Apprentissage supervisé d'une représentation multi-couches à base de dictionnaires pour la classification d'images et de vidéos

Stefen Chan Wai Tim

▶ **To cite this version:**

Stefen Chan Wai Tim. Apprentissage supervisé d'une représentation multi-couches à base de dictionnaires pour la classification d'images et de vidéos. Traitement du signal et de l'image [eess.SP]. Université Grenoble Alpes, 2016. Français. NNT : 2016GREAT089 . tel-01525825

HAL Id: tel-01525825
https://theses.hal.science/tel-01525825

Submitted on 22 May 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Communauté
UNIVERSITÉ Grenoble Alpes

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité : **Signal, Images, Paroles, Télécom**

Arrêté ministériel : 7 août 2006

Présentée par

Stefen CHAN WAI TIM

Thèse dirigée par **Michèle ROMBAUT**
et codirigée par **Denis PELLERIN**

préparée au sein **du Gipsa-Lab**
et de **l'école doctorale Electronique, Electrotechnique, Automatique et Traitement du signal (EEATS)**

Apprentissage supervisé d'une représentation multicouche à base de dictionnaires pour la classification d'images et de vidéos

Thèse soutenue publiquement le **17 novembre 2016**,
devant le jury composé de :

M, Atilla BASKURT
Professeur, Université de Lyon, Président
M, Stéphane CANU
Professeur, INSA Rouen, Rapporteur
M, Nicolas THOME
Maître de conférences HDR, Université Paris VI, Rapporteur
Mme, Michèle ROMBAUT
Professeur, Université Grenoble Alpes, Directeur de thèse
M, Denis PELLERIN
Professeur, Université Grenoble Alpes, Co-Directeur de thèse

Résumé

Dans cette thèse, nous nous intéressons à la classification d'images et de vidéos à l'aide de caractéristiques obtenues par décomposition sur un ensemble multicouche de dictionnaires.
Ces dernières années, de nombreux travaux ont été publiés sur l'encodage parcimonieux et l'apprentissage de dictionnaires. Leur utilisation s'est initialement développée dans des applications de reconstruction et de restauration d'images. Plus récemment, des recherches ont été réalisées sur l'utilisation des dictionnaires pour des tâches de classification en raison de la capacité de ces méthodes à chercher des motifs sous-jacents dans les images et de bons résultats ont été obtenus dans certaines conditions : objet d'intérêt centré, de même taille, même point de vue. Cependant, hors de ce cadre restrictif, les résultats sont plus mitigés.

Dans cette thèse, nous avons proposé une méthode d'apprentissage supervisée des dictionnaires afin de les adapter à la tâche de classification. En effet, les méthodes d'apprentissage classiquement utilisées pour les dictionnaires s'appuient sur des algorithmes d'apprentissage non supervisé.
La structure multicouche a pour objectif d'améliorer le caractère discriminant des codes obtenus. L'architecture proposée s'appuie sur la description locale d'une image en entrée et sa transformation grâce à une succession d'encodage et de traitements. Elle fournit ensuite en sortie un ensemble de descripteurs adaptés à la classification.
La méthode d'apprentissage que nous avons développée est basée sur l'algorithme de rétropropagation du gradient permettant un apprentissage coordonné des différents dictionnaires et une optimisation uniquement par rapport à un coût de classification.

L'architecture proposée a été testée sur les bases de données d'images MNIST et CIFAR-10 avec de bons résultats par rapport aux autres méthodes basées sur l'utilisation de dictionnaires. Dans le cadre du transfert learning, nous avons aussi utilisé cette architecture apprise sur la base CIFAR-10 pour effectuer la tâche de classification sur la base STL-10. Enfin, le cadre proposé a été étendu à l'analyse de vidéos.

Mots-clés : apprentissage de dictionnaires, représentation multicouche, apprentissage supervisé, classification d'images, classification de vidéos

Remerciements

Je souhaite tout d'abord remercier mes directeurs de thèse, Michèle et Denis, qui m'ont encadré pendant ces 3 années de thèse. Merci de m'avoir donné la chance d'effectuer cette aventure, de m'avoir guidé et aidé à acquérir les compétences nécessaires au travail de recherche. Si cette thèse s'est bien passée, c'est aussi grâce à vous. Merci beaucoup.

Je remercie également les membres du jury : Atilla BASKURT, Nicolas THOME et Stéphane CANU pour avoir accepté d'évaluer mes travaux et pour leurs remarques.

Je remercie mes compagnons de thèse, Laetitia, Alexandre, Thibault, Cao (pionnier) et les collègues du laboratoire, Dana, Quentin, Lucas, Alexis, Alexander, Alexandre, Marc, Jocelyn, Lyuba, Victor.. pour leur soutien, les activités et les discussions que nous avons eu. Thibault spécialement, je ne compte plus le nombre de fois où une idée m'est venue pendant une de nos discussions. Bon courage à ceux qui vont bientôt finir.

Je souhaite particulièrement remercier Anuvabh pour son aide sur les architectures de réseaux de neurones. Ton expertise et tes expériences ont contribué à l'écriture de ce manuscrit. Tu as toujours répondu efficacement et avec diligence. Merci.

Merci aussi à l'équipe enseignante d'informatique à laquelle j'ai pu participer pendant deux ans : ce fut une expérience très instructive que je ne peux que recommander.

Un merci aux amis de Grenoble pour les moments partagés, les sorties, les jeux et la bonne humeur.

Pour finir, un grand merci à mon frère et ma famille qui m'a poussé à donner le meilleur de moi-même et qui m'a soutenu durant ces 3 années (avec une mention pour l'organisation du buffet).

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction générale

Sommaire

1.1 Introduction

Ces dernières années, la taille cumulée de l'ensemble des objets multimédias a explosé. En même temps, l'importance des méthodes d'apprentissage automatique pour gérer et classifier ces bases de données de plus en plus importantes s'est accrue : moteurs de recherche basés sur le contenu (i.e Tineye [1] un moteur de recherche utilisant une image en entrée) ou annotation d'archives (i.e INA [2] qui a le besoin d'indexer ses immenses bases de vidéos). En particulier, la classification est une problématique liée à un grand nombre d'applications telles la reconnaissance de personnes ou d'objets, l'annotation automatique d'images, la reconnaissance d'émotions, la reconnaissance de paroles, la détection de défauts industriels, etc.

Dans cette thèse, nous nous intéressons à la classification d'images et de vidéos. Les objets ou concepts à reconnaitre sont souvent d'un haut niveau sémantique ce qui oblige à rechercher des descripteurs adaptés. Par exemple, la tâche peut aller de la reconnaissance d'un animal (chien, chat, oiseau..) jusqu'à des concepts plus abstraits : pour la compétition TRECVid [3], certaines actions à localiser étaient "danser", "jouer d'un instrument" ou encore "demande en mariage". Par ailleurs, la complexité des tâches de classification augmente rapidement pour ce qui est de la taille des bases d'apprentissage mais également du nombre de classes (ILSVRC - "ImageNet Large Scale Visual Recognition Competition" [Rus+15]).

Aux débuts de la classification d'images, les premiers descripteurs étaient des descripteurs bas-niveaux (couleurs, textures) et ceux-ci ont été insuffisants pour représenter de manière efficace le contenu sémantique d'une image. La communauté s'est ensuite intéressée à des descripteurs locaux plus complexes (SIFT [Low04], SURF [BTG06], HOG [DT05]) fournissant

1. www.tineye.com
2. Institut National de l'Audiovisuel
3. http ://trecvid.nist.gov/

une nouvelle représentation du contenu de l'image. Ces descripteurs ont permis des progrès en classification mais sont restés des "fabrications" manuelles, élaborées pour mesurer des critères précis motivés par des connaissances conceptuelles a priori.
L'étape suivante a été l'apparition des descripteurs adaptés aux données obtenus par apprentissage : par exemple par l'intermédiaire des réseaux de neurones [Hay98] ou plus récemment avec l'apparition de nouveaux réseaux de neurones dits convolutifs [KH09]. Ces dernières méthodes permettent d'extraire de manière automatique des descripteurs plus adaptés pour la tâche de classification mais nécessite de grandes bases d'apprentissage.
Une autre approche permettant d'apprendre des descripteurs s'est illustrée, il s'agit de l'apprentissage de dictionnaires [OF96] qui se base sur la recherche d'une représentation parcimonieuse des images.
La construction des bases d'apprentissage de taille suffisante pose un réel problème, même si quelques dispositifs permettent d'apporter une aide à la personne chargée de classer les images [CRP12].

1.2 Contributions

Dans cette thèse, nous nous sommes intéressés à la classification d'images et de vidéos à l'aide des descripteurs obtenus à partir d'une décomposition parcimonieuse utilisant des dictionnaires.

Dans un premier temps, à la suite des travaux de Mairal et al. [MBP12], nous avons implémenté une méthode d'apprentissage supervisé en introduisant un critère de classification dans l'apprentissage des dictionnaires.

Nous avons ensuite proposé de décomposer l'image en un ensemble de patchs ce qui permet une description locale de l'image en plus de conserver sa structure spatiale. L'apprentissage du dictionnaire se fait donc au niveau des patchs d'images.

Après codage des patchs par l'intermédiaire d'un dictionnaire, on obtient donc une représentation de l'image sous la forme de codes parcimonieux. Ces codes peuvent être traités ("pooling", non-linéarité) pour réduire leur dimension et dans l'objectif d'améliorer les performances.

Afin d'obtenir des descripteurs plus pertinents pour la classification, nous avons proposé de regrouper puis d'encoder de nouveau les codes obtenus pour fournir une nouvelle représentation. Cet enchainement d'encodages et de traitements forme notre architecture multicouche.

Pour apprendre les différents dictionnaires de l'architecture, nous avons proposé une méthode basée sur l'algorithme de rétropropagation du gradient afin d'optimiser ceux-ci par rapport à la tâche de classification. Cette méthode nécessite plusieurs étapes par couche de la structure.

Plusieurs architectures sont évaluées sur des bases d'images publiques de l'état de l'art MNIST, CIFAR-10 et STL-10 pour estimer les performances du système proposé. Une analyse

de l'apport de la supervision pour la formulation proposée est fournie.

Nous étudions ensuite la possibilité d'appliquer notre architecture au problème de la reconnaissance d'actions dans les vidéos. Dans cet objectif, les dictionnaires sont modifiés pour représenter du mouvement. De plus, l'architecture est couplée avec un détecteur de personnes pour localiser des régions d'intérêt sur lesquelles l'algorithme sera appliqué.

1.3 Organisation du manuscrit

Le manuscrit se décompose de la façon suivante :

La chapitre 2 définit le concept de dictionnaire et donne les notions de base sur les représentations parcimonieuses. Ce chapitre résume aussi les méthodes dites classiques d'apprentissage de dictionnaires et pose les bases de la problématique de classification.

Le chapitre 3 décrit une méthode d'apprentissage supervisé pour dictionnaires reposant sur l'algorithme de rétro-propagation ce qui permet un apprentissage uniquement basé sur un critère discriminant. L'application considérée est la classification d'images. La méthode, décrite point par point, est le fondement de la contribution de ce manuscrit. Les limites de cette méthode sont abordées en vue de proposer une amélioration.

Le chapitre 4 constitue le coeur du manuscrit et présente une proposition d'architecture multicouche de dictionnaires en vue d'effectuer une tâche de classification. La méthode décrite au chapitre 3 est étendue pour supporter plusieurs couches de dictionnaires. Un algorithme d'apprentissage est proposé.

Le chapitre 5 présente des expériences qui visent à estimer les performances de l'architecture proposée sur plusieurs bases d'images publiques de la littérature. Les performances sont calculées avec et sans l'application de la supervision et nous montrons que la méthode proposée fournit de très bons résultats.

Le chapitre 6 donne une application à la reconnaissance d'actions dans les vidéos et apporte une discussion sur l'adaptation de la méthode proposée à ce contexte.

Chapitre 2

Apprentissage d'un dictionnaire

Sommaire

2.1 Introduction aux dictionnaires

Ces dernières années, il y a eu un intérêt accru dans la recherche de représentations parcimonieuses à partir de dictionnaires [RBE09], [RPE13], [SBJ13], [MBP14], [Bao+16]. En effet, il est intéressant de pouvoir concentrer toute l'information d'un signal dans quelques coefficients seulement, par exemple pour des applications de compression ou de classification.

On peut retrouver ce genre de décompositions en utilisant, par exemple, une décomposition en série de Fourier pour une fonction périodique, ou une décomposition en ondelettes [Mal09]. En pratique, on peut interpréter la recherche de cette décomposition comme un changement vers un nouvel espace de représentation dans lequel un signal s'exprime avec peu de variables (Fig. 2.1).

Dans l'approche qui nous intéresse, les coefficients sont obtenus en décomposant un signal sur une base de signaux élémentaires, que l'on appelle "atomes" ou "prototypes". Ceux-ci sont de même nature que le signal à décomposer et on appelle dictionnaire un ensemble de tels atomes.

La figure 2.2 donne un exemple d'une reconstruction parcimonieuse dans le cas d'un patch d'image de (16×16) pixels. Le patch représenté est un morceau du chapeau de *Lena* et les

reconstructions à 1 atome, 5 atomes et 10 atomes sont données (Fig. 2.3). Pour 10 atomes, on peut visuellement constater qu'on arrive à obtenir une version lissée du patch original. En augmentant davantage le nombre d'atomes utilisés, on peut reconstruire encore plus fidèlement le patch initial.

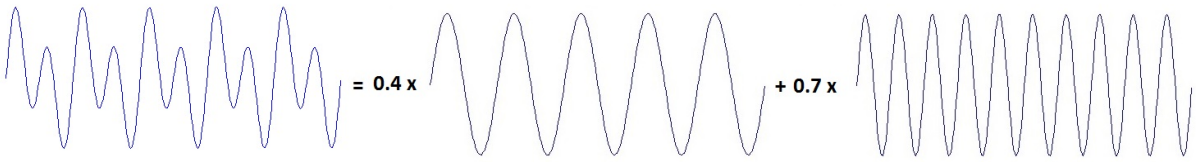


FIGURE 2.1 – Le signal de gauche peut se représenter de différentes manières : dans sa représentation temporelle, on doit conserver un ensemble de valeurs sur tout l'intervalle de temps (ou au moins une période), tandis que dans sa représentation fréquentielle on conserve uniquement les deux fréquences et leurs coefficients associés. La représentation est donc parcimonieuse dans l'espace fréquentiel.

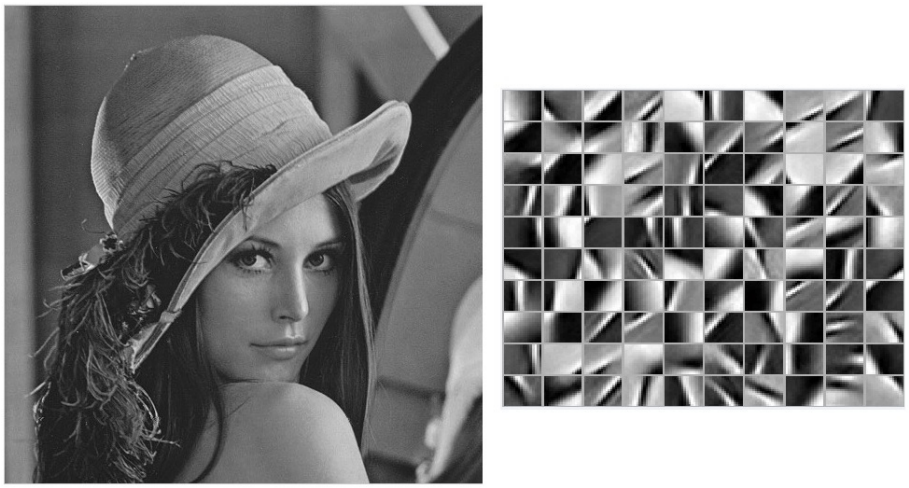


FIGURE 2.2 – Image *Lena* ainsi qu'un exemple de dictionnaire de patchs de taille (16×16) pixels.

Considérons un signal représenté sous la forme d'un vecteur $\mathbf{y} \in \mathcal{R}^n$. Dans la suite du manuscrit, nous nous intéresserons principalement à des images ou des parties d'images (des patchs), un exemple de vecteur $\mathbf{y}$ peut donc être obtenu en vectorisant un patch d'image de taille $(s \times s) = n$. Soit un dictionnaire $\mathbf{D}$ de K atomes $\mathbf{D} = (\mathbf{d}_j)_{j,\in[1,K]} \in \mathcal{R}^{(n \times K)}$. Idéalement, on souhaite pouvoir écrire :

$$\mathbf{y} = \sum_{j=1}^{K} x_j \mathbf{d}_j \tag{2.1}$$

où $\mathbf{x} = (x_j)_{j,\in[1,K]}$ est le vecteur de coefficients qui décrit le signal $\mathbf{y}$ par le dictionnaire $\mathbf{D}$.

Lorsque l'on écrit une telle décomposition, on représente $\mathbf{y}$ par une combinaison linéaire

(a) (b)

FIGURE 2.3 – (a) Patch initial tiré de l'image *Lena*. (b) Reconstruction à l'aide d'un dictionnaire en utilisant respectivement 1 atome, 5 atomes et 10 atomes.

d'atomes du dictionnaire $\mathbf{D}$. La représentation est dite parcimonieuse si une grande proportion des valeurs x_j du vecteur $\mathbf{x}$ est nulle.

Le dictionnaire $\mathbf{D}$ peut être défini *a priori* en vérifiant certaines propriétés, comme par exemple, la capacité à pouvoir représenter n'importe quel signal avec des performances équivalentes. Les dictionnaires dont nous allons discuter sont, au contraire, *appris* pour s'adapter à des ensembles de signaux, ce qui peut se traduire pour les signaux concernés par une parcimonie plus forte quand ils sont représentés par un tel dictionnaire.

2.2 Apprentissage sur un ensemble de signaux

Dans cette section, nous nous intéressons au problème de l'apprentissage d'un dictionnaire : nous disposons d'un ensemble de données $\{\mathbf{y}_k\}_{k\in[1,m]}$ et nous souhaitons trouver un dictionnaire $\mathbf{D}$ qui soit capable de représenter chacun des éléments $\mathbf{y}_k$ de cet ensemble de manière parcimonieuse.

2.2.1 Présentation

L'utilisation de dictionnaires appris sur un ensemble de données s'est popularisé avec les travaux de Olshausen et Field [OF96]. Leur problématique d'étude était de trouver une base de représentation pour les images naturelles possédant des caractéristiques proches du cortex visuel des mammifères. L'objectif était d'apprendre cette représentation en cherchant à maximiser la parcimonie (Fig. 2.4).

La représentation d'un signal $\mathbf{y}$ par une somme pondérée d'atomes $\mathbf{d}_j$ (Eq. 2.1) peut être réécrite sous forme matricielle.

$$\mathbf{y} = \sum_{j=1}^{K} x_j \mathbf{d}_j = \mathbf{D}\mathbf{x}$$

En pratique, on s'autorise une erreur $\mathbf{e}$ dans la reconstruction du signal $\mathbf{y}$:

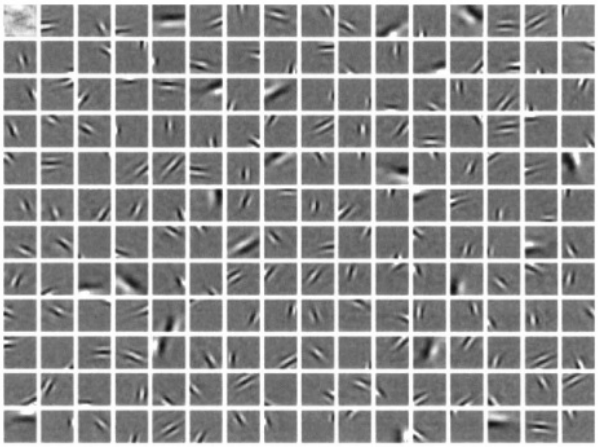


FIGURE 2.4 – Dictionnaire de patchs obtenu dans l'article d'Olshausen et Field [OF96] à partir d'images naturelles. Les atomes ont une taille de (16×16) pixels.

$$\mathbf{y} = \sum_{j=1}^{K} x_j \mathbf{d}_j + \mathbf{e} \approx \mathbf{D}\mathbf{x}$$

L'objectif est d'apprendre un dictionnaire $\mathbf{D}$ de telle sorte qu'on puisse, en moyenne, trouver pour chaque signal d'apprentissage un code $\mathbf{x}$ parcimonieux assurant une bonne reconstruction du signal. Une manière simple pour chercher une solution du problème d'apprentissage est de minimiser une erreur quadratique en utilisant par exemple un algorithme de descente de gradient.

Soit un ensemble de signaux d'apprentissage $\mathbf{y}_k$ pour $k \in [1, m]$, on cherche :

$$\min_{\mathbf{D},\mathbf{x}_k} \quad \frac{1}{m} \sum_{k=1}^{m} \|\mathbf{y}_k - \mathbf{D}\mathbf{x}_k\|_2^2 \tag{2.2}$$

Le dictionnaire obtenu par un tel apprentissage est dépendant de l'ensemble de données utilisé au sens où les structures apprises par le dictionnaire sont celles présentes dans l'ensemble d'apprentissage. Considérons par exemple deux signaux, un signal proche des signaux de l'ensemble d'apprentissage et le second issu d'un groupe différent de signaux. En pratique, sauf cas particuliers, les deux signaux peuvent être bien reconstruits par le dictionnaire appris sur le premier ensemble de signaux. Cependant, pour un nombre de coefficients non-nuls fixé, l'erreur de reconstruction est en moyenne plus faible pour le signal proche des signaux d'apprentissage. C'est pour cela qu'on dit que le dictionnaire est adapté aux signaux d'apprentissage.

On ajoute également une contrainte aux atomes du dictionnaire $\mathbf{D}$, en particulier $\|\mathbf{d}_j\|_2 \leq 1$ pour contrôler que les coefficients du code $\mathbf{x}_k$ ne deviennent pas arbitrairement petits alors

que la norme des atomes $\mathbf{d}_j$ devient arbitrairement grande.

On peut noter que la taille du dictionnaire (le nombre d'atomes K) n'est pas limité. En pratique, on choisit souvent un nombre d'atomes K pour le dictionnaire supérieur ou égal à la dimension des atomes ($K \geq n$), le dictionnaire est alors dit "*overcomplete*". On obtient un dictionnaire dont certains atomes sont corrélés. De manière générale, plus le nombre d'atomes K augmente, plus les atomes du dictionnaire sont redondants. Cela entraine que beaucoup d'entre eux peuvent ne pas être utilisés, ce qui conduit à des représentations encore plus parcimonieuses. Empiriquement, plus le nombre d'atomes augmente, plus on arrivera à diminuer l'erreur de reconstruction à nombre de coefficients non-nuls égal : si on disposait d'une infinité d'atomes, capables de représenter tous les motifs, chaque signal pourrait être représenté de manière parcimonieuse, le cas extrême étant un unique atome égal au signal lui-même. Cependant, un tel dictionnaire est difficile à stocker et aussi difficile à manipuler pour la recherche des coefficients. Il faut aussi tenir compte du coût en calculs pour la recherche du code $\mathbf{x}$ et la mise à jour du dictionnaire qui augmente en fonction du nombre d'atomes K du dictionnaire. De plus, ce qui nous intéresse principalement est l'extraction de caractéristiques communes pour un groupe de signaux : c'est pourquoi en pratique, le nombre d'atomes d'un dictionnaire est toujours petit devant le nombre de signaux de la base d'apprentissage.

2.2.2 Parcimonie

Dans la section précédente, nous avons vu l'apprentissage d'un dictionnaire formulé comme un problème de minimisation (Eq. 2.2). Pour obtenir la parcimonie des vecteurs $\mathbf{x}_k$ représentatifs des signaux $\mathbf{y}_k$ d'apprentissage, on ajoute une contrainte supplémentaire qui prend la forme d'un terme de régularisation.

Le problème devient alors :

$$\min_{\mathbf{D},\mathbf{x}_k} \quad \frac{1}{m}\sum_{k=1}^{m} \|\mathbf{y}_k - \mathbf{D}\mathbf{x}_k\|_2^2 + \phi(\mathbf{x}) \tag{2.3}$$

où ϕ est une fonction de régularisation apportant la parcimonie.

Il existe différentes manières d'introduire la parcimonie à partir du terme de régularisation ϕ de l'équation 2.3.

La manière la plus directe est d'utiliser la pseudo-norme ℓ_0 définie comme suit : $\phi(\mathbf{x}) = \|\mathbf{x}\|_0 = card(\{i|x_i \neq 0\})$ c'est-à-dire que sa valeur correspond au nombre d'éléments non-nuls dans le vecteur $\mathbf{x}$. La pseudo-norme ℓ_0 permet un contrôle strict du nombre de coefficients non-nuls de la décomposition ce qui la rend intéressante dans les applications de décomposition parcimonieuse. Par exemple, cette régularisation a été utilisée avec succès dans l'algorithme K-SVD [AEB06].

Cependant, trouver une solution exacte au problème de décomposition sur un dictionnaire en régularisant par une norme ℓ_0 est difficile (on ne peut pas obtenir une solution

exacte en temps polynomial [BDE09]) et amène par ailleurs certains problèmes comme la non-différentiabilité.

Par conséquent, on simplifie le problème en cherchant une solution approchée. On peut trouver une telle solution de différentes manières : soit en utilisant des algorithmes dits "greedy" tel que l'algorithme "Matching Pursuit" (décrit plus en détails au 2.2.3), ou encore en remplaçant la norme ℓ_0 par une norme ℓ_1. Cette dernière formulation, également appelée "LASSO" (Least absolute selection and shrinkage operator) [Tib96], [Tib12] s'écrit :

$$\min_{\mathbf{D},\mathbf{x}_k} \quad \frac{1}{m}\sum_{k=1}^{m} \|\mathbf{y}_k - \mathbf{D}\mathbf{x}_k\|_2^2 + \lambda \sum_{k=1}^{m} \|\mathbf{x}_k\|_1 \tag{2.4}$$

Cela signifie qu'on cherche à avoir les coefficients de $\mathbf{x}$ les plus petits possibles, en espérant que certains soient nuls. De ce fait, le nombre de coefficients non-nuls est plus élevé qu'avec ℓ_0 (parcimonie plus faible).

La norme ℓ_1 est également non-différentiable en 0 mais il existe de nombreuses méthodes pour résoudre le problème d'optimisation donné par l'équation 2.4 [SFR09].

Mais avant d'aller plus loin, nous allons donner quelques intuitions sur l'utilisation de cette norme dans le contexte de la parcimonie.

Considérons le problème contraint :

$$\begin{aligned} \min_{\mathbf{x}} \quad & \|\mathbf{y} - \mathbf{D}\mathbf{x}\|_2^2 \\ \text{s.t.} \quad & \|\mathbf{x}\|_1 < T \end{aligned} \tag{2.5}$$

où T est un nombre réel positif.

Géométriquement, chercher un minimum à l'équation 2.5 correspond à chercher la tangente entre le contour de la fonction de coût quadratique et la contrainte (voir Fig. 2.5). Intuitivement, on observe que la forme de la contrainte liée à une norme ℓ_1 facilite le positionnement de la tangente sur un des axes de coordonnées, ce qui correspond à une solution parcimonieuse puisque la composante sur les autres axes est nulle.

Par ailleurs, il existe une "équivalence" [Sch05] entre la formulation contrainte (Eq. 2.5) et non-contrainte (Eq. 2.4) (avec T inversement proportionnel à λ) du problème qui généralise donc l'intuition de parcimonie liée à la norme ℓ_1.

Ce modèle a été largement étudié [Tib96], [Tib12] et a l'avantage d'être plus simple à optimiser que la formulation utilisant la norme ℓ_0 au sens où il est "facile" d'obtenir une solution en utilisant des outils appelés "opérateurs proximaux" [PB13].

Il y a bien sûr de nombreuses variantes de fonctions de régularisation qui permettent d'obtenir une décomposition parcimonieuse et qui peuvent être adaptées à des tâches précises,

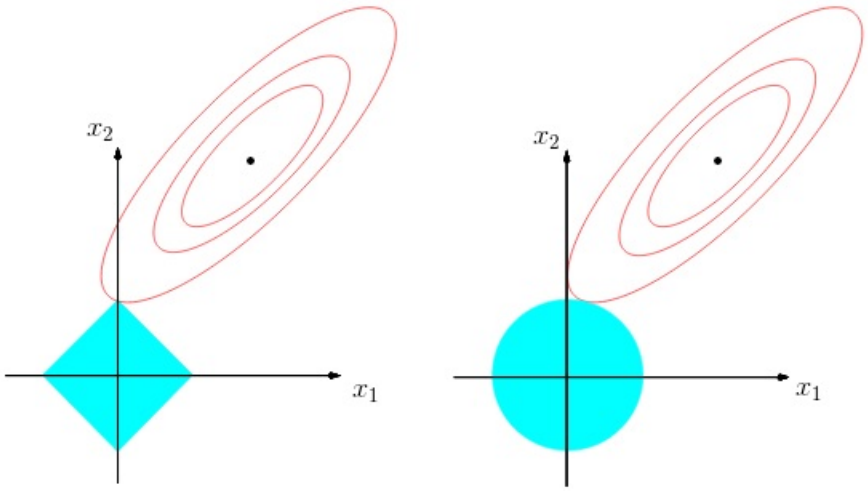


FIGURE 2.5 – À gauche, les contours d'une fonction de coût quadratique (l'ellipse rouge) et d'une contrainte par une norme ℓ_1 (en bleu). À droite, les contours d'une fonction de coût quadratique et d'une contrainte par une norme ℓ_2. La forme des contraintes influence le positionnement de la tangente avec le contour de la fonction de coût. Pour la norme ℓ_1, la tangente a de plus grandes chances d'être localisée sur un des axes ce qui amène des solutions parcimonieuses. (Reproduction extraite de [HTF09])

par exemple : LLC (Locality-constrained Linear Coding) [Wan+10] ou encore "Elastic-net" [ZH05].

Pour l'algorithme LLC, la formulation utilisée est

$$\min_{\mathbf{D},\mathbf{x}} \quad \|\mathbf{y} - \mathbf{D}\mathbf{x}\|_2^2 + \lambda \|l \odot \mathbf{x}\|_1 \tag{2.6}$$

où $\odot$ est la multiplication élément à élément et l est un vecteur contenant les distances entre $\mathbf{y}$ et les atomes de $\mathbf{D}$.

Cette formulation pénalise par exemple le fait d'utiliser des atomes "éloignés" de $\mathbf{y}$ pour la reconstruction en augmentant artificiellement la valeur du coefficient associé (chaque coefficient x_i est pondéré par un terme l_i directement lié à la distance entre $\mathbf{d}_i$ et $\mathbf{y}$).

La régularisation "Elastic-net" combine quant-à-elle linéairement les normes ℓ_1 et ℓ_2. Elle est abordée avec plus de détails au chapitre 4.

2.2.3 Méthodes d'apprentissage

Dans cette section, nous allons présenter différentes manières d'apprendre un dictionnaire $\mathbf{D}$, c'est-à-dire la résolution du problème de minimisation donnée par l'équation 2.3. Les méthodes utilisées peuvent varier en fonction de la fonction de régularisation $\phi(\mathbf{x})$ choisie (par exemple, lorsque l'on utilise une norme ℓ_0).

De manière générale, le problème donnée par l'équation 2.3 n'est pas conjointement convexe lorsque l'on recherche $\mathbf{x}$ et $\mathbf{D}$ simultanément, mais il l'est lorsqu'on les recherche séparément [AEB06]. Une idée simple pour s'en convaincre est qu'il existe plusieurs minima globaux : par exemple, une permutation des colonnes du dictionnaire $\mathbf{D}$. Cependant, on s'aperçoit empiriquement que dans une majorité de cas, chercher un minimum local suffit pour obtenir de bons résultats [AEB06], [Mai+09].

Pour chercher le minimum local pour ce problème, on le décompose habituellement en deux parties distinctes :

— La recherche du code optimal $\hat{\mathbf{x}}_k$ pour chacun des signaux $\mathbf{y}_k$ de la base. Lors de cette étape, on considère que $\mathbf{D}$ est connu et fixé et on cherche à trouver une décomposition parcimonieuse pour chaque signal $\mathbf{y}_k$ individuellement.
On note la solution :

$$\hat{\mathbf{x}}_k = \underset{\mathbf{x}}{\operatorname{argmin}} \quad \|\mathbf{y}_k - \mathbf{D}\mathbf{x}\|_2^2 + \phi(\mathbf{x}) \tag{2.7}$$

On cherche à minimiser l'erreur de reconstruction $\|\mathbf{y}_k - \mathbf{D}\mathbf{x}\|_2^2$ entre $\mathbf{y}_k$ et la décomposition $\mathbf{D}\mathbf{x}$, en ajoutant un terme de régularisation.

— La recherche du meilleur dictionnaire $\mathbf{D}$ pour des codes $\hat{\mathbf{x}}_k$. Dans ce cas, ce sont les coefficients de décomposition pour chaque $\hat{\mathbf{x}}_k$ qui sont connus et on recherche le dictionnaire qui fournit la meilleure reconstruction à partir de ces coefficients.

$$\min_{\mathbf{D}} \quad \frac{1}{m}\sum_{k=1}^{m} \|\mathbf{y}_k - \mathbf{D}\hat{\mathbf{x}}_k\|_2^2 + \sum_{k=1}^{m} \phi(\hat{\mathbf{x}}_k) \tag{2.8}$$

Considérons la fonction de coût :

$$\mathcal{L}(\mathbf{y}, \mathbf{D}, \mathbf{x}) = \|\mathbf{y} - \mathbf{D}\mathbf{x}\|_2^2 + \phi(\mathbf{x}) \tag{2.9}$$

Une des méthodes de résolution les plus simples pour l'apprentissage du dictionnaire consiste à utiliser une descente de gradient stochastique (exemple, [OF96]) en alternant, comme énoncé précédemment, une minimisation par rapport au dictionnaire $\mathbf{D}$ et une recherche du code $\mathbf{x}$. Cette méthode consiste à effectuer des itérations de descente de gradient sur un unique échantillon (ou un petit groupe) de la base d'apprentissage, tiré aléatoirement pour chaque itération.

Les étapes de mise à jour s'écrivent respectivement :

$$\begin{gathered} \mathbf{D} \leftarrow \mathbf{D} - \eta_1 \nabla_{\mathbf{D}} \mathcal{L}(\mathbf{y}_k, \mathbf{D}, \hat{\mathbf{x}}_k) \\ et \\ \mathbf{x}_k \leftarrow \mathbf{x}_k - \eta_2 \nabla_{\mathbf{x}_k} \mathcal{L}(\mathbf{y}_k, \mathbf{D}, \mathbf{x}_k) \end{gathered} \tag{2.10}$$

où η_1 et η_2 sont les pas de descente.

2.2.3.1 Recherche de x

En pratique, la descente de gradient proposée (Eq. 2.10) fonctionne bien lorsque le terme de régularisation est différentiable, ce qui n'est pas toujours le cas (par exemple pour ℓ_0 ou ℓ_1).

Dans le cas de la pseudo-norme ℓ_0, la recherche d'une solution exacte pour le code optimal $\mathbf{x}$ est difficile et couteuse, on utilise donc plutôt des algorithmes "greedy" tels que l'algorithme "Matching Pursuit" [MZ93], [BM98] qui donne en pratique de très bons résultats. Ce type d'algorithme a pour objectif de chercher une solution approchée tout en maintenant la contrainte sur le nombre d'éléments non-nuls. Il s'agit d'un algorithme qui permet de calculer les coefficients de $\mathbf{x}_k$ de façon itérative. L'idée consiste à chercher dans le dictionnaire $\mathbf{D}$ l'atome i qui maximise le produit scalaire avec $\mathbf{y}_k$, on calcule son coefficient $x_{ki} =< \mathbf{y}_k, \mathbf{d}_i >$ puis on calcule le résidu $\mathbf{R}_k = \mathbf{y}_k - x_{ki}\mathbf{d}_i$. On recommence ensuite en cherchant l'atome le plus proche du résidu. De cette manière, à l'itération j, on possède une décomposition contenant au plus j coefficients non-nuls.

Le fonctionnement de l'algorithme est le suivant :

Matching pursuit :

a) Entrées : un signal $\mathbf{y}_k$, le dictionnaire $\mathbf{D}$, le nombre maximum d'éléments non-nuls T (parcimonie) de la solution.

b) Initialisation : $\|\mathbf{R}\|_2 \leftarrow \mathbf{y}_k$

c) Tant que $\|\mathbf{R}\|_2 > \epsilon$ et $\|\mathbf{x}_k\|_0 < T$

   1. chercher $\underset{i}{\text{argmax}} < \mathbf{y}_k, \mathbf{d_i} >, \qquad \mathbf{d_i} \in \mathbf{D}$
   2. $x_{ki} \leftarrow < \mathbf{y}_k, \mathbf{d_i} >$
   3. $\mathbf{R} \leftarrow \mathbf{R} - x_{ki}\mathbf{d_i}$

d) Renvoyer le code $\mathbf{x}_k$

Dans le cas de la norme ℓ_1, une solution peut être obtenue en remplaçant l'étape de recherche du code $\mathbf{x}$ donnée précédemment par :

$$\mathbf{x}_k \leftarrow \text{prox}_{\ell_1}(\mathbf{x}_k - \eta \nabla_{\mathbf{x}_k} \mathcal{L}_q(\mathbf{y}_k, \mathbf{D}, \mathbf{x}_k)) \tag{2.11}$$

où prox_{ℓ_1} désigne l'opérateur proximal de la norme ℓ_1 et $\mathcal{L}_q$ désigne uniquement le terme quadratique de $\mathcal{L}$.

Un autre algorithme permet de trouver un code $\mathbf{x}_k$ dans le cas du LASSO (régularisation par une norme ℓ_1).

La formulation du LASSO donnée par l'équation 2.4 dans le cas où nous cherchons un code parcimonieux $\mathbf{x}_k$ s'écrit :

$$\hat{\mathbf{x}}_k = \underset{\mathbf{x}}{\operatorname{argmin}} \quad \|\mathbf{y}_k - \mathbf{D}\mathbf{x}\|_2^2 + \lambda\|\mathbf{x}\|_1 \tag{2.12}$$

L'algorithme LARS (Least Angle Regression) [Efr+04] permet d'obtenir une solution pour la recherche du code optimal $\hat{\mathbf{x}}_k$ pour la formulation donnée par le LASSO. Une implémentation de cet algorithme a été proposée par J. Mairal dans la librairie *SPAMS* [Mai+10] disponible en ligne [1]. En pratique, cette méthode et son implémentation ont été utilisées pour les travaux présentés dans la suite de ce manuscrit.

L'algorithme LARS amène certaines propriétés particulières. En particulier, lorsque le nombre d'atomes K d'un dictionnaire est supérieur à la dimension n de ses atomes, la solution du LASSO pour un dictionnaire $\mathbf{D}$ fixé n'est plus unique. L'algorithme LARS offre des propriétés de stabilité sur les solutions [Tib12] qui sont intéressantes dans certaines applications (comme la classification).

Soit, par exemple, $\mathcal{A}$ l'ensemble des solutions de l'équation 2.4 pour un signal $\mathbf{y}$ et un dictionnaire $\mathbf{D}$ fixés. Il existe un support maximal d'atomes utilisés pour la reconstruction d'un signal $\mathbf{y}$ particulier : $\{i \in [1, K] \quad | \quad \exists \mathbf{x} \in \mathcal{A}, x_i \neq 0\}$. Une des propriétés intéressantes de l'algorithme LARS est que, dans le cas où la solution du LASSO n'est pas unique, la solution fournie (parmi l'ensemble des solutions parcimonieuses optimales) utilise ce nombre maximal d'atomes. Par ailleurs, une telle solution n'est pas unique non plus.

2.2.3.2 Recherche de D

Pour rechercher $\mathbf{D}$, on utilise souvent les itérations de descente de gradient données par l'équation 2.10. Toutefois, il est possible d'utiliser des méthodes différentes. Par exemple, Aharon et al. [AEB06] utilisent une SVD (Singular Value Decomposition) pour mettre à jour un atome du dictionnaire $\mathbf{D}$. Les atomes sont mis à jour successivement en les remplaçant chacun par le vecteur propre associé à la valeur propre la plus importante d'une certaine matrice d'erreur. En effet, celui-ci donne la direction principale de la matrice d'erreur, c'est donc ce vecteur propre qui minimise, pour des coefficients x_{ki} fixés, l'erreur de reconstruction pour l'ensemble d'apprentissage.

SVD :

a) Entrées : un ensemble de signaux $\mathbf{y}_k, k \in [1, m]$, les codes associés $\mathbf{x}_k$, un dictionnaire initial $\mathbf{D}$.

b) Pour chaque atome $\mathbf{d}_i$ du dictionnaire $\mathbf{D}$, pris individuellement

1. http ://spams-devel.gforge.inria.fr/

1. Calculer l'erreur de reconstruction pour chaque $\mathbf{y}_k$ sans la contribution de l'atome $\mathbf{d}_i$:

$$\mathbf{e}_{ki} = \mathbf{y}_k - \sum_{\substack{j=1 \\ j \neq i}}^{K} x_{kj}\mathbf{d}_j$$

2. Calculer la SVD de la matrice $[\mathbf{e}_{1i}, \cdots, \mathbf{e}_{mi}]$
3. Remplacer $\mathbf{d}_i$ par le vecteur propre associé à la valeur propre la plus importante.

c) Renvoyer le dictionnaire $\mathbf{D}$

2.2.3.3 Algorithme global

Si on reprend maintenant l'idée de la résolution du problème en deux étapes dans sa globalité dans le cas d'une contrainte de type pseudo-norme ℓ_0, on obtient par exemple l'algorithme K-SVD décrit ci-dessous.

K-SVD :

a) Entrées : un ensemble $\mathbf{y}_k$, le dictionnaire $\mathbf{D}$, une parcimonie T, un critère d'arrêt ϵ.

b) Tant que *convergence non atteinte*

1. Recherche du code $\hat{\mathbf{x}}_k$ (par exemple avec l'algorithme "Matching Pursuit")

$$\hat{\mathbf{x}}_k = \underset{\mathbf{x}}{\operatorname{argmin}} \quad \|\mathbf{y}_k - \mathbf{D}\mathbf{x}\|_2^2 \quad \text{tel que} \quad \|\mathbf{x}\| \leq T \tag{2.13}$$

2. Mise à jour du dictionnaire $\mathbf{D}$: algorithme SVD

c) Retourner $\mathbf{D}$

Cette structure est assez générale, et on peut par exemple substituer facilement l'algorithme LARS à l'algorithme "Matching Pursuit" à l'étape (1) et une étape de descente de gradient stochastique à la mise à jour par SVD pour l'étape (2).

2.3 Utilisation des dictionnaires

Dans cette section, nous allons brièvement présenter quelques applications dans lesquelles les méthodes basées sur des dictionnaires ont obtenu d'excellents résultats. Nous allons présenter tout d'abord quelques résultats en restauration d'images (débruitage, inpainting) avant d'aborder l'application qui nous intéresse principalement dans ce manuscrit, c'est-à-dire, la classification.

2.3.1 Inpainting

L'inpainting est le nom donné aux méthodes permettant de restaurer les images endommagées ou de compléter les parties manquantes d'une image (pouvant correspondre à un élément supprimé).

L'idée consiste à apprendre un dictionnaire pouvant être appris soit sur un ensemble d'images proches de l'image à traiter ou même sur l'image corrompue [MES08]. Cette méthode fonctionne bien dans le cas où on considère que la dégradation de l'image ne peut pas être modélisée de manière parcimonieuse. Dans le cas contraire, il faut apprendre le dictionnaire sur des images non dégradées.

A titre d'exemple, dans [Mai10], une formulation est proposée pour la résolution du problème d'inpainting :

$$\underset{\mathbf{D},\mathbf{x}_k}{\operatorname{argmin}} \quad \frac{1}{m}\sum_{k=1}^{m} \|\mathbf{M}_k(\mathbf{y}_k - \mathbf{D}\mathbf{x}_k)\|_2^2 + \lambda\|\mathbf{x}_k\|_1 \tag{2.14}$$

où $\mathbf{M}$ est une matrice diagonale dont la j-ème valeur vaut 1 si la j-ème valeur de $\mathbf{y}_i$ est connue et 0 sinon.

Un exemple de restauration est donné par la figure 2.6.

FIGURE 2.6 – Exemple de restauration : à gauche, une image polluée par des éléments de texte et à droite, l'image restaurée. [Mai+09]

2.3.2 Débruitage

Dans l'application de débruitage, l'objectif est de restaurer une image dégradée par un bruit (souvent un bruit blanc additif gaussien) : chacune des valeurs des pixels de l'image peut s'écrire comme la somme de la valeur réelle du pixel et d'une valeur de bruit.
Dans ce contexte, l'utilisation de dictionnaires appris a eu un grand succès [EA06]. Dans la même optique que pour l'inpainting, l'idée principale utilisée est que les atomes du dictionnaire peuvent capturer la structure de l'image. L'image est traitée comme un ensemble de patchs et on apprend un dictionnaire de patchs sur un ensemble d'images non bruitées. On effectue alors la décomposition de patchs bruités à l'aide du dictionnaire appris. L'ensemble d'apprentissage étant non-bruité, le bruit est mal reconstruit par ce dictionnaire.

2.3.3 Classification

Les dictionnaires peuvent permettre d'apprendre les motifs présents dans les images et de trouver une base permettant une décomposition parcimonieuse. Le fait de pouvoir trouver un code à partir d'un faible nombre d'atomes explicatifs pour une image a naturellement fourni une nouvelle application possible : la classification à partir de ces codes. Il s'agit de découvrir les éléments distinctifs de chacune des différentes classes, qui permettent d'obtenir des descripteurs $\hat{\mathbf{x}}_k$ conduisant à une classification robuste.

Plusieurs approches existent pour répondre à une tâche de classification à l'aide de dictionnaires. La première consiste à coupler directement l'utilisation des codes parcimonieux obtenus par les méthodes décrites précédemment avec un algorithme de classification (par exemple SVM). Ces codes peuvent être obtenus en décomposant l'image entière sur un dictionnaire.

Une autre approche, parmi les plus simples utilisées pour une application de classification est la construction d'un dictionnaire pour chacune des classes de l'ensemble d'apprentissage. Ces dictionnaires sont appris à partir d'une des méthodes évoqués précédemment, c'est-à-dire, de manière non-supervisée et sur un critère de reconstruction.
L'idée principale de cette approche est que chacun des dictionnaires fournit une base dans laquelle la classe associée est mieux représentée que les autres. Lors de la classification d'un nouveau signal, on recherche alors le dictionnaire qui minimise l'erreur de reconstruction pour une contrainte de parcimonie égale : la classe associée à ce dictionnaire fournit alors la classe du signal considéré.

Pour classer des signaux, il est aussi possible de construire des fonctions de régularisation dans un objectif de classification. Par exemple GDDL (Group-structured Dirty Dictionary Learning) [Suo+14] :

GDDL

$$\underset{\mathbf{D},\mathbf{A},\mathbf{B}}{\operatorname{argmin}} \quad \|\mathbf{Y} - \mathbf{D}(\mathbf{A}+\mathbf{B})\|_2^2 + \lambda_1\|\mathbf{A}\|_{1,\infty} + \lambda_2\|\mathbf{B}\|_{1,1} \tag{2.15}$$

$\mathbf{Y} = [\mathbf{y}_1..\mathbf{y}_n]$ correspond à la matrice qui regroupe chaque signal de l'ensemble d'apprentissage et $\|\cdot\|_{1,\infty}$ fait référence à une norme mixte utilisant la norme ℓ_1 pour la "première" dimension et la norme ℓ_∞ pour la "seconde" dimension (i.e dans ce cas, on considère donc la somme des valeurs absolues de la norme ℓ_∞ des lignes de la matrice $\mathbf{A}$).

La formulation décompose la recherche du code optimal $\mathbf{x}$ en une somme de deux termes : $\mathbf{x} = \mathbf{a} + \mathbf{b}$ où $\mathbf{a}$ représente l'aspect "commun" d'une classe, c'est-à-dire que chaque signal d'une même classe doit idéalement utiliser un socle d'atomes commun, et $\mathbf{b}$ représente la "particularité" de chaque signal, c'est à dire la variabilité qui différencie deux éléments (même au sein de la même classe) entre eux.

Considérons un problème de classification à deux classes, un dictionnaire est appris de manière non supervisée à partir des données d'apprentissage. On peut supposer que les atomes sont associés à l'une ou l'autre des deux classes (par exemple, en apprenant un dictionnaire par classe puis en concaténant ces deux dictionnaires). La figure 2.7 résume différents cas de figure pour illustrer la formulation précédente :

1. Le premier cas représente la reconstruction "classique". Pendant l'étape de reconstruction, des atomes associés aux deux classes peuvent être utilisés. En effet, des atomes visuellement proches pourraient être associés à des classes différentes.
2. Dans le second cas, la reconstruction est contrainte à l'utilisation d'atomes spatialement proches. Cependant, les signaux près des limites entre les classes ne sont pas affectés.
3. Dans le troisième cas, les images ne sont pas encodées indépendamment mais par groupe, en forçant une incohérence entre les atomes associés aux deux classes.
4. Dans le quatrième cas, les images sont également encodées par groupe, en forçant cette fois-ci l'utilisation d'un ensemble commun d'atomes.

Dans la pratique, cette dernière méthode est difficilement applicable dans les tests car il faut avoir une connaissance a priori sur l'appartenance à une même classe d'un groupe d'images.

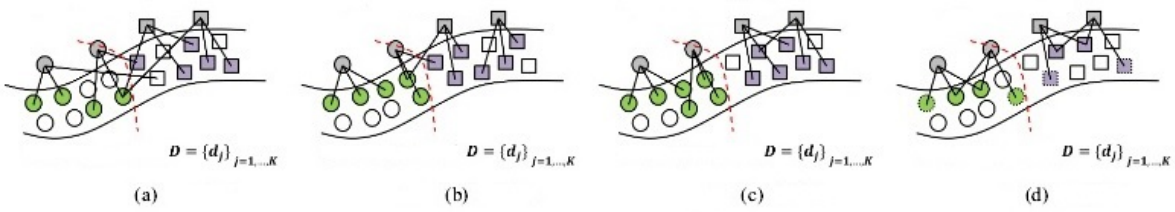


FIGURE 2.7 – (a) Codage classique. (b) Codage avec contrainte de localité. (c) Codage par classe. (d) Codage de groupe. (Reproduction extraite de [Suo+14])

Application :

Un des problèmes de classification dans lequel l'apprentissage de dictionnaires s'est illustré est la reconnaissance et classification de visages (Fig. 2.8) [LHK05]. Pour cette application, on dispose d'une base d'apprentissage contenant une collection de photos étiquetées de visages de plusieurs individus dans différentes conditions de luminosité, de poses ou encore d'occlusions. La tâche consiste alors à retrouver, pour un ensemble d'images non étiquetées à quel individu chaque photo correspond. Pour ce problème, l'apprentissage est souvent effectué sur les images complètes de visages qui font entre 30 et 60 pixels de largeur et de hauteur, et un dictionnaire par individu est appris.
En pratique, les méthodes utilisant des dictionnaires présentent de bons résultats pour cette application. L'article [Wri+09], par exemple, décrit une méthode dans laquelle, le dictionnaire utilisé est construit à partir d'éléments tirés directement de l'ensemble d'apprentissage. Dans un autre exemple [ZL10], l'auteur présente une méthode pour apprendre un dictionnaire qui favorise l'apprentissage d'atomes discriminants.

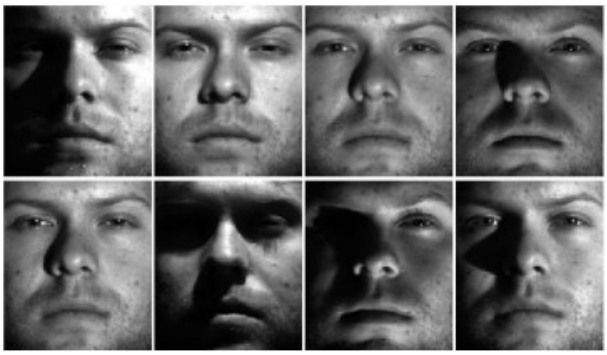


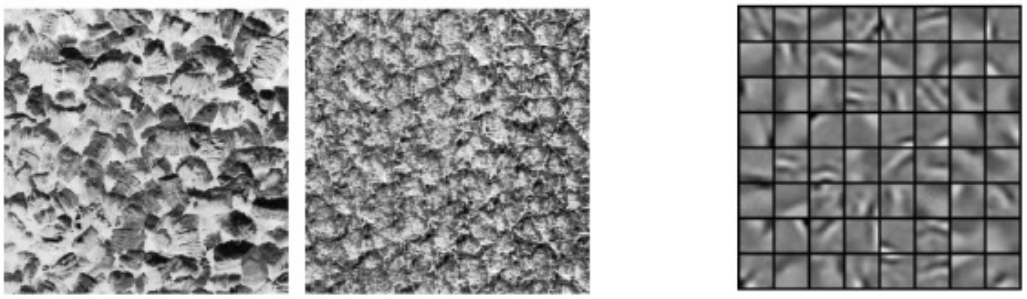
FIGURE 2.8 – Extrait de la base de visages "Extended Yale B" [LHK05]

Un autre exemple de tâches de classification dans laquelle l'apprentissage de dictionnaires s'est illustré est la classification de textures [Mai+08]. Pour cette application, les images sont souvent décomposées en patchs et chacun des patchs peut être classé séparément.

Sur le même principe, si la taille de l'image est trop importante, on peut décomposer celle-ci en un ensemble de patchs puis calculer les codes de chacun des patchs avant d'effectuer une étape d'agrégation des codes (par exemple, sacs de mots visuels (BoW)) puis la classification. Ce type de méthode consiste à compter le nombre d'apparitions de chaque élément d'un dictionnaire sur un ensemble de patchs pour fournir une représentation globale de l'image sous une forme condensée, comme un histogramme. Ces méthodes sont par ailleurs souvent utilisées avec un algorithme de K-moyennes pour lequel les K centroïdes peuvent être vus comme formant un dictionnaire. Dans le cas de l'algorithme K-moyennes, le fonctionnement correspond dans l'idée à un encodage sur un dictionnaire avec une contrainte $\|\mathbf{x}\|_0 = 1$.

Les signaux présentés dans cette section sont principalement des images. Cependant dans le cas de la classification en particulier, il faut noter que les signaux considérés peuvent être eux-mêmes des descripteurs. Les pixels peuvent tout aussi bien représenter une valeur de niveau de gris, une valeur d'un canal de couleur ou même encore une valeur de flot optique.

Pour compléter, certains travaux se sont intéressés à l'apprentissage de représentations



(a) (b)

FIGURE 2.9 – (a) Exemples de textures. (b) Exemple de dictionnaire appris. [Mai+08]

parcimonieuses à l'aide de dictionnaire [Bar13] dans le cas de signaux multivariés. Il s'agit par exemple, d'apprendre un dictionnaire $\mathbf{D}$ quand les signaux $\mathbf{y}_k$ ont plusieurs composantes comme une série temporelle regroupant les mesures de plusieurs capteurs, ainsi que d'étendre certaines méthodes de recherche de codes $\mathbf{x}$ à ce cadre d'étude.

2.4 Conclusion

Dans cette section, nous avons présenté des applications directes de l'apprentissage de dictionnaires, en particulier, la tâche de classification. Les techniques abordées dans cette partie fonctionnent de manière non supervisée et ne prennent pas directement en compte certaines informations (telles que les labels ou classes) associées aux images de l'ensemble d'apprentissage. Une limite à cette dernière réside dans le fait que dans les formulations présentées, le ou les dictionnaires sont obtenus en tant que solutions de la minimisation d'un critère de reconstruction. Or, chercher des atomes qui permettent de bien reconstruire est différent de chercher des atomes qui permettent de bien discriminer. Dans la suite, nous allons présenter une méthode qui permet d'introduire de la supervision dans l'apprentissage des dictionnaires.

Chapitre 3

Apprentissage supervisé de dictionnaires

Sommaire

Nous voulons effectuer de la classification à partir de codes issus de dictionnaires. Dans le chapitre précédent, nous avons présenté une formulation d'apprentissage de dictionnaires basée sur un critère de reconstruction alors que l'on cherche une représentation discriminante. Dans ce chapitre, nous proposons d'étudier plusieurs méthodes qui tiennent compte d'une information supplémentaire sur les signaux $\mathbf{y}_k$ à classer : la classe l_k à laquelle ces signaux sont associés. La problématique à traiter est la suivante : comment introduire la supervision lors de l'apprentissage des dictionnaires. En particulier, nous allons nous intéresser à trois aspects :

— l'apprentissage séparé entre le dictionnaire et le classifieur

— l'intégration de l'information de label au sein du dictionnaire

— l'apprentissage joint du dictionnaire et du classifieur

3.1 Apprentissage séparé du dictionnaire et du classifieur

Le problème peut se formuler de la façon suivante [MBP12] : on considère un signal $\mathbf{y}_k$ associé à un label l_k et le code $\mathbf{x}_k$ obtenu en décomposant ce signal $\mathbf{y}_k$ sur un dictionnaire $\mathbf{D}$. On souhaite alors pouvoir prédire un label $\hat{l}_k$ et minimiser l'erreur de prédiction entre la classe réelle l_k et la classe prédite $\hat{l}_k$.

Dans cette partie, on choisit la notation :

$$\hat{\mathbf{x}}_k = \underset{\mathbf{x}}{\operatorname{argmin}} \quad \|\mathbf{y}_k - \mathbf{D}\mathbf{x}\|_2^2 + \phi(\mathbf{x}) \tag{3.1}$$

où le vecteur $\hat{\mathbf{x}}_k$ désigne spécifiquement la solution au problème d'optimisation pour un signal $\mathbf{y}_k$ et un dictionnaire $\mathbf{D}$ fixé, ici appris suivant un critère de reconstruction comme au chapitre précédent.

Dans le chapitre précédent, nous avons évoqué l'utilisation des codes parcimonieux $\hat{\mathbf{x}}_k$ en tant que descripteurs en entrée d'une fonction de classification. On considère une fonction de coût $\mathcal{L}$ pour le problème d'apprentissage d'un classifieur. Si elle est bien définie et différentiable, de paramètres $\mathbf{W}$, il est alors possible d'appliquer des méthodes d'optimisation classiques, telle qu'une descente de gradient stochastique, pour chercher des paramètres $\mathbf{W}$. L'apprentissage du classifieur se fait indépendamment de celui du dictionnaire $\mathbf{D}$ (Fig. 3.1) :

$$\min_{\mathbf{W}} \quad \frac{1}{m} \sum_{k=1}^{m} \mathcal{L}(l_k, \hat{\mathbf{x}}_k, \mathbf{W}) \tag{3.2}$$

où $\mathbf{W}$ représente l'ensemble des paramètres de $\mathcal{L}$, m représente le nombre de signaux dans l'ensemble d'apprentissage et $\hat{\mathbf{x}}_k$ est solution de l'équation 3.1 pour un certain $\mathbf{y}_k$.

Un inconvénient à cette méthode est que dans la formulation classique, le dictionnaire est appris pour bien reconstruire des signaux $\mathbf{y}_k$ sans tenir compte de la connaissance l_k sur la classe des signaux $\mathbf{y}_k$.

Nous souhaitons apprendre des dictionnaires adaptés pour la classification et pas uniquement pour reconstruire les signaux. Dans la littérature, il existe deux grandes familles de techniques pour apprendre des dictionnaires pour la classification [KW12] : ces méthodes sont présentées dans les deux parties qui suivent.

3.2 Dictionnaires incluant le label

La première famille de techniques consiste à intégrer l'information discriminantes directement dans les dictionnaires. Par exemple, dans le chapitre précédent, nous avons évoqué l'utilisation d'un dictionnaire par classe ainsi qu'une classification utilisant l'erreur de reconstruction [Wri+09], [LWQ13]. Dans ce cas, les labels l_k ne sont pas explicitement contenus dans les dictionnaires mais un ensemble de dictionnaire est appris où chacun des dictionnaires sert à reconnaitre une classe en particulier sur la base du calcul d'une erreur de reconstruction (Fig. 3.2). Très souvent, dans ce genre d'approche, on ajoute également un terme à l'équation d'apprentissage des dictionnaires qui réduit la ressemblance entre dictionnaires de différentes classes [RSS10].

Une autre méthode consiste à intégrer directement l'information de labels au sein du dictionnaire [JLD11], [ZL10] : l'idée ici est par exemple de modifier les signaux $\mathbf{y}_k$ en leur concaténant l'information de label l_k. Par conséquent, chacun des atomes $\mathbf{d}_j$ du dictionnaire

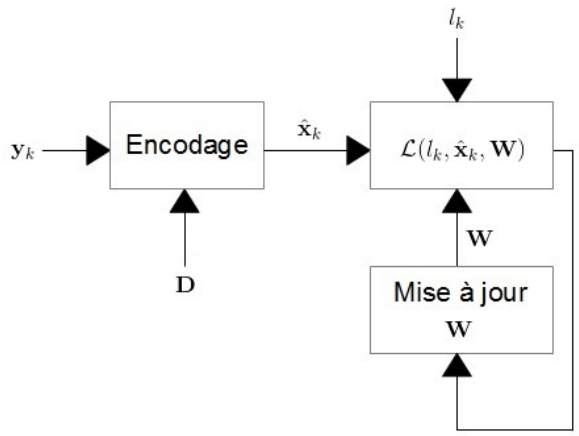


FIGURE 3.1 – Le dictionnaire $\mathbf{D}$ et les paramètres $\mathbf{W}$ du classifieur sont appris séparément. Dans ce cas, le dictionnaire $\mathbf{D}$ est déjà connu et fixé, les paramètres $\mathbf{W}$ sont appris à partir de l'ensemble des codes $\hat{\mathbf{x}}_k$ et des labels l_k associés.

$\mathbf{D}$ appris est associé à une classe. Un exemple d'une telle méthode est présenté par Zhang et Li [ZL10] et de plus amples détails sont fournis dans leur article.

Brièvement, leur algorithme baptisé *D-KSVD* résout le problème suivant :

$$\underset{\mathbf{x},\mathbf{D},\mathbf{W}}{\operatorname{argmin}} \quad \sum_{k=1}^{m} (\|\mathbf{y}_k - \mathbf{D}\mathbf{x}\|_2^2 + \lambda_1 \|l_k - \mathbf{W}\mathbf{x}\|_2^2) + \lambda_2 \|\mathbf{W}\|_2^2 \quad \text{avec} \quad \|\mathbf{x}\|_0 \leq T \tag{3.3}$$

où $\|l_k - \mathbf{W}\mathbf{x}\|_2^2$ joue le rôle d'un coût de classification par une opération linéaire. Le terme de régularisation utilisé pour $\mathbf{x}$ est ici la pseudo-norme ℓ_0.

Les auteurs proposent de reformuler le problème. En concaténant, le label l_k au signal $\mathbf{y}_k$ et la matrice $\mathbf{W}$ au dictionnaire $\mathbf{D}$, on obtient :

$$\underset{\mathbf{x},\mathbf{D},\mathbf{W}}{\operatorname{argmin}} \quad \sum_{k=1}^{m} \| \begin{bmatrix} \mathbf{y}_k \\ \sqrt{\lambda_1} l_k \end{bmatrix} - \begin{bmatrix} \mathbf{D} \\ \sqrt{\lambda_1} \mathbf{W} \end{bmatrix} \mathbf{x} \|_2^2 \quad \text{avec} \quad \|\mathbf{x}\|_0 \leq T \tag{3.4}$$

La nouvelle formulation est similaire à un apprentissage de dictionnaire dans lequel le label a été intégré. De plus, le terme de régularisation $\lambda_2 \|\mathbf{W}\|_2^2$ sur $\mathbf{W}$ a été retiré car nous rappelons que les colonnes du dictionnaire sont généralement normalisées et donc bornées.

En notant $\mathbf{D}^\star = \begin{bmatrix} \mathbf{D} \\ \sqrt{\lambda_1} \mathbf{W} \end{bmatrix}$ et $\mathbf{y}_k^\star = \begin{bmatrix} \mathbf{y}_k \\ \sqrt{\lambda_1} l_k \end{bmatrix}$, on retrouve bien une formalisation similaire à l'apprentissage du dictionnaire (Fig. 3.3) :

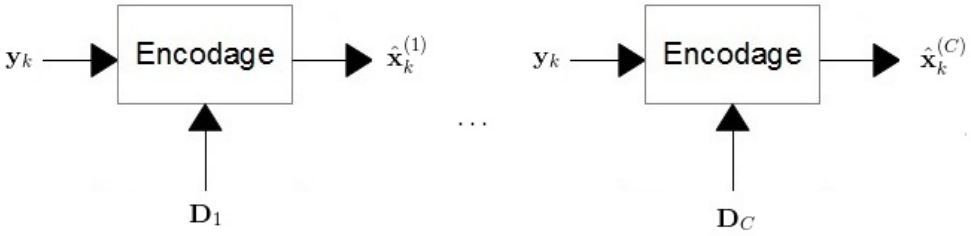


FIGURE 3.2 – Un signal $\mathbf{y}_k$ est décomposé sur chacun des C dictionnaires $\mathbf{D}_i$ donnant le code $\hat{\mathbf{x}}_k^{(i)}$. On effectue ensuite, à partir de chacun des codes, la recherche de l'erreur de reconstruction minimale (par exemple) sur l'ensemble des dictionnaires pour déterminer l'appartenance à une classe.

$$\underset{\mathbf{x},\mathbf{D}^\star}{\operatorname{argmin}} \quad \sum_{k=1}^{m} \|\mathbf{y}_k^\star - \mathbf{D}^\star \mathbf{x}\|_2^2 \quad \text{avec} \quad \|\mathbf{x}\|_0 \leq T \tag{3.5}$$

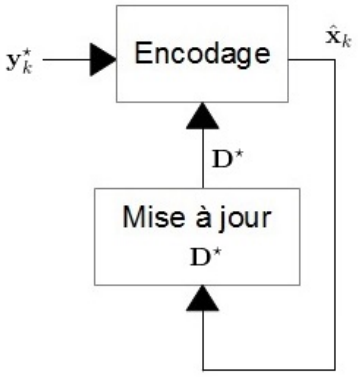


FIGURE 3.3 – Un signal $\mathbf{y}_k^\star$ est décomposé sur un dictionnaire $\mathbf{D}^\star$ donnant le code $\hat{\mathbf{x}}_k$. Le dictionnaire $\mathbf{D}^\star$ est appris sur un critère de reconstruction.

3.3 Apprentissage joint d'un dictionnaire et d'un classifieur

3.3.1 Formulation

La deuxième famille de méthodes pour l'apprentissage de dictionnaires adaptés pour la classification consiste à rendre les coefficients $\hat{\mathbf{x}}_k$ de décomposition discriminants, par exemple en apprenant conjointement un dictionnaire $\mathbf{D}$ et un classifieur. Ces méthodes ont montré de très bonnes performances en classification, par exemple sur la base de données MNIST [MBP12].

Au contraire de l'équation 3.2, dans laquelle on s'intéressait uniquement aux paramètres du classifieur, l'objectif est maintenant d'apprendre conjointement sur un critère de classification le dictionnaire $\mathbf{D}$ et le classifieur ainsi que ses paramètres $\mathbf{W}$ (Fig. 3.4). Cette méthode peut se formuler par l'équation suivante :

$$\min_{\mathbf{D},\mathbf{W}} \quad \frac{1}{m}\sum_{k=1}^{m} \mathcal{L}(l_k, \hat{\mathbf{x}}_k, \mathbf{W}) \tag{3.6}$$

où m est le nombre de signaux.

On note ici que l'on cherche à minimiser le problème par rapport au dictionnaire $\mathbf{D}$. On peut remarquer que $\mathbf{D}$ n'apparait pas explicitement dans la fonction de coût mais il intervient par l'intermédiaire des codes $\hat{\mathbf{x}}_k$.

Pour minimiser une fonction par rapport à l'ensemble des variables utilisées, il est possible de se servir de méthodes reposant sur un concept similaire à la "rétropropagation" [LeC+98a] utilisée dans les réseaux de neurones. La difficulté réside dans la recherche du dictionnaire $\mathbf{D}$ conduisant à un minimum de la fonction $\mathcal{L}$. Pour parvenir à franchir cette difficulté, un problème subsiste : il est nécessaire d'obtenir une expression du gradient qui lie la fonction de coût $\mathcal{L}$ au dictionnaire $\mathbf{D}$. Cependant, cette relation est complexe car ces deux éléments sont liés par les codes $\hat{\mathbf{x}}_k$ qui sont solutions d'un problème de minimisation (Eq. 3.1) potentiellement non-lisse (par exemple si la régularisation utilise la norme ℓ_1).

L'objectif est de calculer les gradients $\nabla_{\mathbf{W}}\mathcal{L}(l_k, \hat{\mathbf{x}}_k, \mathbf{W})$ et $\nabla_{\mathbf{D}}\mathcal{L}(l_k, \hat{\mathbf{x}}_k, \mathbf{W})$. Le gradient $\nabla_{\mathbf{W}}\mathcal{L}(l_k, \hat{\mathbf{x}}_k, \mathbf{W})$ ne pose en général aucun problème et la difficulté se concentre autour du calcul de $\nabla_{\mathbf{D}}\mathcal{L}(l_k, \hat{\mathbf{x}}_k, \mathbf{W})$.
Dans la littérature, plusieurs méthodes ont été utilisées pour résoudre ce problème. Tout d'abord, ce problème a été abordé par Bradley et Bagnell [BB08] en utilisant une fonction de régularisation plus lisse, ce qui a permis d'utiliser la différentiation implicite pour calculer le gradient de $\mathcal{L}$ par rapport à $\mathbf{D}$: avec une fonction de régularisation bien choisie (par exemple ℓ_2), on obtient une formulation explicite pour déterminer le code $\hat{\mathbf{x}}_k$ en fonction de $\mathbf{D}$ et de $\hat{\mathbf{y}}_k$ qui est différentiable. Un des inconvénients de la méthode est que la nouvelle fonction de régularisation proposée renvoie des coefficients $\hat{\mathbf{x}}_k$ de faibles valeurs mais différents de zéro : la parcimonie n'est pas respectée.
Une deuxième approche a été utilisée par Yang et al. [YYH10] et se base sur la rétropropa-

gation du gradient. Elle utilise également la différentiation implicite proposée dans [BB08] en conservant une fonction de régularisation qui apporte la parcimonie (norme ℓ_1) grâce à quelques simplifications. Cependant, l'implémentation proposée est coûteuse en terme de calculs ce qui la rend peu pratique pour des problèmes d'apprentissage de taille importante.
La troisième approche étudiée est proposée par Mairal et al. [MBP12] et regroupe l'avantage de la méthode précédente, c'est-à-dire le respect de la parcimonie, ainsi que l'implémentation d'un algorithme plus rapide. Les détails des calculs et des preuves de différentiabilité sont donnés dans l'article [MBP12]. Cette méthode est détaillée dans la suite.

La méthode proposée par Mairal et al. [MBP12] contient des outils pour le calcul du gradient de $\mathcal{L}$ qui permet de conserver les propriétés de parcimonie. L'équation donnée dans l'article est la suivante :

$$\nabla_{\mathbf{D}}\mathcal{L}(l_k, \hat{\mathbf{x}}_k, \mathbf{W}) = \mathbf{D}\beta\hat{\mathbf{x}}_k^\top + (\mathbf{y} - \mathbf{D}\hat{\mathbf{x}}_k)\beta^\top \tag{3.7}$$

Tout d'abord, on note $\Lambda = \{i | x_i \neq 0\})$, l'ensemble des indices des coefficients non-nuls pour le vecteur de coefficients $\hat{\mathbf{x}}_k$ considéré.

Pour tous les indices $j \in \Lambda$, β est défini comme suit :

$$\beta_\Lambda = (\mathbf{D}_\Lambda^\top \mathbf{D}_\Lambda)^{-1} \nabla_{\hat{\mathbf{x}}_{k\Lambda}} \mathcal{L}(l_k, \hat{\mathbf{x}}_k, \mathbf{W}) \tag{3.8}$$

où $\hat{\mathbf{x}}_{k\Lambda}$ et $\mathbf{D}_\Lambda$ correspondent à $\hat{\mathbf{x}}_k$ et $\mathbf{D}$ restreints aux indices des coefficients non-nuls et $\beta_j = 0$, si $j \notin \Lambda$. Pour déterminer le gradient de $\mathcal{L}$ par rapport à $\mathbf{D}$, on voit apparaître le gradient de $\mathcal{L}$ par rapport à $\hat{\mathbf{x}}_k$.

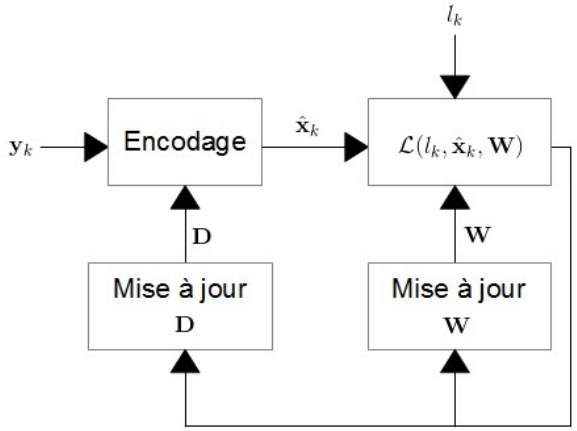


FIGURE 3.4 – Un signal $\mathbf{y}_k$ est décomposé sur un dictionnaire $\mathbf{D}$, le code résultant $\hat{\mathbf{x}}_k$ est envoyé en entrée d'une fonction de coût de paramètres $\mathbf{W}$. On mesure ensuite le coût $\mathcal{L}(l_k, \hat{\mathbf{x}}_k, \mathbf{W})$. On peut alors mettre à jour le dictionnaire $\mathbf{D}$ et les paramètres $\mathbf{W}$ pour s'adapter au problème de classification en calculant les différents gradients par rapport à l'erreur obtenue.

On résume ci-dessous, le pseudo-code de l'apprentissage conjoint du classifieur et du dictionnaire :

Apprentissage conjoint classifieur/dictionnaire :

a) Entrées : un ensemble de signaux $E = \{\mathbf{y}_k, k \in [1, m]\}$, le dictionnaire $\mathbf{D}$, la fonction de coût $\mathcal{L}$.

b) Tant que *Convergence non atteinte*

1. Tirer aléatoirement un signal $\mathbf{y}_k \in E$
2. Calculer $\hat{\mathbf{x}}_k$, solution de l'équation 3.1 avec le dictionnaire $\mathbf{D}$
3. Mettre à jour $\mathbf{D}$ et $\mathbf{W}$:

$$\mathbf{D} \leftarrow \mathbf{D} - \eta_1 \nabla_{\mathbf{D}} \mathcal{L}(l_k, \hat{\mathbf{x}}_k, \mathbf{W})$$
$$\mathbf{W} \leftarrow \mathbf{W} - \eta_2 \nabla_{\mathbf{W}} \mathcal{L}(l_k, \hat{\mathbf{x}}_k, \mathbf{W})$$

c) Renvoyer $\mathbf{D}$ et $\mathbf{W}$

3.3.2 Exemples de méthodes de classification

Dans le cadre de nos travaux, nous avons appliqué la méthode décrite à la sous section précédente pour deux méthodes de classification associées à deux fonctions de coût.

Les deux méthodes de classification choisies sont les suivantes : la classification linéaire couplée à une fonction softmax, ainsi que les SVM [Vap95]. Ces deux méthodes sont très souvent employées dans la littérature.

Classifieur linéaire + Softmax :

Pour un problème de classification à C classes, la fonction softmax donne une probabilité d'appartenance à la classe j. Couplée à un classifieur linéaire, elle se calcule de la manière suivante :

$$p(l_k = j|\hat{\mathbf{x}}_k) = \frac{\exp(\hat{\mathbf{x}}_k^\top \mathbf{w_j})}{\sum_{i=1}^{C} \exp(\hat{\mathbf{x}}_k^\top \mathbf{w_i})} \tag{3.9}$$

où $\mathbf{w_j}$ est un vecteur de poids associé à la classe j.

La prédiction de la classe du signal est alors :

$$\hat{l}_k = \underset{j}{\operatorname{argmax}} \quad p(l_k = j|\hat{\mathbf{x}}_k) \tag{3.10}$$

La fonction de coût utilisée est alors la fonction d'entropie croisée [Bis06] :

$$\mathcal{L}(l_k, \hat{\mathbf{x}}_k, \mathbf{W}) = -\sum_{i=1}^{C} l_k p(l_k = i|\hat{\mathbf{x}}_k) \tag{3.11}$$

Classifieur SVM :

Le ℓ_2-SVM minimise la fonction de coût suivante :

$$\mathcal{L}(l_k, \hat{\mathbf{x}}_k, \mathbf{w}) = \frac{1}{2}\mathbf{w}^\top \mathbf{w} + \lambda \sum_{k=1}^{m} \max(1 - \mathbf{w}^\top \hat{\mathbf{x}}_k l_k, 0)^2 \tag{3.12}$$

Cette formulation tend à favoriser une classification robuste grâce à l'utilisation d'une marge. L'équation 3.12 fonctionne pour un problème de classification binaire. Pour un problème multiclasse, on utilise cette formulation dans un cadre "one versus all".

3.3.3 Gradients des fonctions de coût

Dans cette sous-section, nous donnons les gradients des deux fonctions de coût associées aux méthodes de classification présentées à la section 3.3.2 pour la formulation donnée dans la sous-section 3.3.1 : $\nabla_{\mathbf{W}} \mathcal{L}(l_k, \hat{\mathbf{x}}_k, \mathbf{W})$ et $\nabla_{\hat{\mathbf{x}}_k} \mathcal{L}(l_k, \hat{\mathbf{x}}_k, \mathbf{W})$, ce dernier permettant d'évaluer $\nabla_{\mathbf{D}} \mathcal{L}(l_k, \hat{\mathbf{x}}_k, \mathbf{W})$ par l'équation 3.7. Ces gradients sont utilisés pour propager l'erreur de la fonction de coût du problème de classification vers les éléments à apprendre, en l'occurrence les dictionnaires et les paramètres $\mathbf{W}$.

Classifieur linéaire + Softmax

Gradient par rapport au code $\hat{\mathbf{x}}_k$:

$$\nabla_{\hat{\mathbf{x}}_k} \mathcal{L}(l_k, \hat{\mathbf{x}}_k, \mathbf{W}) = \mathbf{W}(\mathbf{p}_k - \mathbf{l}_k) \tag{3.13}$$

où $\mathbf{p}_k = (p(l_k = j|\hat{\mathbf{x}}_k))_{j \in [1,C]}$ et $\mathbf{l}_k$ est un vecteur de $\mathcal{R}^C$ qui contient des zéros et une valeur 1 à l'indice $i = l_k$, la classe associée au signal $\mathbf{y}_k$.

Gradient par rapport aux paramètres $\mathbf{W} \in \mathcal{R}^{(K \times C)}$:

$$\nabla_{\mathbf{W}} \mathcal{L}(l_k, \hat{\mathbf{x}}_k, \mathbf{W}) = \hat{\mathbf{x}}_k (\mathbf{p}_k - \mathbf{l}_k)^\top \tag{3.14}$$

Classifieur SVM

Les calculs pour le SVM qui suivent sont inspirés du document [Tan13].

Gradient par rapport au code $\hat{\mathbf{x}}_k$:

$$\nabla_{\hat{\mathbf{x}}_k} \mathcal{L}(l_k, \hat{\mathbf{x}}_k, \mathbf{w}) = -2\lambda l_k \cdot \mathbf{w} \cdot \max(1 - \mathbf{w}^\top \hat{\mathbf{x}}_k l_k, 0) \tag{3.15}$$

Gradient par rapport aux paramètres $\mathbf{w}$:

$$\nabla_{\mathbf{w}}\mathcal{L}(l_k, \hat{\mathbf{x}}_k, \mathbf{w}) = \mathbf{w} - 2\lambda l_k \cdot \hat{\mathbf{x}}_k \cdot \max(1 - \mathbf{w}^\top \hat{\mathbf{x}}_k l_k, 0) \tag{3.16}$$

3.3.4 Un dictionnaire versus plusieurs dictionnaires

Dans le cas d'un problème de classification multiclasse, on peut adopter différentes stratégies : la plus directe consiste à traiter l'ensemble des classes dans un unique problème. Cela signifie apprendre un unique dictionnaire qui est apte à distinguer les différentes classes. Ce dictionnaire regroupe donc un ensemble de motifs issus de l'ensemble des classes.

Il est également possible de décomposer un problème multiclasse en un ensemble de problèmes de classification binaire où, pour chaque classe, les éléments de cette classe sont opposés aux éléments de toutes les autres classes ("un contre tous"). Il s'agit donc d'apprendre autant de dictionnaires que de classes présentes dans le problème.

Selon le classifieur utilisé, on peut favoriser l'une ou l'autre des formulations. Par exemple, pour un classifieur de type SVM [MCS06], il est courant d'utiliser un apprentissage de type "un contre tous" car il est souvent formulé pour être utilisé sur des problèmes binaires. De plus, dans certains cas, il peut être plus avantageux en terme de temps d'apprentissage d'apprendre n dictionnaires dans une configuration "un contre tous" qu'un unique dictionnaire dans une configuration multiclasse [MBP12].

3.4 Interprétations

Dans cette section, nous allons essayer de donner quelques intuitions sur les effets du couplage entre classifieur et dictionnaires lors de l'apprentissage.

Tout d'abord, la décomposition sur un dictionnaire peut être vue comme un changement d'espace de représentation [MBP14]. On considère l'exemple suivant : on souhaite classer un ensemble de points dans un espace à 3 dimensions positionnés sur la surface d'une sphère. On peut obtenir un tel positionnement en considérant des signaux normalisés (norme $\|\mathbf{x}\|_2 = 1$), ce qui est couramment utilisé. Ces points sont répartis en deux classes. La figure 3.5 présente deux exemples de dictionnaires ainsi qu'un exemple de configuration de ces points : le premier dictionnaire est appris pour minimiser l'erreur de reconstruction sur chacune des classes séparément. Un classifieur linéaire couplé à la fonction softmax est ensuite appris pour séparer les deux classes (image (a)). On peut constater que dans ce cas, les atomes du dictionnaire sont positionnés au sein des ensembles de points. Ce comportement est attendu étant donné que le critère minimisé est l'erreur de reconstruction.
Dans le deuxième cas, le dictionnaire et le classifieur sont appris conjointement. Cette fois, les atomes se positionnent autour des ensembles de points dans l'objectif de les discriminer. Un tel positionnement présente un avantage : la figure 3.6 montre pour deux atomes particuliers quels sont les points de l'espace pour lesquels ces atomes interviennent dans la décomposition. On constate que pour un point de l'ensemble vert central, seul l'atome entouré en vert est utilisé :

la contribution de l'atome entouré en rouge est nulle pour chacun des points mentionnés. Cette caractéristique permet une bonne séparabilité en terme de classification.

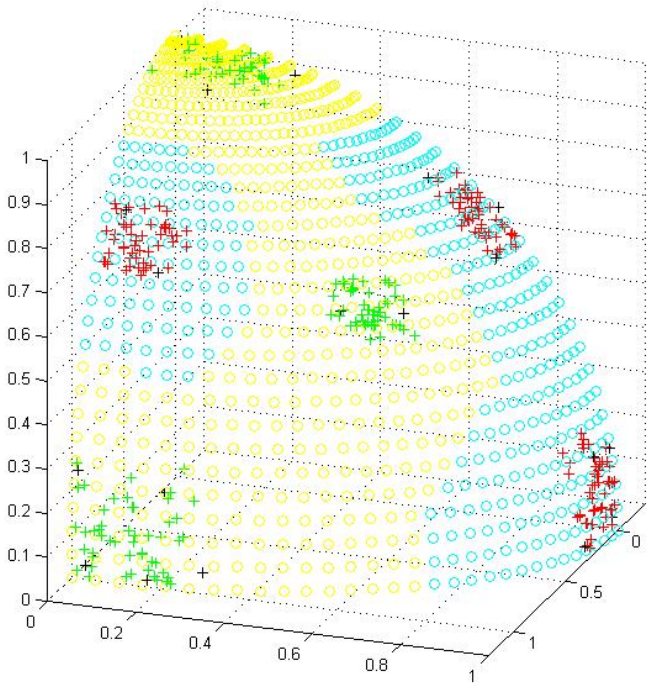
En résumé, pour un critère basé sur l'erreur de reconstruction, la position des atomes est influencée par la densité dans l'espace des signaux à reconstruire pour réduire l'erreur de reconstruction moyenne. Dans le cas d'un critère basé sur une erreur de classification, les atomes semblent se positionner à la frontière des ensembles : la méthode présentée améliore donc le caractère discriminant des codes obtenus.

3.5 Conclusion

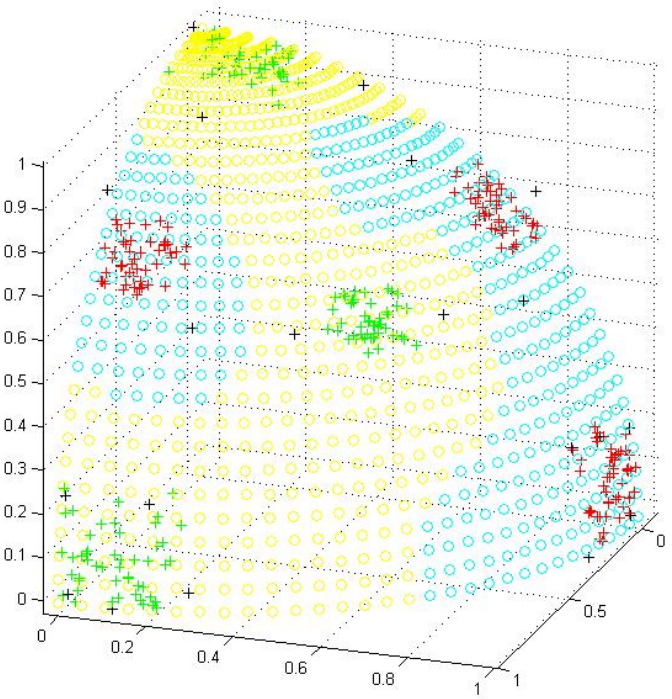
La connaissance de la classe d'un échantillon permet d'adapter l'apprentissage du dictionnaire : la formulation donnée à la section 3.3.1 permet d'apprendre un dictionnaire $\mathbf{D}$ de manière supervisée. Dans ce chapitre, nous avons synthétisé les différentes méthodes de la littérature, et les tests que nous avons réalisés donnent des résultats intéressants suivant les applications traitées. D'autres critères doivent cependant être pris en compte comme le temps de traitement par exemple.

Dans l'article initial [MBP12], la méthode est appliquée sur des images de taille raisonnable donnant lieu à des dictionnaires où un atome est de même dimension qu'un signal de l'ensemble d'apprentissage. Dans le cas d'image de taille plus importante, nous avons évoqué au chapitre précédent la possibilité de décomposer l'image en un ensemble de patchs mais il faut alors ajouter un moyen de fusionner l'information des différents patchs. Yang et al. [YYH10] présente une méthode utilisant un dictionnaire appris de manière supervisée et utilisé au niveau de patchs d'images. Leur méthode s'appuie sur une réduction de dimension par utilisation successive de fonction de "pooling". Cela permet de combiner l'information qui provient de patchs proches tout en introduisant une certaine robustesse à la translation.

Cependant, nous pensons qu'effectuer la décomposition à un unique niveau ne permet pas une représentation assez complexe pour une classification efficace. Dans la suite, nous proposons une manière d'apprendre des dictionnaires à différentes échelles de manière conjointe pour obtenir une représentation discriminante sur des images de plus grandes tailles.



(a)



(b)

FIGURE 3.5 – Les atomes sont représentés par des croix noires. Les croix rouges et vertes montres les éléments appartenant à chacune des deux classes. (a) Dictionnaire appris pour la reconstruction couplé à une classification linéaire apprise séparément. (b) Dictionnaire appris conjointement avec un classifieur linéaire. La région en jaune représente l'ensemble des positions de l'espace qui seraient classées avec les points en vert. La région en cyan représente l'ensemble des positions de l'espace qui seraient classées avec les points en rouge. (Image à voir en couleur)

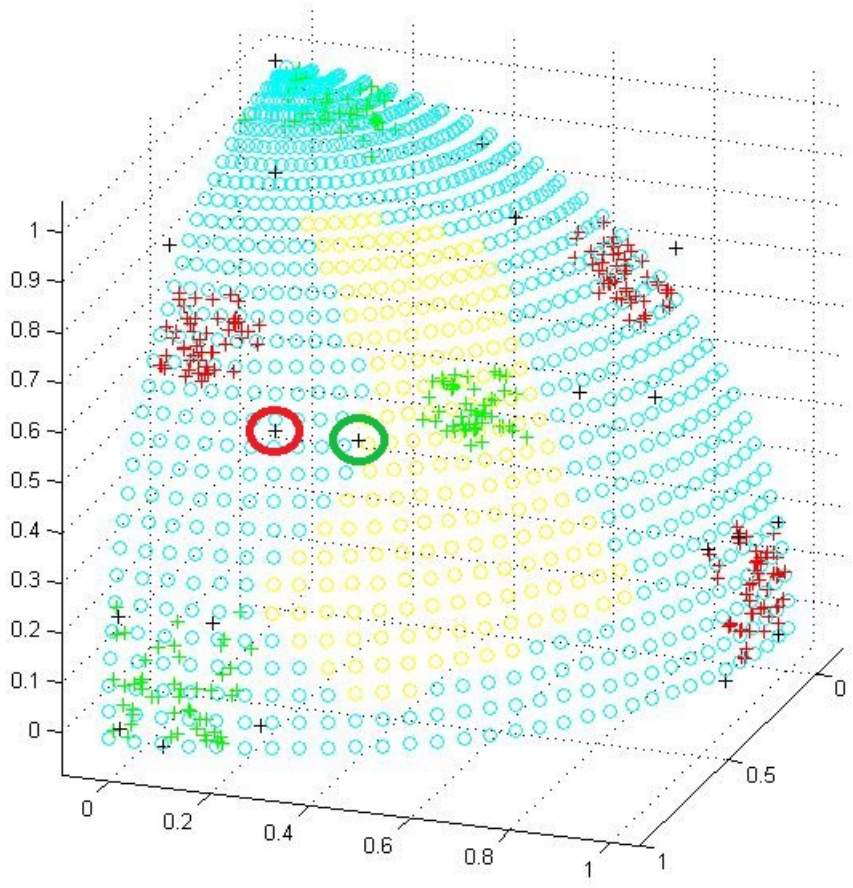


FIGURE 3.6 – La région en jaune représente l'ensemble des positions dans l'espace qui sont reconstruites avec l'atome entouré en vert et pour lesquels la contribution de l'atome entouré en rouge est nulle.

Chapitre 4

Apprentissage supervisé d'une structure multicouche de dictionnaires

Sommaire

4.1 Introduction à la structure multicouche de dictionnaires

Dans ce chapitre, nous allons présenter notre architecture multicouche de dictionnaires pour la classification. Elle repose sur l'apprentissage de manière supervisée d'un ensemble de dictionnaires dans le but d'effectuer plusieurs encodages successifs sur une image et d'obtenir des descripteurs discriminants. L'idée d'utiliser plusieurs couches de dictionnaires n'est pas nouvelle [YYH10], [CS12], [LK14]. Cependant, l'apprentissage de l'ensemble des dictionnaires de manière supervisée n'avait pas encore été réalisé. Les travaux décrits dans cette partie ont été présentés à la conférence ACIVS 2016 [CRP16].

Le but est d'exploiter les codes issus d'un encodage par un dictionnaire comme données d'entrée pour une autre couche. En employant ce procédé, on peut penser que couches après

couches les descripteurs obtenus vont contenir des informations de plus haut niveau utiles pour discriminer les images.

Très récemment, des travaux [Tar16], [Tar+16] ont été publiés sur les structures multicouches de dictionnaires. Ces travaux intitulés "Deep Dictionary Learning" correspondent d'après les auteurs à une déclinaison orientée dictionnaire de l'approche très connue du "Deep learning" ou "Apprentissage profond". L'idée est de reformuler l'équation d'apprentissage de dictionnaire classique en remplaçant le dictionnaire unique par un produit de dictionnaire :

$$\min_{\mathbf{D}^{(1)},\cdots,\mathbf{D}^{(Q)},\mathbf{x}_k} \quad \sum_{k=1}^{m} \|\mathbf{y}_k - \mathbf{D}^{(1)} \cdots \mathbf{D}^{(Q)}\mathbf{x}_k\|_2^2 \; + \phi(\mathbf{x}_k) \tag{4.1}$$

La méthode d'apprentissage proposée pour apprendre l'ensemble des dictionnaires consiste à apprendre chacun des dictionnaires indépendamment en posant des variables intermédiaires, par exemple :

$$\mathbf{z}_{1k} = \mathbf{D}^{(2)} \cdots \mathbf{D}^{(Q)}\mathbf{x}_k \tag{4.2}$$

Après substitution, on obtient alors :

$$\min_{\mathbf{D}^{(1)},\mathbf{z}_{1k}} \quad \sum_{k=1}^{m} \|\mathbf{y}_k - \mathbf{D}^{(1)}\mathbf{z}_{1k}\|_2^2 \tag{4.3}$$

Cette expression est très similaire à l'équation d'apprentissage d'un dictionnaire dans le cas monocouche.

La résolution de l'équation 4.3 permet alors d'obtenir le dictionnaire $\mathbf{D}^{(1)}$ et le code $\mathbf{z}_{1k}$.

Pour obtenir le dictionnaire $\mathbf{D}^{(2)}$, on pose une seconde variable intermédiaire :

$$\mathbf{z}_{2k} = \mathbf{D}^{(3)} \cdots \mathbf{D}^{(Q)}\mathbf{x}_k \tag{4.4}$$

On peut obtient donc une seconde équation d'apprentissage de dictionnaire dans un cas monocouche et on peut continuer ainsi pour apprendre l'ensemble des dictionnaires.

$$\min_{\mathbf{D}^{(2)},\mathbf{z}_{2k}} \quad \sum_{k=1}^{m} \|\mathbf{z}_{1k} - \mathbf{D}^{(2)}\mathbf{z}_{2k}\|_2^2, \tag{4.5}$$

Comme pour les réseaux de neurones profonds, on peut penser que cette méthode permet de définir des dictionnaires de niveaux de caractéristiques différents. Cependant, leur apprentissage n'est pas supervisé, c'est à dire qu'il ne tient pas compte de l'étiquette des images. D'autre part, cette méthode encode l'image complète, ce qui impose que l'objet d'intérêt soit

bien positionné dans l'image. Dans la méthode que nous proposons, les dictionnaires sont, quant à eux, appris de manière supervisée.

Dans le chapitre précédent, nous avons abordé l'apprentissage supervisé de dictionnaires dédiés à des tâches de classification. Dans la pratique, ces méthodes peuvent s'appliquer directement sur des images complètes comme à la section précédente. Cependant, cette manière de procéder a quelques limitations : d'une part, plus la taille de l'image augmente, plus la disparité entre images de même classe augmente et donc plus le nombre d'atomes nécessaires pour pouvoir obtenir une bonne représentation parcimonieuse de l'image va augmenter. D'autre part, le coût en calculs des opérations d'encodage (taille et nombre d'atomes) et de l'apprentissage va également augmenter. De plus, nous avons évoqué précédemment les difficultés à trouver une représentation parcimonieuse discriminante pour l'encodage d'images naturelles, en particulier lorsque l'objet d'intérêt est de taille et de position variable.

4.1.1 Décomposition en patchs

Dans ce chapitre, pour pallier au problème évoqué ci-dessus, nous proposons de décomposer l'image en un ensemble de patchs (évoqués aux chapitres 2 et 3) pour obtenir une représentation locale de l'image. La décomposition de l'image est souvent effectuée à l'aide de patchs qui se recouvrent, pour obtenir une description locale plus précise. Les signaux $\mathbf{y}_k$ considérés deviennent des patchs de taille définie a priori, présentés sous forme de vecteurs contenant les valeurs des pixels (la représentation peut être en couleurs ou en niveaux de gris). Ce type d'approche est utilisé par exemple en débruitage (chapitre 2, section 2.3.2).

Les différents patchs constituant l'image sont encodés individuellement sur le dictionnaire, ce qui donne un ensemble de codes parcimonieux. Il faut alors définir un moyen de traiter cet ensemble de patchs afin d'obtenir une représentation globale de l'image pour pouvoir s'en servir dans la classification. Il existe différentes manières d'arriver à cette fin, comme par exemple le "pooling" (ce point est détaillé dans la suite) qui fusionne les descripteurs d'une région spatiale ou encore la concaténation des descripteurs.

4.1.2 Structure multicouche

La structure que nous avons construite est la suivante : chacun des patchs de l'image est codé à l'aide d'un dictionnaire, les codes issus de cet encodage par le dictionnaire sont regroupés pour former une image à 3 dimensions : pour cela, il suffit de conserver l'information de localisation dans l'image pour chacun des patchs. Une fois cette image à 3 dimensions obtenue, on peut alors répéter le processus d'encodage de cette nouvelle image 3D en appliquant de nouveau les opérations de décomposition en patchs puis d'encodage sur un dictionnaire.

Plus formellement, soit une image $\mathbf{I} = \mathbf{Y}^{(1)}$ décomposée en un ensemble de patchs $\mathbf{y}_k^{(1)}$, $k \in [1, m_1]$. Chacun des patchs $\mathbf{y}_k^{(1)}$ est décomposé sur un dictionnaire $\mathbf{D}^{(1)}$ ce qui résulte en un ensemble de codes $\hat{\mathbf{x}}_k^{(1)}$. Ces codes $\hat{\mathbf{x}}_k^{(1)}$ sont spatialement localisés en conservant l'information

de position des patchs $\mathbf{y}_k^{(1)}$ et forment une nouvelle image en 3 dimensions $\hat{\mathbf{X}}^{(1)}$ (position dans l'espace et des coefficients d'encodage). L'image $\hat{\mathbf{X}}^{(1)}$ est ensuite présentée en entrée d'une nouvelle couche de décomposition $\mathbf{Y}^{(2)} = \hat{\mathbf{X}}^{(1)}$ (Fig. 4.1).

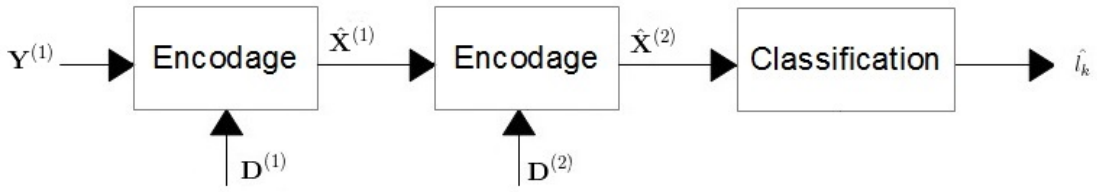


FIGURE 4.1 – Schéma d'un exemple d'encodage avec 2 couches suivies de la classification.

La figure 4.2 illustre les relations entre deux couches successives. En particulier, à l'issue de la première étape d'encodage, on obtient m_1 vecteurs de code $\hat{\mathbf{x}}_k^{(1)}$ (un pour chaque patch de l'image) de dimension K_1 égale au nombre d'atomes du dictionnaire $\mathbf{D}^{(1)}$. Ces vecteurs de code permettent de former l'image en entrée de la seconde couche $\mathbf{Y}^{(2)}$. Les patchs manipulés dans la seconde couche sont des patchs 3D de profondeur K_1.

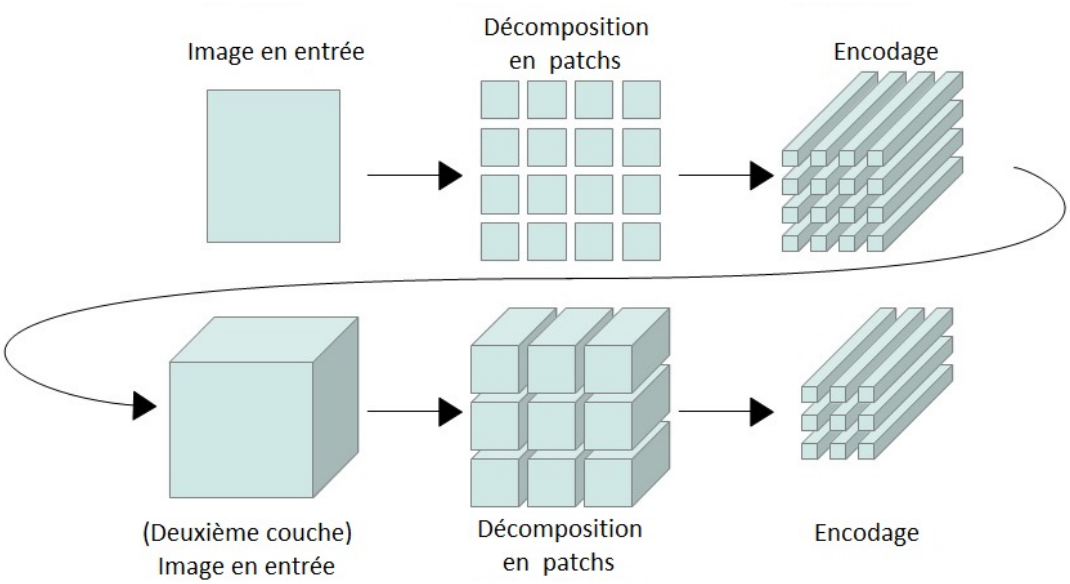


FIGURE 4.2 – Schéma d'illustration de la manipulation de deux couches d'encodage lorsque l'entrée de la seconde couche est traitée comme une image à trois dimensions.

De manière générale, l'architecture peut se décomposer de la manière suivante pour un certain nombre de couches q :

1. Une image $\mathbf{I}$ en entrée est décomposée en un ensemble de patchs $\mathbf{y}_k^{(1)}$ qui se recouvrent. Ces patchs sont ordonnés de sorte à conserver l'information de localisation spatiale.

2. Chaque $\mathbf{y}_k^{(1)}$ est encodé par un dictionnaire $\mathbf{D}^{(1)}$. Comme la localisation des patchs $\mathbf{y}_k^{(1)}$ est conservée, l'ensemble des codes $\hat{\mathbf{x}}_k^{(1)}$ peut être représenté sous la forme d'un volume 3D $\mathbf{X}^{(1)}$ de profondeur égale au nombre d'atomes dans le dictionnaire $\mathbf{D}^{(1)}$.

3. Le volume $\mathbf{X}^{(1)}$ obtenu est traité comme une image 3D et passée en entrée de la couche suivante. (1) et (2) sont répétés pour chacune des couches de l'architecture.

Pour compléter le point 3), si on considère la q-ème couche de l'architecture, on a : $\mathbf{Y}^{(q)} = \hat{\mathbf{X}}^{(q-1)}$. Cela signifie que l'ensemble des codes obtenus à la couche $q-1$ sont utilisés comme image d'entrée $\mathbf{Y}^{(q)}$ de la couche q. L'image $\mathbf{Y}^{(q)}$ est alors décomposée en patchs 3D (eux-même vectorisés $\mathbf{y}_{k_q}^{(q)}$) et des codes $\hat{\mathbf{x}}_{k_q}^{(q)}$ sont calculés grâce au dictionnaire $\mathbf{D}^{(q)}$.

Chacune des couches possède son nombre propre de patchs $k_q \in [1, m_q]$ dépendant de la taille de l'image d'entrée à la couche q ainsi que de la taille des patchs utilisés. Pour ne par alourdir la notation des indices, nous utilisons dans la suite du manuscrit pour chaque couche, l'indice k sans sous-indice. Le reste de l'information est contenu dans l'indice q de la couche. Par exemple, $\hat{\mathbf{x}}_{k_q}^{(q)}$ devient $\hat{\mathbf{x}}_k^{(q)}$.

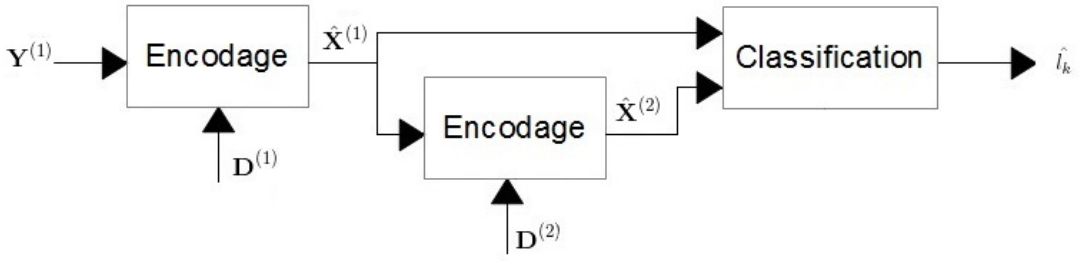


FIGURE 4.3 – Schéma d'une alternative pour l'encodage avec 2 couches suivies de la classification. Dans cet exemple, les codes issus de chaque couche sont utilisés dans l'étape de classification.

On dispose ainsi de plusieurs codes issus chacun d'une couche du système. Il s'agit maintenant de définir la façon d'exploiter ces codes par le classifieur.

Il est possible de traiter les descripteurs de différentes manières en vue d'effectuer la classification. La première méthode consiste à conserver les codes obtenus à chacune des couches intermédiaires et de les concaténer (Fig. 4.3). Cela permet de conserver des informations de chacune des couches : ces informations pourraient avoir été dégradées par la suite par un "pooling" par exemple (opérations détaillées plus loin dans le manuscrit). Cela permet aussi de faire intervenir explicitement les différentes couches dans la classification (dans le cas où le descripteur final est la concaténation des $\hat{\mathbf{x}}_k^{(q)}$). L'inconvénient de procéder ainsi est une augmentation importante de taille du descripteur final en plus d'une certaine redondance : on pose l'hypothèse que les encodages successifs permettent d'une certaine manière d'améliorer la représentation de l'image.

Dans l'architecture que nous allons présenter, on souhaite se servir uniquement des codes issus de la dernière couche en tant que descripteurs utilisés en entrée d'un algorithme de classification. En d'autres termes, tous les codes obtenus dans les couches intermédiaires à partir des différents dictionnaires $\mathbf{D}^{(q)}, q \neq Q$ n'apparaissent pas explicitement pour la classification et sont uniquement représentés par le code final.

Exemple :

Considérons une image 2D à traiter de taille 10×10 pixels en niveau de gris. Nous considérons une architecture à deux couches : la première utilisant un dictionnaire de K_1 atomes de taille 5×5 et la seconde un dictionnaire de K_2 atomes de taille 5×5. On considère pour chaque couche, une décomposition en patchs avec un décalage de 1 pixel.

1. En entrée, l'image de taille 10×10 pixels est décomposée en patchs 5×5 avec un décalage de 1 pixel, ce qui fait $6 \times 6 = 36$ patchs de $5 \times 5 = 25$ pixels.
2. Chacun de ces 6×6 patchs est encodé sur le premier dictionnaire, ce qui donne 6×6 codes de taille K_1. Ces codes sont ensuite restructurés pour former une image de taille $6 \times 6 \times K_1$.
3. L'image de taille $6 \times 6 \times K_1$ est décomposée en 2×2 patchs de taille $5 \times 5 \times K_1$.
4. Chacun de ces 2×2 patchs est alors encodé sur le second dictionnaire, ce qui donne 2×2 codes de taille K_2.

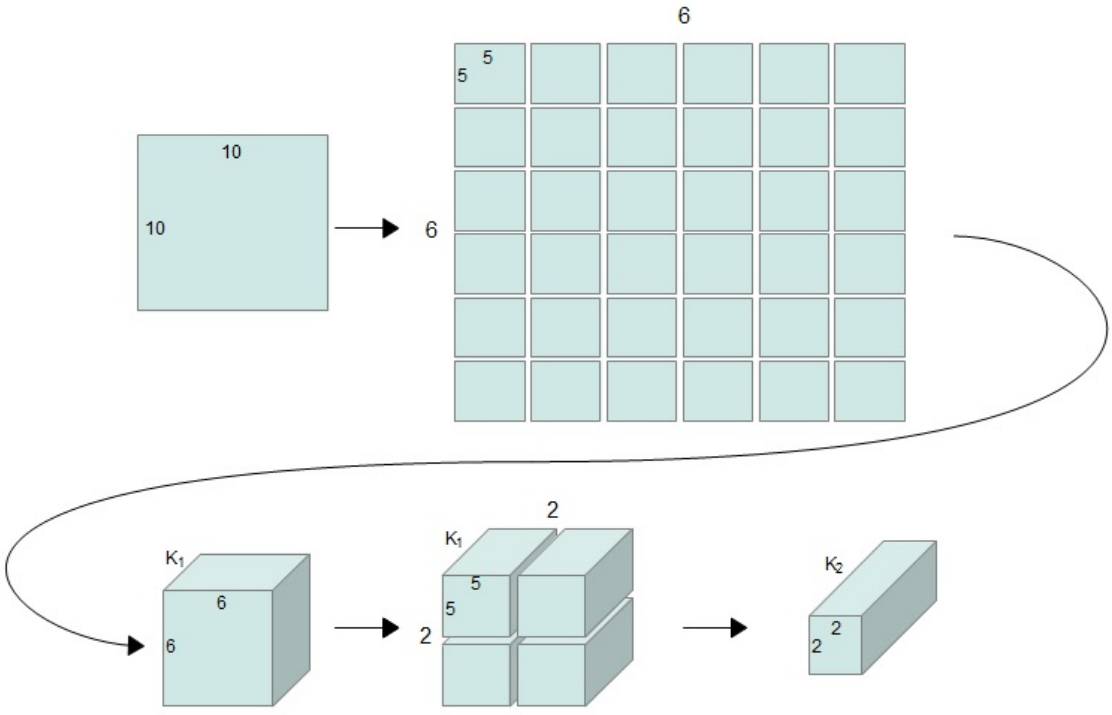


FIGURE 4.4 – Schéma d'illustration de la manipulation de deux couches d'encodage lorsque l'entrée de la seconde couche est traitée comme une image à trois dimensions.

Si le descripteur final est constitué de la concaténation de l'ensemble des codes obtenus à l'ensemble des couches, on obtient alors un vecteur de taille $6\times6\times K_1+2\times2\times K_2 = 36K_1+4K_2$. Si le descripteur final utilise uniquement la concaténation des codes de la dernière couche, par exemple, le vecteur obtenu est de taille $2 \times 2 \times K_2 = 4K_2$, et si on se place dans notre cas particulier où l'on utilise un unique code, le vecteur est de taille K_2.

Dans un choix de simplification, nous pouvons prendre le cas particulier où la sortie de la dernière couche se résume à un unique code. Dans le cas où le choix de la taille des patchs fait que l'on obtient plusieurs codes à l'issue de la dernière couche, on peut alors fusionner ces derniers (par exemple par "pooling") et se ramener au cas d'un unique code.

Les modifications algorithmiques à apporter pour la mise en place des différentes structures présentées dans cette section sont minimes. Si on choisit par exemple d'utiliser les codes obtenus à l'issue de chaque couche, on peut en pratique séparer le problème pour chacune des couches prises séparément : c'est-à-dire que dans le cas d'un problème à deux couches, pendant l'apprentissage on peut dissocier la contribution de la partie du vecteur de descripteurs correspondant à la première couche de celle correspondant à la seconde couche. On considère alors que du point de vue de la partie du descripteur final associée aux codes de la seconde couche, cette dernière est la couche "finale". De même, pour la partie du descripteur associée aux codes de la première couche, on considère également que cette dernière est la couche "finale" (puisqu'un code d'une couche donnée n'est pas influencée par les couches suivantes, mais par les couches précédentes). Il s'agit donc juste de traiter le problème d'apprentissage précédent sur des structures ayant un nombre de couches différents.

Les différences entre les tailles des vecteurs présentés dépendent du choix pour les nombres d'atomes des différents dictionnaires. Ces tailles de dictionnaires sont étudiées plus loin dans le manuscrit.

Comparativement, on peut représenter la structure proposée par Tariyal et al. [Tar16] décrite précédemment. On a représenté (Fig. 4.5) cette approche sous un format identique que celui qui décrit notre approche. Dans ce cas, la décomposition en patch est réduite à l'identité ou en décomposition en un seul patch, et cela à toutes les couches. L'approche de Tariyal et al. est donc un cas particulier de celle que l'on propose.

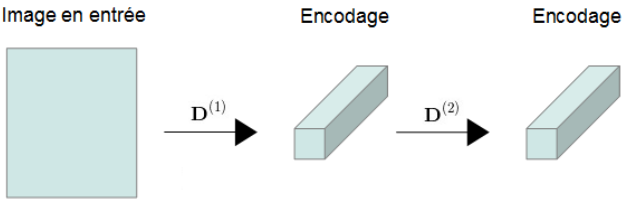


FIGURE 4.5 – Schéma de la succession d'encodage appliquée par la méthode proposée par Tariyal et al. [Tar16]

4.1.3 Traitement intermédiaire des données

Dans un premier temps, dans les sections 4.1 et 4.2.1, nous avons proposé de passer d'une couche (q) à la suivante $(q+1)$ en posant $\mathbf{Y}^{(q+1)} = \hat{\mathbf{X}}^{(q)}$, c'est-à-dire, en présentant directement en entrée de la couche $(q+1)$ l'ensemble des codes $\hat{\mathbf{X}}^{(q)}$ de la couche courante. Maintenant, nous proposons d'appliquer une transformation f sur l'image $\hat{\mathbf{X}}^{(q)}$. Ces transformations peuvent avoir différents objectifs : réduire le nombre de données/dimensions, normaliser, projeter les données. Nous pouvons utiliser n'importe quelle combinaison de fonctions de "pooling" et/ou de fonctions de traitement dans l'architecture proposée, l'unique difficulté restant l'obtention du gradient de cette fonction lors de l'optimisation.

4.1.3.1 Pooling

Le "pooling" [BPL10] sert en général à réduire le nombre de dimensions des descripteurs. En plus de la réduction de dimension, le "pooling" sert également à introduire de l'invariance à de petites translations sur les codes $\hat{\mathbf{x}}^{(q)}$ ainsi qu'aux conditions d'éclairage.

Considérons une image $\mathbf{I}$ que l'on décompose en un ensemble patchs $\mathbf{y}_k$ pouvant se recouvrir. Chacun de ces patchs est encodé sur un dictionnaire $\mathbf{D}$ donnant un ensemble de codes $\hat{\mathbf{x}}_k$. Ces patchs (et donc ces codes) sont spatialement localisés sur une grille respectant la position relative des patchs les uns par rapport aux autres. Effectuer un "pooling" de taille $(s \times s)$ sans recouvrement sur ces codes signifie que l'on regroupe localement les codes obtenus sur un ensemble de patchs adjacents dans un voisinage de taille $(s \times s)$ (voir Fig. 4.6) et qu'on les "fusionne". La fonction utilisée pour y parvenir peut être un moyennage, une fonction *max* ou encore une fonction plus complexe [BPL10]. La caractéristique commune de ces fonctions est qu'elles ne tiennent pas compte de la manière dont est ordonné l'ensemble de vecteurs traités (on peut permuter les vecteurs en entrée de la fonctions de "pooling", le résultat reste le même).

Par exemple, dans leur article, Yang et al. [YYH10] réalisent une étape d'encodage sur un dictionnaire appris de manière supervisée suivi d'une succession d'opérations de "pooling" *max*, qui servent à réduire la dimension et à obtenir une représentation de l'image à plusieurs échelles.

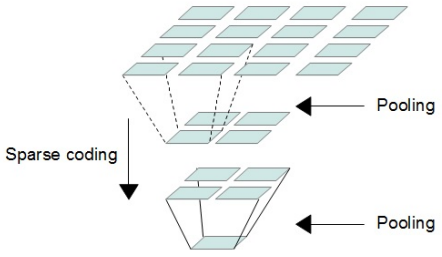


FIGURE 4.6 – Exemple de la combinaison d'opérations d'encodage et de "pooling" sur un voisinage de taille 2×2.

4.1.3.2 Autre transformation

Il est possible de modifier les codes $\hat{\mathbf{x}}^{(q)}$ de chaque couche en leur appliquant un traitement : la fonction max_+, par exemple, consiste à appliquer pour chaque valeur, la fonction $f(b) = max(b, 0)$. Cette fonction, utilisée dans notre architecture a fourni de bons résultats.

4.1.4 Structuration en patchs 2D

Dans la méthode présentée précédemment, les entrées des différentes couches sont traitées comme étant des images sous forme de matrices à 3 dimensions : par exemple l'entrée de la deuxième couche $\mathbf{Y}^{(2)}$ est constituée des K_1 cartes de descripteurs 2D correspondants aux utilisations des différents atomes du dictionnaire $\mathbf{D}^{(1)}$ dans l'image initiale.

Gérer un ensemble de patchs à 3 dimensions peut constituer un problème car la troisième dimension augmente multiplicativement la dimension des atomes 3D de chacun des dictionnaires et influence donc le choix des tailles des dictionnaires (des atomes de grandes dimensions nécessitent des dictionnaires plus grands pour être bien représentés). Cependant, au lieu de traiter des images à 3 dimensions, on peut se ramener à un ensemble d'images 2D : chaque image correspondant à la contribution d'un atome particulier du dictionnaire sur chacun des patchs. On peut alors traiter chaque carte de descripteurs 2D séparément. Pour reprendre l'exemple précédent, au lieu d'utiliser une image à 3 dimensions (de profondeur K_1 correspondant au nombre d'atomes), on la décompose en K_1 images à 2 dimensions (voir Fig. 4.7). Cette approche rappelle la méthode présentée par Bruna et Mallat [BM13] appelée "Invariant Scattering Transform" basée sur des décompositions en ondelettes successives, au lieu de la décomposition parcimonieuse à base de dictionnaires que nous utilisons.

De la même manière que pour la section 4.1.2, l'architecture présentée dans cette section peut se décomposer de la manière suivante pour un certain nombre de couches Q :

1. Une image en entrée est décomposée en un ensemble de patchs qui se recouvrent. Ces patchs sont ordonnés de sorte à conserver l'information de localisation spatiale.
2. Chaque $\mathbf{y}_k^{(1)}$ est encodé par le dictionnaire $\mathbf{D}^{(1)}$. Comme la localisation des patchs est conservée, l'ensemble des codes $\hat{\mathbf{x}}_k^{(1)}$ peut être représenté sous la forme d'un volume 3D de profondeur K_1 égale au nombre d'atomes dans le dictionnaire $\mathbf{D}^{(1)}$.
3. Chacune des K_1 cartes de descripteur est traitée comme une nouvelle image 2D et passée en entrée de la couche suivante. (1) et (2) sont répétés pour chacune des cartes 2D à chacune des couches de l'architecture.

Dans notre contexte, la décomposition en ondelettes est remplacée par la décomposition par dictionnaire, ces atomes pouvant tout aussi bien représenter des motifs particuliers au même titre que des filtres de Gabor représentent des motifs directionnels. Les transformations utilisées entre deux couches sont par exemple celles décrites à la section précédente et le changement d'échelle dans l'utilisation d'une décomposition en ondelettes peut se matérialiser ici par l'utilisation d'une fonction de "pooling".

L'avantage de procéder de la manière présentée ci-dessus est que les patchs traités à chaque couche sont uniquement des patchs en 2 dimensions. Par conséquent, une fois vectorisés, ces patchs appartiennent à un espace de dimension moins importante et donc représentables par un dictionnaire possédant moins d'atomes. Pour prendre un exemple concret : un patch de taille (5×5) est représenté par un vecteur de $\mathcal{R}^{25}$ tandis qu'un patch de taille $(5 \times 5 \times 3)$ donne un vecteur de $\mathcal{R}^{75}$. Comme on l'a vu au chapitre 2, section 2.2, plus un signal est de dimension élevée, plus le nombre d'atomes nécessaires pour une représentation parcimonieuse est important. Il faudra donc plus d'atomes pour représenter les signaux dans le second cas que dans le premier. Une réduction du nombre de dimensions permet une meilleure représentation à nombres d'atomes égaux. Cela permet donc de mieux tenir compte des motifs spécifiques dans chaque carte de descripteurs.

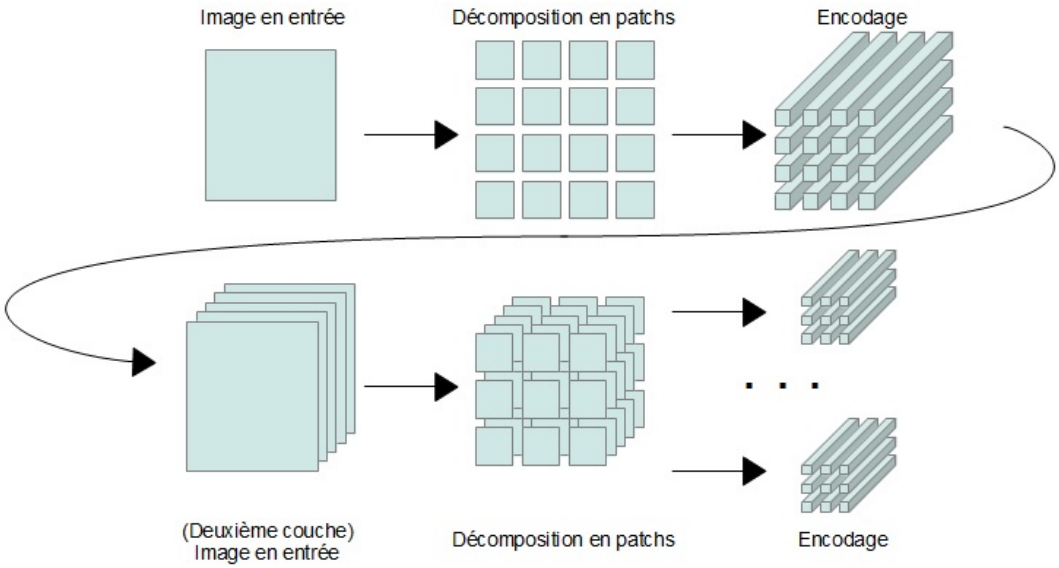


FIGURE 4.7 – Schéma d'illustration de la manipulation de deux couches d'encodage lorsque l'entrée de la seconde couche est traitée comme un paquet d'images à deux dimensions.

En revanche, l'inconvénient principal de cette manière de procéder est la multiplication des données en sortie de chaque couche : si on considère une image en deux dimensions en entrée de la première couche, on obtient K_1 (la taille de $\mathbf{D}^{(1)}$) images en deux dimensions à la sortie de celle-ci. Ces K_1 images sont ensuite présentées séparément en entrée de la seconde couche. Cela va alors produire, pour chacune de ces images, K_2 images de descripteurs : après la deuxième couche, on se retrouve donc avec $K_1 \times K_2$ cartes de descripteurs. La taille du vecteur final de descripteurs utilisé pour la classification augmente donc rapidement, ce qui limite le choix de l'architecture utilisée car on est donc contraint par le nombre de couches et le nombre d'atomes par couche pour ne pas faire exploser la taille des descripteurs.

4.1.5 Architecture choisie

Nous avons présenté différentes possibilités de construction d'architecture. Dans cette sous-section, nous allons résumer les choix d'architectures effectués.

Nous avons décidé d'utiliser l'architecture décrite à la sous-section 4.1.2 en prenant le cas particulier où la sortie de la dernière couche est un unique vecteur de descripteurs. Cela nous contraint donc dans la plupart des cas à adapter l'architecture aux données à traiter. La raison de ce choix est de proposer au classifieur final des descripteurs plus discriminants et plus complexes en cascadant des décompositions parcimonieuses par dictionnaires. Dans cet objectif, l'utilisation par exemple des codes obtenus dans les couches intermédiaires peut être considérée comme redondante par rapport aux codes finaux.

Nous avons choisi d'utiliser la structure basée sur les images et les patchs 3D. En effet, l'utilisation de la structure reposant sur les patchs 2D conduit à une explosion dans la taille des descripteurs ce qui ajoute une contrainte forte dans le dimensionnement de la structure.

Entre les deux couches, les transformations choisies sont un opérateur de "pooling" (par moyenne) et la fonction max_+. Les fonctions de "pooling" souvent utilisées sont la fonction moyenne et la fonction max, le choix de la fonction moyenne a été fait pour simplifier les calculs dans l'étape de rétropropagation. En effet, dans le cas de la fonction max, il est nécessaire de conserver des informations supplémentaires comme la position du maximum et de gérer les cas particuliers dans lesquels il y a plusieurs maxima. La fonction max_+ correspond à appliquer la fonction $max_+(x) = max(x, 0)$. On applique une telle fonction avant la fonction de "pooling", pour éviter de moyenner des valeurs de signes différents. La fonction valeur absolue est également possible, mais max_+ fournit empiriquement de meilleurs résultats.

Le classifieur et la fonction de coût choisis sont un classifieur linéaire couplé à la fonction softmax et l'entropie croisée présentés précédemment.

4.2 Apprentissage

Quelle que soit la structure adoptée, il s'agit maintenant d'apprendre les dictionnaires des différentes couches ainsi que les paramètres afin d'optimiser la classification. Pour cela, nous allons adapter la démarche de l'apprentissage conjoint entre un dictionnaire et un classifieur présentée au chapitre 3 et l'étendre à une application de plusieurs dictionnaires sur plusieurs couches.

4.2.1 Formulation

Considérons une architecture contenant Q couches et donc Q dictionnaires $\mathbf{D}^{(1)}, \cdots, \mathbf{D}^{(Q)}$. On considère de nouveau une fonction de coût $\mathcal{L}$ permettant de mesurer l'erreur de classification. On cherche alors à minimiser la fonction de coût $\mathcal{L}$ par rapport à chacun des dictionnaires

$\mathbf{D}^{(1)}, \cdots, \mathbf{D}^{(Q)}$, et par rapport aux paramètres $\mathbf{W}$ du classifieur.

$$\min_{\mathbf{D}^{(1)},\cdots,\mathbf{D}^{(Q)},\mathbf{W}} \quad \frac{1}{m}\sum_{k=1}^{m} \mathcal{L}(l_k, \hat{\mathbf{x}}_k^{(Q)}, \mathbf{W}) \tag{4.6}$$

où m désigne le nombre d'images dans la base d'apprentissage. Dans l'architecture choisie, seul le dernier code $\hat{\mathbf{x}}_k^{(Q)}$ (indice Q) sert pour le classification.

Pour cela, il est nécessaire de pouvoir calculer les gradients :
$\nabla_{\mathbf{D}^{(1)}}\mathcal{L}(l_k, \hat{\mathbf{x}}_k^{(Q)}, \mathbf{W}), \cdots \nabla_{\mathbf{D}^{(Q)}}\mathcal{L}(l_k, \hat{\mathbf{x}}_k^{(Q)}, \mathbf{W})$ et $\nabla_{\mathbf{W}}\mathcal{L}(l_k, \hat{\mathbf{x}}_k^{(Q)}, \mathbf{W})$. Pour parvenir à cet objectif, nous allons utiliser l'algorithme de rétropropagation évoqué au chapitre 2.

La difficulté supplémentaire pour le calcul de ces gradients est que la fonction de coût $\mathcal{L}(l_k, \hat{\mathbf{x}}_k^{(Q)}, \mathbf{W})$ est évaluée à partir du code $\hat{\mathbf{x}}_k^{(Q)}$ de la dernière couche. Par conséquent, pour évaluer le gradient de $\mathcal{L}$ par rapport au dictionnaire d'une couche intermédiaire $\mathbf{D}^{(q)}$, il faut effectuer la rétropropagation à travers plusieurs étapes d'encodage sur des dictionnaires (Eq. 4.11) et par rapport aux éventuelles transformations f appliqués aux codes $\hat{\mathbf{X}}^{(q)}$.

4.2.2 Méthode d'apprentissage

Nous nous appuyons sur les travaux de Mairal et al. [MBP12] présentés au chapitre 2 que nous étendons à la structure multicouche.

Pour utiliser l'algorithme de rétropropagation, le gradient est calculé de la même manière qu'à la section 3.3.1 en remplaçant $\hat{\mathbf{x}}_k$ par $\hat{\mathbf{x}}_k^{(q)}$ et $\mathbf{y}_k$ par $\mathbf{y}_k^{(q)}$: on rétropropage le gradient sur l'ensemble de couches, de la couche Q à la couche $q = 1$.

Pour un couple image-label $(\mathbf{Y}, l_k)$, le gradient $\nabla_{\mathbf{D}^{(Q)}}\mathcal{L}(l_k, \hat{\mathbf{x}}_k^{(Q)}, \mathbf{W})$ par rapport à la dernière couche est calculé par les équations 3.7 et 4.11. En respectant la notation utilisée pour distinguer les différentes couches, l'équation pour la dernière couche devient :

$$\nabla_{\mathbf{D}^{(Q)}}\mathcal{L}(l_k, \hat{\mathbf{x}}_k^{(Q)}, \mathbf{W}) = \mathbf{D}^{(Q)}\beta\hat{\mathbf{x}}_k^{(Q)\top} + (\mathbf{y}_k^{(Q)} - \mathbf{D}^{(Q)}\hat{\mathbf{x}}_k^{(Q)})\beta^\top \tag{4.7}$$

avec β défini par :

— pour les indices contenus dans l'ensemble Λ (indices pour lesquels $\hat{\mathbf{x}}_k^{(Q)}$ est non-nul),

$$\beta_\Lambda = (\mathbf{D}_\Lambda^{(Q)\top}\mathbf{D}_\Lambda^{(Q)})^{-1}\nabla_{\hat{\mathbf{x}}_{k\Lambda}^{(Q)}}\mathcal{L}(l_k, \hat{\mathbf{x}}_k^{(Q)}, \mathbf{W}) \tag{4.8}$$

— $\beta_j = 0$, si $j \notin \Lambda$.

Pour calculer le gradient de la fonction de coût $\mathcal{L}(l_k, \hat{\mathbf{x}}_k^{(Q)}, \mathbf{W})$ par rapport au dictionnaire $\mathbf{D}^{(q)}$ de la q-ème couche en utilisant l'algorithme de rétropropagation, on effectue :

$$\nabla_{\mathbf{D}^{(q)}}\mathcal{L}(l_k, \hat{\mathbf{x}}_k^{(Q)}, \mathbf{W}) = \mathbf{D}^{(q)}\beta\hat{\mathbf{x}}_k^{(q)\top} + (\mathbf{y}_k^{(q)} - \mathbf{D}^{(q)}\hat{\mathbf{x}}_k^{(q)})\beta^\top \tag{4.9}$$

$$\beta_\Lambda = (\mathbf{D}_\Lambda^{(q)\top}\mathbf{D}_\Lambda^{(q)})^{-1}\nabla_{\hat{\mathbf{x}}_{k\Lambda}^{(q)}}\mathcal{L}(l_k, \hat{\mathbf{x}}_k^{(Q)}, \mathbf{W}) \tag{4.10}$$

où chacun des codes $\hat{\mathbf{x}}_k^{(q)}$ à une couche q donnée est obtenu en résolvant :

$$\hat{\mathbf{x}}_k^{(q)} = \underset{\mathbf{x}}{\operatorname{argmin}} \quad \|\mathbf{y}_k^{(q)} - \mathbf{D}^{(q)}\mathbf{x}\|_2^2 + \lambda_q\|\mathbf{x}\|_1 \tag{4.11}$$

Dans l'équation 4.10, on voit apparaître le gradient de $\mathcal{L}$ par rapport à $\hat{\mathbf{x}}_k^{(q)}$: il faut calculer $\nabla_{\hat{\mathbf{x}}_{k\Lambda}^{(q)}}\mathcal{L}(l_k, \hat{\mathbf{x}}_k^{(Q)}, \mathbf{W})$. Le calcul de ce gradient peut se décomposer de la manière suivante :

$$\frac{\partial\mathcal{L}}{\partial\hat{\mathbf{x}}^{(q)}} = \frac{\partial\mathcal{L}}{\partial\hat{\mathbf{x}}^{(q+1)}}\frac{\partial\hat{\mathbf{x}}^{(q+1)}}{\partial\mathbf{y}^{(q+1)}}\frac{\partial\mathbf{y}^{(q+1)}}{\partial\hat{\mathbf{x}}^{(q)}} \tag{4.12}$$

où $\frac{\partial\hat{\mathbf{x}}^{(q)}}{\partial\mathbf{y}^{(q)}}$:

$$\frac{\partial\hat{\mathbf{x}}^{(q)}}{\partial\mathbf{y}^{(q)}} = (\mathbf{D}^{(q)\top}\mathbf{D}^{(q)})^{-1}\mathbf{D}^{(q)} \tag{4.13}$$

Le calcul de $\frac{\partial\mathbf{y}^{(q+1)}}{\partial\hat{\mathbf{x}}^{(q)}}$ peut être décomposé en deux étapes, $\{\mathbf{y}_k^{(q)}\} = g(\mathbf{Y}^{(q)})$ (où g correspond à la fonction qui prend en entrée une image et renvoie un ensemble de patchs), et $\mathbf{Y}^{(q+1)} = f(\hat{\mathbf{X}}^{(q)})$. Le premier calcul correspond à "inverser" la décomposition en patchs, c'est-à-dire assembler les patchs en une image en les empilant avec le même décalage que pendant la décomposition (Fig. 4.8). Les patchs sont repositionnés en respectant le décalage utilisée lors de la décomposition. Les régions où deux patchs se recouvrent contiennent la somme des valeurs d'erreur de la rétropropagation associées à chacun des patchs (voir Fig. 4.4, pour un exemple plus précis, on peut se référer aux fonction *im2col* et *col2im* de MATLAB).

Le second calcul correspond à chercher le gradient de la transformation qui vaut l'identité dans le cas simple où $\mathbf{Y}^{(q+1)} = \hat{\mathbf{X}}^{(q)}$ et let gradient de f dans les autres cas.

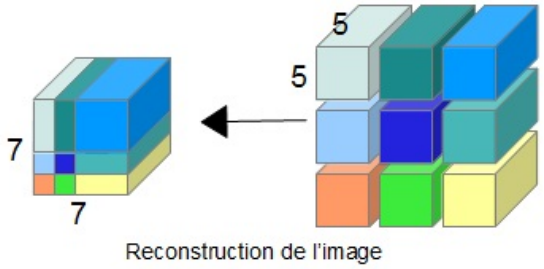


FIGURE 4.8 – Schéma d'illustration de l'étape de reconstruction dans le cas d'un décalage de 1 pixel. Sur cet exemple, les patchs 5×5 sont empilés avec un décalage de 1 pixel pour donner une image 3D de taille 7×7. (Image à voir en couleurs)

Calcul des gradients par rapport aux dictionnaires :

a) Entrées : une image $\mathbf{Y}^{(1)}$ associée à un label l, les dictionnaires $\mathbf{D}^{(1)}, \cdots, \mathbf{D}^{(Q)}, \mathbf{W}$.

b) On calcule le descripteur final $\hat{\mathbf{x}}_k^{(Q)}$ utilisé pour la classification à partir des étapes décrites à la section 4.1.2.

c) On calcule $\nabla_{\hat{\mathbf{x}}_{k\Lambda}^{(Q)}} \mathcal{L}(l_k, \hat{\mathbf{x}}_k^{(Q)}, \mathbf{W})$.

d) On peut alors calculer $\nabla_{\mathbf{D}^{(Q)}} \mathcal{L}(l_k, \hat{\mathbf{x}}_k^{(Q)}, \mathbf{W})$ à partir des équations 4.7 et 4.8.

e) Rétropropagation d'une couche $q+1$ à la couche q :

1. Pour chacun des codes $\hat{\mathbf{x}}_k^{(q+1)}$ de la couche $q+1$, on propage l'erreur jusqu'aux patchs $\frac{\partial \mathcal{L}}{\partial \hat{\mathbf{x}}^{(q+1)}} \frac{\partial \hat{\mathbf{x}}^{(q+1)}}{\partial \mathbf{y}^{(q+1)}}$ à partir de l'équation 4.13.
2. Pour passer à la couche précédente, il faut d'abord effectuer l'opération "inverse" de la décomposition, c'est-à-dire la reconstruction de l'image $\mathbf{Y}^{(q+1)}$ à partir de l'ensemble de patchs, et calculer la dérivée de la transformation f constituée de la composition entre une fonction de "pooling" et la transformation max_+ (Eq. 4.12).
3. à partir de $\nabla_{\hat{\mathbf{x}}_{k\Lambda}^{(q)}} \mathcal{L}(l_k, \hat{\mathbf{x}}_k^{(Q)}, \mathbf{W})$, on peut alors calculer $\nabla_{\mathbf{D}^{(q)}} \mathcal{L}(l_k, \hat{\mathbf{x}}_k^{(Q)}, \mathbf{W})$ avec les équations 4.9 et 4.10.

f) On répète les opérations pour chacune des couches.

Dans le cas où l'on adopte la structuration basée sur des patchs 2D, les gradients s'obtiennent de la manière décrite précédemment, en dissociant les calculs pour chacune des cartes de descripteurs (voir Fig. 4.7). L'étape du passage des patchs vers l'image reconstituée s'effectue donc indépendamment aussi pour chaque carte de descripteurs. Pour la structure reposant sur les patchs 2D, il faut ajouter une seconde étape de "reconstruction" qui consiste à empiler les cartes de descripteurs (Fig. 4.9) : les patchs 2D sont repositionnés en respectant le décalage utilisée lors de la décomposition. Les régions où deux patchs se recouvrent contiennent la somme des valeurs d'erreurs (de la rétro-propagation) associées à chacun des patchs. Cette étape est effectuée pour chacune des cartes de descripteurs. Ces "images" 2D de l'erreur sont ensuite regroupées pour reformer l'image 3D, de même dimension que l'entrée de la couche courante (Voir Fig. 4.7).

4.3 Paramétrisation

Dans l'architecture présentée, il faut choisir un certain nombre de paramètres. Nous allons discuter dans cette section de l'influence des choix effectués.

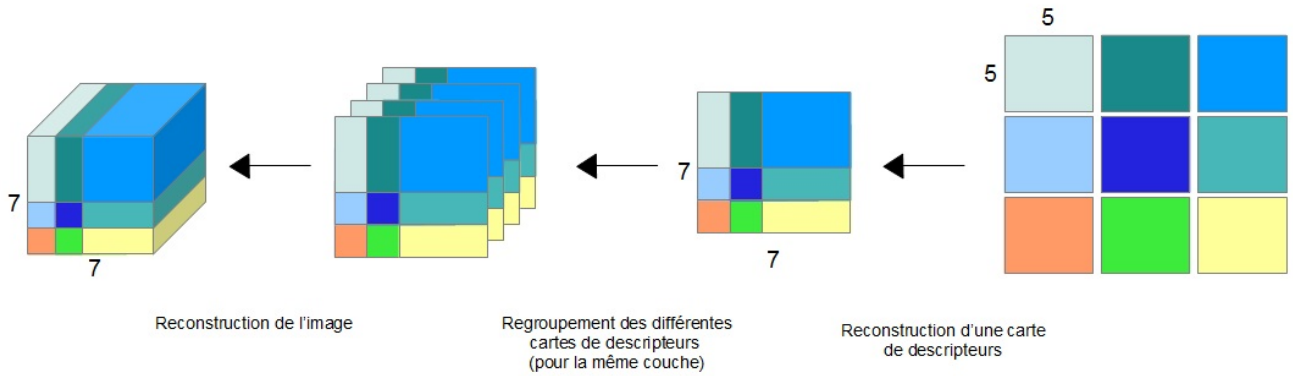


FIGURE 4.9 – Schéma d'illustration de l'étape de reconstruction dans le cas d'une structure 2D. Dans cet exemple 3×3 patchs de dimension 5×5 sont regroupés avec un décalage de 1 pixel dans une image 7×7. L'opération est effectuée pour chacune des cartes de descripteurs qui sont ensuite regroupés pour former une image 3D. (Image à voir en couleurs)

4.3.1 Taille des patchs et des atomes

La taille des atomes influence les motifs qui vont être appris par ceux-ci. Par exemple, des atomes de petite taille (patch 5×5) apprendront des motifs plus simples tandis que des atomes plus grands peuvent apprendre des motifs plus "riches". La figure 4.10 montre des dictionnaires appris sur des images issues de la base de données MNIST [LeC+98b], composées d'images de chiffres écrits de manière manuscrite (Fig. 4.11). Les images ont une taille de 28×28 pixels. Sur le premier dictionnaire, possédant des atomes de taille 5×5, on observe d'une part des motifs qui correspondent aux contours des chiffres (fond noir avec des éléments non-nuls sur les bordures) et d'autres part des motifs pour représenter les courbures. On rappelle aussi que la décomposition d'un patch sur un dictionnaire est signée : par conséquent, il faut également considérer le "négatif" de chaque patch dans l'ensemble des motifs disponibles. Comme on peut s'y attendre, le deuxième dictionnaire de patchs 7×7 contient des versions plus résolues des motifs du premier dictionnaire, ce qui permet une meilleure représentation des détails tels que les motifs courbés.

4.3.2 Nombre d'atomes

Le nombre d'atomes de chaque dictionnaire est un paramètre important de l'architecture. Dans un contexte de reconstruction, le nombre d'atomes influe sur la capacité à représenter fidèlement un signal par un code parcimonieux. C'est pourquoi il est habituel, dans ce contexte, d'avoir un nombre d'atomes supérieur à la dimension du signal à représenter : on parle de dictionnaire "*overcomplete*". Nous rappelons que de manière empirique, pour un nombre fixé de coefficients non-nuls dans la représentation, l'erreur de reconstruction diminue lorsque le nombre d'atomes dans le dictionnaire augmente.

Dans le contexte de classification avec notre architecture et donc d'apprentissage supervisé,

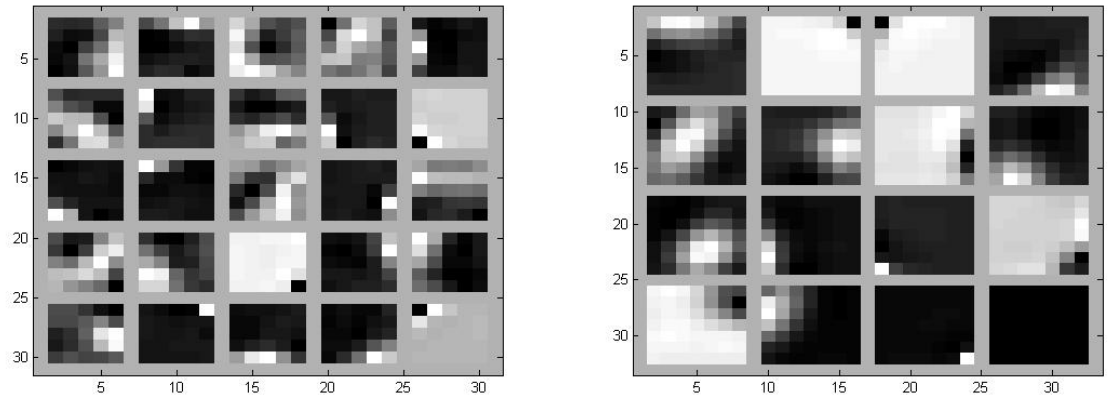


FIGURE 4.10 – Exemples d'atomes dictionnaires appris sur MNIST avec des tailles d'atomes différentes. Le dictionnaire de gauche a des atomes de 5×5 pixels tandis que celui de droite a des atomes de 7×7.

les dimensions des dictionnaires aux différentes couches sont liées, ce qui limite le choix dans la taille des dictionnaires. En effet, comme l'image en entrée de la couche $q+1$ correspond aux codes en sortie de la couche q, la dimension des signaux utilisés à la couche $q+1$ dépendent directement du nombre d'atomes du dictionnaire à la couche q (nous rappelons que ce nombre d'atomes K_q correspond à la profondeur de l'image en sortie de la couche q). Par ailleurs, le nombre de dimensions des signaux (patchs) à traiter est également directement lié à la taille utilisée pour les patchs : la dimension du signal vaut $longeur \times largeur \times profondeur$.

L'algorithme présenté précédemment nécessite le calcul, à chaque couche des codes $\hat{\mathbf{x}}_k^{(q)}$ par l'algorithme LARS [Efr+04] (Eq. 4.11) ainsi que les calculs $\nabla_{\mathbf{D}^{(q)}} \mathcal{L}(l_k, \hat{\mathbf{x}}_k^{(Q)}, \mathbf{W})$. Or, ces calculs nécessitent un temps lié à la dimension des données traitées à chaque couche. Nous avons vu que celles-ci correspondent, d'une part au nombre de patchs et d'autre part au nombre d'atomes K_q.

Soient un signal $\mathbf{y} \in \mathcal{R}^n$ et un dictionnaire $\mathbf{D} \in \mathcal{R}^{(n \times K)}$. D'après Efron [Efr+04], le coût en calcul d'une opération d'encodage est $O(K^3 + nK^2)$ et dépend très fortement du nombre d'atomes K. Dans ces conditions, il est difficile de se placer dans le cas "*overcomplete*" ($K > n$) au risque de voir croître rapidement le nombre d'atomes dans chacune des couches. La limitation est essentiellement due à l'explosion des temps de calculs : la recherche des codes parcimonieux s'effectue une fois pour chacun des patchs de chaque couche. Par conséquent, lors des expérimentations, on se trouve alors souvent dans un cas où le nombre d'atomes est inférieur à la dimension du signal.

4.3.3 Nombre de couches

Dans nos expérimentations, le nombre de couches est déterminé en fonction de taille des images à traiter. Dans l'architecture utilisée, la sortie est fixée à un unique vecteur utilisé

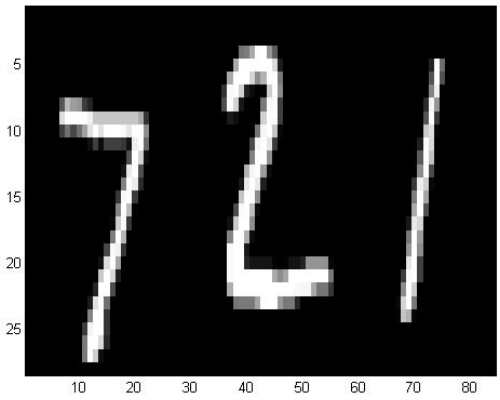


FIGURE 4.11 – Exemples de 3 images de la base MNIST [LeC+98b]. Chaque chiffre est une image différente.

pour la classification. C'est pourquoi un exemple de structure proposée est constituée de 3 couches utilisant des patchs de taille 5×5 décalés de 1 pixel et séparées par des opérations de "pooling" de taille 2×2. Les tailles sont calculées en considérant une décomposition en patchs avec un décalage de 1 pixel. On obtient alors les sorties suivantes :

— On considère une entrée : $32 \times 32 \times 3$ (image couleur)

— Sortie de la couche 1 : $28 \times 28 \times K_1$ (K_1 est le nombre d'atomes de $\mathbf{D}^{(1)}$)

— Pooling 1 : $14 \times 14 \times K_1$

— Sortie de la couche 2 : $10 \times 10 \times K_2$

— Pooling 2 : $5 \times 5 \times K_2$

— Sortie de la couche 1 : $1 \times 1 \times K_3$

Le choix de décalage de 1 pixel est le plus petit possible et également le plus coûteux en calculs. Cependant, c'est aussi celui qui fournit en général les meilleurs résultats : les effets du choix de différentes valeurs de décalage pour la classification sont présentés par Coates et al. [CLN11].

Pour l'exemple présenté, il est possible d'augmenter le nombre de couches en remplaçant une couche d'encodage sur des patchs 5×5 par deux couches d'encodage sur des patchs 3×3 ou encore en retirant une des deux opérations de "pooling". Cependant, même si nous avons constaté que l'ajout de plusieurs couches fournit de meilleurs résultats par rapport à une couche unique, nous avons aussi remarqué expérimentalement qu'ajouter un nombre élevé de couches n'améliore pas systématiquement les performances.

4.3.4 Fonctions de régularisation

Une présentation des fonctions de régularisation a été faite au chapitre 2. Dans cette sous-section, nous allons discuter des avantages et inconvénients liés au choix de la fonction de régularisation et constatés sur notre architecture.

Tout d'abord, nous rappelons que le choix de la norme ℓ_1 dans l'équation 4.11 a été fait pour ses propriétés de parcimonie et parce que son utilisation en optimisation, dans des algorithmes de descente du gradient est bien maîtrisée (davantage qu'avec la pseudo-norme ℓ_0). Cependant, ce choix amène également quelques désavantages. Par exemple, dans le cas où le nombre d'atomes du dictionnaire est supérieur à la dimension des signaux, la solution du problème de décomposition n'est pas unique. Imaginons qu'un certain signal $\mathbf{y}$ soit encodé sur un dictionnaire $\mathbf{D}$ en utilisant LASSO (une régularisation ℓ_1) et que la décomposition utilise un atome $\mathbf{d}_j$. Si on introduit dans le dictionnaire $\mathbf{D}$ un atome $\mathbf{d}_i$ strictement égal à $\mathbf{d}_j$ alors n'importe quelle combinaison $x_j(1-\alpha)\mathbf{d}_j + x_j\alpha\mathbf{d}_i$ est aussi solution.

Un autre inconvénient de l'utilisation d'un terme ℓ_1 [MBP12] est que le gradient de la fonction de coût $\mathcal{L}$ en fonction de $\hat{\mathbf{x}}_k$ n'est pas défini aux endroits où le code $\hat{\mathbf{x}}_k$ change de signe. En pratique, cela ne nuit pas à la convergence (on utilise donc les équations données précédemment) et à l'apprentissage car les occurrences sont rares devant la quantité de signaux traités.

Dans le cas d'une régularisation ℓ_2, le problème mentionné n'apparait pas : si $\mathbf{d}_i$ et $\mathbf{d}_j$ sont strictement égaux, les coefficients optimaux pour ces deux atomes sont égaux. Cependant, comme évoqué au chapitre 2, le norme ℓ_2 n'apporte pas de parcimonie : si on pénalise fortement la norme ℓ_2 d'un vecteur, ses coefficients deviennent faibles mais restent non-nuls.

Dans le cas de dictionnaires possédant des atomes fortement corrélés, il est conseillé d'utiliser la fonction "Elastic net" [ZH05] qui se présente comme la combinaison linéaire d'une norme ℓ_1 et d'une norme ℓ_2. L'avantage est alors de conserver la qualité de parcimonie de la norme ℓ_1 tout en résolvant le problème d'ambiguïté dans le cas d'atomes corrélés.
Pour utiliser cette fonction de régularisation, il suffit de modifier le calcul de β_Λ (Eq. 4.10) de la manière suivante :

$$\beta_\Lambda = (\mathbf{D}_\Lambda^{(q)\top}\mathbf{D}_\Lambda^{(q)} + \lambda_2\mathbf{I})^{-1}\nabla_{\hat{\mathbf{x}}_{k\Lambda}^{(q)}}\mathcal{L}(l_k, \hat{\mathbf{x}}_k^{(Q)}, \mathbf{W}) \tag{4.14}$$

où λ_2 correspond à la constante associée à la norme ℓ_2 de la fonction de régularisation "Elastic net" $\phi(\mathbf{x}) = \lambda_1\|\mathbf{x}\|_1 + \frac{\lambda_2}{2}\|\mathbf{x}\|_2^2$.

4.3.5 Choix de la valeur de λ

Dans cette section, nous allons discuter du paramètre λ (Eq. 4.11) qui contrôle la parcimonie : plus la valeur de ce paramètre est élevée, plus la proportion de coefficients nuls dans les codes $\hat{\mathbf{x}}_k^{(q)}$ augmente. Dans notre architecture, la parcimonie des codes est une propriété

désirée et une proportion élevée de coefficients nuls accélère les étapes d'apprentissages (voir section 4.3.2 et Eq. 4.10). Intuitivement, ce paramètre influence aussi la distance à laquelle un atome peut intervenir dans la représentation d'un signal : si on utilise une valeur λ forte, seuls les atomes fortement corrélés avec le signal seront utilisés.

Par ailleurs, il faut noter que nous travaillons dans des espaces de grandes dimensions et que les dictionnaires utilisés dans nos travaux sont souvent "*undercomplete*", pour des raisons de coûts de calculs. Cela a plusieurs conséquences, d'une part les signaux considérés ne couvrent qu'une petite proportion de l'espace de grandes dimensions et ensuite les atomes doivent être capables de représenter l'intégralité des signaux. Des signaux encodés par le vecteur nul (aucun atome à proximité) n'ont aucune influence sur l'apprentissage : ces signaux ont de grandes chances de rester encodés par le vecteur nul.

4.4 Conclusion

Dans ce chapitre, nous avons exploré un ensemble de structures multicouches basées sur l'utilisation successive de dictionnaires pour encoder une image ou les patchs d'une image. Chacune des couches prend en entrée une image et produit une image à 3 dimensions correspondant à un encodage parcimonieux local sur un dictionnaire. L'ensemble des dictionnaires utilisés (1 par couche) est appris de manière supervisée en optimisant une fonction de coût par descente de gradient. Les calculs des différents gradients nécessaires repose sur les travaux de Mairal et al. [MBP12], initialement réalisés pour fonctionner sur un unique dictionnaire et adaptés pour fonctionner sur notre architecture multicouche.

Le choix des différents paramètres de l'architecture est traité à la section 4.3 et validé empiriquement dans les expérimentations. Cependant, l'interprétation de ces choix reste souvent intuitive et difficile à prouver.

Dans le chapitre suivant, nous abordons les différents tests réalisés sur des bases de données publiques pour valider notre architecture.

Chapitre 5

Experimentations

Sommaire

Dans cette section, nous présentons les tests qui ont été réalisés pour étudier les performances de l'architecture décrite au chapitre précédent. Ces tests sont effectués sur les bases de données pour la tâche de classification d'images MNIST [LeC+98b], CIFAR-10 [KH09] et STL-10 [CLN11]. L'architecture proposée a été implémentée sous MATLAB et nous nous sommes appuyés sur la toolbox SPAMS [Mai+09 ; Mai+10] pour l'implémentation de l'algorithme LARS (encodage d'un signal sur un dictionnaire).

L'utilisation de MATLAB nous a posé des problèmes de temps de calculs. En particulier, les étapes d'encodage à effectuer pour chacun des patchs à chacune des couches sont coûteuses. De plus, chacune de ces opérations est sensible à la taille des dictionnaires choisis.

5.1 Description des expériences

A partir de la base d'apprentissage et pour une structure donnée, il s'agit d'apprendre le ou les dictionnaires, les paramètres du classifieur ainsi que les encodages optimaux des différents éléments de la base.
Nous rappelons que l'apprentissage repose sur les idées de la rétropropagation. De manière générale, on procède de la manière suivante : pour un ensemble de signaux sélectionnés, on calcule les représentations des signaux pour chaque couche de dictionnaires et les code finaux en sortie de l'architecture. On utilise ensuite la fonction de coût (ici, l'entropie croisée) qui donne une erreur entre les codes obtenus et les codes optimaux pour la classification. Le modèle choisi est une classification linéaire. L'erreur est ensuite répercutée sur les dictionnaires à travers le calcul des gradients.

Pour l'algorithme d'optimisation, nous avons opté pour une descente de gradient stochastique [Bot10]. A la différence d'une descente de gradient classique (dite par batch) dans laquelle le gradient est calculé pour l'ensemble de la base d'apprentissage à chaque itération, la descente de gradient stochastique consiste à tirer aléatoirement un élément de l'ensemble d'apprentissage et de réaliser une itération de descente pour cet élément. L'avantage de cette approche est donc de pouvoir effectuer un plus grand nombre d'itérations en limitant les calculs pour chaque itération.

Cependant, dans la descente de gradient stochastique, la direction du gradient pour deux itérations successives est susceptible de varier beaucoup. C'est pourquoi, dans la pratique, on utilisera de petits groupes d'échantillons (de l'ordre de la dizaine) au lieu d'un échantillon unique pour "lisser" la direction de descente.

En terme d'implémentation, les signaux de l'ensemble d'apprentissage sont mélangés aléatoirement. Ensuite, à chaque itération de l'algorithme d'apprentissage, un petit nombre b de signaux sont sélectionnés selon leur ordre d'apparition [Bot12] : à la première itération, on utilise les b premiers signaux (après le mélange aléatoire), à la seconde itération les b suivants, etc. On parcourt ainsi l'ensemble d'apprentissage de manière cyclique.

Il existe différentes méthodes pour fixer le pas d'apprentissage η de la descente de gradient : il est par exemple possible de donner à η une décroissance en $\frac{1}{t}$ [Bot12] où t désigne le nombre d'itérations. Le pas d'apprentissage η utilisé pour la descente de gradient dans l'algorithme proposé décroit périodiquement après un certain nombre de cycles [Spr+15] : il est maintenu constant pendant 2 ou 3 cycles puis il est ensuite divisé par 2.

Pour les tests présentés par la suite, le calcul des gradients au sein de l'implémentation a été vérifié en utilisant les différences finies. Le classifieur utilisé en apprentissage est un classifieur linéaire couplé à la fonction softmax et la fonction de coût associée à ce classifieur est l'entropie croisée (Chapitre 3, Section 3.3.2).

5.2 Base d'images MNIST

La base d'images MNIST [LeC+98b] comporte 60000 images de taille 28×28 pixels et 10 classes, et représentent les dix chiffres, écrits de manière manuscrite puis numérisés. Ces images en niveaux de gris sont réparties de manière prédéterminée et équilibrée en 50000 images d'apprentissage et 10000 images de tests. Cette base de données a certaines caractéristiques : les chiffres sont en blanc sur un fond uniforme noir (valeurs des pixels à 0) et ils sont relativement centrés. Ces conditions sont favorables à un apprentissage de dictionnaires directement sur l'intégralité de l'image. Les tests qui suivent ont été réalisés sur les bases d'apprentissage et de tests sans appliquer de méthodes d'augmentation sur la base de données.

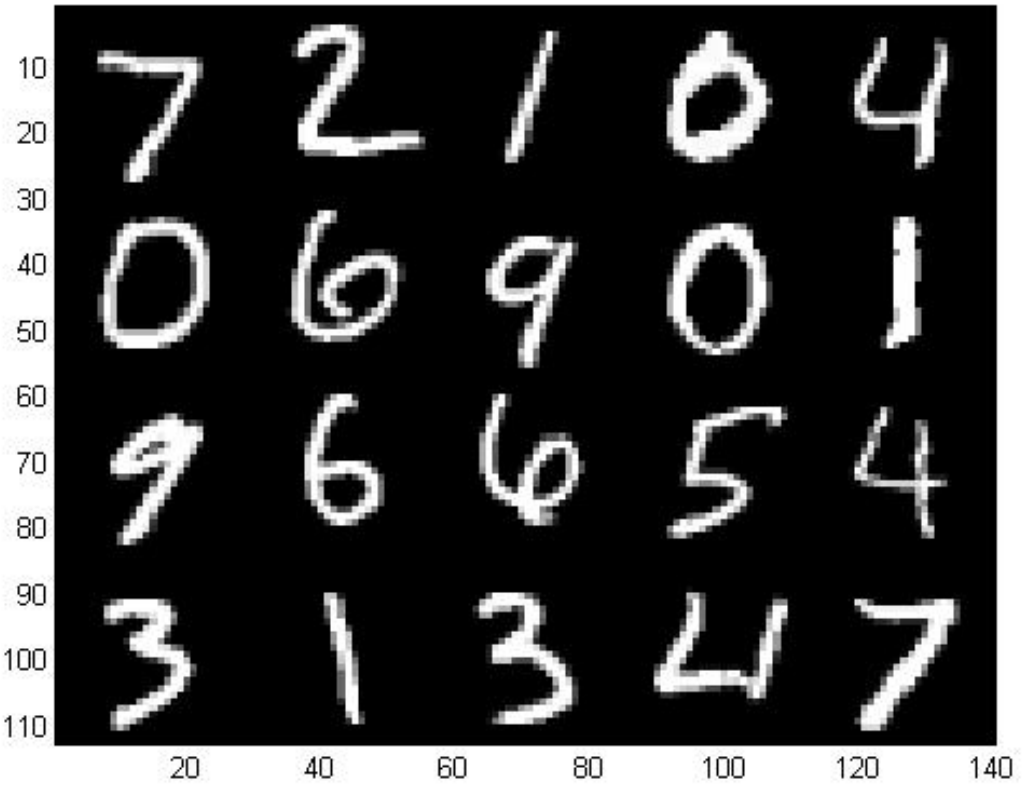


FIGURE 5.1 – Exemples d'images de la base MNIST [LeC+98b]. Chaque chiffre est une image différente de 28×28 pixels.

5.2.1 Tests de référence

Pour commencer, nous avons effectué un certain nombre de tests de référence.
Le premier test consiste à effectuer la classification directe des images à partir des valeurs de

pixels vectorisés. Il s'agit des performances obtenues sans aucun traitement sur les données autre qu'une normalisation. En utilisant un classifieur de type SVM linéaire ("un contre tous", $C = 1000$) [Fan+08], [Vap98], on obtient un taux d'erreur de classification de 8.2%.

Le second test consiste à utiliser une architecture avec une unique couche où le dictionnaire est appris de manière non-supervisée. Le dictionnaire est obtenu par la formulation basée sur la reconstruction (Chapitre 2, Eq. 2.3) pour 3 tailles de dictionnaires $K = 400, 700, 1400$ atomes. Pour ces expériences, les images sont encodées intégralement et ne sont pas décomposées en patchs d'images. La régularisation choisie est "Elastic-net" (Chapitre 4, Section 4.3.4) et les paramètres associées sont $\lambda_1 = 0.1$ et $\lambda_2 = 0.01$. Les taux d'erreurs de classification obtenus avec le classifieur SVM sont de 5.68%, 4.59% et 3.93% respectivement.

On peut déjà constater que le taux d'erreurs de classification entre la classification des pixels de l'image et des codes issus d'un encodage parcimonieux naïfs (LASSO classique) a été divisé par deux et que l'utilisation d'un dictionnaire permet d'extraire des descripteurs qui sont plus robustes que les niveaux de gris des pixels. On remarque aussi qu'augmenter le nombre d'atomes améliore le taux de classification ce qui est raisonnable car dans ce cas, le nombre d'atomes correspond également au nombre de descripteurs utilisés et on obtient donc une description plus précise de chaque image.

5.2.2 Apprentissage d'une architecture multicouche de dictionnaires de manières non supervisée

Le troisième test consiste à calculer les performances de l'architecture proposée sans apprentissage. Pour effectuer ce test, nous allons reproduire l'enchainement d'encodage sur les différentes couches en utilisant des dictionnaires appris pour la reconstruction. Pour obtenir les dictionnaires, les images de l'ensemble d'apprentissage sont décomposés en patchs (couche 1), ces patchs sont ensuite employés pour apprendre le premier dictionnaire $\mathbf{D}^{(1)}$. Une fois celui-ci appris, il est utilisé pour procéder à l'encodage des patchs précédents pour former l'image 3D de la couche 2. Cette image est de nouveau décomposée en patchs qui servent à apprendre le second dictionnaire $\mathbf{D}^{(2)}$. Ce dictionnaire sert à encoder les patchs de la couche 2 pour former la couche 3, et ainsi de suite.
Nous avons choisi de tester deux architectures de forme similaires (3 couches) mais avec un paramétrage différent. Pour effectuer la comparaison proposée, les paramétrages sont présentés dans le Tableau 5.1 : pour ces deux paramétrages, seule la dernière couche se différencie par le nombre d'atomes.
Les images de MNIST ont initialement une taille de 28×28 pixels et nous avons choisi de les agrandir à 32×32 pixels pour pouvoir utiliser des architectures identiques sur les bases MNIST et CIFAR-10. La régularisation choisie est "Elastic-net" avec les même paramètres que pour le test précédent. Avant chaque étape d'encodage, les patchs sont normalisés pour avoir une norme ℓ_2 égale à 1.
En apprenant les dictionnaires de manière non supervisée, les taux d'erreurs obtenus avec le classifieur SVM linéaire ont été de 4.93% pour l'architecture M1 et de 2.2% pour l'architecture M2. Ces résultats confortent les résultats de l'architecture à 1 couche dans le sens où

l'augmentation du nombre de descripteurs (nombre d'atomes à la dernière couche) semblent améliorer les performances. Les résultats sont synthétisés dans le Tableau 5.2.

	Architecture M1	Architecture M2
Entrées	$32 \times 32 \times 1$	$32 \times 32 \times 1$
Couche 1	5×5 - 25 atomes	5×5 - 25 atomes
Transformation	max_+ Pooling 2×2, décalage $= 2$	max_+ Pooling 2×2, décalage $= 2$
Couche 2	5×5 - 25 atomes	5×5 - 25 atomes
Transformation	max_+ Pooling 2×2, décalage $= 2$	max_+ Pooling 2×2, décalage $= 2$
Couche 3	5×5 - 50 atomes	5×5 - 400 atomes
Transformation	max_+	max_+

TABLE 5.1 – Les deux architectures multicouches utilisées. Elles se différencient par le nombre d'atomes de la couche 3.

Pixels	1 couche			3 couches	
	400 atomes	700 atomes	1400 atomes	Architecture M1	Architecture M2
8.2%	5.68%	4.59%	3.93%	4.93%	2.2%

TABLE 5.2 – Récapitulatif des taux d'erreurs pour l'ensemble des tests dans le cas non supervisé (classifieur SVM).

5.2.3 Apprentissage d'une architecture multicouche de dictionnaires de manières supervisée

Dans cette expérience, nous reprenons les conditions du test précédent, en appliquant cette fois un apprentissage supervisé, c'est-à-dire en optimisant par rapport à la tâche de classification. Pour l'apprentissage supervisé, les dictionnaires utilisés sont initialisés avec les dictionnaires obtenus par apprentissage non supervisé (et qui fournissent les résultats décrits précédemment). Le pas d'apprentissage initial est $\eta = 0.6$ et la taille des batchs pour chaque itération de descente est de 10 signaux. Le pas d'apprentissage est divisé par deux tous les deux cycles (nous rappelons qu'un cycle correspond à un parcours de l'intégralité des éléments de l'ensemble d'apprentissage par l'algorithme).

Les résultats obtenus avec la supervision sont respectivement : 0.46% et 0.41 %. On constate que la supervision augmente grandement les performances, que ce soit avec une architecture équivalente ou une architecture utilisant des descripteurs de dimension plus faible : l'architecture M1 (descripteur de taille 50) en apprentissage supervisé est bien plus performante que l'architecture M2 (descripteur de taille 400) en apprentissage non supervisé (Tableau 5.3). Dans la continuité des tests précédents, une augmentation de la taille des descripteurs semble améliorer les performances même si cette amélioration n'est pas très significative.

Apprentissage non supervisé		Apprentissage supervisé	
Architecture M1	Architecture M2	Architecture M1	Architecture M2
4.93%	2.2%	0.46%	0.41 %

TABLE 5.3 – Comparaison entre apprentissages supervisé et non supervisé pour les architectures 1 et 2.

La figure 5.2 donne les courbes d'apprentissage qui correspondent au taux de bonnes classifications en fonction du nombre d'itérations pour les deux architectures. Celle-ci nous montre que l'architecture M2 (en rouge) apprend plus rapidement que l'architecture M1. Cela s'explique par le nombre de paramètres plus important dans l'architecture M2 (environ 48k [1] pour l'architecture M1 et 266k pour l'architecture M2) . Il faut toutefois noter qu'un apprentissage plus rapide en terme de nombre d'itérations ne signifie pas que la durée d'apprentissage est plus courte. En effet, comme l'architecture M2 contient plus d'atomes à la couche 3, le temps requis pour calculer une unique itération d'apprentissage est plus long pour cette architecture.

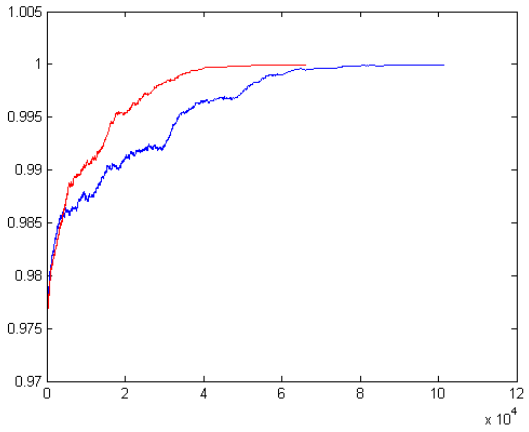


FIGURE 5.2 – Courbe de la précision en classification en fonction du nombre d'itérations pour les architectures 1 (en bleu) et 2 (en rouge).

Nous avons également testé une architecture à 5 couches (voir Tableau 5.4) afin de la comparer aux deux premières architectures proposées. Nous souhaitons estimer la robustesse de la méthode vis-à-vis des différentes opérations effectuées : cette architecture possède 2 couches de plus et une unique opération de pooling au lieu de deux pour les deux architectures précédentes. Chacune des couches utilise un petit nombre d'atomes comme dans les architectures précédentes pour environ 79k paramètres, soit un nombre de paramètres situés entre ceux des architectures M1 et M2. Après apprentissage, le taux d'erreur en classification obtenu est de 0.42% soit environ les performances de l'architecture M2 pour 3 fois moins de paramètres. L'augmentation des paramètres n'a pas entrainé une variation signification dans les performances en classification (architectures M1 et M2). Cela signifie que la complexité de l'architecture est suffisante. Par ailleurs, les résultats des architectures M1, M2 et M3 sont

1. k = 10^3

similaires ce qui semble montrer une certaine robustesse dans le choix des paramètres des architectures.

	Architecture M3
Entrées	$32 \times 32 \times 1$
Couche 1	7×7 - 25 atomes
Transformation	max_+
Couche 2	5×5 - 25 atomes
Transformation	max_+ Pooling 2×2, décalage $= 2$
Couche 3	5×5 - 25 atomes
Transformation	max_+
Couche 4	5×5 - 25 atomes
Transformation	max_+
Couche 5	3×3 - 50 atomes
Transformation	max_+

TABLE 5.4 – L'architecture à 5 couches utilisée. Contrairement aux architectures précédentes, celle-ci n'utilise qu'un seul pooling.

5.2.4 Comparaisons avec l'état de l'art

Méthodes	Taux d'erreur
Yang et al. [YYH10]	0.84%
Yu et al. [YLL11]	0.77%
Mairal et al. [MBP12] 50 atomes	0.96%
Mairal et al. [MBP12] 300 atomes	0.54%
Méthode proposée Architecture M1 (Softmax)	0.46%
Méthode proposée Architecture M2 (Softmax)	0.41%
Méthode proposée Architecture M3 (Softmax)	0.42%

TABLE 5.5 – Comparaison des performances sur la base de données MNIST.

Le Tableau 5.5 donne une comparaison des performances de différentes méthodes de l'état de l'art sur cette base de données. Nous allons donner un descriptif succinct des différentes méthodes à comparer.

L'algorithme proposé par Yang et al. [YYH10] repose sur l'apprentissage supervisé d'un dictionnaire appris sur des patchs d'images. Une image est ensuite décomposée en patchs et

encodée : cela correspond à la première couche de notre algorithme. Ces patchs subissent ensuite successivement plusieurs opérations de pooling donnant une représentation à plusieurs échelles. Les représentations à toutes les échelles sont concaténées pour donner le descripteur final.
L'algorithme proposé par Yu et al. [YLL11] repose sur l'apprentissage non supervisé de deux couches de dictionnaires. La méthode utilise des patchs et fournit une représentation discriminante à deux niveaux.
L'algorithme proposé par Mairal et al. [MBP12] est l'inspiration de l'algorithme proposé. Il s'agit de l'apprentissage supervisé d'une unique couche de dictionnaire, conjointement avec le classifieur : les images sont encodées directement sans passer par des patchs puis classées à partir des codes obtenus. Deux résultats sont donnés : avec un dictionnaire de 50 atomes et un dictionnaire de 300 atomes. Il faut noter que les résultats fournis dans l'article tiennent compte d'une augmentation de l'ensemble d'apprentissage : des images déformées par une légère translation dans chacune des directions ont été ajoutées.

La méthode proposée donne de meilleurs résultats que les méthodes à base de dictionnaires existantes. Les performances obtenues ainsi que la faible dimension des descripteurs montre une capacité à apprendre des descripteurs discriminants pour la classification. On peut noter qu'avec un nombre identique de dimensions pour les descripteurs, notre méthode multicouche fournit de meilleurs résultats que la méthode avec une couche unique.

La base MNIST n'est pas très complexe et l'apprentissage est assez rapide ce qui est utile pour des tests préliminaires. Cependant, les taux d'erreurs déjà très bas avec des méthodes de type dictionnaires et la simplicité des images ne nous permettent pas de tirer toutes les conclusions sur les performances de notre système.

5.3 Base d'images CIFAR-10

5.3.1 Description

La base d'images CIFAR-10 [KH09] est composée d'images réelles en couleurs de 32×32 pixels. Elle comporte 10 classes et contient 60000 images réparties de manière prédéfinie et équilibrée en 50000 images d'apprentissage et 10000 images de tests. Cette base est plus complexe que MNIST et permet de mieux appréhender la capacité d'apprentissage de l'architecture proposée. La figure 5.3 illustre quelques objets d'intérêt dans les images qui ont une grande variabilité de forme, de couleurs et de position. On peut noter que les classes sont d'un haut niveau sémantique (avion, chien ..) et que les images peuvent être difficilement reconnaissables par un humain.

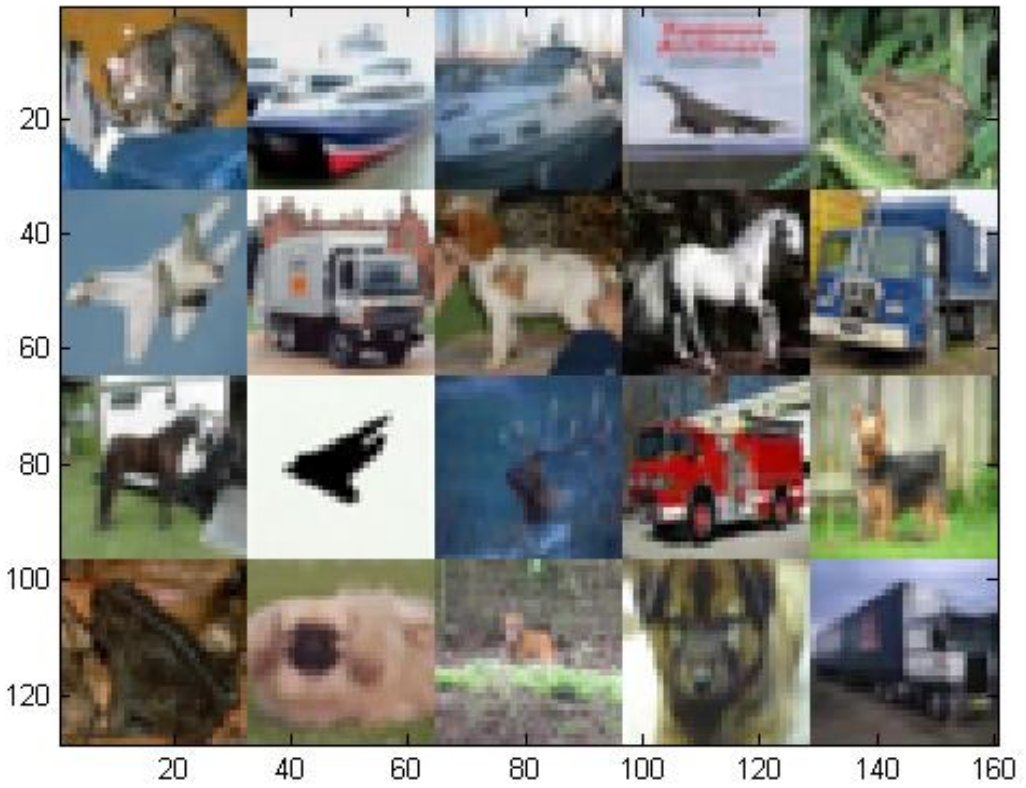


FIGURE 5.3 – Exemples d'images de la base CIFAR-10 [KH09]. Chaque image fait 32×32 pixels.

5.3.2 Blanchiment

L'apprentissage sur la précédente base MNIST a été effectué directement sur les images fournies sans nécessiter de traitements autres qu'une simple normalisation des valeurs (pour ne pas traiter des valeurs de niveaux gris entières entre 0 et 255). Cependant, dans les images naturelles, on observe de fortes corrélations entre les valeurs des pixels proches [KH09]. Lors de l'apprentissage, les atomes vont reproduire ces corrélations et pénaliser la recherche d'éléments discriminants : c'est pourquoi pour des tâches de classification on effectue souvent un pré-traitement nommé blanchiment qui sert à retirer ces corrélations [CN12] [BS97]. Par ailleurs, LeCun et al. [LeC+98a] ont montré que décorréler les variables d'entrées accélère l'apprentissage par descente de gradient.

Le blanchiment d'un patch $\mathbf{y}$ consiste à lui appliquer $\mathbf{Ty}$ où $\mathbf{T}$ est une matrice.
Pour déterminer la matrice de transformation $\mathbf{T}$, on considère l'ensemble des patchs extraits de la base : $\mathbf{Z}$. Ces patchs sont centrés et réduits puis on calcule la matrice de covariance $\mathbf{C}$ de $\mathbf{Z}$. On calcule la décomposition en valeurs propres de $\mathbf{C} = \mathbf{VPV}^\top$. La matrice de transformation

$\mathbf{T}$ pour effectuer le blanchiment est alors : $\mathbf{T} = \mathbf{V} \cdot (\mathbf{P}^{-1} + 0.1\mathbf{I})^{\frac{1}{2}} \cdot \mathbf{V}^{\top}$. Cette transformation a pour objectif de transformer la matrice de covariance pour qu'elle devienne la matrice identité, décorrélant ainsi la valeur des pixels d'une image.

5.3.3 Tests de référence

Nous reprenons les mêmes premiers tests effectués sur la base MNIST afin d'obtenir quelques performances de référence : le premier test consiste à effectuer la classification directe des images à partir des valeurs de pixels vectorisés. Les performances sont obtenues sans aucun traitement sur les données autre qu'une normalisation. En utilisant un classifieur de type SVM linéaire, on obtient un taux de bonnes classifications de 37.85%. Cette performance montre que la complexité des classes dans la base de données CIFAR-10 ne permet pas une discrimination efficace directement à partir de la valeur des pixels : il faut donc se tourner vers des descripteurs plus pertinents.

Dans le second test, nous évaluons les performances avec une unique couche de dictionnaire apprise de manière non-supervisée. Le dictionnaire est obtenu par la formulation basée sur la reconstruction (Chapitre 2, Eq. 2.3) dans 3 configurations $K = 400, 700, 1400$ atomes. Nous nous plaçons dans les même conditions que pour la base de données MNIST. Les taux de bonnes classifications obtenus avec le SVM sont de 43.89%, 45.43% et 46.97% respectivement.

On peut constater que le taux de classification entre l'utilisation des pixels de l'image et des codes issus d'un encodage parcimonieux naïf (LASSO classique) augmente et reste cohérent avec les résultats obtenus sur MNIST. On remarque qu'augmenter le nombre d'atomes améliore ici également les taux de classification.

5.3.4 Apprentissage d'une architecture multicouche de dictionnaires de manière non supervisée

Nous reprenons le test suivant sur l'architecture multicouche proposée sans apprentissage. Pour effectuer ce test, nous reproduisons l'enchainement d'encodage sur les différentes couches en utilisant des dictionnaires qu'on apprend pour la reconstruction. Les conditions d'apprentissage sont les mêmes que pour la base de données MNIST.
Pour effectuer la comparaison proposée, l'architecture utilisée est présentée dans le Tableau 5.6 : la seconde architecture double le nombre des atomes de la première sur chaque couche. La régularisation choisie est toujours "Elastic-net" avec les même paramètres que pour les tests précédents. Avant chaque étape d'encodage, les patchs sont normalisés pour avoir une norme ℓ_2 égale à 1.
En apprenant les dictionnaires de manière non supervisée, les taux de classification obtenus avec le classifieur SVM linéaire ont été de 34.98% pour l'architecture C1 et de 42.39% pour l'architecture C2. Ces résultats suivent les résultats obtenus avec une couche unique dans le sens où l'augmentation du nombre de descripteurs semblent améliorer les performances. Les résultats sont synthétisés dans le Tableau 5.7. Des performances aussi faibles pourraient s'ex-

pliquer par la petite taille des dictionnaires : les dictionnaires utilisés ont moins d'atomes que la dimension des signaux (cas non "overcomplete"). Lorsqu'on apprend un dictionnaire sur un critère de reconstruction, plus le nombre d'atomes est élevé, plus on peut capturer de détails. Habituellement, les autres méthodes d'apprentissage non supervisées dans l'état de l'art utilise des descripteurs entre 50 et 100 fois plus grands [CLN11].

	Architecture C1	Architecture C2
Entrées	$32 \times 32 \times 1$	$32 \times 32 \times 1$
Couche 1	5×5 - 25 atomes	5×5 - 50 atomes
Transformation	max_+ Pooling 2×2, décalage $= 2$	max_+ Pooling 2×2, décalage $= 2$
Couche 2	5×5 - 25 atomes	5×5 - 50 atomes
Transformation	max_+ Pooling 2×2, décalage $= 2$	max_+ Pooling 2×2, décalage $= 2$
Couche 3	5×5 - 50 atomes	5×5 - 100 atomes
Transformation	max_+	max_+

TABLE 5.6 – Deux architectures multicouches utilisées. Elles se différencient par le nombre d'atomes aux différentes couches : l'architecture C2 contient le double d'atomes de l'architecture C1.

Pixels	1 couche			3 couches	
	400 atomes	700 atomes	1400 atomes	Architecture C1	Architecture C2
37.85%	43.89%	45.43%	46.97%	34.98%	42.39%

TABLE 5.7 – Récapitulatif des taux de bonnes classifications pour le cas non supervisé (classifieur SVM).

5.3.5 Apprentissage d'une architecture multicouche de dictionnaires de manières supervisée

5.3.5.1 Performances

Nous reprenons les conditions du test précédent, mais avec l'apprentissage supervisé. Les dictionnaires sont initialisés avec ceux obtenus par apprentissage non supervisé (fournissant les résultats décrits précédemment). Le pas d'apprentissage initial est $\eta = 0.6$ et la taille des batchs pour chaque itération de descente est de 10 signaux. Le pas d'apprentissage est divisé par deux tous les trois cycles (nous rappelons qu'un cycle correspond à un parcours de l'intégralité des éléments de l'ensemble d'apprentissage par l'algorithme).

Les taux de bonnes classifications obtenus avec la supervision sont respectivement : 78.86% et 83.19 %. On constate que la supervision augmente encore une fois grandement les performances pour les deux architectures proposées. De plus, nous pouvons constater qu'entre l'ar-

chitecture C1 et l'architecture C2 les performances augmentent. On peut aussi noter que les performances sur l'ensemble d'apprentissage ne sont pas encore maximales, il est donc possible que le système soit encore sous-paramétré.

Les deux architectures présentées précédemment ne permettent pas d'estimer l'effet de la taille des dictionnaires pour les différentes couches. Nous introduisons donc l'architecture C3 (Tableau 5.8) qui reproduit l'architecture C1 en diminuant le nombre d'atomes dans la première couche.
Pour cette dernière architecture, les performances obtenues sont 67.56 %. Malgré une taille de descripteurs identique à l'architecture C1, le taux de classification est inférieur de plus de 10 % après apprentissage supervisé tandis que la différence de performances avant supervision est faible (environ 0.5 %). La complexité de la première couche est donc très importantes pour la qualité de la représentation obtenue à la dernière couche après apprentissage.

Les performances des différentes architectures sont rassemblées dans le Tableau 5.9.

	Architecture C3
Entrées	$32 \times 32 \times 1$
Couche 1	5×5 - 15 atomes
Transformation	max_+ Pooling 2×2, décalage $= 2$
Couche 2	5×5 - 25 atomes
Transformation	max_+ Pooling 2×2, décalage $= 2$
Couche 3	5×5 - 50 atomes
Transformation	max_+

TABLE 5.8 – L'architecture C3 qui copie l'architecture C1 en diminuant à 15 (au lieu de 25) le nombre d'atomes de la première couche.

Apprentissage non supervisé		
Architecture C1	Architecture C2	Architecture C3
34.98%	42.39%	34.47%
Apprentissage supervisé		
Architecture C1	Architecture C2	Architecture C3
78.86%	83.19 %	67.56 %

TABLE 5.9 – Comparaison des taux de bonnes classifications entre apprentissages supervisé et non supervisé pour les architectures C1, C2 et C3.

5.3.5.2 Courbe d'apprentissage

La figure 5.4 montre l'effet du choix du pas d'apprentissage utilisé, ce pas étant divisé par 2 tous les 3 cycles (Section 5.1). On observe l'apparition de "paliers". Cet effet n'est pas très marqué sur les courbes d'apprentissage de la base MNIST à cause du fait que les

taux de classification atteignent rapidement le maximum. Cependant, la figure 5.4 illustre très bien l'évolution de la précision en classification en fonction de la variation des pas : chaque "palier" correspond au moment où le pas d'apprentissage est divisé par 2. Cette modification se fait à intervalles réguliers, après un certain nombre de cycles sur l'ensemble d'apprentissage. En pratique, l'utilisation d'un tel choix sur le pas d'apprentissage nous a fourni de meilleurs résultats qu'une décroissance par exemple en $\frac{1}{t}$ où t désigne le nombre d'itérations.

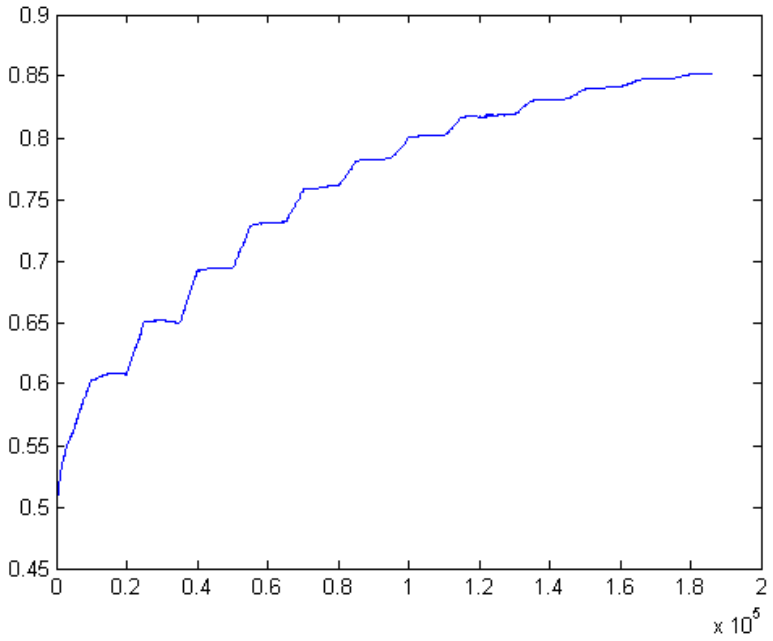


FIGURE 5.4 – Courbe de la précision moyenne en classification sur l'ensemble d'apprentissage en fonction du nombre d'itérations pour l'architecture C1.

5.3.5.3 Comparaisons avec l'état de l'art

Le Tableau 5.10 donne une comparaison des performances de différentes méthodes de l'état de l'art sur la base d'images CIFAR-10. Nous allons donner un descriptif succinct des différentes méthodes comparées.

Peu de méthodes reposant sur des dictionnaires tels que nous les avons définis ont été testées sur cette base d'images car elle est complexe : la variabilité en termes de localisation, de point de vue et d'échelle est un obstacle pour l'apprentissage de dictionnaires de la taille des images. Pour les comparaisons, nous allons donc nous intéresser principalement à des méthodes basées sur l'analyse de patchs.

L'algorithme proposé par Fawzi et al. [FDF15] repose sur l'apprentissage d'un dictionnaire de manière non supervisée couplé à un ensemble de classifieurs simples reposant sur des opérations de seuillage. Le dictionnaire utilise des atomes de la taille des images (32×32 pixels) et fonctionne avec une unique couche.

L'algorithme proposé par Coates et al. [CLN11] repose sur l'algorithme K-moyennes pour l'apprentissage de centroïdes sur des patchs d'images suivi d'un encodage non parcimonieux. Une image est décomposée en patchs et chacun de ces patchs est encodé en calculant sa distance euclidienne par rapport à chacun des centroïdes. Finalement, une opération de pooling sur l'ensemble des patchs est effectuée pour former le descripteur final. Pour obtenir le résultat annoncé, 4000 centroïdes ont été utilisés.
Dans un second article Coates et al. [CN11] modifient l'algorithme et utilise "Orthogonal Matching Pursuit" pour l'encodage. Leur méthode, couplée à un dictionnaire de 6000 centroïdes, améliore leur résultat précédent.
La méthode proposée par T-H Lin et al. [LK14] repose quant à elle sur l'apprentissage d'une structure multicouche de dictionnaires, mais de manière non supervisée. La méthode en question propose d'alterner une couche d'encodage avec une couche de pooling. L'algorithme d'encodage utilisée est une version modifiée de "Orthogonal Matching Pursuit" (OMP) [CW11] reposant sur une contrainte de non-négativité.

La méthode que nous avons proposée donne de meilleurs résultats que celle à base de dictionnaires existante [FDF15]. Nous rappelons que les résultats pour la méthode proposée ont été obtenus avec au plus un descripteur de taille 100. Malgré la faible taille des descripteurs, les performances obtenues surpassent les méthodes de la littérature. Cela montre une capacité à apprendre des descripteurs discriminants pour la classification.

Méthodes	Taux de classification
Fawzi et al. [FDF15]	53.44%
Coates et al. [CLN11]	79.6%
Coates et al. [CN11]	81.5%
T-H Lin et al. [LK14]	81.4%
Méthode proposée Architecture C1 (Softmax)	78.86%
Méthode proposée Architecture C2 (Softmax)	83.19 %
Méthode proposée Architecture C3 (Softmax)	67.56 %

TABLE 5.10 – Comparaison des performances sur la base de données CIFAR-10.

Par rapport aux méthodes basées sur les réseaux de neurones convolutifs (CNN), nous pouvons apporter quelques comparaisons : un test a été réalisé à l'aide de la librairie CAFFE [Jia+14] en reproduisant notre architecture C1 (3 couches de convolution et 2 de pooling, ReLU (type de non-linéarité) et nombre de filtres identiques à nos nombres d'atomes) et notre architecture C2. Aucune autre "amélioration" (augmentation de données et régularisations spécifiques) des CNN n'est utilisée. La performance pour l'architecture C1 est de 66.5 % avec CAFFE contre 78.86 % avec nos dictionnaires et pour l'architecture C2 elle est de 76% avec CAFFE contre 83.19 % avec nos dictionnaires. Cette différence peut s'expliquer par le petit nombre de couches et de filtres utilisés par rapport à ce qui est habituellement fait en utilisant des CNN.

Par ailleurs, Hinton et al. [Hin+12] utilisent une architecture à 3 couches de convolutions, 2 de pooling et 64 filtres pour chaque couche de convolutions (210k paramètres). En guise de classifieur, 3 couches de neurones totalement connectés (2048 neurones) sont ajoutés. C'est l'article qui présente l'amélioration "dropout" pour les réseaux de neurones qui consiste à omettre aléatoirement certains des poids dans la couche de classification pour réduire le sur-apprentissage.
L'architecture totale compte donc 6 couches mais reste proche si l'on compte uniquement les couches utilisant des convolutions. Les performances obtenues sont une précision de 83.4 % sur CIFAR-10. Ces résultats sont proches des performances obtenues avec l'architecture C2 (83.19 %, 191k paramètres).

Avec ces exemples, on remarque que la méthode proposée est compétitive sur cette base de données avec les CNN pour une architecture proche.

Parmi les autres architectures convolutives existantes, on peut également mentionner celle proposée par Oyallon et Mallat [OM15], qui repose sur plusieurs couches de décompositions en ondelettes. Les points forts présentés dans l'article sont de pouvoir construire des descripteurs discriminants et invariants à partir d'une famille d'ondelettes pré-définies. Les performances obtenues sont de 82.3 % de précision pour une taille de descripteur de 2000.
Ici aussi, nous pouvons souligner la différence dans la taille des descripteurs pour des résultats équivalents.

Les résultats mentionnés sont synthétisés dans le Tableau 5.11.

Méthodes	Taux de classification
CAFFE - Architecture C1	66.5%
CAFFE - Architecture C2	76%
Hinton et al. [Hin+12]	83.4%
Oyallon et Mallat [OM15]	82.3%
Méthode proposée Architecture C1 (Softmax)	78.86%
Méthode proposée Architecture C2 (Softmax)	83.19 %

TABLE 5.11 – Comparaison des performances entre des architectures convolutives et la méthode proposée sur la base CIFAR-10.

5.4 Base d'images STL-10

L'apprentissage est la phase importante qui fixe les performances du système mais cette phase est très lourde en temps de calcul. Nous souhaitons savoir si le système ayant été entraîné sur une base de données peut donner des performances correctes sur une autre base.

Pour ce faire, on utilise la base d'images STL-10 [CLN11] (Fig. 5.5) qui est composée

d'images réelles en couleurs de 96×96 pixels. Elle comporte 10 classes et contient 10000 images d'apprentissage et 8000 images de tests. Cette base de données comporte également 100.000 images non labellisées pour faire de l'apprentissage non supervisé.
Nous allons réutiliser l'architecture C2 apprise sur CIFAR-10 et l'appliquer sur les images redimensionnées à la taille 32×32 pixels de la base STL-10. Aucun ré-apprentissage n'est effectué : les descripteurs sont directement extraits à partir de l'ancienne architecture C2 puis couplés avec un classifieur SVM linéaire.

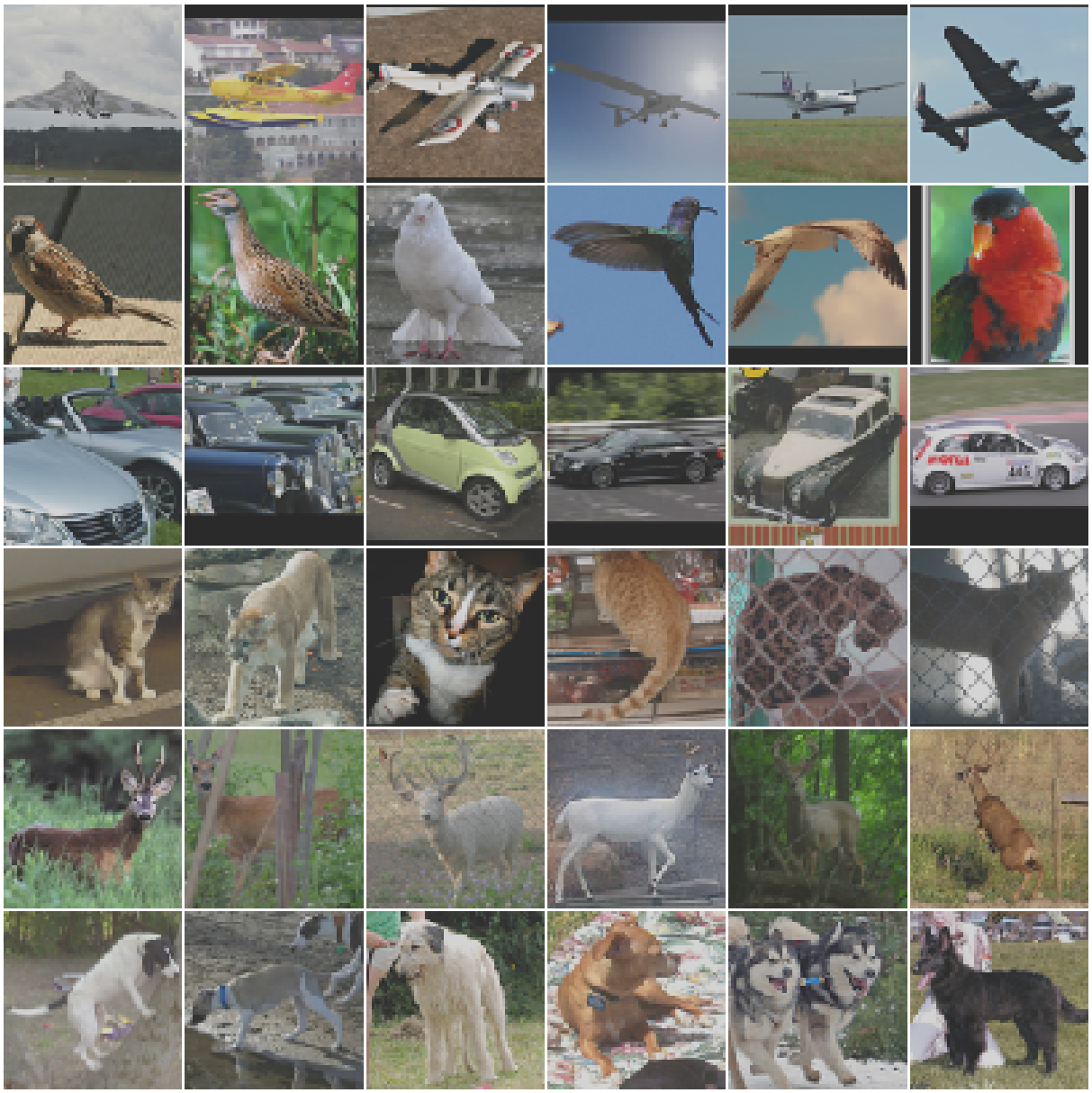


FIGURE 5.5 – Exemples d'images de la base STL-10 [CLN11]. La résolution originale des images est 96×96 pixels.

Le Tableau 5.12 regroupe quelques comparaisons de performances. La méthode proposée

apprend des descripteurs plus performants que les pixels bruts, même lorsque l'apprentissage a été fait sur une autre base d'images. Nous la comparons aux méthodes non supervisées de l'état de l'art car aucun ré-apprentissage n'a été effectué et il s'agit d'estimer la pertinence des descripteurs appris sur d'autres bases de données.
Les méthodes proposées par Coates et al. [CLN11] et T-H Lin et al. [LK14] ont été décrites précédemment pour la base de données CIFAR-10. Pour obtenir les résultats présentés dans le Tableau 5.12, ces méthodes ont été appliquées sur la base STL-10 (apprentissage non-supervisé) contrairement à la méthode que nous proposons qui réutilise l'apprentissage effectué sur la base de données CIFAR-10.
La méthode proposée fournit de meilleurs résultats que les méthodes d'apprentissage non supervisé de l'état de l'art ce qui illustre la capacité de la méthode à extraire des descripteurs génériques et discriminants pour la classification.

Méthodes	Taux de bonnes classifications
Valeurs des pixels [CLN11] (Image - Taille originale)	31.8%
Coates et al. [CLN11] (Image - Taille originale)	51.5%
Coates et al. [CN11] (Image - Redimensionnée 32×32)	59.0%
T-H Lin et al. [LK14] (Image - Redimensionnée 32×32)	60.4%
Méthode proposée (Architecture 2 CIFAR-10) (Image - Redimensionnée 32×32)	66.35%

TABLE 5.12 – Comparaison des performances des méthodes non-supervisées ou entrainées avec la base CIFAR-10 et appliquées à la base de données STL-10.

5.5 Conclusion

Les tests de notre système ont été réalisés en utilisant le même algorithme mais avec des paramétrages différents. Il est à noter que suivant le paramétrage choisi, la phase d'apprentissage est extrêmement gourmande en temps de calcul. Par exemple, pour l'architecture C2 sur la base CIFAR-10, l'expérience a duré environ 1 mois sur une machine en utilisant 8 coeurs @ 2.67GHz et MATLAB.
Ceci explique qu'il ne nous est pas possible, dans cette configuration, de faire une étude exhaustive des effets du paramétrage. Mais le principe de la méthode a été implémenté et donne des résultats tout à fait corrects dans les configurations que nous avons intuitées comme adaptées.
Il faudrait maintenant envisager une implémentation avec une forte parallélisation, ce qui est tout à fait envisageable compte tenu de la structure, pour pousser plus loin les investigations.

Chapitre 6

Extension à l'analyse du mouvement

Sommaire

6.1 Contexte

Notre architecture multi-couches basée sur des dictionnaires a été testée sur des images couleurs et en niveaux de gris. Comme les patchs extraits de ces images sont utilisés sous forme vectorisées, les traitements effectués ensuite dans les deux cas sont exactement les mêmes. Les vidéos peuvent être vus comme des images auxquelles nous ajoutons une dimension supplémentaire : le temps. Il est donc intéressant de voir si l'approche proposée peut-être étendue à l'analyse de vidéos.

En effet, l'équipe AGPIG au GIPSA-Lab travaille depuis plusieurs années sur le traitement vidéo et la classification du mouvement de personne (Thèses de Ramasso [Ram07] et Guironnet [Gui06]). Il s'agit, dans une séquence vidéo, de reconnaitre l'action d'une ou de plusieurs personnes (voir Fig. 6.1). Récemment, l'équipe a également acquis un robot compagnon dont le rôle est de s'assurer de l'état d'une personne âgée ou en situation de faiblesse. Le robot est vu comme un ensemble de capteurs reconfigurables pouvant se déplacer.

C'est dans ce cadre que se situe cette étude préliminaire. L'objectif est d'étudier si les méthodes de classification à base de dictionnaires présentées précédemment peuvent être facilement réutilisées pour analyser par exemple le mouvement de personnes à partir d'un flux vidéo capté par le robot.

Plus généralement, l'application envisagée ici est la reconnaissance d'action qui est une tâche de classification consistant à retrouver à partir d'une séquence vidéo courte l'action exécutée par un personnage.

FIGURE 6.1 – Exemples d'images pour l'action "Tennis". Les points de vues et les conditions d'acquisition sont très variés.

6.2 Modélisation des données

Nous avons déjà manipulé des images couleurs en les traitant comme des matrices d'images à 3 dimensions. Dans le cadre du mouvement, la troisième dimension définie ici est le temps [CRP14], [CRP15]. Nous commençons par définir un bloc *temporel* d'images illustrant un mouvement en considérant des images successives dans une vidéo. Au sein de ces blocs temporels, nous définissons des régions d'images 3D qui correspondent à des morceaux d'images spatialement localisés et pris sur des images successives de la vidéo. Un exemple des régions d'intérêt considérées est illustré par la figure 6.2. Nous nous basons de nouveau sur l'analyse de patchs d'images pour obtenir une représentation locale au sein de ces blocs. Les signaux considérés sont donc des patchs d'images 3D vectorisés où la troisième dimension est le temps (des patchs pris à la même position spatiale sur des images successives dans une vidéo). On a représenté sur la figure 6.3 ces trois niveaux de décomposition de la séquence vidéo : bloc temporel, région 3D et patch 3D.

6.3 Algorithme de reconnaissance d'actions par partie

La reconnaissance du mouvement d'une personne sera plus performante si l'on peut focaliser l'attention sur des régions recouvrant partiellement cette personne.

D'autre part, les vidéos traitées ont une résolution plus grande que les images considérées dans le chapitre précédent. Pour limiter la taille des images à traiter (et ainsi les temps de calculs) nous nous restreignons à certaines zones d'intérêts sur les images.

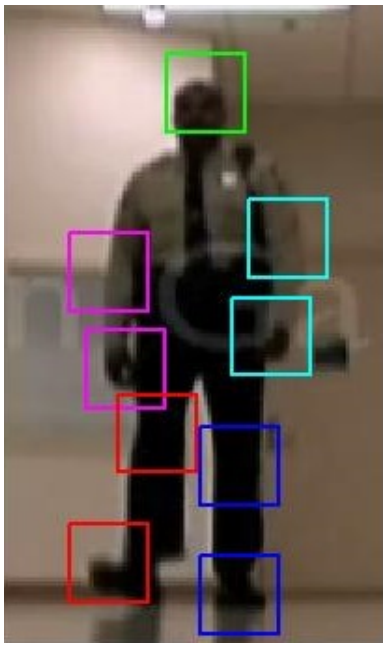


FIGURE 6.2 – Exemple de 9 parties détectées par l'algorithme proposé par [YR13].

6.3.1 Détecteur d'humains par parties

La détection d'humain dans une image est une tâche complexe que nous n'avons pas abordée dans ce travail. Pour effectuer la reconnaissance d'action, il est nécessaire de posséder de bons descripteurs permettant de caractériser le mouvement des personnes. Classiquement, on peut utiliser une boite englobante de la personne en mouvement qui peut par exemple être obtenue par soustraction de fond (caméra fixe) ou par compensation du mouvement (caméra mobile). Dans cette thèse, nous nous sommes intéressés à l'utilisation d'un algorithme de détection de personnages humains par parties développé par Yang et al. [YR13].
L'avantage d'exploiter un tel détecteur est de pouvoir concentrer l'information sur des régions spécifiques de l'image. Cependant, l'inconvénient d'une telle méthode repose dans le fait que, puisque nous donnons plus de poids à l'information de certaines régions, alors la qualité de la classification dépend fortement de la qualité de la détection.

L'algorithme de Yang et al. prend en entrée une image et renvoie, en cas de détection, la localisation d'un humain ainsi qu'une estimation de la position de 26 "parties" faisant office de descripteurs pour la détection. Ces parties, qui sont des régions d'images, couvrent l'ensemble du corps et correspondent approximativement à certaines parties du corps et aux articulations. Ces régions sont détectées par l'application de filtres (plusieurs pour chaque partie) donnant une carte d'énergie. Cette carte, couplée à un ensemble de contraintes de position relative, permet d'obtenir une estimation de la localisation et de la posture d'un individu.

Concernant notre algorithme, pour limiter les tailles des descripteurs, nous avons sélectionné 9 parties qui semblent les plus pertinentes sur les 26 : 1 pour la tête, 4 pour les bras et 4 pour les jambes (voir Fig. 6.2). Pour chacune de ces parties, on définit les régions 3D :

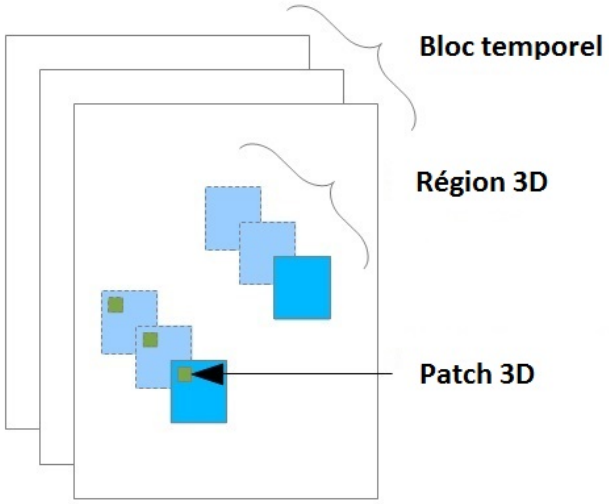
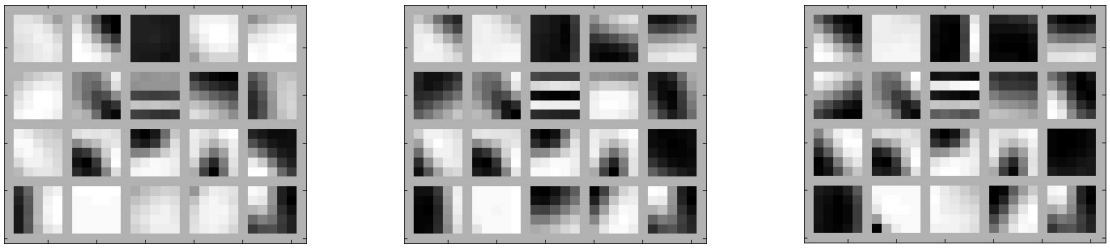


FIGURE 6.3 – Illustration des objets extraits à partir d'une séquence vidéo dans le contexte de la reconnaissance de mouvement.

leur localisation dans les images du bloc temporel est basée sur la position de la partie de la première image. Ensuite, on extrait les patchs 3D temporels associés à chacune des régions 3D.

6.3.2 Application de l'algorithme de classification sur les régions 3D

Une vidéo en entrée est découpée en blocs temporels (avec recouvrement) et chacun de ces blocs est ensuite analysé indépendamment. Cela signifie que si une vidéo est décomposée en blocs temporels de t_b images alors si cette vidéo a une longueur de L images, elle peut donc, par exemple, être découpées en $L - t_b + 1$ blocs en considérant un décalage de 1 image.
Après l'étape de détection des régions 3D à l'aide de l'algorithme de [YR13], nous souhaitons extraire des descripteurs à partir des parties sélectionnées afin de les utiliser pour la tâche de classification. Chacune des parties choisies conduit à la construction d'une région 3D et celle-ci est décomposée en un ensemble de patchs 3D.
Les patchs sont eux-même ensuite encodés sur un dictionnaire. L'idée est alors de représenter un mouvement dans une vidéo en une combinaison simple d'atomes de mouvements appris avec une méthode de type dictionnaire (voir Fig. 6.4). Le descripteur final associé à un bloc temporel est constitué de la concaténation des codes en sortie de l'encodage de chacune des régions 3D.



$t = 1$ $t = 2$ $t = 3$

FIGURE 6.4 – Exemples de 20 atomes de mouvement de taille $(5 \times 5 \times 3)$ pixels. Par exemple, sur l'atome en haut à gauche, on observe un mouvement du côté haut droit vers le côté bas gauche.

6.4 Classification

Il y a peu de modifications à effectuer pour adapter l'algorithme présenté au chapitre 4. En effet, une succession d'images peut être considérée comme une image 3D (si en niveaux de gris) ou 4D (si en couleur) où la troisième ou quatrième dimension respectivement est le temps. Dans l'algorithme proposé, les signaux étudiés sont des patchs en couleurs et sont simplement vectorisés. Cela signifie que l'ajout de la quatrième dimension augmente simplement la dimension du signal considéré en entrée. Dans le cas d'une succession d'images en niveaux de gris, les traitements sont tout à fait identiques.

Par ailleurs, nous pouvons remarquer que l'utilisation de l'architecture présentée au chapitre 4 permet de jouer le même rôle que celui du détecteur, c'est-à-dire de chercher des informations discriminantes dans des régions spécifiques des images (voir tests sur la base CIFAR-10), ce qui permet de considérer également une extraction de descripteurs sur une détection moins complexe (par exemple, les boîtes englobantes).

6.4.1 Classification des blocs temporels

Le seul élément contraignant est la dimension des parties étudiées : celle-ci peut influer sur la taille des patchs étudiés, le nombre de patchs par image, le nombre d'atomes nécessaires et éventuellement le temps d'apprentissage. Chacune des régions 3D est traitée indépendamment des autres (composition en patchs, pooling, encodage). L'appartenance des régions à un bloc temporel se limite à la concaténation des différents codes pour former le vecteur de descripteurs final.

6.4.2 Classification avec rejet

Après l'apprentissage, on souhaite pouvoir classer une nouvelle séquence vidéo. De la même manière que pendant l'apprentissage, on commence par extraire les blocs temporels, les régions

puis les patchs. La particularité du classement ici est le fait de savoir que l'ensemble des blocs extraits appartient à la même vidéo et donc la même classe (sous l'hypothèse que les vidéos ont une étiquette unique).

Dans cette section, nous allons introduire la classification avec rejet. L'idée est de pouvoir séparer les signaux "faciles" à classer des signaux "ambigus" à l'aide d'un algorithme de classification. Cette idée n'est pas totalement nouvelle et a déjà été exploitée dans le cadre d'une classification binaire [KG10].

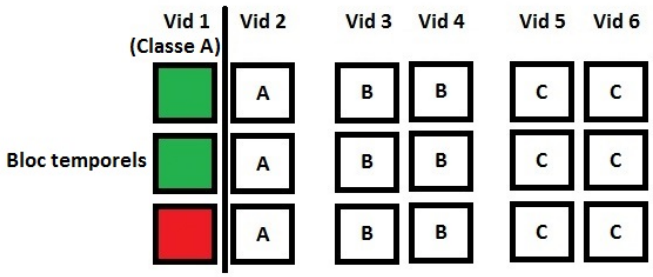
Il est possible d'extraire un vecteur de descripteurs pour chacun des blocs temporels d'une vidéo et la difficulté réside dans la recherche du label de la vidéo à partir de la classification de chacun de ces blocs d'images.

C'est dans ce contexte que nous allons appliquer la classification avec rejet. Considérons un problème à C classes. Parmi l'ensemble des blocs temporels extraits à partir des vidéos, certains sont plus informatifs que d'autres : par exemple, dans le cas d'une action ponctuelle comme "frapper dans ses mains", on peut supposer que les blocs d'images au début du clip vidéo ne permettent pas de représenter seuls l'action. Néanmoins dans la plupart des dispositifs de classification, en soumettant ces blocs au classifieur, celui-ci va proposer un label qui influencera le label final associé à la vidéo à classer. Il s'agit de la situation que nous souhaitons éviter en excluant du processus de décision les blocs d'images considérés comme "ambigus".

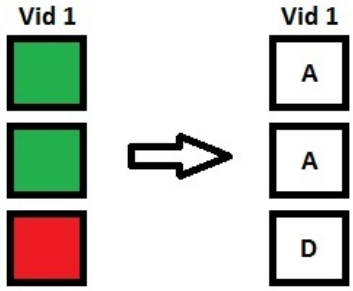
Le procédé de classification avec rejet peut s'effectuer après l'apprentissage des descripteurs, en étant séparé de l'apprentissage de l'architecture de dictionnaires. Il se base sur 3 étapes. La première étape consiste à regrouper les descripteurs des blocs temporels appartenant à une même vidéo. On forme donc N_g groupes où N_g est le nombre de vidéos dans l'ensemble apprentissage (voir Fig. 6.5). On apprend ensuite N_g classifieurs : pour chacun des N_g groupes, on apprend par exemple un classifeur SVM basé sur l'intégralité de l'ensemble d'apprentissage moins le groupe en question (apprentissage "Leave-one-out").
Dans la seconde étape, on teste chacun des classifieurs appris sur le groupe qui a été exclu de l'apprentissage. Pour chaque groupe, les blocs temporels mal classés sont déplacés dans une nouvelle classe que l'on nomme "classe de rejet".
Dans la dernière étape, un dernier classifieur est appris à partir de l'ensemble des blocs de la base d'apprentissage, en tenant compte de la classe supplémentaire de rejet.

Lors des tests, chaque bloc d'une vidéo à classer est soumis au classifieur séparément. On choisit alors de ne pas tenir compte des blocs dont le label proposé est la classe de rejet. On effectue ensuite une opération de fusion (par exemple un vote) sur les labels proposés des blocs temporels restants pour déterminer l'étiquette de la séquence vidéo dans sa globalité.

Il peut arriver que l'ensemble des blocs se voient attribuer le label de rejet. Cela peut signifier que les descripteurs sont peu séparables. Dans ce cas, il est possible d'utiliser une décision plus douce (différente de bien/mal classé) pour la création de la classe de rejet. On peut par exemple utiliser un seuil de probabilité : si la vraie classe a une probabilité inférieure à un seuil donné alors on affecte la classe de rejet.



Une itération d'apprentissage pour un groupe dans la première étape. (vert = bien classé, rouge = mal classé)



Un signal de la vidéo 1 est déplacé dans la nouvelle classe "rejet" pendant la seconde étape.

FIGURE 6.5 – Exemple d'application de la méthode de classification avec rejet à partir de 6 vidéos et 3 classes d'actions A, B and C. L'ensemble d'apprentissage est réparti en 6 groupes (un par vidéo) et un classifieur est appris en excluant le groupe associé à la vidéo 1. Chaque bloc temporel mal classé pendant cette étape est assigné à la classe de "rejet". A la fin de cette étape, l'ensemble d'apprentissage est utilisé en considérant la classe supplémentaire "rejet" pour obtenir le classifieur final.

6.5 Test de la méthode sur la base UCF-10

La méthode décrite ici est en cours d'implémentation et de test sur la base de données UCF-10 [MS08]. Celle-ci est constituée de 10 classes et de 150 vidéos de résolution 720×480 pixels durant entre 2 et 14 secondes. Les classes sont déséquilibrées et contiennent entre 6 et 22 vidéos. Les actions considérées sont liées au sport ("équitation", "course", "golf" ..).

Comparativement aux bases de données images utilisées, cette base a un ensemble d'apprentissage assez petit. Ce problème est également renforcé par le fait qu'aucun ensemble de test n'est clairement défini rendant les comparaisons avec la littérature imprécises.

Pour les premiers tests, nous avons conservé les architectures utilisées précédemment (Tableau 5.6) et nous avons fixé la taille des régions 3D à $32 \times 32 \times 3$ pixels.

Les résultats obtenus à ce jour sont peu probants avec un taux de bonne classification d'environ 70% mais des améliorations restent possibles.

Conclusion et perspectives

Même si des progrès fulgurants ont été réalisés ces dernières années surtout avec les réseaux de neurones profonds, le problème de classification de grandes bases de données d'images et de vidéos reste un problème ouvert, particulièrement quand les classes sont de haut niveau sémantique.

Dans cette thèse, nous avons proposé de décrire des images et des vidéos à l'aide de descripteurs obtenus grâce à des dictionnaires. Après avoir présenté les mécanismes d'apprentissage de dictionnaires dits classiques, nous nous sommes intéressés aux différentes méthodes permettant d'apprendre des dictionnaires adaptés à la classification. Souvent employées de manière non supervisée, ces méthodes n'exploitent pas l'intégralité de l'information disponible a priori. C'est pourquoi nous avons étudié différentes formulations pour modifier l'apprentissage du dictionnaire en un apprentissage supervisé.
Jusqu'à présent, ces dernières formulations avaient été utilisées dans le cadre d'une couche unique de dictionnaire. Cependant, un tel traitement a le désavantage de ne pas pouvoir exploiter efficacement l'information locale dans les images. Nous avons donc étendu la formulation d'apprentissage supervisé pour fonctionner sur des patchs d'images. Nous avons ensuite proposé une architecture multicouche pour fusionner l'information extraite des différents patchs à travers l'ajout de couches de dictionnaires supplémentaires. Chacun des dictionnaires est optimisé pour la tâche de classification grâce à un algorithme similaire à la rétro-propagation que nous avons implémenté.
Pour extraire des descripteurs, une image est découpée en patchs qui sont ensuite encodés sur un dictionnaire et ces codes sont à nouveau découpés en patchs avant d'être de nouveau ré-encodés. Ces dernières opérations peuvent être effectuées plusieurs fois, ce qui forme les différentes couches. Cette succession d'opérations permet d'obtenir des descripteurs de petite taille et a priori de plus en plus discriminants.
La méthode proposée a été testée sur plusieurs bases publiques de classification d'images et a fourni de très bons résultats. Nous avons commencé par évaluer la méthode sur la célèbre base MNIST constituée d'images numérisées de chiffres, sur laquelle les perfomances obtenues ont dépassé l'état de l'art des méthodes similaires. Nous avons ensuite enchaîné des tests sur la base CIFAR-10, un peu plus complexe car constituée d'images réelles, où nous avons également surpassé les méthodes similaires. Pour finir, nous nous sommes tournés vers la base d'images STL-10 sur laquelle, sans effectuer de réapprentissage et en utilisant les dictionnaires appris pour la base CIFAR-10, nous avons obtenu de bonnes performances en classification. Les résultats ainsi que la faible taille des descripteurs utilisés montre les capacités de la méthode proposée à extraire et à condenser les informations pertinentes dans les signaux étudiés.
Enfin, nous avons également proposé une extension de la méthode pour le traitement de vidéos. Cette extension est assez naturelle car les objets à étudier (les vidéos) ne sont pas conceptuellement différents des images et nous avons donc proposé une première étude sur la reconnaissance d'actions d'humains dans les vidéos.

Perspectives

Pour poursuivre ces travaux, différentes pistes s'offrent à nous. La première piste concerne les problèmes de temps de calcul qui ont limité l'étude des architectures et de leur paramétrisation. Comme l'algorithme proposé a été implémenté sous MATLAB, il a été difficile d'explorer la diversité des architectures possibles à cause des grands temps d'apprentissage. Cependant, on peut noter que l'algorithme proposé comporte de nombreuses opérations redondantes, par exemple l'encodage et le traitement de chacun des patchs à chacune des couches de l'architecture ou encore le traitement de chacune des images de la base d'apprentissage. Ces opérations doivent être facilement parallélisables et susceptibles de bénéficier d'une accélération avec un traitement sur processeurs graphiques (GPU).

Il serait également intéressant de continuer d'explorer l'effet du choix de la structure sur les performances. Dans nos expériences, nous nous sommes restreints à un petit nombre de couches à cause des problèmes de calculs évoqués précédemment. Cependant, les expériences montrent que les performances peuvent varier grandement et une meilleure compréhension de l'effet lié aux paramètres du système est nécessaire. Dans un premier temps, nous souhaiterions augmenter le nombre de couches de dictionnaires utilisées pour observer les effets sur les performances : en effet, quelques indices (voir Chapitre 5) suggèrent une possible amélioration des performances. Par ailleurs, il serait également possible de tester assez simplement des architectures différentes, comme des couches de dictionnaires inter-connectées à différents niveaux, la formulation proposée étant flexible.

Actuellement, l'approche par réseaux convolutifs dépasse clairement les autres méthodes, et particulièrement celle basée sur l'apprentissage de dictionnaires. Cependant, pour être efficace, elle nécessite de grandes bases d'apprentissage et qu'il est parfois difficile d'acquérir. Une autre piste possible serait de comparer les performances de la méthode proposée avec les architectures basées sur des réseaux de neurones convolutifs dans le cas où le nombre de données en apprentissage est peu important. Les expériences ont semblé montrer que les dictionnaires étaient plus robustes dans le cas où la base d'apprentissage est réduite. Il serait intéressant de confirmer cette idée d'une part par des tests en limitant la base d'apprentissage, d'autre part en analysant les informations sous-jacentes introduites par la décomposition en patchs qui pourrait expliquer cette différence.

Dans notre architecture, le fait de décomposer l'image en patchs et d'encoder ces patchs indépendamment ne permet pas de tirer profit des relations locales entre les patchs proches. Dans les articles [BEL13], [HHW15], les auteurs proposent une formulation convolutive du problème d'apprentissage et d'encodage sur un dictionnaire. Une telle formulation permettrait de produire des codes encore plus parcimonieux car la reconstruction d'un certain pixel serait affectée non pas par un unique code, mais un ensemble de codes proches. La difficulté est ici d'étendre l'algorithme d'apprentissage pour ce type d'encodage.

Dans le chapitre 2, nous avons évoqué différentes régularisations pour apporter la parcimonie et dans la méthode proposée, nous avons utilisé la norme ℓ_1 et Elastic-Net. La pseudo-norme ℓ_0 a été évoqué pour la parcimonie : l'avantage de pouvoir utiliser cette pseudo-norme ℓ_0 est

qu'il existe des méthode dites "avides" ("greedy") pour trouver une bonne représentation d'un signal sur un dictionnaire. Ces méthodes sont bien plus rapides que la méthode utilisée dans nos travaux. Cependant, elle avait été écartée à cause des difficultés pour calculer le gradient des algorithmes d'encodage (comme "Orthogonal Matching Pursuit"). Il s'avère que les méthodes de différentiation automatique [Gri88] sont particulièrement adaptées pour de telles tâches. La différentiation automatique est une technique qui permet d'évaluer des dérivées de fonctions implémentées par des programmes informatiques. Cette méthode s'appuie sur le fait qu'un programme informatique repose sur l'exécution d'une séquence d'opérateurs de calculs et l'appel à des fonctions de base. En appliquant de manière automatique les règles de dérivation sur les fonctions composées, une dérivée peut être obtenue pour un coût de calcul égal à celui de la fonction elle-même. Quelques tests préliminaires ont été réalisés sur "Matching Pursuit" (et non OMP) et une dérivée a été obtenue avec succès : cette méthode pourrait être une solution pour remplacer la norme ℓ_1 par la pseudo-norme ℓ_0 dans notre algorithme.

Enfin, la transposition de la méthode pour la classification de vidéos pour la reconnaissance d'actions nécessite encore quelques adaptations. On pourrait par exemple améliorer la robustesse du détecteur de parties en faisant du suivi de cible ou encore même retravailler l'architecture pour s'affranchir du détecteur.

Liste des publications

1. S. Chan Wai Tim, M. Rombaut et D. Pellerin. Multi-layer Dictionary Learning for Image Classification, *Advanced Concepts for Intelligent Vision Systems*, Lecce, Italie (2016)
2. S. Chan Wai Tim, M. Rombaut et D. Pellerin. Rejection-based classification for action recognition using a spatio-temporal dictionary, *EUSIPCO*, Nice, France (2015)
3. S. Chan Wai Tim, M. Rombaut et D. Pellerin. Dictionary of Gray-Level 3D Patches For Action Recognition, *IEEE International Workshop on Machine Learning for Signal Processing*, Reims, France (2014)
4. S. Chan Wai Tim, M. Rombaut et D. Pellerin. Adaptive Initialization of a EvKNN Classification Algorithm, *Belief Functions : Theory and Applications*, Compiègne, France (2012)

Bibliographie

[AEB06] M. Aharon, M. Elad et A. Bruckstein. "K-SVD : an algorithm for designing overcomplete dictionaries for sparse representation". In : *IEEE Transactions on Signal Processing* 54.11 (2006), p. 4311–4322.

[Bao+16] C. Bao et al. "Dictionary learning for sparse coding : algorithms and convergence analysis". In : *IEEE transactions on pattern analysis and machine intelligence* 38.7 (2016), p. 1356–1369.

[Bar13] Q. Barthelemy. "Representations parcimonieuses pour les signaux multivaries". Thèse de doct. Universite de Grenoble, 2013.

[BB08] D. M. Bradley et J. A. Bagnell. "Differential Sparse Coding". In : *NIPS* (2008).

[BDE09] A. Bruckstein, D. Donoho et M. Elad. "From sparse solutions of systems of equations to sparse modeling of signals and images". In : *SIAM Review* 51 (2009), p. 34–81.

[BEL13] H. Bristow, A. Eriksson et S. Lucey. "Fast Convolutional Sparse Coding". In : *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2013, p. 391–398.

[Bis06] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[BM13] J. Bruna et S. Mallat. "Invariant Scattering Convolution Networks". In : *IEEE transactions on pattern analysis and machine intelligence* 35.8 (2013), p. 1872–1886.

[BM98] F. Bergeaud et S. Mallat. "Matching pursuit of images". In : *Wavelet Analysis and Its Applications* 7 (1998), p. 285–300.

[Bot10] L. Bottou. "Large-Scale Machine Learning with Stochastic Gradient Descent". In : *Proceedings of COMPSTAT'2010*. 2010, p. 177–186.

[Bot12] L. Bottou. *Stochastic Gradient Descent Tricks*. Rapp. tech. Microsoft Research, 2012.

[BPL10] Y-L. Boureau, J. Ponce et Y. LeCun. "A Theorical Analysis of Feature Pooling in Visual Recognition". In : *Proceedings of the 27th international conference on machine learning (ICML-10)*. 2010, p. 111–118.

[BS97] A. Bell et T.J Sejnowski. "The "independant components" of natural scenes are edge filters". In : *Vision Research* 37 (1997), p. 3327–3338.

[BTG06] H. Bay, T. Tuytelaars et L. Van Gool. "SURF : Speeded Up Robust Features". In : *European conference on computer vision*. 2006, p. 404–417.

[CLN11] A. Coates, H. Lee et A. Ng. "An Analysis of Single-Layer Networks in Unsupervised Feature Learning". In : *AISTATS*. 2011.

[CN11] A. Coates et A. Ng. "The Importance of Encoding Versus Training with Sparse Coding and Vector Quantization". In : *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*. 2011, p. 921–928.

[CN12] A. Coates et A. Ng. "Learning features representations with K-means". In : *Neural Networks : Tricks of the Trade*. 2012, p. 561–580.

[CRP12] S. Chan Wai Tim, M. Rombaut et D. Pellerin. "Adaptive Initialization of a EvKNN Classification Algorithm". In : *Belief Functions : Theory and Applications*. 2012.

[CRP14] S. Chan Wai Tim, M. Rombaut et D. Pellerin. "Dictionary of Gray-Level 3D Patches For Action Recognition". In : *IEEE International Workshop on Machine Learning for Signal Processing*. 2014.

[CRP15] S. Chan Wai Tim, M. Rombaut et D. Pellerin. "Rejection-based classification for action recognition using a spatio-temporal dictionary". In : *EUSIPCO*. 2015.

[CRP16] S. Chan Wai Tim, M. Rombaut et D. Pellerin. "Multi-layer Dictionary Learning for Image Classification". In : *Advanced Concepts for Intelligent Vision Systems*. 2016.

[CS12] A. Castrodad et G. Sapiro. "Sparse Modeling of Human Action from Motion Imagery". In : *International journal of computer vision* 100.1 (2012), p. 1–15.

[CW11] T. Cai et L. Wang. "Orthogonal Matching Pursuit for Sparse Signal Recovery With Noise". In : *IEEE Transaction on Information Theory*. T. 57. 7. 2011, p. 4680–4688.

[DT05] N. Dalal et B. Triggs. "Histograms of oriented gradients for human detection". In : *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)* 1 (2005), p. 886–893.

[EA06] M. Elad et M. Aharon. "Image Denoising Via Sparse and Redundant Representations Over Learned Dictionaries". In : *IEEE Transactions on Image Processing* 15.12 (2006), p. 3736–3745.

[Efr+04] B. Efron et al. "Least Angle Regression". In : *The annals of Statistics* 32.2 (2004), p. 407–499.

[Fan+08] R.-E. Fan et al. "LIBLINEAR : A library for large linear classification". In : *Journal of Machine Learning Research* 9 (août 2008), p. 1871–1874.

[FDF15] A. Fawzi, M. Davies et P. Frossard. "Dictionary Learning for Fast Classification Based on Soft-thresholding". In : *International Journal of Computer Vision* 114 (2015), p. 306–321.

[Gri88] A. Griewank. *On Automatic Differentiation*. Rapp. tech. Argonne National Laboratory, 1988.

[Gui06] M. Guironnet. "Methodes de Resume de Video a partir d'Informations bas niveau, du Mouvement de Camera ou de l'Attention Visuelle". Thèse de doct. Universite Joseph Fourier - Grenoble 1, 2006.

[Hay98] S. Haykin. *Neural Networks : A Comprehensive Foundation*. Prentice Hall PTR, 1998.

[HHW15] F. Heide, W. Heidrich et G. Wetzstein. "Fast and Flexible Convolutional Sparse Coding". In : *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, p. 5135–5143.

[Hin+12] G. E. Hinton et al. "Improving neural networks by preventing co-adaptation of feature detector". In : *arXiv preprint arXiv :1207.0580* (2012).

[HTF09] T. Hastie, R. Tibshirani et J. Friedman. *Elements of Statistical Learning*. Springer, 2009.

[Jia+14] Y. Jia et al. "Caffe : Convolutional Architecture for Fast Feature Embedding". In : *arXiv preprint arXiv :1408.5093* (2014).

[JLD11] Z. Jiang, Z. Lin et L. Davis. "Learning A Discriminative Dictionary for Sparse Coding via Label Consistent K-SVD". In : *Computer Vision and Pattern Recognition (CVPR)*. 2011, p. 1697–1704.

[KG10] M. Arun Kumar et M. Gopal. "A hybrid SVM based decision tree". In : *Journal of pattern Recognition* 43 (2010), p. 3977–3987.

[KH09] A. Krizhevsky et G. Hinton. *Learning multiple layers of features from tiny images*. Rapp. tech. University of Toronto, 2009.

[KW12] S. Kong et D. Wang. "A Brief Summary of Dictionary Learning Based Approach for Classification". In : *arXiv preprint arXiv :1205.6544* (2012). arXiv preprint arXiv :1205.6544.

[LeC+98a] Y. LeCun et al. "Efficient BackProp". In : *Neural Networks : Tricks of the trade*. Sous la dir. de G. Orr et Muller K. Springer, 1998.

[LeC+98b] Y. LeCun et al. "Gradient-based Learning applied to Document recognition". In : *Proceedings of the IEEE* 86 (1998), p. 2278–2324.

[LHK05] K-C. Lee, J. Ho et D. Kriegman. "Acquiring Linear Subspaces for Face Recognition under Variable Lighting". In : *IEEE Transactions on pattern analysis and machine intelligence* 27.5 (2005), p. 684–698.

[LK14] T-H Lin et HT Kung. "Stable and Efficient Representation Learning with Nonnegativity Constraints". In : *International Conference on Machine Learning (ICML)*. 2014, p. 1323–1331.

[Low04] D. G. Lowe. "Distinctive Image Features from Scale-Invariant Keypoints". In : *International journal of computer vision*. T. 60. 2. 2004, p. 91–110.

[LWQ13] J. Luo, W. Wang et H. Qi. "Group Sparsity and Geometry Constrained Dictionary Learning for Action Recognition from Depth Maps". In : *Proceedings of the IEEE International Conference on Computer Vision*. 2013, p. 809–816.

[Mai+08] J. Mairal et al. "Supervised Dictionary Learning". In : *Advances in neural information processing systems* (2008), p. 1033–1040.

[Mai+09] J. Mairal et al. “Online Dictionary Learning for Sparse Coding”. In : *Proceedings of the 26th annual international conference on machine learning* (2009), p. 689–696.

[Mai+10] J. Mairal et al. “Online Learning for Matrix Factorization and Sparse Coding”. In : *Journal of Machine Learning Research* 11 (2010), p. 19–60.

[Mai10] J. Mairal. “Sparse coding for machine learning, image processing and computer vision”. Thèse de doct. ENS Cachan, 2010.

[Mal09] S. Mallat. *A Wavelet Tour of Signal Processing*. Elsevier, 2009.

[MBP12] J. Mairal, F. Bach et J. Ponce. “Task-Driven Dictionary Learning”. In : *IEEE Transactions on Pattern Analysis and Machine Intelligence* 34.4 (2012), p. 791–804.

[MBP14] J. Mairal, F. Bach et J. Ponce. *Sparse Modeling for Image and Vision Processing*. now, 2014.

[MCS06] J. Milgram, M. Cheriet et R. Sabourin. “"One Against One" or "One Against All" : Which One is Better for Handwriting Recognition with SVMs ?” In : *Tenth International Workshop on Frontiers in Handwriting Recognition*. 2006.

[MES08] J. Mairal, M. Elad et G. Sapiro. “Sparse learned representations for image restoration”. In : *Proc. of the 4th World Conf. of the Int. Assoc. for Statistical Computing (IASC)* (2008).

[MS08] J. Ahmed M. D. Rodriguez et M. Shah. “Action MACH : A Spatio-temporal Maximum Average Correlation Height Filter for Action Recognition”. In : *CVPR*. 2008.

[MZ93] S. Mallat et Z. Zhang. “Matching Pursuits with Time-Frequency Dictionaries”. In : *IEEE Transactions on Signal Processing* 41.12 (1993), p. 3397–3415.

[OF96] B. A. Olshausen et D. J. Field. “Emergence of simple-cell receptive field properties by learning a sparse code for natural images”. In : *Nature* 381.6583 (1996), p. 607–609.

[OM15] E. Oyallon et S. Mallat. “Deep Roto-Translation Scattering for Object Classification”. In : *arXiv preprint arXiv :1412.8659v2* (2015).

[PB13] N. Parikh et S. Boyd. *Proximal Algorithms*. 2013.

[Ram07] E. Ramasso. “Reconnaissance de sequences d’etats par le Modele des Croyances Transferables. Application a l’analyse de videos d’athletisme.” Thèse de doct. Universite Joseph Fourier - Grenoble 1, 2007.

[RBE09] R. Rubinstein, A. Bruckstein et M. Elad. “Dictionaries for Sparse Representation Modeling”. In : *IEEE Proceedings*. T. 98. 6. 2009, p. 1045–1057.

[RPE13] R. Rubinstein, T. Peleg et M. Elad. “Analysis K-SVD : A Dictionary-Learning Algorithm for the Analysis Sparse Model”. In : *IEEE Transactions on Signal Processing*. T. 61. 3. 2013, p. 661–677.

[RSS10] I. Ramirez, P. Sprechmann et G. Sapiro. "Classification and clustering via dictionary learning with structured incoherence and shared features". In : *Computer Vision and Pattern Recognition (CVPR)* (2010), p. 3501–3508.

[Rus+15] O. Russakovsky et al. "ImageNet Large Scale Visual Recognition Challenge". In : *International Journal of Computer Vision*. T. 115. 3. 2015, p. 211–252.

[SBJ13] M. Sadeghi, M. Babaie-Zadeh et C. Jutten. "Dictionary Learning for Sparse representation : A Novel Approach". In : *IEEE Signal Processing Letters*. T. 20. 12. 2013, p. 1195–1198.

[Sch05] M. Schmidt. *Least Squares Optimization with L1-Norm Regularization*. Rapp. tech. 2005.

[SFR09] M. Schmidt, G. Fung et R. Rosaless. *Optimization Methods for L1-regularization*. Rapp. tech. University of British Columbia, 2009.

[Spr+15] J. T. Springenberg et al. "Striving for Simplicity : The All Convolutional Net". In : *International Conference on Learning Representations (ICLR)*. 2015.

[Suo+14] Y. Suo et al. "Structured Dictionary Learning for Classification". In : *IEEE Transactions on Signal Processing* (2014).

[Tan13] Y. Tang. "Deep Learning using Linear Support Vector Machines". In : *International Conference on Machine Learning (ICML)* (2013).

[Tar+16] S. Tariyal et al. *Greedy Deep Dictionary Learning*. Rapp. tech. arXiv :1602.00203, 2016.

[Tar16] S. Tariyal. "Deep Dictionary Learning". Thèse de doct. IIIT-Delhi Inde, Juillet 2016.

[Tib12] R. Tibshirani. "The Lasso Problem and Uniqueness". In : *Electronic Journal of Statistics* 7 (2012), p. 1456–1490.

[Tib96] R. Tibshirani. "Regression Shrinkage and Selection via the Lasso". In : *Journal of the Royal Statistical Society* (1996), p. 267–288.

[Vap95] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, 1995.

[Vap98] V. Vapnik. *Statistical Learning Theory*. Sous la dir. de Wiley. 1998.

[Wan+10] J. Wang et al. "Locality-constrained Linear Coding for Image Classification". In : *Computer Vision and Pattern Recognition (CVPR)* (2010).

[Wri+09] J. Wright et al. "Robust Face Recognition via Sparse Representation". In : *IEEE Transactions on Pattern Analysis and Machine Intelligence* 31 (2009), p. 210–227.

[YLL11] K. Yu, Y. Lin et J. Lafferty. "Learning Image Representations from the Pixel Level via Hierarchical Sparse Coding". In : *Computer Vision and Pattern Recognition (CVPR)*. 2011, p. 1713–1720.

[YR13] Y. Yang et D. Ramanan. "Articulated Human Detection with Flexible Mixtures of Parts". In : *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.12 (2013), p. 2878–2890.

[YYH10] J. Yang, K. Yu et T. Huang. "Supervised Translation-Invariant Sparse Coding". In : *Computer Vision and Pattern Recognition (CVPR)* (2010), p. 3517–3524.

[ZH05] H. Zhou et T. Hastie. "Regularization and variable selection via the elastic net". In : *Journal of the Royal Statistical Society, Series B* 67 (2005), p. 301–320.

[ZL10] Q. Zhang et B. Li. "Discriminative K-SVD for Dictionary Learning in Face Recognition". In : *Computer Vision and Pattern Recognition (CVPR)* (2010), p. 2691–2698.