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

# Extracting Multimodal Knowledge from Pretrained Vision Models

Tim Cares

Master's thesis in Applied Computer Science

9. Oktober 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, 9. Oktober 2024 Tim Cares

# Acknowledgements

## Abstract

# Contents

# 1 Background

## 1.1 Basic Loss Functions

Throughout this work we will make use of various loss functions and extend them if necessary. To avoid redundancy, we will define the basic concepts of them here for easier reference.

We denote bold symbols (e.g. $\boldsymbol{v}$) as vectors, and $v_i$ as 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 Mean Squared Error

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:

$$\mathcal{L}_{\text{MSE}}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \|\boldsymbol{y} - \hat{\boldsymbol{y}}\|_2^2 = \frac{1}{D}\sum_{d=1}^{D}(y_d - \hat{y}_d)^2 \tag{1}$$

### 1.1.2 Kullback-Leibler 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 $\boldsymbol{q} \in \mathbb{R}^C$ with a target distribution $\boldsymbol{p} \in \mathbb{R}^C$. Since we are using the KL-Divergence in the context of classification tasks, which are discrete distributions over $C$ 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:

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

Equation 3 is defined analogously for $\boldsymbol{q}$.

### 1.1.3 Cross-Entropy Loss

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:

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

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 always one-hot encoded, meaning one class $i$ has a probability of 1, all others 0 (see Equation 5), the entropy of the target distribution $H(\boldsymbol{p})$ will remain constant and does not affect the minimization. Moreover, again 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 $i$.

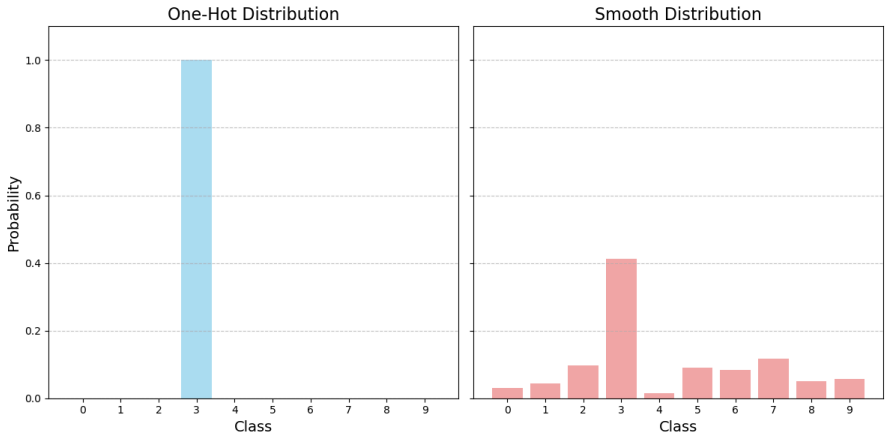


Figure 1: Comparison between different distributions over $C = 10$ classes. The one-hot distribution (left) is used for classification tasks with the cross-entropy loss. The KL-Divergence is used when predicting a smooth distribution (right). A smooth distribution usually results from a model prediction, and is a popular target distribution for knowledge distillation, introduced in Section 1.2.

Often times, the prediction $\boldsymbol{x}$ of a model 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:

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

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 $\boldsymbol{p}$ for cross-entropy and KL-Divergence is shown in the following Figure 1.

## 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]. Based on our approximation, it 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 layers are then used in a different model, they have to adapt to the new environment during finetuning. This, while the model will be smaller and more efficient, requires longer training times.

Another option is knowledge distillation (KD). Here, a smaller model is trained to replicate, or rather predict, the outputs of a larger model for a given sample. The larger model is called the teacher, and the smaller model the student. There are two strategies of KD usually used in practice: Response-based KD and feature-based KD [Gou+21], and we will make use of both 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 [Yu+22]. At a price of 1.449 USD per TPU hour (as of August 2024), the total cost is $1.449\ \mathrm{USD/h} * 24\ \mathrm{h/day} * 5\ \mathrm{days} * 2048\ \mathrm{TPUs} = 356,106.24\ \mathrm{USD}$.

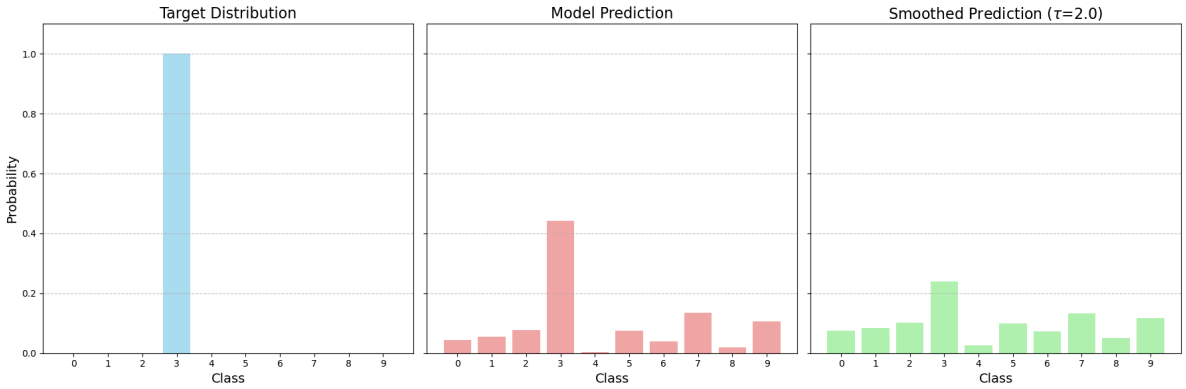


Figure 2: We present a similar figure as Figure 1. A temperature parameter $\tau$ further smoothens an already soft distribution. In response-based KD, this soft distribution is the prediction of a teacher model for a given samle (middle). Further smoothing (right) increases the relative importance of classes with lower scores. Especially in distributions with large number of classes some classes will have semantic similarities. Consider the classes "German Shorthaired Pointer" and "Labrador Retriever" of ImageNet-1K [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 loss function typically used in response-based KD is the KL-Divergence, introduced in Section 1.1.2, measuring 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. 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 ($\boldsymbol{u} \in \mathbb{R}^{1000} \wedge \boldsymbol{z} \in \mathbb{R}^{1000}$). A best practice is to divide the logits by a temperature parameter $\tau$, before applying the softmax function [Gou+21, HVD15], which smooothes the probability distribution further, as illustrated in Figure 2.

The temperature $\tau$ 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.

$$p_i = \frac{\exp\left(\frac{u_i}{\tau}\right)}{\sum_j \exp\left(\frac{u_j}{\tau}\right)} \tag{8}$$

$$q_i = \frac{\exp\left(\frac{z_i}{\tau}\right)}{\sum_j \exp\left(\frac{z_j}{\tau}\right)} \tag{9}$$

$i$ 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.

Consequently, recalling the definition of the KL-Divergence in Equation 2, the training objective of response-based knowledge distillation is to bring the probability distributions of the student model for a given sample, or rather for all sample in the training set, as close as possible to the probability distributions of the teacher model for the respective 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, and therefore 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 Section 1.1.1). 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 Section 3.1. An illustration of response-based vs. feature-based KD is shown in Figure 3.

## 1.3 Transformer

While we previously introduced concepts to train smaller models in a cost-efficient way, we now introduce the architecture of the models used in this work. We will exclusively make use of the Transformer architecture [Vas+17], which has been shown to be highly effective across vision [Bao+22] and language [Dev+19]. Consequtively, we will define the interpretation of image and text in the context of the Transformer.

### 1.3.1 Language Transformer

#### 1.3.1.1 Text Representation

In Transformers, text is represented as a sequence of discrete tokens, which are, as described in word2vec [Mik+13], embedded into D-dimensional vectors using an embedding matrix. A single token $i$, being a (sub)word like "hell", "hello", "a", is represented as $\boldsymbol{e}_i^w \in \mathbb{R}^D$, which is

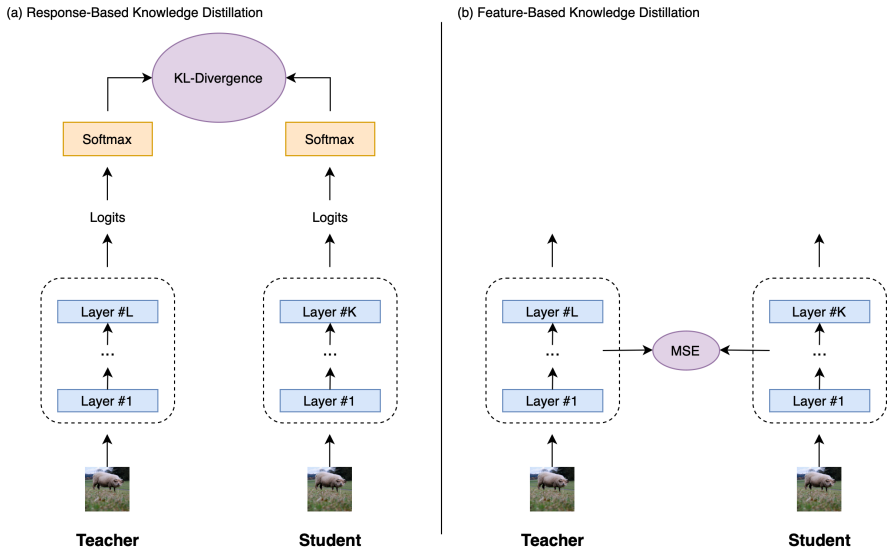


Figure 3: Response-based knowledge distillation (a) requires a teacher to provide logits to generate a probability distribution, which is predicted by the student. Feature-based knowledge distillation (b) is used for predicting the actual activations of the teacher's layer(s). In both cases the weights of the teacher are frozen and the teacher is running in evaluation/ inference mode. It is important to note that in most cases the number of layers between teacher and student differs ($L \neq K$), making the direct regression of the teacher's activations non-trivial. Figure adapted and inspired by [Gou+21], image is taken from COCO train set [Lin+14].

the embedding of the token resulting from the embedding matrix. A text or sentence is represented as a sequence of $M$ token embeddings/represenations $\boldsymbol{E}_w = [\boldsymbol{e}_1^w, \boldsymbol{e}_2^w, ..., \boldsymbol{e}_M^w] \in \mathbb{R}^{M \times D}$.

In addition, each sequence is prepended with a $[\texttt{T\_CLS}] \in \mathbb{R}^D$ token, and appended with an $[\texttt{T\_SEP}] \in \mathbb{R}^D$ token, often referred to as the cls and end-of-sequence token, respectively. As with all word embeddings, they are learnable and part of the embedding matrix. The purpose of the $[\texttt{T\_CLS}]$ token is to aggregate the global information/content of the text, while the $[\texttt{T\_SEP}]$ token is used to indicate the end of the text sequence. The resulting text representation is:

$$\boldsymbol{E}_w = \left[\boldsymbol{e}_{[\texttt{T\_CLS}]}^w, \boldsymbol{e}_1^w, \boldsymbol{e}_2^w, ..., \boldsymbol{e}_M^w, \boldsymbol{e}_{[\texttt{T\_SEP}]}^w\right] \in \mathbb{R}^{(M+2) \times D} \tag{10}$$

The sequence length $M$ is not fixed, but is usuall set to a specific value when training a Transformer model, a popular choice being $M + 2 = 512$.

As will be explained later, unlike RNNs, Transformers do not process a sequence step by step, but all at once. As a result, the Transformer does not have any sense of order in the sequence. It is therefore necessary to add a so called positional encoding to the text embeddings, giving the Transformer a sense of order in the text sequence. This positional encoding

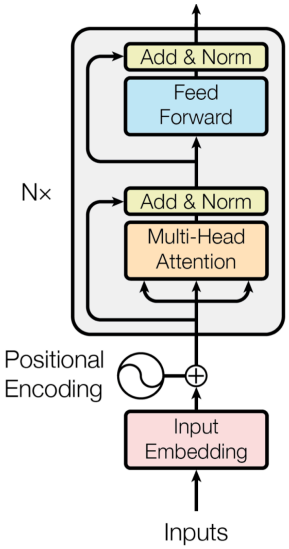


Figure 4: The architecture of the Transformer encoder. The original architecture also includes a Transformer decoder, which is not relevant for our work and therefore omitted. The encoder consists of $N$ (or $L$) Transformer layers. The output of a layer is the input to the subsequent layer. Figure is adjusted from the original Transformer paper [Vas+17].

$\boldsymbol{T}^{\text{pos}}$ is a sequence of $D$-dimensional vectors with the same length as $\boldsymbol{E}_w$, and can therefore simply be added to the text embeddings. The positional encoding, i.e. the embedding of each time step, is either learned or fixed, and we refer to the original Attention is all you need paper [Vas+17] for more details.

$$\boldsymbol{T}_w^{\text{pos}} = \left[\boldsymbol{t}^w_{\text{pos}_{[\text{T\_CLS}]}}, \boldsymbol{t}^w_{\text{pos}_1}, \boldsymbol{t}^w_{\text{pos}_2}, ..., \boldsymbol{t}^w_{\text{pos}_M}, \boldsymbol{t}^w_{\text{pos}_{[\text{T\_SEP}]}}\right] \tag{11}$$

The final text representation is defined as:

$$\boldsymbol{H}_{w,0} = \left[\boldsymbol{h}_{w,0,[\text{T\_CLS}]}, \boldsymbol{h}_{w,0,1}, ..., \boldsymbol{h}_{w,0,M}, \boldsymbol{h}_{w,0,[\text{T\_SEP}]}\right] = \boldsymbol{E}_w + \boldsymbol{T}_w^{\text{pos}} \tag{12}$$

We refer to $0$ as the layer number of the Transformer, which we will clarify in the next section, and $w$ as the modality being text or language, respectively.

#### 1.3.1.2 Transformer Layer

The actual Transformer consists of a set of $L$ layers, also referred to as blocks, which are stacked on top of each other. The input to a Transformer layer $l$ is a sequence of token embeddings $\boldsymbol{H}_{w,l-1}$ produced by the previous layer, and the input to the first layer $l = 1$ is therefore the text representation $\boldsymbol{H}_{w,0}$ as defined in Equation 12. Correspondingly, the output of a Transformer layer $l$ is denoted as $\boldsymbol{H}_{w,l}$, and the length of the sequence does not change across layers. The architecture of a Transformer is displayed in Figure 4.

Independent of modality, we define the operations performed by one Transformer layer as follows:

$$\begin{aligned} \boldsymbol{H}'_l &= \mathrm{LN}(\mathrm{MHA}(\boldsymbol{H}_{l-1}) + \boldsymbol{H}_{l-1}) \\ \boldsymbol{H}_l &= \mathrm{LN}(\mathrm{FFN}(\boldsymbol{H}'_l) + \boldsymbol{H}'_l) \end{aligned} \tag{13}$$

$\mathrm{LN}(\cdot)$ denotes layer normalization, or short LayerNorm. as defined in [BKH16]. Simply put, it normalizes each embedding (or time step respectively) of a sequence individually, so that the mean is zero and the variance is one.

$$h'_j = \frac{h_j - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}, \qquad \mu_k = \frac{1}{D}\sum_{d=1}^{D} h_d, \qquad \sigma_k^2 = \frac{1}{D}\sum_{d=1}^{D} (h_d - \mu_k)^2 \tag{14}$$

$\varepsilon$ is a small constant to avoid division by zero, and $D$ is the embedding dimension of the tokens. The operation is applied to each time step $\boldsymbol{h}$ individually, and its relation to other normalization techniques, like BatchNorm, is shown in Figure 35.

The addition between two sequences of embeddings is a residual connection originally introduced in ResNets [He+16], and adds each time step of one sequence to the corresponding time step of the other sequence, also seen in Equation 12 when adding the positional encoding to the text embeddings.

$\mathrm{FFN}(\cdot)$ denotes a timestep-wise application of a feed-forward network. In essence, it is a 2-layer MLP with Layer Norm and GELU activation, where the first layer usually consists of $D_{\text{ff}}$ neurons, and the second layer of $D$ neurons, with $D$ being the embedding dimension of the tokens. A popular choice for the intermediate dimension $D_{\text{ff}}$ is $D_{\text{ff}} = 4 \times D$. The operation for a single time step $\boldsymbol{h}$, or embedding respectively, is defined as:

$$\mathrm{FFN}(\boldsymbol{h}) = \mathrm{LN}(\mathrm{GELU}(\boldsymbol{h}\boldsymbol{W}_1 + \boldsymbol{b}_1))\boldsymbol{W}_2 + \boldsymbol{b}_2 \tag{15}$$

For a sequence of embeddings $\boldsymbol{H}$, the operation is applied to each time step $\boldsymbol{h}$ individually, which includes the [T_CLS] and [T_SEP] token, though not displayed in Equation 16. Since the output of $\mathrm{FFN}(\cdot)$ is also in $\mathbb{R}^D$, the embedding dimension of the tokens, referred to as *hidden size* or *hidden dim* of a Transformer, does not change across layers.

$$\mathrm{FFN}(\boldsymbol{H}) = [\mathrm{FFN}(\boldsymbol{h}_1), \mathrm{FFN}(\boldsymbol{h}_2), ..., \mathrm{FFN}(\boldsymbol{h}_M)] \tag{16}$$

For the definition of the Multi-Head Attention $\mathrm{MHA}(\cdot)$, performing Self-Attention, we refer to the original Transformer paper [Vas+17] for a detailed explanation.

The output of the last layer $L$ is the final text representation $\boldsymbol{H}_{w,L}$. To use this representation for downstream tasks like classification, the sequence of embeddings has do be reduced to a single representation of the input. This is done by taking the representation of the [T_CLS] token, which is then passed to a subsequent classification head, represented by a single linear layer:

$$\hat{\boldsymbol{y}}_w = \boldsymbol{h}_{w,L,\texttt{[T\_CLS]}}\boldsymbol{W}_{\texttt{[CLS]}} + \boldsymbol{b}_{\texttt{[CLS]}} \in \mathbb{R}^C \tag{17}$$

### 1.3.2 Vision Transformer

After the success of the Transformer architecture in NLP, breaking various benchmarks with architectures like BERT [Dev+19], leading to its widespread adoption especially in Large Language Models (LLMs) like GPT-2 [Rad+19], researchers explored the potential of this architecture beyond text. This exploration led to the development of the Vision Transformer (ViT) [Dos+21], marking a significant shift in computer vision.

Diffent from traditional Convolutional Neural Networks (CNNs), which have been the dominant architecture in computer vision before the ViT, the ViT processes images as 1D sequences of patches, instead of 2D grids of pixels.

We define an image as a 3-dimensional tensor $\boldsymbol{v} \in \mathbb{R}^{C \times H \times W}$. Throughout this work, we will exclusively use an image size of $224 \times 224$ pixels and 3 color channels, so $\boldsymbol{v} \in \mathbb{R}^{3 \times 224 \times 224}$. Before being passed to the Transformer layers, the image first needs to be converted into a sequence of embeddings, similar to text. This is done by first dividing the image into $14 \times 14$ patches, each being a square of size $16 \times 16$ pixels. Each patch $i$ is then flattened into a 256-dimensional vector, and projected into a 768-dimensional embedding $\boldsymbol{e}_i^v \in \mathbb{R}^{768}$ using a fully connected layer. $v$ denotes the image modality.

Similar to text, the resulting sequence of patches is prepended with a special learnable $[\texttt{I\_CLS}] \in \mathbb{R}^{768}$ token, which is used to aggregate the global information/content of the image. A $[\texttt{I\_SEP}]$ does not exist, as images can not be intuitively concatenated like multiple sentences of text. The image representation is defined as:

$$\boldsymbol{E}_v = \left[\boldsymbol{e}^v_{[\texttt{I\_CLS}]}, \boldsymbol{e}^v_1, \boldsymbol{e}^v_2, ..., \boldsymbol{e}^v_N\right] \tag{18}$$

To again give the Transformer a sense of order in the image patches, a unique positional encoding is added to each patch embedding, which is either learned or fixed and also represented as a sequence of 768-dimensional vectors:

$$\boldsymbol{T}^{\text{pos}}_v = \left[0, \boldsymbol{t}^v_{\text{pos}_1}, \boldsymbol{t}^v_{\text{pos}_2}, ..., \boldsymbol{t}^v_{\text{pos}_N}\right] \tag{19}$$

Notice that the positional encoding for the $[\texttt{I\_CLS}]$ token is set to zero. This is because the $[\texttt{I\_CLS}]$ token is not actually part of the image, and can be seen as a type of meta token. The input to a ViT is defined as:

$$\boldsymbol{H}_{v,0} = \left[\boldsymbol{h}_{v,0,[\texttt{I\_CLS}]}, \boldsymbol{h}_{v,0,1}, ..., \boldsymbol{h}_{v,0,N}\right] = \boldsymbol{E}_v + \boldsymbol{T}^{\text{pos}}_v \tag{20}$$

For an image size of $224 \times 224$ pixels, and a patch size of $16 \times 16$ pixels, the number of patches $N$ is 196.

The architecture of a vision Transformer layer is almost identical to a language Transformer layer, with the only difference being that the LayerNorm operation $\text{LN}(\cdot)$ is applied before Multi-Head Attention $\text{MHA}(\cdot)$ [Dos+21], as illustrated in Figure 5. This type of Transformer

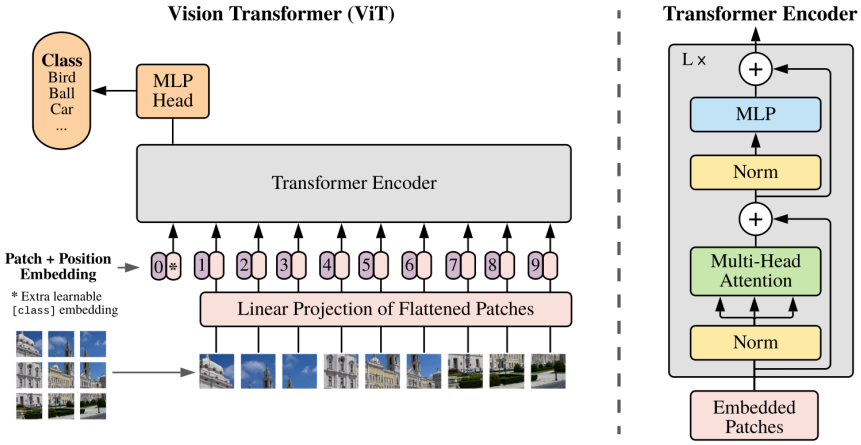


Figure 5: The architecture of the vision Transformer. Its key difference to the language Transformer is the patch embedding and the position of LayerNorm in a layer [Dos+21].

is referred to as the Pre-LN Transformer, and the operations performed by one layer change to:

$$\begin{aligned} \boldsymbol{H}'_l &= \mathrm{MHA}(\mathrm{LN}(\boldsymbol{H}_{l-1})) + \boldsymbol{H}_{l-1} \\ \boldsymbol{H}_l &= \mathrm{FFN}(\mathrm{LN}(\boldsymbol{H}'_l)) + \boldsymbol{H}'_l \end{aligned} \tag{21}$$

For downstream tasks like classification, the ViT follows the procedure of the original Transformer, and passes the representation of the [I_CLS] token to a classification head, which is a single linear (i.e. feed-forward) layer [Dos+21] (see Equation 17).

With the introducion of the vision Transformer came the division into three different model variants, each having the same architecture but different scales. The smallest model is the ViT-B/16, followed by the ViT-L/16. The largest model is the ViT-H/14. The number of layers $L$ and the hidden size $D$ are different for each model, and the ViT-B/16 has $L = 12$ layers and $D = 768$ hidden dim, the ViT-L/16 has $L = 24$ layers and $D = 1024$ hidden dim, and the ViT-H/14 has $L = 32$ layers and $D = 1280$ hidden dim [Dos+21]. Both ViT-B/16 and ViT-L/16 have a patch size of $16 \times 16$ pixels, while the ViT-H/14 has a patch size of $14 \times 14$ pixels. As might have come apparent, our explanation of the Transformer architecture is based on the ViT-B/16 model, which is the model we will make use of in this work.

## 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 of multimodality will be in the context of vision and language.

In the context of Deep Learning this means that the representations, the embedding/representation of e.g. the [I_CLS] or [T_CLS] token, of a concept should be the same (or at least close to each other), no matter if is expressed through image or text, 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 (seperate) 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 of this will be shown in Section 3.2.1.1.4.

To overcome this limitation, a model is required that can produce modality-invariant representations, i.e. representations that are independent of the modality of the input. 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, since they describe the same concept.

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

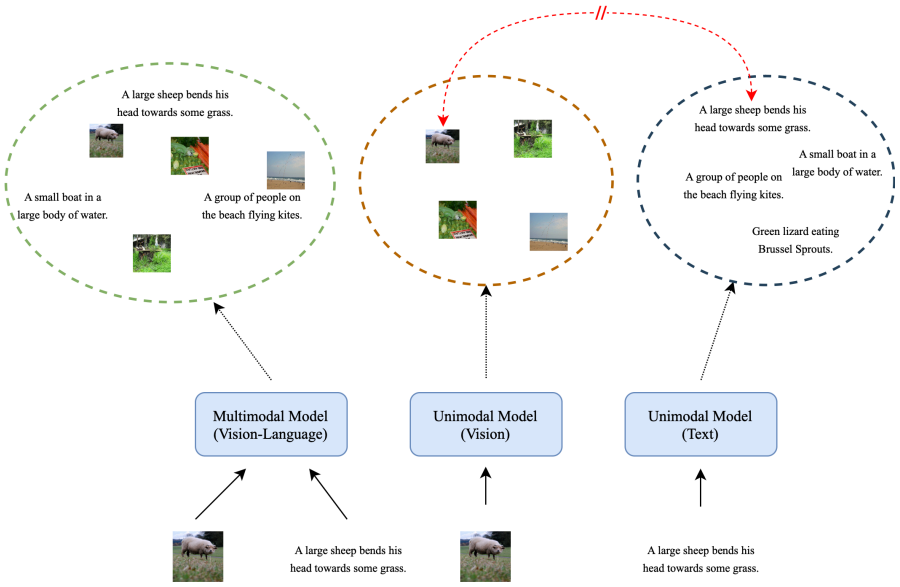


Figure 6: A multimodal model maps multiple modalities into a common representation space, where the representations of the same concept are aligned. In contrast, unimodal models map the input of a single modality into a modality-specific representation space. There is no alignment between the representations 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.

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 Section 1.5.3.2.

When it comes to the actual implementation of multimodal models, Transformers are a very suitable choice, as they can be used for both vision and language, and even other modalities not covered in this work, like audio. Furthermore, both the language and vision Transformer require their input to be a sequence of embeddings, which makes alignment more straightforward: For each unimodal encoder of a multimodal (vision-language) model one distinct Transformer can be used, and the output of a unimodal encoder, which is the respective cls token ([I_CLS] or [T_CLS]) can then be passed (seperately) to a shared encoder, which can be

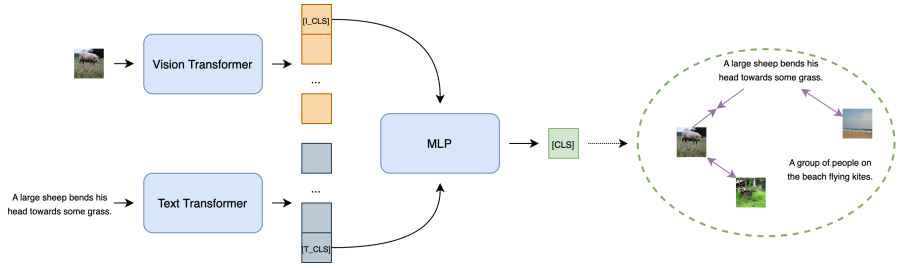


Figure 7: An abstract illustration of a vision-language model. Image and text are first passed through unimodal Transformer encoders, the cls tokens are extracted, and passed seperately into the MLP that maps the modality-specific representations into a common representation space. A contrastive loss ensures the alignment and repulsion of similar and dissimilar concepts, respectively. We indicate this through purple arrows.

implemented as a simple multi-layer MLP (feed forward). The output of the shared encoder is then still *one* representation for the image, and *one* for the text. However, both represenations will be close to each other under cosine similarity, which is useful for multimodal tasks like image-text retrieval, introduced in Section 1.6. An abstract illustration of a vision-language model with Transformer encoders and a shared encoder MLP is shown in Figure 7.

# 1.5 Self-Supervised Learning

## 1.5.1 Motivation

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-supervised 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 per-

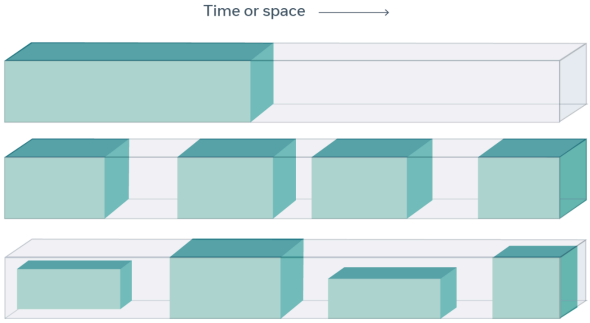


Figure 8: In self-supervised learning parts of the data are masked (grey), and the task of a model is to predict the masked parts using the visible data (green) [LM21].

formance 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.

### 1.5.2 Relationship to Multimodal Models

### 1.5.3 Contrastive Learning

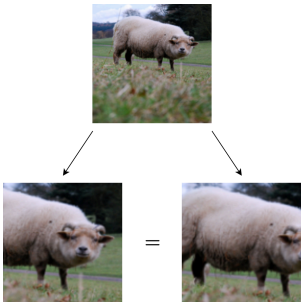

Figure 9: Adding small translations to an image, e.g. a random crop, as illustrated in the figure, will retrain high-level semantic features while changing pixel-level information. The content of the image stays the same, and the same should therefore hold for the representations produced by the model. Image in the figure has been taken from the COCO train set [Lin+14].

#### 1.5.3.1 Vision

In settings where masking discrete tokens and predicting them based on a set of possible tokens, as in language models, is not possible, contrastive learning can be used as a self-supervised method. This is especially useful in vision models, as images are continuous, so there is no discrete set of possible tokens to predict.

Contrastive learning, or the contrastive loss, is a method to learn representations of data without the need for labels, and used in computer vision models like MoCo [He+19], SimCLR [Che+20], and CLIP [Rad+21].

In computer vision, contrastive learning exploits the fact that the high-level semantics of an image are invariant to small (or moderate) changes in pixel-level information. This is achieved by augmenting the input image, e.g., by cropping, rotating, or flipping it. Provided the augmentation is not too drastic (e.g., crop size too large), the high-level semantics of the image will remain the same after augmentation, even though pixel-level information do not. The goal of the image model is then to maximize the cosine similarity between the global representations of two augmented versions of the same image. In Transformers, the global representation is usually the [CLS] token retuned by the final layer of the model, which is a vector that can be compared with the [CLS] token of another image using the cosine similarity. The augmented versions are often referred to as a different *view* of the same image [CH21], as shown in Figure 9.

However, this alone is not sufficient, as the model will collapse to a trivial solution by simply returning the same representation for all inputs, as demonstrated in the papers MoCo [He+19] and SimSiam [CH21]. Producing the same representation for all inputs is the simplest way to maximize the cosine similarity between the original image and its augmented versions, because the representation produced for an image would always be the same, therefore maximizing the cosine similarity (a value of 1). To prevent this, negative samples are introduced. Negative samples are other images that do not contain the same content as

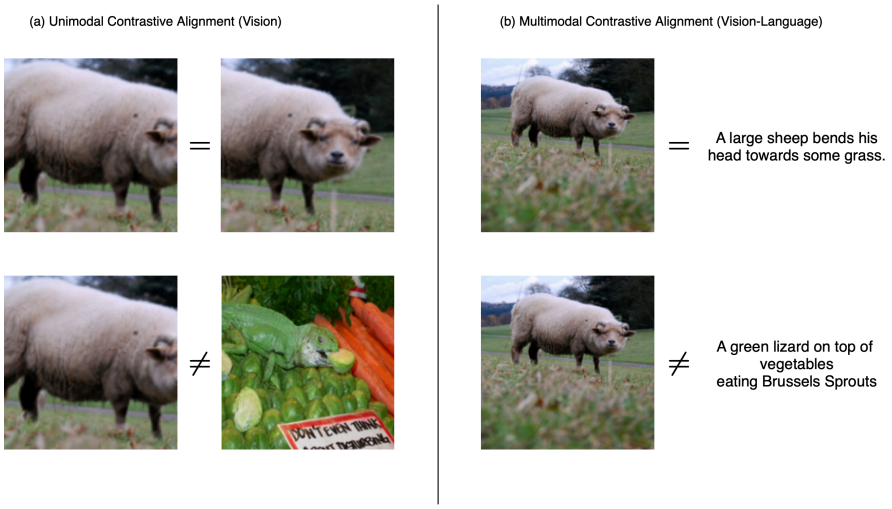


Figure 10: Contrastive learning aims to align the same (or similar) real-world concepts in representation space, while pushing different concepts apart. Multimodal contrastive learning (b) requires existing pairs, e.g. image-text, while for the unimodal case (a) pairs are synthetically created by augmenting the input. Images and text in the figure have been taken from the COCO train set [Lin+14].

the original image, and the cosine similarity between the original image and these negative samples should therefore be minimized (a cosine similarity of 0 indicates no similarity between the input vectors). This prevents the model from collapsing to a constant representation, as it would not minimize the cosine similarity and thus not minimize the loss. A simple yet expressive visualization can be found in [CH20]. This makes self-supervised training of image models possible, and the learned representations represent the high-level semantics of the images, learned without the need for labels.

An implementation and mathematical formulation of the contrastive loss will be introduced in (TODO: cite vision language contrast) on the example of vision-language models.

#### 1.5.3.2 Vision-Language

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 10.

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:

$$\cos(\boldsymbol{a}, \boldsymbol{b}) = \frac{\boldsymbol{a}\boldsymbol{b}^T}{\|\boldsymbol{a}\|_2 * \|\boldsymbol{b}\|_2} = \frac{\boldsymbol{a}}{\|\boldsymbol{a}\|_2}\frac{\boldsymbol{b}^T}{\|\boldsymbol{b}\|_2} \tag{22}$$

$\boldsymbol{a}\boldsymbol{b}^T$ 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, which we denote by the function $\delta(\cdot)$.

$$\delta(\boldsymbol{h}) = \frac{\boldsymbol{h}}{\|\boldsymbol{h}\|_2} \tag{23}$$

$$\boldsymbol{I} = \left[\delta\left(\boldsymbol{h}_{(v,L,[\texttt{I\_CLS}]),1}\right), \delta\left(\boldsymbol{h}_{(v,L,[\texttt{I\_CLS}]),2}\right), ..., \delta\left(\boldsymbol{h}_{(v,L,[\texttt{I\_CLS}]),B}\right)\right] \in \mathbb{R}^{B\times D} \tag{24}$$

$$\boldsymbol{T} = \left[\delta\left(\boldsymbol{h}_{(w,L,[\texttt{T\_CLS}]),1}\right), \delta\left(\boldsymbol{h}_{(w,L,[\texttt{T\_CLS}]),2}\right), ..., \delta\left(\boldsymbol{h}_{(w,L,[\texttt{T\_CLS}]),B}\right)\right] \in \mathbb{R}^{B\times D} \tag{25}$$

$\boldsymbol{I}$ denotes the batch/matrix of image representations, and $\boldsymbol{T}$ 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.

$$\boldsymbol{L} = \boldsymbol{I}\boldsymbol{T}^T, \boldsymbol{L} \in \mathbb{R}^{B\times B} \tag{26}$$

$L_{i,j}$ 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 $L_{i,i}$ 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:

$$\mathcal{L}_{\text{CL}}^{\text{i2t}} = \frac{1}{B}\sum_{i=1}^{B} -\log \frac{\exp(L_{i,i})}{\sum_{k=1}^{B}\exp(L_{i,k})} \tag{27}$$

$\frac{\exp(L_{i,i})}{\sum_{k=1}^{B}\exp(L_{i,k})}$ 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:

$$\mathcal{L}_{\text{CL}}^{\text{t2i}} = \frac{1}{B}\sum_{i=1}^{B} -\log \frac{\exp(L_{i,i})}{\sum_{k=1}^{B}\exp(L_{k,i})} \tag{28}$$

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:

$$\mathcal{L}_{\text{CL}} = \frac{1}{2} * \left(\mathcal{L}_{\text{CL}}^{\text{i2t}} + \mathcal{L}_{\text{CL}}^{\text{t2i}}\right) \tag{29}$$

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 11.

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

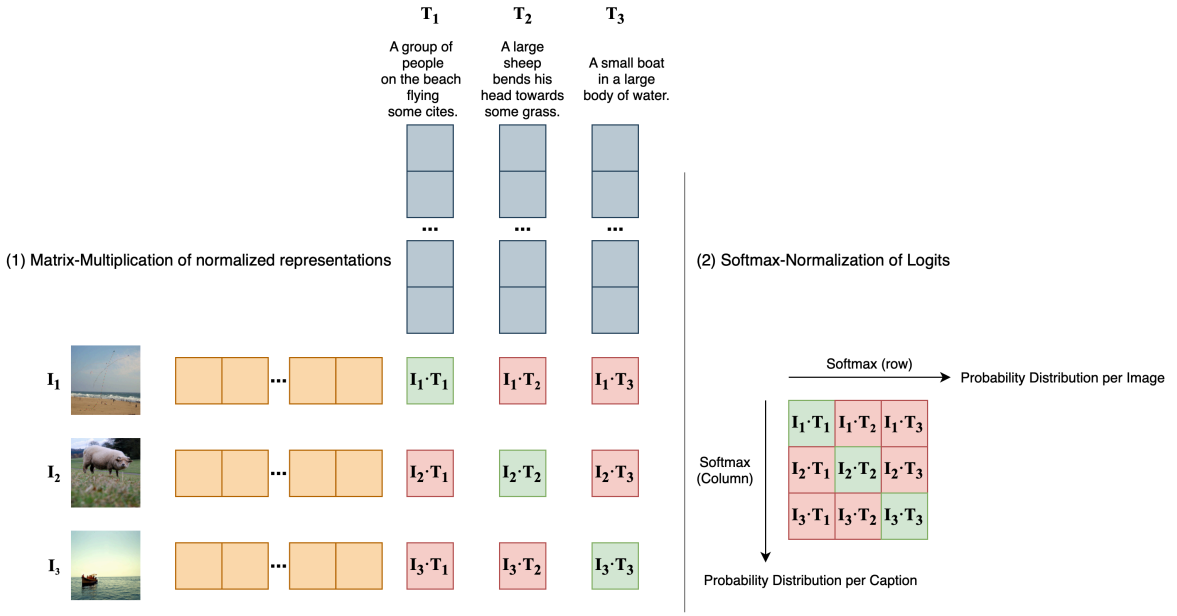


Figure 11: Contrastive Learning is performed using matrix multiplication of normalized representations (1), and the result is matrix $\boldsymbol{L}$ described in Equation 26. The representations are given by the [CLS] token of the respective modality, but are represented as I and T in the figure for 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].

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 multimodal 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

| 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.

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.

# 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 is similar to the definition of a multimodal model as defined in Section 1.4.

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, response based, approach to knowledge distillation, defined in Section 1.2.1.1. 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:

$$\mathcal{L}_{\text{KD}}^{\text{vw}} = \frac{1}{2} * D_{\text{KL}}(g(\boldsymbol{x}^v) \parallel s(f_v(\boldsymbol{x}^v))) + \frac{1}{2} * D_{\text{KL}}(g(\boldsymbol{x}^v) \parallel s(f_w(\boldsymbol{x}^w))) \tag{30}$$

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

$$\mathcal{L}_{\text{KD}}^{\text{va}} = \frac{1}{2} * D_{\text{KL}}(g(\boldsymbol{x}^v) \parallel s(f_v(\boldsymbol{x}^v))) + \frac{1}{2} * D_{\text{KL}}(g(\boldsymbol{x}^v) \parallel s(f_a(\boldsymbol{x}^a))) \tag{31}$$

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 is shown in Figure 39.

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 Section 1.5.3, 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

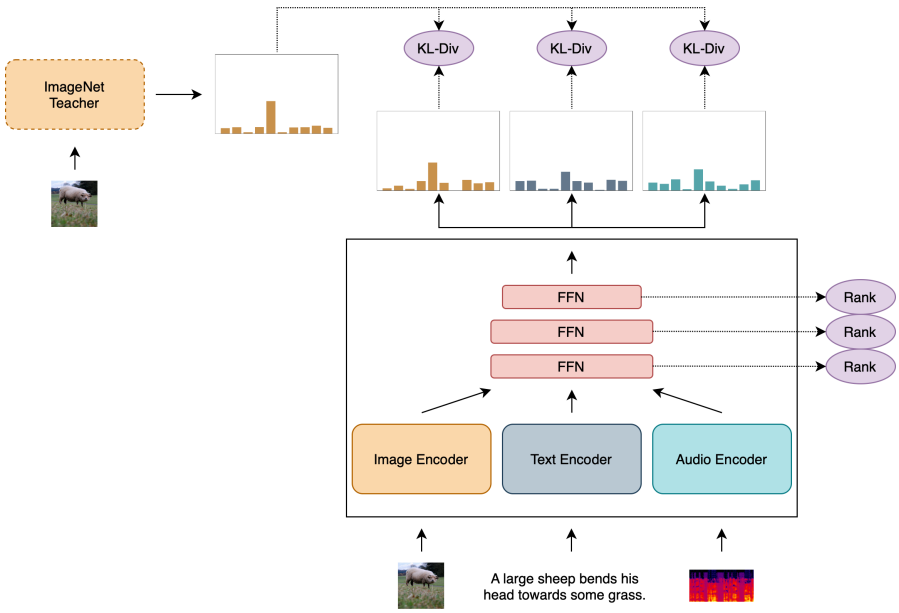


Figure 12: Illustration of the SHRe approach. The model is trained to output the same probability distribution over ImageNet-1K classes between images, image-text pairs, and image-audio pairs. Internal representations are aligned using a ranking loss [AVT17]. Image, text, and audio are always passed individually through the model. For each input, the model outputs a probability distribution over the ImageNet-1K classes (illustrated distributions are smaller for better visibility). 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].

from the same pair closer together, while pushing the representations of inputs from different pairs further apart. It is defined as:

$$\mathcal{L}_{\text{Rank}} = \sum_{i=1}^{B} \sum_{j \neq i} \max\{0, \Delta - \cos(\boldsymbol{x}_i^v, \boldsymbol{x}_i) + \cos(\boldsymbol{x}_i^v, \boldsymbol{x}_j)\} \tag{32}$$

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:

| 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].

$$\mathcal{L}_{\text{SHRe}} = \mathcal{L}_{\text{KD}} + \mathcal{L}_{\text{Rank}} \tag{33}$$

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.

The approach is illustrated in Figure 12. 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.

## 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 ar-

[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.

chitecture, 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:

$$\boldsymbol{L} = \exp(t) * \boldsymbol{I}' \boldsymbol{T}'^T, \boldsymbol{L} \in \mathbb{R}^{B \times B} \tag{34}$$

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).

$$\mathcal{L}_{\text{CLIP}} = \mathcal{L}_{\text{CL}} = \frac{1}{2} * \left(\mathcal{L}_{\text{CL}}^{\text{i2t}} + \mathcal{L}_{\text{CL}}^{\text{t2i}}\right) \tag{35}$$

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

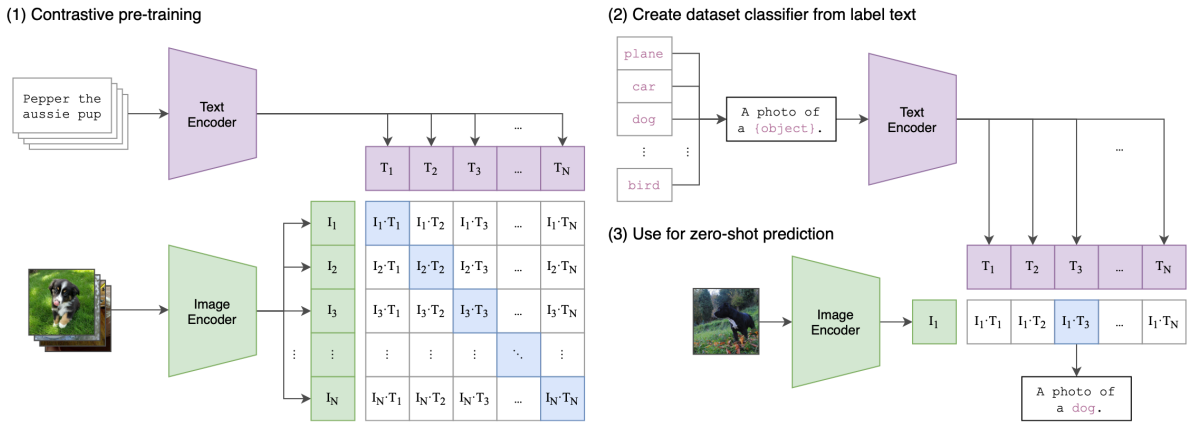


Figure 13: For zero-shot image classification, CLIP uses prompt engineering to create one classifier per image class to predict (2). The class whose classifier has the highest similarity (cosine) with the image representation is the predicted class (3) for the image [Rad+21].

highest similarity to the image representation is predicted as the class for the image, as demonstrated in Figure 13.

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].

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

# 2 Methodology

### 2.1.1 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)
  - 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

### 2.1.2 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

### 2.1.3 Data Collection and Preparation

The data we need to collect has to be both unimodal and multimodal. The requirement for multimodal data is obvious: We aim to align image and text, which requires a dataset of image-text pairs. Unimodal data is required for preliminary tests of classic unimodal knowledge distillation, on which we can then build. Further, we will utilize unimodal data for the evaluation of multimodal models on downstream tasks. After all, a multimodal model should not only excel in aligning modalities, but also in tasks that only involve on of the aligned modalities. This also gives us the opportunity to compare the performance of unimodal and multimodal distilled models on the same tasks.

#### 2.1.3.1 Unimodal Data

Collecting unimodal data does not pose an obstacle, as there are many highly curated and large datasets available. For image data, we select ImageNet-1K [Rus+15], which is an intu-

itive choice, as is features a high variety of content, is widely used for image classification, and, with 1.2 million training images, can be considered a medium-sized dataset. For comparison, current (August 2024) SOTA vision-language models have been trained on datasets spanning at least 14 million samples [Bao+22, Sin+21, Wan+23].

We will use this dataset for both knowledge distillation, and, most importantly, for the evaluation of image models using the ImageNet-1K validation accuracy metric, which is the most popular benchmark for computer vision models by far. We utilize the full dataset of the 2012 version, which contains 1.2 million images for training and 50,000 for validation [Rus+15]. The data can be downloaded from Huggingface's dataset hub[5] without any costs, merely requiring an account.

For raw text data, used in unimodal knowledge distillation of our text model, we select OpenWebText (OWT) [GC19]. This data was developed to replicate the datasets used to train GPT-2, and is also publicly available on HuggingFace[6]. The dataset consists of raw unstructured text, without any labels, which are not necessary for our distillation process. It is published as 21 chunks, and we select the first 6 for training and the 7th for validation, which is around 33% of the data. We do not collect the full dataset, as the training data, when slicing it into sections of 192 tokens, which each slice being one training example, already consists of more than 2.5 billion tokens, which we consider sufficient for our purposes. Even though the data is already preprocessed and cleaned, we further preprocess it by removing empty lines and null bytes, which we found to be quite common and lead to problems during encoding and training, as they provide no learnable information.

For benchmarking language models, including our multimodal models, on downstream tasks, we will use the GLUE benchmark [Wan+19]. GLUE, short for **G**eneral **L**anguage **U**nderstanding **E**valuation, is a collection of NLP datasets spanning four different tasks: Sentiment analysis (SST-2), grammar error detection (CoLA), sentence similarity (STS-B, MRPC, QQP), and natural language understanding (QNLI, MNLI, RTE). All 8 datasets are publicly available, and can also be accessed through HuggingFace[7].

The 8 datasets measure the performance of language models on the following tasks:

#### 2.1.3.1.1 SST-2

Sentence classification of rotten tomatoes movie reviews into "negative" (1), "somewhat negative" (2), "somewhat positive" (3), and "positive" (4) [Soc+13].

[5] https://huggingface.co/datasets/ILSVRC/imagenet-1k
[6] https://huggingface.co/datasets/Skylion007/openwebtext
[7] https://huggingface.co/datasets/nyu-mll/glue

#### 2.1.3.1.2 CoLA

Is a binary classification tasks to test a models understanding of grammar: Model should output whether a sentence is grammatically correct (label: “acceptable” -> 1) or not (label: “unacceptable” -> 0) [WSB18].

#### 2.1.3.1.3 STS-B

A regression task. The model is tasked with predicting the similarity between two sentences. The similarity score is in the interval $[0, 5] \subset \mathbb{R}$ [May21].

#### 2.1.3.1.4 MRPC

Is a binary classification taks. The training objective is paraphrase detection, meaning whether two sentences describe the same semantic concept [DB05].

#### 2.1.3.1.5 QQP

The same as MRPC, instead of a simple sentence pair, the goal is to detect wethere two questions are semantic duplicates, i.e. ask the same thing[8].

#### 2.1.3.1.6 QNLI

A binary classification task, where the model has to predict whether one sentence is the answer to a question represented by another sentence. Examples are of the form (question, sentence) [Raj+16, Wan+19].

#### 2.1.3.1.7 RTE

A dataset of text pairs, where the model has to predict whether a hypothesis (sentence 2) can be inferred from a text (sentence 1) [Ben+09, DGM06, Gia+07, Bar+06, Wan+19]. The task is binary classification (hypothesis can be inferred, or not).

#### 2.1.3.1.8 MNLI

A classification task, where the model has to predict whether a hypothesis can be inferred from the premise (entailment), contradicts the premise (contradiction), or is neutral (neutral). There a two versions available, MNLI matched and MNLI mismatched. Both consists of the same training dataset, but test set of MNLI mismatched consists of out-of-domain data, so sentence pairs about concepts not seen during training. It is therefore a better measure of generalization, compared to MNLI matched [WNB18].

Concrete examples can be found in (TODO: cite glue examples) in the Appendix.

[8] https://quoradata.quora.com/First-Quora-Dataset-Release-Question-Pairs

| Dataset | Training Examples |
|---|---|
| ImageNet-1K [Rus+15] | 1.28M |
| OpenWebText (subset) [GC19] | 13M |
| GLUE [Wan+19] | 990K (total) |
| Total | 15.27M |

Table 3: Unimodal datasets and their sizes used in this work. While the amount of training examples from OpenWebText is indeed correct, it is important to note that collecting text data is significantly cheaper to obtain, and requires less disk space than image data, which is why we were able to collect that much text data without any problems.

#### 2.1.3.2 Multimodal Data

For training multimodal models, specifically image-text models, datasets containing image-text pairs are required. We orient ourselves on the BEiTv3 paper, currently achieving SOTA performance on multimodal benchmarks [Wan+23]. The paper uses the datasets COCO [Lin+14], Visual Genome [Kri+17], Conceptual Captions 3M [Sha+18] and 12M [Cha+21], and SBU captions [OKB11], of which we select COCO, being the most popular and widely used dataset and therefore an intuitive choice, and a subset of both Conceptual Captions 3M and 12M.

While the COCO dataset can be downloaded in its entirety from the COCO website[9], both variants of Conceptual Captions, developed by Google, only provide urls and the caption for each image. This is because the images used come from a variety of sources on the internet, and have been uploaded by humans all over the world. The images stem from blog posts, news articles, social media, and other sources. Since Google does not own the rights to the images, they cannot provide them in a dedicated dataset, which is why there is no guarantee that all images will be available at the time of download. Because of this, we have to utilize not only Conceptual Captions 3M, but also the 12M variant to collect enough data. A favorable side effect this has is that our approach becomes more scalable due to the uncurated nature of CC12M [Cha+21, Sha+18], which we will elaborate on in the next section. The index of CC3M is available on the official Conceptual Captions website (training split)[10], and the index of CC12M can be found in the corresponding GitHub repository[11].

Another popular choice for image-text pairs is the Visual Genome dataset [Kri+17], containing high quality images and detailed annotations. However, we refrain from using this dataset, as we experienced unstable training during preliminary tests, a circumstance we will address again in the experimental part of this work.

[9] https://cocodataset.org/#download
[10] https://ai.google.com/research/ConceptualCaptions/download
[11] https://github.com/google-research-datasets/conceptual-12m

| Dataset | Avg. Caption Length | $\frac{\#\text{ Captions}}{\#\text{ Images}}$ | # Images | # Image-Text Pairs |
|---|---|---|---|---|
| COCO [Lin+14] | 11.0 | 5.0 | 82,783 | 566,747 |
| CC3M (subset) [Sha+18] | 12.0 | 1.0 | 1,516,133 | 1,516,133 |
| CC12M (subset) [Cha+21] | 10.3 | 1.0 | 1,181,988 | 1,181,988 |
| Total | - | - | 2,780,904 | 3,264,868 |

Table 4: Multimodal Dataset used for aligning image and text.

In total, we collect more than **650 GB** of data.

#### 2.1.3.3 On Curated Datasets

The goal of this work is to develop a multimodal model that is cheap to train and does not rely on labeled data in the end-to-end process. That means not only should our multimodal model not require labeled data for training, but also any pretrained models and components used in the process.

Whether image-text datasets, and, in fact, any multimodal dataset consisting of pairs of data, can be seen as curated or even labeled data is a matter of perspective. The difference between curated and labeled data lies in the purpose and level of human involvement: curated data focuses on the careful selection, organization, and cleaning of data to ensure quality and relevance, while labeled data involves explicit tagging or annotation of each example to provide a ground truth for training supervised models (which implies that the data is curated as well).

While image-text datasets are not labeled in the traditional sense, as in having a label for an image or text, the pairs themselves can be seen as labels. Single images or texts can be considered as in-the-wild data, i.e. data that appears naturally in the real world, like in books, articles, or on the internet, image-text pairs however require image and text to be paired together. This can be seen as less natural, as it requires a human to create the caption for an image, or vice versa, which is a form of labeling. The COCO dataset, for example, can be seen as labeled, as for each image a human created a caption with the specific intention of training Machine Learning models [Lin+14]. Consequently, whether multimodal learning can be seen as self-supervised learning, as it is often referred to in the literature [Bao+22, Sin+21, Wan+23], is debatable. With this in mind, creating a multimodal model that is scalable in the sense that it does not rely on labeled data, which is one of the most challenging aspects of AI research, is, if multimodal data is seen as labeled data, not possible.

However, there are multimodal data sources that are at the very least uncurated. One example is the alt-text of images on the internet. Even though the alt-text is created by humans, it is not created with the intention of creating data for Machine Learning, but rather to provide

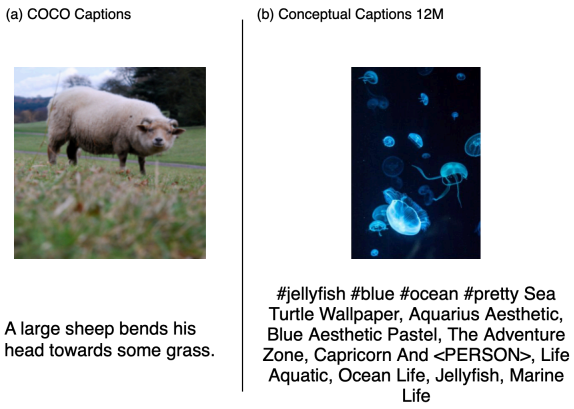


Figure 14: Side-by-side comparison of examples seen in COCO (a) and CC12M (b). While COCO features high quality images and detailed annotations, CC12M consists of in-the-wild image-text pairs from the internet. The latter enables scalability, as more data can be collected without the need for human annotation. The caveat is that the quality of the data is not guaranteed, and image-text pairs might be less correlated. Images and text in the figure have been taken from the COCO train set [Lin+14] and CC12M [Cha+21], respectively.

a description of the image for visually impaired people. Consequently, the data was generated naturally as a byproduct of a different task, and we therefore refer to any uncurated dataset as unlabeled data in this work.

This is exactly why we select both CC3M and CC12M, as they, especially CC12M, consists of in-the-wild image-text pairs from the internet [Cha+21, Sha+18]. This way of collecting data and training models is therefore significantly more scalable than using curated datasets specifically created for Machine Learning, and ensures that our approach to multimodal models can be applied to a wide range of tasks and domains without any explicit human intervention. A comparison between curated and labeled samples, and in-the-wild samples can be seen in Figure 14 below.

#### 2.1.3.4 Data Persistence

How to organize, store, and batch the data during training is an important aspect, as storing this much data is not trivial, and we need to ensure an efficient data pipeline to avoid bottlenecks during training.

For all datasets, except OpenWebText, we organize the data in a format inspired by the authors of BEiT-3. In the file system, each dataset is stored in a separate folder. Each split of a dataset is stored in a jsonl file (inside the respective dataset folder), containing a list of python-parseable dictionaries, with each dictionary representing one (train/val/test) example. Each dictionary contains a sample id, which is especially important for e.g. image-text retrieval, as the sample id is used to check if an image-text pair is a positive (they have the same id) or negative pair. Further, each dictionary contains a key for the image path in the file system, or a key for the text. If the dataset is multimodal, each sample, and therefore

each dictionary, contains both an image path key and a text key. The text, already splitted into a list of tokens, is directly stored in the dictionary, as it is small in size.

All datasets containing images have a separate folder for each split, which contains the raw images in png format.

During training, we load the whole jsonl file into memory, which is manageable, as they rarely exceed 1 GB in size. For each batch of size $B$, $B$ elements are drawn from the list of the jsonl file, where each element is the dictionary representing one example.

If the dataset contains images, the images are loaded from the file system using the image path in the dictionary, and then resized to a fixed size of 224x224 pixels, which is the size for images we use throughout this work. When appropriate, data augmentation is applied. Since all images are of the same size, they can be stacked into a tensor of shape $B \times 3 \times 224 \times 224$.

If the dataset contains text, which is a list of tokens already, each token of the text is converted to its corresponding token id, and the resulting list of token ids is prepended with the id of the [T_CLS] token, and appended with the id of the [T_SEP] token. The text is then padded to a fixed number of tokens, using the id of the [T_PAD] token. To which fixed length the text is padded depends on the model and approach, and will be specified in the respective section. After padding, all texts are stacked into a tensor of shape $B \times T$, where $T$ is the fixed number of tokens.

Since sample ids are simple integers, they can be stacked into a tensor of shape $B \times 1$ without any further processing.

After that, the batch is ready to be fed into the model.

For OpenWebText we take a different approach. The data is stored in a single binary file for each split, which contains the tokenized text data. We choose this strategy for OpenWebText, as the dataset consists of raw unstructured text, from which training examples can easily be created during training by loading the whole dataset into memory, and slicing the text, into consecutive slices of fixed length. Padding is not necessary, as the slices are already of the same length. Each token is then converted to its corresponding token id, and the resulting list of token ids is prepended with the id of the [T_CLS] token, and appended with the id of the [T_SEP] token. Consequently, for a maximum sequence length of $T$ tokens, the whole dataset, which is one long list of tokens, is iterated over during training in steps of $B * (T - 2)$, where $B$ is the batch size. We do not take $B * T$, as for each example the [T_CLS] and [T_SEP] token are added, which increases the length of each example by 2 tokens. After a slice has been taken from the dataset, was converted to token ids, and special tokens were added, it is stacked into a tensor of shape $B \times T$, and then fed into the model.

The different data formats used in this work are compared in Figure 15 below.

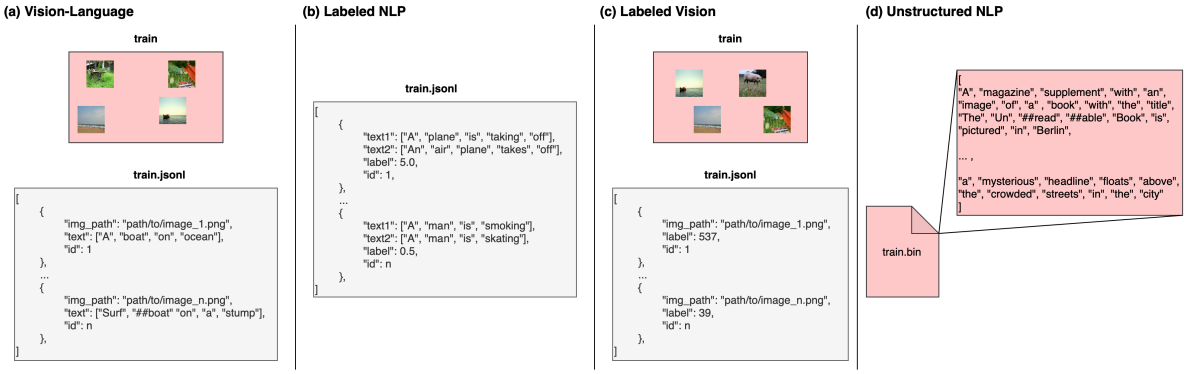


Figure 15: Comparison of different dataset organization and storage formats used in this work. Labeled NLP (d) datasets are not binarized, as they have a structure, i.e. there are fixed examples with labels.

# 3 Experiments

## 3.1 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.

### 3.1.1 Vision

#### 3.1.1.1 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:

$$h'_{j,d} = \frac{h_{j,d} - \mu_d}{\sqrt{\sigma_d^2 + \varepsilon}}, \qquad \mu_k = \frac{1}{T}\sum_{t=1}^{T} h_{t,k}, \qquad \sigma_k^2 = \frac{1}{T}\sum_{t=1}^{T}\left(h_{t,k} - \mu_k\right)^2 \tag{36}$$

Even though the formula might look complicated, it is quite simple in practice. For each embedding dimension $d$, the mean $\mu_d$ and standard deviation $\sigma_d$ 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 Figure 35 in the Appendix. We define the operation $\mathrm{IN}(\cdot)$ as instance normalization on a sequence of embeddings $\boldsymbol{H}$.

$$\mathrm{IN}(\boldsymbol{H}) = [\boldsymbol{h}'_1, \boldsymbol{h}'_2, ..., \boldsymbol{h}'_T] \tag{37}$$

After instance norm and averaging, parameter-less layer normalization, denoted as $\mathrm{LN}(\cdot)$, is performed [Bae+22, Bae+22]. We perform all three operations likewise. The target and prediction are therefore given by:

$$\begin{aligned} \boldsymbol{H'}^{t}_{v,l} &= \mathrm{IN}\Big(\boldsymbol{H}^{t}_{v,l}\Big), l \in \{1, 2, ..., L_t\} \\ \boldsymbol{H'}^{s}_{v,l} &= \mathrm{IN}\Big(\boldsymbol{H}^{s}_{v,l}\Big), l \in \{1, 2, ..., L_s\} \end{aligned} \tag{38}$$

$$\begin{aligned} \widehat{\boldsymbol{H}}^{t}_{v} &= \frac{1}{L_t} \sum_{l=1}^{L_t} \boldsymbol{H'}^{t}_{v,l} \\ \widehat{\boldsymbol{H}}^{s}_{v} &= \frac{1}{L_s} \sum_{l=1}^{L_s} \boldsymbol{H'}^{s}_{v,l} \end{aligned} \tag{39}$$

$$\begin{aligned} \boldsymbol{Y} &= \Big[\boldsymbol{y}_{[\texttt{I\_CLS}]}, \boldsymbol{y}_1, ..., \boldsymbol{y}_N\Big] = \mathrm{LN}\Big(\widehat{\boldsymbol{H}}^{t}_{v}\Big) \\ \widehat{\boldsymbol{Y}} &= \Big[\hat{\boldsymbol{y}}_{[\texttt{I\_CLS}]}, \hat{\boldsymbol{y}}_1, ..., \hat{\boldsymbol{y}}_N\Big] = \mathrm{LN}\Big(\widehat{\boldsymbol{H}}^{s}_{v}\Big) \end{aligned} \tag{40}$$

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

$$\begin{aligned} &\mathcal{L}_{\mathrm{KD}}\Big(\boldsymbol{Y}, \widehat{\boldsymbol{Y}}\Big) = \|\boldsymbol{Y} - \widehat{\boldsymbol{Y}}\|_2^2 = \\ &\frac{1}{N+1}\left(\sum_{n=1}^{N} \mathcal{L}_{\mathrm{MSE}}(\boldsymbol{y}_n, \hat{\boldsymbol{y}}_n) + \mathcal{L}_{\mathrm{MSE}}\Big(\boldsymbol{y}_{[\texttt{I\_CLS}]}, \hat{\boldsymbol{y}}_{[\texttt{I\_CLS}]}\Big)\right) \end{aligned} \tag{41}$$

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. $\mathcal{L}_{\mathrm{MSE}}(\cdot, \cdot)$ is the mean squared error between two vectors, defined in Equation 1.

#### 3.1.1.2 Distillation

We distill the student model by minimizing the loss defined in Equation 41 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 41 also for validation. The total number of parameters

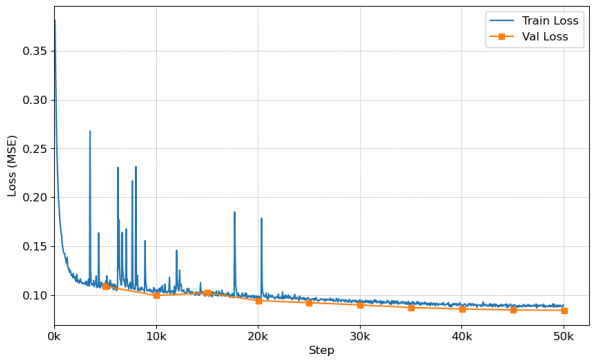


Figure 16: Training loss vs. validation loss during distillation of the Data2Vec2 image model.

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 Table 26.

We show the evolution of the training and validation loss during training in Figure 16. 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.

#### 3.1.1.3 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

| **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 5: 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.

[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 Code 1.

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 Table 27, 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:

$$\mathrm{lr}_l = \mathrm{base\_lr} * \mathrm{layer\_decay}^{L_s+1-l} \tag{42}$$

The learning rates can be seen in the following table:

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

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

Table 6: 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 7: Results of finetuning and linear probing of our distilled Data2Vec2 image model on CIFAR-10 and CIFAR-100.

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 Table 27.

The results, displayed in Table 6 and Table 7, 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.

## 3.1.2 Language

### 3.1.2.1 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, 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. As with our image model, we 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 41. 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 the philosophy of Data2Vec2, which uses the same loss for all modalities [Bae+22].

#### 3.1.2.2 Importance of Sequence Length

Setting the maximum sequence length as long as possible is crucial when regressing the mean activation of all teacher layers for each token. Self-attention mechanisms enable each token to encode not only its own information but also contextual information from other tokens in the sequence. This phenomenon, referred to as "contextualized targets" by the authors of Data2Vec [Bae+22, Bae+22], becomes more pronounced with longer sequences. A larger number of tokens in the sequence increases the opportunities for tokens to attend to one another, thereby enriching contextual representations and enhancing the model's ability to capture complex token relationships.

For instance, in the short phrase "a dog", each token can attend to only one other token, which limits the model's capacity to understand the broader context and relationships. In contrast, the sentence "a dog is playing in the garden with a tennis ball" provides a richer context with more tokens. This expanded context allows the model to capture a wider variety of relationships between tokens, offering the student model more opportunities to learn from the teacher model's outputs. A visualization of the attention between the different tokens of both sentences is provided in Figure 17. Here, we can see that a longer sequence allows tokens to attend to a high variety of other tokens, leading to relationships that represent real-world scenarios. The token "dog" attends mostly to tokens "playing", "garden", "tennis" and "ball", which combined represent the context of the sentence and a real-world scenario. The representation of the token "dog" therefore encodes information about a dog and what it is doing, which can be learned by the student model when regressing the teacher's outputs. In contrast, the token "dog" in the short sentence "a dog" can only attend to the token "a" (and the special tokens [CLS] and [SEP]), which does not provide any context about the dog. However, the disadvantage of longer sequences is that they require more memory and computational resources, as the self-attention mechanism has a quadratic complexity with respect to the sequence length.

A variable sequence length is less relevant for the image model in Section 3.1.1, as the size of an image is fixed to $224 \times 244$ pixels, so there is no variable sequence length.

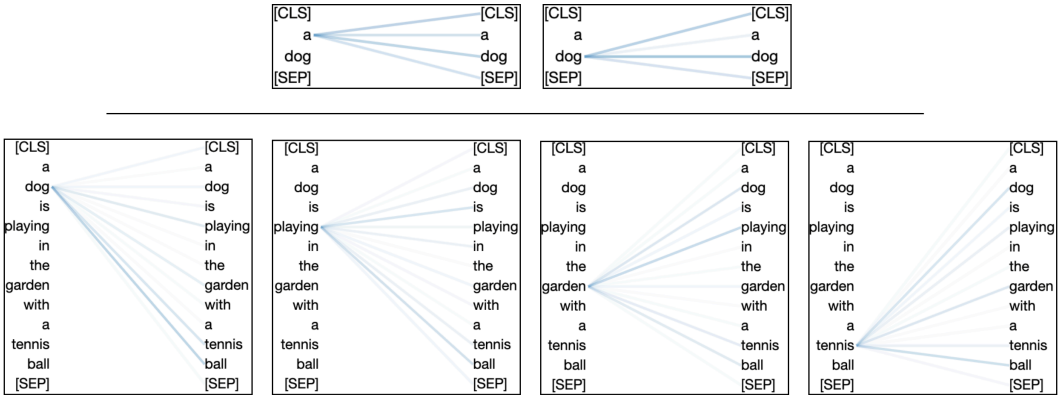


Figure 17: Visualization of the attention between tokens in a short and longer sentence. Longer sequences allow for more complex relationships between tokens that represent the real-world context of the sentence. Attention scores have been taken from the first attention head of the first Transformer layer of the trained uncased BERT model used for distillation.

#### 3.1.2.3 Distillation

During training, we set the maximum sequence length to 256, which is the maximum sequence length that can fit on a single GPU with 24GB of memory (RTX 4090).

The BERT model is distilled on a subset of the OpenWebText dataset [GC19], introduced in Section 2.1.3.1, of which the text is tokenized into subwords using the BERT tokenizer. We use the uncased variant, meaning all tokens are first converted to lowercase: To the model, there is no difference between “Dog” and “dog”.

We validate the student model on the dedicated validation dataset of OWT we introduced in Section 2.1.3.1.

The model is trained for only one epoch, as we found the model to converge quickly, and because the text data we collect yields almost 1M batches of size 256, with a sequence length of 256, which is very large. In comparison, the 1.2M images of ImageNet-1K [Rus+15] yield 5004 batches of size 256 for a single epoch.

Other hyperparameters used, including the learning rate, are similar to that of the image model, and are provided in Table 28.

Before the loss, defined in Equation 47, is applied, we perform a similar preprocessing as for the image model. However, following Data2Vec2 [Bae+22], we do not apply layer normalization on the averaged activations, and the loss is only calculated for non-padding tokens. This was not relevant in the image model, as there is no padding in the image data.

$$\begin{aligned} \boldsymbol{H'}^{t}_{w,l} &= \mathrm{IN}\left(\boldsymbol{H}^{t}_{w,l}\right), l \in \{1, 2, ..., L_t\} \\ \boldsymbol{H'}^{s}_{w,l} &= \mathrm{IN}\left(\boldsymbol{H}^{s}_{w,l}\right), l \in \{1, 2, ..., L_s\} \end{aligned} \tag{43}$$

$$\begin{aligned}\widehat{\boldsymbol{H}}_w^t &= \frac{1}{L_t}\sum_{l=1}^{L_t} \boldsymbol{H'}_{w,l}^t \\ \widehat{\boldsymbol{H}}_w^s &= \frac{1}{L_s}\sum_{l=1}^{L_s} \boldsymbol{H'}_{w,l}^s\end{aligned} \tag{44}$$

$$\begin{aligned}\boldsymbol{Y} &= \left[\boldsymbol{y}_{[\texttt{T\_CLS}]}, \boldsymbol{y}_1, ..., \boldsymbol{y}_M, \boldsymbol{y}_{[\texttt{SEP}]}\right] = \widehat{\boldsymbol{H}}_w^t \\ \widehat{\boldsymbol{Y}} &= \left[\hat{\boldsymbol{y}}_{[\texttt{T\_CLS}]}, \hat{\boldsymbol{y}}_1, ..., \hat{\boldsymbol{y}}_M, \hat{\boldsymbol{y}}_{[\texttt{SEP}]}\right] = \widehat{\boldsymbol{H}}_w^s\end{aligned} \tag{45}$$

$$\mathcal{L}'_{\text{MSE}}(\boldsymbol{y}_m, \hat{\boldsymbol{y}}_m) \coloneqq \begin{cases} 0 & \text{if } \boldsymbol{e}_m^w = \boldsymbol{t}_{[\texttt{PAD}]} \\ \mathcal{L}_{\text{MSE}}(\boldsymbol{y}_m, \hat{\boldsymbol{y}}_m) & \text{else} \end{cases} \tag{46}$$

$$\begin{aligned}\mathcal{L}_{\text{KD}}\left(\boldsymbol{Y}, \widehat{\boldsymbol{Y}}\right) = \|\boldsymbol{Y} - \widehat{\boldsymbol{Y}}\|_2^2 = \\ \frac{1}{M+2}\left(\sum_{m=1}^{M} \mathcal{L}'_{\text{MSE}}(\boldsymbol{y}_m, \hat{\boldsymbol{y}}_m) + \mathcal{L}_{\text{MSE}}\left(\boldsymbol{y}_{[\texttt{T\_CLS}]}, \hat{\boldsymbol{y}}_{[\texttt{T\_CLS}]}\right) + \mathcal{L}_{\text{MSE}}\left(\boldsymbol{y}_{[\texttt{SEP}]}, \hat{\boldsymbol{y}}_{[\texttt{SEP}]}\right)\right)\end{aligned} \tag{47}$$

Here, $\boldsymbol{e}_m^w$ denotes the embedding of token $m$ in the sequence, and the loss is ignored, i.e. 0, if the token is the padding token, denoted $\boldsymbol{t}_{[\texttt{PAD}]}$.

#### 3.1.2.4 Finetuning

To get a sense of the language understanding capabilities of the trained/distilled student model, we finetune it on all GLUE benchmark tasks [Wan+19], which are described in Section 2.1.3.1, and visualized in Table 32.

**Model Setup**

To perform finetuning, we load the weights of the trained student model. After an example (e.g. sentence pair) is passed through the Transformer layer, the representation $\boldsymbol{h}_{w,L,[\texttt{T\_CLS}]}$ of the [CLS] token is extracted, and passed through the BERT pooler [Dev+19], which is a linear layer followed by a Tanh activation function. The linear layer retrains the input dimensionality of the [CLS] token, which is 768. The weights of the pooler come directly from the BERT model, and are also fine-tuned. The pooler is followed by a dropout layer, for which the dropout probability differs between tasks (shown in Table 29), and is followed by a linear classification layer, which maps the representation to the number of classes of the downstream task. If the task is a regression task, for example sentence similarity in [0, 5] for STS-B [May21], the second linear layer returns a scalar value. The classification layer is initialized randomly, and trained from scratch. Pytorch pseudocode for the forward pass is provided in Code 2.

**Details on Tokens** There are two important things to consider when tokenizing the input for the GLUE tasks.

First, we set the maximum sequence length to 256, which is the same as for the distillation process. In theory, it is possible to set the maximum sequence length to 512, which is the maximum sequence length BERT can handle [Dev+19] without interpolation. However, this would (1) cause problems with memory, as those sequences are too long for a 24GB GPU. Furthermore, (2) the positional encoding of the student model comes directly from the teacher model, which is BERT. The positional encoding $\boldsymbol{T}_w^{\text{pos}}$ of BERT is trainable, and has one positional encoding $\boldsymbol{t}_{\text{pos}}^w$ for each position in the sequence. Since we only distill with a sequence length of 256, only the first 256 positional encodings are actually trained during distillation, meaning that the positional encodings for positions 257 to 512 are not trained further, and therefore still the same as in the normal BERT model. That means they are not "used" to a BERT model that has only 6, instead of 12, Transformer blocks, and therefore might not be optimal for the student model.

In general, a sequence length of 256 is acceptable, as most of the GLUE tasks rarely have examples that exceed this length. If an example is longer than 256 tokens, it is truncated to the first 256 tokens. If an example consists of a sentence pair, both sentences are tuncated equally, so that the total length of the sequence is 256.

Second, if the task consists of sentence pairs, we add a special token-type embedding to each token before the positional encoding is added. This, together with the [T_SEP] between both sentences, helps the model to better differentiate between the two sentences in the sentence pair, and the first token-type embedding $\boldsymbol{t}_{[\text{TYP\_1}]}^w$ is added to each token of the first sentence, and the second token-type embedding $\boldsymbol{t}_{[\text{TYP\_2}]}^w$ is added to each token of the second sentence:

$$\begin{aligned}\boldsymbol{E}_w = \Big[&\boldsymbol{e}_{[\text{T\_CLS}]}^w, \boldsymbol{e}_1^w, \boldsymbol{e}_2^w, ..., \boldsymbol{e}_{M_1}^w, \boldsymbol{e}_{[\text{T\_SEP}]}^w, \\ &\boldsymbol{e}_{M_1+1}^w, \boldsymbol{e}_{M_1+2}^w, ..., \boldsymbol{e}_{M_1+M_2}^w, \boldsymbol{e}_{[\text{T\_SEP}]}^w\Big] \in \mathbb{R}^{(M_1+M_2+3)\times D}\end{aligned} \tag{48}$$

$$\begin{aligned}\boldsymbol{E}'_w = \Big[&\boldsymbol{t}_{[\text{TYP\_1}]}^w + \boldsymbol{e}_{[\text{T\_CLS}]}^w, \boldsymbol{t}_{[\text{TYP\_1}]}^w + \boldsymbol{e}_1^w, \\ &\boldsymbol{t}_{[\text{TYP\_1}]}^w + \boldsymbol{e}_2^w, ..., \boldsymbol{t}_{[\text{TYP\_1}]}^w + \boldsymbol{e}_{M_1}^w, \\ &\boldsymbol{t}_{[\text{TYP\_1}]}^w + \boldsymbol{e}_{[\text{T\_SEP}]}^w, \boldsymbol{t}_{[\text{TYP\_2}]}^w + \boldsymbol{e}_{M_1+1}^w, \\ &\boldsymbol{t}_{[\text{TYP\_2}]}^w + \boldsymbol{e}_{M_1+2}^w, ..., \boldsymbol{t}_{[\text{TYP\_2}]}^w + \boldsymbol{e}_{M_1+M_2}^w, \\ &\boldsymbol{t}_{[\text{TYP\_2}]}^w + \boldsymbol{e}_{[\text{T\_SEP}]}^w\Big]\end{aligned} \tag{49}$$

$$\boldsymbol{H}_{w,0} = \Big[\boldsymbol{h}_{w,0,[\text{T\_CLS}]}, \boldsymbol{h}_{w,0,1}, ..., \boldsymbol{h}_{w,0,M_1+M_2}, \boldsymbol{h}_{w,0,[\text{T\_SEP}]}\Big] = \boldsymbol{E}'_w + \boldsymbol{T}_w^{\text{pos}} \tag{50}$$

The representations $\boldsymbol{t}_{[\text{TYP\_1}]}^w$ and $\boldsymbol{t}_{[\text{TYP\_2}]}^w$ of the token-type embeddings are part of the BERT model [Dev+19], but are not used during distillation, as there are no sentence pairs during distillation. However, during finetuning on sentence pairs, they are required, and we take the pretrained token-type embeddings from the BERT model and also train them during finetuning.

| | MNLI | QNLI | RTE | MRPC | QQP | STS-B | CoLA | SST | **Score** |
|---|---|---|---|---|---|---|---|---|---|
| ELMo [Pet+18] | 68.6 | 71.1 | 53.4 | 76.7 | 86.2 | 70.4 | 44.1 | 91.5 | 70.24 |
| BERT [Dev+19] | **86.7** | **91.8** | **69.3** | **88.6** | **89.6** | **89.0** | **56.3** | **92.7** | **83.0** |
| DistilBERT [San+19] | 82.2 | 89.2 | 59.9 | 87.5 | 88.5 | 86.9 | 51.3 | 91.3 | 79.31 |
| F-DistilBERT (ours) | - | 88.0 | - | 85.0 | - | - | <u>55.1</u> | - | - |

Table 8: Comparison of finetuning results on the GLUE benchmark tasks [Wan+19] with other models. Results for BERT [Dev+19], DistilBERT [San+19], and ELMo [Pet+18] are taken from the DistilBERT paper [San+19]. Bold scores indicate the best score for the respective task, while underlined scores indicate the best score for distilled models. The metrics, and therefore the scores shown, to evaluate the models are task-specific, and shown in Table 29.

For examples of sentence pairs, see Table 32.

**Hyperparameters**

Since for most tasks the amount of training data is marginal, e.g. CoLA [WSB18] has only 8.5k training samples, we do not use the layer decay technique used for the image model, and directly select a very low learning rate. Most of the hyperparameters are inspired by BERT [Dev+19], and Data2Vec [Bae+22, Bae+22], and are provided in Table 29. We increase the number of epochs and lower the batch size if the dataset, like CoLA, is very small. This ensures that we have a sufficient number of updates for the model to learn from the data.

**Results**

# 3.2 Multimodal Knowledge Distillation

## 3.2.1 Transformer SHRe

Before we develop an end-to-end self-supervised approach to multimodal knowledge distillation, we first follow the approach of SHRe [AVT17] and develop a multimodal knowledge distillation with a probability distribution over the classes as the prediction target. This allows us to closely observe the impact of switching from a supervised to a self-supervised teacher model on the student model's performance. Moreover, it allows us to gradually increase the complexity of our approach and build on our previous advancements.

### 3.2.1.1 Method

#### 3.2.1.1.1 Architecture

What makes our approach different from SHRe is that we use a language Transformer as the text encoder, and a vision Transformer as the image encoder, bringing us closer to a unified

architecture. In contrast, SHRe uses 2D convolutions and 1D convolutions for the image and text, respectively. For now, the shared encoder remains a 3-layer MLP, as in SHRe [AVT17]. Since the output of the image and text encoder is a sequence of features, but we use a normal MLP as the shared encoder, we have to reduce the sequence of features to a single feature vector. This is done by taking the representation of the [I_CLS] and [T_CLS] token from the image and text encoder, respectively, and aligns with our first implementation of a multimodal model shown in Figure 7. This represenation, namely $\boldsymbol{h}_{v,L_s,[\texttt{I\_CLS}]}$ and $\boldsymbol{h}_{w,L_s,[\texttt{T\_CLS}]}$, is then passed to the shared encoder, which produces the final multimodal representations. Here, $L_s$ denotes the number of Transformer layers in the image and text encoder, respectively, which is defined as $L_s = 6$. We remove the token indicator [I_CLS] and [T_CLS] from the notation, as the MLP only receives those cls token representations as input. There are no more time steps to distinguish between.

To keep the model size manageable, we will resort the the same approach as in our unimodal experiment, and use 6 Transformer layers for the image and text encoder, respectively. This also gives us the opportunity to directly compare the performance of the image and text encoder from our multimodal model to that of the unimodal models (Section 3.1). This can be done by evaluating e.g. the image encoder of the multimodal model on image classification tasks, and then comparing its performance to the results observed for the unimodal image KD model of Section 3.1.1 on the same tasks. A performance similar to that of the strictly unimodal models would indicate that multimodal pretraining yields strong unimodal encoders as a byproduct. As argued by the authors of FLAVA [Sin+21], the aforementioned is in fact even a requirement for multimodal models. This can be attributed to the fact that an alignment in the shared encoder is only possible if the unimodal encoders generate features from which the shared encoder can extract modality-invariant information, like the semantic content of an image or text. If the unimodal encoders are not able to extract high-level features, then neither will the shared encoder be able to extract modality-invariant information, nor will the features be useful in the corresponding unimodal downstream tasks, like for example image classification using the image encoder of the multimodal model.

As done in our unimodal experiments, we initialize the image and text encoder using the embeddings, positional encodings, and the first 6 Transformer layers from Data2Vec2 [Bae+22] and BERT [Dev+19], respectively. The shared encoder will be initialized randomly. For an illustration of the architecture we refer to the same depiction used for SHRe, shown in Figure 12. The only difference is that we only use image and text encoders, which are now Transformers instead of CNNs.

The shared encoder, which is a 3-layer MLP, is implemented by using the same 2-layer MLP module as present in Transformer layers, and adding an additional LayerNorm and linear layer on top of it. Given the output from the text encoder, the forward pass of the shared encoder is then given in the following, and changes for the image encoder accordingly:

| Component | # Params |
|---|---|
| Image Encoder | 42.3M |
| Text Encoder | 66.4M |
| Shared Encoder | 5.5M |
| **Total** | 114.2M |

Table 9: A summary of the number of parameters of the Transformer SHRe model.

$$\begin{aligned} \boldsymbol{h}'_{w,K} &= \boldsymbol{h}_{w,L_s,[\text{T\_CLS}]}\boldsymbol{W}_1 + \boldsymbol{b}_1 \\ \boldsymbol{h}''_{w,K} &= \text{LN}\big(\text{GELU}\big(\boldsymbol{h}'_{w,K}\big)\big)\boldsymbol{W}_2 + \boldsymbol{b}_2 \end{aligned} \tag{51}$$

$$\boldsymbol{h}'''_{w,K} = \text{LN}\big(\boldsymbol{h}''_{w,K}\big)\boldsymbol{W}_3 + \boldsymbol{b}_3 \tag{52}$$

We consider the 3-layer MLP as a single layer stacked on the image and text encoder, and therefore denote the layer number as $K = L_s + 1 = 7$. The operation done in Equation 51 is analogous to the definition of the MLP layers in a Transformer layer, defined in Equation 15. We choose a different notation here to also capture the result $\boldsymbol{h}'_{w,K}$, or $\boldsymbol{h}'_{v,K}$ for an image, of the first linear layer (given by parameters $\boldsymbol{W}_1$ and $\boldsymbol{b}_1$), which is important when defining the loss in the next section.

The whole model has a total of around 114.2M parameters, which is broken down in Table 9.

The 24M parameters the text encoder has more than the image encoder can be attributed to the embedding matrix of the text encoder, which alone has 23.4M parameters. Since we use parts of a pretrained BERT model, we also have to resort to using the BERT tokenizer and vocabulary. The vocabulary consists of 30522 (sub)words, and the embedding matrix has a dimensionality of 768 ($30522 * 768 = 23.4M$). The rest of parameters the text encoder has more is related to BERT's specific implementation of positional encodings.

While 115M parameters can be considered as quite large, considering we strive to build smaller models, it is still significantly smaller than the vision-language models we compare to. For example, VLMo [Bao+22] has 175M[12], CLIP [Rad+21] has more than 400M[13], and BEiT-3 [Wan+23] has 1.9B parameters [Wan+23].

#### 3.2.1.1.2 Training Objective

Since we start with using a supervised teacher, the loss for knowledge distillation will remain KL-Divergence. As the application of the KL-Divergence is two-fold, once for the prediction based on the image and once for the prediction based on the caption, we provide a refined version of the loss function. As a preliminary step, we define the softmax normalization of a vector $\boldsymbol{u}$ as follows:

[12] https://github.com/microsoft/unilm/tree/master/vlmo
[13] https://huggingface.co/openai/clip-vit-large-patch14

$$\pi(\boldsymbol{u}) = \boldsymbol{z} = [z_1, z_2, ..., z_n] \in \mathbb{R}^n$$
$$z_i = \frac{\exp(u_i)}{\sum_{j=1}^{n} \exp(u_j)} \tag{53}$$

This allows us to generate a probability distribution over the classes for the logits generated by the teacher for the image, and for the logits generated by the student for image and caption. The loss for knowledge distillation is then given by:

$$\begin{gathered}\mathcal{L}_{\mathrm{KD}} = \\ \frac{1}{2} * \mathcal{L}_{\mathrm{KD}}^{v} + \frac{1}{2} * \mathcal{L}_{\mathrm{KD}}^{w} = \\ \frac{1}{2} * D_{\mathrm{KL}}\big(\pi(\boldsymbol{p}), \pi\big(\boldsymbol{h}_{v,K}'''\big)\big) + \frac{1}{2} * D_{\mathrm{KL}}\big(\pi(\boldsymbol{p}), \pi\big(\boldsymbol{h}_{w,K}'''\big)\big)\end{gathered} \tag{54}$$

$\boldsymbol{p}$ denotes the logits generated by the teacher for the image, $\boldsymbol{h}_{v,K}'''$ the logits generated by the student for the image, and $\boldsymbol{h}_{w,K}'''$ the logits generated by the student for the caption. All are in $\mathbb{R}^{1000}$ for the 1000 classes of ImageNet.

We decide to replace the ranking loss of SHRe with the contrastive loss introduced by CLIP [Rad+21], and explained in Section 1.5.3.2. We justify this decision with the fact that vision-language contrast has become the de-facto standard for multimodal self-supervised learning, and has lead models like CLIP [Rad+21], VLMo [Bao+22], and CoCa [Yu+22] to reach state-of-the-art results in image-text retrieval.

We apply this loss on the outputs of all three MLP layers of the shared encoder, as we want to enforce the shared encoder to generate aligned representations in all layers. The refined contrastive loss is then given by:

$$\begin{gathered}\mathcal{L}_{\mathrm{CL}} = \\ \frac{1}{3} * (\mathcal{L}_{\mathrm{CL}'} + \mathcal{L}_{\mathrm{CL}''} + \mathcal{L}_{\mathrm{CL}'''}) = \\ \frac{1}{6}\mathcal{L}_{\mathrm{CL}'}^{\mathrm{i2t}} + \frac{1}{6}\mathcal{L}_{\mathrm{CL}'}^{\mathrm{t2i}} + \\ \frac{1}{6}\mathcal{L}_{\mathrm{CL}''}^{\mathrm{i2t}} + \frac{1}{6}\mathcal{L}_{\mathrm{CL}''}^{\mathrm{t2i}} + \\ \frac{1}{6}\mathcal{L}_{\mathrm{CL}'''}^{\mathrm{i2t}} + \frac{1}{6}\mathcal{L}_{\mathrm{CL}'''}^{\mathrm{t2i}}\end{gathered} \tag{55}$$

Let's break this down: The prime ($'$) symbol defines on which outputs from Equation 51 and Equation 52 the contrastive loss is applied. Since we have three linear layers in our shared encoder, and we want to enforce alignment in the whole shared encoder, we apply the contrastive loss on all three layers, but seperately. The superscripts $\mathrm{i2t}$ and $\mathrm{t2i}$ denote if we apply the contrastive loss from image to text or from text to image, and should already

be known from Section 1.5.3.2. To sum up, we apply the contrastive loss on all three linear layers of the shared encoder, and we weight the loss equally for each layer. The contrastive loss itself weights image-to-text and text-to-image equally, which is why each component of the contrastive loss is weighted with $\frac{1}{6}$. For each contrastive loss $\mathcal{L}_{\mathrm{CL}'}$, $\mathcal{L}_{\mathrm{CL}''}$, and $\mathcal{L}_{\mathrm{CL}'''}$ we generate the matrix $\boldsymbol{L}$ from Section 1.5.3.2 once.

Since we use the contrastive loss from CLIP, we also have to define a temperature for the contrastive loss on each layer. In total, we have three temperature parameters, one for each linear layer of the shared encoder. As done in CLIP, we optimize them in log space and initialize them with 0.07 [Rad+21].

The total training objective is then given by:

$$\min \mathcal{L}_{\mathrm{KD}} + \mathcal{L}_{\mathrm{CL}} \tag{56}$$

#### 3.2.1.1.3 Training

For the teacher model we select an improved variant of the ResNet-50 [He+16], called ResNet-50-A1 [WTJ21], which has 25.6M parameters but runs in inference mode, so no gradients are computed. The model was trained on ImageNet-1K [Rus+15] and is available on HuggingFace[14].

We train the student model on all 3.3M image-text pairs we collected (Table 4) for 7 epochs, using a batch size of 256. We do not train for longer, as (1) we want to keep the training time manageable, and (2) we use a lot of pretrained components, which need less training time to converge. As done in prior experiments, we use the AdamW optimizer [LH17], and use a learning rate of $1e^{-}4$. After every epoch, we validate the model on CLIP's zero-shot image classification approach, introduced in Section 1.7.2.2, and select the best model based on the achieved accuracy. The representations for the zero-shot classification are generated by the shared encoder of the student model, which we define as $\boldsymbol{h}'''_{v,K}$ and $\boldsymbol{h}'''_{w,K}$. The classification is performed on the validation set of ImageNet-1K [Rus+15]. At this point it is important to note that the accuracy we report with CLIP zero-shot classification is actually not zero-shot. This is because the teacher model is trained supervised on ImageNet-1K, and the student model is trained using the teacher's probability distribution over the ImageNet-1K classes. Our student model therefore learns the ImageNet-1K classes directly, even though we do not train on ImageNet-1K directly. However, the accuracy we achieve still gives us a good indication of the quality of the student model's representations.

As mentioned before, we tokenize the text using the uncased BERT tokenizer. Again, uncased means that all text is lowercased before tokenization. Inspired by BEiT-3, weset the maximum text length, the length of the captions, to 64 tokens [Wan+23], truncate longer captions and pad shorter captions. This reduces the time required for a forward pass of the

[14] https://huggingface.co/timm/resnet50.a1_in1k

text encoder, and the captions of the data we collect are on average not larger than 15 tokens anyway (see Table 4).

For image augmentation, we use the same techniques as in the unimodal image KD experiment (Section 3.1.1.2). However, we make one important distinction in the size of the random crop: As seen in Table 26, the range of the random crop size is between 0.08 and 1.0 of the original image size. The lower bound is quite small, but because student and teacher receive the same crop, even if it is very small, the student can still learn the teacher's representation for a small part of an image, generated by the crop. However, this gets problematic with image text pairs. If the crop is very small, then important semantic information of the image might be lost, which is still present in the text. Therefore, the resulting probability distribution of the teacher for that small crop might not be representative of the image's high-level content, which is described by the text. This could result in the student predicting a probability distribution that is correct with respect to the whole image, but not with respect to the small crop. To avoid this, we set the lower bound of the random crop size to 0.9, which is also the value used by VLMo [Bao+22]. This ensures that the crop is large enough to capture the high-level content of the image. A visualization of too small crops is shown in Figure 37, and a visualization of a minimum crop size of 0.9 is shown in Figure 38.

All hyperparameters as summarized in Table 30, pytorch pseudocode for the forward pass can be found in Code 3.

#### 3.2.1.1.4 Results

We report the results of image-text retrieval on the test sets of COCO [Lin+14] and Flickr30k [You+14], which is shown in Table 10. Note that we do not report results on unimodal downstream tasks like image classification or text classification. This is because finetuning is expensive, which is why will refrain from doing so until we reach our final improved approach.

While the results on image-text retrieval are significantly worse than the state-of-the-art, we can still observe that we are not far off from the performance of FLAVA [Sin+21]. Considering that FLAVA was developed by a team of researchers from Meta AI, and that this is our first iteration of a multimodal model, the results are promising. From CLIP, VLMo, and BEiT-3 we are still far off, but this can at least partly be attributed to the fact that those model are significantly larger than ours, and that they have been trained on much more data. The latter is shown in Table 33. The increased performance on Flickr30K compared to COCO can be attributed to the fact that the Flickr30K dataset is smaller. For a given image, there are 5 correct captions in only 5k possible captions (25k for COCO), and for a given caption there are only 1k possible images (5k for COCO). We use the representations $\boldsymbol{h}'''_{v,K}$ and $\boldsymbol{h}'''_{w,K}$ for retrieval.

We also proof our statement of Section 1.4 that using two unrelated unimodal models, one image and one text model, for image-text retrieval fails, as the representations produced by

| 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 |
| Data2Vec2 [Bae+22] | 0.02 | 0.08 | 0.19 | 0.01 | 0.10 | 0.19 | 0.02 | 0.12 | 0.18 | 0.02 | 0.06 | 0.12 |
| 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 |
| VLMo [Bao+22] | 74.8 | 93.1 | 96.9 | 57.2 | 82.6 | 89.8 | 92.3 | 99.4 | 99.9 | 79.3 | 95.7 | 97.8 |
| BEiT-3 [Wan+23] | **84.8** | **96.5** | **98.3** | **67.2** | **87.7** | **92.8** | **98** | **100** | **100** | **90.3** | **98.7** | **99.5** |
| $\text{SHRe}_T$ (ours) | 37.06 | 67.74 | 79.7 | 25.3 | 54.73 | 68.19 | 49.9 | 78.5 | 88.5 | 37.04 | 67.34 | 77.38 |

Table 10:

the two models are not aligned. We do this by using the pretrained image and text variant from Data2Vec2 [Bae+22] as the image and text encoder, respectively. Each image is encoded by the image encoder, and each caption is encoded by the text encoder, after which the representation of the [I_CLS] and [T_CLS] token is extracted and used for retrieval according to our formulation from Section 1.5.3.2. This approach does not reach a score of more than 0.2% on any metric, and is therefore inappropriate for image-text retrieval. Consequently, we can conclude that training a multimodal model is essential for the alignment of modalities.

Transformer SHRe reaches 40.26% accuracy on the (zero-shot) ImageNet classification task, which is significantly worse that the 76.2% reached by CLIP [Rad+21], with CLIP being an actual zero-shot application (see previous section).

Since we are using the same approach as SHRe, we also provide a comparison of the average median rank on the COCO test set. How SHRe reports the average median rank is described in Section 1.7.1. Unfortunately, the authors do not provide which pairs they exacly use to calculate the average median rank. We therefore opt for an approach that should closely resemble that of SHRe: We select all 5k images from the COCO test set, and split them into 5 distinct sets of 1k images each. For each set we do the following: For each image one caption, out of the 5 captions available, is selected. We then have 5 splits of 1k image-caption pairs each. In each split, for one candidate there is only one correct target and 999 incorrect targets. We the perform retrieval on each split, and calculate the median rank of the correct target. The average median rank is then the average of the median ranks over all 5 splits. This procedure itself is repeated 5 times, so that each image is paired with each of its 5 correct captions exactly once. The result is 5 average median ranks, which are then averaged to get the metric reported in Table 11. Our approach significantly improves the baseline of SHRe, and the results indicate that the correct pair for a query is mostly either the first or second retrieved item, though the results do not account for outliers, since the median rank is used. The reason for the almost perfect results, considering the minimum possible value is 1, can be attributed to the advances in Deep Learning research since SHRe was published, which was in 2017 [AVT17]. Some of the advances include the Transformer architecture, and the use of contrastive learning. Furthermore, we can assume that the quality of the teacher

| Model | MSCOCO | |
|---|---|---|
| | Image → Text | Text → Image |
| Random | 500 | 500 |
| SHRe [AVT17] | 5.8 | 6.0 |
| $\text{SHRe}_T$ (ours) | **1.5** | **2.0** |

Table 11: Comparison of the average median rank over 5 1k image-caption splits on the COCO test set. Our approach outperforms SHRe by a large margin. The best possible value is 1.

model, which is a ResNet-50-A1 [WTJ21], is also higher than the teacher model used in SHRe, which they do not specify explicitly, but they mention AlexNet as an example teacher model. Lastly, our approach is a vision-language model, while SHRe is a vision-language-audio model, which might make the task of alignment more difficult.

As mentioned in the introduction of SHRe [AVT17], the idea of not only predicting the probability distribution over the ImageNet-1K classes for a given image, but also for a given caption, works because the ImageNet-1K classes can also be used to describe the content of a caption. The main reason for this being that the ImageNet-1K classes describe real-world objects, which are independent of the image modality, and can therefore also be used to describe the content of a text. A visualization of that is shown on image-text pairs from COCO, which is the data we use for training, in Figure 39.

#### 3.2.1.2 Larger Batch Sizes with DDP

As mentioned in the introduction of contrastive learning (Section 1.5.3.2), a large batch size is crucial for the success of contrastive learning. Larger batch sizes allow for more negative samples, which makes the task of finding the correct pair among those negatives more difficult. Unfortunately, we are not able to exceed a batch size of 256, as we run out of memory with our current GPU setup (1 $\times$ NVIDIA RTX 4090).

To overcome this limitation, we utilize Distributed Data Parallel (DDP) [Li+20], which allows us to train our model on multiple GPUs. Each GPU has its own replica (copy) of the model, and for a single forward pass, each GPU processes a different batch of the data. The forward pass and backward pass are then computed on each GPU seperately, based on the mini-batch of data it received. The resulting gradients for each replica a then aggregated across all GPUs/replicas, and each replica updates its weights based on the aggregated gradients. We are therefore able to increase the batch size by the number of GPUs we use, and update the weights of the model based on gradients that have been computed on a larger batch size than a single model received in a single forward pass [Li+20]. An illustration of DDP with 2 GPUs is shown in Figure 40.

A side effect of DDP is that after the forward pass of each replica, there now exists not just a single batch of image-text representations, but as many batches as there are replicas/GPUs. Those representations can, similar to the gradient accumulation, be communicated across all replicas. If we use 2 GPUs with a batch size of 256, then all representations on the first GPU are communicated to the second GPU, and vice versa. Consequently, we now have 512 image-text pairs on which the contrastive loss is computed, which is equivalent to using a batch size of 512 on a single GPU.

This not only increases the effectiveness of the contrastive loss, but also reduces the time required to train the model. This is because when using DDP, the whole dataset is split across all GPUs (see Figure 40), and each GPU always only processes its own part of the dataset. Since the forward pass on each GPU is done in parallel, and the number of steps per epoch is reduced by a factor equal to the number of GPUs used, the training time is reduced. Even though the time required for training the model will not exactly be reduced by the factor of the number of GPUs used, as the distributed communication between the GPUs introduces overhead, it is still significantly faster than training on a single GPU. From a cost perspective, multiple GPUs are obviously more expensive than a single GPU, but the time saved by using DDP outweighs the increased cost to some extent. This is especially important considering that we can traing more models in a shorter amount of time, which allows us to iterate faster.

While it is tempting, also from a technical perspective, to use as many GPUs as possible, we refrain from doing so. This is because we do not want to make the success of our approach too dependent on the hardware we use. After all, the goal is to develop an approach that is feasible even for researchers with limited resources. We therefore limit the number of GPUs to 2, effectively increasing the batch size to 512. The architecture, loss, and training procedure remain the same as described in the previous sections.

As shown in Table 12, introducing DDP with a second GPU lead to an absolute gain of more than 4% on the COCO and Flickr30K retrieval tasks, while almost reaching perfect performance on the average median rank task of SHRe [AVT17]. This underlines the importance of a large batch size for contrastive learning. However, we observe no improvement on the (zero-shot) ImageNet classification task. We hypothesize that the classification task is more dependent on the quality of the text-based class prototypes, which are generated from predefined textual descriptions of the class labels (see Section 1.7.2.2). The quality of these prototypes, which are fixed representations of each class, is not directly influenced by the batch size used during training. In contrast, retrieval tasks rely on contrastive learning, where having a higher diversity of negative samples during training helps the model better distinguish between relevant and irrelevant image-text pairs. As a result, an increase in batch size translates more directly into better retrieval performance, while its effect on zero-shot classification is less pronounced.

The training time, displayed in Table 13 behaves as expected: The wall clock time per batch (the time needed for one full training step, including weight updates) is higher when using

| Approach | COCO Retrieval | | | Flickr30K Avg. R | ImageNet classification |
|---|---|---|---|---|---|
| | Avg. R | AMR-I2T | AMR-T2I | | |
| Default | 55.45 | 1.5 | 2.0 | 66.44 | **40.26** |
| DDP | **59.59** | **1.0** | **1.04** | **70.78** | 40.0 |

Table 12: We report the average R@1, R@5, and R@10 values for image-text and text-image retrieval on the COCO and Flickr30K test sets (Avg. R). Further, the average median rank (AMR) for image-to-text (I2T) and text-to-image (T2I) retrieval on COCO, and the performance on CLIP-like ImageNet classification is listed. DDP improves the performance of the model on the COCO and Flickr30K retrieval tasks, but not on (zero-shot) ImageNet classification.

| Approach | Wall Clock Time / Batch (ms) | Training duration (h) |
|---|---|---|
| Default | **331** | 8.2 |
| DDP | 387 | **4.8** |

Table 13: Comparison of the wall clock time per batch and the total training duration for the default and DDP approach. The training duration is calculated for 7 epochs on the 3.3M image-text pairs we collected. While the wall clock time per batch is higher for the DDP approach, the total training duration is significantly lower.

DDP, but the total training duration is significantly lower. Again, the latter is due to the fact that the number of steps per epoch is reduced by a factor equal to the number of GPUs used, which reduces the total training time.

#### 3.2.1.3 Shared Transformer Encoder

We prepend the name of our approach with "Transformer" to indicate that we use a Transformer architecture for both the image and text encoder. However, this does not hold for the shared encoder, which is, like the original approach, a 3-layer MLP [AVT17]. We now experiment with also replacing the shared encoder with a Transformer, leading to a fully Transformer-based model. We motivate this decision with the architecture of VLMo and BEiT-3, which both use Transformer layers towards the end of the model [Bao+22, Wan+23].

While our shared encoder would then mimic the architecture of VLMo and BEiT-3 in the sense that is uses Transformer layers that process both image and text representations, there is still a significant difference to those models: The upper two Transformer layers, which are the shared ones, of VLMo and BEiT-3 do not process image and text representations separately, but receive a shared representation of an image-text pair. This is created by simply concatenating the image and text representations, and passing them through the Transformer layers [Bao+22, Wan+23]. The input to such a layer $l$ is then given by:

$$\boldsymbol{H}_{\mathrm{vw},l-1} = \left[\boldsymbol{H}_{w,l-1}; \boldsymbol{H}_{v,l-1}\right] \tag{57}$$

This allows the Self-Attention mechanism to attend between individual image and text tokens, leading to a more fine-grained alignment. In contrast, our shared encoder will, just like the shared 3-layer MLP before, process image and text representations separately. The input will therefore either be a single image representation $\boldsymbol{H}_{v,l-1}$, returned by the image encoder, or a single text representation $\boldsymbol{H}_{w,l-1}$, returned by the text encoder.

We hypothesize that our approach might be more challenging to learn, compared to VLMo and BEiT-3's concatenated strategy, due to the following reasons:

- In VLMo and BEiT-3, the input is always a combined image-text representation, ensuring consistency for the Self-Attention mechanism, which can then concentrate solely on fine-grained alignment.
- In our case, the Self-Attention mechanism receives either an image sequence or a text sequence independently, requiring it to not only perform attention but also infer the modality of the input. Since image and text inputs are inherently different, this adds a layer of complexity.

Nevertheless, VLMo demonstrates that this modality-specific complexity may not be an issue. In VLMo, masked language modeling is performed using a Transformer whose Self-Attention weights are frozen and from a pretrained image model, yet the model still effectively handles text-only pretraining [Bao+22].

Given that VLMo can successfully leverage frozen image Self-Attention weights for text tasks, we believe that our approach, where the Self-Attention weights are explicitly trained to work with both image and text inputs, should also be effective, if not more so.

The change in the architecture does not imply a change in the training procedure, loss, or hyperparameters. In fact, the only thing we actually add is one Multi-Head Self-Attention layer to the shared encoder (plus the two residual connections). Recall that we implemented the 3-layer MLP as a 2-layer MLP network as used in Transformer layers, and added an additional LayerNorm and linear layer on top of it (see Section 3.2.1.1.1). Replacing this with a single Transformer layer means we still have the 2-layer MLP network, but now also add a Multi-Head Self-Attention layer before it. Since we are still predicting a probability distribution over the ImageNet classes (the one returned by the teacher), which can be seen as a type of classification task, we still need a classification head on top of the Transformer layer to output logits for the 1000 classes. Fittingly, this is exactly the task of the third MLP layer we had in the shared 3-layer MLP. The number of neurons for each linear layer also remains the same: The first linear layer expands the hidden dimension of the Transformer from 768 to 3072, and the second linear layer reduces it back to 768. The third linear layer then expands it to 1000, the number of classes in ImageNet. In Figure 18, where we illustrate our previous explanation, we indicate the difference in dimensionality between the linear layers by a different breadth.

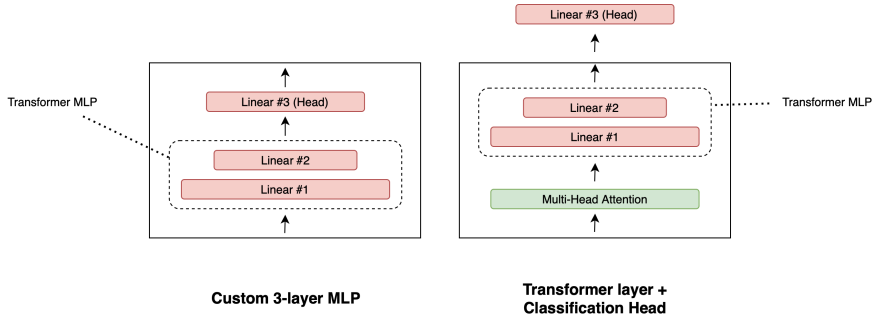


Figure 18: Switching from our implementation of the shared encoder as a 3-layer MLP to a Transformer layer only corresponds to adding a Multi-Head Self-Attention layer before the linear layers. The dimensionality of the linear layers remains the same. On the right, layer normalization, activations, and residual connections are omitted for simplicity.

Since the shared encoder is now can actual Transformer layer plus the classification head, we redefine our previous forward pass of the shared encoder from Equation 51 and Equation 52 to the following (also on the example of the text modality):

$$\boldsymbol{H}^{\text{mha}}_{w,K} = \text{MHA}\Big(\text{LN}\Big(\boldsymbol{H}_{w,L_s}\Big)\Big) + \boldsymbol{H}_{w,L_s} \tag{58}$$

$$\begin{aligned}
\boldsymbol{H}'_{w,K} &= \text{LN}\Big(\boldsymbol{H}^{\text{mha}}_{w,K}\Big)\boldsymbol{W}_1 + \boldsymbol{b}_1 \\
\boldsymbol{H}''_{w,K} &= \text{LN}\big(\text{GELU}\big(\boldsymbol{H}'_{w,K}\big)\big)\boldsymbol{W}_2 + \boldsymbol{b}_2 \\
\boldsymbol{H}^{\text{ln}}_{w,K} &= \text{LN}\Big(\boldsymbol{H}''_{w,K} + \boldsymbol{H}^{\text{mha}}_{w,K}\Big)
\end{aligned} \tag{59}$$

$$\boldsymbol{h}'''_{w,K,[\texttt{T\_CLS}]} = \boldsymbol{h}^{\text{ln}}_{w,K,[\texttt{T\_CLS}]}\boldsymbol{W}_3 + \boldsymbol{b}_3 \tag{60}$$

It holds that $K = L_s + 1 = 7$, since the shared encoder is one additional Transformer layer. The subscripts for the weights $\boldsymbol{W}$ and biases $\boldsymbol{b}$ denote to which linear layer from Figure 18 (right) they belong. Correspondingly, the output of the equation where a pair of $(\boldsymbol{W}_i, \boldsymbol{b}_i)$ is used, is also the output of the linear layer $i$. Equation 58 is the usual operation of the Multi-Head Self-Attention layer known from the ViT architecture, and is therefore also shown in Equation 21 of the section on Vision Transformers (Section 1.3.2).

Equation 59 shows the operation of the Feed-Forward Network (FFN) layer of the Transformer. It is the same operation as defined in Equation 51, but applied pointwise on a sequence. We divide the operations of the FFN into three formulas, as we want to explicitly show that we extract the intermediate representation $\boldsymbol{H}'_{w,K}$ and $\boldsymbol{H}'_{v,K}$, and final representation $\boldsymbol{H}''_{w,K}$ and $\boldsymbol{H}''_{v,K}$ from the FFN. Both the intermediate and final representations we extract are the raw outputs from linear layer #1 and linear layer #2, without any activation function or normalization applied. From those representations, which are still sequences, we take the representation of the [T_CLS] and [I_CLS] token for text and image, respectively.

Concretely, they are: $\boldsymbol{h}'_{w,K,[\texttt{T\_CLS}]}$ and $\boldsymbol{h}'_{v,K,[\texttt{I\_CLS}]}$, and $\boldsymbol{h}''_{w,K,[\texttt{T\_CLS}]}$ and $\boldsymbol{h}''_{v,K,[\texttt{I\_CLS}]}$. Those are then used for the contrastive loss.

The operation in Equation 60 now resembles that of an actual classification head from the vision Transformer architecture, and is only applied on the [I_CLS] or [T_CLS] token. An important distinction we make here, compared to the original shared 3-layer MLP, is that we do not apply the contrastive loss on the output of the classification head $\boldsymbol{h}'''_{w,K,[\texttt{T\_CLS}]}$ and $\boldsymbol{h}'''_{v,K,[\texttt{I\_CLS}]}$, and we explain the reasoning for this in the following.

The task of the classification head is to leverage the knowledge learned from the teacher to provide the student with guidance. This guidance can be described as the fact that objects, or rather the ImageNet-1K classes that describe them, can be found in both the image and its caption (see Figure 39). The student will therefore learn that an image and its caption both describe the same real-world object/concepts. Based on this intuition, the classification head is not meant to output representations of the image or text, and should therefore not be used for retrieval tasks.

For all retrieval tasks, including CLIP-like zero-shot classification, we consequently use the representations $\boldsymbol{h}''_{v,K,[\texttt{I\_CLS}]}$ and $\boldsymbol{h}''_{w,K,[\texttt{T\_CLS}]}$ as final representations for image and text, respectively. The classification head can therefore be discared after training, as it is not needed for retrieval or any other downstream task. It is only used during training to provide the student with guidance from the teacher. An overview which tokens are used in which part of the training objective is shown in Table 14.

The full contrastive loss is now:

$$\begin{aligned} &\mathcal{L}_{\text{CL}} = \\ &\frac{1}{2} * (\mathcal{L}_{\text{CL}'} + \mathcal{L}_{\text{CL}''}) = \\ &\quad \frac{1}{4}\mathcal{L}^{\text{i2t}}_{\text{CL}'} + \frac{1}{4}\mathcal{L}^{\text{t2i}}_{\text{CL}'} + \\ &\quad \frac{1}{4}\mathcal{L}^{\text{i2t}}_{\text{CL}''} + \frac{1}{4}\mathcal{L}^{\text{t2i}}_{\text{CL}''} \end{aligned} \tag{61}$$

#### 3.2.1.4 Results

The influence on retrieval, when adding a shared Transformer layer and changing the representations used for retrieval, can be seen in Table 15. This change not only outperforms our first two experiments in all metrics, but also FLAVA [Sin+21] in R@1 text retrieval on COCO. For other metrics on COCO, we are also surprisingly close to FLAVA. On Flickr30K, we still lack behind FLAVA, but the gap is significantly smaller than before.

We are also able to increase the performance on CLIP-like ImageNet-1K classification from 40.26% to 44.57%, and reach a perfect average median rank for both text and image retrieval on the COCO test set, which is 1.0. As a reference: The previous scores were 1.0 for text

| Loss | Modality | SHRe | Transformer SHRe |
|---|---|---|---|
| $\mathcal{L}_{\text{KD}}$ | Image | $\boldsymbol{h}'''_{v,K}$ | $\boldsymbol{h}'''_{v,K,[\texttt{I\_CLS}]}$ |
| | Text | $\boldsymbol{h}'''_{w,K}$ | $\boldsymbol{h}'''_{w,K,[\texttt{T\_CLS}]}$ |
| $\mathcal{L}_{\text{CL}'}$ | Image | $\boldsymbol{h}'_{v,K}$ | $\boldsymbol{h}'_{v,K,[\texttt{I\_CLS}]}$ |
| | Text | $\boldsymbol{h}'_{w,K}$ | $\boldsymbol{h}'_{w,K,[\texttt{T\_CLS}]}$ |
| $\mathcal{L}_{\text{CL}''}$ | Image | $\boldsymbol{h}''_{v,K}$ | $\boldsymbol{h}''_{v,K,[\texttt{I\_CLS}]}$ |
| | Text | $\boldsymbol{h}''_{w,K}$ | $\boldsymbol{h}''_{w,K,[\texttt{T\_CLS}]}$ |
| $\mathcal{L}_{\text{CL}'''}$ | Image | $\boldsymbol{h}'''_{v,K}$ | - |
| | Text | $\boldsymbol{h}'''_{w,K}$ | - |

Table 14: A comparison between the tokens used in the loss functions for the approach of SHRe [AVT17] with a shared 3-layer MLP and out Transformer SHRe. For Transformer SHRe we do not use the contrastive loss $\mathcal{L}_{\text{CL}'''}$.

| 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** | - |
| Baseline | 37.06 | 67.74 | 79.7 | 25.3 | 54.73 | 68.19 | 49.9 | 78.5 | 88.5 | 37.04 | 67.34 | 77.38 |
| + DDP (↑ 4.24) | 41.88 | 71.86 | 83.06 | 29.4 | 59.3 | 72.06 | 56.1 | 83.5 | 90.7 | 41.80 | 71.36 | 81.14 |
| + Shared $T$ (↑ 5.68) | **44.6** | 75.3 | 85.64 | 31.69 | 62.1 | 74.48 | 58.8 | 85.6 | 92.4 | 43.92 | 74.06 | 82.8 |

Table 15: The overview of retrieval results on the COCO and Flickr30K test sets show significant improvements when using DDP and a shared Transformer encoder. We achieve an absolute gain of almost 7% on the COCO test set, and more than 6% on the Flickr30K test set, compared to our first approach, which we denote as "Baseline". We add FLAVA [Sin+21] as a reference.

retrieval, and 1.04 for image retrieval (see Table 12). While this is indeed the perfect score for the average median rank, it is important to note that we use the *median* rank, and a value of 1.0 therefore means that in at least half of all retrievals (so $\geq 500$) the correct pair is the first retrieved item. This does not account for outliers, and is only applied on a small subset of 1k image-caption pairs. Nevertheless, this is the benchmark provided by SHRe [AVT17], so a perfect score is still worth mentioning. Unless there will be a decrease in performance on the average median rank, we will refrain from reporting it in the future, as it is not a very informative metric and we have already reached the best possible score. A visualization of the retrieved samples for captions (image retrieval) and images (text retrieval) is shown in Figure 41 and Figure 42, respectively.

### 3.2.2 Self-Supervised Teacher

#### 3.2.2.1 Challenges of Self-Supervision

With the architecture and the first benchmarks in place, we can now focus on the main goal of this research: Applying the idea of SHRe [AVT17] using a self-supervised teacher model. A consequence of a self-supervised teacher is that the teacher's prediction for a given image is not a probability distribution over a set of classes, but merely a representation of the input sample. This raises the question which training objective to use when using a self-supervised teacher, as we cannot use the KL-Divergence loss anymore. This only works on probability distributions, and not on representations.

In preliminary experiments on unimodal knowledge distillation, a self-supervised teacher did not pose a problem, as both the teacher and student received the same input, and the latter was able to regress all time steps of the teacher model. However, this was only possible because the teacher and student received exactly the same input: For a given time step, the patch or text token at that time step was the same for both models, allowing the student to learn the teacher's representation for each patch or text token, respectively. Since we predict the output of an image teacher model, in which ever form it may be, the aforementioned still holds true when the multimodal model receives an image as the input. The output will be a representation for each image patch, which is also the prediction of the teacher model, allowing for the same approach used in unimodal knowledge distillation (see Section 3.1.1).

When the multimodal model receives a text (an image's caption) as the input however, the teacher's prediction is still a representation of the image. This poses the following problems:

1. The number of patches (and therefore time steps) in an image is usually not the same as the number of text tokens of its corresponding caption. Consequently, we do not have a one-to-one correspondence between the time steps of the image and text models, and the student cannot regress every time step of the teacher model.
2. Even if the number of time steps were the same, the content of the time steps would not be aligned: The text token at a time step does not necessarily correspond to the content of the image patch at the same time step. Moreover, text naturally contains fill words such as "the", "a", "is", which do not have any meaning with respect to the content of an image. Another example are padding tokens, which do not contain any information at all, and are merely used for batching a set of texts.

Consequently, regressing the representation of individual patches when the multimodal student model receives a text as the input is not possible, and we have to resort to regressing the global representation of the image, which is the `[I_CLS]` token. This choice solves both of the aforementioned problems, as we do not rely on the representations of individual time steps. To clarify, the concept of the forward pass remains the same as in SHRe [AVT17] and our Transformer variant of the previous chapter (Section 3.2.1): For a single image-text pair, the image can be passed to the teacher, and the image and its caption can be passed to the student seperately. When we focus on the global representation (the cls token) returned, the

teacher will always return a representation of the [I_CLS] token, aggregating global information of the image. The same holds true for the student, producing a representation of the [I_CLS] token. As a training objective we can then require:

$$\min_{\boldsymbol{h}^s_{v,K,[\texttt{I\_CLS}]}} \|\boldsymbol{h}^s_{v,K,[\texttt{I\_CLS}]} - \boldsymbol{h}^t_{v,L_t,[\texttt{I\_CLS}]}\| \tag{62}$$

We denote a representation generate by the student with the superscript $s$, and the teacher with the superscript $t$. The objective forces the student to push its representation of the [I_CLS] token as close as possible to the teachers representation of the same token. Most importantly, the student can also be trained to push the representation of the caption, given by the [T_CLS] token, as close as possible to the teacher's representation of the image:

$$\min_{\boldsymbol{h}^s_{w,K,[\texttt{T\_CLS}]}} \|\boldsymbol{h}^s_{w,K,[\texttt{T\_CLS}]} - \boldsymbol{h}^t_{v,L_t,[\texttt{I\_CLS}]}\| \tag{63}$$

An illustration of the problem posed by the misalignment of time steps and the solution of regressing the global representation of the image is shown in Figure 19.

The combined training objective when regressing only global information is then:

$$\min_{\boldsymbol{h}^s_{v,K,[\texttt{I\_CLS}]},\boldsymbol{h}^s_{w,K,[\texttt{T\_CLS}]}} \|\boldsymbol{h}^s_{v,K,[\texttt{I\_CLS}]} - \boldsymbol{h}^t_{v,L_t,[\texttt{I\_CLS}]}\| + \|\boldsymbol{h}^s_{w,K,[\texttt{T\_CLS}]} - \boldsymbol{h}^t_{v,L_t,[\texttt{I\_CLS}]}\| \tag{64}$$

While this objective in theory forces the student to output the same global representation for an image and its caption as the teacher, it requires the teacher to produce a representation of the [I_CLS] token that is abstract enough to also describe the content of the caption. Conretely, the representation of the [I_CLS] token should **not** contain any image-specific information, as this would make it impossible for the student to align the representation of the caption with that of the image: It is not possible to extract any image-specific information, like the exact position of an object in the image, from the caption. Consequently, whether the representation of the [I_CLS] token produced by the teacher is abstract enough to also describe the content of the caption remains to be seen in the following experiments.

The challenges that come with the choice of a self-supervised teacher raises the question why we do not directly use a multimodal model as the teacher. This reason behind this choice can be attributed to the fact that the goal of this research is to train a multimodal model without using any existing, especially pretrained, *multimodal* components. Instead, we aim to extract knowledge from purely unimodal models and learn to generate modality-invariant features from it (1), and to not rely on labeled data in the end-to-end training of the multimodal model (2). This includes the teacher model, which should not be trained on labeled data, but only self-supervised.

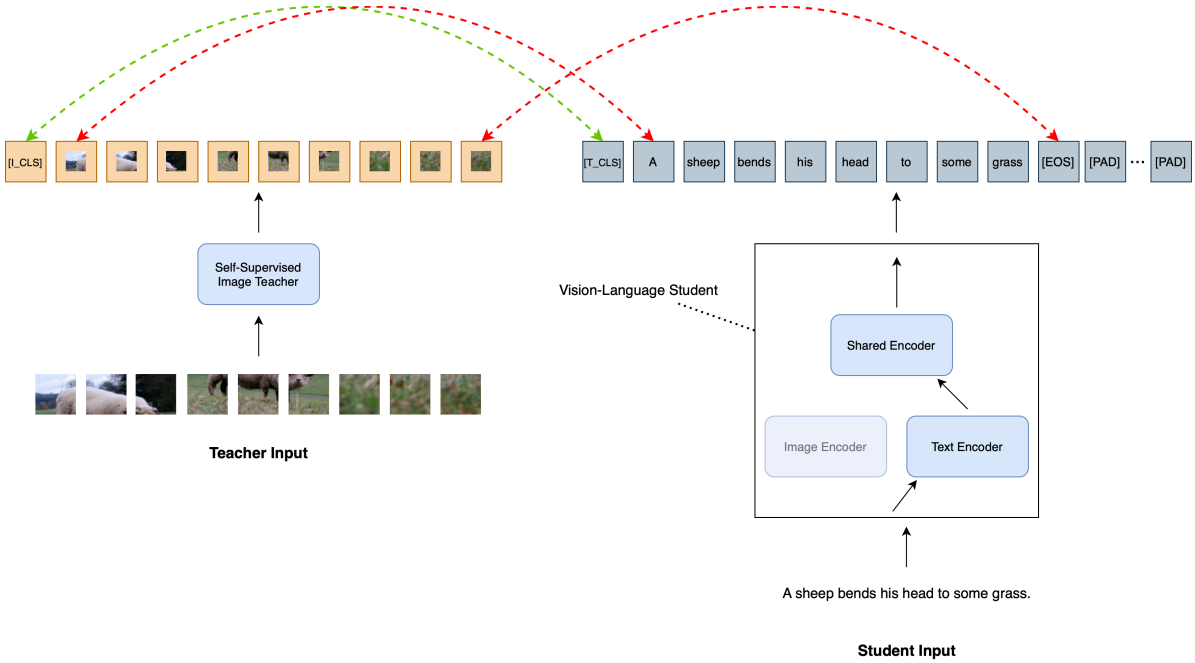


Figure 19: The meaning/content of time steps across modalities is not aligned, and the number of time steps will differ between modalities. This makes alignment on the level of individual time steps impossible. The cls token aggregates global information independent of time steps, and captures the meaning and interpretation of the respective input, making alignment possible. However, this requires the teacher cls token to not contain any modality-specific (in this case image) information. Image-Text example is taken from the COCO train set [Lin+14].

#### 3.2.2.2 Feature-based Knowledge Distillation

Based on the previous discussion, we propose an adjustment to the loss used to train the multimodal student. We formulate the knowledge distillation loss to a feature-based loss, which implements the training objective discussed in the previous section:

$$\begin{aligned} \mathcal{L}_{\text{KD}} = \\ \frac{1}{2} * \mathcal{L}^{v}_{\text{KD}} + \frac{1}{2} * \mathcal{L}^{w}_{\text{KD}} = \\ \frac{1}{2} * \|\boldsymbol{h}'''^{s}_{v,K,[\texttt{I\_CLS}]} - \boldsymbol{h}^{t}_{v,L_t,[\texttt{I\_CLS}]}\|_2^2 + \frac{1}{2} * \|\boldsymbol{h}'''^{s}_{w,K,[\texttt{T\_CLS}]} - \boldsymbol{h}^{t}_{v,L_t,[\texttt{I\_CLS}]}\|_2^2 \end{aligned} \tag{65}$$

The loss is the mean of the mean squared error between the student's and teacher's representation of the [I_CLS] token, and between the student's representation of the [T_CLS] token and the teacher's representation of the [I_CLS] token.

The target representation $\boldsymbol{h}^{t}_{v,L_t,[\texttt{I\_CLS}]} \in \mathbb{R}^{768}$ stems from a new, now self-supervised, teacher model, which is BEiTv2 [Pen+22]. Since BEiTv2 is a Transformer-based self-supervised image model it also returns a sequence of patch representations, and we extract the represen-

| 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** |
| $\text{SHRe}_{\text{T}}$ | 44.6 | 75.3 | 85.64 | 31.69 | 62.1 | 74.48 | 58.8 | 85.6 | 92.4 | 43.92 | 74.06 | 82.8 |
| S-SMKE (↑ 4.38) | 51.66 | 79.9 | **88.66** | 36.17 | 66.55 | **78.28** | 64.5 | 88.4 | 93.0 | 51.78 | 78.54 | 86.46 |

Table 16: An updated contrastive loss with a self-supervised teacher, which we denote as "S-SMKE" leads to an average gain of 4.38% compared to our previous approach $\text{SHRe}_{\text{T}}$. On Flickr30K, we reach substantial improvements while still being behind FLAVA [Sin+21] and CLIP [Rad+21].

tation $\boldsymbol{h}^{t}_{v,L_t,[\texttt{I\_CLS}]}$ of the [I_CLS] token from the output sequence. The teacher has $L_t = 12$ Transformer layers, and is based on the ViT-B/16 architecture. It therefore is with almost 86M parameters significantly larger than the previous ResNet-50-A1 [WTJ21] teacher model, which had 25M parameters. The teacher being based on the ViT-B/16 architecture also means that the dimensionality of the output representations is 768.

The architecture of the student remains largely same as before, the prediction of the student for the [I_CLS] token of the teacher, which is the global image represenation, is generated by the student's classification head. A consequence of this is that the classification head now also outputs 768-dimensional representations, instead of the 1000-dimensional output used previously for the ImageNet-1K classes. Therefore: $\boldsymbol{h'''}^{s}_{v,K,[\texttt{I\_CLS}]} \in \mathbb{R}^{768}$ and $\boldsymbol{h'''}^{s}_{w,K,[\texttt{T\_CLS}]} \in \mathbb{R}^{768}$. As a side note, the student's classification head is technically not a classification head anymore, as it does not predict any classes but rather a representation of the teacher. We will refer to it as the regression head from now on.

#### 3.2.2.3 Results

Apart from the change in the teacher and the impact on the loss, the training setup remains the same, and no hyperparameters are changed. As we now do not follow the approach of SHRe, that is, predicting a probability distribution over ImageNet-1K classes [AVT17], we name the new approach "**S**elf-**S**upervised **M**ultimodal **K**nowledge **E**xtraction" (S-SMKE).

Surprisingly, the results show a significant improvement in retrieval performance compared to the previous benchmarks, as shown in Table 16. We achieve competetive performance with FLAVA [Sin+21] and CLIP [Rad+21] on COCO, and even outperform CLIP on COCO R@10 image retrieval by more than 6 percentage points.

Less surprisingly, we observe a drop in ImageNet-1K classification performance of more than 14 percentage points compared to the supervised teacher approach SHRe [AVT17], as shown in Table 17 (left). We consider this as not surprising, because we previously used representations $\boldsymbol{h'''}^{s}_{v,K,[\texttt{I\_CLS}]}$ and $\boldsymbol{h'''}^{s}_{w,K,[\texttt{T\_CLS}]}$ for the classification, which were also the representa-

| Approach | Accuracy |
|---|---|
| Visual N-Grams [Li+17] † | 11.5 |
| CLIP [Rad+21] † | **76.2** |
| $\text{SHRe}_\text{T}$ (ours) | 44.57 |
| S-SMKE (ours) | 30.4 |

| Approach | Training time (h) |
|---|---|
| $\text{SHRE}_\text{DDP}$ | **4.8** |
| $\text{SHRe}_\text{T}$ | 5.27 |
| S-SMKE | 6.9 |

Table 17: (Left) Accuracy of S-SMKE on the validation set of ImageNet-1k, using CLIP zero-shot classification, compared to CLIP [Rad+21] and Visual N-Grams [Li+17]. The latter was developed as a first proof-of-concept of zero-shot image classification in 2017 [Li+17, Rad+21]. We denote a full zero-shot approach with †. (Right) Development of the training time for the different approaches. Adding additional parameters through Self-Attention ($\text{SHRe}_\text{T}$) and a large self-supervised teacher (S-SMKE) increases the training time by more than two hours.

tions used for predicting the ImageNet-1K classes as given by a supervised teacher trained on ImageNet-1K. Therefore, those representations were optimized for predicting ImageNet-1K classes. Now, we regress the teacher's representation of the [I_CLS] token, and since the teacher was not trained on the ImageNet-1K classes, the student's representations do not contain any information about them. This leads to a drop in classification performance.

While we now train the student without the ImageNet-1K classes, the CLIP-like zero-shot classification still cannot be considered as zero-shot. This can be attributed to the fact that, although trained without the classes/labels, our teacher model BEiTv2 [Pen+22] has been trained on ImageNet-1K. The knowledge contained in the teacher's representations of the [I_CLS] token could therefore leak information about the images of ImageNet-1K to the student. In order for it to be a true zero-shot approach, the teacher would have to be trained on a dataset that is disjoint from ImageNet-1K.

Speaking of the teacher, its increased size leads to a longer training time, as shown in Table 17 (right), which would be even more pronounced if we wouldn't use DDP with 2 GPUs.

### 3.2.3 Token-Type Embeddings

In order for the shared Transformer approach to work, the linear layers and Self-Attention mechanism in the shared Transformer layer need to be able to somewhat differentiate between both modalities (image and text). Even though we desire an aligned representation to form in the shared block, especially the Self-Attention mechanism still needs to be able to differentiate between the two modalities. This can be explained by the fact that for images the model needs to find 2-dimensional spatial relationships, while for text only 1-dimensional relationships are required. Even though we learned from the previous experiments that the shared Transformer block is able to learn a good representation without any modality-specific information, we still want to investigate how the performance of the model changes when we explicitly provide the shared Transformer block with the information

| TTE | ImageNet-1K | Retrieval | |
|---|---|---|---|
| | | MSCOCO | Flickr30K |
| × | **30.4** | **66.87** | **77.1** |
| ✓ | 29.6 | 66.0 | 76.5 |

Table 18: Introducing token-type embeddings (TTE) after the positional encoding degrades performance slightly. We compare to the previous model, which does not use TTE.

which modality it is processing. This nessessitates a special embedding for both image and text, which is added to each token of the respective modality, even the special tokens. This is called a token-type embedding (TTE) [Bao+22], and is also used in multimodal models such as VLMo [Bao+22].

The intuitive approach to implement this would be to directly follow VLMo and add the token type embeddings after the positional encoding, before the input is fed into the Transformer blocks [Bao+22], which corresponds to our modality-specific encoders. Our definition of the Transformer input changes as follows for text:

$$\begin{aligned} \boldsymbol{H}_{w,0} &= \left[\boldsymbol{h}_{w,0,[\texttt{T\_CLS}]}, \boldsymbol{h}_{w,0,1}, ..., \boldsymbol{h}_{w,0,M}, \boldsymbol{h}_{w,0,[\texttt{T\_SEP}]}\right] = \boldsymbol{E}_w + \boldsymbol{T}_w^{\text{pos}} + \boldsymbol{T}_w^{\text{type}} \\ \boldsymbol{T}_w^{\text{type}} &= \left[\boldsymbol{t}^w_{\text{type}_{[\texttt{T\_CLS}]}}, \boldsymbol{t}^w_{\text{type}_1}, ..., \boldsymbol{t}^w_{\text{type}_M}, \boldsymbol{t}^w_{\text{type}_{[\texttt{T\_SEP}]}}\right] \end{aligned} \tag{66}$$

Similar holds for images:

$$\begin{aligned} \boldsymbol{H}_{v,0} &= \left[\boldsymbol{h}_{v,0,[\texttt{I\_CLS}]}, \boldsymbol{h}_{v,0,1}, ..., \boldsymbol{h}_{v,0,N}\right] = \boldsymbol{E}_v + \boldsymbol{T}_v^{\text{pos}} + \boldsymbol{T}_v^{\text{type}} \\ \boldsymbol{T}_v^{\text{type}} &= \left[\boldsymbol{t}^v_{\text{type}_{[\texttt{I\_CLS}]}}, \boldsymbol{t}^v_{\text{type}_1}, ..., \boldsymbol{t}^v_{\text{type}_N}\right] \end{aligned} \tag{67}$$

The token type embedding is the same for every token and patch in the sequence, respectively.

$$\begin{aligned} \boldsymbol{t}^w_{\text{type}_i} = \boldsymbol{t}^w_{\text{type}_j}, \forall i, j \in \{[\texttt{T\_CLS}], 1, ..., M, [\texttt{T\_SEP}]\} \\ \boldsymbol{t}^v_{\text{type}_i} = \boldsymbol{t}^v_{\text{type}_j}, \forall i, j \in \{[\texttt{I\_CLS}], 1, ..., N\} \end{aligned} \tag{68}$$

This follows VLMo [Bao+22] and is because each token/patch of a text/image input sequence is of the same modality. The parameters added to the model are negligible, as they only include two additional embeddings of size $D$, with $D = 768$ this accounts for just $768 * 2 = 1536$ parameters.

We present the results in Table 18, and show that the variant with TTE does not improve the performance of the model.

We suspect that this is due to two reasons. First, the the TTE is added before the modality-specific encoders, so the image and text encoder. Their task is to extract features from the input, independent of the **other** respective modality. Consequently, TTE is of no use

to them, as the same embedding is added to every token/patch, and both encoders do not need to differentiate between image and text in their input: The image encoder will always receive an image, and the text encoder always a text. Second, even worse, we use pretrained layers for the image and text encoder, which already extract rich features from the input. Adding the same token-type embedding to their input, initialized randomly, will destroy the patch (token) embeddings the image (text) encoder receives as its input, and thus the feature extraction will be less effective. Even though we are also training the pretrained image and text encoder, it will take time until the encoders learn to adapt to the TTE, and until the TTE has been trained in itself.

We therefore opt for the following change: We add the TTE after the modality-specific encoders, meaning the token type embedding is added to the output of the image and text encoder. This way, the TTE will not disturb the feature extraction of the image and text encoder. However, what will inevitably happen is that adding the TTE after the modality-specific encoders will destroy the features extracted by the encoders, which is exactly the second problem mentioned before, just at a different stage in the model.

A possible solution can be found in the Transformer block of extremely deep and large models, such as BEiT-3 [Wan+23]. Here, each embedding dimension of the output generated by the Self-Attention and MLP in a Transformer layer is multiplied with a seperate, learnable, scalar. What makes this approach special is that the scalars are all initialized with values close to zero. As this multiplication is done before the residual connection, i.e. the addition of the input of the Self-Attention and MLP respectively, the contribution of the Self-Attention and MLP to the output of the Transformer block is very small at the beginning of training. What follows is that the initial input to the model is carried very far through the model, and the actual contribution of the Self-Attention and other parameters is added gradually as the model lears to extract meaningful features from the input [Tou+21].

We will use LayerScale not for the purpose it was intended for, that is to allow training of extremely deep models [Tou+21], but to allow the TTE to be added gradually after the modality-specific encoders, without destroying the features extracted by the encoders. Before the TTE is added to the output sequence of the image and text encoder, we multiply each dimension of the TTE with its own learnable scalar, as given by LayerScale. We initialize the weights of the LayerScale to 1e-5, so that each dimension of the TTE is multiplied with a very small scalar, and the contribution of the TTE, when adding it to the output of the image and text encoder, is very small. This way, the shared Transformer block will receive the almost unaltered features from the encoders, and since the scale is learned, the TTE will be added as the models sees fit - as much as it helps the model to learn. Implementing LayerScale is straightforward, the representation of each token/patch in the output of the image and text encoder is multiplied element-wise ($\odot$) with the LayerScale embedding $\boldsymbol{t}_{\text{scale}}$:

| TTE | LayerScale | ImageNet-1K | Retrieval | |
|---|---|---|---|---|
| | | | MSCOCO | Flickr30K |
| × | × | **30.4** | 66.87 | 77.1 |
| ✓ | × | 30.3 | **67.2** | 78.08 |
| ✓ | ✓ | 30.3 | **67.2** | **78.29** |

Table 19: Comparison of S-SMKE with different TTE and LayerScale settings **after** the modality-specific encoders. We observe that adding TTE and LayerScale especially improves performance on the retrieval tasks, with Flick30K showing an absolute gain of 1.19%.

$$\boldsymbol{H}'_{w,L_s} = \boldsymbol{H}_{w,L_s} + \boldsymbol{t}_{\text{scale}} * \boldsymbol{T}^{w}_{\text{type}} = \boldsymbol{H}_{w,L_s} + \left[\boldsymbol{t}_{\text{scale}} \odot \boldsymbol{t}^{w}_{\text{type}_{[\texttt{T\_CLS}]}}, \boldsymbol{t}_{\text{scale}} \odot \boldsymbol{t}^{w}_{\text{type}_1}, ..., \boldsymbol{t}_{\text{scale}} \odot \boldsymbol{t}^{w}_{\text{type}_M}, \boldsymbol{t}_{\text{scale}} \odot \boldsymbol{t}^{w}_{\text{type}_{[\texttt{T\_SEP}]}}\right] \tag{69}$$

$$\boldsymbol{H}'_{v,L_s} = \boldsymbol{H}_{v,L_s} + \boldsymbol{t}_{\text{scale}} * \boldsymbol{T}^{v}_{\text{type}} = \boldsymbol{H}_{v,L_s} + \left[\boldsymbol{t}_{\text{scale}} \odot \boldsymbol{t}^{v}_{\text{type}_{[\texttt{I\_CLS}]}}, \boldsymbol{t}_{\text{scale}} \odot \boldsymbol{t}^{v}_{\text{type}_1}, ..., \boldsymbol{t}_{\text{scale}} \odot \boldsymbol{t}^{v}_{\text{type}_N}\right] \tag{70}$$

We use the same LayerScale $\boldsymbol{t}_{\text{scale}} \in \mathbb{R}^{768}$ for both image and text type embeddings, as the contribution of the type embeddings should be the same for both modalities. We do not want to bias the model towards one modality.

Table 19 shows that this approach is able to slightly improve the performance of the model, and we achieve a gain of at least 0.3% in all tasks. While this is not a significant improvement, it shows that the TTE [Bao+22], together with LayerScale [Tou+21], can be beneficial, which is why we keep this approach.

After training, to validate whether the TTE is actually utilized, we check the average value of the LayerScale weights. If they show a value lower or equal to the initial value of 1e-5, then we can conclude that even though the model performance improves, the relative importance of TTE is low. We measure a mean of 2e-4 and observe the maximum weight for a channel to be 0.37. This shows that the model utilizes the TTE to some extend, although it does not seem to be that useful. This is also reflected in the low gain achieved from adding LayerScale to the TTE, as seen in Table 19. Nevertheless, we keep the TTE and LayerScale in the model, as they increase the performance slightly, and only add few parameters to the model (1,536 for both token type embeddings, and 768 for the learnable scaling weights of LayerNorm).

## 3.2.4 Enhancing Alignment

### 3.2.4.1 Region Descriptions for Contrastive Learning

Many papers we compare to, namely VLMo [Bao+22], FLAVA [Sin+21], and BEiT-3 [Wan+23], use Visual Genome [Kri+17] as one of the datasets to train their Vision-Language models. Developed for a variety of descriptive visual tasks, like visual question answering, it also contains images with so-called region descriptions. These are human annotated and highly curated descriptions of small regions in an image, often focusing on specific objects or parts of the image [Kri+17]. This makes them an attractive source for training vision-language models, as they provide a more fine-grained description of the image content than the global image caption, while still being abstract enough to accurately describe real-world objects and scenes.

Because the dataset is so highly curated, and the region descriptions often only cover a small part of the image, each image in the dataset has up to 50 region descriptions. Given that the dataset contains 108,077 images, and each region description can be considered as a seperate image-text pair, this results in a quite large dataset of 5.4M image-text pairs. Since the number of images is also relatively small, it does not require as much disk space as the other datasets we use, like CC3M [Sha+18] or CC12M [Cha+21]. All of this makes Visual Genome an attractive source for our experiments, however, as seen in our overview of the datasets we use in Table 4, we do not use it throughout this work.

This is because we encountered an unstable training behavior when using Visual Genome in combination with contrastive learning, as shown in Figure 20. As a side note, even though it looks like that without Visual Genome the accuracy saturates at around 70% after the first 6k steps, this is not the case: Figure 21 shows that the accuracy of the contrastive loss continues to increase with more training steps. We did not however observe this behavior when using Visual Genome.

We assume that the reason for this behavior is that the region descriptions are too specific, i.e. they focus on parts too small to capture the overall content of the image, and generally shorter than those of COCO, CC3M, and CC12M: While the data we collect has a mean caption length of 11.1 tokens (see Table 4), the region descriptions of Visual Genome only have a mean length of 4.7 tokens. Since our alignment of image and text is based on the global information carried by the image and the caption, too specific or too short descriptions might not be helpful, but rather confuse the model.

The reason why this works for other models, like VLMo, FLAVA, or BEiT-3, is because they use a concatenation of the image sequence and the text sequence as input to the shared encoder [Bao+22, Sin+21, Wan+23]. This approach itself allowes for a more fine-grained alignment of image and text, where individual text tokens can attend to individual image patches, and vice versa. Therefore, even descriptions of small regions can be helpful for these models.

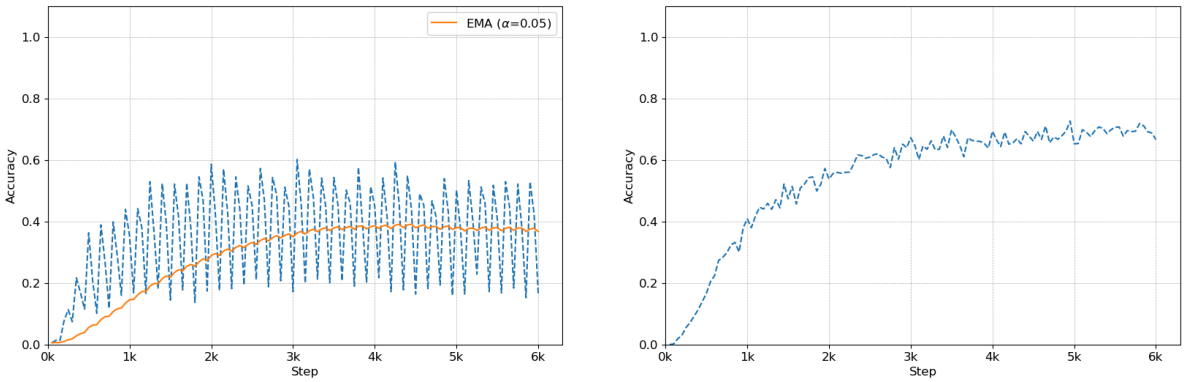


Figure 20: Training accuracy of Image-Text Contrast with Visual Genome (left) vs. without Visual Genome (right). Removing Visual Genome from the training data leads to a more stable training. Even though we only show the first 6k steps, this behavior continues throughout training.

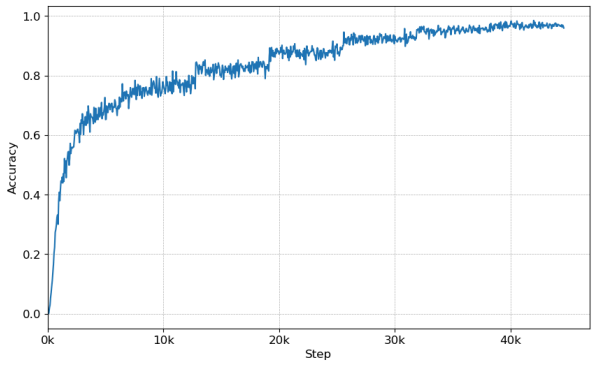


Figure 21: Visualizing the training accuracy of image-text contrast without Visual Genome shows that the accuracy continues to increase with longer training. This is in contrast to the training with Visual Genome, where the accuracy saturates after the first 6k steps.

Furthermore, larger models, like the VLMo's ViT-L/16 variant [Bao+22], have empirically shown to be robust to noisy image-text pairs. Since we "only" perform global image-text alignment, the fine-grained captions of Visual Genome can be considered as noisy, as they not always capture the global semantics of their image. This, and the fact that our model is relatively small compared to models like VLMo or BEiT-3, might also be a reason why we encounter this unstable training behavior when using Visual Genome.

#### 3.2.4.2 Image-Text Matching with Feature Fusion

Based on the success we experienced so far with applying contrastive learning to learn image-text representations, we now turn our attention to Image-Text Matching (ITM). Used as a training objective by VLMo [Bao+22] and METER [Dou+22], this tasks involves predicting whether an image and text pair match. In both papers, the implementation is straightforward: Since the Self-Attention mechanism in both works processes image and text jointly,

i.e. the text and image tokens are concatenated (see Equation 71), allowing the text tokens to attend to image patches and vice versa, the representation of the [T_CLS] can be used as the representation of the image-text pair. This representation is then passed into a linear layer, which is the classification head, to predict whether the image-text pair matches. In practice, this rather simple approach actually leads to the best results in both papers.

$$\boldsymbol{H}_{vw,l} = \left[\boldsymbol{h}_{w,l,\texttt{[T\_CLS]}}, \boldsymbol{h}_{w,l,1}, ..., \boldsymbol{h}_{w,l,M}, \boldsymbol{h}_{w,l,\texttt{[T\_SEP]}}, \boldsymbol{h}_{v,l,\texttt{[I\_CLS]}}, \boldsymbol{h}_{v,l,1}, ..., \boldsymbol{h}_{v,l,N}\right] \quad (71)$$

The approach of using the [T_CLS] token as the image-text representation in ITM stands in contrast to our approach, where the image and text representations are separate, and the [T_CLS] token only contains information about the text, and the [I_CLS] token only about the image. If we want to apply ITM to our model, we need to find a way to combine the image and text representations.

An intuitive way to combine, or rather compare, the image and text representations is to compute the cosine similarity between the global text representation $\boldsymbol{h}''_{w,K,\texttt{[T\_CLS]}}$ and the image representation $\boldsymbol{h}''_{v,K,\texttt{[I\_CLS]}}$. However, this is already performed by the contrastive loss. Instead, we propose an approach we call feature fusion, where we perform a linear transformation of the representation of the [T_CLS] and [I_CLS] token, and then simply add them together. The linear transformation is modeled by a linear layer. We justify approach by the intuition that a linear projection of the representations might transform them in a way that adding them together gives a meaningful representation of the image-text pair. Using the actual representations $\boldsymbol{h}''_{v,K,\texttt{[I\_CLS]}}$ and $\boldsymbol{h}''_{w,K,\texttt{[T\_CLS]}}$ for the addition is not a good idea, as they are already enforced to be similar under the contrastive loss. Simply adding them together does not express whether image and text match or not. Adding a linear projection however, allows the model to keep the representations aligned, while learning a projection of the aligned representations that is useful for ITM.

As with the contrastive loss, we apply ITM to the raw output of the intermediate and final output of the FFN from the shared Transformer block. That also means that we have seperate projections and classification heads for both outputs:

$$\begin{aligned}
\boldsymbol{h}'_{v,\texttt{[I_ITM]}} &= \boldsymbol{h}'_{v,K,\texttt{[I_CLS]}} \boldsymbol{W}'_{\texttt{[I_ITM]}} \\
\boldsymbol{h}'_{w,\texttt{[T_ITM]}} &= \boldsymbol{h}'_{w,K,\texttt{[T_CLS]}} \boldsymbol{W}'_{\texttt{[T_ITM]}} \\
\boldsymbol{h}''_{v,\texttt{[I_ITM]}} &= \boldsymbol{h}''_{v,K,\texttt{[I_CLS]}} \boldsymbol{W}''_{\texttt{[I_ITM]}} \\
\boldsymbol{h}''_{w,\texttt{[T_ITM]}} &= \boldsymbol{h}''_{w,K,\texttt{[T_CLS]}} \boldsymbol{W}''_{\texttt{[T_ITM]}}
\end{aligned} \quad (72)$$

$$\begin{aligned}
\boldsymbol{h}'_{\texttt{[IT_ITM]}} &= \left(\boldsymbol{h}'_{v,\texttt{[I_ITM]}} + \boldsymbol{h}'_{w,\texttt{[T_ITM]}}\right) \boldsymbol{W}'_{\text{itm_cls}} + \boldsymbol{b}'_{\text{itm_cls}} \\
\boldsymbol{h}''_{\texttt{[IT_ITM]}} &= \left(\boldsymbol{h}''_{v,\texttt{[I_ITM]}} + \boldsymbol{h}''_{w,\texttt{[T_ITM]}}\right) \boldsymbol{W}''_{\text{itm_cls}} + \boldsymbol{b}''_{\text{itm_cls}}
\end{aligned} \quad (73)$$

In general, $\boldsymbol{h}_{[\texttt{IT\_ITM}]} \in \mathbb{R}^2$ denotes the logits for ITM, where $h_{[\texttt{IT\_ITM}],0}$ and $h_{[\texttt{IT\_ITM}],1}$ denote the score that the image-text pair is not matching and matching, respectively. For clarity, we name the learnable parameters with the same primes ($'$) as the representations they are applied to.

Since at its core ITM is a binary classification task, we can use cross-entropy as the loss function:

$$\mathcal{L}_{\text{ITM}} = \frac{1}{2} * \mathcal{L}_{\text{CE}}\Big(\boldsymbol{h}'_{[\texttt{IT\_ITM}]}, \boldsymbol{y}_{\text{ITM}}\Big) + \frac{1}{2} * \mathcal{L}_{\text{CE}}\Big(\boldsymbol{h}''_{[\texttt{IT\_ITM}]}, \boldsymbol{y}_{\text{ITM}}\Big) = \tag{74}$$

With the loss for a single component being (on the example of the output from linear #1):

$$\mathcal{L}_{\text{CE}}\Big(\boldsymbol{h}'_{[\texttt{IT\_ITM}]}, \boldsymbol{y}_{\text{ITM}}\Big) = -\sum_{i=0}^{1} y_i \log \frac{\exp\Big(h'_{[\texttt{IT\_ITM}],i}\Big)}{\sum_{j=0}^{1} \exp\Big(h'_{[\texttt{IT\_ITM}],j}\Big)}, \qquad y_i \in \{0, 1\} \tag{75}$$

The target $\boldsymbol{y}_{\text{ITM}} \in \mathbb{R}^2$ is defined such that $y_0 = 1$ and $y_1 = 0$ if the image-text pair does not match, and $y_0 = 0$ with $y_1 = 1$ if they do. Since $h_{[\texttt{IT\_ITM}],0}$ and $h_{[\texttt{IT\_ITM}],1}$ are just logits, they have to be softmax-normalized to get probabilities, hence the exponentiation and division in the loss function. We do not use the tradiational binary cross-entropy loss, as applying softmax, followed by the logarithm, is numerically more stable.

The loss is added as a third loss term to the total loss function, which now reads:

$$\mathcal{L}_{\text{S-SMKE}} = \mathcal{L}_{\text{KD}} + \mathcal{L}_{\text{ITC}} + \mathcal{L}_{\text{ITM}} \tag{76}$$

Image-Text Matching requires both positive and negative examples, and there are two approaches to generating the latter. The first, more intuitive approach, is to sample a random text from the current batch for a given image, and vice versa. While straightforward, this method might make the task too easy, as negative example will usually be completely unrelated to the candidate.

To address this issue, the papers follow VLMo [Bao+22] and ALBEF [Li+21] employ hard-negative mining, which involves sampling difficult negative examples that are hard to distinguish from positive examples. ALBEF and VLMo utilize the cosine similarities between all possible image-text pairs of the current batch, which are generated as part of contrastive learning. They are stored in the matrix $\boldsymbol{L}$, explained in Section 1.5.3.2. A negative text for a given image is generated by randomly sampling a text/caption based on the cosine similarities between the image and all texts in the batch, where the similarity to the actual text/caption of the image is set to 0 to avoid selecting it as a negative example. This means that captions with a high cosine similarity to the image are more likely to be sampled as negative examples. The same process is applied to generate negative examples for captions.

Given a batch size $B$, $B$ positive examples (the actual image-text pairs), and $2B$ negative examples are generated. The $2B$ negative examples consists of the $B$ images in the batch

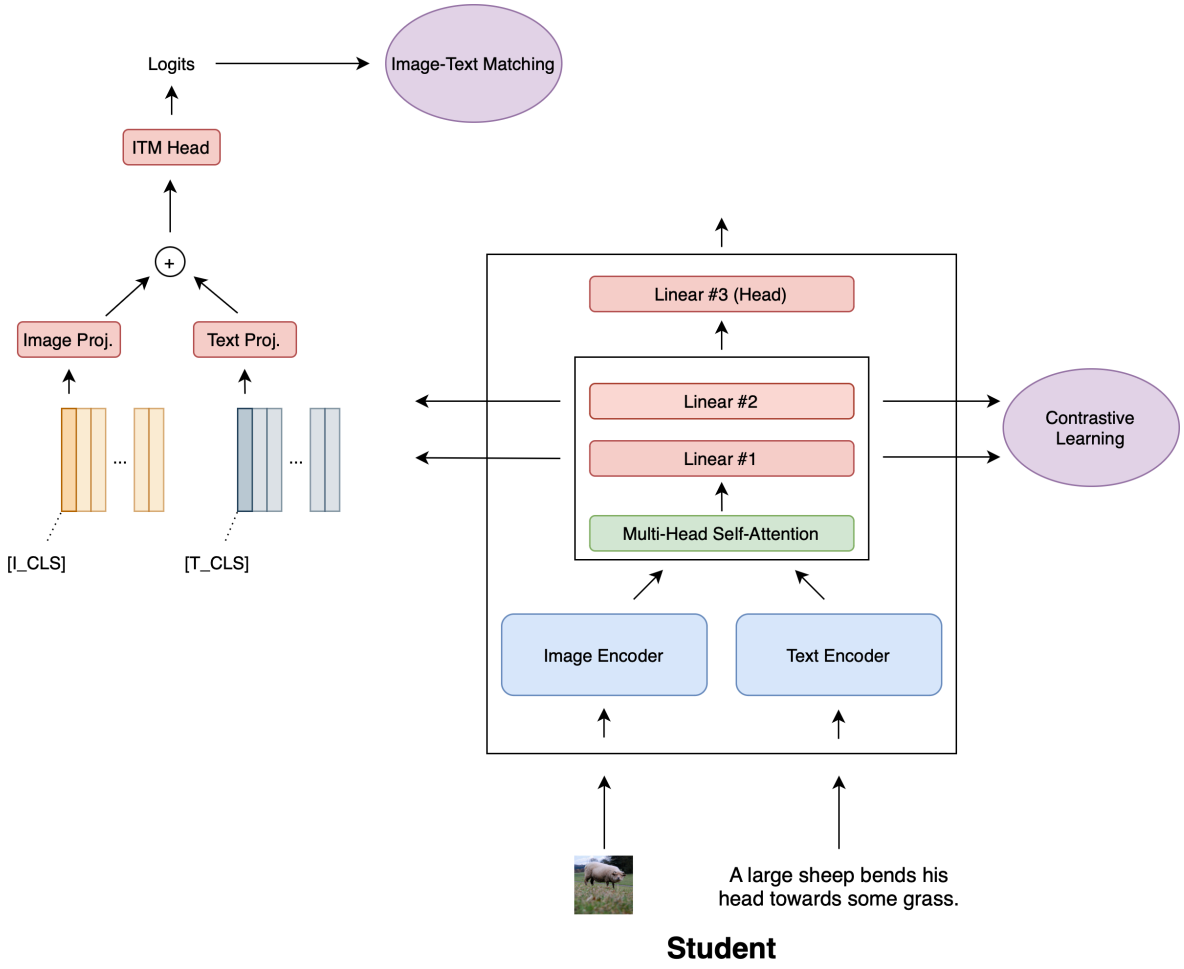


Figure 22: In our approach to Image-Text Matching (ITM), the global representations of the image and text are first transformed through seperate learnable linear projections, and then added to form a joint representation of the image-text pair. This represenation is then passed to a classification head to predict whether the image-text pair matches. This is done for both the output of linear #1 and linear #2, and for both there are seperate classification and linear projection heads, leading to a total of six additional linear layers. The Knowledge Distillation loss (MSE), and the teacher, are omitted for simplicity. The image-text example is taken from the COCO train set [Lin+14].

paired with their hard-negative captions, and the $B$ captions in the batch paired with their hard-negative images.

The remainder of the training setup remains unchanged, the classification heads and and the linear projections add around 1.1M parameters to the model, which is negligible compared to the total number of trainable parameters, which now stands at 117 million.

We compare both ITM with random negatives as well as ITM with hard-negative mining with our previous baselines.

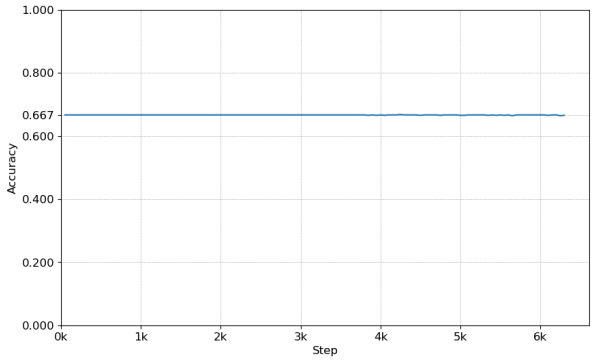


Figure 23: Training accuracy of the student model for Image-Text Matching (ITM) in the first epoch. The accuracy does not change from the initial value of 66.67%, and always predicts that image-text pairs do not match.

Plotting the training accuracy (Figure 23) shows a continuous score of 66.67% for both approaches, which remains constant throughout the first epoch with no deviation from this value. This suggests that our approach is not effective. We suspect this outcome arises because the classification head fails to learn anything meaningful from the fused representations, leading it to predict the most frequent class in the batch to somehow minimize the loss.

In this setting, the most frequent class corresponds to the scenario where the image-text pair does not match, as each batch contains $2B$ negative examples and only $B$ positive examples. Consequently, 66.67% of the fused representations in a batch are negative examples. If the classification head predicts that all examples are negative, it will achieve an accuracy of exactly 66.67%.

As this result gives no reason to continue the training, we stop it after the first epoch. We suspect that ITM might not be a suitable task for our model, as the representations of the image and text are separate. Generating a meaningful joint representation of an image-text pair is only possible if the tokens of the image and text can attend to each other using Self-Attention. This so-called cross-modal attention is given in models like VLMo [Bao+22], METER [Dou+22], and ALBEF [Li+21], but not in our model.

Although, as mentioned before, a logical step would be to simply compute the cosine similarity between an image-text pair, and using the score as an indicator of matching or non-matching, as this is already performed in the contrastive loss. Contrastive learning directly optimizes for similarity between image and text embeddings by maximizing the cosine similarity for positive pairs and minimizing it for negatives. Adding an cosine similarity step for ITM would only duplicate this process, without introducing any new information on which the model can improve.

#### 3.2.4.3 Increasing Negative Examples for ITC

Our current approach utilizes Distributed Data Parallel (DDP) [Li+20] with a batch size of 256 per GPU. With two GPUs, the combined batch size is 512 and image/text features are

gathered from all devices to increase the number of negative examples, as described in VLMo [Bao+22]. This method, as demonstrated in Section 3.2.1.2, improves performance.

However, implementations of image-text models that leverage contrastive learning typically use much larger batch sizes, and therefore have much more negative examples available. For instance, CLIP uses a batch size of 32,768 [Rad+21]. Achieving such large batch sizes would require more GPUs, as batch sizes exceeding 256 (per device) lead to out-of-memory (OOM) errors on the GPU (NVIDIA RTX 4090).

Although adding more GPUs to our setup is costly, the training time will be reduced proportionally to the number of GPUs used. For example, using two GPUs instead of one halves the training time. This is because DDP shardes the whole training dataset between all devices, such that each device processes a different part of the dataset [Li+20] (already mentioned in Section 3.2.1.2 about DDP). From a cost perspective, this is acceptable, however, we decide against this approach, because we want to keep the number of GPUs manageable. Moreover, going with more GPUs would make the success of our approach dependent on the number of GPUs one has available, which is why we search for alternatives that allow us to increase the number of negative samples without requiring more GPUs.

One intuitive approach to increasing the number of negative examples is to use a FIFO queue-based memory bank. The queue stores image and text embeddings, i.e. tokens [I_CLS] and [T_CLS], from previous batches. With each new batch, the memory bank is updated by adding the current batch embeddings and discarding those of the oldest batch. These stored embeddings, combined with the embeddings of the current batch, are then used as negative examples.

Given a batch size of $B$ and a memory bank size of $U$, we can achieve $B + U - 1$ negative samples. For instance, with a batch size of 256 and a memory bank size of 768, we can attain 1023 negative samples. This configuration effectively simulates a contrastive loss with a batch size of 1024, and is comparable to four GPUs with DDP and a batch size of 256 per GPU. However, it is important to note that this "simulation" of larger batch sizes with a memory bank only applies to the number of negative examples: The actual gradients are still computed using the effective batch size of 512 (… because we still use the double GPU setup with a batch size of 256 per GPU).

Implementing this approach actually requires to maintain four distinct memory banks. This is because we apply the contrastive loss to both the intermediate representations $\boldsymbol{h}'_{v,K,[\texttt{I\_CLS}]}$ and $\boldsymbol{h}'_{w,K,[\texttt{T\_CLS}]}$, and the output of the shared Transformer block $\boldsymbol{h}''_{v,K,[\texttt{I\_CLS}]}$ and $\boldsymbol{h}''_{w,K,[\texttt{T\_CLS}]}$. An overview of the memory banks required for the approach is shown in Table 20.

Illustrated in Table 21, we observed a significant drop in performance, and the resulting model is not usable. The accuracy on ImageNet-1K, which is 0.001% and therefore corresponds to a random classification (1000 classes), indicates that the model did not learn any-

| | Modality | Layer | Stores negatives for |
|---|---|---|---|
| 1 | Image | Linear #1 | [T_CLS] of Linear #1 |
| 2 | Text | | [I_CLS] of Linear #1 |
| 3 | Image | Linear #2 | [T_CLS] of Linear #2 |
| 4 | Text | | [I_CLS] of Linear #2 |

Table 20: Each memory bank stores the representations of one modality, created by a layer, to use as negative examples for the represenations of the other modality, created by the same layer.

| Memory Bank | ImageNet-1K | Retrieval | |
|---|---|---|---|
| | | MSCOCO | Flickr30K |
| × | **30.3** | **67.2** | **78.29** |
| ✓ | 0.001 | 0.12 | 0.58 |

Table 21: Using a memory bank to store negative examples leads to unusable results. Especially noticeable is an accuracy of 0.001% on ImageNet-1K, which corresponds to a random classification.

thing useful. This suggests that simply increasing the number of negative examples via a memory bank does not help in learning richer represenations.

We suspect this drop in performance originates because of the following reasons: When using an actual batch size of 512 without a memory bank, all negative examples are generated during the same step, and therefore by the same model with the same weights. This is the case even with DDP, as all model replicas are synchronized, i.e. share the same weights. Therefore, the representations share the same latent space and are consistent with each other. A similarity measure, in our case the cosine similarity, can then provide the model with a meaning of distance between these, in our case an image and text, representations.

However, when using a memory bank, most negative examples come from previous batches that were stored in the memory bank. As the model's weights constantly change, especially at the beginning of training, there is a continous shift in the representation space. This shift is so pronounced that even representations from the immediate previous steps differ significantly from the current representations, and a similarity measure will not provide meaningful information to the model. To demonstrate, say we generate a representation $\boldsymbol{h}''_{v,K,[\texttt{I\_CLS}]}$ for an image, and store it in the memory bank. If we generate a new representation for the same image in the next step, then both representations will not be the same, because the weights with which the representations have been generated were not the same. Consequently, the cosine similarity will not be at its maximum value of 1, even though it is the same image. It follows that in general, the representations stored in the memory bank are

not consistent with the representations of the current training step, and comparing them with cosine similarity does not provide a meaningful measure of similarity to the model.

This can be thought of as a less extreme case of comparing the representations of an image-only and text-only model, which are not associated with each other. In the beginning of our experiments, we tested image-text retrieval with the Data2Vec2 image and text model (see Table 10), and observed that this approach is ineffective for image-text retrieval. The representations produced by both models do not have any relationship with each other, and therefore the cosine similarity does not provide any meaningful information to the model.

While this effect is less pronounced with a memory bank, as the representations are still generated by the same model, the shift in the model's weights is still significant enough to make the representations inconsistent with each other.

#### 3.2.4.3.1 Relation to Initial Memory Bank Approach

The memory bank was initially introduced by [Wu+18] as a mapping of the complete training dataset: The embedding of each sample in the dataset is stored in the memory bank (illustrated in Figure 24). For each batch/step, $K$ samples are randomly drawn from the memory bank to be used as negative examples. The representations of samples in the current batch are then updated in the memory bank [Wu+18]. This approach is similar to ours, but faces the same problem: The representations come from an older variant of the model with different weights. Even worse, the representation of an example in the dataset is updated only when it was last seen in a batch, which can be a long time ago for large datasets. However, the authors mitigate this issue by using proximal regularization, as shown in Equation 77.

$$-\log h(i, \boldsymbol{v}_i^{t-1}) + \lambda * \|\boldsymbol{v}_i^t - \boldsymbol{v}_i^{t-1}\|_2^2 \tag{77}$$

While the term $-\log h(i, \boldsymbol{v}_i^{t-1})$ can be ignored for now, as it just denotes a form of contrastive loss, the other term $\lambda * \|\boldsymbol{v}_i^t - \boldsymbol{v}_i^{t-1}\|_2^2$ serves as the proximal regularization. It describes the mean squared error between the representation $\boldsymbol{v}_i^t$ of a training example $i$ in the current batch, e.g. an image, and the representation of the same training example $\boldsymbol{v}_i^{t-1}$ stored in the memory bank, which was updated the last time the image was in the current batch. This time is denoted as time step $t-1$.

The goal is to minimize Equation 77, and therefore to also minimize the proximal regularization, which is the mean squared error. The mean squared error is only at its minimum, when both inputs are the same. Therefore, the proximal regularization enforces the representation of a training example to not change too rapidly between updates, and allows for more stable/consistent negative examples, depending on the value of weight $\lambda$. The authors report improved results with a value of $\lambda = 30$ [Wu+18], meaning that the proximal regularization term is 30 times more important than the contrastive loss term.

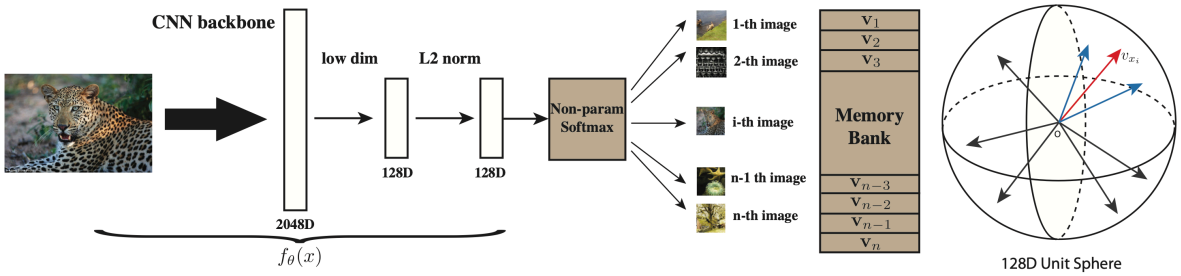


Figure 24: The memory bank, originally developed for self-supervised image representation learning, stores a 128-dimensional embedding of each training example. Contrary to what can be observed, instead of the whole dataset, just $K$ samples are drawn from the memory bank in each iteration to be used as negative examples [Wu+18].

This forces the model to keep the representations of the training examples in the memory bank consistent, so that a similarity measure can provide meaningful learning signals to the model. Our approach does not take this into account.

#### 3.2.4.3.2 Momentum Encoder

Inconsistent representations in a memory bank is a problem also identified by the authors of MoCo [He+19], which employ a different approach to address this issue. They also use a queue-based memory bank, which is, similar to our approach, much smaller than the training dataset. However, in MoCo, the negative examples in the memory bank are not updated by the model that is being trained, but instead by a momentum encoder. The momentum encoder is a copy of the model, but its weights are an exponential moving average of the actual model weights, defined in Equation 78 [He+19]. Consequently, the momentum encoder's weights are not updated by gradient descent, and the negative examples do not come from the model that is trained, but only from the momentum encoder.

$$\theta_k = m * \theta_k + (1 - m) * \theta_q \tag{78}$$

With $\theta_k$ being the momentum encoder weights, $\theta_q$ the actual model weights, and $m$ the momentum coefficient, the momentum encoder can updated very slowly. Typical values for $m$ are usually between $m = 0.99$ and $m = 0.999$ [Che+20, CXH21, He+19, Li+21].

The results are weights that change very slowly, which will also hold for the representations the momentum encoder produces. This approach can be seen as maintaining consistency in the model that produces the negative examples, rather than making the negative examples consistent themselves, as is done through the regularization term in Equation 77. An illustration of this method for our image-text contrastive learning can be seen in Figure 25.

Even though no gradients are needed to update the momentum encoder, it still requires additional GPU memory to keep it in memory, which is the disadvantage of this variant.

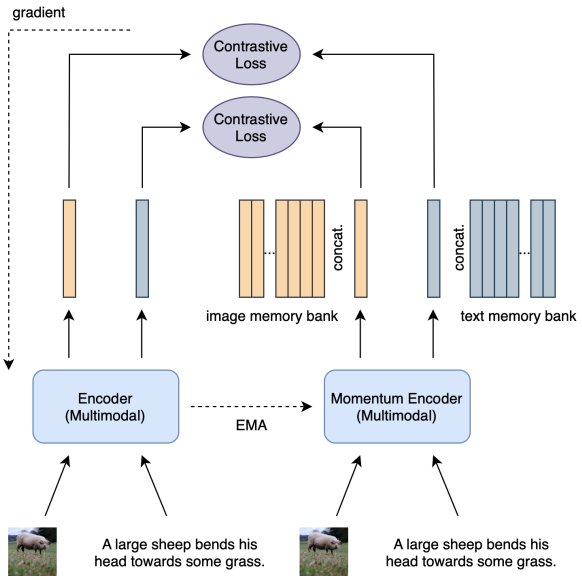


Figure 25: A momentum encoder generates negative examples for ITC, which are stored in a memory bank that discards the representations of the oldest batch when new ones are added after every step. Figure inspired by MoCo v2 [Che+20], image and text taken from COCO [Lin+14].

#### 3.2.4.3.3 Resolution

We can't use the memory bank style of [Wu+18] since we have 3,264,868 training examples (see Table 4). Each embedding has a size of 768, and storing them at full precision (float32) would require $3,264,868 * 768 * 4$ bytes $\approx 10$ GB of additional GPU memory. However, with our current setting we only have around 1.2 GB of GPU memory remaining.

We suspect that using a proximal regularization term, as in Equation 77, could also stabilize our memory bank approach. However, we cannot apply it, since the term is based on the difference (MSE) between the representation of an individual training example when it was last updated in the memory bank, and its current representation in the batch. This requires the exact approach of [Wu+18], which we just deemed as infeasible. Our memory bank is significantly smaller than the training dataset, and older samples are dequeued, so the same training example will never be in the memory bank and the current batch simultaneously.

In conclusion:

1. We cannot use a FIFO queue-based memory bank, as representations between samples are inconsistent.
2. We cannot use a memory bank that stores all training examples, as it would require too much additional GPU memory.
3. Proximal regularization is not applicable to a memory bank that is smaller than the training dataset.

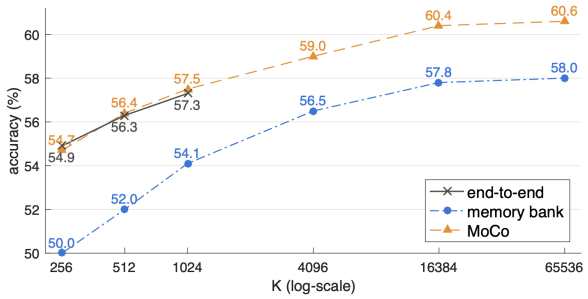


Figure 26: Experiments done by the authors of MoCo show that increasing the memory bank size up to 65k is beneficial to performance [He+19].

The only alternative is to use a momentum encoder as in MoCo [He+19], which is why we opt for this approach in the next experiment. Our experimental setup remains the same, but we add a momentum encoder, which is a copy of our student model (as shown in Figure 25). Oriented on ALBEF [Li+21], we use a memory bank of size 65,536 and a momentum factor of 0.995. Both hyperparameters also lead to good results in MoCo, where this approach was first introduced [He+19].

However, we encounter an OOM error, which is not surprising, considering the large memory bank size of 65k, and that we need to maintain four of those in total (see Table 20). Considering that the size of the memory bank is crucial for performance (illustrated by MoCo [He+19] in Figure 26), we would like to keep it as large as possible. Based on this goal, we apply two optimizations to reduce the GPU memory.

First, we only apply the contrastive loss on the output of the shared Transformer layer ($\boldsymbol{h}''_{v,K,[\texttt{I\_CLS}]}$ and $\boldsymbol{h}''_{w,K,[\texttt{T\_CLS}]}$), and remove the contrastive loss $\mathcal{L}_{\text{CL}'}$ on the intermediate representations $\boldsymbol{h}'_{v,K,[\texttt{I\_CLS}]}$ and $\boldsymbol{h}'_{w,K,[\texttt{T\_CLS}]}$. This reduces the GPU memory by a margin, as we only need two memory banks of size 65,536 each, instead of four (see Table 20).

We also identify a further optimization: Usually, the forward pass of the momentum encoder is done after the forward pass of the model that is trained, which is an approach MoCo [He+19] and ALBEF [Li+21] follow. It follows that during the forward pass of the momentum encoder, GPU memory is already allocated to store all activations of the actual model, as they are needed for the backward pass later. Because we did not encounter an OOM error before using the momentum encoder, and we observed that the GPU memory in previous experiments was almost fully utilized (up to 98%) even without a momentum encoder, we suspect that the forward pass of the momentum encoder is the reason for the overflow.

This can be remedied by performing the update and forward pass of the momentum encoder in each training step before any other work is done (this includes the forward pass of the teacher and student). This way, the activations of intermediate layers of the momentum encoder are freed before the forward pass of the actual model. The result is the same performance, as the work done per step remains the same, but now reordered, while we avoid the memory overflow. We illustrate this in Figure 27.

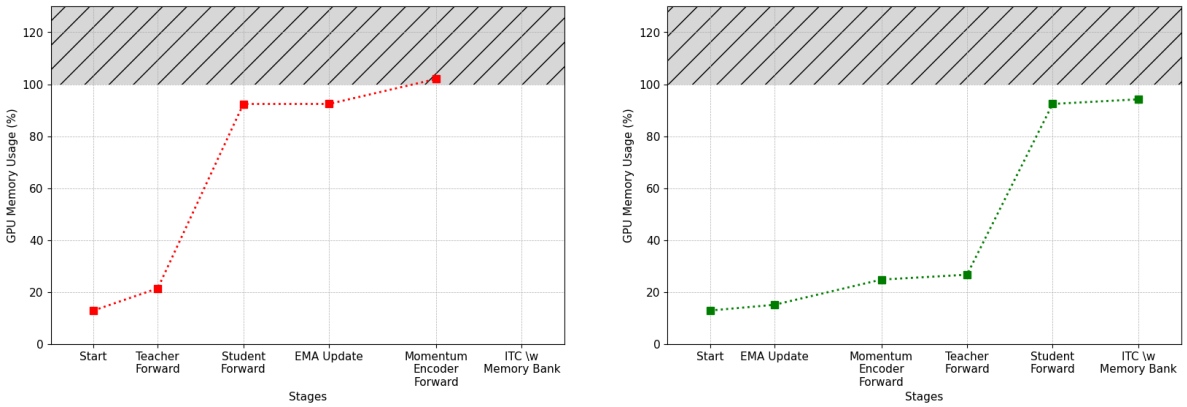


Figure 27: Doing the forward pass of the momentum encoder after the forward pass of the model, when the model's activations are already stored on the GPU memory (left), leads to a cuda OOM error. This can be avoided by reordering the operations, so that the momentum encoder (EMA) update and forward pass are performed before the forward pass of the model (right). Results are based on NVIDIA RTX 4090 with 24 GB of GPU memory.

| Momentum Encoder | ImageNet-1K | Retrieval | |
|---|---|---|---|
| | | MSCOCO | Flickr30K |
| × | **30.3** | **67.2** | **78.29** |
| ✓ | 30.0 | 64.28 | 76.17 |

Table 22: Comparison of the Standard ITC approach with momentum encoder and a memory bank of size 65,536. The momentum encoder approach does not exceed the performance of the standard ITC approach.

The result is shown in Table 22. The performance does not exceed that of the the standard gathering from all devices with just 511 negative examples (effective batch size of 512). However, the experiment seems more promising to achieve a better retrieval performance with more epochs compared to the previous approach (see Figure 28), as the latter appears to saturate towards the end of training.

We consider efficiency and simplicity as a key aspect of our work. Since adding a momentum encoder

1. increases the complexity of our approach,
2. increases the training time from approx. 7 hours to 10.3 hours, and
3. does not lead to a significant improvement in performance,

we decide to abandon this approach. The increase training time can be attributed to the additional forward pass of the momentum encoder and a large matrix multiplication, resulting from the large memory bank.

### 3.2.5 Target Cross-Modal Late Interaction

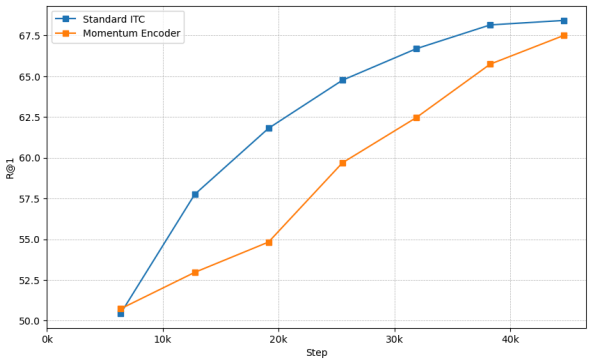


Figure 28: A momentum encoder shows a promising trend to achieve better retrieval performance towards the end of training.

#### 3.2.5.1 Cross-Modal Late Interaction

Until now, we used the global text and image representations [T_CLS] and [I_CLS] for contrastive learning. This has the disadvantage that only global information is utilized, and fine-grained, token/patch-specific, information is not considered. This can make retrieval, and alignment in general, difficult, especially if real-world concepts described by an image and a text differ in small, yet important, details. An example of this can be seen in (TODO: vis retrievals on full coco), where multiple retrievals are incorrect, even though they are semantically very similar to the query. The differences between the query and the retrieved samples are often so small that they are not captured by the global representations. This is one of the reasons why models like BEiT-3 [Wan+23] or VLMo [Bao+22], which allow fine-grained alignment of text and image through cross-attention, perform better than models that only use global representations, like CLIP [Rad+21] and our model. To address the issue of fine-grained alignment, the authors of FILIP [Yao+21] introduced Cross-Modal Late Interaction (CMLI) for contrastive learning, which showed improvements in retrieval performance even for a model that already uses cross-attention.

As shown in Figure 29, no cosine similarity between [T_CLS] and [I_CLS] is computed, but instead the cosine similarity between all image patches $\left[\boldsymbol{h}_{v,l,k}\right]_{1\leq k\leq N}$ and text tokens $\left[\boldsymbol{h}_{w,l,j}\right]_{1\leq j\leq M}$, with $N$ being the number of image patches, and $M$ being the number of text tokens. Specifically, $N$ and $M$ denote the number of patches/tokens in a sequence that are not the cls token ([I_CLS]/[T_CLS]) or padding token ([PAD]) [Yao+21]. The choice to exclude padding tokens is obvious, as they do not carry any semantic information. The cls token is excluded, as it is not specific to any token/patch, and is used to represent global information. We additionally exclude the end-of-sequence token ([EOS]), as it is not specific to any token/patch either. The result is the cosine similarity between all image patches and text tokens of an image-text pair.

The next step is to find for each image patch $k$, the text token with the maximum cosine similarity to this image patch.

$$m_k^{\text{i2t}} = \underset{1 \leq j \leq M}{\text{argmax}} \left[\boldsymbol{h}_{v,l,k}\right] \left[\boldsymbol{h}_{w,l,j}\right]^T \tag{79}$$

Likewise, for each text token $j$, we get the image patch with the maximum cosine similarity to this text token

$$m_j^{\text{t2i}} = \underset{1 \leq k \leq N}{\text{argmax}} \left[\boldsymbol{h}_{v,l,k}\right] \left[\boldsymbol{h}_{w,l,j}\right]^T \tag{80}$$

This has an intersting effect: For each image patch, the semantically most similar text token is found, and vice versa for each text token - the result of this operation can be seen in (2) of Figure 29. Consequently, the model will be able to associate small details of an image with individual text tokens, and vise versa. The actual cosine similarity between an image-text pair is then the average of all associations between an image patch and a text token.

$$s^{\text{i2t}}_{\boldsymbol{H}_{v,l},\boldsymbol{H}_{w,l}} = \frac{1}{N} \sum_{k=1}^{N} \left[\boldsymbol{h}_{v,l,k}\right] \left[\boldsymbol{h}_{w,l,m_k^{\text{i2t}}}\right]^T \tag{81}$$

$$s^{\text{t2i}}_{\boldsymbol{H}_{v,l},\boldsymbol{H}_{w,l}} = \frac{1}{M} \sum_{j=1}^{M} \left[\boldsymbol{h}_{v,l,m_j^{\text{t2i}}}\right] \left[\boldsymbol{h}_{w,l,j}\right]^T \tag{82}$$

Here, for one image-text pair, $m_k^{\text{i2t}}$ denotes the index of the text token with the highest cosine similarity to image patch $k$, and $m_j^{\text{t2i}}$ the index of the image patch with the highest cosine similarity to text token $j$. $s^{\text{i2t}}_{\boldsymbol{H}_{v,l},\boldsymbol{H}_{w,l}}$ denotes the the similarity score between an image representation $\boldsymbol{H}_{v,l}$ and text representation $\boldsymbol{H}_{w,l}$. Vice versa, $s^{\text{t2i}}_{\boldsymbol{H}_{v,l},\boldsymbol{H}_{w,l}}$ denotes the similarity score between a text representation $\boldsymbol{H}_{w,l}$ and an image representation $\boldsymbol{H}_{v,l}$. $l$ can denote any layer of the model, but is usually the last layer, as the representations are most meaningful there.

In contrast to the standard contrastive learning, this similarity measure is not necessarily symmetric, as e.g. a text token might have a maximum cosine similarity to another image patch, than the image patch that has its maximum similarity to the text token [Yao+21]. The process in illustrated in Figure 29.

While this approach allows for a fine-grained alignment of image and text, its practical implementation is very computationally and memory intensive. For standard constrastive learning, is is sufficient to compute the cosine similarity of the global representation (cls token) between every possible image-text pair in a batch. If negative examples are gathered from all devices, then the number of dot products to compute is defined as $(B * P)^2$, with $B$ being the batch size per device, and $P$ being the number of devices (in our case GPUs). As we use a batch size of $B = 256$ per device, and use $P = 2$ GPUs, the number of dot products to compute is $(256 * 2)^2 = 262,144$. Considering that we perform this efficiently using matrix multiplication, and the embedding size is 768 with float32 precision, we already need

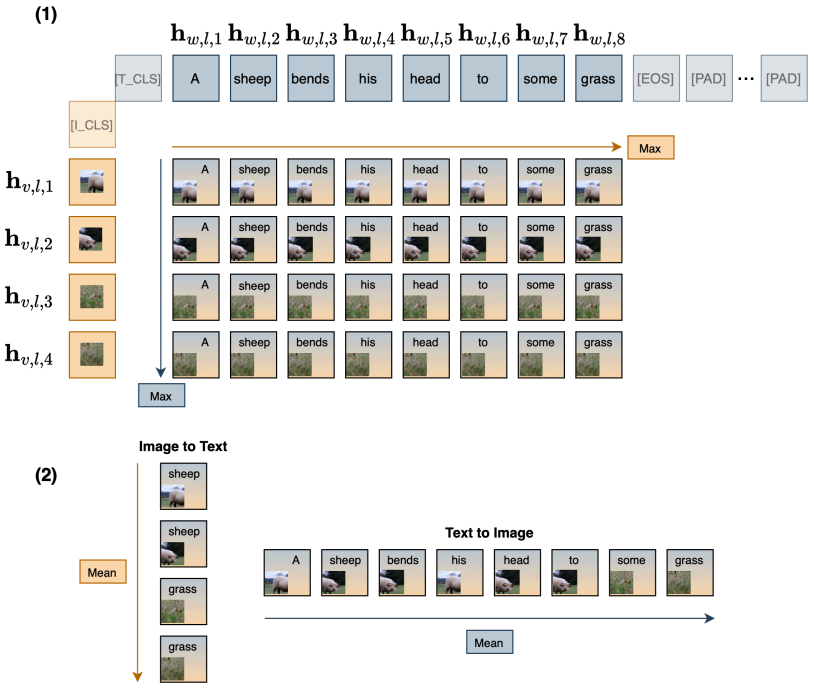


Figure 29: For a token/patch, CMLI finds the timestep with the highest semantic match from the other modality. This enables the model to associate small details of image and text with each other. Notice how through the max-operation patches containing grass are always associated with the word "grass", and the words "sheep" and "head" are matched with the head of the sheep (associations created through max are shown in (2)). The cosine similarity is then the average of all associations between an image-text pair. Figure inspired and adapted from [Yao+21].

$262,144 * 768 * 4$ bytes $= 805.31$ MB of GPU memory, which is still manageable, since we have around 2 GB of GPU memory remaining for a step.

However, with CMLI, we need to compute the similarity between all possible image-text pairs, where the similarity for one pair requires the computation of the cosine similarity between all image patches and text tokens of the image-text pair. With a maximum text sequence length of 64 tokens [Wan+23], two of which are ignored as they are the cls and eos token, and an image sequence length of 196 (without cls token), the number of dot products to compute for just one image-text pair is $196 * 64 = 12,544$. With a batch size of 256 per device, and 2 GPUs, the number of dot products increases from $262,144$ to $256 * 12,544 = 6,422,528$. Even if the embedding dimension is reduced to 256, which is a simplification done in FILIP [Yao+21], we need $6,422,528 * 256 * 4$ bytes $= 6.58$ GB of additional GPU memory, just to store the result. Consequently, this approach is not feasible in our setup.

#### 3.2.5.2 Method

What is feasible though, is to apply CMLI to a setting where the computation is more lightweight. As the driving factor behind the memory requirements in contrastive learning is that the similarity between all possible image-text pairs in a batch is computed, this is removed when just the similarity between positive pairs is computed, which is what we call Target-CMLI.

Target-CMLI is not used for contrastive learning, but rather to alleviate the problem of regressing patch-level information of the teacher model. Recall that in the current setting, which is multimodal knowledge distillation, it is merely possible to regress the global representations of the teacher model, and not the patch-level information. This is because the teacher model only outputs patch-level predictions for the image modality, and not for the text modality. Consequently, the student model can replicate the output of the teacher model for the image modality, but not for the text modality, as it is not possible to assign a text token to a specific image patch. This is illustrated in Figure 19, and discussed in Section 3.2.2.1.

However, as seen in Figure 29, when computing the cosine similarity between all image patches and text tokens of an image-text pair, and selecting the $\mathrm{argmax}$ over all image patches with respect to a text token, then a corresponding, or at least similar, image patch can be found for each text token ((2) of Figure 29). This means that the student model can replicate the output of the teacher model for the text modality by first selecting the most similar image patch for each text token, and then minimizing the Mean Squared Error (MSE) between the teacher's representation of the selected image patch and the representation of the text token.

For the patch-level image representation of the student model that means that we can now also regress the patch-level information of the teacher, and not only the global information. This was also possible in all previous experiments, as the order of the image patches does not change between student and teacher image representations, however, this would heavily bias the parameters of the shared Transformer block towards the image modality.

The definition of the loss changes as follows:

$$\mathcal{L}_{\text{KD}}^{\text{i2t}} = \text{MSE}\Big(\boldsymbol{H'''}^{s}_{v,K}, \boldsymbol{H}^{t}_{v,L_t}\Big) = \sum_{n=1}^{N} \|\boldsymbol{h'''}^{s}_{v,K,n} - \boldsymbol{h}^{t}_{v,L_t,n}\|_2^2 \tag{83}$$

For a given text representation we first need to find $m_j^{\text{t2i}}$ for each text token $j$, and then define the loss as:

$$\mathcal{L}_{\text{KD}}^{\text{t2i}} = \text{MSE}\Big(\boldsymbol{H'''}^{s}_{w,K}, \boldsymbol{H}^{t}_{v,L_t}\Big) = \sum_{z=1}^{Z} \|g\big(\boldsymbol{h'''}^{s}_{w,K,z}\big) - g\Big(\boldsymbol{h}^{t}_{v,L_t,m_z^{\text{t2i}}}\Big)\|^2 \tag{84}$$

We denote $g(\cdot)$ as a linear projection to reduce the dimensionality of the image representation from the teacher, and the text representation from the student, to 32. This is done to

| T-CMLI | ImageNet-1K | Retrieval | |
|---|---|---|---|
| | | MSCOCO | Flickr30K |
| × | **30.3** | **67.2** | **78.29** |
| ✓ | 26.7 | 65.68 | 76.2 |

Table 23: Comparison of adjusting the loss function $\mathcal{L}_{\text{KD}}$ from regressing the [I_CLS] of the teacher model to regressing the patch-level information of the teacher model using Target-CMLI. We observe a decrease in all metrics.

(1) reduce memory consumption when computing the similarity between all image patches and text tokens for all image-text pairs in a batch, and (2) to reduce the information that can be expressed by $g(\boldsymbol{x})$. We motivate (2) by the fact that matching individual image patches to text tokens is a very fine-grained task, and we want to aviod having pixel-level information in the patch representations of the teacher model, which would, despite the matching via $\text{argmax}$, cause problems with predicting those representations, as we can't extract those pixel-level information from the text tokens. The total Knowledge-Distillation loss remains the same, and is the mean of the two losses:

$$\mathcal{L}_{\text{KD}} = \frac{1}{2} * (\mathcal{L}_{\text{KD}}^{\text{i2t}} + \mathcal{L}_{\text{KD}}^{\text{t2i}}) \tag{85}$$

We use the mean instead of the sum, as we also use the contrastive loss, and we want both losses to have the same weight. We find $m_k^{\text{t2i}}$ using an embedding dimension of 32, which is achieved through the same projection $g(\cdot)$. The hidden size of the model remains at 768.

What makes the implementation of Target-CMLI feasible is that we only need to compute the similarity between the positive pairs, and not all possible image-text pairs in a batch. This is because we only need to find the most similar image patch for each text token of a positive pair. For a per-device batch size of 256, Target-CMLI requires $12,544 * 256 = 3,211,264$ dot products to compute. With an embedding dimension of 32, just $3,211,264 * 32 * 4 \text{ bytes} = 411 \text{ MB}$ of additional GPU memory is required. This is feasible in our setup.

#### 3.2.5.3 Results

The results, as seen in Table 23, can be considered as disappointing. We lose more than 2 percentage points in average retrieval on both MSCOCO and Flickr30K, while the performance on ImageNet-1K decreases by nearly 4 percentage points.

A look at a visualization (Figure 30) of matches between text tokens and their top-5 most similar image patches under cosine similarity reveals that while there are some cases where the token has the highest similarity to an actual image patch belonging the the correct object, the matching is far from perfect, and matches are often either completely unrelated, or scattered across almost random regions. A matching that would be expected from a successful approach is illustrated by the authors of FILIP [Yao+21], and can be seen in Figure 43. While the exaples of FILIP are based on CMLI for contrastive learning, and not our Target-CMLI

for knowledge distillation, the principle of matching text tokens to image patches remains the same, and the quality of the matches is expected to be similar.

In general, we observe that the patch with the highest similarity for a text token is often a patch completely unrelated to the token. While it is true that, thanks to self-attention, the representation of an image patch not only contains information about the patch itself, but also about patches it considers important, the self-attention map of the image patch with the highest similarity to a text token, which is the matched image patch, does not show any clear signs that it contains aggregated information from patches that are part of the object the text token describes. If that were the case, then the self-attention map of the matched image patch would show a clear focus on the object the text token describes/represents. This is illustrated in Figure 30, where for each example, the left image shows the top 5 most similar image patches for a text token under cosine similarity, and the right image shows the self-attention map of the image patch with the highest similarity to the text token, i.e. the image patch $m_j^{\text{t2i}}$ for a text token $j$. For the aforementioned, we observe a very inconsistent matching between text tokens and image patches. Especially examples "planes" and "birds" show that a mismatch is not rare.

As a reference, we provide the self-attention map, w.r.t. the [I_CLS] token, of the final Transformer layer from the self-supervised image model DINO [Car+21] in Figure 44 (Appendix). We would expect the self-attention map of the matched image patch $m_j^{\text{t2i}}$ to the example text token $j$ (Figure 30) to be similar to that of DINO. However, this is not always the case.

Moreover, the top 5 matched image patches are often scattered across the image, and not focused on a specific region, underlining that the approach does not work as intended. Instead, a result similar to that of FILIP in Figure 43 is expected, where the matched image patches are focused on the object the text token describes, and lie next to each other. Our results are much more similar to that of CLIP, also illustrated by the authors of FILIP (Figure 43), leading us to believe that a missing cross-modal attention might be the reason.

We suspect this, because FILIP [Yao+21], like VLMo [Bao+22] and BEiT-3 [Wan+23], uses cross-attention to align text and image. This allows individual text tokens to attend to specific image patches, and vice versa, leading to e.g. text tokens to be able to "locate" the objects they describe in an image. Cross-Modal Late Interaction then allows to apply this "locating of the matched image patch" to contrastive learning and retrieval.

CLIP however, like our model, does not use cross-attention, and only uses global representations for contrastive learning. Therefore, CLIP is not able to find relationships between individual text tokens and image patches, leading to a scattered matching, as seen in Figure 43.

While we do not use CMLI for contrastive learning, but for knowledge distillation, the same principles apply. Even worse, since the image patches that we try to match to text tokens are not even the result of the student model, but of the teacher model, the model does not have the possibility to somehow learn patch-level representations that are suitable for matching

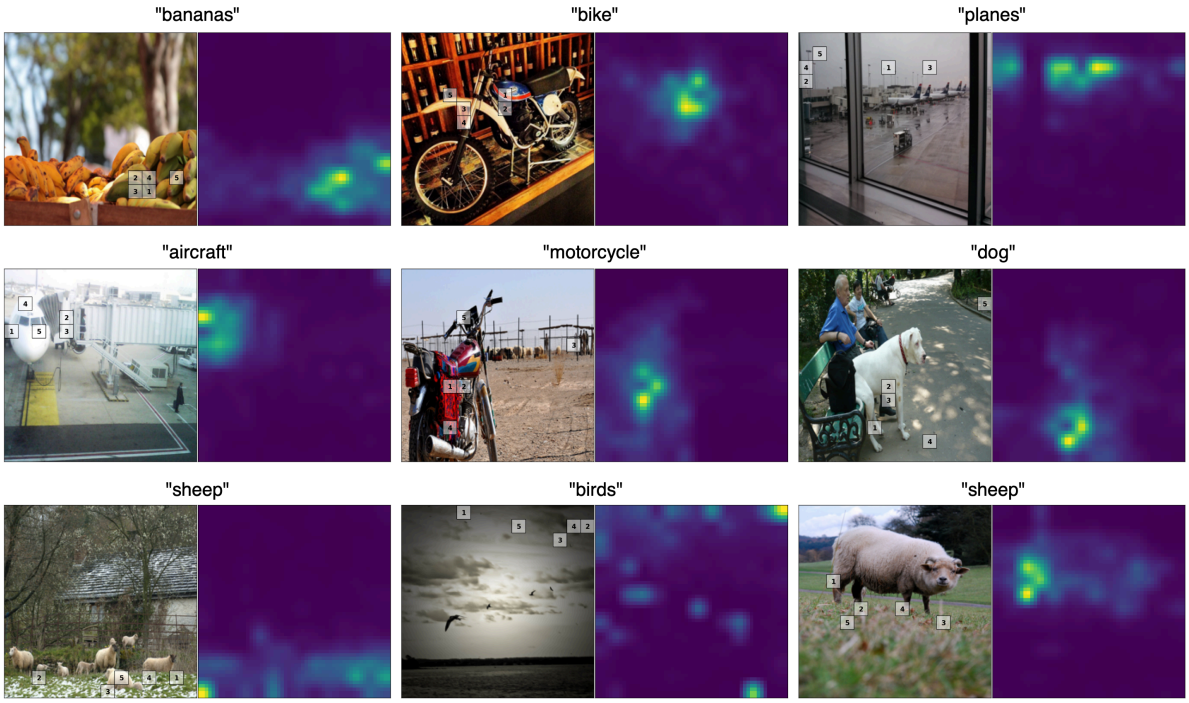


Figure 30: Visualization of selected text tokens from the image's caption and the tokens top 5 most similar image patches under cosine similarity (left). The self-attention map (right) of the image patch with the highest similarity to the text token is shown next to the original image. We observe that matched image patches are often scattered across the image, are not part of the object the text token represents, and their self-attention map, indicating the composition of the token, does not always show a clear focus on the object the text token describes. The title shows a text token $j$, and the self-attention map with respect to image patch $m_j^{\mathrm{t2i}}$ (i.e. the matched one). Image-text pairs taken from COCO test set [Lin+14].

to text tokens. Consequently, there really is no guidance for the model to learn matching its text tokens to the right image patches of the teacher model.

Lastly, Figure 30 shows the problem based on text tokens that have a real-world meaning, and therefore a counterpart in images: The text token "plane" can also be present in an image as an actual plane. However, text tokens like "a", "the", and even a full stop ".", which is a valid token, are also matched to image patches. They are merely fill words or grammatic nuances in a sentence, and do not carry any semantic information that can be mapped to an image patch. Because CMLI works over all text tokens, and only few text tokens actually have an object-level counterpart in images, like "plane", most of the matchings between text tokens and image patches are not meaningful to begin with. Recall that this was one of the reasons why we decided to only regress the teacher's `[I_CLS]` token (see Section 3.2.2.1).

To come to a conclusion, even though Target-CMLI seems to be a promising approach to alleviate the mismatch between text tokens and image patches, especially considering that

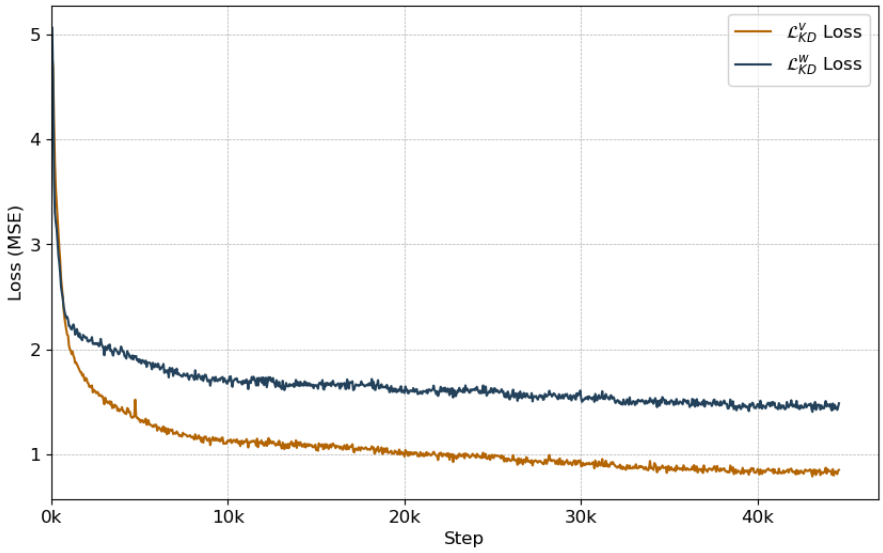


Figure 31: Training loss for the image and text component of the $\mathcal{L}_{\text{KD}}$ loss. The image component $\mathcal{L}_{\text{KD}}^{v}$ shows a significantly lower loss compared to the text component $\mathcal{L}_{\text{KD}}^{t}$, indicating that the [I_CLS] of the teacher still contains image-specific information.

some examples in Figure 30 show a self-attention map that is focused on the object the text token describes, the results are far from consistent, and are constrained to only a few text tokens that have a real-world counterpart in images, and even this are not reliable.

### 3.2.6 Modality-Invariant Targets

Throughout the previous experiments, we have seen that the misalignment between image patches and text tokens leads to problems when regressing the image features of the teacher. While we are already exceeding the performance of the supervised teacher by predicting the [I_CLS] of the teacher, there is still room for improvement. A glance at the loss $\mathcal{L}_{\text{KD}}$, which is now again the previous one (defined again in Equation 86 for ease of access), since Target-CMLI (Section 3.2.5.2) proved to be ineffective, shows that the loss for the image features is clearly lower than that for the text features. This indicates that the [I_CLS] of the teacher still contains image-specific information. If that were not the case, the loss for both components, i.e. $\mathcal{L}_{\text{KD}}^{v}$ and $\mathcal{L}_{\text{KD}}^{w}$, would be similar. Consequently, we aim to introduce a modality-invariant target loss that is less affected by the image-specific information in the [I_CLS] of the teacher.

$$\begin{gathered}\mathcal{L}_{\text{KD}} = \\ \frac{1}{2} * \mathcal{L}_{\text{KD}}^{v} + \frac{1}{2} * \mathcal{L}_{\text{KD}}^{w} = \\ \frac{1}{2} * \|\boldsymbol{h'''}^{s}_{v,K,[\text{I\_CLS}]} - \boldsymbol{h}^{t}_{v,L_t,[\text{I\_CLS}]}\|_2^2 + \frac{1}{2} * \|\boldsymbol{h'''}^{s}_{w,K,[\text{T\_CLS}]} - \boldsymbol{h}^{t}_{v,L_t,[\text{I\_CLS}]}\|_2^2\end{gathered} \tag{86}$$

#### 3.2.6.1 Contrastive Target Loss

We identify the MSE loss, used as a criterion for knowledge distillation, as unfavourable, as it enforces the student representation of the caption $\boldsymbol{h'''}^{s}_{w,K,[\texttt{T\_CLS}]}$ to be the same as the teacher representation of the image $\boldsymbol{h}^{t}_{v,L_t,[\texttt{I\_CLS}]}$. The loss only becomes zero if both are identical. This is less of a problem for $\boldsymbol{h'''}^{s}_{v,K,[\texttt{I\_CLS}]}$, as this is the image representation of the student, which can contain the image-specific information. However, the student is not able to extract the image-specific information from the caption.

To address this issue, we propose a contrastive target loss, which is inspired by contrastive learning. This loss collects all teacher representations from the current batch $\left\{\boldsymbol{h}^{t}_{(v,L_t,[\texttt{I\_CLS}]),b}\right\}_{b=1}^{B}$, and computes the cosine similarity between those representations and the student representation of a candidate caption $\boldsymbol{h'''}^{s}_{w,K,[\texttt{T\_CLS}]}$. Similar to the contrastive loss, we aim to maximize the similarity between the student representation of caption $\boldsymbol{h'''}^{s}_{(v,K,[\texttt{I\_CLS}]),i}$ and the teacher representations of image $\boldsymbol{h}^{t}_{(v,L_t,[\texttt{I\_CLS}]),i}$ in the batch, while minimizing the similarity between the student representation of caption $\boldsymbol{h'''}^{s}_{(v,K,[\texttt{I\_CLS}]),i}$ and the teacher representations of all other images $\boldsymbol{h}^{t}_{(v,L_t,[\texttt{I\_CLS}]),j}, j \neq i$ in the batch.

This has the advantage that we are now focusing less on making $\boldsymbol{h'''}^{s}_{(v,K,[\texttt{I\_CLS}]),i}$ and $\boldsymbol{h}^{t}_{(v,L_t,[\texttt{I\_CLS}]),i}$ identical, but rather on what representations match, and which do not This puts more emphasis on the relative similarity with respect to the similarity to other images in the batch. Maximizing the cosine similarity is further a less strict criterion than minimizing the MSE, as the cosine similarity only requires both representations to be in the same direction, but not necessarily to be identical.

Like in the contrastive loss, if the representations are good enough $\boldsymbol{h'''}^{s}_{(v,K,[\texttt{I\_CLS}]),i}$ and $\boldsymbol{h}^{t}_{(v,L_t,[\texttt{I\_CLS}]),i}$ will have more in common than $\boldsymbol{h'''}^{s}_{(v,K,[\texttt{I\_CLS}]),i}$ and $\boldsymbol{h}^{t}_{(v,L_t,[\texttt{I\_CLS}]),j}$ for $in = j$, so the student representation of the caption $i$ is more likely match the teacher representations of the image $i$.

#### 3.2.6.2 Implementation

The implementation closely follows that of the image-text contrast of Section 1.5.3.2. We concatenate all teacher representations of the image [I_CLS] in the batch to a tensor, and all student representations of the caption [T_CLS] to another tensor:

$$\begin{aligned}\boldsymbol{I}_t &= \left[\boldsymbol{h}^{t}_{(v,L_t,[\texttt{I\_CLS}]),1}, \boldsymbol{h}^{t}_{(v,L_t,[\texttt{I\_CLS}]),2}, ..., \boldsymbol{h}^{t}_{(v,L_t,[\texttt{I\_CLS}]),B'}\right] \in \mathbb{R}^{B' \times D} \\ \boldsymbol{T}_s &= \left[\boldsymbol{h'''}^{s}_{(w,K,[\texttt{T\_CLS}]),1}, \boldsymbol{h'''}^{s}_{(w,K,[\texttt{T\_CLS}]),2}, ..., \boldsymbol{h'''}^{s}_{(w,K,[\texttt{T\_CLS}]),B'}\right] \in \mathbb{R}^{B' \times D}\end{aligned} \tag{87}$$

Additionally, we also collect all image representations of the student [I_CLS] in the batch to a tensor:

$$\boldsymbol{I}_s = \left[\boldsymbol{h}^s_{(v,K,[\texttt{I\_CLS}]),1}, \boldsymbol{h}^s_{(v,K,[\texttt{I\_CLS}]),2}, ..., \boldsymbol{h}^s_{(v,K,[\texttt{I\_CLS}]),B'}\right] \in \mathbb{R}^{B' \times D} \tag{88}$$

We define $B'$ as the combined batch size over all devices, as we can gather all representations from all devices (see Section 3.2.1.2). It therefore holds that $B' = B * P$, where $P$ is the number of devices. In our case we use $P = 2$ devices and $B = 256$ samples per device, resulting in $B' = 2 * 256 = 512$.

The cosine similarity can again be computed efficiently using matrix multiplication of the normalized representations:

$$\begin{aligned} \boldsymbol{L}^w &= \delta(\boldsymbol{T}_s)\delta(\boldsymbol{I}_t)^T \in \mathbb{R}^{B' \times B'} \\ \boldsymbol{L}^v &= \delta(\boldsymbol{I}_s)\delta(\boldsymbol{I}_t)^T \in \mathbb{R}^{B' \times B'} \end{aligned} \tag{89}$$

Here, $\delta$ denotes the normalization:

$$\begin{aligned} \delta(\boldsymbol{X}) &= [\delta(\boldsymbol{x}_1), \delta(\boldsymbol{x}_2), ..., \delta(\boldsymbol{x}_{B'})] \in \mathbb{R}^{B' \times D} \\ \delta(\boldsymbol{x}) &= \frac{\boldsymbol{x}}{\|\boldsymbol{x}\|_2} \in \mathbb{R}^D \end{aligned} \tag{90}$$

Notice how the contrastive target loss is computed both between the student representation of the caption and the teacher representations of the image, which is the important part we covered in the previous section, and between the student representations of the image and the teacher representations of the image. The latter is the image-to-image part of the knowledge distillation loss, which does not suffer from image-specific information in the teacher representation of the image, but we also adapt it to the contrastive loss for consistency.

Analogous to image-text contrast, we define the loss, which is just cross-entropy, as follows:

$$\begin{aligned} \mathcal{L}^{\text{t2i}}_{\text{KD}} &= \frac{1}{B'} \sum_{i=1}^{B'} -\log \frac{\exp\left(L^w_{i,i}\right)}{\sum_{k=1}^{B'} \exp\left(L^w_{i,k}\right)} \\ \mathcal{L}^{\text{i2i}}_{\text{KD}} &= \frac{1}{B'} \sum_{i=1}^{B'} -\log \frac{\exp\left(L^v_{i,i}\right)}{\sum_{k=1}^{B'} \exp\left(L^v_{i,k}\right)} \\ \mathcal{L}_{\text{KD}} &= \frac{1}{2} * \mathcal{L}^{\text{t2i}}_{\text{KD}} + \frac{1}{2} * \mathcal{L}^{\text{i2i}}_{\text{KD}} \end{aligned} \tag{91}$$

We rename the components of the loss from $\mathcal{L}^w_{\text{KD}}$ to $\mathcal{L}^{\text{t2i}}_{\text{KD}}$, and $\mathcal{L}^v_{\text{KD}}$ to $\mathcal{L}^{\text{i2i}}_{\text{KD}}$, in order to reflect the text-to-image and image-to-image parts of contrastive learning, respectively.

When comparing our current configuration to our previous best, which was reached when introducing a token-type embedding after the modality specific encoders (Section 3.2.3), we find that while performance on MSCOCO and Flickr30K retrieval remains unchanged, performance on ImageNet-1K improves by nearly 3 percentage points. This improvement on ImageNet-1K is likely due to our use of CLIP zero-shot classification, which involves retriev-

| KD Loss | ImageNet-1K | Retrieval | |
|---|---|---|---|
| | | MSCOCO | Flickr30K |
| MSE | 30.3 | **67.2** | 78.29 |
| Contrastive | **33.0** | 67.15 | **78.3** |

Table 24: Replacing the MSE loss with the contrastive target loss improves the performance on ImageNet-1K by almost 3 percentage points, while matching the performance on MSCOCO and Flickr30K retrieval.

ing the most similar class prototype using a contrastive loss (see Section 1.7.2.2). Since we now employ a contrastive loss for knowledge distillation, and the weights of the teacher model — responsible for generating one component of the contrastive target loss (the teacher representations $\boldsymbol{I}_t$ of the images) — were originally trained on ImageNet-1K, the student model's learning method aligns more closely with CLIP zero-shot classification. This alignment is likely to enhance performance on ImageNet-1K.

#### 3.2.6.3 Memory Bank

In Section 3.2.4.3, we evaluated the possibility of storing representations, produced by the student, from previous batches in a memory bank. The idea was that since the contrastive loss requires a large number of negative samples to be effective, we could use representations from previous batches as additional negative samples. However, we found that using representations from previous batches leads to a significant performance drop, as the representations were inconsistent.

Fortunately, the contrastive target loss does not suffer from outdated representations. This is because the representations we compare the student representation to are from the teacher. The teacher's weights are frozen during training, meaning that the teacher's representations are consistent over the entire training process. Therefore, all negative examples that are used in the contrastive target loss are consistent with each other, meaning we can savely use a simple memory bank to store the teacher representations from previous batches. Workarounds like a momentum encoder (Section 3.2.4.3.2) or proximal regularization of the features (Section 3.2.4.3.1) are not necessary.

The formulation of the loss does not change, but only the concatenated teacher representations.

$$\boldsymbol{I}_t' = \boldsymbol{I}_t \| \boldsymbol{V} = \left[\boldsymbol{h}^t_{(v,L_t,[\texttt{I\_CLS}]),1}, ..., \boldsymbol{h}^t_{(v,L_t,[\texttt{I\_CLS}]),B'}, \boldsymbol{v}^t_{(v,L_t,[\texttt{I\_CLS}]),1}, ..., \boldsymbol{v}^t_{(v,L_t,[\texttt{I\_CLS}]),G}\right] \in \mathbb{R}^{(B'+G)\times D} \tag{92}$$

$$\boldsymbol{L}^w = \delta(\boldsymbol{T}_s)\delta(\boldsymbol{I}_t')^T \in \mathbb{R}^{B'\times G} \tag{93}$$

We denote $\boldsymbol{v}^t_{(v,L_t,[\texttt{I\_CLS}]),i}$ as the teacher representation of the image $i$ from the memory bank, so from a previous batch, and $G$ as the number of representations stored in the memory

| 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** |
| S-SMKE | 51.66 | 79.9 | 88.66 | 36.17 | 66.55 | **78.28** | 64.5 | 88.4 | 93.0 | 51.78 | 78.54 | 86.46 |
| S-SMKE$_{\text{CTL}}$ | 52.68 | 80.56 | 88.3 | 36.5 | 66.58 | **78.28** | 69.0 | 89.2 | 94.3 | 51.48 | 79.18 | 86.66 |
| S-SMKE$_{\text{CTL_MB}}$ | 53.54 | 81.1 | **89.52** | 35.65 | 66.0 | 77.77 | 70.9 | 92.1 | 96.0 | 52.72 | 80.2 | 87.46 |

Table 25: A contrastive target loss with memory bank especially improves text retrieval, while the performance on COCO image retrieval degrades slightly. S-SMKE$_{\text{CTL}}$ denotes the contrastive target loss without memory bank, and S-SMKE$_{\text{CTL_MB}}$ denotes the contrastive target loss with memory bank.

bank, i.e. the size. We set $G = 65536$ in our experiments, which we orientate on the ideal size found by MoCo [He+19] for contrastive learning (see Figure 26).

A detailed look on retrieval performance in Table 25 shows a gain especially on text retrieval tasks, and we are able to increase text retrieval on Flickr30K by approx. 2-3 percentage points in each metric. This is surprising, as we would expect an increase in image retrieval performance, because the contrastive target loss uses text-image retrieval, expressed through the loss $\mathcal{L}_{\text{KD}}^{\text{t2i}}$ in Equation 91. This should have a positive effect on text-image retrieval in COCO and Flickr30K, but we instead observe a slight decrease in performance. Simultaneously, the performance on image-text retrieval increases consistently, even though the contrastive target loss does not use image-text retrieval. We would expect the opposite.

The performance on ImageNet-1K is increasd by an impressive 4 percentage points to 37% (33% before). Considering the increase in image-text retrieval on Flickr30K and COCO, this is actually to be expected: In CLIP zero-shot classification, a candidate image is compared to all class prototypes, which have been created from text descriptions of the classes. The class prototype, which is a text representation, with the highest similarity to the image is then chosen as the predicted class, which is essentially image-text retrieval. Therefore, if the performance on image-text retrieval increases, we would expect an increase in performance on ImageNet-1K.

## 3.2.7 Quantizing Visual Features

### 3.2.7.1 Motivation

Even though we were able to reduce the impact of the image-specific information in the teacher's [I_CLS] token by introducing the contrastive target loss, the problem remains the same. A repeated glance on the comparison between the training loss for the image and text component of the $\mathcal{L}_{\text{KD}}$ loss (Figure 32), which is now based on a contrastive loss with memory bank, shows that the loss for the image component $\mathcal{L}_{\text{KD}}^{\text{i2i}}$ is still significantly lower than

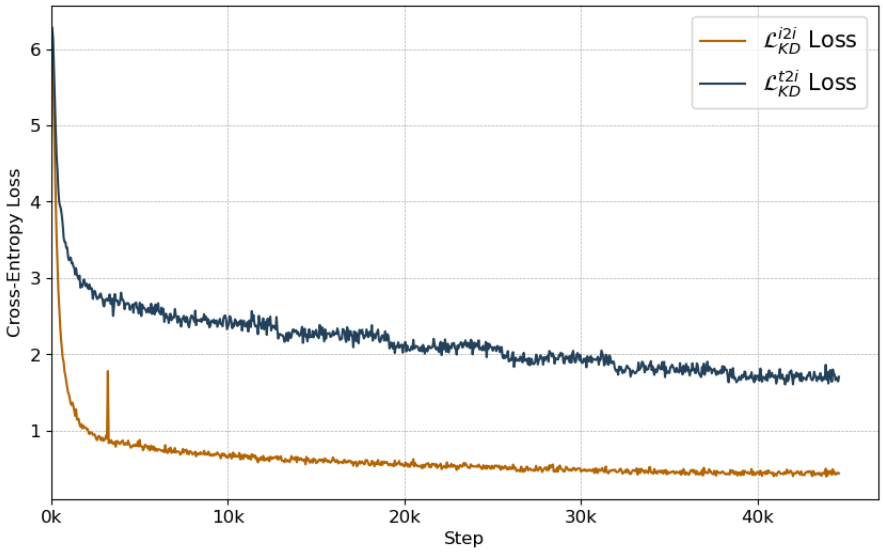


Figure 32: Training loss for the image and text component of the $\mathcal{L}_{\text{KD}}$ loss. Even though the contrastive target loss is able to increase the performance of the model, the loss for the image component $\mathcal{L}_{\text{KD}}^{\text{i2i}}$ is still significantly lower than that for the text component $\mathcal{L}_{\text{KD}}^{\text{t2i}}$.

that for the text component $\mathcal{L}_{\text{KD}}^{\text{t2i}}$. While our approach is able to achieve impressive results even with this imbalance, we push the boundaries by introducing an additional component that aims to further reduce the impact of the image-specific information.

The paper "Neural Discrete Representation Learning" by van den Oord et al. [OVK17] first introduced the concept of quantizing images to discrete values in order to learn representations. The idea is to learn a so-called codebook that contains a finite set of prototype representations/embeddings. An autoencoder architecture is then used to first encode and compress the image to a continuous representation, which could be a sequence of representations for vision Transformers, or a set of 2D feature maps for convolutional neural networks.

Staying at the Transformer architecture, each token of the sequence, which represents an image patch, is then compared to all prototype representations in the codebook using e.g. consine similarity. The closest prototype to each token is then selected, and each token is replaced by the prototype representation it is closest to. Based on this sequence of prototype representations, the decoder is then trained to reconstruct the original image. During training, each prototype is updated using for example the mean over all patch representations that were closest to it. This is similar to the k-means algorithm, and is called vector-quantization Section 3.2.7 [Pen+22].

The consequence is that it is possible to discretize patch-level representations, which are usually continuous, to a finite set of prototype representations. This technique is using in papers like BEiTv2 [Pen+22] to perform e.g. masked image modeling to learn visual representations [Bao+22, Pen+22].

We aim to apply vector-quantization not on the patch-level representations, but on the image-level representation [I_CLS]. The goal is discretize the representation space of a self-supervised image model, such as the teacher we use, to obtain a set of prototype embeddings we can map the representations produced by the model to. This way, we could find something similar to classes produced by a supervised model, but without the need for labels, which should remove large amounts image-specific information. Intuitively, this can be seen as some form of clustering the representations produced by a self-supervised model, and closely resembles applying k-means clustering on representations produced by a model from a set of images.

#### 3.2.7.2 Method

To achieve this, we first take our pretrained teacher model BEiTv2 [Pen+22], pass an image through it, and extract the representation $\boldsymbol{h}^t_{v,L_t,[\texttt{I\_CLS}]} \in \mathbb{R}^{768}$ of the [I_CLS] token. This functions as the representation of the image, and the weights of the teacher remain frozen throughout the whole process. The representation is then passed through a learnable linear projection $\boldsymbol{W}_{q\downarrow} \in \mathbb{R}^{D\times S}$ that projects the representation into a lower-dimensional space with $s$ dimensions, where $D$ is the dimensionality of the original representation.

The projected representation $\boldsymbol{h}^t_{v,L_t,[\texttt{I\_CLS}]}\boldsymbol{W}_{q\downarrow}$ is then compared to all prototype embeddings $\left\{\boldsymbol{q}_j\right\}_{j=1}^{J}$ in the codebook $\boldsymbol{Q} \in \mathbb{R}^{J\times S}$ using cosine similarity, and the index $m$ of the closest prototype is extracted.

$$m = \underset{1\leq s\leq S}{\operatorname{argmin}}\, \delta\Big(\boldsymbol{h}^t_{v,L_t,[\texttt{I\_CLS}]}\boldsymbol{W}_{q\downarrow}\Big)\delta(\boldsymbol{q}_s)^T \tag{94}$$

We use the projection $\boldsymbol{W}_{q\downarrow}$, because this already reduces the dimensionality of the representation, forcing only the most important information to be kept. At best, pixel-level information is completely removed. Furthermore, a projection to a lower-dimensional space has shown to alleviate the problem of codebook collapse, where all representations are mapped to only a few, or even just one, prototype [Pen+22]. If codebook collapse occurs, the model is not able to learn distinct representations, and there would not be a set of prototypes that represent different semantic concepts, like classes in a supervised model.

The index $m$ of the closest prototype is then used to select the corresponding prototype embedding $\boldsymbol{q}_m$ from the codebook $\boldsymbol{Q}$, which is then projected back to the original embeddings dimension $D$ using yet another learnable linear projection $\boldsymbol{W}_{b\uparrow} \in \mathbb{R}^{S\times D}$. The projected prototype embedding $\boldsymbol{q}_m\boldsymbol{W}_{b\uparrow}$ is then concatenated with the original image representation and input to the Transformer layers of the BEiTv2 model, and serves as the [I_CLS] token.

Then, each patch/token representation of the image $\left\{\boldsymbol{h}^t_{v,0,n}\right\}_{n=1}^{N}$ is replaced by a special learnable mask token $\boldsymbol{h}_{[\texttt{MASK}]}$ with a certain probability $p$. We sample the probability from a uniform distribution $\alpha \sim U(0,1)$:

$$\Omega(\boldsymbol{x}) := \begin{cases} \boldsymbol{h}_{\texttt{[MASK]}} & \text{if } p > \alpha \\ \boldsymbol{x} & \text{else} \end{cases}$$
$$\boldsymbol{H}'_{v,L_t} = \left[\boldsymbol{q}_m \boldsymbol{W}_{b\uparrow}, \Omega\left(\boldsymbol{h}^t_{v,0,1}\right), ..., \Omega\left(\boldsymbol{h}^t_{v,0,N}\right)\right] \tag{95}$$

Let's break down why we do this: The idea is that $\boldsymbol{H}'_{v,K}$ becomes input to a set of Transformer layers, which form the decoder. The task of the decoder is to reconstruct the original representation $\boldsymbol{h}^t_{v,L_t,\texttt{[I\_CLS]}}$ of the image, produced by the frozen teacher. Because of self-attention in the Transformer layers, the decoder is able to use both the content and the spacial information of the unmasked tokens to reconstruct the teacher's [I_CLS] token. To make the whole process rely on quanized global image representation $\boldsymbol{q}_m$, we select $p$ to a values close to 1, which is $p = 0.9$ in our case, so 90% of the tokens are replaced by the mask token. This way, the decoder will not be able to adequately reconstruct the image representation $\boldsymbol{h}^t_{v,L_t,\texttt{[I\_CLS]}}$ without fully utilizing the quantized global image representation $\boldsymbol{q}_m$. The only way through which high level information about the image content can reach the decoder is through $\boldsymbol{q}_m$, which should force the model to learn a set of prototypes that represent different semantic concepts in the image. The few unmasked tokens should help the model to have access to (1) spacial information, and (2) low-level information about the image content, which should not be captured by $\boldsymbol{q}_m$, and is generally not needed in $\boldsymbol{q}_m$, because the unmasked tokens provide this information.

Say an image contains a picture of a sheep on a meadow with a blue sky in the background. Since $\boldsymbol{q}_m$ can only represent relatively little information (due to its low dimensionality and discrete nature), the model may only compress the most important information, like the sheep, the meadow, and the sky, into $\boldsymbol{q}_m$. The few image patches that are not masked out might contain information about the spacial arrangement, or any other information originally captured by $\boldsymbol{h}^t_{v,L_t,\texttt{[I\_CLS]}}$. Those information should be image-specific. That way, the model should be able to learn a set of prototypes that represent different semantic concepts in the image. Examples of masking random image patches are illustrated in Figure 45.

The training objective is then:

$$\begin{aligned} \mathcal{L}_{I-VQ} = & \\ & 1 - \delta\left(\boldsymbol{h}^t_{v,F,\texttt{[I\_CLS]}}\right)\delta\left(\boldsymbol{h}^t_{v,L_t,\texttt{[I\_CLS]}}\right)^T \\ & + \|\delta\left(\boldsymbol{h}^t_{v,L_t,\texttt{[I\_CLS]}}\boldsymbol{W}_{q\downarrow}\right) - sg[\delta(\boldsymbol{q}_m)]\|_2^2 \end{aligned} \tag{96}$$

Here, $\boldsymbol{h}^t_{v,F,\texttt{[I\_CLS]}}$ denotes the decoder output for the [I_CLS] token, which is the one we want to reconstruct. $F$ denotes the number of all Transformer layers of the encoder, the frozen BEiTv2 model, and the decoder. It holds that $F = L_t + U$, where $U$ is the number of Transformer layers in the decoder. Since for our teacher it holds that $L_t = 12$, we have $F = 12 + U$.

The first term, $1 - \delta\left(\boldsymbol{h}^t_{v,F,[\texttt{I\_CLS}]}\right)\delta(\boldsymbol{q}_m)^T$, aims to maximize the cosine similarity between the reconstructed [I_CLS] token and the original representation $\boldsymbol{h}^t_{v,L_t,[\texttt{I\_CLS}]}$ for the image, produced by the teacher. If the cosine similarity is maximized, so 1.0, then this part becomes 0. This is the reconstruction loss. The other component aims to minimize the distance between the projected representation $\boldsymbol{h}^t_{v,L_t,[\texttt{I\_CLS}]}\boldsymbol{W}_{q\downarrow}$ and the closest, and therefore selected, prototype $\boldsymbol{q}_m$. It is the mean squared error.

The form $sg[\cdot]$ denotes a stop-gradient operation. This operation is required, because the codebook lookup, i.e. replacing the representation $\boldsymbol{h}^t_{v,L_t,[\texttt{I\_CLS}]}\boldsymbol{W}_{q\downarrow}$ with the closest prototype $\boldsymbol{q}_m$, is not differentiable. It is not possible to compute gradients that express how $\boldsymbol{W}_{q\downarrow}$, $\boldsymbol{W}_{q\uparrow}$ and $\boldsymbol{Q}$ should be updated to minimize the loss, since the lookup cannot be expressed as a continuous function.

The solution is to compute the loss, and therefore the gradients, only with respect to the encoder and decoder weights. Since the encoder weights are frozen, gradients only need to be computed for $\boldsymbol{W}_{q\downarrow}$, $\boldsymbol{W}_{q\uparrow}$, and the decoder weights. We treat the selected codebook embedding $\boldsymbol{q}_m$ as constant, meaning that we do not compute gradients for it. That is why in the loss, $sg[\cdot]$ turns its input into a constant, and therefore avoids gradient computation for $\boldsymbol{q}_m$.

The problem this creates is that the gradients flow through the decoder and $\boldsymbol{W}_{q\uparrow}$, but are they "cut off", leading to no update of the projection $\boldsymbol{W}_{q\downarrow}$, and the codebook $\boldsymbol{Q}$. To solve this, the gradients with respect to $\boldsymbol{q}_m$ are copied to $\boldsymbol{h}^t_{v,L_t,[\texttt{I\_CLS}]}\boldsymbol{W}_{q\downarrow}$, bypassing how $\boldsymbol{q}_m$ was selected. For the weight update of $\boldsymbol{W}_{q\downarrow}$ this essentially means $\boldsymbol{h}^t_{v,L_t,[\texttt{I\_CLS}]}\boldsymbol{W}_{q\downarrow}$ and $\boldsymbol{q}_m$ its "replacement" are treated as the same.

This loss however does not lead to an optimization of the codebook $\boldsymbol{Q}$, as, again, $\boldsymbol{q}_m$ is treated as constant whose origin is basically unknown to the model. This is done on purpose, as each codebook embedding is optimized/updated as an exponential moving average (EMA) of the representations in the current batch (size $B$) it replaced. For a single codebook embedding $\boldsymbol{q}_j$, this set can be defined as:

$$\boldsymbol{A}_j = \left\{ \left(\boldsymbol{h}^t_{v,L_t,[\texttt{I\_CLS}]}\boldsymbol{W}_{q\downarrow}\right)_i \mid i \in [1, B] \wedge m_i = j \right\} \tag{97}$$

Here, $m_i$ is the index of the closest prototype for the $i$-th image in the batch. The update of the codebook embedding $\boldsymbol{q}_j$ is then:

$$\boldsymbol{q}_j = d * \boldsymbol{q}_j + (1 - d) * \frac{1}{Z}\sum_{z=1}^{Z}\left(\boldsymbol{A}_j\right)_z \tag{98}$$

The codebook embedding $j$ is updated with the mean over all representations in the batch it was closest to. $d$ denotes the decay factor of the EMA update, normally specified as $m$. However, we use $d$ to avoid confusion with the index $m$ of the closest prototype. We show can illustration of the concept in Figure 33. The loss formulation in Equation 96, as well as the EMA update, is heavily inspired by the vector quantization process in BEiTv2 [Pen+22].

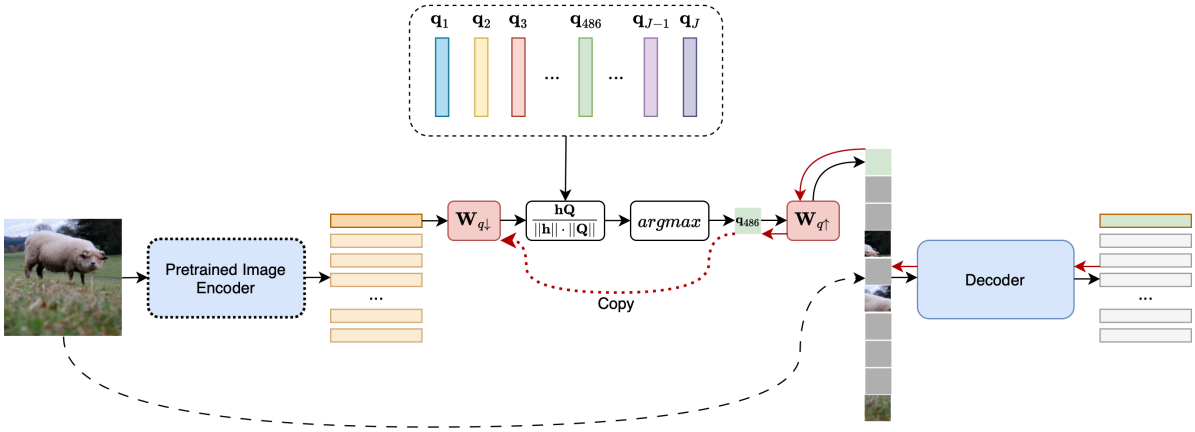


Figure 33: Illustration of the vector quantization of a self-supervised image model's representation space. Models with dotted borders are frozen, and red lines indicate gradient flow. Opaque outputs, in this case patch representations, are ignored in the loss computation. The figure is inspired by papers "Neural Discrete Representation Learning" [OVK17], and BEiT [Bao+22], heavily oriented on the depiction of the vector quantization process in BEiTv2 [Pen+22], and the image stems from the COCO test set [Lin+14].

#### 3.2.7.3 Training Image VQ

For training, we use an embedding dimension of the codebook of $S = 16$. For comparison, BEiTv2 uses $S = 32$ for its patch-level codebook. As might have come apparent from the previous section, we heavily orient on the vector quantization process in BEiTv2 [Pen+22], which is why we originally used $S = 32$ as well. However, we found that this led to codebook collapse in preliminary experiments, which is why we opt for a lower dimensionality. We experiment with two different codebook sizes, $J = 1024$ and $J = 8192$. The former is motivated by the fact that we aim to learn a set of prototypes that represent different semantic concepts in the image, like classes in a supervised model. Since we BEiTv2, which is our encoder, was pretrained on ImageNet-1K [Pen+22], we orient the number of possible concepts on the number of classes in ImageNet-1K, which is 1000 [Rus+15]. The latter is used to investigate whether a larger codebook size can still capture semantic concepts without image-specific information.

For the decoder, we merely use a single Transformer layer initialized from scratch. Few decoder layer means lower capacity to reconstruct the image representation, which should force the model to rely even more on the quantized global image representation $\boldsymbol{q}_m$. This is also done in BEiTv2, where the authors experiment with a single, and three decoder layers. Consequently, for us it holds that $U = 1$, and $F = L_s + U = 12 + 1 = 13$. For the decay factor, we again orient on BEiTv2, and use $d = 0.99$. As mentioned in the previous section, we use a masking probability of $p = 0.9$. All hyperparameters, including the ones described here, are shown in Table 31.

We train the model for 10 epochs on ImageNet-1K, and use a relatively large learning rate of $1e-3$. We do this, because we use two GPUs with a combined batch size of 512, which is relatively large. Further, we do not have have a large number of trainable parameters, only the projection matrices $\boldsymbol{W}_{q\downarrow}$, $\boldsymbol{W}_{q\uparrow}$, and the weights of one Transformer layer (decoder). Simultaneously, we use a frozen teacher model, which acts as a feature extractor and provides high quality representations.

After each epoch, we validate the loss $\mathcal{L}_{I-VQ}$ on the validation set of ImageNet-1K, and, most importantly, calculate the codebook usage over the whole validation set to check for codebook collapse.

#### 3.2.7.4 Training S-SMKE with Image VQ

## 3.2.8 Teacher Ablation Studies

## 3.2.9 Limitations and Insights

While the proposed methods is an efficient way to traing comparatively small multimodal models, and can easily be adapted to other modalities, e.g. audio, it has two main limitations.

First, our method relies on knowledge distillation of a self-supervised image model as the teacher. The fact that there has been no incentive for the teacher to learn a representation that is independent of the image modality makes it difficult to learn a representation that is truly modality-invariant and aligned across the modalities of the student model. This has repeatedly been shown when comparing the loss between image-to-image and text-to-image distillation, where the former is consistently lower. Interestingly, we were still able to outperform the approach of a supervised teacher, showing that even though the ImageNet-1K classes, which we predict (with KL-Divergence, see Section 3.2.1), are independent of the image modality, they might not capture the content of an image's caption better than an image-specific representation (used with the self-supervised teacher), which is what we first assumed.

A glance at the comparison between the components of the knowledge distillation loss when using a supervised teacher (so KL-Divergence) also shows that this approach suffers from the same problem as when using a self-supervised teacher (see Figure 34). Here, the KL-Divergence for the image-to-image loss $\mathcal{L}^{v}_{\mathrm{KD}}$ is also consistently lower than for the text-to-image loss $\mathcal{L}^{w}_{\mathrm{KD}}$, and the loss components between both approaches (Transformer SHRe and S-SMKE) generally perform very similar.

We conclude that using an unimodal teacher in general is a limitation regarding alignment, and therefore performance. Still, due to the fact that it does not require a multimodal teacher, the approach is far more flexible and can be applied to any modality, as long as a suitable teacher (of one of the modalities used) is available.

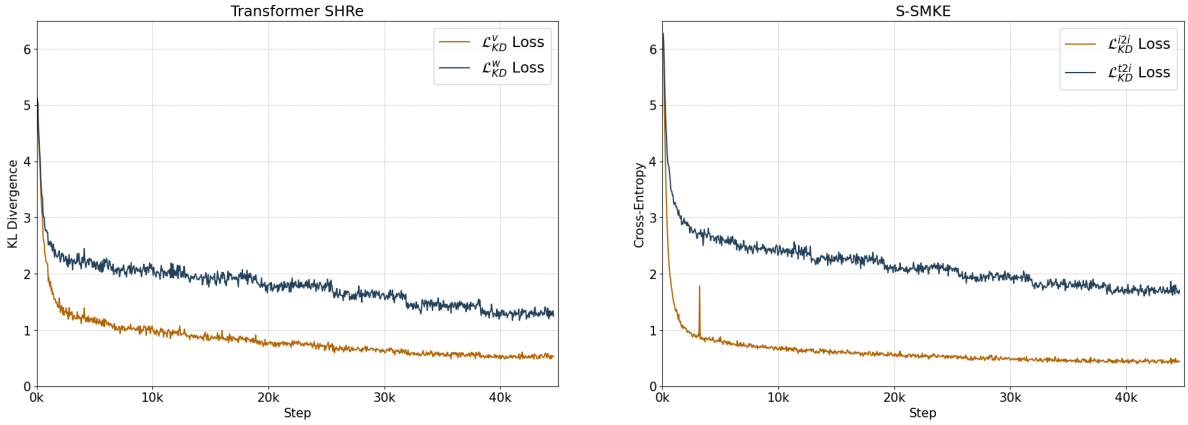


Figure 34: Both approaches, the supervised as well as the self-supervised (ours), show that the image component is consistently lower than the text component of the knowledge distillation loss. Moreover, the loss behavior troughout the training process is very similar between both approaches, although the loss functions are different.

Second, S-SMKE (and (Transformer) SHRe) processes image and text seperately, in a forward pass for the image and a forward pass for the text. This is similar to CLIP [Rad+21] (see Section 1.7.2). Because there is no attention mechanism between individual image patches and text tokens, both approaches miss a fine-grained alignment between the modalities. Even though our model performs quite well on the retrieval task, even outperforming well-established research papers on some metrics, there is still room for improvement. However, we believe that there is not much more performance to gain for our approach. Wrong retrievals are often still similar to the query, and only differ in little token-level details. Since our representations are based on the global content of the image and text, those details are not captured in most cases, leading to a “false” retrieval. Examples of this can be seen with example retrievals on the full MSCOCO test set in /@, which is the exact dataset we publish our results on (previous visualizations are based on 1k subsets as in SHRe [AVT17], and are therefore a simpler task since there are less possible retrieval candidates).

At this point it has to be noted that when it comes to a production-grade retrieval system, e.g. a system for searching images by text, the retrievals of our model can still be considered as correct, or “good enough”, as most of them, even though they are flagged as “false” under the evaluation metrics, are still semantically related to the query.

Still, the problem remains, and is actually relevant when it comes to applications where the fine-grained details are crucial. This includes tasks like visual question answering and visual reasoning, benchmarked by the datasets VQAv2 [Agr+16] and NLVR2 [Suh+19], respectively.

For instance, in NLVR2, the task is to decide whether a given sentence about an image is either true or false. The sentence, so the statement about the image, often focuses on fine-grained details, for which there is no guarantee that they are captured just by global repre-

sentations. Furthermore, the statement can often not even be considered as a caption, as is can also be a false statement about the image. Examples of this can be seen in Figure 46.

That simply aligning global representations is not enough to excel in such tasks was also shown by the authors of ViLT [KSK21], who were one of the first to propose a model that aligns image and text on a fine-grained level. They found that when finetuning CLIP [Rad+21] on the NLVR2 dataset, using the dot-product of the global representations of image and statement as the binary classifier, the model achieved an accuracy of only 51% [KSK21]. Considering that the task is a binary classification, and both classes are equally distributed, this is only slightly better than random guessing and therefore unusable for practical applications. Since our model works on a similar level of alignment, the same will likely apply with our approach.

## 3.3 Fine-Grained Alignment

## 3.4 Finetuning on Downstream Tasks

## 3.5 Ablation Study: Reduction of Data

## 3.6 Ablation Study: Scaling up the final Approach

## 3.7 Discussion of Results

# 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 $\varepsilon$ | 1e-06 |
| | AdamW $\beta$ | (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 26: Hyperparameters used for distilling a Data2Vec2 image model.

| Type | Hyperparameters | ImageNet | | CIFAR10 | | CIFAR100 | |
|---|---|---|---|---|---|---|---|
| | | Finetune | Linear probe | Finetune | Linear probe | Finetune | Linear probe |
| **Training** | Num classes | 1k | 1k | 10 | 10 | 100 | 100 |
| | Epochs | 15 | | | | | |
| | Batch size | 256 | | | | | |
| | Optimizer | AdamW | | | | | |
| | AdamW $\varepsilon$ | 1e-8 | | | | | |
| | AdamW $\beta$ | (0.9, 0.999) | | | | | |
| | Weight decay | 0.01 | | | | | |
| | Base learning rate | 1e-3 | | | | | |
| | Layer Decay | 0.81 | - | 0.75 | - | 0.75 | - |
| | 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 27: 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 | Values |
|---|---|---|
| **Model** | Layers | 6 |
| | Hidden size | 768 |
| | FFN inner hidden size | 3072 |
| | Attention Heads | 12 |
| | Max sequence length | 256 |
| | Vocabulary size | 30522 |
| **Training** | Epochs | 1 |
| | Total steps | 1M |
| | Batch size | 256 |
| | Optimizer | AdamW |
| | AdamW $\varepsilon$ | 1e-06 |
| | AdamW $\beta$ | (0.9,0.98) |
| | Weight decay | 0.01 |
| | Base learning rate | 1e-4 |
| | Learning rate schedule | Cosine |
| | Warmup steps | 10k (1% of total steps) |
| | Hardware | 1 × RTX 4090 24GB |

Table 28: Hyperparameters used for distilling a BERT model.

| Type | Hyperparameters | MNLI | QNLI | RTE | MRPC | QQP | STS-B | CoLA | SST |
|---|---|---|---|---|---|---|---|---|---|
| **Training** | Num classes | 3 | 2 | 2 | 3 | 2 | 1 (Regression) | 2 | 2 |
| | Head Dropout prob. | 0.2 | 0.2 | 0.1 | 0.1 | 0.1 | 0.1 | 0.1 | 0.1 |
| | Epochs | 10 | 10 | 20 | 20 | 10 | 20 | 20 | 10 |
| | Total steps | 61360 | 32733 | 3113 | 5095 | 31563 | 7187 | 10689 | 21047 |
| | Batch size | 64 | 32 | 16 | 16 | 128 | 16 | 16 | 32 |
| | Optimizer | AdamW | | | | | | | |
| | AdamW $\varepsilon$ | 1e-6 | | | | | | | |
| | AdamW $\beta$ | (0.9, 0.98) | | | | | | | |
| | Weight decay | 0.1 | | | | | | | |
| | Base learning rate | 2e-5 | 2e-5 | 2e-5 | 2e-5 | 2e-5 | 4e-5 | 1e-5 | 2e-5 |
| | Learning rate schedule | Polynomial decay | | | | | | | |
| | Warmup steps | 10% of total steps | | | | | | | |
| | Metric | Accuracy | Accuracy | Accuracy | F1 | F1 | Spearman | Accuracy | Accuracy |
| | Hardware | 1 × RTX 4090 24GB | | | | | | | |

Table 29: Hyperparameters for the GLUE [Wan+19] benchmark tasks of the distilled BERT model.

| Type | Hyperparameters | Values |
|---|---|---|
| **Model** | Image/Text layers | 6 (Transformer) |
| | Shared layers | 3 (MLP) |
| | Hidden size | 768 |
| | FFN inner hidden size | 3072 |
| | Attention Heads | 12 |
| | Patch size | 16×16 |
| | Input resolution | 224×224 |
| | Max. caption length | 64 |
| **Training** | Epochs | 7 |
| | Total steps | 89273 |
| | Batch size | 256 |
| | Optimizer | AdamW |
| | AdamW $\varepsilon$ | 1e-06 |
| | AdamW $\beta$ | (0.9,0.98) |
| | Weight decay | 0.01 |
| | Base learning rate | 1e-4 |
| | Learning rate schedule | Cosine |
| | Warmup steps | 8927 (10% of total steps) |
| | Hardware | 1 × RTX 4090 24GB |
| **Augmentations** | Horizontal flipping prob. | 0.5 |
| | RandomResizeCrop range | [0.9, 1.0] |

Table 30: Hyperparameters used for training the Transformer SHRe model.

| Type | Hyperparameters | Values |
|---|---|---|
| **Model** | Encoder | BEiTv2 ViT-B/16 |
| | # Decoder layers | 1 |
| | Codebook size | {1024×16, 8192×16} |
| | Patch size | 16×16 |
| | EMA decay | 0.99 |
| | Mask prob. | 0.9 |
| **Training** | Epochs | 10 |
| | Total steps | 50040 |
| | Batch size | 256 |
| | Optimizer | AdamW |
| | AdamW $\varepsilon$ | 1e-06 |
| | AdamW $\beta$ | (0.9,0.98) |
| | Weight decay | 0.01 |
| | Base learning rate | 1e-3 |
| | Learning rate schedule | Cosine |
| | Warmup steps | 5004 (10% of total steps) |
| | Hardware | 2 × RTX 4090 24GB |
| **Augmentations** | Horizontal flipping prob. | 0.5 |
| | RandomResizeCrop range | [0.5, 1.0] |

Table 31: Hyperparameters used for training the image vector quantizer.

# 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
```

Code 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.

```
# model: pretrained (e.g. distilled) model
# bert_pooler: pretrained BERT pooler layer with tanh activation
# dropout: dropout layer
# cls_head: roberta classification head
# x: batch text tokens (B, M+2) -> M: max sequence length, 2: cls and sep
token
# target: batch of labels (B, )
# regression: boolean flag for regression tasks
# metric: either Accuracy, F1, or Spearman correlation
def glue_forward(model, bert_pooler, dropout, cls_head,
                 x, target, regression, metric):

    x = model(x) # (B, M+2, D)

    x = x[:, 0] # take cls token (B, D)
    x = dropout(bert_pooler(x)) # (B, D)
    x = cls_head(x) # (B, C) -> C: number of classes

    if regression:
      x = x.squeeze() # (B, ) -> remove the last dimension, as C=1
      pred = x
      loss = mse_loss(x, target)
    else:
      pred = x.argmax(dim=-1) # (B, )
      loss = cross_entropy(x, target)

    score = metric(pred, target)

    return loss, pred, score
```

Code 2: Pytorch pseudocode for the forward pass during finetuning on GLUE benchmark tasks.

```
# teacher_model: ResNet-50-A1 model
# image_encoder: Image encoder of the multimodal student model
# text_encoder: Text encoder of the multimodal student model
# shared_encoder: Shared encoder of the multimodal student model
# imgs: batch of images (B, 3, H, W)
# captions: batch of image captions (B, 64)
# kl_div: KL-Divergence
# clip_loss: Contrastive loss used in CLIP
def forward(teacher_model, image_encoder, text_encoder,
            shared_encoder, imgs, captions):

    with torch.no_grad():
      target = teacher_model(imgs) # (B, 1000)

    img_layer_res = shared_encoder(image_encoder(imgs)[:, 0])
    # [(B, 768), (B, 3072), (B, 1000)]

    text_layer_res = shared_encoder(text_encoder(captions)[:, 0])
    # [(B, 768), (B, 3072), (B, 1000)]

    kl_loss = 1/2*kl_div(target, img_layer_res[2]) +
              1/2*kl_div(target, text_layer_res[2])

    itc_loss = 1/3*clip_loss(img_layer_res[0], text_layer_res[0]) +
               1/3*clip_loss(img_layer_res[1], text_layer_res[1]) +
               1/3*clip_loss(img_layer_res[2], text_layer_res[2])

    loss = kl_loss + itc_loss

    return loss
```

Code 3: Abstract code used in the forward pass for distilling the multimodal Transformer SHRe from a pretrained ResNet-50-A1 model.

# AC Figures and Visualizations

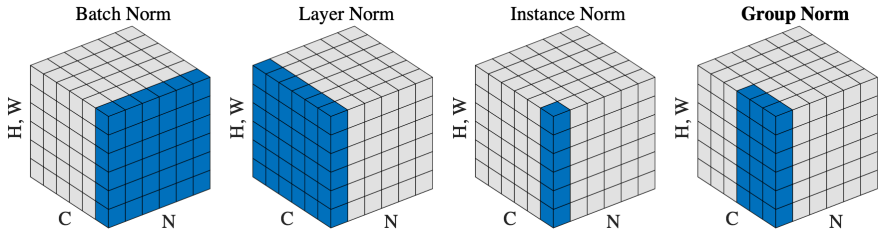


Figure 35: Comparison of different normalization operations on the example of images. Dimension “H, W” refers to the height and width of the input, “C” to the number of channels or embedding dimensions, and “N” to the number of samples, i.e. the batch dimension. Since we are working with sequences of 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].

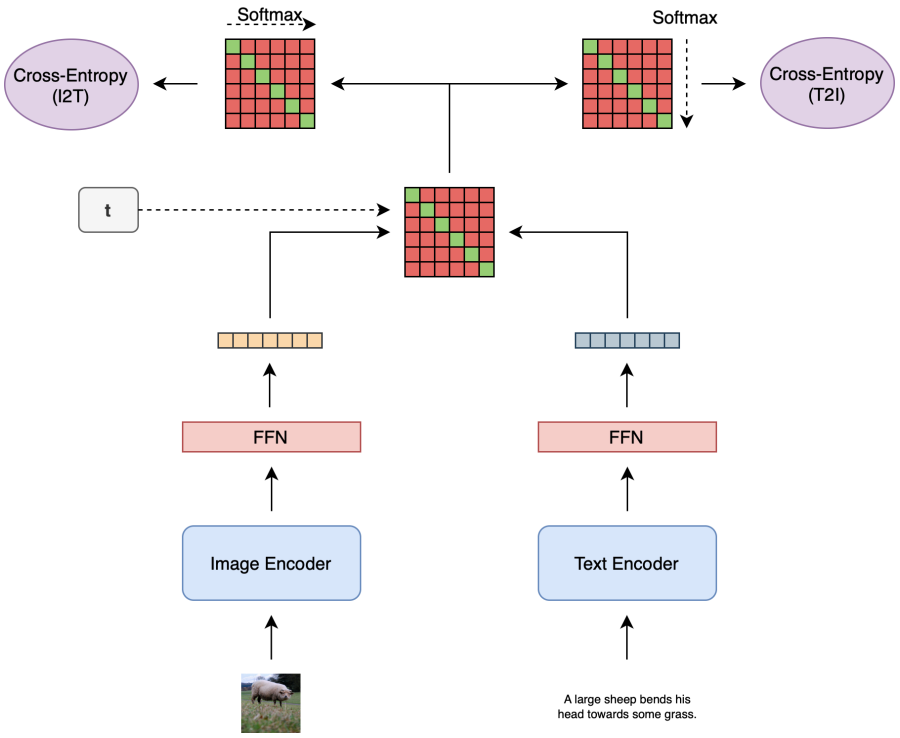


Figure 36: Illustration of CLIP training. A batch of image-text pairs is passed through the model and embedded into a shared latent space. The cosine similarity between all pairs is computed and softmax-normalized to calculate the image-to-text and text-to-image loss. The final loss is the mean of both 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.

| Dataset | Example | Label |
|---|---|---|
| CoLA | [CLS] Our friends won't buy this analysis, let alone the next one we propose. [SEP] | 1 |
| SST-2 | [CLS] hide new secretions from the parental units [SEP] | 0 |
| MRPC | [CLS] Amrozi accused his brother, whom he called "the witness", of deliberately distorting his evidence. [SEP] Referring to him as only "the witness", Amrozi accused his brother of deliberately distorting his evidence. [SEP] | 1 |
| STS-B | [CLS] A plane is taking off. [SEP] An air plane is taking off. [SEP] | 5.0 |
| QQP | [CLS] How is the life of a math student? Could you describe your own experiences? [SEP] Which level of preparation is enough for the exam "jlpt5"? [SEP] | 0 |
| MNLI | [CLS] Conceptually cream skimming has two basic dimensions - product and geography. [SEP] Product and geography are what make cream skimming work. [SEP] | 1 |
| QNLI | [CLS] When did the third Digimon series begin? [SEP] Unlike the two seasons before it and most of the seasons that followed, Digimon Tamers takes a darker and more realistic approach to its story featuring Digimon who do not reincarnate after their deaths and more complex character development in the original Japanese. [SEP] | 1 |
| RTE | [CLS] No Weapons of Mass Destruction Found in Iraq Yet. [SEP] Weapons of Mass Destruction Found in Iraq. [SEP] | 1 |

Table 32: Training examples of the GLUE benchmark tasks (one example per task). Examples are taking from the GLUE dataset card on Hugging Face[15].

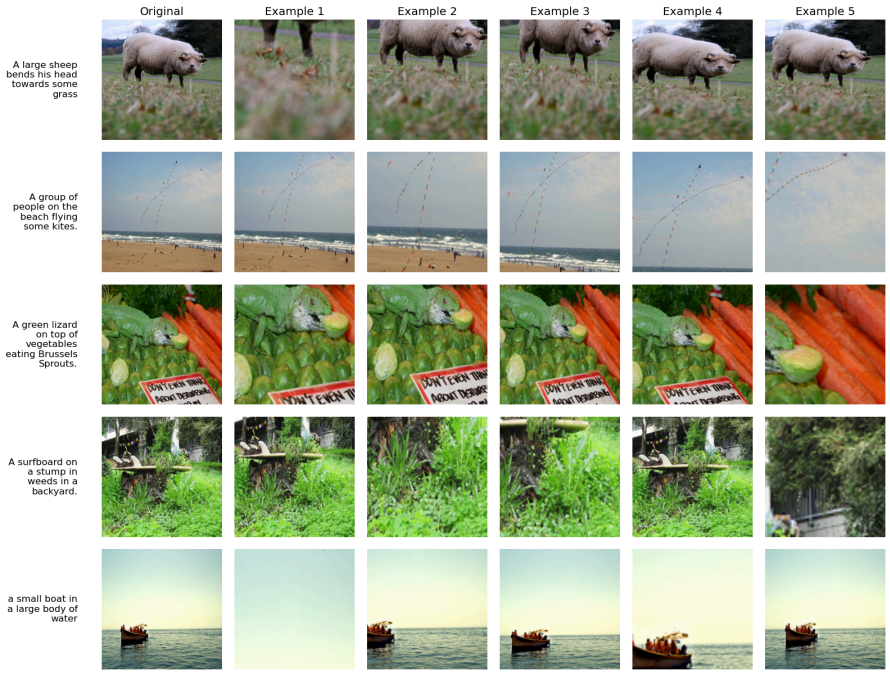


Figure 37: A small lower bound (8%) for a random crop erases high-level semantic features which are important for aligning text and image. Image-text pairs have been taken from the COCO train set [Lin+14].

[15] https://huggingface.co/datasets/nyu-mll/glue

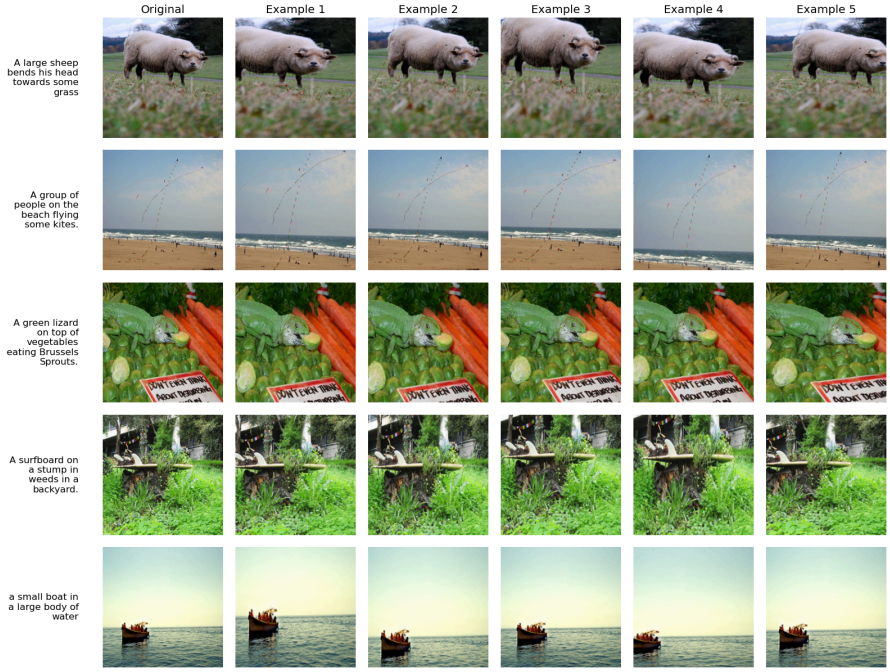


Figure 38: A larger lower bound (90%) for a random crop retains high-level semantic features. Notice how this is not the case for very low values, as shown in Figure 37. Image-text pairs have been taken from the COCO train set [Lin+14].

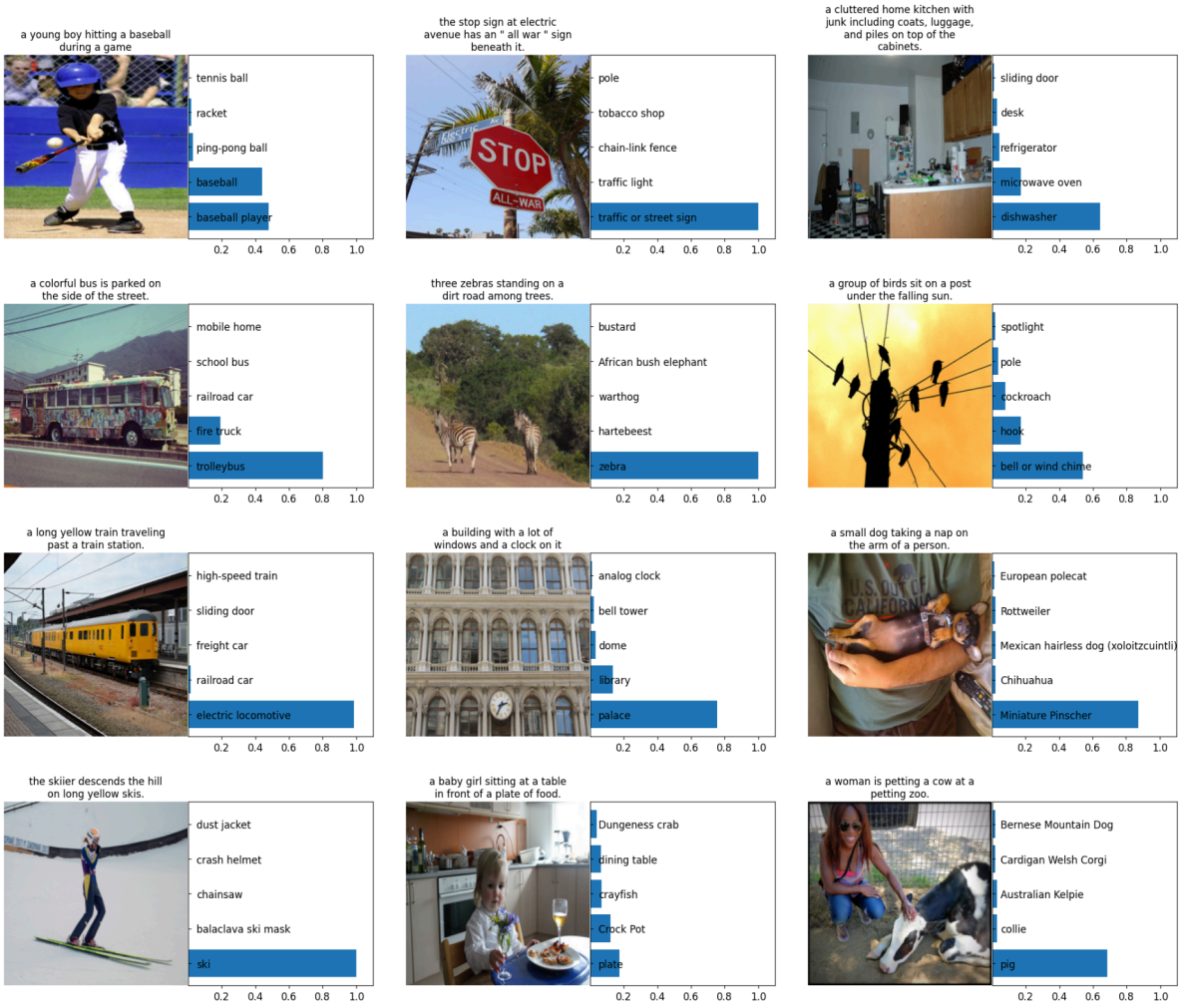


Figure 39: Visualization of the predicted probabilities for the top-5 ImageNet-1K [Rus+15] classes on image-text pairs from the COCO train set [Lin+14]. While the predicted classes are not always correct, e.g. bottom right, they are able to capture to some extend the semantic content of the image, and even the text. The latter is crucial for the approach of SHRe [AVT17]. Note: The figure does not stem from the SHRe paper [AVT17], but is a custom visualization of the concept. However, it is inspired by the CLIP paper [Rad+21]. The ResNet-50-A1 [WTJ21] model is used for the predictions.

| Approach | # Image-Text pairs |
|---|---|
| FLAVA [Sin+21] | 70M |
| CLIP [Rad+21] | 400M |
| VLMo [Bao+22] | 10M |
| CoCa [Yu+22] | >3B |
| BEiT-3 [Wan+23] | 21M |
| This work | 3.3M |

Table 33: A comparison of the number of image-text pairs used for pretraining in different approaches. We use significantly fewer pairs compared to other approaches.

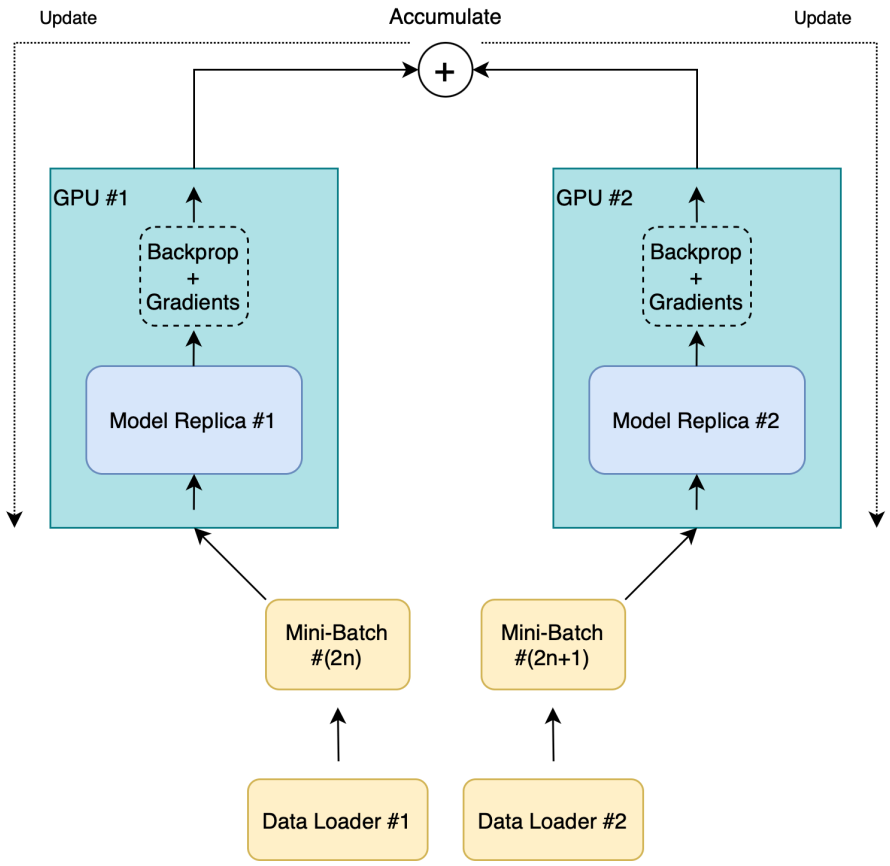


Figure 40: Distributed Data Parallel allows training a model with larger batch sizes than possible on a single GPU. The actual batch size per GPU does not change, but gradient updates are synchronized across GPUs, leading to weight updates that are equivalent to a larger batch size [Li+20].

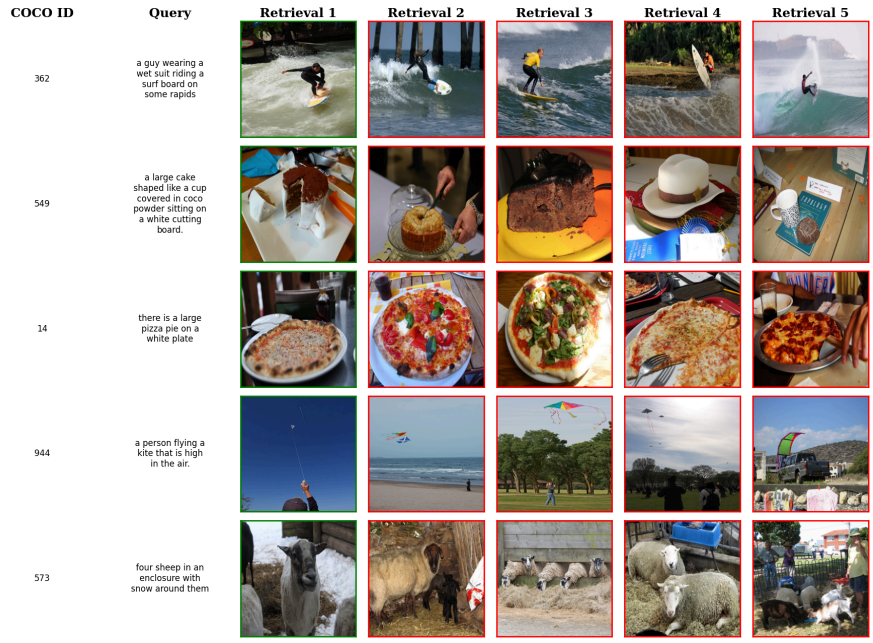


Figure 41: Results of image retrieval on a 1k subset of the COCO test set [Lin+14], as described in [AVT17].

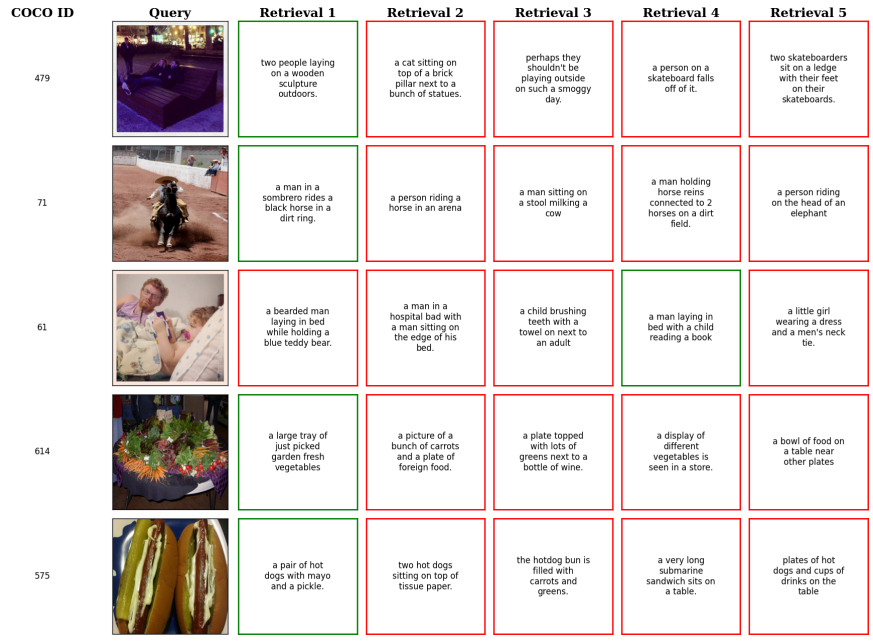


Figure 42: Results of text/caption retrieval on a 1k subset of the COCO test set [Lin+14], as described in [AVT17].

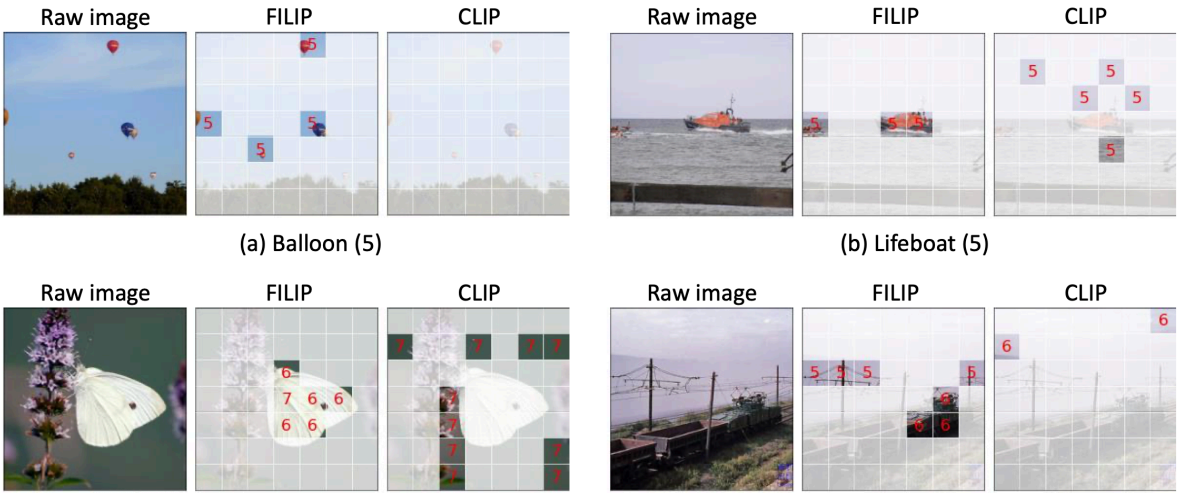


Figure 43: Cross-Model Late Interaction (CMLI) on FILIP [Yao+21] and CLIP [Rad+21]. Numbers in parentheses describe the index of the text token in the text sequence, and numbers in patches to which text token a patch is matched. While FILIP is able to achieve a localization of the correct image patches, CLIP fails to do so. The figure is directly taken from FILIP [Yao+21].

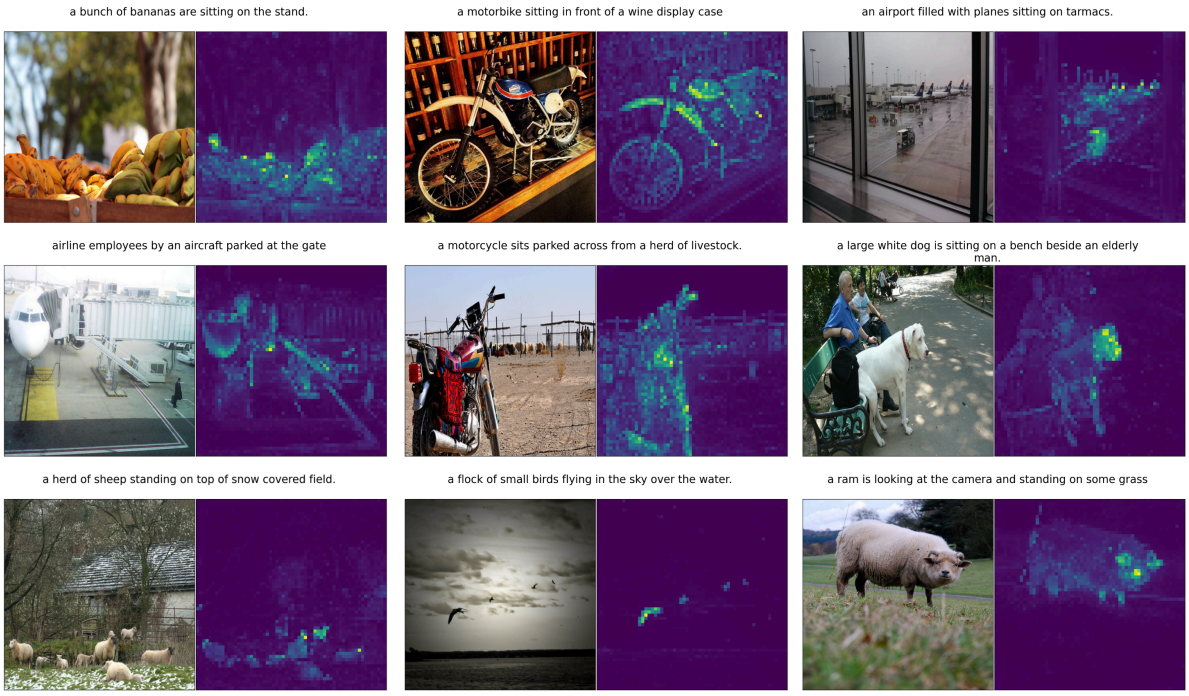


Figure 44: Fine-grained DINO [Car+21] self-attention map of the [I_CLS] token on image-text pairs of the COCO test set [Lin+14]. Heatmap is average over all attention heads.


Figure 45: Examples of random masking applied to images from the COCO test set [Lin+14]. The masking probability is set to $p = 0.9$.

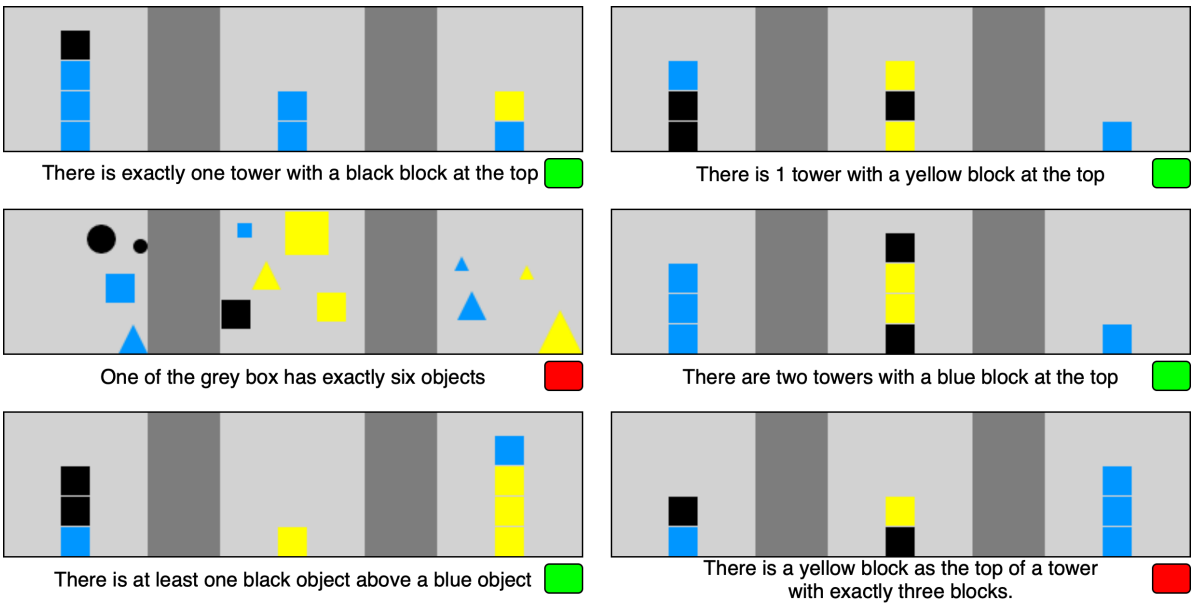


Figure 46: Example images with corresponding statements from the NLVR2 dataset [Suh+19]. Green and red boxes at the bottom right of each example indicate a correct and incorrect statement, respectively. The examples are taken from the official NLVR2 website[16].

[16] https://lil.nlp.cornell.edu/nlvr/nlvr.html

# AD Technical Details

## ADA Data

## ADB Hardware

To train the models and store the data, we used the GPU cloud platform runpod.io[17]. This platform provides access to a wide range of GPU types, including the NVIDIA RTX 4090 24GB, which we use for training almost all of our models. The reason for choosing this platform over traditional cloud providers like AWS or GCP is that the price per GPU hour is significantly lower, which allows us to train more models for the same budget. For example, as of September 2024, the price per GPU hour for an NVIDIA RTX 4090 24GB on runpod.io is $0.69. A comparable GPU on AWS, e.g. the NVIDIA V100 16GB, costs $3.06[18] per GPU hour. This price difference is despite the fact that the V100 was released in 2017, while the RTX 4090 was released in 2022. The RTX 4090 is also faster than the V100, with a higher memory bandwidth and more CUDA cores, so training on the RTX 4090 is more cost-effective by a margin.

To store all of our data, which is a total of >900 GB, we use a network volume provided by runpod.io. This volume is mounted on a virtual machine on start-up, allowing to access the data for training on the GPUs and provides high flexibility.

The RTX 4090 instances we used have 61 GB of memory and 8 virtual CPUs for one GPU, and around 132 GB of memory with 16 virtual CPUs if using two GPUs, which is similar to that of AWS.

runpod.io also provides on-demand vm instances, as well as spot instances. The latter can be automatically terminated if demand is high, but the price is significantly lower. On-demand instances are more expensive but are non-interruptible, which is, even though we create a model checkpoint after each training epoch, important for long-running experiments, which is the case for most training runs in this work. As of September 2024, the price for an on-demand instance with a single RTX 4090 is $0.69 per GPU hour, while the price for a spot instance is just $0.35 per GPU hour. The price for an instance increases proportionally with the number of GPUs used, so for us, a two-GPU instance costs $1.38 per GPU hour.

## ADC Costs Breakdown

[17] https://www.runpod.io

[18] Price is taken from the official AWS pricing page (https://aws.amazon.com/de/ec2/instance-types/p3/) and based on the on-demand price for the region us-east (North-Virginia).

# Bibliography


[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.

[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

[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

[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.

[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.

[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.

[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.

[Bao+22] H. Bao, L. Dong, S. Piao, and F. Wei, "BEiT: BERT Pre-Training of Image Transformers," in *International Conference on Learning Representations*, 2022. [Online]. Available: https://openreview.net/forum?id=p-BhZSz59o4

[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.

[Mik+13] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient Estimation of Word Representations in Vector Space." 2013.

[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.

[BKH16] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer Normalization." 2016.

[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.

[Rad+19] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language Models are Unsupervised Multitask Learners," 2019.

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

[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/

[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.

[Che+20] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A Simple Framework for Contrastive Learning of Visual Representations," *arXiv preprint arXiv:2002.05709*, 2020.

[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.

[CH21] X. Chen and K. He, "Exploring Simple Siamese Representation Learning," in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021, pp. 15745–15753. doi: 10.1109/CVPR46437.2021.01549.

[CH20] T. Chen and G. Hinton, "Advancing Self-Supervised and Semi-Supervised Learning with SimCLR." [Online]. Available: https://research.google/blog/advancing-self-supervised-and-semi-supervised-learning-with-simclr/

[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.

[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

[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

[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.

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

[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.

[Soc+13] R. Socher *et al.*, "Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank," in *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, Seattle, Washington, USA: Association for Computational Linguistics, Oct. 2013, pp. 1631–1642. [Online]. Available: https://www.aclweb.org/anthology/D13-1170

[WSB18] A. Warstadt, A. Singh, and S. R. Bowman, "Neural Network Acceptability Judgments," *arXiv preprint arXiv:1805.12471*, 2018.

[May21] P. May, "Machine translated multilingual STS benchmark dataset.," 2021. [Online]. Available: https://github.com/PhilipMay/stsb-multi-mt

[DB05] W. B. Dolan and C. Brockett, "Automatically constructing a corpus of sentential paraphrases," in *Proceedings of the International Workshop on Paraphrasing*, 2005.

[Raj+16] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, "SQuAD: 100,000+ Questions for Machine Comprehension of Text," in *Proceedings of EMNLP*, Austin, Texas: Association for Computational Linguistics, 2016, pp. 2383–2392.

[Ben+09] L. Bentivogli, I. Dagan, H. T. Dang, D. Giampiccolo, and B. Magnini, "The Fifth PASCAL Recognizing Textual Entailment Challenge," 2009.

[DGM06] I. Dagan, O. Glickman, and B. Magnini, "The PASCAL recognising textual entailment challenge," *Machine learning challenges. evaluating predictive uncertainty, visual object classification, and recognising tectual entailment*. Springer, pp. 177–190, 2006.

[Gia+07] D. Giampiccolo, B. Magnini, I. Dagan, and B. Dolan, "The third PASCAL recognizing textual entailment challenge," in *Proceedings of the ACL-PASCAL workshop on textual entailment and paraphrasing*, 2007, pp. 1–9.

[Bar+06] R. Bar Haim *et al.*, "The second PASCAL recognising textual entailment challenge," 2006.

[WNB18] A. Williams, N. Nangia, and S. R. Bowman, "A Broad-Coverage Challenge Corpus for Sentence Understanding through Inference," in *Proceedings of NAACL-HLT*, 2018.

[Kri+17] R. Krishna *et al.*, "Visual Genome: Connecting Language and Vision Using Crowdsourced Dense Image Annotations," *Int. J. Comput. Vision*, vol. 123, no. 1, pp. 32–73, May 2017.

[Sha+18] P. Sharma, N. Ding, S. Goodman, and R. Soricut, "Conceptual captions: A cleaned, hypernymed, image alt-text dataset for automatic image captioning," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers*, I. Gurevych and Y. Miyao, Eds., 2018, pp. 2556–2565.

[Cha+21] S. Changpinyo, P. Sharma, N. Ding, and R. Soricut, "Conceptual 12M: Pushing Web-Scale Image-Text Pre-Training To Recognize Long-Tail Visual Concepts," in *CVPR*, 2021.

[OKB11] V. Ordonez, G. Kulkarni, and T. L. Berg, "Im2text: Describing images using 1 million captioned photographs," in *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011. Proceedings of a meeting held 12-14 December 2011, Granada, Spain*, J. Shawe-

Taylor, R. S. Zemel, P. L. Bartlett, F. C. N. Pereira, and K. Q. Weinberger, Eds., 2011, pp. 1143–1151.

[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.

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

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

[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.

[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

[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.

[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.

[Pet+18] M. Peters *et al.*, "Deep Contextualized Word Representations," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, Association for Computational Linguistics, 2018. doi: 10.18653/v1/n18-1202.

[WTJ21] R. Wightman, H. Touvron, and H. Jégou, "ResNet strikes back: An improved training procedure in timm." 2021.

[Li+20] S. Li *et al.*, "PyTorch distributed: experiences on accelerating data parallel training," *Proc. VLDB Endow.*, vol. 13, no. 12, pp. 3005–3018, Aug. 2020, doi: 10.14778/3415478.3415530.

[Li+17] A. Li, A. Jabri, A. Joulin, and L. van der Maaten, "Learning visual n-grams from web data," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 4183–4192.

[Tou+21] H. Touvron, M. Cord, A. Sablayrolles, G. Synnaeve, and H. Jegou, "Going deeper with Image Transformers," in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, IEEE, Oct. 2021. doi: 10.1109/iccv48922.2021.00010.

[Dou+22] Z.-Y. Dou *et al.*, "An Empirical Study of Training End-to-End Vision-and-Language Transformers," in *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, Jun. 2022. doi: 10.1109/cvpr52688.2022.01763.

[Li+21] J. Li, R. R. Selvaraju, A. D. Gotmare, S. Joty, C. Xiong, and S. Hoi, "Align before Fuse: Vision and Language Representation Learning with Momentum Distillation," in *NeurIPS*, 2021.

[Wu+18] Z. Wu, Y. Xiong, S. X. Yu, and D. Lin, "Unsupervised Feature Learning via Non-parametric Instance Discrimination," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 3733–3742. doi: 10.1109/CVPR.2018.00393.

[Che+20] X. Chen, H. Fan, R. Girshick, and K. He, "Improved Baselines with Momentum Contrastive Learning," *arXiv preprint arXiv:2003.04297*, 2020.

[CXH21] X. Chen, S. Xie, and K. He, "An Empirical Study of Training Self-Supervised Vision Transformers," *arXiv preprint arXiv:2104.02057*, 2021.

[Car+21] M. Caron *et al.*, "Emerging Properties in Self-Supervised Vision Transformers," in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, IEEE, Oct. 2021. doi: 10.1109/iccv48922.2021.00951.

[OVK17] A. van den Oord, O. Vinyals, and K. Kavukcuoglu, "Neural Discrete Representation Learning." 2017.

[Agr+16] A. Agrawal *et al.*, "VQA: Visual Question Answering: www.visualqa.org," *International Journal of Computer Vision*, vol. 123, no. 1, pp. 4–31, Nov. 2016, doi: 10.1007/s11263-016-0966-6.

[Suh+19] A. Suhr, S. Zhou, A. Zhang, I. Zhang, H. Bai, and Y. Artzi, "A Corpus for Reasoning about Natural Language Grounded in Photographs," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, Association for Computational Linguistics, 2019. doi: 10.18653/v1/p19-1644.

[KSK21] W. Kim, B. Son, and I. Kim, "ViLT: Vision-and-Language Transformer Without Convolution or Region Supervision." 2021.

[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.