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

Titel der Arbeit

Tim Cares

Master's thesis in Applied Computer Science

25. September 2024

Autor Tim Cares
Matrikelnummer
Email Adresse

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

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

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

Hannover, 25. September 2024 Tim Cares

Acknowledgements

Abstract

Contents

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.

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:

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

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

A comparison between the target distribution $\boldsymbol{p}$ 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

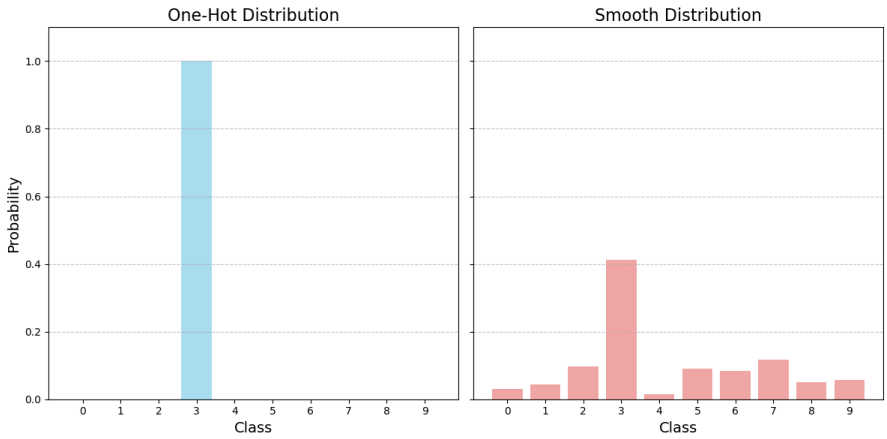

Figure 1: Comparison between different distributions over $C = 10$ classes. The one-hot distribution (left) is used for classification tasks with the cross-entropy loss. The KL-Divergence is used when predicting a smooth distribution (right). A smooth distribution usually results from a model prediction,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

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

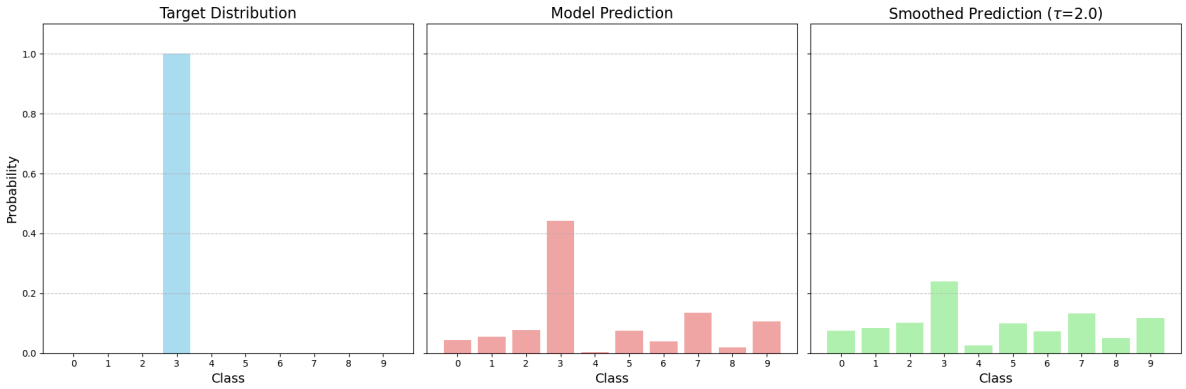

Figure 2: We present a similar figure as Figure 1. A temperature parameter τ further smoothens an already soft distribution. In response-based KD, this soft distribution is the prediction of a teacher model for a given samle (middle). Further smoothing (right) increases the relative importance of classes 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 τ, before applying the softmax function [Gou+21, HVD15], which smooothes the probability distribution further, as illustrated in Figure 2.

The temperature τ is usually a tuneable hyperparameter, but research has shown that it can also be a learned parameter, especially in other settings such as contrastive learning [Bao+22], introduced later.

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

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

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

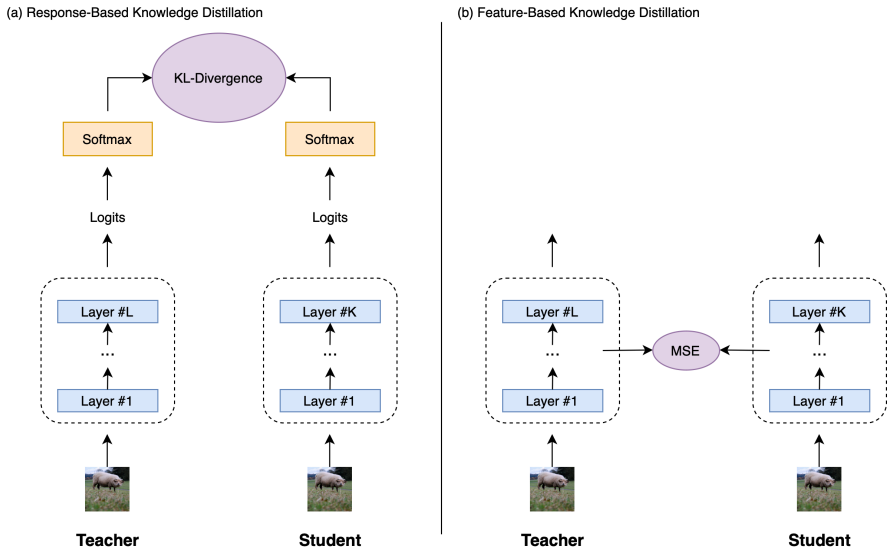

Figure 3: Response-based knowledge distillation (a) requires a teacher to provide logits to generate a probability distribution, which is predicted by the student. Feature-based knowledge distillation (b) is used for predicting the actual activations of the teacher's layer(s). In both cases the weights 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

$\boldsymbol{T}^{\text{pos}}$ 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.

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

The final text representation is defined as:

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

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:

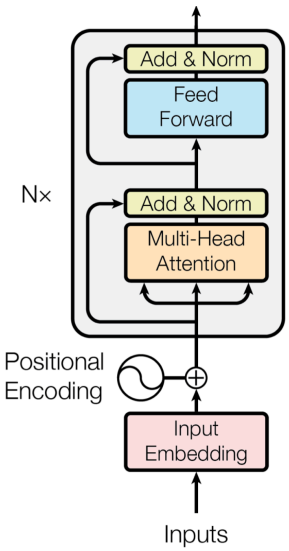

Figure 4: The architecture of the Transformer encoder. The original architecture also includes a Transformer decoder, which is not relevant for our work and therefore omitted. The encoder consists of N (or L) Transformer layers. The output of a layer is the input to the subsequent layer. Figure is adjusted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

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

$\mathrm{LN}(\cdot)$ 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.

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

ε 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 18.

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_{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_{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 224×224 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 14×14 patches, each being a square of size 16×16 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:

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

For an image size of 224×224 pixels, and a patch size of 16×16 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

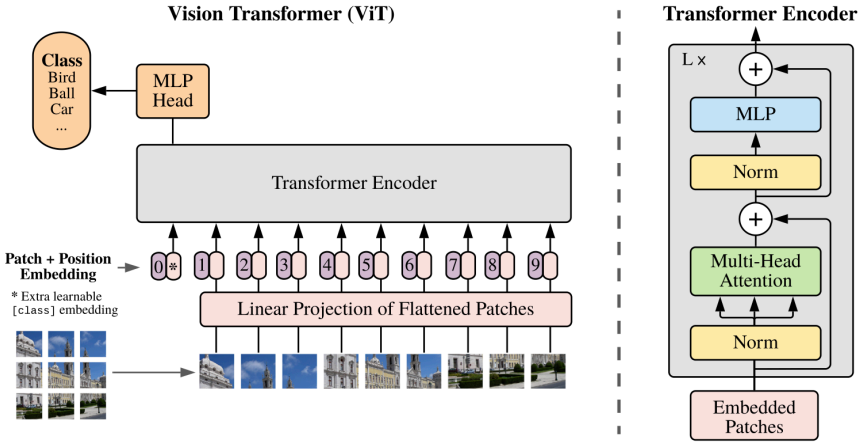

Figure 5: The architecture of the vision Transformer. Its key difference to the language Transformer is the patch embedding and the position of LayerNorm in a layer [Dos+21].

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 16×16 pixels, while the ViT-H/14 has a patch size of 14×14 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.

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

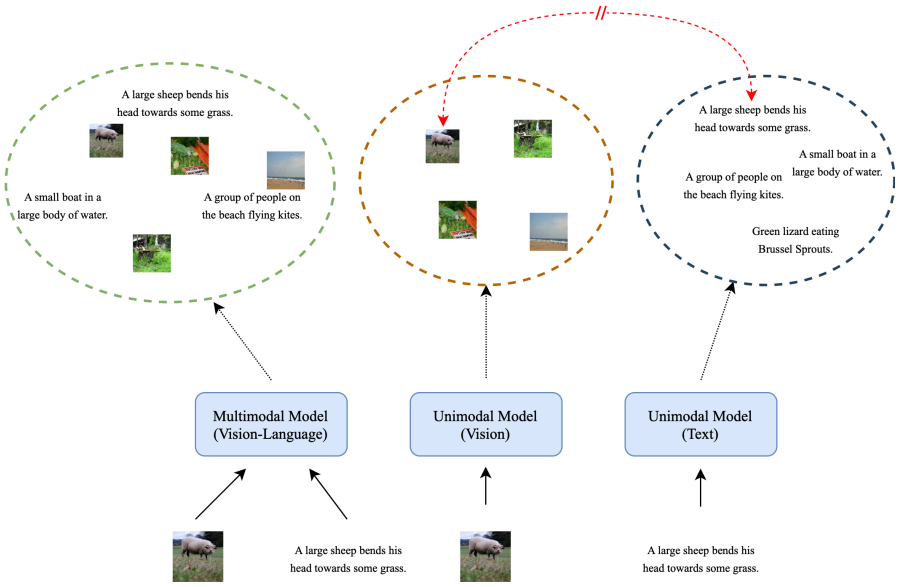

Figure 6: A multimodal model maps multiple modalities into a common representation space, where the representations of the same concept are aligned. In contrast, unimodal models map the input of a single modality into a modality-specific representation space. There is no alignment between the representations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

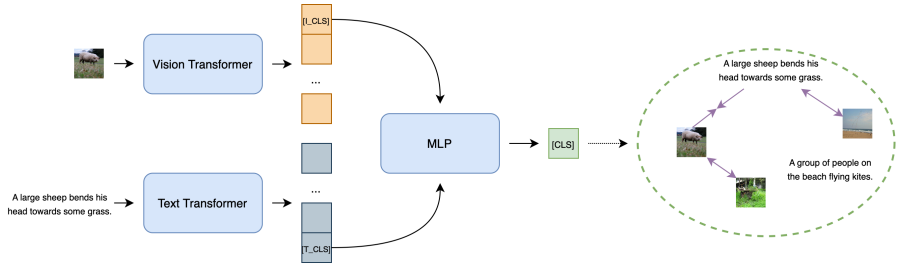

Figure 7: An abstract illustration of a vision-language model. Image and text are first passed through unimodal Transformer encoders, the cls tokens are extracted, and passed seperately into the MLP that maps the modality-specific representations into a common representation space. A contrastive loss 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-

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

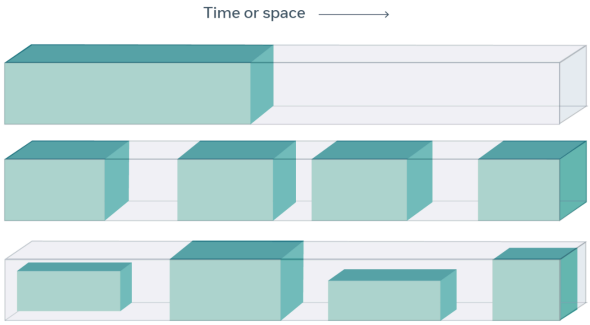

Figure 8: In self-supervised learning parts of the data are masked (grey), and the task of a model is to predict the masked parts using the visible data (green) [LM21].

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

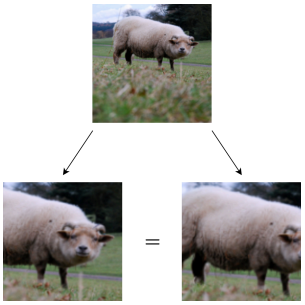

Figure 9: Adding small translations to an image, e.g. a random crop, as illustrated in the figure, will retrain high-level semantic features while changing pixel-level information. The content of the image stays the same, and the same should therefore hold for the representations produced by the model.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

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.

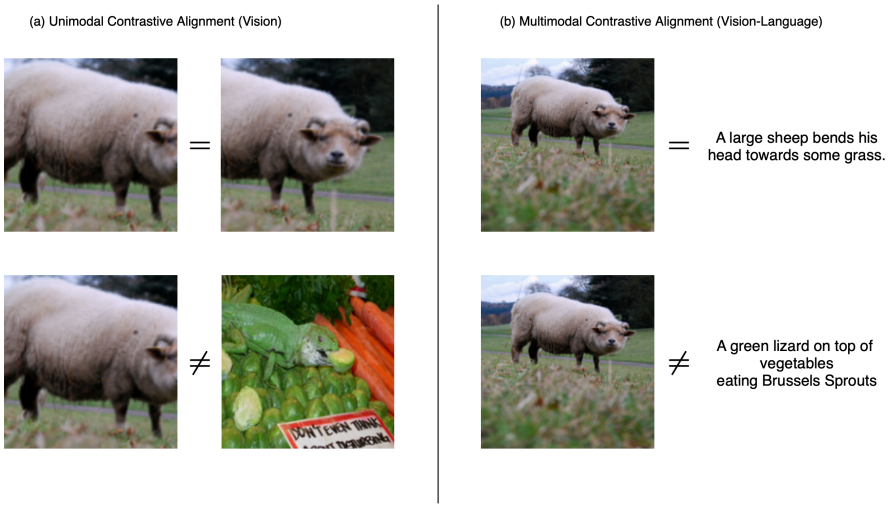

Figure 10: Contrastive learning aims to align the same (or similar) real-world concepts in representation space, while pushing different concepts apart. Multimodal contrastive learning (b) requires existing pairs, e.g. image-text, while for the unimodal case (a) pairs are synthetically created by augmenting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

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.

$$\boldsymbol{h}' = \frac{\boldsymbol{h}}{\|\boldsymbol{h}\|_2} \tag{23}$$

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

$$\boldsymbol{T} = \left[\boldsymbol{h}'_{(w,L,[\texttt{T\_CLS}]),1}, \boldsymbol{h}'_{(w,L,[\texttt{T\_CLS}]),2}, ..., \boldsymbol{h}'_{(w,L,[\texttt{T\_CLS}]),B}\right] \in \mathbb{R}^{B \times D} \tag{25}$$

$\boldsymbol{I}$ 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.

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

$\boldsymbol{L}_{i,j}$ then denotes the similarity between image i and text j in the batch. The diagonal of the matrix contains the similarity between positive pairs, i.e., the correct image-text pairs (i, i),

with $\boldsymbol{L}_{i,i}$ describing their similarity. For an image, all other texts in the batch are considered as negative samples, and vice versa for text. The superscript T denotes the transpose of a matrix, and is not to be confused with the batch of text representations $\boldsymbol{T}$.

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

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

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

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

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

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

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

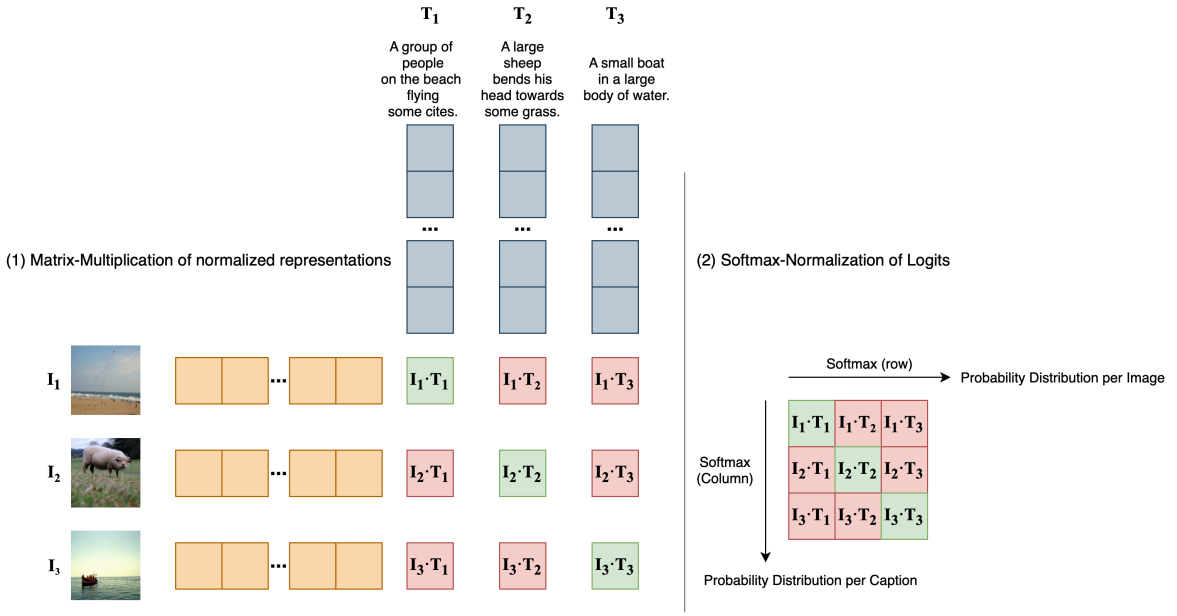

Figure 11: Contrastive Learning is performed using matrix multiplication of normalized representations (1), and the result is matrix $\boldsymbol{L}$ described in Equation 26. The representations are given by the [CLS] token of the respective modality, but are represented as I and T in the figure for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

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

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

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

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

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

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

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

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 will be shown when we apply this approach in (TODO: Transformer SHRe section).

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

However, the shared encoder is meant to have the same internal representation for e.g. an image and its caption/text: Since they describe the same concept, the activations in the shared encoder should be similar, which is, as described in (TODO: cite image-text contrast), crucial for tasks such as retrieval. To achieve this, the authors add a ranking loss to the

training, which functions similarly to a contrastive loss. This ranking loss drives the representations of inputs from the same pair closer together, while pushing the representations of inputs from different pairs further apart. It is defined as:

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

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:

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

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.

	MSCOCO		Flickr (Custom)[2]		Unspecified[3]	
Model	Image ↓	Text ↓	Image ↓	Sound ↓	Text ↓	Sound ↓
	Text	Image	Sound	Image	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].

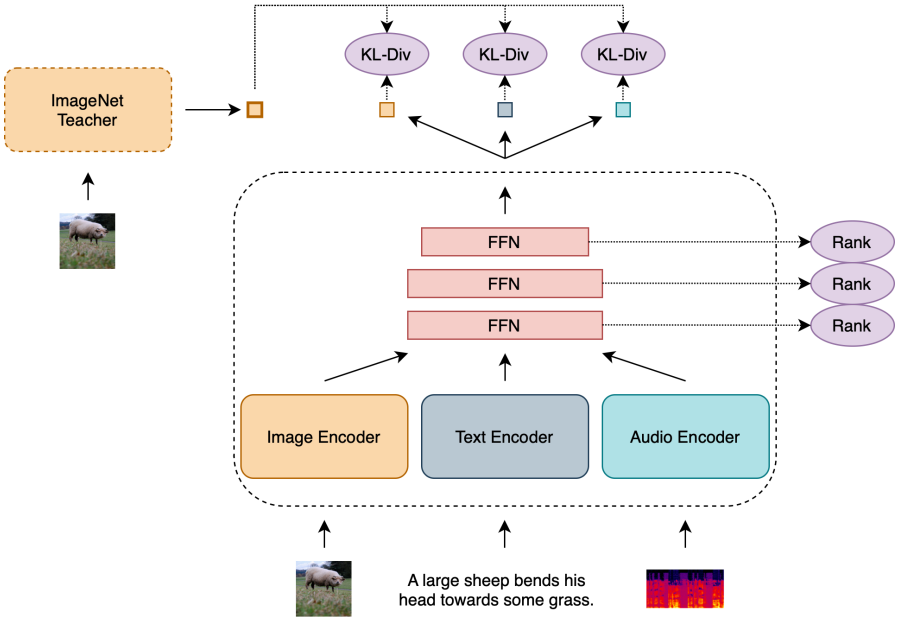

Figure 12: Illustration of the SHRe approach. The model is trained to output the same probability distribution over ImageNet-1K classes between images, image-text pairs, and image-audio pairs. Internal representations are aligned using a ranking loss [AVT17]. Image, text, and audio are always passed individually through the model. The output of the model, shown as colored squares, represent 1000-dimensional vectors, with each element representing the probability of the input belonging to a specific ImageNet-1K class. The figure does not originate from the original paper, but is a custom visualization of the concept. Image and text example is taken from the MSCOCO train set [Lin+14], the spectogram originates from the SHRe paper [AVT17].

[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.

1.7.2 CLIP

1.7.2.1 Method

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

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

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

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

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

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

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

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}.”.

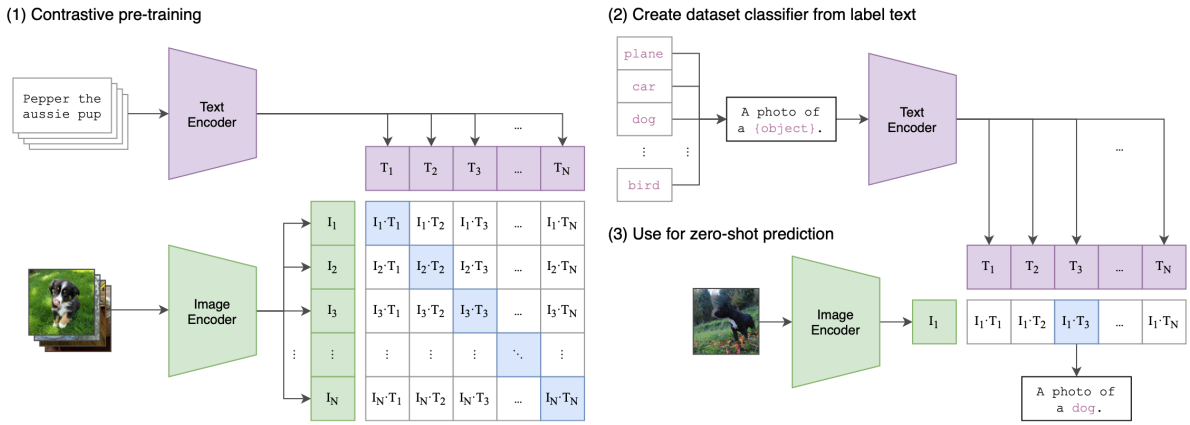

Figure 13: For zero-shot image classification, CLIP uses prompt engineering to create one classifier per image class to predict (2). The class whose classifier has the highest similarity (cosine) with the image representation is the predicted class (3) for the image [Rad+21].

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

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

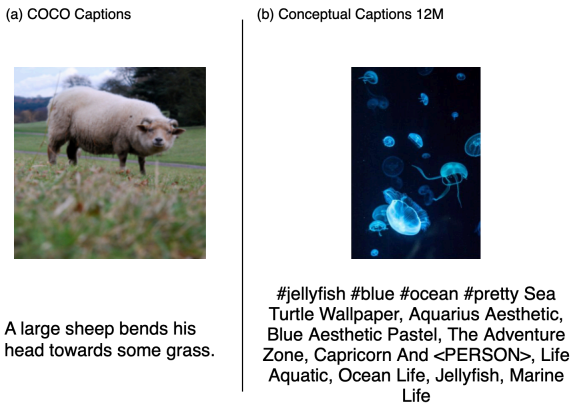

Figure 14: Side-by-side comparison of examples seen in COCO (a) and CC12M (b). While COCO features high quality images and detailed annotations, CC12M consists of in-the-wild image-text pairs from the internet. The latter enables scalability, as more data can be collected without the need for human annotation. 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.

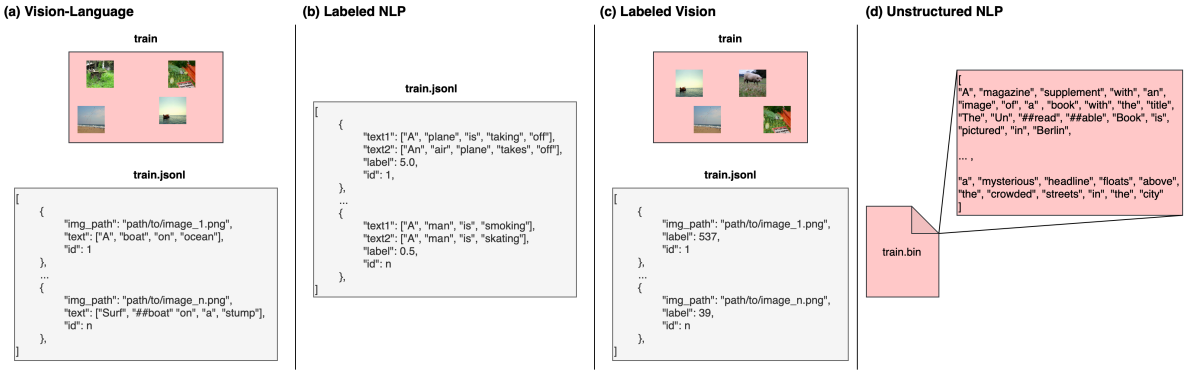

Figure 15: Comparison of different dataset organization and storage formats used in this work. Labeled NLP (d) datasets are not binarized, as they have a structure, i.e. there are fixed examples with labels.

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 μ_d and standard deviation σ_d are calculated over all time steps T. In the case of an embedding dimension of $D = 768$, this means for one sample (e.g. a sequence representing an image) 768 means and standard deviations are calculated, one for each embedding dimension. Then, for each time step j, the embedding at time step j is normalized by normalizing each dimension of the embedding independently, using the

corresponding mean and standard deviation computed for that dimension [UVL17]. During the normalization, a small epsilon, e.g. $1e^{-8} = 10^{-8}$, is added to the standard deviation to prevent division by zero. For an illustrative comparison between instance normalization, batch normalization and layer normalization, see (TODO: cite normalization) in the appendix. We define the operation $\mathrm{InstanceNorm}(\cdot)$ as instance normalization on a sequence of embeddings $\boldsymbol{H}$.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

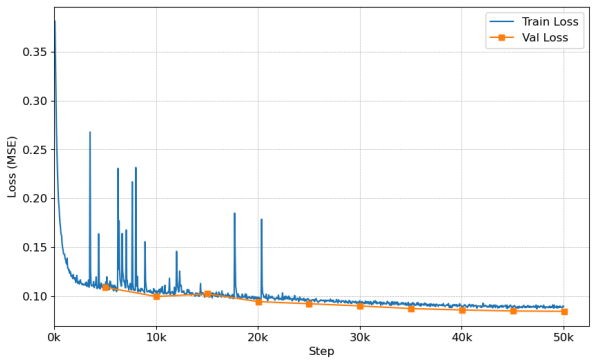

Figure 16: Training loss vs. validation loss during distillation of the Data2Vec2 image model.

[Kri09], CIFAR-100 [Kri09] and ImageNet-1K [Rus+15]. For that, we load the trained student model and add Layer Normalization and a linear classifier on top of it. The output of the student model is a sequence of embeddings, one embedding for each image patch, and one cls token embedding. We follow the approach of Data2Vec [Bae+22, Bae+22] and BEiTv2 [Pen+22], and take the mean over all patch embeddings as the output of the student model, which is then passed to the layer normalization and linear classifier (cls token embedding is ingnored). For all three tasks we perform full finetuning, i.e. we finetune all layers of the student model on the downstream task, and linear probing, we only train the added layer norm and linear classifier on top of the student model. For pytorch pseudocode of linear probing and full finetuning see (TODO: cite pseudocode).

For data augmentation during finetuning we use RandAugment [Cub+20], mixup [Zha+18] and cutmix [Yun+19] augmentation, and random erasing [Zho+20]. The hyperparameters for these augmentations are provided in (TODO: cite hyperparameters), and have been selected based on the values used in BEiTv2 [Pen+22], Data2Vec [Bae+22], and Data2Vec2 [Bae+22]. We refrain from explaining the augmentation techniques in detail here, as they are well documented in the respective papers.

For finetuning on ImageNet-1K we use a base learning rate of 1-e3 in combination with layer decay. Layer decay is a technique to reduce the base learning rate for each layer of the model by a certain factor. The goal is to have lower learning rates for layers closer to the input, and higher learning rates for layers closer to the output. This ensures that low-level features learned during pretraining or distillation are not destroyed during finetuning. We use a decay factor of 0.81, which is derived by scaling the layer decay used in Data2Vec2 [Bae+22], from which we extract layers for the student model, by the square root. We use scaling instead of the value used in Data2Vec2, which is 0.65 ($\sqrt{0.65} \approx 0.81$), as we only have half the number of layers in the student model, and can therefore afford larger learning rates for the lower layers. The actual learning rate for a layer l is then calculated by:

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

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

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.

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.

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

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

The results, displayed in Table 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, for reasons later explained later, we will also use BERT as the text encoder in our multimodal model, so for consistency we will use BERT for the unimodal distillation as well. Second, as mentioned before, there already exists a distilled version of BERT, DistilBERT [San+19], to which we can directly compare our results.

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

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.

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 Data2Vec2, which uses the same loss [Bae+22].

3.1.2.2 Distillation

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

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

3.1.2.3 Finetuning

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

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

Table 8: Results for BERT and DistilBERT are taken from the DistilBERT paper [San+19].

3.2.1.1.1 Architecture

What makes our approach different from SHRe is that we use a langauge 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 $\boldsymbol{h}_{v,K}$ and $\boldsymbol{h}_{w,K}$. Here, L_s denotes the number of Transformer layers in the image and text encoder, respectively, which is defined as $L_s = 6$. K denotes the number of the output layer, which is defined as $K = 9$. We set $K = 9$ because we have a 3-layer MLP and we count each MLP layer as a distinct layer. Since the MLP is stacked on top of the image and text encoder the last MLP layer, being the output layer, is the 9th layer. Further, we remove the token indicator [I_CLS] and [T_CLS] from the notation, as the MLP only receives those representations as input. There are no more time steps to distinguish between.

The 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.

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.

The shared encoder, which is a 3-layer MLP, is implemented by using the 2-layer MLP module as implemented in a Transformer layer, and adding an additional LayerNorm and MLP 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:

$$\begin{aligned} \boldsymbol{h}_{w,K-2} &= \boldsymbol{h}_{w,L_s,[\texttt{T\_CLS}]}\boldsymbol{W}_1 + \boldsymbol{b}_1 \\ \boldsymbol{h}_{w,K-1} &= \mathrm{LN}\big(\mathrm{GELU}\big(\boldsymbol{h}_{w,K-2}\big)\big)\boldsymbol{W}_2 + \boldsymbol{b}_2 \\ \boldsymbol{h}_{w,K} &= \mathrm{LN}\big(\boldsymbol{h}_{w,K-1}\big)\boldsymbol{W}_3 + \boldsymbol{b}_3 \end{aligned} \tag{43}$$

The computation of $\boldsymbol{h}_{v,K-2}$ and $\boldsymbol{h}_{v,K-1}$ is analogous to the definition of the MLP layers in a Transformer layer, defined in Equation 15. We choose a different notation here to be more precise 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

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

version of the loss function. As a preliminary step, we define the softmax normalization of a vector $\boldsymbol{u}$ as follows:

$$\begin{aligned} \pi(\boldsymbol{u}) = \boldsymbol{z} &= [z_1, z_2, ..., z_n] \in \mathbb{R}^n \\ z_i &= \frac{\exp(u_i)}{\sum_{j=1}^{n} \exp(u_j)} \end{aligned} \tag{44}$$

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:

$$\begin{aligned} \mathcal{L}_{\text{KD}} &= \\ \frac{1}{2} * \mathcal{L}_{\text{KD}}^{v} + \frac{1}{2} * \mathcal{L}_{\text{KD}}^{w} &= \\ \frac{1}{2} * D_{\text{KL}}\big(\pi(\boldsymbol{p}), \pi\big(\boldsymbol{h}_{v,K}\big)\big) + \frac{1}{2} * D_{\text{KL}}\big(\pi(\boldsymbol{p}), \pi\big(\boldsymbol{h}_{w,K}\big)\big) & \end{aligned} \tag{45}$$

$\boldsymbol{p}$ 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:

$$\begin{aligned} \mathcal{L}_{\text{CL}} &= \\ \frac{1}{3} * \big(\mathcal{L}_{\text{CL}}^{K-2} + \mathcal{L}_{\text{CL}}^{K-1} + \mathcal{L}_{\text{CL}}^{K}\big) &= \\ \frac{1}{6}\mathcal{L}_{\text{CL}}^{K-2,\ \text{i2t}} + \frac{1}{6}\mathcal{L}_{\text{CL}}^{K-2,\ \text{t2i}} &+ \\ \frac{1}{6}\mathcal{L}_{\text{CL}}^{K-1,\ \text{i2t}} + \frac{1}{6}\mathcal{L}_{\text{CL}}^{K-1,\ \text{t2i}} &+ \\ \frac{1}{6}\mathcal{L}_{\text{CL}}^{K,\ \text{i2t}} + \frac{1}{6}\mathcal{L}_{\text{CL}}^{K,\ \text{t2i}} & \end{aligned} \tag{46}$$

Let's break this down: The superscript $K-2$, $K-1$, and K denote on which layer the contrastive loss is applied. Since we have three layers MLP layers in our shared encoder, and

we want to enforce alignment in all layers, we apply the contrastive loss on all three layers. As introduced in the previous section (Section 3.2.1.1.1), K denotes the number of layers that are stacked on top of each other. Since we have 6 layers for each modality-specific encoder, and 3 layers for the shared encoder, $K = 9$. Therefore, we apply the contrastive loss on layers 7-9, which conviniently are the the three MLP layers forming the shared encoder. The second 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 MLP layers of the shared encoder, and we weight the loss equally for each layer. The contrastive loss also 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 $\mathcal{L}_{\mathrm{CL}}^{Q}$, with $Q \in [K-2, K-1, K]$, we generate the matrix $\boldsymbol{L}$ from Section 1.5.3.2 once, but since we to not directly make use of the [I_CLS] and [T_CLS] token, we have to redefine the batched representations $\boldsymbol{I}$ and $\boldsymbol{T}$ as follows:

$$\boldsymbol{I}_Q = \left[\boldsymbol{h}'_{(v,Q),1}, \boldsymbol{h}'_{(v,Q),2}, ..., \boldsymbol{h}'_{(v,Q),B}\right] \in \mathbb{R}^{B \times D_Q} \tag{47}$$

$$\boldsymbol{T}_Q = \left[\boldsymbol{h}'_{(w,Q),1}, \boldsymbol{h}'_{(w,Q),2}, ..., \boldsymbol{h}'_{(w,Q),B}\right] \in \mathbb{R}^{B \times D_Q} \tag{48}$$

Here, D_Q denotes the dimensionality of the representations $\boldsymbol{h}_{v,Q}$ and $\boldsymbol{h}_{w,Q}$ from MLP layer Q, which is essentially the number of neuros in the MLP layer. B denotes the batch size, and we refer to Section 1.5.3.2 for the definition of $\boldsymbol{h}'$. It follows for the matrix $\boldsymbol{L}$ of image-to-text and text-to-image similarities:

$$\boldsymbol{L}_Q = \boldsymbol{I}_Q \boldsymbol{T}_Q^T, \boldsymbol{L}_Q \in \mathbb{R}^{B \times B} \tag{49}$$

The training objective is then given by:

$$\min \mathcal{L}_{\mathrm{KD}} + \mathcal{L}_{\mathrm{CL}} \tag{50}$$

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. 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

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

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. 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 10, 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 20, and a visualization of a minimum crop size of 0.9 is shown in Figure 21.

Pseudocode

3.2.2 Challenges of Self-Supervision

Our approach differs to that of SHRe [AVT17] in that we will make use of a self-supervised teacher model. The consequence is that the teacher's prediction for a given sample is not a probability distribution over a set of classes, but merely a representation of the input sample. As mentioned in Section 1.7.1, a probabilty distribution over a set of classes is to some extent independent of the input modality, as each class can be seen as a semantic concept that can be present in both images and text. SHRe works well with a supervised image teacher model, as the probability distribution over the classes of an image can somewhat describe the content of the image's caption. Examples are shown in TODO.

For a unimodal self-supervised model however, this probability distribution does not exists, raising the question which training objective to use when using a self-supervised teacher. 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 as the input, the teacher's prediction is still a representation of the image, and not the text (the image's caption). 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:

$$\min_{\boldsymbol{h}^s_{v,L_s,[\texttt{I\_CLS}]}} \left\| \boldsymbol{h}^s_{v,L_s,[\texttt{I\_CLS}]} - \boldsymbol{h}^t_{v,L_t,[\texttt{I\_CLS}]} \right\| \tag{51}$$

This 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 [T_CLS] as close as possible to the representation of the image [I_CLS] produced by the teacher:

$$\min_{\boldsymbol{h}^s_{w,L_s,[\texttt{T\_CLS}]}} \left\| \boldsymbol{h}^s_{w,L_s,[\texttt{T\_CLS}]} - \boldsymbol{h}^t_{v,L_t,[\texttt{I\_CLS}]} \right\| \tag{52}$$

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 17.

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

$$\min_{\boldsymbol{h}^s_{v,L_s,[\texttt{I\_CLS}]}, \boldsymbol{h}^s_{w,L_s,[\texttt{T\_CLS}]}} \left\| \boldsymbol{h}^s_{v,L_s,[\texttt{I\_CLS}]} - \boldsymbol{h}^t_{v,L_t,[\texttt{I\_CLS}]} \right\| + \left\| \boldsymbol{h}^s_{w,L_s,[\texttt{T\_CLS}]} - \boldsymbol{h}^t_{v,L_t,[\texttt{I\_CLS}]} \right\| \tag{53}$$

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 the extract any image-specific information like the pixel values of the exact position of an object in the image from the caption of the image. 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

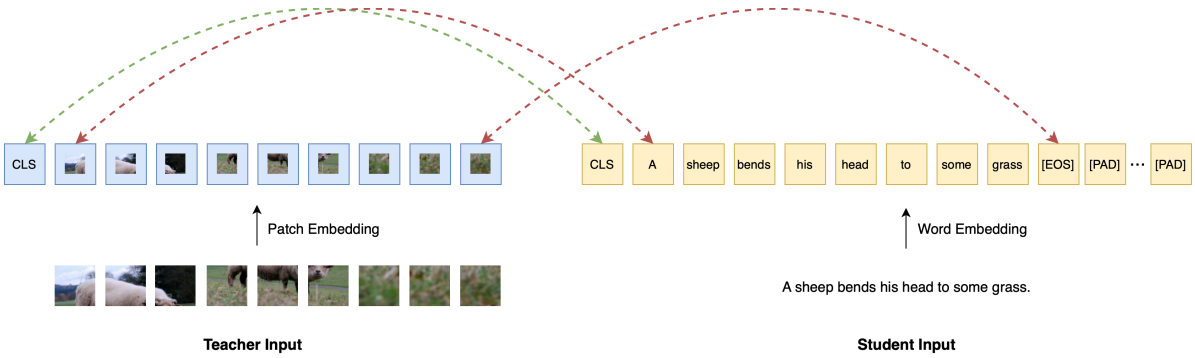

Figure 17: The meaning/content of time steps across modalities is not aligned, and the number of time steps will differ between modalities. This makes alignment on the level of individual time steps impossible. The cls token aggregates global information independent of time steps, and captures the meaning 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].

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.

A Appendix

AA Hyperparameters

AB Pseudocode

AC Figures and Visualizations

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

Table 10: Hyperparameters used for distilling a Data2Vec2 image model.

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

Table 11: 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].

AD Technical Details

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

Table 12: Hyperparameters for the GLUE [Wan+19] benchmark tasks of the distilled Data2Vec2 image model.

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

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

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

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

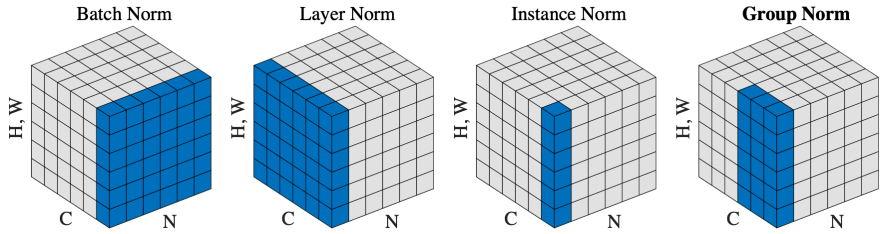

Figure 18: Comparison of different normalization operations on the example of images. Dimension “H, W” refers to the height and width of the input, “C” to the number of channels or embedding dimensions, and “N” to the number of samples, i.e. the batch dimension. Since we are working with sequences of 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].

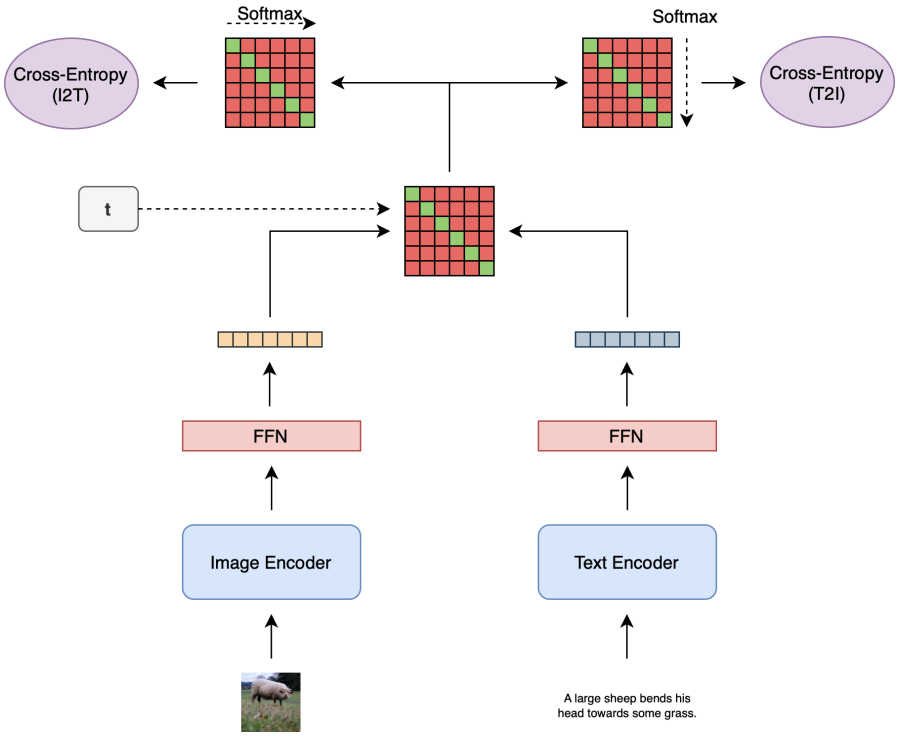

Figure 19: Illustration of CLIP training. A batch of image-text pairs is passed through the model and embedded into a shared latent space. The cosine similarity between all pairs is computed and softmax-normalized to calculate the image-to-text and text-to-image loss. The final loss is the mean of both 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	Our friends won't buy this analysis, let alone the next one we propose.	1
SST-2	hide new secretions from the parental units	0
MRPC	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.	1
STS-B	A plane is taking off. [SEP] An air plane is taking off.	5.0
QQP	How is the life of a math student? Could you describe your own experiences? [SEP] Which level of prepration is enough for the exam "jlpt5"?	0
MNLI	Conceptually cream skimming has two basic dimensions - product and geography. [SEP] Product and geography are what make cream skimming work.	1
QNLI	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.	1
RTE	No Weapons of Mass Destruction Found in Iraq Yet. [SEP] Weapons of Mass Destruction Found in Iraq.	1

Table 13: Training examples of the GLUE benchmark tasks (one example per task). Examples are taking from the GLUE dataset card on Hugging Face[15].

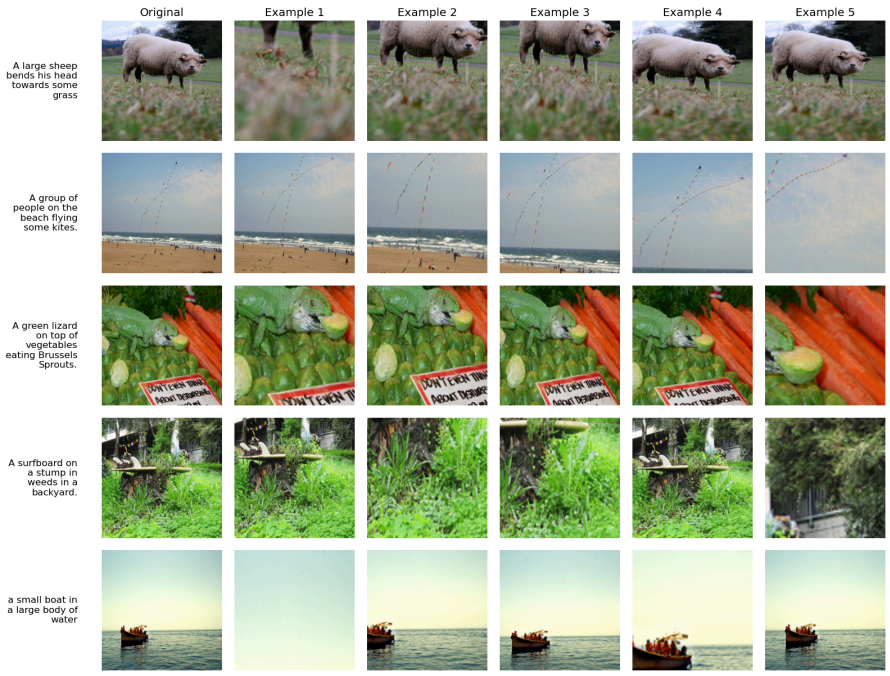

Figure 20: A small lower bound (8%) for a random crop erases high-level semantic features which are important for aligning text and image. Image-text pairs have been taken from the COCO train set [Lin+14].

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

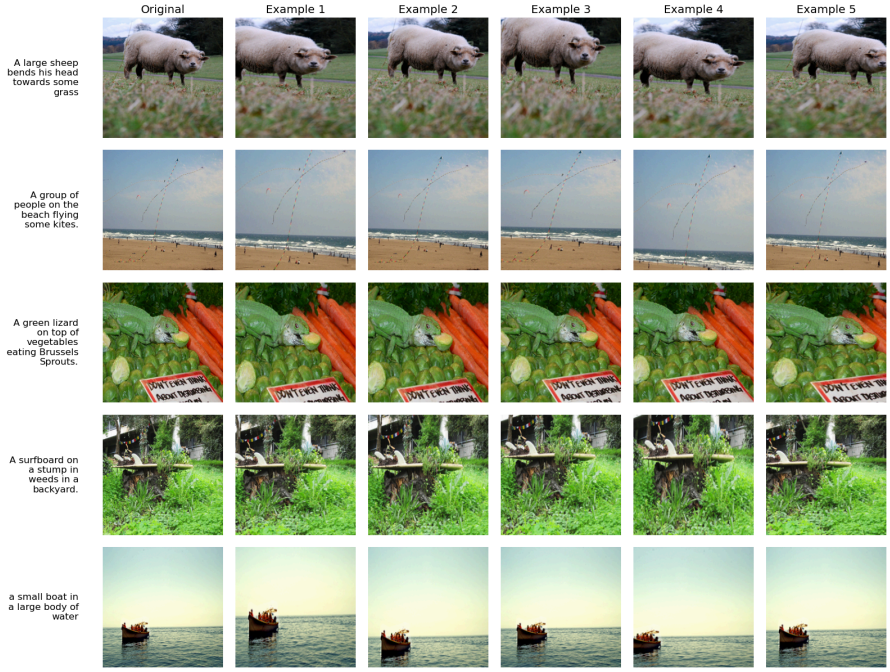

Figure 21: A larger lower bound (90%) for a random crop retains high-level semantic features. Notice how this is not the case for very low values, as shown in Figure 20. Image-text pairs have been taken from the COCO train set [Lin+14].

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.

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

[WH18] Y. Wu and K. He, "Group Normalization," in *Computer Vision -- ECCV 2018: 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part XIII*, Munich, Germany: Springer-Verlag, 2018, pp. 3–19. doi: 10.1007/978-3-030-01261-8_1.