Method

Model Architecture

- We start with an image only (unimodal) model, based on a ViT architecture
- We orient the architecture on the Data2Vec architecture
  - We want to create a multimodal Data2Vec
- Therefore, model features a seperate modality (specific) encoder,

and then the Transformer blocks

- those are later used to process multiple modalities and align them

- As with the target creation, we want to reuse as much as possible from d2v directly
  - We are constrained by GPUs+memory, and goal is generally to take a step back from huge models (500+ million)
    - (we are only talking about training smaller models, we are not saying that we do not use those bigger models, or parts of them, for KD in general, so taking those, later three, large teacher is allowed base on our philosophy, especially because we only run them in inference)
  - as will come apparent later, it is not feasible to create very small multimodal models, due to the difficulty of the task, and because each multimodal model needs one modality encoder per modality
    - it doesn't matter whether the model uses a multi-encoder approach (like data2vec) or a fuzed encoder approach, build using Mixture of Modality Experts (MoME)
    - in either case we need seperate layers for each modality in the lower layers
    - will make the model very large, and therefore expensive to train
  - important is that training is feasible, i.e. training does not take too long with moderate resources, so GPUs

- we have at our disposal the already pretrained D2V models for

image, text and audio
- for this first step, we are only using the image model
- image model is based on ViT-B/16 model
- has around 86 million params, embed dim of 768
  - other d2v models have also based on this, but are larger due to

  their modality encoders (mention more on that in earlier sections)
- also finetuned versions of d2v models avaialble, trained on labeled data, but we won't use them
  - We want to build self-supervised models, i.e. without labeled data
    - using targets to regress that where created by models trained on labeled data would be "cheating" and not self-supervised anymore
- because, again, we want to reuse as much as possible from D2V

as possible, we are going the reuse the image modality encoder of D2V image

- That way we can profit immensely from the features produced by them

- We freeze the modality specific encoder -> We do not want them to forget what they have learned + plus they are already really good at their task
  - modality encoder runs in inference

- we only train the transformer layers/blocks
  - so the part that we train does not receive raw images, but already patch embeddings not containing pixel level information anymore (rather a little higher level information)
  - this is high quality information, and a big advantage for us

  - also, we save memory for the gradients normally needed to train a modality encoder, and training will be faster
- disadvantage: we are now bound to the dimension/size of the encoders (768)
  - embed dim set be PatchEmbed layer in image modality encoder
- means linear layers of Multi-Head Attention and MLP layers have to be adjusted to this size, can't be smaller
- therefore, our embed dim will always be 768
- more params, longer forward pass, more memory, longer training time
- in theory we could reduce this dimension by adding a linear projection between the image modality encoder and our trainable Transformer Blocks, so just one MLP layer, with number of neurons <768
- problems is that we would compress a lot of information, so we are potentially losing the advantage of the high quality features that the pretraining modality encoder produces
- also, because this projection is has to be training itself, training time would be longer, as the layer first has to learn to compress the information while retaining the important features
- there is also no guarentee that one layer is sufficient to do that
- we therefore refrain from this approach
- another advantage however, not directly apparent comes from the

fact that the teacher d2v (image) model, and our trainable student model both need to be on the gpu during training

- trainer runs in inference
  - this includes the modality encoder of the teacher model
- student model also uses the modality encoder, that is, like the entire teacher, frozen and running in inference
- we can share the modality encoder between the teacher and the student model
- saves parameters -> less memory needed
- also, since we need for each step do a forward pass for teacher and student, we can simply use the result of the modality image encoder for both models -> we do not need to do it twice -> less compute needed
- So even though we are now constrained with embed dim and modality encoder size, reusing it and sharing it between teacher and student saves, time and memory, plus will probably yield a better student model, as it already receives rich features
- This should amortize/compensate the increased model size

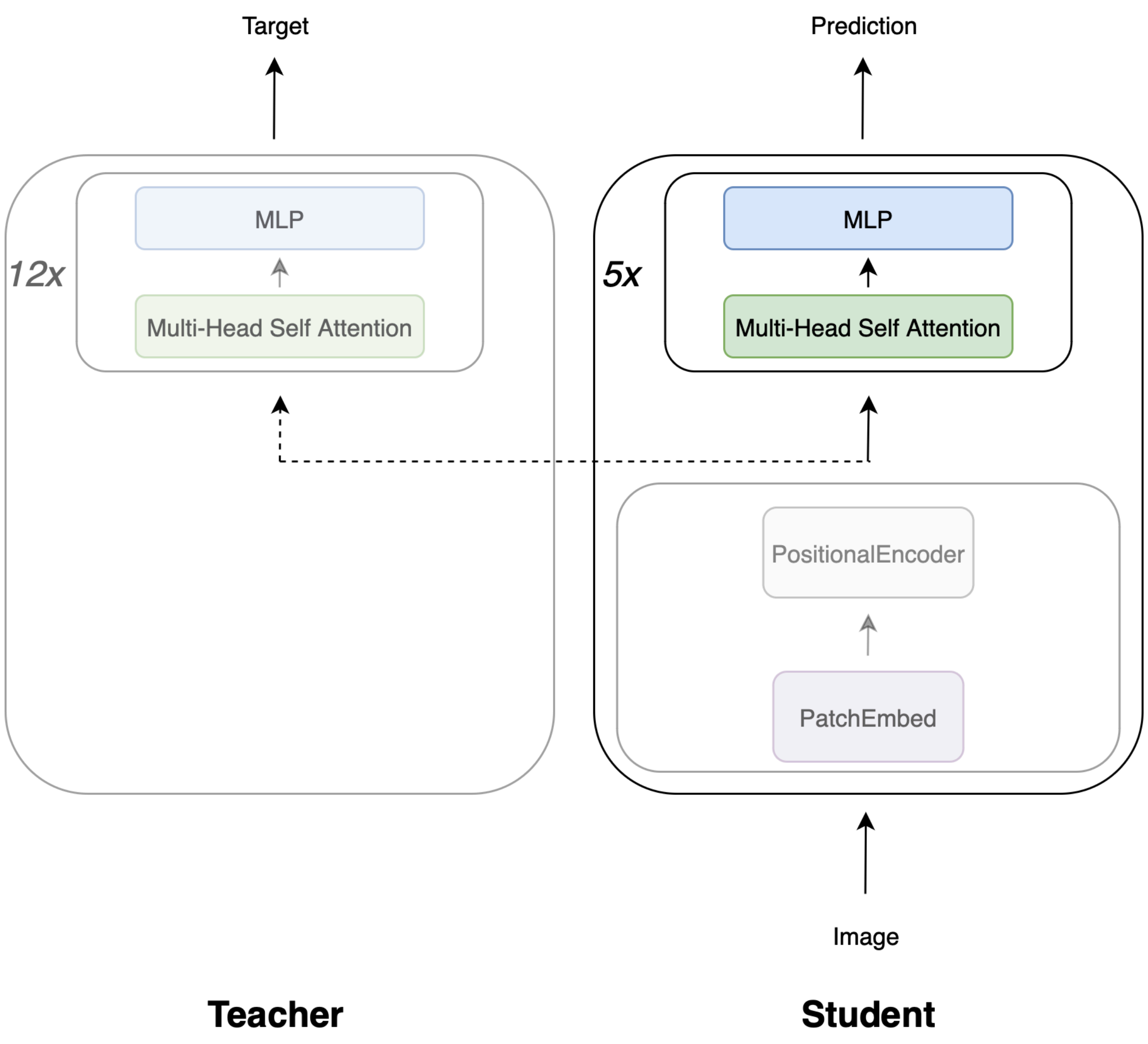

Figure 1: Test

Distillation Targets

Training Setup
Training and Validation

- In experiments on unimodal models we follow the same training setup
- We train the model for 30_000 steps (batches), with a batch size of 256
- We could in theory train it longer, but here we are just doing preliminary experiments to see

if the overall setup and idea works
- how do we know if the setup works, i.e. the model learns useful representations?
- usually done by running validation on the same dataset it is trained on
- but we do not have any lables, so not possible
- selecting some metric for that is not as straightforward as in supervised learning -> e.g. accuracy
- train loss also no good indicator (not expressive) -> we regress all timesteps using mse loss, so low/decreasing loss

is generally a sign that model adapts to data, but does not tell us anything about the quality of the representations

- also overfitting could also be the case (however it is unlikely, as the model is relatively small and imagenet is a large dataset)
- What we want to know is how good the embeddings are that the network learns
  - goal is to later utilize the ability of the model to create semantic representation of its input on downstream tasks, whether it is fine-tuning, low-, or zero-shot
- Therefore it is important/advantageous to already know during training how the model performs in the task it shall later be used for
- Obviously fine-tuning and even low-shot would take too much time during pre-training, as we do it repeatedly
- Also, especially fine-tuning might distort how the model is improving/performing during pre-training as we would give the model

the incentive to adjust the produced representations to the specific task, based on the labels

- So it could happen that we might get decent results through fine-tuning while to model performs poorly during pre-training, meaning the model does not necessarily has to rely on what is has learning during pre-training, or maybe is hasn't learned at all

- Instead, we use zero-shot validation
  - We do not need specific fine tuning, we just do predictions, which is fast
  - In this case the performance on the downstream task fully depends on the ability of the model to produce semantically rich representations, which is exactly what the model is supposed to learn during pre-training

- What options/downstream applications and tasks are there for zero-shot?

- One popular one is retrieval (of images/text/audio)

- Here we have a dataset of samples that belong together in a certain way, for example samples have the same class (in the sense of a classification dataset)

- Here we create a so called memory bank of embeddings for a selected set of samples of a dataset

- And then have a set of "query" samples for which we try to find similar samples in the memory bank by also computing representations/embeddings for the query

samples and matching them through the cosine similarity to samples in the memory bank, before we use to cosine similarity we also normalize the embeddings

- In order to here match a sample to other samples of the same class in the memory bank, the process is fully reliant on the representations created by the model

- in this case we can use accuracy
  - we consider a prediction correct, if the highest cosine similarity for a query sample is with a sample in the memory bank that has the same class
  - else it is incorrect
  - we can also use the top-k accuracy, in which we consider a prediction correct if one of the k highest cosine similarities for a query sample with respect to sampels in the memory bank belongs to a sample of the same class
  - we use top-3 accuracy
    - means we consider a prediction correct if among the 3 most similar samples in the memory bank one is of the same class as the query sample
  - often top-5 accuracy used, as in imagenet benchmarks
  - but would be too easy, especially for CIFAR-10, where we only have 10 classes -> random guessing would already give us 50% top-5 accuracy

  - top-1 accuracy is basically normal accuracy
- we do zero-shot validation for CIFAR-10 and CIFAR-100 seperatly
- we use the images of the training subsets of CIFAR-10 (CIFAR-100) as memory bank, and the images of the test subsets of CIFAR-10 (CIFAR-100) as query samples
  - mimics real world application, e.g. recommendation of similar images,
  - we have an image that we have not seen before (test set), for which we want to find similar images from images we have seen before (training set)
- we can also invert the process, and use the test set as memory bank and the training set as query samples
- gives two measures of representation quality for each dataset

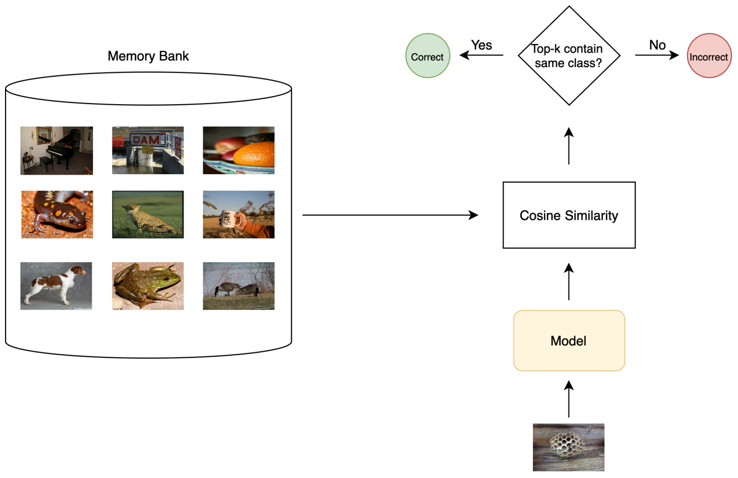

Figure 2: Test

- As a dataset we use CIFAR-10 and CIFAR-100
- Small enough so retrieval can be done fast, enough examples so that it can be challenging and not too easy
- Especially the case for CIFAR-100, where we have 100 classes
- If the model creates rich representations, it should be able to distinguish between the classes
- The better we can distinguish between the classes, based on the

produced representations, the better the model has learned

$$\cos(\|f(x)\|_2)$$

```python
def zero_shot_retrieval(model, train_images, test_images,
    train_labels, test_labels, top_k):

    memory_bank = model(train_images)
    queries = model(test_images)

    memory_bank = normalize(memory_bank, axis=-1)
    queries = normalize(queries, axis=-1)

    scores = queries @ memory_bank.T

    indices = similarity_scores.topk(top_k, dim=-1).indices

    matches = (train_labels[indices]==test_labels.unsqueeze(1)).any(dim=-1)

    top_k_accuracy = matches.sum() / len(test_labels)

    return top_k_accuracy
```

Listing 1: Test