Unimodal Knowledge Distillation

To validate whether unimodal knowledge distillation even works, which is undoubtedly a simpler task than the multimodal knowledge distillation we are trying to develop, we will first conduct experiments on unimodal knowledge distillation. That is, the usual knowledge distillation introduced in section (TODO: cite kd).

Vision

Method

Our approach to vision KD involves using BEiTv2, self-supervised pretrained on ImageNet-1K [1], [2], as the teacher model and training a shallow version of the vision variant from Data2Vec2 [3] as the student model. This approach might be seen as unusual by some, as a more intuitive choice would be to construct a small BEiTv2 variant for the student model, and then training it to mimic the large model. DistilBERT [4] for example select half of the layers from a pretrained BERT [5] model, and organizes them into a smaller model, which is then trained to replicate the output of the pretrained BERT. However, we use layers of a different pretrained model and organize them into our student, as this step will be inevitable in the multimodal case. This is becuase the multimodal student model will later not only be far more complex, but its multimodal nature also requires a different architecture than the teacher. We believe that taking a similar approach here will give us valuable insights about the feasibility (of multimodal KD) on a comparatively simple exampe, and provide us with a foundation on which we can build our approach to multimodal KD.

BEiTv2 is a self-supervised model, and therefore does not provide a probability distribution over classes that can be predicted using KL-Divergence. Instead, we only have access to the model's activations for each layer, so we have to resort to feature-based knowledge distillation. One option would be to predict the teacher's output for the cls token $\boldsymbol{h}^t_{v,L,[\texttt{I\_CLS}]}$, which aggregates the high level content of the image, and then use the mean squared error as the loss function. However, this neglects the activations for individual image patches and activations of intermediate layers.

This argument is quite similar to that of Data2Vec, a general framework for self-supervised pretraining of unimodal image, text and audio models [6]. The authors introduce "contextualized representations", which are the activations of all layers of a model for each time step of the input. Because of Self-Attention in Transformers, the activations for each image patch (time step) are influenced by all other image patches, and therefore not only encode information about a patches content, but also about its context in the image, i.e. the relationship to other patches. Consequently, contextualized representations are more informative than a single cls token, as they encode information about the image at different levels of abstraction, and how the model aggregates low level features to high level concepts. Since the goal of KD is to "mimic" the behavior of a teacher model for a given input in a compressed way, this is the exact information that should be transferred from the teacher to the student. Simply predicting the cls token would only "mimic" what information the teacher extracts from the image, but not how the information is extracted.

While the dimensions of our student model match those of the teacher model, they both have a hidden size of $d = 768$ and intermediate size of $d_{\text{ff}} = 3072$ for the feed-forward layers in the Transformer blocks, the number of layers in the student model ($L_s = 12$) is only half of that of the teacher model ($L_t = 12$). It is therefore not possible for each layer of the student model to mimic the behavior of the corresponding layer in the teacher model. Fortunately, experiments of the Data2Vec authors show that predicting the mean of all layer activations for each time step works as well as predicting the activations of each layer individually [6]. This suits our approach well, as the only mismatch between the teacher and student model is the number of layers, which is irrelevant when predicting the mean of all layer activations for each time step. The authors apply instance

normalization to the activations of each layer before averaging. Instance normalization is a normalization technique that works on each dimension of a sequence independently. For a sequence of embeddings/representations with length T, instance normalization is defined as follows:

$$h'_{j,d} = \frac{h_{j,d} - \mu_d}{\sqrt{\sigma_d^2 + \varepsilon}}, \qquad \mu_k = \frac{1}{T}\sum_{t=1}^{T} h_{t,k}, \qquad \sigma_k^2 = \frac{1}{T}\sum_{t=1}^{T}\left(h_{t,k} - \mu_k\right)^2 \tag{1}$$

Even though the formula might look complicated, it is quite simple in practice. For each embedding dimension d, the mean μ_d and standard deviation σ_d are calculated over all time steps T. In the case of an embedding dimension of $D = 768$, this means for one sample (e.g. a sequence representing an image) 768 means and standard deviations are calculated, one for each embedding dimension. Then, for each time step j, the embedding at time step i is normalized by normalizing each dimension of the embedding independently, using the corresponding mean and standard deviation computed for that dimension [7]. During the normalization, a small epsilon, e.g. $1e^{-8} = 10^{-8}$, is added to the standard deviation to prevent division by zero. For an illustrative comparison between instance normalization, batch normalization and layer normalization, see (TODO: cite normalization) in the appendix. We define the operation $\text{InstanceNorm}(\cdot)$ as instance normalization on a sequence of embeddings $\boldsymbol{H}$.

$$\text{InstanceNorm}(\boldsymbol{H}) = [\boldsymbol{h}'_1, \boldsymbol{h}'_2, ..., \boldsymbol{h}'_T] \tag{2}$$

After instance norm and averaging, parameter-less layer normalization is performed. We perform all three operations likewise. The target and prediction are therefore given by:

$$\begin{aligned} \boldsymbol{H'}^{t}_{v,l} &= \text{InstanceNorm}\Big(\boldsymbol{H}^{t}_{v,l}\Big), l \in \{1, 2, ..., L_t\} \\ \boldsymbol{H'}^{s}_{v,l} &= \text{InstanceNorm}\Big(\boldsymbol{H}^{s}_{v,l}\Big), l \in \{1, 2, ..., L_s\} \end{aligned} \tag{3}$$

$$\begin{aligned} \widehat{\boldsymbol{H}}^{t}_{v} &= \frac{1}{L_t}\sum_{l=1}^{L_t} \boldsymbol{H'}^{t}_{v,l} \\ \widehat{\boldsymbol{H}}^{s}_{v} &= \frac{1}{L_s}\sum_{l=1}^{L_s} \boldsymbol{H'}^{s}_{v,l} \end{aligned} \tag{4}$$

$$\begin{aligned} \boldsymbol{Y} &= \Big[\boldsymbol{y}_{[\texttt{I\_CLS}]}, \boldsymbol{y}_1, ..., \boldsymbol{y}_N\Big] = \text{LayerNorm}\Big(\widehat{\boldsymbol{H}}^{t}_{v}\Big) \\ \widehat{\boldsymbol{Y}} &= \Big[\hat{\boldsymbol{y}}_{[\texttt{I\_CLS}]}, \hat{\boldsymbol{y}}_1, ..., \hat{\boldsymbol{y}}_N\Big] = \text{LayerNorm}\Big(\widehat{\boldsymbol{H}}^{s}_{v}\Big) \end{aligned} \tag{5}$$

The loss for a single sample (image) is defined in the following:

$$\mathcal{L}_{\text{KD}}\Big(\boldsymbol{Y}, \widehat{\boldsymbol{Y}}\Big) = \|\boldsymbol{Y} - \widehat{\boldsymbol{Y}}\|_2^2 = \frac{1}{N+1}\left(\sum_{n=1}^{N}\mathcal{L}_{\text{MSE}}(\boldsymbol{y}_n, \hat{\boldsymbol{y}}_n) + \mathcal{L}_{\text{MSE}}\Big(\boldsymbol{y}_{[\texttt{I\_CLS}]}, \hat{\boldsymbol{y}}_{[\texttt{I\_CLS}]}\Big)\right) \tag{6}$$

We denote $\boldsymbol{y}_i$ and $\hat{\boldsymbol{y}}_i$ as the average representation for image patch i over all layers from the teacher and student model, respectively. This includes instance norm before averaging, and layer norm after averaging. For the definition of $\text{LayerNorm}(\cdot)$, see (TODO: cite notation). $\mathcal{L}_{\text{MSE}}(\cdot, \cdot)$ is the mean squared error between two d-dimensional vectors, defined in (TODO: cite equation).

Pretraining

Finetuning

To get a sense of how well the student model has learned from the teacher, we evaluate the student model by finetuning it on the downstream image classification tasks of CIFAR-10 [8] and CIFAR-100 [8] and ImageNet-1K [1]. For that, we load the trained student model and add Layer Normalization and a linear classifier on top of it. The output of the student model is a sequence of embeddings, one embedding for each image patch, and one cls token embedding. We follow the approach of Data2Vec [6], [3] and BEiTv2 [2], and take the mean over all patch embeddings as the output of the student model, which is then passed to the layer normalization and linear classifier. For all three tasks we perform full finetuning, i.e. we finetune all layers of the student model on the target task, and linear probing, i.e. we only traing the added linear classifier on top of the student model. For pytorch pseudocode of linear probing and full finetuning, see (TODO: cite pseudocode).

For data augmentation during finetuning, we use RandAugment [9], mixup [10] and cutmix [11] augmentation, and random erasing [12]. The hyperparameters for these augmentations are provided in (TODO: cite hyperparameters), and have been selected based on the values used in BEiTv2 [2], Data2Vec [6], and Data2Vec2 [3]. We refrain from explaining the augmentations in detail here, as they are well documented in the respective papers. As for distillation, we use one RTX 4090 24GB GPU for finetuning.

For finetuning of ImageNet-1K, we use a base learning rate of 1-e3 in combination with layer decay. Layer decay is a technique to reduce the base learning rate for each layer of the model by a certain factor. The goal is to have lower learning rates for layers closer to the input, and higher learning rates for layers closer to the output. This ensures that low-level features learned during pretraining or distillation are not destroyed during finetuning. We use a decay factor of 0.81, which is derived by scaling the layer decay used in Data2Vec2 [3], from which we extract layers for the student model, by the square root. We use scaling instead of the value used in Data2Vec2, which is 0.65 ($\sqrt{0.65} \approx 0.81$), as we only have half the number of layers in the student model, and can therefore afford larger learning rates for the lower layers. The actual learning rates a layer l is then calculated by:

$$\mathrm{lr}_l = \mathrm{base\_lr} * \mathrm{layer\_decay}^{L_s+1-l} \tag{7}$$

The learning rates can be seen in the following table:

Layer no.	0	*1*	*2*	*3*	*4*	*5*	*6*	7
Learning rate	2.3e-4	2.8e-4	3.5e-4	4.3e-4	5.3e-4	6.6e-4	8.1e-4	1e-3

Table 1: Cursive layer numbers indicate Transformer blocks. The learning rates are calculated using a base learning rate of 1e-3 and a layer decay of 0.81.

Language

Method

Pretraining

Finetuning

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] Z. Peng, L. Dong, H. Bao, Q. Ye, and F. Wei, "BEiT v2: Masked Image Modeling with Vector-Quantized Visual Tokenizers," 2022.

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

[4] 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.

[5] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, J. Burstein, C. Doran, and T. Solorio, Eds., Minneapolis, Minnesota: Association for Computational Linguistics, Jun. 2019, pp. 4171–4186. doi: 10.18653/v1/N19-1423.

[6] 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.

[7] D. Ulyanov, A. Vedaldi, and V. Lempitsky, "Instance Normalization: The Missing Ingredient for Fast Stylization." [Online]. Available: https://arxiv.org/abs/1607.08022

[8] A. Krizhevsky, "Learning multiple layers of features from tiny images," 2009.

[9] E. D. Cubuk, B. Zoph, J. Shlens, and Q. V. Le, "Randaugment: Practical automated data augmentation with a reduced search space," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2020, pp. 3008–3017. doi: 10.1109/CVPRW50498.2020.00359.

[10] H. Zhang, M. Cissé, Y. N. Dauphin, and D. Lopez-Paz, "mixup: Beyond Empirical Risk Minimization," in *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, OpenReview.net, 2018. [Online]. Available: https://openreview.net/forum?id=r1Ddp1-Rb

[11] S. Yun, D. Han, S. Chun, S. Oh, Y. Yoo, and J. Choe, "CutMix: Regularization Strategy to Train Strong Classifiers With Localizable Features," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, Los Alamitos, CA, USA: IEEE Computer Society, Nov. 2019, pp. 6022–6031. doi: 10.1109/ICCV.2019.00612.

[12] Z. Zhong, L. Zheng, G. Kang, S. Li, and Y. Yang, "Random Erasing Data Augmentation," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 7, pp. 13001–13008, Apr. 2020, doi: 10.1609/aaai.v34i07.7000.