 (Figure 4.1).

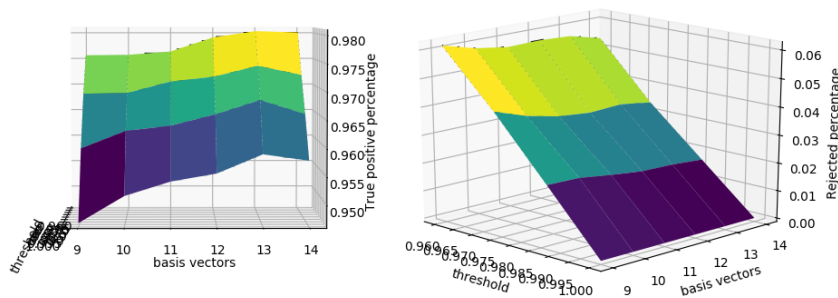


FIGURE 4.3 – A gauche, pourcentage d’images correctement identifiées et a droite, pourcentage d’images rejetées en fonction de nombre de vecteurs de base et du seuil de distance minimale

On remarque que quelque soit la base, l’efficacité de l’algorithme évolue approximativement de la même manière en fonction de la valeur du seuil. On peut donc par la suite se permettre d’étudier indépendamment l’influence du seuil et celle du nombre de bases.

Seuil

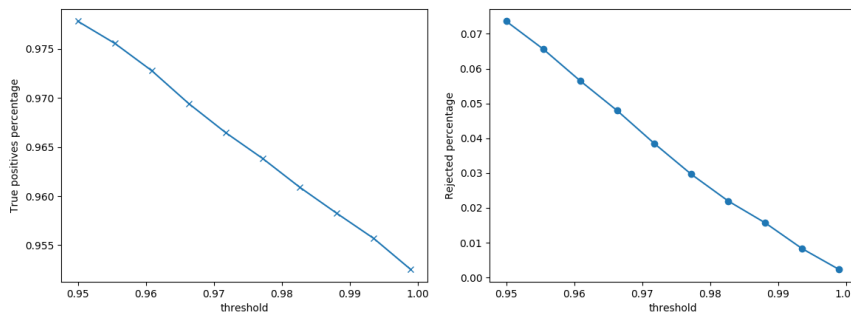


FIGURE 4.4 – Pourcentage de vrai positifs et de rejetés en fonction du niveau du seuil.

On remarque dans la figure 4.2 le pourcentage de vrais positifs et le pourcentages de rejetés sont proportionnelles et linéairement décroissantes en fonction du seuil. On n'a donc pas de choix optimal objectif pour le seuil, on choisira donc le seuil en fonction de la tolérance à l'erreur et le taux de rejet acceptable.

Nombre de vecteurs de base

On remarque que l'efficacité de l'algorithme est croissante en fonction du nombre k de vecteurs de bases pour ceux observés. Cependant après notre analyse théorique, on peut dire que si k devient trop grand, les espaces vectoriels qu'on associe à chaque chiffre (étant de dimension k) ont de fortes chances de s'intersecter (Figure 4.3). Il est donc intéressant de chercher la valeur de k optimale pour notre algorithme.

En faisant varier k entre 1 et 30, on remarque que l'efficacité de l'algorithme se stabilise à partir de $k=20$. De plus le nombre de vecteurs de base k choisi n'influe pas significativement sur la complexité des calculs : en effet seul le calcul de $I - U_k U_k^T$ dépend de k et il est effectué une seule fois pour chaque chiffre. On fixe donc par la suite $k=20$.

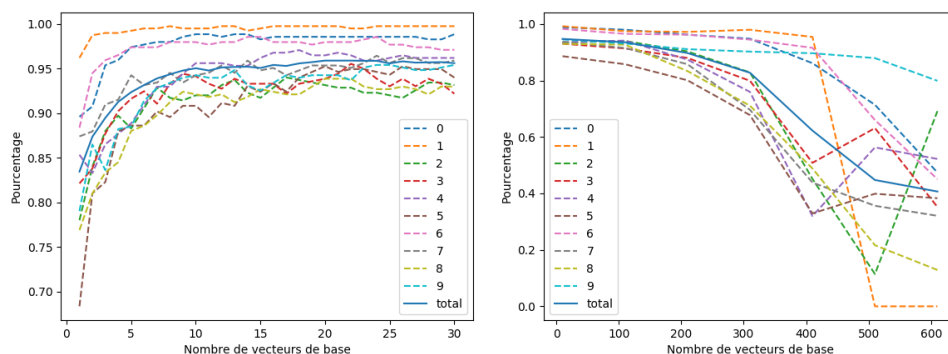


FIGURE 4.5 – Performance pour chaque chiffre et en moyenne en fonction du nombre de vecteurs de base. A gauche lorsque le nombre de vecteurs de base est encore faible l'algorithme devient plus performant en augmentant ce paramètre. A droite on assiste à l'effondrement du modèle lorsque le nombre de vecteurs de base est trop élevé et que les espaces relatifs à chacun des nombres s'intersectent.

Chapitre 5

Distance tangente

5.0.1 Théorie

On souhaite désormais affranchir notre algorithme des erreurs liées aux petites variations usuelles que nos chiffres peuvent connaître lorsqu'un humain les écrits. En effet il est courant dans une base de données réelle que les chiffres soient légèrement décalés, dilatés ou encore épaissis. Afin d'explicitier la méthode plus simplement, on va d'abord s'intéresser uniquement à rendre notre algorithme insensible à la translation horizontale (en X).

Si on considère une image I comme un point dans \mathbf{R}^{784} , l'ensemble des I translatés est une courbe continue dans \mathbf{R}^{784} . On peut décrire l'ensemble des points de cette courbe, comme l'image de \mathbf{R} par la fonction $f(x) = s(I, x)$, où $s(I, \alpha)$ est l'élément de \mathbf{R}^{784} correspondant à I décalé de α sur l'axe des X. P étant un autre élément de \mathbf{R}^{784} , on cherche alors à établir la distance minimale entre un translaté de I et de P. Idéalement on aimerai donc définir la distance suivante :

$$d(I, P) = \min_{\alpha, \beta \in \mathbf{R}} \|s(I, \alpha) - s(P, \beta)\|$$

Cependant il serait inutile d'essayer d'implémenter cette distance, puisque son calcul est très complexe et qu'on ne s'intéresse ici qu'à de petites translations. On va donc utiliser des approximations à l'ordre 1 des courbes des translatés, valables pour des petites valeurs de α et β . On effectue un développement de Taylor et on obtient :

$$s(I, \alpha) \approx s(I, 0) + \frac{ds}{d\alpha}(I, 0) * \alpha = I + \frac{ds}{d\alpha}(I, 0) * \alpha$$

Ainsi on peut définir une nouvelle distance plus simple, à peu près équivalente au voisinage de 0 :

$$d_2(I, P) = \min_{\alpha, \beta \in \mathbf{R}} \|I + \frac{ds}{d\alpha}(I, 0) * \alpha - P - \frac{ds}{d\beta}(P, 0) * \beta\|$$

Or $\frac{ds}{d\alpha}(I, 0)$ est indépendant de α , on peut donc le noter T_I et ainsi réécrire notre distance ainsi :

$$d_2(I, P) = \min_{\alpha, \beta \in \mathbf{R}} \|(I - P) - (-T_I \quad T_P) \begin{pmatrix} \alpha \\ \beta \end{pmatrix}\|$$

A condition que la norme utilisée soit la norme euclidienne, le calcul de cette distance est un problème des moindres carrés. On le résout grâce au théorème suivant : soit A une matrice de rang plein de décomposition en valeurs singulières $A = U\Sigma V^T$, le problème des moindres carrés $\min_x \|Ax - b\|_2$ a pour unique solution

$$x = V\Sigma^{-1}U_1^T b$$

Ici $A = (-T_I \quad T_P)$ et $b = (I - P)$.

Généralisons maintenant cette méthode au cas où l'image I peut avoir subi n transformations continues distinctes. L'espace affine des éléments correspondant aux I transformés, noté C_I est alors de dimension n , car il est engendré par n courbes de dimension 1 (on peut à partir de I se déplacer dans \mathbf{R}^{784} suivant n courbes pour obtenir un I transformé, on voit bien qu'on peut décrire cet espace avec n paramètres). Soit P un autre élément de \mathbf{R}^{784} on souhaiterait comme précédemment pouvoir calculer la distance minimale entre C_I et C_P mais c'est impossible. On utilise donc à nouveau l'hypothèse que les transformations sont de petite ampleur, et on va approximer C_I et C_P par leur plans tangents en 0. Soit α dans \mathbf{R}^n , on définit la fonction tel que $s(I, \alpha)$ soit l'élément de \mathbf{R}^{784} correspondant à I ayant subi pour tout i inférieur à n , la i -ème transformation avec un coefficient α_i . C_I est alors l'image par $f(x) = s(I, x)$ de \mathbf{R}^n , un développement de Taylor à l'ordre 1 permet alors d'explicitier le plan tangent à C_I en 0. Au voisinage de 0 on a :

$$f(x) = s(I, x) \approx I + \sum_{i=1}^n \frac{ds}{d\alpha_i}(I, 0) \alpha_i$$

On définit alors la distance tangente entre I et P comme la distance euclidienne minimale entre un point du plan tangent de I et un point du plan tangent de P :

$$d_{\text{tangente}}(I, P) = \min_{\alpha, \beta \in \mathbf{R}^n} \|I - P + (-T_I \quad T_P) \begin{pmatrix} \alpha \\ \beta \end{pmatrix}\|$$

où T_I est la matrice des dérivées partielles de $s(I, \alpha)$ selon α .

Il ne reste désormais plus qu'à réappliquer notre premier algorithme en utilisant cette distance tangente, afin d'obtenir un code insensible aux modifications humaines usuelles.

5.0.2 Pratique

Pour implémenter l'algorithme on considérera les transformations suivantes : la translation (en x et en y), la rotation, l'agrandissement et le rétrécissement, l'étirement (horizontal, vertical et diagonal), et l'épaississement. On implémente chaque transformation et sa dérivée.

Par la suite on considérera les images comme des fonctions de deux variables $p(x, y)$ et on notera les dérivées partielles par rapport à x et y respectivement $p_x = \frac{dp}{dx}$ et $p_y = \frac{dp}{dy}$. Les images étant discrètes on calcule les dérivées par différences finies.

Translation

La translation en x de α_x pixels est donnée par la fonction

$$s(p, \alpha_x)(x, y) = p(x + \alpha_x, y)$$

En utilisant le développement de Taylor à l'ordre 1, on obtient :

$$\frac{d}{d\alpha_x}(s(p, \alpha_x)(x, y))|_{\alpha_x=0} = \frac{d}{d\alpha_x}p(x + \alpha_x, y)|_{\alpha_x=0} = p_x(x, y)$$

De façon analogue on obtient la translation en y et sa dérivée.

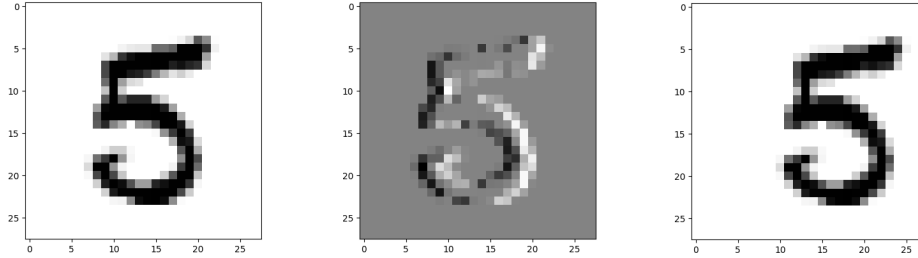


FIGURE 5.1 – Une image d'un 5, sa dérivée en x et l'image translatée.

Rotation

La rotation d'angle α_r est donnée par la fonction

$$s(p, \alpha_r)(x, y) = p(x \cos \alpha_r + y \sin \alpha_r, -x \sin \alpha_r + y \cos \alpha_r)$$

Et sa dérivée par :

$$\frac{d}{d\alpha_r}(s(p, \alpha_r)(x, y)) = (-x \sin \alpha_r + y \cos \alpha_r)p_x - (x \cos \alpha_r + y \sin \alpha_r)p_y$$

d'où

$$\frac{d}{d\alpha_r}(s(p, \alpha_r)(x, y))|_{\alpha_r=0} = yp_x - xp_y$$

où les dérivées sont évaluées en (x, y)

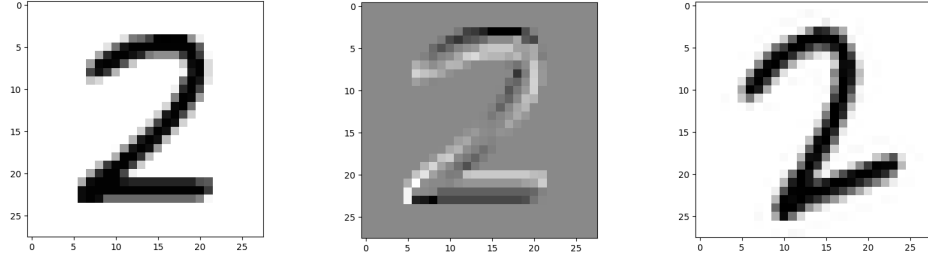


FIGURE 5.2 – Une image d’un deux, sa dérivée de rotation et l’image ayant subi une rotation.

Agrandissement et rétrécissement

Les fonctions de changement d’échelle sont donnée par :

$$s(p, \alpha_s)(x, y) = p((1 + \alpha_s)x, (1 + \alpha_s)y)$$

Et sa dérivée par :

$$\frac{d}{d\alpha_s} s(p, \alpha_s)(x, y)|_{\alpha_s=0} = xp_x + yp_y$$

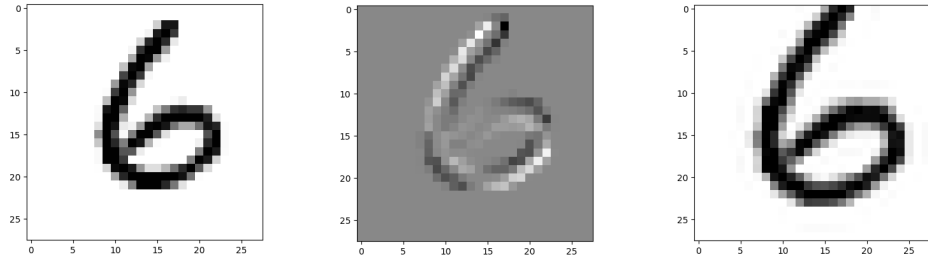


FIGURE 5.3 – Une image d’un six, sa dérivée d’agrandissement et l’image agrandie.

Étirement

L’étirement horizontal ou vertical est donné par :

$$s(p, \alpha_p)(x, y) = p((1 + \alpha_p)x, (1 - \alpha_p)y)$$

Et sa dérivée par :

$$\frac{d}{d\alpha_p} s(p, \alpha_p)(x, y)|_{\alpha_p=0} = xp_x - yp_y$$

L'étirement diagonal est donné par :

$$s(p, \alpha_p)(x, y) = p(x + \alpha_p y, y + \alpha_p x)$$

Et sa dérivée par :

$$\frac{d}{d\alpha_p} s(p, \alpha_p)(x, y)|_{\alpha_p=0} = yp_x + xp_y$$

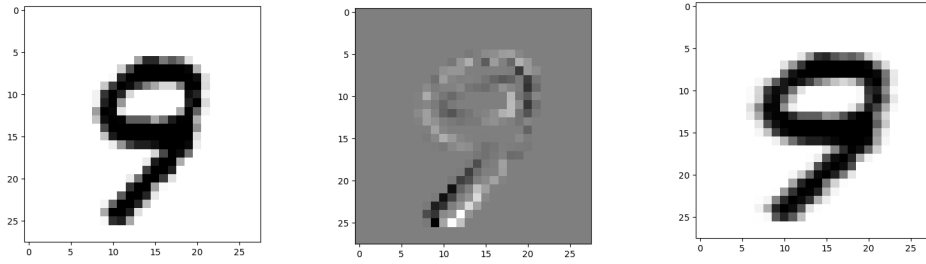


FIGURE 5.4 – Une image d'un neuf, sa dérivée d'étirement et l'image étirée.

Épaississement

Pour l'épaississement on considérera seulement la dérivée [1, p. 127] :

$$(p_x)^2 + (p_y)^2$$

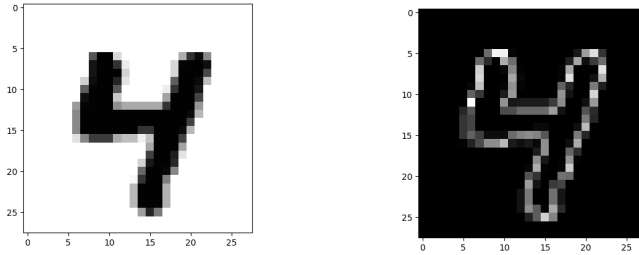


FIGURE 5.5 – Une image d'un quatre et sa dérivée d'épaississement.

Comparaison entre la distance euclidienne et la distance tangente

Notre algorithme diffère de celui des écarts à la moyenne uniquement par la distance utilisée. Il est donc intéressant de comparer la distance d'un 3 modifié, à la moyenne de chaque chiffre, en fonction de la distance choisie et de l'intensité de la modification. Dans le cas où la transformation est l'identité, les distances sont égales car le 3 utilisé est le 3 moyen.

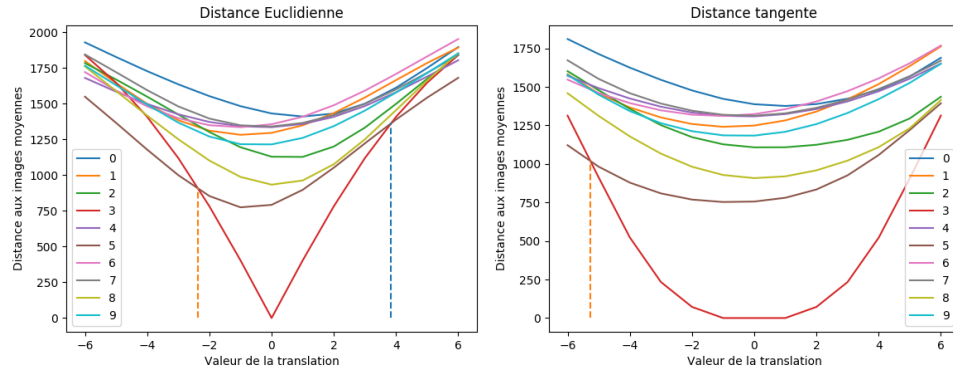


FIGURE 5.6 – Distances des images moyennes à l’image moyenne des 3 translaté horizontalement en fonction de la valeur translation.

On remarque qu’avec la distance euclidienne, un décalage de 2 pixels vers la gauche de l’image suffit à avoir une mauvaise identification, alors que pour la distance tangente, on identifie correctement l’image même après des décalages allant jusqu’à 6 pixels.

5.0.3 Algorithme

On calcule pour chaque chiffre la moyenne des images du jeu d’entraînement (comme dans le premier algorithme) puis on calcule la matrice des dérivées de chaque transformation appliquée à chaque image moyenne. Pour tester une image, on calcule de même la matrice des dérivées pour celle-ci. On l’identifie ensuite au chiffre tel que la distance tangente à l’image moyenne associée soit minimale.

5.0.4 Tests

Pour vérifier l’efficacité de l’algorithme, nous avons implémenté les différentes transformations à l’aide des modules python de traitement d’image (scipy et openCV) et généré des jeux de tests où chaque image a subi un nombre de transformations aléatoirement choisies. On a ensuite comparé l’efficacité de cet algorithme avec les deux autres algorithmes.

Nous avons créé trois bases de test, avec différents niveaux de ”bruit” : chaque image subit une, deux ou trois transformations choisies aléatoirement.

TABLE 5.1 – Performance des trois algorithmes sur une base ayant subi une transformation

Algorithme	0	1	2	3	4	5	6	7	8	9	Total
1	72.0	85.3	65.4	62.6	74.7	59.7	75.0	73.9	57.1	61.4	69.0
2	97.3	94.4	94.0	94.8	96.9	94.7	95.1	95.0	91.3	91.6	94.6
3	84.8	92.5	73.2	78.4	86.4	75.9	86.0	83.7	66.7	72.9	80.3

TABLE 5.2 – Performance des trois algorithmes sur une base ayant subi deux transformations

Algorithme	0	1	2	3	4	5	6	7	8	9	Total
1	58.9	72.8	57.4	51.8	67.5	53.1	61.2	64.7	46.0	46.3	58.2
2	94.0	87.5	91.2	91.2	93.3	93.6	92.3	91.9	88.8	85.3	90.9
3	73.4	82.4	67.5	69.1	79.9	71.7	73.9	75.2	57.4	62.1	71.4

TABLE 5.3 – Performance des trois algorithmes sur une base ayant subi trois transformations

Algorithme	0	1	2	3	4	5	6	7	8	9	Total
1	47.0	66.9	50.7	41.3	61.2	47.6	50.7	54.2	37.4	35.3	49.5
2	92.9	82.4	90.3	83.8	90.6	89.5	89.7	88.4	85.4	77.3	87.1
3	62.8	76.8	61.7	58.9	73.6	65.0	66.2	66.2	48.2	52.8	63.4

On remarque que la distance tangente permet d'améliorer substantiellement la résistance au bruit de l'algorithme des écarts à la moyenne. Cependant la méthode de décomposition en valeurs singulières offre de meilleurs résultats, même sur une base bruitée.

Chapitre 6

Test globaux

Pour tester la performance des algorithmes implémentés on a effectué une série de tests sur l'ensemble de la base de données : on a réservé 20% des images de la base de données comme ensemble de test, puis nous avons sélectionné aléatoirement la moitié des images restantes comme ensemble d'entraînement.

Pour évaluer la complexité en temps de calcul, on a exécuté chaque algorithme sur un ensemble de 1000 images avec les résultats suivants : l'algorithme des écarts à la moyenne prend 0.1 secondes, celui de la décomposition en valeurs singulières 37.7 secondes et celui de la distance tangente 1137.9 secondes (environ 20 minutes).

6.0.1 Écarts a la moyenne

Pour le premier algorithme nous avons testé 50 fois, obtenant en moyenne des performances (taux d'identification correct) de 83% pour la norme 2 et 84% pour la norme 3 (Table 5.1) avec des résultats variées en fonction des chiffres analysés.

TABLE 6.1 – Performance Algorithme 1, pour les normes 2 et 3

Norme	0	1	2	3	4	5	6	7	8	9	Total
2	88.2	96.4	77.2	81.7	83.9	70.3	87.9	85.7	75.8	79.5	83
3	90.2	88.6	84.6	80.7	82.5	80.7	88.0	82.1	82.2	79.9	84

6.0.2 Décomposition en valeurs singulières

L'algorithme de décomposition en valeurs singulières étant significative-ment plus coûteux quant au temps de calcul, on a effectuée 30 tests, obtenant une performance moyenne de 96% avec un minimum de 94% (Table 5.2).

TABLE 6.2 – Performance Algorithme 2 avec 20 vecteurs de base

Chiffre	0	1	2	3	4	5	6	7	8	9	Total
Performance	98.9	99.2	94.6	94.8	97.1	94.3	97.9	94.8	93.9	94.4	96.0

En ajoutant un seuil de ressemblance minimale, on est capable de mettre de coté les chiffres les plus difficiles à identifier (Tables 5.3 et 5.4). si on accepte un taux de rejet d'environ 5% on peut atteindre un pourcentage d'identification correcte d'environ 98%.

TABLE 6.3 – Performance et taux de rejet de l'Algorithme 2 avec 20 vecteurs base et un seuil de 0.99, pour les chiffres de 0 à 9 et en moyenne.

	0	1	2	3	4	5	6	7	8	9	Total
Performance	99.0	99.3	95.1	95.5	97.4	95.2	98.1	95.5	94.8	95	96.6
Rejetées	0.2	0.1	1.1	1.6	0.7	1.9	0.4	1.4	1.7	1.3	1.0

TABLE 6.4 – Performance et taux de rejet de l'Algorithme 2 avec 20 vecteurs base et un seuil de 0.95, pour les chiffres de 0 à 9 et en moyenne.

	0	1	2	3	4	5	6	7	8	9	Total
Performance	99.5	99.6	96.7	97.7	98.7	97.9	98.9	97.5	97.5	97.4	98.2
Rejetées	1.2	0.6	5.1	7.6	4.1	9.9	2.5	6.7	9.2	6.5	5.3

6.0.3 Distance tangente

L'algorithme de distance tangente étant très coûteux en temps de calcul (environ 40 fois plus lent que le deuxième en 10000 fois plus lent que le premier), on a effectué 10 tests sur la base de données obtenant une performance moyenne de 89.6% (Table 6.5).

Cette performance est plus élevée que celle du premier algorithme, mais reste très inférieure à celle de l'algorithme de décomposition en valeur singulière. Bien que l'algorithme des distances tangentes soit intéressant du point de vu théorique, il ne sera pas utilisé en pratique à cause de son temps d'exécution trop élevé pour des résultats moins bons que ceux du second algorithme.

TABLE 6.5 – Performance algorithme distance tangente pour les chiffres de 0 à 9 et moyenne.

	0	1	2	3	4	5	6	7	8	9	Total
Performance	94.9	99.6	79.2	92.5	90.3	83.6	94.5	90.9	80.7	88.0	89.6

Chapitre 7

Conclusion

En comparant les résultats obtenus avec ceux de l'ouvrage de référence (Elden, 2007) on remarque que l'on a de meilleures performances avec les deux premiers algorithmes. Cela est dû à deux facteurs significatifs : la base que nous avons utilisée est environ 7 fois plus grande, et les images traitées ont une résolution plus élevée (28×28 par rapport à 16×16 pixels dans l'ouvrage de référence).

Pour l'algorithme des distances tangentes, Elden compare les images à tester à chaque image du jeu d'entraînement. Bien que cela donne de bons résultats, pour l'ensemble de données utilisé et les ressources informatiques à notre disposition, cette implémentation est irréalisable. Nous avons donc implémenté la distance tangente comme une amélioration de l'algorithme des écarts à la moyenne. Si l'on disposait d'une plus grande puissance de calcul, on pourrait essayer d'implémenter cet algorithme comme une amélioration de la méthode de la svd (en calculant la distance avec l'espace vectoriel des trois au lieu de celle avec la moyenne des trois dans le corps de l'algorithme).

Nos tests globaux nous permettent de conclure que pour la reconnaissance des chiffres, l'algorithme basé sur la décomposition en valeurs singulières est celui des trois algorithmes que l'on choisira en pratique. Il donne en effet les meilleurs résultats pour un temps de calcul restant raisonnable.

Bibliographie

- [1] L. Elden, “Matrix methods in data mining and pattern recognition,” *SIAM Fundamentals of Algorithms*, 2007.
- [2] P. S. Yann Le Cun John Denker Bernard Victorri, “Transformation invariance in pattern recognition - tangent distance and tangent propagation,” in *Neural Networks : Tricks on the Trade*, pp. 239–274, Springer-Verlag, 1524 ed., 1998.