Knowledge Distillation for Smaller Models in Computer Vision

Wonjun Park*, **Abhinay Kotla***
Computer Science and Engineering
University of Texas at Arlington
Arlington, TX, USA
{ *wxp7177, axk5827* }*@mavs.uta.edu*

***Abstract*—Knowledge distilation is a technique that allows deep learning models to transfer knowledge from a larger, more complex model (the teacher) to a smaller, more efficient model (the student). The technique can be adopted in various tasks including image classification and image generation. Among computer vision tasks, the paper specifically focuses on the image classification task to demonstrate the effectiveness of knowledge distillation, further highlighting the framework to apply the technique to other tasks. The results from the image classification task that is addressed in this paper show that knowledge distillation is able to successfully decrease the size of the model while preserving the performance of the model. In particular, the distilled model achieves slightly better performance than the teacher model. We expect that the framework can be applied to other tasks in computer vision without affecting performance. The code is available at here.**

***Index Terms*—deep learning, computer vision, knowledge distillation, teacher-student learning, and quantization.**

I. Introduction

Deep neural networks have achieved remarkable success across a wide spectrum of computer vision problems, ranging from simple image classification tasks, such as ImageNet [1], to high-fidelity image generation using diffusion models [2]–[4] and Generative Adversarial Network (GAN) models [5], [6]. Much of this progress, however, has been driven by increasingly larger model capacities: billions of parameters, expansive training datasets, and considerable computational budgets — even though such computationally expensive models deliver state-of-the-art performance. Their intensive resource requirements pose serious obstacles to deploying them on resource-constrained devices or in real-time and latency-sensitive back-end services, such as smartphones or even low-end GPUs. Closing the gap between accuracy and efficiency therefore remains one of the central research challenges in deep learning.

Image classification is a fundamental task in computer vision. The goal is to predict the class of an image from a predefined set of categories. Many other tasks, including object detection, image segmentation, and image generation, are successfully built on top of image classification, especially since AlexNet [7] revolutionized computer vision by outperforming traditional methods with its deep learning-based approach.

In this project, the initial plan was to implement a knowledge distillation framework for an image inpainting task using diffusion models, aiming to restore masked regions of images. Unfortunately, although training scripts for student models were successfully implemented and run with guidance from the teacher model DiffIR [4], both memory limitations and the computational burden led to extremely long training times. Training on the Places dataset [8] required more than 200 hours per epoch, even when using 8-bit floating-point precision. Given these constraints, we decided to pivot from image inpainting to image classification in order to demonstrate the effectiveness of knowledge distillation and to lay the groundwork for applying the technique to more complex tasks in the future.

Building on the code implemented in Assignment 2 of the CSE 5368 course, we developed a knowledge distillation framework for the image classification task. In the subsequent sections, we describe our knowledge distillation method, present experimental results demonstrating its efficiency, and discuss potential directions for future work.

II. Related Works

In this section, we review performance optimization techniques for the image classification task through three main approaches: *II-A lightweight backbones*, *II-B knowledge distillation*, and *II-C quantization*. The first approach focuses on the design of efficient architectures, the second emphasizes the transfer of knowledge from a larger model to a smaller one, and the third discusses the quantization of models, a widely used technique for reducing model size and improving performance on low-compute devices.

A. Lightweight Backbones

Early breakthroughs in large-scale image classification were driven by deeper and wider Convolutional Neural Network (CNN) architectures such as AlexNet, VGG, Inception, and ResNet [7], [9]–[11]. While these architectures steadily improved top-1 accuracy on ImageNet [1], their memory footprint and computational cost grew proportionally, prompting a parallel line of research into efficiency-focused approaches. Lightweight backbones such as MobileNet [12],

* Equal Contribution.

This paper is a part of the final project of the course for Spring 2025, CSE-5368-001: Neural Networks at the University of Texas at Arlington under Dr. Francklin Rivas.

[13], EfficientNet [14], and the Data-efficient Vision Transformer (DeiT) [15] demonstrated that accuracy can be retained through careful depth-width scaling and attention re-use. Nevertheless, these models still require substantial training resources when trained from scratch.

B. Knowledge Distillation

Knowledge distillation is a technique used to transfer knowledge from a large, complex model to a smaller, more efficient model, enabling deployment on resource-constrained devices without significant performance loss. Among several aspects of knowledge distillation, the paper specifically focuses on the mimicry between the teacher and student models, which enables the small model to follow the teacher's probability distribution. Hinton et al. [16] introduced logit-matching between a cumbersome teacher and a compact student. Subsequent works enriched the transfer signal by aligning intermediate feature maps [17] or the spatial attention of convolutional layers [18]. Another notable approach, self-distillation [19], has been shown to be effective in improving the performance of a single model by training it with its own architecture.

C. Quantization

Quantization techniques have recently gained significant attention for their ability to enable Large Language Model (LLM) deployment on low-compute devices. In particular, Microsoft[1] has pioneered the quantization of LLM through their Phi-3 [20] and Phi-4 [21] models, as well as the 1-bit quantized model, BitNet [22]. Floating-point quantization techniques can also be applied to the image classification task, enabling the deployment of large models on low-compute devices.

III. Methodology

A. Intel Image Dataset

Fig. 1. Intel Image Dataset

The Intel Image Classification Dataset [23] was curated as part of a data hackathon organized by Intel[2] on the online platform Kaggle[3], with the aim of fostering engagement and innovation within data science communities. The dataset comprises 25,000 RGB color images, each with a resolution of 150 $\times$ 150 pixels, depicting a variety of natural and urban scenes from around the world. The images, originally captured by Jan Böttinger on Unsplash[4], are labeled into six distinct classes: buildings, forest, glacier, mountain, sea, and street.

The dataset is partitioned into three subsets: approximately 14,000 images for training, 3,000 images for testing, and 7,000 images for prediction tasks. Each subset is provided as a separate zipped file on Kaggle.

The primary objective of this dataset is to support participants in developing and evaluating image classification models. By providing a diverse and well-labeled collection of scenes, the dataset encourages the development of robust classification algorithms capable of distinguishing objects in various environments with reasonable accuracy.

B. Model Architecture

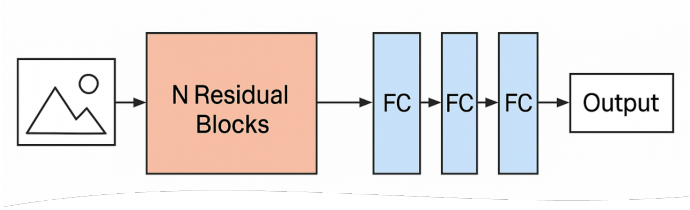

Fig. 2. Custom CNN model Architecture

In this section, we describe the architecture of the teacher model. We implemented a custom CNN model, incorporating residual blocks [11], which hierarchically extract and aggregate spatial features from the input images before mapping them to one of six scene categories.

The model consists of 12 residual blocks, progressively increasing the channel capacity while reducing the spatial resolution, and a fully connected classifier that sequentially reduces feature dimensions through 1024, 512, 256, 64, and finally 6 outputs. In total, the model contains 26,534,358 parameters.

All operations in the model are performed using 32-bit floating point precision, which is the default in PyTorch [24] for most GPUs.

C. Weight Reduction

To create lighter student models from the teacher architecture, we reduced both the number of residual blocks and the depth of the classifier. Four reduced models were created, excluding the teacher model, with 10, 8, 6, and 4 residual blocks, respectively, and classifier configurations of (512, 256, 64, 16), (256, 64, 16), (128, 64, 16), and (64, 32) layers. The number of parameters in these models are 6,601,686, 1,648,470, 408,854, and 98,294, respectively.

D. Precision Quantization

Quantization was applied to both model weights and floating-point operations to further reduce computational cost. We used the model with 1.6 million parameters for this test,

[1]https://www.microsoft.com/en-us/
[2]https://www.intel.com/content/www/us/en/homepage.html
[3]https://www.kaggle.com/
[4]https://unsplash.com/

aiming to achieve the best balance between performance and efficiency. While other models were also tested, they were unable to match the performance achieved by this model. Depending on the application, different models can be selected for quantization based on the trade-off between accuracy and computational cost. Two different levels of precision were explored: 16-bit and 8-bit floating point. To implement quantization, we used `torch.set_default_dtype()` along with the `bitsandbytes` [25] Python framework.

IV. Experiments

A. Training Details

Several student models were configured based on the teacher model described in Section III-B, applying the reduction and quantization techniques introduced in Sections III-C and III-D. All training was conducted on a single NVIDIA GeForce RTX 4070 Laptop GPU. The implementation is based on the PyTorch framework [24], leveraging its extensive libraries and tools for deep learning. The Adam optimizer [26] was employed with a learning rate of 0.001. A mini-batch size of 224 was used, and KL Divergence was utilized as the loss function, effectively measuring the difference between the distributions predicted by the teacher and those output by the student models. Specifically, the the loss quantifies how well the student model mimics the teacher model's output distribution, minimized when the student model closely approximates the teacher model's output distribution. The following formulations shows the KL Divergence,

$$\mathrm{KL}\big(P \parallel Q\big) = \sum_{c=1}^{C} P(c|x)\, \log \frac{P(c|x)}{Q(c|x)}\,, \tag{1}$$

where P is the teacher's distribution and Q is the student's distribution. Early stopping was applied with a patience of 3 epochs to prevent overfitting.

B. Evaluation

The performance of the teacher model is summarized in Table I. Table II shows the performance of the student models along with the reduction in the number of parameters. Additionally, Table III presents the performance achieved through quantization of model weights and floating-point operations.

TABLE I
Teacher Model Performance

	Teacher model
Number of parameters	26,534,358
Accuracy	85.77%
Training time per epoch (seconds)	72.32

As expected, the training time per epoch decreases as the number of parameters is reduced. The smallest student model achieves the fastest training time per epoch, at 64.7 seconds. Interestingly, the performance does not strictly correlate with the number of parameters: the 8-block student model outperforms others with an accuracy of 86.43%, highlighting that architectural choices and dataset characteristics can outweigh mere parameter counts.

TABLE II
Weight Reduction Comparison

	10 blocks, (512, 256, 64, 16)
Number of parameters	6,601,686
Accuracy	83.23%
Training time per epoch (seconds)	68.6

	8 blocks, (256, 64, 16)
Number of parameters	1,648,470
Accuracy	**86.43%**
Training time per epoch (seconds)	67.0

	6 blocks, (128, 64, 16)
Number of parameters	408,854
Accuracy	85.77%
Training time per epoch (seconds)	66.7

	4 blocks, (64, 32)
Number of parameters	98,294
Accuracy	84.57%
Training time per epoch (seconds)	**64.7**

TABLE III
Precision Quantization on Teacher Model

	fp16	*fp8*
Accuracy	85.54%	**86.38%**
Training time per epoch (seconds)	**61.3**	70.5

Using the best-performing student model (8 blocks), we further experimented with precision quantization. The results in Table IV demonstrate that the 8-block student model benefits significantly from fp16 precision. It achieved an accuracy of 87.00% with the fastest training time per epoch at 60.8 seconds, while fp8 precision resulted in slightly lower accuracy (86.07%) and a slower training time (69.0 seconds). This highlights that selecting the appropriate precision can significantly improve both model performance and computational efficiency.

V. Conclusion

In this paper, we presented a comprehensive study on parameter reduction and quantization of image classification models using knowledge distillation. We demonstrated that knowledge distillation is an effective technique for transferring knowledge from a larger, more complex model (the teacher) to a smaller, more efficient model (the student) in the context of image classification tasks. Based on these results, we believe that our knowledge distillation framework can be extended to other computer vision tasks, including image inpainting, without significant performance degradation. We anticipate that this framework will contribute to the development of more efficient models across a wide range of computer vision applications.

TABLE IV
PRECISION QUANTIZATION ON 8-BLOCK MODEL

	fp16	*fp8*
Accuracy	**87.00%**	86.07%
Training time per epoch (seconds)	**60.8**	69.0

REFERENCES

[1] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[2] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," *Advances in neural information processing systems*, vol. 33, pp. 6840–6851, 2020.

[3] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 10 684–10 695.

[4] B. Xia, Y. Zhang, S. Wang, Y. Wang, X. Wu, Y. Tian, W. Yang, and L. Van Gool, "Diffir: Efficient diffusion model for image restoration," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 13 095–13 105.

[5] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Advances in neural information processing systems*, vol. 27, 2014.

[6] K. Nazeri, E. Ng, T. Joseph, F. Z. Qureshi, and M. Ebrahimi, "Edgeconnect: Generative image inpainting with adversarial edge learning," *arXiv preprint arXiv:1901.00212*, 2019.

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

[8] B. Zhou, A. Lapedriza, A. Khosla, A. Oliva, and A. Torralba, "Places: A 10 million image database for scene recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 6, pp. 1452–1464, 2017.

[9] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[10] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[11] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[12] A. G. Howard, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[13] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.

[14] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International conference on machine learning*. PMLR, 2019, pp. 6105–6114.

[15] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou, "Training data-efficient image transformers & distillation through attention," in *International conference on machine learning*. PMLR, 2021, pp. 10 347–10 357.

[16] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[17] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, "Fitnets: Hints for thin deep nets," *arXiv preprint arXiv:1412.6550*, 2014.

[18] S. Zagoruyko and N. Komodakis, "Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer," *arXiv preprint arXiv:1612.03928*, 2016.

[19] L. Zhang, J. Song, A. Gao, J. Chen, C. Bao, and K. Ma, "Be your own teacher: Improve the performance of convolutional neural networks via self distillation," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 3713–3722.

[20] M. Abdin, J. Aneja, H. Awadalla, A. Awadallah, A. A. Awan, N. Bach, A. Bahree, A. Bakhtiari, J. Bao, H. Behl *et al.*, "Phi-3 technical report: A highly capable language model locally on your phone," *arXiv preprint arXiv:2404.14219*, 2024.

[21] M. Abdin, J. Aneja, H. Behl, S. Bubeck, R. Eldan, S. Gunasekar, M. Harrison, R. J. Hewett, M. Javaheripi, P. Kauffmann *et al.*, "Phi-4 technical report," *arXiv preprint arXiv:2412.08905*, 2024.

[22] J. Wang, H. Zhou, T. Song, S. Mao, S. Ma, H. Wang, Y. Xia, and F. Wei, "1-bit ai infra: Part 1.1, fast and lossless bitnet b1. 58 inference on cpus," *arXiv preprint arXiv:2410.16144*, 2024.

[23] P. Singh, "Intel image classification," https://www.kaggle.com/datasets/puneet6060/intel-image-classification, 2019, accessed: 2025-04-21.

[24] A. Paszke, "Pytorch: An imperative style, high-performance deep learning library," *arXiv preprint arXiv:1912.01703*, 2019.

[25] bitsandbytes Foundation, "bitsandbytes: Accessible large language models via k-bit quantization for pytorch," https://github.com/bitsandbytes-foundation/bitsandbytes. [Online]. Available: https://github.com/bitsandbytes-foundation/bitsandbytes

[26] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.