Compressibility of WGANs in a Distributed Learning Environment

Luka Lukač, Damjan Strnad, Štefan Horvat
University of Maribor
Faculty of Electrical Engineering and Computer Science
Koroška cesta 46, 2000 Maribor, Slovenia
{luka.lukac, damjan.strnad, stefan.horvat}@um.si

Abstract. *This work presents a method for compressing Wasserstein generative adversarial networks (WGANs) in a distributed learning environment to reduce network load and server memory usage. Instead of transferring raw data from edge devices to a central server, a WGAN is trained locally on each edge device. The generator part of a trained WGAN is then pruned according to the lottery ticket hypothesis, and compressed using a general-purpose codec. Experimental results demonstrate that the generator can be compressed by up to 15x and still retain the ability to generate high-quality samples.*

Keywords. machine learning, generative adversarial network, Wasserstein distance, lottery ticket hypothesis, data compression

1 Introduction

In the recent era, a rapid development of low-cost sensor devices and their widespread deployment have led to the collection of vast amounts of data. This presents a significant challenge for centralized machine learning (ML) approaches, as transferring massive datasets from edge devices to the server can be resource intensive (Margara et al., 2023). Furthermore, the data distribution across clients can be non-IID in a real-world environment, therefore, it is essential to capture local data distributions of all clients. To address these issues, a special type of the distributed learning paradigm has been proposed (X. Cao et al., 2023), in which separate clients train their local models and send them to the server where they are aggregated into a global model.

Generative ML models that enable generating synthetic data are often used for reducing the communication between clients and the server within a distributed learning environment (Kasturi and Hota, 2023; Merugu and Ghosh, 2003; Sajjadi Mohammadabadi et al., 2024). The idea behind the approach is that the generative models are trained on edge devices and transferred to the server instead of the raw data. This way, the network load and the server memory consumption can be greatly reduced. Aggregated generative models on the server enable on-the-fly generation of data that resembles the local samples of clients, which can significantly increase the performance of trained models (Wijesinghe et al., 2024).

Modern generative ML models are becoming increasingly more complex (C. Li et al., 2016), which presents a significant burden on the communication channels in a distributed setting. Several model compression techniques for ML models were proposed to reduce the memory consumption and network loads (Z. Li et al., 2023), including model pruning (He et al., 2019; Yang et al., 2017), parameter quantisation (Wu et al., 2016), low-rank decomposition (H. Li et al., 2023), and knowledge distillation (Qin et al., 2021; Tung and Mori, 2019). One of the most promising and efficient model pruning techniques is known as the lottery ticket hypothesis (Frankle and Carbin, 2019). In many cases, this method is able to prune more than 90% of neural network (NN) parameters, and obtain the so called winning tickets (Malach et al., 2020). Pruning by itself does not reduce the model size, as it only sets the redundant parameters to zero (Z. Li et al., 2023). However, this reduces the information entropy of the data, and consequently, enables a more efficient model compression. Therefore, compressing the obtained winning tickets from the locally trained generators using general-purpose data compression algorithms before sending them to the server can significantly reduce network load and improve communication efficiency.

The main contributions of this paper are:

- A novel approach for compression of WGAN generators using the lottery ticket pruning and general-purpose compression methods.
- Experimental work, which indicates the efficiency of the proposed compression scheme for WGANs, having a negligible impact on the generator performance.

The remainder of this paper is structured in the following way: Section 2 describes the existing methods for compression of NN models along with the related generative adversarial networks (GANs). Section 3 explains the methodology behind the proposed Wasserstein GAN (WGAN), the lottery ticket extraction, and

the pruned model compression. Section 4 summarises the results and analyses them, while Section 5 concludes the paper and presents future work.

2 Related Work

This section consists of two parts. Firstly, the existing methods for compression of neural network models are reviewed. After that, the basic principles of GANs are presented.

2.1 Compression of Neural Network Models

The most common and earliest methods for the compression of NNs are known as pruning methods. Their main principle is to remove components inside an NN that have negligible impact on its performance (Z. Li et al., 2023). From the early days of weight decay (Hanson and Pratt, 1988), many different pruning methods have been developed. Structured pruning methods typically remove redundant channels in separate layers of the NN (Luo et al., 2019; Xiang et al., 2021) where the importance of each channel is calculated using different heuristics (Chang et al., 2022; Kuang et al., 2022; Q. Li et al., 2022; Lin et al., 2018). On the other hand, unstructured pruning methods set separate, unrelated NN parameters to zeros (Han et al., 2016; Molchanov et al., 2019). Therefore, such pruned NNs are not compressed by itself after the pruning, and their representation needs to be compressed with data compression algorithms (Frankle and Carbin, 2019).

Parameter quantisation lowers the precision of internal NN parameters (Z. Li et al., 2023), and, consequently, reduce the total NN storage space. Some methods perform the quantisation after the training phase (Cai et al., 2020; Fang et al., 2020; Shomron et al., 2021) while others quantise the NN parameters during each forward and backward propagation of the training (Rastegari et al., 2016; Tailor et al., 2021). The advantage of the first group of methods is simplicity and high-speed quantisation, while the post-training quantisation methods yield more accurate results (Z. Li et al., 2023).

Low-rank decomposition approximates the weight matrix of the NN with the product of lower-rank matrices (H. Li et al., 2023). Although some decomposition methods deal with fully connected layers (Denil et al., 2013; Lu et al., 2017; Yu et al., 2017), most of the procedures for low-rank decomposition operate on convolutional layers (Denton et al., 2014; S. Lee et al., 2021; Lin et al., 2019; Rigamonti et al., 2013). Generally, such form of model compression removes redundancy and is applicable to a wide variety of NN architectures. It may, however, be less effective in state-of-the-art networks due to common occurrences of 1x1 convolution filters, which cannot be efficiently decomposed (Z. Li et al., 2023).

Knowledge distillation is a form of model compression where a smaller student network is trained using the results of a large pre-trained teacher network (Gou et al., 2021). Different approaches for training such teacher-student architectures were proposed. The simplest methods use the output of the last layer of the teacher network and train the student network to mimic such outputs (Ba and Caruana, 2014; Hinton et al., 2015). On the other hand, more advanced distillation models also match outputs of intermediate layers and/or their relations to train the student (S. H. Lee et al., 2018; Yim et al., 2017), which can increase the performance of the student model.

2.2 Generative Adversarial Networks

A generative adversarial network (GAN) is a popular generative ML model that was proposed in 2014 (Goodfellow et al., 2014). It consists of two NNs: generator, which is trained to produce samples that resemble the original data distribution, and discriminator that evaluates the realism of samples (either real or artificially generated). The main goal of the alternating training of both models is to train the generator to the point where the discriminator can no longer reliably distinguish between real and fake samples. The GAN structure is depicted in Figure 1. Conditional GAN (cGAN) was introduced shortly after the vanilla GAN, providing the user with control over generated samples' classes (Mirza and Osindero, 2014).

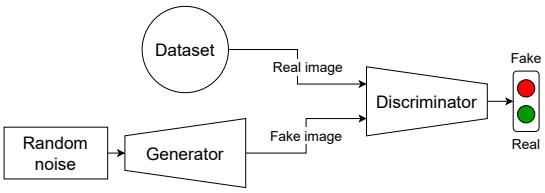

Figure 1. Structure of a GAN.

One of the most common problems that can occur during the training of GANs is mode collapse (Ding et al., 2022). Different ways of tackling this issue were proposed in the past (Bhagyashree et al., 2020; Dai et al., 2024; Durall et al., 2020). A major improvement in training of GANs was the introduction of Wasserstein GAN (WGAN) (Arjovsky et al., 2017), which uses the Wasserstein distance instead of the conventional divergence-based losses. Consequently, the alternating training process of a discriminator (often referred to as a critic) and a generator leads to greater stability and reduces the likelihood of a mode collapse. Due to their great performance and ability to generate a wide variety of data types, WGANs are successfully used in many modern applications (J. Li et al., 2020; Liu et al., 2023; Vo et al., 2024; Wang et al., 2019).

3 Method

This section consists of three parts: firstly, the principles behind the WGAN are described, after that, the lottery ticket pruning is explained, and lastly, the compression techniques of pruned models are presented.

3.1 WGAN

Let G_i be a generator, and let C_i be a critic of a conditional WGAN that belongs to the client $c_i \in \mathcal{C}$. The main goal of G_i is to fool C_i by generating samples that resemble the real data (i.e., critic assigning high scores to fake samples). During the training, G_i is optimised based on the generator loss function $\mathcal{L}_{G_i}$ in Equation 1.

$$\mathcal{L}_{G_i} = -\mathbb{E}_{z\sim\mathcal{N}(0,I)}\left[C_i(G_i(z|y)|y)\right] \tag{1}$$

where z is a latent vector sampled from the standard Gaussian distribution $\mathcal{N}(0, I)$, $\sim$ is the sampling operation, and y is the class label.

The critic C_i estimates the Wasserstein distance between the real and generated data distributions. The main idea is that a well-trained C_i should produce higher scores for real samples and lower values for fake samples. Additionally, the critic loss function $\mathcal{L}_{C_i}$ should impose 1-Lipschitz constraint (bounding the gradient to range $[0, 1]$) (Zhang et al., 2020). This constraint is enforced with a gradient penalty technique (Gulrajani et al., 2017). The critic loss function $\mathcal{L}_{C_i}$ is given in Equation 2.

$$\mathcal{L}_{C_i} = \mathbb{E}_{x\sim\mathbb{P}_r^i}\left[C_i(x|y)\right] - \mathbb{E}_{z\sim\mathcal{N}(0,I)}\left[C_i(G_i(z|y)|y)\right] + \lambda\cdot\mathbb{E}_{\hat{x}}\left[\left(\left\|\nabla_{\hat{x}}C_i(\hat{x}|y)\right\|_2 - 1\right)^2\right] \tag{2}$$

where x represents a real data sample from the i-th client's data distribution $\mathbb{P}_r^i$, λ the strength of the gradient penalty term, and $\hat{x}$ a synthetic sample interpolated between a real and a fake sample.

The training loop of the WGAN with gradient penalty is presented in Algorithm 1. The training is performed for N iterations (Line 8), during which the G_i and C_i are trained in alternating cycles. Usually, C_i is trained for R iterations (Line 10) where $\mathcal{L}_{C_i}$ is calculated in Line 12 using real samples x, latent vectors z, and class labels y. The parameters of C_i are recalculated in Line 13. After that, in Line 16, $\mathcal{L}_{G_i}$ is calculated using the newly sampled latent vectors z. The parameters of G_i are updated in Line 17. After the training is performed for N iterations, the trained G_i and C_i are returned as a result of the function in Line 19.

Algorithm 1 Training loop of the WGAN with gradient penalty on $c_i \in \mathcal{C}$.

1: **function** WGAN-TRAIN(G_i, C_i, N, B, R, η_G, η_C, λ)
2: ▷ N: number of iterations
3: ▷ B: batch size
4: ▷ R: no. of critic iterations per generator iteration
5: ▷ η_G, η_C: generator and critic learning rate
6: ▷ λ: gradient penalty term strength
7:
8: **for** N iterations **do**
9: $x, y \leftarrow$ LoadRealSamples(B)
10: **for** R iterations **do**
11: $z \leftarrow$ SampleLatentVectors(B)
12: $\mathcal{L}_{C_i} \leftarrow$ CriticLoss(x, z, y, λ) ▷ Eq. 2
13: $C_i \leftarrow$ Backpropagation($\mathcal{L}_{C_i}$, η_C)
14: **end for**
15: $z \leftarrow$ SampleLatentVectors(B)
16: $\mathcal{L}_{G_i} \leftarrow$ GeneratorLoss(z, y) ▷ Eq. 1
17: $G_i \leftarrow$ Backpropagation($\mathcal{L}_{G_i}$, η_G)
18: **end for**
19: **return** G_i, C_i
20: **end function**

3.2 Lottery Ticket Pruning

The lottery ticket hypothesis states that a dense, randomly initialised NN contains a subnetwork that can match the accuracy of the original network after training for at most the same number of iterations. To prune models efficiently, non-pruned parameters should be reset to their initial values and retrained after each pruning iteration (Frankle and Carbin, 2019).

Let $f(x, \theta)$ be a NN with initial parameters $\theta \in \mathbb{R}^d$. The network achieves accuracy a after N iterations. There exists a subnetwork $f(x, M \odot \theta)$ with mask M that achieves accuracy a' after at most N' iterations where $\odot$ denotes elementwise multiplication. The lottery ticket hypothesis predicts the existence of a subnetwork:

- $\|M\|_1 \ll d$
  The subnetwork will always have fewer parameters.
- $a' \geq a$
  The subnetwork will have comparable accuracy.
- $N' \leq N$
  The subnetwork will learn in a comparable number of NN training iterations.

The subnetwork $f(x, M \odot \theta)$ is called the winning ticket (Frankle and Carbin, 2019), and is extracted by the iterative pruning procedure shown in Algorithm 2. The default initialisation of the generator and critic parameters, denoted as G_i^0 and C_i^0, is remembered in Line 7. The pruning mask M is initialised to ones in Line 8, and determines which generator parameters have been pruned (i.e., mask value of zero denotes a

pruned parameter). After the initialisation, iterative pruning of G_i is performed for P iterations. The intensity of pruning is determined by the pruning rate δ, which specifies the percentage of remaining parameters that are pruned in each iteration. This means that the total share ρ of pruned parameters after p iterations is given by:

$$\rho = 1 - (1 - \delta)^p \tag{3}$$

In Line 11, the parameters of G_i are reset to the default initialisation according to M, where the previously pruned parameters are set to zero and frozen during subsequent training. The C_i parameters are always reset to the default initialisation and remain unaffected by the pruning mask, since only G_i is being pruned. The models are retrained for a fixed number of training iterations in Line 12. After training, the pruning is performed in Line 13 by sorting the parameters by increasing magnitude, and marking the share ρ of the lowest for pruning by setting the corresponding mask values to zero. After P pruning iterations, the final G_i is declared a winning ticket.

Algorithm 2 Iterative pruning lottery ticket algorithm.

1: **function** ITERATIVELOTTERYTICKET(G_i, C_i, P, δ)
2: ▷ G_i: initialised generator model
3: ▷ C_i: initialised critic model
4: ▷ P: no. of lottery ticket pruning iterations
5: ▷ δ: pruning rate
6:
7: G_i^0, $C_i^0 \leftarrow$ GetInitParams(G_i, C_i)
8: $M \leftarrow$ InitPruningMask()
9: **for** $p \leftarrow 1$ **to** P **do**
10: $\rho \leftarrow 1 - (1 - \delta)^p$ ▷ Eq. 3
11: G_i, $C_i \leftarrow$ ResetToInitParams(G_i^0, C_i^0, M)
12: G_i^e, $C_i^e \leftarrow$ WGAN-Train(G_i, C_i) ▷ Alg. 1
13: G_i, C_i, $M \leftarrow$ Prune(G_i^e, C_i^e, M, ρ)
14: **end for**
15: **return** G_i
16: **end function**

There are two approaches to model pruning. The first approach is local pruning where the proportion ρ of parameters is pruned in each layer separately. This technique is simple and stable, but can prune parameters with larger magnitudes, potentially resulting in reduced model performance. The second approach is global pruning where the parameters are pruned with respect to the global magnitude order. This can lead to higher efficiency and flexibility, but if layers have low numbers of parameters, a layer crash can appear, where all layer parameters are set to zero and the pruned network becomes unusable (Frankle and Carbin, 2019).

3.3 Compression of Pruned Models

The obtained winning ticket model $f(x, M \odot \theta)$ has the same size as the original, non-pruned G_i. However, as it contains many zero parameters (depending on the number of pruning iterations P), the information entropy of $f(x, M \odot \theta)$ is significantly lower. Low information entropy signifies that the pruned $f(x, M \odot \theta)$ is much more compressible than G_i. Consequently, $f(x, M \odot \theta)$ is compressed with an arbitrary general-purpose data compression algorithm, such as zlib (Gailly, Jean-loup and Adler, Mark, 2004), LZMA ("XZ Utils", 2025), bzip2 ("bzip2 and libbzip2", 2025), or paq9a (Mahoney, 2025).

4 Results

The purpose of the experiments was to examine how WGANs can be compressed within a distributed learning environment. The experiments are demonstrated for a single client c_i yet the proposed training and compression pipeline can easily be re-used on multiple clients. In the continuation of this section, the results of the WGAN training, the lottery ticket pruning, and the compression of the pruned model using general-purpose compressors are presented in detail.

The experiments were conducted on our own vector dataset named **H**and**w**ritten **C**haracter **D**ataset (HWCD). As per its name, it consists of 26 handwritten letters of the English, 3 additional letters of the Slovenian alphabet (Č, Š, and Ž), and digits from 0 to 9. Altogether, the HWCD contains 3,314 handwritten character samples across 39 classes. Each sample consists of 10 consecutive vectors with 3 respective components (movement by x-axis, movement by y-axis, and pen down/up). Examples of the character classes that belong to the HWCD are shown in Figure 2.

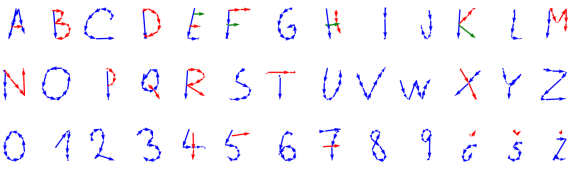

Figure 2. The set of characters in the HWCD. The blue vector color symbolises the first pen stroke, the red color denotes the second stroke, and the green color the third stroke.

The generator G_i that was used in the WGAN training consists of a fully connected layer followed by two 1D convolutional layers. None of the generator layers uses biases. On the other hand, the critic C_i is a multilayer perceptron composed of three fully connected layers. The training hyperparameters of the WGAN are collected in Table 1. The WGAN was implemented in Python 3.11 using the module PyTorch (Paszke et al., 2019).

Table 1. The hyperparameters for the training of WGAN.

Hyperparameter	Symbol	Value
No. of iterations	N	150,000
Batch size	B	256
Critic/generator iterations	R	256
Generator learning rate	η_G	0.0001
Critic learning rate	η_C	0.0001
Gradient penalty term	λ	10

The outputs of the trained WGAN are displayed in Figure 3. It can be seen that all generated samples are recognisable and resemble the real samples from the HWCD, although some samples carry different characteristic that the ones depicted in Figure 2 due to the diverse set of training samples (e.g., number of strokes and distinct local shapes).

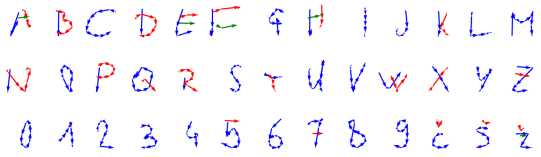

Figure 3. Results of a trained WGAN.

Both pruning approaches yielded similar results, with global pruning achieving marginally superior performance. Therefore, the global pruning method was used throughout the experiments. As critics do not need to be compressed and transferred to the server in a distributed environment, pruning was applied only to G_i. The pruning rate δ of 20% was chosen based on a series of preliminary tests. This means that approximately half of generator weights were pruned after 5 iterations, and approximately 90% of them were frozen after 10 iterations.

In Figure 4, outputs of a pruned WGAN are visualised after different numbers of iterations P. It can be observed that the ability to generate artificial samples that resemble the real samples is pertained even when the pruning rate ρ is high. The generator performance starts to gradually deteriorate after many pruning iterations as some generated characters become unrecognisable (e.g., Q, X, and 8) in Figure 4f. The generator is still able to produce realistic samples even when over 97% of weights are pruned, indicating high efficiency of such approach (Figure 4e).

The generator model is compressed after each iteration of the lottery ticket pruning in order to observe the compressibility of the pruned model. The compression ratios of the generator with the general-purpose compression methods zlib, LZMA, bzip2, and paq9a are collected in Table 2. It can be observed that in the case of a lower number of iterations P, paq9a is the superior compressor, while in the case of a higher number of iterations, LZMA yields better compression ratios. All in all, it can be seen that the generator model can be pruned to the extend where it can be compressed up to 15x without a significant loss of the ability to produce high-quality samples. This can significantly reduce the network load when transferring the model to the server within a distributed learning environment.

Table 2. Compressibility of the pruned generators. The compression algorithm with the highest compression ratio at each is marked bold.

P	% pruned	Compression ratio			
		zlib	LZMA	bzip2	paq9a
/	0.00%	1.08	1.08	1.05	**1.17**
1	20.00%	1.21	1.24	1.23	**1.33**
2	36.00%	1.43	1.49	1.48	**1.59**
3	48.80%	1.70	1.80	1.79	**1.91**
4	59.04%	2.05	2.19	2.17	**2.30**
5	67.23%	2.50	2.68	2.64	**2.79**
6	73.79%	3.08	3.33	3.27	**3.42**
7	79.03%	3.60	3.98	3.92	**4.06**
8	83.22%	4.16	4.68	4.63	**4.75**
9	86.58%	4.79	5.49	5.41	**5.53**
10	89.26%	5.52	**6.40**	6.30	**6.40**
11	91.41%	6.31	**7.46**	7.32	7.35
12	93.13%	7.16	**8.57**	8.40	8.36
13	94.50%	8.11	**9.83**	9.56	9.50
14	95.60%	9.05	**11.12**	10.84	10.64
15	96.48%	9.96	**12.43**	12.02	11.76
16	97.19%	10.86	**13.74**	13.38	12.88
17	97.75%	11.85	**15.17**	14.72	14.14

5 Conclusion

This paper introduces a new method for compression of WGAN generators inside a distributed learning environment. Trained WGANs on separate clients are iteratively pruned with the lottery ticket pruning procedure in the first step. After that, each WGAN is compressed using one of the state-of-the-art general-purpose compression algorithms. The method was extensively tested on vectorised characters of Handwritten Character Dataset. The experiments indicate that the generators can be highly compressed while their ability of generating real-like samples is preserved, which can significantly decrease the network load in a distributed learning environment.

In the future, it would be beneficial for the method to adapt to client data distribution changes with continual learning and on-the-fly pruning of generator models. Generator models could then be periodically sent to the server and aggregated into a global model. Furthermore, using the differential privacy mechanisms, the method could also be modified to operate within a federated learning environment.

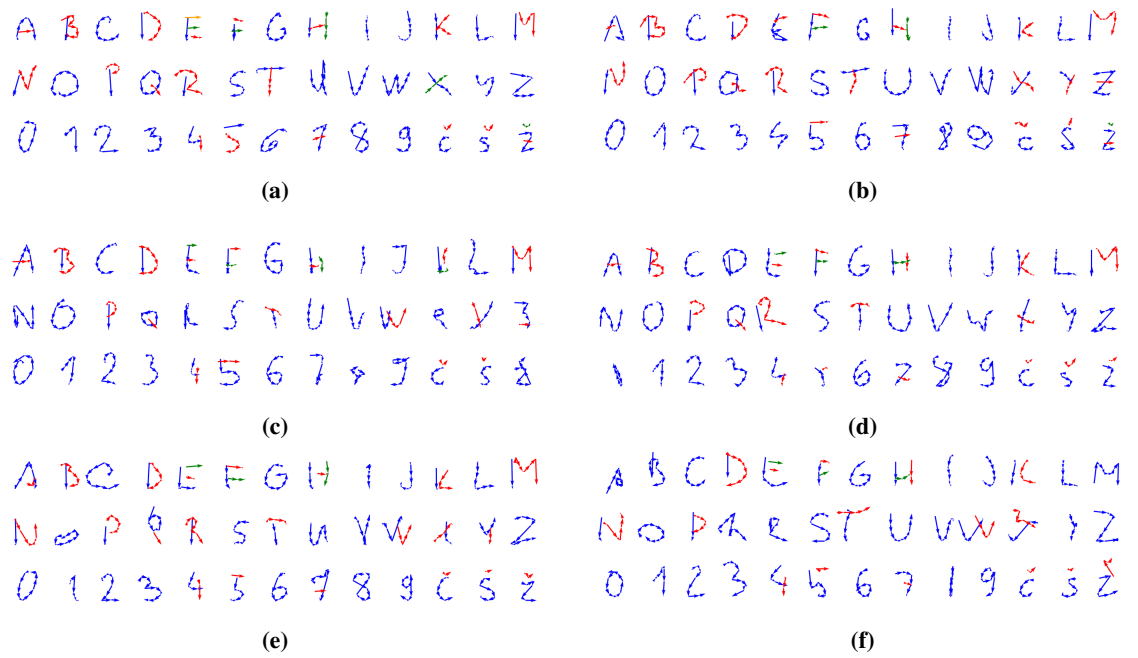

Figure 4. Results of a pruned WGAN: (a) $P = 1$, (b) $P = 5$, (c) $P = 10$, (d) $P = 15$, (e) $P = 16$, (f) $P = 17$.

Acknowledgments

The authors acknowledge the financial support from the Slovenian Research and Innovation Agency under Research Project J2-4458, Young Researcher Funding under Grant 0796-59772, and Research Core Funding P2-0041.

References

Arjovsky, M., Chintala, S., & Bottou, L. (2017). Wasserstein generative adversarial networks. *ICML'17: Proceedings of the 34th International Conference on Machine Learning*, 214–223.

Ba, L. J., & Caruana, R. (2014). Do deep nets really need to be deep? *NIPS'14: Proceedings of the 28th International Conference on Neural Information Processing Systems*, *2*, 2654–2662.

Bhagyashree, V. Kushwaha, & G. C. Nandi. (2020). Study of prevention of mode collapse in generative adversarial network (GAN). *Proceedings of the 2020 IEEE 4th Conference on Information & Communication Technology (CICT)*, 1–6.

Bzip2 and libbzip2 [Available: https://sourceware.org/bzip2/.]. (2025, May).

Cai, Y., Yao, Z., Dong, Z., Gholami, A., Mahoney, M. W., & Keutzer, K. (2020). ZeroQ: A novel zero shot quantization framework. *Proceedings of the 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 13166–13175.

Chang, J., Lu, Y., Xue, P., Xu, Y., & Wei, Z. (2022). Automatic channel pruning via clustering and swarm intelligence optimization for CNN. *Appl. Intell.*, *52*(15), 17751–17771.

Dai, Z., Zhao, L., Wang, K., & Zhou, Y. (2024). Mode standardization: A practical countermeasure against mode collapse of GAN-based signal synthesis. *Appl. Soft Comput.*, *150*, 111089.

Denil, M., Shakibi, B., Dinh, L., Ranzato, M., & de Freitas, N. (2013). Predicting parameters in deep learning. *Proceedings of the 27th International Conference on Neural Information Processing Systems*, *2*, 2148–2156.

Denton, E., Zaremba, W., Bruna, J., LeCun, Y., & Fergus, R. (2014). Exploiting linear structure within convolutional networks for efficient evaluation. *NIPS'14: Proceedings of the 28th International Conference on Neural Information Processing Systems*, *1*, 1269–1277.

Ding, Z., Jiang, S., & Zhao, J. (2022). Take a close look at mode collapse and vanishing gradient in GAN. *Proceedings of the 2022 IEEE 2nd International Conference on Electronic Technology, Communication and Information (ICETCI)*, 597–602.

Durall, R., Chatzimichailidis, A., Labus, P., & Keuper, J. (2020). Combating mode collapse in GAN training: An empirical analysis using Hessian eigenvalues. *arxiv:2012.09673*.

Fang, J., Shafiee, A., Abdel-Aziz, H., Thorsley, D., Georgiadis, G., & Hassoun, J. (2020). Post-training piecewise linear quantization for deep neural networks. *arXiv:2002.00104*.

Frankle, J., & Carbin, M. (2019). The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv:1803.03635*.

Gailly, Jean-loup & Adler, Mark. (2004, December). Zlib compression library [Available: http://www.dspace.cam.ac.uk/handle/1810/3486.].

Goodfellow, I. J., Pouget-Abadie, J., & Mirza, e. a., Mehdi. (2014, December). Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, & K. Q. Weinberger (Eds.), *Advances in Neural Information Processing Systems (NIPS 2014)* (Vol. 27). Curran Associates, Inc.

Gou, J., Yu, B., Maybank, S. J., & Tao, D. (2021). Knowledge distillation: A survey. *Int. J. Comput. Vis.*, *129*(6), 1789–1819.

Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., & Courville, A. (2017). Improved training of Wasserstein GANs. *NIPS'17: Proceedings of the 31st International Conference on Neural Information Processing Systems*, 5769–5779.

Han, S., Mao, H., & Dally, W. J. (2016). Deep compression: Compressing deep neural network with pruning, trained quantization and Huffman coding. *Proceedings of the 4th International Conference on Learning Representations (ICLR 2016)*.

Hanson, S., & Pratt, L. Y. (1988, January). Comparing biases for minimal network construction with back-propagation. In D. Touretzky (Ed.), *NIPS'88: Proceedings of the 2nd International Conference on Neural Information Processing Systems* (pp. 177–185, Vol. 1). MIT Press.

He, Y., Liu, P., Wang, Z., Hu, Z., & Yang, Y. (2019). Filter pruning via geometric median for deep convolutional neural networks acceleration. *Proceedings of the 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 4335–4344.

Hinton, G., Vinyals, O., & Dean, J. (2015). Distilling the knowledge in a neural network. *arXiv:1503.02531*.

Kasturi, A., & Hota, C. (2023). OSGAN: One-shot distributed learning using generative adversarial networks. *J. Supercomput.*, *79*(12), 13620–13640.

Kuang, J., Shao, M., Wang, R., Zuo, W., & Ding, W. (2022). Network pruning via probing the importance of filters. *Int. J. Mach. Learn. Cybern.*, *13*(9), 2403–2414.

Lee, S. H., Kim, D. H., & Song, B. C. (2018). Self-supervised knowledge distillation using singular value decomposition. *Computer Vision – ECCV 2018: 15th European Conference*, 339–354.

Lee, S., Kim, H., Jeong, B., & Yoon, J. (2021). A training method for low rank convolutional neural networks based on alternating tensor compose-decompose method. *Appl. Sci.*, *11*(2), 643.

Li, C., Yang, Y., Feng, M., Chakradhar, S., & Zhou, H. (2016). Optimizing memory efficiency for deep convolutional neural networks on GPUs. *SC '16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 633–644.

Li, H., Wang, Z., Yue, X., Wang, W., Tomiyama, H., & Meng, L. (2023). An architecture-level analysis on deep learning models for low-impact computations. *Artif. Intell. Rev.*, *56*(3), 1971–2010.

Li, J., Niu, K., Liao, L., Wang, L., Liu, J., Lei, Y., & Zhang, M. (2020, July). A generative steganography method based on WGAN-GP. In X. Sun, J. Wang, & E. Bertino (Eds.), *Proceedings of the Artificial Intelligence and Security (ICAIS 2020)* (pp. 386–397). Springer Singapore.

Li, Q., Li, H., & Meng, L. (2022). Feature map analysis-based dynamic CNN pruning and the acceleration on FPGAs. *Electronics*, *11*(18), 2887.

Li, Z., Li, H., & Meng, L. (2023). Model compression for deep neural networks: A survey. *Computers*, *12*(3), 60.

Lin, S., Ji, R., Chen, C., Tao, D., & Luo, J. (2019). Holistic CNN compression via low-rank decomposition with knowledge transfer. *IEEE Trans. Pattern Anal. Mach. Intell.*, *41*(12), 2889–2905.

Lin, S., Ji, R., Li, Y., Wu, Y., Huang, F., & Zhang, B. (2018). Accelerating convolutional networks via global & dynamic filter pruning. *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, 2425–2432.

Liu, M., Wang, Z., Li, H., Wu, P., Alsaadi, F. E., & Zeng, N. (2023). AA-WGAN: Attention augmented Wasserstein generative adversarial network with application to fundus retinal vessel segmentation. *Comput. Biol. Med.*, *158*, 106874.

Lu, Y., Kumar, A., Zhai, S., Cheng, Y., Javidi, T., & Feris, R. (2017). Fully-adaptive feature sharing in multi-task networks with applications in person attribute classification. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 1131–1140.

Luo, J. .-., Zhang, H., Zhou, H. .-., Xie, C. .-., Wu, J., & Lin, W. (2019). ThiNet: Pruning CNN filters for a thinner net. *IEEE Trans. Pattern Anal. Mach. Intell.*, *41*(10), 2525–2538.

Mahoney, M. (2025, May). Data Compression Programs [Available: https://mattmahoney.net/dc/].

Malach, E., Yehudai, G., Shalev-Schwartz, S., & Shamir, O. (2020). Proving the lottery ticket hypothesis: Pruning is all you need. *Proceedings of the 37th International Conference on Machine Learning*, *119*, 6682–6691.

Margara, A., Cugola, G., Felicioni, N., & Cilloni, S. (2023). A model and survey of distributed data-intensive systems. *ACM Comput. Surv.*, *56*(1), 1–69.

Merugu, S., & Ghosh, J. (2003, November). Privacy-preserving distributed clustering using generative models. In X. Wu, A. Tuzhilin, & J. Shavlik (Eds.), *Proceedings of the Third IEEE International Conference on Data Mining* (pp. 211–218). IEEE.

Mirza, M., & Osindero, S. (2014). Conditional generative adversarial nets. *arXiv:1411.1784*.

Molchanov, P., Mallya, A., Tyree, S., Frosio, I., & Kautz, J. (2019). Importance estimation for neural

network pruning. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 11256–11264.

Paszke, A., Gross, S., Massa, F., Lerer, A., & Bradbury, e. a. (2019). PyTorch: An imperative style, high-performance deep learning library. *arXiv:1912.01703*.

Qin, D., Bu, J.-J., Liu, Z., Shen, X., Zhou, S., Gu, J.-J., Wang, Z.-H., Wu, L., & Dai, H.-F. (2021). Efficient medical image segmentation based on knowledge distillation. *IEEE Trans. Med. Imaging*, *40*(12), 3820–3831.

Rastegari, M., Ordonez, V., Redmon, J., & Farhadi, A. (2016, September). XNOR-Net: ImageNet classification using binary convolutional neural networks. In B. Leibe, J. Matas, N. Sebe, & M. Welling (Eds.), *Computer Vision – ECCV 2016* (pp. 525–542). Springer International Publishing.

Rigamonti, R., Sironi, A., Lepetit, V., & Fua, P. (2013). Learning separable filters. *Proceedings of the 2013 IEEE Conference on Computer Vision and Pattern Recognition*, 2754–2761.

Sajjadi Mohammadabadi, S. M., Entezami, M., Karimi Moghaddam, A., Orangian, M., & Nejadshamsi, S. (2024). Generative artificial intelligence for distributed learning to enhance smart grid communication. *Int. J. Intell. Netw.*, *5*, 267–274.

Shomron, G., Gabbay, F., Kurzum, S., & Weiser, U. (2021). Post-training sparsity-aware quantization. *Proceedings of the 35th International Conference on Neural Information Processing Systems*.

Tailor, S. A., Fernandez-Marques, J., & Lane, N. D. (2021). Degree-quant: Quantization-aware training for graph neural networks. *arXiv:2008.05000*.

Tung, F., & Mori, G. (2019). Similarity-preserving knowledge distillation. *Proceedings of the 2019 IEEE/CVF International Conference on Computer Vision (ICCV)*.

Vo, H. V., Du, H. P., & Nguyen, H. N. (2024). APELID: Enhancing real-time intrusion detection with augmented WGAN and parallel ensemble learning. *Comput. Secur.*, *136*, 103567.

Wang, Q., Zhou, X., Wang, C., Liu, Z., Huang, J., Zhou, Y., Li, C., Zhuang, H., & Cheng, J. .-. (2019). WGAN-based synthetic minority over-sampling technique: Improving semantic fine-grained classification for lung nodules in CT images. *IEEE Access*, *7*, 18450–18463.

Wijesinghe, A., Zhang, S., & Ding, Z. (2024). PS-FedGAN: An efficient federated learning framework with strong data privacy. *IEEE Internet Things J.*, *11*(16), 27584–27596.

Wu, J., Leng, C., Wang, Y., Hu, Q., & Cheng, J. (2016). Quantized convolutional neural networks for mobile devices. *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 4820–4828.

X. Cao, T. Başar, S. Diggavi, & Y. C. Eldar, e. a. (2023). Communication-efficient distributed learning: An overview. *IEEE J. Sel. Areas Commun.*, *41*(4), 851–873.

Xiang, Q., Wang, X., Song, Y., Lei, L., Li, R., & Lai, J. (2021). One-dimensional convolutional neural networks for high-resolution range profile recognition via adaptively feature recalibrating and automatically channel pruning. *Int. J. Intell. Syst.*, *36*(1), 332–361.

XZ Utils [Available: https://tukaani.org/xz/.]. (2025, April).

Yang, T.-J., Chen, Y.-H., & Sze, V. (2017). Designing energy-efficient convolutional neural networks using energy-aware pruning. *Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 6071–6079.

Yim, J., Joo, D., Bae, J., & Kim, J. (2017). A gift from knowledge distillation: Fast optimization, network minimization and transfer learning. *Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 7130–7138.

Yu, X., Liu, T., Wang, X., & Tao, D. (2017). On compressing deep models by low rank and sparse decomposition. *Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 67–76.

Zhang, Z., Zeng, Y., Bai, L., Hu, Y., Wu, M., Wang, S., & Hancock, E. R. (2020). Spectral bounding: Strictly satisfying the 1-Lipschitz property for generative adversarial networks. *Pattern Recognit.*, *105*, 107179.