Encodage de caractéristiques de documents administratifs

Eiko Dekkers, Luc Enée, Adam Germain et Evan L'Huissier

Rhône, Université Lumière Lyon 2, 5 Avenue Pierre Mendès, Bron, 69500, France.

Résumé

Dans le contexte actuel de l'évolution technologique, de plus en plus de documents se retrouvent numérisés. Il est donc devenu encore plus important de pouvoir vérifier l'authenticité de ces documents. Dans cette optique, nous avons travaillé sur une solution d'encodage et d'authentification de documents. Notre approche consiste à encoder chaque caractère scanné, à l'aide d'une méthode alliant squelettisation et encodage de Freeman. L'encodage devant être résistant à de multiples impressions et scans, un pré-traitement particulier permet d'améliorer la précision de notre programme.

Mots clés: Traitement d'image, Freeman, Squelettisation, Encodage

1 Introduction

La falsification de documents est un problème grandissant dans un contexte où les scans de documents sont de plus en plus communs. En effet, il est dur de vérifier l'authenticité d'un document. Il est possible de réaliser un hachage d'un document numérique afin de prévenir la modification non autorisée dudit document, mais ces méthodes ne fonctionnent pas sur des documents imprimés puis scannés, car l'impression et le scanne créé des défauts qui changent le digest (code généré par le hachage). Notre idée est de proposer une méthode de hachage permettant de vérifier l'authenticité d'un document qui fut imprimé puis scanné plusieurs fois. Nous allons pour cela nous inspirer du code de Freeman pour générer l'encodage de chaque lettre du document afin de générer un hash permettant la reconnaissance du caractère.

2 Etat de l'art

De nombreux papiers traitent du problème de la sécurité des documents hybrides, notamment un papier paru en 2014 [1] propose de remplacer le hash du document (sous forme d'image) directement par le hash des données extraites de celui-ci (à l'aide d'OCR), cette méthode, bien que présentant une précision de 99,83%, la probabilité d'un faux positif est de 53% ce qui est trop élevé.

Une deuxième méthode est celle dite des "crossing numbers" paru en 2021 [2], elle propose de vérifier les points de croisements des caractères d'une manière semblable à la reconnaissance d'empreintes digitales. Un autre document [3] utilise aussi la position de ces points de croisement pour caractériser un document. Il existe plusieurs papiers de recherches qui utilisent la méthode de Freeman pour faire de l'OCR (optical character recognition) que ce soit en versions imprimées ou manuscrite. L'OCR de texte imprimé en elle-même ne nous intéresse pas particulièrement, mais pour la reconnaissance manuscrite, la logique est similaire à la nôtre : c.-à-d. Détecter les caractères semblables malgré de légères différences (parallèle entre l'écriture pour la version manuscrite et les défauts d'impression/scan pour notre recherche).

Une des pistes pour l'utilisation de Freeman est un papier de l'université de Tikrit [3]. Dans ce papier, il est question d'utiliser le code de Freeman afin d'encoder les caractères manuscrits de l'alphabet arabe. Le code de Freeman est généré à l'aide des contours, puis il récupère le nombre d'occurrences de chaque chiffre. Il retire ensuite les chiffres à faible occurence (dans l'exemple seules les occurrences simples sont retirées, dans leurs tests, ils sont allés jusqu'à cinq) afin de normaliser les résultats. La reconnaissance de caractères isolés utilisant cette méthode est fiable à 95% plusieurs papiers utilise ce traitement sur le code de Freeman généré avec de très bons résultats [4].

Le second papier [5] utilise la même méthode de Freeman mais remplace la normalisation par une fonction de comparaison de chaîne de caractères calculant la distance selon le nombre de substitutions, d'insertion et de suppression. Les résultats de cette méthode se montrent efficace pour des caractères montrant de faibles variations entre eux.

Certains utilisent le code de Freeman comme base pour une autre méthode, notamment une méthode basée sur une vectorisation des caractères [6] qui permet de décrire la forme d'une lettre à l'aide de 28 formes. Ils obtiennent alors un code plus court. D'autres utilisent de l'apprentissage supervisé [7], permettant de faciliter le processus de reconnaissance de caractère.

3 Methode proposé

La méthode proposée pour un encodage de documents administratifs est la suivante : on encode chaque caractère du document grâce à un encodage de Freeman.

Cet encodage est réalisé après le prétraitement et la squelettisation de l'image. Nous ne nous occuperons pas de l'extraction des caractères du document que l'on souhaite authentifier, mais seulement de l'encodage des caractères individuellement. La plupart des fonctions de traitement d'image sont issues de la bibliothèque *opencv-python*, sauf l'encodage de Freeman et la fonction qui retire les serifs.

3.1 Prétraitement

Le prétraitement est assez important, car il va permettre de réduire les défauts produits par l'impression et le scan. Le prétraitement est effectué dans cet ordre :

- un filtre moyenneur, pour uniformiser les défauts d'impression
- un seuillage
- une inversion des couleurs
- une fermeture pour corriger les défaut restants

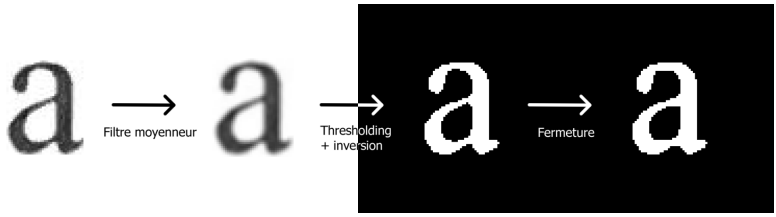

Fig. 1: Etapes du pré-traitement sur une image de la lettre 'a' obtenue après une impression et scan

Ci-dessus, en *Fig.1*, on peut observer les différentes étapes du pré-traitement qu'on applique. L'image initiale en niveaux de gris (dus à l'impression et au scan) est transformée en une image noire ou blanc nette.

3.1.1 Filtre moyenneur

Le filtre moyenneur permet d'éliminer les artéfacts produits par le scan. En effet, quand on squelettise un caractère, les défauts dus à l'impression peuvent donner un squelette très déformé comparé au squelette du caractère initial. On utilisera la fonction `cv2.filter2D()` avec un noyau de taille 3.

3.1.2 Seuillage et inversion

Le seuillage permet de passer l'image en noir ou blanc, selon un paramètre seuil. Nous ne ferons pas varier ce paramètre, car un seuil de 180 paraît adéquat dans toutes les situations. On utilisera la fonction `cv2.threshold()`.

Ensuite, on veut inverser l'image afin que le caractère soit en blanc sur fond noir, car la méthode de squelettisation considère les pixels blancs comme objet.

3.1.3 Fermeture

La fermeture est une opération morphologique basique de traitement d'image.

Elle est utile pour boucher les éventuels trous générés par le scan d'un document, mais pas pour retirer le bruit ou de petits objets parasites. Pour ces types de parasites, on compte sur le fait que le filtre moyenneur suivi du thresholding les élimine. On peut même enchaîner plusieurs dilatations suivies du même nombre d'érosions afin d'augmenter l'effet, on réfèrera à ce nombre en parlant de l'ordre de la fermeture. On utilisera les fonctions `cv2.dilate()` et `cv2.erode()`, avec un noyau de taille 3 également.

3.2 Squelettisation

La fonction de squelettisation utilisée est `skeletonize()` de scikit-image. Elle produit une image avec un squelette blanc sur fond noir, d'un seul pixel de large.

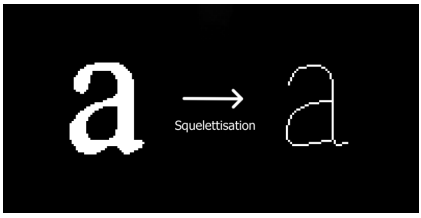

Fig. 2: Squelettisation, après un pré-traitement, sur la lettre 'a' obtenue après une impression et scan

Ci-dessus, en *Fig.2*, on a un exemple d'une squelettisation réussie, i.e le squelette est fidèle à ce qu'on pourrait attendre d'une lettre 'a'.

Après squelettisation, il peut être intéressant de retirer les serifs. En effet, notre encodage de Freeman est assez sensible aux extrémités du squelette, donc les retirer limiterait les erreurs. Pour cela, une méthode similaire à celle du document [8] qui emploie aussi de la squelettisation pour encoder les caractéristiques du document. Leur méthode consiste à repérer les jonctions et extrémités dans le squelette du caractère, à évaluer la distance entre ces différents points et à retirer les jonctions "extrémités" trop proches d'une

autre jonction.

Notre méthode repère seulement les extrémités et remonte le long du squelette jusqu'à atteindre une autre jonction, et s'arrête si le segment est trop long (auquel cas l'extrémité n'est pas considérée comme un serif).

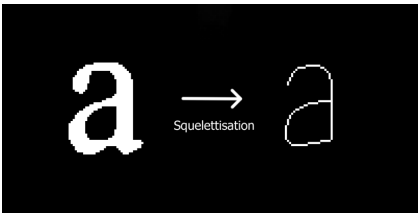

Fig. 3: Squelettisation avec serifs retirés, après un pré-traitement, sur la lettre 'a' obtenue après une impression et scan

Sur l'exemple en *Fig.3*, on voit que le serif en bas à droite de la lettre 'a' n'est plus présent, mais que l'extrémité du haut du 'a' n'a pas été considérée comme un serif car trop longue; elle est donc toujours présente dans le squelette.

3.3 Encodage de Freeman

L'encodage de Freeman est habituellement utilisé pour détecter les contours d'un objet dans une image : à partir d'un point de départ, on parcourt le contour, et suivant la direction, on ajoute un chiffre au code.

7	0	1
6		2
5	4	3

Fig. 4: Représentation des 8 voisins du pixel gris et de leur ordre

Les codes de Freeman sont donc des chaines de caractères composés des chiffres de 0 à 7. Dans *Fig.4*, on voit comment sont agencé les voisins d'un pixel, et surtout dans quel ordre ils sont évalués.

Dans notre cas, on veut utiliser la même idée mais sur un objet qui est un squelette (1 pixel de large). Nous avons donc une version modifiée de l'algorithme, qui parcours le squelette jusqu'à une extrémité, et on reprend un autre point de départ jusqu'à que tous les pixels du squelette soient parcourus. Le code de Freeman est crée de la même façon, et le point de départ

est une extrémité du squelette (quand elle existe).

Pour avoir une idée de l'allure d'un code de Freeman, voici celui du squelette obtenu en *Fig.3* : **0101011122212222223233344 3444444444444444444444444444456666566666566666677700000010 0101112121212222122222**. Cependant, on peut aussi se rendre compte qu'avoir l'encodage de Freeman du squelette tel quel n'est pas très utile et qu'il faut donc trouver la manière de les comparer.

3.4 Comparaison entre les codes de Freeman

Après avoir encodé des caractères, on veut pouvoir comparer et mesurer les différences entre eux, en particulier entre un même caractère scanné zéro, une et deux fois.

On peut proposer une mesure avec par exemple la distance de Levenshtein, aussi appelée distance d'édition. Elle permet de calculer la distance entre deux chaînes de caractères, ce qui permet de l'appliquer facilement aux codes de Freeman qu'on obtient avec notre algorithme.

C'est la méthode que nous avons choisi d'employer. Les comparaisons sont effectuées entre les caractères numériques (sans défauts) et ceux scannés plusieurs fois. Une fois qu'on obtient la mesure des différences pour un certain ensemble de caractères, on peut comparer ces différences avec celles d'un autre ensemble de caractères scannés un nombre différent de fois, et ainsi évaluer la pertinence du pré-traitement appliqué.

4 Résultats

4.1 Données utilisées

Dans cette section, nous allons appliquer le pré-traitement et la squelettisation sur une base de données de caractères imprimés et scannés un différent nombre de fois, avec différentes résolutions. Les caractères sont en taille de police 12, en police est *Times New Roman*.

Il est composé d'images numériques des caractères, de ces mêmes caractères imprimés puis scannés à 300 dpi (auquel on se réfèrera par **PS300**), à 600 dpi (**PS600**), et imprimés et scannés deux fois à 600 dpi (**2PS600**). Il y a 10 images pour chaque caractère dans chaque catégorie, ce qui nous fait un total de 260 images de contrôle (les images numériques des caractères) et de 780 images imprimées et scannées un différent nombre de fois. Des marges ont été rajoutées aux images afin que la taille soit de 100 pixels par 100 pixels.

On voit dans la *Fig.5* que même si la lettre reste reconnaissable, l'ensemble de lettres **2PS600** a fortement subi les effets des impressions et des scans, comparé aux ensembles **PS300** et **PS600**. On peut s'attendre à avoir de

Fig. 5: Un exemplaire de la lettre 'a' du dataset. Dans cet ordre : **numérique**, **PS300**, **PS600**, **2PS600**

grandes différences entre les encodages avec le caractère numérique.

Par la suite, nous allons mesurer les différences entre les encodages de Freeman avec différentes options de pré-traitement, afin de prouver qu'un pré-traitement bien pensé peut changer beaucoup de chose pour avoir un encodage fiable et robuste vis-à-vis des successions d'impressions et de scans.

Quand ça n'est pas précisé, les comparaisons sont faites par défaut avec filtre moyenneur, fermeture d'ordre 2 et avec les serifs retirés. On comparera les résultats avec ceux de la *Fig.6* ci-dessous :

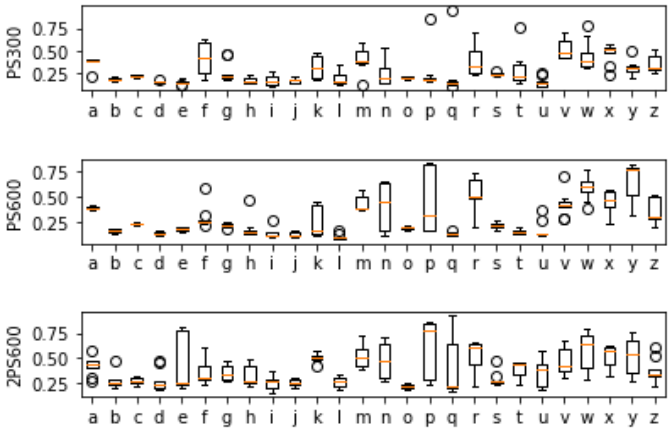

Fig. 6: Différences en avec le pré-traitement complet

4.2 Comparaison sans filtre moyenneur

Dans *Fig.7* ci-dessous, on voit que les caractères de **PS600** ont bien moins de différences en moyenne que ceux des ensembles **PS300** et **2PS600**, même si certains caractères ont des différences qui restent élevées, aux alentours des 50% d'erreur. Ces caractères sont *'h'*, *'n'*, *'m'*, *'v'* et *'w'*.

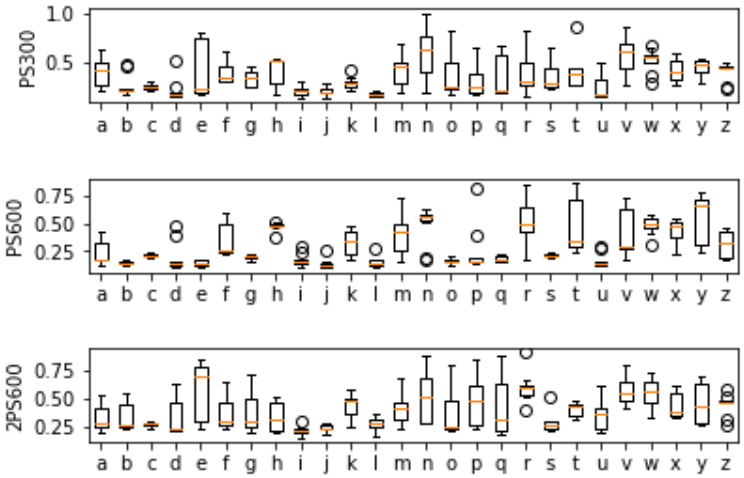

Fig. 7: Diagramme à barre représentant la différence relative entre l'encodage des caractères de chaque ensemble et celui des caractères numériques, sans filtre moyenneur dans le pré-traitement

4.3 Comparaison sans fermeture

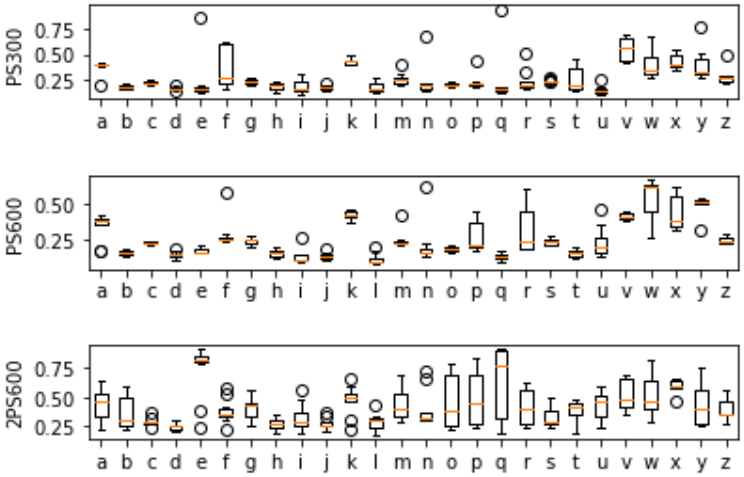

Fig. 8: Différences sans fermeture dans le pré-traitement

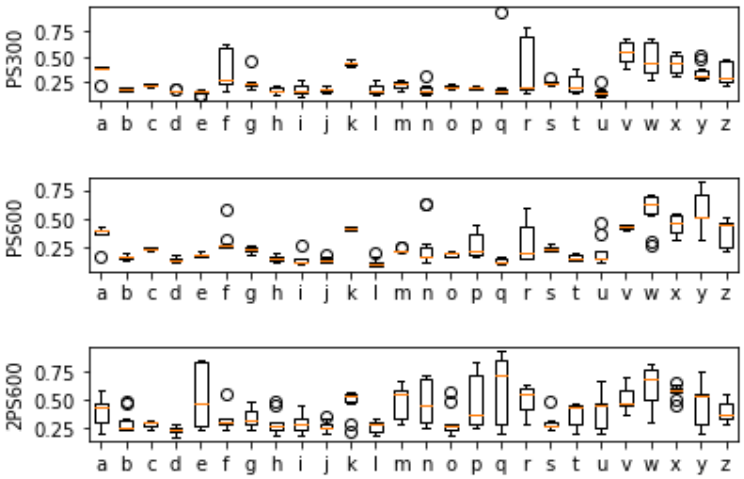

Fig. 9: Différences avec une fermeture d'ordre 1

Avec *Fig.8*, on voit que la fermeture permet par exemple de fortement réduire l'erreur chez quelques caractères comme le *'e'* dont la moyenne des différences passe d'environ 75% à moins de 50% dans l'ensemble **2PS600** (comparé à *Fig.6*, qui emploie une fermeture d'ordre 2 avec les autres paramètres identiques).

Cependant, on peut aussi remarquer que la fermeture d'ordre 2 a aussi tendance à faire apparaître des extrêmes, par exemple quelques caractères de **PS600** se retouvent avec une différence avec l'ensemble numérique au-dessus des 75%, alors qu'avant la fermeture toutes les différences sont inférieures à 75%. D'une manière générale, l'impact de ce pré-traitement a l'air neutre, des erreurs sont amoindries, mais d'autres apparaissent.

La fermeture d'ordre 2 (*Fig.6*) donne des résultats assez volatils, il est donc important de se demander si son utilisation est pertinente pour l'authentification de documents. En effet, faire apparaître des valeurs extrêmes tout comme les retirer aura tendance à générer de faux positif tout comme des faux négatifs, deux résultats qui sont à l'opposé de ce que nous cherchons à faire.

En revanche, la fermeture d'ordre 1 (*Fig.9*) a l'air d'être le juste milieu, surtout pour les caractères de **PS300** et **PS600**.

4.4 Comparaisons sans retirer les serifs

Quand on observe *Fig.10* ci-dessous, on remarque que les différences des ensembles **PS300** et **PS600** sont extrêmement réduites par rapport aux différences

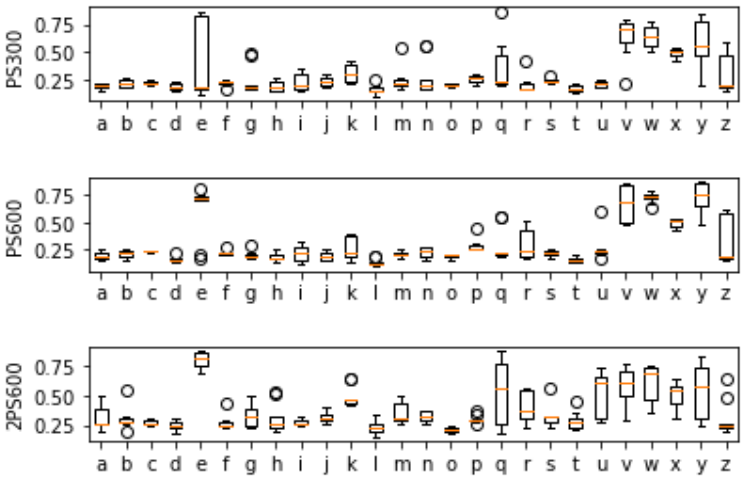

Fig. 10: Différences en gardant les serifs

obtenues sur des squelettes auxquels on a retiré les serifs (*Fig.6*). En revanche, certains caractères ne sont pas ou peu affectés par la présence ou non des serifs.

4.5 Resultats avec un choix de paramètres pertinents

Enfin, on a vu que les meilleurs résultats en moyenne étaient donnés par le filtre moyenneur, une fermeture d'ordre 1, et en gardant les serifs (chaque paramètre réglé ainsi individuellemen). Ci-dessous, en *Fig.11*, voici la combinaison de ces paramètres :

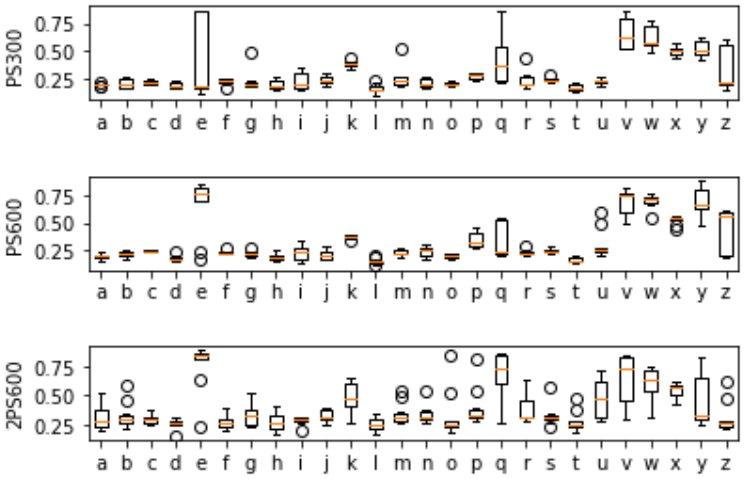

Fig. 11: Différences en gardant les serifs, avec filtre moyenneur et fermeture d'ordre 1

On observe en *Fig.11* que les ensembles de caractères **PS300** et **PS600** présentent relativement peu d'extrêmes différences. L'ensemble **2PS600** présente quant à lui toujours une variance plus prononcée. On voit aussi que certains caractères ne sont toujours pas ou peu affectés par le pré-traitement; ceux de la fin de l'alphabet, ou *'e'* et *'q'*.

5 Conclusion et perspectives

Pour conclure, notre méthode permet d'encoder chaque caractère d'un document qu'on voudrait authentifier. Les étapes de pré-traitement que nous avons utilisé sont dans l'ensemble pertinentes :

- le filtre moyenneur apporte d'une manière générale une baisse des différences entre les encodages, ce qui en fait un élément essentiel d'un bon pré-traitement.
- le seuillage et l'inversion des couleurs sont deux éléments essentiels du pré-traitement, à cause de la fonction de squelettisation de scikit-image. Une possibilité serait de tester beaucoup de seuils pour essayer d'avoir le meilleur, mais dans notre cas un seuil de 180 apportait de bons résultats.
- la fermeture était sans doute le paramètre le plus sensible du pré-traitement que nous avons effectué, et il s'avère qu'une fermeture d'ordre 1 est plus pertinent qu'une fermeture d'ordre 2.
- retirer les serifs après la squelettisation n'a en revanche pas apporté grand chose, si bien que dans le résultat final nous n'avons pas appliqué cette opération.

Enfin, il convient d'ajouter que le pré-traitement ne change pas en fonction du caractère étudié (on ne cherche pas à identifier un caractère, mais plutôt à retrouver les mêmes caractéristiques entre les différentes successions d'impressions et de scans). Ainsi, le pré-traitement est parfois inefficace, voire désavantageux pour certains caractères, entre autres *'e'*, *'q'*, *'v'* et *'w'*, *'x'*, *'y'* et *'z'*.

Afin d'améliorer le programme, il faudrait dans un premier temps corriger le pré-traitement des serifs sur certains caractères. Et ensuite bien sûr d'adapter la méthode sur un document complet et non sur chaque caractère isolé comme c'est le cas ici. Il faudrait penser une solution de reconnaissance des lettres dans le but de pouvoir adapter le pré-traitement en fonction du caractère détecté.

Il peut aussi être intéressant de considérer la normalisation des codes générés. La méthode la plus pertinente serait celle utilisée dans [3] et dans [4] qui consiste simplement à retirer les directions à faibles occurrences. Cela nous permettrait de retirer les petites impuretés pour ne garder que les formes pertinentes et obtenir le même code pour la lettre scannée et l'originale et donc un code unique et propre a chaque caractère permettant un hachage plus

fiable. Sinon, utiliser la suppression des faibles occurrences et la création d'un code utilisant le nombre d'apparitions de chaque chiffre restant, ce qui permet de rendre les résultats encore plus homogènes, cette méthode est d'ailleurs utilisée dans [3].

Pour finaliser ce projet, il faut trouver une méthode de hachage qui rendrait la comparaison suffisamment fiable. On pourrait penser à une méthode de fuzzy hash (qui consiste à avoir un hash qui change peu aux changements légers). Cependant, comme le montre [1], le fuzzy hash n'est pas assez fiable pour détecter les modifications : un seul scan suffirait à le classifier comme frauduleux. Pour ceci, il faut soit un algorithme de hachage permettant de réellement différencier le faux du vrai ou comme mentionné précédemment un code de Freeman qui est identique et unique pour une lettre comme dans[4], ce qui n'est pas le cas pour notre méthode.

Une perspective possible pourrait être aussi d'utiliser l'apprentissage supervisé [7] qui semble donner des résultats très satisfaisants.

Il est important de noter aussi que notre méthode s'applique sur les données textuelles des documents et non sur le reste du contenu (photos, graphes, etc.) donnant une possibilité de falsification. Il faut donc la coupler à une autre méthode pour assurer une authentification totale et fiable.

Nous tenons à exprimer notre plus profonde gratitude à Iuliia Tkachenko et Pauline Puteaux pour leur contribution à notre projet de recherche. Leur expertise a été indispensables pour l'avancement de ce projet. Nous sommes particulièrement reconnaissants pour leur temps et leur engagement à chaque étape de la recherche.

References

[1] Eskenazi, S., Gomez-Krämer, P., Ogier, J.-M.: When document security brings new challenges to document analysis. HAL Open science (2014). https://doi.org/10.1007/978-3-319-20125-2_10

[2] Puteaux, P., Tkachenko, I.: Crossing number features: from biometrics to printed character matching. iwcdf 2021 - 3rd international workshop on computational document forensics. HAL Open science, 437–450 (2021). https://doi.org/10.1007/978-3-030-86198-8_31

[3] Abed, M.A.: Freeman chain code contour processing for handwritten isolated arabic characters recognition. College of Computers Sciences Mathematics – University of Tikrit – Iraq (2012)

[4] Althobaiti, H., Lu, C.: A survey on arabic optical character recognition and an isolated handwritten arabic character recognition algorithm using encoded freeman chain code. Department of Computer and Information Sciences Towson University Towson, USA (2017)

[5] Pradhan, S.K., Sarkar, S., Das, S.K.: A character recognition approach using freeman chain code and approximate string matching (2013)

[6] Nasien, D., Yulianti, D., Omar, F.S., Adiya, M.H., Desnelita, Y., Chandra, T.: New feature vector from freeman chain code for handwritten roman character recognition. 2018 2nd International Conference on Electrical Engineering and Informatics (ICon EEI 2018), Batam - Indonesia, 16th-17th October 2018 (2018)

[7] Alam, M.M., Kashem, D.M.A.: A complete bangla ocr system for printed chracters (2010)

[8] Tan, L., Sun, X.: Robust text hashing for content-based document authentication. Information Technology Journal 10(8), 1608–1613 (2011). https://doi.org/10.3923/itj.2011.1608.1613