Convolutional Auto-Encoder with Tensor-Train Factorization

Manish Sharma, Panos P. Markopoulos, Eli Saber
Rochester Institute of Technology
Rochester NY, USA
{ms8515, panos, esseee}@rit.edu

M. Salman Asif
University of California Riverside
Riverside CA, USA
sasif@ece.ucr.edu

Ashley Prater-Bennette
Air Force Research Laboratory
Rome NY, USA
ashley.prater-bennette@us.af.mil

Abstract

Convolutional auto-encoders (CAEs) are extensively used for general purpose feature extraction, image reconstruction, image denoising, and other machine learning tasks. Despite their many successes, similar to other convolutional networks, CAEs often suffer from over-parameterization when trained with small or moderate-sized datasets. In such cases, CAEs suffer from excess computational and memory overhead as well as decreased performance due to parameter over-fitting. In this work we introduce CAE-TT: a CAE with tunable tensor-train (TT) structure to its convolution and transpose-convolution filters. By tuning the TT-ranks, CAE-TT can adjust the number of its learning parameters without changing the network architecture. In our numerical studies, we demonstrate the performance of the proposed method and compare it with alternatives, in both batch and online learning settings.

1. Introduction

Deep learning frameworks such as convolutional neural networks (CNNs) have emerged as a solution of choice for computer vision applications, including image classification [1], object detection [2, 3, 4], and image segmentation [5], to name a few. Most CNNs are designed for supervised learning and consider available labels for the training data. In many real-world settings, however, labels are scarce or unavailable and unsupervised learning methods are needed. Convolutional Auto-Encoders (CAEs) are unsupervised CNNs that serve as general-purpose feature extractors, by learning convolution filters that minimize an objective loss. CAEs are quite popular for dimensionality reduction, image compression/reconstruction, and denoising, among other tasks [6, 7].

Deep and wide CAEs have many learning parameters and, accordingly, high learning capacity that allows them to identify complex underlying structures in the data. At the same time, they require large amounts of training examples for generalizing well to unseen data. In real-world applications with small to moderate-sized training datasets, deep and wide networks tend to memorize the training data (over-fitting), thus compromising their ability to generalize inference to unseen data. In such cases, limiting the overall number of trainable parameters reduces the network capacity, but can also resolve over-fitting, thus leading to improved performance while also reducing computational and memory overhead. On the other hand, when more training data become available in a streaming way, which is a common case in many real-world applications such as real-time streaming video frames in computer vision, increasing the number of parameters can increase the network capacity and, thus, allow to further improve inference [8, 9]. It is therefore desirable to design CAEs with a tunable number of parameters that can adjust to the complexity and size of the available data.

A conceptually simple way to adjust the number of trainable parameters is by tuning the width and depth of the network, i.e., the number of convolution filters per layer and the number of layers in the network. Certainly, these approaches would result in significant changes in the network architecture which is impractical for on-the-fly adjusting the

This material is based upon work that was supported in part by NSF Awards OAC-1808582 and CCF-2046293, AFOSR Awards FA9550-20-1-0039 and FA9550-21-1-0330, and NGA Award HM0476-19-1-2014.

Cleared for public release by US Air Force Security and Policy Review on 16 July 2021, case number AFRL-2021-2294.

This research is funded by an academic grant from the National Geospatial-Intelligence Agency (Award No. # HM0476-19-1-2014, Project Title: Target Detection/Tracking and Activity Recognition from Multimodal Data). Approved for public release, 21-812.

parameters of AI systems in real-world deployments. In this work, we propose an alternative approach based on tensor analysis, which allows one to adjust the number of parameters without changing the network architecture. A high-level description follows.

Since CAEs are fully-convolutional networks, all trainable parameters are in the form of convolution filters, which are typically modeled as 4-way tensors (modes 1 and 2 are spatial; modes 3 and 4 correspond to input and output channels, respectively). Similar to matrices, tensor variables are often modeled as a product of latent factors. Such tensor factorization is a standard practice in data analysis and machine learning, with applications including data compression, denoising, multi-modal feature extraction, and parameter estimation. Standard tensor factorization methods include Tucker Decomposition (TD) [10], Parallel Factor Analysis (PARAFAC) [11], and Tensor-Train Factorization (TT) [12], among others. Among these tensor factorizations, TT has exhibited top performance in modeling convolution filters in supervised convolutional networks, such as classification CNNs, object-detection CNNs, and convolutional long short-term memory networks (LSTMs) for spatio-temporal learning [4, 13, 14, 15].

In this work, for the first time, we use TT factorization to model the convolution filters of a CAE for unsupervised learning. Specifically, we parameterize the convolution and transpose-convolution filters of CAEs as a product of TT-cores, which contain all trainable parameters and are trained in an end-to-end fashion. The number and size of the cores can be tuned, adjusting the number of trainable parameters, without changing the size of the convolution filter and, accordingly, without changing the network architecture. We refer to the proposed method as *CAE-TT*.

For a low number of moderately-sized TT-cores, the total number of parameters is significantly lower than that of standard CAE and, thus, over-fitting is avoided. On the other hand, to prevent under-fitting and increase the network capacity as more training data become available, we can simply increase the number and size of trainable TT-cores while, still, the sizes of the filters and the network architecture remain invariant. We evaluate the efficiency of the proposed method in image compression and denoising experiments, on the standard MNIST [16] and FMNIST [17] datasets. The merits of CAE-TT are clearly illustrated for both batch and online training.

2. Technical Background

2.1. Convolutional Auto-Encoder (CAE)

CAE is a sequential arrangement of convolutional encoding and decoding layers. Based on their convolutional structure, CAEs are preferred for image data processing over standard auto-encoders. The encoding convolution layers

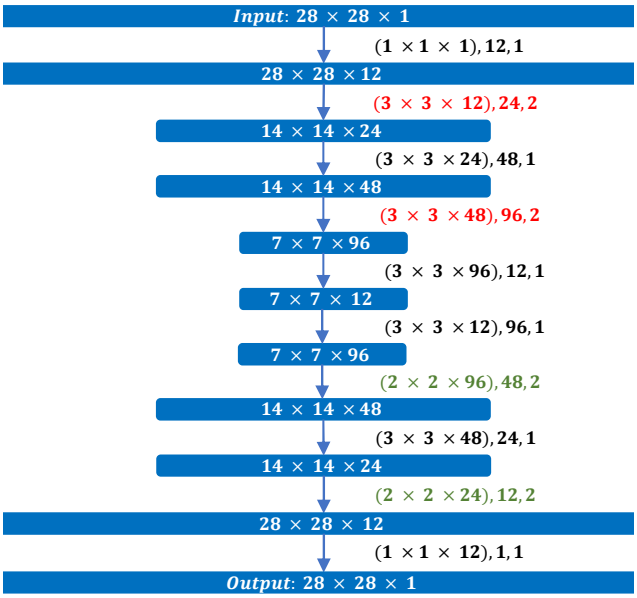

Figure 1: Schematic of standard CAE with 10 layers. The top and bottom rectangles present input and output, respectively. The rectangles in between represent the feature maps between successive layers. In each blue rectangle, we note the 3-mode dimensions. Between the feature maps, we present the parameters of the convolution filter in the form: $(l_1 \times l_2 \times C), S, z_i$. The red and green color font denotes encoding (down-sampling) and decoding (up-sampling) layers respectively. Black color layers are standard convolutional layers.

(encoder) map the input image $\underline{\mathbf{I}} \in \mathbb{R}^{W \times H \times F}$ ($W \times H$ pixels, F channels) to a compact representation $\underline{\mathbf{M}} \in \mathbb{R}^{w \times h \times f}$, for $w < W$, $h < H$, and $f < F$. The decoding transpose-convolution layers (decoder) map the latent representation back to the output $\underline{\mathbf{Y}} \in \mathbb{R}^{W \times H \times F}$. That is, the encoder extracts features from the input and the decoder uses these features to reconstruct it.

Each layer of the CAE receives as input a tensor feature map $\underline{\mathbf{X}}$ of size $L_1^{(in)} \times L_2^{(in)} \times C$. Modes 1 and 2 refer to space and mode 3 refers to channels. Then the layer convolves $\underline{\mathbf{X}}$ with a convolution filter tensor $\underline{\mathbf{K}}$ of size $l_1 \times l_2 \times C \times S$. Denoting by $*$ the convolution opearation, the filter output is $\underline{\mathbf{Y}} = \underline{\mathbf{X}} * \underline{\mathbf{K}}$ of size $L_1^{(out)} \times L_2^{(out)} \times S$, where $L_i^{(out)} = (L_i^{(in)} - l_i + p_i)/z_i + 1$, z_i and p_i are the convolution stride and padding for mode i, respectively, and S is the number of output channels. Fig. 1 offers a schematic illustration of a standard CAE with 10 layers.

2.2. Tensor Preliminaries

A tensor is d-way array, each entry of which is addressed with d indices. For example, a vector is a 1-way tensor

$$\underline{\mathbf{C}}[i_1,\ldots,i_{k_1-1},j_1,\ldots,j_{k_2-1},j_{k_2+1},\ldots,j_{d_2},i_{k_1+1},\ldots i_{d_1}]$$
$$=\sum_{t=1}^{n_{k_1}}\underline{\mathbf{A}}[i_1,\ldots,i_{k_1-1},t,i_{k_1+1},\ldots i_{d_1}]\underline{\mathbf{B}}[j_1,\ldots,j_{k_2-1},t,j_{k_2+1},\ldots,j_{d_2}] \quad (1)$$

while a matrix is a 2-way tensor.

Contraction is the tensor generalization of matrix product. Considering tensors $\underline{\mathbf{A}} \in \mathbb{R}^{n_1\times n_2\times\cdots\times n_{d_1}}$ and $\underline{\mathbf{B}} \in \mathbb{R}^{m_1\times m_2\times\cdots\times m_{d_2}}$ satisfying $n_{k_1} = m_{k_2}$, their (k_1,k_2)-contraction $\underline{\mathbf{C}} = \underline{\mathbf{A}} \circ_{k_1}^{k_2} \underline{\mathbf{B}}$ is defined as in equation (1) and has size $n_1\times\cdots\times n_{k_1-1}\times m_1\times\cdots m_{k_2-1}\times m_{k_2+1}\times\cdots\times m_{d_2}\times n_{k_1+1}\times\cdots\times n_{d_1}$. For simplicity in notation, we denote $\underline{\mathbf{A}}\circ_{d_1}^{1}\underline{\mathbf{B}}$ simply as $\underline{\mathbf{A}}\circ\underline{\mathbf{B}}$. For $\mathbf{D}\in\mathbb{R}^{n_k\times m}$, the mode-$k$ tensor-matrix product $\underline{\mathbf{A}}\times_k\mathbf{D}$ is equal to $(k,1)$-contraction $\underline{\mathbf{A}}\circ_k^1\underline{\mathbf{D}}$, where $\underline{\mathbf{D}}$ is the (2-way) tensor representation of matrix $\mathbf{D}$ [18].

3. Proposed Method

In this paper, we parameterize the convolution and transpose-convolution filters of the CAE by means of TT factorization. TT factorization of the learning parameters has already been successfully applied before in neural networks [4, 13, 14, 15, 19]. In this work, we extend it for the first time to a convolutional auto-encoder.

By tuning the ranks of the TT factorization we can limit redundancy in the number of learnable parameters, avoid over-fitting, and improve performance in applications where training data are limited. At the same time, when more training data become available, we can increase the number of trainable parameters (and, thus, the network capacity) by increasing the TT factorization ranks without altering the filter dimension and the network structure.

In order to TT parameterize a convolution filter $\underline{\mathbf{K}} \in \mathbb{R}^{l_1\times l_2\times C\times S}$, we first factorize C and S as $C = c_1\cdots c_d$ and $S = s_1\cdots s_d$, respectively, for some modeling parameter $d\geq 2$. Then, we reshape $\underline{\mathbf{K}}$ into a $(d+1)$-way tensor $\underline{\mathbf{K}}'$ of size $l_1l_2\times c_1s_1\times\ldots\times c_ds_d$ and factorize it as a train of tensor contractions (tensor train) of the form

$$\underline{\mathbf{K}}' = \underline{\mathbf{G}}_0\circ\underline{\mathbf{G}}_1\circ\cdots\circ\underline{\mathbf{G}}_d. \quad (2)$$

The proposed TT parameterization of the convolution filter is presented in Fig. 2.

The tensors $\{\underline{\mathbf{G}}_i\}_{i=0}^d$ are known as TT-cores of $\underline{\mathbf{K}}$ and comprise the trainable parameters of the proposed network. $\underline{\mathbf{G}}_0$ has size $1\times l_1l_2\times r_1$, $\underline{\mathbf{G}}_d$ has size $r_d\times c_ds_d\times 1$ and for every $0<i<d$, $\underline{\mathbf{G}}_i$ has size $r_i\times c_is_i\times r_{i+1}$. The modeling parameters $\{r_i\}_{i=1}^d$ are known as TT-ranks and determine the number of learning parameters of the layer. Specifically, before TT parameterization, this layer would

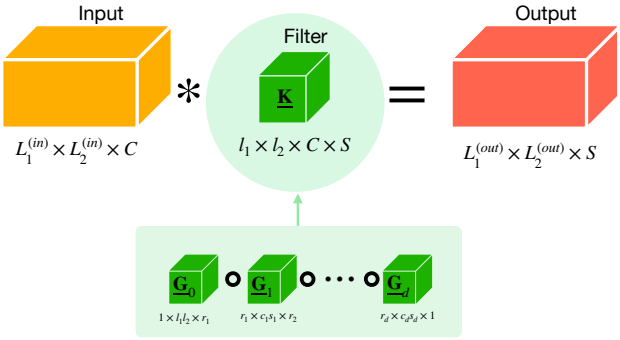

Figure 2: TT parameterization of a convolution filter.

have $P = l_1l_2CS$ learning parameters; after TT modeling, the number of learning parameters is $P_{TT}(\{r_i\}_{i=1}^d) = l_1l_2r_1 + r_dc_ds_d + \sum_{i=1}^{d-1} r_ir_{i+1}c_is_i$. For any given l, C, and S, P_{TT} is a function of the TT modeling parameters d and $\{r_i\}_{i=1}^d$. Finding optimal values for these parameters is an involved task. In this work, we select TT-ranks $\{r_j\}_{j=1}^d$ by means of the method introduced in [4]. That is, we set $r_j = z_j(b) = \lfloor br_j^{max}\rfloor$, where r_j^{max} is an upper bound for r_j and $b\in(0,1)$ is a tunable coefficient, common across all ranks, that determines the overall compression of the TT model. Accordingly, the number of learning parameters for this layer becomes

$$P_{TT}(b,d) = l_1l_2z_1(b) + z_d(b)c_ds_d + \sum_{j=1}^{d-1} z_j(b)z_{j+1}(b)c_js_j \quad (3)$$

and can be adjusted by tuning only b and d.

To demonstrate how tuning b and d changes the total number of parameters, we consider the CAE of Fig. 1 and factorize its filters according to the proposed method, for varying values of b and d, while keeping the same number of layers and number of channels per layer. In Fig. 3, we plot the combined $P_{TT}(b,d)$ vs. d for various values of b. We notice that, interestingly, the number of parameters is a monotonically increasing function of b, but not of d. That is, a lower number of TT-cores could possibly result to a higher overall number of learning parameters.

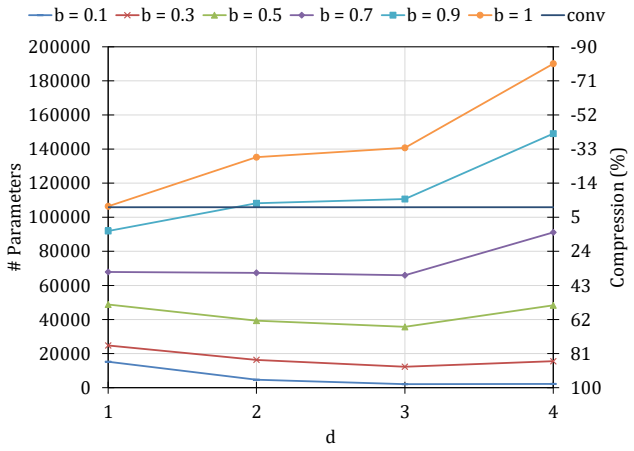

Figure 3: Number of parameters in CAE-TT vs. number of TT factors d, for various values of compression coefficient b along with baseline uncompressed CAE (conv).

4. Experimental Studies

In this section, we present the performance of the proposed CAE-TT with experiments on image compression/reconstruction and denoising. For our experiments we employ the standard MNIST [16] and FMNIST [17] datasets. Both datasets consist of $60,000$ training and $10,000$ testing grayscale images of size 28×28 capturing 10 distinct classes. To demonstrate the parameter-tuning ability of the proposed method, we also present experiments with online training. To that end, we partition training data from MNIST and FMNIST into 5 and 6 streaming batches, respectively. Each stream batch consists of $1,000$ examples from each of the 10 classes. All training and testing images are normalized to take values in $[0, 1]$. In all our studies, we consider the basic CAE of Fig. 1 that constitutes of 10 layers that include 2 down-sampling convolution layers and 2 up-sampling transpose-convolution layers. Our optimizer is ADAM [20] with a learning rate 10^{-3} and mini-batch size of $5,000$ images. For the proposed CAE-TT, we consider $d = 2$, which means that each filter is factorized into three TT-cores. Our performance evaluation metrics are Peak Signal-to-Noise Ratio (PSNR) and the Structural Similarity Index Measure (SSIM) [21]. Finally, all the results are averaged over three experimental realizations.

4.1. Results

4.1.1 Reconstruction

In this first set of studies, we consider simple compression and reconstruction of images in the MNIST and FMNIST datasets. The CAE and CAE-TT architectures are as presented above.

In our first study, we consider that all FMNIST training data are at-once available (i.e., not streaming) to the network and we investigate how different numbers of learning parameters affect the reconstruction performance. As discussed in Section I, a simple approach to adjust the number of learning parameters is to tune the network width – i.e., the number of filters per convolution and transpose-convolution layer. Alternatively, the proposed approach adjusts the number of parameters by tuning the TT-ranks of the filters. In Fig. 4, we plot the image reconstruction performance (PSNR on the top and SSIM on the bottom) versus the number of learning parameters, for network-width tuning and TT-rank tuning. We observe that CAE-TT lies closer to the top-left corner. This is particularly clear in the SSIM plots where CAE-TT attains very high performance, exceeding 0.96, with fewer than $40,000$ parameters, while network-width tuning needs more than $70,000$ parameters to reach similar performance.

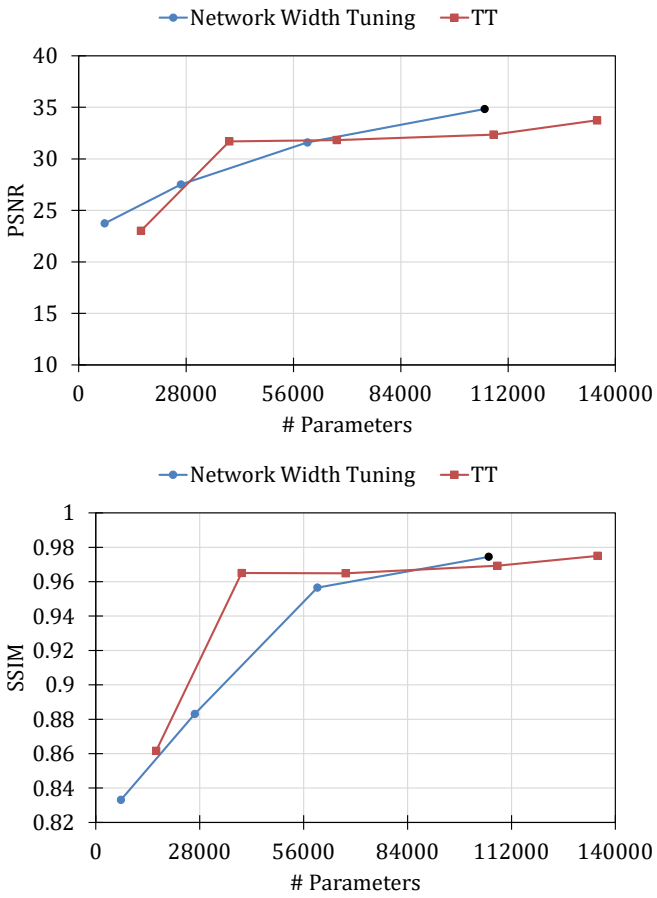

Figure 4: PSNR (top) and SSIM (bottom) performance vs. number of parameters for network-width tuning and the proposed TT-rank tuning (CAE-TT).

Next, we demonstrate the utility of the proposed method in online learning, when more training data become available in a streaming fashion. To that end, we first consider

fixed filter TT-ranks ($b = 0.1$) and, accordingly, a fixed number of approximately $5,000$ learning parameters. We consider that a new batch of training data ($1,000$ examples per class) streams in every 300 training epochs (i.e., at epochs $1, 301, 601, 901, 1201,$ and 1501). In Fig. 5 we observe that the streaming training data help the network increase its performance. However, since the network capacity is limited due to the small number of learning parameters, the performance curves seem to reach a plateau after $1,200$ epochs. For example, the SSIM performance plateaus at the low value of 0.6, which demonstrates that the network has under-fitted the training data. It is therefore clear that, as new training data become available, the network capacity has to increase in order to allow for higher performance.

Next, we repeat this experiment, this time increasing the TT-ranks (and, thus, the network capacity) every time new training data batch streams in. Specifically, at epochs $1, 301, 601, 901, 1201,$ and 1501 b is tuned to the values $0.1, 0.3, 0.5, 0.7, 0.9,$ and 1, respectively. In Fig. 6 we plot reconstruction performance (PSNR on the left and SSIM on the right) at each training epoch (gray curve) and after training on each new/streaming batch (blue curve). The gray curves naturally exhibit performance dips as training switches from one streaming batch to another. These dips result from the newly introduced and randomly initialized untrained parameters in the TT-cores that have been augmented by the TT-rank increase. The secondary vertical axis presents the corresponding number of learning parameters at each epoch. Comparing Fig. 6 with Fig. 5, it is clear that the progressive increase of the TT-ranks allows for significantly superior learning. In Fig. 7 we repeat the same experiment on the MNIST dataset. This time we consider just 5 streaming batches, received at epochs $1, 301, 601, 901,$ and 1201. Once again, tuning the TT-ranks shows significant performance enhancement.

We observe dips in PSNR (per epoch) and SSIM (per epoch) for both datasets as training switches from one streaming batch to the next one. These dips result from the newly introduced and randomly initialized untrained parameters in the TT-cores corresponding to the increased rank values. We also observe an overall increasing trend in PSNR (per batch) and SSIM (per batch) performance while testing on the same dataset. This illustrates the ability of the proposed CAE-TT method to increase the learning capacity of the network by increasing the ranks of the underlying TT-cores of the convolution and transpose-convolution filters as new training data batches are processed.

4.1.2 Denoising

In this study, we repeat the above online learning experiment, but this time for noisy data. Specifically, we consider

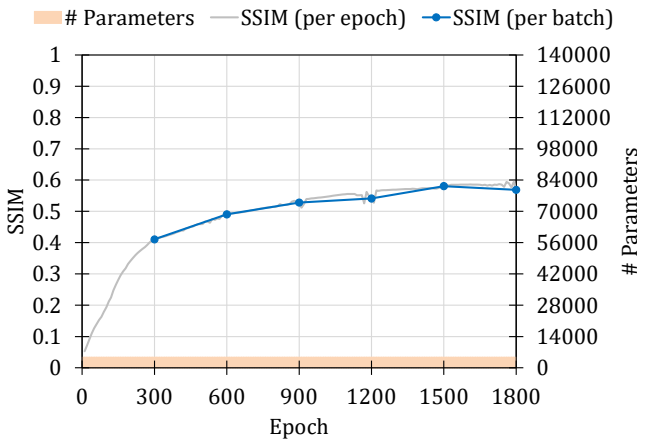

Figure 5: SSIM at each epoch, for fixed TT-rank compression coefficient $b = 0.1$.

Method	SSIM	Num. Parameters
NLM	0.5440	
BM3D	0.5593	
CAE-NWT	0.8272	$26,664$
CAE-NWT	0.8725	$59,706$
Proposed CAE-TT ($b = 0.5$)	0.8714	$39,304$

Table 1: Comparison of SSIM performances in image denoising experiment.

that half of the pixels in each training image are additively corrupted by white Gaussian noise with mean 0 and variance 0.6 and then clipped back in the range $[0, 1]$. In Figs. 8 and 9, we plot PSNR and SSIM at each epoch along with the number of parameters on the secondary vertical axis. Our observations are similar to those of our reconstruction studies in Section 4.1.1 but this time the performance curves plateau to lower values, due to the noise corruption. Interestingly, this plateau can be reached with as few as $70,000$ learning parameters.

In Table 1, we compare the SSIM performance of Non-Local Means (NLM) [22], Block-Matching and 3D filtering (BM3D) [23], CAE with network-width tuning (CAE-NWT), and the proposed CAE-TT (for $b = 0.5$) on the MNIST dataset. NLM and BM3D are standard and popular non-deep learning methods for image denoising. We observe that both CAE methods outperform the non-deep learning counterparts. Also, we observe that CAE-TT offers similar performance to CAE-NWT, but for far fewer learning parameters.

In Figs. 10 and 11, we show examples of clean and corrupted MNIST and FMNIST images, respectively, as well as denoised reconstructions by means of NLM, BM3D, CAE-NWT ($59,706$ parameters), and the proposed CAE-

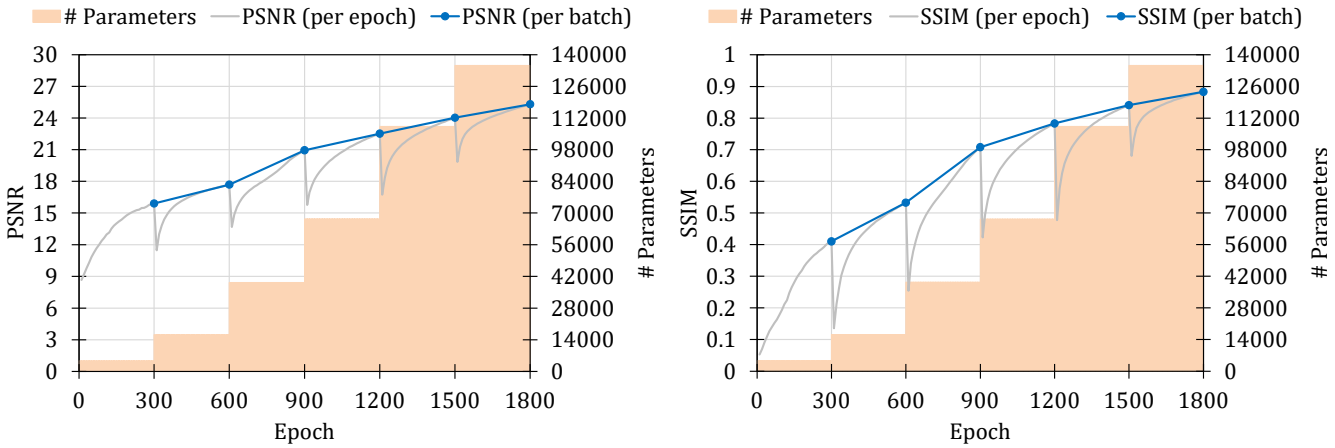

Figure 6: PSNR (left) and SSIM (right) at each epoch, for increasing TT-ranks (reconstruction on the FMNIST dataset).

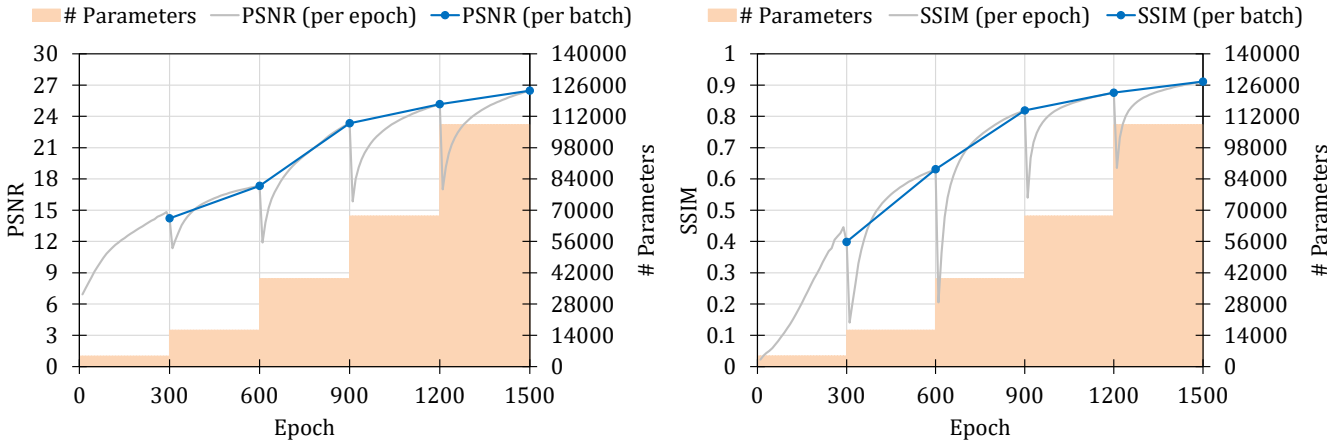

Figure 7: PSNR (left) and SSIM (right) at each epoch, for increasing TT-ranks (reconstruction on the MNIST dataset).

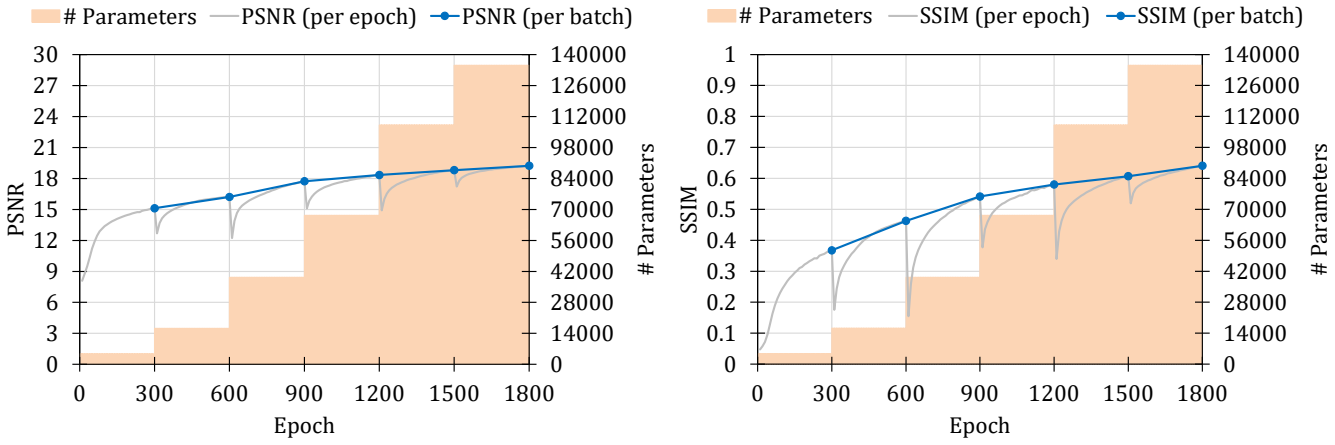

Figure 8: PSNR (left) and SSIM (right) at each epoch, for increasing TT-ranks (denoising on the FMNIST dataset).

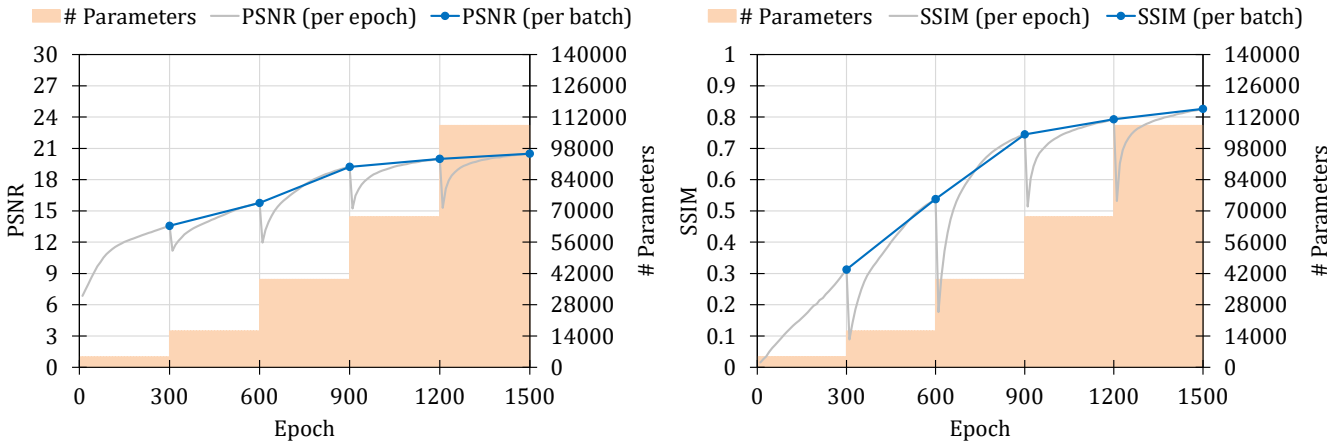

Figure 9: PSNR (left) and SSIM (right) at each epoch, for increasing TT-ranks (denoising on the MNIST dataset).

TT (39, 304 parameters). The CAE methods exhibit similar performance (with the proposed CAE-TT using 34% fewer parameters than CAE-NWT), significantly outperforming the non-deep-learning counterparts.

4.2. Conclusion

We introduced CAE-TT: a convolutional auto-encoder the filters of which have tunable Tensor-Train (TT) structure. The proposed method allows for tuning the learning parameters of the network without changing its architecture. The number of parameters can be reduced in the case of limited data in order to avoid over-fitting. They can also increase, as more training data become available, in order to increase the network capacity and facilitate learning improvements. Our numerical studies demonstrate that adjusting the number of learning parameters by means of TT-rank tuning can attain higher performance than adjusting them by means of network-width tuning.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.

[2] M. Dhanaraj, M. Sharma, T. Sarkar, S. Karnam, D. G. Chachlakis, R. Ptucha, P. P. Markopoulos, and E. Saber, "Vehicle detection from multi-modal aerial imagery using YOLOv3 with mid-level fusion," in *Proc. SPIE Defense Commercial Sens.*, (Anaheim, CA), pp. 1139506:1–1139506:11, May 2020.

[3] M. Sharma, M. Dhanaraj, S. Karnam, D. G. Chachlakis, R. Ptucha, P. P. Markopoulos, and E. Saber, "YOLOrs: Object detection in multimodal remote sensing imagery," *IEEE JSTARS*, vol. 14, pp. 1497–1508, 2020.

[4] M. Sharma, P. P. Markopoulos, and E. Saber, "YOLOrs-lite: A lightweight CNN for real-time object detection in remote-sensing," in *IEEE IGARSS*, p. accepted to appear, 2021.

[5] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-Decoder with atrous separable convolution for semantic image segmentation," in *Proc. ECCV*, pp. 801–818, 2018.

[6] B. Du, W. Xiong, J. Wu, L. Zhang, L. Zhang, and D. Tao, "Stacked convolutional denoising auto-encoders for feature representation," *IEEE trans. on cybernetics*, vol. 47, no. 4, pp. 1017–1027, 2016.

[7] A. Tewari, M. Zollhofer, H. Kim, P. Garrido, F. Bernard, P. Perez, and C. Theobalt, "MoFA: Model-based deep convolutional face autoencoder for unsupervised monocular reconstruction," in *Proc. IEEE ICCV Workshops*, pp. 1274–1283, 2017.

[8] A. Bifet, R. Gavaldà, G. Holmes, and B. Pfahringer, *Machine learning for data streams: with practical examples in MOA*. MIT press, 2018.

[9] D. Sahoo, Q. Pham, J. Lu, and S. C. Hoi, "Online deep learning: Learning deep neural networks on the fly," *arXiv preprint arXiv:1711.03705*, 2017.

[10] Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin, "Compression of deep convolutional neural networks for fast and low power mobile applications," *arXiv preprint arXiv:1511.06530*, 2015.

[11] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky, "Speeding-up convolutional neural networks using fine-tuned CP-decomposition," *arXiv preprint arXiv:1412.6553*, 2014.

[12] I. V. Oseledets, "Tensor-train decomposition," *SIAM J. Scientific Comput.*, vol. 33, no. 5, pp. 2295–2317, 2011.

[13] T. Garipov, D. Podoprikhin, A. Novikov, and D. Vetrov, "Ultimate tensorization: compressing convolutional and FC layers alike," *arXiv preprint arXiv:1611.03214*, 2016.

Figure 10: (a) Examples of MNIST images. (b)-(e) Image denoising examples (top row: corrupted; bottom row: denoised) using methods: (b) NLM; (c) BM3D; (d) CAE-NWT (59, 706 parameters); (e) CAE-TT (39, 304 parameters).

Figure 11: (a) Examples of FMNIST images. (b)-(e) Image denoising examples (top row: corrupted; bottom row: denoised) using methods: (b) NLM; (c) BM3D; (d) CAE-NWT (59, 706 parameters); (e) CAE-TT (39, 304 parameters).

[14] Y. Panagakis, J. Kossaifi, G. G. Chrysos, J. Oldfield, M. A. Nicolaou, A. Anandkumar, and S. Zafeiriou, "Tensor methods in computer vision and deep learning," *Proceedings of the IEEE*, vol. 109, no. 5, pp. 863–890, 2021.

[15] J. Su, W. Byeon, J. Kossaifi, F. Huang, J. Kautz, and A. Anandkumar, "Convolutional tensor-train LSTM for spatio-temporal learning," *arXiv preprint arXiv:2002.09131*, 2020.

[16] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[17] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[18] L. Li, W. Yu, and K. Batselier, "Faster tensor train decomposition for sparse data," *arXiv preprint arXiv:1908.02721*, 2019.

[19] Q. Zhang, L. T. Yang, Z. Chen, and P. Li, "A tensor-train deep computation model for industry informatics big data feature learning," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 7, pp. 3197–3204, 2018.

[20] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980v9 (ICLR, May 2015,* San Diego, CA*)*, Jan. 2017.

[21] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Trans Image Process*, vol. 13, no. 4, pp. 600–612, 2004.

[22] A. Buades, B. Coll, and J.-M. Morel, "Non-local means denoising," *Image Processing On Line*, vol. 1, pp. 208–212, 2011.

[23] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian, "Image denoising by sparse 3-d transform-domain collaborative filtering," *IEEE Trans Image Process*, vol. 16, no. 8, pp. 2080–2095, 2007.