Identification des scripts dans les manuscrits anciens

Résumé : La reconnaissance de scripts est la distinction d'une langue écrite par rapport à une autre dans un manuscrit ancien. Les manuscrits anciens sont des documents complexes de par la diversité de leurs contenus, par exemple la présence de lettrines, artefacts et différentes images. Mais aussi, par le niveau de détérioration qu'ils ont subit qu'elle soit au travers d'interventions humaines ou naturelles. Cette complexité rend la tâche de l'identification des scripts difficile, car l'algorithme doit avoir une bonne capacité de généralisation. Par exemple, l'algorithme doit être capable de reconnaître le script d'un ou plusieurs fragments provenant du même manuscrits, mais éparpillés dans plusieurs sources de données. Notre approche consiste à aborder le problème de l'identification des scripts par la détermination des caractéristiques en prenant le signal de l'image du document dans sa globalité, prenant donc en compte l'environnement où se trouve le script. Nous faisons l'hypothèse qu'un apprentissage effectif des caractéristiques utiles à la classification est possible dans un premier temps. Et, dans un second temps, l'extraction de caractéristiques fines est abordable par un empilement de réseaux de neurones de types auto-encodeurs (CAE pour Convolutional Auto-Encoders) afin de fournir une représentation alternative des données reçues en entrée. Ces représentations fines mélangent les propriétés du script aux propriétés de la page où ils se trouvent. Les représentations sont des plus générales dans les premières couches, par exemple le changement de contraste dans l'image, vers les plus particulières, par exemple le tracé d'un scribe (écrivain) par rapport à un autre. Les résultats obtenus avec les représentations CAE sont comparables à ceux fabriqués par des experts. Nous avons rapporté les résultats obtenus avec les caractéristiques apprises, et ce en les comparants aux caractéristiques extraites à l'aide d'une expertise humaine. Nous avons enfin relevé l'importance de l'initialisation des paramètres lors de l'apprentissage.

7.1 Introduction

7.1.1 Position du problème

L'évolution technologique de l'acquisition d'images, par scanner ou appareil photo de type réflexe, a permis de réduire les coûts liés à la numérisation de nombreux documents anciens, dans les bibliothèques, archives municipales, musées, etc. Ceci a résulté en une rapide expansion du nombre de travaux sur l'analyse d'images de documents ou DIA, *Document Image Analysis*. Le DIA est un domaine qui se trouve au croisement de l'analyse d'images, la reconnaissance de formes et les

(*a*) (*b*)

(*c*) (*d*)

FIGURE 7.1 – Exemple des quatre scripts correspondant aux quatre classes identifiées (a) Éthiopien (b) Syriaque (c) Grec (d) Latin

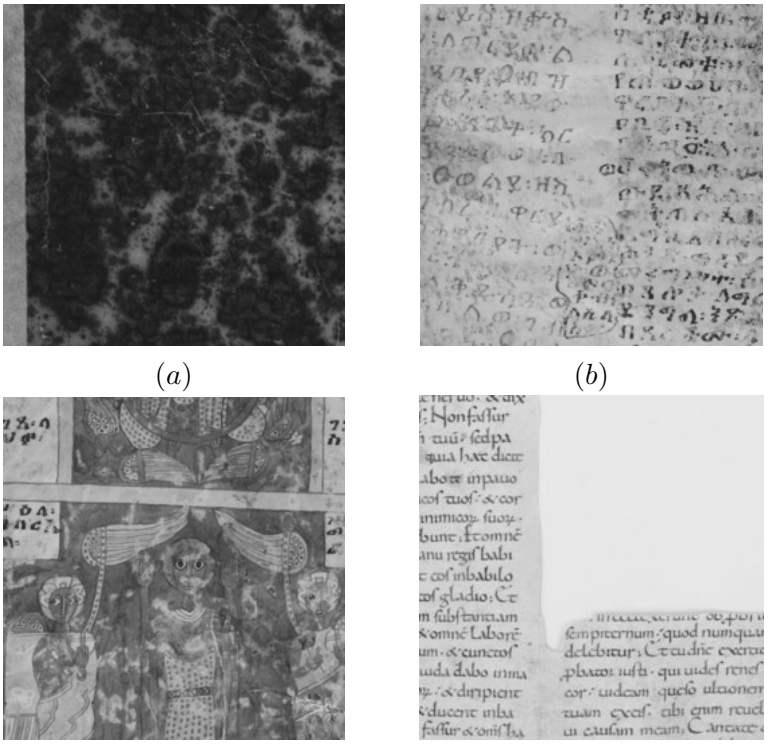
sciences humaines, avec par exemple l'implication de la philologie ou encore la paléographie.

Dans ce chapitre, nous nous intéressons à un cas particulier de DIA qui est l'identification des *scripts*[1] Dans les pages d'un manuscrit ancien. Le but d'exécuter une telle tâche est de proposer une méthodologie qui permettra de rechercher un document en émettant une requête sur son *script*, ou encore de transcrire son contenu avec un algorithme adapté de reconnaissance de caractères OCR *Optical Character Recognition* en anglais.

La figure 7.1 montre quatre scripts issus de différents manuscrits anciens qui seront étudiés dans ce chapitre (voir la table 7.1). Nous appelons donc script, un texte écrit à la main, à ne pas confondre avec une langue qui est la lecture du texte écrit ou le parlé dans une discussion.

Les manuscrits anciens sont des documents complexes. D'abord, leur mise en forme est complexe avec une variété d'artefacts comme des lettrines, des dessins, et des contenus textuels de différentes formes (e.g. figure 7.2 a et c). Ensuite, leur qualité est souvent détériorée par le temps, ou par des interventions humaines (e.g taches, coupures) comme cela est illustré par exemple dans les figures 7.2 (b) et 7.2 (d)). Puis, vu qu'il n'existe pas encore de normes communes à l'ensemble des

1. À ne pas confondre avec le terme "script" d'un langage informatique.



(a) (b) (c) (d)

FIGURE 7.2 – Exemple de différentes formes de dégradation, artefacts, intervention humaine que l'on peut trouver dans les manuscrits étudiés. (a) Une page de garde noire. (b) Taches sur la page et dilution de l'encre. (c) Artefacts. (d) Coupures et interventions humaines.

bibliothèques (e.g. Cambridge, Stanford, Bibliothèque National de France, etc.), la numérisation des manuscrits qui sont complets est souvent non homogène, c'est-à-dire que parfois le code couleur lors de la numérisation n'est pas le même, ou que la résolution n'est pas la même. Enfin, nous pouvons également avoir affaire à plusieurs fragments d'un seul manuscrit qui sont distribués dans plusieurs bibliothèques à travers le monde.

En pratique, les scripts sont qualifiés à l'aide d'un ensemble de caractéristiques (e.g. largeur du trait, contraste, forme des lettres). Ces caractéristiques sont accessibles au travers du traitement de l'image représentée par une matrice de pixels. Le traitement aux termes des pixels s'effectue par l'application d'un filtre sur un ou plusieurs niveaux qui correspondent à différentes régions d'intérêts (ROI pour *Region Of Interest* en anglais). Ces ROI peuvent représenter une ligne, un mot, ou encore un caractère. Dans ce cas, les documents illustrés dans la figure 7.2 sont le plus souvent considérés comme du bruit et sont supprimés lors de l'extraction des caractéristiques, et aussi de la discrimination des scripts.

En raison de la nature complexe des documents anciens, nous pensons que les méthodes qui sont entraînées sur un seul manuscrit avec des caractéristiques ex-

traites au niveau des ROI cités perdent en information et ne sont pas directement applicables à d'autres manuscrits anciens. Dans cette partie, nous proposons de travailler au niveau de la page sans aucun nettoyage ni pré traitement préalable sur les pixels du manuscrit.

7.1.2 Spécification de notre jeu de données

Dataset	#Pages	Scripts	Code manuscrit
BNF_eth_226	222	Ethiopic	Eth 226
BNF_eth_248	188	Ethiopic	Eth 248
BNF_eth_31	169	**Ethiopic**	Eth 31
BNF_eth_32	415	**Ethiopic**	Eth 32
BNF_syr_438	217	**Syriaque**	Syr 438
N41_BNF_grec_2465	476	**Greek**	Gr 2465
N97jpg	93	Greek	Suppl Gr 687
N43jpg	529	Greek	Gr 2795
N65jpg	377	**Greek**	N/A
N71jpg	417	Greek	Gr 164
N73jpg	824	Greek	N/A
N84	531	**Latin**	Latin 2246
N89jpg	338	**Latin**	Latin 9452

TABLE 7.1 – Description du jeu de données utilisé. Les scripts qui sont en gras sont ceux qui ont été utilisés lors de la phase d'entraînement pour l'identification des scripts.

Pour mener notre étude sur l'identification des scripts, nous avons choisi d'utiliser un jeu de données composé de 13 manuscrits avec un total de 4796 pages divisées en quatre principaux scripts : Éthiopien, Syriaque, Grec et Latin. Des exemples représentés par des captures sont illustrés dans la figure 7.1. Les manuscrits ont été téléchargés sur le site de la Bibliothèque Nationale de France (BNF). [2]. Les manuscrits écrits en éthiopien sont issus des collections de Mondon-Vidailhet, Griaule et Marcel Cohen. Ceux en Syrique sont issus des collections de l'Ancien Testament dans la version peshitta. Ceux en Grec sont issus de la collection ayant appartenu au chancelier Séguier. Ceux en latin sont issus de Gregorius Magnus, Homiliae in Ezechielem et d'un lectionnaire romain composé vers 670-680.

Les images des documents téléchargés ont été convertis du canal RGB (i.e. Red Green Blue) en des images niveaux de gris, c'est-à-dire vers un canal à une seule valeur. Téléchargées en `jpeg`, les images ont été converties dans le format `png` d'une taille moyenne de 300 `ko`. La taille des pages est de 1024×1400 pixels. Une description du jeu de données est décrite dans la Table 7.1.

Pour chaque script (i.e. Éthiopien, Syriaque, Grec et Latin) nous avons choisi au hasard deux manuscrits, mis en gras dans la table 7.1. Notre *vérité terrain* [3] (Ground Truth en anglais) est composée d'un total de 2523 pages étiquetées chacune avec le

2. http://www.bnf.fr/fr/collections_et_services/catalogues.html

3. Une *vérité terrain* est un ensemble d'images étiquetées par les différentes classes d'objets que nous souhaitons reconnaître en utilisant un algorithme d'apprentissage. L'étiquetage peut être fait par des experts ou automatiquement par une analyse des images.

script qui lui correspond. La *vérité terrain* est divisée en un jeu d'entraînement, en un jeu de validation pour l'optimisation des paramètres des différents algorithmes, et en un jeu de test pour valider les résultats obtenus par les algorithmes présentés. En se basant sur les bonnes pratiques tirées de l'état de l'art [Krig 2014], les proportions des trois jeux de données sont les suivantes : 60% pour l'entraînement, 25% pour la validation et 15% pour le test. La *vérité terrain* construite contient des images découpées dans des positions aléatoires de l'image, avec une dimension de 256×256. Nous n'avons pas effectué de traitement ni de déformation sur les données, ce qui veut dire que nous avons également gardé le type d'images données en exemple dans la figure 7.2 (c'est-à-dire des images dégradées ou contenant divers artefacts).

7.1.3 Notre approche pour la classification des scripts

Nous avons considéré deux principales méthodes pour l'extraction de caractéristiques. Nous nous référons à la première méthode comme une extraction faite main des caractéristiques, en anglais *handcrafted features method*. Nous nous référons à la deuxième méthode comme une extraction des caractéristiques apprises, en anglais *learned features method*.

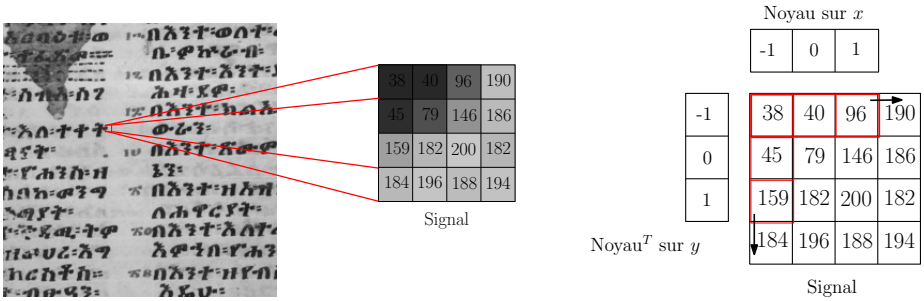
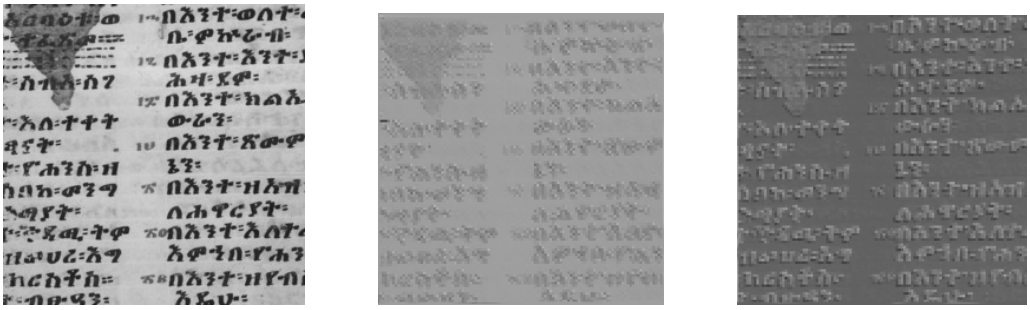


FIGURE 7.3 – (a) Extraction du signal au niveau du pixel de chaque image. (b) Application successive du masque $[-1, 0, 1]$ sur l'axe des x et $[-1, 0, 1]^T$ sur l'axe des y.

Comme dans tous les problèmes de traitement d'images, la page du manuscrit numérisé est représentée par une matrice de pixels. Nous nous référons à cette matrice comme étant un signal. L'extraction des caractéristiques à partir d'une image passe par la transformation du signal de cette dernière, usuellement représenté par une matrice de pixel, par exemple celle de la figure 7.3. La transformation s'effectue par une application d'une fonction de pondération appelée aussi masque. En pratique ceci revient à appliquer par translation sur la matrice de l'image un vecteur. Par exemple, le vecteur $[-1, 0, 1]$ traduit la différence entre la valeur du pixel d'avant et la valeur du pixel suivant. Comme c'est montré dans la figure 7.3. Un masque est appliqué sur une surface de pixels qui va être utilisée pour un certain traitement.

L'application de $[-1, 0, 1]$ sur l'axe x et de $[-1, 0, 1]^T$ sur l'axe y de l'image nous permet de calculer la dérivée du signal de l'image d'origine et de capturer

les changements au niveau des pixels. De grands changements nous permettront de capturer des traits ou de calculer les gradients dans une image. Notons que le processus que nous venons de décrire est commun à plusieurs techniques d'analyse des images nous y référons comme l'action d'appliquer un masque. Notamment lors de l'utilisation des réseaux de neurones artificiels, nous parlerons de filtre convolutif ou encore appliquer une convolution sur le signal de l'image. [4].



(a) (b) (c)

FIGURE 7.4 – Exemple de l'application du masque $[-1, 0, 1]$ sur l'axe x de l'image (a), obtention des caractéristiques représentées dans l'image (b). Application du masque $[-1, 0, 1]^T$ et obtention des caractéristiques représentées dans l'image (c).

Dans la figure 7.4, nous avons appliqué le masque $[-1, 0, 1]$ sur l'axe x de l'image (a), nous avons obtenu l'image (b). Ensuite, nous avons appliqué sa transposée sur l'axe y de l'image (a) et nous avons obtenu l'image (c). Ces caractéristiques obtenues sont qualifiées de faites main, car le masque $[-1, 0, 1]$ est déjà prédéfini. Les caractéristiques sont donc rigoureusement choisies par un expert pour y être ensuite remises en entrée dans un algorithme de discrimination ou classification. Par exemple, dans le domaine d'analyse des documents à partir d'images, nous pouvons citer les caractéristiques de Gabor [Hamamoto 1998], la technique du motif binaire local [Mu 2008], l'histogramme de gradients orientés [Dalal 2005], la transformation de caractéristiques visuelles invariante à l'échelle plus connue sous l'acronyme SIFT [Lowe 2004], etc.

L'approche que nous proposons d'étudier consiste à discriminer les scripts avec des caractéristiques apprises au terme d'une suite de convolutions successives appliquées sur les pixels au niveau de la **page** qui constituera notre ROI. Une convolution est une fonction qui est appliquée au signal, elle joue le rôle de filtre qui permet de distinguer certaines caractéristiques de ce signal par rapport à d'autres. Nous utiliserons une suite de convolutions, c'est-à-dire des convolutions de convolutions. Cette suite est structurée dans un ensemble de couches, l'ensemble de la structure est appelé architecture profonde par référence à l'approche communément appelée *Deep Learning* (voir la section 7.2 pour plus de détails).

L'objectif de l'utilisation de ces architectures profondes consiste à extraire des

4. En pratique une convolution est décrite par une matrice que l'on applique sur le signal (ou la matrice de pixels de l'image) comme c'est montré dans la figure 7.3

caractéristiques invisibles à l'oeil humain, mais interprétable et utiles pour les algorithmes de classification. Ces caractéristiques peuvent correspondre à des informations diverses telles que la courbure des traits, la forme des mots. Mais aussi d'autres informations qui sont considérées, par les autres méthodes, comme un bruit, par exemple, les informations qui concernent l'arrière-plan, les images, et d'autres artefacts présents dans les manuscrits. Ces bruits sont souvent supprimés dans des procédures de prétraitement. Nous pensons qu'un environnement non contrôlé avec un minimum d'intervention sur le jeu de données permettrait de réduire le biais et d'avoir une meilleure généralisation sur d'autres jeux de données [Torralba 2011].

Cette approche ne requiert pas d'expertise préalable sur les images et permet d'obtenir une représentation différente, mais satisfaisante comparée à l'extraction faite main. Comme pour l'apprentissage automatique en général, l'apprentissage des caractéristiques peut être divisé en deux sous catégories : un apprentissage non supervisé des caractéristiques et un apprentissage supervisé. Nous utiliserons respectivement les deux types d'apprentissages dans la section 7.4 et dans la section 7.5.

L'apprentissage non supervisé des caractéristiques ne requiert pas un étiquetage préalable des données d'entraînement. Nous pouvons apprendre et extraire des caractéristiques d'une image en utilisant *k-means* [Coates 2011], l'analyse en composantes principales (PCA) [Dorko 2003], les auto-encodeurs [Chen 2015, Wei 2015], les machines de Boltzman [Sahu 2015], etc. En revanche, l'apprentissage supervisé des caractéristiques requiert l'utilisation de la *vérité terrain* où nous disposons d'un étiquetage rigoureux des données.

À travers notre étude empirique de différentes architectures profondes, présentées tout au long de cette partie, nous voulons étudier l'hypothèse qui stipule qu'une architecture profonde est capable de se construire une représentation des données à partir d'une quantité limitée d'exemples. Et qui de plus n'a pas subi de prétraitement.

En travaillant avec les architectures profondes, les problèmes auxquels nous nous confrontons sont : la perte du signal d'origine au cours des convolutions successives avec une mauvaise optimisation des algorithmes, la qualité des données disponibles, et le non-contrôle de ce qui peut être appris. Par exemple, l'algorithme peut se faire une représentation des données à travers les variations des contrastes dans l'image ou la largeur des traits dans le texte, etc.

Dans un autre registre, le défi avec les architectures profondes comme d'autres algorithmes d'apprentissage automatique est la réduction du coût relatif aux erreurs commises lors de l'apprentissage (e.g. une instance mal classifiée). En particulier, ceci revient à optimiser une fonction objectif non convexe dans un espace de paramètres à plusieurs solutions. La différence entre une fonction convexe et non convexe est illustrée en une dimension dans la figure 7.5 (a) et 7.5 (b) respectivement. Dans la première, toute ligne reliant deux points sur (a) sera toujours au-dessus du minimum global ou de la solution optimale, tandis que pour la deuxième, toute ligne reliant deux points sur (b) peut être en dessous du minimum local.

Tandis que la recherche du minimum global d'une fonction convexe est un pro-

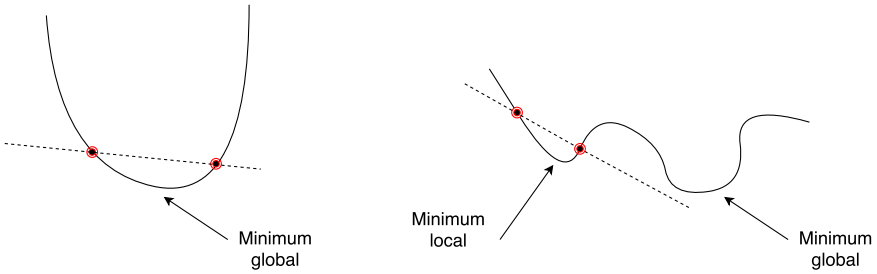


FIGURE 7.5 – (a) Illustration d'une fonction convexe avec une seule solution représentée par un minimum global. (b) Illustration d'une fonction non convexe avec plusieurs solutions, dont un minimum global.

blème résolu théoriquement, la recherche d'un minimum global d'une fonction non convexe est sujette à l'application d'une heuristique qui parcourt l'espace sans être sûre de trouver la solution optimale. Car, elle peut être prise au piège dans un minimum local et ne plus jamais en sortir. L'approche d'un tel problème consiste à explorer, empiriquement, l'espace des solutions en entreprenant plusieurs expérimentations, expliquées au travers des trois sections centrales de ce chapitre.

7.2 Méthode de travail

Les réseaux de neurones artificiels (ANN pour *Artificial Neural Network*) ont vu leur popularité augmenter au cours des dernières années. Tandis que l'implémentation des ANN a connu un succès modéré à leurs débuts dans les années 80, l'avancée technologique et l'arrivée des processeurs sur les cartes graphiques ont redonné à ses techniques une place importante. Sur certaines tâches telles que la reconnaissance d'objets, les ANN ont gagné une position difficile à détrôner par les autres algorithmes d'apprentissage telles que les séparateurs à vastes marges (SVM) ou encore les forêts aléatoires. Comme le montre l'étude de [Schmidhuber 2015], les ANN ont subi de nombreuses modifications et évolution au fils des années afin qu'ils soient adaptés à certaines tâches plus qu'à d'autres.

Nous distinguons deux grands groupes de ANN, les réseaux de neurones à convolutions (CNN pour *Convolutional Neural Network*) qui ont fait leurs preuves essentiellement en vision et en reconnaissance d'objet. Et les réseaux de neurones récurrents (RNN pour *Reccurent Neural Network*), qui sont utilisés pour apprendre une séquence de données par exemple la suite de caractères dans un texte numérique. Les CNN appliquent un filtre (décrit par une fonction) sur les données pour faire émerger des caractéristiques utiles à la classification. Les RNN donnent en sortie une combinaison des transformations linéaire appliquée sur les données en entrée, l'apprentissage de la séquence se fait par l'intégration d'un cycle dans la structure du réseau.

Nous nous intéressons dans cette partie à l'extraction des caractéristiques par

l'application d'un filtre appris. Nous utiliserons pour cela deux types de CNN, le premier appelé auto-encodeur entreprend un apprentissage non supervisé du filtre. Le deuxième est un CNN classique, il entreprend un apprentissage supervisé du filtre. Les auto-encodeurs sont des réseaux de neurones qui sont entraînés pour fournir une représentation alternative des données x qu'ils reçoivent en entrée, ceci est effectué par l'application d'une suite de convolutions, ensuite une suite de dé-convolutions.

Nous nous référons à la structure de dispositions des couches de convolutions et les couches de classifications comme une architecture. Nous appellerons architecture profonde, une architecture avec plusieurs couches de convolutions ou couches d'auto-encodeurs. L'apport méthodologique dans cette partie consiste en l'exploration empirique des architectures que nous avons introduites pour la tâche de l'identification des scripts dans les manuscrits anciens. Dans ce qui suit, nous commencerons par un rappel sur les ANN et introduirons par la suite les outils méthodologiques utilisés dans cette partie.

7.2.1 Rappels sur les réseaux de neurones artificiels

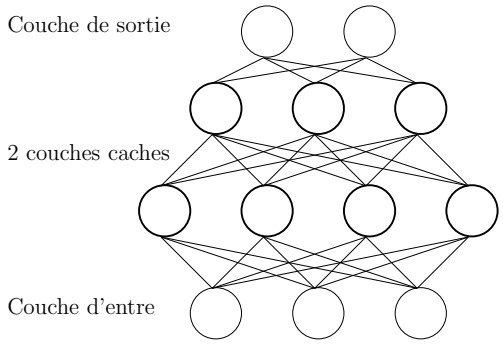


FIGURE 7.6 – Exemple d'un réseau de neurones artificiel avec une couche d'entrée, deux couches cachées et une couche de sortie. Nous avons illustré des liens bi directionnels entre les noeuds pour illustrer le sens de la convolution et le sens de l'algorithme de rétropropagation.

Le réseau de neurones artificiels (ANN), ou perceptron multicouche (ANN), peut être défini comme un ensemble d'unités de traitement simples (neurones) qui sont reliées entre elles par des connexions pondérées. Dans la figure 7.6, le réseau de neurones artificiels est constitué :

— d'une couche d'entrée de dimension $d = 3$. La valeur d dépend du vecteur d'observation reçu par la couche d'entrée.
— Ensuite de deux couches composées d'unités cachées qui appliquent une fonction non linéaire sur les transformations linéaires qu'elles reçoivent de la couche d'entrée.
— Et enfin, la couche de sortie, de dimension m.

Dans la figure 7.6, $m = 2$. Dans ce cas, ce réseau, cité en exemple, est utilisé pour une tâche de classification binaire. Pour d'autres valeurs de m, le réseau sera

utilisé pour une classification multi classes. Dans la couche de sortie, le neurone avec la valeur la plus élevée donnera la classe.

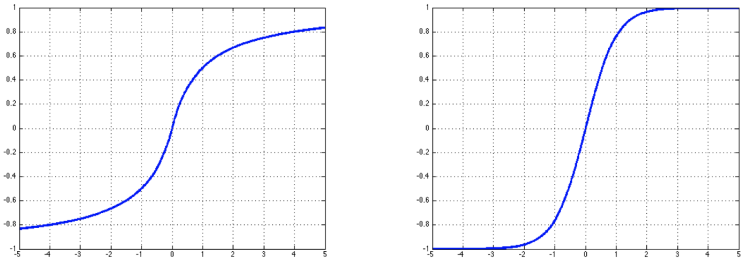
Afin de simplifier l'explication, prenons le cas d'un réseau de neurones à une seule couche cachée comportant d_h unités cachées [5]. La sortie du réseau est obtenue par le calcul de la fonction $f : \mathbb{R}^d \rightarrow \mathbb{R}^m$, où d est la taille du vecteur d'entrée x, et m la taille du vecteur de sortie $f(x)$, telle que :

$$f(x) = g(b_2 + W_2(h(b_1 + W_1 x))) \tag{7.1}$$

$b_1 \in \mathbb{R}^{d_h}$ et $b_2 \in \mathbb{R}^m$ sont les vecteurs de biais. $W_1 \in \mathbb{R}^{d_h \times d}$ représente la matrice des poids connectant le vecteur d'entrée à la couche cachée, $W_2 \in \mathbb{R}^{m \times d_h}$ est la matrice des poids connectant la couche cachée au vecteur de sortie. La fonction 7.1, calcul la sortie du neurone artificiel vers l'environnement extérieur par l'application de la fonction d'activation h sur la matrice des poids en entrée $W1$ et l'application de la fonction d'activation g sur la sortie de h qui est passée aux neurones de la couche cachée. L'interprétation de la sortie de cette fonction 7.1 dépend du problème de classification considéré. Par exemple dans le cas binaire 1 signifierait que l'entré appartient à une classe A et 0 signifierait que l'entrée appartienne à la classe B. Les fonctions d'activation g et h sont définies comme suit : typiquement pour h l'on choisit la fonction sigmoïde ou la fonction tangente hyperbolique :

$$sigmoide(a) = \frac{1}{1 + e^{-a}} : [-\infty, +\infty] \rightarrow [0, 1] \tag{7.2}$$

$$tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}} : [-\infty, +\infty] \rightarrow [-1, 1] \tag{7.3}$$



(a) (b)

FIGURE 7.7 – Illustration des réponses non linéaires des fonctions d'activations *softsign* (a) et *than*(b)

Pour la discrimination des scripts avec des réseaux de neurones, nous utiliserons la fonction *tanh* dans nos travaux, car elle permet un apprentissage plus rapide en raison de sa saturation en 1. Dans le cas de l'apprentissage des caractéristiques, comme nous le verrons plus tard, nous utiliserons la fonction *softsign*. Comme nous pouvons observer dans la figure 7.7, la dérivée de la fonction *softsign* illustrée en (a)

5. d_h dimension des couches cachées, h est utilisée pour *hidden* : cachée

converge polynomialement vers 0, ce qui est plus intéressant pour la phase d'apprentissage qu'une dérivée de fonction qui converge de façon exponentielle et sature en 1 rapidement comme la fonction *tanh* illustrée en (b). La première raison de ce choix revient à la nature de nos données qui sont bruitées et en quantité limitée qui nous a poussés à penser qu'un apprentissage des caractéristiques qui ne converge pas rapidement pourrait apporter plus d'informations. La seconde raison de ce choix porte sur la rapidité de cette fonction d'activation lors de l'apprentissage des caractéristiques. La fonction *softsign* est définie comme suit :

$$softsign(x) = \frac{x}{|x|+1} : [-\infty, +\infty] \rightarrow [-1, 1] \tag{7.4}$$

On peut réécrire le vecteur de sortie $f(x)$ dans la fonction 3.12 comme suit :

$$f(x) = g(c + V(h(x))) \tag{7.5}$$

Dans le cas d'une classification multi classes, le choix de g se porte le plus souvent sur la fonction $softmax$ décrite comme suit :

$$softmax(a)_i = \frac{e^{a_i}}{\sum_j e^{a_j}} \tag{7.6}$$

Algorithme 13 Apprentissage des paramètres d'un réseau de neurones artificiels avec la descente de gradient

```
1: Initialiser aléatoirement les poids du réseau
2: tant que condition d'arrêt n'est pas atteinte faire
3:     pour tout exemple dans l'ensemble d'entraînement faire
4:         Calculer le gradient de la fonction de perte L
5:         Appliquer la technique de rétropropagation du gradient
6:         Mettre à jour les poids W du réseau
```

Pour entraîner un réseau ANN, nous devons apprendre tous les paramètres $\theta = \{W, V, b, c\}$. Pour cela, nous avons utilisé dans nos travaux l'algorithme de descente de gradient (voir pseudo-code dans Algorithme 13). L'algorithme est divisé en deux étapes : une propagation en avant pour calculer le vecteur de sortie $f(x)$ et une propagation en arrière où la dérivée partielle de la fonction de perte est calculée par rapport aux paramètres du réseau. Ceci est décrit dans la fonction suivante :

$$W \leftarrow W - \eta \frac{\partial L}{\partial W} \tag{7.7}$$

Où η est un pas d'apprentissage, et $\frac{\partial L}{\partial W}$ le gradient de la perte L par rapport aux paramètres W.

Un réseau ANNs ayant suffisamment d'unités cachées est en mesure d'apprendre à représenter n'importe quelle fonction continue. Dans ce cas, il est considéré comme un approximateur universel [Hornik 1989]. Cependant, le nombre de paramètres à apprendre augmente de façon exponentielle avec le nombre de couches cachées [Hastad 1991] ce qui nous empêche d'avoir un temps d'apprentissage "souhaitable"

sur des architectures multicouches [Bengio 2007c]. De nos jours, grâce aux nouvelles architectures, fonctions d'activations et méthodes d'entraînement combinées à l'augmentation de la puissance des ordinateurs, l'utilisation des architectures dites profondes est rendue possible. Nous nous concentrerons dans cette partie sur l'utilisation des réseaux de neurones à convolution pour l'extraction des caractéristiques apprises.

7.2.2 Les réseaux de neurones à convolutions (CNN)

Les réseaux de neurones à convolution (CNN pour *Convolutional Neural Networks*) sont inspirés des travaux de Hubel sur le cortex visuel des mammifères et l'arrangement des cellules responsables de l'interprétation du champ visuel [Hubel 1968]. On distingue essentiellement deux types de cellules. Les cellules dites simples, ils détectent les caractéristiques liées à la forme et les cellules dites complexes qui sont sensibles à la forme entière.

Les CNNs ont été popularisées avec les travaux de [Lecun 1998a] et son implémentation appelée LeNet5 (5 pour les 5 régions de l'aire visuelle), pour la reconnaissance de caractères. Cependant, les CNNs trouvent leur origine dans les travaux de [Fukushima 1980] sur la simulation du fonctionnement des cellules dans cortex visuel des mammifères.

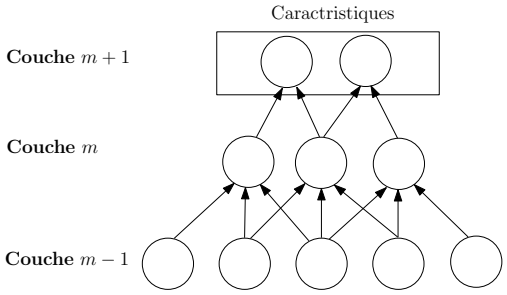


FIGURE 7.8 – Caractéristiques/Poids partagées

Dans les CNNs, chaque neurone est connecté à un sous-ensemble de la couche précédente. Dans la figure 7.8, la couche $m-1$ comporte 5 neurones, les trois neurones de la couche m sont connectés à trois neurones voisins de la couche $m-1$. On peut considérer chaque neurone comme une unité de détection d'une caractéristique locale. La carte des caractéristiques (*features map*)[6] est construite par convolution c'est-à-dire par application d'un filtre linéaire dont le principe consiste à utiliser une fenêtre glissante pour parcourir les données. Comme c'est décrit dans l'équation 7.2.2.1, chaque connexion entre les neurones est pondérée par une matrice des poids W et un vecteur de biais b qui contrôle le potentiel d'activation des neurones. La sortie est obtenue par l'application d'une transformation non linéaire g. Le passage

6. Ensemble de neurones qui sont considérés comme des unités de détection des caractéristiques locales.

d'une couche à une autre a pour effet de réduire le signal global en des caractéristiques locales.

$$h_{ij}^{k} = g((W^{k} * x)_{ij} + b_{k}) \tag{7.8}$$

L'architecture du réseau construit avec des convolutions impose le partage des poids pour toute carte k de caractéristiques. Car, les neurones calcul la même fonction avec des entrées différentes provenant de différentes parties du système, dont il faut qu'ils aient le même poids. Ce principe de partage de poids réduit considérablement la complexité du modèle et facilite également la généralisation, puisque l'apprentissage se fait avec les mêmes paramètres pour extraire les mêmes caractéristiques des différentes parties du signal global.

7.2.2.1 Couche de *Max-Pooling* ou max-agrégation

Enfin, un autre concept important des CNNs est le sous-échantillonnage appliqué sur la fenêtre glissante de la convolution. Appelé max-pooling ou max-agrégation, il a pour effet de ne prendre que la valeur maximale de la sortie de la fonction d'activation g appliquée sur différentes régions, qui ne se chevauchent pas, du signal global. De la même manière que l'équation , un vecteur de biais est rajouté aux résultats en sortie. Le max-pooling permet de réduire la sensibilité aux translations de la fenêtre et aux faibles rotations. Il de ce fait permet d'éviter aussi des calculs superflus.

7.2.3 Les auto-encodeurs (AE)

Les auto-encodeurs [Ranzato 2007, Bengio 2007a] sont des réseaux de neurones qui sont entraînés pour fournir une représentation alternative des données x qu'ils reçoivent en entrée. Un auto-encodeur qui aura appris la bonne représentation des données x est capable d'encoder puis de décoder de manière à retrouver approximativement les mêmes données x en entrée. L'intérêt d'utiliser les auto-encodeurs n'est pas dans le fait de retrouver x, mais dans la transformation qu'aura appris l'auto-encodeur pour arriver à ce résultat. C'est cette représentation apprise que nous avons fournie comme un pré entraînement au réseau, afin de remplacer l'initialisation aléatoire des connexions neuronales ou d'extraire de bonnes caractéristiques apprises.

Dans la figure 7.9, nous avons illustré un exemple d'auto-encodeur avec une seule couche. Le vecteur des caractéristiques apprises, à droite de la figure, est issu de la représentation que l'auto-encodeur a construite avec l'exécution d'une descente de gradient (voir l'algorithme 13) et ensuite l'exécution de l'algorithme de rétropropagation pour ajuster l'initialisation des connexions neuronales.

Le principe de fonctionnement de l'auto-encodeur consiste en l'application de deux tâches : l'encodage et le décodage. L'encodage est une transformation déterministe, que l'on note z, du vecteur en entrée, $x \in [0, 1]^d$. Notons que dans nos travaux, nous utiliserons un vecteur d'entrée $x \in [-1, 1]^d$.

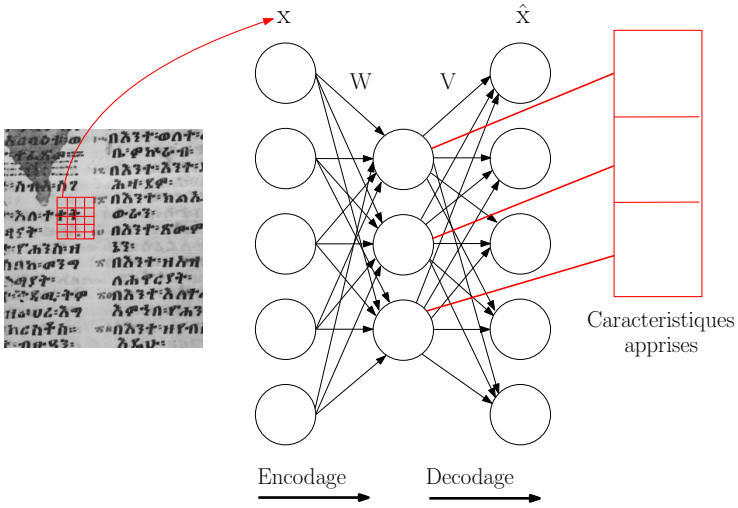


FIGURE 7.9 – Un auto-encodeur avec une seule couche cachée

$$z = f_{enc}(Wx + b) \tag{7.9}$$

Où W est une matrice $\in R^{d_h \times d}$ et $b \in R^{d_h}$ est un vecteur de biais. Le décodage est la reconstruction de la transformation h, que l'on note r.

$$\hat{x} = f_{dec}(Vz + c) \tag{7.10}$$

Où V est la transposée de W, s'il s'agit de matrices liées, et c un biais de dimension d. Comme pour un réseau de neurones, les paramètres sont $\theta = \{W, V, b, c\}$ et les poids sont $V = W^T$ s'il s'agit de matrices liées. Ces paramètres sont optimisés avec la méthode de descente de gradient qui a pour but de trouver un optimum global. Pour une entrée x binaire ou quasi binaire, il est préférable d'optimiser l'erreur de reconstruction par le calcul d'une entropie croisée. L'entropie croisée mesure la distance entre la sortie donnée par le réseau et la sortie attendue. L'entropie croisée est définie dans l'équation 7.11, quant à l'équation de la perte résultante, elle est définie dans l'équation 7.12.

$$L(x, \hat{x}) = -\sum_{i=1}^{d} [x_i \log(\hat{x}_i) + (1 - x_i) \log(1 - \hat{x}_i)] \tag{7.11}$$

$$L(x, \hat{x}) = \sum_{i=1}^{d} [(x_i - \hat{x}_i)^2] \tag{7.12}$$

7.2.4 Les auto-encodeurs convolutif empilés (SCAE)

Les auto-encodeurs peuvent être empilés, où la sortie d'un auto-encodeur est utilisée comme entrée pour l'auto-encodeur suivant. Cette utilisation séquentielle augmente le taux de compression des données en entrées et réduit le nombre de

caractéristiques dans la couche $l+1$ par rapport à la couche l. Cette structure peut être associée à un réseau de neurones à convolution pour avoir un vecteur de caractéristiques en plusieurs dimensions, ceci en appliquant le même auto-encodeur sur plusieurs grilles de pixels.

Dans cette thèse, et pour apprendre les caractéristiques, nous utiliserons un empilement d'auto-encodeurs avec l'application d'une convolution à chaque couche pour la transformation du signal de l'image d'origine. Les auto-encodeurs convolutifs empilés ou SCAE (Stacked convolutional autoencoders) sont un réseau de neurones avec plusieurs couches d'auto-encodeurs. Cet empilement d'auto-encodeurs forme une architecture profonde où le résultat de la couche l_n est repris par la couche l_{n+1}.

L'apprentissage non supervisé se fait sur cette structure en appliquant un algorithme glouton, une heuristique qui va chercher le minimum local pour chaque couche [Bengio 2007b]. Les poids du réseau sont affinés en utilisant la méthode de descente de gradient, le but est d'arriver à un minimum global pour minimiser le risque empirique. Le vecteur de caractéristique produit par la couche finale peut être donné en entrée à un modèle de classification analogue à un réseau de neurones à convolution afin d'initialiser ses poids.

En ce qui concerne la fonction d'activation des auto-encodeurs empilés, nous avons choisi de travailler avec la fonction *softsign*. L'auto-encodeur avec lequel nous avons travaillé encode une entrée x de dimension n en une sortie y de dimension m comme suit :

$$y_i = f\left(b_i^e + \sum_{j=1}^{n}\left(w_{ij}^e \cdot x_j\right)\right) \tag{7.13}$$

Où b_i^e est le biais et w^e les poids utilisés pour l'encodage.

Le décodage de la sortie y pour reconstruire l'entrée est exécuté d'une façon similaire :

$$\hat{x}_j = f\left(b_j^d + \sum_{i=1}^{m}\left(w_{ji}^d \cdot y_i\right)\right) \tag{7.14}$$

Où $\hat{x}$ est l'approximation du vecteur x encodé par y, b_j est le biais, et w^d sont les poids utilisés pour le décodage. Pour calculer la fonction 7.14, l'auto-encodeur a besoin d'apprendre un total de poids $m \times (n+1) + n \times (m+1)$, durant la phase d'entraînement. Pour optimiser le temps d'entraînement, il est plus judicieux d'utiliser des convolutions successives d'auto-encodeurs empilés avec des signaux de petite taille que d'entraîner, en une seule fois, un auto-encodeur qui couvre une grande portion de l'image ou un signal de grande taille. Car, rappelons que les auto-encodeurs donnent une représentation alternative à l'ensemble du signal. Tandis que, les piles d'auto-encodeurs fournissent des représentations alternatives de plusieurs ordres. C'est-à-dire de l'ordre des traits, contours, etc. Jusqu'à arriver à l'ordre de l'objet en entier.

Les convolutions sont créées comme suit : d'abord, un auto-encodeur couvrant une surface de $W_1 \times H_1$ pixels avec un nombre de sortie m_1 est

entraîné. Ensuite, nous le copions $W_2 \times H_2$ fois, et rajoutons les copies dans une grille avec un décalage de $O_{1x} \times O_{1y}$ pixels. La grille couvre alors, $((O_{1x} \cdot (W_2 - 1) + W_1) \times (O_{1y} \cdot (H_2 - 1) + H_1))$ pixels. La sortie des auto-encodeurs dans cette grille peut être considérée comme un tableau composé de $W_2 \times H_2 \times m_1$ valeurs. Ce tableau peut être par la suite donné en entrée à un auto-encodeur dans un second niveau dans la convolution.

Notons qu'il faut faire attention à la dimension des convolutions pour pouvoir empiler plusieurs niveaux d'auto-encodeurs. Enfin, tous les niveaux des auto-encodeurs empilés avec une convolution sont entraînés avec l'algorithme de descente de gradient (voir algorithme 13) et une rétropropagation sur les poids, le but étant de minimiser $(\hat{x} - x)^2$.

Les poids des connexions neuronales ont été initialisés avec une distribution uniforme telle que

$$W \in]-\frac{1}{\sqrt{n_{in}}}; \frac{1}{\sqrt{n_{in}}}[\tag{7.15}$$

Où n_{in} est le nombre de neurones en entrée. La valeur des neurones en sortie est dans l'intervalle $[-1, 1]$. Le réseau avec lequel nous travaillons n'est pas *Sparse*, ou en d'autres termes, il ne contient pas de zéros. À quantité égale d'informations encodées, ce type de réseau possède une phase d'entraînement plus rapide que les réseaux *Sparse*.

7.2.4.1 Les couches d'*Extremum Pooler* : concept additionnel

Nous appelons *Extremum Pooler* une opération de réduction sur un réseau dense (i.e. non *Sparse*) par la sélection exclusive des neurones avec la valeur maximale. Cette fonction est similaire au *Maxpooling* qui est exécutable sur un réseau *Sparse*. La fonction d'*Extremum Pooler* est définie par

$$a_j = \max_{n \times n}(a_i^{m \times m} J(m, m)) \tag{7.16}$$

où $J(m, m)$ est une fenêtre glissante sur le *patch* en sortie d'une couche d'un SCAE ou d'une convolution. J calcule la valeur maximale dans le voisinage défini par la taille de la fenêtre glissante sur le *patch* d'une taille $m \times m$ pixels.

7.2.5 Architectures

Dans le contexte de ce travail, nous appelons une architecture une structure d'interconnexion entre différentes couches de convolutions, de max-pooling, d'extremum-pooling ou d'auto-encodeurs. Une architecture comprend dans son entrée une image et en sa sortie un vecteur de caractéristiques ou un réseau de neurones pour la classification. Par exemple, la figure 7.15 est une suite d'auto-encodeurs empilés qui transforme l'image en entrée en un vecteur de caractéristiques en appliquant différents filtres. Chaque filtre réduit la taille du signal d'origine et possède plusieurs

sorties. Par exemple, l'application du premier filtre dans la figure 7.15 résulte en 8 images de taille 26×26.

Au vu de la complexité de nos données, nous explorons dans ce travail plusieurs architectures dans le but d'obtenir une bonne taille du filtre appris. Car, il faut qu'il soit assez large pour capturer les variations des différents traits des scripts et assez petit pour ne pas capturer trop de variations et perdre l'information essentielle à la classification.

7.3 Construction d'un référentiel empirique obtenu

Nous introduisons dans cette section notre référentiel empirique. Ce référentiel est élaboré avec trois algorithmes qui sont entraînés avec des caractéristiques extraites avec l'histogramme de gradients orientés (HOG en anglais, *Histogram of Oriented Gradients*). Ce référentiel a pour but de comparer les solutions à base d'architectures profondes que nous proposons.

7.3.1 Histogramme de gradients orientés

FIGURE 7.10 – Détection des piétons avec des histogrammes de gradients orientés

Dans le référentiel empirique, les algorithmes de classification sont entraînés avec des caractéristiques extraites avec un filtre fait main, en anglais *handcrafted features*. Nous avons choisi d'utiliser l'histogramme de gradients orientés (HOG) qui a été proposé par [Dalal 2005] pour détecter les piétons dans une image par exemple dans la figure 7.10. Les caractéristiques de type HOG ont également été utilisées par [Minetto 2013] avec succès pour la reconnaissance des textes dans les images et par [Yilmaz 2011] pour la reconnaissance du tracé d'une signature. Étant donné la nature complexe des images de manuscrits, notre choix s'est porté sur l'extraction des caractéristiques avec HOG. Car, il est plus résistant aux transformations géométriques et photométriques dans les images.

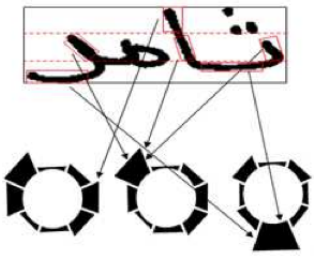


FIGURE 7.11 – Des histogrammes de gradients orientés divisés en 8 *bins*. Chacun des trois histogrammes correspond à la partie supérieure, celle du milieu et la partie inférieure de l'image du script arabe, donné ici en exemple. La flèche indique la contribution d'une partie spécifique du tracé à l'histogramme [Saidani 2015]

HOG caractérise une image par une distribution de gradients. Chaque gradient représente un changement de direction dans l'intensité du pixel ou la couleur dans une image. Les caractéristiques HOG sont représentées par un histogramme qui compte l'orientation du gradient des pixels d'une image donnée. Le principe est le suivant : d'abord les orientations du gradient de chaque pixel sont calculées, puis un histogramme des orientations est calculé sur une petite portion de l'image. Enfin, tous les histogrammes de toutes les portions sont rassemblés dans un vecteur. Ce vecteur est ce que l'on appelle vecteur de caractéristiques et il sera donné en entrée aux algorithmes de classification du référentiel empirique. Ce principe est décrit en détail dans ce qui suit.

7.3.1.1 Extraction des caractéristiques

En pratique, le vecteur des caractéristiques HOG est extrait comme suit : d'abord, l'image est divisée en des petites portions (e.g. 6×6 pixels), où l'on applique les filtres $D_x = [-1, 0, 1]$ et $D_y = [-1, 0, 1]^T$, respectivement sur l'axe x et y de l'image I. Cette opération de convolution, comme ça a été montré précédemment dans la figure 7.3, nous permet d'obtenir les dérivées $\delta_x = I \times D_x$ et $\delta_y = I \times D_y$, comme ça été également montré dans la figure 7.4 où δ_x correspond à la figure (a) 7.4 et δ_y correspond à la figure (b) 7.4.

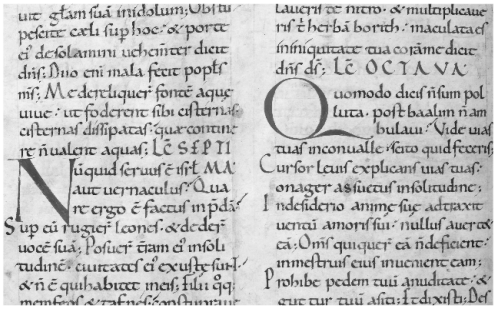
Puis, la magnitude et l'orientation du gradient sont calculées avec les équations 7.17 et 7.18.

$$Magnitude_{x,y} = \sqrt{(\delta y_{x,y})^2 + (\delta x_{x,y})^2} \tag{7.17}$$

$$Angle_{x,y} = atan(\delta y_{x,y}, \delta x_{x,y}) \tag{7.18}$$

Ensuite, un histogramme des orientations est calculé pour chaque portion en utilisant l'équation suivante :

$$H_i = \sum_{x,y \in I, Angle_{x,y} \in bin_i} Magnitude_{x,y}, i = 1, \ldots, n \tag{7.19}$$



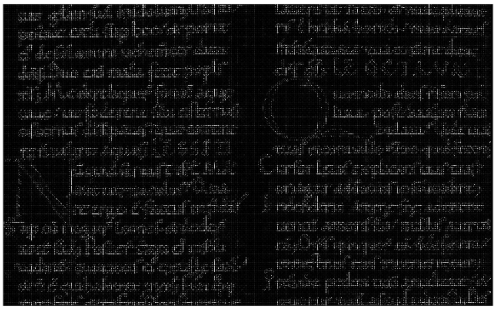


FIGURE 7.12 – Calcul de l'histogramme des gradients orientés sur une image d'une page d'un manuscrit écrit en latin.

Suivant les gradients calculés, les histogrammes sont uniformes de 0 à 180° (cas non signé) ou de 0 à 360° (cas signé). Dans nos expérimentations, l'intervalle $[0, 2\pi]$ (i.e. 0 à 360°) est divisé en ce que l'on appelle des *bins*, comme c'est montré dans la figure 7.11. Chaque *bin* couvre une orientation de $\frac{\pi}{4}$. Ce qui nous donne un n = 8 bins. Ce choix est expliqué par les expérimentations précédentes menées par [Saidani 2015] sur la reconnaissance du tapé et du tracé des mots dans une image.

Chaque pixel de la portion vote pour un *bin* de l'histogramme, en fonction de l'orientation du gradient au point (x,y). Le vote du pixel est pondéré par l'intensité du gradient en ce point.

Enfin, les gradients de tous les blocs sont normalisés dans un histogramme et régularisés avec la norme L_2 décrite par l'équation 7.20, où les histogrammes de chaque bloc sont désigné par v.

$$f = \frac{v}{\sqrt{||v||_2^2 + \varepsilon^2}} \tag{7.20}$$

Une fois normalisées, les valeurs de l'histogramme sont collectées dans un vecteur de caractéristiques et sont par la suite données en entrée aux algorithmes de classification présentée dans la sous-section suivante. La figure 7.12 illustre la sortie obtenue avec HOG après son application sur une page d'un manuscrit latin.

7.3.2 Classification

Pour construire notre référentiel de comparaison des différentes stratégies d'analyse des scripts, nous avons donné, en entrée, à trois différents algorithmes de classification les caractéristiques extraites avec HOG. Nous avons choisi le perceptron multicouche appelée aussi réseaux de neurones artificiels, les séparateurs à vaste marge et les forêts aléatoires.

7.3.2.1 Réseaux de neurones artificiels

Les réseaux de neurones artificiels sont constitués de plusieurs unités logiques appelées neurones, qui répondent que par 0 ou 1. Les neurones sont disposés dans trois différentes couches : la couche d'entrée, la couche cachée et la couche de sortie. La couche de sortie permet de diviser l'espace d'entrée de façon linéaire, les ou la couche cachée permet applique une division non linéaire sur l'espace. Pour plus de détails, veuillez vous référer à la section de rappel 7.2.1. Nos expérimentations ont été menées avec un nombre de couches cachées égales à $\frac{\#attributs+\#classes}{2}$. Par exemple si nous avons six attributs et les quatre classes correspondantes aux différents scripts alors le nombre de couches est égal à 5.

7.3.2.2 Forêts aléatoires

Les forêts aléatoires (RF pour Random Forest en anglais) introduites par [Breiman 2001] sont un ensemble d'arbres de décisions entraînés aléatoirement. Chaque arbre produit une prédiction et l'ensemble des prédictions est ensuite agrégé avec différentes techniques [Breiman 2001], afin de produire un résultat de prédiction final. Pour plus de détails, veuillez vous référer à la section 5.2.

7.3.2.3 Séparateurs à vaste marge

Les séparateurs à vastes marges (SVM) ont été introduits par [Vapnik 1998]. La méthode consiste à chercher un hyperplan qui sépare deux classes de données de manière à maximiser la marge entre les exemples les plus proches des deux différentes classes. On se réfère à ses exemples comme des vecteurs de support. Pour plus de détails, veuillez vous référer à la section de l'annexe A.2.3.

7.3.3 Résultats

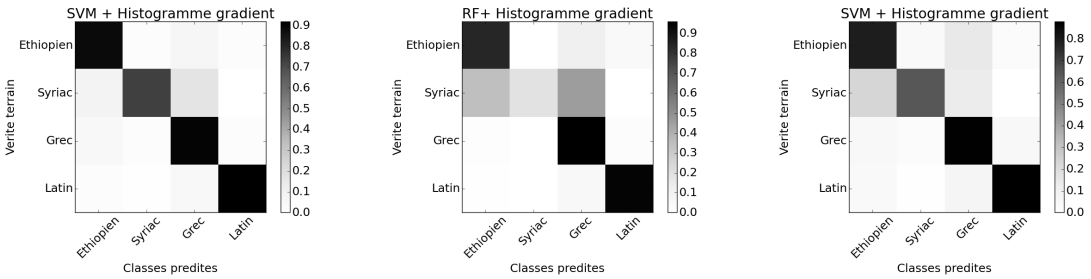
Afin de juger la fiabilité des trois algorithmes de classification (i.e. ANN, RF, SVM) entraînée avec les caractéristiques HOG, nous avons utilisé la validation croisée sur une base de 2523 images de pages de manuscrits. Nous avons fixé le nombre de portions (*folds*) à 10 pour chaque itération, c'est-à-dire que l'évaluation de l'erreur est calculée sur 10 sous-échantillons tirés de la base des images.

Les résultats sont rapportés dans la table 7.2 où nous pouvons constater que les réseaux de neurones artificiels ont obtenu le meilleur résultat avec un taux de justesse égal à 89.07% suivi des forêts aléatoires avec un taux de justesse égale à 86.15%. Cependant, selon la matrice de confusion illustrée dans la figure (b) 7.13, nous constatons que les forêts aléatoires n'arrivent pas à généraliser leur apprentissage pour discriminer le script syriaque.

L'algorithme SVM paraît plus résistant à la quantité de données fournies lors de l'apprentissage et arrive à reconnaître les quatre scripts malgré une performance moindre par rapport aux deux autres. Notons que la quantité de données pour ce

HOG		Precision moy.	Rappel moy.	F1 moy.	Justesse	RMSE
ANN	Ethiopien	0.86	0.88	0.87	**89.07%**	0.12
	Syriaque	0.78	0.73	0.75		
	Grec	0.87	0.90	0.88		
	Latin	**0.95**	**0.92**	**0.93**		
RF	Ethiopien	0.85	0.83	0.84	86.15%	0.15
	Syriaque	0.96	0.19	0.32		
	Grec	0.80	0.96	0.87		
	Latin	**0.94**	**0.94**	**0.94**		
SVM	Ethiopien	0.79	0.79	0.79	83.70%	0.18
	Syriaque	0.72	0.64	0.67		
	Grec	0.81	0.87	0.84		
	Latin	**0.92**	**0.88**	**0.90**		

TABLE 7.2 – Les résultats des algorithmes de discrimination entraînée avec les caractéristiques issues de l'histogramme de gradients orientés. Nous avons mené un ensemble de 10 tests sur les données jamais vues auparavant par les algorithmes de discrimination. ANN est l'abréviation pour le réseau de neurones artificiels, RF est l'abréviation des forêts aléatoires, et SVM est l'algorithme de machine à vecteurs de support.



(a) (b) (c)

FIGURE 7.13 – Matrice de confusion des algorithmes de classification avec les caractéristiques extraites avec l'histogramme de gradient orienté. La figure (a) illustre les résultats de discrimination du ANN. La figure (b) illustre les résultats de discrimination des forêts aléatoires. Enfin, la figure (c) illustre les résultats de discrimination exécutés par SVM. Les meilleurs résultats sont mis en gras.

script est réduite par rapport aux trois autres, ce qui explique notamment la faible performance des trois algorithmes pour la reconnaissance du Syriaque en particulier.

7.4 Contribution pour l'apprentissage non supervisé de scripts avec les auto-encodeurs

Dans cette section, nous étudions l'apprentissage non supervisé des caractéristiques pour la reconnaissance des scripts. L'intérêt pour l'apprentissage non supervisé des caractéristiques a été porté par [Hinton 2006a, Bengio 2007b, Huang 2007, Coates 2010, LeCun 2012]. Il est présenté comme une alternative logique des ca-

ractéristiques extraites par des filtres choisis ou pré fabriqué par des experts (i.e. *handcrafted features*), et comme une technique qui exploite la puissance des réseaux de neurones à plusieurs couches, par rapport à l'apprentissage supervisé. Défendu donc par Lecun, Bengio, Hinton et leurs étudiants, l'apprentissage non supervisé des caractéristiques a été récemment introduit dans l'analyse des images des documents.

Nous citons par exemple le travail de [Sahu 2015] sur la reconnaissance des caractères à partir d'une image d'un document avec un ROI au niveau des mots. Les auteurs ont utilisé un empilement de machines de Boltzman restrictives (RBM) pour obtenir des représentations compactes des images contenant du texte. Les RBMs sont des réseaux de neurones avec des unités stochastiques par comparaison aux unités déterministes comme la sigmoïde ou le softsign (voir le rappel sur les réseaux de neurones dans la sous-section 7.2.1). Les auteurs ont ensuite appliqué des algorithmes classiques des reconnaissances de caractères (OCR) sur les caractéristiques apprises par les RBMs et ont démontré de meilleures performances comparées aux caractéristiques extraites manuellement.

Pour notre travail, nous utilisons un empilement d'auto-encodeurs, principe présenté dans la sous-section 7.2.4. Un auto-encodeur, rappelons-le, est un réseau de neurones qui apprend à travers des convolutions successives une représentation des données en entrée. Cette représentation, si elle est bonne, doit permettre à l'auto-encodeur de recréer avec une suite de déconvolution le signal d'entrée (voir sous-section 7.2.3). Le principe est le même que les RBM sauf qu'un auto-encodeur utilise des unités déterministes et dans ce cas en particulier des unités de type *softsign* (voir le rappel de la sous-section 7.2.1).

L'extraction des caractéristiques avec un filtre appris par un empilement d'auto-encodeurs et leur utilisation pour la classification reste encore mal explorée dans l'état de l'art. Nous citons par exemple le travail de [Chen 2015] sur l'extraction et la sélection des caractéristiques apprises pour la tâche de l'analyse de la mise en page des manuscrits anciens. Les auteurs ont utilisé l'algorithme des séparateurs à vaste marge (SVM) entraîné avec les caractéristiques apprises. Les auteurs de [Wei 2015] ont utilisé la même architecture que [Chen 2015] et ont travaillé sur la sélection des caractéristiques apprises par l'utilisation de l'algorithme de *feed forward selection*.

7.4.1 Apprentissage des caractéristiques

Dans ce qui suit, nous présentons trois différentes architectures [7] avec différentes dispositions de SCAE. La première est un empilement classique d'auto-encodeurs convolutifs (SCAE), la deuxième est un empilement de SCAE avec une dernière couche d'*Extremum pooler* à la suite de la dernière couche SCAE, et la troisième est une architecture en Sandwich où chaque couche SCAE est suivie d'une couche d'*Extremum pooler*. Nous utilisons l'espace de travail *N-light-N* proposé par [Seuret 2016] [8], qui nous a permis d'implémenter les trois différentes architectures. À la sortie de chaque architecture, les caractéristiques apprises sont récupérées dans

7. Tel que c'est défini dans la sous-section 7.2.5.
8. https ://github.com/seuretm/n-light-n

un vecteur et utilisées pour entraîner les mêmes algorithmes proposés dans notre référentiel.

Dans la première architecture, nous souhaitons expérimenter l'effet sur le taux de justesse quand nous exécutons une rétropropagation sur les poids des SCAE. Dans la deuxième et troisième architecture, nous souhaitons expérimenter deux différentes dispositions des couches d'*Extremum pooler*. Ces dispositions sont inspirées des bonnes pratiques de l'état de l'art [Krizhevsky 2012, LeCun 2012]. Notons également qu'une rétropropagation sur des couches de sous-échantillonnage comme l'*Extremum pooler* est possible si nous pouvons garder une trace de tous les poids avant l'exécution d'un *Extremum pooler*. Ceci est en cours de développement dans l'espace de travail *N-light-N*.

Les images qui sont données en entrée aux différentes architectures sont d'une taille égale à $256 \times 256 \times 1$. Cette taille correspond à celle de la capture prise à un endroit aléatoire de la page d'un manuscrit appartenant aux données d'entraînement, le chiffre 1 correspond au canal gris, si les images été en RGB (i.e. Red, Green, Blue), nous aurions mis 3. Cette taille nous a permis d'obtenir un traitement plus rapide sur les processus CPU. Notons que nous avons également essayé des résolutions supérieures e.g. 512×512, ceci n'a pas donné de meilleurs résultats au moment de la classification.

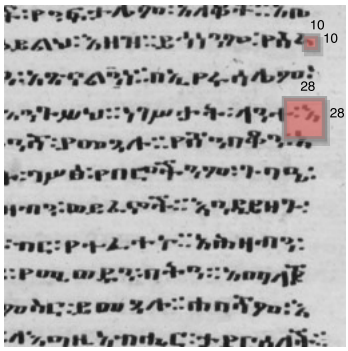


FIGURE 7.14 – Exemple des informations capturer par les filtres, dessinés en rouge, obtenus avec la deuxième, filtre de taille 10×10, et troisième architecture, filtre de taille 28×28.

7.4.1.1 Première architecture

La première architecture correspond à un empilement classique d'auto-encodeurs. La première couche filtre l'image en entrée de $256 \times 256 \times 1$ avec 8 masques de taille $3 \times 3 \times 1$ et une foulée de 1 (*stride* en anglais), c'est-à-dire que le masque est appliqué sans chevauchement de pixels. La couche suivante prend en entrée la sortie de la première couche, qui est un filtre de taille $11 \times 11 \times 8$ et le filtre avec 16 masques d'une taille de $3 \times 3 \times 1$. La troisième couche prend en entrée le filtre de la couche précédente de taille $9 \times 9 \times 16$ et y applique 36 masques de taille $5 \times 5 \times 1$.

Enfin, la dernière couche prend la sortie de la couche précédente, le filtre avec 36 masques et nous obtenons une représentation d'une taille de 36 neurones. Une fois l'image recrée à partir de cette représentation, nous obtenons après calcul de $\hat{x}$ un filtre de taille 13×13. Dans cette première architecture, nous n'utilisons pas de couche de sous-échantillonnage, c'est à dire de couche d'extremum pooler ou max-pooling.

7.4.1.2 Deuxième architecture

La deuxième architecture est similaire à la première, à l'exception de la dernière couche qui est remplacée par un *extremum pooler* qui nous donne un filtre de plus petite taille, égale à 10×10. Avec cette architecture, nous souhaitons étudier l'effet de l'extremum-pooler, mis en dernière couche, sur les résultats de classification.

Les couches d'extremum pooling ne contribuent pas à l'apprentissage du réseau, mais réduisent la taille du problème en obligeant le filtre de convolution à ne prendre que la valeur maximale ou extrême d'un ensemble de neurones. Notons que nous ne pouvons pas exécuter de rétropropagation sur ce type d'architecture. Car nous ne pouvons calculer que la dérivée $\frac{\partial f_n(x_{n-1}, W_n)}{\partial x_n}$ de l'erreur de la sortie de la dernière couche. Un calcul qui nécessite de dériver l'erreur totale obtenue par le réseau et calculée par l'équation 7.21.

$$\frac{\partial E}{\partial x_{n-1}} = \frac{\partial E}{\partial x_n} \times \frac{\partial f_n(x_{n-1}, W_n)}{\partial x_n} \tag{7.21}$$

Où f_n est la fonction de sortie calculée pour la couche n, x_{n-1} est le vecteur donné en entrée par la couche $n-1$ et W_n est la matrice des poids des connexions entre les neurones de la couche n. Enfin, E est l'erreur totale calculée avec la fonction de perte 7.11 à la sortie du réseau.

Ceci est une limite actuelle de notre implémentation framework *N-light-N* que nous utilisons.

7.4.1.3 Troisième architecture

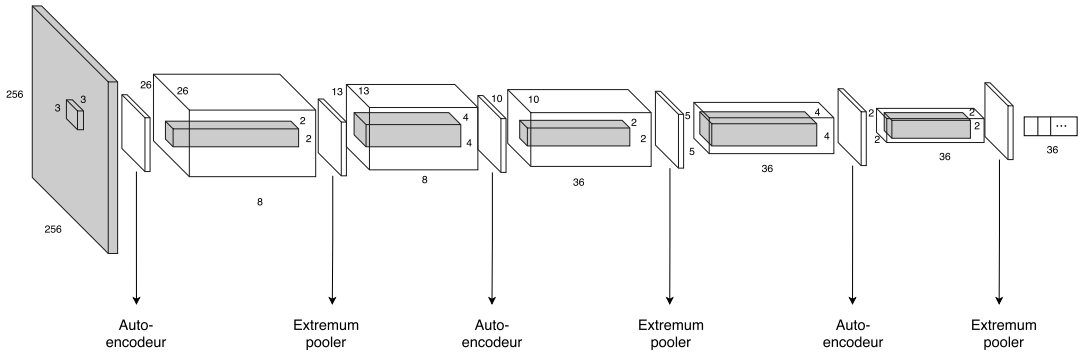
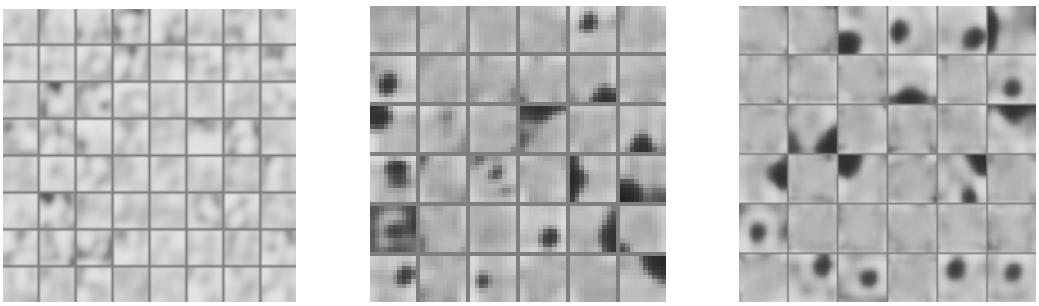


FIGURE 7.15 – Un auto-encodeur avec une seule couche cachée

La troisième architecture correspond à un *sandwich* d'auto-encodeurs empilés avec des couches d'*extremum pooler*, elle est illustrée dans la figure 7.15. Dans la première couche, l'auto-encodeur apprend à partir d'un petit patch 3×3 pixels. La dimension du vecteur d'entrée x est la concaténation des valeurs du signal au niveau de gris des pixels dans le voisinage du patch $3 \times 3 \times 1 = 9$. La foulée est initialisée à 1. Le nombre de neurones ou d'unités cachées est initialisé à 8.

Donc, la première couche de convolution dans l'architecture des auto-encodeurs empilés filtre l'image en entrée de $256 \times 256 \times 1$ avec 8 masques de taille $3 \times 3 \times 1$, avec une foulée (*stride* en anglais) de 1 pixel. On applique à la première couche un *extremum pooler*, qui prend en entrée la sortie de la première couche et la filtre avec 8 masques de taille $2 \times 2 \times 8$. La deuxième couche de convolution prend en entrée la couche sur laquelle nous avons appliqué l'*extremum pooler* et filtre la sortie avec 8 masques. On y applique ensuite un *extremum pooler*. La troisième couche de convolution a 36 filtres de taille $4 \times 4 \times 36$.

Nous obtenons avec la dernière couche une représentation de l'image compressée par 36 neurones. Nous appliquons, par la suite, le processus inverse pour reproduire l'image en entrée. La taille de patch à la sortie du réseau après le calcul de $\hat{x}$ est égale à 28×28.



(*a*) (*b*) (*c*)

FIGURE 7.16 – Exemples des caractéristiques apprises à partir des quatre scriptes (a) première architecture (b) deuxième architecture (c) troisième architecture

Les filtres obtenus avec la dernière couche des trois différentes architectures de SCAE sont illustrés dans la figure 7.16. Nous constatons dans la figure (a) 7.16, que le réseau a été capable d'apprendre des traits courbés qui ne sont pas clairement identifiables comme étant des lettres. La deuxième et la troisième architecture, illustrées respectivement dans la figure (b) 7.16 et (c) 7.16, nous donnent des filtres où nous pouvons distinguer des structures en gouttes, rondes ou plus allongées pour prendre une forme plus ovale. Après une inspection des filtres, nous pouvons passer à la phase d'extraction des caractéristiques.

Pour extraire le vecteur de caractéristiques apprises par l'architecture, que nous avons décrites ci-dessus, nous appliquons aléatoirement 100 fois, pour tous les scripts, et sur chaque page, de chaque manuscrit le patch de taille 28×28. Dans nos expérimentations, pour la phase d'entraînement, ainsi que la phase de test, les pixels

Auto-encodeurs empilés		Precision moy.	Rappel moy.	F1 moy.	Justesse
RF	Ethiopien	**0.59**	0.54	0.56	**55.57%**
	Syriaque	0.52	0.11	0.19	
	Grec	0.49	0.51	0.50	
	Latin	0.58	**0.71**	**0.64**	
SVM	Ethiopien	**0.70**	0.36	0.48	48.91%
	Syriaque	0.57	0.02	0.04	
	Grec	0.41	0.41	0.41	
	Latin	0.48	**0.75**	**0.59**	
ANN	Ethiopien	**0.24**	**0.84**	**0.37**	25.83%
	Syriaque	0.13	0.00	0.01	
	Grec	0.17	0.33	0.19	
	Latin	0.00	0.00	0.00	

TABLE 7.3 – Les résultats des algorithmes de discrimination de scripts entraînée avec les caractéristiques extraites avec l'auto-encodeur convolutif empilé. Les meilleurs résultats sont mis en gras.

dans la bordure de chaque page ne sont pas pris en compte ; seuls les pixels centrés dans un environ de 256×256 sont considérés. La figure 7.9 illustre comment nous appliquons le patch, dessiné en rouge, sur la page manuscrite. La même opération est valable pour la première et deuxième couche.

7.4.2 Classification

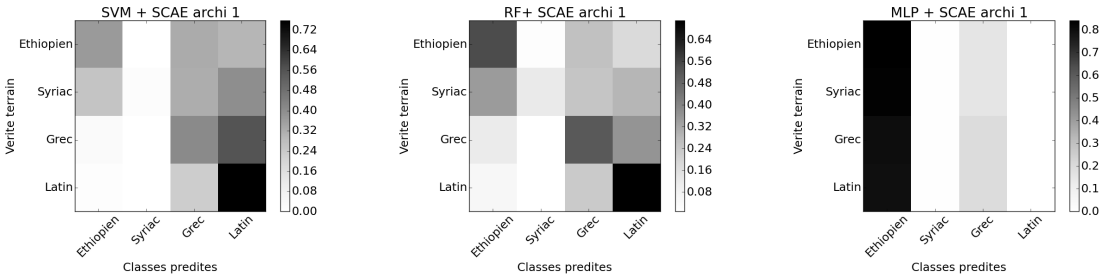
Afin de discriminer les quatre scripts que nous étudions dans ce chapitre, nous avons exploité les caractéristiques apprises par les trois architectures de SCAE pour entraîner les trois algorithmes de notre référentiel empirique à savoir un réseau de neurones artificiels (ANN) à $\frac{\#attributs+\#classes}{2}$ couches, l'algorithme des forêts aléatoires (RF) et l'algorithme des séparateurs à vaste marge (SVM).

7.4.2.1 Résultats

Dans cette partie, nous présentons les résultats d'identifications des scripts : Éthiopien, Syriaque, Grecque et Latin. La présentation des résultats est divisée en deux parties. La première correspond à la discrimination des scripts suite à l'extraction des caractéristiques apprises par les trois architectures SCAE. La deuxième correspond au préapprentissage des deux architectures à convolution présentées précédemment.

7.4.2.2 Première architecture

Dans la table 7.3, nous rapportons les résultats de la première architecture de SCAE qui ne comporte pas de couches d'extremum pooler. Les performances des trois algorithmes, c'est à dire, les forêts aléatoires (RF), les réseaux de neurones artificiels (ANN) et l'algorithme des séparateurs de vastes marges (SVM), ont nettement baissé par rapport à notre référentiel empirique. La détérioration la plus



(a) (b) (c)

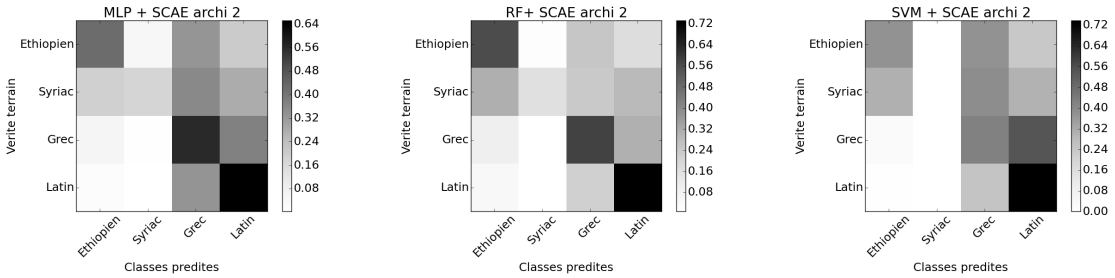
FIGURE 7.17 – Matrice de confusion des algorithmes de classification avec les caractéristiques apprises extraites avec l'architecture 1 de SCAE. La figure (a) illustre les résultats de discrimination du ANN. La figure (b) illustre les résultats de discrimination des forêts aléatoires. Enfin, la figure (b) illustre les résultats de discrimination exécutés par SVM.

Auto-encodeurs empilés		Precision moy.	Rappel moy.	F1 moy.	Justesse
RF	Ethiopien	**0.63**	0.56	0.59	**59.28%**
	Syriaque	0.61	0.16	0.25	
	Grec	0.54	**0.58**	0.56	
	Latin	0.61	0.73	**0.66**	
ANN	Ethiopien	**0.71**	0.42	0.53	52.77%
	Syriaque	0.56	0.17	0.26	
	Grec	0.44	0.56	0.49	
	Latin	0.54	**0.65**	**0.59**	
SVM	Ethiopien	0.68	0.38	0.48	48.77%
	Syriaque	**0.80**	0.00	0.00	
	Grec	0.38	0.42	0.40	
	Latin	0.40	**0.50**	**0.73**	

TABLE 7.4 – Les résultats des algorithmes de discrimination de scripts entraînés avec les caractéristiques extraites par l'auto-encodeur convolutif empilé. Les meilleurs résultats sont mis en gras.

remarquable est celle du ANN qui est passé de 89.07% à 25.83%. Notons également les faibles performances pour la discrimination du script syriaque, et ce pour tous les algorithmes dans la table 7.3.

En observant les matrices de confusion illustrées dans la figure 7.17, nous pouvons constater que les performances de la détection du script syriaque sont faibles comparées aux trois autres scripts. Étonnamment, ANN n'a pas pu détecter une seule instance des pages en Latin donnée lors du test. Nous ne nous pouvons pas, pour l'instant, donner une explication pour les résultats obtenus avec le ANN, surtout en ce qui concerne la discrimination du script Latin, qui comme nous le verrons dans la suite, est toujours bien détecté.



(a) (b) (c)

FIGURE 7.18 – Matrices de confusions des algorithmes de classification avec les caractéristiques apprises extraites avec l'architecture 2 de SCAE. La figure (a) illustre les résultats de discrimination du ANN. La figure (b) illustre les résultats de discrimination des forêts aléatoires. Enfin, la figure (b) illustre les résultats de discrimination exécutés par SVM.

Auto-encodeurs empilés		Precision moy.	Rappel moy.	F1 moy.	Justesse
RF	Ethiopien	0.70	**0.82**	0.75	**71.18%**
	Syriaque	0.69	0.24	0.36	
	Grec	0.68	0.67	0.67	
	Latin	**0.74**	0.78	**0.76**	
ANN	Ethiopien	**0.76**	0.66	**0.71**	64.46%
	Syriaque	0.49	0.34	0.41	
	Grec	0.55	0.65	0.60	
	Latin	0.69	**0.70**	0.69	
SVM	Ethiopien	**0.66**	0.71	**0.68**	56.96%
	Syriaque	0.13	0.00	0.00	
	Grec	0.50	0.43	0.46	
	Latin	0.55	**0.74**	0.63	

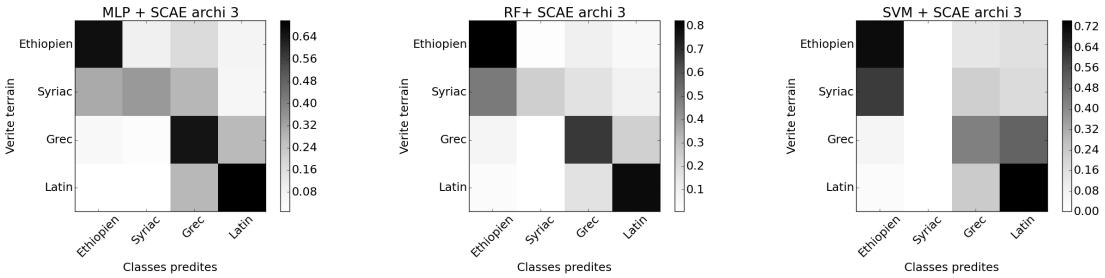
TABLE 7.5 – Les résultats des algorithmes de discrimination de scripts entraînée avec les caractéristiques extraites avec l'auto-encodeur convolutif empilé. Les meilleurs résultats sont mis en gras.

7.4.2.3 Deuxième architecture

Dans la table 7.4, nous rapportons les résultats des trois algorithmes : RF, ANN et SVM. Ces derniers sont entraînés avec les caractéristiques apprises par la deuxième architecture SCAE que nous avons présentées dans la section 7.4.1.1. Les performances globales des algorithmes se sont améliorées par rapport à l'architecture précédente. Les forêts aléatoires ont pris les devants sur les réseaux de neurones artificiels (ANN) avec une justesse de 59.28% en comparaison avec le référentiel construit avec les caractéristiques manuelles. Notons les performances très faibles pour l'identification du script syriaque, qui rappelons-le, a le moins de données dans notre *vérité terrain*.

En analysant les matrices de confusion dans la figure 7.19, nous constatons que les ANNs ont une meilleure capacité de généralisation pour l'Éthiopien, le Grecque

et le Latin. Malgré un score de justesse plus élevé, les forêts aléatoires ont plus de mal que les ANNs à identifier le script syriaque. L'algorithme SVM fournit les performances les plus faibles, et n'arrive même pas à identifier une seule instance du script syriaque.



(a) (b) (c)

FIGURE 7.19 – Matrice de confusion des algorithmes de classification avec les caractéristiques apprises extraites avec l'architecture 3 de SCAE. La figure (a) illustre les résultats de discrimination du ANN. La figure (b) illustre les résultats de discrimination des forêts aléatoires. Enfin, la figure (b) illustre les résultats de discrimination exécutés par SVM.

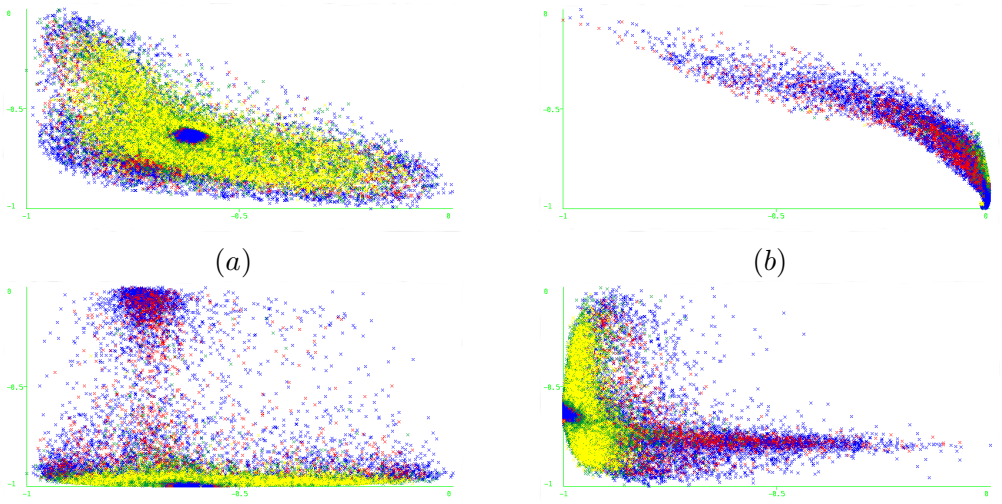
7.4.2.4 Troisième architecture

Dans la table 7.5, nous présentons les résultats de la discrimination des scripts après extraction des caractéristiques apprises avec la troisième architecture de SCAE. Contrairement au référentiel construit avec les caractéristiques manuelles, les forêts aléatoires obtiennent de meilleurs résultats que les réseaux de neurones artificiels, avec une justesse égale à 71.18%. Les résultats des trois algorithmes se sont nettement améliorés par rapport aux deux premières architectures de SCAE, mais n'ont pas atteint les performances des algorithmes entraînés avec les caractéristiques HOG. Nous pouvons clairement observer à travers la figure 7.19, la pauvre performance pour la discrimination du script syriaque, surtout pour le SVM qui, comparé à ces résultats, possédait une bonne capacité de généralisation quand il a été entraîné avec les caractéristiques manuelles de HOG.

Dans cette partie, nous avons présenté les résultats obtenus avec différentes stratégies de classification. Le point commun entre ces stratégies ce sont les caractéristiques apprises avec les trois différentes architectures de SCAE que nous avons conçues dans la section 7.4.1. Après plusieurs exécutions, en optimisant les différents paramètres des classifieurs (e.g. nombre d'arbres dans les forêts aléatoires ou nombre de neurones dans les réseaux de neurones artificiels), nous avons constaté qu'il n'y avait pas de changement significatif dans les résultats. Nous avons donc conclu que les caractéristiques apprises étaient seules responsables des résultats obtenus et nous avons décidé, après plusieurs optimisations des trois architectures, de visualiser et par la suite analyser ces caractéristiques.

7.4.3 Analyse des résultats

À partir d'un ensemble réduit de données sans un prétraitement préalable (e.g. nettoyage du bruit dans l'image, binarisation, détection des mots, détection des composantes connexes), nous espérions qu'un pré entraînement non supervisé, couche après couche, nous permettrait d'extraire des caractéristiques de plus en plus abstraites et donc, par la suite, pertinentes pour une classification selon un critère supervisé. Avec la série d'expériences que nous avons menées, nous voulions tester notre hypothèse qui stipule qu'un empilement d'auto-encodeurs nous permettra d'obtenir ce genre de caractéristiques.



(a) (b) (c) (d)

FIGURE 7.20 – Visualisation des caractéristiques apprises par l'auto-encodeur. Chaque point correspond à une valeur dans le vecteur de caractéristiques apprises après la phase d'encodage dans les différentes architectures. La couleur bleue correspond au script Éthiopien, la couleur rouge au Syriaque, la couleur verte au Grec et enfin le jaune au Latin.

Nous nous attendions, donc, à ce que les représentations apprises par les trois architectures SCAE nous donnent de meilleurs résultats que ceux présentés dans notre référentiel empirique construit avec l'extraction de caractéristiques manuelles. Cependant, la meilleure performance obtenue est celle des forêts aléatoires (RF), entraînées avec les caractéristiques extraites de la troisième architecture SCAE. Les RF ont obtenu un taux de justesse égale à 71.18% (voir table 7.5) comparé au 89.07% (voir table 7.2), obtenu avec un réseau de neurones artificiel ou ANN, entraînés avec les caractéristiques manuelles HOG.

En observant les caractéristiques de la troisième architecture dans la figure 7.16 (b), nous pouvons distinguer des structures en gouttes laissant à présager que ces représentations étaient utiles pour le RF. Malgré que les caractéristiques illustrées dans la figure 7.16 (a), apprises par la première architecture semblent plus intéressantes. Car, nous pouvons distinguer des contours de lettres et des courbatures de

traits, laissant à penser que nous pouvons avoir de meilleurs résultats de discrimination. Ceci n'a pas empêché d'avoir le plus mauvais taux de justesse avec 25.83% obtenu avec un ANN entraîné sur ces caractéristiques.

Suite au mauvais résultat que nous avons obtenu avec la première architecture, nous avons eu l'intuition suivante : comme nous travaillons au niveau des pixels d'une page manuscrite, le risque que l'architecture SCAE apprenne une représentation inintéressante est plus fort comparé à un SCAE entraîné au niveau d'un caractère ou un mot ou une composante connexe. En effet, le patch de l'architecture SCAE a plus de chance de tomber sur du blanc ou du bruit dans l'arrière-plan que sur des lettres manuscrites.

Cette intuition revient, donc, à attaquer le problème à la phase de l'encodage afin que le SCAE puisse apprendre une représentation des images des documents données en entrée. C'est pour cette raison que nous avons décidé d'introduire, dans la deuxième et troisième architecture SCAE, de la dispersion dans les valeurs des neurones en ajoutant des couches qui orientent le gradient vers le neurone avec la valeur maximale.

Les caractéristiques extraites avec les auto-encodeurs à convolution sont visualisées dans la figure 7.20, les classes correspondant aux scripts Éthiopien, syriaque, Grec et Latin sont colorées respectivement en bleu, rouge, vert et jaune. Nous pouvons clairement observer que dans les figures 7.20 (a), 7.20 (c) et 7.20 (d) les caractéristiques correspondant au script latin sont clairement prédominantes, les autres scripts y sont juste interposés.

Nous observons dans la figure (c) 7.20 que les deux scripts Éthiopien et syriaque se détachent un peu, mais sans pour autant constituer un nuage complètement indépendant des scripts grec et latin. Ces derniers sont toujours interposés. Selon la visualisation, nous pouvons conclure que le SCAE confond entre le latin et le grec, le Syriaque et l'Éthopien. Ceci n'est pas vraiment étonnant vu la manière dont ils sont écrits avec un large trait (voir figure 7.1). Il nous faut, donc, plus de données pour que le CAE généralise son apprentissage et distingue des caractéristiques plus fines et qu'il soit plus sensible à la variation.

Discussion

Les résultats de visualisation montrés dans la figure 7.20 viennent appuyer ceux démontrés dans les tables 7.3, 7.4, 7.5. La capacité de généraliser l'apprentissage d'une représentation à caractère non supervisé apprise à partir d'une quantité limitée de données et qui sont, de plus, non traitées est mise en doute. Il est vrai que l'ajout des couches d'extremum pooling a aidé à améliorer considérablement la justesse des résultats en les faisant passer de 55.57% à 71.18% pour les forêts aléatoires. Cependant, ceci n'égale pas les performances de notre référentiel empirique avec un taux de justesse égale à 86.15% pour le même algorithme (i.e. forêt aléatoire).

Nous ne pouvons pas avoir une seule explication à ces résultats, vu l'espace de paramètres que nous devons explorer. Par exemple, comme nous le savons, les auto-encodeurs construisent une représentation compacte d'une image, cette repré-

sentation doit permettre une reconstruction optimale de l'image d'origine, ce qui revient à optimiser la dimension des caractéristiques utilisées lors de la reconstruction. Cependant, nous sommes encore incertains si la dimension optimisée lors de l'apprentissage de l'auto-encodeur est optimale en entrée d'un algorithme de discrimination ou classification.

Aussi, dans la procédure d'entraînement, nous prenons en compte un nombre de patchs aléatoires à partir de l'image. Ces patchs peuvent être tout blancs, ce qui rend la généralisation de la sortie du masque au moment de la convolution encore plus difficile. On est toujours aussi exposé au risque de perte du signal d'origine à cause d'une mauvaise initialisation des poids au sein de la toute première couche des architectures. Une possible solution à entreprendre consiste en l'utilisation du *finetuning*.

L'ajustement fin ou *finetuning* en anglais est l'approximation de la meilleure valeur possible des paramètres du réseau que nous avons déjà entraîné et est basé sur le principe de l'apprentissage par transfert [Pan 2010]. Dans le cas des SCAE, cette technique peut augmenter considérablement les performances du réseau où nous traitons toutes les couches comme un seul modèle. Nous n'avons pas utilisé l'ajustement fin dans nos expérimentations, car, et comme expliquée précédemment, nous ne pouvons pas utiliser de rétropropagation sur les couches avec un gradient orienté vers le neurone avec la valeur maximale. Ceci est pour l'instant une limitation du framework *N-light-N*.

Notons que nous avons, aussi, essayé d'optimiser nos architectures en explorant les auto-encodeurs débruitant [Vincent 2008] où il s'agit d'apprendre une représentation utile à la corruption synthétique de l'entrée. L'auto-encodeur débruitant ne diffère pas de l'auto-encodeur ordinaire, à part le fait que les données x en entrée sont corrompues, dénotées $\tilde{x}$, et que les données en sortie sont aussi données pour effectuer la reconstruction $z = g_{\theta'}(f_\theta(\tilde{x}))$ de la représentation cachée $y = f_\theta(\tilde{x})$. Malheureusement, la nature de nos données ne nous permet pas d'appliquer ce genre de technique qui a besoin de données nettoyées et normalisées.

Notons, aussi, que nous avons changé la résolution de l'image en entrée en passant de 256×256 à 512×512, afin de vérifier si une augmentation dans le nombre de pixels aurait un impact sur le taux de justesse obtenu. Nous n'avons pas eu de changement sensible qui mériterait un intérêt particulier.

7.5 Contribution pour l'apprentissage supervisé de scripts avec les architectures à convolution

Dans la section précédente, nous nous sommes intéressés à l'apprentissage non supervisé des caractéristiques que nous avons utilisées par la suite pour entraîner les trois algorithmes utilisés pour construire notre référentiel empirique. Nous avons également analysé les résultats de discrimination obtenus avec le rajout de deux couches connexes de neurones pré entraînés.

Dans cette section, nous explorons une architecture plus simple faite d'un petit

nombre de couches convolutives qui sont normalisées par des couches de *max pooling* et donc appliquées sur un réseau de type *Sparse*. Nous analysons d'abord le rôle de la fonction d'initialisation des poids des connexions neuronales. Ensuite, l'optimisation des différents paramètres de l'algorithme de descente de gradient. Nous voulons jauger, à travers les expérimentations, l'effet de l'optimisation des paramètres sur la justesse des résultats de la discrimination des quatre scripts étudiés.

La motivation d'entraîner des couches profondes vient des travaux de Y. Bengio, qui stipule que si nous avons beaucoup de poids dans notre réseau, alors les minimums locaux deviennent, à peu près, égaux aux minimums globaux. Autrement dit, lorsque nous exécutons une descente de gradient, la probabilité de la direction de la descente est proportionnelle au nombre de poids. Néanmoins, dans les travaux sur les architectures profondes [Hinton 2006b, Bengio 2006, Erhan 2009, Larochelle 2009] initiés après l'année 2005, les chercheurs ont supposé que c'était difficile d'entraîner un réseau constitué de plusieurs couches avec l'algorithme de descente de gradient, ceci à cause des mauvais résultats obtenus. C'est, entre autres, avec les travaux de [Krizhevsky 2012] que les architectures profondes ont été remises dans la course.

Les auteurs de [Krizhevsky 2012] ont entraîné un large réseau à convolution avec 60 millions de paramètres et un jeu de données se composant de plus d'un million d'images annotées avec 1000 classes. Les bons résultats obtenus ont prouvé que c'était possible d'entraîner des architectures profondes à condition d'apporter une attention particulière à l'optimisation des paramètres de la descente de gradient, mais aussi à l'initialisation aléatoire des poids des connexions neuronales.

Notons, aussi, que les auteurs ont utilisé deux améliorations majeures pour les architectures profondes : la transformation non linéaire appelée *Rectified Linear Unit* (ReLU) [Nair 2010] pour améliorer la convergence de l'apprentissage lors de l'activation des neurones dans le réseau, et la technique du *Dropout* [Srivastava 2014] pour réduire le sur apprentissage. Nous n'utiliserons pas de *Dropout* dans nos expérimentations, mais la méthode de dégradation des pondérations appelée *weight decay*. Cette méthode consiste à rajouter une pénalité à la fonction d'erreur. Une étude comparative des deux méthodes de régulation de l'apprentissage (i.e. le *Dropout* et le *weight decay*) a été présentée dans les travaux de [Shi 2016]. Notons que nous remplacerons également, dans ce qui suit, la fonction d'activation *softsign* par la fonction ReLU.

Dans le cadre les auteurs de [Bluche 2013] ont utilisé les réseaux de neurones à convolution afin d'extraire les caractéristiques pour la reconnaissance des mots écrits à la main. Dans leur article, les auteurs proposent une comparaison entre les caractéristiques extraites manuellement et celles apprises de manière supervisée. Pour la tâche de la classification, les auteurs ont combiné les réseaux de neurones à convolution et les chaînes de Markov cachées. Leurs résultats prouvent que la combinaison avec les architectures profondes permet d'obtenir de meilleurs résultats que l'utilisation seule d'une chaîne de Markov cachée. Également, les auteurs de ce travail [Du 2015] ont utilisé les réseaux de neurones à convolution pour apprendre de manière supervisée les caractéristiques au niveau des caractères chinois. Ils ont par la suite, donné en entrée les caractéristiques apprises à un algorithme de classification

par prototypage des caractéristiques.

Nous utiliserons, dans cette partie, la plateforme *Caffe* pour la conception de nos architectures profondes convolutives. *Caffe* est développé par l'équipe de [Jia 2014] à Berkley et a connu, auprès de la communauté travaillant sur les architectures profondes, un franc succès. Notons qu'il y a d'autres plateformes aussi efficaces que *Caffe*, cet article [Bahrampour 2015] les compare au niveau de la vitesse, de la consommation de ressources, etc. Notons aussi que *Caffe* offre la possibilité d'implémenter des réseaux *Sparse* c'est-à-dire que la valeur des neurones en sortie est dans l'intervalle $[0, 1]$.

7.5.1 Sur l'importance de l'initialisation des paramètres

L'implication majeure des travaux de [Krizhevsky 2012], à notre sens, est la mise en évidence de l'importance de l'initialisation des paramètres (i.e. poids des connexions neuronales, paramètres de la descente de gradient) au début de chaque entraînement. Ceci a été souligné par les travaux de [Sutskever 2013b, Sutskever 2013a] sur les réseaux de neurones récurrents. En effet, la méthode d'entraînement d'une structure neuronale, qu'elle soit profonde ou pas consiste en la transformation de la problématique d'apprentissage supervisée en une problématique d'optimisation des paramètres $\theta = \{W, V, b, c\}$.

L'optimisation est traduite par une approximation des valeurs qui minimisent la fonction objectif ou de perte $f(\theta)$ (un exemple de $f(\theta)$ est illustré dans la figure 7.21). L'approximation est exécutée par l'algorithme de descente de gradient et par une rétropropagation sur les poids des connexions neuronales. Cependant, cette optimisation qui est basée sur une initialisation aléatoire des paramètres en amont résulte en de mauvaises performances de généralisation et même de mauvaises solutions au niveau des données d'entraînement [Larochelle 2009, Erhan 2009, Hinton 2006a].

Ceci s'explique par le choix d'un mauvais bassin d'attraction dans l'espace des paramètres du réseau de neurones, résultant en l'exploration de mauvais minimaux locaux, plateaux, mais aussi des points col (*saddles point* en anglais), comme l'avaient démontré les auteurs de ce travail [Dauphin 2014]. Néanmoins, même avec le choix du bon bassin d'attraction, nous ne pouvons pas être sûrs que le modèle aura une bonne généralisation, comme c'était le cas avec notre première architecture de SCAE étudiée dans la section 7.4.1.

Face à une mauvaise généralisation, l'on parle de sur apprentissage qui nécessite l'usage de la régularisation pour le choix du modèle le plus simple. Nous nous retrouvons dans la problématique de la sélection du modèle le plus simple ou de l'hypothèse la plus courte (Définition A.9) qui aura le plus de capacité à généraliser ses prédictions en vue de nouveaux exemples.

Avec les travaux de [Hinton 2006a], une nouvelle stratégie de régularisation basée sur un apprentissage non supervisé a vu le jour. Il s'agit d'initialiser les paramètres (poids et biais) avec un critère non supervisé avant d'entamer l'apprentissage supervisé avec la descente de gradient. Les auteurs ont utilisé une architecture d'empilement de machines de Boltzmann restreintes (RBM pour *Restricted Boltzmann*

Machines).

D'autres travaux ont exploré l'empilement d'auto-encodeurs ordinaires [Bengio 2007b] (SAE pour *Stacked Autoencoders*, ou encore l'empilement d'auto-encodeurs débruiteurs [Vincent 2010] (SDAE pour *Stacked Denoising Autoencoders*). Dans nos travaux présentés dans la section 7.4, nous avons utilisé un empilement d'auto-encodeurs de type convolutif (SCAE). Le rajout de plusieurs couches d'auto-encodeurs peut être une manière d'éviter l'initialisation aléatoire des poids du réseau de neurones dédié à la classification, cependant il faut toujours que l'on choisisse les poids du réseau de préapprentissage que ce soit un RBM, un SDAE, ou un SCAE.

Rappelons que la convolution du signal en entrée, dans ce type d'architecture, est aussi dépendante des valeurs initiales que nous attribuons aux poids des connexions neuronales. Si les poids des connexions entre les neurones du réseau sont trop petits, alors le signal d'origine rétrécit avec chaque multiplication et transformation linéaire (i.e. une opération de convolution), jusqu'à ce qu'il soit dégradé, atteins une valeur nulle (*decay to zero*), et ne soit plus utilisable en sortie du réseau. Par contre si les poids dans le réseau sont trop grands, alors le signal d'origine croît lors de la convolution, jusqu'à ce qu'il devient trop grand pour être utile. Ceci a une conséquence directe sur l'entraînement qui ne convergera jamais vers une valeur et le système peut devenir par la suite chaotique.

Initialisation des poids

Nous avons utilisé dans nos travaux sur les SCAE une initialisation aléatoire des poids des connexions neuronales qui est tirée d'une distribution uniforme où $W \in]-\frac{1}{\sqrt{n_{in}}};\frac{1}{\sqrt{n_{in}}}[$. Cet intervalle a été déterminé empiriquement, où nous avons constaté après plusieurs essais que l'utilisation de la racine nous évitait de perdre le signal en entrée n_{in}.

Nous utiliserons dans cette partie une autre méthode pour l'initialisation des poids des connexions neuronales. La valeur initiale des poids est tirée d'une distribution gaussienne, avec une moyenne nulle et une variance décrite par l'équation suivante

$$Var(G) = \frac{1}{n_{in}} \tag{7.22}$$

Où G est la distribution gaussienne, et n_{in} est le nombre de neurones donné en entrée à la distribution. Afin de stabiliser le signal en entrée du réseau (i.e. éviter qu'il explose ou diminue jusqu'à disparaître complètement), notre objectif consiste à garer la même variance du signal lors de l'exécution de l'algorithme de descente de gradient. L'on appelle ce type d'initialisation aléatoire, l'initialisation de Xavier [Glorot 2010].

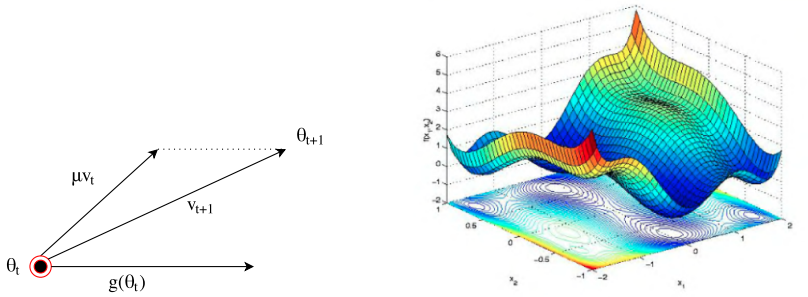
La variance de la distribution gaussienne telle qu'elle est calculée dans l'équation 7.22, est une implémentation de la plateforme d'apprentissage des architectures profondes appelée *Caffe* [Jia 2014], que nous utilisons dans cette partie afin de mener

nos expérimentations. La variance telle qu'elle est calculée dans les travaux d'origine de [Glorot 2010] est donnée par l'équation 7.23.

$$Var(G) = \frac{2}{n_{in} + n_{out}} \tag{7.23}$$

Où n_{out} est le nombre de neurones en sortie. Afin de préserver la variance du signal en entrée ainsi qu'en sortie après l'application de l'algorithme de rétropropagation, la variance est calculée sur la moyenne des neurones donnés en entrée et en sortie.

Élan du gradient



(a) (b)

FIGURE 7.21 – (a) Calcul de la direction du gradient à l'étape θ_{t+1}, Où $g(\theta_t) = \varepsilon \nabla f(\theta_t)$. (b) Illustration d'une fonction objectif $f(\theta)$.

Étant donné la fonction objectif $f(\theta)$ que nous souhaitons minimiser, l'élan μ est donné par :

$$v_{t+1} = \mu v_t - \varepsilon \nabla f(\theta_t) \tag{7.24}$$
$$\theta_{t+1} = \theta_t + v_{t+1} \tag{7.25}$$

Où $\varepsilon > 0$ est le taux d'apprentissage, $\mu \in [0, 1]$ est le coefficient de l'élan et $\nabla f(\theta_t)$ est le gradient à θ_t. Nous avons appelé le coefficient μ élan, par abus de langage et suite à la traduction du terme anglais *momentum* que nous retrouvons dans l'état de l'art. Cependant, μ représente en réalité le frottement exercé sur la vitesse v, il a pour effet la réduction et l'alourdissement de l'énergie cinétique du système.

Si nous considérons l'espace de la fonction objectif $f(\theta)$, illustrée dans la figure (b) [9] 7.21, comme étant un massif rocheux, que nous nous positionnons en haut d'une colline et que notre but est d'atteindre le bas de cette colline, alors le frottement μ que nous exerçons sur la vitesse de déplacement nous "aide" à nous arrêter, une

9. Figure prise des slides Multilayer Neural Networks par Leon Bottou présenté dans *Deep learning summer school, Montreal, 2015*

fois la descente terminée. La direction du gradient à θ_{t+1} est calculée par l'équation 7.25 qui est illustrée dans la figure (a) 7.21.

Pour l'optimisation de la fonction objectif, nous nous sommes concentrés sur les paramètres suivants :

— Le taux d'apprentissage ε.
— L'élan de l'apprentissage μ. Notons que nous nous intéressons uniquement à l'élan classique de l'apprentissage. Nous n'explorons pas l'effet du paramètre de Nestrov pour l'accélération du gradient (NAG), bien que dans leur article [Sutskever 2013b] les auteurs démontrent que le NAG est sensiblement meilleur que le paramètre de l'élan de l'apprentissage classique.
— La fonction de l'initialisation aléatoire des poids des connexions des neurones.
— La profondeur du réseau.

7.5.2 Classification

Dans cette partie, nous avons fait varier le taux d'apprentissage, l'élan de l'apprentissage, changé la fonction d'initialisation aléatoire des poids et proposé une nouvelle architecture convolutive illustrée dans la figure 7.22. Nous rapportons également, les résultats de l'entraînement avec nos données de l'architecture proposée par [Krizhevsky 2012].

Les deux architectures utilisent la fonction d'activation *ReLU* (*Rectified Linear Unit*) où la sortie de chaque neurone est calculée avec la fonction $f(x) = \max(0, x)$. En comparaison avec la fonction $f(x) = tanh(x)$ par exemple, l'utilisation de ReLU ne nous confronte pas au problème de la perte du signal suite à l'application de convolutions successives. De plus, et comme a démontré [Krizhevsky 2012], ReLU est 6 fois plus rapide en temps d'entraînement que les autres fonctions d'activations présentées dans la sous-section 7.2.1, ceci est essentiellement dû à la suppression de la normalisation ainsi que l'absence de tout calcul exponentiel.

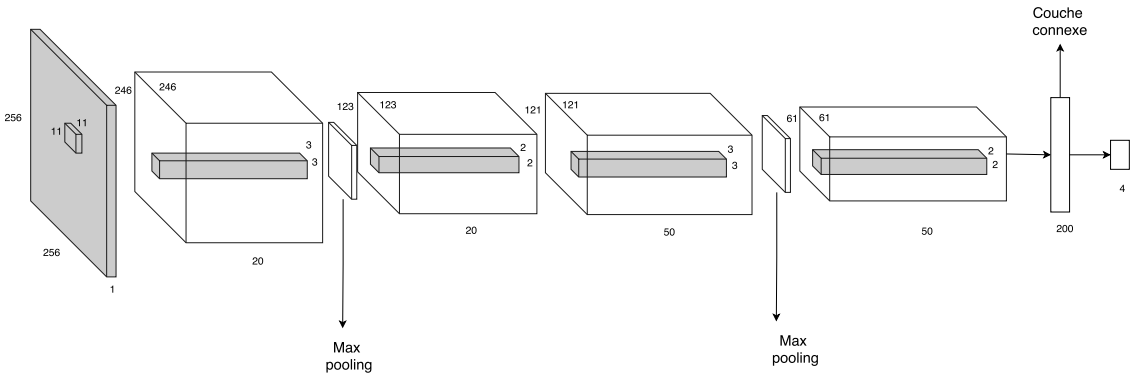


FIGURE 7.22 – Architecture d'un réseau convolutif avec des couches de *max pooling*

Dans la figure 7.22, l'architecture du réseau est constituée de quatre couches

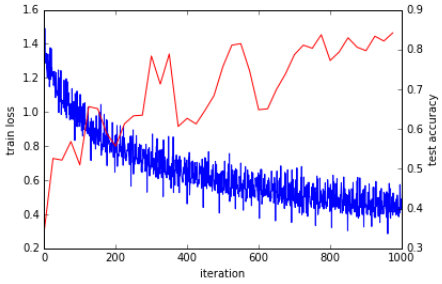


FIGURE 7.23 – illustration de la sortie de la fonction de perte lors de l'entraînement et de la justesse des résultats calculés sur les données de test

de convolution, 2 couches de *maxpooling* et deux couches de neurones connexes [10]. La sortie de la première couche connexe composée de 200 neurones est donnée en entrée à la deuxième couche connexe. Une fonction *softmax* calcule alors un ordre de probabilités afin de classifier le signal sur lequel nous avons appliqué une convolution avec l'un des quatre labels correspondants à l'Éthiopien, le Syriaque, le Grec ou le Latin.

La première couche convolutive filtre l'image en entrée de $256 \times 256 \times 1$ avec 20 masques d'une taille de $11 \times 11 \times 1$ et une foulée d'un pixel. La couche convolutive suivante prend en entrée la sortie de la première couche et y applique un filtre avec 50 masques d'une taille de $3 \times 3 \times 20$. La deuxième et quatrième couche, elles correspondent à des couches de *maxpooling*. La deuxième couche exécute 20 masques d'une taille de $2 \times 2 \times 20$ et une foulée de 2 pixels. La quatrième couche exécute 50 masques d'une taille de $2 \times 2 \times 50$ et une foulée de 2 pixels.

Notons que nous avons également essayé d'autres configurations avec par exemple plus de couches convolutives, un maxpooling à la sortie de chaque convolution ou aussi un seul maxpooling. À la fin de toutes les convolutions. Nous n'avons pas eu de meilleurs résultats par rapport à l'architecture présentée au-dessus.

	Taux ε	**Élan μ**	**weight decay**	**justesse**
Configuration # 1	10^{-4}	0.7	4×10^{-4}	**85%**
Configuration # 2	10^{-4}	0.6	4×10^{-4}	76%
Configuration # 3	10^{-5}	0.7	4×10^{-4}	43%

TABLE 7.6 – Les différentes initialisations de paramètres, pour la même architecture.

Nous rapportons dans le tableau 7.6 trois différentes configurations de l'architecture illustrée dans la figure 7.22. La configuration avec un taux de 85% de justesse est composée d'un taux d'apprentissage $\varepsilon = 10^{-4}$, un élan $\mu = 0.7$ et un *weight decay* de 4×10^{-4}. Ces résultats sont comparables en performance à ceux rapportés dans le référentiel empirique (voir tableau 7.2) et plus précisément les 89% de justesse obtenue par le réseau de neurones (ANN) entraîné avec des caractéristiques HOG.

10. Dans une couche connexe, tous les neurones sont connectés entre eux au sein d'une même couche et avec ceux de la couche suivante.

Dans la figure 7.23, nous avons illustré deux courbes. La première en rouge (ascendante) est celle du taux de justesse à travers les itérations. La deuxième en bleu (descendante) est la valeur de la fonction objectif que nous souhaitons minimiser, appelée aussi perte (en anglais *train loss*). La plus petite valeur atteinte est de 0.27, la valeur moyenne de perte est de 0.41. Nous avons atteint une justesse de prédiction sur les données de test d'une valeur de 85% et ce après 1000 itérations. Quant au comportement "bruité" de la courbe de la justesse sur les données de test, il est expliqué par l'utilisation du *weight decay* pour éviter le sur apprentissage.

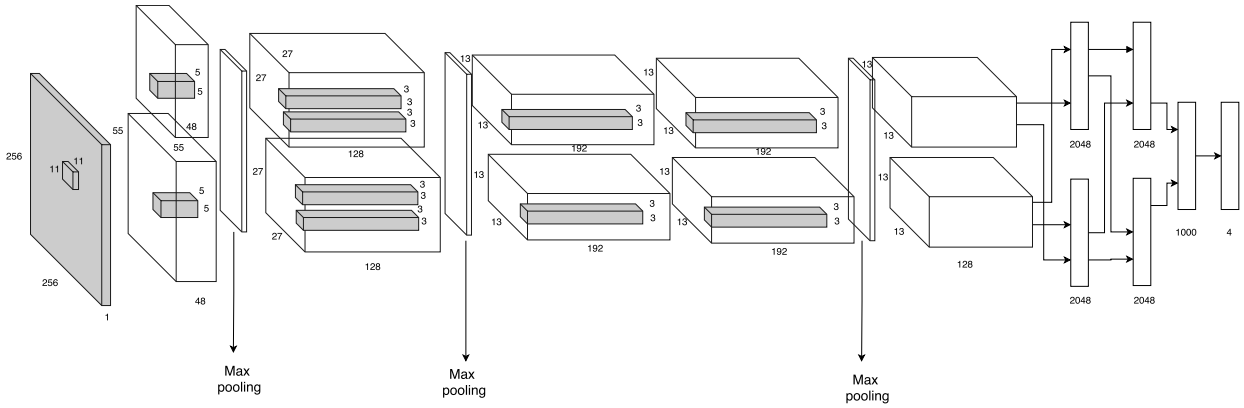


FIGURE 7.24 – Architecture d'un réseau convolutif avec des couches de *max pooling*

Dans la figure 7.24, l'architecture du réseau de [Krizhevsky 2012] appelé Alexnet est constituée de cinq couches de convolution, trois d'entre elles sont suivies par des couches de *maxpooling*, à la fin des trois couches de neurones connexes nous avons connecté une quatrième avec quatre neurones dont la distribution est calculée avec un *softmax*.

La première couche convolutive filtre l'image en entrée de $224 \times 224 \times 1$ avec 96 masques d'une taille de $11 \times 11 \times 1$ et une foulée de 4 pixels. La seconde couche convolutive prend la sortie de la couche de *maxpooling* et filtre cette dernière avec 256 masques d'une taille de $5 \times 5 \times 48$. La troisième couche prend la sortie de la deuxième couche après normalisation et application d'un *maxpooling* sur les neurones, elle y applique 384 masques d'une taille de $1 \times 1 \times 256$. La quatrième applique sur la sortie de la troisième 384 masques d'une taille de $1 \times 1 \times 192$. Quant à la cinquième, elle possède 256 masques avec une taille de $1 \times 1 \times 192$. Les couches connexes possèdent chacune 4096 neurones.

Alexnet a été originellement construit pour la classification de 1.2 million d'images en 1000 catégories. Le réseau a été entraîné sur les trois canaux RGB (i.e. sur des images en couleur). Le signal en entrée est tridimensionnel. Comme nous l'avons illustré dans la figure 7.1, nous travaillons avec des images au niveau du gris. Nous n'avons donc pas pu utiliser un réseau déjà entraîné et avons dû en entraîner un nouveau en précisant qu'il ne devait prendre en considération qu'un seul canal, celui du gris.

La fonction d'activation non linéaire ReLU est utilisée après chaque couche de convolution ou couche connexe. Pour la régularisation de l'apprentissage, le *Dropout* est utilisé avec une valeur de 0.5 et ce dans les deux couches connexes . Nous n'avons pas appliqué de *Dropout* à la couche des quatre neurones connexes que nous avons ajoutés à la fin de l'architecture. Nous avons gardé un taux d'apprentissage $\varepsilon = 10^{-4}$ et un élan $\mu = 0.7$. Nous avons obtenu avec cette configuration un taux de justesse égale à 82%.

7.5.3 Analyse des résultats

Nous constatons que les deux architectures convolutives ont obtenu de meilleurs résultats que les auto-encodeurs. Nous pensons que ceci est principalement dû aux caractéristiques extraites. Face à des données réduites comportant une quantité non négligeable de bruits (voir figure 7.2), il semblerait d'après nos résultats empiriques qu'une architecture convolutive simple soit plus appropriée qu'une architecture de type auto-encodeurs.

Les résultats empiriques que nous avons obtenus nous laissent penser que les auto-encodeurs nécessiteraient plus de données afin de construire une meilleure représentation de toutes les caractéristiques que nous pouvons trouver dans un manuscrit ancien. Le compromis entre le choix du modèle d'apprentissage et le rajout de plus de données lors de l'entraînement de ce dernier, a été traité dans les travaux de [Zhu 2012].

Notons également que l'architecture que nous avons proposée, et qui est illustrée dans la figure 7.22, a obtenu en moyenne un résultat légèrement meilleur (une différence de 3% dans le taux de justesse) que celui d'Alexnet. Cependant, nous pensons que l'architecture Alexnet est capable de nous fournir un meilleur résultat si nous avons eu la possibilité de récupérer une version entraînée sur plusieurs images de documents au niveau du gris. En effet, en appliquant un apprentissage par transfert plusieurs travaux [Shin 2016] rapportent de meilleurs résultats par rapport à un apprentissage à partir de zéro.

Les résultats obtenus démontrent également l'importance des différents paramètres que nous avons utilisés lors de l'apprentissage. Comme nous avons observé dans le tableau 7.6, un changement sensible dans la valeur de certains paramètres peut résulter dans une chute importante dans le taux de justesse. Par exemple, passant de 85% à 43% lors d'une modification du taux d'apprentissage ε. Ou encore de 85% à 76% lors d'une modification de l'élan μ.

Notons finalement l'importance du schéma de l'initialisation aléatoire des poids des connexions neuronales. Dans la partie des SCAE, nous avons utilisé une fonction d'initialisation que nous avons tirée d'un ensemble de tests empiriques. Dans cette partie, nous avons utilisé une fonction d'initialisation avec une distribution gaussienne où nous avons évité de perdre le signal en réduisant la variance. Il semblerait que cette dernière soit plus efficace toujours d'après nos résultats empiriques.

Dans cette partie, nous nous sommes intéressés au problème de l'identification des scripts dans des manuscrits comportant l'Éthiopien, le Syriaque, le Grec et le Latin. Nous avons approché le problème en menant une étude empirique où nous avons adopté une analyse en termes des pixels au niveau d'une page entière issue d'un manuscrit donné. Cette analyse est basée sur l'apprentissage d'un ensemble de caractéristiques par l'application d'une suite de transformations non linéaires sur les pixels de la page donnée. Cette suite de transformations, appelée convolutions, est appliquée sur ce signal (i.e. ensemble de pixels) à travers une couche de neurones en entrée, une ou plusieurs couches cachées et une couche de sortie. Ces couches forment ce que l'on appelle une architecture profonde.

Deux catégories d'architectures ont été étudiées empiriquement dans ce chapitre. La première appelée autoencodeur, nous fournit des caractéristiques qui sont issues de la représentation non supervisée qu'elle se fait du signal. La deuxième est une architecture convolutive avec une représentation supervisée. Nous avons mené une étude empirique afin de jauger la capacité d'une architecture par rapport à une autre à se construire une bonne représentation à partir d'une quantité de données limitée et en mauvais état.

Les résultats obtenus sont les suivants : utilisant des filtres construits avec des algorithmes conçus par des experts, nous avons obtenu sur notre jeu de données un taux de justesse de 89.07%. Puis, en construisant un filtre de manière non supervisée, nous avons obtenu des résultats modestes avec un taux de justesse de 71.8%. Nous avons remédié à ces résultats par l'optimisation d'une liste de paramètres et en changeant la fonction d'initialisation des poids des différentes connexions neuronales dans les architectures étudiées. Nous avons obtenu des résultats plus proche de ceux du filtre choisi par l'expert, avec un taux de justesse de 85%.

Nos conclusions sur ce chapitre sont les suivantes :

— Le compromis entre la quantité de données utilisée lors de l'entraînement et l'utilisation de meilleurs modèles d'apprentissage doit être considéré avec attention.

— Le sous-apprentissage est une problématique aussi importante que le sur apprentissage et celle-ci n'est pas toujours résolue par l'extension du jeu de données.

— Comme perspectives, nous comptons implémenter la fonction ReLU dans les auto-encodeurs et permettre une rétropropagation sur les couches comportant un extremum pooler afin de permettre l'utilisation du *finetuning*.