ICT Focus

http://www.ictfocus.org

JOURNAL ARTICLE OPEN ACCESS

CLOS network: A parameter-efficient alternative to linear layers in neural networks

J.Orgil [1*], Z.Nomin-Erdene [1] G.Magvan-Erdene[1], E.Battsetseg[1]
[1] Department of Computer Science, SICT, MUST, Ulaanbaatar, Mongolia
e-mail: orgilsict@must.edu.mn, z.nominna@gmail.com, magvan-erdene@must.edu.mn, battsetseg@must.edu.mn

Abstract

In this paper, we propose utilizing the CLOS network architecture to replace traditional linear layers in deep learning models, including transformers. The CLOS network, commonly used in networking systems, is adapted to neural networks to reduce parameter sizes while maintaining model performance. Our experiments show that the CLOS network achieves the same accuracy and loss as the conventional linear layer, but with fewer parameters. However, this efficiency comes at the cost of increased processing time, which is 1.5x to 3x slower. Despite this trade-off, the CLOS network can be an effective alternative for parameter reduction in various architectures, including large models like transformers.

Keywords: **Linear layer, CLOS network, Transformers, parameter decrease.**

Academic Editor: Orgil Jargalsaikha

Received: date
Accepted: date
Published: date

Publisher's Note: ICTFocus Open Journal stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

1. INTRODUCTION

Linear layers are fundamental building blocks of modern deep learning architectures, including transformers, convolutional neural networks, and fully connected networks. Despite their simplicity and wide usage, linear layers can contribute to significant model

size, which can be a limiting factor in memory-constrained environments or when deploying models at scale. Addressing this issue by reducing the number of parameters while maintaining model performance has become an active area of research.

The CLOS network, initially designed for efficient communication in high-performance networking systems, can potentially replace linear layers in neural networks. The CLOS architecture features a multi-stage interconnection of more miniature switches, which can be adapted to neural network designs to reduce the number of trainable parameters. This approach promises to make neural networks memory-efficient while maintaining comparable accuracy and loss performance.

This work proposes replacing conventional linear layers with a CLOS network-based architecture across different neural network models, including transformers. Our experimental results demonstrate that the CLOS network can significantly reduce the number of parameters in these models without compromising performance metrics such as accuracy and loss. However, this reduction comes at the cost of increased computational time, with our findings showing a 1.5x to 3x slower processing time compared to traditional linear layers.

This paper is organized as follows: Section 2 reviews related work on parameter-efficient neural network designs. Section 3 describes the implementation of the CLOS network within neural network architectures. Section 4 presents the experimental setup and results. Finally, Section 5 discusses the findings and potential future research directions.

2. Related work

Parameter Reduction and Compression Techniques: Methods like pruning and quantization reduce neural network size while maintaining accuracy, as in "Deep Compression" by Han et al. [6], which removes less essential weights, and Jacob et al.'s [11] quantization for hardware-constrained deployment. These focus on compression rather than architectural changes like CLOS networks. Relatedly, neural network compression includes Hinton et al.'s [7] knowledge distillation to transfer performance from large to small models, and Howard et al.'s [8] MobileNets for lightweight convolutional architectures, offering insights applicable to CLOS strategies. Sparse networks and pruning, such as Zhou et al.'s [24] filter pruning for efficient convnets and Louizos et al.'s [13] L0 regularization for sparsity, complement this by creating models with fewer parameters while sustaining performance, potentially enhancing CLOS networks.

Efficient Architectures and Alternatives: Efficient architectures like Szegedy et al.'s [18] Inception and Iandola et al.'s [10] SqueezeNet use factorized convolutions for high performance with reduced parameters, similar to CLOS for efficiency. Alternatives to fully connected layers, including Sridhar et al.'s [17] random projections and low-rank factorization, cut costs in dense layers, aligning with CLOS as substitutes. Structured matrices, per Sindhwani et al.'s [16] approximations (e.g., circulant, Toeplitz) and Cheng et al.'s [4] circulant projections for redundancy exploration, reduce complexity in fully connected layers, mirroring CLOS goals. Weight sharing and low-rank decompositions, such as Zhang et al.'s [3] attention with fusion for model size reduction and Novikov et al.'s [14] tensor decompositions for compression, provide efficient alternatives to linear layers like CLOS.

Memory Efficiency and Dynamic Methods: Memory-efficient methods include Chen et al.'s [2] sub-linear memory training and Huang et al.'s [9] CondenseNet for dynamic connection learning, offering strategies combinable with CLOS. Neural architecture search (NAS) for efficient models, like Tan et al.'s [19] EfficientNet for balanced scaling and Cai et

al.'s [1] ProxylessNAS for hardware optimization, provides insights for comparing CLOS overhead to traditional layers.

Applications to Advanced Models and Acceleration: Applying CLOS networks to machine learning reduces communication and parameters for fast inference, showing promise in adapting traditional architectures for neural efficiency. For Transformers, Vaswani et al.'s [21] self-attention architecture could benefit from CLOS to cut dense layer demands, while Dehghani et al.'s [5] Universal Transformers use shared weights for efficiency, relating to CLOS enhancements. Acceleration techniques include Wang et al.'s [22] SkipNet for dynamic layer skipping to reduce computation, and Ratul et al.'s [15] analysis of frameworks for faster inference, paralleling CLOS trade-offs in processing time.

3. Proposed Method

We propose a novel implementation of the CLOS network architecture for neural networks, which we call the CLOS Layer. This approach aims to reduce computational complexity while maintaining model expressiveness.

3.1. CLOS Layer Architecture

The CLOS Layer is designed as a three-stage network, inspired by the CLOS network topology used in telecommunications. It consists of input, middle, and output switches, represented by weight matrices W_1, W_2, and W_3 respectively.

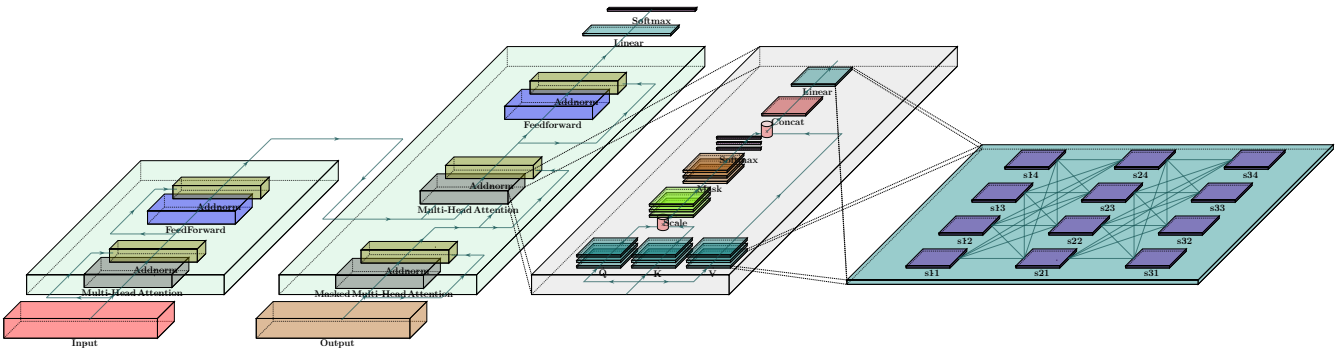

Figure 1. The CLOS Transformer model architecture builds on the standard Transformer design. In a typical Transformer, the attention mechanisms in the encoder and decoder are very similar. Multi-head attention is then expanded using QKV projections and a feed-forward (dense) layer. Each of these QKV projections and the output dense layer consists of linear transformations that encode and store knowledge about the input information. The right side of the figure illustrates the CLOS layer, which replaces the components described in the algorithm section.

In Figure 1 represents Multi-head attention based transformer encoder and decoder model. We illustrated decoders Multi-head attention's Query, Key and Value gates and replaced by CLOS layer. Moreover, we can replace each linear layer of transformer based model including dense layers.

3.2. Algorithm

The forward pass of the CLOS Layer is defined as follows:

1. **Input:** Tensor $x \in \mathbb{R}^{B \times C \times N}$, where B is batch size, C is number of channels (2 or 3), and N is input features

2. Reshape x into (b_{in}, b_1) blocks
3. Compute first stage: $x_1 = xW_1 + b_1$
4. Compute second stage: $x_2 = x_1W_2 + b_2$
5. Compute third stage: $y = x_2W_3 + b_3$
6. Reshape y to (B, C, M), where M is output features
7. **Output:** Tensor $y \in \mathbb{R}^{B\times C\times M}$

Where:

- $W_1 \in \mathbb{R}^{b_{in}\times b_1\times b_2}$
- $W_2 \in \mathbb{R}^{b_1\times b_2\times b_3}$
- $W_3 \in \mathbb{R}^{b_2\times b_3\times b_{out}}$
- b_1, b_2, b_3 are bias vectors (if used)
- $b_{in}, b_1, b_2, b_3, b_{out}$ are switch sizes determined by factorizing input and output feature dimensions

The algorithm supports 2D and 3D inputs, adapting the tensor operations accordingly.

3.3. Complexity Analysis and Non-blocking

The time complexity of the CLOS Layer is $O(N \cdot \max(b_{in}, b_1, b_2, b_3, b_{out}))$, which is lower than the $O(N^2)$ complexity of a standard fully connected layer when the switch sizes are chosen appropriately.

Non-blocking situation: CLOS networks are a type of multistage switching network commonly used in high-performance computing and telecommunications. Named after Charles CLOS, who introduced the concept in 1953, these networks are designed to be non-blocking, meaning that any unused input port can always be connected to any unused output port, regardless of the network's current state.

The non-blocking property of a CLOS network depends on the number of switches in the middle stage. For a network to be strictly non-blocking, the number of middle-stage switches (m) must satisfy the following equation:

$$m \geq 2n - 1 \tag{1}$$

Where:

m - is the number of switches in the middle stage;

n - is the number of input/output ports on each switch in the input and output stages;

The following equation ensures sufficient paths through the network to accommodate all possible input-output connections without blocking. The parameter count for a standard dense linear layer with input dimension d_{in} and output dimension d_{out} is simply:

$$\text{Params}_{\text{dense}} = d_{\text{in}} \cdot d_{\text{out}}. \tag{2}$$

In contrast, the proposed CLOS layer adopts a three-stage structure with a middle dimension m and k switches per stage (where k is typically chosen as $k = \lceil \max(d_{\text{in}}, d_{\text{out}})/m \rceil$ to ensure connectivity).

The total number of parameters in the CLOS layer is:

$$\text{Params}_{\text{CLOS}} = k \cdot m \cdot (d_{\text{in}} + 2m + d_{\text{out}}). \tag{3}$$

This can also be expressed explicitly as:

$$\text{Params}_{\text{CLOS}} = k \cdot (d_{\text{in}} \cdot m + m \cdot m + m \cdot d_{\text{out}}). \tag{4}$$

By selecting a small middle dimension $m \ll \min(d_{\text{in}}, d_{\text{out}})$, the CLOS layer achieves substantial parameter reduction compared to the dense baseline while retaining expressive capacity through its multi-stage routing. Calculation of CLOS parameters shown in Fig. 2

3.4. Implementation Details

The CLOS Layer has been implemented as a custom PyTorch module, leveraging the efficiency of advanced tensor operations, particularly the `einsum` function for optimized matrix multiplications. This implementation autonomously determines the optimal switch dimensions based on the input and output feature spaces while allowing manual configuration when necessary. Furthermore, we have developed a methodology to systematically replace all linear layers within a given model with their CLOS Layer counterparts, subject to a specific constraint. This substitution is executed only when the ratio of input size to output size (or vice versa) does not exceed a factor of 4 due to most model expansion and reduction parameters. This constraint can be formally expressed as:

$$\max\left(\frac{\text{input\_size}}{\text{output\_size}}, \frac{\text{output\_size}}{\text{input\_size}}\right) < 4 \tag{5}$$

This approach ensures that the CLOS Layer is applied judiciously, maintaining model efficiency while exploiting the benefits of the CLOS network architecture in neural network design.

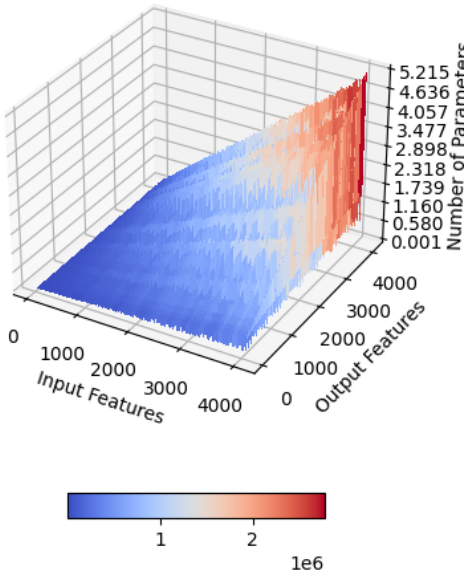

Figure 2. Number of Parameters in CLOS network.

In Fig. 2, in and out features are related to Million parameter size.

4. Experimental setup and Results

We trained traditional linear layer and transformer-related networks. Firstly, our research aimed at Standard small dataset such as MNIST and CIFAR, which can show us the accuracy and processing time performance in small custom models easy to compare traditional linear layers. After that, we changed our model to a deep transformer model such as BERT, VIT and NLLB. The following section will discuss these models' training setups and results.

4.1. Training setup and results in CLOS and Linear on MNIST dataset

We built two custom small models with the same setup, in which both network weights were started from scratch during CPU and GPU training. Therefore, the MNIST dataset image shape is reshaped as $Batch \times 1 \times 784$ that can feed directly into traditional Linear and CLOS layers. Traditional linear and CLOS model layers were set as the same hidden units of $Layer1$, $Layer2$ was $x[784, 512, 256, 128]$ and $Layer3(x, 10)$. SGD optimizer both networks with learning rate 0.001. $ReLU$ activation with $cross\_entroty\_loss$ and training epochs 10. The following figures show that our method, CLOS, and traditional neural networks have no big difference in accuracy and loss in processing time, the same at both networks, in GPU 1.50 and in CPU 3.43 minutes.

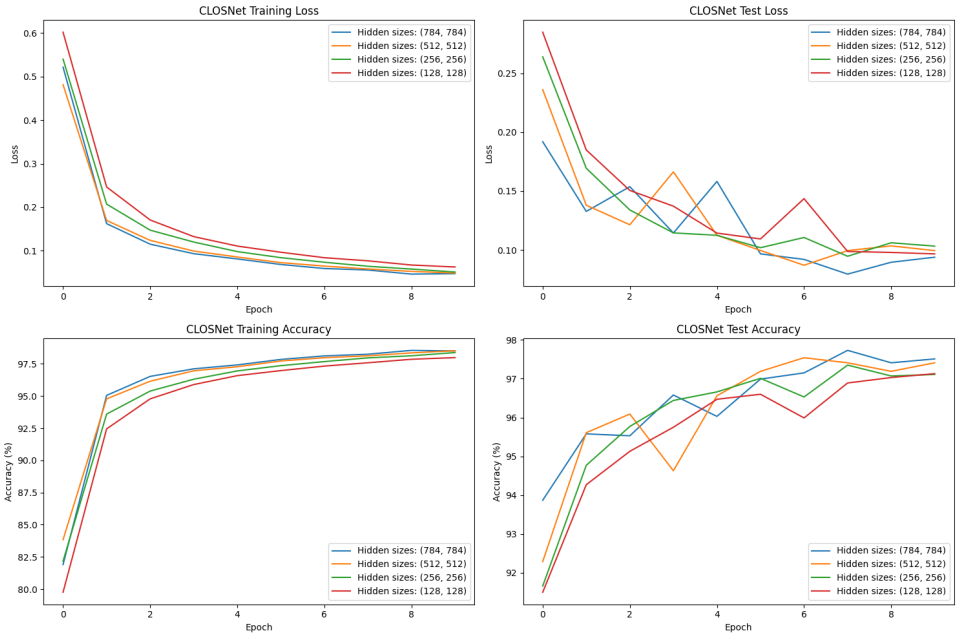

Figure 3. Simulation results for the CLOS network.

4.2. Training setup and results in CLOS and Linear on VIT model

We implemented two parallel models with identical architectures, training both networks from scratch on both CPU and GPU platforms. The implementation details are as follows:

The CIFAR10 dataset was reshaped to accommodate the network architecture, with input tensors formatted as Batch $\times$ 3 $\times$ 224 $\times$ 224 to facilitate direct input into both the traditional Linear and CLOS Transformer attention layers.

The training process utilized the following hyperparameters:

- Batch size: 64

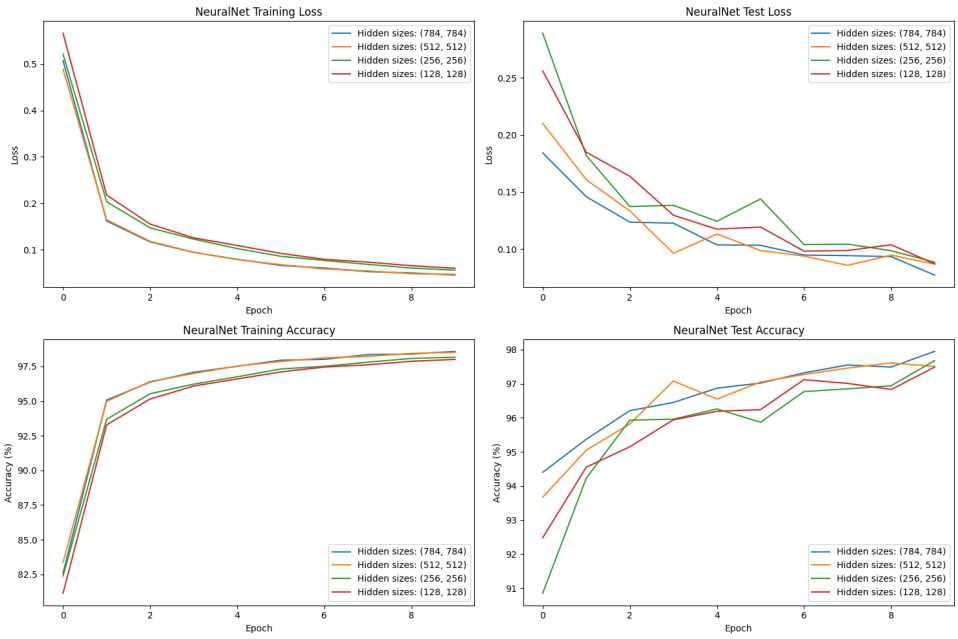

Figure 4. Simulation results for the Neural network.

- Training epochs: 120
- Learning rate: 3×10^{-4}
- Learning rate decay (gamma): 0.7
- Optimizer: Adam optimizer
- Loss function: Cross-entropy loss
- Learning rate scheduler: Step scheduler with step size 1

Performance evaluation was conducted on both computational platforms:

- **GPU Processing Time**: 01:37 minutes per epoch
- **CPU Processing Time**: 03:43 minutes per epoch
- **Linear Layer FLOPs**: 1,229,312
- **Clos Layer FLOPs**: 258,720

The experimental results demonstrated comparable performance between the CLOS and traditional neural network architectures in terms of both accuracy and loss metrics. The processing time remained consistent across both network implementations on their respective platforms.

Both networks demonstrated equivalent convergence characteristics, suggesting that the CLOS architecture maintains computational efficiency while preserving the learning capabilities of traditional linear layers.

4.3. Training setup and results for English–Mongolian Machine Translation Models

In the development of an English–Mongolian machine translation system, we conducted comprehensive experiments using multilingual and large-scale language models, including MBart [12], Qwen [23], NLLB-600M, and NLLB-3.3B [20]. Although these models differ in architecture, they share common mechanisms based on the Transformer framework for encoding, decoding, and generating token sequences. Due to the rich morphology, agglutinative structure, and syntactic characteristics of the Mongolian language—which differs substantially from English—the translation quality can be significantly influenced. Therefore, each model was fine-tuned on a domain-specific Mongolian dataset to adapt the pretrained architecture to the linguistic properties of the target language and to improve translation performance.

The training process utilized the following hyperparameters:

- Batch size: 2 – 16 (depending on model size)
- Training epochs: 3
- Learning rate: $2 \times 10^{-5} - 5 \times 10^{-5}$
- Optimizer: Adam optimizer
- Loss function: Cross-entropy loss
- Accuracy function: 1-WER

TABLE I

English Mongolian translation 1 Million sentence training result on CLOS replaced.

En-Mn / 1M			
Model	**Train Loss**	**Test Loss**	**Test Acc**
MBart	0.6719	0.1638	0.2171
Qwen	0.3619	0.1350	0.3424
NLLB-200-600M	0.7383	0.1435	0.3424
NLLB-200-3.3B	0.4524	0.1239	0.3424

TABLE II

English Mongolian translation 14 Million sentence training result on CLOS replaced.

En-Mn / 14M			
Model	**Train Loss**	**Test Loss**	**Test Acc**
Qwen	1.8535	3.8000	0.025
NLLB-200-600M	0.5018	3.3238	0.020
NLLB-200-3.3B	0.3423	4.0527	0.030

We ran our experiments on English-to-Mongolian machine translation with the CLOS-modified Transformer. On the smaller 1M sentence dataset (Table I), everything looks good: all models get nice low test losses (0.12–0.16) and reasonable accuracy (up to 0.34). Clearly, this amount of data is enough for solid learning. Things change with the bigger 14M dataset (Table II). Training loss stays decent, but test loss shoots way up (over 3.3) and accuracy crashes to almost nothing (0.02–0.03). That's classic underfitting—the

Figure 5. Training curve for the NLLB-200-600M model.

models just haven't been trained long enough to digest all the extra data. The training curve in Figure 5 shows they're still improving steadily. So we took the most balanced performer, NLLB-200-600M, and kept training it on the 14M data using CLOS. After another 15,000 steps, the training loss dropped sharply to 0.1435, which tells us the model is adapting really well to English-Mongolian translation. Bottom line: CLOS performs great on smaller datasets and, given enough training time, handles much larger ones effectively too. This makes it especially useful for low-resource languages like Mongolian. Going forward, the simplest way to get better results is probably just to train longer on the big datasets.

5. Conclusion

Our experiments on MNIST (simple networks) and CIFAR10 (vision transformers) validate CLOS networks as efficient alternatives to traditional linear layers, maintaining performance while reducing parameters. Key findings include:

Parameter efficiency: Comparable accuracy with fewer parameters, ideal for memory-constrained settings.

Performance trade-off: 1.5x–3x slower processing times. Cause of the CLOS 3 stages.

Architecture versatility: Integrates well into various models, including transformers.

Equivalent convergence: Similar learning curves without compromising effectiveness.

Future directions involve optimizing CLOS computational efficiency, exploring hardware-specific implementations, and distilling knowledge from pretrained transformer QKV weights into CLOS layers without training data. This approach promises parameter-efficient architectures for size-constrained deployments.

REFERENCES

[1] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. *ArXiv*, abs/1812.00332, 2018.

[2] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost, 2016.

[3] Yong Cheng, Wei Wang, Wenjie Zhang, Ling Yang, Jun Wang, Huan Ni, Tingzhao Guan, Jiaxin He, Yakang Gu, and Ngoc Nguyen Tran. A multi-feature fusion and attention network for multi-scale object detection in remote sensing images. *Remote Sensing*, 15(8), 2023.

[4] Yu Cheng, Felix X. Yu, Rogerio S. Feris, Sanjiv Kumar, Alok Choudhary, and Shi-Fu Chang. An exploration of parameter redundancy in deep networks with circulant projections. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ICCV '15, page 2857–2865, USA, 2015. IEEE Computer Society.

[5] Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers, 2019.

[6] Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding, 2016.

[7] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. *ArXiv*, abs/1503.02531, 2015.

[8] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017.

[9] Gao Huang, Shichen Liu, Laurens van der Maaten, and Kilian Weinberger. Condensenet: An efficient densenet using learned group convolutions, 06 2018.

[10] 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, 2016.

[11] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference, 06 2018.

[12] Yinhan Liu, Jiatao Gu, Naman Goyal, and et al. Multilingual denoising pre-training for neural machine translation. *Transactions of the Association for Computational Linguistics*, 8:726–742, 2020.

[13] Christos Louizos, Max Welling, and Diederik P. Kingma. Learning sparse neural networks through l0 regularization. In *International Conference on Learning Representations*, 2018.

[14] Alexander Novikov, Dmitry Podoprikhin, Anton Osokin, and Dmitry Vetrov. Tensorizing neural networks. In *Proceedings of the 29th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, page 442–450, Cambridge, MA, USA, 2015. MIT Press.

[15] Ishrak Jahan Ratul, Yuxiao Zhou, and Kecheng Yang. Accelerating deep learning inference: A comparative analysis of modern acceleration frameworks. *Electronics*, 14(15), 2025.

[16] Vikas Sindhwani, Tara Sainath, and Sanjiv Kumar. Structured transforms for small-footprint deep learning. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[17] Sridhar Swaminathan, Deepak Garg, Rajkumar Kannan, and Frederic Andres. Sparse low rank factorization for deep neural network compression. *Neurocomputing*, 398:185–196, 2020.

[18] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions, 2014.

[19] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *ArXiv*, abs/1905.11946, 2019.

[20] NLLB Team, Marta R. Costa-jussà, James Cross, Onur Çelebi, and et al. No language left behind: Scaling human-centered machine translation, 2022.

[21] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[22] Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E. Gonzalez. Skipnet: Learning dynamic routing in convolutional networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[23] An Yang, Anfeng Li, Baosong Yang, and et al. Qwen3 technical report, 2025.

[24] Zhengguang Zhou, Wengang Zhou, Houqiang Li, and Richang Hong. Online filter clustering and pruning for efficient convnets. In *2018 25th IEEE International Conference on Image Processing (ICIP)*, pages 11–15, 2018.

BIOGRAPHIES

Orgil Jargalsaikhan , is an Telecommunication engineer (2009), master of Information Technology (2011) in SICT, MUST and doctor in Systems Innovation (2022) with the Tokushima University of Japan.

Between 2009 and 2018 he was the Lecturer of Telecommunication Department of MUST SICT. He is currently a full associated professor at University of SICT Computer Science department. His areas of interest are: Information and Computer science, Deep learning and Graph networks , Machine learning and Image Processing.

Z.Nomin-Erdene is 4th degree Bachelor student of MUST, SICT, Computer Science department.
G.Magvan-Erdene is Master student of MUST, SICT, Computer Science department.
E.Battsetseg is Graduated MUST, PES, master of Power Electronics (2015), Now Lecturer of Computer Science department. Her areas of interest are: Information and Computer science, Deep learning and Graph networks , Machine learning and Image Processing.