Increasing Negative Examples for ITC

- current approach uses DDP with batch size of 256
- we use two GPUs, so combined batch size is 512
- negative samples are gathered from all devices to get more negative samples [1]
- as shown in experiments with supervised teacher (SHRe), improves performance
- implementation of image-text models that use contrastive learning, use much larger batch sizes
  - e.g. CLIP uses batch size of 32,768 [2]
- larger batch sizes would require more GPUs, as higher batch sizes per device than 256 lead to OOM errors
  - cost aspect would not be a big problem
  - even though multiple GPUs are more expensive, training time is reduced by a factor of the number of GPUs used
  - so if using two GPUs instead of one, training time is halved
- search for alternatives, that allows for more negative samples
- before we search for alternatives for more negative samples, we study the effect of using the same approach described prior, i.e. DDP with batch size of 512, but we do not gather negative samples from all devices, as done in the previous experiments
- means effective batch size is 512, but ITC done on 256 negative samples
- results will show us how important increased number of negative samples truly is for the performance of our model
- one alternative is to use a FIFO queue based memory bank
- memory bank stores image and text embeddings of previous batches
- with every batch, we update the memory bank with the current batch embeddings, while discarding the oldest embeddings
- embeddings from memory bank are then used, together with the embeddings of the current batch, as negative examples
- given a batch size of N, and a memory bank size of M, we can have $N - 1 + M$ negative samples
- given a batch size of 256, and a memory bank size of 768, we can have 1023 negative samples, which is the same as having an effective batch size of 1024, or using four GPUs with DDP and a batch size of 256
  - this "simulation" of bigger batch sizes with a memory bank would however only apply to the negative examples of the contrastive loss
  - at the end in our case the gradients are still computed with an batch size 512 (256 per device)
- nonetheless, we experiment with the aforementioned approach
- the training stays the same, but we additionall maintain a memory bank
- doing this, we observe a significant drop in performance

Model	ImageNet-1k Accuracy	Retrieval MSCOCO	Retrieval Flickr30K	Batch Size	ITC # Negative Examples	Wall Clock Time (in h)
CLIP [2]	**72.6**	**66.73**	**90.1**	32,768	32,767	-
Sx3HRe$_{\text{DDP}}$	26.1	66.3	42.4	512	511	7.09
Sx3HRe$_{\text{MB}}$	17.8	54.4	30.3	256	511	11.88
Sx3HRe$_{\text{DDP}_{\text{NG}}}$	17.8	54.4	30.3	512	255	7.09

Table 1: CLIP

- might come as a surprise, but is a well known
- problem lies is rapidly changing model weights during training
- when using actual batch size of 512, or DDP, all negative examples (the representations) come from the same model with the same weights
- when using a memory bank, the negative examples are stored from previous batches, or steps respectively
- means from an older model with different weights
- leads to a fast shift in embedding space, and in the embeddings in general
- so pronounced, that even representations from the immediate previous steps are so different to the represenations of the current step, that they are not useful as negative examples
- memory bank was initially introduced as a mapping of the complete training dataset
- embedding of each sample is stored in the memory bank
- for each batch, K samples are drawn from the memory bank to be used as negative examples
- representations of samples that are currently in the batch are then updated in the memory bank
- has the same problem that representations come from an older model with different weights, especially if the dataset is large
- this is partly mitigated by using a proximal regularization, show in Equation 1

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

- term $-\log h(i, \boldsymbol{v}_i^{t-1})$ can be ignored for now, and just be considered as the contrastive loss for simplicity
- more interesting is proximal regularization term $\lambda * \|\boldsymbol{v}_i^t - \boldsymbol{v}_i^{t-1}\|_2^2$
- is the mean squared error between the representation $\boldsymbol{v}_i$ of a training example i in the current batch, e.g. an image, at time step t, and the representation of the same training example at time step $t-1$ (the last update)
- enforces that representation of a training example does not change too rapidly
- λ controls how strong this regularization is
- allowes for more stable negative examples

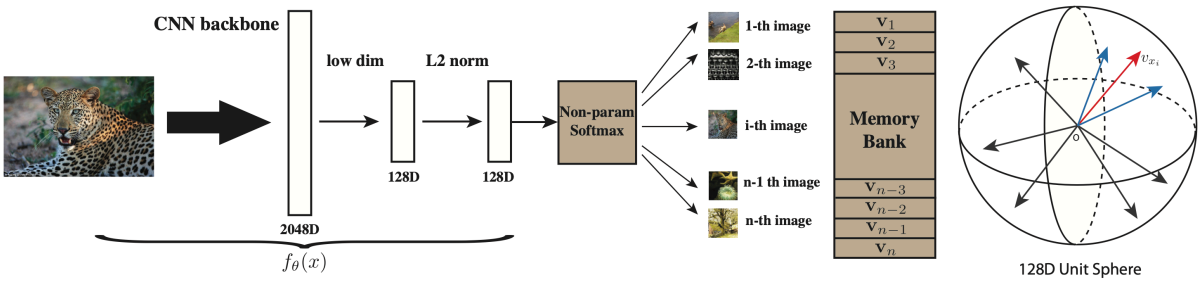

Figure 1:

- problem also identified by authors of MoCo [3]
- authors experiments, displayed in Figure 2, show that even a memory bank that maps all training examples performs significantly worse than using the actual batch size it tries to mimic
- especially with lower batch sizes
- this type of memory bank need to sample up to 65,536 ($K = 65,536$) negative examples to match actual, comparatively small, batch size of 1024

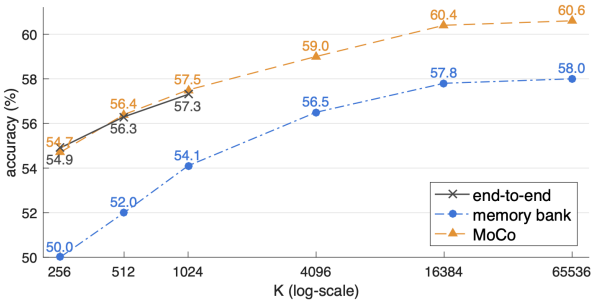

Figure 2:

- we can't use the memory bank style of [4], since we have 3,264,868 training examples
  - each embedding has a size of 768, considering storing them at full precision, so float32, we would need $3,264,868 * 768 * 4B > 10e^9 B = 10$ GB of additional GPU memory
- we suspect that using a proximal regularization term, as in Equation 1, might help to stabilize our memory bank approach
- however, we cannot use it -> the term is based on the representation of the same training example at the previous time step
- but our memory bank is significantly smaller than the training dataset, and older samples are dequeued, so the same training example will never be in the memory bank and at the same time in the current batch
- authors of MOCO propose a different approach to solving this problem
- they use a queue based memory bank, much smaller than the training dataset (cite MoCo v2), so same approach as we use
- but the negative examples in the memory bank are not updated by the model that is trained, but instead by a momentum encoder, which is a moving average of the actual model weights
- weight update of the momentum encoder is given by

$$\theta_k = m * \theta_k + (1 - m) * \theta_q \tag{2}$$

- with θ_k being the momentum encoder weights, θ_q the actual model weights, and m the momentum factor
- set to to $m = 0.999$, meaning the momentum encoder is updated very slowly
- that way negative examples in the memory bank are updated by a model with similar weights as the model that is trained
- makes them more stable and consistent
- can be seen as keeping the model consistent that produces the negative examples, instead of making the negative examples themselves consistent through an additional regularization term (Equation 1)
- disadvantage is that a copy of the same model that is being trained has to be kept in memory
- even though no gradients are needed for the momentum encoder, it still requires additional GPU memory

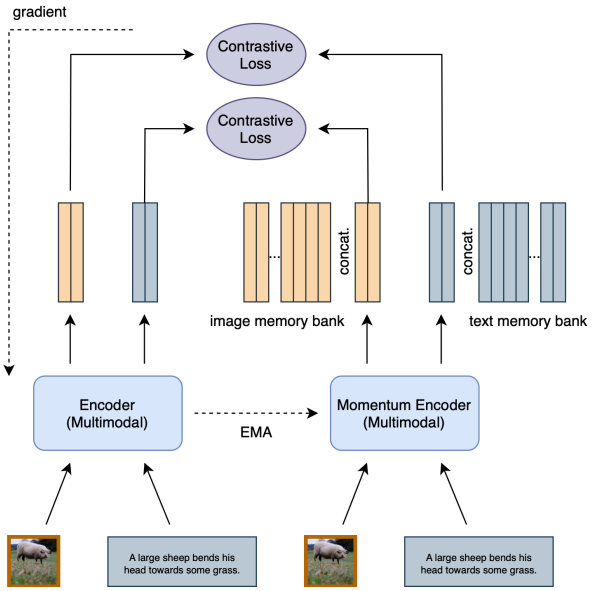

Figure 3: Figure inspired by MoCo v2 [5]

- in conclusion:

1. can't use FIFO queue based memory bank, as representations of negative examples are inconsistent
2. can't use memory bank that maps all training examples, as it would require too much additional GPU memory
3. proximal regularization not applicable to a memory bank that is smaller than the training dataset

- only alternative is to use a momentum encoder as in MoCo
- therefore, we opt for this approach
- experimental setup remains the same, but we add a momentum encoder, which is a copy of our (student) model that is trained
- oriented on ALBEF [6], we use a memory bank of size 65,536 and a momentum factor of 0.995
  - both are hyperparameters that also lead to good results in MoCo, where this approach was first introduced [3]
- however, we get an OOM error

- this is not surprising, as we have a very large memory bank of size 65,536
- as seen in Figure 3, we can't just have one memory bank, as done in unimodal models like MoCo [3], because we need one memory bank for storing negative text examples and one for storing negative image examples
- also recall that we do contrastive loss on both FFN layers of the shared Transformer layer
- so we need two memory banks for each FFN layer
- so we need four memory banks of size 65,536 each
- we would like to keep the size of the memory bank, as this is important for performance (seen in Figure 2)
- therefore, we perform two optimizations:
  - we only do the contrastive loss on the last FFN layer of the shared Transformer layer, so on its cls token output
  - will reduce the GPU memory by a margin, as we only need two memory banks of size 65,536 each
  - also, while the memory bank of the last FFN layer stores embeddings of size 768, because this is the hidden dim used for our model, the memory bank of the first FFN layer stores embeddings of size 3072
    - this is because in Transformer layers the first FFN layer of the MLP expands the hidden dim by a factor of four ($4 * 768 = 3072$)
  - just removing this memory bank will save us 1.6 GB of GPU memory
  - reason: assuming embeddings are in full precision, so float32, we would need $3072 * 4B = 12,288B$, so approximately 12.3 KB per embedding
  - for one memory bank, this means $12,288 * 65,536 = 805,8$ MB of GPU memory is required
  - for two memory banks, this means $805,8 \text{ MB} * 2 \approx 1.6 \text{ GB}$ of GPU memory is required
- this is still not enought to prevent OOM errors, and we identify a further optimization
- usually, forward pass of momentum encoder is done after the forward pass of the model that is trained
- that means that during the forward pass of the momentum encoder, where GPU memory is required to store the activations, gradients and activations of the actual model are also stored on the GPU
- this approach, using in MoCo [3] and ALBEF [6], is illustrated left in Figure 4
- because we did not encoder OOM errors before using the momentum encoder, and we observed that the GPU memory in previous experiments was almost fully utilized, even without a momentum encoder, we suspect that the forward pass of the momentum encoder is the reason
- we therefore perform the forward pass of the momentum encoder before any work is done in one step of the training loop
- so momentum update and forward pass of the momentum encoder is done first
- this way, activations of momentum encoder are freed before the forward pass of the model that is trained
- performance stays the same, as the work that is done remains the same
- with this strategy, we avoid OOM errors

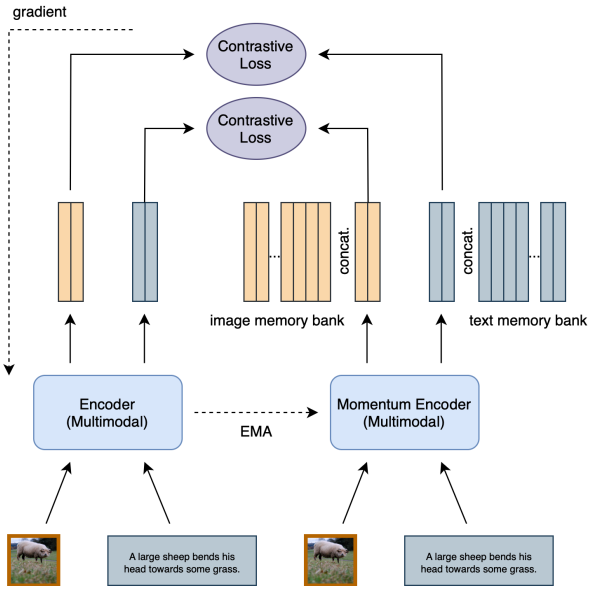

Figure 4: me_forward_comparison

...

- best approach would be to use no contrastive loss -> all the work we did before was because contrastive learning requires large number of negative examples
- BEiT-3 give good reason to discard it

Bibliography

[1] 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

[2] 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.

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

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

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

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