Compression efficace d'images basée sur un modèle de représentation d'état

Bouzid Arezki, Anissa Mokraoui, Fangchen Feng
L2TI, Université Sorbonne Paris Nord
99, avenue Jean-Baptiste Clément, 93430 Villetaneuse, France
{bouzid.arezki, anissa.mokraoui, fangchen.feng}@univ-paris13.fr

Résumé

Les réseaux neuronaux profonds ont révolutionné le domaine de la compression d'images en surpassant les approches traditionnelles. Cependant, leur complexité élevée limite souvent leur utilisation. Pour pallier ce problème, diverses stratégies telles que la distillation des connaissances et les architectures légères ont été explorées pour réduire la complexité tout en maintenant de bonnes performances. Cet article présente une nouvelle architecture de compression d'images basée sur un modèle de représentation d'état. Cette architecture trouve un équilibre entre performance et complexité de calcul, la rendant adaptée aux applications pratiques. Les évaluations expérimentales confirment que notre architecture obtient de bonnes performances en termes de BD-rate, tout en réduisant considérablement la complexité de calcul et la latence par rapport aux autres méthodes compétitives de l'état de l'art.

Mots clefs

Compression d'images, Modèle d'espace d'état, Complexité de calcul, Débit-distorsion

1 Introduction

Les méthodes de compression basées sur les réseaux neuronaux profonds [1, 2] offrent des performances en constante amélioration par rapport aux approches traditionnelles. Elles incluent généralement un autoencodeur variationnel hiérarchique à deux niveaux avec un *hyper-prior* comme modèle d'entropie, composé de deux ensembles d'encodeurs et de décodeurs. Ces méthodes sont souvent trop complexes pour une utilisation en temps réel. Des modèles réduits sont proposés pour diminuer cette complexité, souvent en sacrifiant la performance débit-distorsion [2]. Pour pallier cela, des techniques comme la distillation des connaissances, les architectures légères et les mécanismes d'attention légers [3] ont été développées. Récemment, les modèles à espace d'état (SSM) et leur variante, le modèle Mamba, ont suscité un grand intérêt en vision par ordinateur. Cependant, leur adoption a été limitée par des exigences élevées en calcul et en mémoire. Mamba [4] surmonte ces limitations en intégrant un mécanisme de sélection, améliorant ainsi le raisonnement contextuel.

Dans cet article, nous introduisons une architecture de compression d'images basée sur un modèle à espace d'état, qui met l'accent sur la performance en termes de débit-distorsion, la complexité de calcul et la latence. Des expériences approfondies montrent que notre méthode parvient à atteindre de bonnes performances en termes de compression tout en réduisant de manière significative la complexité de calcul et la latence par rapport aux méthodes compétitives de l'état de l'art. Cette affirmation est étayée par la Fig. 1 (BD-rate versus complexité de calcul), qui situe clairement notre méthode de compression parmi les approches les plus compétitives de l'état de l'art.

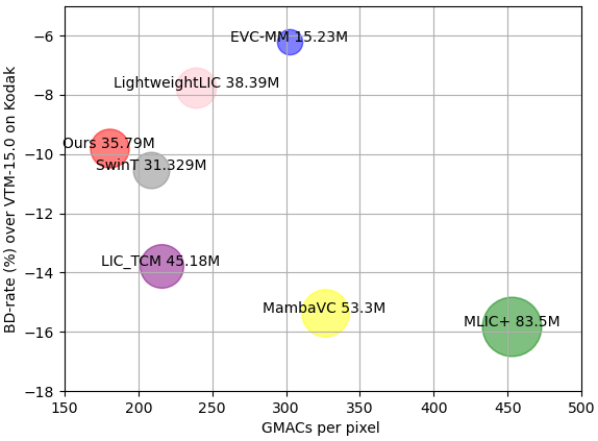

FIGURE 1 – *BD-rate (VTM-15.0 [5] comme référence) vs complexité de calcul (GMAC) sur Kodak [6]. Le rayon des cercles indique le nombre de paramètres du modèle.*

2 Préliminaires

Introduisons ci-dessous quelques notions utiles sur les modèles de représentation d'état *space state model* (SSM) sur lesquels notre méthode de compression s'appuie.

La transformation du SSM dans S4 [4] est dérivée du modèle classique d'espace d'état, qui associe un signal d'entrée unidimensionnel $x(t) \in \mathbb{R}$ à un signal de sortie unidimensionnel $y(t) \in \mathbb{R}$ via un état latent $h(t) \in \mathbb{R}^N$ de dimension N, nous pouvons formuler le processus à l'aide d'équations différentielles ordinaires linéaires (EDO) :

$$\begin{aligned} h'(t) &= \mathbf{A}h(t) + \mathbf{B}x(t), \\ y(t) &= \mathbf{C}h(t), \end{aligned} \tag{1}$$

où $\mathbf{A} \in \mathbb{R}^{N\times N}$, $\mathbf{B} \in \mathbb{R}^{N\times 1}$, $\mathbf{C} \in \mathbb{R}^{1\times N}$ sont des paramètres du réseau de neurones.

Pour traiter la séquence d'entrée discrète $x = [x_0, x_1, \ldots, x_{L-1}] \in \mathbb{R}^L$, les paramètres de l'équation (1) sont discrétisés en utilisant une taille de pas Δ, représentant la résolution de l'entrée continue $x(t)$ [4]. En particulier, les paramètres continus $\mathbf{A}$ et $\mathbf{B}$ sont convertis en paramètres discrets $\overline{\mathbf{A}}$ et $\overline{\mathbf{B}}$ par la technique *zero-order hold (ZOH)*, définie comme suit :

$$\begin{aligned} \overline{\mathbf{A}} &= \exp(\Delta\mathbf{A}), \\ \overline{\mathbf{B}} &= (\Delta\mathbf{A})^{-1}(\exp(\Delta\mathbf{A}) - \mathbf{I}).\Delta\mathbf{B}. \end{aligned} \tag{2}$$

Aprés la discrétisation de $\mathbf{A}$ et $\mathbf{B}$, l'equation (1) est reformulée comme suit :

$$\begin{aligned} h_t &= \overline{\mathbf{A}}h_{t-1} + \overline{\mathbf{B}}x_t, \\ y_t &= \mathbf{C}h_t. \end{aligned} \tag{3}$$

Les SSM peuvent alors être calculés efficacement à l'aide de RNNs. Le processus récursif peut être reformulé et calculé comme une convolution :

$$\begin{aligned} \overline{\mathbf{K}} &= (\mathbf{C}\overline{\mathbf{B}}, \mathbf{C}\overline{\mathbf{A}\mathbf{B}}, \ldots, \mathbf{C}\overline{\mathbf{A}}^{L-1}\overline{\mathbf{B}}), \\ y &= x * \overline{\mathbf{K}}, \end{aligned} \tag{4}$$

où L représente la longueur de la séquence d'entrée x ; et $\overline{\mathbf{K}} \in \mathbb{R}^L$ le noyau de convolution SSM.

Mamba [7] intègre la dépendance des données pour capturer l'information contextuelle dans l'équation (1) en introduisant une nouvelle méthode de paramétrage pour les SSM incluant un mécanisme de sélection conditionnelle basé sur l'entrée, appelé S6. Bien que la nature récurrente des SSM limite la possibilité d'une parallélisation complète, Mamba propose une optimisation grâce à des techniques de paramétrage structuré et à un algorithme de balayage parallèle efficace sur le plan matériel. En conséquence, de nombreux travaux ont adapté Mamba du traitement du langage naturel (NLP) au domaine de la vision [8].

3 Méthode de compression proposée

Notre méthode repose sur une architecture à deux niveaux. Dans un premier temps, l'image d'entrée x est encodée par l'encodeur génératif pour obtenir $y = g_a(x)$. Ensuite, le hyper-latent $z = h_a(y)$ est extrait via l'encodeur du réseau hyper-prior. Le hyper-latent quantifié $\hat{z}$ est alors modélisé et codé à l'aide d'un codage entropique basé sur un modèle factorisé *Factorized model* avant d'être transmis à travers $h_s(\hat{z})$. Les sorties de h_s et du modèle contextuel *context model* sont ensuite utilisées par le réseau de paramètres d'entropie *Entropy Parametres* [1], qui génère les paramètres μ et σ d'un modèle d'entropie gaussien conditionnel $P(y|\hat{z}) = \mathcal{N}(\mu, \sigma^2)$ pour modéliser y. Le vecteur latent quantifié $\hat{y} = Q(y)$ est codé au final via un codage entropique (codage/décodage arithmétique AE/AD) et transmis à $\hat{x} = g_s(\hat{y})$ pour reconstruire l'image $\hat{x}$. L'architecture détaillée de notre méthode est schématisée dans la Fig. 2.

Nous proposons d'utiliser le *context model* décrit dans [1]. Contrairement à l'architecture présentée dans [12], notre choix vise à développer une architecture optimisée pour l'efficacité computationnelle.

Les encodeurs génératifs et hyper-prior, g_a et h_a, sont construits avec le bloc de fusion *patch merge* et le bloc *Visual State Space (VSS)* illustré dans la Fig 3. Le bloc de *patch split* utilise l'opération *Depth-to-Space* [2] pour le sous-échantillonnage, une couche de normalisation et une couche linéaire pour projeter l'entrée à une certaine profondeur C_i. Dans g_a, la profondeur C_i de vecteur latent augmente à mesure que le réseau devient plus profond, ce qui permet une représentation de l'image de plus en plus abstraite. À chaque étape, nous réduisons la résolution par un facteur 2. Comparé à la couche convolutionnelle utilisée dans MambaVC [12], le bloc de fusion *patch merge* que nous avons choisi est plus simple à implémenter et offre également une complexité de calcul moindre.

Le bloc VSS, proposé initialement dans [8], consiste en une seule branche de réseau avec deux modules résiduels, suivant l'architecture du bloc Transformer classique. Plus précisément, chaque niveau de profondeur de notre méthode se compose d'une séquence de blocs VSS ou le nombre de blocs à chaque niveau i est noté d_i (voir Fig. 2). A partir d'une carte de caractéristiques d'entrée $\mathbf{f} \in \mathbb{R}^{H\times W\times C}$, nous obtenons $\mathbf{f}''$ à partir d'un premier module résiduel :

$$\mathbf{f}'' = \mathbf{f} + \mathbf{f}', \tag{5}$$

où $\mathbf{f}'$ est obtenu comme suit :

$$\begin{aligned} \mathbf{f}' = \mathrm{MLP}_2(\mathrm{LN}_2(\mathrm{SS2D}(\\ \sigma(\mathrm{DWConv}(\mathrm{MLP}_1(\mathrm{LN}_1((f)))))))). \end{aligned} \tag{6}$$

Comme illustré par la Fig. 3 (voir (a) and (b)), la sortie du VSS bloc est donnée par :

$$\mathbf{f}_{out} = \mathrm{MLP}_3(\mathrm{LN}_3(\mathbf{f}'')) + \mathbf{f}'', \tag{7}$$

où LN représente la couche de normalisation ; SS2D le module de balayage sélectif 2D ; σ la fonction d'activation SiLU ; DWConv la convolution depthhwise ; et MLP la projection linéaire. Contrairement au bloc VSS dans [8], nous utilisons RMSNorm [16] à la place de LayerNorm comme couche de normalisation. Nous avons observé empiriquement que RMSNorm améliore significativement la vitesse de convergence lors de l'entraînement. Nous adoptons l'approche de balayage sélectif proposée dans [8], qui adapte un mécanisme de sélection dépendant de l'entrée [7] aux données 2D sans compromettre ses avantages. SS2D comprend trois étapes illustrées dans la Fig. 3 (c) : Balayage croisé *Cross-scan* : déplie les caractéristiques d'entrée en séquences le long de quatre chemins de traversée distincts ; Balayage sélectif *Selective scan* : traite chaque chemin avec un S6 distinct en parallèle ; Fusion croisée *Cross-merge* : remodèle et fusionne ensuite les séquences résultantes pour former les caractéristiques de sortie, intégrant efficacement l'information provenant d'autres pixels dans différentes directions par rapport à chaque pixel courant. Cela facilite l'établissement de champs réceptifs globaux dans l'espace 2D.

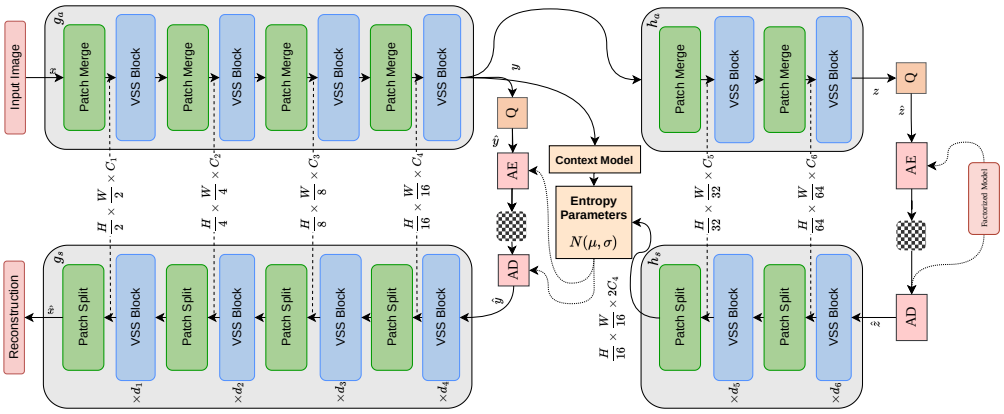

FIGURE 2 – *Architecture de notre méthode proposée de compression d'images.*

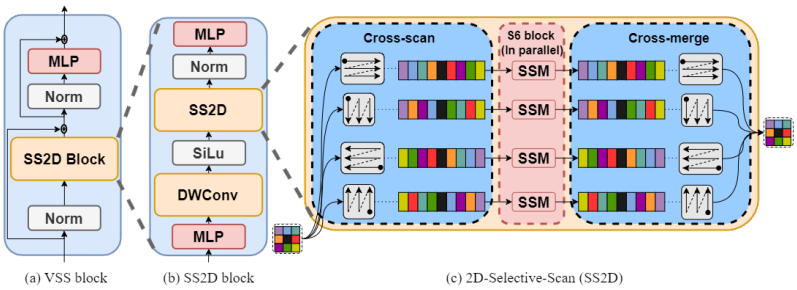

FIGURE 3 – *Le bloc VSS [8] se compose d'un bloc SS2D qui effectue des balayages sélectifs selon 4 parcours parallèles.*

4 Analyse des performances

Notre modèle a été entraîné sur le jeu de données CLIC20 [9] en utilisant une fonction de perte $L = D + \lambda R$, où R représente le débit binaire et D la distorsion. Nous utilisons l'erreur quadratique moyenne (MSE) comme mesure de distorsion dans l'espace de couleur RGB. Le multiplicateur de Lagrange λ ajuste le compromis débit-distorsion (RD), avec des valeurs choisies $\lambda \in \{100, 50, 30, 10\}$. Chaque lot d'entraînement comporte 8 découpes aléatoires des images de taille 256×256. Nous avons effectué 1 million (1M) d'itérations avec l'optimiseur ADAM et un taux d'apprentissage fixé à 10^{-4}. notre modèle a été implémentée avec Pytorch en utilisant la bibliothèque CompressAI [17].

Nous proposons d'évaluer les performances de notre modèle sur trois jeux de données : Kodak [6], JPEG-AI [10], et CLIC20 [9] incluant les catégories mobile et professionnelle. Toutes les images ont été complétées par des zéros dans cas où leur taille n'est pas un multiple de 256. Nous comparons notre modèle aux modèles compétitifs de l'état de l'art [2,3,12–15] choisis pour leurs performances (débit-distorsion, complexité de calcul). Les expériences ont été réalisées sur un GPU A100 80 Go et un CPU Intel Xeon Gold 6330 3,10 GHz.

Le tableau 1 présente le BD-rate des modèles compétitifs sur les 3 datasets, avec un débit couvrant la plage de 0.4 à 1.2 bits par pixel (bpp), et le VTM-15.0 [5] comme référence. Notre modèle atteint un BD-rate de -21.75% et une augmentation relative de 4.17% par rapport au modèle LIC_TCM [15]. Ce dernier reconnu pour ses performances supérieures parmi les méthodes comparées. Cependant, ce modèle présente une complexité de calcul et un nombre de paramètres significativement plus élevés, comme illustré par Fig. 1. Celle-ci compare les performances en termes de BD-rate et de GMACs [1] des différentes méthodes sur le jeu de données Kodak. L'analyse du graphique confirme bien que notre modèle offre un excellent compromis entre le BD-rate, la complexité de calcul et le nombre de paramètres.

Le tableaux 2 compare la complexité de calcul évaluée en termes de MACs[1] pour trois résolutions différentes, soulignant la réduction significative apportée par notre approche en termes de complexité de calcul.

Le tableau 3 donne la latence moyenne sur 2000 images à une résolution de 256×256 sur le GPU utilisé. Notre modèle affiche des temps de décodage compétitifs par rapport à lightweightLIC [3], tout en maintenant des temps de codage similaires.

1. Les MAC et les FLOP sont calculés à l'aide de la bibliothèque calflops. `https://github.com/MrYxJ/calculate-flops.pytorch`

Methodes	Kodak [6]	CLIC2020 [9]	JPEG-AI [10]	La moyenne
BPG444 [11]	29.86%	32.77%	43.77%	35.46%
SwinT* [2]	-10.52%	-6.47%	-2.78%	-6.03%
MambaVC* [12]	-15.37%	-16.69%	-12.47%	-14.69%
SwinNPE [13]	-5.85%	-17.50%	-23.56%	-15.63%
LightweightLIC [3]	-7.76%	-23.60%	-29.86%	-20.40%
MLIC+* [14]	**-15.86%**	-	-15.89%	-
LIC_TCM [15]	-13.76%	**-30.65%**	**-33.37%**	**-25.92%**
Méthode proposée	-9.81%	-29.91%	-25.55%	-21.75%

TABLEAU 1 – *BD-rate performance (VTM-15.0 [5] comme référence). $*$ Chiffres extraits des articles (car les modèles pré-entrainés ne sont pas disponibles). L'évaluation de MLIC+ sur le jeu de données CLIC2020 n'est pas fournie dans [14].*

Résolution	Méthode proposée	SwinT [2]	LIC_TCM [15]	LightweightLIC [3]	MambaVC [12]	MLIC+ [14]
768×512	**180.053G**	208.789G	215.316G	239.213G	326.112G	452.622G
1024×768	**360.106G**	417.570G	430.632G	478.425G	652.224G	905.243G
1280×1280	**750.222G**	869.956G	897.150G	996.719G	OM	1.8859T
#paramètres (M)	35.79	32.34	45.18	38.39	53.30	83.50

TABLEAU 2 – *Multiply-Add Cumulation (MACs), mesuré sur un GPU A100 80 Go et un CPU Intel Xeon Gold 6330 3,10 GHz, a différentes résolutions d'images. OM signifie Out of Memory.*

Méthodes	Latence GPU (ms)	
	Encodage	Décodage
MambaVC [12]	**14.01**	73.36
LIC_TCM [15]	15.23	52.46
SwinT [2]	14.25	20.86
LightweightLIC [3]	15.62	**15.56**
Méthode proposée	20.24	19.93

TABLEAU 3 – *Latence moyenne sur 2000 images (résolution 256×256) sur un GPU A100 80 Go et un processeur Intel Xeon Gold 6330 3,10 GHz.*

5 Conclusion

Nous avons proposé une approche de compression efficace d'images basée sur le modèle de représentation d'état. Celle-ci présente des performances compétitives en termes de débit-distorsion tout en réduisant de manière significative la complexité de calcul et la latence.

Références

[1] David Minnen, Johannes Ballé, et George D Toderici. Joint autoregressive and hierarchical priors for learned image compression. *NeurIPS*, 31, 2018.

[2] Yinhao Zhu, Yang Yang, et Taco Cohen. Transformer-based transform coding. Dans *ICLR*, 2021.

[3] Ziyang He, Minfeng Huang, Lei Luo, Xu Yang, et Ce Zhu. Towards real-time practical image compression with lightweight attention. *Expert Systems with Applications*, 252 :124142, 2024.

[4] Albert Gu, Karan Goel, et Christopher Ré. Efficiently modeling long sequences with structured state spaces. *arXiv*, 2021.

[5] Benjamin Bross, Ye-Kui Wang, Yan Ye, Shan Liu, Jianle Chen, Gary Sullivan, et Jens-Rainer Ohm. Overview of the versatile video coding (vvc) standard and its applications. *TCSVT*, 31 :3736–3764, 10 2021.

[6] Rich Franzen. Kodak lossless true color image suite. `http://r0k.us/graphics/kodak`, 1999.

[7] Albert Gu et Tri Dao. Mamba : Linear-time sequence modeling with selective state spaces. *arXiv*, 2023.

[8] Yue Liu et Tian et al. Vmamba : Visual state space model. *arXiv*, 2024.

[9] Lucas Theis George Toderici et al. Clic 2020 : Challenge on learned image compression. 2020.

[10] JPEG-AI. Jpeg-ai test images. `https://jpegai.github.io/test_images/`, 2020.

[11] Fabrice Bellard. Bpg image format. `http://bellard.org/bpg/`, 2018.

[12] Shiyu Qin, Jinpeng Wang, Yimin Zhou, Bin Chen, Tianci Luo, Baoyi An, Tao Dai, Shutao Xia, et Yaowei Wang. Mambavc : Learned visual compression with selective state spaces. *arXiv*, 2024.

[13] Bouzid Arezki, Fangchen Feng, et Anissa Mokraoui. Convolutional transformer-based image compression. Dans *SPA*, pages 154–159. IEEE, 2023.

[14] Wei Jiang, Jiayu Yang, Yongqi Zhai, Peirong Ning, Feng Gao, et Ronggang Wang. Mlic : Multi-reference entropy model for learned image compression. ACM, Octobre 2023.

[15] Jinming Liu, Heming Sun, et Jiro Katto. Learned image compression with mixed transformer-cnn architectures. Dans *CVPR*, pages 14388–14397, 2023.

[16] Biao Zhang et Rico Sennrich. Root mean square layer normalization. *NeurIPS*, 32, 2019.

[17] Jean Bégaint, Fabien Racapé, Simon Feltman, et Akshay Pushparaja. Compressai : a pytorch library and evaluation platform for end-to-end compression research, 2020.