Knowledge Distillation

Training large Deep Learning models is computationally expensive, and therefore finanically infeasible for researchers outside of large corporations. Models often need more than 100 million parameters to achieve state-of-the-art (SOTA) performance, and training those models requires a lot of computational resources, e.g. GPUs, time and data. For example, CoCa, a vision model reaching SOTA performance of 91% validation accuracy on ImageNet-1K [1], [2], has 2.1 billion parameters, was trained on more than 3 billion images [3], and, based on our approximation, should have cost over 350 thousand USD to train[1].

One strategy to avoid high computational costs is transfer learning. Here, a, potentially large, pretrained model is used as a starting point, and finetuned on a specific task, for a potentially different use case. That way, features learned by the model during pretraining can be reused, and the model can be adapted to the new task with less data. The disadvantage of this approach is that the model size does not change, so finetuning is still computationally expensive, especially for large models. A viable strategy would be to only use a few layers from the pretrained model, but since the environment in which those layers were trained is different from the one in which they are used during finetuning, this approach requires longer training times to adapt to the new environment.

Another option is knowledge distillation (KD). Here, a smaller model, the student model, is trained to replicate, or rather predict the outputs of a larger model, the teacher model, for a given sample. There are two strategies of KD usually used in practice: Response-based KD and feature-based KD [4]. Both will be used in this work.

Knowledge distillation has the advantage that the student model can be much smaller, and have a different architecture, compared to the teacher model. Since the teacher is running in inference mode no backpropagation is needed, and thus no gradients have to be computed (this is still required in simple transfer learning). This makes KD faster and requires less memory compared to finetuning. Most importantly, it has been empirically shown that student models much smaller than their teachers can achieve similar performance. For example, the distilled model of BERT, DistilBERT, reduced the model size by 40%, while retraining 97% of the performance of the original model [5].

Response-based Knowledge Distillation

In response-based KD, the teacher must provide a probability distribution over a set of classes for a given sample, which is the prediction of the teacher. The student model tries to replicate this probability distribution. This is also called soft targets, because the probability distribution is, unless the teacher is 100% sure, not one-hot encoded, but rather a smooth distribution over the classes. This increases the relative importance of logits with lower values, e.g. the classes with the second and third highest logits, and Hinton et al. [6] argue that this makes the model learn hidden encoded information the teacher model has learned, which are not represented when focusing on just the class with the highest logit/probability. This helps the student model to generalize better, especially on less data, compared to a model trained from scratch [6], [4].

The loss function typically used in response-based KD is the Kullback-Leibler divergence (KL), which measures the difference between two probability distributions. The mathematical formulation is as follows: Let $f(\cdot)$ be the teacher model, $g(\cdot)$ the student model, and $\boldsymbol{x}$ the input sample of any modalitiy, e.g. an image or a text. We omit the notation $\boldsymbol{H}_{v,0}$ and $\boldsymbol{H}_{w,0}$ for the input, as defined in (TODO: cite notation section), as knowledge distillation is independent of the modality and model architecture. Therefore, the input can be imagined as e.g. an image or text.

[1] Calculation done based on the price per TPU hour of the CloudTPUv4 on Google Cloud Platform with a three year commitment. CoCa was trained for 5 days using 2048 CloudTPUv4s. At a price of 1.449 USD per TPU hour (as of August 2024), the total cost is $1.449\ \text{USD/h} * 24\ \text{h/day} * 5\ \text{days} * 2048\ \text{TPUs} = 356,106.24\ \text{USD}$.

We define $\boldsymbol{u} = g(\boldsymbol{x})$ and $\boldsymbol{z} = f(\boldsymbol{x})$ as the output of the teacher and student model, respectively. Those are the logits, and for a classification task of e.g. 1000 classes, vectores of length 1000. A best practice is to divide the logits by a temperature parameter τ, before applying the softmax function [6], [4], which smooothes the probability distribution further, as illustrated in Figure 1.

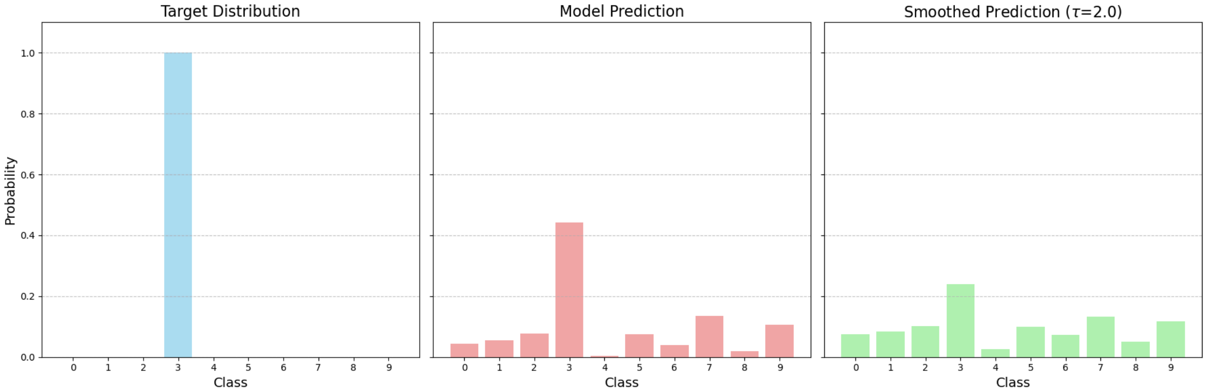

Figure 1: Comparison between the distribution for a classification task of 10 classes. The target distribution is used to train a model from scratch on a classification taks, while the model prediction over the classes can be used for knowledge distillation. The temperature parameter τ further smoothens the distribution, and increases the relative importance of classes with lower scores. Especially in distributions with large number of classes, e.g. ImageNet-1K [1], some classes will have semantic similarities, like "German Shorthaired Pointer" and "Labrador Retriever" [1], which are both dog breeds. The temperature parameter brings the scores for those classes closer together, which helps the student model to learn the hidden encoded information the teacher model has learned. In the setting of the dog breeds that means: Both classes are related/similar.

The temperature τ is usually a tuneable hyperparameter, but research has shown that it can also be a learned parameter, especially in other settings such as contrastive learning [3], introduced later.

$$p_i = \frac{\exp\left(\frac{u_i}{\tau}\right)}{\sum_j \exp\left(\frac{u_j}{\tau}\right)} \tag{1}$$

$$q_i = \frac{\exp\left(\frac{z_i}{\tau}\right)}{\sum_j \exp\left(\frac{z_j}{\tau}\right)} \tag{2}$$

i and j denote indices of the classes, and p_i and q_i the probabilities of class i according to the teacher $g(\cdot)$ and student model $f(\cdot)$, respectively. The goal is to minimize the difference between both distributions (the probabilities over all classes), computed by the KL-Divergence (introduced in (TODO:cite)).

Consequently, recalling the definition of the KL-Divergence in (TODO: cite), the training objective of response-based knowledge distillation is to bring the probability distributions of the student model, for a given sample (or all samples in general), as close as possible to the probability distributions of the teacher model for the same sample(s).

Feature-based Knowledge Distillation

In feature-based KD, the teacher model does not need to provide a probability distribution over classes. Instead, the student model tries to replicate the (intermediate) activations of the teacher model, so the student model does not necessarily have to only predict the final output (probability distribution) of the teacher model.

The activations of the teacher model are usually regressed using the Mean Squared Error (MSE) as the loss function (defined in (TODO: cite)). The Mean Absolute Error (MAE) can also be used as a criterion, although it is less common [4], [7], [8].

How the activations of the teacher model are regressed by the student model can be adjusted to the specific use case. However, this choice is greatly influenced by the architecture of the student model. For example, if the student model has the same architecture as the teacher, but only half the number of layers, then the student model can't replicate the activations of the teacher 1:1. In this case, other strategies have to be used, and we will introduce one of them in (TODO: cite unimodal kd experiments). An illustration of response-based vs. feature-based KD is shown in Figure 2.

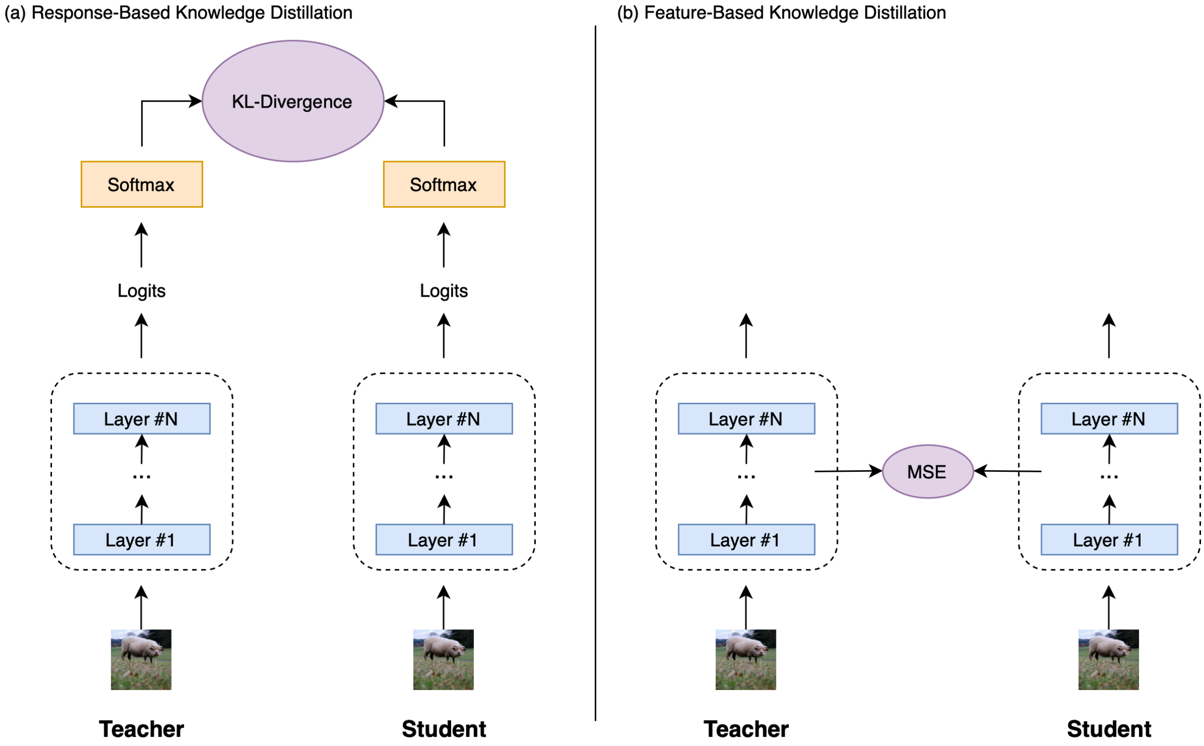

Figure 2: Response-based knowledge distillation (a) requires a teacher to provide logits, from which a probability distribution can be created, which is predicted by the student. Feature-based knowledge distillation (b) is used for predicting the actual activations of the teacher's layer(s). In both cases the weights of the teacher are frozen and the teacher is running in evaluation/inference mode. Figure adapted and inspired by [4], image is taken from COCO train set [9].

Bibliography

[1] O. Russakovsky *et al.*, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015, doi: 10.1007/s11263-015-0816-y.

[2] J. Yu, Z. Wang, V. Vasudevan, L. Yeung, M. Seyedhosseini, and Y. Wu, "CoCa: Contrastive Captioners are Image-Text Foundation Models," *Transactions on Machine Learning Research*, 2022, [Online]. Available: https://openreview.net/forum?id=Ee277P3AYC

[3] H. Bao *et al.*, "VLMo: Unified Vision-Language Pre-Training with Mixture-of-Modality-Experts," in *Advances in Neural Information Processing Systems*, 2022. [Online]. Available: https://openreview.net/forum?id=bydKs84JEyw

[4] J. Gou, B. Yu, S. J. Maybank, and D. Tao, “Knowledge Distillation: A Survey,” *International Journal of Computer Vision*, vol. 129, no. 6, pp. 1789–1819, 2021, doi: 10.1007/s11263-021-01453-z.

[5] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, “DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter,” in *NeurIPS EMC^2 Workshop*, 2019.

[6] G. Hinton, O. Vinyals, and J. Dean, “Distilling the Knowledge in a Neural Network.” [Online]. Available: https://arxiv.org/abs/1503.02531

[7] A. Baevski, W.-N. Hsu, Q. Xu, A. Babu, J. Gu, and M. Auli, “data2vec: A general framework for self-supervised learning in speech, vision and language,” *arXiv abs/2202.03555*, 2022.

[8] A. Baevski, A. Babu, W.-N. Hsu, and M. Auli, “Efficient Self-supervised Learning with Contextualized Target Representations for Vision, Speech and Language.” 2022.

[9] T.-Y. Lin *et al.*, “Microsoft COCO: Common Objects in Context,” in *Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V*, D. J. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Eds., in Lecture Notes in Computer Science, vol. 8693. Springer, 2014, pp. 740–755.