UNIVERSITE BRETAGNE LOIRE / MATHSTIC

UNIVERSITÉ DE RENNES 1

THESE DE DOCTORAT DE

L'UNIVERSITE DE RENNES 1
COMUE UNIVERSITE BRETAGNE LOIRE

ECOLE DOCTORALE N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : Signal, Image, Vision

Thierry DUMAS

Deep learning for image compression

Thèse présentée et soutenue à Rennes, le 07 juin 2019.
Unité de recherche : centre de recherche INRIA Rennes Bretagne Atlantique.

Rapporteurs avant soutenance

Murat TEKALP	Professeur, Koç Univ. Istanbul
Peter SCHELKENS	Professeur, Vrije Universiteit Brussel

Composition du Jury

Murat TEKALP	Professeur, Koç Univ. Istanbul
Peter SCHELKENS	Professeur, Vrije Universiteit Brussel
Patrice BRAULT	Ingénieur de recherche, Centrale Supélec
Luce MORIN	Enseignante/chercheuse, INSA Rennes
Christine GUILLEMOT	Directrice de Recherche, INRIA Rennes
Aline ROUMY	Chargée de Recherche, INRIA Rennes

Président

Directrice de thèse

Christine GUILLEMOT	Directrice de recherche, INRIA Rennes

Co-directrice de thèse

Aline ROUMY	Chargée de recherche, INRIA Rennes

Contents

Résumé en français

La quantité d'images et de vidéos transmises a augmenté significativement au cours des vingt dernières années avec le développement de plateformes comme Facebook et Netflix. Et cette tendance va se poursuivre. Par exemple, Cisco a prédit en 2016 que le trafic de contenu vidéo sur internet va se multiplier par quatre entre 2016 et 2021 [1]. Malgré l'accroîssement des capacités de transmission, l'augmentation des contenus images et vidéos exige de meilleures méthodes de compression. Actuellement, le meilleur standard de compression d'images et de vidéos est *High Efficiency Video Coding* (HEVC), aussi appelé H.265. Il a été lancé en 2013. H.265 a remplacé l'ancien standard de compression *Advanced Video Coding* (AVC), aussi appelé H.264, en améliorant les performances de compression par deux. L'effort de standardisation en compression se poursuit toujours. Un nouveau standard, qui sera appelé *Versatile Video Coding* (VVC), est en cours de développement [2].

Principes de la compression d'images

Les standards de compression utilisés couramment sont avec perte. Cela signifie qu'une image est encodée en un train binaire, et le train binaire est décodé en une approximation de l'image d'origine. L'erreur entre l'image d'origine et son approximation est caractérisée par une mesure de distorsion. Un algorithme de compression d'image a pour objectif de fournir le taux de compression le plus petit possible, c'est-à-dire le train binaire le plus petit possible, tout en engendrant la plus petite distorsion possible.

L'étape nécessaire à la compression des échantillons de données est leur quantification. Plus précisément, la plage de valeurs prises par les échantillons de données est transformée en un relativement petit nombre de symboles discrets du côté de l'encodeur. Ensuite, la transformation est approximativement inversée du côté du décodeur afin d'obtenir une approximation des échantillons d'origine. La théorie débit-distorsion montre qu'un quantificateur vectoriel peut induire le plus petit taux de compression parmi tous les codes possibles à une distorsion moyenne maximale donnée, à condition que la taille des vecteurs soit très grande [1]. Malheureusement, la quantification vectorielle n'est pas mise en place dans les algorithmes de compression couramment utilisés à cause de la forte complexité de la quantification vectorielle. Une alternative à la quantification vectorielle est la quantification scalaire. Lorsque le taux de compression est élevé et la

1. https://www.cisco.com/c/dam/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.pdf
2. http://phenix.it-sudparis.eu/jvet/

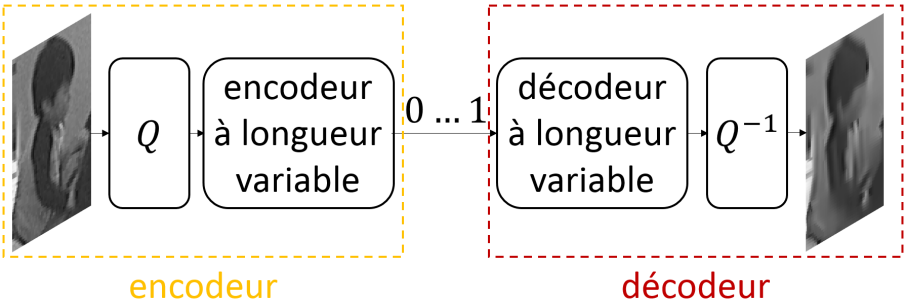


Figure 1: Illustration du quantificateur scalaire uniforme et du codeur à longueur variable sans perte en tant que fondement des standards de compression d'image. Q et Q^{-1} représentent le quantificateur scalaire uniforme du côté encodeur et son inverse approximative du côté décodeur respectivement.

mesure de distorsion est quadratique, un quantificateur scalaire uniforme combiné avec un codeur sans perte à longueur variable fournissent un compromis débit-distorsion 1.53 dB en dessous du meilleur compromis possible. En d'autres termes, c'est 0.254 bits par échantillon supérieur au meilleur compromis débit-distorsion possible. Cette affirmation est valable pour n'importe quelle distribution de données indépendantes et identiquement distribuées [2, 3]. Il est même possible d'aller plus loin. Il est prouvé que, pour n'importe quel taux de compression, lorque la mesure de distorsion est quadratique, sans aucune hypothèse au sujet de la distribution de données, un quantificateur scalaire uniforme avec une codeur sans perte à longueur variable fournissent un compromis débit-distorsion 0.754 bits par échantillon supérieur au meilleur compromis possible [4]. Étant donné ces résultats, l'association d'un quantificateur scalaire uniforme et d'un codeur sans perte à longueur variable est la pierre fondatrice de tous les standards de compression (voir Figure 1).

Le problème au sujet du quantificateur scalaire uniforme est qu'il ne peut pas exploiter les corrélations statistiques entre les différents échantillons de données. Ceci implique que ces corrélations statistiques doivent être prises en compte par le codeur sans perte à longueur variable mentionné ci-dessus. Par exemple, notons $\mathbf{X}$ un vecteur aléatoire discret de taille $m \in \mathbb{N}^*$ qui représente m pixels déjà quantifiés. L'entropie jointe $H(\mathbf{X})$ de $\mathbf{X}$ satisfait

$$H(\mathbf{X}) \leq \sum_{i=1}^{m} H(X_i). \tag{1}$$

$H(X_i)$ désigne l'entropie de la variable aléatoire discrète X_i. Si le codeur sans perte à longueur variable compresse les pixels quantifiés comme des variables mutuellement indépendantes, le taux de compression obtenu est égal à la somme des entropies individuelles. En revanche, si le codeur sans perte à longueur variable compresse jointement les pixels quantifiés, le taux de compression obtenu est égal à l'entropie jointe. Sachant que les pixels d'une image ou d'un bloc d'image sont localement très dépendants (voir Figure 2), (1) révèle que la compression sans perte des pixels quantifiés comme des variables mutuellement indépendantes occasionne une perte importante en performance de compression par rapport à la compression sans perte jointe des pixels quantifiés. Afin d'atteindre le plus petit taux de compression envisageable, c'est-à-dire l'entropie jointe, la fonction de masse de probabilité jointe des pixels quantifiés doit être modélisée, et le

Figure 2: Dépendances entre les pixels d'une image. Deux blocs de pixels dans la même image, chacun représenté dans un rectangle jaune, sont très corrélés.

modèle doit être stocké à la fois du côté encodeur et du côté décodeur. Cependant, l'étape de modélisation est extrêmement complexe. Par ailleurs, le coût de stockage croît exponentiellement avec le nombre de pixels quantifiés. C'est pourquoi le codeur sans perte à longueur variable ne peut pas exploiter seul toutes les corrélations statistiques entre les pixels d'une image, ce qui se traduit en corrélations statistiques entre les pixels quantifiés dans notre exemple précédent.

Une solution est de transformer les pixels en employant une transformée inversible de façon à ce que, suite à l'étape de quantification scalaire uniforme, les coefficients transformés quantifiés sont quasiment indépendants. Notons $\mathbf{Y}$ un vecteur aléatoire discret de taille m représentant les coefficients transformés quantifiés. Sous condition d'une quasi-indépendance des coefficients transformés quantifiés, on peut écrire

$$H\left(\mathbf{Y}\right) \lesssim \sum_{i=1}^{m} H\left(Y_i\right). \tag{2}$$

$H\left(Y_i\right)$ désigne l'entropie de la variable aléatoire discrète Y_i. (2) montre que la compression de chaque coefficient transformé quantifié individuellement en utilisant le codeur sans perte à longueur variable est à peu près aussi efficace que leur compression jointe. C'est pourquoi tous les standards de compression d'image avec perte incluent une transformée inversible, un quantificateur scalaire uniforme et un codeur sans perte à longueur variable (voir Figure 3). Il faut mentionner que les standards de compression d'image avec perte n'emploient pas un quantificateur scalaire uniforme à strictement parlé, mais plutôt certaines de ses variantes. Par exemple, le quantificateur scalaire uniforme dans JPEG2000 possède une zone morte dans laquelle la taille du pas de quantification autour de 0 est doublée [5]. H.265 inclut un quantificateur à reconstruction uniforme [6] dont la partie décodeur est identique à la partie décodeur du quantificateur scalaire uniforme pour une taille de pas de quantification équivalente, et sa fonction du côté encodeur contient un décalage par rapport à la fonction du côté encodeur de la quantification scalaire uniforme. Il faut aussi noter qu'une transformée appliquée à un bloc de pixels pourrait ne pas fournir des coefficients transformés totalement indépendants. Par conséquent, un modèle de contexte complète souvent le codeur sans perte à longueur variable de façon à exploiter les corrélations restantes au sein des coefficients transformés quantifiés [7, 8].

Une autre solution visant à rendre les composantes en entrée du codeur sans perte à longueur variable aussi indépendantes que possible est le codage prédictif. Pour le moment, ignorons la transformée et le codeur sans perte à longueur variable pour simplifier l'explication du codage prédictif. De plus, considérons une séquence d'entrée

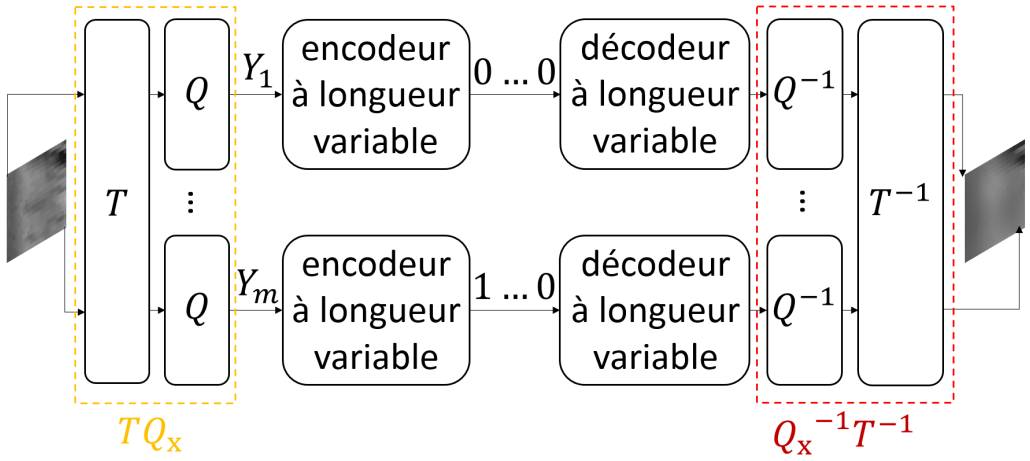
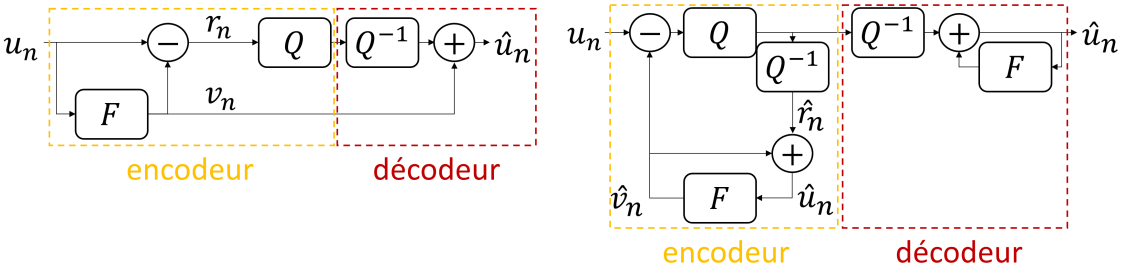
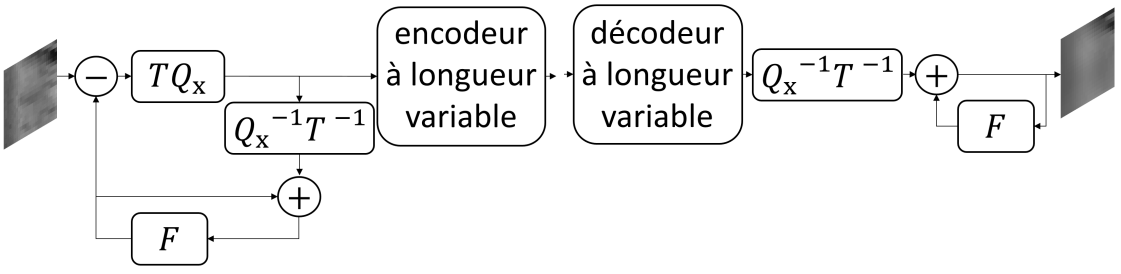


Figure 3: Illustration d'une transformée inversible, un quantificateur scalaire uniforme, et un codeur à longueur variable sans perte en tant qu'amélioration du schéma fondateur affiché dans Figure 1. T et T^{-1} indiquent la transformée directe du côté encodeur et sa rétro-transformée du côté décodeur respectivement.



(a) (b)

Figure 4: Illustration du codage prédictif. Contrairement au schéma montré en (a), le schéma présenté en (b) n'exige pas la transmission de la prédiction de l'entrée actuelle u_n de l'encodeur au décodeur.



(a) (b)

Figure 5: Illustration de (a) l'encodeur et (b) le décodeur pour la compression d'image par l'intermédiaire d'un prédicteur et d'une transformée.

$\{u_i\}_i$. C'est typiquement une séquence de pixels dans le cadre de la compression d'image. Dans la formulation basique du codage prédictif, du côté encodeur, une prédiction v_n de l'entrée actuelle u_n est calculée à partir de la séquence passée en entrée $\{u_i\}_{i=0\ldots n-1}$ par l'intermédiaire du filtre prédictif F. La différence entre u_n et v_n produit le résidu r_n. Le résidu est ensuite quantifié (voir Figure 4a). Mais le problème concernant ce schéma prédictif est que v_n doit être envoyé de l'encodeur au décodeur. Afin d'éviter cet envoi, le filtre prédictif est dupliqué du côté décodeur, et la prédiction de u_n est calculée à partir de la reconstruction $\{\hat{u}_i\}_{i=0\ldots n-1}$ de la séquence passée après encodage et décodage (voir Figure 4b). À noter que JPEG2000 n'implique aucun schéma prédictif. Par contre, H.265 combine un prédicteur et une transformée. Tout d'abord, le prédicteur renvoie un résidu de prédiction qui est aussi décorrélé que possible des pixels précédemment encodés et décodés. Ensuite, la transformée est appliquée au résidu. Le schéma de référence en compression d'image via l'association de la prédiction et de la transformée est résumé dans Figure 5.

Objectif de la thèse

Étant donné l'explication précédente, un élément décisif pour l'obtention d'une compression d'image performante est la capacité du filtre prédictif et de la transformée à fournir au codeur sans perte à longueur variable un vecteur avec des composantes indépendantes.

Le choix de la transformée dans un standard de compression d'image est motivé par des résultats d'optimalité. Cependant, la distribution des pixels d'une image ne se conforme pas exactement aux hypothèses nécessaires pour aboutir à ces résultats. Afin d'approfondir ce point, considérons le schéma de compression d'image contenant une transformée, un quantificateur scalaire uniforme, et un codeur sans perte à longueur variable. Lorsque l'entrée de la transformée est un vecteur Gaussien de moyenne zéro et que la taille du pas de quantification est petite, pour un taux de compression donné, la transformée de Karhunen-Loève (KLT) minimise l'espérance de la distorsion parmi toutes les transformées linéaires [9, Proposition 7.2]. Il est essentiel d'insister sur le fait que le résultat précédent portant sur la KLT n'est pas le résultat classique provenant de la théorie de l'approximation, qui énonce que, lorsque seulement les $k \in \mathbb{N}^*$ premiers coefficients transformés sont préservés, la KLT minimise la distorsion parmi toutes les transformées linéaires [9, Proposition 7.1]. La base de la KLT contient les vecteurs propres de la matrice de covariance des données. Il convient de préciser que la KLT génère des coefficients transformés décorrélés et, puisque les échantillons de données en entrée de la KLT sont jointement Gaussiens, décorrélation implique indépendance. Par conséquent, dans le cas précédent, chaque coefficient transformé quantifié peut être encodé séparément par le codeur sans perte à longueur variable. Dans le contexte de la compression d'image, le problème est que la distribution des pixels d'une image n'est pas connue. La matrice de covariance des pixels doit être estimée et transmise de l'encodeur au décodeur. Afin d'éviter de l'envoyer, la transformée en Cosinus Discrète (DCT) est couramment choisie plutôt que la KLT. La DCT est indépendante des données. Si l'entrée de la transformée suit un processus de Gauss-Markov d'ordre 1 à haute corrélation, la DCT est une approximation fiable de la KLT [9, Section 7.1.1], [10, Section 7.5.3]. Étant donné les deux

points précédents, la DCT est utilisée dans JPEG et H.265. En remarque, il convient de noter que les auteurs dans [11] montrent que, lorsque l'entrée de la transformée suit un processus auto-régressif d'ordre 1 avec des excitations parcimonieuses et que la taille de l'entrée est grande, la transformée en ondelettes fournit des coefficients transformés plus indépendants que ceux fournis part la DCT.

Malheureusement, la distribution des pixels d'une image n'est ni exactement un processus de Gauss-Markov d'ordre 1 à haute corrélation ni un processus auto-regressif d'ordre 1 avec des excitations parcimonieuses. C'est pourquoi le premier objectif de cette thèse est l'apprentissage à partir d'une grande base d'images d'une transformée pour la compression d'image. L'apprentissage est effectué en amont, la transformée apprise étant ensuite enregistrée à la fois du côté encodeur et du côté décodeur de façon à ne transmettre aucune information supplémentaire liée à la transformée.

Par ailleurs, comme les pixels d'une image ne sont pas jointement Gaussiens, décorrélation n'implique pas indépendance. C'est pourquoi l'application de la KLT à des pixels n'engendre pas des coefficients transformés indépendants. Si elle existe, la transformation des pixels qui fournit des coefficients transformés indépendants est probablement non-linéaire. L'algorithme d'apprentissage doît être capable d'approximer cette transformée non-linéaire. Par conséquent, il est nécessaire de se doter d'un algorithme d'apprentissage qui peut approximer quasiment n'importe quelle fonction. La capacité d'approximation universelle est une caractéristique des réseaux de neurones [12, 13, 14]. La transformée pour la compression d'image sera ainsi apprise par l'intermédiaire de réseaux de neurones.

Dans la continuité de ce qui a été mentionné concernant la transformée, l'élaboration des filtres de prédiction ne prend pas en compte la distribution complexe des pixels d'une image. Afin de clarifier ce point, la manière classique d'élaborer des filtres de prédiction est d'abord justifiée en partant du cas de la prédiction optimale. Considérons un pixel, modélisé par une variable aléatoire discrète X, à prédire à partir de pixels voisins précédemment encodés et décodés. Ces pixels voisins sont représentés par un ensemble $\mathcal{B}$ de variables aléatoires observées. La prédiction optimale $\hat{X}^*$ de X, c'est-à-dire la prédiction qui minimise l'espérance de l'erreur de prédiction élevée au carré, est l'espérance conditionnelle $\mathbb{E}[X|\mathcal{B}]$ [10, Section 6.1]. Mais, le nombre de paramètres de la distribution conditionnelle croît exponentiellement avec le nombre de pixels dans l'ensemble $\mathcal{B}$ et la distribution conditionnelle n'est pas stationnaire. Par conséquent, il est difficile de l'estimer et de la transmettre. La manière traditionnelle de corriger ces deux problèmes dans le cadre de la compression d'image est de définir dans un premier temps un ensemble de fonctions de prédiction qui est connu de l'encodeur et du décodeur. Cet ensemble de fonctions de prédiction est fini et fixe. Ensuite, pour un bloc de pixels donné à prédire, l'encodeur trouve la meilleure fonction de prédiction en se basant sur un critère débit-distorsion et l'indice de la fonction de prédiction choisie est envoyé de l'encodeur au décodeur. Cette approche permet d'adapter la prédiction à la texture du bloc à prédire tout en limitant le coût d'envoi du modèle de prédiction. Cependant, deux facteurs, indiqués par (i) et (ii), empêchent cette approche de prédire précisément des blocs d'image avec des textures complexes. (i) Le nombre de fonctions de prédiction est restreint puisque le coût d'envoi du modèle de prédiction est maintenu relativement bas. (ii) Les fonctions de prédiction sont linéaires. Pour illustrer les raisons pour lesquelles (ii) peut poser problème, concentrons nous sur la prédiction intra dans H.265. H.265 a 35 fonctions de prédiction intra linéaires et prédéfinies. Hormis les deux premières fonctions de pré-

diction intra, appelées DC et planaire (voir Figures A.7, A.12, et A.15 dans Appendice A), chaque fonction de prédiction intra consiste à propager les valeurs des pixels voisins précédemment encodés et décodés selon une direction spécifique [15]. Cette approche est appropriée en présence de contours, et donc dans des petites zones contenant des bords orientés [16, 17, 18] (voir Figures A.1 à A.8 dans Appendice A). Par contre, elle échoue dans des larges zones renfermant souvent des textures plus complexes [19, 20, 21] (voir Figures A.9 à A.16 dans Appendice A). Le défi est de construire une fonction de prédiction intra pouvant prédire à la fois de simples textures dans des petits blocs d'image, aussi bien que des textures complexes dans des larges blocs.

Pour relever ce défi, cette fonction de prédiction intra a besoin d'un modèle fiable de la distribution jointe des pixels d'une image. Les réseaux de neurones ont demontré leur capacité à apprendre ce modèle. Par exemple, dans [22, 23], des réseaux de neurones récurrents mettent séquentiellement à jour leur représentation interne des dépendances entre les pixels dans une région connue d'une image et ensuite génère le prochain pixel dans la région inconnue de l'image. Par conséquent, les réseaux de neurones seront utilisés pour apprendre un modèle fiable des dépendances entre un bloc, contenant potentiellement une texture complexe, et ses pixels voisins qui ont été précédemment encodés et décodés.

Structure de la thèse

Ce document peut être divisé en deux parties. La première partie introduit la compression d'image (voir Chapitre 2) et se concentre sur les méthodes les plus récentes portant sur l'apprentissage de la compression d'image via des réseaux de neurones (voir Chapitre 3). Le deuxième partie rassemble les contributions de ce doctorat (voir Chapitres 4 et 5).

Chapitre 2 Ce chapitre fournit des informations préliminaires sur la compression d'image qui seront réutilisées à travers le document. Notamment, la structure de chaque composante clé dans H.265 est justifiée. L'attention est portée sur H.265 pour deux raisons. (i) H.265 est à ce jour le standard de compression d'image [24] et de vidéo [25] le plus performant. (ii) H.265 inclut à la fois un prédicteur intra et une transformée, apparaîssant ainsi comme le fondement le plus pertinent pour développer nos travaux par la suite.

Chapitre 3 La littérature sur les réseaux de neurones est extrêmement vaste. Parmi tous les modèles de réseaux de neurones existants, la famille des réseaux de neurones à propagation avant entraînés par *backpropagation*, qui est la famille d'intérêt dans ce document, est introduite. Puis, ce chapitre analyse deux questions ouvertes essentielles au sujet des réseaux de neurones et explique comment elles ont influencé nos recherches. Enfin, les approches les plus récentes en apprentissage de la compression d'image via des réseaux de neurones sont présentées. Chapitre 3 se termine par une comparaison entre des algorithmes de décomposition parcimonieuse sur un dictionaire et des auto-encodeurs peu profonds et parcimonieux dans le cadre de l'apprentissage d'une transformée pour la compression d'image. Le lecteur est prié de se référer à Section 3.1.1.4 pour une définition des auto-encodeurs. Les résultats de cette comparaison servent de point d'ancrage pour Chapitre 4.

Chapitre 4 Dans ce premier chapitre de contribution, l'apprentissage d'une transformée via les réseaux de neurones pour la compression d'image est étudié. Dans ce contexte, deux problèmes majeurs émergent. (i) La manière traditionelle d'apprendre une transformée est d'utiliser un auto-encodeur, et d'entraîner l'auto-encodeur en minimisant l'erreur de reconstruction. L'auto-encodeur contient habituellement moins de neurones dans la dernière couche de son encodeur que la dimension de son vecteur d'entrée, ce qui signifie que la représentation est sous-complète. Ceci s'appelle un auto-encodeur avec une représentation en goulot d'étranglement. Dans le contexte de la compression d'image, le problème réside dans le fait qu'une transformée apprise via un auto-encodeur avec une représentation en goulot d'étranglement n'est pas efficace en termes débit-distorsion. Il y a deux raisons à cela. Tout d'abord, lors de la phase d'apprentissage, la compressibilité de la représentation est mesurée en utilisant le nombre de coefficients dans la représentation, ce qui est très éloigné du taux de compression obtenu par la quantification de la représentation via un quantificateur scalaire uniforme et ensuite l'encodage du résultat via un codeur sans perte à longueur variable. Ensuite, le fait que la représentation est sous-complète implique que l'espace de code est très restreint. Le premier défi est donc de rendre la transformée apprise efficace en termes débit-distorsion en intégrant une mesure fiable du taux de compression au sein de la phase d'apprentissage et en augmentant le nombre de neurones dans la représentation. (ii) L'apprentissage d'un réseau de neurones, comme n'importe quel modèle de *Machine Learning*, est séparé en deux phases: un graphe est construit et les paramètres du graphe sont appris en utilisant un large ensemble de données d'apprentissage. Par conséquent, un réseau de neurones est dédié à une tâche spécifique après son entraînement. Dans le contexte de la compression d'image, cela implique que le réseau de neurones est entraîné pour un taux de compression donné. Ici, le défi est d'apprendre une transformée unique pour compresser à n'importe quel taux de compression. Ceci est appelé une transformée agnostique quant au taux de compression.

Pour résoudre (i), c'est-à-dire apprendre une transformée efficace en termes débit-distorsion, notre première solution est de forcer, pendant la phase d'apprentissage et la phase de test, une contrainte de parcimonie sur la représentation. Cela signifie que, pour un large sous-ensemble des coefficients dans la représentation, la valeur de chaque coefficient doit être 0. Ensuite, uniquement la position de chaque coefficient quantifié non-nul et sa valeur sont encodés par un codeur sans perte à longueur variable. De cette manière, le taux de compression est exprimé comme une fonction du nombre de coefficients non-nuls dans la représentation, ce qui est toujours une approximation assez grossière du vrai taux de compression. Mais, dans cette approche, la représentation peut-être soit complète soit sur-complète tout en étant compressible. La deuxième solution est d'imposer, pendant la phase d'apprentissage, une contrainte sur l'entropie de la représentation quantifiée. Cette fois-ci, le taux de compression est approximé par l'entropie de la représentation quantifiée. Pour résoudre (ii), c'est-à-dire avoir une transformée agnostique quant au taux de compression, dans le cas de la contrainte de parcimonie, la contrainte varie au cours de la phase d'apprentissage. Dans le cas de la contrainte d'entropie, la taille du pas de quantification est simplement agrandie au cours de la phase de test. La première solution est proche de JPEG en termes de performance débit-distorsion alors que le codeur sans perte à longueur variable utilisé n'a pas de modèle de contexte. La seconde solution fournit de meilleurs performances que JPEG2000 en termes débit-distorsion alors que le codeur sans perte à longueur variable utilisé n'a pas non plus de modèle de contexte.

Chapitre 5 Dans ce deuxième chapitre de contribution, l'apprentissage d'une fonction de prédiction intra via des réseaux de neurones pour la compression d'image est considéré. Un réseau de neurones prédit un bloc de pixels donné à partir de pixels voisins déjà encodés et décodés. Comme pour l'apprentissage d'une transformée via des réseaux de neurones, les problèmes portent sur l'adaptibilité de la fonction apprise. Ici, l'adaptabilité a trois aspects. (i) La fonction apprise doit prédire des blocs de différentes tailles. (ii) Elle doit prédire à partir d'un nombre variable de pixels voisins déjà encodés et décodés. (iii) Le niveau de bruit de quantification dans le voisinage du bloc à prédire pourrait fluctuer.

Afin de résoudre (i), une architecture de réseau de neurones est dédié à la prédiction des blocs de pixels de chaque taille. Concernant (ii), certains des pixels voisins déjà encodés et décodés sont masqués aléatoirement pendant la phase d'apprentissage de façon à ce que le réseau de neurones apprenne le fait qu'une partie de l'information dans son voisinage de pixels pourrait manquer. Par rapport à (iii), nous montrons que la fonction de prédiction apprise a une robustesse au bruit de quantification dans son entrée. En integrant l'ensemble de réseaux de neurones dans H.265, un gain significatif en termes débit-distorsion est observé. 1.46% à 5.20% de bits dans le train binaire est économisé à qualité de reconstruction équivalente.

Chapter 1

Introduction

The amount of transmitted images and videos has grown significantly over the last twenty years with the advent of platforms such as Facebook or Netflix and that trend will carry on. For instance, Cisco predicted in 2016 that the internet video traffic will grow fourfold from 2016 to 2021 [1]. Despite improved broadcast capacities, this increasing amount of images and videos requires increasingly efficient compression methods. The currently most efficient image and video compression standard is High Efficiency Video Coding (HEVC), also called H.265. It was released in 2013. It has replaced the older compression standard Advanced Video Coding (AVC), also called H.264, by improving the compression performance by two. The standardization effort is still pursued. A new standard, to be called Versatile Video Coding (VVC), is currently under development [2].

This introduction first justifies the design of the recent image compression standards. In particular, it explains why the transform and the prediction are crucial to image compression. Based on this, it is presented how neural networks can improve the transform and the prediction, which is the purpose of the thesis.

1.1 Principles of image compression

The commonly used image compression standards are lossy. This means that an image is encoded into a bitstream, and the bitstream is decoded into an approximation of the original image. The error between the original image and its approximation is characterized by a distortion measure. An image compression algorithm aims at giving the smallest possible rate, i.e. the shortest possible bitstream, while providing the smallest possible distortion.

The necessary step for compressing data samples is to quantize them. More precisely, the range of data samples values is mapped to a relatively small number of discrete symbols on the encoder side. Then, the mapping is approximatively inversed on the decoder side to get an approximation of the original data samples. Rate-distortion theory shows that a vector quantizer can achieve the smallest compression rate among all possible codes for a given maximum average distortion, provided that the vector size is very

1. https://www.cisco.com/c/dam/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.pdf
2. http://phenix.it-sudparis.eu/jvet/

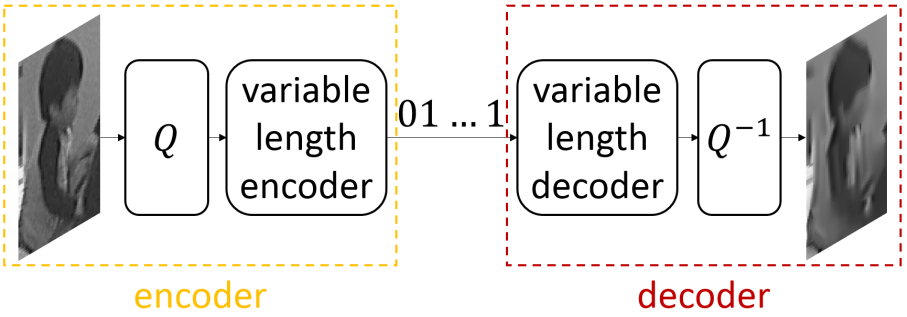


Figure 1.1: Illustration of a uniform scalar quantizer and a variable length lossless coder as baseline for image compression standards. Q and Q^{-1} denote respectively the uniform scalar quantizer on the encoder side and its approximate inverse on the decoder side.

large [1]. Unfortunately, vector quantization is not implemented in common compression algorithms due to its enormous complexity. An alternative to vector quantization is scalar quantization. When the rate is large and the distortion measure is quadratic, a uniform scalar quantizer with a variable length lossless coder is 1.53 dB below the best possible rate-distortion performance. In other words, it is 0.254 bits per sample larger than the best possible rate-distortion performance. This is true for any i.i.d data distribution [2, 3]. Even better, it is demonstrated that, for any rate and when the distortion measure is quadratic, without any assumption regarding the data distribution, a uniform scalar quantizer with a variable length lossless coder is 0.754 bits per sample larger than the best possible rate-distortion performance [4]. Given these results, uniform scalar quantization and variable length lossless coding form the backbone of all image compression standards (see Figure 1.1).

The problem concerning uniform scalar quantization is that it cannot exploit the statistical correlations between data samples. This implies that these statistical correlations have to be considered by the above-mentioned variable length lossless coding. For instance, let $\mathbf{X}$ be a discrete random vector of size $m \in \mathbb{N}^*$ representing a block of m quantized image pixels. The joint entropy $H(\mathbf{X})$ of $\mathbf{X}$ satisfies

$$H(\mathbf{X}) \leq \sum_{i=1}^{m} H(X_i). \tag{1.1}$$

$H(X_i)$ denotes the entropy of the discrete random variable X_i. If the variable length lossless coder compresses the quantized image pixels as mutually independent variables, its rate equals the sum of the individual entropies. However, if the variable length lossless coder jointly compresses the quantized image pixels, its rate equals the joint entropy. Given that the image pixels are highly dependent (see Figure 1.2), (1.1) reveals that the lossless compression of the quantized image pixels as mutually independent variables incurs a substantial loss in terms of compression performance compared to the joint lossless compression of the quantized image pixels. To reach the lowest rate, i.e. the joint entropy, the joint probability mass function of the quantized image pixels must be modeled, and the model must be stored on both the encoder side and the decoder side. But, the modelling step is extremely complex. Moreover, the storage grows exponentially with the number of quantized image pixels. That is why the variable length lossless coder cannot

Figure 1.2: Dependencies between image pixels. Two blocks of pixels in the same image, each in a yellow box, are highly correlated.

exploit alone all the statistical correlations between the image pixels, which translate into the statistical correlations between the quantized image pixels in our previous example.

A solution is to transform the image pixels via an invertible mapping such that, after uniform scalar quantization, the resulting quantized transform coefficients are almost independent. Let $\mathbf{Y}$ be a discrete random vector of size m representing the quantized transform coefficients. Under quasi-independence of the quantized transform coefficients, it holds

$$H(\mathbf{Y}) \lesssim \sum_{i=1}^{m} H(Y_i). \tag{1.2}$$

$H(Y_i)$ denotes the entropy of the discrete random variable Y_i. (1.2) shows that the separate compression of each quantized transform coefficient via the variable length lossless coder is nearly as efficient as their joint compression. That is why all image compression standards involve an invertible transform, a uniform scalar quantizer and a variable length lossless coder (see Figure 1.3). Note that common image compression standards do not use, strictly speaking, uniform scalar quantization, but rather use variants thereof. For instance, the uniform scalar quantizer in JPEG2000 has a dead-zone which doubles the quantization step size around 0 [5]. H.265 includes a uniform reconstruction quantizer [6] whose decoder mapping is identical to the decoder mapping of the uniform scalar quantizer at equivalent quantization step size and its encoder mapping is offset with respect to the encoder mapping of the uniform scalar quantizer. Note also that a transform applied to the image pixels may not yield fully independent transform coefficients. Therefore, a context model usually supplements the variable length lossless coder to exploit the correlations left among the quantized transform coefficients [7, 8].

Another solution to make the components of the input to the variable length lossless coder as independent as possible is predictive coding. Let us ignore for the moment the transform and the variable length lossless coder to simplify the explanation of predictive coding. Let $\{u_i\}_i$ be an input sequence. It is typically a sequence of image pixels in image compression. In the basic form of predictive coding, on the encoder side, a prediction v_n of the current input u_n is computed from the past input sequence $\{u_i\}_{i=0\ldots n-1}$ via the prediction filter F. v_n is subtracted from u_n, yielding the residue r_n. The residue is then quantized (see Figure 1.4a). But, the problem regarding this prediction scheme is that v_n has to sent from the encoder to the decoder. To avoid sending it, the prediction filter is reproduced on the decoder side, and the prediction of u_n is computed from the reconstruction $\{\hat{u}_i\}_{i=0\ldots n-1}$ of the past sequence after encoding and decoding (see Figure

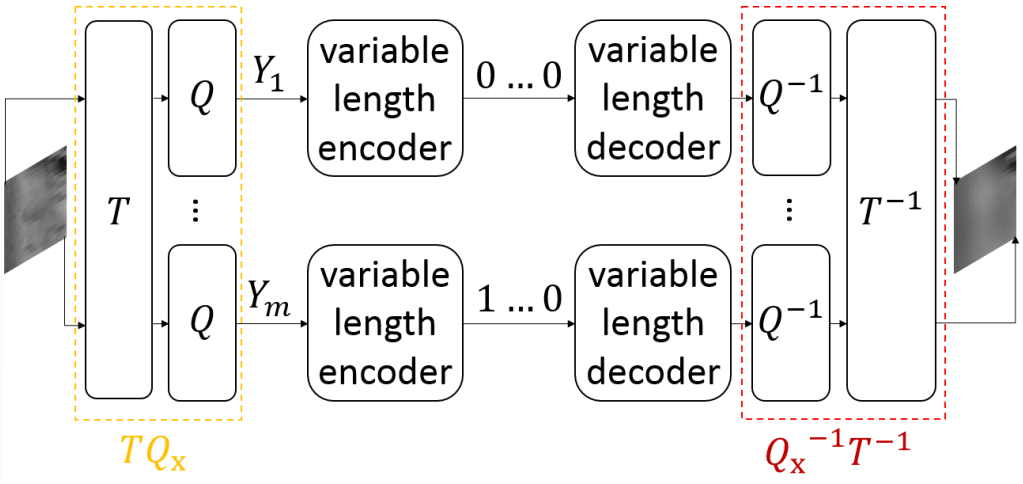
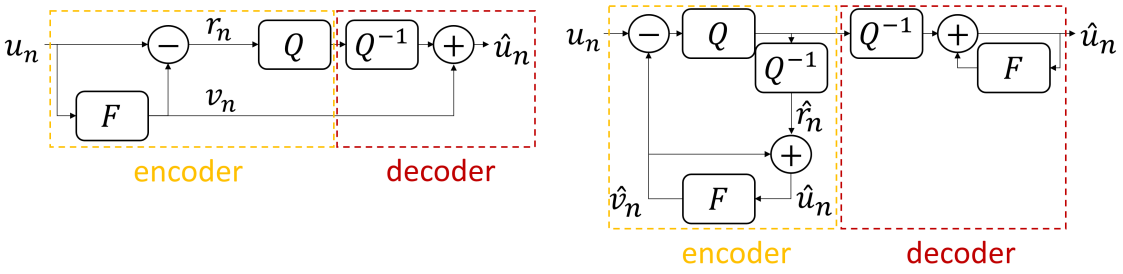
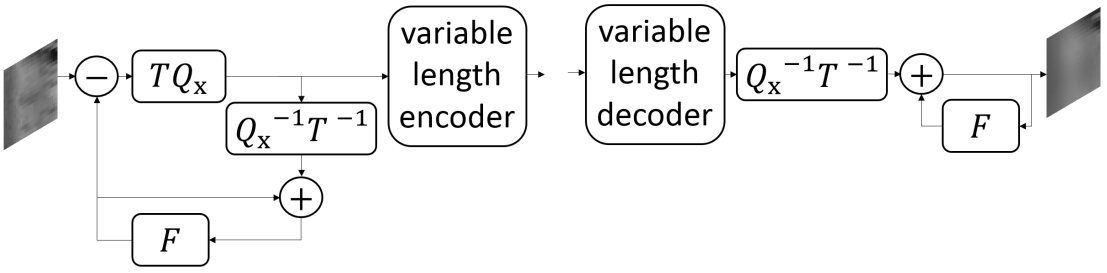


Figure 1.3: Illustration of an invertible transform, a uniform scalar quantizer and a variable length lossless coder as improvement of the baseline in Figure 1.1. T and T^{-1} denote respectively the forward transform on the encoder side and the inverse transform on the decoder side.



(a) (b)

Figure 1.4: Illustration of predictive coding. Unlike the scheme shown in (a), the scheme presented in (b) does not require to transmit the prediction of the current input u_n from the encoder to the decoder.



(a) (b)

Figure 1.5: Illustration of (a) the encoder and (b) the decoder for image compression via the cascade of the prediction and the transform.

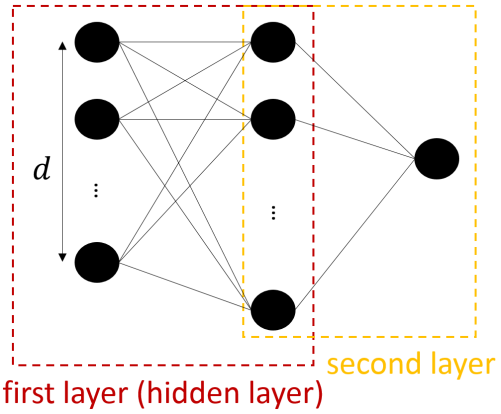


Figure 1.6: Fully-connected feedforward neural network for approximating continuous functions $[0,1]^d \to \mathbb{R}$ in [12]. The activation function σ in the first layer is sigmoidal, meaning that $\lim_{x\to-\infty} \sigma(x) = 0$ and $\lim_{x\to+\infty} \sigma(x) = 1$. The activation function in the second layer is the identity.

1.4b). Note that JPEG2000 does not involve any prediction. In contrast, H.265 combines the prediction and the transform. The prediction first computes a residue of prediction which is as uncorrelated with the past encoded and decoded pixels as possible. Then, the transform applies to the residue. In brief, the baseline for image compression via the prediction and the transform is summarized in Figure 1.5.

1.2 Purpose of the thesis

Given the above explanation, a key for efficient image compression is the capacity of the prediction filter and the transform to provide the variable length lossless coder with inputs of independent components.

The choice of the transform in an image compression standard is justified by optimality results. However, the distribution of the image pixels does not comply with the exact assumptions to derive these results. To develop this point, let us first consider the image compression scheme containing a transform, a uniform scalar quantizer, and variable length lossless coder. When the input to the transform is a zero-mean Gaussian vector and the quantization step size is small, at a given compression rate, the Karhunen-Loève Tranform (KLT) minimizes the expected distortion among all linear transforms [9, Proposition 7.2]. Note that the previous result on the KLT is not the classical result from the approximation theory, which states that, when only the first $k \in \mathbb{N}^*$ transform coefficients are kept, the KLT minimizes the distorsion among all linear transforms [9, Proposition 7.1]. The basis of the KLT is the eigenvectors of the covariance matrix of the data. Note that the KLT leads to uncorrelated transform coefficients, and, as the input samples to the KLT are jointly Gaussian, decorrelation implies independence. Therefore, in the previous case, each quantized transform coefficient can be separately encoded by the variable length lossless coder. The problem in image compression is that the distribution of the image pixels is unknown. The covariance matrix of the image pixels has to be estimated and transmitted from the encoder to the decoder. To avoid sending it, the Discrete Cosine

Transform (DCT) is usually preferred over the KLT. The DCT is data-independent. If the input to the transform follows a Gaussian-Markov process of order 1 with high correlation, the DCT is a close approximation of the KLT [9, Section 7.1.1], [10, Section 7.5.3]. For these two reasons, the DCT is used in JPEG and H.265. As a side note, the authors in [11] show that, when the input to the transform follows an auto-regressive process of order 1 with sparse excitations and the input length is large, the wavelet transform provides more independent transform coefficients than the DCT.

Unfortunately, the distribution of the image pixels is neither exactly a Gaussian-Markov process of order 1 with high correlation nor an auto-regressive process of order 1 with sparse excitations. That is why the first aim of this thesis is to learn from a large image dataset a transform for image compression. The learning is offline, the learned transform being then stored on both the encoder side and the decoder side so that no additional information related to the transform has to be transmitted.

Moreover, as the image pixels are not jointly Gaussian, decorrelation does not imply independence. That is why the application of the KLT to image pixels does not yield independent transform coefficients. If it exists, the transformation of the image pixels that provides independent transform coefficients is likely to be non-linear. The learning algorithm must be able to approximate this non-linear transform. Therefore, a learning algorithm that can approximate almost any function is needed. The capacity of universal approximation is a characteristic of neural networks. This has been well studied in the literature. One of the first result on neural networks as universal approximators states that any continuous function $[0,1]^d \to \mathbb{R}$ can be approximated with arbitrary precision by a fully-connected feedforward neural network composed of a first layer, called hidden layer, with a finite number of neurons and a sigmoidal activation function, and a second layer with one output neuron and the identity as activation function [12] (see Figure 1.6). But, the main issue concerning this result is that the number of neurons in the hidden layer can be unrealistically large. A solution to this problem is given by recent works [13, 14]. They show that, in order to achieve a given approximation error via a fully-connected feedforward neural network while setting a realistic constraint on its number of neurons, its number of layers must be increased. In details, the authors in [13] first fix specific functions to be approximated and an approximation error $\varepsilon \in \mathbb{R}_+^*$ to be reached. Then, they search for a minimum number of layers and a minimum number of neurons of a given type in a fully-connected feedforward neural network, the two numbers depending on ε, such that this fully-connected feedforward neural network is able to achieve ε as approximation error. Amongs other results, the authors in [13] demonstrate that, for any function from a large class of smooth piecewise functions $[0,1]^d \to \mathbb{R}$ to be approximated, using the binary threshold and ReLU [26] as activation functions, a shallow fully-connected feedforward neural network requires exponentially more neurons than its deep counterpart. In [13], *deep* refers to a neural network whose number of layers grows with $1/\varepsilon$. The authors in [14] find a simple function $\mathbb{R}^d \to \mathbb{R}$ for which a fully-connected feedforward neural network with 3 layers and a reasonable number of neurons in each of its first two layers can achieve a given approximation error. However, this approximation error cannot be achieved by any fully-connected feedforward neural network with 2 layers, unless the number of neurons in its first layer grows exponentially with d. Note that the third layer of the 3-layer fully-connected feedforward neural network and the second layer of a 2-layer fully-connected feedforward neural network have one output neuron as the output space

of the function to be approximated is uni-dimensional. The previous result holds for all the standard activation functions in the literature. Given [12, 13, 14], the transform for image compression will be learned via neural networks with multiple layers.

Like the transform, the classic design of the prediction filter does not capture the complex distribution of the image pixels. To clarify this point, the classic design of the prediction filter is first justified by starting from the case of optimal prediction. Let us consider a pixel, modelled by the random variable X, to be predicted from its previously encoded and decoded neighboring pixels. These neighboring pixels are represented as a set $\mathcal{B}$ of observed random variables. The optimal prediction $\hat{X}^*$ of X, i.e. the prediction that minimizes the mean squared prediction error, is the conditional expectation $\mathbb{E}[X|\mathcal{B}]$ [10, Section 6.1]. However, the number of parameters of the conditional distribution grows exponentially with the number of pixels in the set $\mathcal{B}$ and the conditional distribution is not stationary. So it is difficult to estimate and difficult to transmit. The common way to correct these two issues in image compression is to first define a set of prediction functions which is known by the encoder and the decoder. The set of prediction functions is finite and fixed. Then, given an image block to be predicted, the encoder finds the best prediction function according to a rate-distortion criterion and the index of the selected function is transmitted from the encoder side to the decoder side. This approach enables to adapt the prediction to the texture of the image block to be predicted while restricting the cost of sending the prediction model. However, two factors, indicated by (i) and (ii), prevent this approach from predicting accurately image blocks with complex textures. (i) The number of prediction functions is limited as the cost of sending the prediction model is kept low. (ii) The prediction functions are linear. To illustrate why (ii) can be problematic, let us focus on the intra prediction in H.265. H.265 has 35 predefined linear intra prediction functions. Apart from the first two intra prediction functions, called DC and planar (see Figures A.7, A.12, and A.15 in Appendix A), each intra prediction function consists in simply propagating previously encoded and decoded neighboring pixel values along a specified direction [15]. This approach is suitable in the presence of contours, hence in small regions containing oriented edges [16, 17, 18] (see Figures A.1 to A.8 in Appendix A). However, it fails in large areas usually containing more complex textures [19, 20, 21] (see Figures A.9 to A.16 in Appendix A). The challenge is to design an intra prediction function that can predict both simple textures in small image blocks, as well as complex textures in larger ones.

To this end, this intra prediction function needs a reliable model of the joint distribution of image pixels. Neural networks have proved capable of learning this model. For example, in [22, 23], recurrent neural networks sequentially update their internal representation of the dependencies between the pixels in the known region of an image and then generate the next pixel in the unknown region of the image. Therefore, neural networks will be used to learn a reliable model of dependencies between a block, possibly containing a complex texture, and its previously encoded and decoded neighborhood.

1.3 Outline of the thesis

This document can be divided into two parts. The first one gives background on image compression (see Chapter 2) and deals with the state-of-the-art in the learning of

image compression via neural networks (see Chapter 3). The second part gathers the contributions of the PhD (see Chapters 4 and 5).

Chapter 2 This chapter provides background information on image compression. Notably, it justifies the design of the key components of H.265. The focus is on H.265 for two reasons. (i) H.265 is to date the most efficient standardized image [24] and video [25] compression algorithm. (ii) H.265 involves both an intra prediction and a transform, thus being the most relevant foundation for explaining our work later on.

Chapter 3 The literature on neural networks is extremely vast. Among all the existing neural network models, the family of feedforward neural networks trained via backpropagation, which is the family of interest in this document, is introduced. Then, this chapter analyzes two major open issues regarding neural networks and explains how they influence our research. Finally, the state-of-the-art approaches in the learning of image compression via neural networks are reviewed. Chapter 3 ends with a comparison between algorithms for sparse decomposition over a dictionary and shallow sparse autoencoders in the context of the learning of a transform for image compression. Please, refer to Section 3.1.1.4 for a detailed definition of autoencoders. This is the starting point for Chapter 4.

Chapter 4 In this first chapter of contribution, we study the learning of a transform via neural networks for image compression. In this context, two main issues emerge. (i) The traditional way of learning a transform is to use an autoencoder, i.e. a neural network whose encoder turns an input vector into a representation and whose decoder reconstructs the input vector from this representation, and train the autoencoder by minimizing the error of reconstruction. The autoencoder usually has much less neurons in the last layer of its encoder than the dimension of the input vector, meaning that the representation is undercomplete. This is called an autoencoder with a bottleneck representation. In the context of image compression, the problem is that a transform learned via an autoencoder with a bottleneck representation is not efficient in terms of rate-distortion. This has two reasons. First, during the training phase, the compressibility of the representation is measured using the number of coefficients in the representation, which is very different from the rate obtained by quantizing the representation via a uniform scalar quantization and then encoding it via a variable length entropy coder, as it is done in common image compression schemes. Second, the undercomplete representation implies that the code space is very limited. The first challenge is thus to make the learned transform efficient in terms of rate-distortion by integrating a reliable measure of the rate into the training phase and increasing the number of neurons in the representation. (ii) The training of a neural network, as for any other Machine Learning model, is split into two phases: a graph is built and the parameters of the graph are learned on a large dataset. Therefore, a neural network is dedicated to a specific task after its training. For image compression, this means that the neural network is trained to meet a given rate constraint. Here, the challenge is to learn a unique transform for compressing at any rate, which is called *rate-agnostic*.

To solve (i), i.e learning an efficient transform in terms of rate-distortion, our first solution is to enforce, during the training and test phases, a sparsity constraint on the representation. This means that, for a large subset of the coefficients in the representation, the value of each coefficient has to be 0. Then, only the position of each non-zero

quantized coefficient and its value are encoded via a variable length lossless coder. In this case, the rate is viewed as a function of the number of non-zero coefficients in the representation, which is still a coarse approximation of the true rate. But, in this approach, the representation can be either complete or overcomplete while being compressible. The second solution is to set, during the training phase, a constraint on the entropy of the quantized representation. This time, the rate is approximated by the entropy of the quantized representation, which is a more reliable approximation. To solve (ii), i.e. having a *rate-agnostic* transform, in the case of the sparsity constraint, the constraint varies during the training phase. In the case of the entropy constraint, the quantization step size is simply increased during the test phase. The first solution is near JPEG in terms of rate-distortion performance, while using a variable length entropy coder without any context model. The second solution provides better performance than JPEG2000 in terms of rate-distortion, while also using a variable length entropy coder without any context model.

Chapter 5 In this second chapter of contribution, the learning of an intra prediction function via neural networks for image compression is considered. A neural network predicts a given block of image pixels from already encoded and decoded pixels surrounding this block. As for the learning of a transform via neural networks, the issues relate to the adaptibility of the learned function. Here, the adaptibility has three aspects. (i) The learned function has to predict blocks of various sizes. (ii) It has to predict from a variable number of already encoded and decoded neighboring pixels. (iii) The level of quantization noise in these neighboring pixels may vary.

To solve (i), one neural network architecture is dedicated to the prediction of the blocks of each size. For (ii), some of the already encoded and decoded neighboring pixels are randomly masked during the training phase so that the neural network integrates the fact that some information may be missing in the neighborhood of the block to be predicted. Regarding (iii), we demonstrate that the learned function of prediction exhibits robustness to the quantization noise in its input. When integrating the set of neural networks into H.265, a significant gain in terms of rate-distortion is reported. 1.46% to 5.20% of bits in the bitstream are saved for equivalent quality of image reconstruction.

Chapter 2

Overview of H.265

This section gives background information on image compression through an overview of H.265. H.265 was developped by the collaboration between the Video Coding Expert Group (VCEG) from the International Telecommunication Union (ITU-T) and the Moving Picture Experts Group (MPEG) from the International Organization for Standardization/International Electrotechnical Commission (ISO/IEC). H.265 will be presented as an image compression algorithm. This means that the components specific to video compression will not be considered.

2.1 Chrominance subsampling

Before the actual compression of an image via H.265, the first method to reduce the image size consists in converting the image color space into the $Y'C_bC_r$ color space and subsampling the two channels C_b and C_r.

2.1.1 Why use the $Y'C_bC_r$ color space in image compression?

To understand the origin of the $Y'C_bC_r$ color space and the reason for its use in image compression, the notion of color space and its link to the human eye physiology must be first explained.

Under normal light conditions, photoreceptors in the retina of the human eye, called cones, are responsible for color vision [27]. There are three kinds of cone cells, namely S-cones, M-cones, and L-cones. Each cone type has a different spectral sensitivity. S-cones have a peak sensitivity around 440 nanometers (see Figure 2.1). For M-cones and L-cones, the peak sensitivity are around respectively 545 nanometers and 580 nanometers [28]. The color of a light entering the human eye is represented as three cone stimuli. As an example, yellow, near 600 nanometers, is perceived when L-cones are slightly more stimulated than M-cones and much more stimulated than S-cones. In 1924, the Commission Internationale de l'Eclairage (CIE) defined the CIE luminous efficiency function $V(\lambda)$ [29], $\lambda \in \mathbb{R}_+^*$ denoting the wavelength. The CIE luminous efficiency function, dimensionless, is the average spectral sensitivity of human brightness perception under normal light conditions. Two light stimuli with difference radiance spectra $\Phi(\lambda)$ and $\Phi'(\lambda)$, expressed in watts per nanometer, are perceived as equally bright if $\int_0^{+\infty} \Phi(\lambda) V(\lambda) d\lambda = \int_0^{+\infty} \Phi'(\lambda) V(\lambda) d\lambda$.

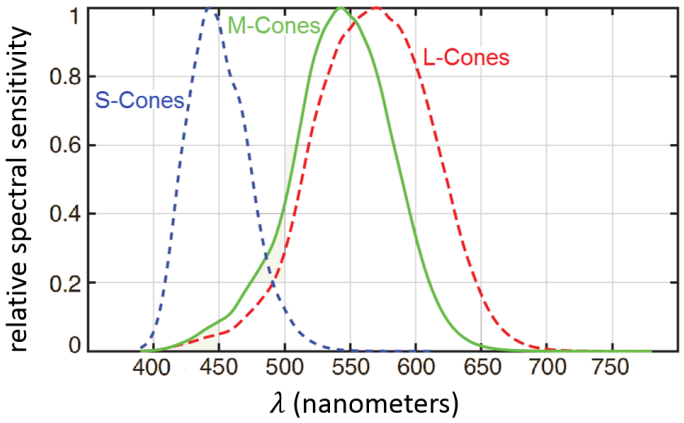
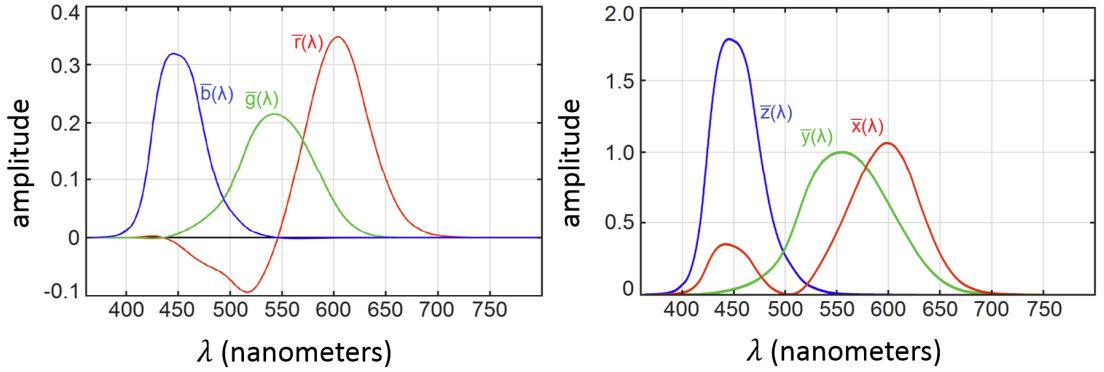


Figure 2.1: Evolution of the relative spectral sensitivity of L-cones, M-cones, and S-cones with the wavelength λ.



(a) (b)

Figure 2.2: Evolution of the CIE 1931 (a) RGB and (b) XYZ color-matching functions with the wavelength λ.

The concept of color space draws on the human eye physiology. A color space is a defined range of colors where each color is described using three channels [1], just as three cone stimuli in the human eye characterize a color. More precisely, each color C can be reproduced using a linear combination of three primaries $\mathcal{R}$, $\mathcal{G}$, and $\mathcal{B}$ with the channels R, G, and B as respective weights,

$$C = \mathrm{R}\mathcal{R} + \mathrm{G}\mathcal{G} + \mathrm{B}\mathcal{B}. \tag{2.1}$$

In 1931, the CIE standardized the CIE 1931 RGB color space with $\mathcal{R}$ at red 700 nanometers, $\mathcal{G}$ at green 546.1 nanometers, and $\mathcal{B}$ at blue 435.8 nanometers. Given this standard, the amount R, G, and B of each respective primary $\mathcal{R}$, $\mathcal{G}$, and $\mathcal{B}$ in the color perceived by the human eye for a given radiance spectrum $\Phi(\lambda)$ must be known. That is why color-matching functions $\overline{r}(\lambda)$, $\overline{g}(\lambda)$, and $\overline{b}(\lambda)$ (see (2.2)) were determined from experiments with human observers [30, 31] and tabulated for wavelengths from 380 to 780 nanometers at intervals of 5 nanometers [32, 33].

$$\mathrm{R} = \int_0^{+\infty} \frac{\Phi(\lambda)}{\Phi_0}\overline{r}(\lambda)\,d\lambda,\ \mathrm{G} = \int_0^{+\infty} \frac{\Phi(\lambda)}{\Phi_0}\overline{g}(\lambda)\,d\lambda,\ \mathrm{B} = \int_0^{+\infty} \frac{\Phi(\lambda)}{\Phi_0}\overline{b}(\lambda)\,d\lambda \tag{2.2}$$

1. Note that some color spaces involve four or more channels. But, these color spaces are less common than those with three channels. For instance, the CMYK color space uses four channels.

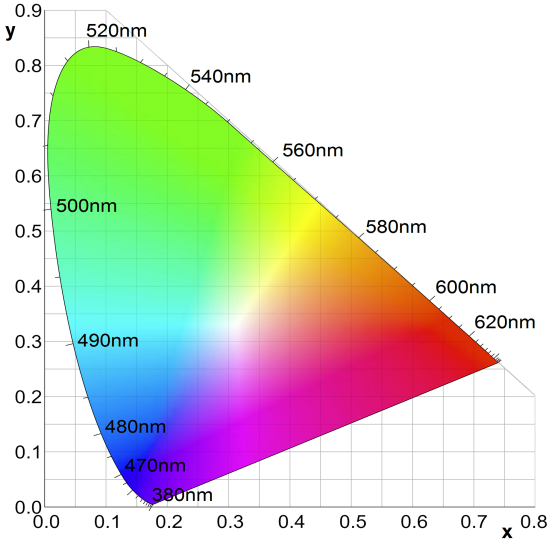


Figure 2.3: CIE 1931 chromaticity diagram. z is not represented as $z = 1 - x - y$.

The reference radiance Φ_0 defines the range of the three channels for the considered viewing condition. Figure 2.2a shows that $\overline{r}(\lambda)$ has negative values inside the range from 435.8 to 546.1 nanometers. This implies that most of the monochromatic lights cannot be represented by a physically mixture of the three standardized primaries. To correct this, the CIE introduced, also in 1931, the CIE 1931 XYZ color space with $\mathcal{X}$, $\mathcal{Y}$, and $\mathcal{Z}$ as primaries and $\overline{x}$, $\overline{y}$, and $\overline{z}$ as color-matching functions (see Figure 2.2b, (2.3), (2.4), and (2.5)).

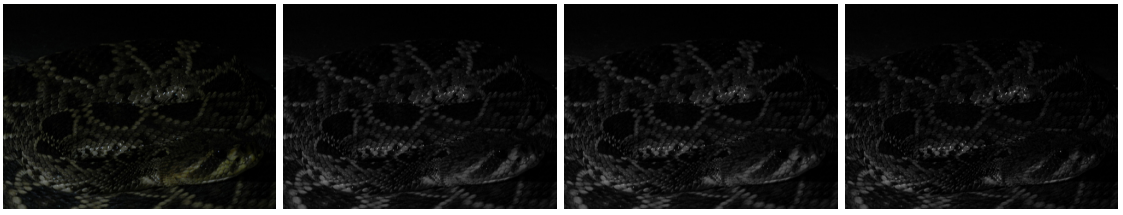
$$C = \mathrm{X}\mathcal{X} + \mathrm{Y}\mathcal{Y} + \mathrm{Z}\mathcal{Z} \tag{2.3}$$

$$\mathrm{X} = \int_0^{+\infty} \frac{\Phi(\lambda)}{\Phi_0}\overline{x}(\lambda)\,d\lambda,\ \mathrm{Y} = \int_0^{+\infty} \frac{\Phi(\lambda)}{\Phi_0}\overline{y}(\lambda)\,d\lambda,\ \mathrm{Z} = \int_0^{+\infty} \frac{\Phi(\lambda)}{\Phi_0}\overline{z}(\lambda)\,d\lambda \tag{2.4}$$

$$\begin{bmatrix}\overline{x}(\lambda)\\ \overline{y}(\lambda)\\ \overline{z}(\lambda)\end{bmatrix} = \frac{1}{0.17697}\begin{bmatrix}0.49000 & 0.31000 & 0.20000\\ 0.17697 & 0.81240 & 0.01063\\ 0.00000 & 0.01000 & 0.99000\end{bmatrix}\begin{bmatrix}\overline{r}(\lambda)\\ \overline{g}(\lambda)\\ \overline{b}(\lambda)\end{bmatrix} \tag{2.5}$$

The derivation of the linear relation in (2.5) is based upon two key conditions, indicated by (i) and (ii). (i) The color-matching functions $\overline{x}(\lambda)$, $\overline{y}(\lambda)$, and $\overline{z}(\lambda)$ have to be non-negative. (ii) $\overline{y}(\lambda)$ has to be equal to the CIE luminous efficiency function $V(\lambda)$. Note that (ii) will be essential when considering image compression in the next paragraph. The details concerning the remaining conditions for finding this linear relation can be found in [34]. Note also that a common visualization focusing on the ratio between the channels X, Y, and Z is the CIE 1931 chromaticity diagram. The two coordinates x and y in the CIE 1931 chromaticity diagram correspond to respectively X and Y inside the plane X + Y + Z = 1 (see Figure 2.3),

$$x = \frac{\mathrm{X}}{\mathrm{X}+\mathrm{Y}+\mathrm{Z}} \quad y = \frac{\mathrm{Y}}{\mathrm{X}+\mathrm{Y}+\mathrm{Z}} \quad z = \frac{\mathrm{Z}}{\mathrm{X}+\mathrm{Y}+\mathrm{Z}}. \tag{2.6}$$



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

Figure 2.4: Visualization of (a) a natural image, (b) its channel R, (c) its channel G, and (d) its channel B. The three channels are displayed using a grayscale.

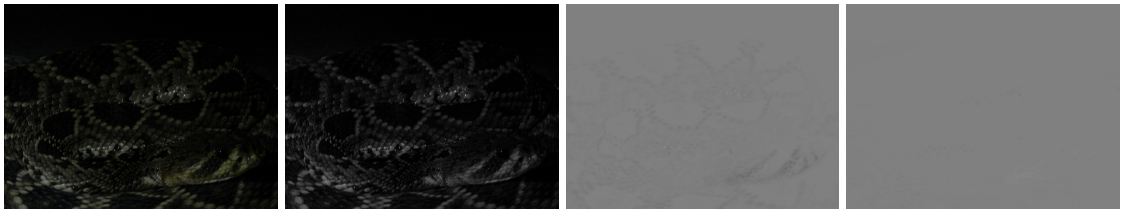
From the CIE 1931 XYZ color space, a RGB color space, different from the CIE 1931 RGB color space, can be created by setting the first primary in the red area in Figure 2.3, the second primary in the green area, the third primary in the blue area, and defining the color when its three channels take their maximum value. Given (2.6), there is a linear relation between the channels X, Y, and Z and the three RGB color space channels. But, neither a RGB color space nor the CIE 1931 XYZ color space is suitable for image compression. Indeed, the three RGB color space channels have redundant information. Put differently, using a RGB color space, if a natural image pixel has a channel with large value, the other two channels also have large values with high probability (see Figure 2.4). In this sense, the CIE 1931 XYZ color space seems to be more appropriate for image compression since, as said in the previous paragraph, the color-matching function $\overline{y}(\lambda)$ is equal to the CIE luminous efficiency function $V(\lambda)$, i.e. the luminance information is decorrelated from the color information. But, the fact that the CIE 1931 XYZ color space has non-uniformly distributed colors [35], i.e. the Euclidean distance between two points in the CIE 1931 color space is not a valid metric for the color change perceived by the human eye, raises another problem for image compression. Let us illustrate this via an example. In Figure 2.3, the green area is larger than the blue area. As a consequence, if a compression algorithm encodes and decodes a pixel in the green area, making a small error of reconstruction for its channel X for instance. The same compression algorithm encodes and decodes a pixel in the blue area, making the same error of reconstruction for its channel X. Then, the color change due to the compression perceived by the human eye will be more pronounced for the pixel in the blue area. That is why color spaces with uniformly distributed colors are preferred in image compression. The $\mathrm{Y}'\mathrm{C_bC_r}$ color space corrects these two problems simultaneously.

- The $\mathrm{Y}'\mathrm{C_bC_r}$ color space is derived so that its channel Y' is proportional to the channel Y. This implies that the luminance information is decorrelated from the color information.
- It has almost uniformly distributed colors.

Besides, two additional properties are advantageous for image compression.

- The $\mathrm{Y}'\mathrm{C_bC_r}$ color space is defined as a linear transformation of a RGB color space,

$$\begin{aligned} \mathrm{Y}' &= k_r\mathrm{R}' + k_g\mathrm{G}' + k_b\mathrm{B}' \\ \mathrm{C_b} &= \frac{1}{2\left(1-k_b\right)}\left(\mathrm{B}' - \mathrm{Y}'\right) + O \\ \mathrm{C_r} &= \frac{1}{2\left(1-k_r\right)}\left(\mathrm{R}' - \mathrm{Y}'\right) + O. \end{aligned} \tag{2.7}$$



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

Figure 2.5: Visualization of (a) same natural image as in Figure 2.4, (b) its channel $Y^{'}$, (c) its channel C_b, and (d) its channel C_r. The three channels are displayed using a grayscale.

Table 2.1: Subsampling of the two chrominance channels.

$Y^{'}C_bC_r$ $4:4:4$	No subsampling of the two chrominance channels.
$Y^{'}C_bC_r$ $4:2:2$	The two chrominance channels are subsampled horizontally by a factor of two.
$Y^{'}C_bC_r$ $4:2:0$	The two chrominance channels are subsampled both horizontally and vertically by a factor of two.

O is an offset depending on the range of the channels. k_r, k_g, and k_b depend on the RGB color space associated to the $Y^{'}C_bC_r$ color space. $R^{'}$, $G^{'}$, and $B^{'}$ are the result of the so-called gamma encoding of respectively R, G, and B. Gamma encoding is a non-linear transformation to account for the perceptual non-uniformity of luminance and colors [36]. (2.7) implies that the conversion of an image from a RGB color space to its associated $Y^{'}C_bC_r$ color space can be quickly computed.

- The $Y^{'}C_bC_r$ color space belongs to the family of the color opponent spaces [37] as the channels C_b and C_r correspond to a difference between the channel $Y^{'}$ and a channel of the associated RGB color space. This induces a decorrelation of the two color channels.

Gamma encoding is beyond the scope of the study of the $Y^{'}C_bC_r$ color space for image compression. Yet, it is essential to stress that, if the gamma encoding was not applied before the linear transformation in (2.7), the color information would be effectively decorrelated from the luminance information. But, with this gamma encoding, changes in the C_b and C_r channels slightly impact the channel $Y^{'}$. By comparing Figures 2.4 and 2.5, we see the superiority of the $Y^{'}C_bC_r$ color space over its associated RGB color space in terms of channels decorrelation. Given the four reasons listed above, the $Y^{'}C_bC_r$ color space will be systematically used thoughout the document.

2.1.2 Chrominance subsampling as preliminary compression

The human eye perceives more spatial details with brightness discrimination than with color discrimination [38]. That is why the spatial resolution of the channels C_b and C_r can be reduced whereas that of the channel $Y^{'}$ remains unchanged. Table 2.1 summarizes the different types of subsampling for the two chrominance channels. Note that the input to H.265 usually has already subsampled chrominance channels. This means that the measures of the H.265 compression performance do not take into account the compression due to this subsampling.

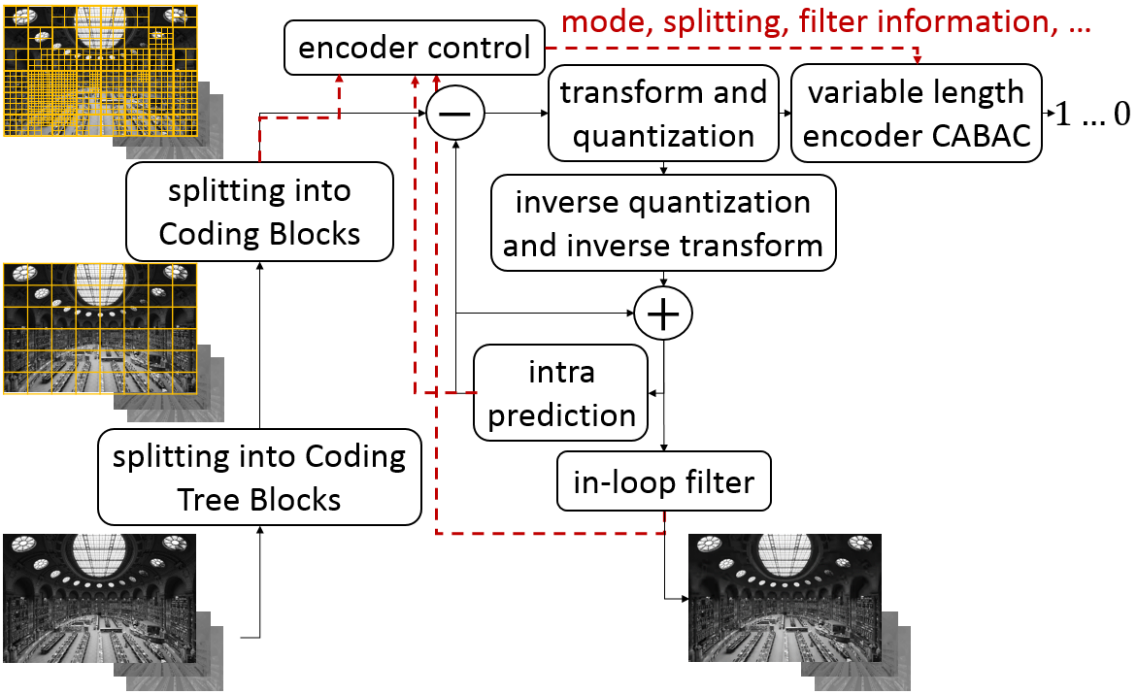


Figure 2.6: Structure of the encoder of H.265 for encoding a $Y'C_bC_r$ image. Note that the splitting into Coding Tree Blocks and the splitting into Coding Blocks is performed on the three channels Y', C_b, and C_r. But, these splittings are illustrated on the channel Y' exclusively for readability. The subsampling of the two channels C_b and C_r is $4:2:0$.

2.2 Global structure of H.265

The structure of H.265 is based on the principle presented in Figure 1.5. Figure 2.6 illustrates the encoder of H.265 as an image compression algorithm. To get the encoder of H.265 as a video compression algorithm, motion compensation and inter prediction [39, 40], i.e. the prediction of a block in the current video frame to be encoded from a block inside an already encoded and decoded video frame, must be added to Figure 2.6.

2.3 Block-based structure

The most important aspect of H.265 is its block-based structure. This means that a $Y'C_bC_r$ image is split into square blocks of different size (see Figure 2.6). Then, one block at a time is encoded and decoded. The reason for using a block-based structure is that it enables the combination of the prediction and the transform, which are two keys for an efficient image compression scheme (see Section 1.2). The splitting of the image into square blocks in H.265, called quadtree structure partitioning, can generate blocks of sizes 4×4, 8×8, 16×16, 32×32, and 64×64 pixels [2]. This choice can only be justified after describing the H.265 intra prediction in Section 2.4. In details, the quadtree structure partitioning involves four types of splitting.

2. In image and video compression, the image size in pixels is written $w \times h$, where $w \in \mathbb{N}^*$ and $h \in \mathbb{N}^*$ denote the image width and height respectively. This convention is adopted throught the document.

- **Coding Tree Block (CTB)**. An image is first split into Coding Tree Units (CTUs) of the same size. A CTU contains a luminance CTB, two chrominance CTBs, and syntax elements. From now on, let us focus on luminance CTBs, simply called CTBs. The differences between luminance CTBs and chrominance CTBs will be pointed out later on. The CTB size is a design parameter. It can take the following values: 64×64, 32×32, or 16×16 pixels [41]. After the first splitting, the CTBs are processed one at a time in raster-scan order, i.e. from left to right and top to bottom.
- **Coding Block (CB)**. Each CTB is further split into CBs (see Figure 2.7). The CTB can be first split into either one or four CBs. Then, if the CTB is split into four CBs, each of them can be recursively split into four CBs. The largest CB size is equal to the selected CTB size. The smallest CB size is a design parameter, which is usually set to 8×8 pixels. The CBs are processed in a Z-scanning order (see Figure 2.7).
- **Prediction Block (PB)**. The PB is the block on which the intra prediction is performed. If the CB size is not equal to the smallest CB size, the PB is similar to the CB. Otherwise, the CB can be split into four PBs. Note that, when dealing with video compression, i.e. when inter-prediction exists, the CB partitioning into PBs has more options [42].
- **Transform Block (TB)**. The TB is the block on which the transform is performed. As for the PBs, the CB can be further split into TBs.

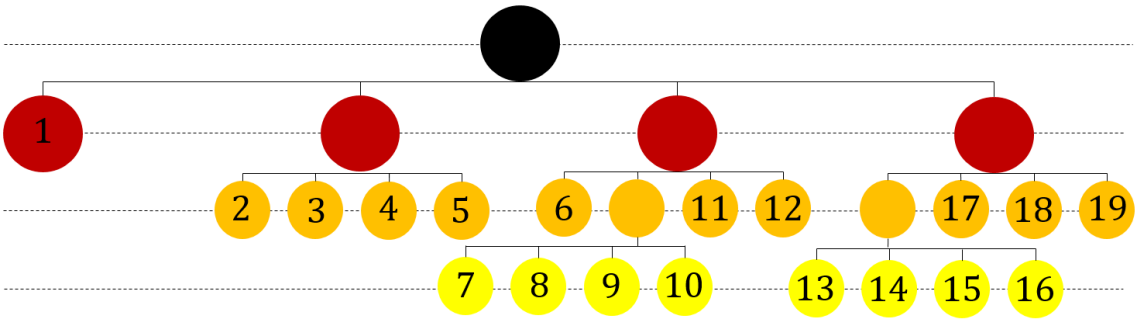
In a CTU, the height of the two chrominance CTBs are smaller than the height of the luminance CTB by a factor depending on the type of subsampling for the two chrominance channels. The same rule goes for the width of the two chrominance CTBs. For instance, in $4:2:0$, the factor for the height and that of the width are equal to 2. As another example, in $4:2:2$, the factor for the height is equal to 1 whereas that for the width is equal to 2. The same sequence of splittings is applied to the luminance CTB and the two chrominance CTBs (see Figure 2.8). But, there exists an exception to this rule. When a luminance CB with the smallest CB size is split into four luminance PBs, the two corresponding chrominance CBs are not split.

It is essential to stress that the common splitting of a luminance CTB and its two corresponding chrominance CTBs on the encoding side has to be signalled to the decoder so that the luminance CTB and its two corresponding chrominance CTBs can be reconstructed on the decoding side. This signalling is illustrated in Figure 2.6 by the gray arrow leaving the component that splits into CBs, going through the encoder control before pointing to the variable length encoder which generates bits. The finer is the splitting, the larger is the signalling cost. To find the optimal splitting among all the possible splittings, a recursive rate-distortion optimization is run on the encoding side [3].

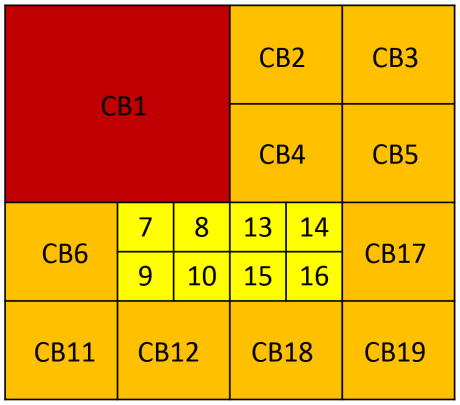
2.4 Intra prediction

Intra prediction consists in creating a prediction for the current PB using previously encoded and decoded pixels in its close neighborhood. Due to the raster-scanning order

3. See the method "TEncCU::xCompressCU" in the file "TEncCu.cpp" of the reference H.265 software at https://hevc.hhi.fraunhofer.de/trac/hevc/browser/tags/HM-16.15/source/Lib/TLibEncoder/TEncCu.cpp for details on this optimization.



(a)



(b)

Figure 2.7: Partitioning of a luminance CTB of size 64×64 pixels into luminance CBs viewed as (a) a tree and (b) blocks.

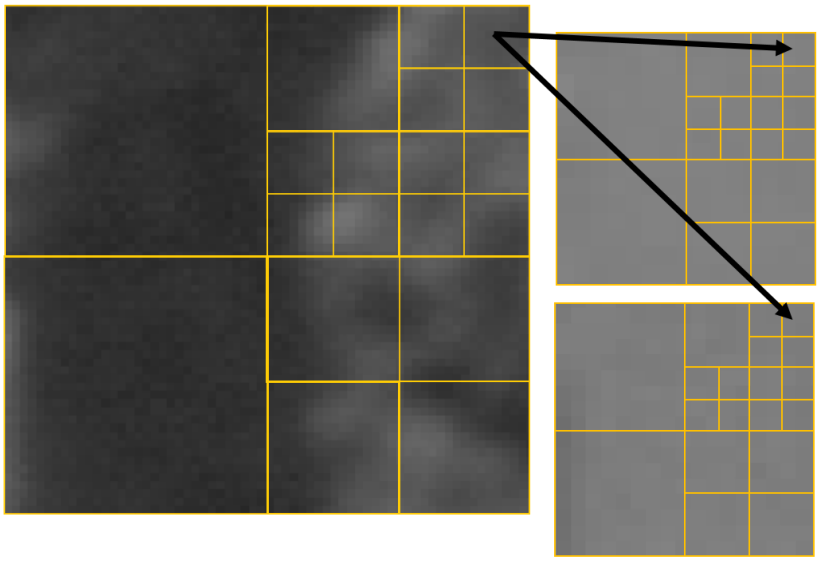
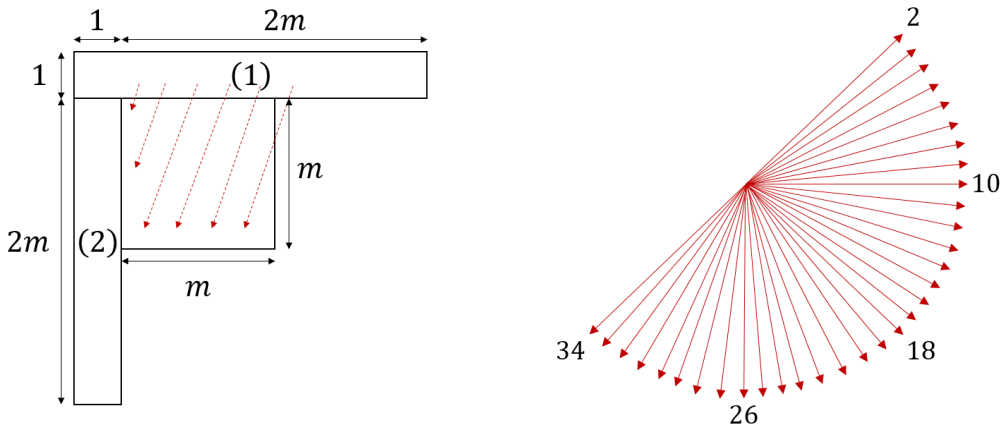


Figure 2.8: Partitioning of a luminance CTB into luminance CBs and its two corresponding chrominance CTBs into chrominance CBs. Here, the type of subsampling for the two chrominance channels is $4 : 2 : 0$. The black arrows show the association between a luminance CB and its two corresponding chrominance CBs.



(a) (b)

Figure 2.9: HEVC intra prediction: (a) the prediction of the current luminance PB from already encoded and decoded pixels via the mode of index 30 and (b) the 33 directional modes. In (a), (1) denotes the group of $2m+1$ already encoded and decoded pixels located above the current luminance PB of width $m \in \mathbb{N}^*$ pixels. (2) refers to the group of $2m$ already encoded and decoded pixels located on the left side of the current luminance PB.

of the CTBs, the Z-scanning order of the CBs, and the quadtree structure partitioning, the used pixels are located above and on the left side of the current PB (see Figures 2.9a). Note that, in Figure 2.9a, when the rightmost pixels in the group of pixels located above the current PB, denoted (1), or the bottommost pixels in the group of pixels located on the left side of the current PB, denoted (2), are not encoded and decoded yet, they are padded from the available ones.

The intra prediction can be derived with several modes. For luminance PBs, HEVC has 35 intra prediction modes: one DC mode, one planar mode, and 33 directional modes. These directional modes consist in propagating the neighboring pixels values in a given direction (see Figure 2.9b). The DC mode simply predicts all the pixels of the current luminance PB with the average value of all neighboring pixels. Finally, the planar mode, designed to prevent the discontinuities along the block boundaries that can occur with the DC mode, averages the horizontal mode of index 10 and the vertical mode of index 26. Unlike the 35 available modes for a luminance PB, only 5 modes can be used for the two corresponding chrominance PBs: DC, planar, horizontal of index 10, vertical of index 26, and the direct mode. The direct mode is the mode selected for the luminance PB.

For each PB, the best intra prediction mode according to a rate-distortion criterion [4] is selected on the encoding side and signalled in the bitstream using only the chosen mode index. This signalling is shown in Figure 2.6 by the gray arrow leaving the intra prediction component, going trough the encoder control before pointing to the variable length encoder which generates bits. On the decoding side, the prediction mode to be used for reconstructing the PB is determined by this index.

4. To avoid overloading the explanation of the H.265 intra prediction with details, this rate-distortion criterion will be developped later on when needed. See the method "TEncSearch::estIntraPredLumaQT" in the file "TEncSearch.cpp" of the H.265 reference software at https://hevc.hhi.fraunhofer.de/trac/hevc/browser/tags/HM-16.15/source/Lib/TLibEncoder/TEncSearch.cpp to understand the criterion exactly.

Now that the mechanism of the intra prediction in H.265 is clarified, the motivations for designing a quadtree structure partitioning that produces PBs of sizes 4×4, 8×8, 16×16, 32×32, and 64×64 pixels can be explained. The quadtree structure partitioning adapts the size of the PBs so that their texture can be fairly accurately predicted by the H.265 intra prediction, while considering the cost of signalling the partitioning. Indeed, small regions in natural images usually contain oriented edges, cf. Section 1.2. As directional modes are designed for predicting oriented edges, the intra prediction in H.265 can yield a very accurate prediction of a small PB. However, the intra prediction in H.265 struggles to predict a large PB with slightly complex textures. That is why, in the presence of a CTB with complex textures, it is worth in terms of quality of the reconstructed CTB to split the CTB into very small PBs, then predict each PB separately during the encoding and the decoding of the CTB. And this even if it requires more bits to signal to the decoder the fine splitting of the CTB. Conversely, in the presence of a CTB with a basic texture, the quality of the reconstructed CTB will be sufficient if the CTB is split into large PBs. This time, the advantage is that the cost of signalling the splitting of the CTB to the decoder is low.

2.5 Transform and quantization

After the prediction step, we obtain a residual CB which can be further split into residual TBs, as explained in Section 2.3. Each of these residual TBs is then transformed via a Discrete Cosine Transform (DCT) (see Figure 2.10). An advantage of the DCT that was not mentioned in Section 1.2 is its separability. The DCT can be computed independently on each line and then on each column of a residual TB, giving the possibility to compute the DCT at low complexity. If the size of a residual TB is 4×4 pixels, a Discrete Sine Transform (DST) is used instead of the DCT as the DST yields a gain in terms of rate-distortion compared to the DCT in the context of intra predictive coding.

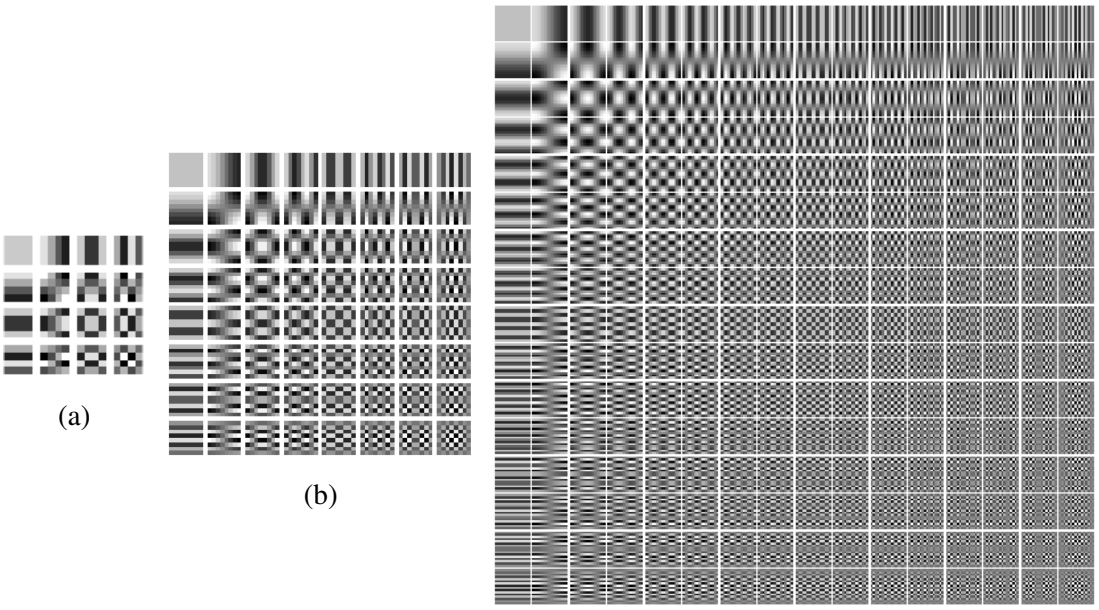
The coefficients resulting from the application of the DCT to a residual TB are scaled to give more weight to lower frequency coefficients and quantized via a uniform reconstruction quantizer. The scaling is justified by the fact that the human eye is much less sensitive to high frequencies than low ones. The fixed quantization step size Δ of the uniform reconstruction quantizer is defined as a function of a Quantization Parameter (QP), $\mathrm{QP}\in[|0,51|]$,

$$\Delta = 2^{\frac{\mathrm{QP}-4}{6}}. \tag{2.8}$$

The amount of distortion and the resulting rate of the block is controlled by this QP value.

2.6 In-loop filtering

When QP is large, a strong quantization is applied to the scaled transform coefficients, generating artifacts such as banding in uniform areas or ringing around edges in the reconstructed image. These artifacts are characterized by the appearance of lines in the reconstructed image that are not present in the original image. Due to the block structure of H.265, the block boundaries are also prone to the appearance of artifacts, referred to



(a) (b) (c)

Figure 2.10: Basis of the DCT applied to (a) 4×4 TBs, (b) 8×8 TBs, and (c) 16×16 TBs.

as blocking since the blocks can be distinguished in the reconstructed image (see Figures 2.11, 2.12, and 2.13).

To attenuate these banding, ringing, and blocking artifacts, two filters are performed during the HEVC encoder: the DeBlocking Filter (DBF) [43] and the Sample Adaptive Offset (SAO) [44]. The DBF reduces the blocking artifacts. It consists in performing an adaptive filtering to the block boundaries. The SAO attenuates the banding and ringing artifacts. It selects regions of the image containing one or multiple CTBs where two SAO types can be applied: Edge Offset (EO) or Band Offset (BO). In both cases, each pixel is classified into a category, 5 for EO and 32 for BO. Then, a common offset is added to each sample depending on its category. The offset is signalled in the bitstream for the decoder. This signalling is indicated in Figure 2.6 by the gray arrow leaving the in-loop filter component, crossing the encoder control before pointing to the variable length encoder. The DBF and the SAO are referred to as in-loop filters since they are included in the encoding and decoding loops.

2.7 Entropy coding

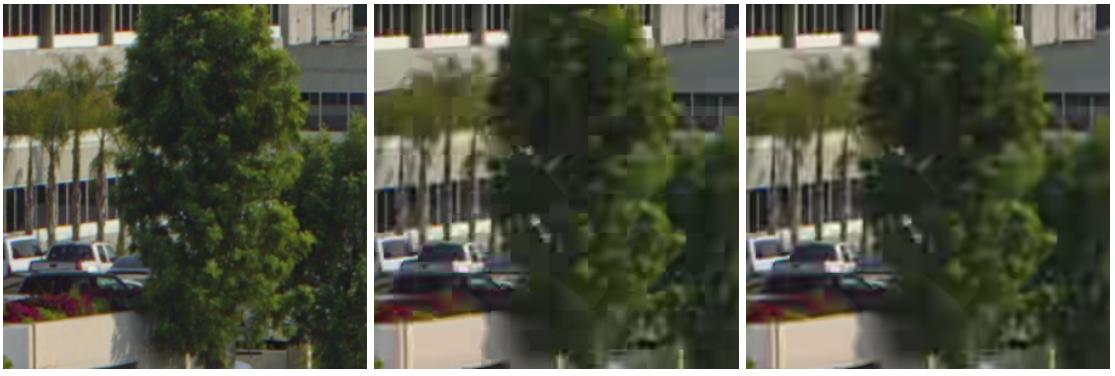
The data to be entropy-coded in H.265 are divided into two categories. The first category contains the quantized transform coefficients resulting from the chaining of the transform of a residual TB and the quantization of the transform coefficients. The second category gathers all the signalling information, i.e., inter alia, the quadtree partitioning of each CTB, the selected intra prediction mode of each PB, and the SAO offsets. The entropy coding of all the signalling information is shown in Figure 2.6 by the gray arrow leaving the encoder control and pointing to the variable length encoder. The entropy

(a) (b) (c)

Figure 2.11: Visualization of blocking artifacts: (a) a 300×300 crop of the first frame of PeopleOnStreet, (b) its reconstruction via H.265 with the DBF turned off, and (c) its reconstruction via H.265 with the DBF turned on. QP $= 42$.

(a) (b) (c)

Figure 2.12: Visualization of blocking artifacts: (a) another 300×300 crop of the first frame of the video PeopleOnStreet, (b) its reconstruction via H.265 with the DBF turned off, and (c) its reconstruction via H.265 with the DBF turned on. QP $= 42$.



(a) (b) (c)

Figure 2.13: Visualization of blocking artifacts: (a) a 300×300 crop of the first frame of the video Traffic, (b) its reconstruction via H.265 with the DBF turned off, and (c) its reconstruction via H.265 with the DBF turned on. QP $= 42$.

coding of the two categories of data is done by a Context-Adaptive Binary Arithmetic Coder (CABAC) [8].

2.7.1 Entropy encoding of the quantized transform coefficients

Let us focus on the quantized transform coefficients to be entropy-encoded by CABAC. More specifically, the beginning of this entropy encoding is considered as it involves an interesting dependency with the intra prediction in H.265.

A key characteristic of the transform coefficients obtained by applying the DCT to a residual TB is that few of them, with relatively large absolute values, concentrate most of the information on the spatial distribution of the residual samples. This stems from the strong energy compaction of the DCT [45]. Due to this, after quantization, the set of quantized transform coefficients is sparse. A way to encode efficiently this set is sparse encoding, i.e. the values of the non-zero quantized transform coefficients exclusively and the position of the quantized transform coefficients within the 2D array of quantized transform coefficients are encoded. The sparse encoding in H.265 has four steps.

1. Encoding a flag indicating whether the 2D array of quantized transform coefficients contains at least one non-zero quantized transform coefficient.
2. Selecting an order for scanning the 2D array of quantized transform coefficients.
3. Encoding the position of the last non-zero quantized transform coefficient.
4. From the position of the last non-zero quantized transform coefficient to the position of the DC coefficient, in reverse scanning order, encoding the values of the non-zero quantized transform coefficients exclusively and the position of the quantized transform coefficients.

Each of the four steps is further detailed below.

1. A flag, called *coded_block_flag*, is encoded to indicate whether the 2D array of quantized transform coefficients contains at least one non-zero quantized transform coefficient. If not, i.e. *coded_block_flag* $= 0$, the sparse encoding stops.
2. The scan enables to convert the 2D array of quantized transform coefficients into a 1D array. The default scan is diagonal (see Figure 2.14). But, for 4×4 and 8×8 arrays of quantized transform coefficients, horizontal and vertical scans are allowed (see Figure 2.15). The choice of scan is determined by the intra prediction mode selected for predicting the PB associated to the residual TB [46]. The vertical scan is used when the intra prediction mode is close to horizontal, i.e. the mode of index 10. The horizontal scan is used when the intra prediction mode is close to vertical, i.e. the mode of index 26. For other intra prediction modes, the diagonal scan is used. This choice aims at exploiting the horizontal or vertical correlation among the residual samples, depending on the intra prediction mode. For instance, in the case of vertical intra prediction mode, the correlation among the residual samples is likely to be vertical. Therefore, the quantized transform coefficients with relatively large absolute values are likely to be clustered in the first few rows of the 2D array of quantized transform coefficients. The previous statement is justified when looking at the DCT basis (see Figure 2.10). In the third step of the sparse encoding, the horizontal scan will reach the last non-zero quantized transform coefficient

earlier than the two other scans, thus saving bits during the fourth step of the sparse encoding.

For 8×8 arrays of quantized transform coefficients with diagonal scan and larger ones, for which the scan is necessarily diagonal, the 2D array of quantized transform coefficients is split into 4×4 subblocks. Then, the scan is nested, i.e. it goes diagonally from one 4×4 subblock to another while, within each subblock, scanning diagonally (see Figure 2.16). Two reasons motivate the splitting of the 2D array of quantized transform coefficients into 4×4 subblocks, indicated by (i) and (ii). (i) The implementation complexity of diagonal scan for the entire 2D array is higher than the 4×4 subblock scan. (ii) The position of the quantized transform coefficients can be encoded in a multi-level approach during the fourth step of the sparse encoding. Note that, in the case of 8×8 arrays of quantized transform coefficients, if the horizontal scan is used, the 2D array is split into 2×8 subblocks instead of 4×4 subblocks. Similarly, if the vertical scan is used, the 2D array is split into 8×2 subblocks instead of 4×4 subblocks (see Figure 2.15).

3. The horizontal and vertical coordinates of the last non-zero quantized transform coefficient in the scanning order are encoded [47] (see Figure 2.17).

4. The scan goes from the position of the last non-zero quantized transform coefficient to the position of the DC coefficient in reverse scanning order. This approach incurs low encoding cost as the last non-zero quantized transform coefficient is usually close to the DC coefficient in terms of spatial positioning. Indeed, as the spatial frequencies of residual TBs are known to be majoritarily low, the quantized transform coefficients with relatively large absolute values are often all grouped near the top-left corner of the 2D array of quantized transform coefficients.

   Each subblock is scanned via up to five passes [48, 49], each pass encoding a different syntax element, denoted from (a) to (e) below.

   (a) *significant_coeff_flag*, a flag indicating whether the current quantized transform coefficient is non-zero. Note that *significant_coeff_flag* indirectly encodes the position of the current quantized transform coefficient.

   (b) *coeff_abs_level_greater1_flag*, a flag indicating whether the absolute value of the current quantized transform coefficient is strictly larger than 1.

   (c) *coeff_abs_level_greater2_flag*, a flag indicating whether the absolute value of the current quantized transform coefficient is strictly larger than 2.

   (d) *coeff_sign_flag*, a flag indicating the sign of the current quantized transform coefficient, 0 meaning positive and 1 negative.

   (e) *coeff_abs_level_remaining*, being the remaining value for the absolute value of the current quantized transform coefficient. Let *coeff_abs_level* denote the absolue value of the current quantized transform coefficient.

$$\begin{aligned} \textit{coeff\_abs\_level\_remaining} = \textit{coeff\_abs\_level} \\ - \textit{significant\_coeff\_flag} \\ - \textit{coeff\_abs\_level\_greater1\_flag} \\ - \textit{coeff\_abs\_level\_greater2\_flag} \end{aligned}$$

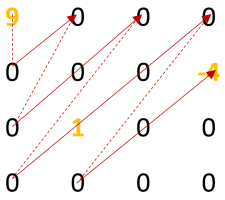
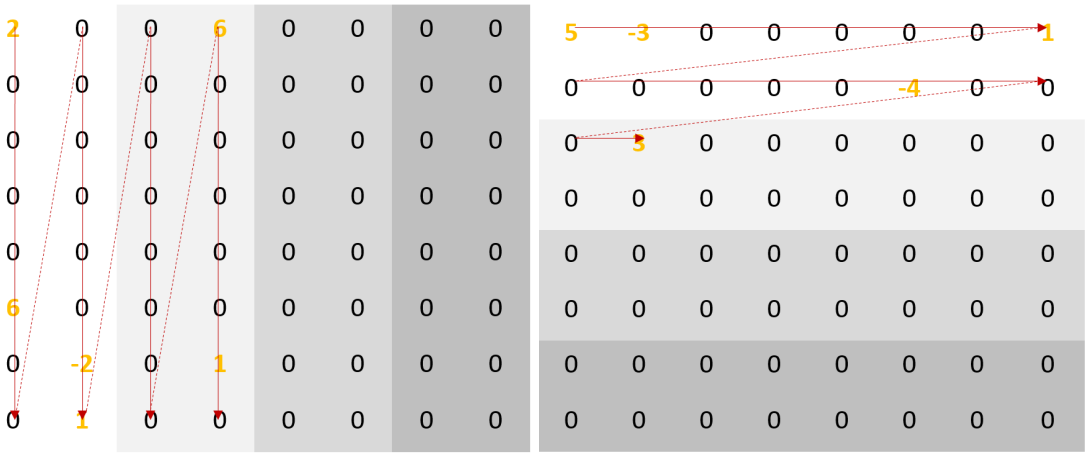


Figure 2.14: Diagonal scan of a 4×4 array of quantized transform coefficients.



(a) (b)

Figure 2.15: Two non-diagonal scans of 8×8 arrays of quantized transform coefficients: (a) vertical and (b) horizontal scans. Each subblock has a background colored with a different gray level.

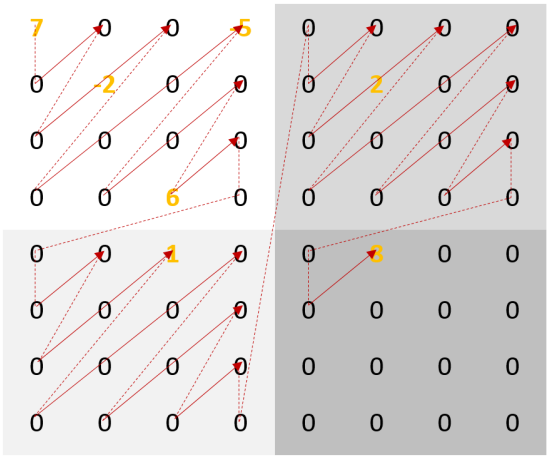


Figure 2.16: Diagonal scan of a 8×8 array of quantized transform coefficients. Each subblock has a background with a different gray level.

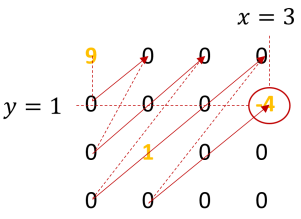


Figure 2.17: Coordinates xy of the last non-zero quantized transform coefficient.

coeff_abs_level_remaining syntax element is encoded if *coeff_abs_level* is strictly larger than 2 or whenever a specified maximum number of *coeff_abs_level_greater1_flag* or *coeff_abs_level_greater2_flag* per subblock is reached.

Note that in each pass, a syntax element is encoded only if necessary. For instance, if a quantized transform coefficient is equal to zero, the syntax elements from (b) to (e) are not encoded.

It is essential to stress that the above-mentioned 5 different flags are intermediate flags. This means that a flag is never written to the bitstream as it is. Each flag undergoes further processings to get their bit code in the bitstream. For example, each *coded_block_flag* goes through a binary arithmetic encoding with context model [5]. Similarly, each *significant_coeff_flag* is encoded in a multi-level fashion, considering this flag at the subblock level [50]. A method called sign data hiding [51] is applied to each *coeff_sign_flag*.

The horizontal and vertical coordinates of the last non-zero coefficient in the scanning order are binarized with a truncated unary prefix and a fixed length suffix [52] before running the binary arithmetic encoding of the prefix with context model. Each *coeff_abs_level_remaining* is binarized using a combination of a truncated Golomb-Rice code [53] and a 0^{th}-order Exponential-Golomb code.

2.7.2 Entropy encoding of the signalling information

The entropy encoding of the signalling information is a topic that involves plenty of implementation details, thus falling outside the scope of a simple overview of H.265 for image compression. Therefore, this topic, and in particular the entropy encoding of the selected intra prediction mode for a given PB, will be explained at a later stage, when needed.

2.8 Quality evaluation

The quality evaluation measures the amount of distortion brought by an image compression algorithm. An image compression standard can be considered better than another one if it gives reconstructed images of better quality for the same compression rate or conversely a lower rate for the same quality of image reconstruction.

The most obvious way of evaluating the quality of image reconstruction is to use subjective experiments, as defined in [54]. In these approaches, a panel of human observers assess the quality of reconstructed images. Then, the results are averaged as a mean opinion score. This way, an estimation of the perceived distortion is obtained. These methods remain the most reliable, yet being generally impractical as a significant number of viewers are needed over a long time. In practice, objective metrics enable to evaluate quickly an image compression algorithm with many different parameters on large sets of images. Below is presented three objective metrics widely used in image compression: the Peak Signal to Noise Ratio (PSNR), the Structural SIMilarity (SSIM), and Bjontegaard's metric.

5. See the method "TEncSbac::codeQtCbf" in the file "TEncSbac.cpp" of the reference H.265 software at https://hevc.hhi.fraunhofer.de/trac/hevc/browser/tags/HM-16.15/source/Lib/TLibEncoder/TEncSbac.cpp for details on this encoding.

2.8.1 PSNR

The PSNR, in decibels (dB), between an image channel $\mathbf{C}$ of height $h_c \in \mathbb{N}^*$ and width $w_c \in \mathbb{N}^*$ and its reconstruction $\hat{\mathbf{C}}$ is derived from the mean-squared error $\text{MSE}\left(\mathbf{C}, \hat{\mathbf{C}}\right)$ between the channel and its reconstruction,

$$\text{MSE}\left(\mathbf{C}, \hat{\mathbf{C}}\right) = \frac{1}{h_c \times w_c} \sum_{i=1}^{h_c} \sum_{j=1}^{w_c} \left(\mathbf{C}\left[i,j\right] - \hat{\mathbf{C}}\left[i,j\right]\right)^2 \tag{2.9}$$

$$\text{PNSR}\left(\mathbf{C}, \hat{\mathbf{C}}\right) = 10 \log_{10} \left(\frac{\left(2^b - 1\right)^2}{\text{MSE}\left(\mathbf{C}, \hat{\mathbf{C}}\right)} \right). \tag{2.10}$$

$\mathbf{C}\left[i,j\right]$ is the pixel at the position $[i,j]$ in the image channel $\mathbf{C}$. b denotes the bit depth of the channel, usually $b = 8$. The PSNR is commonly computed on each of the three channels Y', C_b, and C_r independently.

2.8.2 SSIM

In contrast with the pixelwise PSNR, the SSIM is computed on blocks of the image displaced pixel by pixel. This method detects structural or contrast changes. Since the human visual system is specialized in detecting structural information, this metric well approximates the perceived image quality. The blockwise SSIM is defined for a block $\hat{\mathbf{X}}$ in the reconstructed image and its colocated block $\mathbf{X}$ in the original image,

$$\text{SSIM}\left(\mathbf{X}, \hat{\mathbf{X}}\right) = \frac{\left(2\mu_\mathbf{X}\mu_{\hat{\mathbf{X}}} + c_1\right)\left(2\sigma_{\mathbf{X}\hat{\mathbf{X}}} + c_2\right)}{\left(\mu_\mathbf{X}^2 + \mu_{\hat{\mathbf{X}}}^2 + c_1\right)\left(\sigma_\mathbf{X}^2 + \sigma_{\hat{\mathbf{X}}}^2 + c_2\right)}. \tag{2.11}$$

$\mu_\mathbf{X} \in \mathbb{R}_+^*$ and $\mu_{\hat{\mathbf{X}}} \in \mathbb{R}_+^*$ are the averages of the blocks $\mathbf{X}$ and $\hat{\mathbf{X}}$ respectively. $\sigma_\mathbf{X} \in \mathbb{R}_+^*$ and $\sigma_{\hat{\mathbf{X}}} \in \mathbb{R}_+^*$ denote the variances of each block respectively. $\sigma_{\mathbf{X}\hat{\mathbf{X}}} \in \mathbb{R}_+^*$ is the covariance. $c_1 = 0.01\left(2^b - 1\right)$. $c_2 = 0.03\left(2^b - 1\right)$. The typical block size is 8×8 pixels. All blockwise SSIMs are averaged over the entire image to obtain the imagewise SSIM.

2.8.3 Bjontegaard's metric

It is difficult to compare two compression standards using only the distortion because it implies to have two images encoded and decoded with the exact same compression rate.

To overcome this problem, Bjontegaard suggests to compute an average difference between two rate-distortion curves [55]. The principle is to encode and decode an image with both compression standards at different compression rates and compute the resulting PSNR in each case. The rate-distortion points are then put onto a graph (see Figures 2.18, 2.19, and 2.20). An approximate rate-distortion for each compression standard is obtained via a third order polynomial interpolation of its rate-distortion points. The gain of one compression standard over the other one corresponds to the area between the two approximate rate-distortion curves. This area can be computed in two ways, either at equivalent rate or at equivalent distortion.

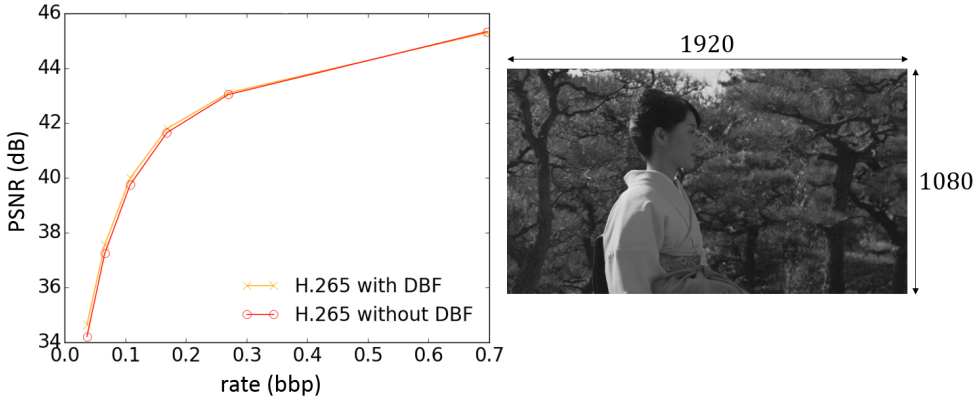


Figure 2.18: Comparison of the rate-distortion curve of H.265 with DeBlocking Filter (DBF) and that of H.265 without DBF for the luminance channel of the first frame of the video Kimono. For each of the two rate-distortion curves, each of the 6 rate-distortion points is associated to a different QP in $\{17, 22, 27, 32, 37, 42\}$. Note that the rate-distortion points are linked to each other; the third order polynomial interpolation is not shown. Here, Bjontegaard's metric of H.265 with DBF with respect to H.265 without DBF is 4.63% of rate savings at equivalent distortion.

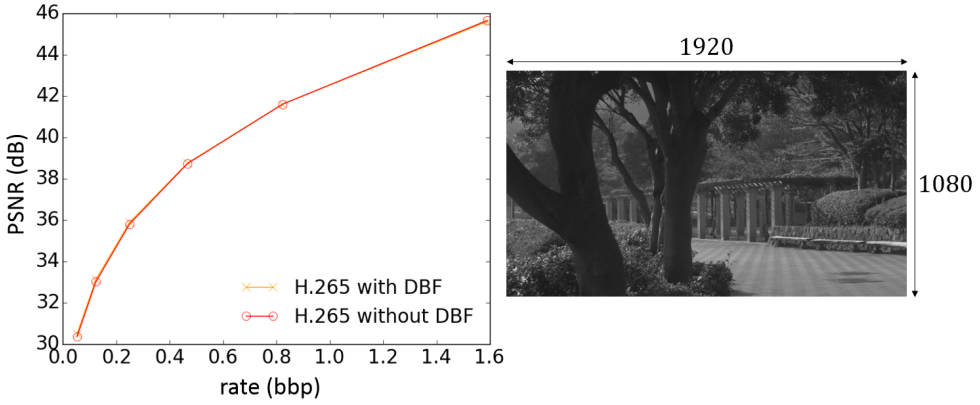


Figure 2.19: Comparison of the rate-distortion curve of H.265 with DBF and that of H.265 without DBF for the luminance channel of the first frame of the video ParkScene. Here, Bjontegaard's metric of H.265 with DBF with respect to H.265 without DBF is 1.25%.

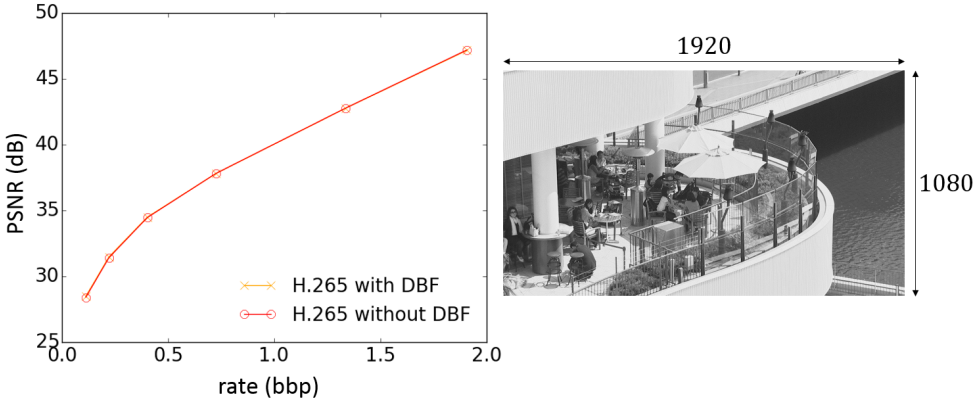


Figure 2.20: Comparison of the rate-distortion curve of H.265 with DBF and that of H.265 without DBF for the luminance channel of the first frame of the video ParkScene. Here, Bjontegaard's metric of H.265 with DBF with respect to H.265 without DBF is 0.37%.

Chapter 3

Learning models for image compression

This section introduces the neural network model, discusses two major open issues about neural networks that have influenced our research, and review the use of neural networks for learning image compression. Then, two families of methods for learning a transform yielding sparse representation, which are algorithms for sparse decomposition over a dictionary and shallow sparse autoencoders, are introduced and compared in the context of image compression. More precisely, an experimental framework is set up to analyze what makes a method for learning a transform yielding sparse representation efficient in terms of rate-distortion. The conclusions arising from this study will form the basis for Chapter 4.

3.1 Neural networks for image compression

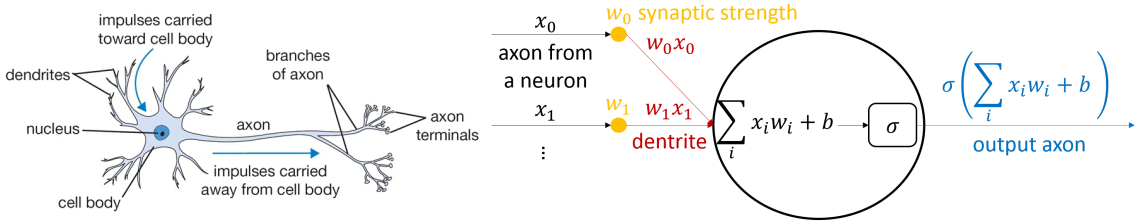
3.1.1 Neural network model

The motivation behind the neural network model, the layerwise organisation in a neural network, and the neural network training are explained. It is important to stress that there exists many families of neural networks, e.g. restricted boltzmann machines [56] or recurrent neural networks [57]. In this document, we only deal with the family of feedforward neural networks trained via backpropagation.

3.1.1.1 Modeling a neuron

The human brain is very efficient for solving tasks such as the object recognition in images perceived by the human eye [58] or the object segmentation in the images. This has motivated the research community to understand the human brain by modeling it since 1943 [59, 60]. The basic computational unit of the brain is a neuron. Approximately 86 billion neurons are inside the human nervous system. The neurons are connected with approximately 10^{14} synapses.

A neuron receives input signals from its dendrites and produces an output signal along its axon. Its axon is connected to dendrites of other neurons (see Figure 3.1a). In the standard mathematical model of a neuron, the signal x_0 that travels along the axon of a neuron interacts multiplicatively with the dendrite of the current neuron based on the synaptic strength of the dendrite w_0. The dendrites carry different signals to the body



(a) (b)

Figure 3.1: Comparison of (a) the sketch of a neuron from the point of view of biologists and (b) the standard mathematical model of a neuron.

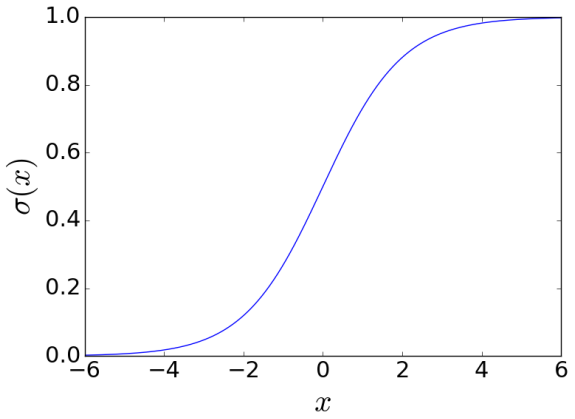


Figure 3.2: Logistic non-linearity $\sigma(x) = 1/(1+\exp(-x))$

of the current neuron where they get summed and a bias b is added. If the result of the previous linear combination exceeds a certain threshold, the neuron fires, sending a spike along its axon. A common assumption is that the scheduling of the spikes along the axon does not matter. The information emitted by the current neuron is coded in the frequency of the spikes. That is why the previous linear combination is converted into the frequency of the spikes via an activation function σ (see Figure 3.1b). The basic activation function is the logistic non-linearity $\sigma(x) = 1/(1+\exp(-x))$, $x \in \mathbb{R}$ as the logistic non-linearity takes a real-valued input and squashes it to $]0,1[$ (see Figure 3.2).

3.1.1.2 Neural networks as neurons in a graph

Neural networks are viewed as groups of neurons that are connected in an acyclic graph. This way, the outputs of some neurons become inputs to other neurons. Instead of using unstructured groups of connected neurons, neurons are organized into layers.

Fully-connected layer The most common type of layer is the fully-connected layer in which each output neuron is connected to all the input neurons. However, the output neurons are not connected to one another. A fully-connected layer with m input neurons and n output neurons (see Figure 3.3) is expressed as

$$\mathbf{z} = \sigma(\mathbf{V}\mathbf{x} + \mathbf{b}). \tag{3.1}$$

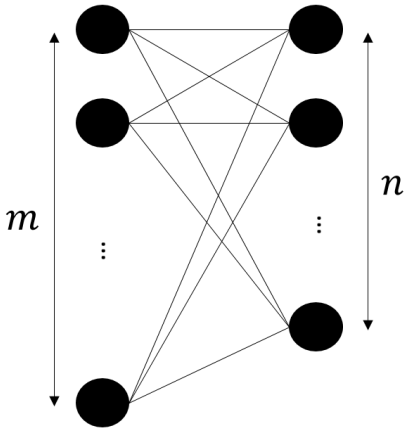


Figure 3.3: Fully-connected layer with m inputs neurons and n output neurons.

$\mathbf{x} \in \mathbb{R}^m$ and $\mathbf{z} \in \mathbb{R}^n$ store respectively the m input neurons and the n output neurons. $\mathbf{V} \in \mathbb{R}^{n \times m}$ and $\mathbf{b} \in \mathbb{R}^n$ denote respectively the weights connecting the output neurons to the input neurons and the biases.

Fully-connected feedforward neural network A fully-connected feedforward neural network is a stack of fully-connected layers. For instance, a fully-connected feedforward neural network composed of two fully-connected layers (see Figure 3.4) is expressed as

$$\hat{\mathbf{y}} = \sigma\left(\mathbf{\Phi}\sigma\left(\mathbf{V}\mathbf{x} + \mathbf{b}\right) + \mathbf{c}\right). \tag{3.2}$$

$\hat{\mathbf{y}} \in \mathbb{R}^l$ stores the l output neurons of the fully-connected feedforward neural network. $\mathbf{\Phi} \in \mathbb{R}^{l \times n}$ and $\mathbf{c} \in \mathbb{R}^l$ denote respectively the weights and the biases of the second fully-connected layer.

Convolutional layer Unlike fully-connected layers, convolutional layers forces local connectivity between neurons. This means that, instead of connecting each output neuron to all the input neurons, each output neuron is connected to neighboring input neurons. Moreover, the connection weights are shared among the output neurons.

In details, in the case where the input to the convolutional layer is an image with m channels, this image is first convolved with n 3D filters, the third dimension of each 3D filter being equal to m. Then, biases are added to the result of the convolution and a non-linearity is applied. The output of the convolutional layer is a 3D tensor whose third dimension is equal to n. This tensor is said to be a stack of n 2D feature maps. The convolutional layer (see Figure 3.5) is expressed as

$$\mathbf{Z} = \sigma\left(\mathbf{X} * \mathbf{W} +_t \mathbf{b}\right). \tag{3.3}$$

$\mathbf{X}$ and $\mathbf{Z}$ denote respectively the input image and the output stack of n 2D feature maps. $\mathbf{W} \in \mathbb{R}^{h \times w \times m \times n}$ and $\mathbf{b} \in \mathbb{R}^n$ denote respectively the bank of n convolutional filters and the biases. The operator $+_t$ in (3.3) is not the common elementwise addition between two tensors. $+_t$ means that the i^{th} bias is added to all the coefficients in the output resulting from the convolution of the input image with the i^{th} convolutional filter, $i \in [|1, n|]$. In the literature $+_t$ is simply denoted $+$.

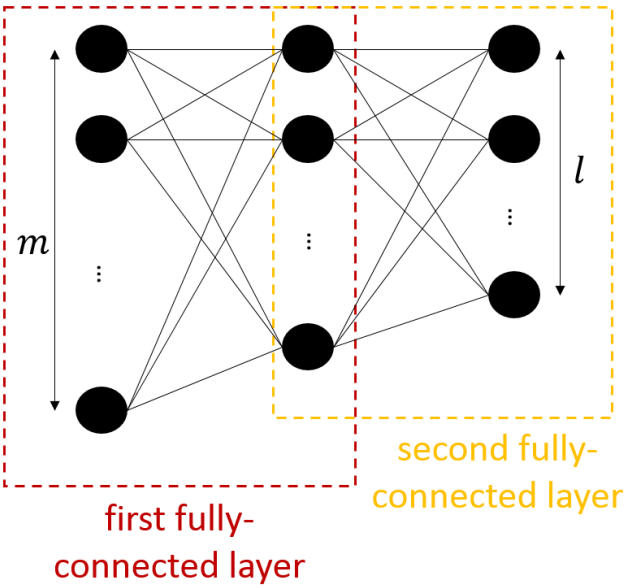


Figure 3.4: Fully-connected feedforward neural network made of two fully-connected layers. It has m inputs neurons and l output neurons.

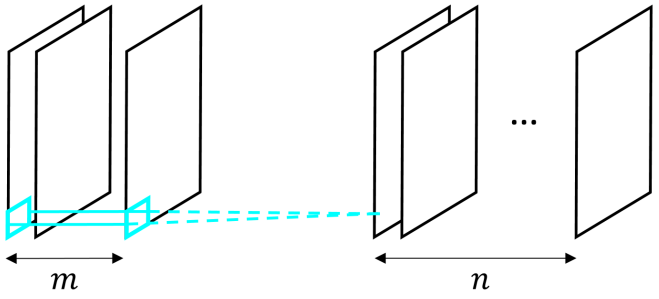


Figure 3.5: Convolutional layer when its input is an image with m channels and n 3D convolutional filters are involved.

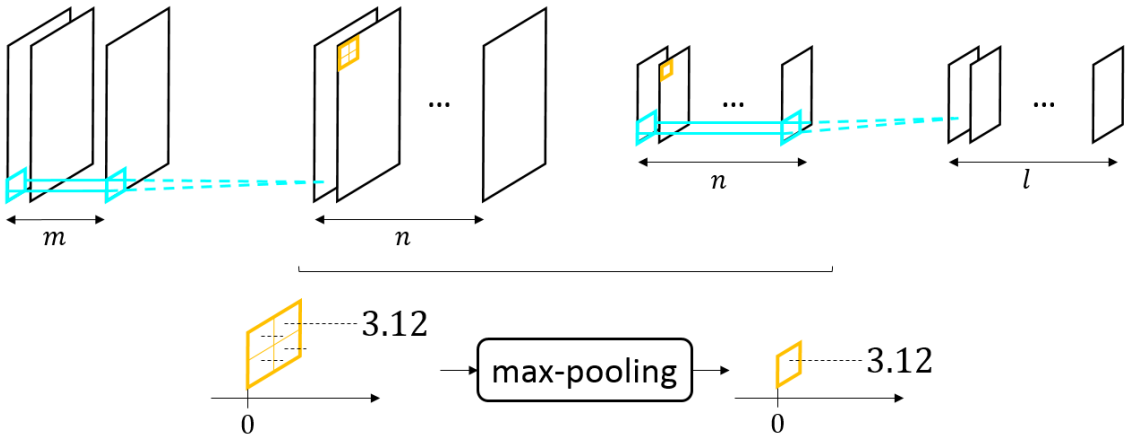


Figure 3.6: Convolutional feedforward neural network containing a convolutional layer, a max-pooling operation, and another convolutional layer. The input image has m channels and the output stack contains l 2D feature maps. In this illustration, the non-overlapping sub-regions in max-pooling are 2×2.

Convolutional feedforward neural network In its standard form, a convolutional feedforward neural network is a feedforward neural network containing an alternance between convolutional layers and max-pooling operations. In the case where the input to max-pooling is a stack of 2D feature maps, max-pooling downsamples each input 2D feature map by appling a max filter to non-overlapping sub-regions. For a given sub-region, the max filter picks the maximum value over this sub-region. The dimension of the non-overlapping sub-regions is often 2×2 as it is observed that, in many applications, a convolutional feedforward neural network yields better performances when the non-overlapping sub-regions in each max-pooling operation are very small and max-pooling is alternated many times with convolutional layers [61, 62]. For instance, a convolutional feedforward neural network composed of the convolutional layer in (3.3), a max-pooling operation, and another convolutional layer (see Figure 3.6) is expressed as

$$\hat{\mathbf{Y}} = \sigma\left(\text{max-pooling}\left(\sigma\left(\mathbf{X} * \mathbf{W} +_t \mathbf{b}\right)\right) * \mathbf{U} +_t \mathbf{c}\right). \tag{3.4}$$

$\hat{\mathbf{Y}}$ is the stack of $l \in \mathbb{N}$ 2D feature maps returned by the convolutional feedforward neural network. $\mathbf{U} \in \mathbb{R}^{\overline{h} \times \overline{w} \times n \times l}$ and $\mathbf{c} \in \mathbb{R}^l$ denote respectively the l convolutional kernels and the biases in the second convolutional layer.

The main difference between a convolutional feedforward neural network and a fully-connected feedforward neural network is that prior knowledge about the input data is incorporated into the architecture of the convolutional feedforward neural network. First, filters with local connectivity are applied because it is assumed that the data components exhibit high local correlations. Then, a multi-layer architecture with max-pooling is used as the data is assumed to have a multi-resolution structure. The high local correlations and the multi-resolution structure are two key properties of images [63]. This prior knowledge reduces the number of weights and biases to be learned, thus enhancing the trainability of the neural network.

3.1.1.3 Training a feedforward neural network

Let us consider a regression task to explain the training of a feedforward neural network $f\left(\,.\,;\boldsymbol{\theta}\right)$ whose weights and biases are inside the vector $\boldsymbol{\theta}$. For instance, a regression task will be considered in Chapter 5 with the learning of intra prediction for image compression. A training set containing $N \in \mathbb{N}^*$ inputs vectors, each paired with its target vector, $\left\{\left(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}\right), ..., \left(\mathbf{x}^{(N)}, \mathbf{y}^{(N)}\right)\right\}$ is given. The goal of the training is to minimize over the trainable parameters $\boldsymbol{\theta}$ an objective function $\mathcal{L}\left(\left(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}\right), ..., \left(\mathbf{x}^{(N)}, \mathbf{y}^{(N)}\right); \boldsymbol{\theta}\right)$ measuring the error between the training target vector and its approximation provided by the feedforward neural network when feeding the associated input vector into it, averaged over the training set,

$$\min_{\boldsymbol{\theta}} \mathcal{L}\left(\left(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}\right), ..., \left(\mathbf{x}^{(N)}, \mathbf{y}^{(N)}\right); \boldsymbol{\theta}\right). \tag{3.5}$$

For simplicity, the training does not take into consideration the problem of the generalization to new input vectors during the test phase. As an example, if the training target vectors have binary values, the objective function is usually the cross-entropy between the training target vector and its approximation provided by the feedforward neural network

when feeding the associated input vector into it, averaged over the training set,

$$\begin{aligned}\mathcal{L}\left(\left(\mathbf{x}^{(1)},\mathbf{y}^{(1)}\right),...,\left(\mathbf{x}^{(N)},\mathbf{y}^{(N)}\right);\boldsymbol{\theta}\right) = \\ -\frac{1}{N}\sum_{i=1}^{N}\sum_{j}\left(y_j^{(i)}\log\left(\hat{y}_j^{(i)}\right)+\left(1-y_j^{(i)}\right)\log\left(1-\hat{y}_j^{(i)}\right)\right) \qquad (3.6)\\ \hat{\mathbf{y}}^{(i)} = f\left(\mathbf{x}^{(i)};\boldsymbol{\theta}\right).\end{aligned}$$

In the case of training target vectors with binary values, the last layer of the feedforward neural network usually has the logistic non-linearity as activation function so that the coefficients of $\hat{\mathbf{y}}$ belong to $]0,1[$.

For the minimization, a gradient descent is used. The step $t \in \mathbb{N}^*$ of the gradient descent is expressed as

$$\boldsymbol{\theta}^{(t)} \leftarrow \boldsymbol{\theta}^{(t-1)} - \varepsilon^{(t)}\frac{\partial\mathcal{L}\left(\left(\mathbf{x}^{(1)},\mathbf{y}^{(1)}\right),...,\left(\mathbf{x}^{(N)},\mathbf{y}^{(N)}\right);\boldsymbol{\theta}^{(t-1)}\right)}{\partial\boldsymbol{\theta}^{(t-1)}}. \qquad (3.7)$$

$\varepsilon^{(t)} \in \mathbb{R}_+^*$ is the learning rate for the gradient descent at step t. For the trainable parameters at initialization $\boldsymbol{\theta}^{(0)}$, the weights are drawn from specific distributions [64, 65] whereas the biases are set to a constant [66].

Backpropagation is an algorithm for computing the derivative of the objective function with respect to the trainable parameters $\boldsymbol{\theta}$ at low computational cost. Backpropagation is based on the decomposition of this derivative via iterative chain rules [67, 68].

To speed up the gradient descent, a coarse approximation of the derivative in (3.7) can be computed using a single training input vector randomly drawn from the training set, paired with its target vector. This is called online stochastic gradient descent [69, 70]. Alternatively, a more precise approximation of the derivative in (3.7) can be calculated using a mini-batch of training input vectors randomly drawn from the training set, each paired with its target vector. In this case, the name is mini-batch stochastic gradient descent [71].

3.1.1.4 Autoencoders

An autoencoder is a neural network that is trained to first compute a representation of the input and then, from this representation, reproduce the input at the output. Its architecture can be viewed as consisting of an encoder and a decoder. The encoder $g_e(\,.\,;\boldsymbol{\theta})$, parametrized by $\boldsymbol{\theta}$, takes an input vector and computes a representation of the input vector. The decoder $g_d(\,.\,;\boldsymbol{\phi})$, parametrized by $\boldsymbol{\phi}$, takes the representation and returns a reconstruction of the input vector.

$$\mathbf{z} = g_e\left(\mathbf{x};\boldsymbol{\theta}\right) \qquad (3.8)$$

$$\hat{\mathbf{x}} = g_d\left(\mathbf{z};\boldsymbol{\phi}\right) \qquad (3.9)$$

$\mathbf{x} \in \mathbb{R}^m$, $\mathbf{z} \in \mathbb{R}^n$, and $\hat{\mathbf{x}} \in \mathbb{R}^m$ denote respectively the input vector, its representation, and its reconstruction.

An autoencoder is trained by minimizing over its trainable parameters $\{\boldsymbol{\theta};\boldsymbol{\phi}\}$ the reconstruction error, averaged over a training set of N input vectors $\left\{\mathbf{x}^{(1)},...,\mathbf{x}^{(N)}\right\}$. For

instance, if the measure of error is the l_2-norm of the difference between the input vector and its reconstruction, the minimization over the trainable parameters $\{\boldsymbol{\theta}; \boldsymbol{\phi}\}$ is

$$\min_{\boldsymbol{\theta},\boldsymbol{\phi}} \frac{1}{N} \sum_{i=1}^{N} \| \mathbf{x}^{(i)} - g_d \left(g_e \left(\mathbf{x}^{(i)}; \boldsymbol{\theta} \right); \boldsymbol{\phi} \right) \|_2^2. \tag{3.10}$$

If the autoencoder learns the identity function, it is not useful. It is more interesting to obtain a representation that captures the most salient features of the training data. To this end, a solution is to set $n < m$, i.e. to have an undercomplete representation [72].

Another solution is to set $n \geq m$, i.e. to have an overcomplete representation, and constrain the representation space to be contractive in the neighborhood of the training input vectors. In other words, the representation is stable when the training input vector slightly changes. This constraint is introduced by adding to the objective function to be minimized over the trainable parameters $\{\boldsymbol{\theta}; \boldsymbol{\phi}\}$ the Frobenius norm of the Jacobian of the encoder mapping, averaged over the training set of input vectors. The minimization over the trainable parameters $\{\boldsymbol{\theta}; \boldsymbol{\phi}\}$ then becomes

$$\begin{aligned} &\min_{\boldsymbol{\theta},\boldsymbol{\phi}} \frac{1}{N} \sum_{i=1}^{N} \left(\| \mathbf{x}^{(i)} - g_d \left(g_e \left(\mathbf{x}^{(i)}; \boldsymbol{\theta} \right); \boldsymbol{\phi} \right) \|_2^2 + \gamma \| J_{g_e} \left(\mathbf{x}^{(i)} \right) \|_F^2 \right), \gamma \in \mathbb{R}_+^* \\ &\| J_{g_e} \left(\mathbf{x}^{(i)} \right) \|_F^2 = \sum_{j=1}^{m} \sum_{k=1}^{n} \left(\frac{\partial g_e \left(\mathbf{x}^{(i)}; \boldsymbol{\theta} \right)_k}{\partial x_j^{(i)}} \right)^2. \end{aligned} \tag{3.11}$$

This autoencoder is called *contractive autoencoder* [73].

3.1.2 Open issues regarding neural networks

Two major open issues regarding neural networks, i.e. the generalization of neural networks and the choice of the neural network architecture, are explained. Then, this section mentions the impact of the two issues on our research process.

3.1.2.1 Generalization of neural networks

The problem of generalization comes from the fact that a Machine Learning algorithm is trained on training data but the measure of performance is computed on test data.

The common wisdom justifies a small generalization error by either properties of the model family, regularization techniques used during the training phase, or the number of training examples. But, these reasons are no longer valid when it comes to the ability of large neural networks to generalize well.

This is shown experimentally by the authors in [74] in the context of image classification. Two different image classification datasets, the CIFAR10 dataset and the ILSVRC 2012 ImageNet dataset, are considered. Three deep feedforward neural network architectures from the literature are set up. For each pair of a dataset and a feedforward neural network architecture, the authors in [74] study the case where all the labels in the training set are replaced by random ones, which is called the corrupted training set, and the case where the training set is uncorrupted. For the uncorrupted training set, the deep feedforward neural network reaches almost perfect accuracy on the training set at the end of the

training phase. The accuracy on the test set is high. For the corrupted training set, the accuracy on the training set is still almost perfect whereas the accuracy of the test set is at the level of the random guess. This shows that a deep feedforward neural network has the capacity to memorize a large training set. The problem is that, for the uncorrupted training set, the deep feedforward neural network converges to a solution generalizing well. Why did not it memorize the training set?

Another evidence shakes up the conventional view of regularization. l_2-norm weight decay is the addition of the l_2-norm of all the weights of the neural networks to the objective function to be minimized over the neural network parameters. This is equivalent to setting a Gaussian prior on these weights [75]. It is observed that, in the context of image classification, the l_2-norm weight decay increases the classifcation accuracy on the training set [74, 66]. This is not consistent with the fact that a regularizer is usually expected to decrease the training accuracy.

The question of the generalization of large neural networks is still an open issue. In this document, we resort to experimentation to select the number of examples in the training set and the regularizers during the training phase.

3.1.2.2 Choice of the neural network architecture

Recent works have established links between the architecture of deep neural networks and their state-of-the-art results in classification and regression in high dimension. For instance, let us consider the problem of supervised learning where a deep convolutional feedforward neural network approximates a function f_o using training data examples. The author in [76] shows that the cascade of convolutional layers including a contractive pointwise non-linearity progressively scatters the data at different scales and reduce the range of variations of the result along directions for which f_o does not vary. This yields a transformation of the data having the main invariant of f_o.

Despite this progress, the design of a neural network requires extensive empirical evaluations. This is due to the fact that the neural network architecture often depends on the application. The example of the Batch Normalization layer (BN) perfectly illustrates this point. BN takes a batch of input examples and normalizes each input coefficient over all the examples in the batch by centering it and reducing its scale [77]. In the context of image classification, a deep neural network containing several BNs is less prone to overfitting and its training is faster. However, in the context of image super-resolution, a deep neural network containing BNs returns images of worse quality than its counterparts without any BN [78].

Consequently, in Chapter 5, the proposed neural network architectures cannot be justified theoretically. They are rather motivated by the structure of the data to be processed.

3.1.3 Review of neural networks for image compression

The approaches for learning image compression via neural networks can be classified into two categories. The first category gathers the works aiming at learning a simple image compression scheme. The second category contains propositions for improving components of H.265 via learning.

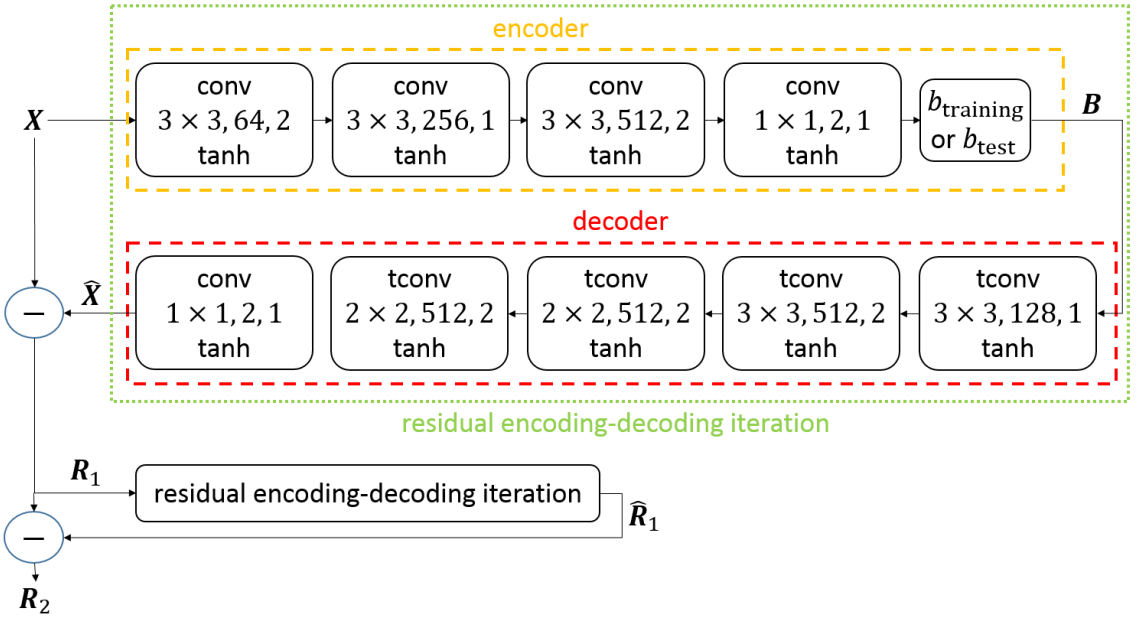


Figure 3.7: Convolutional residual autoencoder in [79], where the three numbers separated by commas are respectively the height and width of the convolutional kernels, the number of convolutional kernels, and the convolution stride. *tanh* denotes the hyperbolic tangent non-linearity. Two iterations of residual encoding and decoding are depicted. In the first iteration, the encoder takes a 32×32 image patch $\mathbf{X}$ and generates a stack of feature maps $\mathbf{B}$ whose coefficients belong to $\{-1, 1\}$. This stack contains two 8×8 feature maps. The decoder takes the stack of feature maps and computes a reconstruction $\hat{\mathbf{X}}$ of the image patch $\mathbf{X}$. The second iteration encodes and decodes the residue $\mathbf{R}_1 = \mathbf{X} - \hat{\mathbf{X}}$ from the first iteration. In each block, *conv* and *tconv* mean respectively convolution and transpose convolution [80]. Note that the hyperbolic tangent non-linearity is a rescaled version of the logistic non-linearity as $\tanh(x) = 2\sigma(2x) - 1, x \in \mathbb{R}$. At the end of the two-iteration residual encoding and decoding, the refined reconstruction of the image patch is $\hat{\mathbf{X}} + \hat{\mathbf{R}}_1$. Note that the pixels values in the input patches are rescaled to $[-0.9, 0.9]$, which is within the range of the hyperbolic tangent non-linearity.

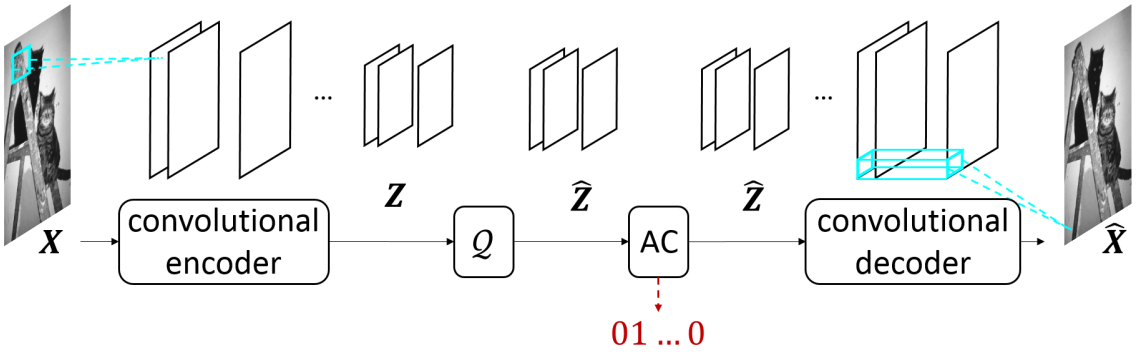


Figure 3.8: Convolutional autoencoder in [81, 82, 83]. The Arithmetic Coder (AC), gathering the arithmetic encoder and the arithmetic decoder, is lossless and fixed. It has no impact on the training of the convolutional autoencoder. That is why it is no longer drawn in the illustrations in Chapter 4.

3.1.3.1 Towards end-to-end learning of an image compression scheme

As alternatives to traditional compression standards, several works attempt to learn a simple image compression scheme in an end-to-end manner. To date, most of the learned image compression schemes include a transform, a quantizer, and a context model for entropy coding.

Joint learning of a transform and a quantizer The first work proposing an end-to-end learned image compression scheme based on neural networks is [79]. In this work, a convolutional residual autoencoder turns an image patch into a binary representation and then computes a reconstruction of this image patch from the representation. A residue between the image and its reconstruction is computed. Finally, the encoding and decoding iteration is applied iteratively to the residue from the previous iteration (see Figure 3.7). To get a binary representation, the hyperbolic tangent non-linearity is chosen for the last convolutional layer in the encoder and the output of this layer is fed into the binary threshold b_{test},

$$b_{\text{test}}(x) = \begin{cases} -1 \text{ if } x < 0 \\ 1 \text{ if } x \geq 0 \end{cases}, x \in]-1, 1[. \tag{3.12}$$

The binary threshold is problematic during the training phase via gradient descent as its derivative with respect to its input always equals to 0. Therefore, a stochastic approximation b_{training} of the binary threshold b_{test} is used during the training phase,

$$\begin{aligned} & b_{\text{training}}(x) = x + \epsilon, x \in]-1, 1[, b_{\text{training}}(x) \in \{-1, 1\} \\ & \epsilon = \begin{cases} 1 - x \text{ with probability } (1 + x)/2 \\ -1 - x \text{ with probability } (1 - x)/2 . \end{cases} \end{aligned} \tag{3.13}$$

Then, as $\mathbb{E}[b_{\text{training}}](x) = x$, b_{training} is replaced by $\mathbb{E}[b_{\text{training}}]$ when computing the derivative of the stochastic approximation of the binary threshold with respect to its input during the training phase. The convolutional residual autoencoder is trained via the minimization over its trainable parameters of the l_2-norm of the difference between the input and the reconstruction of the input, cumulated over all residual encoding and decoding iterations, and averaged over a training set of image patches. The residual convolutional autoencoder has the capability of variable rate image compression as, at each encoding and decoding iteration, bits are added to the bistream while the reconstruction of the image is refined. The problem regarding this approach is that it is applied to 32×32 image patches. The subsequent works [84, 85] extend it to full resolution image compression. Likewise, all the approaches presented below apply to full resolution images.

Unlike in [79], there is no residual mechanism in [81, 82, 83]. A convolutional autoencoder encodes and decodes an image via a single pass (see Figure 3.8). The binary threshold is replaced by a uniform scalar quantizer $\mathcal{Q}$ followed by an arithmetic coder for converting the representation $\mathbf{Z}$ into a bitstream. The uniform scalar quantization is problematic during the training phase via gradient descent as its derivative with respect to its input always equals to 0. Therefore, in [81], the uniform scalar quantizer is replaced by the identity during the training phase. In [82], the uniform scalar quantizer is replaced by the identity only when computing its derivative with respect to its input during the training phase. In contrast, the authors in [83] fix the quantization step size to

1 during the test phase and approximate the uniform scalar quantization with a uniform noise of support $[-0.5, 0.5]$ during the training phase. The main difference between [81] and [82, 83] lies in the objective function to be minimized over the trainable parameters of the convolutional autoencoder. In [81], the rate is controlled via a constraint on the number of non-zero coefficients in the representation. The objective function only contains a distortion term, which is the l_2-norm of the difference between the input image and its reconstruction, averaged over a training set of images. However, in [82, 83], the objective function combines this distortion term and a term approximating the entropy of the quantized representation, averaged over the training set.

Note that the approaches in [82, 83] learn one transform per target rate. During the test phase, each learned transform compresses at a single rate. The question of a single learned transform compressing at several rates during the test phase is not addressed. This question is a key topic in [81, 86] (see Chapter 4).

Differently from [81, 82, 83], the image compression scheme in [87] integrates a vector quantizer instead of a uniform scalar quantizer. Let say that the representation $\mathbf{Z}$ at the output of the convolutional encoder is reshaped into $\overline{\mathbf{Z}} = \left[\overline{\mathbf{z}}^{(1)}, ..., \overline{\mathbf{z}}^{(p)}\right]$ where, $\forall i \in [|1, p|], \overline{\mathbf{z}}^{(i)} \in \mathbb{R}^d$. $\mathcal{C} = \left\{\mathbf{c}^{(1)}, ..., \mathbf{c}^{(L)}\right\}$ denotes a set of $L \in \mathbb{N}^*$ centers vectors. For $j \in [|1, L|]$, $\mathbf{c}^{(j)} \in \mathbb{R}^d$. Each of the p vectors in the quantized representation is defined as a linear combination of the centers vectors,

$$Q\left(\overline{\mathbf{z}}^{(i)}\right) = \sum_{j=1}^{L} \mathbf{c}^{(j)} \phi_{\text{test},j}\left(\overline{\mathbf{z}}^{(i)}\right). \tag{3.14}$$

$\boldsymbol{\phi}_{\text{test}}\left(\overline{\mathbf{z}}^{(i)}\right) \in \{0,1\}^L$ contains the coefficients of the linear combination for the vector $Q\left(\overline{\mathbf{z}}^{(i)}\right)$ in the quantized representation. During the test phase, the coefficients are determined by the nearest neighbor assignment,

$$\phi_{\text{test},j}\left(\overline{\mathbf{z}}^{(i)}\right) = \begin{cases} 1 \text{ if } j = \underset{k \in [|1,L|]}{\operatorname{argmin}} \|\overline{\mathbf{z}}^{(i)} - \mathbf{c}^{(k)}\|_2 \\ 0 \text{ otherwise} \end{cases}. \tag{3.15}$$

But, the nearest neighbor assignment makes $\phi_{\text{test},j}\left(\overline{\mathbf{z}}^{(i)}\right)$ non-differentiable with respect to either $\overline{\mathbf{z}}^{(i)}$ or $\mathbf{c}^{(k)}, k \in [|1, L|]$. That is why the authors in [87] use a relaxed version of the nearest neighbor assignment during the training phase,

$$\phi_{\text{training},j}\left(\overline{\mathbf{z}}^{(i)}; \lambda\right) = \frac{\exp\left(-\lambda \|\overline{\mathbf{z}}^{(i)} - \mathbf{c}^{(j)}\|_2^2\right)}{\sum_{k=1}^{L} \exp\left(-\lambda \|\overline{\mathbf{z}}^{(i)} - \mathbf{c}^{(k)}\|_2^2\right)}, \lambda \in \mathbb{R}_+^*. \tag{3.16}$$

This way, the trainable parameters of the convolutional autoencoder and the set of centers vectors can be learned jointly. Note that $\phi_{\text{training},j}\left(\overline{\mathbf{z}}^{(i)}; \lambda\right)$ tends to $\phi_{\text{test},j}\left(\overline{\mathbf{z}}^{(i)}\right)$ as $\lambda \to +\infty$. That is why, during the training phase, the relaxed version of the nearest neighbor assigment can get close to the nearest neighbor assignment by increasing progressively λ.

Joint learning of a transform, a quantizer, and a context model for the entropy coder
In many applications, neural networks show efficiency when they learn the entire processing pipeline. This point is well illustrated by the case of the generation of image

descriptions, where the transformation of an image into an embedded representation via a feedforward neural network and the transformation of text descriptions into another embedded representation via a recurrent neural network are jointly learned [88].

Given this trend, it seems natural in image compression to try to learn jointly as many components in an image compression scheme as possible. That is why, the next step is to learn jointly a transform, a quantizer, and a context model for entropy coding. The authors in [89] extends [83] by pursuing this aim. A branch convolutional encoder is placed before the uniform scalar quantizer to map the representation $\mathbf{Z}$ to a stack of feature maps $\mathbf{C}$, which is a representation of the scale of spatially neighboring coefficients in the quantized representation $\hat{\mathbf{Z}}$. $\mathbf{C}$ is quantized, yielding $\hat{\mathbf{C}}$. Then, a branch convolutional decoder maps $\hat{\mathbf{C}}$ to an approximation $\hat{\mathbf{\Sigma}}$ of the scale of each coefficient in the quantized representation $\hat{\mathbf{Z}}$ (see Figure 3.9). This mechanism acts as a context model for the quantized representation $\hat{\mathbf{Z}}$. The entire architecture in Figure 3.9 is trained by minimizing over its trainable parameters an objective function combining the distortion term, a term approximating the entropy of the quantized representation $\hat{\mathbf{Z}}$, and another term approximating the entropy of $\hat{\mathbf{C}}$. The last two terms are averages over the training set, like the distortion term. It is essential to stress that, in [89], the compressed context model is sent from the encoder side to the decoder side. However, to save rate, the two approaches presented below do not send the context model. It is inferred from either already decoded bits [90] or from the quantized representation [91].

The authors in [90] propose another way of learning a context model. A bitplane decomposition is first applied to the quantized representation. Then, in the bitplane decomposition, for a given bit to be encoded via the arithmetic encoder, features are extracted from the bits that have already been decoded by the arithmetic decoder in the neighborhood of the current bit. A classifier infers the probability of the current bit being 1 from these features. The probability is fed into the arithmetic encoder (see Figure 3.10).

Finally, the authors in [91] extend their previous contributions in [87] also by targeting the learning of a context model for entropy coding. A convolutional feedforward neural network provides an approximation of the probability mass function of each coefficient in the quantized representation given the previous coefficients in the raster-scan order of the arithmetic encoding and decoding. To setup this method, the number of quantization levels first has to be fixed. That is why the vector quantizer in [87] is turned into a scalar variant with scalar centers $\mathcal{C} = \{c_1, ..., c_L\}$ where, for $l \in [|1, L|]$, $c_l \in \mathbb{R}$. During the test phase, each coefficient in the representation $\mathbf{Z}$ is quantized via the scalar nearest neighbor assignment,

$$\hat{z}_{ijk} = \min_{c_1,...,c_L} |z_{ijk} - c_l|. \tag{3.17}$$

z_{ijk} and $\hat{z}_{ijk}$ are the coefficients at the position $[i, j, k]$ in $\mathbf{Z}$ and $\hat{\mathbf{Z}}$ respectively. (3.17) gives L quantization levels. During the training phase, to resolve the non-differentiability of the scalar nearest neighbor assignment with respect to its input, the scalar nearest neighbor assignment is replaced by its relaxed version,

$$\tilde{z}_{ijk} = \sum_{j=1}^{L} \frac{\exp\left(-\lambda|z_{ijk} - c_j|\right)}{\sum_{k=1}^{L} \exp\left(-\lambda|z_{ijk} - c_k|\right)} c_j, \lambda \in \mathbb{R}_+^*. \tag{3.18}$$

A convolutional feedforward neural network maps the quantized representation $\hat{\mathbf{Z}}$ to a 4D

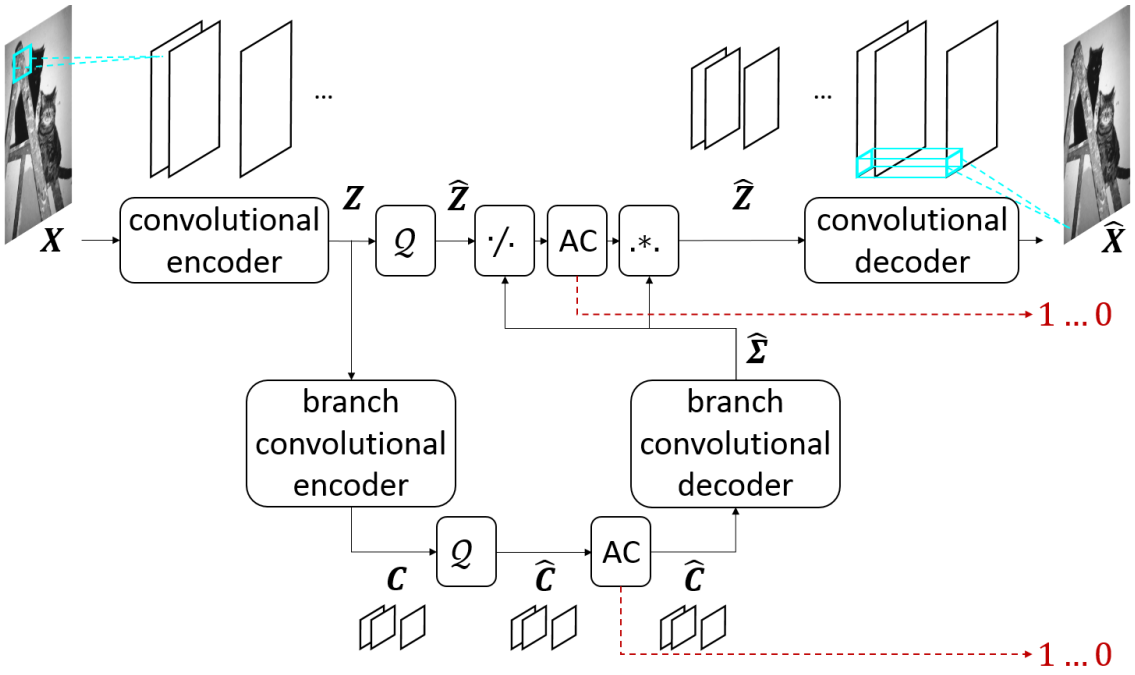


Figure 3.9: Convolutional autoencoder supplemented by a branch convolutional autoencoder that compresses a context model for entropy coding. For a given input image to be compressed, the bitstream is obtained by grouping the two bitstreams provided by the two Arithmetic Coders (ACs). $./.$ denotes the elementwise division between two tensors. $.*.$ denotes the elementwise multiplication between two tensors. The approximation $\hat{\mathbf{\Sigma}}$ of the scale of each coefficient in the quantized representation $\hat{\mathbf{Z}}$ has the same shape as the quantized representation $\hat{\mathbf{Z}}$.

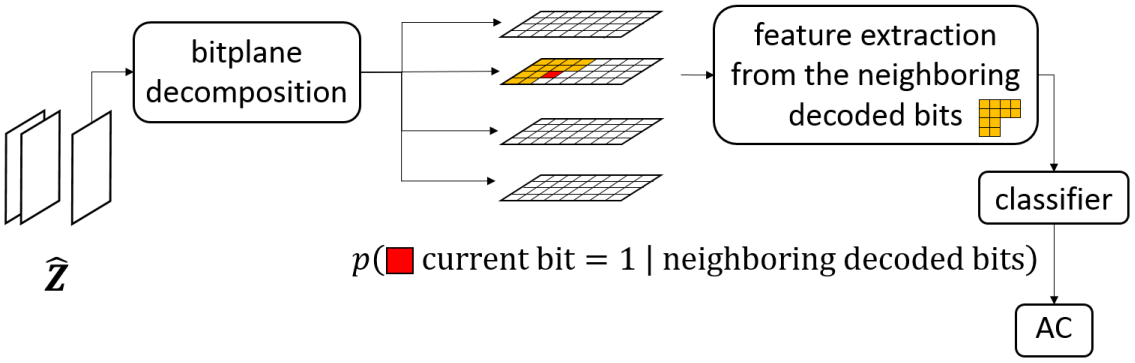


Figure 3.10: Context model at the bitplane decomposition level. The current bit to be encoded via the Arithmetic Coder (AC) is colored in red. Its neighboring decoded bits in the same bitplane are colored in orange.

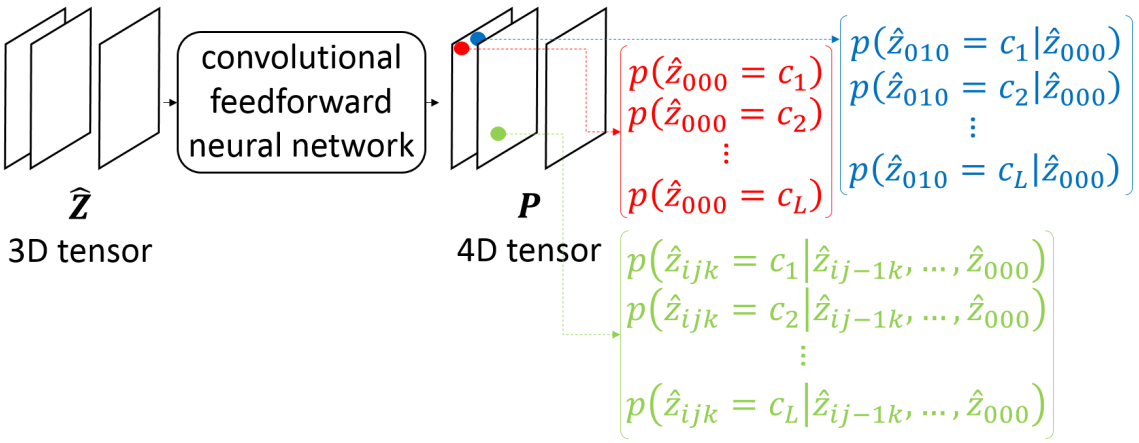


Figure 3.11: Context model $\mathbf{P}$ for entropy coding directly inferred from the quantized representation via a convolutional feedforward neural network.

tensor $\mathbf{P}$ whose coefficient at the position $[i, j, k, l]$ gives the probability that $\hat{z}_{ijk}$ equals to c_l, $l \in [|1, L|]$, given the value of the coefficients that have already been decoded by the arithmetic decoder in the quantized representation (see Figure 3.11). $\mathbf{P}$ is thus the context model for arithmetic coding. Let $I(\hat{z}_{ijk})$ be the index of $\hat{z}_{ijk}$ in $\mathcal{C}$. The convolutional feedforward neural network is trained by minimizing over its trainable parameters the maximum likelihood, averaged over a training set of images. For a single training image, providing a single quantized representation, the maximum likelihood is

$$\mathcal{L}\left(\hat{\mathbf{Z}}\right) = -\sum_{i,j,k} \log\left(P_{ijk\, I\left(\hat{z}_{ijk}\right)}\right). \tag{3.19}$$

In Figure 3.11, masks are applied to the convolutions in the convolutional feedforward neural network so that $\hat{z}_{ijk}$ can only depend on the coefficients in the quantized representation $\hat{\mathbf{Z}}$ that have already been decoded by the arithmetic decoder.

End-to-end scalable image compression scheme The originality of the approach in [92] is the learning of an end-to-end scalable image compression scheme. Indeed, a recurrent variational auto-encoder creates a representation of an image in which the information is ordered. Taking the first few bits in the bitstream, the variational decoder provides the semantic content of the decoded image. With the remaining bits, the variational decoder refines the low level details in the decoded image. The residual mechanism of the convolutional residual autoencoder in [79, 84] also enables scalable image compression.

Rate-distortion performance The take-home message regarding the rate-distortion performance of the end-to-end learning of an image compression scheme is that the rate-distortion performance improves very fast. Indeed, [84], which appeared for the first time in late 2016 in *arXiv*, shows rate-distortion curves close to that of JPEG whereas the work in [93], which is an improvement over [89] and was published in September 2018 in *arXiv*, gives rate-distortion curves slightly above that of H.265.

Table 3.1: Summary of the characteristics of the different approaches for the end-to-end learning of image compression. For the neural network type, ReAE, RAE, and FAE mean respectively Residual Auto-Encoder, Recurrent Auto-Encoder, and Feedforward Auto-Encoder respectively. For the quantization type, B, S, and V denote binary, scalar, and vector respectively. Note that, in [79, 84, 85], both ReAEs and RAEs are proposed. ✓ = YES and × = NO.

Reference	structure				capability	
	network type	quantization type	learned approximate quantization	learned context model	scalable	variable rate
[79]	ReAE, RAE	B	✓	×	✓	✓
[84, 85]	ReAE, RAE	B	✓	✓	✓	✓
[81]	FAE	S	×	×	×	✓
[82]	FAE	S	×	×	×	×
[83]	FAE	S	✓	×	×	×
[86]	FAE	S	✓	×	×	✓
[87]	FAE	V	✓	×	×	×
[89, 93]	FAE	S	✓	✓	×	×
[90]	FAE	S	×	✓	×	×
[91]	FAE	S	✓	✓	×	×
[92]	RAE	S	×	×	✓	✓

3.1.3.2 Improving components of H.265 via learning

The quadtree partitioning in H.265 takes approximatively 80% of the encoding time because of the recursive rate-distortion optimization. One of the first application of learning in H.265 is the reduction of the complexity of the quadtree partitioning. To avoid running the recursive rate-distortion optimization, the quadtree partitioning is predicted in advance. Let us begin with two approaches that are not based on neural networks to highlight the difference between them and a neural network based approach. In [94], given a CB to be possibly split in the current CTB, features are first computed from the quadtree partitioning history. For instance, a feature is the average number of splits in the already encoded and decoded CTBs around the current CTB. Another feature is the selected intra or inter prediction mode for predicting this CB. Then, a decision tree infers from these features whether the iterative splitting of this CB should cease. In [95], a support vector machine replaces the decision tree. Unlike the two previous approaches in which hand-crafted features are fed into classifiers, in [96], a convolutional feedforward neural network maps the pixels in a given CB to be possibly split in the current CTB and the pixels in its neighboring CBs inside the same CTB to the decision of stopping the iterative splitting of this CB.

A convolutional feedforward neural network can also give a context model for encoding the index of the selected intra prediction mode in H.265 for predicting a given luminance PB. For instance, the convolutional feedforward neural network in [97] takes 3 already encoded and decoded luminance PBs around the given luminance PB and the 3 most probable modes to infer the probability of each of the 35 intra prediction modes of being the selected mode.

Another problem learning approaches tackle is the reduction of the blocking artifacts in the images H.265 decodes. As H.265 uses block-based prediction and transform coding, discontinuites appear around the block boundaries in the decoded images. To limit this, H.265 applies successively the DBF and the SAO (see Section 2.6) to the decoded images. The authors in [98] replace these two filters by a convolutional feedforward neural network. In [99], the approach differs slightly. The DBF and the SAO are kept and a convolutional feedforward neural network is added as a complementary filter. Similarly, the authors in [100] train a neural network whose architecture is very similar to the neural network called EDSR [78] to remove the compression artifacts on the images decoded by H.265. In [101], the authors show that an approach for compression artifact removal based on generative adversarial networks provides reconstructed images with higher fidelity to the uncompressed versions of images than the approaches based on standard convolutional neural networks, according to the human eyes.

Regarding the intra prediction in H.265, a feedforward neural network is used to predict a given PB from already encoded and decoded pixels in the neighborhood of this PB [102, 103]. In H.265, the size of a PB can be either 4×4, 8×8, 16×16, 32×32, or 64×64 pixels (see Chapter 2). The PBs of each size are predicted via a different feedforward neural network. In [102], only fully-connected feedforward neural networks are used. Besides, PBs of size 4×4, 8×8, 16×16, and 32×32 only are considered. The PBs of size 64×64 pixels are not considered as a fully-connected feedforward neural network for predicting these PBs would contain too many trainable parameters. In contrast, in [103], convolutional feedforward neural networks predict large PBs. In H.265,

pixels may not be encoded and decoded yet in the neighborhood of the current PB. That is why the feedforward neural networks are trained to predict from a variable number of already encoded and decoded neighboring pixels by randomly masking some of these pixels during the training phase.

From now on, a feedforward neural network will be simply called neural network as only the focus will be on this family of neural networks exclusively.

3.2 Learning sparse decompositions versus learning shallow sparse autoencoder transforms

Sparse decompositions share similarities with shallow sparse autoencoder transforms. In this section, we first review sparse decomposition and its learning, and compare it with shallow sparse autoencoder transforms.

3.2.1 Sparse representation algorithms

The *sparse approximation* problem (P_0) looks for the best approximation of a data vector as a linear combination of a given small number $K \in \mathbb{N}^*$ of atoms from a redundant dictionary [104],

$$\mathbf{z} = \min_{\mathbf{z}_v} \|\mathbf{x} - \mathbf{D}\mathbf{z}_v\|_2^2 \text{ such that } \|\mathbf{z}_v\|_0 \leq K. \tag{3.20}$$

$\mathbf{z} \in \mathbb{R}^n$ denotes the sparse decomposition of the data vector $\mathbf{x} \in \mathbb{R}^m$ over the dictionary $\mathbf{D} \in \mathbb{R}^{m \times n}$. Note that *best* means *with minimum approximation error*. (P_0) has a so-called *sparse-constrained* formulation. Another formulation of (P_0) is *error-constrained*. The target is now to find the sparsest approximation of the data vector that achieves a given error $\varepsilon \in \mathbb{R}_+^*$ [105],

$$\mathbf{z} = \min_{\mathbf{z}_v} \|\mathbf{z}_v\|_0 \text{ such that } \|\mathbf{x} - \mathbf{D}\mathbf{z}_v\|_2^2 \leq \varepsilon. \tag{3.21}$$

Throughout Section 3.2.1, only (P_0) in its *sparse-contrained* formulation will be considered. This is because a given number of non-zero coefficients in the sparse decomposition of an image patch over a dictionary is a type of fixed rate constraint, and the compression task usually aims at obtaining the most accurate reconstruction of the image patch at a given rate.

The major issue concerning (P_0) is that it is NP-hard [106, 107]. At the present time, two approaches to this non-linear approximation exist. Convex relaxation methods turn (P_0) into a convex program (see LARS-Lasso and CoD described below). Greedy methods do a series of locally optimal choices to provide an approximate solution of (P_0) (see OMP described below).

Sparse autoencoders also provides a sparse decomposition of a data vector. More precisely, the coefficients of the sparse decomposition are computed by multiplying the data vector with a fixed matrix and then only the most significant terms are kept (see the T-sparse Auto-Encoder (T-sparse AE) and the Winner-Take-All Auto-Encoder (WTA AE) described below). Note that, as the rate-distortion criterion is a key, we pay attention

to sparse autoencoders able to create codes containing coefficients with 0 as value. For instance, the type of sparse autoencoder used for image denoising in [108, 109] is not relevant for image compression. Indeed, the autoencoder in [108] has the logistic non-linearity in the last layer of its encoder which computes the representation. Each coefficient in the representation is constrained to have a small positive value on average during the training phase. This does not yield coefficients with 0 as value in the representation.

From now on, the data vector $\mathbf{x}$ will be an image patch.

LARS-Lasso and Coordinate Descent (CoD) [110, 111] LARS-Lasso and CoD attempt to solve a relaxed version of (P_0) to find a sparse decomposition of an image patch over the dictionary,

$$\mathbf{z} = \min_{\mathbf{z}_v} \left(\|\mathbf{x} - \mathbf{D}\mathbf{z}_v\|_2^2 + \lambda \|\mathbf{z}_v\|_1 \right), \lambda \in \mathbb{R}_+^* \tag{3.22}$$

$$\tilde{\mathbf{x}} = \mathbf{D}\mathbf{z}. \tag{3.23}$$

Orthogonal Matching Pursuit (OMP) [112] OMP is the canonical greedy algorithm for (P_0) (see Algorithm 1). At each iteration $l \in [|1, K|]$, OMP computes a new approximation $\tilde{\mathbf{x}}^l$ of the image patch $\mathbf{x}$. The approximation residue $\mathbf{r}^l = \mathbf{x} - \tilde{\mathbf{x}}^l$ is used to determine which new atom in the dictionary is selected during the next iteration for refining the approximation. More precisely, the selected atom is the one whose inner product with the current residue $\mathbf{r}^l$ is the largest in absolute value.

Algorithm 1 : OMP.

Inputs: $\mathbf{x}$, $\mathbf{D}$, and K.

1: At initialization, $\mathbf{r}^0 = \mathbf{x}$ and $S^0 = \emptyset$.
2: **for all** $l \in [|1, K|]$ **do**
3: $\quad j = \underset{i \in \overline{S}^{l-1}}{\operatorname{argmin}} |\mathbf{D}_i^T \mathbf{r}^{l-1}|$ where $\overline{S}^{l-1} = [|1, n|] \setminus S^{l-1}$
4: $\quad S^l = S^{l-1} \cup j$
5: $\quad \mathbf{z}^l_{S^l} = (\mathbf{D}_{S^l})^\dagger \mathbf{x}$ and $\mathbf{z}^l_{\overline{S}^l} = \mathbf{0}$
6: $\quad \tilde{\mathbf{x}}^l = \mathbf{D}\mathbf{z}^l$
7: $\quad \mathbf{r}^l = \mathbf{x} - \tilde{\mathbf{x}}^l$
8: **end for**

Output: $\mathbf{z} = \mathbf{z}^K$.

$\mathbf{D}_i$ denotes the i^{th} column of $\mathbf{D}$. $\mathbf{D}_{S^l}$ is the subdictionary obtained by keeping the columns of $\mathbf{D}$ of indices in S^l. $(\mathbf{D}_{S^l})^\dagger$ is the More-Penrose pseudo inverse of $\mathbf{D}_{S^l}$. As for LARS-Lasso and CoD, the product between the dictionary and the sparse decomposition provides a reconstruction of the image patch,

$$\tilde{\mathbf{x}} = \mathbf{D}\mathbf{z}. \tag{3.24}$$

T-sparse Auto-Encoder (T-sparse AE) [113] The non-linearity in the encoder of the T-sparse AE imposes a given number $T \in \mathbb{N}^*$ of non-zero coefficients in the sparse representation of an image patch. To do this, it keeps the T largest coefficients in its input

vector and sets the rest to 0 (see (3.25)). Then, the decoder of the T-sparse AE gives a reconstruction of this image patch from the sparse representation (see (3.26)).

$$\mathbf{z} = f_T\left(\mathbf{V}\mathbf{x} + \mathbf{b}\right) \tag{3.25}$$

$$\tilde{\mathbf{x}} = \mathbf{\Phi}\mathbf{z} + \mathbf{c} \tag{3.26}$$

The trainable parameters of the T-sparse AE are the encoder weights $\mathbf{V} \in \mathbb{R}^{n \times m}$ and biases $\mathbf{b} \in \mathbb{R}^n$, the decoder weights $\mathbf{\Phi} \in \mathbb{R}^{m \times n}$ and biases $\mathbf{c} \in \mathbb{R}^m$. $\tilde{\mathbf{x}} \in \mathbb{R}^m$ is the reconstruction of $\mathbf{x}$. $f_T : \mathbb{R}^n \to \mathbb{R}^n$ denotes the T-sparse AE non-linearity.

We also introduce a variant of the T-sparse AE called T-absolute AE. It is identical to the T-sparse AE except that its application f_T keeps the T coefficients in its input vector which absolute values are the largest and sets the rest to 0.

Winner-Take-All Auto-Encoder (WTA AE) [114] The non-linearity in the encoder of the WTA AE allows only a proportion $\alpha \in]0, 1[$ of the coefficients in the set of sparse representations of the patches in an image to be different from 0 (see (3.27)). Then, the decoder of the WTA AE provides a reconstruction of the patches in this image from the set of sparse representations (see (3.28)). Unlike the case of the T-sparse AE, the sparsity constraint for the WTA AE considers the entire image representation.

$$\mathbf{M_z} = g_\alpha\left(\mathbf{W}\mathbf{M_x} + \mathbf{C_d}\right) \tag{3.27}$$

$$\mathbf{M}_{\tilde{\mathbf{x}}} = \mathbf{U}\mathbf{M_z} + \mathbf{C_e} \tag{3.28}$$

p image patches $\mathbf{x}^{(1)}, ..., \mathbf{x}^{(p)}$ are concatenated into a matrix $\mathbf{M_x} \in \mathbb{R}^{m \times p}$. p sparse vectors $\mathbf{z}^{(1)}, ..., \mathbf{z}^{(p)}$ are concatenated into a matrix $\mathbf{M_z} \in \mathbb{R}^{n \times p}$. The trainable parameters of the WTA AE are the encoder weights $\mathbf{W} \in \mathbb{R}^{n \times m}$ and biases $\mathbf{d} \in \mathbb{R}^n$, the decoder weights $\mathbf{U} \in \mathbb{R}^{m \times n}$ and biases $\mathbf{e} \in \mathbb{R}^m$. p copies of $\mathbf{d}$ are concatenated into a matrix $\mathbf{C_d} \in \mathbb{R}^{n \times p}$. p copies of $\mathbf{e}$ are concatenated into a matrix $\mathbf{C_e} \in \mathbb{R}^{m \times p}$. $g_\alpha : \mathbb{R}^{n \times p} \to \mathbb{R}^{n \times p}$ denotes the WTA AE non-linearity that keeps the $\alpha \times n \times p$ largest coefficients in its input matrix and sets the rest to 0.

(3.22), (3.25), (3.27), and Algorithm 1 illustrate that, unlike the T-sparse AE and the WTA AE, LARS-Lasso, CoD, and OMP go through an iterative process for encoding.

3.2.2 Comparison in terms of rate-distortion

In this section, LARS-Lasso, CoD, OMP, the T-sparse AE, and the WTA AE are compared on the same image compression experiment in terms of rate-distortion.

3.2.2.1 Training phase

Before running any comparison, LARS-Lasso, CoD, and OMP need pre-trained dictionaries. Similarly, the parameters of the T-sparse AE and those of the WTA AE must be learned.

The first aspect of the training phase to consider is the training data extraction. The Caltech-256 dataset [115] is a set of 256 objects categories containing 30607 images. The first 200 categories are dedicated to training which makes a total of 22823 images for training. The RGB color space is transformed into the $\mathrm{Y'C_bC_r}$ color space and we

only keep the luminance channel. $N \in \mathbb{N}^*$ patches of size $\sqrt{m} \times \sqrt{m}$ are randomly extracted from the 22823 luminance images. We remove the DC component from each patch. It yields a bank of patches $B = \left\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, ..., \mathbf{x}^{(N)}\right\}$. We define μ_j and σ_j as respectively the mean and the standard deviation of the j^{th} pixel x_j over all the patches in B. For $m \leq 144$ and $N \geq 600000$, we have noticed that, $\forall j \in [|1, m|]$, $\mu_j \approx 0$ and $\sigma_j \approx \sigma \in \mathbb{R}_+^*$. This remark must be kept aside as it will hold importance in the next paragraph.

The learning algorithms developped below are all gradient-based methods. This is because gradient-based methods have interesting convergence bounds in the context of large-scale Machine Learning [116]. In our case, the number of training image patches is 600000, i.e. close to the large-scale Machine Learning regime.

Learning D_λ for LARS-Lasso Given B and λ, the algorithm in [117] [1] generates $\mathbf{D}_\lambda$ by solving

$$\begin{aligned} &\min_{\mathbf{D}_\lambda \in \mathcal{C}, \mathbf{z}^{(1)}, ..., \mathbf{z}^{(N)}} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}\left(\mathbf{x}^{(i)}, \mathbf{z}^{(i)}, \mathbf{D}_\lambda, \lambda\right) \\ &\mathcal{L}\left(\mathbf{x}^{(i)}, \mathbf{z}^{(i)}, \mathbf{D}_\lambda, \lambda\right) = \left\|\mathbf{x}^{(i)} - \mathbf{D}_\lambda \mathbf{z}^{(i)}\right\|_2^2 + \lambda \left\|\mathbf{z}^{(i)}\right\|_1 \\ &\mathcal{C} = \left\{\mathbf{D} \in \mathbb{R}^{m \times n} \text{ such that}, \forall j \in [|1, n|], \left\|\mathbf{D}_j\right\|_2 = 1\right\}. \end{aligned} \tag{3.29}$$

In (3.29), the l_2-norm of each dictionary atom is constrained to prevent the dictionary from having arbitrarily large values, leading to arbitrarily small values of the coefficients in the sparse decomposition of an image patch over the dictionary.

Learning D_δ for CoD Given S and $\delta \in \mathbb{R}_+^*$, Algorithm 2 generates $\mathbf{D}_\delta$ by solving

$$\min_{\mathbf{D}_\delta \in \mathcal{C}, \mathbf{z}^{(1)}, ..., \mathbf{z}^{(N)}} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}\left(\mathbf{x}^{(i)}, \mathbf{z}^{(i)}, \mathbf{D}_\delta, \delta\right). \tag{3.30}$$

Algorithm 2 alternates between sparse decomposition steps that involve CoD and dictionary updates that use online stochastic gradient descent.

Learning D_o for OMP K-SVD [118] is the commonly used dictionary learning algorithm for OMP. But K-SVD is too slow when the number of examples in the training set is large. Unlike K-SVD, Algorithm 3 can quickly generate the dictionary $\mathbf{D}_o$ when the training set is large. Section 3.2.2.4 compares K-SVD and Algorithm 3 inside an image compression scheme in terms of rate-distortion. Given B and K, Algorithm 3 generates $\mathbf{D}_o$ by solving

$$\begin{aligned} &\min_{\mathbf{D}_o \in \mathcal{C}, \mathbf{z}^{(1)}, ..., \mathbf{z}^{(N)}} \frac{1}{N} \sum_{i=1}^{N} \mathcal{G}\left(\mathbf{x}^{(i)}, \mathbf{z}^{(i)}, \mathbf{D}_o\right) \text{ such that, } \forall i \in [|1, N|], \left\|\mathbf{z}^{(i)}\right\|_0 \leq K \\ &\mathcal{G}\left(\mathbf{x}^{(i)}, \mathbf{z}^{(i)}, \mathbf{D}_o\right) = \left\|\mathbf{x}^{(i)} - \mathbf{D}_o \mathbf{z}^{(i)}\right\|_2^2. \end{aligned} \tag{3.31}$$

1. Its implementation is the one from the SPAMS library at http://spams-devel.gforge.inria.fr/

Algorithm 2 : gradient descent dictionary learning for CoD.

Inputs: B, δ, and $\varepsilon \in \mathbb{R}_+^*$.

The coefficients of $\mathbf{D}_\delta$ are initialized from $\mathcal{U}\left[-0.2, 0.2\right]$.
$\forall j \in \left[|1, n|\right], \mathbf{D}_{\delta,j} \leftarrow \mathbf{D}_{\delta,j} / \left\|\mathbf{D}_{\delta,j}\right\|_2$
for several epochs **do**
  for all $i \in \left[|1, N|\right]$ **do**
    $\mathbf{z}^{(i)} = \min_{\mathbf{z}_v} \mathcal{L}\left(\mathbf{x}^{(i)}, \mathbf{z}_v, \mathbf{D}_\delta, \delta\right)$ using CoD.
    $\mathbf{D}_\delta \leftarrow \mathbf{D}_\delta - \varepsilon \ \partial \mathcal{L}\left(\mathbf{x}^{(i)}, \mathbf{z}^{(i)}, \mathbf{D}_\delta, \delta\right) / \partial \mathbf{D}_\delta$
    $\forall j \in \left[|1, n|\right], \mathbf{D}_{\delta,j} \leftarrow \mathbf{D}_{\delta,j} / \left\|\mathbf{D}_{\delta,j}\right\|_2$
  end for
end for

Output: $\mathbf{D}_\delta$.

Algorithm 3 alternates between sparse decomposition steps that involve OMP and dictionary updates that use online stochastic gradient descent. All the details regarding these dictionary updates are provided in Appendix B.1.

Algorithm 3 : gradient descent dictionary learning for OMP.

Inputs: B, K, and ε.

The coefficients of $\mathbf{D}_o$ are initialized from $\mathcal{U}\left[-0.2, 0.2\right]$.
$\forall j \in \left[|1, n|\right], \ \mathbf{D}_{o,j} \leftarrow \mathbf{D}_{o,j} / \left\|\mathbf{D}_{o,j}\right\|_2$
for several epochs **do**
  for all $i \in \left[|1, N|\right]$ **do**
    $\mathbf{z}^{(i)} = \text{OMP}\left(\mathbf{x}^{(i)}, \mathbf{D}_o, K\right)$
    $\mathbf{D}_o \leftarrow \mathbf{D}_o - \varepsilon \ \partial \mathcal{G}\left(\mathbf{x}^{(i)}, \mathbf{z}^{(i)}, \mathbf{D}_o\right) / \partial \mathbf{D}_o$
    $\forall j \in \left[|1, n|\right], \ \mathbf{D}_{o,j} \leftarrow \mathbf{D}_{o,j} / \left\|\mathbf{D}_{o,j}\right\|_2$
  end for
end for

Output: $\mathbf{D}_o$.

Algorithm 3 and K-SVD mainly differ in the scheduling of dictionary updates. At epoch t, for a given training image patch of B, Algorithm 3 updates all the atoms of $\mathbf{D}_o$ used in the sparse decomposition of that patch over $\mathbf{D}_o$. At epoch t, K-SVD updates only one time each atom of the dictionary; each atom update takes into account the approximation error of all the training patches of B using that atom in their sparse decomposition.

Training the T-sparse AE The T-sparse AE is trained by simply minimizing the average reconstruction error over the training set of image patches, i.e. given B and T,

$$\begin{aligned} &\min_{\mathbf{V},\mathbf{b},\mathbf{\Phi},\mathbf{c}} \frac{1}{N} \sum_{i=1}^{N} \left\|\mathbf{x}^{(i)} - \tilde{\mathbf{x}}^{(i)}\right\|_2^2 \\ &\text{such that, } \forall i \in \left[|1, N|\right], \left\|f_T\left(\mathbf{V}\mathbf{x}^{(i)} + \mathbf{b}\right)\right\|_0 \leq T. \end{aligned} \tag{3.32}$$

The non-linearity of the T-sparse AE alone defines the constraint on the number of non-zero coefficients in the sparse representation of an image patch during the training phase.

Training the WTA AE In a similar manner to the T-sparse AE, only the non-linearity of the WTA AE sets the constraint on the number of non-zero coefficients in the set of sparse representations of a group of image patches. Given B, α, and p,

$$\begin{aligned} &\min_{\mathbf{W},\mathbf{d},\mathbf{U},\mathbf{e}} \frac{1}{N} \sum_{i=1}^{N_p} \left\| \mathbf{M}_{\mathbf{x}}^{(i)} - \mathbf{M}_{\tilde{\mathbf{x}}}^{(i)} \right\|_F^2 \text{ where } N_p = \frac{N}{p} \\ &\text{such that, } \forall i \in [|1, N_p|], \left\| g_\alpha \left(\mathbf{W}\mathbf{M}_{\mathbf{x}}^{(i)} + \mathbf{C_d} \right) \right\|_0 \leq \alpha \times n \times p. \end{aligned} \tag{3.33}$$

$\|.\|_F$ stands for the Frobenius norm and $\|.\|_0$ counts the number of non-zero elements in its input matrix. p must be a divisor of N.

For the minimizations in (3.32) and (3.33), mini-batch stochastic gradient descent with momentum [119] is used. [120] gives many tricks to run backpropagation. In particular, we must be careful that the mean of each input to a neural network over all training examples be zero and all inputs have the same standard deviation. As explained in the previous paragraph, B satisfies this requirement.

3.2.2.2 Test phase

After the training phase in Section 3.2.2.1, the algorithms for sparse decomposition over a dictionary and the shallow sparse autoencoders undergo the same image compression experiment.

Before introducing the image compression scheme, test luminance images must be generated. In the Caltech-256 dataset, the 250^{th} category gathers 96 images of zebras. We pick them out of the dataset, convert the RGB color space into the $\text{Y}'\text{C}_\text{b}\text{C}_\text{r}$ color space and only keep the luminance channel. It yields the bank of luminance images denoted $\Gamma = \left\{ \mathbf{X}^{(1)}, ..., \mathbf{X}^{(96)} \right\}$. We choose the images of zebras for testing because they have disparate backgrounds. This makes them difficult to reconstruct. Note that the training set B does not contain any patches of zebras.

For each $i \in [|1, 96|]$, the luminance image $\mathbf{X}^{(i)}$ passes through the system displayed in Figure 3.12. Note that the 96 luminance images in Γ have different sizes. For $i \in [|1, 96|]$, $N^{(i)}$ varies. The quantizer applies a uniform scalar quantization on the values of the non-zero coefficients in the set $\left\{ \mathbf{z}^{(1)}, ..., \mathbf{z}^{\left(N^{(i)}\right)} \right\}$. The coding part draws on the codec in [121]. The set of indexes $\left\{ a_k^{(j)} \right\}_{j \in \left[|1, N^{(i)}|\right],\, k \in \left[|1, \|\mathbf{z}^{(j)}\|_0|\right]}$ is encoded using a fixed length code. The set of quantized values $\left\{ \tilde{\gamma}_k^{(j)} \right\}_{j \in \left[|1, N^{(i)}|\right],\, k \in \left[|1, \|\mathbf{z}^{(j)}\|_0|\right]}$ is entropy coded with an Huffman code. Here, we call PSNR, in decibels (dB), the average PSNR over the 96 luminance images in Γ. The mean rate over the 96 luminance images in Γ, denoted R, in bits per pixel (bbp), is defined as

$$\begin{aligned} R &= \frac{1}{96} \sum_{i=1}^{96} R^{(i)} \\ R^{(i)} &= \left(E^{(i)} + \log_2(n) \right) \frac{1}{m \times N^{(i)}} \sum_{j=1}^{N^{(i)}} \|\mathbf{z}^{(j)}\|_0. \end{aligned}$$

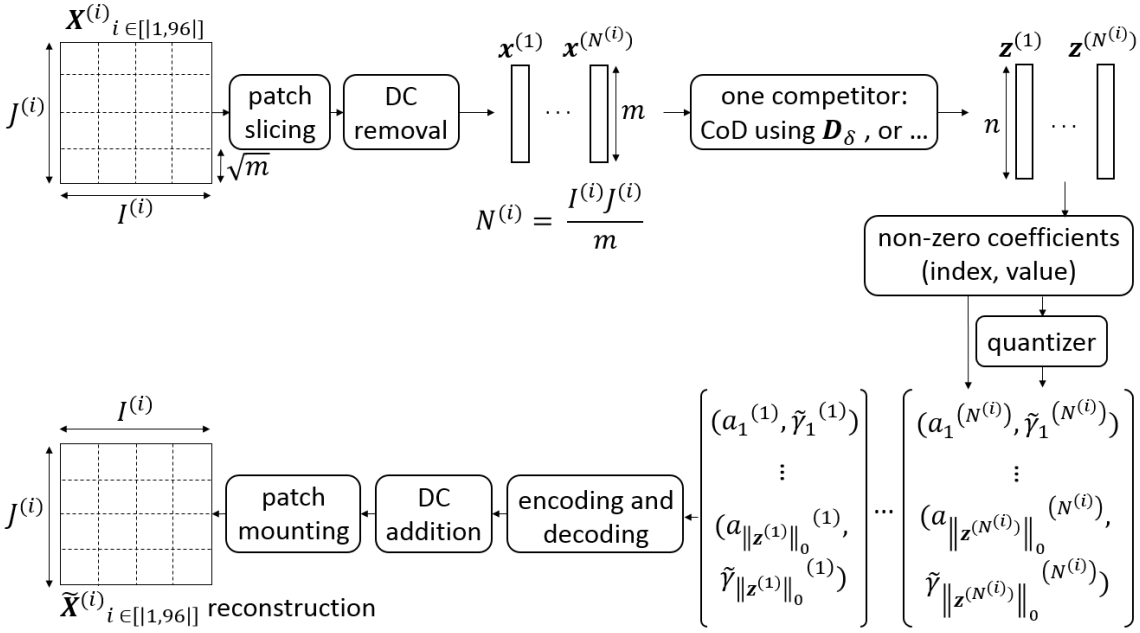


Figure 3.12: Image compression scheme on the test bank of luminance images.

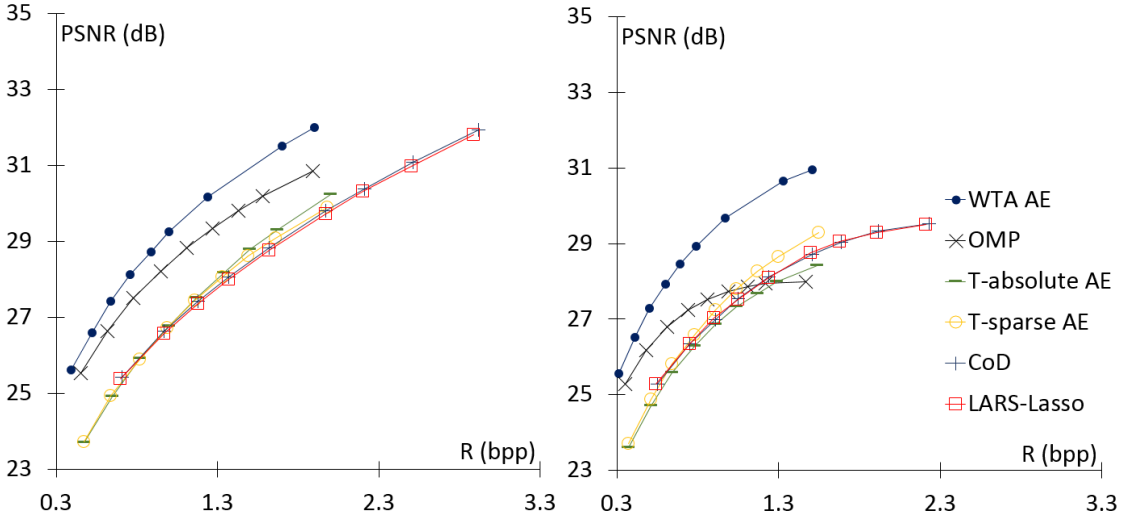
For $i \in [|1, 96|]$, $E^{(i)}$ refers to the entropy of the distribution of the set of quantized values $\left\{\tilde{\gamma}_k^{(j)}\right\}_{j\in\left[|1,N^{(i)}|\right],\, k\in\left[|1,\|\mathbf{z}^{(j)}\|_0|\right]}$. Note that R does not take into account the encoding cost of the DC components, as this has to be encoded for each evaluated method. R should also include the encoding cost of the number of non-zero coefficients. Note that, for the T-sparse AE and OMP, $\forall j \in \left[|1, N^{(i)}|\right]$, $\|\mathbf{z}^{(j)}\|_0 = \|\mathbf{z}\|_0$ independent of both i and j. For the WTA AE, LARS-Lasso, and CoD, for each $j \in \left[|1, N^{(i)}|\right]$, $\|\mathbf{z}^{(j)}\|_0$ varies. However, we neglect this contribution as its cost is small with respect to the cost of encoding the set of indexes $\left\{a_k^{(j)}\right\}_{j\in\left[|1,N^{(i)}|\right],\, k\in\left[|1,\|\mathbf{z}^{(j)}\|_0|\right]}$ and the set of quantized values $\left\{\tilde{\gamma}_k^{(j)}\right\}_{j\in\left[|1,N^{(i)}|\right],\, k\in\left[|1,\|\mathbf{z}^{(j)}\|_0|\right]}$.

Moreover, for OMP, R can be further optimized. Indeed, the coefficients associated to the first iterations of OMP have larger absolute values than the coefficients associated to the last iterations of OMP. Therefore, an entropy coding can be performed per iteration of OMP. For $k \in [|1, \|\mathbf{z}\|_0|]$, let $E_k^{(i)}$ be the entropy of the distribution of the set of quantized values $\left\{\tilde{\gamma}_k^{(j)}\right\}_{j\in\left[|1,N^{(i)}|\right]}$. For OMP only, R becomes R_o,

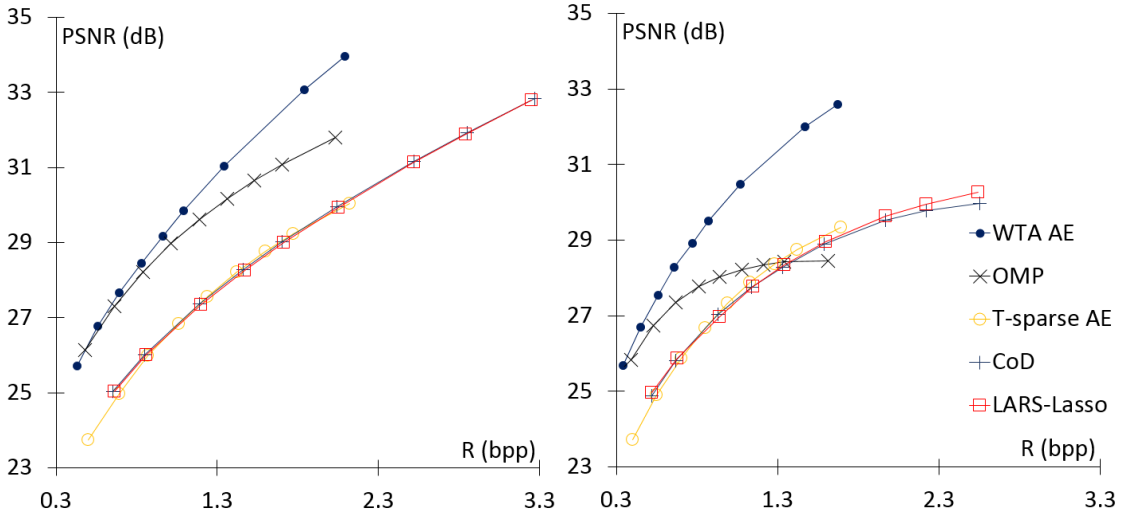
$$R_o = \frac{1}{96 \times m} \sum_{i=1}^{96} \sum_{k=1}^{\|\mathbf{z}\|_0} \left(E_k^{(i)} + \log_2(n)\right).$$

3.2.2.3 Rate-distortion analysis

For each dimension n of the sparse decompsition of an image patch over a dictionary, we start with the training phase in Section 3.2.2.1. $N = 600000$, $m = 144$, $T = K = 10$, $p = 20000$, $\alpha = \frac{10}{n}$, and $\lambda = \frac{\delta}{2} = 0.1$. The relationship between λ and δ comes from



(a) $n = 144$, 8-bit uniform quantization (b) $n = 144$, 5-bit uniform quantization



(c) $n = 288$, 8-bit uniform quantization (d) $n = 288$, 5-bit uniform quantization

Figure 3.13: Evolution of PNSR with R for three shallow sparse autoencoders (T-sparse AE, T-absolute AE, and WTA AE) and three algorithms for sparse decomposition over a dictionary (LARS-Lasso, CoD, and OMP).

the fact that the minimization problem in [117] differs from (3.29) by a factor 0.5. We have observed that these values for T, K, p, α, λ, and δ bring to each algorithm the best model for Section 3.2.2.2. For the T-sparse AE and the WTA AE, the learned weights are displayed in Appendix B.2. For CoD and OMP, the learned dictionaries are shown.

During the test phase, several values for the sparsity parameters of the T-sparse AE, the WTA AE, LARS-Lasso, CoD, and OMP are used to draw Figure 3.13. Note that, during the test phase, for the WTA AE, for $i \in [|1, 96|]$, $p = N^{(i)}$ varies.

Analysis for 8-bit uniform quantization The experiments in Figure 3.13 are conducted when LARS-Lasso uses $\mathbf{D}_\lambda$, CoD uses $\mathbf{D}_\delta$, and OMP uses $\mathbf{D}_o$. We also ran other experiments where LARS-Lasso, CoD, and OMP shared a common dictionary. We noticed

insignificant differences from what is displayed in Figure 3.13. The domination of OMP over LARS-Lasso and CoD has nothing to do with pre-trained dictionaries. It must be related to the nature of OMP. Note that this observation coincides with the conclusions of the authors in [122]. Their framework differs from ours as it is classification. They also suggest that the quality of pre-trained dictionaries has little impact as long as the dictionaries express relatively well the diversity of image patches. However, the sparse decomposition mechanism matters a lot.

For OMP, we previously compared R with its optimized variant R_o used in Figure 3.13. Over the range of PSNRs in Figure 3.13, $R - R_o$ typically belonged to $[0.01, 0.05]$ dB. It was not large. The good rate-distortion trade-offs of OMP compared to those of LARS-Lasso and CoD cannot be attributed to the fact that the measure of rate for OMP has been optimized.

We observe that, for equivalent rates, the PSNRs provided by the T-sparse AE are below those of OMP. We try to put forward an explanation. The main difference between autoencoders and the algorithms for sparse decomposition over a dictionary lies in the relation linking an image patch to its sparse vector. For autoencoders, this relation simply couples a linear combination and a point-wise non-linear function (see (3.25)). However, in the case of the algorithms for sparse decomposition over a dictionary, it is an iterative process. Given an image patch, autoencoders put more restrictions on the form its sparse vector can take. This limited diversity of sparse vectors leads to poorer reconstruction of image patches.

The WTA AE beats all its competitors in terms of rate-distortion trade-off. This can be explained by the way the sparsity constraint is handled. For OMP and the T-sparse AE, the number of non-zero coefficients in the sparse decomposition is fixed per image patch (see (3.25) and 1). Instead, for the WTA AE, it is fixed per set of image patches (see (3.27)). This allows to spread the resources over the patches, using less non-zero coefficients for patches with less complex texture. This WTA AE approach can be viewed as a suboptimal form of the *block sparsity selection using a global rate-distortion criterion* for OMP [121], where the coefficients leading to the largest distortion decrease are kept. Yet, the approach in [121] is highly complex.

Analysis for variable uniform scalar quantization Unlike OMP, the T-sparse AE and the WTA AE seem to be rarely affected by the quantization level. For $i \in [|1, 96|]$, the non-zero coefficients in the set $\left\{\mathbf{z}^{(1)}, ..., \mathbf{z}^{\left(N^{(i)}\right)}\right\}$ created by OMP are either positive or negative and most of them gather around zero. However, the T-sparse AE produces a set $\left\{\mathbf{z}^{(1)}, ..., \mathbf{z}^{\left(N^{(i)}\right)}\right\}$ whose non-zero coefficients are positive and are spreaded over a wide range. We assume that the impact of quantization arises from this difference in the distribution of the non-zero coefficients in the set $\left\{\mathbf{z}^{(1)}, ..., \mathbf{z}^{\left(N^{(i)}\right)}\right\}$. To verify this, we look at the behavior of the T-absolute AE. Basically, the only distinction between the T-sparse AE and the T-absolute AE comes from the fact that the T-absolute AE generates a set $\left\{\mathbf{z}^{(1)}, ..., \mathbf{z}^{\left(N^{(i)}\right)}\right\}$ that contains both positive and negative non-zero coefficients. Like OMP, the 5-bit uniform scalar quantization harms the PSNRs in the T-absolute AE curve. For OMP and the T-absolute AE, the quantization operation might cause many confusions between the small positive and negative non-zero coefficients in the set $\left\{\mathbf{z}^{(1)}, ..., \mathbf{z}^{\left(N^{(i)}\right)}\right\}$.

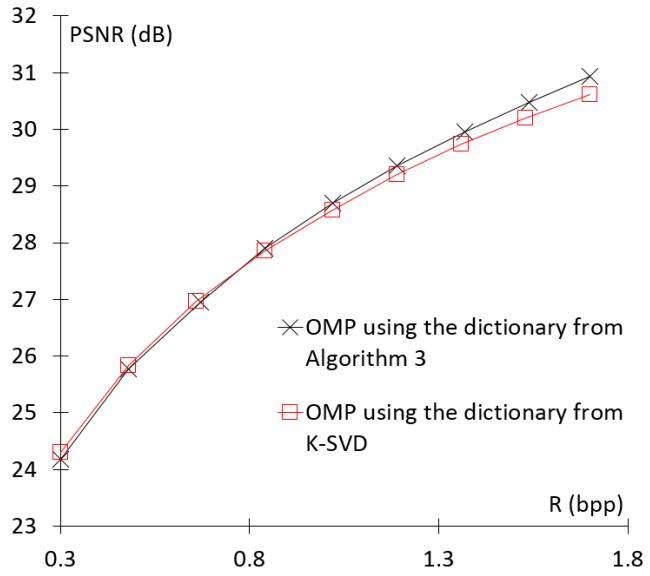


Figure 3.14: Evolution of PSNR with R when $n = 288$ and a 8-bit uniform scalar quantization is used. Algorithm 3, called gradient descent dictionary learning for OMP and defined in Section 3.2.2.1, considers one training image patch at a time and updates all the dictionary atoms used in the sparse decomposition of that patch over the dictionary via stochastic gradient descent. In contrast, K-SVD updates each dictionary atom by considering the approximaton error of all the training image patches that use that atom in their sparse decomposition. The dictionary learned via the gradient descent dictionary learning for OMP and that learned via K-SVD yield similar rate-distortion performance inside the image compression experiment.

The reconstruction of image patches is damaged by these confusions.

3.2.2.4 Comparison between K-SVD and the gradient descent dictionary learning for OMP

We restart the entire scheme in Sections 3.2.2.1 and 3.2.2.2. $N = 50000$ and $n = 288$. This way, K-SVD quickly trains a dictionary. Figure 3.14 compares two cases: OMP uses the dictionary from K-SVD and OMP uses the dictionary from the gradient descent dictionary learning for OMP. The dictionary learned via the gradient descent dictionary learning for OMP and the dictionary learned via K-SVD provide equivalent rate-distortion curves in the image compression scheme.

3.2.3 Complexity comparison

As additional analysis, LARS-Lasso, CoD, OMP, the T-sparse AE, and the WTA AE are compared in terms of algorithmic complexity.

Table 3.2 summarizes the complexity per image patch for OMP based on a fast QR-1 decomposition [123], the T-sparse AE, and the WTA AE. The basic implementation of LARS-Lasso [124] has complexity $\mathcal{O}(n^3)$, assuming that $m \leq n$. [117] suggests an efficient Cholesky-based implementation of LARS-Lasso and processes batches of $\eta \in \mathbb{N}^*$ image patches. Using these optimizations, LARS-Lasso becomes only slightly slower

Table 3.2: complexity per image patch assuming that $\|\mathbf{z}\|_0 \ll m \leq n$. The T-sparse AE and the WTA AE are the least complex methods.

Method	Complexity
OMP	$\mathcal{O}(mn \|\mathbf{z}\|_0)$ [123]
T-sparse AE	$\mathcal{O}(mn)$
WTA AE	$\mathcal{O}(mn)$

than OMP. One iteration of CoD has complexity $\mathcal{O}(n)$ [111] and the number of iterations is related to a threshold on the reconstruction error. CoD can be either faster than OMP or much slower depending on this threshold so we do not show the complexity of CoD. Note that, in our experiments, CoD is much slower than OMP as we set a low threshold to get the best PSNRs from CoD.

We clarify the complexity of the WTA AE displayed in Table 3.2. In (3.27), the complexity of the matrix product $\mathbf{W}\mathbf{M}_{\mathbf{x}}$ is $\mathcal{O}(nmp)$. In (3.27), g_α requires a sorting operation of complexity $\mathcal{O}(np\log(np))$. During the test phase, $200 \leq p \leq 8000$ so $\mathcal{O}(nmp)$ dominates $\mathcal{O}(np\log(np))$. The complexity per image patch of the WTA AE is $\mathcal{O}(nm)$.

3.2.4 Conclusion on sparse decompositions versus shallow sparse autoencoders

Sparse decompositions and shallow sparse autoencoders share many similarities. In Section 3.2, both approaches applied to image compression have been compared. This is a contribution per se. In particular, the following conclusions can be drawn.

- On the encoder side, the iterative decomposition over a dictionary (OMP) is more efficient in terms of rate-distortion than a single-step decomposition onto a basis (T-sparse AE), see Section 3.2.2.3.
- A global sparsity constraint spread over the sparse decompositions of several image patches (WTA AE) is more efficient in terms of rate-distortion than a sparsity constraint on each sparse decomposition separately, see Section 3.2.2.3.
- The two considered shallow sparse autoencoders are robust to quantization noise.
- The two considered shallow sparse autoencoders are much less complex than LARS-Lasso, CoD, and OMP.
- The algorithms for sparse decomposition over a dictionary solving a relaxed version of (P_0) (LARS-Lasso and CoD) are not efficient in terms of rate-distortion, see Section 3.2.2.3.

3.3 Conclusion

In this section, the family of feedforward neural networks trained via backpropagation is introduced. The generalization of neural networks and the choice of the neural network architecture are two open issues. That is why, in the rest of the document, we resort to experimentation to take decisions regarding the regularizations during the training phase, the number of training examples in the training sets, and the architectures of the neural

networks. This section ends with a comparison between the algorithms for sparse decomposition over a dictionary and the shallow sparse autoencoders in the context of the learning of a transform for image compression. This comparison indicates that a shallow sparse autoencoder is promising in terms of rate-distortion, provided that a global sparsity constraint is applied to the representation of the whole image. This serves as basis for Chapter 4.

Chapter 4

Learning a transform for image compression

The learning of a transform for image compression, which is reviewed in Chapter 3, is motivated by the mismatch between the assumptions of the results of optimality in a rate-distortion sense for transform coding on the distribution of the image pixels and their true distribution. Indeed, there exists a result of optimality in a rate-distortion sense for transform coding in two different cases. (i) If the input to the transform is a zero-mean Gaussian vector, the KLT is optimal [9, Proposition 7.2]. (ii) If the input to the transform follows a Gaussian-Markov process of order 1 with high correlation, the DCT is optimal [9, Section 7.1.1] [10, Section 7.5.3] (see Section 1.2). But, the distribution of the image pixels is neither jointly Gaussian nor a Gaussian-Markov process of order 1 with high correlation. Hence, the idea of learning a transform from a large image database for image compression.

The main challenge regarding the learning of a transform for image compression is to make the learned transform efficient in terms of rate-distortion. This requires to integrate a reliable measure of the rate into the training phase and increase the number of neurons in the representation. For this purpose, two approaches are studied.

- **Expressing the rate as a function of the number of non-zero coefficients in the representation**. A sparsity constraint is set on the representation. The number of non-zero coefficients in the sparse representation of the entire image is constrained as a global sparsity constraint is shown to be more efficient in terms of rate-distortion than a sparsity constraint on the sparse representation of each image patch separately (see Section 3.2.2.3). Thanks to this sparsity constraint, the representation can be either complete or overcomplete while being compressible.
- **Approximating the rate via the entropy of the quantized representation**. The number of non-zero coefficients in the representation is still an unprecise measure of the rate as the rate depends on the number of non-zero coefficients in the representation and their distribution. An alternative is to integrate into the objective function to be minimized an approximate measure of the entropy of the quantized representation.

Besides, deep autoencoders rather than shallow ones will be explored for the following reason. Recent works studying the approximation power of neural networks seem to converge towards a common conclusion regarding the benefit of increasing the number of

layers in a neural network. In order for a neural network to achieve a given approximation error, the total number of required neurons decreases as the number of layers increases [13, 14] (see Section 1.2). It is important to stress that, in the demonstrations of the previous results, only some families of simple functions are considered. These functions might be very different from the transform we want to approximate. The previous results lead to the intuition that, without changing the number of neurons in the representation of an autoencoder, i.e. without changing the rate computed from the representation, the addition of layers to this autoencoder may raise its attainable accuracy of approximation of the hypothetic optimal transform in terms of rate-distortion. This holds true for the WTA AE introduced in Chapter 3.

The second challenge about the learning of a transform for image compression is to learn a unique transform that can compress at any rate during the test phase. This kind of transform is called *rate-agnostic*. We have observed that our shallow sparse autoencoders are naturally *rate-agnostic* whereas, in the case of the deep sparse autoencoders, the sparsity level in the sparsity constraint on the representation need to vary during the training phase to make the transform *rate-agnostic*. In the case of the autoencoders trained via an entropy constraint on their quantized representation, the learned transform is viewed as non-*rate-agnostic* as the level of the constraint is fixed during the training phase. However, it is shown that, during the test phase, the compression scheme including the learned transform can compress efficiently at different rates by varying the quantization step size according to a specific scheduling.

4.1 Rate as a function of the number of non-zero coefficients in the representation

The goal in Chapter 4 is to learn a transform that is both efficient in terms of rate-distortion and *rate-agnostic*. This section presents the first approach to reach this goal. To be efficient in terms of rate-distortion, a deep sparse autoencoder with an overcomplete representation is designed and the rate is expressed as a function of the number of non-zero coefficients in the representation. The architecture of this deep sparse autoencoder is based on the architecture of the WTA AE presented in Section 3.2.1 as the WTA AE has proven to be relatively efficient in terms of rate-distortion.

The contributions are:

- A deep sparse autoencoder architecture for image compression.
- A novel way to code the sparse quantized representation for rate-distortion efficient.
- A training with a stochastically varying sparsity parameter for adapting the deep sparse autoencoder to the compression at different rates.

4.1.1 Stochastic Winner-Take-All Auto-Encoder (SWTA AE)

We now present a deeper version of the WTA AE, called Stochastic Winner-Take-All Auto-Encoder (SWTA AE), whose architecture is shown in Figure 4.1. Two key choices for the architecture of the SWTA AE are explained below. Then, details on the sparsity constraint are provided.

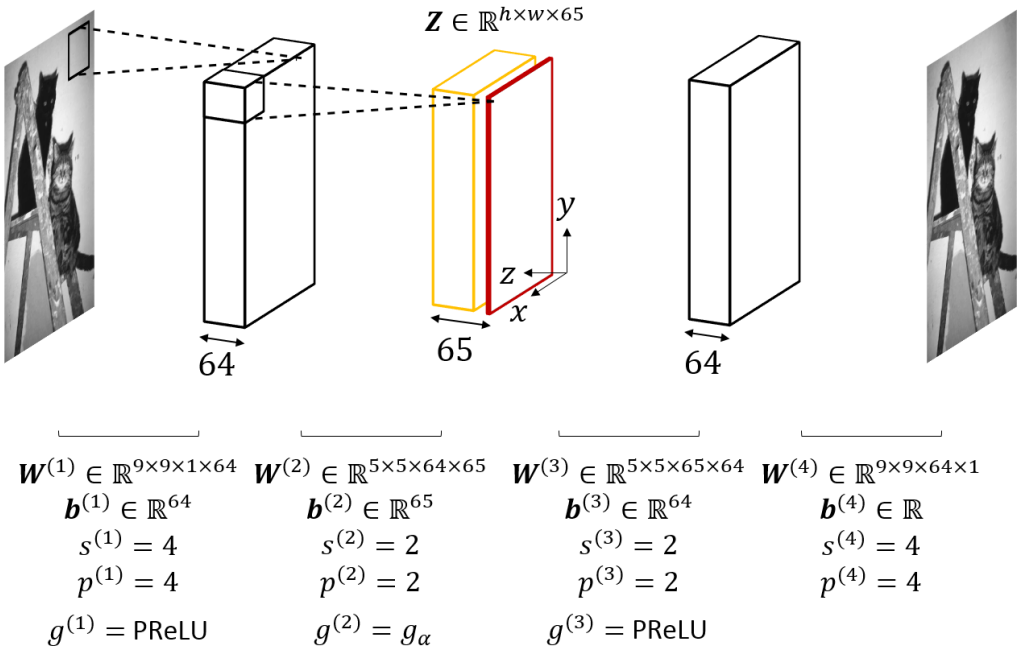


Figure 4.1: SWTA AE architecture. PReLU refers to Parametrized Rectified Linear Unit [65].

Fully convolutional A compression algorithm must process images of various sizes. However, the commonly used neural network architectures [66, 65] include both convolutional layers and fully-connected layers, and the number of parameters of the latters directly depends on the image size. This imposes to train one architecture per image size. That is why our proposed SWTA AE only contains convolutional layers. Its encoder has two convolutional layers and its decoder has two deconvolutional layers [125]. Each layer indexed by $i \in [|1, 4|]$ consists in convolving the layer input with the bank of filters $\mathbf{W}^{(i)}$, adding the biases $\mathbf{b}^{(i)}$, and applying a non-linear mapping $g^{(i)}$ if $i \neq 4$, producing the layer output. For the borders of the layer input, zero-padding of width $p^{(i)}$ is used.

Strided convolution Max-pooling is a core component of neural networks [126] that downsamples its input representation by applying a max filter to non-overlapping subregions. But max-pooling increases the rate. Indeed, if the encoder contains a max-pooling layer, the locations of maximum activations selected during pooling operations must be recorded and transmitted to the corresponding unpooling layer in the decoder [127, 128]. Instead, for $i \in [|1, 2|]$, we downsample using a fixed stride $s^{(i)} > 1$ for convolution, which does not need any signalling.

Semi-sparse representation In Figure 4.1, $\mathbf{Z}$ is the representation of the input image that is quantized and entropy coded to give the bitstream. The WTA constraint controls the coding cost of the representation. The WTA constraint is defined via a mapping $g_\alpha : \mathbb{R}^{h\times w\times 64} \to \mathbb{R}^{h\times w\times 64}$, where $\alpha \in]0, 1[$ is the WTA parameter. g_α keeps the $\alpha \times h \times w \times 64$ most representative coefficients of all input feature maps, i.e. those whose absolute values are the largest, and sets the rest to 0. g_α only applies to the output of the convolution in the second layer involving the first 64 filters in $\mathbf{W}^{(2)}$, producing the first 64 sparse feature

maps in the output of the encoder $\mathbf{Z}$. Figure 4.1 displays these sparse feature maps in orange. Varying α leads to various coding costs of $\mathbf{Z}$. Note that [114] uses WTA, but our WTA rule is different and g_α does not apply to specific dimensions of its input tensor as this constraint is not relevant for image compression.

A patch of the input image might be represented by a portion of the first 64 sparse feature maps in $\mathbf{Z}$ that only contains zeros. We want to ensure that each image patch has a minimum code in $\mathbf{Z}$ to guarantee a sufficient quality of reconstruction per patch. That is why the last feature map in $\mathbf{Z}$ is not sparse. Figure 4.1 displays it in red. Interestingly, we have noticed that, at the end of the training phase in Section 4.1.3, the SWTA AE stores in the last feature map a subsampled version of its input image.

Bitstream generation The coding of the positions of the non-zero coefficients represents an important part of the rate. The coding cost for a brute force coding of the positions of the non-zero coefficients increases linearly with the number of non-zero coefficients inside the first 64 features maps in $\mathbf{Z}$ and logarithmically with the number of coefficients. By contrast, we propose to code the positions of the non-zero coefficients as explained hereafter. Figure 4.1 defines a coordinate system (x, y, z) for $\mathbf{Z}$. The non-zero coefficients inside the first 64 feature maps in $\mathbf{Z}$ are scanned along (x, y, z) where z changes the fastest. For each pair (x, y), the number of non-zero coefficients along z is coded with a Huffman code and the position of each non-zero coefficient along z is coded with a fixed-length code. A Huffman code is used for coding the number of non-zero coefficients along z as, at a given position (x, y), a small number of non-zero coefficients is much more frequent than a large one. This coding unequivocally characterizes the position of each non-zero coefficient inside the first 64 feature maps in $\mathbf{Z}$.

The values of the non-zero coefficients inside the first 64 feature maps in $\mathbf{Z}$ are uniformly quantized over 8 bits and coded with a Huffman code.

Number of trainable parameters The trainable parameters of the SWTA AE are all the weights and biases displayed in Figure 4.1. By counting them, the number of trainable parameters is 218562.

4.1.2 Winner-Take-All Orthogonal Matching Pursuit (WTA OMP)

OMP, with a learned dictionary, is an algorithm for sparse decomposition yielding better rate-distortion trade-offs than the T-sparse AE, which is a shallow sparse autoencoder with a sparsity constraint on the representation of each image patch individually. However, OMP provides worse rate-distortion trade-offs than the WTA AE, which is a shallow sparse autoencoder with a global sparsity constraint on the representations of all the patches in an image (see Section 3.2.2.3). For fairness, the SWTA AE, a deep version of the WTA AE, must be compared to a variant of OMP that incorporates a global sparsity constraint on the representations of all the patches in an image. We thus introduce this variant of OMP, called Winner-Take-All Orthogonal Matching Pursuit (WTA OMP).

In a given set of image patches, WTA OMP first decomposes each image patch over a dictionary. Then, it keeps a given number of coefficients with largest absolute value in the set of sparse decompositions and sets the rest to 0. The support of the sparse

decomposition of each image patch over the dictionary has therefore been changed. Hence the need for a final least-square minimization (see Algorithm 4).

Algorithm 4 : WTA_OMP

Inputs: $\mathbf{X}$, $\mathbf{D}$, $K < m$, and γ.

1: $\forall j \in [|1, p|]$, $\mathbf{Y}_j = \text{OMP}(\mathbf{X}_j, \mathbf{D}, K)$
2: $\mathbf{I} = f_\gamma(\mathbf{Y})$
3: $\forall j \in [|1, p|]$, $\mathbf{Z}_j = \min_{\mathbf{z} \in \mathbb{R}^n} \|\mathbf{X}_j - \mathbf{D}\mathbf{z}\|_2^2$ such that $\text{supp}(\mathbf{z}) = \text{supp}(\mathbf{I}_j)$

Output: $\mathbf{Z}$.

$\mathbf{X} \in \mathbb{R}^{m \times p}$ is a matrix whose columns are formed by p image patches of m pixels. $\mathbf{Y} \in \mathbb{R}^{n \times p}$ is a matrix whose columns are formed by the sparse decompositions of the image patches over the dictionary $\mathbf{D}$. For $\gamma \in]0, 1[$, $f_\gamma : \mathbb{R}^{n \times p} \to \mathbb{R}^{n \times p}$ keeps the $\gamma \times n \times p$ coefficients with largest absolute value in its input matrix and sets the rest to 0. For $j \in [|1, p|]$, $\mathbf{X}_j$ denotes the j^{th} column of $\mathbf{X}$. $\text{supp}(\mathbf{z}) = \{i \in [|1, n|] \mid \mathbf{z}_i \neq 0\}$ is the support of vector $\mathbf{z}$.

4.1.3 Training phase

Before moving on to the image compression experiment in Section 4.1.4, the SWTA AE needs training. Similarly, the dictionary must be learned for WTA OMP.

1.0×10^5 RGB images are extracted from the ILSVRC2012 ImageNet training set [129]. The RGB color space is transformed into the $\text{Y}'\text{C}_\text{b}\text{C}_\text{r}$ color space and only the luminance channel is kept. For the SWTA AE, the luminance images are resized to 321×321. $\mathbf{M} \in \mathbb{R}^{321 \times 321}$ denotes the mean of all luminance images. $\sigma \in \mathbb{R}_+^*$ is the mean of the standard deviation over all luminance images. Each luminance image is subtracted by $\mathbf{M}$ and divided by σ. These images are concatenated into a training set $\mathbf{\Delta} \in \mathbb{R}^{321 \times 321 \times 10^5}$. For WTA OMP, $\eta = 1.2 \times 10^6$ image patches of size $\sqrt{m} \times \sqrt{m}$ are randomly sampled from the luminance images. We remove the DC component from each patch. These patches are concatenated into a training set $\mathbf{\Gamma} \in \mathbb{R}^{m \times \eta}$.

Training the SWTA AE The WTA parameter α, which is the proportion of non-zero coefficients in the first 64 feature maps of $\mathbf{Z}$, tunes the coding cost of $\mathbf{Z}$. If the WTA parameter is fixed during the training phase, all the filters and the biases of the SWTA AE are learned for one rate. That is why the WTA parameter is turned into a stochastic hyperparameter during the training phase. This justifies the prefix *Stochastic* in SWTA AE. Since there is no reason to favor some rates during the training phase, α is sampled according to the uniform distribution $\mathcal{U}[\mu - \epsilon, \mu + \epsilon]$, where $\mu - \epsilon > 0$ and $\mu + \epsilon < 1$. $\mu = 1.8 \times 10^{-1}$ and $\epsilon = 1.7 \times 10^{-1}$ are chosen so that the support of α is large. At each training epoch, α is drawn for each training image of the training set $\mathbf{\Delta}$.

The SWTA AE is able to process images of various sizes. During the training phase, we feed the SWTA AE with random crops of size 49×49 of the training images of the training set $\mathbf{\Delta}$. This accelerates training considerably. The training objective is to minimize the mean squared error between these cropped images and their reconstruction plus l_2-norm weights decay. Mini-batch stochastic gradient descent with momentum is

used. The gradient descent learning rate is fixed to 2.0×10^{-5}, the momentum is 0.9, and the size of mini-batches is 5. The weights decay coefficient is 5.0×10^{-4}. Our implementation is based on Caffe [130]. It adds to Caffe the tools for the semi-sparse representation and the bitstream generation in Section 4.1.1.

Learning $\mathbf{D}_w$ for WTA OMP WTA OMP is a novel algorithm for sparse decomposition we introduced. It therefore requires a specific training, which is detailed below. Given $\mathbf{\Gamma}$, $K < m$, and γ, Algorithm 5 generates $\mathbf{D}_w$ by solving

$$\min_{\mathbf{D}_w, \mathbf{Z}_1, ..., \mathbf{Z}_\eta} \frac{1}{\eta} \sum_{j=1}^{\eta} \|\mathbf{\Gamma}_j - \mathbf{D}_w \mathbf{Z}_j\|_2^2 \text{ such that } \forall j \in [|1, \eta|], \|\mathbf{Z}_j\|_0 \leq K \text{ such that } \sum_{i=j}^{\eta} \|\mathbf{Z}_j\|_0 \leq \gamma \times n \times \eta. \tag{4.1}$$

Algorithm 5 alternates between sparse decomposition steps that involve WTA OMP and dictionary updates that use mini-batch stochastic gradient descent. Given $\mathbf{\Gamma}$ and p, let ϕ be a function that randomly partitions $\mathbf{\Gamma}$ into $\eta_p = \eta \ / \ p$ mini-batches $\left\{\mathbf{X}^{(1)}, ..., \mathbf{X}^{(\eta_p)}\right\}$, where, for $i \in [|1, \eta_p|]$, $\mathbf{X}^{(i)} \in \mathbb{R}^{m \times p}$.

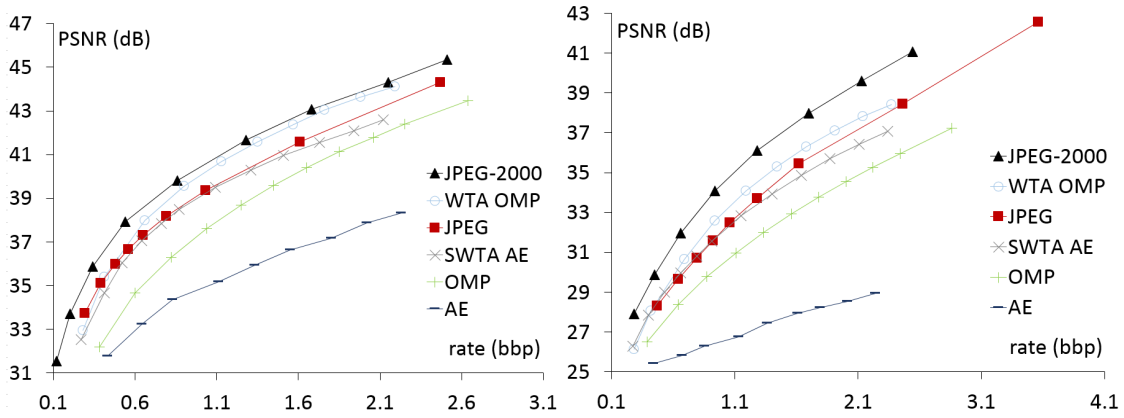
Algorithm 5 : learning $\mathbf{D}_w$ for WTA OMP.

Inputs: $\mathbf{\Gamma}$, $K < m$, γ, p, and $\varepsilon \in \mathbb{R}_+^*$.

The coefficients of $\mathbf{D}_w$ are initialized from $\mathcal{U}[-0.2, 0.2]$.
$\forall j \in [|1, n|]$, $\mathbf{D}_{w,j} \leftarrow \mathbf{D}_{w,j} / \|\mathbf{D}_{w,j}\|_2$
for several epochs **do**
    $\left[\mathbf{X}^{(1)}, ..., \mathbf{X}^{(\eta_p)}\right] = \phi(\mathbf{\Gamma}, p)$
    for all $i \in [|1, \eta_p|]$ **do**
        $\mathbf{Z}^{(i)} = \text{WTA\_OMP}\left(\mathbf{X}^{(i)}, \mathbf{D}_w, K, \gamma\right)$
        $\mathbf{D}_w \leftarrow \mathbf{D}_w - \varepsilon \ \partial \left\|\mathbf{X}^{(i)} - \mathbf{D}_w \mathbf{Z}^{(i)}\right\|_F^2 \Big/ \partial \mathbf{D}_w$
        $\forall j \in [|1, n|]$, $\mathbf{D}_{w,j} \leftarrow \mathbf{D}_{w,j} / \|\mathbf{D}_{w,j}\|_2$
    end for
end for

Output: $\mathbf{D}_w$.

Learning $\mathbf{D}_o$ for OMP In the image compression experiment in Section 4.1.4, OMP is used as baseline for comparison. That is why OMP also needs a learned dictionary $\mathbf{D}_o$. Given $\mathbf{\Gamma}$, the dictionary $\mathbf{D}_o$ is learned via Algorithm 3. m and n are optimized with an exhaustive search. This leads to $m = 64$ and $n = 1024$. For the SWTA AE and WTA OMP, the same values for m and n are used. Moreover, $K = 15$, $\gamma = 4.5 \times 10^{-3}$, $p = 10$, and $\varepsilon = 2.0 \times 10^{-2}$. For this configuration, the learned dictionaries $\mathbf{D}_w$ for WTA OMP and $\mathbf{D}_o$ for OMP are displayed in Appendix B.3.



(a) Lena luminance 512×512 pixels. (b) Barbara luminance 480×384 pixels.

Figure 4.2: Evolution of PNSR with the rate for a deep sparse autoencoder (SWTA AE), a regular autoencoder (AE), two algorithms for sparse decomposition over a dictionary (OMP and WTA OMP), JPEG, and JPEG2000.

4.1.4 Image compression experiment

After the training phase in Section 4.1.3, the rate-distortion curves of OMP, WTA OMP, the SWTA AE, JPEG, and JPEG2000 are compared. For this comparison, four test luminance images are extracted from Lena, Barbara, Mandrill, and peppers respectively. None of these four images exists in the training set.

Image codec for the SWTA AE Each input test luminance image is pre-processed similarly to the training in Section 4.1.3. The mean learned image $\mathbf{M}$ is interpolated to match the size of the input image. Then, the input image is subtracted by this interpolated mean image and divided by the learned σ. The encoder of the SWTA AE computes the representation $\mathbf{Z}$. The bitstream is obtained by processing the representation as detailed in Section 4.1.1.

Image codec for OMP and WTA OMP A luminance image is split into 8×8 non-overlapping patches. The DC component is removed from each patch. The DC components are uniformly quantized over 8 bits and coded with a fixed-length code. OMP (or WTA OMP) finds the sparse decompositions of the image patches over $\mathbf{D}_o$ (or $\mathbf{D}_w$). The non-zero coefficients are uniformly quantized over 8 bits and coded with a Huffman code while their position is coded with a fixed-length code.

Then, for WTA OMP only, the number of non-zero coefficients of the sparse decomposition of each patch over the dictionary $\mathbf{D}_w$ is coded with a Huffman code.

4.1.5 Rate-distortion analysis

We compare the SWTA AE with JPEG and JPEG2000 [1]. The target of the analysis is to see whether the SWTA AE can exploit the statistical dependencies between the image

1. The code of both JPEG and JPEG2000 can be found at http://www.imagemagick.org/script/index.php

pixels. That is why a simple Huffman code is used for coding the values of the non-zero coefficients in the case of the SWTA AE, not a context-based entropy coder, as in JPEG and JPEG2000. Furthermore, the SWTA AE is compared with its non-sparse Auto-Encoder counterpart (AE). The AE has the same architecture as the SWTA AE but its representation only contains non-sparse feature maps. Note that, to draw a new point in the AE rate-distortion curve, the AE must be first re-trained with a different number of feature maps in its representation.

Figure 4.2 shows the rate-distortion curves of OMP, WTA OMP, the AE, the SWTA AE, JPEG, and JPEG2000 for two of the most common images: Lena and Barbara. In terms of rate-distortion trade-off, the SWTA AE outperforms the AE and WTA OMP is better than OMP. This highlights the value of the Winner-Take-All approach for image compression. When comparing the SWTA AE and WTA OMP, we observe that the iterative decomposition is more efficient for image compression via sparse representations. Moreover, the SWTA AE can compete with JPEG. We also ran this image compression experiment on several crops of Lena and Barbara and observed that the relative position of the six rate-distortion curves was comparable to the relative positioning in Figure 4.2. The size of the test image does not affect the performance of the SWTA AE. More simulation results and a complexity analysis for OMP, WTA OMP, and the SWTA AE can be found online [2].

4.1.6 Limitations of the deep sparse autoencoders

We have introduced a deep sparse autoencoder (SWTA AE) with a Winner-Take-All non-linearity. The sparsity parameter of the non-linearity is stochastically driven during the training phase so that the SWTA AE adapts to the compression at different rates. Moreover, an efficient way of coding the quantized representation in the SWTA AE is proposed. These contributions enable to improve the rate-distortion trade-offs provided by the SWTA AE. But, the SWTA AE does not beat WTA OMP in terms of rate-distortion.

A problem is that the WTA constraint causes an extreme sparsity in the representation of the SWTA AE. This implies that, during the training of the SWTA AE, the gradients flow through the representation, and most of their components are set to 0. In other words, the gradients vanish. As a consequence, the decoder of the SWTA AE is trained much faster than the encoder. The vanishing gradients hamper the training of the SWTA AE, especially when its depth is large. Note that the cause of the vanishing gradients we observe is different from the cause of the vanishing gradients appearing during the training of recurrent neural networks [131].

Given this issue, Section 4.2 develops another approach for learning a transform for image compression that meets the two targets: efficiency in terms of rate-distortion and a *rate-agnostic* transform. This approach is not based on sparsity but on the minimization of the entropy of the quantized representation.

2. https://www.irisa.fr/temics/demos/NeuralNets/AutoEncoders/swtaAE.htm

4.2 Rate as the entropy of the quantized representation

There are two apparent limitations regarding the learning of a transform for image compression via a deep sparse autoencoder. (i) The gradient vanishing issue hampers the training of the deep sparse autoencoder (see Section 4.1.6). (ii) The expression of the rate as a function of the number of non-zero coefficients in the sparse representation of the image is not close enough to the true rate as, for instance, the fact that the quantized non-zero coefficients are losslessly encoded via an entropy coder is not taken into account. Another idea would be to train a deep autoencoder via the minimization over its parameters of an objective function combining a distortion term and a term measuring the entropy of the quantized representation.

But, the issue is that the deep autoencoder trained via the minimization of this objective function over its parameters must be *rate-agnostic*. Indeed, during the training phase, the rate is determined by the weight between the two terms in the objective function. But, during the test phase, this weight has no influence. In this case, a way to modify the rate during the test phase is to change the quantization step sizes of the uniform scalar quantizer. Now, the problem is the choice of quantization step sizes during both the training phase and the test phase so that the entire compression scheme is efficient in terms of rate-distortion at any rate. Should the quantization step sizes be imposed during the training phase? To answer this, we propose an approach where the transform and the uniform scalar quantizer are learned jointly. Then, we investigate whether, during the test phase, the compression falls apart when the representation obtained via the learned transform is quantized using quantization step sizes which differ from those in the training stage.

4.2.1 Joint learning of the transform and the quantizer

Section 4.2.1 introduces the autoencoder trained via the minimization over its parameters of an objective function combining a distortion term and a term measuring the entropy of the quantized representation. Then, it details our proposal for learning jointly this autoencoder transform and the uniform scalar quantizer.

4.2.1.1 Autoencoder trained via a constraint on the entropy of the quantized representation

The encoder g_e of the autoencoder, parametrized by $\boldsymbol{\theta}$, computes a representation $\mathbf{Y}$ from the image $\mathbf{X}$. Its decoder g_d, parametrized by $\boldsymbol{\phi}$, gives a reconstruction $\hat{\mathbf{X}}$ of the image $\mathbf{X}$ from the quantized representation $\hat{\mathbf{Y}} = \mathcal{Q}(\mathbf{Y})$ (see Figure 4.3). If an autoencoder has fully-connected layers [132, 133, 134], the number of parameters depends on the image size. This implies that one autoencoder has to be trained per image size. To avoid this, an architecture without fully-connected layer is chosen. It exclusively comprises convolutional layers and non-linear operators. In this case, $\mathbf{Y} \in \mathbb{R}^{h \times w \times m}$ is a set of m feature maps of size $n = h \times w$ (see Figure 4.3).

The basic autoencoder training minimizes the image reconstruction error [72]. In order to perform a rate-distortion optimization, the authors in [83] add the minimization of the entropy of the quantized representation. Moreover, a bit allocation is performed by learning a normalization for each feature map of $\mathbf{Y}$. The encoder followed by the normalizations at the encoder side, parametrized by $\boldsymbol{\varphi}_e$, are denoted $\overline{g}_e(\,.\,;\boldsymbol{\theta},\boldsymbol{\varphi}_e)$. Similarly,

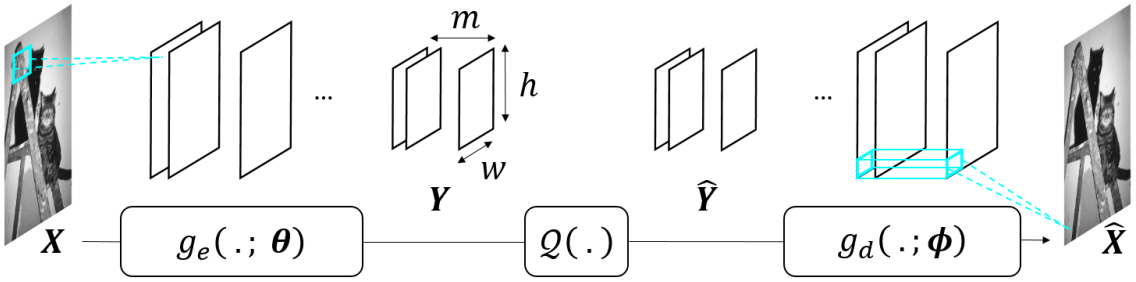


Figure 4.3: Illustration of a baseline autoencoder for image compression.

the normalizations at the decoder side, parametrized by $\boldsymbol{\varphi}_d$, followed by the decoder are denoted $\overline{g}_d\left(\,.\,;\boldsymbol{\varphi}_d,\boldsymbol{\phi}\right)$. Finally, this leads to

$$\min_{\substack{\boldsymbol{\theta},\boldsymbol{\varphi}_e,\\ \boldsymbol{\varphi}_d,\boldsymbol{\phi}}} \mathbb{E}\left[\left\|\mathbf{X}-\overline{g}_d\left(\mathcal{Q}\left(\overline{g}_e\left(\mathbf{X};\boldsymbol{\theta},\boldsymbol{\varphi}_e\right)\right);\boldsymbol{\varphi}_d,\boldsymbol{\phi}\right)\right\|_F^2+\gamma\sum_{i=1}^{m}H_i\right]$$

$$H_i=-\frac{1}{n}\sum_{j=1}^{n}\log_2\left(\hat{p}_i\left(\hat{y}_{ij}\right)\right),\gamma\in\mathbb{R}_+^*. \tag{4.2}$$

$\hat{p}_i$ is the probability mass function of the i^{th} quantized feature map coefficients $\{\hat{y}_{ij}\}_{j=1\ldots n}$. The expectation $\mathbb{E}[.]$ is approximated by averaging over a training set of images. Unfortunately, $\mathcal{Q}$ makes minimization (4.2) unusable. Indeed, the derivative of any quantization with respect to its input is 0 at any point. Consequently, $\boldsymbol{\theta}$ and $\boldsymbol{\varphi}_e$ cannot be learned via gradient-based methods [135]. To get around this issue, [83] fixes the quantization step size to 1 and approximates the uniform scalar quantization with the addition of a uniform noise of support [-0.5, 0.5]. Note that, even though the quantization step size is fixed, the bit allocation varies over the different feature maps via the normalizations. In the next section, we consider instead to remove the normalizations and learn explicitly the quantization step size for each feature map of $\mathbf{Y}$.

4.2.1.2 Learning the quantization step sizes

We address the problem of optimizing the quantization step size for each feature map of $\mathbf{Y}$. Because of the uniform scalar quantizer, the function to be minimized is an implicit function of the quantization step sizes $\{\delta_i\}_{i=1\ldots m}$. The target is to make it an explicit function of $\{\delta_i\}_{i=1\ldots m}$. For $q\in\{\ldots,-\delta_i,0,\delta_i,\ldots\}$,

$$\hat{p}_i\left(q\right)=\int_{q-0.5\delta_i}^{q+0.5\delta_i}p_i\left(t\right)dt=\delta_i\tilde{p}_i\left(q\right). \tag{4.3}$$

$\tilde{p}_i=p_i*l_i$ where p_i is the probability density function of the i^{th} feature map coefficients $\{y_{ij}\}_{j=1\ldots n}$ and l_i denotes the probability density function of the continuous uniform distribution of support $[-0.5\delta_i,0.5\delta_i]$. The normalizations are removed from (4.2) and, using (4.3), (4.2) becomes

$$\min_{\boldsymbol{\theta},\boldsymbol{\phi}} \mathbb{E}\left[\left\|\mathbf{X}-g_d\left(g_e\left(\mathbf{X};\boldsymbol{\theta}\right)+\boldsymbol{\mathcal{E}};\boldsymbol{\phi}\right)\right\|_F^2+\gamma\sum_{i=1}^{m}\tilde{h}_i\right] \tag{4.4}$$

$$\tilde{h}_i = -\log_2(\delta_i) - \frac{1}{n}\sum_{j=1}^{n}\log_2\left(\tilde{p}_i\left(y_{ij}+\varepsilon_{ij}\right)\right).$$

The i^{th} matrix of $\boldsymbol{\mathcal{E}} \in \mathbb{R}^{h\times w\times m}$ contains n realizations $\{\varepsilon_{ij}\}_{j=1\ldots n}$ of $\mathcal{E}_i$, $\mathcal{E}_i$ being a continuous random variable of probability density function l_i. In (4.4), the function to be minimized is differentiable with respect to $\boldsymbol{\theta}$. $\boldsymbol{\theta}$ can thus be learned via gradient-based methods. However, $\{\delta_i\}_{i=1\ldots m}$ cannot yet be learned as the function to be minimized in (4.4) is not differentiable with respect to $\{\delta_i\}_{i=1\ldots m}$. This is resolved using the change of variable $\mathcal{E}_i = \delta_i\mathcal{T}$ where $\mathcal{T}$ is a random variable following the continuous uniform distribution of support $[-0.5, 0.5]$. Now, the minimization over $\{\delta_i\}_{i=1\ldots m}$ is feasible:

$$\min_{\substack{\boldsymbol{\theta},\boldsymbol{\phi},\\ \delta_1,\ldots,\delta_m}} \mathbb{E}\left[\left\|\mathbf{X} - g_d\left(g_e\left(\mathbf{X};\boldsymbol{\theta}\right)+\boldsymbol{\Delta}\odot\mathbf{T};\boldsymbol{\phi}\right)\right\|_F^2 + \gamma\sum_{i=1}^{m}\tilde{h}_i\right]$$

$$\tilde{h}_i = -\log_2(\delta_i) - \frac{1}{n}\sum_{j=1}^{n}\log_2\left(\tilde{p}_i\left(y_{ij}+\delta_i\tau_{ij}\right)\right). \tag{4.5}$$

The i^{th} matrix of $\mathbf{T} \in \mathbb{R}^{h\times w\times m}$ contains n realizations $\{\tau_{ij}\}_{j=1\ldots n}$ of $\mathcal{T}$. All the coefficients in the i^{th} matrix of $\boldsymbol{\Delta} \in \mathbb{R}^{h\times w\times m}$ are equal to δ_i. $\boldsymbol{\Delta}\odot\mathbf{T}$ denotes the elementwise multiplication between $\boldsymbol{\Delta}$ and $\mathbf{T}$. A detail has been left out so far: $\tilde{p}_i$ is unknown. In a similar manner to [82, 83], $\tilde{p}_i$ can be replaced by a function $\tilde{f}_i$, parametrized by $\boldsymbol{\psi}^{(i)}$, and $\boldsymbol{\psi}^{(i)}$ is learned such that $\tilde{f}_i$ fits $\tilde{p}_i$.

In the end, we end up with three groups of parameters: $\{\boldsymbol{\theta}, \boldsymbol{\phi}\}$, $\{\delta_i\}_{i=1\ldots m}$ and $\{\boldsymbol{\psi}^{(i)}\}_{i=1\ldots m}$. These three groups are learned by alternating three different stochastic gradient descents. All the training heuristics are detailed in the code[4].

Section 4.2.1.2 has developped an approach for learning explictly the transform and a quantization step size for each feature map of $\mathbf{Y}$. Before evaluating this approach in Section 4.2.3, Section 4.2.2 studies what would happen if, during the test phase, the coefficients in $\mathbf{Y}$ are quantized using quantization step sizes that differ from those in the training stage. This first requires understanding the internal structure of $\mathbf{Y}$ after the training phase.

4.2.2 Inside the learned representation

This section studies the different feature maps of $\mathbf{Y}$ after the training phase. To this end, a deep convolutional autoencoder must first be built and trained. g_e is the composition of a convolutional layer, a generalized divisive normalization (GDN) [136], a convolutional layer, a GDN, and a convolutional layer. g_d is the reverse composition, replacing each GDN with an inverse generalized divisive normalization (IGDN) [136] and each convolutional layer with a transpose convolutional layer [80]. It is important to stress that $m = 128$, $\mathbf{X}$ has one channel and the convolutional strides and paddings are chosen such that h and w are 16 times smaller than respectively the height and the width of $\mathbf{X}$. Therefore, the number of pixels in $\mathbf{X}$ is twice the number of coefficients in $\mathbf{Y}$. The training set contains 24000 luminance images of size 256×256 that are extracted from ImageNet [129]. The minimization is (4.5), $\gamma = 10000.0$. Note that, if a GDN was placed immediately after g_e, a IGDN was placed immediately before g_d and, $\forall i \in [|1, m|]$, $\delta_i = 1.0$ was not learned, the autoencoder architecture and the training would correspond to [83].

4.2.2.1 Distribution of the learned representation

After the training phase, a test set of 24 luminance images of size 768×512 is created from the Kodak suite [3]. Here, $\mathbf{X}$ refers to a test luminance image. Figure 4.4 shows the normed histograms of 9 feature maps of $\mathbf{Y} = g_e(\mathbf{X}; \boldsymbol{\theta})$, averaged over the test set. Every feature map of $\mathbf{Y}$, except the 90^{th}, has a normed histogram similar to those displayed. To be more precise, let us write the probability density function of the Laplace distribution with mean $\mu \in \mathbb{R}$ and scale $\lambda \in \mathbb{R}_+^*$, denoted $f(\,.\,; \mu, \lambda)$.

$$f(x; \mu, \lambda) = \frac{1}{2\lambda} \exp\left(-\frac{|x-\mu|}{\lambda}\right)$$

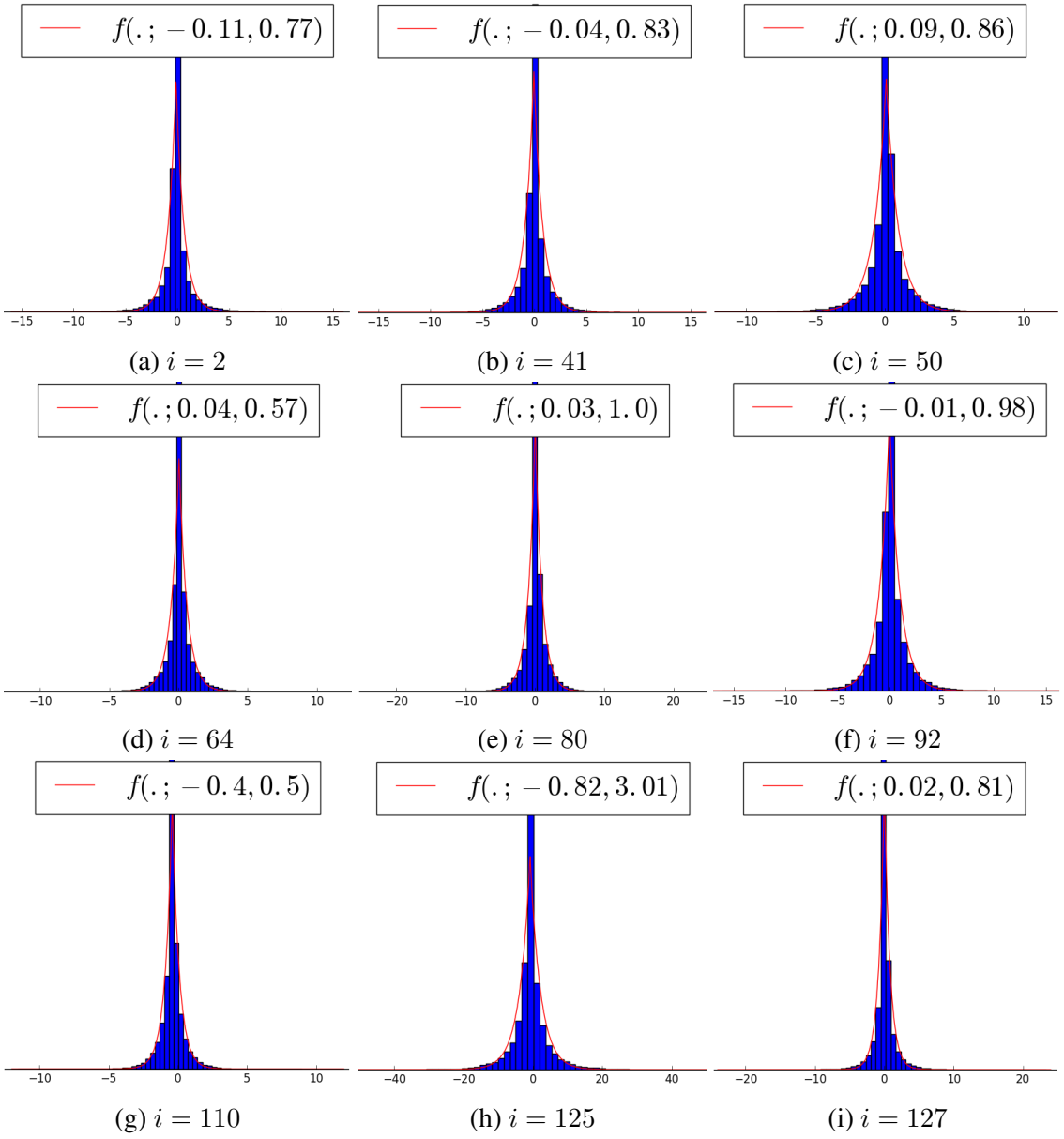
$\forall i \in [|1, m|]$, $i \neq 90$, there exists $\mu_i \in \mathbb{R}$ and $\lambda_i \in \mathbb{R}_+^*$ such that $f(\,.\,; \mu_i, \lambda_i)$ fits well the normed histogram of the i^{th} feature map of $\mathbf{Y}$. Note that most of the $m-1$ scales belong to $[0.5, 2.0]$ (see Figure 4.5). For transformed coefficients having a zero-mean Laplace distribution, [137] proves that a uniform reconstruction quantizer (URQ) with constant decision offsets approaches the optimal scalar quantizer in terms of squared-error distortion for any quantization step size. Yet, in our case, (i) the $m-1$ Laplace probability density functions are not zero-mean, (ii) uniform scalar quantizers are used instead of this URQ. The point (i) is not problematic as an extra set of luminance images is used to compute an approximation $\overline{\mu}_i \in \mathbb{R}$ of the mean of the i^{th} feature map of $\mathbf{Y}$, then, during the test phase, the i^{th} feature map of $\mathbf{Y}$ is centered via $\overline{\mu}_i$ before being quantized. Note that $\{\overline{\mu}_i\}_{i=1\ldots m}$ does not depend on the test luminance images, thus incurring no transmission cost. Regarding the point (ii), it must be noted that the decoder mapping of the URQ is exactly the decoder mapping of the uniform scalar quantization with same quantization step size. Since our case comes close to the requirements of the proof in [137], during the test phase, the rate-distortion trade-off should not collapse as the quantization step sizes deviate from the learned values. This will be verified in Section 4.2.3.

4.2.2.2 Internal structure of the learned representation

The shortcoming of the previous fitting is that it does not reveal what information each matrix of $\mathbf{Y}$ encodes. To discover it, further visualizations are needed. The most common way of exploring a deep convolutional neural network (CNN) trained for image recognition is to look at the image, at the CNN input, resulting from the maximization over its pixels of a given neural activation in the CNN [138, 139, 140]. Precisely, [138, 139] maximize over the image pixels a given neural activation at the CNN output, i.e a class probability. This shows what image features characterize this class according to the CNN. In our case, the maximization over the image pixels of a given coefficient in $\mathbf{Y}$ does not yield interpretable images. Indeed, the coefficients in $\mathbf{Y}$ are not bounded. This may explain why the maximization often returns saturated images.

Alternatively, the information the j^{th} feature map of $\mathbf{Y}$ encodes, $j \in [|1, m|]$, can be seen as follows. $\forall i \in [|1, m|]$, all the coefficients in the i^{th} feature map of $\mathbf{Y}$ are set to $\overline{\mu}_i$. This way, the feature maps of $\mathbf{Y}$ contains no significant information. Then, a single coefficient in the j^{th} feature map of $\mathbf{Y}$ is set to $\alpha \in \mathbb{R}$ and $\hat{\mathbf{X}} = g_d(\mathcal{Q}(\mathbf{Y}); \boldsymbol{\phi})$ is displayed. α is selected such that it is near one of the two tails of the Laplace distribution

3. http://r0k.us/graphics/kodak/



(a) $i = 2$ (b) $i = 41$ (c) $i = 50$

(d) $i = 64$ (e) $i = 80$ (f) $i = 92$

(g) $i = 110$ (h) $i = 125$ (i) $i = 127$

Figure 4.4: Normed histogram of the i^{th} feature map of $\mathbf{Y}$.

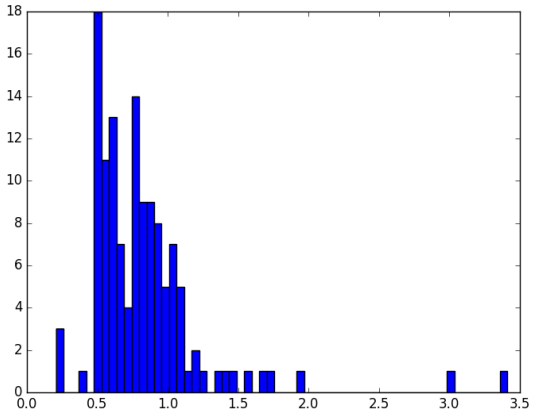
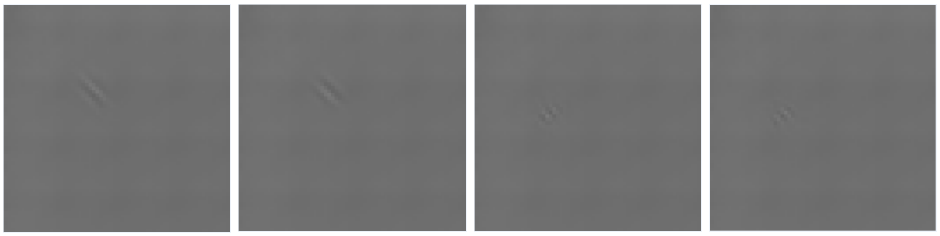
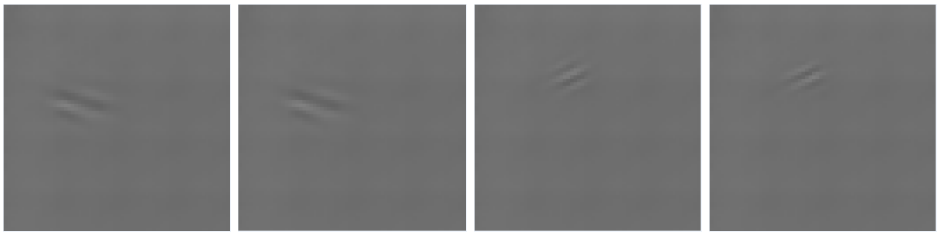


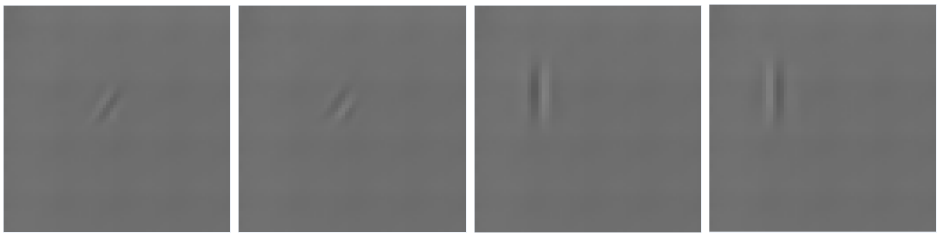
Figure 4.5: Histogram of the $m - 1$ scales provided by the fitting.



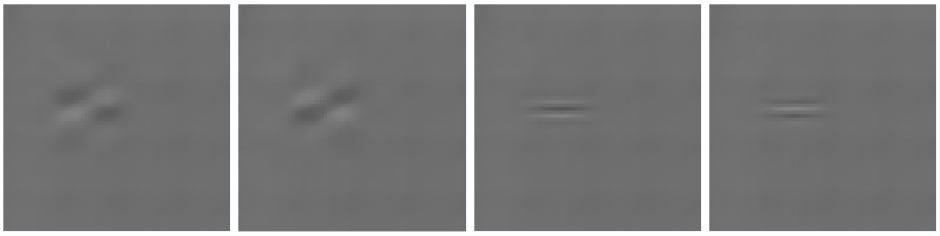
(a) $j = 2$; $\alpha = 8.0$ (b) $j = 2$; $\alpha = -8.0$ (c) $j = 5$; $\alpha = 6.5$ (d) $j = 5$; $\alpha = -6.5$



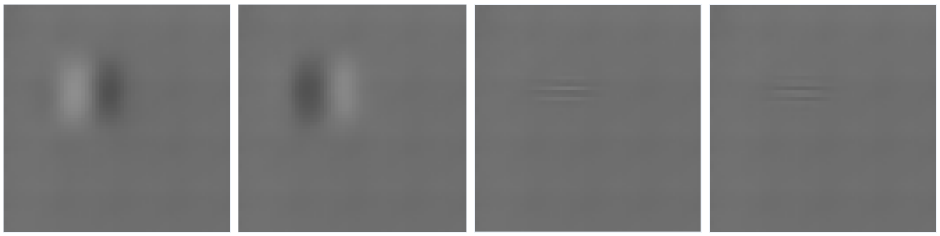
(e) $j = 19$; $\alpha = 10.0$ (f) $j = 19$; $\alpha = -10.0$ (g) $j = 41$; $\alpha = 8.5$ (h) $j = 41$; $\alpha = -8.5$



(i) $j = 50$; $\alpha = 8.0$ (j) $j = 50$; $\alpha = -8.0$ (k) $j = 82$; $\alpha = 10.0$ (l) $j = 82$; $\alpha = -10.0$

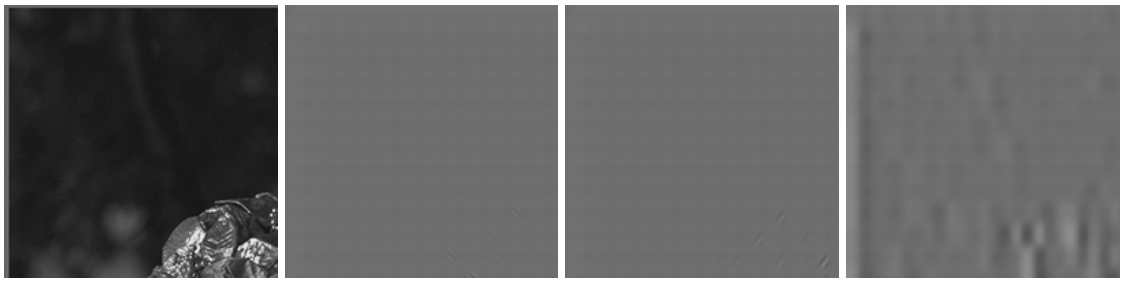


(m) $j = 98$; $\alpha = 11.0$ (n) $j = 98$; $\alpha = -11.0$ (o) $j = 107$; $\alpha = 10.0$ (p) $j = 107$; $\alpha = -10.0$



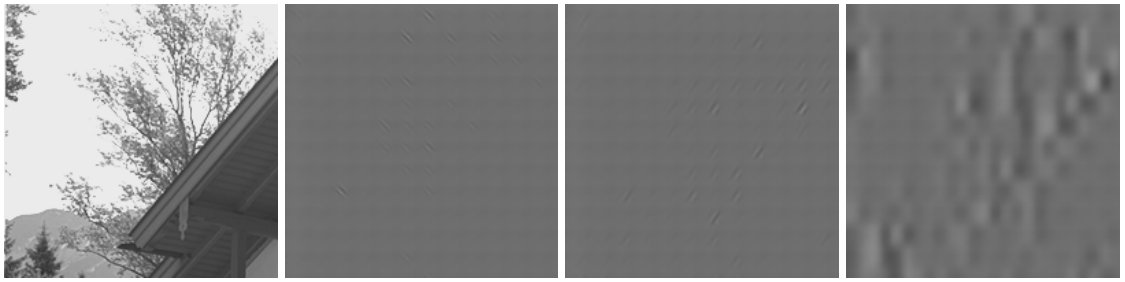
(q) $j = 125$; $\alpha = 20.0$ (r) $j = 125$; $\alpha = -20.0$ (s) $j = 127$; $\alpha = 9.0$ (t) $j = 127$; $\alpha = -9.0$

Figure 4.6: 64×64 crop at the top-left of $\hat{\mathbf{X}}$.



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

Figure 4.7: Reconstructed luminance image when the 17^{th} Kodak luminance image is inserted into the encoder $g_e(\,.\,;\boldsymbol{\theta})$ and all the feature maps of $\mathbf{Y}$, except the j^{th} feature map, are masked. (a) 200×200 crop at the top-left of the 17^{th} Kodak luminance image, (b) 200×200 crop at the top-left of $\hat{\mathbf{X}}$ when $j = 2$, (c) 200×200 crop at the top-left of $\hat{\mathbf{X}}$ when $j = 50$, and (d) 200×200 crop at the top-left of $\hat{\mathbf{X}}$ when $j = 125$.



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

Figure 4.8: Same visualization as in Figure 4.7 using the 24^{th} Kodak luminance image instead of the 17^{th} Kodak luminance image.

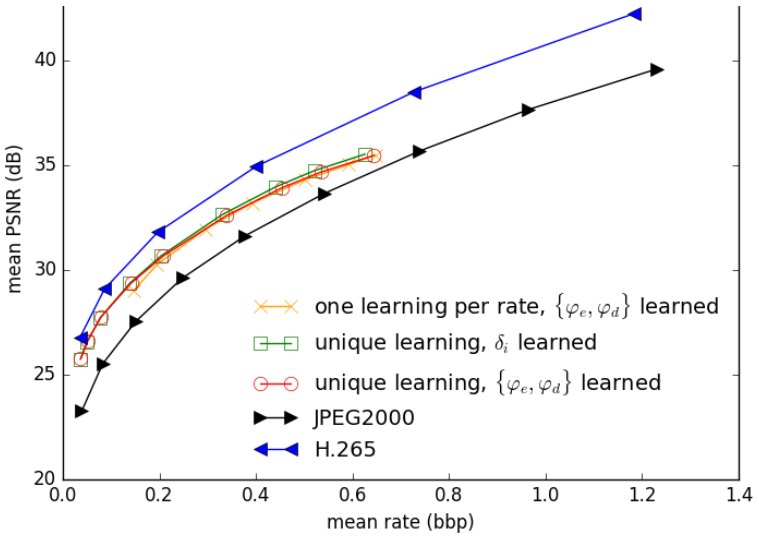


Figure 4.9: Rate-distortion curves averaged over the 24 Kodak luminance images.

of the j^{th} feature map of $\mathbf{Y}$. Figure 4.6 shows the 64×64 crop at the top-left of $\hat{\mathbf{X}}$ when the single coefficient is located at the top-left corner of the j^{th} feature map of $\mathbf{Y}$, $j \in \{2, 5, 19, 41, 50, 82, 98, 107, 125, 127\}$. We notice that the 2^{nd}, 41^{st} and 50^{th} feature maps of $\mathbf{Y}$ each encodes a spatially localized image feature whereas the 98^{th} and 125^{th} feature maps each encodes a spatially extended image feature. Moreover, the image feature is turned into its symmetrical feature, with respect to the mean pixel intensity, by moving α from the right tail of the Laplace distribution of the j^{th} feature map of $\mathbf{Y}$ to the left tail. This linear behaviour is observed for each feature map of $\mathbf{Y}$.

It is worth remarking that, given the fitting in Section 4.2.2.1, $\mathbf{Y}$ is similar to the DCT coefficients for blocks of prediction error samples in H.265 [6] in terms of distribution. However, when looking at the information each feature map of $\mathbf{Y}$ encodes, $\mathbf{Y}$ has nothing to do with these DCT coefficients.

To validate this conclusion, another visualization of the information the j^{th} feature map of $\mathbf{Y}$ encodes can be used. The difference between the previous visualization and the following one lies in the fact that an activation in a feature map of $\mathbf{Y}$ is artificially triggered in the previous visualization whereas a feature map of $\mathbf{Y}$ is activated by inserted a luminance image into the encoder in the following visualization. Precisely, after inserting a Kodak luminance image into the encoder $g_e(\,.\,;\boldsymbol{\theta})$, $\forall i \in [|1, m|], i \neq j$, all the coefficients in the i^{th} feature map of $\mathbf{Y}$ are set to $\overline{\mu}_i$. Now, only the j^{th} feature map of $\mathbf{Y}$ keeps relevant information about the luminance image. Then, $\hat{\mathbf{X}} = g_d(\mathcal{Q}(\mathbf{Y}); \boldsymbol{\phi})$ is displayed. Again, the 2^{nd} and 50^{th} feature maps of $\mathbf{Y}$ each encodes a spatially localized image feature, unlike the 125^{th} feature map (see Figures 4.7 and 4.8).

4.2.3 Experiments

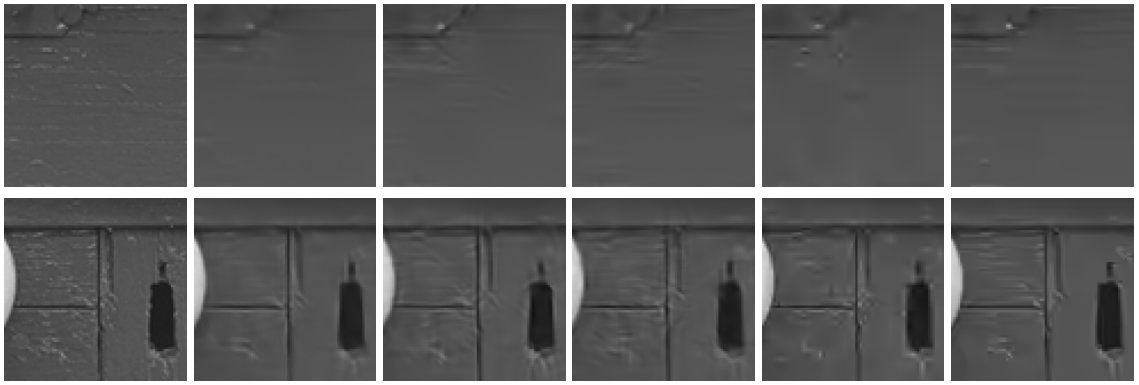
We now evaluate in terms of rate-distortion performances: (i) whether the way of learning the quantization matters, (ii) whether, during the test phase, it is efficient to quantize the coefficients obtained with the learned transform using quantization step sizes which differ from those in the training stage. This is done by comparing three cases.

The first case follows the approach in [83]. One transform is learned per rate-distortion point, the bit allocation being learned via the normalizations. In details, an autoencoder is trained for each $\gamma \in S = \{10000.0, 12000.0, 16000.0, 24000.0, 40000.0, 72000.0, 96000.0\}$. During the training and test phases, the quantization step size is fixed to 1.0.

In the second case, a unique transform is learned, the bit allocation being done by learning a quantization step size per feature map. More precisely, a single autoencoder is trained for $\gamma = 10000.0$ and $\{\delta_i\}_{i=1\ldots m}$ is learned (see Section 4.2.1). During the test phase, the rate varies as the quantization step sizes are equal to the learned quantization step sizes multiplied by $\beta \in \mathcal{B} = \{1.0, 1.25, 1.5, 2.0, 3.0, 4.0, 6.0, 8.0, 10.0\}$.

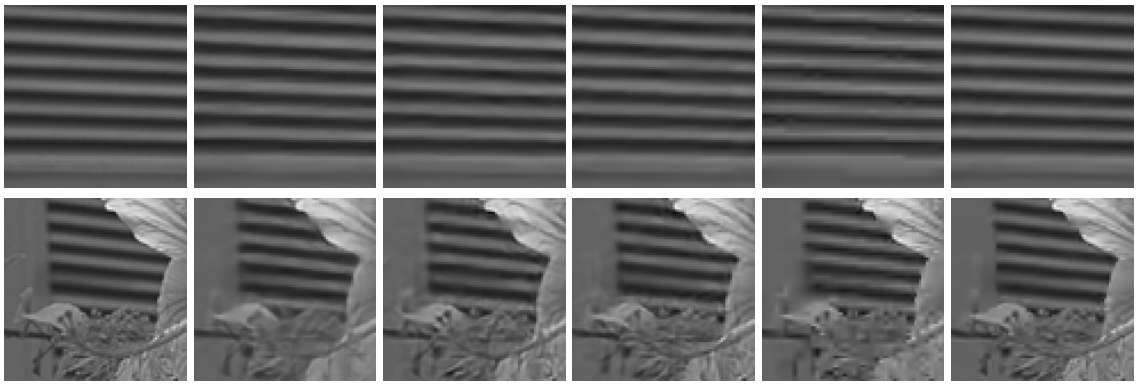
In the third case, a unique transform is learned, the bit allocation being learned via the normalizations. In details, a single autoencoder is trained for $\gamma = 10000.0$ and, during the training phase, the quantization step size is 1.0. During the test phase, the rate varies as the quantization step size spans $\mathcal{B}$.

In the second case, the autoencoder has the architecture described at the beginning of Section 4.2.2. In the first and third cases, a GDN is also placed after g_e and a IGDN is placed before g_d. The autoencoders are trained on 24000 luminance images of size 256×256 that are extracted from ImageNet. Then, during the test phase, the 24 lumi-



(a) original crops	(b) orange	(c) green	(d) red	(e) JPEG2000	(f) H.265
	$(0.238, 34.815)$	$(0.191, 34.446)$	$(0.198, 34.421)$	$(0.191, 33.799)$	$(0.183, 35.458)$

Figure 4.10: Reconstruction of two 80×80 crops of the 2^{nd} Kodak luminance image via five compression algorithms. To associate (b) orange, (c) green, and (d) red to their respective algorithm, please refer to the color code in Figure 4.9. Below each letter (b), (c), (d), (e), and (f) is written inside brackets the rate (bbp) and the PSNR (dB) when the corresponding algorithm is applied to the **entire** 2^{nd} Kodak luminance image.



(a) original crops	(b) orange	(c) green	(d) red	(e) JPEG2000	(f) H.265
	$(0.242, 34.211)$	$(0.258, 34.830)$	$(0.266, 34.797)$	$(0.253, 33.391)$	$(0.260, 37.003)$

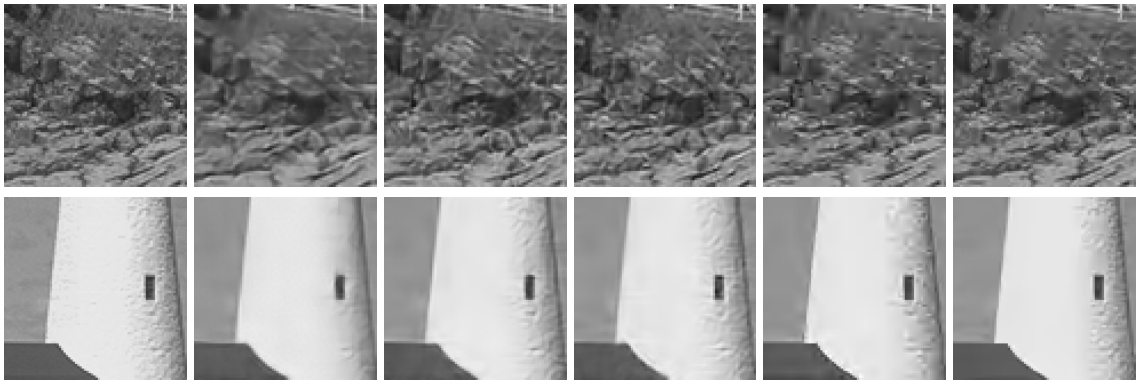
Figure 4.11: Same visualization as in Figure 4.10 using the 7^{th} Kodak luminance image.

(a) original crops	(b) orange	(c) green	(d) red	(e) JPEG2000	(f) H.265
	$(0.179, 34.123)$	$(0.120, 33.123)$	$(0.124, 33.193)$	$(0.153, 33.641)$	$(0.178, 36.280)$

Figure 4.12: Same visualization as in Figure 4.10 using the 12^{th} Kodak luminance image.

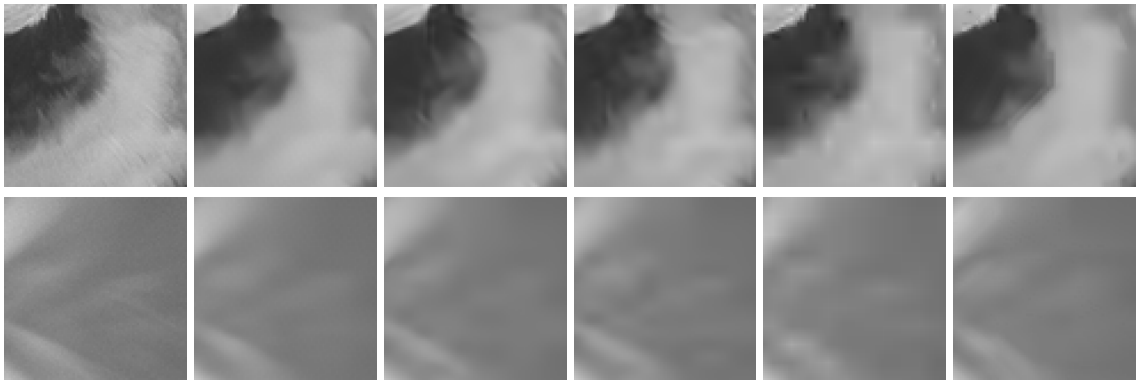
(a) original crops | (b) orange $(0.320, 33.822)$ | (c) green $(0.263, 33.295)$ | (d) red $(0.272, 33.336)$ | (e) JPEG2000 $(0.322, 33.685)$ | (f) H.265 $(0.290, 35.062)$

Figure 4.13: Same visualization as in Figure 4.10 using the 16^{th} Kodak luminance image. Below each letter (b), (c), (d), (e), and (f) is written inside brackets the rate and the PSNR when the corresponding algorithm is applied to the **entire** 16^{th} Kodak luminance image.



(a) original crops | (b) orange $(0.344, 31.031)$ | (c) green $(0.378, 31.543)$ | (d) red $(0.387, 31.514)$ | (e) JPEG2000 $(0.393, 31.639)$ | (f) H.265 $(0.469, 34.570)$

Figure 4.14: Same visualization as in Figure 4.10 using the 21^{st} Kodak luminance image.



(a) original crop | (b) orange $(0.145, 36.144)$ | (c) green $(0.103, 34.770)$ | (d) red $(0.105, 34.750)$ | (e) JPEG2000 $(0.134, 35.512)$ | (f) H.265 $(0.081, 35.597)$

Figure 4.15: Same visualization as in Figure 4.10 using the 23^{rd} Kodak luminance image.

nance images from the Kodak suite are inserted into the autoencoders. The rate is estimated via the empirical entropy of the quantized coefficients, assuming that the quantized coefficients are i.i.d. Note that, for the second and third cases, we have also implemented a binarizer and a binary arithmetic coder to compress the quantized coefficients losslessly (see the code[4]). The difference between the estimated rate and the exact rate via the lossless coding is always smaller than 0.04 bbp.

4.2.3.1 Rate-distortion analysis

Figure 4.9 shows the rate-distortion curves averaged over the 24 luminance images. The JPEG2000 curve is obtained using ImageMagick. The H.265 [39] curve is computed via the version HM-16.15. There is hardly any difference between the second case and the third case. This means that the explicit learning of the transform and the quantization step sizes is equivalent to learning the transform and the normalizations while the quantization step size is imposed. The second and the third cases perform as well as the first case. The minimization in (4.5) and the training in [83] provide learned transforms which can be used with various quantization step sizes during the test phase. It is convenient not to train one autoencoder per compression rate as a single training takes 4 days on a NVIDIA GTX 1080. Finally, we see that the learned transforms yield better rate-distortion performances than JPEG2000. The quality of image reconstruction for the experiment in Figure 4.9 can be checked in Figures 4.10, 4.11, 4.12, 4.13, 4.14, and 4.15. Much more visualizations for the experiment in Figure 4.9 and another experiment on luminance images created from the BSDS300 [141] are displayed online [4]. Besides, the code to reproduce all the numerical results in Section 4.2 and train the autoencoders is available online[4].

4.2.3.2 Number of trainable parameters of each autoencoder

In the first and third cases, which are the orange curve and the red curve respectively in Figure 4.9, the detailed architecture of each autoencoder corresponds exactly to the architecture in [83] in order to have a fair comparison. Please, refer to [83] to know all the details about the autoencoder architecture. The number of trainable parameters is 1758848.

The second case is the green curve in Figure 4.9. In this case, the autoencoder does not contain the normalization parameters $\{\varphi_e, \varphi_d\}$, which removes 33024 parameters, and the quantization step sizes $\{\delta_i\}_{i=1\ldots m}$ are learned, which adds 128 parameters. The number of trainable parameters is thus 1725952.

4.3 Conclusion

To conclude, two approaches are proposed for learning a transform for image compression that achieves rate-distortion efficiency and is *rate-agnostic*.

In the first approach, a deep sparse autoencoder is trained using a constraint on the number of non-zero coefficients in the sparse representation of an entire image. The sparsity parameter controlling this constraint is stochastically driven during the training

4. www.irisa.fr/temics/demos/visualization_ae/visualizationAE.htm

phase so that the deep sparse autoencoder adapts to compressing at multiple rates. During the test phase, the sparsity parameter provides a direct control over the rate.

In the second approach, a deep autoencoder is trained via the minimization of an objective function combining a distortion term and a term approximating the rate via the entropy of the quantized representation. It is shown that the uniform scalar quantizer can be fixed during the training phase, or in other words, there is no apparent gain resulting from the joint learning of the transform and the uniform scalar quantizer. During the test phase, the quantization step sizes are increased from their value at the end of the training phase in order to compress at multiple rates.

Chapter 5

Learning intra prediction for image compression

Besides the transform, which is learned for image compression in Chapter 4, a key of efficient image compression schemes is the predictor (see Section 1.2).

In the regular intra prediction for image compression, a set of prediction functions is first predefined. It is known by both the encoder and the decoder. Then, given an image block to be predicted, the encoder identifies the best prediction function according to a rate-distortion criterion and the index of the best prediction function is sent to the decoder. As the number of prediction functions is restricted and the functions are linear, a regular intra predictor struggles to predict complex textures in large image blocks (see Section 1.2). This brings about the idea of learning, with the help of neural networks, a reliable model of dependencies between a block to be predicted, possibly containing a complex texture, and its neighborhood that we refer to as its context.

Neural networks have already been considered in [102] for intra block prediction. However, the authors in [102] only take into consideration blocks of sizes 4×4, 8×8, 16×16, and 32×32 pixels and use fully-connected neural networks. Here, we consider both fully-connected and convolutional neural networks. We show that, while fully-connected neural networks give good performance for small block sizes, convolutional neural networks are more appropriate, both in terms of prediction PSNR and PSNR-rate performance gains, for large block sizes. The choice of neural network is block size dependent, hence does not need to be signalled to the decoder. This set of neural networks, called Prediction Neural Networks Set (PNNS), has been integrated into a H.265 codec, showing PSNR-rate performance gains from 1.46% to 5.20%.

The contributions of this chapter are as follows:

- A set of neural network architectures is proposed, including both fully-connected and convolutional neural networks, for intra image prediction.
- It is shown that, in the case of large block sizes, convolutional neural networks yield more accurate predictions compared with fully-connected ones.
- Thanks to the use of masks of random sizes during the training phase, the neural networks of PNNS well adapt to the available context that may vary. E.g. in H.265, the available context, hence the number of known pixels in the neighborhood, depends on the position of the considered PB within the CB and within the CTB.
- Unlike the H.265 intra prediction modes, which are each specialized in predicting a

specific texture, the proposed PNNS, trained on a large unconstrained set of images, is able to model a large set of complex textures.
- A surprising property of the neural networks for intra prediction is proved experimentally: they do not need to be trained on distorted contexts, i.e. the neural networks trained on undistorted contexts generalize well on distorted contexts, even for severe distortions.

The code to reproduce all the numerical results and train the neural networks of PNNS will be made available online [1] after the PhD defense.

5.1 Conditions for efficient neural network based intra prediction

The use of neural networks for intra prediction within an image compression scheme raises several questions that we address in this paper. What neural network architecture provides enough power of representation to map causal and distorted data samples to an accurate prediction of a given image block? What context size should be used? Section 5.2 looks for a neural network architecture and the optimal number of causal and distorted data samples for predicting a given image block. Moreover, the amount of causal and distorted data samples available at the decoder varies. It depends on the partitioning of the image and the position of the block to be predicted within the image. Section 5.3 trains the neural networks so that they adapt to the variable context size. Finally, can neural networks compensate for the quantization noise in its input and be efficient in a rate-distortion sense? Sections 5.4 and 5.5 answer these two questions with experimental evidence.

5.2 Proposed neural network based intra prediction

Unlike standard intra prediction in which the encoder chooses the best mode in a rate-distortion sense among several pre-defined modes, only one neural network among a set of neural networks does the prediction here. Unlike [102], our set contains both fully-connected and convolutional neural networks. This section first presents our set of neural networks. Then, it explains how one neural network is selected for predicting a given image block and the context is defined according to the block size. Finally, the specificities of the integration of our neural networks into H.265 are detailed.

5.2.1 Fully-connected and convolutional neural networks

Let $\mathbf{X}$ be a context containing decoded pixels above and on the left side of a square image block $\mathbf{Y}$ of width $m \in \mathbb{N}^*$ to be predicted (see Figure 5.1). The transformation of $\mathbf{X}$ into a prediction $\hat{\mathbf{Y}}$ of $\mathbf{Y}$ via either a fully-connected neural network f_m, parametrized by $\boldsymbol{\theta}_m$, or a convolutional neural network g_m, parametrized by $\boldsymbol{\phi}_m$, is described in (5.1).

1. https://www.irisa.fr/temics/demos/prediction_neural_network/PredictionNeuralNetwork.htm

The corresponding architectures are depicted in Figures 5.2 and 5.3.

$$\begin{aligned} \mathbf{X}_c &= \mathbf{X} - \alpha \\ \hat{\mathbf{Y}}_c &= \begin{cases} f_m\left(\mathbf{X}_c; \boldsymbol{\theta}_m\right) \\ g_m\left(\mathbf{X}_c; \boldsymbol{\phi}_m\right) \end{cases} \\ \hat{\mathbf{Y}} &= \max\left(\min\left(\hat{\mathbf{Y}}_c + \alpha, 255\right), 0\right) \end{aligned} \tag{5.1}$$

During optimization, each input variable to a neural network must be approximatively zero-centered over the training set to accelerate convergence [120]. Besides, since the pixel space corresponds to both the input and output spaces in intra prediction, it makes sense to normalize the input and output similarly. One could subtract from each image block to be predicted and its context their respective mean during the training phase. But, this entails sending the mean of a block to be predicted to the decoder during the test phase. Instead, the mean pixel intensity α is first computed over all the training images. Then, α is subtracted from each image block to be predicted and its context during training. During the test phase, α is subtracted from the context (see (5.1) where the subscript c stands for centered).

This preprocessing implies a postprocessing of the neural network output. More precisely, the learned mean pixels intensity is added to the output and the result is clipped to $[0, 255]$ (see (5.1)).

The first operation for both architectures consists in formatting the context to ease the computations in the neural network. In the case of a fully-connected neural network, all elements in the context are connected such that there is no need to keep the 2D structure of the context [63]. Therefore, the context is first vectorized, and fast vector-matrix arithmetics can be used (see Figure 5.2). However, in the case of a convolutional neural network, fast computation of 2D filtering requires to keep the 2D structure of the context. Moreover, again for fast computation, the shape of the input to the convolution has to be rectangular. That is why the context is split into two rectangles $\mathbf{X}_0$ and $\mathbf{X}_1$ (see Figures 5.1 and 5.3). $\mathbf{X}_0$ and $\mathbf{X}_1$ are then processed by distinct convolutions.

The proposed fully-connected architecture is composed of 4 fully-connected layers. The first layer computes an overcomplete representation of the context to reach $p \in \mathbb{N}^*$ output coefficients. Overcompleteness is chosen as it is observed empirically that overcomplete representations in early layers boost the performance of neural networks [142, 134, 61]. The next two layers keep the number of coefficients unchanged, while the last layer reduces it to provide the predicted image block. The first three layers have LeakyReLU [143] with slope 0.1 as non-linear activation. The last layer has no non-linear activation. This is because the postprocessing discussed earlier contains already a non-linearity, which consists in first adding the learned mean pixels intensity to the output and clipping the result to $[0, 255]$.

The first task of the convolutional architecture is the computation of features characterizing the dependencies between the elements in $\mathbf{X}_0$. $\mathbf{X}_0$ is thus fed into a stack of convolutional layers. This yields a stack $\mathbf{Z}_0$ of $l \in \mathbb{N}^*$ feature maps (see Figure 5.3). Similarly, $\mathbf{X}_1$ is fed into another stack of convolutional layers. This yields a stack $\mathbf{Z}_1$ of l feature maps.

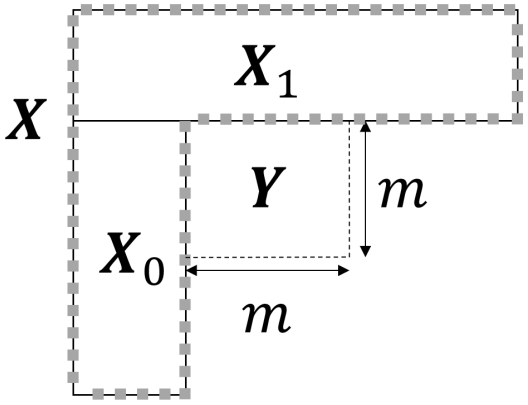


Figure 5.1: Illustration of the relative positions of $\mathbf{X}$, $\mathbf{X}_0$, $\mathbf{X}_1$, and $\mathbf{Y}$.

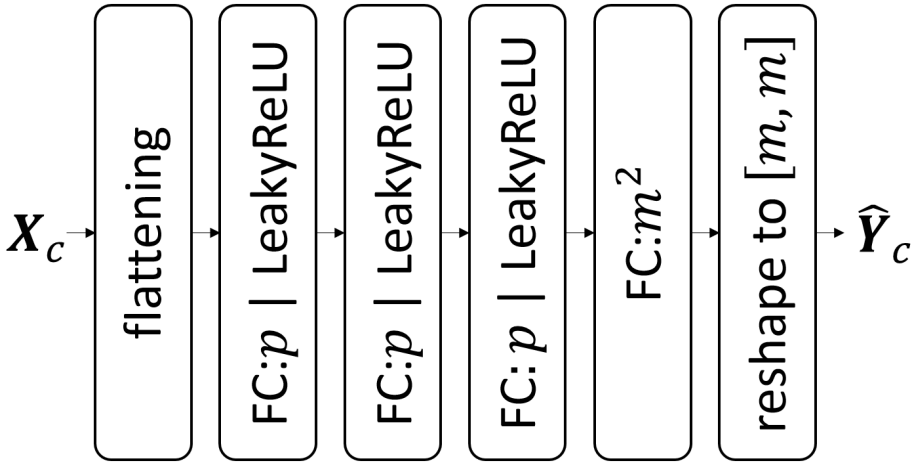


Figure 5.2: Illustration of the fully-connected architecture f_m. FC:p|LeakyReLU denotes the fully-connected layer with p output neurons and LeakyReLU with slope 0.1 as non-linear activation. Unlike FC:p|LeakyReLU, FC:p has no non-linear activation. $\boldsymbol{\theta}_m$ gathers the weights and biases of the 4 fully-connected layers of f_m.

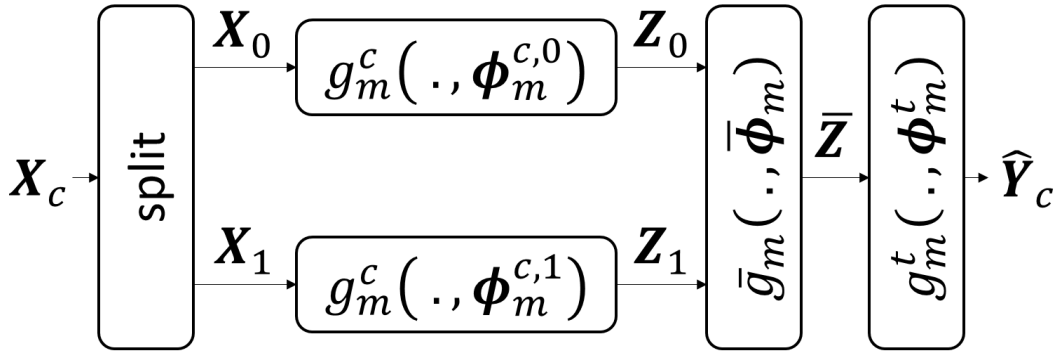


Figure 5.3: Illustration of the convolutional architecture g_m in which g_m^c, $\overline{g}_m$, and g_m^t denote respectively a stack of convolutional layers, the merger, and the stack of transpose convolutional layers. $\boldsymbol{\phi}_m^c$, $\overline{\boldsymbol{\phi}}_m$, and $\boldsymbol{\phi}_m^t$ gather the weights and biases of respectively g_m^c, $\overline{g}_m$, and g_m^t. $\boldsymbol{\phi}_m = \left\{\boldsymbol{\phi}_m^{c,0}, \boldsymbol{\phi}_m^{c,1}, \overline{\boldsymbol{\phi}}_m, \boldsymbol{\phi}_m^t\right\}$.

All the elements in the context can be relevant for predicting any image block pixel. This implies that the information associated to all spatial locations in the context has to be merged. Unfortunately, convolutions only account for short-range spatial dependencies. That is why the next layer in the convolutional architecture merges spatially $\mathbf{Z}_0$ and $\mathbf{Z}_1$ (see Figure 5.3). More precisely, for $i \in [|1, l|]$, all the coefficients of the i^{th} feature map of $\mathbf{Z}_0$ and of the i^{th} feature map of $\mathbf{Z}_1$ are merged through affine combinations. Then, LeakyReLU with slope 0.1 is applied, yielding the merged stack $\overline{\mathbf{Z}}$ of feature maps. Note that this layer bears similarities with the *channelwise fully-connected layer* [144]. But, unlike the *channelwise fully-connected layer*, it merges two stacks of feature maps of different height and width. Its advantage over a fully-connected layer is that it contains l times less weights.

The last task of the convolutional architecture is to merge the information of the different feature maps of $\overline{\mathbf{Z}}$. $\overline{\mathbf{Z}}$ is thus fed into a stack of transpose convolutional layers [125, 80]. This yields the predicted image block (see Figure 5.3). Note that all convolutional layers and transpose convolutional layers, apart from the last transpose convolutional layer, have LeakyReLU with slope 0.1 as non-linear activation. The last transpose convolutional layer has no non-linear activation due to the postprocessing discussed earlier.

5.2.2 Growth rate of the context size with the block size

Now that the architectures and the shape of the context are defined, the size of the context remains to be optimized. The causal neighborhood of the image block to be predicted used by the H.265 intra prediction modes is limited to one row of $2m + 1$ decoded pixels above the block and one column of $2m$ decoded pixels on the left side of the block. However, a context of such a small size is not sufficient for neural networks as a neural network relies on the spatial distribution of the decoded pixels intensity in the context to predict complex textures. Therefore, the context size has to be larger than $4m + 1$.

But, an issue arises when the size of the context grows too much. Indeed, if the image block to be predicted is close to either the top edge of the decoded image $\mathbf{D}$ or its left edge, a large context goes out of the bounds of the decoded image. The neural network prediction is impossible. There is thus a tradeoff to find a suitable size for the context.

Let us look at decoded pixels above and on the left side of $\mathbf{Y}$ to develop intuitions regarding this tradeoff. When m is small, the long range spatial dependencies between these decoded pixels are not relevant for predicting $\mathbf{Y}$ (see Figure 5.4). In this case, the size of the context should be small so that the above-mentioned issue is limited. However, when m is large, such long range spatial dependencies are informative for predicting $\mathbf{Y}$. The size of the context should now be large, despite the issue. Therefore, the context size should be a function of $m^q, q \geq 1$.

From there, we ran several preliminary experiments in which $q \in \{1, 2, 3\}$ and the PSNR between $\hat{\mathbf{Y}}$ and $\mathbf{Y}$ was measured. The conclusion is that a neural network yields the best PSNRs when the size of the context grows with m^2, i.e. the ratio between the size of the context and the size of the image block to be predicted is constant. This makes sense as, in the most common regression tasks involving neural networks, such as super-resolution [145, 78], segmentation [128, 146] or video interpolation [147, 148, 149], the

ratio between the input image dimension and the output image dimension also remains constant while the height and width of the output image increase. Given the previous conclusions, $\mathbf{X}_0$ is a rectangle of height $2m$ and width m. $\mathbf{X}_1$ is a rectangle of height m and width $3m$.

5.2.3 Criterion for selecting a proposed neural network

Now that the context is defined with respect to m, all that remains is to choose between a fully-connected neural network and a convolutional one according to m.

Convolutional neural networks allow better extracting 2D local image structures with fewer parameters compared to fully-connected neural networks [63]. This is verified by the better prediction performances of the convolutional neural networks for $m > 8$ (see Sections 5.3.5 and 5.3.6). However, this property becomes less critical for small block sizes. We indeed experimentally show that, when $m \leq 8$, for the prediction problem, fully-connected architectures outperform convolutional architectures.

Therefore, when integrating the neural network based intra predictor into a codec like H.265, the criterion is to predict a block of width $m > 8$ via a proposed convolutional neural network, and a block of width $m \leq 8$ via a fully-connected neural network.

5.2.4 Integration of the neural network based intra prediction into H.265

A specificity of H.265 is the quadtree structure partitioning, which determines the range of values for m and the number of available decoded pixels in the context.

Due to the partioning in H.265 of an image channel into Coding Tree Blocks (CTBs), and the partioning of each CTB into Coding Blocks (CBs), $m \in \{4, 8, 16, 32, 64\}$. For each $m \in \{4, 8\}$, a fully-connected neural network is constructed with internal size $p = 1200$. Similarly, one convolutional neural network is constructed per block width $m \in \{16, 32, 64\}$. The convolutional architecture for each $m \in \{16, 32, 64\}$ is detailed in Appendix C.1.

Another consequence of this partitioning is that the number of available decoded pixels in the context depends on m and the position of image block to be predicted in the current CTB. For instance, if the block is located at the bottom of the current CTB, the bottommost m^2 pixels in the context are not decoded yet. More generally, it might happen that a group of $m \times n_0$ pixels, $n_0 \in \{0, 4, ..., m\}$, located at the bottom of the context, is not decoded yet. Similarly, a group of $n_1 \times m$ pixels, $n_1 \in \{0, 4, ..., m\}$, located furthest to the right in the context, may not have been decoded yet (see Figure 5.5). When pixels are not decoded yet, the solution in H.265 is to copy a decoded pixel into its neighboring undecoded pixels. But, this copy process cannot be re-used here. Indeed, it would fool the neural network and make it believe that, in an undecoded group of pixels and its surroundings, the spatial distribution of pixels intensity follows the regularity induced by the copy process.

Alternatively, it is possible to indicate to a neural network that the two undecoded groups are unknown by masking them. The mask is set to the learned mean pixels intensity over the training set so that, after subtracting it from the context, the average of each input variable to a neural network over the training set is still near 0. More precisely regarding

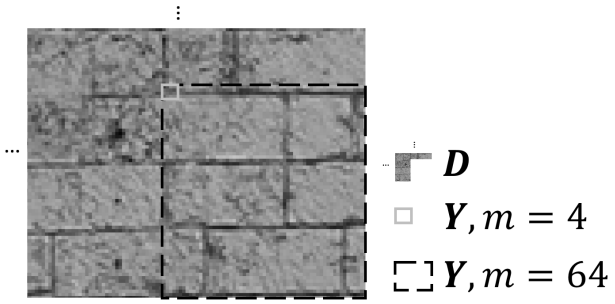


Figure 5.4: Dependencies between $\mathbf{D}$ and $\mathbf{Y}$. The luminance channel of the first image in the Kodak suite [150] is being encoded via H.265 with Quantization Parameter QP $= 17$.

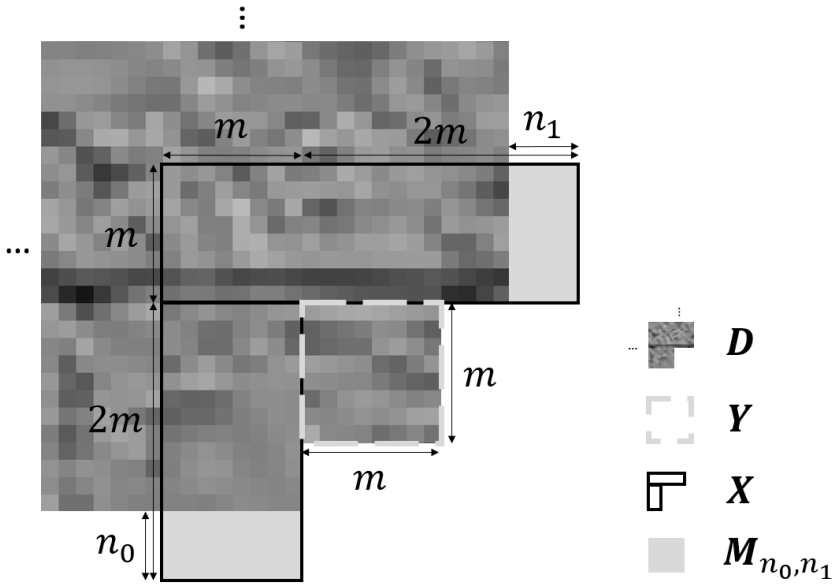


Figure 5.5: Masking of the undecoded pixels in $\mathbf{X}$ for H.265. The luminance channel of the first image in the Kodak suite is being encoded via H.265 with QP $= 37$. Here, $m = 8$ and $n_0 = n_1 = 4$.

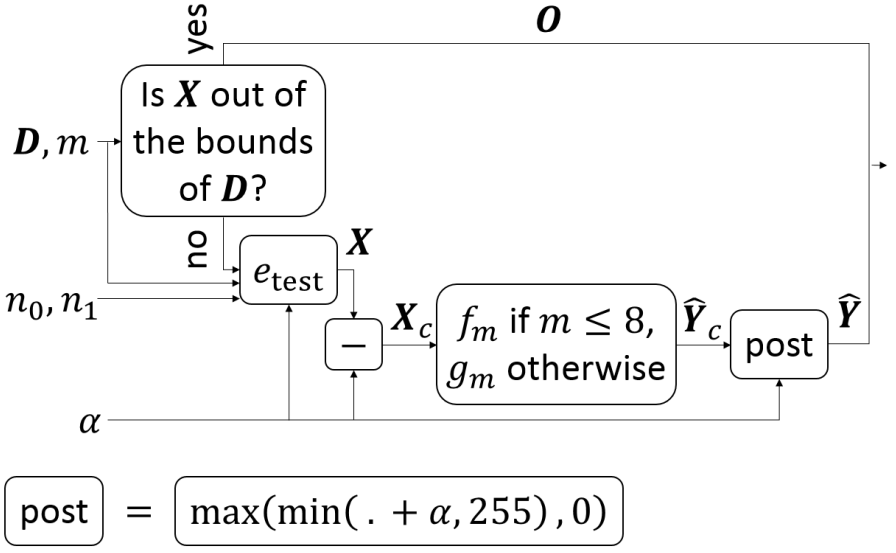


Figure 5.6: Neural network based intra prediction scheme inside H.265.

the masking, the first undecoded group in the context is fully covered by an α-mask of height n_0 and width m. The second undecoded group in the context is fully covered by an α-mask of height m and width n_1 (see Figure 5.5). The two α-masks together are denoted $\mathbf{M}_{n_0,n_1}$. In Section 5.3.1, the neural networks will be trained so that they adapt to this variable number of available decoded pixels in the context.

Figure 5.6 summarizes the integration of the neural network based intra prediction scheme into H.265. The last issue to address is when no context is available. This occurs when the context goes out of the bounds of the decoded image, i.e. the pixel at the top-left of the context is not inside the decoded image. In this case, no neural network is used, and a zero prediction $\mathbf{O}$ of $\mathbf{Y}$ is returned. In Figure 5.6, e_{test} extracts $\mathbf{X}$ from $\mathbf{D}$ with respect to the position of $\mathbf{Y}$ while applying $\mathbf{M}_{n_0,n_1}$ to $\mathbf{X}$, following Figure 5.5.

5.3 Neural networks training

This section explains how our neural networks are trained. Notably, an adaptation to the changing number of available decoded pixels in the input context is proposed. Then, an experiment shows the effectiveness of this approach. Moreover, this experiment compares the predictions of convolutional neural networks and those of fully-connected neural networks in terms of prediction PSNR in the case of large image blocks to be predicted.

5.3.1 Adaptation of the neural networks to the variable number of available decoded pixels via random context masking

The fact that n_0 and n_1 vary during the test phase, e.g. in H.265, has to be considered during the training phase. It would be unpractical to train one set of neural networks for each possible pair $\{n_0, n_1\}$. Instead, we propose to train the neural networks while feeding them with contexts containing a variable number of known pixels. More precisely, during the training phase, n_0 and n_1 are sampled uniformly from the set $\{0, 4, ..., m\}$. This way, the amount of available information in a training context is viewed as a random process the neural networks have to cope with.

5.3.2 Objective function to be minimized

The goal of the prediction is to minimize the Euclidean distance between the image block to be predicted and its estimate, or in other words to minimize the variance of the difference between the block and its prediction [10], also called the residue. The choice of the l_2 norm is a consequence of the l_2 norm chosen to measure the distortion between an image and its reconstruction. So, this Euclidean distance is minimized to learn the parameters $\boldsymbol{\theta}_m$. Moreover, regularization by l_2 norm of the weights (not the biases), denoted $[\boldsymbol{\theta}_m]_W$, is applied:

$$\begin{aligned} &\min_{\boldsymbol{\theta}_m} \mathbb{E}\left[\left\|\mathbf{Y}_c - f_m\left(\mathbf{X}_c; \boldsymbol{\theta}_m\right)\right\|_2\right] + \lambda \left\|[\boldsymbol{\theta}_m]_W\right\|_2^2 \\ &\mathbf{Y}_c = \mathbf{Y} - \alpha. \end{aligned} \tag{5.2}$$

The expectation $\mathbb{E}[.]$ is approximated by averaging over a training set of image blocks to be predicted, each paired with its context. $\lambda = 0.0005$. For learning the parameters $\boldsymbol{\phi}_m$, (5.2) is used, replacing $\boldsymbol{\theta}_m$ with $\boldsymbol{\phi}_m$ and f_m with g_m.

The optimization algorithm is ADAM [151] with mini-batches of size 100. The learning rate is 0.0001 for a fully-connected neural network and 0.0004 for a convolutional one. The number of iterations is 800000. The learning is divided by 10 after 400000, 600000, and 700000 iterations. Regarding the other hyperparameters of ADAM, the recommended values [151] are used. The learning rates and $\lambda = 0.0005$ were found via an exhaustive search. For a given m, a neural network is trained for each combination of learning rate and λ value in $\{0.007, 0.004, 0.001, 0.0007, 0.0004, 0.0001, 0.00007, 0.00004\} \times \{0.001, 0.0005, 0.0001\}$. We select the pair of parameter values giving the largest mean prediction PSNR on a validation set.

For both fully-connected and convolutional neural networks, the weights of all layers excluding the first one are initialized via Xavier's initialization [64]. Xavier's initialization allows to accelerate the training of several consecutive layers by making the variance of the input to all layers equivalent. The problem concerning the first layer is that the pixel values in the input contexts, belonging to $[-\alpha, 255 - \alpha]$, have a large variance, which leads to instabilities during training. That is why, in the first layer, the variance of the output is reduced with respect to that of the input by initializing the weights from a Gaussian distribution with mean 0 and standard deviation 0.01. The biases of all layers are initialized to 0.

5.3.3 Training data

The experiments in Sections 5.3.4 and 5.5 involve luminance images. That is why, for training, image blocks to be predicted, each paired with its context, are extracted from luminance images.

One 320×320 luminance crop $\mathbf{I}$ is, if possible, extracted from each RGB image in the ILSVRC2012 ImageNet training set [129]. This yields a set $\boldsymbol{\Gamma} = \left\{\mathbf{I}^{(i)}\right\}_{i=1...1048717}$.

The choice of the training set of pairs of contexts and image blocks to be predicted is a critical issue. Moreover, it needs to be handled differently for fully-connected and convolutional neural networks. Indeed, a convolutional neural network predicts large image blocks with complex textures, hence its need for high power of representation (see Appendix C.1). As a consequence, it overfits during the training phase if no training data augmentation [152, 153, 154] is used. In constrast, a fully-connected neural network predicts small blocks with simple textures, hence its need for relatively low power of representation. Thus, it does not overfit during the training phase without training data augmentation. Moreover, we have noticed that a training data augmentation scheme creates a bottleneck in training time for a fully-connected neural network. Therefore, training data augmentation is used for a convolutional neural network exclusively. The dependencies between a block to be predicted and its context should not be altered during the training data augmentation. Therefore, in our training data augmentation scheme, the luminance crops in $\boldsymbol{\Gamma}$ are exclusively randomly rotated and flipped. Precisely, for each step of ADAM, the scheme is Algorithm 6. s_{rotation} rotates its input image by angle $\psi \in \{0, \pi / 2, \pi, 3\pi / 2\}$ radians. s_{flipping} flips its input image horizontally with probability 0.5. e_{train} is the same function as e_{test}, except that e_{train} extracts $\left\{\mathbf{X}^{(i)}, \mathbf{Y}^{(i)}\right\}$ from

potentially rotated and flipped $\mathbf{I}$ instead of extracting $\mathbf{X}$ from $\mathbf{D}$ and the position of the extraction is random instead of being defined by the order of decoding. For training a fully-connected neural network, $\left\{\mathbf{X}_c^{(i)}, \mathbf{Y}_c^{(i)}\right\}_{i=1...10000000}$ is generated offline from $\mathbf{\Gamma}$, i.e. before the training starts.

Algorithm 6 : training data augmentation for the convolutional neural networks.

Inputs: $\mathbf{\Gamma}$, m, α.

for all $i \in [|1, 100|]$ **do**
  $\bar{i} \sim \mathcal{U}[1, 1048717]$
  $\psi \sim \mathcal{U}\left\{0, \frac{\pi}{2}, \pi, \frac{3\pi}{2}\right\}$
  $n_0, n_1 \sim \mathcal{U}\{0, 4, ..., m\}$
  $\mathbf{J} = s_{\text{flipping}}\left(s_{\text{rotation}}\left(\mathbf{I}^{(\bar{i})}, \psi\right)\right)$
  $\left\{\mathbf{X}^{(i)}, \mathbf{Y}^{(i)}\right\} = e_{\text{train}}\left(\mathbf{J}, m, n_0, n_1, \alpha\right)$
  $\mathbf{X}_c^{(i)} = \mathbf{X}^{(i)} - \alpha$
  $\mathbf{Y}_c^{(i)} = \mathbf{Y}^{(i)} - \alpha$
end for

Output: $\left\{\mathbf{X}_c^{(i)}, \mathbf{Y}_c^{(i)}\right\}_{i=1...100}$.

The issue regarding this generation of training data is that the training contexts have no quantization noise whereas, during the test phase in an image compression scheme, a context has quantization noise. This will be discussed during several experiments in Section 5.5.4.

5.3.4 Effectiveness of the random context masking

A natural question is whether the random context masking applied during the training phase to adapt the neural networks to the variable number of known pixels in the context degrades the prediction performance. To address this question, a neural network trained with random context masking is compared to a set of neural networks, each trained with a fixed mask size. The experiments are performed using fully-connected neural networks for block sizes 4×4 and 8×8 pixels (f_4 and f_8), and convolutional neural networks for block sizes 16×16, 32×32, and 64×64 pixels (g_{16}, g_{32}, and g_{64}).

The experiments are carried out using the 24 RGB images in the Kodak suite [150], converted into luminance. 960 image blocks to be predicted, each paired with its context, are extracted from these luminance images. Table 5.1 shows the PSNR, denoted $\text{PSNR}_{\text{PNNS},m}$, between the image block and its prediction via PNNS, averaged over the 960 blocks, for each block width m and each test pair $\{n_0, n_1\}$. We see that a neural network trained with a fixed mask size has performance in terms of PSNR that significantly degrades when the mask size during the training phase and the test phase differ. By contrast, a neural network trained with random context masking allows to get the best performance (written in bold) or the second best performance (written in italic) in terms of PSNR for all the possible mask sizes during the test phase. Moreover, when the second best PSNR performance is achieved, the second best PSNR is very close to the best one.

A second set of experiments is related to the success rate $\mu_{\text{PNNS},m}$ of the neural network based intra prediction, i.e. the rate at which PNNS provides a better prediction in terms of

Table 5.1: Comparison of (a) $\text{PNSR}_{\text{PNNS},4}$, (b) $\text{PSNR}_{\text{PNNS},8}$, (c) $\text{PSNR}_{\text{PNNS},16}$, (d) $\text{PSNR}_{\text{PNNS},32}$, and (e) $\text{PSNR}_{\text{PNNS},64}$ for different pairs $\{n_0, n_1\}$ during the training and test phases. $\mathcal{U}\{0, 4, ..., m\}$ denotes the fact that, during the training phase, n_0 and n_1 are uniformly drawn from the set $\{0, 4, 8, 12, ..., m\}$

.

Test $\{n_0, n_1\}$	Training f_4 with $\{n_0, n_1\}$				
	$\{0,0\}$	$\{0,4\}$	$\{4,0\}$	$\{4,4\}$	$\mathcal{U}\{0,4\}$
$\{0,0\}$	**34.63**	34.39	34.44	34.23	*34.57*
$\{0,4\}$	32.90	*34.39*	32.82	34.23	**34.42**
$\{4,0\}$	32.79	32.68	**34.44**	34.23	*34.39*
$\{4,4\}$	30.93	32.68	32.82	**34.23**	*34.20*

(a)

Test $\{n_0, n_1\}$	Training f_8 with $\{n_0, n_1\}$				
	$\{0,0\}$	$\{0,8\}$	$\{8,0\}$	$\{8,8\}$	$\mathcal{U}\{0,4,8\}$
$\{0,0\}$	**31.85**	31.73	31.68	31.62	*31.79*
$\{0,8\}$	30.93	**31.73**	30.83	31.62	*31.66*
$\{8,0\}$	30.70	30.75	**31.68**	31.62	**31.68**
$\{8,8\}$	29.58	30.75	30.83	**31.62**	*31.56*

(b)

Test $\{n_0, n_1\}$	Training g_{16} with $\{n_0, n_1\}$				
	$\{0,0\}$	$\{0,16\}$	$\{16,0\}$	$\{16,16\}$	$\mathcal{U}\{0,4,...,16\}$
$\{0,0\}$	*29.23*	22.69	24.85	20.76	**29.25**
$\{0,16\}$	28.65	**29.12**	24.66	23.99	*29.11*
$\{16,0\}$	28.43	22.60	*29.06*	22.99	**29.12**
$\{16,16\}$	27.87	28.37	28.35	**28.98**	*28.97*

(c)

Test $\{n_0, n_1\}$	Training g_{32} with $\{n_0, n_1\}$				
	$\{0,0\}$	$\{0,32\}$	$\{32,0\}$	$\{32,32\}$	$\mathcal{U}\{0,4,...,32\}$
$\{0,0\}$	**25.93**	22.13	22.29	18.85	*25.92*
$\{0,32\}$	25.39	**25.79**	22.18	21.75	*25.76*
$\{32,0\}$	25.41	22.02	**25.82**	20.38	*25.80*
$\{32,32\}$	25.00	25.31	25.32	**25.63**	*25.62*

(d)

Test $\{n_0, n_1\}$	Training g_{64} with $\{n_0, n_1\}$				
	$\{0,0\}$	$\{0,64\}$	$\{64,0\}$	$\{64,64\}$	$\mathcal{U}\{0,4,...,64\}$
$\{0,0\}$	*21.46*	19.41	19.78	18.17	**21.47**
$\{0,64\}$	21.27	*21.35*	19.68	19.95	**21.38**
$\{64,0\}$	21.11	19.18	**21.34**	19.06	**21.34**
$\{64,64\}$	20.94	21.08	21.16	**21.27**	**21.27**

(e)

Table 5.2: Comparison of success rates in percentage (a) $\mu_{\text{PNNS},4}$, (b) $\mu_{\text{PNNS},8}$, (c) $\mu_{\text{PNNS},16}$, (d) $\mu_{\text{PNNS},32}$, and (e) $\mu_{\text{PNNS},64}$ for different pairs $\{n_0, n_1\}$ during the training and test phases. $\mathcal{U}\{0, 4, ..., m\}$ denotes the fact that, during the training phase, n_0 and n_1 are uniformly drawn from the set $\{0, 4, 8, 12, ..., m\}$

.

Test $\{n_0, n_1\}$	Training f_4 with $\{n_0, n_1\}$				
	$\{0,0\}$	$\{0,4\}$	$\{4,0\}$	$\{4,4\}$	$\mathcal{U}\{0,4\}$
$\{0,0\}$	**22**%	17%	19%	16%	*19%*
$\{0,4\}$	15%	**18**%	13%	15%	*17%*
$\{4,0\}$	11%	11%	**20**%	17%	*19%*
$\{4,4\}$	11%	12%	13%	**16**%	*15%*

(a)

Test $\{n_0, n_1\}$	Training f_8 with $\{n_0, n_1\}$				
	$\{0,0\}$	$\{0,8\}$	$\{8,0\}$	$\{8,8\}$	$\mathcal{U}\{0,4,8\}$
$\{0,0\}$	**33**%	30%	30%	30%	*32%*
$\{0,8\}$	21%	**31**%	20%	**31**%	29%
$\{8,0\}$	21%	20%	**34**%	31%	*32%*
$\{8,8\}$	16%	20%	20%	**31**%	*30%*

(b)

Test $\{n_0, n_1\}$	Training g_{16} with $\{n_0, n_1\}$				
	$\{0,0\}$	$\{0,16\}$	$\{16,0\}$	$\{16,16\}$	$\mathcal{U}\{0,4,...,16\}$
$\{0,0\}$	**55**%	18%	20%	9%	*54%*
$\{0,16\}$	42%	**51**%	18%	18%	*50%*
$\{16,0\}$	40%	15%	*51%*	17%	**53**%
$\{16,16\}$	33%	36%	40%	**52**%	*49%*

(c)

Test $\{n_0, n_1\}$	Training g_{32} with $\{n_0, n_1\}$				
	$\{0,0\}$	$\{0,32\}$	$\{32,0\}$	$\{32,32\}$	$\mathcal{U}\{0,4,...,32\}$
$\{0,0\}$	**63**%	30%	30%	14%	**63**%
$\{0,32\}$	54%	**61**%	27%	27%	**61**%
$\{32,0\}$	54%	28%	**63**%	22%	*62%*
$\{32,32\}$	46%	53%	52%	**60**%	**60**%

(d)

Test $\{n_0, n_1\}$	Training g_{64} with $\{n_0, n_1\}$				
	$\{0,0\}$	$\{0,64\}$	$\{64,0\}$	$\{64,64\}$	$\mathcal{U}\{0,4,...,64\}$
$\{0,0\}$	**68**%	39%	43%	27%	*67%*
$\{0,64\}$	63%	*64%*	41%	44%	**65**%
$\{64,0\}$	62%	36%	*66%*	38%	**68**%
$\{64,64\}$	57%	61%	63%	*64%*	**66**%

(e)

Figure 5.7: Prediction of a block of size 16×16 pixels via the best H.265 mode in terms of PSNR and PNNS: (a) H.265 causal neighborhood, (b) PNNS context, (c) block to be predicted, (d) predicted block via the best H.265 mode (of index 27) in terms of PSNR, and (e) predicted block via PNNS.

PSNR than any other prediction of H.265. Again, the neural network trained with random context masking achieves the best rate (written in bold) or the second best rate (written in italic), where the second best rate is always very close to the best rate (see Table 5.2).

Therefore, we conclude that random context masking does not alter the performance of the trained neural networks. Besides, random context masking is an effective way of dealing with the changing number of known pixels in the context. Section 5.5 will discuss rate-distortion performance. Note that, in order to fix n_0 and n_1 during the test phase, the previous experiments have been carried out outside H.265.

5.3.5 Relevance of convolutional networks for predicting large blocks

Let us have a look at the overall efficiency of our convolutional neural networks for predicting large image blocks before comparing convolutional neural networks and fully-connected neural networks in this case. Figures 5.7 to 5.14 each compares the prediction of an image block provided by the best H.265 intra prediction mode in terms of prediction PSNR and the prediction provided by PNNS. Note that, the neural networks of PNNS yielding these predictions are trained via random context masking. Note also that $n_0 = n_1 = 0$ during the test phase. In Figure 5.13, the image block to be predicted contains the frame of a motorbike. There is no better H.265 intra prediction mode in terms of prediction PSNR than the DC mode in this case. In contrast, PNNS can predict a coarse version of the frame of the motorbike. In Figure 5.14, the block to be predicted contains lines of various directions. PNNS predicts a combination of diagonal lines, vertical lines and horizontal lines, which is not feasible for a H.265 intra prediction mode. However, Figures 5.11 and 5.12 show failure cases for the convolutional neural network. The quality of the prediction provided by the convolutional neural network looks worse than that provided by the best H.265 mode. Indeed, the prediction PSNRs are 27.09 dB and 23.85 dB for the H.265 mode of index 25 and the convolutional neural network respectively in Figure 5.11. They are 29.35 dB and 27.37 dB for the H.265 mode of index 29 and the convolutional neural network respectively in Figure 5.12. Therefore, unlike the H.265 intra prediction modes, the convolutional neural networks of PNNS can model a large set of complex textures found in large image blocks.

Table 5.3a compares our set of neural networks $\{f_4, f_8, g_{16}, g_{32}, g_{64}\}$ with the set of 4 fully-connected neural networks in [102], called Intra Prediction Fully-Connected Networks Single (IPFCN-S), in terms of prediction PSNR. *Single* refers to the single set of 4 fully-connected neural networks. This differs from the two sets of fully-connected neural networks mentioned later on in Section 5.5.3. The 4 fully-connected neural networks in

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

Figure 5.8: Prediction of a block of size 16×16 pixels via the best H.265 mode in terms of PSNR and PNNS: (a) H.265 causal neighborhood, (b) PNNS context, (c) block to be predicted, (d) predicted block via the best H.265 mode (planar) in terms of PSNR, and (e) predicted block via PNNS.

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

Figure 5.9: Prediction of a block of size 16×16 pixels via the best H.265 mode in terms of PSNR and PNNS: (a) H.265 causal neighborhood, (b) PNNS context, (c) block to be predicted, (d) predicted block via the best H.265 mode (planar) in terms of PSNR, and (e) predicted block via PNNS.

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

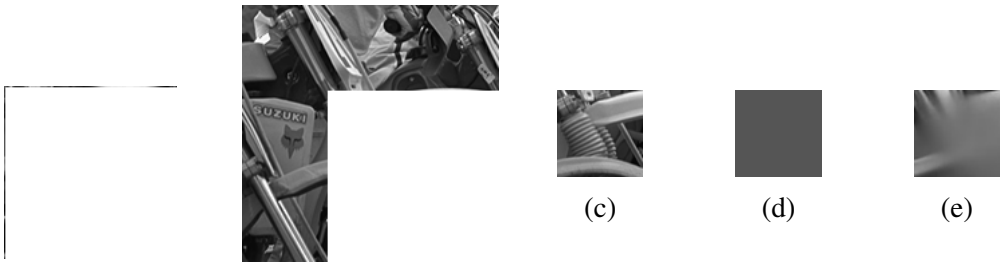
Figure 5.10: Prediction of a block of size 16×16 pixels via the best H.265 mode in terms of PSNR and PNNS: (a) H.265 causal neighborhood, (b) PNNS context, (c) block to be predicted, (d) predicted block via the best H.265 mode (of index 11) in terms of PSNR, and (e) predicted block via PNNS.

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

Figure 5.11: Prediction of a block of size 16×16 pixels via the best H.265 mode in terms of PSNR and PNNS: (a) H.265 causal neighborhood, (b) PNNS context, (c) block to be predicted, (d) predicted block via the best H.265 mode (of index 25) in terms of PSNR, and (e) predicted block via PNNS.

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

Figure 5.12: Prediction of a block of size 16×16 pixels via the best H.265 mode in terms of PSNR and PNNS: (a) H.265 causal neighborhood, (b) PNNS context, (c) block to be predicted, (d) predicted block via the best H.265 mode (of index 29) in terms of PSNR, and (e) predicted block via PNNS.



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

Figure 5.13: Prediction of a block of size 64×64 pixels via the best H.265 mode in terms of PSNR and PNNS: (a) H.265 causal neighborhood, (b) PNNS context, (c) block to be predicted, (d) predicted block via the best H.265 mode (DC) in terms of PSNR, and (e) predicted block via PNNS.

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

Figure 5.14: Prediction of a block of size 64×64 pixels via the best H.265 mode in terms of PSNR and PNNS: (a) H.265 causal neighborhood, (b) PNNS context, (c) block to be predicted, (d) predicted block via the best H.265 mode (planar) in terms of PSNR, and (e) predicted block via PNNS.

Table 5.3: Comparison of (a) $\text{PSNR}_{\text{PNNS},m}$ and $\text{PSNR}_{\text{IPFCN-S},m}$ [102] and (b) success rates $\mu_{\text{PNNS},m}$ and $\mu_{\text{IPFCN-S},m}$ [102] in percentage for $m \in \{4, 8, 16, 32, 64\}$. f_4, f_8, g_{16}, g_{32}, and g_{64} are trained via random context masking. During the test phase, $n_0 = n_1 = 0$. Note that, there is no comparison for block width 64 pixels as this block width is not considered in [102].

m	$\text{PSNR}_{\text{PNNS},m}$	$\text{PSNR}_{\text{IPFCN-S},m}$ [102]
4	**34.57**	33.70
8	**32.01**	31.44
16	**29.25**	28.71
32	**25.52**	24.96
64	21.47	–

(a)

m	$\mu_{\text{PNNS},m}$	$\mu_{\text{IPFCN-S},m}$ [102]
4	**19**%	14%
8	**31**%	26%
16	**54**%	35%
32	**60**%	41%
64	67%	–

(b)

[102] predict image blocks of sizes 4×4, 8×8, 16×16, and 32×32 pixels respectively. Let $\text{PSNR}_{\text{IPFCN-S},m}$ be the PSNR between the image block and its prediction via IPFCN-S, averaged over the 960 blocks. We thank the authors in [102] for sharing the trained model of each fully-connected neural network of IPFCN-S. Table 5.3a reveals that the performance of PNNS in terms of prediction PSNR is better than that of IPFCN-S for all sizes of image blocks to be predicted. More interestingly, when looking at the success rate $\mu_{\text{IPFCN-S},m}$ of IPFCN-S, i.e. the rate at which IPFCN-S provides a better prediction in terms of PSNR than any other prediction of H.265, the difference $\mu_{\text{PNNS},m} - \mu_{\text{IPFCN-S},m}$ increases with m (see Table 5.3b). This shows that convolutional neural networks are more appropriate than fully-connected ones in terms of prediction PSNR for large block sizes.

5.3.6 Justification of fully-connected neural networks for predicting small image blocks

Now that the benefit of using convolutional neural networks rather than fully-connected ones for predicting large image blocks is shown, the choice of fully-connected neural networks for predicting small blocks must be justified. Table 5.4 displays the average PSNR between the image block to be predicted and its prediction over the 960 blocks for both a fully-connected architecture and a convolutional one for each block width $m \in \{4, 8, 16\}$. The two fully-connected architectures f_4 and f_8 are already presented in Figure 5.2. The fully-connected architecture f_{16} is also that in Figure 5.2. The three convolutional architectures g_4, g_8, and g_{16} are shown in Table C.1 in Appendix C.1. We see that the convolutional neural network provides better prediction PSNRs on average than its fully-connected counterpart only when $m > 8$. This justifies why f_4 and f_8 are selected for predicting blocks of width 4 and 8 pixels respectively.

5.3.7 Different objective functions

In the objective function to be minimized over the neural network parameters, the distortion term involves the $L2$ norm of the difference between the image block to be predicted and its estimate (see (5.2)). We also tried an alternative. This alternative consisted

Table 5.4: Comparison of the average prediction PSNR given by the fully-connected architecture and that given by the convolutional one for different pairs $\{n_0, n_1\}$ during the test phase: (a) $\text{PSNR}_{\text{PNNS},4}$ given by f_4 and g_4, (b) $\text{PSNR}_{\text{PNNS},8}$ given by f_8 and g_8, and (c) $\text{PSNR}_{\text{PNNS},16}$ given by f_{16} and g_{16}. All neural networks are trained by drawing n_0 and n_1 uniformly from the set $\mathcal{U}\{0, 4, 8, 12, ..., m\}$.

Test $\{n_0, n_1\}$	f_4	g_4
$\{0, 0\}$	34.57	33.82
$\{0, 4\}$	34.42	33.66
$\{4, 0\}$	34.39	33.79
$\{4, 4\}$	34.20	33.54

(a)

Test $\{n_0, n_1\}$	f_8	g_8
$\{0, 0\}$	31.79	31.40
$\{0, 8\}$	31.66	31.41
$\{8, 0\}$	31.68	31.32
$\{8, 8\}$	31.56	31.32

(b)

Test $\{n_0, n_1\}$	f_{16}	g_{16}
$\{0, 0\}$	29.07	29.25
$\{0, 16\}$	28.95	29.11
$\{16, 0\}$	28.95	29.12
$\{16, 16\}$	28.86	28.97

(c)

in replacing the $L2$ norm by the $L1$ norm. The choice of the $L1$ norm could be justified by the fact that, in H.265, for a given image block to be predicted, the criterion for selecting several intra prediction modes among all modes involves the sum of absolute differences between the image block to be predicted and its estimate. But, this alternative distortion term did not yield any increase of the mean prediction PSNRs compared to those shown in Table 5.1.

A common alternative to the regularization by $L2$ norm of the weights is the regularization by $L1$ norm of the weights [75]. Both approaches have been tested and it was observed that the regularization by $L2$ norm of the weights slightly reduces the mean prediction error on a validation set at the end of the training.

5.4 Signalling of the prediction modes in H.265

Before moving on to the experiments in Section 5.5 where PNNS is integrated into a H.265 codec, the last issue is the signalling of the prediction modes inside H.265. Indeed, the integration of PNNS into H.265 requires to revisit the way all modes are signalled. Section 5.4 describes two ways of signalling PNNS into H.265. The purpose of setting up these two ways is to identify later on which signalling yields the largest PSNR-rate performance gains and discuss why a difference between them exists. The first signalling is the substitution of a H.265 intra prediction mode with PNNS. The second signalling is a switch between PNNS and the H.265 intra prediction modes.

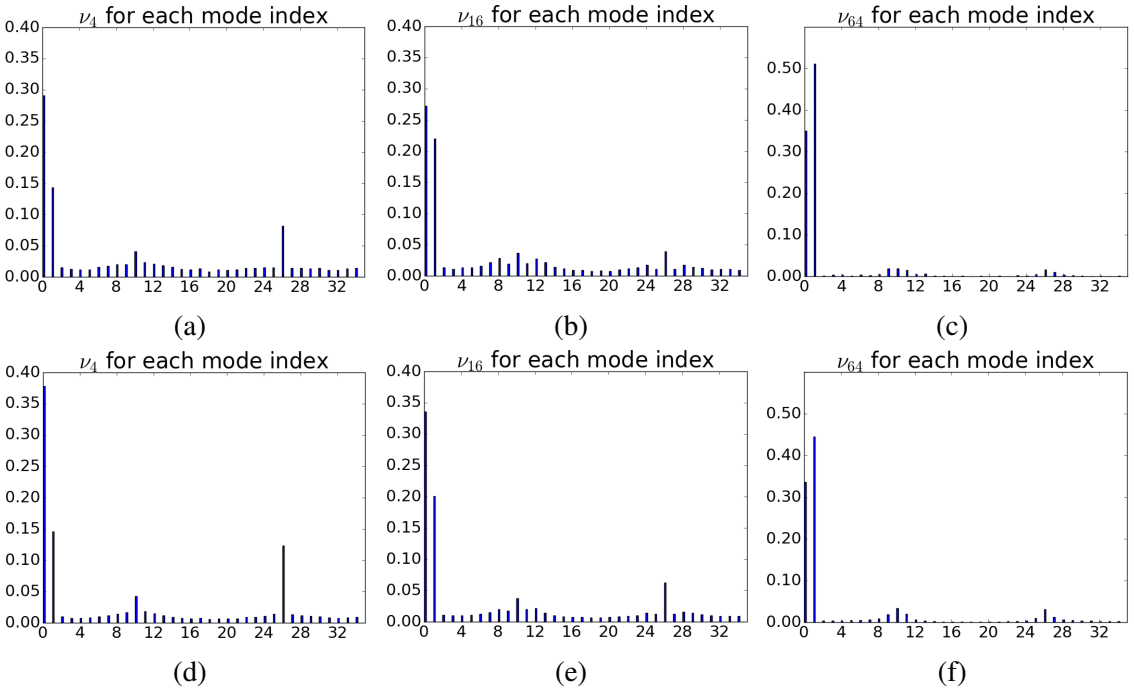


Figure 5.15: Analysis of (a, d) ν_4, (b, e) ν_{16}, and (c, f) ν_{64} for each mode. 100 luminance crops from either (a, b, c) the BSDS300 dataset or (d, e, f) the INRIA Holidays dataset are encoded via H.265 with QP $= 32$.

Table 5.5: Signalling of the modes described in [102].

Mode	codeword
Neural network mode	1
First MPM	010
Second MPM	0110
Third MPM	0111
Non-MPM and non-neural network mode	00 {5bits}

5.4.1 Substitution of a H.265 intra prediction mode with PNNS

Section 5.4.1 first describes how H.265 selects the best of its 35 intra prediction modes for predicting a given image block. Based on this, a criterion for finding the mode to be replaced with PNNS is built.

To select the best of its 35 intra prediction modes for predicting a given image block, H.265 proceeds in two steps. During the first step, the 35 modes compete with one another. Each mode takes the causal neighborhood of the block to compute a different prediction of the block. The sum of absolute differences between the input block and its prediction is linearly combined with the mode signalling cost, yielding the mode *fast* cost[2]. The modes associated to a given number of lowest *fast* costs are put into a *fast* list. During the second step, only the modes in the *fast* list compete with one another. The mode with the lowest rate-distortion cost is the best mode.

Knowing this, it seems natural to replace the mode that achieves the least frequently

2. *fast* stresses that the cost computation is relatively low.

the lowest rate-distortion cost. Therefore, the frequency of interest $\nu_{\overline{m}} \in [0,1], \overline{m} \in \{4,8,16,32,64\}$, is the number of times a mode has the lowest rate-distortion cost when $m = \overline{m}$ divided by the number of times the above-mentioned selection process is run when $m = \overline{m}$. To be generic, $\nu_{\overline{m}}$ should not be associated to luminance images of a specific type. That is why 100 480×320 luminance crops extracted from the BSDS300 dataset [141] and 100 1600×1200 luminance crops extracted from the INRIA Holidays dataset [155] are encoded with H.265 to compute $\nu_{\overline{m}}$. In this case, the mode of index 18 has on average the lowest $\nu_{\overline{m}}$ when $\overline{m} \in \{4,16,32,64\}$ (see Figure 5.15). Note that this conclusion is verified with QP $\in \{22,27,32,37,42\}$. Note also that statistics about the frequency of selection of each H.265 intra prediction mode have already been analyzed [156, 157]. But, they are incomplete for our case as they take into account few videos and do not differentiate each value of $\overline{m}$. Thus, PNNS replaces the H.265 mode of index 18.

As explained thereafter, the signalling cost the H.265 intra prediction mode of index 18 is variable and can be relatively large. When substituting the H.265 intra prediction mode of index 18 with PNNS, this variable cost transfers to PNNS. In contrast, the purpose of the second signalling of PNNS inside H.265 is to induce a fixed and relatively low signalling cost of PNNS.

5.4.2 Switch between PNNS and the H.265 intra prediction modes

The authors in [102] propose to signal the neural network mode with a single bit. This leads to the signalling of the modes summarized in Table 5.5. In addition, we modify the process of selecting the 3 Most Probable Modes (MPMs) [39] of the current PB to make the signalling of the modes even more efficient. More precisely, if the neural network mode is the mode selected for predicting the PB above the current PB or the PB on the left side of the current PB, then the neural network mode belongs to the MPMs of the current PB. As a result, redundancy appears as the neural network mode has not only the codeword 1 but also the codeword of a MPM. That is why, in the case where PNNS belongs to the MPMs of the current PB, we substitute each MPM being PNNS with either planar, DC or the vertical mode of index 26 such that the 3 MPMs of the current PB are different from each other. Besides, planar takes priority over DC, DC having priority over the vertical mode of index 26. See the code[1] for further details regarding this choice.

The key remark concerning Section 5.4 is that there is a tradeoff between the signalling cost of PNNS versus the signalling cost of the H.265 intra prediction modes. Indeed, the substitution (see Section 5.4.1) keeps the signalling cost of the H.265 intra prediction modes constant but the signalling cost of PNNS can be up to 6 bits. Instead, the switch decreases the signalling cost of PNNS and raises that of each H.265 intra prediction mode.

5.5 Experiments

Now that two ways of signalling PNNS inside H.265 are specified, these two ways can be compared in terms of PSNR-rate performance gains. Moreover, PNNS integrated into H.265 can be compared to IPFCN-S integrated into H.265 in terms of PSNR-rate performance gains.

5.5.1 Experimental settings

The H.265 HM16.15 software is used in all the following experiments. The configuration is all-intra main. Note that the following settings only mention the integration of PNNS into H.265 via the substitution of the H.265 intra prediction mode of index 18 with PNNS, so called *PNNS substitution*. But, the same settings apply to the integration of PNNS into H.265 via the switch between PNNS and the H.265 intra prediction modes, so called *PNNS switch*. The PSNR-rate performance of *PNNS substitution* with respect to H.265 is computed using Bjontegaard's metric, which is the average saving in bitrate of the rate-distortion curve of *PNNS substitution* with respect to the rate-distortion curve of H.265.

It is interesting to analyze whether there exists a range of bitrates for which *PNNS substitution* is more beneficial. That is why 3 different ranges of bitrates are presented. The first range, called *low rate*, refers to QP $\in \{32, 34, 37, 39, 42\}$. The second range, called *high rate*, corresponds to QP $\in \{17, 19, 22, 24, 27\}$. The third range, called *all rate*, computes Bjontegaard's metric with the complete set of QP values from 17 to 42. The H.265 common test condition [158] recommends $\{22, 27, 32, 37\}$ as QPs setting. Yet, we add several QPs to the recommended setting. This is because, to compute Bjontegaard's metric for *low rate* for instance, a polynom of degree 3 is fitted to rate-distortion points, and at least 4 rate-distortion points, i.e. 4 different QPs, are needed for a good fit.

Four test sets are used to cover a wide variety of images. The first test set contains the luminance channels of respectively Barbara, Lena, Mandrill and Peppers. The second test set contains the 24 RGB images in the Kodak suite, converted into luminance. The third test set gathers the 13 videos sequences of the classes B, C, and D of the H.265 CTC, converted into luminance. The fourth test set contains 6 widely used videos sequences [3], converted into luminance. Our work is dedicated to image coding. That is why only the first frame of each video sequence in the third and fourth test sets are considered. It is important to note that the training in Section 5.3, the extraction of the frequency of selection of each H.265 intra prediction mode in Section 5.4, and the current experiments involve 7 distinct sets of luminance images. This way, PNNS is not tuned for any specific test luminance image.

5.5.2 Analysis of the two ways of signalling the PNNS mode inside H.265

The most striking observation is that the PSNR-rate performance gains generated by *PNNS switch* are always larger than those provided by *PNNS substitution* (see Tables 5.6, 5.7, 5.8, and 5.9). This has two explanations. Firstly, *PNNS substitution* is hampered by the suppression of the original H.265 intra prediction mode of index 18. Indeed, the PSNR-rate performance gain is degraded when the original H.265 intra prediction mode of index 18 is a relevant mode for encoding a luminance image. The most telling example is the luminance channel of the first frame of BasketballDrill. When this channel is encoded via H.265, for the original H.265 intra prediction mode of index 18, $\nu_4 = 0.085$, $\nu_8 = 0.100$, $\nu_{16} = 0.116$, $\nu_{32} = 0.085$, and $\nu_{64} = 0.088$. This means that, compared to the average statistics in Figure 5.15, the original H.265 intra prediction mode of index 18

3. ftp://ftp.tnt.uni-hannover.de/pub/svc/testsequences/

Table 5.6: PSNR-rate performance gains compared with H.265 of *PNNS substitution* and *PNNS switch* for the first test set.

Image name	PSNR-rate performance gain			
	PNNS substitution			*PNNS switch*
	Low rate	*High rate*	*All rate*	*All rate*
Barbara (480×384)	2.47%	1.31%	1.79%	2.77%
Lena (512×512)	1.68%	2.11%	2.05%	3.78%
Mandrill (512×512)	0.77%	0.58%	0.67%	1.46%
Peppers (512×512)	1.61%	1.50%	1.71%	3.63%

Table 5.7: PSNR-rate performance gains compared with H.265 of *PNNS substitution* and *PNNS switch* for the second test set. The Kodak image size is 768×512 pixels.

Kodak image index	PSNR-rate performance gain			
	PNNS substitution			*PNNS switch*
	Low rate	*High rate*	*All rate*	*All rate*
1	1.20%	0.74%	0.95%	2.06%
2	0.59%	0.91%	0.88%	2.16%
3	0.91%	2.04%	1.59%	3.22%
4	1.78%	1.75%	1.80%	3.23%
5	1.40%	2.45%	2.08%	4.01%
6	1.43%	0.81%	1.12%	2.11%
7	1.12%	2.36%	1.76%	3.86%
8	1.01%	0.83%	0.98%	1.79%
9	1.54%	1.43%	1.46%	3.05%
10	2.20%	2.37%	2.42%	3.84%
11	0.93%	0.91%	1.00%	2.41%
12	0.96%	1.02%	1.07%	2.33%
13	0.83%	0.5%	0.64%	1.76%
14	0.96%	1.28%	1.20%	2.76%
15	1.53%	1.19%	1.37%	2.62%
16	0.66%	0.62%	0.70%	1.69%
17	1.35%	2.03%	1.80%	3.69%
18	0.68%	0.96%	0.88%	1.98%
19	1.44%	0.86%	1.05%	2.06%
20	0.92%	1.61%	1.38%	2.71%
21	0.99%	0.83%	0.94%	2.28%
22	0.56%	0.88%	0.78%	2.22%
23	1.20%	2.45%	2.03%	4.20%
24	0.68%	0.87%	0.80%	1.73%

Table 5.8: PSNR-rate performance gains compared with H.265 of *PNNS substitution* and *PNNS switch* for the third test set.

Video sequence		PSNR-rate performance gain			
		PNNS substitution			*PNNS switch*
		Low rate	*High rate*	*All rate*	*All rate*
B (1920×1080)	BQTerrace	1.66%	0.95%	1.29%	2.44%
	BasketballDrive	4.80%	2.87%	3.65%	5.20%
	Cactus	1.48%	1.51%	1.58%	3.05%
	ParkScene	0.64%	1.16%	0.97%	2.58%
	Kimono	1.28%	1.55%	1.65%	2.92%
C (832×480)	BQMall	1.20%	1.30%	1.30%	3.14%
	BasketballDrill	-1.18%	1.34%	0.39%	3.50%
	RaceHorsesC	1.34%	1.58%	1.53%	3.29%
	PartyScene	1.02%	0.91%	0.96%	2.42%
D (416×240)	BQSquare	0.79%	0.86%	0.86%	2.21%
	BasketballPass	1.61%	1.80%	1.48%	3.08%
	BlowingBubbles	0.66%	1.22%	1.02%	2.65%
	RaceHorses	1.32%	1.63%	1.54%	3.28%

Table 5.9: PSNR-rate performance gains compared with H.265 of *PNNS substitution* and *PNNS switch* for the fourth test set.

Video sequence	PSNR-rate performance gain			
	PNNS substitution			*PNNS switch*
	Low rate	*High rate*	*All rate*	*All rate*
Bus (352×288)	1.67%	1.17%	1.45%	2.74%
City (704×576)	1.55%	1.19%	1.35%	2.51%
Crew (704×576)	1.56%	1.24%	1.38%	3.10%
Football (352×288)	1.44%	1.78%	1.78%	3.52%
Harbour (704×576)	1.80%	0.73%	1.25%	2.51%
Soccer (704×576)	0.96%	0.95%	1.03%	1.90%

is used approximatively 10 times more frequently. This explains why the PSNR-rate performance gain provided by *PNNS substitution* is only 0.39% (see Table 5.8). The other way round, the luminance channel of the first frame of BasketballDrive is an insightful example. When this channel is encoded via H.265, for the original H.265 intra prediction mode of index 18, $\nu_4 = 0.004$, $\nu_8 = 0.005$, $\nu_{16} = 0.004$, $\nu_{32} = 0.004$, and $\nu_{64} = 0.000$. In this case, the original H.265 intra prediction mode of index 18 is almost never used. *PNNS substitution* thus yields 3.65% of PSNR-rate performance gain.

There is another explanation for the gap in PSNR-rate performance gain between *PNNS substitution* and *PNNS switch*. As shown in Section 5.3.5, PNNS is able to model a large set of complex textures found in large image blocks. PNNS is also able to model a large set of simple textures found in small blocks. Following the principle of Huffman Coding, an intra prediction mode that gives on average predictions of good quality, such as PNNS, should be signalled using fewer bits. However, an intra prediction mode that seldom yields the highest prediction quality, such as the H.265 intra prediction mode of index 4, should be signalled using more bits. This corresponds exactly to the principle of

(a) (b) (c)

Figure 5.16: Comparison of (a) a 100×100 crop of the luminance channel of the first frame of BQMall, (b) its reconstruction via H.265, and (c) its reconstruction via *PNNS switch*. In the two cases (b) and (c), QP $= 27$. For the entire luminance channel of the first frame of BQMall, for H.265, $\{\text{rate} = 0.670 \text{ bpp}, \text{PSNR} = 38.569 \text{ dB}\}$. For *PNNS switch*, $\{\text{rate} = 0.644 \text{ bpp}, \text{PSNR} = 38.513 \text{ dB}\}$.

(a) (b) (c)

Figure 5.17: Same comparison as in 5.16, using the luminance channel of the first frame of BlowingBubbles and QP $= 32$. For H.265, $\{\text{rate} = 0.538 \text{ bpp}, \text{PSNR} = 33.462 \text{ dB}\}$. For *PNNS switch*, $\{\text{rate} = 0.516 \text{ bpp}, \text{PSNR} = 33.435 \text{ dB}\}$.

(a) (b) (c)

Figure 5.18: Same comparison as in 5.16, using the luminance channel of the first frame of City and QP $= 37$. For H.265, $\{\text{rate} = 0.257 \text{ bpp}, \text{PSNR} = 30.992 \text{ dB}\}$. For *PNNS switch*, $\{\text{rate} = 0.249 \text{ bpp}, \text{PSNR} = 30.972 \text{ dB}\}$.

the switch between PNNS and the H.265 intra prediction modes. Therefore, *PNNS switch* beats *PNNS substitution* in terms of PSNR-rate performance gains. Figures 5.16, 5.17, and 5.18 each compares the reconstruction of a luminance image via H.265 and its reconstruction via *PNNS switch* at similar reconstruction PSNRs. More visual comparisons are available online[1]. Appendix C.3 shows the intra prediction mode selected for each PB in H.265, *PNNS substitution*, and *PNNS switch*. This reveals that, in *PNNS switch*, the PNNS mode is very often selected.

Another interesting conclusion emerges when comparing *low rate* and *high rate*. There is no specific range of bitrate for which *PNNS substitution* is more profitable (see Tables 5.6, 5.7, 5.8, and 5.9). Note that, in few cases, the PNSR-rate performance gain in *all rate* is slightly larger than those in *low rate* and *high rate*. This happens when the area between the rate-distortion curve of *PNNS substitution* and the rate-distortion curve of H.265 gets relatively large in the range QP $\in [27, 32]$.

5.5.3 Comparison with the state-of-the-art

Now, *PNNS switch* is compared to IPFCN-S integrated into H.265 in terms of PSNR-rate performance gains. It is important to note that the authors in [102] develop two versions of their set of 4 fully-connected neural networks for intra prediction. The first version, called IPFCN-S, is the one used in Section 5.3.5. The 4 fully-connected neural networks are trained on an unconstrained training set of image blocks to be predicted, each paired with its context. The second version is called Intra Prediction Fully-Connected Networks Double (IPFCN-D). The training data are dissociated into two groups. One group gathers image blocks exhibiting textures with angular directions, each paired with its context. The other group gathers image blocks exhibiting textures with non-angular directions, each paired with its context. In IPFCN-D, there are two sets of 4 fully-connected neural networks, each set being trained on a different group of training data. Then, the two sets are integrated into H.265. IPFCN-D gives slightly larger PSNR-rate performance gains than IPFCN-S. The comparison below involves IPFCN-S as our training set is not dissociated. But, this dissociation could also be applied to the training set of the neural networks of PNNS.

PNNS switch and IPFCN-S integrated into H.265 are compared on the third test set. The PSNR-rate performance gains of IPFCN-S are reported from [102]. We observe that the PSNR-rate performance gains of *PNNS switch* are larger than those of IPFCN-S integrated into H.265, apart from the case of the video sequence ParkScene (see Table 5.10). Note that, for several videos sequences, the difference in PSNR-rate performance gains between *PNNS switch* and IPFCN-S integrated into H.265 is significant. For instance, for the video sequence BasketballPass, the PSNR-rate performance gain of *PNNS switch* is 3.08% whereas that of IPFCN-S integrated into H.265 is 1.1%. Therefore, the use of both fully-connected neural networks and convolutional neural networks for intra prediction, the training with random context masking and the training data augmentation for training the convolutional neural networks of PNNS help boost the PSNR-rate performance gains. This is consistent with the conclusion in Section 5.3.5. Note that, even when comparing the PSNR-rate performance gains of *PNNS switch* with those of IPFCN-D integrated into H.265 which are reported in [102], *PNNS switch* often yields larger gains.

Table 5.10: PSNR-rate performance gains of our proposed *PNNS switch* and IPFCN-S [102] inside H.265 for the third test set. The reference is H.265.

Video sequence		PSNR-rate performance gain	
		our *PNNS switch*	IPFCN-S [102] inside H.265
B (1920×1080)	BQTerrace	$\mathbf{2.44}\%$	1.8%
	BasketballDrive	$\mathbf{5.20}\%$	3.4%
	Cactus	$\mathbf{3.05}\%$	2.7%
	ParkScene	2.58%	$\mathbf{2.8}\%$
	Kimono	$\mathbf{2.92}\%$	2.7%
C (832×480)	BQMall	$\mathbf{3.14}\%$	2.0%
	BasketballDrill	$\mathbf{3.50}\%$	1.1%
	RaceHorsesC	$\mathbf{3.29}\%$	2.9%
	PartyScene	$\mathbf{2.42}\%$	1.3%
D (412×240)	BQSquare	$\mathbf{2.21}\%$	0.6%
	BasketballPass	$\mathbf{3.08}\%$	1.1%
	BlowingBubbles	$\mathbf{2.65}\%$	1.6%
	RaceHorses	$\mathbf{3.28}\%$	2.8%

Table 5.11: Average computation time ratio with respect to H.265. 1 means that the computation time of the method is as long as that of H.265.

	our *PNNS switch*	IPFCN-S [102] inside H.265
Encoding	51	46
Decoding	191	190

5.5.4 Robustness of the neural networks to quantization noise in their input context

Section 5.5.3 just showed the effectiveness of the proposed PNNS in a rate-distortion sense. The last issue is that the neural networks of PNNS are trained on contexts without quantization noise but, during the test phase inside H.265, these neural networks are fed with contexts containing H.265 quantization noise. It is thus natural to ask whether, during the test phase inside H.265, the quality of the predictions provided by the neural networks of PNNS is degraded by the fact that no quantization noise exists in the contexts during their training.

To answer this, let us consider two different *PNNS switch*. In the first *PNNS switch*, our 5 neural networks, one for each block size, are dedicated to all QPs. Note that the first *PNNS switch* corresponds to the *PNNS switch* that has been used so far. In the second *PNNS switch*, a first set of 5 neural networks is dedicated to QP ≤ 27 whereas a second set is dedicated to QP > 27. Unlike the first set of neural networks, the second set is trained on contexts that are encoded and decoded via H.265 with QP $\sim \mathcal{U}\{32, 37, 42\}$ for each training context. For the third test set, the difference in PSNR-rate performance gain between the first *PNNS switch* and the second *PNNS switch* ranges between 0.0% and 0.1%. This means that there is no need to train the neural networks of PNNS on contexts with quantization noise.

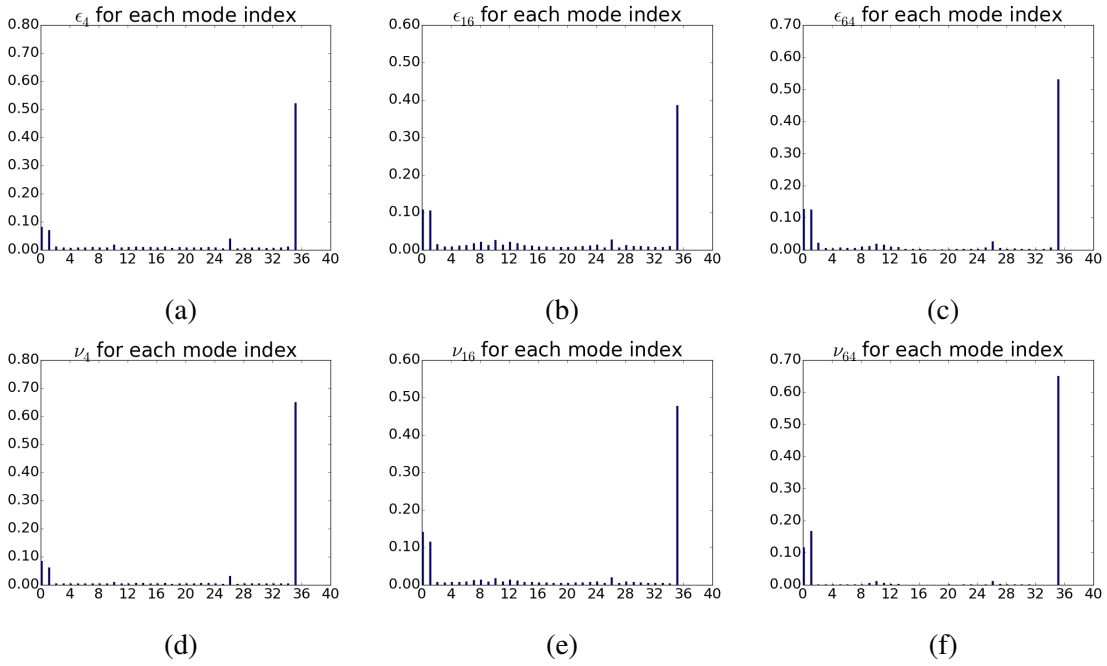


Figure 5.19: Analysis of (a) ϵ_4, (b) ϵ_{16}, (c) ϵ_{64}, (d) ν_4, (e) ν_{16}, and (f) ν_{64} for each mode. 100 480×320 luminance crops from the BSDS300 dataset are encoded via *PNNS switch* with QP $= 32$. The PNNS mode is the mode of index 35 in this figure.

5.5.5 Complexity

A fully-connected neural network needs an overcomplete representation to provide predictions with high quality. That is why the number of neurons in each fully-connected layer is usually much larger than the size of the context. Likewise, the number of feature maps in each convolutional layer of a convolutional neural network is usually large. This incurs a high computational cost. Table 5.11 gives the encoding and decoding times for *PNNS switch* and IPFCN-S, showing that *PNNS switch* has larger running times. But, the differences in running times between *PNNS switch* and IPFCN-S are not large in comparison with the increases of running times for both approaches with respect to H.265. A Bi-Xeon CPU E5-2620 is used for *PNNS switch*.

5.5.6 Memory consumption

In addition to the complexity, another issue arising from the integration of PNNS into H.265 is the increase of the memory consumption. For a regular intra prediction mode in H.265, the context around the image block to be predicted consists of one row of $2m + 1$ pixels above the block and one column of $2m$ pixels on the left side of the block. However, for a neural network of PNNS, the context around the image block consists of m rows of $3m$ pixels above the block and m columns of $2m$ pixels on the left side of the block (see Figure 5.5). Therefore, the hardware must be adapted to handle the large memory required by the input contexts to the neural networks of PNNS.

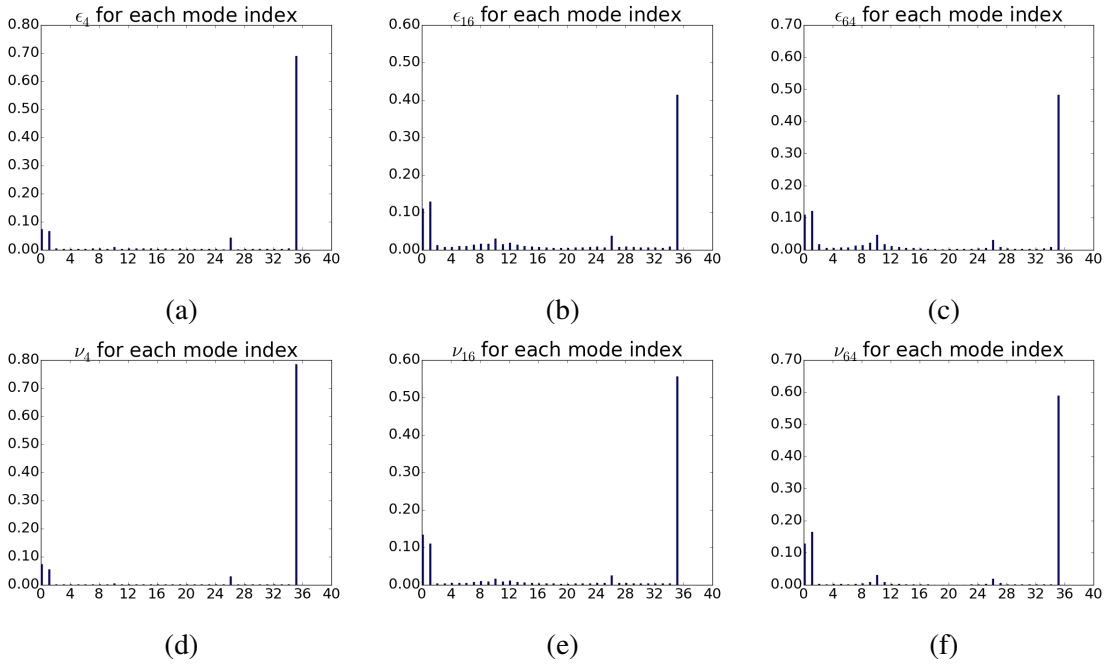


Figure 5.20: Analysis of (a) ϵ_4, (b) ϵ_{16}, (c) ϵ_{64}, (d) ν_4, (e) ν_{16}, and (f) ν_{64} for each mode. 100 1600×1200 luminance crops from the INRIA Holidays dataset are encoded via *PNNS switch* with QP $= 32$. The PNNS mode is the mode of index 35 in this figure.

5.5.7 Need for a rate-distortion objective function?

In an image codec, the ultimate criterion for selecting a prediction function among a predefined set of prediction functions must be a rate-distortion criterion. For example, the criterion in H.265 for selecting the best mode in terms of rate-distortion among the set of intra prediction modes is briefly described in Section 5.4.1. However, as mentioned in Section 5.3.2, the objective function for training a neural network of PNNS is based on the Euclidean distance between the image block to be predicted and the neural network prediction. Therefore, it is intuitive to think that the neural networks of PNNS should become more efficient for the intra prediction inside an image codec when a term reflecting the cost of encoding the residue of prediction is added to the objective function to be minimized.

To verify whether this intuition is correct, we can compare two frequencies inside *PNNS switch*. The first frequency $\epsilon_{\overline{m}} \in [0, 1]$, $\overline{m} \in \{4, 8, 16, 32, 64\}$, is the number of times an intra prediction mode has the lowest *fast* cost when the width of the block to be predicted $m = \overline{m}$ divided by the number of times the entire process of selecting the best intra prediction mode in terms of rate-distortion is run when $m = \overline{m}$. The second frequency $\nu_{\overline{m}} \in [0, 1]$ is defined in Section 5.4.1. In fact, $\forall \overline{m} \in \{4, 8, 16, 32, 64\}$, QP $= 32$, for the PNNS mode, $\epsilon_{\overline{m}}$ is slightly smaller than $\nu_{\overline{m}}$ (see Figures 5.19 and 5.20). This means that the PNNS mode is not penalized by the rate-distortion selection occuring after the *fast* selection of the intra prediction modes. This surprising observation may be justified by the fact that a neural network of PNNS tends to provide a blurry prediction of the image block to be predicted, implying that the residue of prediction rarely contains large coefficients in absolute value, which usually incur large encoding costs. Given this

conclusion, we did not give priority to the search for an objective function that reflects the rate-distortion target of image compression in this work on intra prediction. Note that, when the QP value changes, there exist exceptions to the previous observation. For instance, for $\overline{m} = 64$, QP $= 22$, for the PNNS mode, $\epsilon_{64} - \nu_{64} = 0.1$ using the luminance crops from the INRIA Holidays dataset.

5.6 Conclusion

To conclude, a set of neural network architectures, including both fully-connected neural networks and convolutional neural networks, is presented for intra prediction. It is shown that fully-connected neural networks are well adapted to the prediction of image blocks of small sizes whereas convolutional ones provide better predictions in large blocks. Our neural networks are trained via a random context masking of their context so that they adapt to the variable number of available decoded pixels for the prediction in an image compression scheme. When integrated into a H.265 codec, the proposed neural networks are shown to give rate-distortion performance gains compared with the H.265 intra prediction. Moreover, it is shown that these neural networks can cope with the quantization noise present in the prediction context, i.e. they can be trained on undistorted contexts, and then generalize well on distorted contexts in an image compression scheme. This greatly simplifies training as quantization noise does not need to be taken into account during the training phase.

Chapter 6

Conclusion and perspectives

6.1 Conclusion

This thesis investigated the learning of a transform and the learning of an intra predictor for image compression via neural networks. The results of rate-distortion optimality for transform coding requires assumptions on the distribution of the image pixels that do not exactly meet their true distribution (see Section 1.2). Likewise, the complex distribution of the image pixels is not fully considered in the design of the classic prediction filters (see Section 1.2). The learning from a set of natural images is a way to discover a model of the transform and a model of the prediction filter that best suit the true distribution of the image pixels.

The major challenge regarding the learning of a transform for image compression is to obtain a learned transform that is efficient in terms of rate-distortion while being able to compress at different rates during the test phase, which is called *rate-agnostic*. To achieve rate-distortion efficiency, it is essential to integrate into the training of a deep autoencoder a reliable measure of the rate. That is why a first approach is to train a deep sparse autoencoder using a constraint on the number of non-zero coefficients in the sparse representation of the image and approximate the rate via a function of the number of the non-zero coefficients in this representation. In this case, a way to make the deep sparse autoencoder *rate-agnostic* is to drive the sparsity parameter stochastically during the training phase. A second approach to reach rate-distortion efficiency is to insert into the objective function to be minimized during the training of the deep autoencoder a term approximating the entropy of the quantized representation. In this case, during the test phase, the quantization step sizes can be increased from their value at the end of the training phase in order to compress at multiple rates. This means that the image compression scheme is *rate-agnostic* by influencing the uniform scalar quantizer.

Regarding the learning of an intra predictor for image compression via neural networks, the challenge is that the learned intra predictor must be adaptive in a number of respects. The intra predictor must be able to predict accurately blocks of different sizes. That is why we propose a set of neural network architectures. Fully-connected architectures are dedicated to the prediction of blocks of small sizes whereas convolutional architectures are dedicated to the prediction of large blocks. Moreover, in the input context to a neural network, pixels surrounding the block to be predicted may have not been encoded and decoded yet. This means that some information may be missing. Therefore,

the neural networks are trained via a random masking of their input context. This way, they learn the prediction from a variable amount of information. Finally, a neural network must predict accurately a given image block, whatever the level of quantization noise in its input context. We show experimentally that the neural networks trained on undistorted contexts generalize well on distorted contexts.

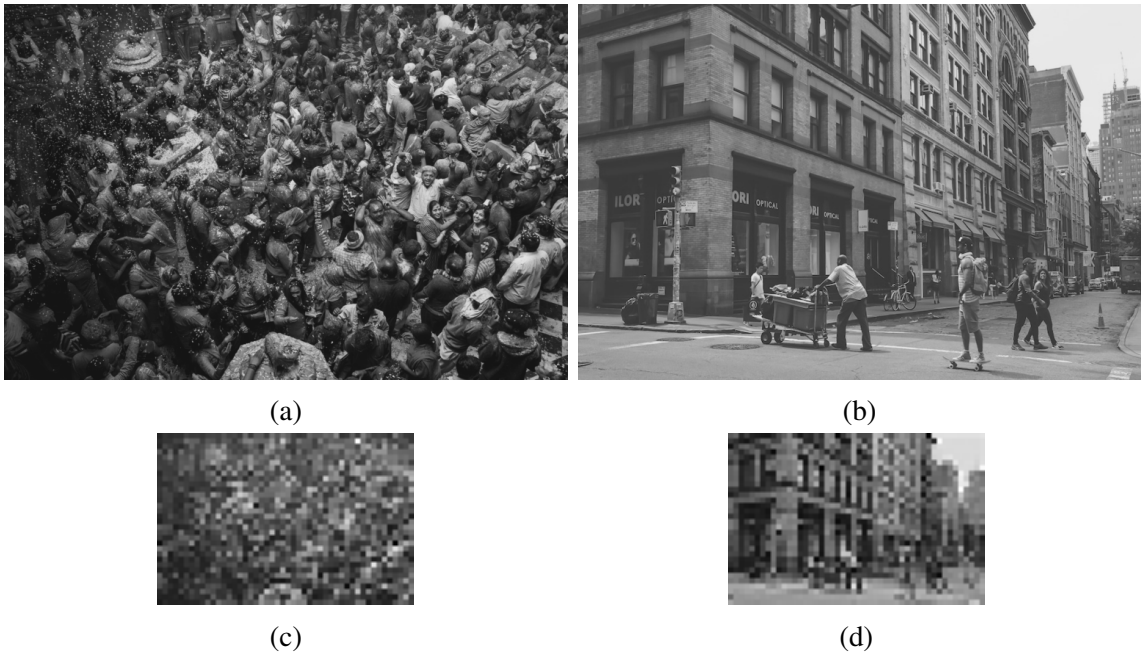
6.2 Perspectives

This section describes perspectives arising from the results of the thesis. These perspectives can be split into two categories. First, one may wonder whether the learned transform via neural networks has lived up to the expectation of providing quasi-independent transform coefficients (see Section 6.2.1). Then, the focus will be on a neural network based intra predictor that runs faster during the test phase and consumes less memory so that it better meets the requirements of the image and video compression normalization (see Section 6.2.2). Finally, this section discusses how to further improve the neural network based intra predictor in terms of quality of prediction (see Section 6.2.3).

6.2.1 Dependencies between the transform coefficients

In the recent approaches for learning transforms via neural networks for image compression, some evidence indicate that the transform coefficients provided by the encoder exhibit spatial dependencies. Let us consider the autoencoder in Section 4.2.2. Figures 6.1b and 6.1d show respectively an input luminance image to the encoder and the feature map with the largest amount of spatial dependencies in the stack of feature maps returned by the encoder. The spatial structure in this feature map is obvious. Moreover, in all recent end-to-end learned image compression schemes, improvements in rate-distortion performance have emerged thanks to the joint learning of a transform and a context model for entropy coding [159] (see Section 3.1.3.1). This means that the learned transform does not manage to exploit alone all the statistical dependencies within the input images. Finally, during the thesis, we tried to learn transforms that exploit longer-range spatial dependencies by designing deeper neural networks. For instance, convolutional layers with various strides, i.e. spatial subsamplings, were added to the autoencoder architecture in Section 4.2.2 . However, none of these attempts resulted in better rate-distortion tradeoffs on average. Given these remarks, one might believe that the learning of a transform for image compression via neural networks has plateaued in terms of rate-distortion performance.

The term *on average* in the above remarks might be an essential detail. In transform coding, one looks for a single transform domain in which, for any image, the transform coefficients are ideally independent and the signal energy is compacted in few of them. Given the tremendeous variability from one natural image to another, it might be difficult to find one transform domain with such properties for all images. For instance, Figures 6.1a and 6.1b display two luminance images. For each of them, Figures 6.1c and 6.1d show the feature map with the largest amount of spatial dependencies in the stack of feature maps returned by the encoder of the autoencoder in Section 4.2.2. For the first image, the displayed feature map looks like noise. The transform domain is suitable. But, this is not the case for the second image. In this case, a context model is necessary.



(a) (b)

(c) (d)

Figure 6.1

To further improve the learning of a transform via neural networks, several transforms can be learned jointly, each transform specializing in one class of natural images.

6.2.2 Neural network based intra predictor for the image and video compression normalization

The integration of a neural network based intra prediction mode into H.265 yields significant gains in terms of rate-distortion. But, the neural network based intra prediction mode incurs a large increase of the running time and the memory consumption of H.265.

6.2.2.1 Reduction of the running time

On the encoder side, the running time of H.265 with the neural network based intra prediction mode is around 50 times larger than the running time of H.265. On the decoder side, the running time of H.265 with the neural network based intra prediction mode is approximatively 200 times larger than the running time of H.265 (see Section 5.5.5). This is especially critical on the decoder side as a slower decoding damages the user experience for those who are watching images and videos.

To speed up the running time of the neural network based intra predictor, the heuristics in the literature for designing smaller neural network architectures with less parameters can be explored. These heuristics have been mainly developped in the context of image recognition. For instance, SqueezeNet [160] is a small convolutional neural network that meets the accuracy of AlexNet [66] on the ILSVRC2012 image recognition challenge. SqueezeNet contains 50 times less parameters than AlexNet.

6.2.2.2 Reduction of the memory consumption

The input contexts to the neural networks of the neural network based intra prediction mode are much larger than the input contexts to the regular H.265 intra prediction modes. Moreover, the parameters of the neural networks must be stored on both the encoder and decoder sides. These cause an increase of the memory consumption when integrating the neural network based intra prediction mode.

Two lines of research can yield a reduction of the memory cost of storing the parameters of the neural networks. First, the design of smaller neural network architectures with less parameters (see Section 6.2.2.1) decreases both the running time and this memory cost. Then, the parameters of the neural networks can be compressed [161, 162, 163]. This compression takes several forms. First, the neural network can be pruned by learning only the important connections. In addition, the weights of the neural network can be quantized and entropy coded.

6.2.3 Neural network based intra predictor with improved quality of prediction

6.2.3.1 Matching the block to be predicted with relevant context

A problem appears when a convolutional neural network takes a large context of pixels previously encoded and decoded around a given image block to be predicted to infer a prediction of this block. Most regions of the large context may be irrelevant for the prediction whereas a few regions may contain all the information of interest for the prediction. But, convolutional neural networks are not designed to copy regions of their input context and paste them into their output prediction. Note that, unlike convolutional neural network, the locally linear embedding methods [164, 165] have the capability of performing this copy-paste task.

To correct this defect, a neural network module can match each region of a coarse prediction of the block to be predicted with a region of the input context, and use the selected regions of the input context to infer a refined prediction. More specifically, the convolutional neural network for intra prediction first provides a coarse prediction of the image block to be predicted. Then, the neural network module evaluates which region of the input context best matches each region of the coarse prediction. Finally, the selected regions of the input context and the coarse prediction can be transformed into a refined version of the prediction via a neural network, in a similar manner to [166].

6.2.3.2 Learned side information

The main limitation of the neural network based intra prediction mode is that it struggles at accurately predicting an image block that is uncorrelated from its neighboring already encoded and decoded pixels. For instance, let us say that, for a given image block, there is a discontinuity in the distribution of pixel intensity between this block and its neighboring already encoded and decoded pixels. In the rest of this section, this case will be called an unpredictable case. The neural network based intra prediction mode can only rely on the input context of neighboring already encoded and decoded pixels to infer a prediction of the block, hence its failure.

A way to circumvent the problem of unpredictable cases would be to supplement the neural network for intra prediction with a learned module that encodes side information. In details, this learned module takes the image block to be predicted and its context as inputs to generate a compressible representation. This compressible representation can be quantized and converted into a bitstream via binary arithmetic encoding. Then, the reconstructed quantized representation obtained via binary arithmetic decoding is fed into an intermediate layer of the neural network for intra prediction. By constraining the coding cost of the quantized representation and jointly optimizing the neural network for intra prediction and the learned module, the compressible representation should contain exclusively information that is not redundant with the context information.

Author's publications

Conference Papers

Thierry Dumas, Aline Roumy, and Christine Guillemot. Shallow spase autoencoders versus sparse coding algorithms for image compression. In *ICME Workshops*, 2016.
Thierry Dumas, Aline Roumy, and Christine Guillemot. Image compression with stochastic winner-take-all autoencoder. In *ICASSP* 2017.
Thierry Dumas, Aline Roumy, and Christine Guillemot. Auto-encoder optimisé au sens débit-distorsion : indépendant de la quantification? In *GRETSI*, 2017.
Thierry Dumas, Aline Roumy, and Christine Guillemot. Autoencoder based image compression: can the learning be quantization independent? In *ICASSP*, 2018.

Journal Paper

Thierry Dumas, Aline Roumy, and Christine Guillemot. Context-adaptive neural network based intra prediction for image compression. Submitted to *Transactions on Image Processing* in July 2018.

Appendix A

A.1 Intra prediction in H.265

The prediction of a luminance image block via the best intra prediction function among the 35 H.265 intra prediction functions according to a rate-distortion criterion is shown in Figures from A.1 to A.16. In Figures A.1, A.2, A.3, A.4, A.5, A.6, A.7, and A.8, the image block to be predicted is small, i.e. of size smaller than 8×8 pixels. In Figures A.9, A.10, A.11, A.12, A.13, A.14, A.15, and A.16, the image block to be predicted is large, i.e. of size larger than 32×32 pixels.

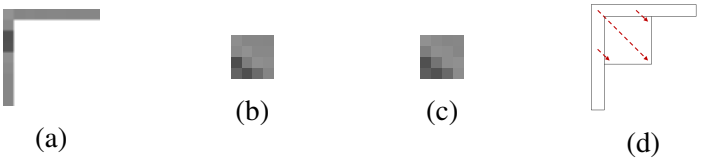


Figure A.1: Prediction of a luminance image block of size 4×4 pixels via the selected H.265 function: (a) previously encoded and decoded neighboring pixels, (b) block to be predicted, (c) predicted block via the selected H.265 function, and (d) prediction sketch (the dotted arrows showing the direction of propagation). The index of the selected H.265 function is 17.

Figure A.2: Prediction of a luminance image block of size 4×4 pixels via the selected H.265 function. The legend is the one in Figure A.1. The index of the selected H.265 function is 7.

Figure A.3: Prediction of a luminance image block of size 4×4 pixels via the selected H.265 function. The index of the selected H.265 function is 8.

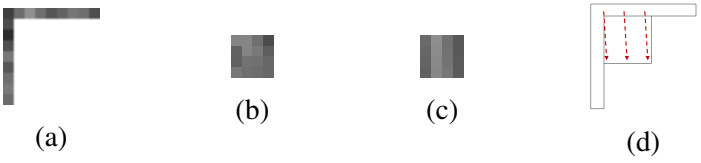
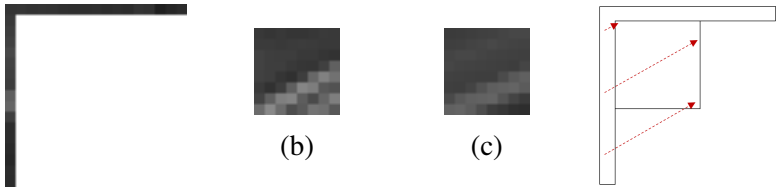
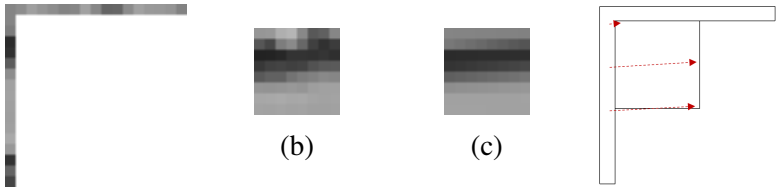


Figure A.4: Prediction of a luminance image block of size 4×4 pixels via the selected H.265 function. The index of the selected H.265 function is 25.



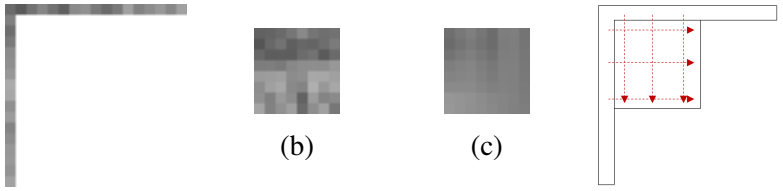
(a) (b) (c) (d)

Figure A.5: Prediction of a luminance image block of size 8×8 pixels via the selected H.265 function: (a) previously encoded and decoded neighboring pixels, (b) block to be predicted, (c) predicted block via the selected H.265 function, and (d) prediction sketch (the dotted arrows showing the direction of propagation). The index of the selected H.265 function is 6.



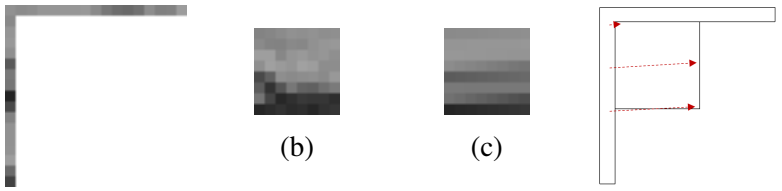
(a) (b) (c) (d)

Figure A.6: Prediction of a luminance image block of size 8×8 pixels via the selected H.265 function. The legend is the one in Figure A.5. The index of the selected H.265 function is 9.



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

Figure A.7: Prediction of a luminance image block of size 8×8 pixels via the selected H.265 function. The selected H.265 function, called planar, has 0 as index.



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

Figure A.8: Prediction of a luminance image block of size 8×8 pixels via the selected H.265 function. The index of the selected H.265 function is 9.

Figure A.9: Prediction of a luminance image block of size 32×32 pixels via the selected H.265 function: (a) previously encoded and decoded neighboring pixels, (b) block to be predicted, (c) predicted block via the selected H.265 function, and (d) prediction sketch (the dotted arrows indicating the direction of propagation). The index of the selected H.265 function is 7.

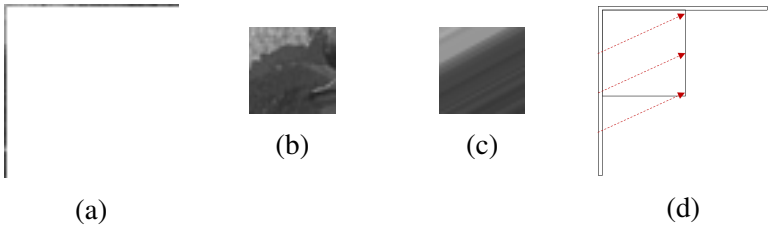


Figure A.10: Prediction of a luminance image block of size 32×32 pixels via the selected H.265 function. The legend is the one in Figure A.9. The index of the selected H.265 function is 5.

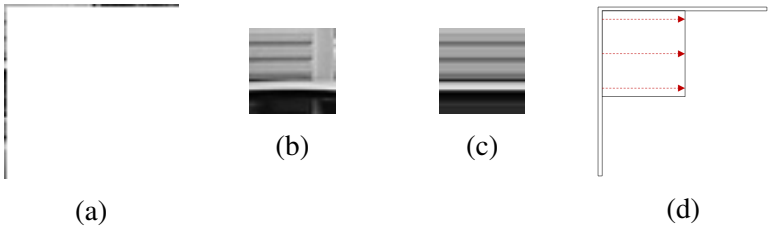


Figure A.11: Prediction of a luminance image block of size 32×32 pixels via the selected H.265 function. The index of the selected H.265 function is 10.

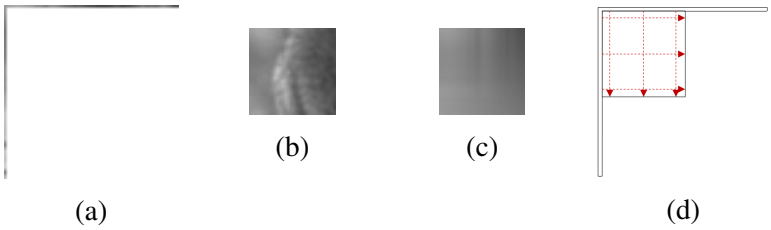


Figure A.12: Prediction of a luminance image block of size 32×32 pixels via the selected H.265 function. The selected H.265 function, called planar, has 0 as index.

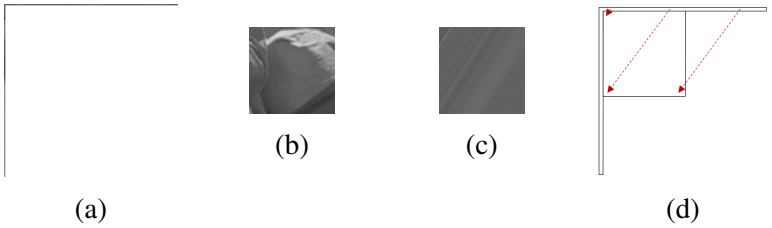


Figure A.13: Prediction of a luminance image block of size 64×64 pixels via the selected H.265 function: (a) previously encoded and decoded neighboring pixels, (b) block to be predicted, (c) predicted block via the selected H.265 function, and (d) prediction sketch (the dotted arrows indicating the direction of propagation). The index of the selected H.265 function is 32.

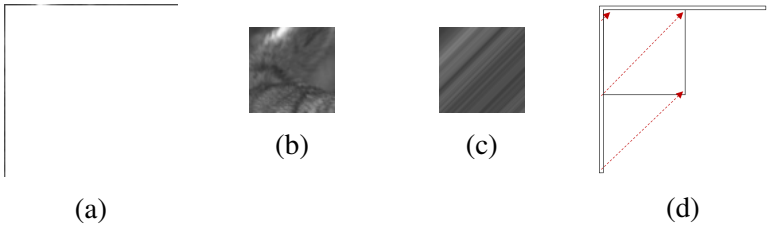


Figure A.14: Prediction of a luminance image block of size 64×64 pixels via the selected H.265 function. The legend is the one in Figure A.13. The index of the selected H.265 function is 2.

Figure A.15: Prediction of a luminance image block of size 64×64 pixels via the selected H.265 function. The selected H.265 function, called DC, predicts each block pixel as the average of the neighboring pixels colored in red. Its index is 1.

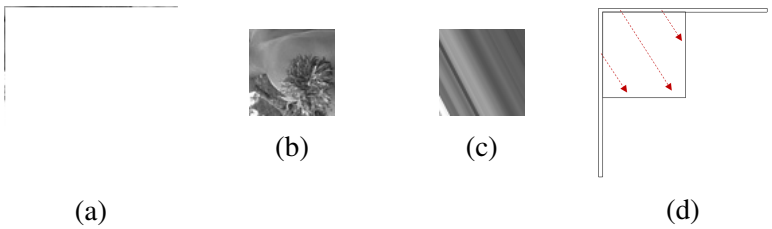


Figure A.16: Prediction of a luminance image block of size 64×64 pixels via the selected H.265 function. The index of the selected H.265 function is 21.

Appendix B

B.1 Details on the dictionary updates in Algorithm 3

The minimization in (3.31) over the dictionary $\mathbf{D}_o$ and the set of sparse decompositions $\left\{\mathbf{z}^{(1)}, \ ..., \mathbf{z}^{(N)}\right\}$ is not jointly convex. However, if the set of sparse decompositions is fixed, i.e. it is assumed not to depend on the dictionary, the minimization over the dictionary is convex. That is why the common way of solving problems like (3.31) is to alternate between the set of sparse decompositions and the dictionary, minimizing over one while keeping the other one fixed [167, 168, 117]. Algorithm 3 draws on this approach. For $i \in [|1, N|]$, given a training image patch $\mathbf{x}^{(i)}$ and the fixed dictionary, a sparse decomposition $\mathbf{z}^{(i)}$ of $\mathbf{x}^{(i)}$ over the dictionary is computed via OMP. Then, given the training image patch and its fixed sparse decomposition, a gradient descent step is applied to the dictionary.

Appendix B.1 first details the gradient descent step. Then, it is demonstrated that, to derive the gradient descent step, it is not necessary to fix this sparse decomposition. It is sufficient to assume that the support of the sparse decomposition does not change when a coefficient of the dictionary slightly varies.

B.1.1 Gradient descent step applied to the dictionary

$$\begin{aligned}
\mathcal{G}\left(\mathbf{x}^{(i)}, \mathbf{z}^{(i)}, \mathbf{D}_o\right) &= \left(\mathbf{x}^{(i)} - \mathbf{D}_o\mathbf{z}^{(i)}\right)^T \left(\mathbf{x}^{(i)} - \mathbf{D}_o\mathbf{z}^{(i)}\right) \\
&= \left(\mathbf{x}^{(i)}\right)^T \mathbf{x}^{(i)} - 2\left(\mathbf{x}^{(i)}\right)^T \mathbf{D}_o\mathbf{z}^{(i)} + \left(\mathbf{z}^{(i)}\right)^T \left(\mathbf{D}_o\right)^T \mathbf{D}_o\mathbf{z}^{(i)} \\
\frac{\partial \mathcal{G}\left(\mathbf{x}^{(i)}, \mathbf{z}^{(i)}, \mathbf{D}_o\right)}{\partial \mathbf{D}_o} &= -2\frac{\partial\left(\left(\mathbf{x}^{(i)}\right)^T \mathbf{D}_o\mathbf{z}^{(i)}\right)}{\partial \mathbf{D}_o} + \frac{\partial\left(\left(\mathbf{z}^{(i)}\right)^T \left(\mathbf{D}_o\right)^T \mathbf{D}_o\mathbf{z}^{(i)}\right)}{\partial \mathbf{D}_o} \\
&= -2\mathbf{x}^{(i)}\left(\mathbf{z}^{(i)}\right)^T + 2\mathbf{D}_o\mathbf{z}^{(i)}\left(\mathbf{z}^{(i)}\right)^T \text{ as } \mathbf{z}^{(i)} \text{ is fixed.} \\
&= 2\left(\mathbf{D}_o\mathbf{z}^{(i)} - \mathbf{x}^{(i)}\right)\left(\mathbf{z}^{(i)}\right)^T
\end{aligned} \tag{B.1}$$

B.1.2 Assumption on the sparse decomposition

From now on, the equations are written using vector coefficients to avoid introducing complicated notations, like tensors of dimension 3.

$$\begin{aligned}
\frac{\partial \mathcal{G}\left(\mathbf{x}^{(i)}, \mathbf{z}^{(i)}, \mathbf{D}_o\right)}{\partial D_{o,\alpha\beta}} &= \frac{\partial\left(\sum_{j=1}^{m}\left(x_j^{(i)} - \hat{x}_j^{(i)}\right)^2\right)}{\partial D_{o,\alpha\beta}} \text{ where } \hat{\mathbf{x}}^{(i)} = \mathbf{D}_o\mathbf{z}^{(i)} \\
&= 2\sum_{j=1}^{m}\left(\hat{x}_j^{(i)} - x_j^{(i)}\right)\frac{\partial \hat{x}_j^{(i)}}{\partial D_{o,\alpha\beta}} \\
&= 2\sum_{j=1}^{m}\left(\hat{x}_j^{(i)} - x_j^{(i)}\right)\sum_{k=1}^{n}\left(\frac{\partial D_{o,jk}}{\partial D_{o,\alpha\beta}} z_k^{(i)} + D_{o,jk}\frac{\partial z_k^{(i)}}{D_{o,\alpha\beta}}\right) \\
&= 2\left(\hat{x}_\alpha^{(i)} - x_\alpha^{(i)}\right) z_\beta^{(i)} + 2\sum_{j=1}^{m}\left(\hat{x}_j^{(i)} - x_j^{(i)}\right)\sum_{k=1}^{n} D_{o,jk}\frac{\partial z_k^{(i)}}{\partial D_{o,\alpha\beta}} \\
&= 2\left(\hat{x}_\alpha^{(i)} - x_\alpha^{(i)}\right) z_\beta^{(i)} - 2\sum_{k=1}^{n}\left(\mathbf{r}^{(i)}\right)^T \mathbf{D}_{o,k}\frac{\partial z_k^{(i)}}{\partial D_{o,\alpha\beta}} \qquad \text{(B.2)}
\end{aligned}$$

$\mathbf{r}^{(i)} = \mathbf{x}^{(i)} - \hat{\mathbf{x}}^{(i)}$ is the residue of the sparse decomposition of $\mathbf{x}^{(i)}$ over $\mathbf{D}_o$. As $\mathbf{z}^{(i)}$ was previously computed via OMP, $\mathbf{z}^{(i)}$ is defined as

$$\mathbf{z}_{S^{(i)}}^{(i)} = \left(\mathbf{D}_{o,S^{(i)}}\right)^{\dagger}\mathbf{x}^{(i)}.$$

$S^{(i)} = \text{supp}\left(\mathbf{z}^{(i)}\right) = \left\{k \in [|1, n|] \mid z_k^{(i)} \neq 0\right\}$ is the support of $\mathbf{z}^{(i)}$. $\mathbf{D}_{o,S^{(i)}}$ is the dictionary obtained by keeping the columns of $\mathbf{D}_o$ whose indices belong to $S^{(i)}$. $\mathbf{z}_{S^{(i)}}^{(i)}$ is the vector obtained by keeping the coefficients of $\mathbf{z}^{(i)}$ whose indices belong to $S^{(i)}$.

Now, if we assume that $S^{(i)}$ does not change when $D_{o,\alpha\beta}$ slightly varies, (B.2) becomes

$$\frac{\partial \mathcal{G}\left(\mathbf{x}^{(i)}, \mathbf{z}^{(i)}, \mathbf{D}_o\right)}{\partial D_{o,\alpha\beta}} = 2\left(\hat{x}_\alpha^{(i)} - x_\alpha^{(i)}\right) z_\beta^{(i)} - 2\sum_{k \in S^{(i)}}\left(\mathbf{r}^{(i)}\right)^T \mathbf{D}_{o,k}\frac{\partial z_k^{(i)}}{\partial D_{o,\alpha\beta}}. \qquad \text{(B.3)}$$

Knowing that the residue $\mathbf{r}^{(i)}$ is orthogonal to any column in $\mathbf{D}_o$ whose index belongs to $S^{(i)}$ [169, Section 3.1.2], (B.3) becomes

$$\frac{\partial \mathcal{G}\left(\mathbf{x}^{(i)}, \mathbf{z}^{(i)}, \mathbf{D}_o\right)}{\partial D_{o,\alpha\beta}} = 2\left(\hat{x}_\alpha^{(i)} - x_\alpha^{(i)}\right) z_\beta^{(i)}. \qquad \text{(B.4)}$$

By writing (B.4) in vector form, (B.4) is turned into (B.5), which is similar to (B.1).

$$\frac{\partial \mathcal{G}\left(\mathbf{x}^{(i)}, \mathbf{z}^{(i)}, \mathbf{D}_o\right)}{\partial \mathbf{D}_o} = 2\left(\mathbf{D}_o\mathbf{z}^{(i)} - \mathbf{x}^{(i)}\right)\left(\mathbf{z}^{(i)}\right)^T \qquad \text{(B.5)}$$

This shows that the assumption that the support of the sparse decomposition does not change when a coefficient of the dictionary slightly varies is sufficient to derive the update rule in (B.1).

B.2 Visualization of the learned weights of the shallow sparse autoencoders

The learned weights of the T-sparse AE and those of the WTA AE are displayed in Figure B.1. The learned dictionary for CoD and that for OMP are also shown. The matrix of encoder weights of the T-sparse AE look almost identical to the transpose of the matrix of decoder weights of the T-sparse AE. Likewise, the matrix of encoder weights of the WTA AE is almost identical to the transpose of the matrix of decoder weights of the WTA AE. The weight features for the shallow sparse autoencoders appear to be more distributed around high spatial frequencies than the dictionary features in the case of the algorithms for sparse decomposition over a dictionary.

B.3 Visualization of the learned dictionaries for OMP and WTA OMP

The learned dictionary for WTA OMP and that for OMP are displayed in Figure B.2. The dictionary features for WTA OMP seem to be better balanced between low and high spatial frequencies than the dictionary features for OMP.

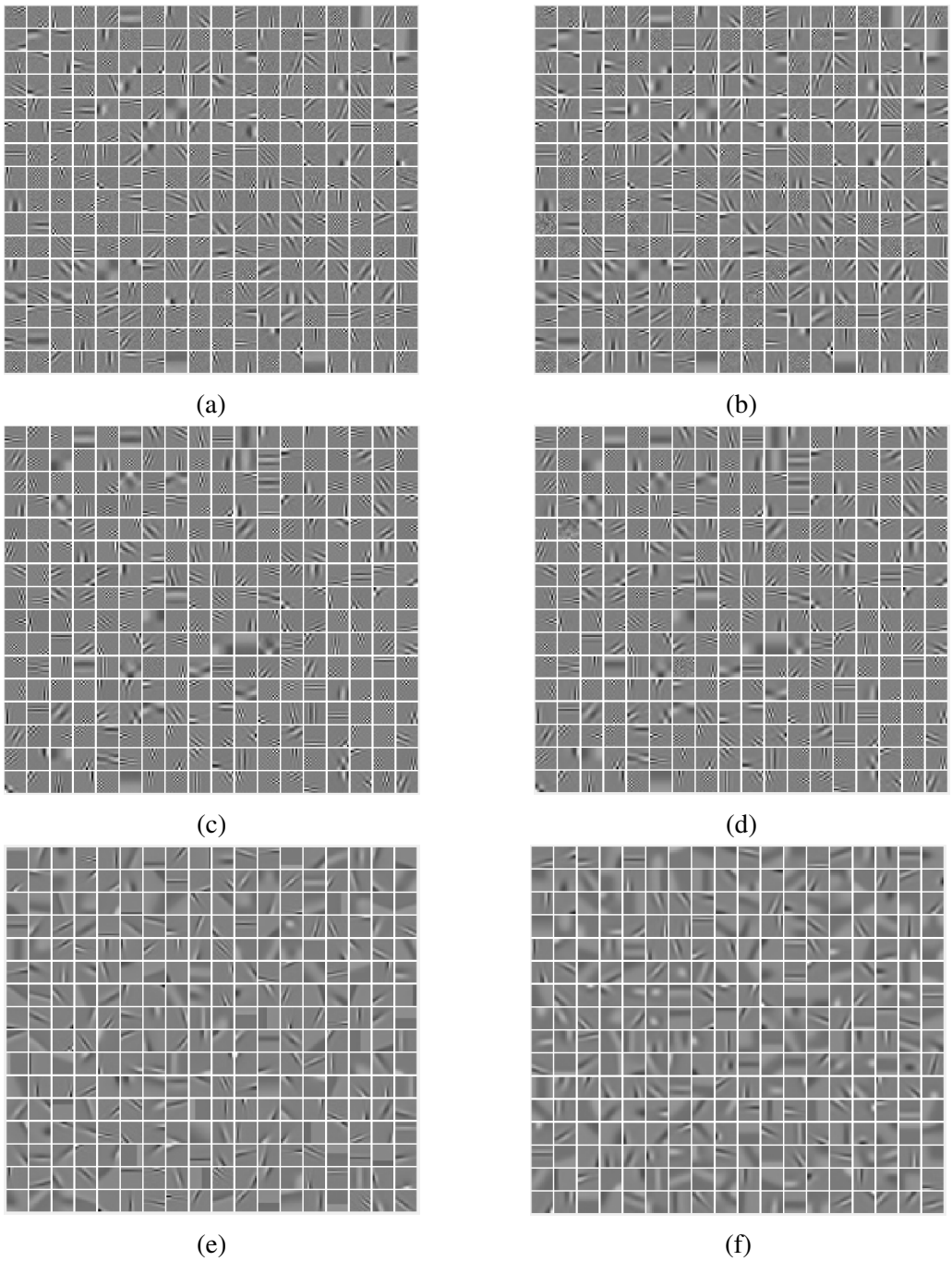
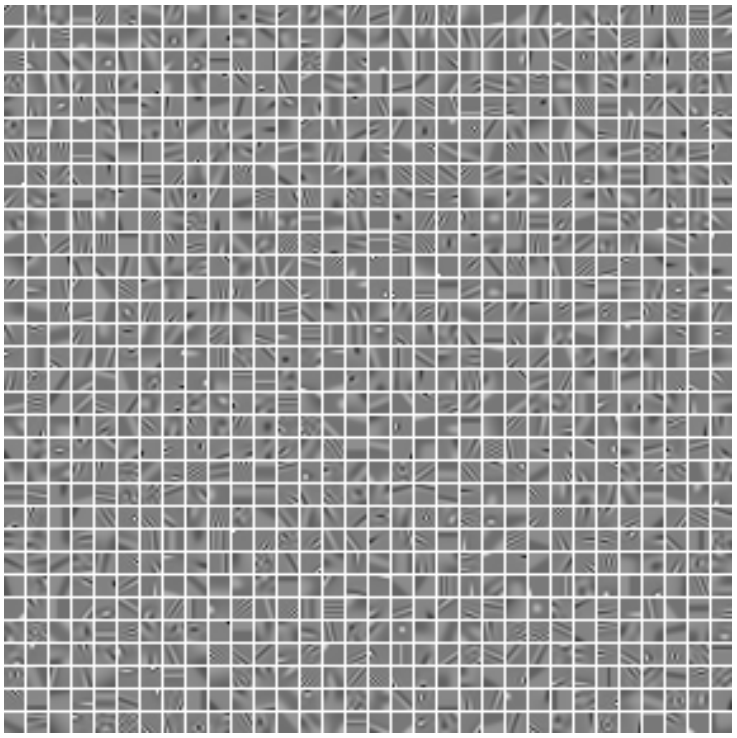
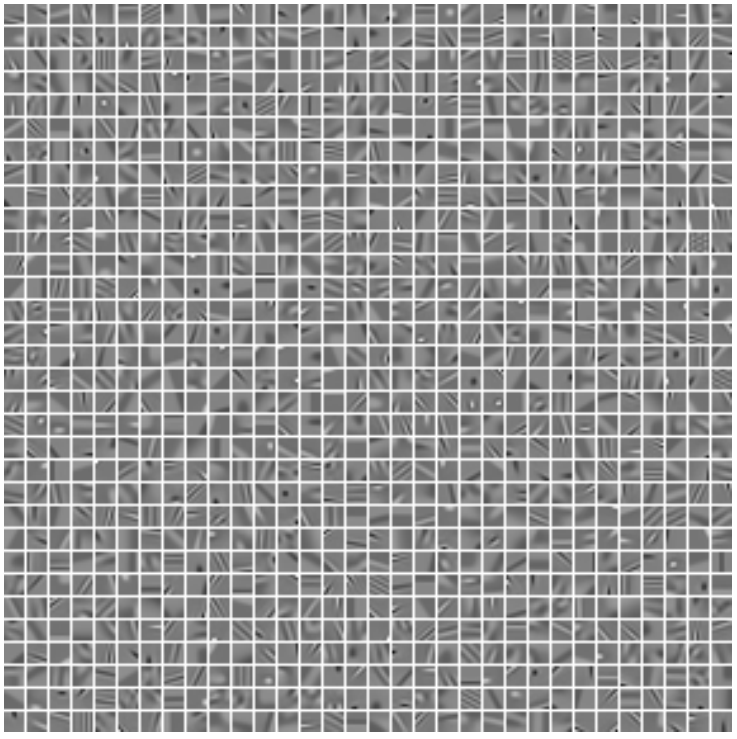


Figure B.1: Visualization of (a) the encoder weights $\mathbf{V}$ of the T-sparse AE, (b) the decoder weights $\mathbf{\Phi}$ of the T-sparse AE, (c) the encoder weights $\mathbf{W}$ of the WTA AE, (d) the decoder weights $\mathbf{U}$ of the WTA AE, (e) the dictionary $\mathbf{D}_\delta$ for CoD, and (f) the dictionary $\mathbf{D}_o$ for OMP. $n = 288$. For $\mathbf{\Phi}$, $\mathbf{U}$, $\mathbf{D}_\delta$, and $\mathbf{D}_o$, a block corresponds to a column, i.e. a dictionary atom in the cases of $\mathbf{D}_\delta$ and $\mathbf{D}_o$, reshaped to $(\sqrt{m}, \sqrt{m})$. For, $\mathbf{V}$ and $\mathbf{W}$, a block corresponds to a row reshaped to $(\sqrt{m}, \sqrt{m})$.



(a)



(b)

Figure B.2: Visualization of (a) the dictionary $\mathbf{D}_w$ for WTA OMP and (b) the dictionary $\mathbf{D}_o$ for OMP. $n = 1024$.

Appendix C

C.1 Convolutional architectures for H.265

The architecture of the stack of convolutional layers $g_m^c\left(\,.\,;\boldsymbol{\phi}_m^{c,0}\right)$ that is applied to the context portion $\mathbf{X}_0$ located on the left side of the image block to be predicted is shown in Table C.1 for $m \in \{4, 16, 64\}$. The architecture of the stack of convolutional layers $g_m^c\left(\,.\,;\boldsymbol{\phi}_m^{c,1}\right)$ that is applied to the context porton $\mathbf{X}_1$ located above the image block is identical to that of $g_m^c\left(\,.\,;\boldsymbol{\phi}_m^{c,0}\right)$. The slope of LeakyReLU is equal to 0.1.

The architecture of $g_8^c\left(\,.\,;\boldsymbol{\phi}_8^{c,0}\right)$ is similar to that of $g_4^c\left(\,.\,;\boldsymbol{\phi}_4^{c,0}\right)$ but, in the first layer, the filter size is $5 \times 5 \times 1$, the number of filters is 64, and the stride is 2. Moreover, in the second layer, the filter size is $3 \times 3 \times 64$ and the number of filters is 64. The architecture of $g_{32}^c\left(\,.\,;\boldsymbol{\phi}_{32}^{c,0}\right)$ is similar to that of $g_{64}^c\left(\,.\,;\boldsymbol{\phi}_{64}^{c,0}\right)$ but, in the third layer, the filter size is $3 \times 3 \times 128$, the number of filters is 128, and the stride is 1. Moreover, in the fourth layer, the filter size is $5 \times 5 \times 128$ and the number of filters is 256. In the fifth layer, the filter size is $3 \times 3 \times 256$ and the number of filters is 256.

To obtain the architecture of g_m^t, each sequence of layers in g_m^c is reversed. Besides, each convolution is replaced by a transposed convolution (see Table C.1).

C.2 Number of trainable parameters

It is interesting to count the number of trainable parameters in each neural network of PNNS to highlight how the number of trainable parameters evolves with the width m of the image block to be predicted.

Fully-connected neural network for $\mathbf{m = 4}$ The number of weights in a fully-connected layer is equal to the number of input neurons times the number of output neurons. The number of biases in a fully-connected layer is equal to the number of output neurons. Therefore, the number of trainable parameters in the fully-connected architecture in Figure 5.2 is equal to $5m^2p + p + p^2 + p + p^2 + p + pm^2 + m^2 = m^2\left(6p + 1\right) + 2p^2 + 3p$.

For $m = 4$, $p = 1200$, this gives 2998816 trainable parameters.

Fully-connected neural network for $\mathbf{m = 8}$ For $m = 8$, $p = 1200$, the number of trainable parameters is equal to 3344464.

Table C.1: Architectures of the stack of convolutional layers $g_m^c\left(\,.\,;\boldsymbol{\phi}_m^{c,0}\right)$ that is applied to the context portion $\mathbf{X}_0$ located on the left side of the image block to be predicted ((a) $m = 4$, (c) $m = 16$, and (e) $m = 64$) and the stack of transposed convolutional layers $g_m^t\left(\,.\,;\boldsymbol{\phi}_m^t\right)$ ((b) $m = 4$, (d) $m = 16$, and (f) $m = 64$). *conv* means convolution and *tconv* means transposed convolution.

Layer number	Layer type	Filter size	Number of filters	Stride	Non-linearity
1	conv	$3 \times 3 \times 1$	32	1	LeakyReLU
2	conv	$3 \times 3 \times 32$	32	1	LeakyReLU

(a)

Layer number	Layer type	Filter size	Number of filters	Stride	Non-linearity
1	tconv	$3 \times 3 \times 32$	32	1	LeakyReLU
2	tconv	$3 \times 3 \times 32$	1	1	-

(b)

Layer number	Layer type	Filter size	Number of filters	Stride	Non-linearity
1	conv	$5 \times 5 \times 1$	64	2	LeakyReLU
2	conv	$3 \times 3 \times 64$	64	1	LeakyReLU
3	conv	$5 \times 5 \times 64$	128	2	LeakyReLU
4	conv	$3 \times 3 \times 128$	128	1	LeakyReLU

(c)

Layer number	Layer type	Filter size	Number of filters	Stride	Non-linearity
1	tconv	$3 \times 3 \times 128$	128	1	LeakyReLU
2	tconv	$5 \times 5 \times 128$	64	2	LeakyReLU
3	tconv	$3 \times 3 \times 64$	64	1	LeakyReLU
4	tconv	$5 \times 5 \times 64$	1	2	-

(d)

Layer number	Layer type	Filter size	Number of filters	Stride	Non-linearity
1	conv	$5 \times 5 \times 1$	64	2	LeakyReLU
2	conv	$5 \times 5 \times 64$	128	2	LeakyReLU
3	conv	$5 \times 5 \times 128$	256	2	LeakyReLU
4	conv	$5 \times 5 \times 256$	512	2	LeakyReLU
5	conv	$3 \times 3 \times 512$	512	1	LeakyReLU

(e)

Layer number	Layer type	Filter size	Number of filters	Stride	Non-linearity
1	tconv	$3 \times 3 \times 512$	512	1	LeakyReLU
2	tconv	$5 \times 5 \times 512$	256	2	LeakyReLU
3	tconv	$5 \times 5 \times 256$	128	2	LeakyReLU
4	tconv	$5 \times 5 \times 128$	64	2	LeakyReLU
5	tconv	$5 \times 5 \times 64$	1	2	-

(f)

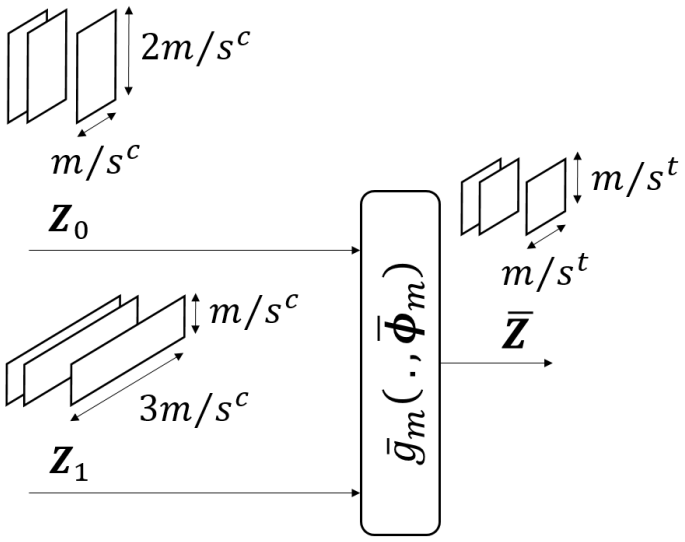


Figure C.1: Illustration of the dimensions of the two input stacks of feature maps to the merger and the dimensions of the output stack of feature maps.

Convolutional neural network for $\mathbf{m = 16}$ The number of weights in a convolutional layer is equal to the number of input feature maps times the number of output feature maps times the height of the convolutional kernels times their width. The number of biases in a convolutional layer is equal to the number of output feature maps.

For the merger denoted $\bar{g}_m$ in Figure 5.3, the counting of the number of trainable parameters is not straightforward. First, it is essential to stress that the height of each feature map in the output stack of feature map $\overline{\mathbf{Z}}$ is equal to $m \,/\, s^t$, where s^t is the product of the convolutional strides in the stack of transpose convolutional layers g_m^t. This is because the application of g_m^t to $\overline{\mathbf{Z}}$ provides a prediction of the image block to be predicted of size $m \times m$. Likewise, the width of each feature map in $\overline{\mathbf{Z}}$ is equal to $m \,/\, s^t$. Now, let s^c be the product of the convolutional strides in the stack of convolutional layers g_m^c. Note that $s^c = s^t$. Figure C.1 summarizes the dimensions of the two input stacks of feature maps to the merger and the dimensions of the output stack of feature maps. Now, as each input feature map to the merger is fully-connected to a different output feature map, the number of weights connecting a pair input-output feature maps is equal to $5m^4 \,/\, (s^c)^4$. For this pair input-output feature maps, $m^2 \,/\, (s^c)^2$ biases are involved.

For $m = 16$, given the architecture displayed in Tables C.1c and C.1d, the number of trainable parameters in the merger alone is equal to 165888. The total number of trainable parameters is equal to 947968.

Convolutional neural network for $\mathbf{m = 32}$ The number of trainable parameters in the merger alone is equal to 331776. The total number of trainable parameters is equal to 3858944.

Convolutional neural network for $\mathbf{m = 64}$ Given the architecture displayed in Tables C.1e anf C.1f, the number of trainable parameters in the merger alone is equal to 663552. The total number of trainable parameters is equal to 13989376.

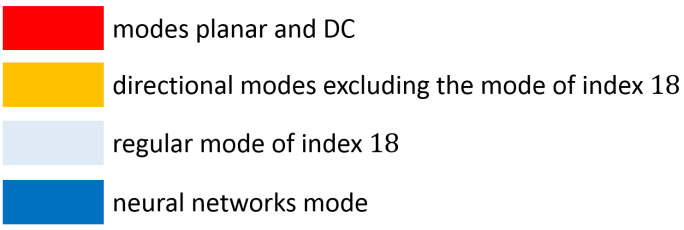
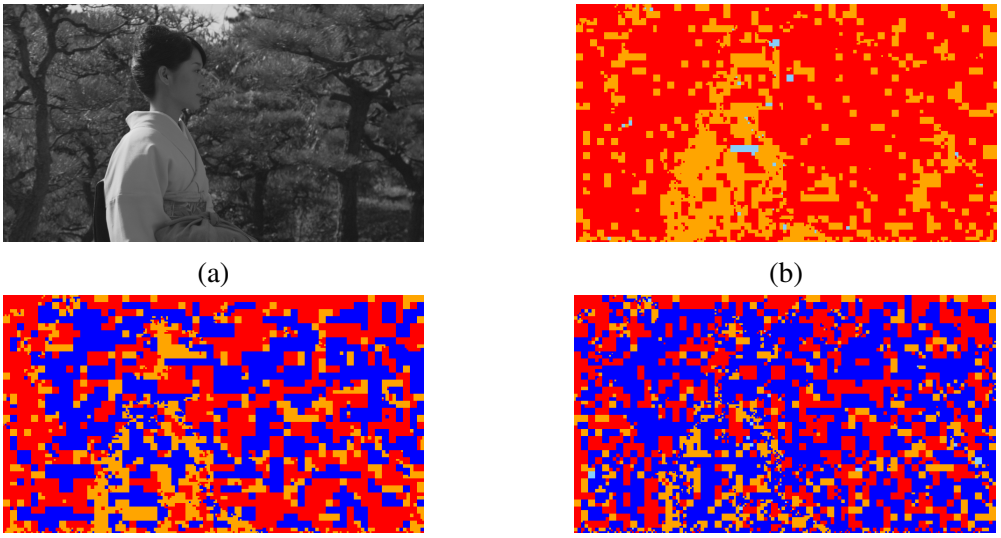


Figure C.2: Color legend for coloring each PB depending on its selected intra prediction mode.

Figure C.3: Comparison of the three maps of intra prediction modes: (a) 832×480 luminance channel of the first frame of BQMall, (b) map of intra prediction modes via H.265, (c) map of intra prediction modes via *PNNS substitution*, and (d) map of intra prediction modes via *PNNS switch*. For the three encodings, QP $= 32$.

C.3 Selected intra prediction mode for each PB

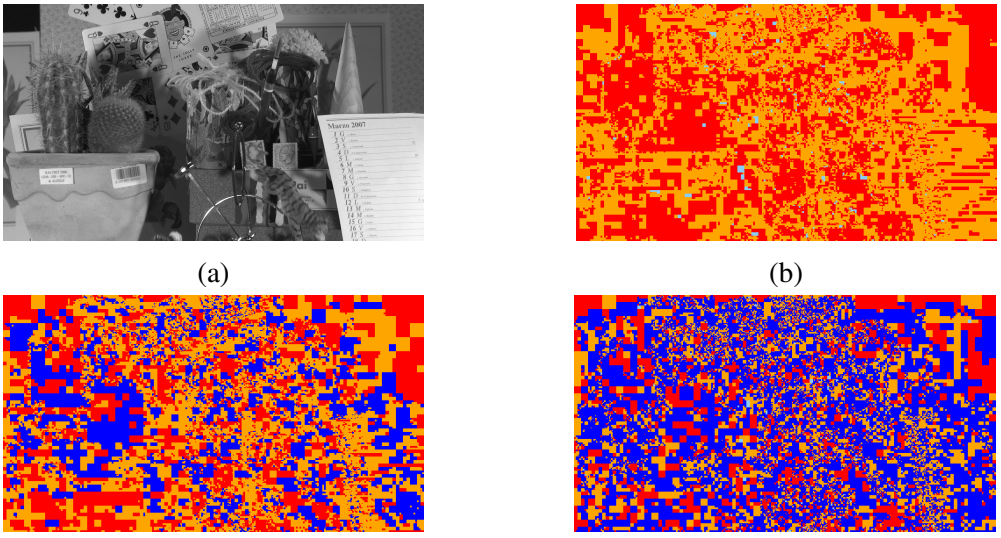
For a given image channel encoded by a given variant of H.265, let us define the map of intra prediction modes as the image obtained by coloring each PB using the color associated to its selected intra prediction mode. The legend for associating each intra prediction mode to a color is explained in Figure C.2. Figures C.3 to C.9 compare the three maps of intra prediction modes obtained via H.265, *PNNS substitution*, and *PNNS switch* respectively for a different luminance image to be encoded. Note that, for *PNNS substitution*, the color of the regular intra prediction mode of index 18 never appears in the maps of intra prediction modes as this mode is replaced by the neural networks mode.



(a) (b)

(c) (d)

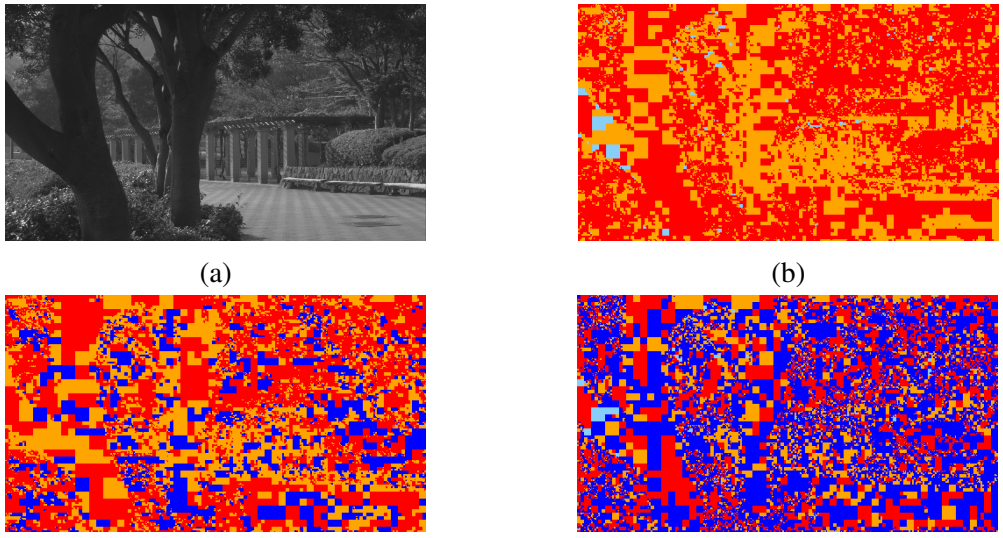
Figure C.4: Comparison of the three maps of intra prediction modes: (a) 1920×1080 luminance channel of the first frame of Kimono, (b) map of intra prediction modes via H.265, (c) map of intra prediction modes via *PNNS substitution*, and (d) map of intra prediction modes via *PNNS switch*. For the three encodings, QP $= 32$.



(a) (b)

(c) (d)

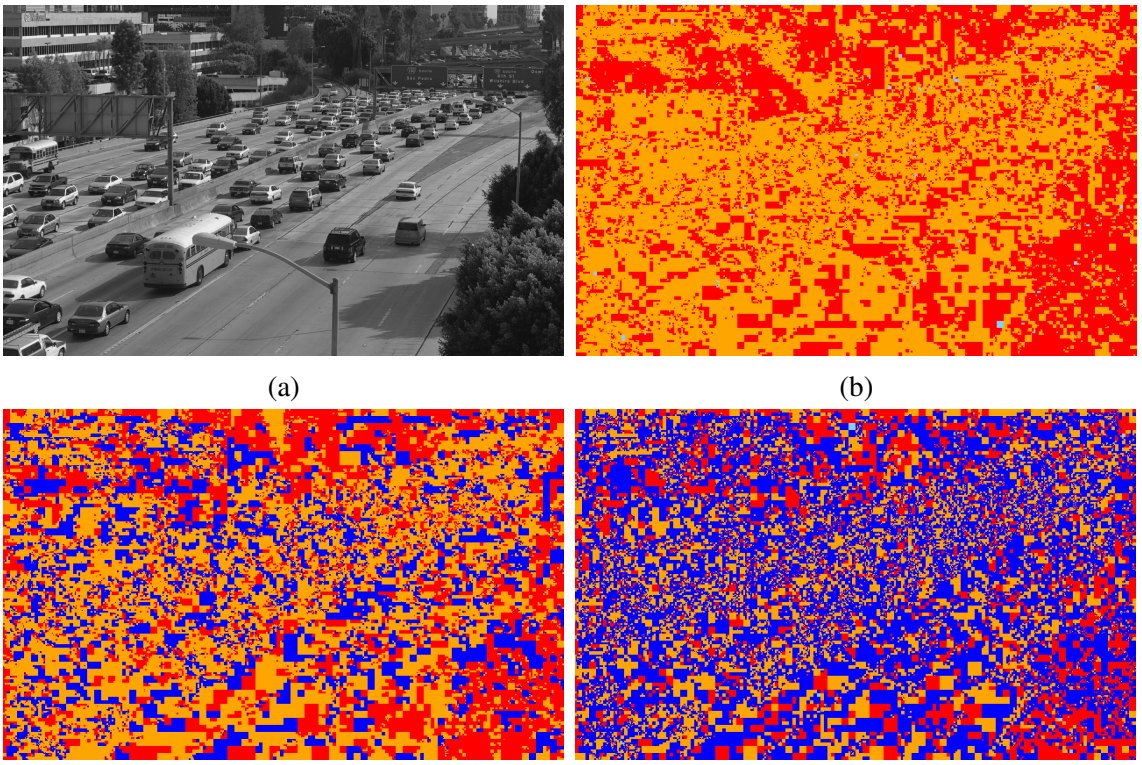
Figure C.5: Comparison of the three maps of intra prediction modes: (a) 1920×1080 luminance channel of the first frame of Cactus, (b) map of intra prediction modes via H.265, (c) map of intra prediction modes via *PNNS substitution*, and (d) map of intra prediction modes via *PNNS switch*. For the three encodings, QP $= 32$.



(a) (b)

(c) (d)

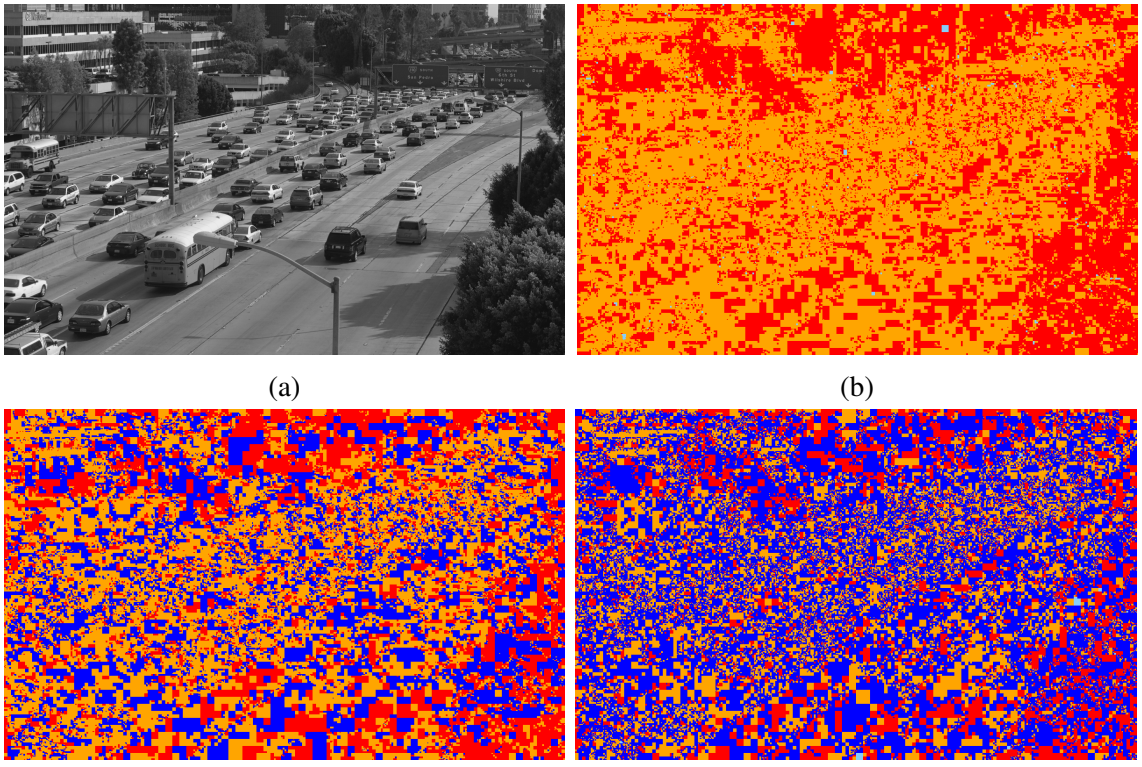
Figure C.6: Comparison of the three maps of intra prediction modes: (a) 1920×1080 luminance channel of the first frame of ParkScene, (b) map of intra prediction modes via H.265, (c) map of intra prediction modes via *PNNS substitution*, and (d) map of intra prediction modes via *PNNS switch*. For the three encodings, QP $= 32$.



(a) (b)

(c) (d)

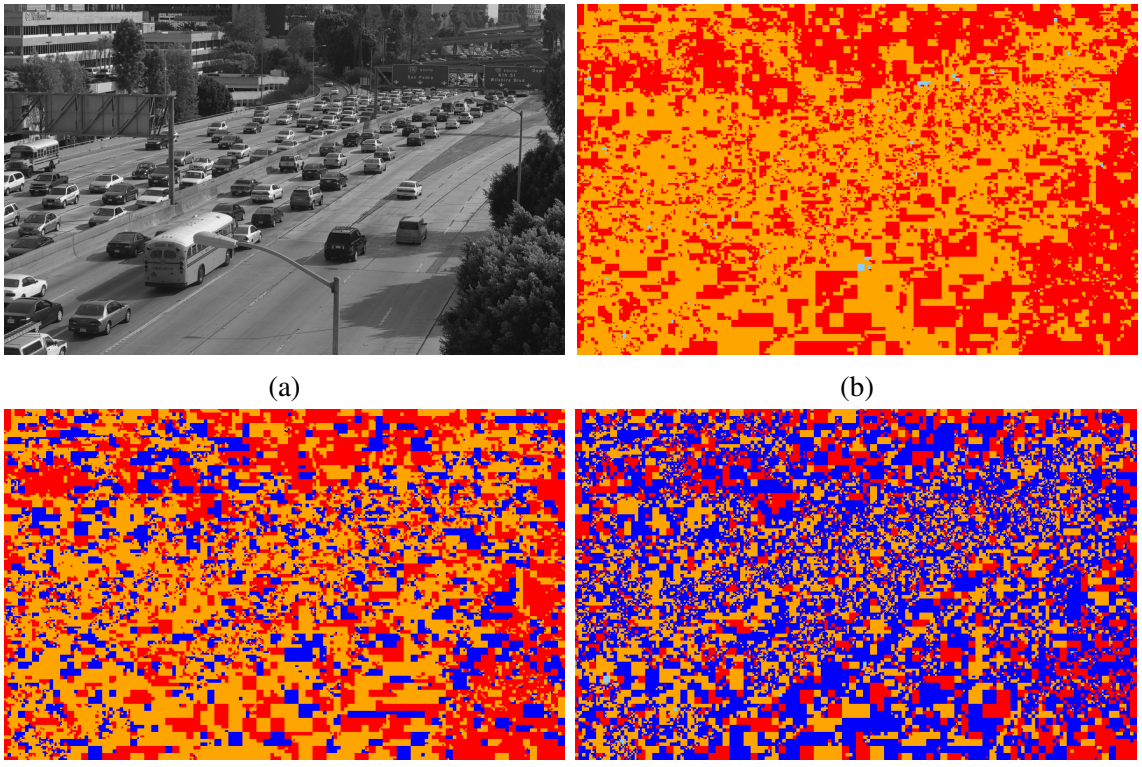
Figure C.7: Comparison of the three maps of intra prediction modes: (a) 2560×1600 luminance channel of the first frame of Traffic, (b) map of intra prediction modes via H.265, (c) map of intra prediction modes via *PNNS substitution*, and (d) map of intra prediction modes via *PNNS switch*. For the three encodings, QP $= 32$.



(a) (b)

(c) (d)

Figure C.8: Same comparison as in Figure C.7 but QP $= 27$.



(a) (b)

(c) (d)

Figure C.9: Same comparison as in Figure C.7 but QP $= 37$.

Bibliography

[1] Allen Gersho. Asymptotically optimal block quantization. *IEEE Transactions on Information Theory*, vol. 25, no. 4:373–380, July 1979.

[2] Herbert Gish and John N. Pierce. Asymptotically efficient quantizing. *IEEE Transactions on Information Theory*, vol. 14, no. 5:676–683, September 1968.

[3] Robert M. Gray and David L. Neuhoff. Quantization. *IEEE Transactions on Information Theory*, vol. 44, no. 6:2325–2383, October 1998.

[4] Jacob Ziv. On universal quantization. *IEEE Transactions on Information Theory*, vol. 31, no. 3:344–347, May 1985.

[5] Michael W. Marcellin, Margaret A. Lepley, Ali Bilgin, Thomas J. Flohr, Troy T. Chinen, and James H. Kasner. An overview of quantization in JPEG2000. *Signal Processing: Image Communication*, vol. 17, issue 1:73–84, January 2002.

[6] Heiko Schwarz and Thomas Wiegand. Video coding: part II of fundamentals of source and video coding. *Foundations and Trends in Signal Processing*, vol. 10, no.1-3:1–346, December 2016.

[7] Majid Rabbani and Rajan Joshi. An overview of the JPEG 2000 still image compression standard. *Signal Processing: Image Communication*, vol. 17, issue 1:3–48, January 2002.

[8] Detlev Marpe, Heiko Schwarz, and Thomas Wiegand. Context-based adaptive binary arithmetic coding in the H.264/AVC video compression standard. *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7:620–636, July 2003.

[9] Martin Vetterli and Jelena Kovačević. Wavelets and subband coding. *Englewood Cliffs, NJ: Prentice Hall*, 1995.

[10] Thomas Wiegand and Heiko Schwarz. Source coding: part I of fundamentals of source and video coding. *Foundations and Trends in Signal Processing*, vol. 4, no. 1:1–222, January 2011.

[11] Pedram Pad and Michael Unser. Optimality of operator-like wavelets for representing sparse AR(1) processes. *IEEE Transactions on Signal Processing*, vol. 63, issue 18:4827–4837, September 2015.

[12] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control Signal Systems*, vol. 2, no. 4:303–314, 1989.

[13] Shiyu Liang and R. Srikant. Why deep neural networks for function approximation. In *ICLR*, 2017.

[14] Ronen Eldan and Ohad Shamir. The power of depth for feedforward neural networks. In *Conference On Learning Theory*, 2016.

[15] Jani Lainema, Frank Bossen, Woo-Jin Han, Junghye Min, and Kemal Ugur. Intra coding of the HEVC standard. *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12:1792–1801, December 2012.

[16] Anthony J. Bell and Terrence J. Sejnowski. Edges are the independent components of natural scenes. In *NIPS*, 1996.

[17] J. H. van Hateren and A. van der Schaaf. Independent component filters of natural images compared with simple cells in primary visual cortex. *Proc. R. Soc. Lond. B*, vol. 265, no. 1394:359–366, March 1998.

[18] Honglak Lee, Chaitanya Ekanadham, and Andrew Y. Ng. Sparse deep belief net model for visual area V2. In *NIPS*, 2007.

[19] Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *ICML*, 2009.

[20] Haruo Hosoya and Aapo Hyvärinen. A hierarchical statistical model of natural images explains tuning properties in V2. *The Journal of Neuroscience*, vol. 35, no. 29:10412–10428, July 2015.

[21] Marc'Aurelio Ranzato, Volodymyr Mnih, Joshua M. Susskind, and Geoffrey E. Hinton. Modeling natural images using gated MRFs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 9:2206–2222, September 2013.

[22] Lucas Theis and Matthias Bethge. Generative image modeling using spatial LSTMs. In *NIPS*, 2015.

[23] Aäron van der Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. In *ICML*, 2016.

[24] Touradj Ebrahimi, Karel Fliegel, Antonio Pinheiro, Martin Rerabek, Thomas Richter, and Thanos Skodras. Overview and benchmarking summary for the ICIP 2016 compression challenge. In *ICIP*, 2016.

[25] M.P. Sharabayko and N.G. Markov. Contemporary video compression standards: H.265/HEVC, VP9, vp10, daala. In *SIBCON*, 2016.

[26] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010.

[27] Ronan Boitard, Mahsa T. Pouzarad, Panos Nasiopoulos, and Jim Slevinsky. Demystifying high-dynamic range technology: a new evolution in digital media. *IEEE Consumer Electronics Magazine*, vol. 4, no. 4:72–86, October 2015.

[28] R.W.G. Hunt. *The reproduction of colour*. Wiley-IS&T Series in Imaging Science and Technology, 6^{th} Edition, November 2004.

[29] CIE. *CIE Proceedings 1924*. Cambridge University Press, Cambridge, 1926.

[30] W. D. Wright. A re-determination of the trichromatic coefficients of the spectral colours. *Transactions of the Optical Society*, vol. 30, no. 4:141–164, March 1929.

[31] J. Guild. The colorimetric properties of the spectrum. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 230:149–187, January 1932.

[32] CIE. *CIE Proceedings 1931*. Cambridge University Press, Cambridge, 1932.

[33] T. Smith and J. Guild. The C.I.E colorimetric standards and their use. *Transactions of the Optical Society*, vol. 33, no. 3:73–134, January 1932.

[34] Hugh S. Fairman, Michael H. Brill, and Henry Hemmendinger. How the CIE 1931 color-matching functions were derived from wright-guild data. *Color Research & Application*, vol. 22, no. 1:11–23, February 1997.

[35] Charles Poynton. *Digital video and HD: algorithms and interfaces*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2nd edition, 2012.

[36] J. C. Stevens and S. S. Stevens. Brightness function: effects of adaptation. *Journal of the Optical Society of America*, vol. 53, no. 3:375–385, March 1963.

[37] Brian A. Wandell. *Foundations of vision*. Sinauer Associates, 1995.

[38] Christian J. van den Branden Lambrecht. *Vision models and applications to image and video processing*. Springer US, 2001.

[39] Gary J. Sullivan, Jens-Rainer Ohm, Woo-Jin Han, and Thomas Wiegand. Overview of the High Efficiency Video Coding (HEVC) Standard. *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, issue 12:1649–1667, December 2012.

[40] Jean Bégaint, Franck Galpin, Philippe Guillotel, and Christine Guillemot. Region-based models for motion compensation in video compression. In *PCS*, 2018.

[41] Vivienne Sze, Madhukar Budagavi, and Gary J. Sullivan. *High Efficiency Video Coding (HEVC): algorithms and architectures*. Springer International Publishing, 1st edition, 2014.

[42] Mathias Wien. High efficiency video coding: coding tools and specification. *Springer*, September 2014.

[43] Andrey Norkin, Gisle Bjontegaard, Arild Fuldseth, Matthias Narroschke, Masaru Ikeda, Kenneth Andersson, Minhua Zhou, and Geert Van der Auwera. HEVC deblocking filter. *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, issue 12:1746–1754, December 2012.

[44] Chih-Ming Fu, Elena Alshina, Alexander Alshin, Yu-Wen Huang, Ching-Yeh Chen, Chia-Yang Tsai, Chih-Wei Hsu, Shaw-Min Lei, Jeong-Hoon Park, and Woo-Jin Han. Sample adaptive offset in the HEVC standard. *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, issue 12:1755–1764, December 2012.

[45] Rao Kamisetty Ramamohan and Yip Ping. Discrete cosine transform: algorithms, advantages, applications. *Academic Press*, 1990.

[46] Yunfei Zheng, Muhammed Coban, Xianglin Wang, Joel Sole, Rajan Joshi, and Marta Karczewicz. CE11: mode dependent coefficient scanning. *JCTVC-D393, 4th JCTVC Meeting, Daegu*, January 2011.

[47] Joel Sole, Rajan Joshi, and Marta Karczewicz. CE11: parallel context processing for the signifance map in high coding efficiency. *JCTVC-E338, 5th JCTVC Meeting, Geneva*, March 2011.

[48] Joel Sole, Rajan Joshi, and Marta Karczewicz. CE11: unified scans for the significance map and coefficient level coding in high efficiency. *JCTVC-F288,* 6^{th} *JCTVC Meeting, Torino*, July 2011.

[49] Joel Sole, Rajan Joshi, and Marta Karczewicz. CE11: scanning passes of residual data in HE. *JCTVC-G320,* 7^{th} *JCTVC Meeting, Geneva*, November 2011.

[50] Nguyen Nguyen, Tianying Ji, Dake He, Gäelle Martin-Cocher, and Lin Song. Multi-level significant maps for large transform units. *JCTVC-G644,* 7^{th} *JCTVC Meeting, Geneva*, November 2011.

[51] Gordon Clare, Félix Henry, and Joël Jung. Sign data hiding. *JCTVC-G271,* 7^{th} *JCTVC Meeting, Geneva*, November 2011.

[52] Wei-Jung Chien, Joel Sole, and Marta Karczewicz. Last position coding for CABAC. *JCTVC-G704,* 7^{th} *JCTVC Meeting, Geneva*, November 2011.

[53] Tung Nguyen. CE11: coding of transform coefficient levels with Golomb-Rice codes. *JCTVC-E253,* 5^{th} *JCTVC Meeting, Geneva*, March 2011.

[54] Recommendation P.910. Subjective video quality assessment methods for multimedia applications. Technical report, International Telecommunication Union, 2008.

[55] Gisle Bjontegaard. Calculation of average PSNR differences between RD-curves. Technical report, ITU-T SG16/Q6, Austin, TX, USA, 2001.

[56] Geoffrey E. Hinton. A pratical guide to training restricted boltzmann machines. Technical report, UTML2010-003, University of Toronto, August 2010.

[57] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, vol. 9, no. 8:1735–1780, December 1997.

[58] James J. DiCarlo, Davide Zoccolan, and Nicole C. Rust. How does the brain solve visual object recognition? *Neuron*, vol. 73, issue 3:415–434, February 2012.

[59] McCulloch Warren and Walter Pitts. A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, vol. 5, issue 4:115Ű133, December 1943.

[60] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, vol. 65, issue 6:386Ű408, November 1958.

[61] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

[62] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *CVPR*, 2016.

[63] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, vol. 86, no. 11:2278–2324, November 1998.

[64] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTAT*, 2010.

[65] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: surpassing human-level performance on imagenet classification. In *ICCV*, 2015.

[66] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[67] Paul Werbos. *Beyond regression: new tools for prediction and analysis in the behavioral sciences*. PhD thesis, Harvard University, January 1974.

[68] David E. Rumelhart and James L. McClelland. *Parallel distributed processing: explorations in the microstructure of cognition, vol. 1*. Foundations, MIT Press, 1986.

[69] Léon Bottou. Online learning and stochastic approximations. *Online Learning in Neural Networks*, pages 9–42, Cambridge University Press, 1998.

[70] Léon Bottou. Large scale machine learning with stochastic gradient descent. *Proceedings of COMPSTAT*, pages 177–186, 2010.

[71] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Lecture 6a, overview of mini-batch gradient descent. http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

[72] Yoshua Bengio. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, vol. 2, no. 1:1–127, 2009.

[73] Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio. Contracting auto-encoders: explicit invariance during feature extraction. In *ICML*, 2011.

[74] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. In *ICLR*, 2017.

[75] Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. *Neural Networks: Tricks of the Trade*, Springer, 2013.

[76] Stëphane Mallat. Understanding deep convolutional networks. *Philosophical Transactions of the Royal Society A*, 374(2065):20150203, March 2016.

[77] Serge Ioffe and Christian Szegedy. Batch normalization: accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

[78] Bee Lim, Sanghyun Son, Heewon Kim, Seungjun Nah, and Kyoung Mu Lee. Enhanced deep residual networks for single image super-resolution. In *CVPR Workshops*, 2017.

[79] George Toderici, Sean O'Malley, Sung Jin Hwang, Damien Vincent, David Minnen, Shumeet Baluja, Michele Covell, and Rahul Sukthankar. Variable rate image compression with recurrent neural networks. In *ICLR*, 2016.

[80] Vincent Dumoulin and Francesco Visin. A guide to convolutional arithmetic for deep learning. *arXiv:1603.07285v2*, January 2018.

[81] Thierry Dumas, Aline Roumy, and Christine Guillemot. Image compression with stochastic winner-take-all auto-encoder. In *ICASSP*, 2017.

[82] Lucas Theis, Wenzhe Shi, Andrew Cunningham, and Ferenc Huszár. Lossy image compression with compressive autoencoders. In *ICLR*, 2017.

[83] Johannes Ballé, Valero Laparra, and Eero P. Simoncelli. End-to-end optimized image compression. In *ICLR*, 2017.

[84] George Toderici, Damien Vincent, Nick Johnston, Sung Jin Hwang, David Minnen, and Joel Shor. Full resolution image compression with recurrent neural networks. In *CVPR*, 2017.

[85] Nick Johnston, Damien Vincent, David Minnen, Michele Covell, Saurabh Singh, Troy Chinen, and Sung Jin Hwang. Improved lossy image compression with priming and spatially adaptive bit rates for recurrent networks. *arXiv preprint arXiv:1703.10114v1*, March 2017.

[86] Thierry Dumas, Aline Roumy, and Guillemot Christine. Autoencoder based image compression: can the learning be quantization independent? In *ICASSP*, 2018.

[87] Eirikur Agustsson, Fabian Mentzer, Michael Tschannen, Lukas Cavigelli, Radu Timofte, Luca Benini, and Luc Van Gool. Soft-to-hard vector quantization for end-to-end learning compressible representations. In *NIPS*, 2017.

[88] Andrea Karpathy and Li Fei-Fei. Deep visual semantic alignments for generating image descriptions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 4:664–676, April 2017.

[89] Johannes Ballé, David Minnen, Saurabh Singh, Sung Jin Hwang, and Nick Johnston. Variational image compression with a scale hyperprior. In *ICLR*, 2018.

[90] Oren Rippel and Lubomir Bourdev. Real-time adaptive image compression. In *ICML*, 2017.

[91] Fabian Mentzer, Eirikur Agustsson, Michael Tschannen, Radu Timofte, and Luc Van Gool. Conditional probability models for deep image compression. In *CVPR*, 2018.

[92] Karol Gregor, Frederic Besse, Danilo J. Rezende, Ivo Danihelka, and Daan Wierstra. Towards conceptual compression. *arXiv preprint arXiv:1604.08772*, April 2016.

[93] David Minnen, Johannes Ballé, and George Toderici. Joint autoregressive and hierarchical priors for learned image compression. In *arXiv:1809.02736v1*, September 2018.

[94] Mateus Grellert, Guilherme Correa, Bruno Zatt, Sergio Bampi, and Luis A. da Silva Cruz. Learning-based complexity reduction and scaling for HEVC encoders. In *ICASSP*, 2018.

[95] Xiaolin Shen and Lu Yu. CU splitting early termination based on weighted SVM. *EURASIP Journal on Image and Video Processing*, vol. 2013, no. 4, December 2013.

[96] Tianyi Li, Mai Xu, and Xin Deng. A deep convolutional neural network approach for complexity reduction on intra-mode HEVC. In *ICME*, 2017.

[97] Riu Song, Dong Liu, Houqiang Li, and Feng Wu. Neural network-based arithmetic coding of intra prediction modes in HEVC. In *VCIP*, 2017.

[98] Yuanying Dai, Dong Liu, and Feng Wu. A convolutional neural network approach for post-processing in HEVC intra coding. *Proceedings of the 23rd International Conference on Multimedia Modeling*, pages 28–39, January 2017.

[99] Tingting Wang, Mingjin Chen, and Hongyang Chao. A novel deep learning-based method for improving coding efficiency from the decoder-end for HEVC. In *DCC*, 2017.

[100] Ogün Kirmemiş, Gonca Bakar, and A. Murat Tekalp. Learned compression artifact removal by deep residual networks. In *CVPR*, 2018.

[101] Leonardo Galteri, Lorenzo Seidenari, Marco Bertini, and Alberto Del Bimbo. Deep generative adversarial compression artifact removal. In *ICCV*, 2017.

[102] Jiahao Li, Bin Li, Jizheng Xu, Ruiqin Xiong, and Wen Gao. Fully-connected network-based intra prediction for image coding. *IEEE Transactions on Image Processing*, vol. 27, no. 7:3236–3247, July 2018.

[103] Thierry Dumas, Aline Roumy, and Christine Guillemot. Context-adaptive neural network based intra prediction for image compression. *submitted to Transactions on Image Processing, arXiv:1807.06244v1*, July 2018.

[104] Kyle K. Herrity, Anna C. Gilbert, and Joel A. Tropp. Sparse approximation via iterative thresholding. In *ICASSP*, 2006.

[105] Gagan Rath and Christine Guillemot. A complementary matching pursuit algorithm for sparse approximation. In *EUSIPCO*, 2008.

[106] B. K. Natarajan. Sparse approximate solutions to linear systems. *SIAM J. Comput.*, vol. 24, no. 2:227–234, April 1995.

[107] Geoffrey Davis, Stéphane Mallat, and Marco Avellaneda. Adaptive greedy approximations. *Constructive Approximation*, vol. 13, issue 1:57–98, March 1997.

[108] Junyuan Xie, Linli Xu, and Enhong Chen. Image denoising and inpainting with deep neural networks. In *NIPS*, 2012.

[109] Forest Agostinelli, Michael R. Anderson, and Honglak Lee. Adaptive multi-column deep neural networks with application to robust image denoising. In *NIPS*, 2013.

[110] M. R. Osborne, Brett Presnell, and B. A. Turlach. A new approach to variable selection in least squares problems. *IMA Journal of Numerical Analysis*, 20, no. 3:389–404, July 2000.

[111] YingYing Li and Stanley Osher. Coordinate descent optimization for l1 minimization with application to compressed sensing; a greedy algorithm. *Inverse Problem and Imaging*, vol. 3, no. 3:487–503, 2009.

[112] Joel A. Tropp. Greed is good: algorithmic results for sparse approximation. *IEEE Transactions on Information Theory*, vol. 50, no. 10:2231–2242, October 2004.

[113] Alireza Makhzani and Brendan Frey. k-sparse autoencoders. In *ICLR*, 2014.

[114] Alireza Makhzani and Brendan Frey. A winner-take-all method for training sparse convolutional autoencoders. In *NIPS*, 2015.

[115] Greg Griffin, Alex Holub, and Pietro Perona. Caltech-256 object category dataset. Technical report, California Institute of Technology, 2007.

[116] Léon Bottou, Franck E. Curtis, and Jorge Nocedal. Optimization methods for large scale machine learning. *arXiv preprint arXiv:1606.04838, 2016*.

[117] Julien Mairal, Francis Bach, Jean Ponce, and Guillermo Sapiro. Online learning for matrix factorization and sparse coding. *JMLR*, 11:19–60, March 2010.

[118] Ori Bryt and Michael Elad. Compression of facial images using the K-SVD algorithm. *Journal of Visual Communication and Image Representation*, vol. 19, no. 4:270–283, May 2008.

[119] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *ICML*, 2013.

[120] Yann LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. Efficient backprop. *Neural Networks: Tricks of the Trade*, Springer, 1998.

[121] Joaquin Zepeda, Christine Guillemot, and Ewa Kijak. Image compression using sparse representations and the iteration-tuned and aligned dictionary. *IEEE Journal of Selected Topics in Signal Processing*, vol. 5, no. 5:1061–1073, September 2011.

[122] Adam Coates and Andrew Y. Ng. The importance of encoding versus training with sparse coding and vector quantization. In *ICML*, 2011.

[123] Bob L. Sturm and Mads G. Christensen. Comparaison of orthogonal matching pursuit implementations. In *EUSIPCO*, 2012.

[124] Bradley Efron, Trevor Hastie, Iain Johnstone, and Robert Tibshirani. Least angle regression. *The Annals of Statistics*, vol. 32, no. 2:407–499, 2004.

[125] Hyeonwoo Hoh, Seunghoon Hong, and Bohyung Han. Learning deconvolution network for semantic segmentation. In *ICCV*, 2015.

[126] Dominik Scherer, Andreas Müller, and Sven Behnke. Evaluation of the pooling operations in convolutional architectures for object recognition. In *ICANN*, 2010.

[127] Matthew Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *ECCV*, 2014.

[128] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: a deep convolutional encoder-decoder architecture for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 12:2481–2495, December 2017.

[129] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li. ImageNet: a large-scale hierarchical image database. In *CVPR*, 2009.

[130] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: convolutional architecture for fast feature embedding. In *ACM MM*, 2014.

[131] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, and Jürgen Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. *S. C. Kremer and J. F. Kolens editions, A Field Guide to Dynamical Recurrent Neural Networks, IEEE Press*, 2001.

[132] Ruslan R. Salakhutdinov and Geoffrey E. Hinton. Semantic hashing. In *SIGIR Workshop on Information Retrieval and Applications of Graphical Models*, 2007.

[133] L. Deng, M. Seltzer, D. Yu, A. Acero, A. Mohamed, and G. Hinton. Binary coding of speech spectrograms using a deep auto-encoder. In *INTERSPEECH*, 2010.

[134] Alex Krizhevsky and Geoffrey E. Hinton. Using very deep autoencoders for content-based image retrieval. In *ESANN*, 2011.

[135] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, vol. 323:533–536, October 1986.

[136] Johannes Ballé, Valero Laparra, and Eero P. Simoncelli. Density modeling of images using a generalized normalization transform. In *ICLR*, 2016.

[137] Gary J. Sullivan. Efficient scalar quantization of exponential and laplacian random variables. *IEEE Transactions on Information Theory*, vol. 42:1365–1374, September 1996.

[138] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: visualizing image classification models and saliency maps. In *ICLR*, 2014.

[139] Anh Nguyen, Jason Yosinki, and Jeff Clune. Deep neural networks are easily fooled: high confidence predictions for unrecognizable images. In *CVPR*, 2015.

[140] Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson. Understanding neural networks through deep visualization. In *ICML*, 2015.

[141] David Martin, Charless Fowlkes, Doron Tal, and Jitendra Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *ICCV*, 2001.

[142] Geoffrey E. Hinton and Ruslan R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, vol. 313, no. 5786:504–507, July 2006.

[143] Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *ICML*, 2013.

[144] Deepak Pathak, Philipp Krähenbühl, Jeff Donahue, Trevor Darell, and Alexei A. Efros. Context encoders: feature learning by inpainting. In *CVPR*, 2016.

[145] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. Image super-resolution using deep convolutional networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, no. 2:295–307, February 2016.

[146] Pauline Luc, Natalia Neverova, Camille Couprie, Jakob Verbeek, and Yann LeCun. Predicting deeper into the future of semantic segmentation. In *ICCV*, 2017.

[147] Simone Meyer, Oliver Wang, Henning Zimmer, Max Grosse, and Alexander Sorkine-Hornung. Phase-based frame interpolation for video. In *CVPR*, 2015.

[148] Simon Niklaus, Long Mai, and Feng Liu. Video frame interpolation via adaptive convolution. In *CVPR*, 2017.

[149] Joost van Amersfoort, Wenzhe Shi, Alejandro Acosta, Francisco Massa, Johannes Totz, Zehan Wang, and Jose Caballero. Frame interpolation with multi-scale deep loss functions and generative adversarial networks. *arXiv:1711.06045v1*, November 2017.

[150] Kodak suite.

[151] Diederik P. Kingma and Jimmy Lei Ba. Adam: a method for stochastic optimization. In *ICLR*, 2015.

[152] Dan Claudiu Ciresan, Ueli Meier, Luca Marcia Gambardella, and Jüergen Schmidhuber. Deep, big, simple neural nets for handwritten digit recognition. *Neural Computation*, vol. 22, no. 12:3207–3220, December 2010.

[153] Dan Ciresan, Ueli Meier, and Jüergen Schmidhuber. Multi-column deep neural networks for image classification. In *CVPR*, 2012.

[154] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[155] Herve Jegou, Matthijs Douze, and Cordelia Schmid. Hamming embedding and weak geometry consistency for large scale image search. In *ECCV*, 2008.

[156] Roman I. Chernyak. Analysis of the intra predictions in H.265/HEVC. *Applied Mathematical Sciences*, vol. 8, no. 148:7389–7408, 2014.

[157] Ons Bougacha, Hassan Kibeya, Nidhameddine Belhadj, Mohamed Ali Ben Ayed, and Nouri Masmoudi. Statistical analysis of intra prediction in HEVC video encoder. In *IPAS*, 2016.

[158] C. Rosewarne, K. Sharman, and D. Flynn. Common test conditions and software reference configurations for HEVC range extensions. Document JCTVC-P1006, 16th meeting, JCT-VC, San Jose, CA, USA, January 2014.

[159] Jan P. Klopp, Yu-Chiang Frank Wang, Shao-Yi Chien, and Liang-Gee Chen. Learning a code space predictor by exploiting intra image dependencies. In *BMVC*, 2018.

[160] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size. *arXiv:1602.07360v4*, November 2016.

[161] Song Han, Huizi Mao, and William J. Dally. Deep compression: compressing deep neural networks with pruning, trained quantization and huffman coding. In *ICLR*, 2016.

[162] Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. Incremental weight quantization: towards lossless CNNs with low precision weights. In *ICLR*, 2017.

[163] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: training neural networks with low precision weights and activations. *Journal of Machine Learning Research*, 18:1–30, April 2018.

[164] Martin Alain, Cheriguin Safa, Christine Guillemot, and Dominique Thoreau. Locally linear embedding methods for inter image prediction. In *ICIP*, 2013.

[165] Jean Bégaint, Dominique Thoreau, Philippe Guillotel, and Mehmet Türkan. Locally-weighted template-matching based prediction for cloud-based image compression. In *DCC*, 2016.

[166] Jiahui Yu, Zhe Lin, Jimei Yang, Xiaohui Shen, Xin Lu, and Thomas S. Huang. Generative image inpainting with contextual attention. In *CVPR*, 2018.

[167] Honglak Lee, Alexis Battle, Rajat Raina, and Andrew Y. Ng. Efficient sparse coding algorithms. In *NIPS*, 2006.

[168] Michal Aharon, Michael Elad, and Alfred Bruckstein. K-SVD: an algorithm for designing overcomplete dictionaries for sparse representation. *IEEE Transactions on Signal Processing*, vol. 54, no. 11:4311–4322, November 2006.

[169] Michael Elad. Sparse and redundant representations: from theory to applications in signal and image processing. *Springer*, 2010.

UNIVERSITE BRETAGNE LOIRE / MATHSTIC

UNIVERSITÉ DE RENNES 1

Titre : apprentissage profond pour la compression d'image

Mots clés : réseaux de neurones profonds, compression d'image, codage par transformée, prédiction intra.

Résumé : Ces vingt dernières années, la quantité d'images et de vidéos transmises a augmenté significativement, ce qui est principalement lié à l'essor de Facebook et Netflix. Même si les capacités de transmission s'améliorent, ce nombre croissant d'images et de vidéos transmises exige des méthodes de compression plus efficaces. Cette thèse a pour but d'améliorer par l'apprentissage deux composants clés des standards modernes de compression d'image, à savoir la transformée et la prédiction intra. Plus précisément, des réseaux de neurones profonds sont employés car ils ont un grand pouvoir d'approximation, ce qui est nécessaire pour apprendre une approximation fidèle d'une transformée optimale (ou d'un filtre de prédiction intra optimal) appliqué à des pixels d'image.
En ce qui concerne l'apprentissage d'une transformée pour la compression d'image via des réseaux de neurones, un défi est d'apprendre une transformée unique qui est efficace en termes de compromis débit-distorsion, à différents débits. C'est pourquoi deux approches sont proposées pour relever ce défi. Dans la première approche, l'architecture du réseau de neurones impose une contrainte de parcimonie sur les coefficients transformés. Le niveau de parcimonie offre un contrôle sur le taux de compression. Afin d'adapter la transformée à différents taux de compression, le niveau de parcimonie est stochastique pendant la phase d'apprentissage. Dans la deuxième approche, l'efficacité en termes de compromis débit-distorsion est obtenue en minimisant une fonction de débit-distorsion pendant la phase d'apprentissage. Pendant la phase de test, les pas de quantification sont progressivement agrandis selon un schéma afin de compresser à différents débits avec une unique transformée apprise.
Concernant l'apprentissage d'un filtre de prédiction intra pour la compression d'image via des réseaux de neurones, le problème est d'obtenir un filtre appris qui s'adapte à la taille du bloc d'image à prédire, à l'information manquante dans le contexte de prédiction et au bruit de quantification variable dans ce contexte. Un ensemble de réseaux de neurones est conçu et entraîné de façon à ce que le filtre appris soit adaptif à ces égards.

Title: deep learning for image compression

Keywords: deep neural networks, image compression, transform coding, intra prediction.

Abstract: Over the last twenty years, the amount of transmitted images and videos has increased noticeably, mainly urged on by Facebook and Netflix. Even though broadcast capacities improve, this growing amount of transmitted images and videos requires increasingly efficient compression methods. This thesis aims at improving via learning two critical components of the modern image compression standards, which are the transform and the intra prediction. More precisely, deep neural networks are used for this task as they exhibit high power of approximation, which is needed for learning a reliable approximation of an optimal transform (or an optimal intra prediction filter) applied to image pixels.
Regarding the learning of a transform for image compression via neural networks, a challenge is to learn an unique transform that is efficient in terms of rate-distortion while keeping this efficiency when compressing at different rates. That is why two approaches are proposed to take on this challenge. In the first approach, the neural network architecture sets a sparsity on the transform coefficients. The level of sparsity gives a direct control over the compression rate. To force the transform to adapt to different compression rates, the level of sparsity is stochastically driven during the training phase. In the second approach, the rate-distortion efficiency is obtained by minimizing a rate-distortion objective function during the training phase. During the test phase, the quantization step sizes are gradually increased according a scheduling to compress at different rates using the single learned transform.
Regarding the learning of an intra prediction filter for image compression via neural networks, the issue is to obtain a learned filter that is adaptive with respect to the size of the image block to be predicted, with respect to missing information in the context of prediction, and with respect to the variable quantization noise in this context. A set of neural networks is designed and trained so that the learned prediction filter has this adaptibility.