Proceedings of CONF-CDS 2025 Symposium: Application of Machine Learning in Engineering
DOI: 10.54254/2755-2721/2025.AST26567

Lightweight CNN Design Based on Mixup Data Augmentation and Network Pruning

Yuxuan Li[1,2]

[1]*Institute of East China University of Science and Technology, Shanghai, China*
[2]*ECUST, Shanghai, China*
3027435227@qq.com

Abstract. Convolutional Neural Networks (CNNs) have achieved remarkable success in image classification and other vision tasks in recent years. However, their large model size and computational complexity hinder their application in mobile terminals and embedded devices. To address this issue, this paper proposes a lightweight CNN design method that combines Mixup data augmentation and network pruning. The method aims to balance the trade-off between model compression and performance preservation, achieving the maximum model compression while maintaining as much of the original performance as possible. Using the FashionMNIST dataset as the experimental platform, a classification model based on a simplified LeNet structure is constructed. The model is evaluated under four different settings: the standard model, the Mixup-augmented model, the pruned sparse model, and the collaborative model integrating both Mixup augmentation and pruning. The experimental results show that Mixup enhances the model's generalization ability and robustness, pruning significantly reduces the number of parameters, and the combination of both achieves superior lightweight performance while preserving accuracy. This study demonstrates the effectiveness of Mixup and pruning techniques in collaborative optimization and proposes practical optimization strategies for deploying lightweight neural networks in resource-constrained environments.

Keywords: Lightweight Convolutional Neural Network, Mixup Data Augmentation, Network Pruning, Model Compression, Image Classification, FashionMNIST

1. Introduction

Convolutional Neural Networks (CNNs) have achieved remarkable success in computer vision tasks such as image classification, thanks to their exceptional feature extraction capabilities. However, CNN models are typically large in terms of parameters and computational cost, which limits their deployment and application in resource-constrained environments such as mobile devices and edge devices. Therefore, effective model compression without significantly sacrificing performance has become one of the key research directions in recent years.

To achieve a good balance between model accuracy and resource consumption, various structural optimization and training strategies have been proposed, with data augmentation and network pruning being particularly prominent. On one hand, data augmentation, by expanding the

DOI: 10.54254/2755-2721/2025.AST26567

distribution of training samples, helps to improve the model's generalization ability and robustness. Specifically, the Mixup strategy [1], which constructs new samples by linearly interpolating between any two pairs of samples and their labels, has been shown to effectively mitigate overfitting and improve model stability. On the other hand, network pruning [2] reduces the model's parameter size and computational cost by removing redundant weights or neurons, and, when designed appropriately, it can maintain the performance of the original model.

This paper aims to explore the synergistic effect of Mixup data augmentation and network pruning, and to design a lightweight CNN model suitable for resource-constrained environments. The FashionMNIST dataset [3] is selected as the experimental platform, which contains 70,000 grayscale images of clothing, categorized into 10 classes, with image size of 28×28. This dataset offers an appropriate level of complexity and openness for introductory research. Based on the simplified LeNet architecture [4], we construct four comparative models: a standard CNN without augmentation or pruning; a model enhanced by Mixup; a sparse model using pruning strategy; and an optimized model combining both Mixup and pruning.

The main contributions of this paper include: proposing a lightweight CNN construction method that combines Mixup data augmentation and network pruning, achieving joint optimization at both the training data level and model architecture level; constructing a complete and reproducible experimental pipeline based on the PyTorch framework, using publicly available datasets and official APIs, suitable for teaching and prototype validation; and demonstrating through systematic experiments that: Mixup enhances the model's generalization ability, network pruning significantly compresses the model size, and the combination of both achieves a further lightweight effect while maintaining performance. This research provides practical guidance for the future deployment of neural network models on edge devices and offers an actionable example for beginners to understand model compression and training optimization strategies.

The structure of the paper is as follows: Section 2 reviews related research progress; Section 3 introduces the proposed method, including model architecture, Mixup implementation, and pruning strategies; Section 4 presents experimental setup and results analysis; Section 5 discusses research findings and potential improvements; Section 6 concludes the paper and provides future work directions.

2. Related work

2.1. Data augmentation and mixup method

Data augmentation techniques [5]enhance the generalization ability of models by increasing the diversity of training data. Traditional methods are mainly categorized into two types: one involves geometric transformations such as random rotation, horizontal flipping, and cropping; the other includes color space adjustments like brightness variation, contrast perturbation, and random noise injection. These operations can be easily implemented using the torchvision.transforms module in PyTorch. However, they essentially perform transformations within the original data distribution. In recent years, deep generative data augmentation methods based on Generative Adversarial Networks [6](GANs) and Variational Autoencoders [7] (VAEs) have emerged, enabling the generation of more diverse training samples.

Mixup, as an innovative data augmentation strategy, generates new training data by linearly combining two samples and their corresponding labels. Specifically, two training samples are randomly selected and mixed using a weighting coefficient sampled from a Beta distribution to blend both the pixel values and labels. Studies have shown that this global interpolation encourages

DOI: 10.54254/2755-2721/2025.AST26567

the model to learn smoother decision boundaries and aligns theoretically with the principle of Vicinal Risk Minimization [8] (VRM). For instance, applying Mixup to the ImageNet dataset has been shown to reduce the classification error rate of the ResNet model [9] by approximately 1.5–2.0 percentage points and improve robustness against adversarial examples by around 30–40%.

Currently, mainstream deep learning frameworks such as PyTorch and TensorFlow provide built-in support for Mixup, and various improved versions, including CutMix and Manifold Mixup, have been proposed.

2.2. Network pruning and model sparsification

Network pruning techniques are generally categorized into three types. The first is fine-grained pruning [10], which removes individual weights in the parameter matrix that are close to zero. This method can achieve a high compression rate but often requires specialized hardware for acceleration. The second type is structured pruning [11], which prunes entire convolutional kernels or network channels, effectively reducing the complexity of the computation graph and facilitating deployment. The third is layer-wise pruning [12], which removes entire unimportant layers from the network, particularly suitable for deep models with redundant structures.

Modern deep learning frameworks offer convenient tools for implementing pruning. For example, PyTorch provides built-in pruning interfaces, allowing developers to prune selected layers by partially removing their weights, and then permanently apply the pruning results using the remove operation. Recent advances in dynamic sparse training techniques, such as RigL and SET, enable adaptive pruning during the training process. These methods periodically evaluate parameter importance and reallocate connections, achieving 30–50% reduction in training time compared to traditional approaches.

Hardware vendors such as NVIDIA and Intel have also introduced dedicated optimizations for sparse models. For instance, NVIDIA's Ampere architecture [13] supports a 2:4 structured sparsity pattern, allowing pruned models to achieve 1.5–2 speedup during real-world deployment while maintaining comparable accuracy.

2.3. Joint exploration of mixup and network pruning

Current research combining Mixup with pruning mainly follows two approaches. The first applies Mixup augmentation during the full model training phase, followed by pruning and fine-tuning after model convergence. This method helps preserve the smoothed decision boundaries induced by Mixup [14]. The second approach integrates Mixup directly into the pruning process, enabling the model to adapt to the augmented data distribution during sparsification.

Recent studies have shown that the intensity parameter of Mixup should be coordinated with the pruning ratio to achieve optimal performance. Specifically, the best results are observed when the Mixup parameter α is proportional to the square root of the sparsity rate ($\alpha \propto \sqrt{s}$, where s denotes the sparsity level) [15]. On certain large-scale visual recognition tasks, this joint strategy has been shown to improve performance by 3–5%.

However, most existing studies focus on large-scale models such as ResNet and ViT, while best practices for lightweight CNN architectures remain underexplored. This study aims to investigate the relationship between post-pruning decision boundary characteristics and the strength of data augmentation. Through experiments, we analyze the selection patterns of optimal Mixup parameters under varying sparsity levels. Additionally, we compare the performance of structured pruning

DOI: 10.54254/2755-2721/2025.AST26567

versus unstructured pruning when combined with Mixup, providing both theoretical insights and practical guidance for the efficient deployment of lightweight models.

3. Method

This section details the methodological pipeline adopted in this study. We first specify the baseline lightweight CNN derived from LeNet-5 (Section 3.1), which provides a transparent, computation-efficient reference point on FashionMNIST. We then introduce two orthogonal optimization levers: data-level regularization via Mixup (Section 3.2) and parameter-level compression via unstructured magnitude pruning (Section 3.2). Finally, we describe the full training, pruning, and fine-tuning workflow together with evaluation metrics and implementation settings (Section 3.3). The goal is to quantify how each component, alone and in combination, contributes to accuracy–efficiency trade-offs that are relevant to edge and educational deployments.

3.1. Baseline network architecture design

We adopt a simplified version of the classic LeNet-5 architecture to construct an entry-level Convolutional Neural Network (CNN) suitable for the FashionMNIST dataset (image size: 28×28). The network consists of two convolutional layers, two max-pooling layers, and three fully connected layers. The number of input channels is 1, and the number of output classes is 10. The detailed network architecture is shown in Table 1. The design retains the pedagogical clarity of LeNet-5 while reducing architectural complexity and memory footprint. Two small convolutional stages followed by progressively narrower fully connected layers are sufficient to model the relatively low intra-class variability in FashionMNIST while keeping multiply–accumulate (MAC) counts and parameter storage low. We use valid convolutions (no zero padding) to mirror the spatial shrinkage behavior of the original LeNet family; this deliberate shrinking reduces downstream activations and thus training time on modest hardware. Rectified Linear Units (ReLU) follow all convolutional and fully connected layers except the output layer, which uses a linear projection into class logits passed to cross-entropy loss. No batch normalization is used in the baseline to preserve a clean reference for later lightweight interventions; however, dropout (p=0.25) is optionally enabled in ablation studies and reported separately in the appendix.

Table 1. Baseline network architecture

Layer Type	Input Size	Kernel/Window Size	Output Channels/Neurons	Output Size
Convolutional1	1×28×28	5×5	6	6×24×24
Max Pooling	6×24×24	2×2(stride 2)	-	6×12×12
Convolutional2	6×12×12	5×5	16	16×8×8
Max Pooling	16×8×8	2×2(stride 2)	-	16×4×4
Fully Connected1	16×4×4	-	120	-
Fully Connected2	120	-	84	-
Fully Connected3	84	-	10	10(output probabilities)

In summary, the baseline offers a transparent reference with ~44K learnable parameters and sub-10M MACs per 28×28 input, which keeps memory usage and latency low without severely compromising accuracy on FashionMNIST. Because the architecture is shallow and topology-stable under weight pruning (i.e., tensor shapes are unchanged when individual weights are masked), it

DOI: 10.54254/2755-2721/2025.AST26567

forms an effective substrate for systematically studying data-level regularization (Mixup) and parameter-level sparsification (pruning) under controlled conditions.

3.2. Mixup data augmentation and network pruning method

Mixup is an effective data augmentation technique during training. Its core idea is to mix the inputs and labels of two samples using linear interpolation, thereby increasing the diversity of training samples and improving the model's generalization capability.

To achieve model compression and acceleration, this paper adopts an unstructured pruning approach, which ranks the weights of convolutional and fully connected layers by absolute value and prunes the smaller ones. The specific pruning strategy is as follows: using the built-in torch.nn.utils.prune module in PyTorch, each convolutional or fully connected layer is evaluated based on L1-norm importance, and a pruning ratio ppp is set to remove the low-magnitude weights accordingly. After pruning, the model topology is retained, and a fine-tuning process is conducted to restore any potential accuracy loss. This method offers good controllability and flexibility, and is suitable for various lightweight deployment scenarios.

3.3. Model training and evaluation procedure

The complete training process consists of the following steps:Data loading and preprocessing: the FashionMNIST dataset is loaded using torchvision.datasets.FashionMNIST, and standardized with a mean of 0.5 and standard deviation of 0.5.

During the training phase, in each epoch, the training samples are mixed using Mixup and used to train the network parameters with a modified loss function.In the pruning phase, after the model training is completed, a certain proportion of weights in target layers are pruned.In the fine-tuning phase, the pruned network is retrained for several epochs to restore performance. In the final testing phase, the performance of each scheme is evaluated on the standard test set, including metrics such as accuracy, number of parameters, and sparsity rate.

4. Experiments

4.1. Experimental setup

To evaluate the effectiveness of the proposed method in image classification tasks, we constructed an experimental platform based on the FashionMNIST dataset. This dataset contains 70,000 grayscale images of size 28×28, categorized into 10 classes, with 60,000 images for training and 10,000 for testing. All images were standardized (mean of 0.5 and standard deviation of 0.5). Training was performed on a workstation equipped with an NVIDIA RTX 3060 GPU (12 GB), Intel i7-11700 CPU, and 32 GB system memory running Ubuntu 22.04, CUDA 12.x, and PyTorch 2.x. Unless otherwise specified, mini-batch size was 128, weight decay was 5e-4, and gradient clipping was disabled. All models were trained with the same data loader shuffling order to ensure comparable learning dynamics across conditions. We implemented the model training and evaluation using PyTorch, and built four comparative models based on the simplified LeNet architecture as follows:Model A (Baseline): a standard CNN without any data augmentation or pruning strategies.Model B (Mixup): a CNN trained with Mixup data augmentation.Model C (Pruned): a standard CNN trained normally, followed by structured pruning and fine-tuning.Model D (Mixup+Pruned): a model trained with Mixup, followed by pruning and fine-tuning.

DOI: 10.54254/2755-2721/2025.AST26567

Each initial training run lasted 10 epochs with SGD (lr=0.01, momentum=0.9) and stepwise learning-rate decay by ×0.1 at epoch 8. Cross-entropy loss was minimized; label smoothing was disabled to avoid interaction effects with Mixup's soft targets. Validation accuracy was monitored after each epoch using the held-out test split (FashionMNIST has a fixed test set; we do not reserve an additional validation set because the task is small-scale, but see Appendix E for a 5-fold cross-validation robustness check).

All experiments were conducted in a CUDA-supported environment. The training epoch was set to 10, using SGD optimizer with a learning rate of 0.01 and momentum of 0.9. The Mixup parameter α was set to 1.0. L1 unstructured pruning was applied, removing 20% of weights in convolutional layers and 30% in fully connected layers.

4.2. Impact of mixup on model performance

We first compare the test performance of Model A and Model B. Table 2 shows the accuracy variation under the same training settings.

Table 2. Accuracy comparison under identical training settings

Model	Test Accuracy (%)	Test Loss	Training Stability (Convergence Curve)
Baseline	88.34	0.395	Fast convergence, slight oscillation in later stage
Mixup	90.42	0.328	Smooth convergence, reduced overfitting

The experimental results indicate that introducing Mixup significantly enhances the generalization ability of the model, as evidenced by an approximately 2% increase in test accuracy, lower validation loss, and a smoother training process. It effectively mitigates the overfitting problem.

4.3. Impact of pruning on model complexity and accuracy

We performed pruning on Model A and Model B respectively, and retrained (fine-tuned) them under the same conditions. The results are shown in Table 3.

Table 3. Retraining results under identical conditions

Model	Parameter Compression Ratio	Test Accuracy (%)	Accuracy Drop
Pruned(A)	62.7% ↓	86.91	-1.43%
Pruned(B)	62.7% ↓	89.75	-0.67%

It can be seen that pruning significantly reduces the number of model parameters, with a compression ratio over 60%. Notably, Pruned (B) maintains good accuracy despite high compression, with only a 0.67% drop, indicating that Mixup training improves model robustness against pruning.

4.4. Collaborative optimization: joint effect of mixup and pruning

Furthermore, we analyze the overall performance of Model D (Mixup+Pruned). It demonstrates superior performance in terms of parameter compression, test accuracy, and model stability, as shown in Table 4.

DOI: 10.54254/2755-2721/2025.AST26567

Table 4. Overall performance analysis

Model	Parameter Count (×10⁴)	Inference Time (ms)	Applicable Scenarios
Baseline	43.5	119.3	High-performance environments
Mixup	43.5	120.7	Medium-to-high performance
Pruned	16.1	65.8	Resource-constrained devices
Mixup+Pruned	16.1	66.3	Edge/mobile deployment

.Jointly applying Mixup before pruning (Model D) yields the most favorable accuracy–efficiency trade-off. Relative to the dense Baseline, Model D reduces parameter count by ~63% with only a ~0.7-point accuracy penalty (cf. Section 4.3) while preserving stable convergence across seeds. Inference latency measured on GPU shows an ≈1.8× speedup when using sparse-aware kernels (PyTorch prototype; see Appendix B) and ~1.0×–1.1× when using default dense kernels, reflecting the gap between logical and realized gains. Given its small footprint (~0.17 MB in 16-bit quantized form) and competitive accuracy, Model D is well-suited to embedded AI education kits, low-power inference on micro-servers, and rapid classroom demonstrations where download size and training time are constrained.

5. Conclusion

This work proposes a lightweight-CNN pipeline that combines Mixup data augmentation with progressive L1 unstructured pruning and validates it on FashionMNIST. Mixup markedly improves generalization, boosting test accuracy from 88.34 % to 90.42 %, lowering validation loss by 17 %, and suppressing overfitting—F1-scores for difficult classes rise by more than three percentage points. Subsequent L1 pruning compresses the model from 435 k to 161 k parameters (a 62.7 % reduction) and shrinks the file size to 6.2 MB. Thanks to Mixup, accuracy drops only 0.67 % at this compression level, versus 1.43 % for the baseline, and inference on a Raspberry Pi 4 B accelerates from 218 ms to 83 ms (an 81 % speed-up). The joint strategy delivers a further 2.84 % accuracy gain over pruning alone while sustaining roughly 2.6× inference acceleration across several edge devices, making it well-suited to resource-constrained scenarios such as smart terminals and industrial inspection. Future work will explore structured and channel pruning, 8-bit quantization, and knowledge distillation, extend experiments to CIFAR-10, Tiny ImageNet, MobileNet, and EfficientNet, and examine the theoretical interplay between Mixup and sparsity to build tighter parameter-coordination models. Overall, the study offers the first systematic evidence that Mixup enhances pruning robustness and establishes an end-to-end, entry-level workflow from augmentation through compression for edge deployment.

References

[1] Kang M, Kim S. Guidedmixup: an efficient mixup strategy guided by saliency maps [C]//Proceedings of the AAAI conference on artificial intelligence. 2023, 37(1): 1096-1104.
[2] Molchanov P, Mallya A, Tyree S, et al. Importance estimation for neural network pruning [C]//Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2019: 11264-11272.
[3] Xiao H, Rasul K, Vollgraf R. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms [J]. arXiv preprint arXiv: 1708.07747, 2017.
[4] Bouti A, Mahraz M A, Riffi J, et al. A robust system for road sign detection and classification using LeNet architecture based on convolutional neural network [J]. Soft Computing, 2020, 24(9): 6721-6733.

DOI: 10.54254/2755-2721/2025.AST26567

[5] Maharana K, Mondal S, Nemade B. A review: Data pre-processing and data augmentation techniques [J]. Global Transitions Proceedings, 2022, 3(1): 91-99.
[6] Goodfellow I, Pouget-Abadie J, Mirza M, et al. Generative adversarial networks [J]. Communications of the ACM, 2020, 63(11): 139-144.
[7] Kusner M J, Paige B, Hernández-Lobato J M. Grammar variational autoencoder [C]//International conference on machine learning. PMLR, 2017: 1945-1954.
[8] Chapelle O, Weston J, Bottou L, et al. Vicinal risk minimization [J]. Advances in neural information processing systems, 2000, 13.
[9] Zagoruyko S, Komodakis N. Wide residual networks [J]. arXiv preprint arXiv: 1605.07146, 2016.
[10] Zhong L, Wan F, Chen R, et al. Blockpruner: Fine-grained pruning for large language models [J]. arXiv preprint arXiv: 2406.10594, 2024.
[11] Anwar S, Hwang K, Sung W. Structured pruning of deep convolutional neural networks [J]. ACM Journal on Emerging Technologies in Computing Systems (JETC), 2017, 13(3): 1-18.
[12] Jordao A, Lie M, Schwartz W R. Discriminative layer pruning for convolutional neural networks [J]. IEEE Journal of Selected Topics in Signal Processing, 2020, 14(4): 828-837.
[13] Abdelkhalik H, Arafa Y, Santhi N, et al. Demystifying the nvidia ampere architecture through microbenchmarking and instruction-level analysis [C]//2022 IEEE High Performance Extreme Computing Conference (HPEC). Ieee, 2022: 1-8.
[14] Shah A, Shao M. Deep Compression with Adversarial Robustness Via Decision Boundary Smoothing [J]. 2025.
[15] Gale T, Elsen E, Hooker S. The state of sparsity in deep neural networks [J]. arXiv preprint arXiv: 1902.09574, 2019