EE5102/CS6302 - Advanced Topics in Machine Learning – Fall 2025

Assignment 3 — *Release Date: 19 October 2025*

Instructions: Please read the following instructions carefully and abide by while preparing your submissions:

- This is the third graded assignment of the course which counts towards 7% of your final assignment aggregate.
- We need only one submission per group. If you submit the report for your only task and/or not in the required format, we will assign you zero marks.
- This assignment is due by **Friday, 31 October 2025 on LMS**.
- As a submission, you need to prepare and submit you deliverables according to the instructions in **Task 4 of this assignment**.
- **AI Usage Policy:** You are not allowed to use any generative AI model—including LLMs—to write any part of your PDF report. The language/text and analysis must be entirely your own, in the report.

  For the coding part, you ***may*** use public libraries, built-in functions, or even get help from an LLM if needed. That's acceptable.

  However, once you submit your assignment, you are fully responsible for everything in it—text and code both. If your submission overlaps significantly with another group's work, you cannot later claim that some part was LLM-generated. That will not be accepted as an excuse.

Overview & Motivation: As larger neural networks with more layers and nodes are being widely used, reducing their storage and computational cost becomes critical, especially for some real-time applications such as online learning and incremental learning. In addition, recent years witnessed significant progress in virtual reality, augmented reality, and smart wearable devices, creating unprecedented opportunities for researchers to tackle fundamental challenges in deploying deep-learning systems to portable devices with limited resources [e.g., memory, central processing units (CPUs), energy, bandwidth]. Efficient deep learning methods can have a significant impact on distributed systems, embedded devices, and field-programmable gate array (FPGA) for artificial intelligence (AI). For example, the residual network-50 (ResNet-50), which has 50 convolutional layers, needs more than 95 megabytes of memory for storage, and numerous floating number multiplications for calculating each image. After discarding some redundant weights, the network still works as usual but saved more than 75% of parameters and 50% computational time. For devices like cell phones and FPGAs with only several megabyte resources, how to compact the models used on them becomes really important.

Model compression techniques have emerged as essential tools to reduce model size and improve efficiency without sacrificing too much performance. In this assignment, we will explore three primary approaches to model compression: **Pruning**, **Quantization**, and **Knowledge Distillation**. Each of these methods tackles the challenge of reducing computational demand in different ways, offering unique benefits and trade-offs.

Pruning: Pruning aims to reduce the number of parameters in a neural network by eliminating redundant or less important connections. By carefully selecting which weights to remove, pruning can dramatically shrink a model's size while maintaining its accuracy. We will focus on both **unstructured pruning**, where individual weights are removed, and **structured pruning**, where entire neurons or channels are eliminated. The goal is to help you understand how pruning can be tailored to specific architectures and use cases, as well as its impact on the model's decision-making process.

Quantization: Quantization reduces the precision of the model's weights and activations, typically converting 32-bit floating-point numbers to lower bit-width representations, such as 16-bit, 8-bit, or even 4-bit. While this reduces the memory footprint and accelerates inference, it can also affect model accuracy, particularly in sensitive layers. In this assignment, you will explore the trade-offs between precision and performance through **Post-training Quantization (PTQ)** and **Quantization-aware Training (QAT)**. These techniques are critical for deploying models on edge devices or specialized hardware.

Knowledge Distillation: In Knowledge Distillation, a smaller "student" model learns to mimic a larger, more powerful "teacher" model. By transferring knowledge through softened probability distributions or internal feature representations, the student model achieves competitive performance at a fraction of the size. We will examine state-of-the-art approaches in this domain: **logit matching**, **feature matching**, and **contrastive representation distillation** and investigate whether larger teachers always yield better student models. Additionally, you'll explore how KD can improve properties like robustness and out-of-distribution (OOD) performance.

These three techniques will provide a comprehensive look at the current state of model compression. Through hands-on experimentation, you will not only understand the mechanisms behind these approaches but also their practical implications, setting the stage for deeper exploration in real-world scenarios.

Task 1: Pruning: Pruning, one of the earliest forms of model compression, was first introduced by [1] in the seminal "Optimal Brain Damage" paper, which laid the foundation for removing redundant parameters from neural networks. Over time, pruning has evolved into both structured and unstructured variants, each offering different trade-offs in terms of computational efficiency and performance. In this assignment, we will explore both **structured and unstructured pruning** in the context of Convolutional Neural Networks (CNNs), specifically focusing on the **VGG-11** architecture. Our approach will closely follow the methodology of [2] and will draw inspiration from the first assignment used in this course: https://hanlab.mit.edu/courses/2024-fall-65940

1. **Unstructured Pruning:** In unstructured pruning, we aim to remove individual weights within the network, focusing on those that contribute least to the overall performance. For this assignment, you will implement unstructured pruning from scratch, using an L2-norm magnitude-based approach. By pruning weights with the smallest magnitudes (those closest to zero), you will iteratively reduce the network's complexity, setting the target sparsity ratios yourself.

   Before and after pruning, it is essential to visualize the distribution of the weights. You'll notice that pruning primarily affects weights near zero, which is advantageous when the weight distribution follows a bell curve. Visualizing these changes will help you understand the impact of pruning on the network's internal structure.

   A key part of this section is the **sensitivity analysis** where you can see the effect of increasing sparsity on model accuracy. You will independently prune each layer and observe how varying the target sparsity impacts accuracy. This will allow you to determine whether early or later layers are more sensitive to pruning, and you'll be expected to reason why this is the case. The final goal is to adjust the sparsity ratios across layers such that your final pruned model achieves 70% overall sparsity. To summarize:

   (a) Use magnitude-based unstructured pruning to compress a VGG-11 model, plotting the distribution of the weight tensors before and after pruning.

   (b) Perform a sensitivity analysis and visualize which layers are more sensitive to pruning, across a set of sparsity ratio values.

   (c) Set the values for the sparsity ratios for each layer yourself, commenting on why you are making the choices you are. The goal is to have a model that has been pruned with a 70% ratio, and save this for later.

2. **Structured Pruning:** Structured pruning, on the other hand, removes entire channels or filters rather than individual weights, leading to more hardware-efficient networks. In this assignment, you will conduct **channel-wise pruning**, which involves jointly pruning output channels of one layer and corresponding input channels of the next. This is more complex than unstructured pruning, as the pruning decisions across layers are interdependent (assuming no special layers like BatchNorm in between). You are, again, encouraged to follow the structure and guidance from the course assignment of hanlab.mit.edu/courses/2024-fall-65940, particularly for implementing magnitude-based pruning across channels.

   Similar to unstructured pruning, you will conduct a sensitivity analysis for structured pruning, albeit at a coarser level. You will investigate whether the sensitivity trends (early vs. later layers) remain consistent with unstructured pruning. Your final model in this section should also maintain 70% of the original size in terms of memory footprint. More precisely:

   (a) Use magnitude-based structured pruning to compress a VGG-11 model, plotting the distribution of the weight tensors before and after pruning.

   (b) Perform a sensitivity analysis and visualize which layers are more sensitive to pruning, across a set of sparsity ratio values.

   (c) Set the values for the sparsity ratios for each layer yourself, commenting on why you are making the choices you are. The goal is to have a model that has been pruned with a 70% ratio (approximately), on a per-layer basis, and save this for later.

3. **Comparison and Analysis:** After pruning, you will compare the results of the original model, the unstructured pruned model, and the structured pruned model. You are required to generate Grad-CAM visualizations [3] for each model to analyze how robust pruning is in terms of accuracy and feature localization, refer to this repository: https://github.com/jacobgil/pytorch-grad-cam. In your report, discuss the accuracy drops observed across the different models, as well as the visual differences in the Grad-CAM outputs.

   Finally, you will compare structured and unstructured pruning in terms of their ease of application and hardware requirements. Consider the fact that structured pruning can be more beneficial for certain hardware

accelerators like GPUs, whereas unstructured pruning might run effectively even on CPUs. Additionally, report the inference times for all three models and graph the accuracy vs. target sparsity (taking 5 readings for this graph). Put more succinctly,

(a) Compare the three models (original, unstructured-pruned, structured-pruned), using a Grad-CAM analysis to highlight how distinct the models are from each other.

(b) Give a qualitative analysis of Structured vs. Unstructured Pruning, highlighting whether there are special hardware considerations that must be taken into account for either approach or any inherent limitations of these approaches in general.

(c) Compare inference times for a given query image (averaging over a few runs), and compare the storage on-device (trying to match the sparsity levels as best you can).

(d) Using a Pruning approach of your choice, create a single plot of validation/test accuracy vs. target sparsity (over the entire model), showing how the severity of pruning affects performance. You can take up to 5 readings for this.

Task 2: Quantization: Quantization is a model compression technique that reduces the precision of weights and activations, allowing for significant reductions in memory usage and computational demand. In this assignment, we will explore both Post-Training Quantization (PTQ) and Quantization-Aware Training (QAT) across several bit-widths: `fp16`, `bf16`, `int8`, and `int4`. These experiments will give you a hands-on understanding of how quantization affects model accuracy and how QAT can be used to recover performance. For a review on the basics of Quantization, and a supplementary discussion on the implications for hardware and other aspects of implementation, please refer to [4].

For this task, it is **highly recommended** for you to use the `torchao` library for this task: https://github.com/pytorch/ao/tree/main - there are many quantization schemes already defined for you to plug-and-play with for the purposes of a comparative evaluation. You can get started with it here: https://pytorch.org/blog/pytorch-native-architecture-optimization/. You may also find following short courses useful:

- https://learn.deeplearning.ai/courses/quantization-fundamentals/lesson/1/introduction
- https://learn.deeplearning.ai/courses/quantization-in-depth/lesson/1/introduction

After you have gone through these resources, you will be able to do the following tasks:

1. **Post-Training Quantization (PTQ) and Quantization-Aware Training (QAT):** You will start by performing Post-Training Quantization (PTQ), which involves converting a pretrained model's weights to lower precision after the training process is complete. PTQ is simple to apply but may result in accuracy loss, particularly at very low bit-widths such as `int4`.

   Next, you will implement Quantization-Aware Training (QAT), which introduces quantization effects into the model during the training process, allowing the model to adapt and recover some of the performance lost in PTQ. QAT is typically more computationally expensive, but for this assignment, you will experiment with a few training epochs to observe how quickly the model recovers performance at different bit-widths.

   For each method, you will apply quantization at the following bit-widths:

   - `fp16`: 16-bit floating point
   - `bf16`: 16-bit brain floating point (a format optimized for training stability)
   - `int8`: 8-bit integer
   - `int4`: 4-bit integer

   By comparing PTQ and QAT, you will quantify the trade-off between simplicity and performance recovery. Make sure to carefully analyze the accuracy drop after PTQ and how many epochs of QAT are required to regain performance across different bit-widths.

2. **Scaling Law Analysis:** Using the results from your PTQ and QAT experiments, you will perform a scaling law analysis to examine how model performance changes with decreasing bit precision. Plot the accuracy vs. bit-width for each quantization method (PTQ and QAT), and note any trends. Are there diminishing returns with lower bit-widths? Does QAT offer significant improvements over PTQ at higher bit-widths like `fp16`, or is its impact more noticeable at lower bit-widths like `int8` and `int4`?

   You are **highly** recommended to read the following paper [5] - this is based on Large-Language Models but also examines the scaling laws of the model performance as we vary the total number of bits in the model. The grading of this part will hold you up to a comparable standard.

   By analyzing the scaling behavior of quantization, you can better understand how aggressively you can reduce precision without severely impacting model performance.

3. **Mixed-Precision Quantization Analysis:** Now, let us explore **mixed-precision quantization**, where different layers of the network are assigned different numeric precisions based on their sensitivity to quantization error. Using the same VGG-11 architecture on the CIFAR-100 dataset, start by identifying critical layers such as the first convolution, final classifier, and residual connections (if any) that are typically more sensitive to precision loss. These layers should retain a higher precision format, such as `fp16` or `bf16`, while the remaining intermediate layers are quantized to lower bit-widths (e.g., `int8` or `int4`). Your goal is to examine how selective precision assignment can help balance **model accuracy, storage compression, and inference latency**.

   Train and evaluate three configurations: (i) uniform quantization where all layers use the same bit-width (baseline), (ii) simple mixed-precision quantization where only the first and last layers use higher precision, and (iii) adaptive mixed-precision quantization where each layer's precision is chosen based on its observed

activation variance or Hessian sensitivity [6] – read about them. Record the model size, test accuracy, and inference latency for each case, and analyze the trade-offs. Plot accuracy versus model size to visualize how precision allocation impacts performance efficiency. Discuss whether using higher precision in early and late layers substantially mitigates accuracy degradation, and comment on any diminishing returns when further fine-tuning layer-wise precision.

4. **Effect of Outliers on Quantization Performance:** In this sub-task, you will study how **outliers in activations or weights** affect quantization quality and the overall performance of low-precision models. Using the VGG-11 model on CIFAR-100, begin by analyzing activation distributions across several layers after training a full-precision baseline model. Use histograms or box plots to visualize the presence of outliers—large activation values that dominate the scale of the tensor and force quantizers to allocate a large dynamic range for a few extreme values. Such outliers can lead to inefficient quantization because they reduce the effective resolution available for most of the distribution, thereby increasing quantization noise for the majority of values.

   Next, perform experiments with **different outlier-handling strategies**. Compare two variants: (i) standard uniform quantization (no clipping), and (ii) clipped quantization, where activations are truncated to a fixed percentile range (e.g., 99.9%) [7] (another useful variant is to use logarithmic or adaptive scaling, where quantization levels are distributed more densely around smaller magnitudes [8]). Quantize both weights and activations to low bit-widths such as `int8` and `int4`, then evaluate the resulting models on CIFAR-100. Report accuracy and visualizations of the modified distributions after clipping or scaling.

   Analyze how outlier suppression affects performance across layers—does clipping activations in deeper layers yield more stable results? Discuss whether certain channels or feature maps are more prone to producing large outliers and how this might depend on batch normalization or ReLU saturation. Conclude by connecting your observations to broader challenges in quantization: why outlier-robust quantizers are crucial for deployment in large models (e.g., Vision Transformers).

Task 3: Knowledge Distillation: Knowledge Distillation (KD) is a model compression technique in which a smaller model (student) learns to mimic the behavior of a larger, more powerful model (teacher). This technique was first introduced by [9], where the student is trained to match the "softened" output logits of the teacher - hence termed **Logit Matching (LM)**. The softened logits carry more information than hard labels, allowing the student to learn from the richer knowledge embedded in the teacher's predictions. Over time, KD has expanded beyond just logit-matching to include other methods like **Hint-based Distillation ("Hint")** [10], where intermediate feature representations of the teacher are transferred to the student, and **Contrastive Representation Distillation (CRD)** [11], which focuses on aligning the learned representations of the teacher and student models in a contrastive manner.

These approaches open different avenues for understanding what knowledge gets distilled during the process and how information transfer between models works. The essence of KD lies not only in compressing models but also in imparting specific properties from the teacher, such as improved generalization or robustness. Several works have explored whether larger teachers always provide better students [12] and what aspects of knowledge are effectively transferred [13]. Furthermore, research like [14] investigates whether the true benefits of distillation lie beyond just reducing model size, contributing to a deeper understanding of KD's inner workings.

Note: For previous tasks, you used a single VGG-11 on the CIFAR-100 dataset. For this setup, you will use the independent student, S_I, to be VGG-11 and the teacher, T, to be VGG-16 unless specified otherwise. **Note: use VGG-16 Teacher pretrained and train student VGG-11 from scratch.** You are also advised to go through [13] since we will take inspiration from their methodology for this section. You may also find following repositories useful:

- https://github.com/HobbitLong/RepDistiller
- https://github.com/AberHu/Knowledge-Distillation-Zoo

1. **Logit Matching:** In this task, you will compare three approaches: *Basic Logit Matching*, *Label Smoothing regularization* [15], and *Decoupled Knowledge Distillation* [16]. These methods, while different in their application, share a common theme of transferring or regularizing knowledge to improve model performance. However, the differences in how each technique defined distillation loss which may result in varying performance outcomes.

   Logit Matching, as explored earlier, focuses on mimicking the softened logits of a teacher model during knowledge distillation. The student learns from the teacher's richer softmax outputs, which convey more information than hard labels alone.

   In contrast, *Label Smoothing* is a regularization technique that prevents the model from becoming overconfident by softening the target labels. Instead of one-hot encoded labels (where one class is fully correct and others are incorrect), Label Smoothing assigns a small probability to incorrect classes, ensuring that the model doesn't overfit on the exact class distributions. The referenced paper discusses this in Section 7.

   Decoupled Knowledge Distillation separates the knowledge transfer into two components: the learning of the target class and the learning of the non-target class. This method gives more control over the optimization process, potentially leading to more effective distillation by allowing the student to focus on specific components of the teacher's knowledge for each category or channel.

   Your goal for this task is to implement these approaches and to compare and analyze the results. You should holistically discuss why you achieved the results to it, what you expected vs. what you got, and rationalize why a particular approach may be better than another.

2. **Comparing Performance of SoTA Approaches:** The first step is to implement and train models using the three different KD methods mentioned above.

   For all four models (LM, Hints, CRD, and S_I), train them on the same dataset and report the classification accuracy on a validation set - note that it is on you to decide how to set up this experiment in terms of training (fairly). This will provide a baseline comparison of how the different KD methods improve performance over an independent student. Your analysis should discuss how each KD method affects the final accuracy of the student model compared to both T and S_I.

3. **Comparing Probability Distributions:** Once the models are trained, the next step is to investigate how well the student models approximate the probability distributions generated by the teacher. You will focus on comparing the output probability distributions (softened labels) between:

   - T vs. S_D for each distillation method (Logit-Matching, Hints, and CRD).
   - T vs. S_I for the independent student model.

This analysis will allow you to see whether the student models' predictions are more aligned with the teacher's predictions than those of the independent student.

For a set of query images, compute the average discrepancy of the probability distributions for the teacher T and compare them to the distributions of the student models S for each KD method as well as the independent student - it is up to you to choose and justify your choice of measure such as Kl divergence. Analyze whether the distributions of the KD-based students are closer to the teacher's distributions compared to the independent student. Discuss whether any of the KD approaches leads to a significantly closer alignment with the teacher's predictions.

4. **Examining Localization Knowledge Transfer:** KD can transfer not only predictive power but also the teacher model's ability to focus on relevant features within an image. In this section, you will use **GradCAM** [3] to generate visualizations that highlight which parts of an image are most important for making a classification decision.

   For the teacher T, each of the KD-based student models S_D (LM, Hints, CRD), and the independent student S_I, generate GradCAM visualizations for a set of query images. These visualizations will show which areas of the image the models focus on when making predictions. Compare the GradCAM outputs for all models (**quantifying** the similarity similar to [13]), and discuss whether the student models trained with KD are able to better replicate the teacher's focus compared to the independent student. Analyze whether the KD methods help the student models localize important features in a manner similar to the teacher.

5. **Checking for Color Invariance with CRD:** Invariance to transformations, such as color variations, is an important property that models should ideally learn. In this section, you will test the ability of the student model to learn **color invariance** using the **CRD** method.

   To do this, you will finetune T with **color jitter (or other color-related) augmentations** applied to the training data and assess whether the model can handle color variations during inference - after training, evaluate the performance of the model on a color-jittered version of the validation set to see if it has learned to ignore color variations.

   Using this color-invariant teacher, perform the distillation process on S_I, using *regular augmentations* - we don't want the student to become invariant to color explicitly, rather to see if the properties of the teacher can be ambiently distilled into the student.

   Using this student S_D, evaluate the performance on a color-jittered version of the validation set. Discuss whether CRD helps the student model achieve color invariance, and compare this result with the other KD methods, if applicable.

6. **Testing the Efficacy of a Larger Teacher:** Finally, you will explore whether the size of the teacher model affects the performance of the student. In this section, you will use the **LM** approach to distill knowledge from two different teachers:

   - **VGG-16**: The teacher used in the previous sections.
   - **VGG-19**: A larger version of the teacher model.

   Train the VGG-11 student using Logit-Matching, first with VGG-16 as the teacher (you can use your results from previous tasks), and then with VGG-19 as the larger teacher. Compare the performance of the student models in terms of classification accuracy and any other relevant metrics. Discuss whether the larger teacher (VGG-19) leads to significantly better performance, or if the improvements are marginal. Analyze whether the size of the teacher model plays an important role in KD efficacy.

Task 4: Synthesis of results & Report Writing Guidelines: Now that you have conducted the experiments, the final task is to synthesize your findings into a coherent analysis and present it in a professional manner.

Deliverables:

- GitHub Repo: containing code (with documentation), any saved models or data subsets, and a README explaining how to run your analysis.
- PDF Report: ICML style, 6-10 pages including figures, addressing all points above. Treat it as a professional paper – clarity, structure, and correctness are key. Guidelines are given below. If you have additional results, you may add them in appendix after 10 pages. Write clearly and succinctly. Use bullet points or subheadings only if necessary.

Remaining instructions for report writing remain same as for previous assignments. However, for this particular assignment you can focus on comparing the three compression techniques—*Pruning*, *Quantization*, and *Knowledge Distillation*—in a holistic manner, while reflecting on the thoughts towards the end of end of the report.

Your comparison should consider various factors, including convenience, practical implementation, potential advantages and drawbacks, and the effectiveness of each technique under different constraints. One nice comparative analysis for reference is [17]. Here are some pointers you can consider:

1. **Convenience and Practicality**: Consider how easy each of these methods is to implement. Which technique is more intuitive or straightforward, and which one might require more in-depth tuning and experimentation? Reflect on the trade-offs in terms of complexity and time required for proper implementation.
2. **Potential Drawbacks and Advantages**: Discuss any drawbacks or weaknesses that might arise when using these techniques in practice. Do some methods introduce additional vulnerabilities or reduce model robustness? Are there specific scenarios where one technique offers a clear advantage over the others? Think about edge cases, performance degradation, and any unexpected outcomes based on your experiments.
3. **Effectiveness Given a Fixed Compute Budget**: With limited computational resources, which technique would offer the best trade-off between simplicity, accuracy, and overall efficiency? Consider both training and inference stages and reflect on how resource constraints may affect the choice of compression technique.
4. **Can These Approaches Be Used Together?**: Finally, ponder whether these techniques can or should be used simultaneously. Are they mutually exclusive, or is there potential for synergy when combining them? Discuss whether applying multiple compression techniques in tandem (e.g., pruning and quantization, or distillation and quantization) could lead to enhanced model efficiency without compromising performance.

Summarize your thoughts and present a final recommendation on which technique (or combination of techniques) is the most effective overall. Consider the balance between simplicity and effectiveness, and reflect on how these techniques align with different goals, such as reducing memory usage, speeding up inference, or preserving accuracy.

Your analysis should demonstrate a deep understanding of the strengths and limitations of each approach, supported by the results from your experiments. A well-rounded discussion that touches on all the points above will be essential for achieving full credit - do not cheap out here.

References

[1] Yann LeCun, John Denker, and Sara Solla. Optimal brain damage. In D. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2. Morgan-Kaufmann, 1989.

[2] Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural networks, 2015.

[3] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. *International Journal of Computer Vision*, 128(2):336–359, October 2019.

[4] Markus Nagel, Marios Fournarakis, Rana Ali Amjad, Yelysei Bondarenko, Mart van Baalen, and Tijmen Blankevoort. A white paper on neural network quantization, 2021.

[5] Tim Dettmers and Luke Zettlemoyer. The case for 4-bit precision: k-bit inference scaling laws, 2023.

[6] Zhen Dong, Zhewei Yao, Daiyaan Arfeen, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. Hawq-v2: Hessian aware trace-weighted quantization of neural networks. *Advances in neural information processing systems*, 33:18518–18529, 2020.

[7] Yefei He, Luoming Zhang, Weijia Wu, and Hong Zhou. Data-free quantization with accurate activation clipping and adaptive batch normalization. *Neural Processing Letters*, 55(8):10555–10568, 2023.

[8] Sangyun Oh, Hyeonuk Sim, Jounghyun Kim, and Jongeun Lee. Non-uniform step size quantization for accurate post-training quantization. In *European Conference on Computer Vision*, pages 658–673. Springer, 2022.

[9] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015.

[10] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. Fitnets: Hints for thin deep nets, 2015.

[11] Yonglong Tian, Dilip Krishnan, and Phillip Isola. Contrastive representation distillation, 2022.

[12] Jang Hyun Cho and Bharath Hariharan. On the efficacy of knowledge distillation, 2019.

[13] Utkarsh Ojha, Yuheng Li, Anirudh Sundara Rajan, Yingyu Liang, and Yong Jae Lee. What knowledge gets distilled in knowledge distillation?, 2023.

[14] Samuel Stanton, Pavel Izmailov, Polina Kirichenko, Alexander A. Alemi, and Andrew Gordon Wilson. Does knowledge distillation really work?, 2021.

[15] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision, 2015.

[16] Borui Zhao, Quan Cui, Renjie Song, Yiyu Qiu, and Jiajun Liang. Decoupled knowledge distillation, 2022.

[17] Andrey Kuzmin, Markus Nagel, Mart van Baalen, Arash Behboodi, and Tijmen Blankevoort. Pruning vs quantization: Which is better?, 2024.