HOCHSCHULE
HANNOVER
UNIVERSITY OF
APPLIED SCIENCES
AND ARTS
–
Fakultät IV
Wirtschaft und
Informatik

Titel der Arbeit

Tim Cares

Master's thesis in Applied Computer Science

25. September 2024

Autor Tim Cares
Matrikelnummer
Email Adresse

Erstprüfer Prof. Dr. Vorname Name
Abteilung Informatik, Fakultät IV
Hochschule Hannover
Email Adresse

Zweitprüfer Prof. Dr. Vorname Name
Abteilung Informatik, Fakultät IV
Hochschule Hannover
Email Adresse

I hereby declare that I have written and submitted this thesis independently, without any external help or use of sources and aids other than those specifically mentioned by me. I also declare that I have not taken any content from the works used without proper citation and acknowledgement.

Hannover, 25. September 2024 Tim Cares

Acknowledgements

Abstract

Contents

List of Figures

List of Tables

1 Background

1.1 Notations and Definitions

Throughout this work we will make use of various concepts and their notations, which we will define here for easier reference, and to avoid redundancy. Bold symbols (e.g. $\boldsymbol{v}$) denote vectors, and v_i the i-th element of the respective vector. Upper-cased bold symbols (e.g. $\boldsymbol{M}$) denote matrices, and M_{ij} the element in the i-th row and j-th column of the respective matrix.

1.1.1 Loss Functions

1.1.1.1 Mean Squared Error (MSE)

The Mean Squared Error (MSE) is a loss function used in regression tasks, and describes the average of the squared differences between the prediction $\hat{\boldsymbol{y}} \in \mathbb{R}^d$ and the target $\boldsymbol{y} \in \mathbb{R}^d$. Since in this work the predictions and targets will exclusively be in the form of d-dimensional vectors, the MSE is defined as:

$$\mathcal{L}_{\text{MSE}}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \|\boldsymbol{y} - \hat{\boldsymbol{y}}\|_2^2 = \frac{1}{d} \sum_{j=1}^{d} \left(y_j - \hat{y}_j\right)^2 \tag{1}$$

1.1.1.2 Kullback-Leibler Divergence (KL-Divergence)

The Kullback-Leibler Divergence (KL-Divergence) is used to measure the difference between two probability distributions. Specifically, in the context of Machine Learning, we are comparing a predicted probability distribution $\boldsymbol{q} \in \mathbb{R}^d$ with a target distribution $\boldsymbol{p} \in \mathbb{R}^d$. Since we are using the KL-Divergence in the context of classification tasks, which are discrete distributions over classes, the KL-Divergence is defined as:

$$\mathcal{L}_{\text{KD}}(\boldsymbol{p} \parallel \boldsymbol{q}) = D_{\text{KL}}(\boldsymbol{p} \parallel \boldsymbol{q}) = \sum_j p_j \log \frac{p_j}{q_j} \tag{2}$$

p_j and q_j are the probabilities of class j according to the target and predicted distribution, respectively. For both distributions, there are potentially multiple classes with a non-zero probability:

$$\forall j\big(p_j \in [0,1]\big) \wedge \sum_j p_j = 1 \tag{3}$$

1.1.1.3 Cross-Entropy Loss (CE)

The Cross-Entropy Loss (CE) is quite similar to the KL-Divergence in that it compares two probability distributions in classification tasks. It is defined as:

$$\mathcal{L}_{\mathrm{CE}}(\boldsymbol{p}, \boldsymbol{q}) = H(\boldsymbol{p}, \boldsymbol{q}) = H(\boldsymbol{p}) + D_{\mathrm{KL}}(\boldsymbol{p} \parallel \boldsymbol{q}) = -\sum_j p_j \log p_j + \sum_j p_j \log \frac{p_j}{q_j} \tag{4}$$

Here $H(\boldsymbol{p})$ denotes the entropy of the target distribution $\boldsymbol{p}$, and $D_{\mathrm{KL}}(\boldsymbol{p} \parallel \boldsymbol{q})$ the KL-Divergence between the target and predicted distribution.

The difference between KL-Divergence and cross-entropy is that the latter is used in traditional classification tasks, where the target distribution $\boldsymbol{p}$ is fixed and one-hot encoded, meaning that there is only one correct class:

$$\exists! j\big(p_j = 1\big) \wedge \forall k(k \neq j \rightarrow p_k = 0) \tag{5}$$

This strengthens the condition of the KL-Divergence, which we defined previously in Equation 3. Since the goal is to minimize the cross-entropy loss $H(\boldsymbol{p}, \boldsymbol{q})$ and $\boldsymbol{p}$ is fixed, the entropy of the target distribution $H(\boldsymbol{p})$ is a constant, and does not affect the minimization. Moreover, given the constraint in Equation 5, only one term in the sum of the KL-Divergence is non-zero. Consequently, we can simplify the cross-entropy loss, so that the training objective for classification tasks is:

$$\begin{aligned} \min H(\boldsymbol{p}, \boldsymbol{q}) &= H(\boldsymbol{p}) + D_{\mathrm{KL}}(\boldsymbol{p} \parallel \boldsymbol{q}) \\ &= D_{\mathrm{KL}}(\boldsymbol{p} \parallel \boldsymbol{q}) \\ &= \sum_j p_j \log \frac{p_j}{q_j} \\ &= \log \frac{1}{q_i} \\ &= -\log q_i \end{aligned} \tag{6}$$

The cross entropy loss therefore minimizes the negative log-likelihood of the correct class i.

Often times, the prediction of a model $\boldsymbol{x}$ is returned as raw logits, and not as probabilities. To convert logits into probabilities, the softmax function is used. For ease of use, without

having to mention a softmax-normalization every time we make use of the cross-entropy loss, we redefine the cross-entropy loss *actually used in this work* as:

$$\mathcal{L}_{\mathrm{CE}}(\boldsymbol{p}, \boldsymbol{x}) = H(\boldsymbol{p}, \boldsymbol{x}) = -\log \frac{\exp(x_i)}{\sum_j \exp(x_j)} \tag{7}$$

We denote $\boldsymbol{x}$ as the raw logits (the model prediction), and $\boldsymbol{p}$ as the one-hot encoded target distribution. i is the index of the correct class, and hence each element in $\boldsymbol{x}$ corresponds to the raw logit for one class.

A comparison between the target distribution predicted using KL-Divergence, and another predicted by cross-entropy is shown in the following figure.

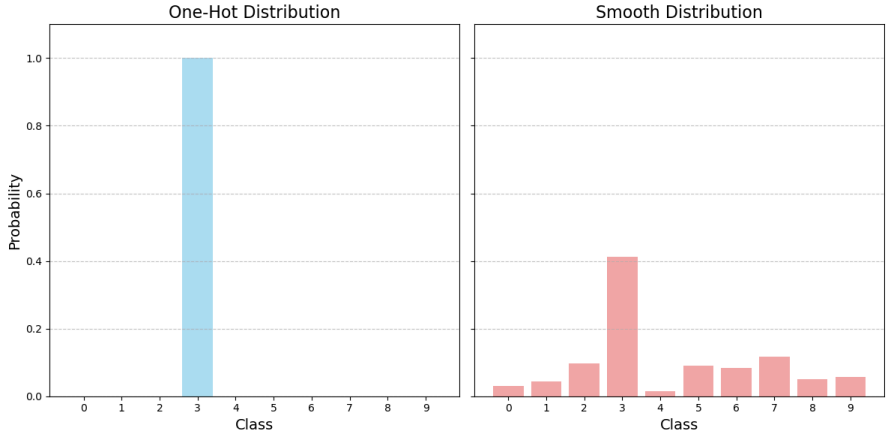

Figure 1: Comparison between the distributions with 10 classes. The one-hot distribution (left) is used for classification tasks with the cross-entropy loss. The KL-Divergence is used when predicting a smooth distribution (right). A smooth distribution usually results from a model prediction, and is a popular target distribution for knowledge distillation, introduced in a later section.

1.1.2 Modality Representations

Since the architectures used in the experiments of this work are based on the Transformer [Vas+17] and vision Transformer [Dos+21] architecture, both image and text are represented as sequences of embeddings, which are processed by the Transformer blocks.

1.1.3 Image Representation

We define an image as a 3-dimensional tensor $\boldsymbol{v} \in \mathbb{R}^{C \times H \times W}$. Because we will use the base variant of the vision Transformer, ViT-B/16 [Dos+21], the image is patchified into 14x14 patches, each being a square of size 16x16 pixels. Each image patch represents one timestep in the sequence, and the number of patches N is given by $N = H \times \frac{W}{P^2}$, with P being the number of patches per dimension, and $P = 14$. Since we use an image size of 224x224

pixels, so $\boldsymbol{v} \in \mathbb{R}^{3\times244\times244}$, we will have $N = 244 \times \frac{244}{14^2} = 196$ patches, or timesteps respectively. Each patch is flattened into a 256-dimensional vector, and then projected into a 768 dimensions $\boldsymbol{e}_i^v \in \mathbb{R}^{768}$, using a fully connected layer. The image sequence is prepended with a special learnable $[\texttt{I\_CLS}] \in \mathbb{R}^{768}$ token, which is, following [Dos+21], used to aggregate the global information/content of the image. The result is a sequence of patch embeddings, which we define as $\boldsymbol{E}_v$, where v indicates the image modality:

$$\boldsymbol{E}_v = \left[\boldsymbol{e}^v_{[\texttt{I\_CLS}]}, \boldsymbol{e}^v_1, \boldsymbol{e}^v_2, ..., \boldsymbol{e}^v_N\right] \tag{8}$$

To give the Transformer a sense of order in the image patches/timestep, a unique positional encoding is added to each patch embedding. This can either be learned or fixed, with the latter being for example a sinusoidal positional encoding [Vas+17]. This positional encoding is also represented as a sequence of 768-dimensional vectors:

$$\boldsymbol{T}^{\text{pos}}_v = \left[0, \boldsymbol{t}^v_{\text{pos}_1}, \boldsymbol{t}^v_{\text{pos}_2}, ..., \boldsymbol{t}^v_{\text{pos}_N}\right] \tag{9}$$

Since the $[\texttt{I\_CLS}]$ token is not part of the image, the positional encoding for the $[\texttt{I\_CLS}]$ token is set to zero, so nothing is added to it. An image representation is defined as:

$$\boldsymbol{H}^s_{v,l} = \left[\boldsymbol{h}^s_{v,l,[\texttt{I\_CLS}]}, \boldsymbol{h}^s_{v,l,1}, ..., \boldsymbol{h}^s_{v,l,N}\right] \tag{10}$$

In Equation 10, l denotes the layer of the Transformer block that returned the image representation, and v indicates that the representation is an image. Since we use Knowledge Distillation (KD) in some parts of this thesis, representations will be, if neccessary, superscripted with s or t, for a student and teacher representation, respectively.

We define $l = 0$ as the input to the Transformer, and $l = L$ as the output of the Transformer, where L is the number of layers in the Transformer. Consequently, the image input to the Transformer is defined as:

$$\boldsymbol{H}^s_{v,0} = \left[\boldsymbol{h}^s_{v,0,[\texttt{I\_CLS}]}, \boldsymbol{h}^s_{v,0,1}, ..., \boldsymbol{h}^s_{v,0,N}\right] = \boldsymbol{E}_v + \boldsymbol{T}^{\text{pos}}_v \tag{11}$$

The output of the Transformer is defined as:

$$\boldsymbol{H}^s_{v,L} = \left[\boldsymbol{h}^s_{v,L,[\texttt{I\_CLS}]}, \boldsymbol{h}^s_{v,L,1}, ..., \boldsymbol{h}^s_{v,L,N}\right] \tag{12}$$

1.1.4 Text Representation

We define a text as a sequence of discrete tokens, which are, similiar to image patches, embedded into 768-dimensional vectors using an embedding matrix. A single token i is represented as $\boldsymbol{e}^t_i \in \mathbb{R}^{768}$, and the sequence of tokens, representing the text, is prepended with a start-of-sequence token $[\texttt{T\_CLS}] \in \mathbb{R}^{768}$, and appended with an end-of-sequence token $[\texttt{T\_SEP}] \in \mathbb{R}^{768}$. The purpose of the $[\texttt{T\_CLS}]$ token is, as with $[\texttt{I\_CLS}]$, to aggregate the global

information/content of the text. The [T_SEP] token is used to indicate the end of the text sequence. A text sequence consists of M tokens, and we use w to denote a text sequence:

$$\boldsymbol{E}_w = \left[\boldsymbol{e}^w_{[\texttt{T\_CLS}]}, \boldsymbol{e}^w_1, \boldsymbol{e}^w_2, ..., \boldsymbol{e}^w_M, \boldsymbol{e}^w_{[\texttt{T\_SEP}]}\right] \tag{13}$$

The maximum text sequence length M is not fixed, and will be defined when neccessary in the experimental part of this work.

A positional encoding is also added to the text embeddings, to give the Transformer a sense of order in the text sequence. Since the special token [T_SEP] denotes the end of the text sequence, it is part of the sequence, and therefore has a positional encoding. The latter does not hold for the [T_CLS] token, as it is used to aggregate the global information/content of the text.

$$\boldsymbol{T}^{\text{pos}}_w = \left[0, \boldsymbol{t}^w_{\text{pos}_1}, \boldsymbol{t}^w_{\text{pos}_2}, ..., \boldsymbol{t}^w_{\text{pos}_M}, \boldsymbol{t}^w_{\text{pos}_{[\texttt{T\_SEP}]}}\right] \tag{14}$$

A text representation is defined as:

$$\boldsymbol{H}^s_{w,l} = \left[\boldsymbol{h}^s_{w,l,[\texttt{T\_CLS}]}, \boldsymbol{h}^s_{w,l,1}, ..., \boldsymbol{h}^s_{w,l,M}, \boldsymbol{h}^s_{w,l,[\texttt{T\_SEP}]}\right] \tag{15}$$

Equation 15 denotes the representation denoted by a student model s, but it can also be a teacher representation t.

The input to the Transformer for text is Equation 15 with $l = 0$, and the output of the Transformer is Equation 15 with $l = L$.

1.1.5 Transformer Block

Unless we use pretrained architectures that follow a different architecture, which we will then specify, we follow the Pre-LayerNorm definition of the Transformer block as given in [Xio+20]. As the name suggests, it applies LayerNorm before the Multi-Head Attention, instead of after.

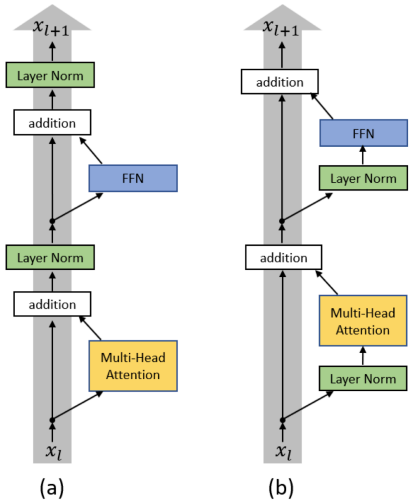

Figure 2: Comparison of a Post-Norm Transformer block/layer (a), and a Pre-Norm Transformer block/layer (b). (a) is the architecture as defined in the original "Attention is all you need" paper [Vas+17]. We follow the Pre-Norm architecture [Xio+20].

One Transformer block performs the following operations:

$$\boldsymbol{H}'_l = \text{MHA}(\text{LN}(\boldsymbol{H}_{l-1})) + \boldsymbol{H}_{l-1} \tag{16}$$

$$\boldsymbol{H}_l = \text{FFN}(\text{LN}(\boldsymbol{H}'_l)) + \boldsymbol{H}'_l \tag{17}$$

We denote LN as LayerNorm, MHA as Multi-Head Attention, and FFN as a 2 layer MLP, all following the original Transformer of [Vas+17]. As previously mentioned, the only difference is the order of operations [Xio+20]. $\boldsymbol{H}^s_{v,l}$ and $\boldsymbol{H}^s_{w,l}$ can be used as a drop-in replacement for image and text, respectively. Both equations are, with slight adjustment, taken from VLMo [Bao+22].

We define a Transformer as multiple Transformer blocks stacked on top of each other.

1.2 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 [Rus+15, Yu+22], has 2.1 billion parameters, was trained on more than 3 billion images

[Bao+22], 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 [Gou+21]. 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 [San+19].

1.2.1.1 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. [HVD15] 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 [Gou+21, HVD15].

[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 $1.449\ \mathrm{USD/h} * 24\ \mathrm{h/day} * 5\ \mathrm{days} * 2048\ \mathrm{TPUs} = 356,106.24\ \mathrm{USD}$.

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.

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 [Gou+21, HVD15], which smooothes the probability distribution further, as illustrated in Figure 3.

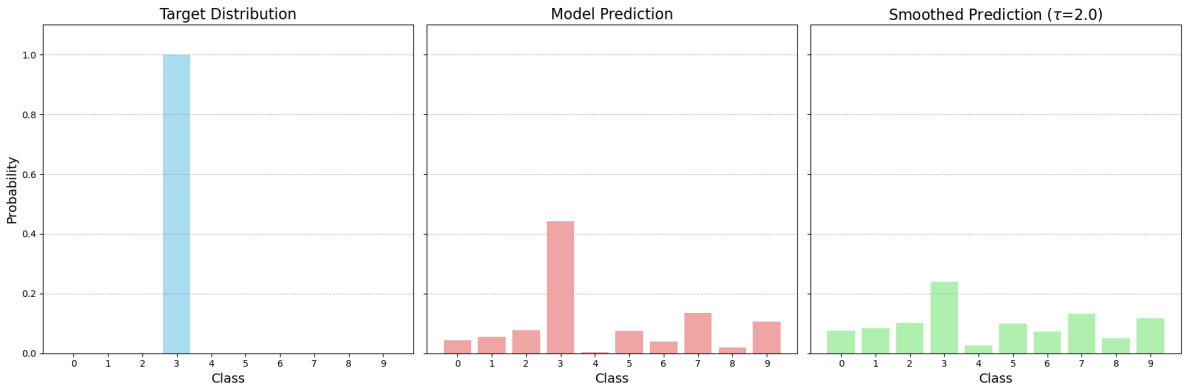

Figure 3: Comparison between the distribution for a classification task of 10 classes. The target distribution is used to train a model from scratch on a classification taks, while the model prediction over the classes can be used for knowledge distillation. The temperature parameter τ further smoothens the distribution, and increases the relative importance of classes with lower scores. Especially in distributions with large number of classes, e.g. ImageNet-1K [Rus+15], some classes will have semantic similarities, like "German Shorthaired Pointer" and "Labrador Retriever" [Rus+15], 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 [Bao+22], introduced later.

$$p_i = \frac{\exp\left(\frac{u_i}{\tau}\right)}{\sum_j \exp\left(\frac{u_j}{\tau}\right)} \tag{18}$$

$$q_i = \frac{\exp\left(\frac{z_i}{\tau}\right)}{\sum_j \exp\left(\frac{z_j}{\tau}\right)} \tag{19}$$

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).

1.2.1.2 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 [Bae+22, Bae+22, Gou+21].

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 4.

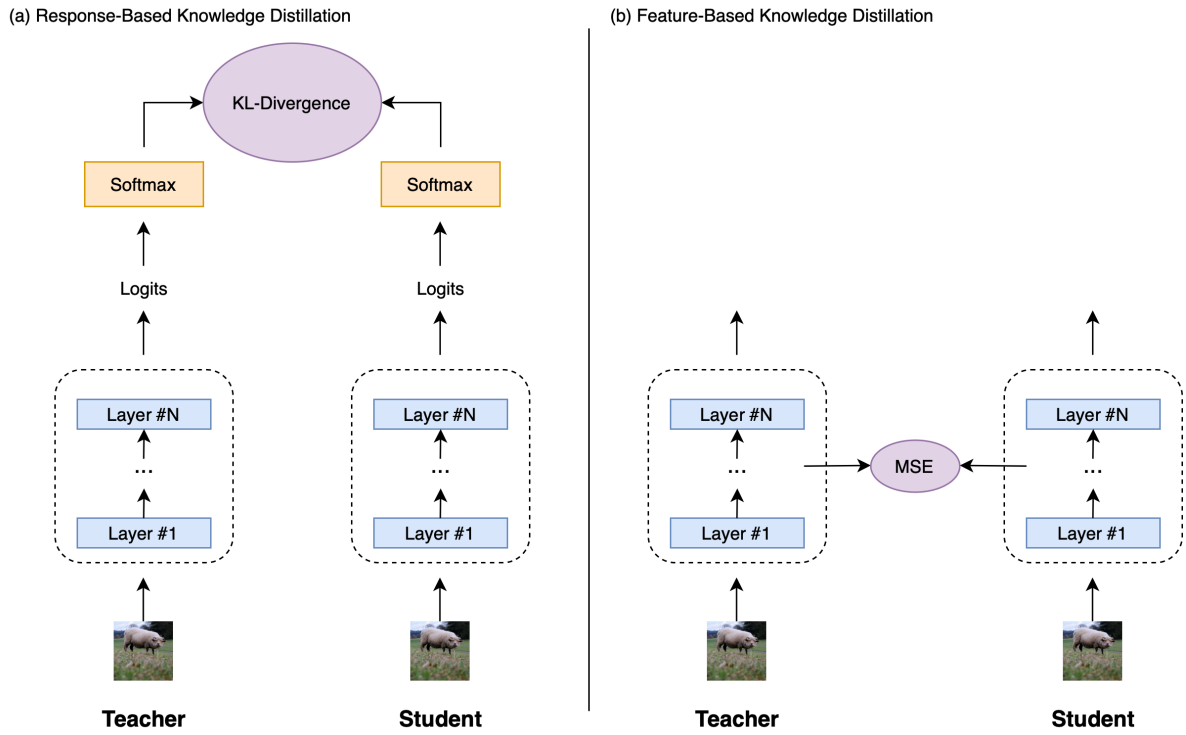

Figure 4: Response-based knowledge distillation (a) requires a teacher to provide logits, from which a probability distribution can be created, which is predicted by the student. Feature-based knowledge distillation (b) is used for predicting the actual activations of the teacher's layer(s). In both cases the weights of the teacher are frozen and the teacher is running in evaluation/inference mode. Figure adapted and inspired by [Gou+21], image is taken from COCO train set [Lin+14].

1.3 Self-Supervised Learning

While we previously identified large Deep Learning models as generally expensive to train, we now focus on the problem of scalability in the context of supervised learning, the most common form of training AI models.

Supervised models, while powerful, are not inherently scalable. Although their architecture can be extended to create larger models that achieve better performance, these larger models require more data for training. In the context of supervised learning, this data must be labeled, which presents a significant challenge. Labeled data is scarce and expensive to obtain, as it requires human annotation, thereby limiting the scalability of supervised models.

The primary objective of self-supervised learning is to learn representations of data without relying on human-annotated labels. However, self-supervised learning is not unsupervised learning. Unsupervised learning operates without any form of supervision, meaning that no labels are required at all, as seen in clustering methods like K-means. In contrast, self-super-

vised learning requires, as supervised learning, labeled data, but in contrast to supervised learning labels are generated directly from the data itself.

A prominent example of self-supervised learning is Masked Language Modeling (MLM) in Natural Language Processing (NLP), which is used in the popular NLP model BERT, being one of the first models trained using self-supervised methods to achieve state-of-the-art performance in NLP [Dev+19]. In BERT, certain tokens, or words, are masked, i.e., removed, from a sentence, and the model is tasked with predicting the masked tokens. Since the labels are derived from the data itself — the words to predict are part of the original data — no human annotation is needed [Dev+19]. This allows for the utilization of large amounts of unlabeled data, as any text data can be used.

What makes self-supervised learning particularly powerful is its applicability to any type of data with a hierarchical structure, such as text, images, audio, or video. In these cases, part of the data can be masked, and the model must predict the masked part based on the context provided by the remaining data. An intuitive example, presented by Yann LeCun and Ishan Misra of Meta, illustrates why this approach is effective. Consider the sentence "The lions chase the wildebeests in the savanna." If "lions" and "wildebeests" are masked, the input becomes "The [MASK] chases the [MASK] in the savanna.". To successfully predict the masked words, the model must understand the real-world concepts expressed by the sentence. While "The cat chases the mouse in the savanna" might be a valid prediction in the context of "chase," the word "savanna" provides additional context, as it is not a typical habitat for cats and mice, but rather for lions and wildebeests. Thus, the model must understand that lions and wildebeests are animals that inhabit savannas, in order to make a correct prediction. Through this process of predicting masked words, the model learns about the concepts of the world we live in [LM21].

While this example is specific to text data, the same principle can be applied to other types of hierarchical data, e.g. images and audio.

Consequently, self-supervised learning allows makes models scalable, as they can be trained on large amounts of unlabeled data. A fact we will come back to in the experimental part of this work, where will will approach multimodal models with this philosophy.

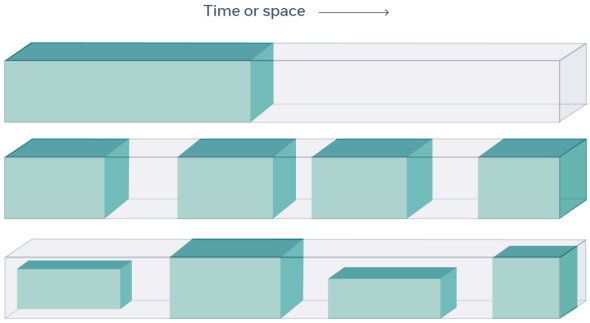

Figure 5: In self-supervised learning parts of the data are masked (grey), and the task of a model is to predict the masked parts using the visible data (green) [LM21].

1.4 Multimodal Models

Multimodal models are characterized by their ability to process multiple modalities, such as text, images, audio, or video, within a single model. The motivation behind these models lies in the idea that models should be able to understand real-world concepts in a way similar to humans. Humans can express the same concept across different modalities, “a cat”, for example, can be represented in text, image, or audio, and regardless of how the concept is expressed, the interpretation and understanding remains the same.

Please note that since our focus is on vision-language models, all further explanations will be based on the multimodality in the context of vision and language.

In the context of Deep Learning, this means that the representations of a concept should be the same (or at least close to each other), no matter if is expressed through text or image, which is also called alignment. However, in most existing models, this is not the case. These models are typically unimodal, meaning they process only one modality, making alignment of multiple modalities impossible. A naive approach would be to pass an image into an image model, and its caption into a text model. Even though the generated representations describe the same concept, they will not be the same, as both models are not related to each other. Each model will have a seperate latent space, as there has been no incentive for the models to learn a representation that is aligned across modalities (Figure 6), resulting in different representations for the same concept. While it is possible to compare the representations of two unimodal models, e.g. through cosine similarity, a similarity close to 1 (the maximum) does not necessarily mean that the concepts expressed in the representations are the same. There simply is no semantic relationship between the representations of the same concept produced by two unimodal models. A proof will be shown in (TODO: cite section where d2v2 image+text is used with retrieval).

To overcome this limitation, we need to develop models that can understand the same concept across different modalities, or input types respectively. They should map the input of

different modalities into a common representation space, where the representations of the same concept are aligned, i.e. close to each other.

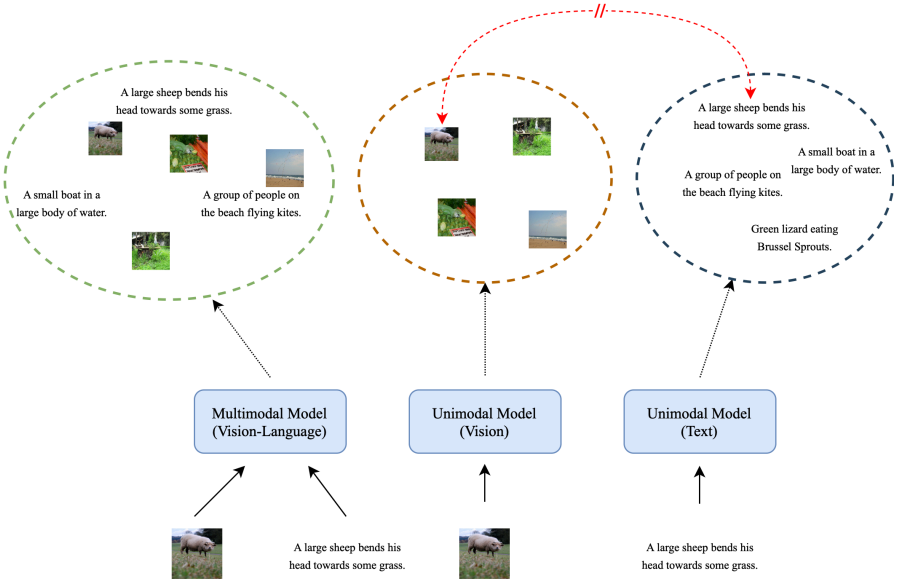

Figure 6: A multimodal model maps multiple modalities into a common representation space, where the representations of the same concept are aligned. In contrast, unimodal models map the input of a single modality into a modality-specific representation space. There is no alignment between the representations of the same concept produced by two unimodal models (indicated by the double slashed [//] arrow). While a comparison between the representations of two unimodal models is numerically possible, e.g. through cosine similarity, the similarity cannot be interpreted in a meaningful way.

Multimodal models consist of both unimodal encoders and multimodal components. Unimodal encoders are needed because of the inherent differences between modalities, e.g. image and text: Images are 2D and composed of pixels, while text is 1D and composed of words. Unimodal encoders encode the input into a modality-specific representation space, so they are normal unimodal models, e.g. a ResNet for images. In this work, all encoders will be based on the Transformer architecture.

Multimodal models require components that enforce a common representation space for the different modalities. There are two options: A multimodal (or shared) encoder, or a loss function (training objective).

The multimodal encoder is responsible for mapping the modality-specific representations into a unified/shared representation space, where representations should be independent of the modality. That means, the representations should not contain any modality-specific information, e.g. pixel information in images or single-word information in text. Only then representations of the same concept can be aligned, or close to each other under some distance metric, respectively.

To actually ensure that the representations of the same concept are aligned, and not only in the same space, a training objective is needed that pushes the representations of the same concept closer together in representation space, while pushing the representations of different concepts further apart. For vision-language models this translates to pushing the representations of an image and its caption closer together, while pushing the representations of an image and an unrelated caption (or vice versa) further apart. To quantify the similarity between two representations, a distance metric is used, e.g. cosine similarity. The loss function is usually the contrastive loss, and its implementation for vision-language models will be introduced in the next section. An illustration of a multimodal model is provided in Figure 7, concrete examples will be introduced in (TODO: cite related work).

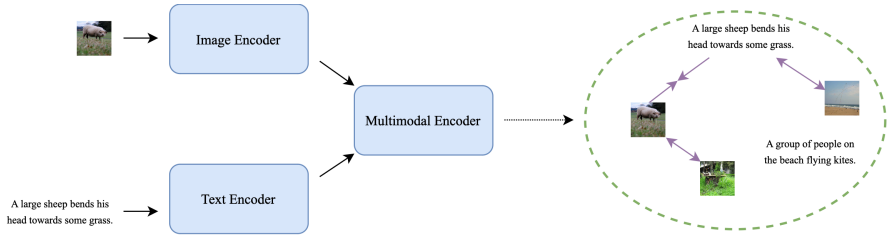

Figure 7: An abstract representation of a vision-language model. Image and text are first passed through unimodal, modality-specific, models (encoders), and then through a multimodal encoder that maps the modality-specific representations into a common representation space. A contrastive loss ensures the alignment and repulsion of similar and dissimilar concepts, respectively. We indicate this through purple arrows.

1.5 Vision-Language Contrast

Introduced as a method for self-supervised learning of image models ((TODO: cite contrastive learning section)), contrastive learning can be extended from unimodal (image) to multimodal applications, such as image and text. As mentioned in the previous section, we aim to maximize the cosine similarity between an image and its corresponding text (i.e., caption), and vice versa. Augmentation is not needed, as we always have pairs: one image and one text. Negative samples for images are captions of other images, and vice versa. In this setting, the model learns to produce similar representations for an image and its caption, describing the same real-world concept, and dissimilar representations for an image and caption that are unrelated. A conceptual example for both vision and vision-language contrastive learning can be seen in Figure 8.

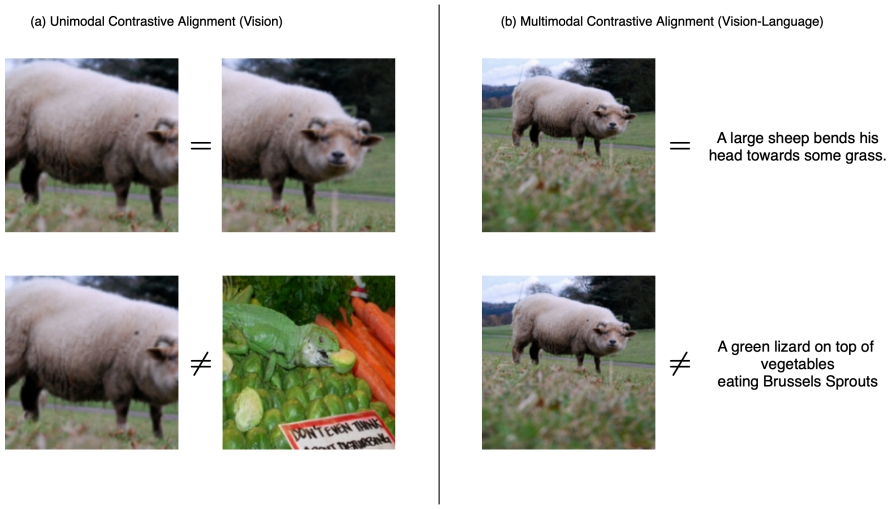

Figure 8: Contrastive learning aims to align the same (or similar) real-world concepts in representation space, while pushing different concepts apart. Multimodal contrastive learning (b) requires existing pairs, e.g. image-text, while for the unimodal case (a) pairs are synthetically created by augmenting the input. Images and text in the figure have been taken from the COCO train set [Lin+14].

Contrastive learning requires a (global) representation of the input, which is then used to compare it with other inputs. Since the introduction of the vision Transformer in 2020 by Dosovitskiy et al. [Dos+21], most vision-language models are exclusively based on the Transformer architecture, which is why the [CLS] token is used as the global representation for both image ([I_CLS]) and text ([T_CLS]), respectively. There have been other approaches, such as Cross-Modal Late Interaction introduced in FLILP [Yao+21], but they usually require significantly more compute [Yao+21] and do not outperform global contrastive learning [Wan+23], which is what we use here.

The representations are generated by passing the image sequence $\boldsymbol{H}_{v,0}$ and text sequence $\boldsymbol{H}_{w,0}$ through the vision-language model f, and extracting the representations for both tokens ($\boldsymbol{h}_{v,L,[\texttt{I\_CLS}]}$ and $\boldsymbol{h}_{w,L,[\texttt{T\_CLS}]}$) from the output of the final layer $\boldsymbol{H}_{v,L}$ and $\boldsymbol{H}_{w,L}$, which is the output of the Transformer. For the resulting batch of image and text representations $\left\{\boldsymbol{h}_{(v,L,[\texttt{I\_CLS}]),k}, \boldsymbol{h}_{(w,L,[\texttt{T\_CLS}]),k}\right\}_{k=1}^{B}$, where B is the batch size, the cosine similarity between all possible image-text pairs is computed. The cosine similarity is given by:

$$\cos(\boldsymbol{a}, \boldsymbol{b}) = \frac{\boldsymbol{a}\boldsymbol{b}^T}{\|\boldsymbol{a}\|_2 * \|\boldsymbol{b}\|_2} = \frac{\boldsymbol{a}}{\|\boldsymbol{a}\|_2}\frac{\boldsymbol{b}^T}{\|\boldsymbol{b}\|_2} \tag{20}$$

$\boldsymbol{a}\boldsymbol{b}^T$ denotes the simple dot product between both representations. $\|\boldsymbol{a}\|_2$ and $\|\boldsymbol{b}\|_2$ denote the L2-norm of the representations.

The cosine similarity between all possible image-text pairs can be computed efficiently by organizing all image and text representations in a matrix, which is already given in a batch-wise training, and normalizing every representation.

$$\boldsymbol{h}' = \frac{\boldsymbol{h}}{\|\boldsymbol{h}\|_2} \tag{21}$$

$$\boldsymbol{I} = \left[\boldsymbol{h}'_{(v,L,[\texttt{I\_CLS}]),1}, \boldsymbol{h}'_{(v,L,[\texttt{I\_CLS}]),2}, ..., \boldsymbol{h}'_{(v,L,[\texttt{I\_CLS}]),B}\right] \in \mathbb{R}^{B\times D} \tag{22}$$

$$\boldsymbol{T} = \left[\boldsymbol{h}'_{(w,L,[\texttt{T\_CLS}]),1}, \boldsymbol{h}'_{(w,L,[\texttt{T\_CLS}]),2}, ..., \boldsymbol{h}'_{(w,L,[\texttt{T\_CLS}]),B}\right] \in \mathbb{R}^{B\times D} \tag{23}$$

I denotes the batch/matrix of image representations, and T contains the text representations. D is the dimensionality of the representations, often referred to as the hidden size or hidden dimension in Transformers.

A matrix multiplication of both batches of representations then computes the dot product between every image with every text, and vice versa. Since the representations are normalized, the result will be the cosine similarity between all possible image-text pairs in the batch.

$$\boldsymbol{L} = \boldsymbol{I}\boldsymbol{T}^T, \boldsymbol{L} \in \mathbb{R}^{B\times B} \tag{24}$$

$\boldsymbol{L}_{i,j}$ then denotes the similarity between image i and text j in the batch. The diagonal of the matrix contains the similarity between positive pairs, i.e., the correct image-text pairs (i, i), with $\boldsymbol{L}_{i,i}$ describing their similarity. For an image, all other texts in the batch are considered as negative samples, and vice versa for text. The superscript T denotes the transpose of a matrix, and is not to be confused with the batch of text representations $\boldsymbol{T}$.

For a batch size of 256 ($B = 256$), each image has 255 negative samples (i.e., captions of other images) and one positive sample (i.e., its own caption), the same holds vice versa. This can be seen as a classification problem with 256 classes, where the model has to predict the correct class out of 256 classes, and each class representing one caption or image, respectively. For an image, the logit for the correct class is the similarity (cosine) to its own caption, and the logits for the negative classes are the similarities to the captions of other images. The same holds vice versa for text.

To calculate the loss, the cross-entropy loss is used. For a batch, the loss for selecting the correct caption for each image is given by:

$$\mathcal{L}_{\text{CL}}^{\text{i2t}} = \frac{1}{B}\sum_{i=1}^{B} -\log \frac{\exp(\boldsymbol{L}_{i,i})}{\sum_{k=1}^{B}\exp(\boldsymbol{L}_{i,k})} \tag{25}$$

$\frac{\exp(\boldsymbol{L}_{i,i})}{\sum_{k=1}^{B}\exp(\boldsymbol{L}_{i,k})}$ denotes the softmax-normalized similarity between an image and its correct caption, which is the usual way for calculating the cross-entropy. The result of this normalization is a probability distribution for each image, where each caption in the batch has a probability of being the correct caption for the image, and vice versa. The probability that the correct caption belongs to the current image is then used to calculate the negative log-likelihood, which is the loss.

Accordingly, the loss for selecting the correct image for each caption is given by:

$$\mathcal{L}_{\text{CL}}^{\text{t2i}} = \frac{1}{B}\sum_{i=1}^{B} -\log \frac{\exp(\boldsymbol{L}_{i,i})}{\sum_{k=1}^{B}\exp(\boldsymbol{L}_{k,i})} \tag{26}$$

Here, the softmax-normalization is with respect to the similarity of a text with all other images in the batch. The final loss is the mean of the image-to-text and text-to-image loss:

$$\mathcal{L}_{\text{CL}} = \frac{1}{2} * \left(\mathcal{L}_{\text{CL}}^{\text{i2t}} + \mathcal{L}_{\text{CL}}^{\text{t2i}}\right) \tag{27}$$

Returning to the concept of contrastive learning, this process ensures that the similarity between the representation of an image and its caption is maximized, i.e. close to each other, while the similarity between an image and an unrelated caption is minimized, i.e. far apart. Only this would appropriately minimize the loss, and thus the model learns to align the representations of the same concept across modalities. An illustration of multimodal contrastive learning can be found in Figure 9.

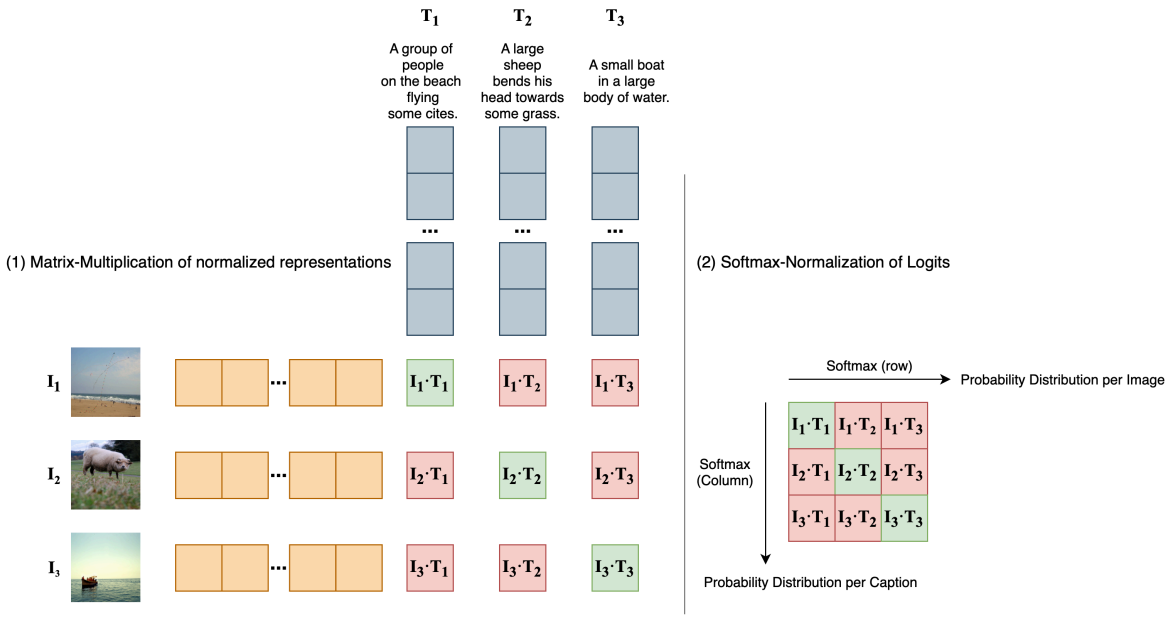

Figure 9: Contrastive Learning is performed using matrix multiplication of normalized representations (1), and the result is matrix $\boldsymbol{L}$ described in Equation 24. The representations are given by the [CLS] token of the respective modality, but are represented as I and T in the figure for simplicity. The diagonal of the resulting matrix contains the cosine similarity between positive samples. The softmax operation along the rows yields a probabilty distribution for each image over all captions, and the softmax operation along the columns vice versa (2). The cross-entropy loss is then used to calculate the loss for the distributions. The final loss is the mean of both losses. Image-Text pairs in the figure have been taken from the COCO train set [Lin+14].

The performance of contrastive learning is highly dependent on the number of negative samples available, which directly translates to the batch size. For instance, with a batch size of two, the model only needs to differentiate between one caption that belongs to the image and one that does not (a negative sample), and vice versa. This task is significantly simpler than with 255 negative samples or more, where there might be captions that are semantically similar to the image, but do not belong to it. So with increased negative samples, to probability of encountering hard-negative examples increases, forcing the model to aggregate as much information as possible in [I_CLS] and [T_CLS] to even differentiate between semantically similar concepts.

The results improve with an increased number of negative examples [He+19, Wan+23], which we will also show later, in the experiments section. More negative samples are usually achieved by using larger batch sizes [He+19, Rad+21, Wan+23]. However, this typically requires higher VRAM GPUs, or multiple GPUs, which is costly.

1.6 Image-Text Retrieval

The goal of image-text retrieval (ITR) is to find the matching caption for a given image in a set of captions, and likewise, finding the matching image for a given caption in a set of images. The process begins with embedding and normalizing a set of samples, which become a set of keys. For some normalized candidate representation, called the query, the most similar key is retrieved among the set of keys is the retrieved sample. This is exactly what is learned through contrastive learning, where we try to maximize the similarity between an image or caption (query) and its paired caption or image among other samples (keys), respectively. For that, we can use the same batch-wise computation introduced in the previous section about the contrastive loss. The similarity is computed by the cosine similarity, which is, again, computed by matrix multiplication of the normalized embeddings.

Image-Text retrieval can be viewed as a form of semantic search, which has significant practical relevance in areas like recommendation systems, e.g. to find fitting images based on a given text query. This is precisely what is learned through multimodal contrastive learning.

Image-Text retrieval is a simple and efficient way to benchmark the quality of the learned representations of a vision-language model, as it does not require any finetuning, just the embeddings produced by the model. The metric used for benchmarking is Rank@K (R@K), where K determines at which rank the paired/correct sample has to be in the ranking of keys, in order for the retrieval to be considered correct. We use R@1, R@5, and R@10, where R@1 is the normal accuracy, i.e., the paired sample has to be the most similar one. R@5 means that the paired sample has to be in the top 5 most similar samples, and for R@10, it has to be in the top 10 most similar samples.

In this thesis, we use the 5K test set of MSCOCO [Lin+14], and the 1K test set of Flickr30k [You+14] for benchmarking, which are the standard benchmarking dataset for multi-

modal models like FLAVA [Sin+21], CLIP [Rad+21], VLMo [Bao+22], and BEiT-3 [Wan+23]. MSCOCO contains 5K images with 5 captions for each image [Lin+14], and Flickr30k contains 1K images with 5 captions each [You+14]. For both datasets, all images and all texts are embedded and normalized, so that each image and each text is represented by the respective [CLS] token returned by the model. Then, matrix multiplication between all images and all captions of a dataset is performed, resulting in a matrix of shape (N, M), where N is the number of images and M is the number of captions in the dataset. So for MSCOCO, the matrix is of shape (5K, 25K), and for Flickr30k, the matrix is of shape (1K, 5K).

For each image, R@1, R@5, and R@10 are computed. The mean of R@1, R@5, and R@10 over all images are then called text-retrieval of the respective metrics (e.g. R@1-text-retrieval). We call this text-retrieval, because we are trying to retrieve the correct caption for a given image. The same is done for each caption, resulting in image-retrieval of the respective metrics (e.g. R@1-image-retrieval). For each dataset, we have 6 metrics in total: R@1, R@5, and R@10 for text-retrieval and image-retrieval, respectively. We will report the results of image-text retrieval in the format seen in Table 1.

Model	**MSCOCO (5K test set)**						**Flickr30K (1K test set)**					
	Image → Text			Text → Image			Image → Text			Text → Image		
	R@1	R@5	R@10	R@1	R@5	R@10	R@1	R@5	R@10	R@1	R@5	R@10
FLAVA [Sin+21]	42.74	76.76	-	38.38	67.47	-	67.7	94.0	-	65.22	89.38	-
CLIP [Rad+21]	58.4	81.5	88.1	37.8	62.4	72.2	88.0	98.7	99.4	68.7	90.6	95.2
BEiT-3 [Wan+23]	84.8	96.5	98.3	67.2	87.7	92.8	98.0	100.0	100.0	90.3	98.7	99.5

Table 1: Benchmarks of different vision-language models on the MSCOCO and Flickr30K datasets for image-text retrieval.

1.7 Related Work

1.7.1 Deep Aligned Representations

The motivation for the knowledge distillation driven approach in this work is provided by the paper "See, Hear, and Read: Deep Aligned Representations" by Aytar et al. (2017) [AVT17]. For simplicity, this paper will be referred to as "SHRe" (for **S**ee, **H**ear, **Re**ad) in this work.

In SHRe, the authors propose a method to align representations of image, text, and audio through knowledge distillation from a supervised image model. The student model is a multimodal model with separate modality-specific encoders for image, text, and audio, with a shared encoder on top. The approach utilizes 1D convolutions for audio and text, and 2D convolutions for images. The output feature maps of these encoders are flattened and then passed separately through a shared encoder, consisting of 3 linear layers [AVT17]. The approach is generally independent of the specific architecture of the components (encoders),

meaning that any architecture can be used. Notice how the aforementioned exactly matches the definition of a multimodal model as defined in (TODO: cite multimodal_models).

The teacher model was trained in a supervised manner, with the authors utilizing a model pretrained on ImageNet-1K, though it is not specified what exact model was used. The training objective is to minimize the KL-Divergence between the teacher and student models.

Specifically, the method involves using image-text $\{\boldsymbol{x}^v, \boldsymbol{x}^w\}$ and image-audio $\{\boldsymbol{x}^v, \boldsymbol{x}^a\}$ pairs. $\boldsymbol{x}^v$ is a 2D image, $\boldsymbol{x}^w$ a sequence of text tokens, and $\boldsymbol{x}^a$ a spectrogram of audio. For each pair, the image $\boldsymbol{x}^v$ is passed through the teacher model $g(\cdot)$, producing a probability distribution over the ImageNet-1K classes (1000 classes), denoted as $g(\boldsymbol{x}^v)$. The same image $\boldsymbol{x}^v$ is also passed through the image encoder $f_v(\cdot)$ of the student model, followed by the shared encoder $s(\cdot)$, also resulting in a probability distribution over the ImageNet-1K classes, defined as $s(f_v(\boldsymbol{x}^v))$.

The other part of the pair, for example, the text $\boldsymbol{x}^w$ in an image-text pair, is passed through the text encoder $f_w(\cdot)$ of the student model, and then through the shared encoder $s(\cdot)$. Since the shared encoder is the same as the one used for the image, the output is, again, a probability distribution over the ImageNet-1K classes, represented as $s(f_w(\boldsymbol{x}^w))$.

The probability distribution generated by the teacher model for the image can be compared with the probability distribution produced by the student model for the same image, using KL-Divergence. This is the usual approach to knowledge distillation, defined in (TODO: cite knowledge distillation section). What makes the approach unique, however, is that the probability distribution of the teacher model for the image can be compared with the probability distribution of the student model for the text. For a single image-text pair, the loss is defined as:

$$\mathcal{L}_{\text{KD}}^{\text{vw}} = \frac{1}{2} * D_{\text{KL}}(g(\boldsymbol{x}^v) \parallel s(f_v(\boldsymbol{x}^v))) + \frac{1}{2} * D_{\text{KL}}(g(\boldsymbol{x}^v) \parallel s(f_w(\boldsymbol{x}^w))) \tag{28}$$

With D_{KL} being the KL-Divergence. The loss changes accordingly for image-audio pairs, where the probability distribution over audio is defined as $s(f_a(\boldsymbol{x}^a))$.

$$\mathcal{L}_{\text{KD}}^{\text{va}} = \frac{1}{2} * D_{\text{KL}}(g(\boldsymbol{x}^v) \parallel s(f_v(\boldsymbol{x}^v))) + \frac{1}{2} * D_{\text{KL}}(g(\boldsymbol{x}^v) \parallel s(f_a(\boldsymbol{x}^a))) \tag{29}$$

The goal of this approach is to make the probability distributions between teacher and student as similar as possible. Since an image and its corresponding text in an image-text pair describe the same real-world concept, the distribution of the teacher model for the image, over the ImageNet-1K classes, can directly be transferred to the caption of the image. That way, the model can learn to output the same probabilities over the ImageNet-1K classes for both the image and the text. This enables the alignment of modalities at the level of real-world objects. The same process can be applied to image-audio pairs, allowing the model to

align representations across multiple modalities. A visualization of this will be shown when we apply this approach in (TODO: Transformer SHRe section).

Even though all modalities share the same shared encoder $s(\cdot)$, the output of the intermediate layers in the shared encoder will still differ for each modality. This is because KL-Divergence only ensures alignment at the level of classes, which corresponds to the output layer (the last fully-connected layer of the shared encoder outputs the probability distribution over ImageNet-1K classes). The internal representations in $s(\cdot)$, meaning the first two layers, can still be different between the modalities of a pair. They can vary, as long as the resulting probability distribution of the last fully-connected/linear layer is the same as the teacher model's output.

However, the shared encoder is meant to have the same internal representation for e.g. an image and its caption/text: Since they describe the same concept, the activations in the shared encoder should be similar, which is, as described in (TODO: cite image-text contrast), crucial for tasks such as retrieval. To achieve this, the authors add a ranking loss to the training, which functions similarly to a contrastive loss. This ranking loss drives the representations of inputs from the same pair closer together, while pushing the representations of inputs from different pairs further apart. It is defined as:

$$\mathcal{L}_{\text{Rank}} = \sum_{i=1}^{B} \sum_{j \neq i} \max\{0, \Delta - \cos(\boldsymbol{x}_i^v, \boldsymbol{x}_i) + \cos(\boldsymbol{x}_i^v, \boldsymbol{x}_j)\} \tag{30}$$

Here, B represents the batch size, $\boldsymbol{x}_i^v$ is an image, and $\boldsymbol{x}_i$ is the corresponding text or audio, depending if an image-text or image-audio pair is used. j iterates over negative samples in the batch ($j \neq i$).

Different from contrastive loss, for a given input, e.g. an image, the ranking loss does not normalize the similarity scores of a positive pair (e.g. image-text) with respect to all other possible pairings (all other texts) for a sample (image) in the batch. The authors did not provide intuitions for the choice of the ranking loss over the contrastive loss, and we can only assume that since the paper was published in 2017 [AVT17], the contrastive loss was not as widely adapted as it is today.

The final loss is a combination of the KL-Divergence loss and the ranking loss:

$$\mathcal{L}_{\text{SHRe}} = \mathcal{L}_{\text{KD}} + \mathcal{L}_{\text{Rank}} \tag{31}$$

The authors evaluate SHRe on retrieval tasks, and the results (Table 2) show that SHRe performs significantly better than a random baseline. Interestingly, even though the model is only trained on image-text and image-audio pairs, the alignment also generalizes to text-audio pairs, and the model can retrieve text-audio pairs, albeit not as well as between the modalities it was trained on [AVT17]. This indicates that the image modality acts as an anchor between text and audio, enabling the model to align representations between modalities it was not explicitly trained on. The alignment between modalities becomes transitive.

Model	MSCOCO		Flickr (Custom)[2]		Unspecified[3]	
	Image ↓ Text	Text ↓ Image	Image ↓ Sound	Sound ↓ Image	Text ↓ Sound	Sound ↓ Text
Random	500	500	500	500	500	500
SHRe	5.8	6.0	47.5	47.8	135.0	140.5

Table 2: Retrieval results of SHRe on different datasets. Each dataset contains 5k sample pairs (e.g. image-text pairs) for evaluation, and is splitted into 5 chunks of 1k samples each. Retrieval is then performed on each chunk, and metric used is the median rank of the correct pair in the ranked list. The median rank is averaged over all chunks for each datasets, so the results seen describe the average median rank over all chunks for each dataset. The results are taken from the SHRe paper [AVT17].

The approach is illustrated in Figure 10. It is important to note that SHRe is only trained with image-text and image-audio pairs, and not, how it might seem from the figure, with image-text-audio triplets.

The SHRe approach is a crucial foundation for this work, as it demonstrates how the knowledge from a **supervised** unimodal (image) model can be *extracted* and *transferred* to a multimodal model.

[2]Datasets used consists of videos collected from Flickr, from which frames were extracted and used as images with the corresponding audio [AVT17].

[3]Data has been collected and annotated using Amazon Mechanical Turk [AVT17, SF08]. Where the data originates from is not specified in the paper.

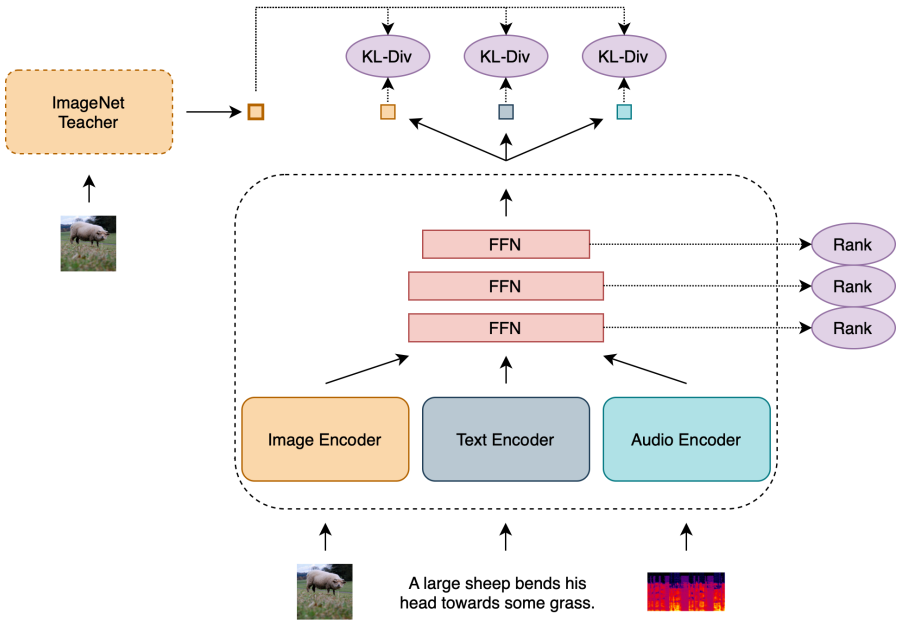

Figure 10: Illustration of the SHRe approach. The model is trained to output the same probability distribution over ImageNet-1K classes between images, image-text pairs, and image-audio pairs. Internal representations are aligned using a ranking loss [AVT17]. Image, text, and audio are always passed individually through the model. The output of the model, shown as colored squares, represent 1000-dimensional vectors, with each element representing the probability of the input belonging to a specific ImageNet-1K class. The figure does not originate from the original paper, but is a custom visualization of the concept. Image and text example is taken from the MSCOCO train set [Lin+14], the spectogram originates from the SHRe paper [AVT17].

1.7.2 CLIP

1.7.2.1 Method

CLIP is a method developed by OpenAI to train a vision-language model using contrastive learning. CLIP stands for (**C**ontrastive **L**anguage-**I**mage **P**retraining). The architecture consists of a separate image encoder $f(\cdot)$ and text encoder $g(\cdot)$, both of which can be any architecture, and a linear projection (linear layer without bias and activation function) on top of the modality-specific encoders.

The forward pass works as follows: For a batch of image-text pairs, the images $\left\{\boldsymbol{H}_{(v,0),i}\right\}_{i=1}^{B}$ (B denotes the batch size) are passed through the image encoder, resulting in an image representation $\boldsymbol{I} \in \mathbb{R}^{B \times D}$. Similarly, the texts $\left\{\boldsymbol{H}_{(w,0),i}\right\}_{i=1}^{B}$ are passed through the text encoder, producing a text representation T. Recall that $\boldsymbol{I}$ and $\boldsymbol{T}$ correspond to the batched representations of the [I_CLS] and [T_CLS] tokens, as defined in (TODO: cite equ for I) and (TODO: cite equ for T), respectively.

Both the image and text representations produced by the encoders are in separate embedding spaces — one for images and one for text - they are not related to each other initially. However, for contrastive learning to be effective, the embeddings should exist in the same latent space. After all, the embedding for an image and its corresponding text should be the same (or at least very close to each other).

In SHRE, discussed in the previous section, this shared latent space is achieved through a shared encoder on top of the modality-specific encoders, and through a ranking loss [AVT17]. CLIP maps the image and text representations into a shared latent space using linear projections $\boldsymbol{o}_v$ and $\boldsymbol{o}_w$ for image and text, respectively. These linear projections allow the model to map the image and text embeddings in a shared latent space, which is ensured by the contrastive loss. Note that the linear projections $\boldsymbol{o}_v$ and $\boldsymbol{o}_w$ can also be defined as functions, but for consistency with the original paper we use dot product notation, as shown in the following.

The image representation in the shared embedding space is denoted as $\boldsymbol{I}' = \|\boldsymbol{o}_v \boldsymbol{I}^T\|_2$, and the text representation as is given by $\boldsymbol{T}' = \|\boldsymbol{o}_w \boldsymbol{T}^T\|_2$. Since cosine similarity is used as the similarity metric in the contrastive loss, the embeddings are normalized, which is indicated by the l2 norm $\|\cdot\|_2$ around the result of the linear projections. It is important to note that the superscript T denotes the transpose of a matrix, not the batch of text representations.

Then, it is sufficient to perform matrix multiplication of the normalized representations in order to compute the cosine similarity between each pair. The result is given by:

$$\boldsymbol{L} = \exp(t) * \boldsymbol{I}' \boldsymbol{T}'^T, \boldsymbol{L} \in \mathbb{R}^{B \times B} \tag{32}$$

The opteration is quite similar to the batched cosine similarity operation introduced in (TODO: cite vision-lang-contrast). However, it is notable that the cosine similarities $\boldsymbol{L}$ are scaled by $\exp(t)$, where t is a temperature parameter. This parameter is used to control the smoothness of the softmax function, and is a scalar applied element-wise to the cosine similarities, which should be a familiar concept from knowledge distillation (TODO: cite KD section).

In knowledge distillation, the temperature was introduced as a tunable hyperparameter [AVT17, Gou+21]. However, in CLIP it is a learnable parameter that is optimized during training, just like any other parameter in the model, eliminating the need for manual tuning. The temperature t is optimized in log-space, which is why the actual temperature by which logits are scaled, is given by $\exp(t)$ [Rad+21].

Although the authors did not provide a specific reason for the optimization in log-space, it is likely that this approach ensures that the temperature is always positive, since $\exp(t)$ always returns a positive value. Optimizing in log-space may also contribute to greater numerical stability (the logarithm grows at a low rate), resulting in less drastic changes in the temperature during optimization and thereby making training more stable.

In the matrix $\boldsymbol{L}$, the cosine similarity between image i and text j in the batch is denoted by $\boldsymbol{L}_{i,j}$, where the diagonal elements contain the similarity for positive pairs. To maximize the similarity between positive pairs (i, i), and minimize the similarity between negative pairs (i, j), with $i \neq j$, cross-entropy loss is used.

The loss for selecting the correct caption for each image and vice versa is exacly the same as given in (TODO: cite vision-lang-contrast i2t) and (TODO: cite vision-lang-contrast t2i), respectively. The final loss of CLIP is the vision-language contrastive loss, given in (TODO: cite vision-lang-contrast).

$$\mathcal{L}_{\text{CLIP}} = \mathcal{L}_{\text{CL}} = \frac{1}{2} * \left(\mathcal{L}_{\text{CL}}^{\text{i2t}} + \mathcal{L}_{\text{CL}}^{\text{t2i}}\right) \tag{33}$$

CLIP only relies on contrastive learning to train a vision-language model, and therefore requires a high batch size to achieve good results. The authors use a very large batch size of 32,768 [Rad+21]. An abstract illustration of the end-to-end training process of CLIP is shown in (TODO: cite figure) in the Appendix.

1.7.2.2 Zero-Shot Image Classification

What makes CLIP special is its method of zero-shot image classification using the trained model. This capability is achieved through prompt engineering on the text encoder. For each class in the dataset, where image classification is desired, the name of the class is injected into a prompt template. The prompt template follows a structure like this: “a photo of a {class name}.”.

CLIP uses 80 different prompts, so for each class in the dataset, 80 distinct prompts are generated (similar to the example shown above). These 80 prompts are passed through the text encoder and text projection, resulting in 80 different text embeddings for one class. These embeddings are then averaged and normalized, yielding a single embedding per class. This embedding captures the semantic meaning of the class name, which the model learned through contrastive pretraining.

To classify an image, the image is passed through the image encoder and image projection, resulting in an image embedding. The cosine similarity between this image embedding and all class embeddings is calculated. The class corresponding to the text embedding with the highest similarity to the image representation is predicted as the class for the image, as demonstrated in Figure 11.

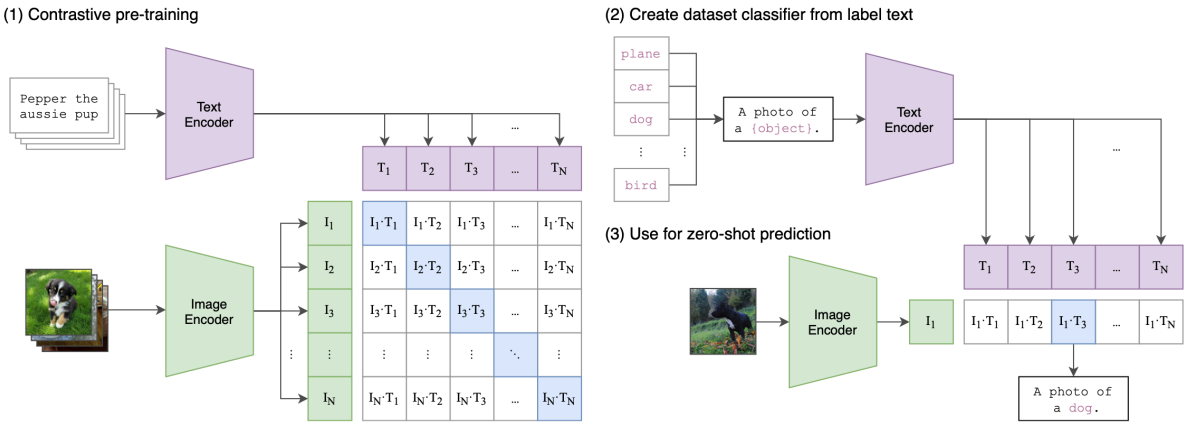

Figure 11: For zero-shot image classification, CLIP uses prompt engineering to create one classifier per image class to predict (2). The class whose classifier has the highest similarity (cosine) with the image representation is the predicted class (3) for the image [Rad+21].

The approach reaches a zero-shot accuracy of 76.2% on the validation set of ImageNet-1K [Rus+15], with a top-5 accuracy of 95% [Rad+21]. This is particularly impressive given that the model has never seen any images from the ImageNet-1K dataset during training, nor has it been trained on any image classification task. It merely achieves this accuracy through its cross-modal understanding between text and image. The model effectively "knows" how the ImageNet-1K classes look visually.

However, it is important to note that these results were based on a vision Transformer following the ViT-L/14@336px architecture for the image encoder. This architecture consists of 24 layers, 16 attention heads, a hidden size of 1024, and processes images at a resolution of 336x336 [Rad+21]. For the text encoder, a 12-layer Transformer was used, consisting of 12 attention heads and a hidden size of 768 [Rad+21]. According to HuggingFace, the model is 428 million parameters large[4]. Additionally, the model was trained on a custom dataset specifically developed for CLIP, consisting of 400 million image-text pairs [Rad+21].

1.8 Methodology

1.8.1 Tools

Software:

- for all implementations we use pytorch lightning
- provides high-level functionalities on top of pytorch
  - like checkpointing, logging, distributed training, etc.
- we do not need to implement them in pytorch manually (to some extend)

[4] https://huggingface.co/openai/clip-vit-large-patch14

  - pytorch already provides a high-level API but it is more prone to errors
- we save time and can focus on the actual implementation
- errors in vanilla pytorch are likely and hard to debug
- research will enevitably involve a lot of trial and error (experimentation)
- to keep track of all experiments, we use the experiment tracking tool Weights & Biases

Hardware:
- it is not possible to train the models, used in this work, on the CPU

-> GPUs are a requirement
- should be relatively new -> should be able to handle models upwards of 50 million parameters, but should not be too expensive
- we are severely limited by financial constraints, as there is not external funding for this work
- GPUs rented in the cloud
- we do not use popular cloud services like AWS or GCP -> too expensive
- instead, we use the smaller provider runpod.io
- has a high variety of consumer-grade, and enterprise-grade GPUs, much more affordable
- we opt for the (consumer-grade) NVIDIA RTX 4090
  - has one of the highest speeds (TODO: cite?) but lacks high VRAM (only 24GB)
    - is a problem we will address later
  - at the time of this research (June 2024), comes in at around 0.75 USD per hour
  - higher VRAM GPUs, like the A100, are available for 1.89 USD per hour
    - too expensive in the long run

1.8.2 Experimental Approach

- we will start as simple as possible
- always build on the results and knowledge of the previous steps
- to first validate if Knowledge-Distillation, the approach we will use throughout this work, even works for us, we will first test KD of unimodal models (e.g. distilling a ResNet-50 from a ResNet-101 on ImageNet), an area which has already been researched extensively
- from this, we will advance to the actual goal of this work: Multimodal Knowledge-Distillation
- as this is increasingly more difficult than distilling a unimodal model from another unimodal model of the same architecture, we will start with a supervised teacher
  - means, the teacher model has been trained on labeled data, and provides us with logits, and therefore a probabilty distribution, to regress
    - is basically a reproduction of SHRe [AVT17]
    - has been proven to work with this paper as a proof-of-concept
- if this approach works likewise for us, we will advance to a self-supervised teacher

- recall that goal was build a model/procedure for multimodal KD completly unreliant on labeled data
  - also means teacher, or any pretrained module that might be used, can't be trained on labeled data
  - goal of this work is to check if this is possible
  - as mentioned before, VLMo for example use a BEiT module pretrained on labeled data as part of their model
    - this is not end-to-end self-supervised

2 Experiments

2.1 Unimodal Knowledge Distillation

To validate whether traditional unimodal knowledge distillation, an undoubtedly simpler task than multimodal knowledge distillation, even works, we will first conduct experiments on unimodal knowledge distillation. We will then build on the results to develop a multimodal knowledge distillation.

2.1.1 Vision

2.1.1.1 Method

Our approach to vision KD involves using a pretrained Data2Vec2 [Bae+22] image model as the teacher model, and distilling a shallow version of this model, which is the student. We attribute our choice of Data2Vec2 to its effectiveness and consistency in self-supervised learning across image, text and audio. Data2Vec2 is a general framework to pretrained *unimodal* image, text, and audio models using self-supervised learning [Bae+22, Bae+22], which fits our philosophy of aligning modalities.

We approach the distillation by taking the first 6 Transformer blocks of the *pretrained* teacher model, which are exactly half of the 12 Transformer blocks in the teacher, and organize them into a smaller model. This smaller model also includes the pretrained cls token, patch projection, and positional encodings. Consequently, the student model is not trained from scratch, but already initialized with a subset of the teacher's weights. This is inspired by DistilBERT [San+19], a small BERT variant distilled from the normal BERT model, selecting every second layer from a pretrained BERT [Dev+19] model and organizing them into a smaller model. As mentioned before, we use the first 6 Transformer blocks of the teacher model, which we found leads to better results than using every second layer. The resulting student model is with 43.1M parameters almost half the size of the teacher model, which has 85.9M parameters.

Data2Vec2 is a self-supervised model [Bae+22], 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. 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 [Bae+22, Bae+22]. 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 $D_{\text{ff}} = 3072$, the number of layers in the student model ($L_s = 6$) 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 (or image patch, respectively) works as well as predicting the activations of each layer individually [Bae+22]. 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, which 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:

$$h'_{j,d} = \frac{h_{j,d} - \mu_d}{\sqrt{\sigma_d^2 + \varepsilon}}, \qquad \mu_k = \frac{1}{T}\sum_{t=1}^{T} h_{t,k}, \qquad \sigma_k^2 = \frac{1}{T}\sum_{t=1}^{T}\left(h_{t,k} - \mu_k\right)^2 \tag{34}$$

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 j is normalized by normalizing each dimension of the embedding independently, using the

corresponding mean and standard deviation computed for that dimension [UVL17]. 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 $\mathrm{InstanceNorm}(\cdot)$ as instance normalization on a sequence of embeddings $\boldsymbol{H}$.

$$\mathrm{InstanceNorm}(\boldsymbol{H}) = [\boldsymbol{h}'_1, \boldsymbol{h}'_2, ..., \boldsymbol{h}'_T] \tag{35}$$

After instance norm and averaging, parameter-less layer normalization is performed [Bae+22, Bae+22]. We perform all three operations likewise. The target and prediction are therefore given by:

$$\begin{aligned} \boldsymbol{H'}^{t}_{v,l} &= \mathrm{InstanceNorm}\Big(\boldsymbol{H}^{t}_{v,l}\Big), l \in \{1, 2, ..., L_t\} \\ \boldsymbol{H'}^{s}_{v,l} &= \mathrm{InstanceNorm}\Big(\boldsymbol{H}^{s}_{v,l}\Big), l \in \{1, 2, ..., L_s\} \end{aligned} \tag{36}$$

$$\begin{aligned} \widehat{\boldsymbol{H}}^{t}_{v} &= \frac{1}{L_t} \sum_{l=1}^{L_t} \boldsymbol{H'}^{t}_{v,l} \\ \widehat{\boldsymbol{H}}^{s}_{v} &= \frac{1}{L_s} \sum_{l=1}^{L_s} \boldsymbol{H'}^{s}_{v,l} \end{aligned} \tag{37}$$

$$\begin{aligned} \boldsymbol{Y} &= \Big[\boldsymbol{y}_{[\texttt{I\_CLS}]}, \boldsymbol{y}_1, ..., \boldsymbol{y}_N\Big] = \mathrm{LayerNorm}\Big(\widehat{\boldsymbol{H}}^{t}_{v}\Big) \\ \widehat{\boldsymbol{Y}} &= \Big[\hat{\boldsymbol{y}}_{[\texttt{I\_CLS}]}, \hat{\boldsymbol{y}}_1, ..., \hat{\boldsymbol{y}}_N\Big] = \mathrm{LayerNorm}\Big(\widehat{\boldsymbol{H}}^{s}_{v}\Big) \end{aligned} \tag{38}$$

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

$$\mathcal{L}_{\mathrm{KD}}\Big(\boldsymbol{Y}, \widehat{\boldsymbol{Y}}\Big) = \|\boldsymbol{Y} - \widehat{\boldsymbol{Y}}\|_2^2 = \frac{1}{N+1}\left(\sum_{n=1}^{N} \mathcal{L}_{\mathrm{MSE}}(\boldsymbol{y}_n, \hat{\boldsymbol{y}}_n) + \mathcal{L}_{\mathrm{MSE}}\Big(\boldsymbol{y}_{[\texttt{I\_CLS}]}, \hat{\boldsymbol{y}}_{[\texttt{I\_CLS}]}\Big)\right) \tag{39}$$

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 $\mathrm{LayerNorm}(\cdot)$, see (TODO: cite notation). $\mathcal{L}_{\mathrm{MSE}}(\cdot, \cdot)$ is the mean squared error between two vectors, defined in (TODO: cite equation).

2.1.1.2 Distillation

We distill the student model by minimizing the loss defined in Equation 39 using the AdamW optimizer [LH17] with a base learning rate of 5e-4. We train for 10 epochs with a batch size of 256 on the training set of ImageNet-1K [Rus+15], and run validation after every epoch on the validation set of ImageNet-1K. As Data2Vec2 our approach does not involve labels, we use the loss defined in Equation 39 also for validation. The total number of parameters

involved in the distillation process is 129M, of which 43.1M trainable belong to the student model, and 85.9M frozen parameters to the teacher model.

Since we use the same architecture as Data2Vec2 for our teacher model, the images, being of size 224×224, which is the size we will consistently use for all experiments, are split into 16x16 patches, which results in a sequence length of $N = 196$. A cls token is added to the sequence, which results in a total sequence length of $N + 1 = 197$. All embeddings have a dimension of $D = 768$.

For data augmentation we decide to use the same augmentation strategy using during the training of the teacher model. This ensures that we get the training targets from the same distribution the teacher has seen, and that we do not generate data for which the teacher might give inaccurate representations. The augmentations involve (1) cropping a random portion of an image and resizing it back to the original image resolution (224×224), (2) performing a random horizontal flip with probability 0.5, and (3) normalizing the image RGB channels with the mean and standard deviation of the ImageNet-1K dataset [Bae+22].

Detailed hyperparameters are provided in (TODO: cite hyperparameters).

We show the evolution of the training and validation loss during training in Figure 12. We observe the traditional convergence behavior of a model during training, and the validation loss is consistently lower than the training loss, which is a sign of good generalization. There are some peaks in the training loss, which are likely due to a high learning rate, but they do not affect the validation loss, which is why do not investigate them further. As we do not have access to any other metric than the MSE loss during training we have to evaluate the student by finetuning on downstream tasks, which follows in the next section. This will answer whether the distillation actually yields a model competetive in performance to the teacher model and if the knowledge transfer was successful.

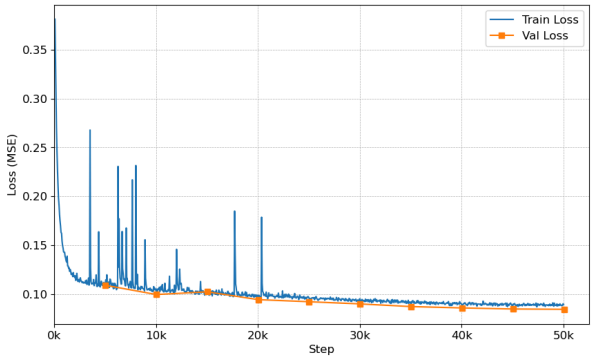

Figure 12: Training loss vs. validation loss during distillation of the Data2Vec2 image model.

2.1.1.3 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

[Kri09], CIFAR-100 [Kri09] and ImageNet-1K [Rus+15]. 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 [Bae+22, Bae+22] and BEiTv2 [Pen+22], 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 (cls token embedding is ingnored). For all three tasks we perform full finetuning, i.e. we finetune all layers of the student model on the downstream task, and linear probing, we only train the added layer norm and 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 [Cub+20], mixup [Zha+18] and cutmix [Yun+19] augmentation, and random erasing [Zho+20]. The hyperparameters for these augmentations are provided in (TODO: cite hyperparameters), and have been selected based on the values used in BEiTv2 [Pen+22], Data2Vec [Bae+22], and Data2Vec2 [Bae+22]. We refrain from explaining the augmentation techniques in detail here, as they are well documented in the respective papers.

For finetuning on 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 [Bae+22], 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 rate for a layer l is then calculated by:

$$\mathrm{lr}_l = \mathrm{base\_lr} * \mathrm{layer\_decay}^{L_s+1-l} \tag{40}$$

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 3: Learning rates for different blocks/layers when finetuning on ImageNet-1K. 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.

We show the learning rates for 8 layers in total, even though the student model only has 6 Transformer blocks. This is because we count all weights before the first Transformer block as layer 0, which includes the weights used for projecting patches to embeddings, the cls token, and the positional encodings. Correspondingly, layer 7 includes the weights for the

layer norm and linear classifier on top of the student model, which are initialized randomly and can be assigned a higher learning rate than the other layers.

For all hyperparameters used on the downstream tasks, see (TODO: cite hyperparameters).

The results, displayed in Table 4 and Table 5, show that while the student model is not able to outperform the teacher model (Data2Vec2), as well as all other models we compare to, it is able to achieve acceptable performance on all 6 evaluations considering both BEiTv2 and Data2Vec2 are based on the ViT-B/16 architecture [Bae+22, Pen+22], which is twice as large as the student model. We also compare to the ResNet-101 model from the original ResNet paper [He+16], which has 44.5M parameters, and is therefore comparable in size to the student model, but has been trained only supervised.

Method	Finetune	Linear Probe
Data2Vec2	84.5	-
BEiTv2	85.0	80.1
ResNet-101	80.1	-
DistilData2Vec2 (ours)	75.0	56.2

Table 4: Comparison of finetuning and linear probing results with SOTA self-supervised models on ImageNet-1K.

Dataset	Approach	Accuracy
CIFAR-10	Finetune	97.0
	Linear Probe	68.4
CIFAR-100	Finetune	85.1
	Linear Probe	46.2

Table 5: Results of finetuning and linear probing of our distilled Data2Vec2 image model on CIFAR-10 and CIFAR-100.

2.1.2 Language

2.1.2.1 Method

For knowledge distillation of a language model, we decide against the intuitive choice of distilling the corresponding Data2Vec2 language model, and opt for distilling a BERT model instead. We do this for two reasons. First, for reasons later explained later, we will also use BERT as the text encoder in our multimodal model, so for consistency we will use BERT for the unimodal distillation as well. Second, as mentioned before, there already exists a distilled version of BERT, DistilBERT [San+19], to which we can directly compare our results.

We use the same approach as for the image model, and distill a smaller version of BERT from a pretrained BERT model. Different to DistilBERT, we again take the first 6 Transformer

blocks of the teacher model, and organize them into a smaller model together with the embedding layer and positional encodings. The student model is therefore, again, initialized with a subset of the teacher's weights.

The distillation loss is defined analogously to the distilled image model, and is defined in Equation 39. We do not need to change anything, as the loss is applicable to any Transformer, regardless of the modality, making it a universal loss function for feature-based knowledge distillation. This follows Data2Vec2, which uses the same loss [Bae+22].

2.1.2.2 Distillation

The BERT model is distilled on a subset of the OpenWebText dataset [GC19], introduced in (TODO: cite data), of which the text is tokenized into subwords using the BERT tokenizer. During training, the tokenized text of the dataset is split into sequences of 196 tokens, which is the maximum sequence length that can fit on a single GPU with 24GB of memory (RTX 4090).

We validate the student model on the dedicated validation dataset of OWT we introduced in (TODO: cite data). The same loss as used as for training.

2.1.2.3 Finetuning

	MNLI	QNLI	RTE	MRPC	QQP	STS-B	CoLA	SST	**Score**
BERT	86.7	91.8	69.3	88.6	89.6	92.7	56.3	92.7	83.5
DistilBERT	82.2	89.2	59.9	87.5	88.5	86.9	51.3	91.3	79.6
DistilData2Vec2 (ours)	-	-	-	-	-	-	-	-	-

Table 6: Results for BERT and DistilBERT are taken from the DistilBERT paper [San+19].

A Appendix

AA Hyperparameters

Type	Hyperparameters	Values
Model	Layers	6
	Hidden size	768
	FFN inner hidden size	3072
	Attention Heads	12
	Patch size	16×16
	Input resolution	224×224
Training	Epochs	10
	Total steps	50040
	Batch size	256
	Optimizer	AdamW
	AdamW ε	1e-06
	AdamW β	(0.9,0.98)
	Weight decay	0.01
	Base learning rate	1e-4
	Learning rate schedule	Cosine
	Warmup steps	5004 (10% of total steps)
	Hardware	1 × RTX 4090 24GB
Augmentations	Horizontal flipping prob.	0.5
	RandomResizeCrop range	[0.08, 1.0]

Table 7: Hyperparameters used for distilling a Data2Vec2 image model.

Type	Hyperparameters	ImageNet		CIFAR10		CIFAR100	
		Finetune	Linear probe	Finetune	Linear probe	Finetune	Linear probe
Training	Epochs	15					
	Batch size	256					
	Optimizer	AdamW					
	AdamW ε	1e-8					
	AdamW β	(0.9, 0.999)					
	Weight decay	0.01					
	Base learning rate	1e-3					
	Layer Decay	0.81					
	Learning rate schedule	Cosine					
	Warmup steps	10% of total steps					
	Hardware	1 × RTX 4090 24GB					
Mixup [Zha+18]/**Cutmix** [Yun+19]	Mixup prob.	0.8					
	Cutmix prob.	1.0					
	Prob.	0.9					
	Switch prob.	0.5					
	Label smooting	0.1					
RandAugment [Cub+20]	Magintude	9					
	Magnitude std.	0.5					
	Magnitude inc.	1					
	# ops	2					
RandomErase [Zho+20]	Prob.	0.25					
	Mode	pixel					
	# erase	1					

Table 8: Hyperparameters used for the ImageNet-1K [Rus+15], CIFAR10 [Kri09], and CIFAR100 [Kri09] of the distilled Data2Vec2 image model. We refer to the respective papers for details on the augmentation techniques [Cub+20, Yun+19, Zha+18, Zho+20].

Type	Hyperparameters	MNLI	QNLI	RTE	MRPC	QQP	STS-B	CoLA	SST
Training	Epochs	15							
	Batch size	256							
	Optimizer	AdamW							
	AdamW ε	1e-8							
	AdamW β	(0.9, 0.999)							
	Weight decay	0.01							
	Base learning rate	1e-3							
	Layer Decay	0.81							
	Learning rate schedule	Cosine							
	Warmup steps	10% of total steps							
	Metric	Accuracy	Accuracy	Accuracy	F1	F1	Spearman	Accuracy	Accuracy
	Hardware	1 × RTX 4090 24GB							

Table 9: Hyperparameters for the GLUE [Wan+19] benchmark tasks of the distilled Data2Vec2 image model.

AB Pseudocode

```
# model: pretrained (e.g. distilled) model
# layer_norm: layer normalization layer
# cls_head: linear classifier -> nn.Linear(D, C)
# x: batch of images (B, 3, H, W)
def image_downstream_forward(model, layer_norm, cls_head, x, linear_probe):

    if linear_probe:
      with torch.no_grad():
        x = model(x) # (B, T, D)
    else:
      x = model(x) # (B, T, D)

    x = x[:, 1:] # remove cls token (B, T-1, D)
    x = x.mean(dim=1) # mean over all patches (B, D)
    x = layer_norm(x)
    x = cls_head(x) # (B, C)
    pred = x.argmax(dim=-1) # (B, )
    return pred
```

Listing 1: Pytorch pseudocode for the forward pass during finetuning or linear probing of a pretrained model on an image classification tasks. The output of the forward pass is the predicted class index for each image in the batch.

AC Figures and Visualizations

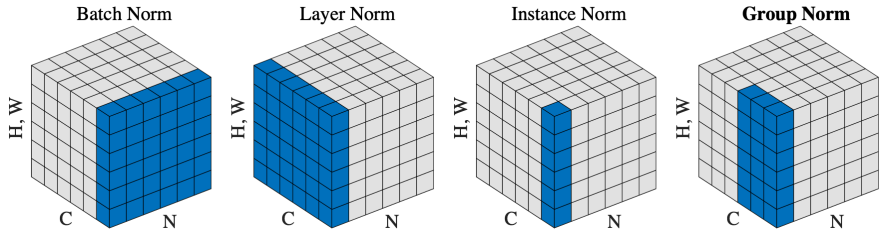

Figure 13: Comparison of different normalization operations on the example of images. Dimension “H, W” refers to the height and width of the input, “C” to the number of channels or embedding dimensions, and “N” to the number of samples, i.e. the batch dimension. Since we are working with sequences of embeddings, the height and width dimension correspond to the time steps (“H, W” -> “T”). The normalization operations work correspondingly on text sequences, where we also have time steps, so dimension “H, W” can also be replaced by “T” [WH18]. Please note that group norm, even though it is displayed in bold , is not used in this work. The figure merely was introduced in the paper of Group Norm [WH18].

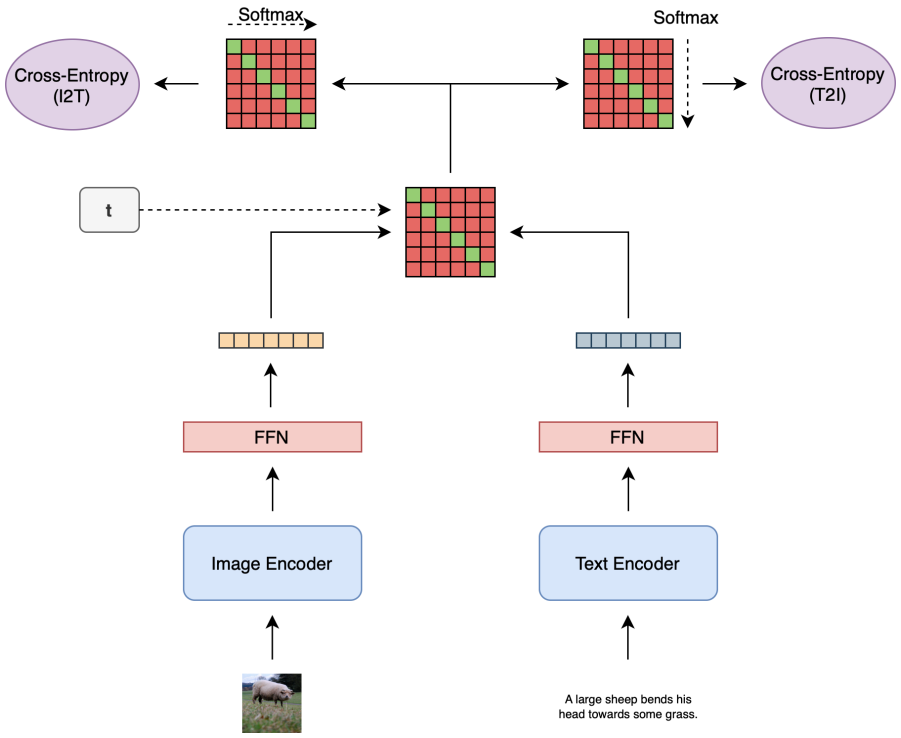

Figure 14: Illustration of CLIP training. A batch of image-text pairs is passed through the model and embedded into a shared latent space. The cosine similarity between all pairs is computed and softmax-normalized to calculate the image-to-text and text-to-image loss. The final loss is the mean of both losses [Rad+21]. The example is shown with a batch size of 6. The figure does not originate from the original paper, but is a custom visualization of the concept. Image-Text pair is taken from the MSCOCO train set [Lin+14], and do not refer to the contrastive loss of 6 pairs at the top of the figure. They are merely indicators of the intput to the model.

AD Technical Details

Bibliography

[Vas+17] A. Vaswani *et al.*, "Attention is all you need," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, in NIPS'17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 6000–6010.

[Xio+20] R. Xiong *et al.*, "On layer normalization in the transformer architecture," in *Proceedings of the 37th International Conference on Machine Learning*, in ICML'20. JMLR.org, 2020.

[Rus+15] 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.

[Gou+21] 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.

[Lin+14] 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.

[LM21] Y. LeCun and I. Misra, "Self-supervised learning: The dark matter of intelligence." [Online]. Available: https://ai.meta.com/blog/self-supervised-learning-the-dark-matter-of-intelligence/

[AVT17] Y. Aytar, C. Vondrick, and A. Torralba, "See, Hear, and Read: Deep Aligned Representations," *arXiv preprint arXiv:1706.00932*, 2017, [Online]. Available: https://arxiv.org/abs/1706.00932

[Rad+21] A. Radford *et al.*, "Learning transferable visual models from natural language supervision," in *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, M. Meila and T. Zhang, Eds., in Proceedings of Machine Learning Research, vol. 139. PMLR, 2021, pp. 8748–8763.

[WH18] Y. Wu and K. He, "Group Normalization," in *Computer Vision -– ECCV 2018: 15th European Conference, Munich, Germany, September 8-14, 2018, Pro-*

ceedings, Part XIII, Munich, Germany: Springer-Verlag, 2018, pp. 3–19. doi: 10.1007/978-3-030-01261-8_1.

[San+19] 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.

[Kri09] A. Krizhevsky, "Learning multiple layers of features from tiny images," 2009.

[Cub+20] 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.

[Yun+19] 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.

[Zha+18] 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

[Zho+20] 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.

[Wan+19] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman, "GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding," 2019.

[Dos+21] A. Dosovitskiy *et al.*, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale," *ICLR*, 2021.

[Bao+22] 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

[Yu+22] 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

[HVD15] G. Hinton, O. Vinyals, and J. Dean, "Distilling the Knowledge in a Neural Network." [Online]. Available: https://arxiv.org/abs/1503.02531

[Bae+22] A. Baevski, A. Babu, W.-N. Hsu, and M. Auli, "Efficient Self-supervised Learning with Contextualized Target Representations for Vision, Speech and Language." 2022.

[Bae+22] 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.

[Dev+19] 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.

[Yao+21] L. Yao *et al.*, "FILIP: Fine-grained Interactive Language-Image Pre-Training," *CoRR*, 2021.

[Wan+23] W. Wang *et al.*, "Image as a Foreign Language: BEIT Pretraining for Vision and Vision-Language Tasks," in *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023, pp. 19175–19186. doi: 10.1109/CVPR52729.2023.01838.

[He+19] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick, "Momentum Contrast for Unsupervised Visual Representation Learning," *arXiv preprint arXiv:1911.05722*, 2019.

[You+14] P. Young, A. Lai, M. Hodosh, and J. Hockenmaier, "From image descriptions to visual denotations: New similarity metrics for semantic inference over event descriptions," *Transactions of the Association for Computational Linguistics*, vol. 2, pp. 67–78, Feb. 2014.

[Sin+21] A. Singh *et al.*, "FLAVA: A foundational language and vision alignment model," *CoRR*, 2021, [Online]. Available: https://arxiv.org/abs/2112.04482

[SF08] A. Sorokin and D. Forsyth, "Utility data annotation with Amazon Mechanical Turk," in *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, 2008, pp. 1–8. doi: 10.1109/CVPRW.2008.4562953.

[UVL17] D. Ulyanov, A. Vedaldi, and V. Lempitsky, "Instance Normalization: The Missing Ingredient for Fast Stylization." [Online]. Available: https://arxiv.org/abs/1607.08022

[LH17] I. Loshchilov and F. Hutter, "Decoupled Weight Decay Regularization." 2017.

[Pen+22] Z. Peng, L. Dong, H. Bao, Q. Ye, and F. Wei, "BEiT v2: Masked Image Modeling with Vector-Quantized Visual Tokenizers," 2022.

[He+16] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, Jun. 2016. doi: 10.1109/cvpr.2016.90.

[GC19] A. Gokaslan and V. Cohen, "OpenWebText Corpus." 2019.