ACDistill: ACNet and Distillation

Yan Zhu, Bohan Diao, Jian Rong Lee

Abstract This report presents a framework for optimizing convolutional neural networks (CNNs) through asymmetric convolution and knowledge distillation, aiming to balance high performance with efficient deployment on resource-constrained devices. First, we integrate Asymmetric Convolution Blocks (ACB) into standard backbones (e.g., LeNet, VGG) and lightweight custom architectures. These ACB modules introduce additional convolutional branches, enriching representational power at train time. After training, the branches are fused into a single standard kernel, preserving the improved accuracy without increasing inference overhead.

Second, we incorporate knowledge distillation (KD) to compress the model size further and boost performance. In this approach, a larger "teacher" model transfers its learned representations to a smaller "student", guiding the student's training with both logits-based and feature-based supervision. We employ a scheduling mechanism for the distillation loss where the student initially relies heavily on the teacher's outputs, then gradually shifts toward ground-truth labels in later epochs.

Our experiments on the CIFAR dataset demonstrate consistent accuracy gains with ACB-enhanced architectures over baseline CNNs. Moreover, combining ACB with KD provides additional improvements, especially for student-teacher pairs that differ substantially in capacity. These findings underscore the viability of structural re-parameterization and knowledge distillation as complementary strategies for constructing efficient yet accurate networks. Finally, we discuss extensions such as distilling into stronger student architectures, adapting the pipeline to transformer based models, and exploring more dynamic loss-weighting strategies to further enhance performance across diverse tasks and deployment scenarios.

1. Introduction

In recent years, the rapid development of large-scale machine learning models—such as large language models (LLMs) and diffusion-based generative models—has brought remarkable performance gains across a wide range of applications. However, these models often consist of billions of parameters, leading to substantial challenges in deployment, particularly in resource-constrained environments such as mobile devices, embedded systems, and edge computing platforms. This growing gap between model capacity and deployment feasibility has created a strong demand for techniques that can optimize models while preserving their inference efficiency.

To address this issue, our project aims to develop an automated system that performs a sequence of optimizations on a set of machine learning models. The core objective is to improve training-time performance while keeping inference-time deployment simple and efficient. Specifically, we focus on two key strategies: re-parameterization and knowledge distillation.

Re-parameterization, exemplified by the Asymmetric Convolution Block[1], introduces additional convolutional branches during the training phase—such as horizontal (1×3), vertical (3×1), and standard square (3×3) convolutions. These branches are later merged into a single 3×3 kernel for inference, preserving the original inference speed while benefiting from the richer representational power during training. This method is particularly effective in scenarios where deployment constraints are strict, such as in mobile or automotive systems.

Knowledge distillation, first introduced by Hinton et al. [4], provides a framework for training smaller models (students) to mimic the behavior of larger, pre-trained models (teachers). This approach not only improves the convergence rate and performance of lightweight models but also significantly reduces the computational resources required for training and deployment. In this project, we apply both soft-label prediction matching and intermediate feature mimicking to transfer knowledge from teacher to student, enabling smaller models to achieve competitive accuracy with a fraction of the parameters.

The remainder of this paper is structured as follows: Section II reviews relevant work in re-parameterization and knowledge distillation. Section III describes the system design and implementation details. Section IV presents our experimental results and analysis. Finally, Section V concludes the paper and discusses potential future work. The source code for this project is available at: `https://github.com/chedana/ADLS/`.

2. Related Work

2.1. ACB Branch Fusion during Inference

During training, an Asymmetric Convolution Block (ACB) consists of three parallel convolutional branches: a standard 3×3 convolution, a vertical 1×3 convolution, and a horizontal 3×1 convolution. Each of these branches is followed by a Batch Normalization (BN) layer. However, during inference, to reduce computational overhead and speed up inference, the three branches are fused into a single equivalent convolution. Each branch performs:

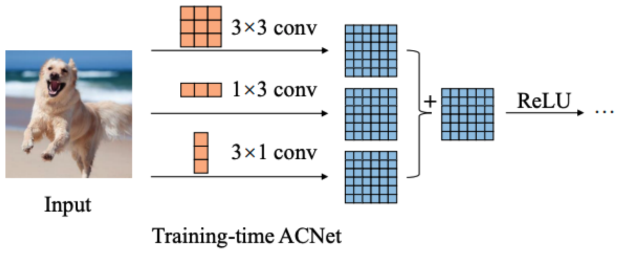


Figure 1. train-mode[1]

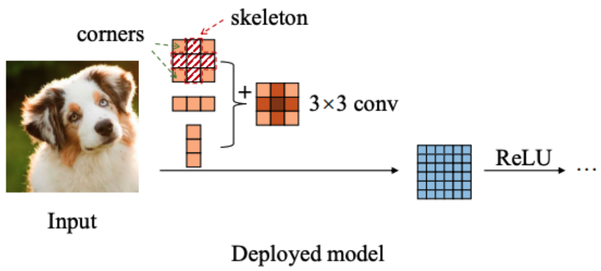


Figure 2. inference mode[1]

$$\mathrm{BN}(I * W) = \frac{I * W - \mu}{\sigma} \cdot \gamma + \beta$$

This can be rewritten as:

$$I * \left(\frac{\gamma}{\sigma} W\right) - \frac{\mu\gamma}{\sigma} + \beta$$

So for the three branches:

$$\begin{aligned}
\text{Branch}_1 : \quad & I * \left(\frac{\gamma}{\sigma} W_1\right) - \frac{\mu\gamma}{\sigma} + \beta \\
\text{Branch}_2 : \quad & I * \left(\frac{\bar{\gamma}}{\bar{\sigma}} W_2\right) - \frac{\bar{\mu}\bar{\gamma}}{\bar{\sigma}} + \bar{\beta} \\
\text{Branch}_3 : \quad & I * \left(\frac{\hat{\gamma}}{\hat{\sigma}} W_3\right) - \frac{\hat{\mu}\hat{\gamma}}{\hat{\sigma}} + \hat{\beta}
\end{aligned}$$

The outputs from the three branches are then summed:

$$\text{Output} = I * \left(\frac{\gamma}{\sigma} W_1 + \frac{\bar{\gamma}}{\bar{\sigma}} W_2 + \frac{\hat{\gamma}}{\hat{\sigma}} W_3\right) + b$$

Where the bias term b is:

$$b = -\frac{\mu\gamma}{\sigma} + \beta - \frac{\bar{\mu}\bar{\gamma}}{\bar{\sigma}} + \bar{\beta} - \frac{\hat{\mu}\hat{\gamma}}{\hat{\sigma}} + \hat{\beta}$$

This process is referred to as BN fusion followed by branch fusion, resulting in a single equivalent convolution kernel and bias that replaces the original three-branch structure for efficient inference.

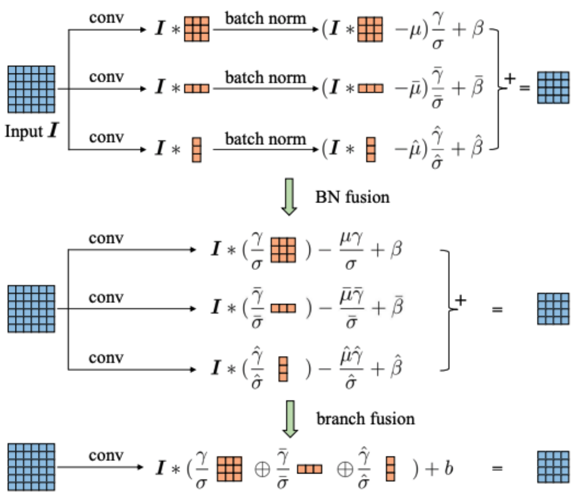


Figure 3. branch-fusion[1]

This process removes all additional branches and achieves the same representational capacity as the multi-branch structure while significantly reducing computational cost.

ACB exemplifies a broader research trend in *structural re-parameterization*, which seeks to decouple the complexity of training-time architectures from their efficient inference-time counterparts. Similar ideas have been adopted in RepVGG [3] and RepViT [8].

While effective, the benefits of ACB are primarily constrained to convolution-based architectures and require additional engineering to implement the fusion correctly. Its advantages may diminish when applied to networks already possessing high representational capacity or those based on transformer designs.

Nevertheless, in this project, ACB serves as a valuable building block for model optimization. Its ability to enhance training performance while preserving inference efficiency makes it particularly well-suited for deployment on mobile and edge devices with constrained computational resources.

2.2. Knowledge Distillation

Knowledge Distillation (KD), first proposed by Hinton et al. in the seminal paper *Distilling the Knowledge in a Neural Network* [4], is a widely-used technique for transferring knowledge from a large, well-trained *teacher* model to a smaller, more efficient *student* model. This method enables the student to approximate the teacher's performance while requiring significantly fewer computational resources.

The primary motivation behind KD lies in the practical constraints of deploying large-scale models such as large language models and diffusion models, which often contain billions of parameters. These models are difficult to deploy in resource-limited environments and present challenges for small companies, academic institutions, or individual developers. By using KD, it becomes possible to develop lightweight models that maintain reasonable accuracy while drastically reducing memory, computation, and deployment costs.

KD has also been shown to improve the training efficiency and convergence speed of student models, particularly when compared to training from scratch. In our estimation, student models with only 1/10 or even 1/100 of the teacher's size can retain approximately 80–90% of the teacher's performance, depending on the task and model architecture, making them suitable for real-world use with a fraction of the resources.

There are two common strategies in KD:

1. **Mimic Prediction (Soft Label Loss):** This approach minimizes the difference between the soft output probabilities of the teacher and the student.

2. **Mimic Feature (Feature-based Distillation):** This method help the student network learn not just the final predictions, but also the internal representations learned by the teacher, we choose L2 loss as loss functions.

$$\mathcal{L}_{\text{feature}} = \|f_T - f_S\|^2$$

2.3. Code Base

Our system is based on the open-source **ACNet** codebase, originally designed for classification tasks using *Asymmetric Convolution Blocks (ACB)*. However, we have extensively refactored and extended the original code to support a broader optimization framework.

3. Methods

Key functionalities include:

- Integrating ACB into a modular and configurable pipeline suitable for automated re-parameterization and deployment;
- Extending the backbone network to support knowledge distillation;
- Implementing feature and Logits mimicking for more effective knowledge transfer;
- Optimizing arguments parsing pipeline for experiments efficiency improvement;

This flexible design enables the system to optimize a variety of models while maintaining deployment efficiency, especially in constrained environments.

3.1. Automation

To streamline experimentation and minimize manual code changes, our system is driven by a unified configuration file (`config.yaml`) that defines all core components of the training and optimization pipeline. This includes model architecture, training hyperparameters, and knowledge distillation settings. The file is loaded at runtime using the `--config` argument, allowing the entire pipeline to be executed automatically.

The configuration specifies the student model (e.g., `lenet5bn`) and whether it should use the Asymmetric Convolution Block (ACB) via the `block_type` field. It also supports toggling knowledge distillation through the `KD` flag under the `teacher` section. When KD is enabled, additional fields such as `teacher_net`, `block_type`, `ckpt`, and `method` control the teacher model architecture, checkpoint path, and the type of distillation loss (e.g., feature or logits).

Other training parameters, such as batch size, number of epochs, learning rate, and whether to continue training or start from scratch (`--conti_or_fs`), are also specified in the configuration. This design enables automated switching between training modes, consistent experiment management, and reproducibility of results.

By centralizing all experiment settings into a single file, our system supports rapid experimentation across various model configurations and optimization strategies without modifying the core codebase.

3.2. ACNet

The ACNet framework operates through a systematic workflow that begins with configuration loading from .yaml files, supporting various network architectures such as vgg, lenet5bn, and mobilev1cifar and etc. The system conditionally employs either ACNetBuilder for asymmetric convolution blocks (when block_type is set to 'acb') or the standard ConvBuilder.

Upon completion of training, when ACNet is utilized, the convert_acnet_weights function transforms the multi-branch asymmetric convolutions used during training into a single standard convolution. The framework then tests these converted weights using a deployment-mode ACNetBuilder. This architectural design features a unique building mechanism where ACNetBuilder creates parallel convolution branches during training, which are subsequently merged into a single standard convolution layer during deployment.

The fundamental advantage of ACNet lies in its approach of utilizing richer feature extraction structures during training while maintaining computational efficiency during inference through mathematical equivalence transformations that combine multiple convolution kernels into one. This ”complex training, simple inference” paradigm constitutes the central mechanism of this implementation, enhancing model performance without increasing computational cost at deployment.

3.3. KD

The Knowledge Distillation (KD) mechanism implemented in this codebase represents a model compression technique where a smaller student model learns from a larger, better-performing teacher model. The framework begins by accessing teacher model parameters from ’teacher’ section in config file, when the KD flag is enabled. Essential parameters include the teacher network architecture, checkpoint file path, and convolution block type.

During the distillation process, the code invokes train_kd_main instead of the standard train_main function, passing the teacher configuration, student model parameters, and the appropriate builder. Throughout training, the teacher model generates soft labels, enabling the student model to learn not only from hard labels (actual categories) but also from the teacher’s output probability distributions. This dual learning approach transfers the teacher’s generalization capabilities to the student model.

For organizational purposes, the framework creates log directories that incorporate teacher network information in a structured format, documenting the teacher network type, block structure, and distillation methodology. The code supports two distillation methods, which are ’logits’ and ’feature’. The logits method encourages the student model to learn from the output predictions of the teacher network, typically by matching the soft probabilities produced before the softmax layer. The feature method, on the other hand, guides the student to mimic the internal feature representations of the teacher, specifically the output of the last convolutional layer before the fully connected (FC) layer. This allows the student to align its intermediate understanding of the input with that of the teacher.

The fundamental principle underlying this knowledge distillation approach involves extracting representational knowledge from a more capable model into a more compact one, potentially leveraging ACNet’s ”complex training, simple inference” paradigm to enhance model performance without increasing inference computational demands.

4. Experiments

We conduct extensive experiments to evaluate the effectiveness of ACNet in enhancing CNN performance across diverse architectures and the CIFAR dataset[5]. Specifically, we select three architectures as baselines, construct corresponding ACNet variants, and train them from scratch. To ensure fair comparison, each ACNet model is converted back to the baseline structure for evaluation. In addition, we incorporate knowledge distillation (KD), using a deeper, ACBlock version of baseline as the teacher to further guide the baseline student model. All models are trained to full convergence with identical hyperparameter configurations, such as learning rate schedules and batch sizes, to guarantee comparability.

4.1. Architectures and Configurations

Model and training for ACB To compare the performance between the baseline convolutional block and the Asymmetric Convolution Block (ACB), we conduct experiments using three different network architectures: Cifar-quick, VGG-16, and LeNet(Figure 4).

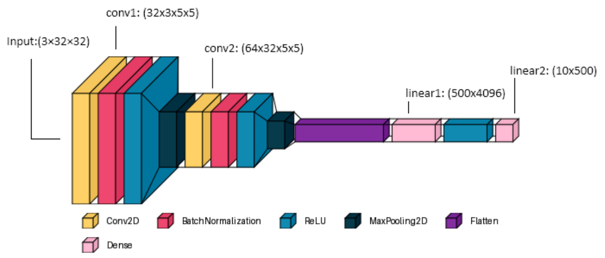


Figure 4. LeNet-Cifar

All experiments are carried out under the same training configuration to ensure fairness. Specifically, we train each model for 50 epochs using the same data augmentation strategy, a batch size of 128, and the SGD optimizer. The learning rate is scheduled using the CosineAnnealingLR scheduler, it is a learning rate adjustment strategy that follows a cosine annealing schedule[6], used to improve convergence and generalization in deep learning training. It gradually decreases the learning rate from an initial value $\eta_{\max}$ to a minimum value $\eta_{\min}$ over a predefined number of epochs $T_{\max}$, following a half-cosine curve. This approach allows for larger learning rates at the beginning for faster convergence and smaller rates toward the end for fine-tuning. The learning rate at epoch t is computed using the formula:

$$\eta_t = \eta_{\min} + \frac{1}{2}(\eta_{\max} - \eta_{\min}) \left(1 + \cos\left(\frac{t\pi}{T_{\max}}\right)\right)$$

Model and training for KD For knowledge distillation (KD) experiments, we design three student-teacher model pairs. In the first setting, both student and teacher are based on the Cifar-quick architecture, where the student uses the original Cifar-quick, and the teacher, denoted as *Cifar-quick_acb_deep*, doubles the number of convolutional layers and replaces standard convolution with Asymmetric Convolution Blocks (ACB). The second setting uses a shallow version of VGG, *vgg_shallow*, with only 5 convolutional layers as the student, and *vgg_acb*, a full VGG-16 model with ACB applied to all convolutional layers, as the teacher. In the third setting, we use LeNet as the student, while the teacher, *lenet_deep_acb*(Figure 5), is a deeper variant with doubled convolutional layers and ACB applied throughout. These configurations are chosen to evaluate KD performance across different depths and architectural complexities.

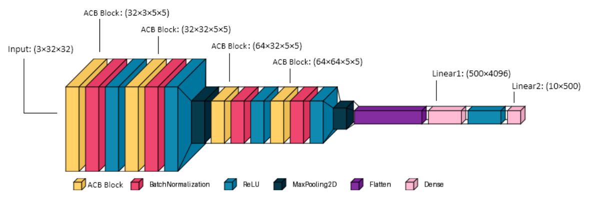


Figure 5. LeNet-deep-ACB

Model	**Student Top-1**	**Teacher Top-1**	**Top-1 ↑**
Cifar-quick	84.64%	88.44%	3.80%
VGG-shallow	90.19%	92.82%	2.63%
LeNet	83.72%	88.96%	5.42%

Table 1. Student vs Teacher

To balance the influence of the teacher and the ground-truth labels during training, we introduce a KD loss scheduler that dynamically adjusts the weight α in the total loss function. The total loss is computed as:

$$\mathcal{L}_{\text{total}} = \alpha \cdot \mathcal{L}_{\text{KD}} + (1 - \alpha) \cdot \mathcal{L}_{\text{sup}},$$

where $\mathcal{L}_{\text{KD}}$ is the knowledge distillation loss and $\mathcal{L}_{\text{sup}}$ is the standard supervised loss (e.g., cross-entropy with labels). The value of α is scheduled across three training phases: during the first one third of the epochs, $\alpha = 1.0$, meaning the model is fully guided by the teacher; in the middle third, $\alpha = 0.5$, balancing between teacher supervision and label supervision; and in the final third, $\alpha = 0.0$, where the model is trained purely with ground-truth labels. This gradual transition encourages effective knowledge transfer early on, while allowing the student to fine-tune independently in later stages.

4.2. Performance improvement on Cifar

We achieve significant performance improvements with the use of Asymmetric Convolution Blocks (ACB). Each baseline-ACB pair demonstrates consistent gains in Top-1 accuracy when ACB is integrated into the teacher network. These results confirm that incorporating ACB enhances the representational capacity of the network.

Model	**Baseline Top-1**	**ACNet Top-1**	**Top-1 ↑**
Cifar-quick	84.64%	85.75%	1.11%
VGG	91.70%	92.82%	1.12%
LeNet	83.72%	85.30%	1.58%

Table 2. Baseline vs ACB

presents the Top-1 accuracy comparison between baseline models and their KD-enhanced using logit-based and feature-based distillation. Across all architectures, both KD strategies lead to performance improvements over the baseline. Cifar-quick benefits from KD-logits and KD-feature, improving by 0.97% and 0.87% respectively. LeNet shows even larger gains, with KD-logits achieving a 2.26% increase and KD-feature reaching a 2.43% improvement. However, the improvement on VGG-shallow is less pronounced, with only marginal gains of 0.93% (KD-logits) and 0.03% (KD-feature). This can be attributed to the small performance gap (2.63%) between the student and teacher models in this case. In knowledge distillation, a larger gap between the teacher and student typically enables more effective transfer of knowledge, as the teacher can provide richer and more informative supervision. The limited difference here suggests that the teacher does not have a substantial advantage, thus limiting the benefit of distillation.

Model	**Baseline Top-1**	**KD-logits**	**KD-feature**
Cifar-quick	84.64%	85.61%	85.51%
VGG-shallow	90.19%	91.12%	90.22%
LeNet	83.72%	85.98%	86.15%

Table 3. Baseline vs KD

4.3. Design Decisions

The choice of model architectures—LeNet, VGG, and Cifar-quick—was driven by their simplicity and fully convolutional designs, making them well-suited for image classification tasks under limited computational resources and time constraints. Their relatively shallow depth makes them ideal for a proof-of-concept study where rapid iteration and clarity of analysis are prioritized. To construct more capable teacher models, we used deeper versions of the student networks and replaced standard convolutional layers with Asymmetric Convolution Blocks (ACB). This design increases the performance gap between student and teacher, which is known to be beneficial for effective knowledge distillation, as a more powerful teacher provides richer supervision.

We explored both logits-based and feature-based knowledge distillation methods. These two strategies are among the most widely used in the field, and employing both allows for a more comprehensive understanding of how different forms of supervision affect student learning. Instead of using a fixed weighting between distillation loss and supervised loss, we introduced a dynamic loss scheduler. At the early stages of training, the model is guided entirely by the teacher ($\alpha = 1$), as its parameters are more randomly and less reliable for interpreting raw labels. As training progresses, the influence of the teacher gradually decreases, with α reduced to 0.5 in the middle third of training and finally to 0 in the last third. This transition reflects a learning process similar to how humans learn: starting under close guidance by parents and teachers, then gradually shifting toward self-driven learning through direct exposure to labels and data.

We also chose the CosineAnnealingLR scheduler to control the learning rate during training. This approach starts with a high learning rate to encourage exploration and prevent early convergence to suboptimal minima, and gradually decreases it to fine-tune the model in later epochs. Combined with the SGD optimizer and consistent training configurations (batch size 128, 50 epochs, and uniform data augmentation), these choices help ensure fair evaluation and effective convergence across all experimental setups.

4.4. Discussion and Future Work

As large-scale models continue to grow in size and complexity, transferring their knowledge to smaller, more efficient models becomes increasingly important. Knowledge distillation (KD) is a promising solution to this challenge, and our current work lays the foundation for future extensions in several directions. While we applied KD to baseline convolutional student models, a natural next step is to distill into stronger student models enhanced with Asymmetric Convolution Blocks (ACB), potentially yielding even better performance. Beyond classification, this KD framework can be adapted to more complex tasks such as object detection, semantic segmentation, and generative models like diffusion.

Additionally, although our experiments focused on convolutional networks, the KD pipeline can also be extended to transformer-based architectures. Given the increasing use of vision and language transformers, developing distillation methods tailored for these models—such as attention-based feature alignment—would significantly enhance the generalizability of our approach.

Our current α scheduling strategy is deliberately simple, using a fixed three-stage transition from full KD supervision to full label supervision. In future work, more dynamic or learnable scheduling strategies could be explored, possibly adapting based on training signals or confidence gaps. Moreover, combining logits-based and feature-based KD into a unified loss function may improve generalization by capturing both output behavior and internal representations. Incorporating online hard example mining (OHEM[7]) can further enhance the process by prioritizing difficult samples where the student struggles most compared to the teacher. Another exciting direction is to integrate LoRA (Low-Rank Adaptation) into the KD pipeline, enabling parameter-efficient fine-tuning guided by distillation losses. For instance, LoRA-based adapters could allow lightweight student models to benefit from high-capacity teacher knowledge while remaining efficient and adaptable. Altogether, these extensions present a rich landscape for future exploration, especially as model scaling continues to outpace deployment constraints.

5. Conclusion

In summary, our work shows that structural reparameterization and knowledge distillation improve the accuracy and efficiency of CNNs. The ACBs enrich representational capacity during training without adding inference overhead, while knowledge distillation bridges the gap between large teacher models and more lightweight student networks. Future work may extend these techniques to transformer-based architectures, more dynamic KD scheduling strategies, and advanced feature alignment methods. These extensions will allow the work to be applied to a larger variety of tasks and deployment environments.

6. Acknowledgment

This work is based on the publicly available ACNet codebase [2], which we adopted and extended for our experiments. We reused their training pipeline and overall structure, but made modifications to adapt it to our knowledge distillation framework. Specifically, we modified selected backbone files to support different student-teacher configurations and integrated knowledge distillation mechanisms into the training loop. We introduced both logits-based and feature-based KD, added a flexible α scheduler to control the contribution of KD loss over time, and optimized the argument parsing procedure to improve ease of configuration and experiment management. These changes allowed the original codebase to support our experiments efficiently while preserving its core functionality. We sincerely thank the original authors for their excellent implementation.

References

[1] X. Ding, Y. Guo, G. Ding, and J. Han. Acnet: Strengthening the kernel skeletons for powerful cnn via asymmetric convolution blocks, 2019.

[2] X. Ding, Y. Sun, X. Wang, J. Han, and G. Ding. Acnet: Strengthening the kernel skeletons for powerful cnn via asymmetric convolution blocks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1911–1920, 2019.

[3] X. Ding, X. Zhang, N. Ma, J. Han, G. Ding, and J. Sun. Repvgg: Making vgg-style convnets great again, 2021.

[4] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network, 2015.

[5] A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

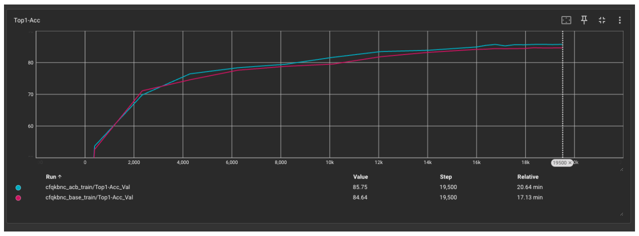
[6] I. Loshchilov and F. Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

[7] A. Shrivastava, A. Gupta, and R. Girshick. Training region-based object detectors with online hard example mining. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 761–769, 2016.

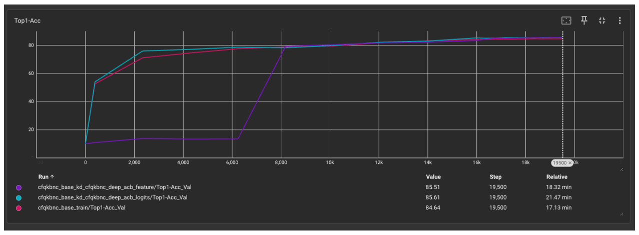
[8] A. Wang, H. Chen, Z. Lin, J. Han, and G. Ding. Repvit: Revisiting mobile cnn from vit perspective, 2024.

7. Appendix

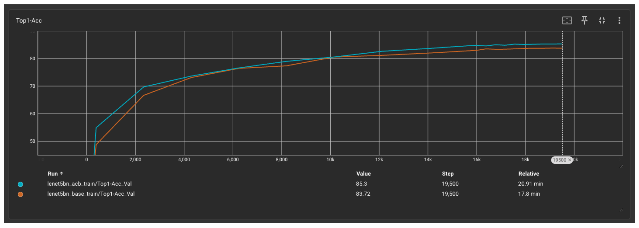
Validation curve for baseline vs ACB and KD: As shown in Figure, the feature-based KD model fails to converge during the initial training stage when $\alpha = 1$. This is because, in that stage, the student model is fully supervised by the teacher's convolutional feature map output (i.e., the feature vector before the fully connected layer). While this allows the student's convolutional layers to learn from the teacher, the final fully connected (FC) layers receive no supervision, as there is no label loss or logits loss to guide the output layer. As a result, the student cannot produce meaningful predictions in this phase, which explains the low Top-1 accuracy. However, once the training enters the second phase ($\alpha = 0.5$), where both the KD loss and the supervised label loss are combined, the model quickly begins to converge. This is because the FC layers now receive gradient signals from the label supervision, allowing the entire network to align both structurally (via features) and functionally (via labels), leading to rapid improvement in accuracy.



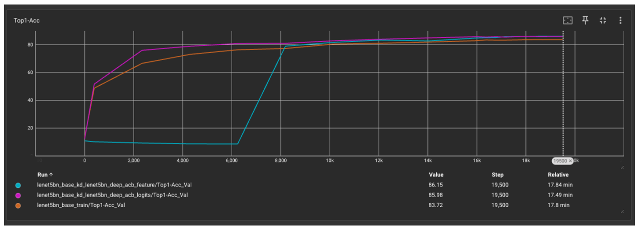
	Run ↑	Value	Step	Relative
●	cfqkbnc_acb_train/Top1-Acc_Val	85.75	19,500	20.64 min
●	cfqkbnc_base_train/Top1-Acc_Val	84.64	19,500	17.13 min



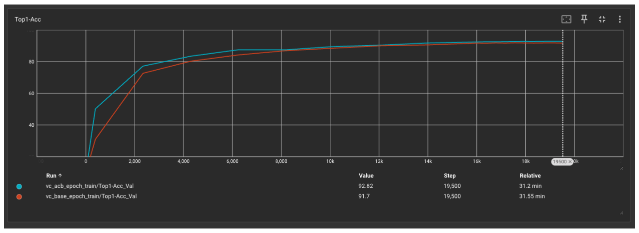
	Run ↑	Value	Step	Relative
●	cfqkbnc_base_kd_cfqkbnc_deep_acb_feature/Top1-Acc_Val	85.51	19,500	18.32 min
●	cfqkbnc_base_kd_cfqkbnc_deep_acb_logits/Top1-Acc_Val	85.61	19,500	21.47 min
●	cfqkbnc_base_train/Top1-Acc_Val	84.64	19,500	17.13 min



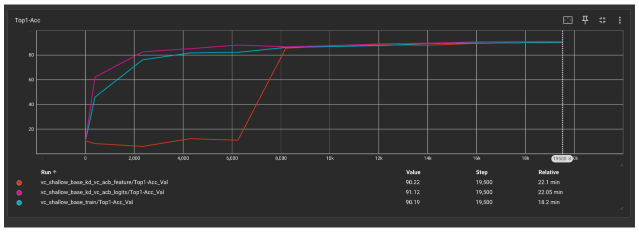
	Run ↑	Value	Step	Relative
●	lenet5bn_acb_train/Top1-Acc_Val	85.3	19,500	20.91 min
●	lenet5bn_base_train/Top1-Acc_Val	83.72	19,500	17.8 min



	Run ↑	Value	Step	Relative
●	lenet5bn_base_kd_lenet5bn_deep_acb_feature/Top1-Acc_Val	86.15	19,500	17.84 min
●	lenet5bn_base_kd_lenet5bn_deep_acb_logits/Top1-Acc_Val	85.98	19,500	17.49 min
●	lenet5bn_base_train/Top1-Acc_Val	83.72	19,500	17.8 min



	Run ↑	Value	Step	Relative
●	vc_acb_epoch_train/Top1-Acc_Val	92.82	19,500	31.2 min
●	vc_base_epoch_train/Top1-Acc_Val	91.7	19,500	31.55 min



	Run ↑	Value	Step	Relative
●	vc_shallow_base_kd_vc_acb_feature/Top1-Acc_Val	90.22	19,500	22.1 min
●	vc_shallow_base_kd_vc_acb_logits/Top1-Acc_Val	91.12	19,500	22.05 min
●	vc_shallow_base_train/Top1-Acc_Val	90.19	19,500	18.2 min