A Appendix

The appendix is composed of 9 parts. In Sec. A.1, we discuss the gradient of multi-modal contrastive loss. In Sec. A.2, we elaborate the derivations and implementations of decoupled gradient accumulation. In Sec. A.3, we introduce the detailed calculation of coin flipping mixup loss. In Sec A.4, we explore another sampling strategy related to our proposed debiased sampling. In Sec A.5, we show that debiased sampling tackles various kinds of data bias. In Sec A.6, we show that debiased sampling works well on a single dataset. In Sec A.7, we provide linear probing results on more datasets. In Sec. A.8, we detail open-source and web pre-training data. In Sec. A.9, we provide training details for reproducing our strong baseline.

A.1 Gradient of Multi-Modal Contrastive Loss

Formulating gradients of contrastive loss. Within each training batch, define the similarity between the *query* j and the *key* k as s_{jk}. The ground-truth label corresponding to s_{jk} is represented by $y_{jk} \in \{0, 1\}$. The contrastive loss can be formulated as:

$$\mathcal{L} = \sum_j \sum_k y_{jk} \log \left(\frac{\exp(s_{jk})}{\sum_l \exp(s_{jl})} \right), \tag{1}$$

where the temperature parameter is omitted for simplification. Then, the gradient of the popular contrastive loss could be written as:

$$\begin{aligned}
\nabla_\theta \mathcal{L} &= -\sum_j \sum_k y_{jk} \nabla_\theta \log \left(\frac{\exp(s_{jk})}{\sum_l \exp(s_{jl})} \right) \\
&= -\sum_j \sum_k y_{jk} \left(\nabla_\theta s_{jk} - \nabla_\theta \log \sum_l \exp(s_{jl}) \right) \\
&= -\sum_j \sum_k y_{jk} \left(\nabla_\theta s_{jk} - \frac{1}{\sum_l \exp(s_{jl})} \nabla_\theta \sum_l \exp(s_{jl}) \right) \\
&= -\sum_j \sum_k y_{jk} \left(\nabla_\theta s_{jk} - \sum_l \frac{\exp(s_{jl})}{\sum_m \exp(s_{jm})} \nabla_\theta s_{jl} \right) \\
&= -\sum_j \sum_k y_{jk} \left(\nabla_\theta s_{jk} - \sum_l \bar{p}_{jl} \nabla_\theta s_{jl} \right) \\
&= -\sum_j \sum_k y_{jk} \nabla_\theta s_{jk} + \sum_j \sum_k y_{jk} \sum_l \bar{p}_{jl} \nabla_\theta s_{jl},
\end{aligned} \tag{2}$$

where we place a vinculum on a value to indicate its gradient is detached. Due to $\sum_k y_{jk} = 1$, we rewrite Eqn.(2) as:

$$\begin{aligned}\nabla_\theta \mathcal{L} &= -\sum_j \sum_k y_{jk} \nabla_\theta s_{jk} + \sum_j \sum_l \bar{p}_{jl} \nabla_\theta s_{jl} \\ &= -\sum_j \sum_k y_{jk} \nabla_\theta s_{jk} + \sum_j \sum_k \bar{p}_{jk} \nabla_\theta s_{jk} \\ &= \sum_j \sum_k \left(\bar{p}_{jk} - y_{jk}\right) \nabla_\theta s_{jk} \\ &= \sum_j \sum_k \left(\bar{p}_{jk} - y_{jk}\right) \left(\bar{x}_j \nabla_\theta x_k + \bar{x}_k \nabla_\theta x_j\right),\end{aligned} \tag{3}$$

where x_j and x_k are embeddings of sample j and k. Regarding the sample j as the *query*, its gradient comes to $\sum_k \left(\bar{p}_{jk} - y_{jk}\right) \left(\bar{x}_j \nabla_\theta x_k + \bar{x}_k \nabla_\theta x_j\right)$. If sample j and k are from different machines, detaching gradients makes the term $\bar{x}_j \nabla_\theta x_k$ to 0, since x_k serves as a constant term in the gradient calculation process.

Detaching gradients in multi-modal contrastive loss. Subsequently, we study the gradients of multi-modal contrastive loss. We start with minor notation adjustments to cater for the multi-modal setting. The calculation of multi-modal contrastive loss can be divided into image-to-text (I2T) matching and text-to-image (T2I) matching parts. Gradients of I2T and T2I matching losses are:

$$\nabla_\theta \mathcal{L}^{I2T} = \sum_j \sum_k \left(\bar{p}_{jk}^{I2T} - y_{jk}^{I2T}\right) \left(\bar{i}_j \nabla_\theta t_k + \bar{t}_k \nabla_\theta i_j\right), \tag{4}$$

$$\nabla_\theta \mathcal{L}^{T2I} = \sum_j \sum_k \left(\bar{p}_{jk}^{T2I} - y_{jk}^{T2I}\right) \left(\bar{t}_j \nabla_\theta i_k + \bar{i}_k \nabla_\theta t_j\right), \tag{5}$$

where i and t represent image and text embeddings. For pairs (i_j, t_k) and (t_j, i_k) from *different* machines, gather operations with detaching gradients would produce the following gradients on the machine of j:

$$\tilde{\nabla}_\theta \mathcal{L}^{I2T} = \left(\bar{p}_{jk}^{I2T} - y_{jk}^{I2T}\right) \bar{t}_k \nabla_\theta i_j, \tag{6}$$

and the gradient on k's machine:

$$\tilde{\nabla}_\theta \mathcal{L}^{T2I} = \left(\bar{p}_{kj}^{T2I} - y_{kj}^{T2I}\right) \bar{i}_j \nabla_\theta t_k. \tag{7}$$

We add a tilde symbol on the gradient $\tilde{\nabla}_\theta \mathcal{L}$, indicating the calculation involves detaching gradients. Then, we have:

$$\nabla_\theta \mathcal{L}^{I2T} + \nabla_\theta \mathcal{L}^{T2I} \neq \left(\tilde{\nabla}_\theta \mathcal{L}^{I2T} + \tilde{\nabla}_\theta \mathcal{L}^{T2I}\right). \tag{8}$$

Mathematically, detaching gradients in multi-modal contrastive loss yields incorrect gradients. Experiments in Sec. 4.1 of the manuscript prove that gradient reserved gather operations are beneficial in multi-modal contrastive learning.

A.2 Decoupled Gradient Accumulation

Decoupling the gradient of multi-modal contrastive loss. Inspired by a technical report[1] which decouples the gradient of single-modal contrastive loss,

[1] https://spaces.ac.cn/archives/8471

we further generalize it to the multi-modal scenario. According to Eqn.(4) and (5), we have:

$$\begin{aligned}
\nabla_\theta \mathcal{L}^{I2T} &= \sum_j \sum_k \left(\bar{p}_{jk}^{I2T} - y_{jk}^{I2T}\right) \left(\bar{i}_j \nabla_\theta t_k + \bar{t}_k \nabla_\theta i_j\right) \\
&= \sum_k \nabla_\theta \left(\sum_j \left(\bar{p}_{jk}^{I2T} - y_{jk}^{I2T}\right) \bar{i}_j \right) t_k \\
&+ \sum_j \nabla_\theta \left(\sum_k \left(\bar{p}_{jk}^{I2T} - y_{jk}^{I2T}\right) \bar{t}_k \right) i_j
\end{aligned} \tag{9}$$

$$\begin{aligned}
\nabla_\theta \mathcal{L}^{T2I} &= \sum_j \sum_k \left(\bar{p}_{jk}^{T2I} - y_{jk}^{T2I}\right) \left(\bar{t}_j \nabla_\theta i_k + \bar{i}_k \nabla_\theta t_j\right) \\
&= \sum_k \nabla_\theta \left(\sum_j \left(\bar{p}_{jk}^{T2I} - y_{jk}^{T2I}\right) \bar{t}_j \right) i_k \\
&\quad + \sum_j \nabla_\theta \left(\sum_k \left(\bar{p}_{jk}^{T2I} - y_{jk}^{T2I}\right) \bar{i}_k \right) t_j \\
&= \sum_j \nabla_\theta \left(\sum_k \left(\bar{p}_{kj}^{T2I} - y_{kj}^{T2I}\right) \bar{t}_k \right) i_j \\
&\quad + \sum_k \nabla_\theta \left(\sum_j \left(\bar{p}_{kj}^{T2I} - y_{kj}^{T2I}\right) \bar{i}_j \right) t_k.
\end{aligned} \tag{10}$$

Then, the total gradient can be written as:

$$\begin{aligned}
\nabla_\theta \mathcal{L} &= \nabla_\theta \mathcal{L}^{I2T} + \nabla_\theta \mathcal{L}^{T2I} \\
&= \sum_j \nabla_\theta \left(\sum_k \left(\bar{p}_{jk}^{I2T} - y_{jk}^{I2T} + \bar{p}_{kj}^{T2I} - y_{kj}^{T2I}\right) \bar{t}_k \right) i_j \\
&\quad + \sum_k \nabla_\theta \left(\sum_j \left(\bar{p}_{jk}^{I2T} - y_{jk}^{I2T} + \bar{p}_{kj}^{T2I} - y_{kj}^{T2I}\right) \bar{i}_j \right) t_k.
\end{aligned} \tag{11}$$

As suggested in Eqn.(11), we mathematically decouple the gradient into two parts. One part of gradient is only related to stop-gradient embeddings ($\bar{t}_k$ and $\bar{i}_j$), and the other part only depends on embeddings with gradients (t_k and i_j).

Implementation of decoupled gradient accumulation. In the conventional multi-step gradient accumulation, we are not allowed to obtain embeddings (with gradients) from different training sub-iterations. However, we can cache stop-gradient embeddings of the large batch, and then calculate the correct gradient with Eqn.(11) in each sub-iteration. With forwarding the large batch and caching stop-gradient embeddings, our decoupled gradient accumulation can accurately produce the gradient produced by large-batch training.

Complete pseudo code in a PyTorch-like style. In Sec. 4.2 of the manuscript, we provide a simplified pseudo code of decoupled gradient accumulation. In Algorithm 1, we provide a detailed and complete pseudo code of decoupled gradient accumulation for better understanding. In the implementation

of previous methods [11,5], the temperature of contrastive loss is learnable. Thus, in the implementation of decoupled gradient accumulation, we need consider the gradient of the temperature variable. As shown in Algorithm 1, we detach the gradient of temperature (with `torch.no_grad`) for forwarding the large batch, and then calculate the gradient of temperature in each sub-iteration. Besides, a square-root should be applied on the value of temperature for correctly calculating the scale of temperature. Note that encoders could contain modules of randomness, *e.g.*, dropout layers are widely applied in the BERT [4]. Thus, forwarding the same sample two times could produce different embeddings. To this end, we set the identical random seed for twice forwarding processes, eliminating the randomness and stabilizing the training.

A.3 Coin Flipping Mixup Loss

We detail the coin flipping mixup loss function by following notations defined in Sec. 3.2 of the manuscript. We first define a batch $\{(I_1, T_1), (I_2, T_2), \ldots, (I_N, T_N)\}$ of N image-text pairs. Then, we uniformly sample a γ from the range $[0, 1]$.

$\gamma > 0.5$**:** We apply the mixup on the images, and the mixed batch can be denoted as $\{(\tilde{I}_1, T_1), (\tilde{I}_2, T_2), \ldots, (\tilde{I}_N, T_N)\}$. The image-to-text matching part can be formulated as:

$$\begin{aligned}\mathcal{L}_{\tilde{I}2T} = \lambda * &\left(-\frac{1}{N}\sum_{j=1}^{N}\log\frac{\exp(\tilde{i_j}\cdot t_j/\tau)}{\sum_{k=1}^{N}\exp(\tilde{i_j}\cdot t_k/\tau)}\right) \\ &+ (1-\lambda) * \left(-\frac{1}{N}\sum_{j=1}^{N}\log\frac{\exp(\tilde{i_j}\cdot t_{N-j}/\tau)}{\sum_{k=0}^{N-1}\exp(\tilde{i_j}\cdot t_{N-k}/\tau)}\right).\end{aligned} \tag{12}$$

And the text-to-image part can be formulated as:

$$\begin{aligned}\mathcal{L}_{T2\tilde{I}} = \lambda * &\left(-\frac{1}{N}\sum_{j=1}^{N}\log\frac{\exp(t_j\cdot\tilde{i_j}/\tau)}{\sum_{k=1}^{N}\exp(t_j\cdot\tilde{i_k}/\tau)}\right) \\ &+ (1-\lambda) * \left(-\frac{1}{N}\sum_{j=1}^{N}\log\frac{\exp(t_j\cdot\tilde{i}_{N-j}/\tau)}{\sum_{k=0}^{N-1}\exp(t_j\cdot\tilde{i}_{N-k}/\tau)}\right).\end{aligned} \tag{13}$$

$\gamma \leq 0.5$**:** We apply the mixup on the texts, and the mixed batch can be denoted as $\{(I_1, \tilde{T}_1), (I_2, \tilde{T}_2), \ldots, (I_N, \tilde{T}_N)\}$. The image-to-text matching part can be formulated as:

$$\begin{aligned}\mathcal{L}_{I2\tilde{T}} = \lambda * &\left(-\frac{1}{N}\sum_{j=1}^{N}\log\frac{\exp(i_j\cdot\tilde{t_j}/\tau)}{\sum_{k=1}^{N}\exp(i_j\cdot\tilde{t_k}/\tau)}\right) \\ &+ (1-\lambda) * \left(-\frac{1}{N}\sum_{j=1}^{N}\log\frac{\exp(i_j\cdot\tilde{t}_{N-j}/\tau)}{\sum_{k=0}^{N-1}\exp(i_j\cdot\tilde{t}_{N-k}/\tau)}\right).\end{aligned} \tag{14}$$

Algorithm 1 Pseudo code in a PyTorch-like style.

```
# stable_random_seed: random seed generated by time.time()
# temp: temperature

# fix dropout with fixed random seed
setup_seed(random_seed)

with torch.no_grad():
    # stop-grad forward
    img_emb_local, text_emb_local = [], []
    for _idx_l in range(0, bs, bs_train):
        _data_batch = data_batch[_idx_l: _idx_l + bs_train]
        _img_embs, _text_embs, temp = model(_data_batch)
        img_emb_local.append(_img_embs)
        text_emb_local.append(_text_embs)

    # concatenate embeddings of each GPU
    img_emb_local = torch.cat(img_emb_local, dim = 0)
    text_emb_local = torch.cat(text_emb_local, dim = 0)

    # gather embeddings of all GPUs
    img_emb_global = torch.cat(gather(img_emb_local), dim = 0)
    text_emb_global = torch.cat(gather(text_emb_local), dim = 0)

    # calculate cosine similarity
    sim_i2t_nm = img_emb_global @ text_emb_local.T / temp
    sim_i2t_mn = img_emb_local @ text_emb_global.T / temp

    # calculate the normalized factor in softmax function
    sim_i2t_esum_local = torch.sum(torch.exp(sim_i2t_mn), dim = 1)
    sim_t2i_esum_local = torch.sum(torch.exp(sim_i2t_nm.T), dim = 1)
    sim_i2t_esum = torch.cat(gather(sim_i2t_esum_local), 0).unsqueeze(dim = 1)
    sim_t2i_esum = torch.cat(gather(sim_t2i_esum_local), 0).unsqueeze(dim = 1)

    # calculate the probability matrix
    prob_i2t_mn = torch.exp(sim_i2t_mn) / sim_i2t_esum[bs * rank: bs * (rank + 1), :]
    prob_t2i_nm = torch.exp(sim_i2t_mn.T) / sim_t2i_esum
    prob_i2t_nm = torch.exp(sim_i2t_nm) / sim_i2t_esum
    prob_t2i_mn = torch.exp(sim_i2t_nm.T) / sim_t2i_esum[bs * rank: bs * (rank + 1), :]

    left_I = (prob_i2t_mn + prob_t2i_nm.T) @ text_emb_global - text_emb_local * 2
    left_I /= torch.sqrt(temp)
    left_T = (prob_i2t_nm.T + prob_t2i_mn) @ img_emb_global - img_emb_local * 2
    left_T /= torch.sqrt(temp)

# Fix dropout with fixed random seed
setup_seed(random_seed)

# forward with grad
for _idx_l in range(0, bs, bs_train):
    _left_I = left_I[_idx_l: _idx_l + bs_train]
    _left_T = left_T[_idx_l: _idx_l + bs_train]
    _data_batch = data_batch[_idx_l: _idx_l + bs_train]

    _img_embs, _text_embs, temp = model(_data_batch)

    loss_i = _left_I * _img_embs
    loss_t = _left_T * _text_embs
    # loss corresponds to Eqn.(11)
    loss = (loss_i + loss_t).sum() / 2 / bs / torch.sqrt(temp)
    # backward propagation
    loss.backward()

# update model parameters
update(model.param)
```

And the text-to-image part can be formulated as:

$$\begin{aligned}\mathcal{L}_{\tilde{T}2I} = \lambda * \left(-\frac{1}{N}\sum_{j=1}^{N} \log \frac{\exp(\tilde{t_j} \cdot i_j/\tau)}{\sum_{k=1}^{N} \exp(\tilde{t_j} \cdot i_k/\tau)} \right) \\ + (1-\lambda) * \left(-\frac{1}{N}\sum_{j=1}^{N} \log \frac{\exp(\tilde{t_j} \cdot i_{N-j}/\tau)}{\sum_{k=0}^{N-1} \exp(\tilde{t_j} \cdot i_{N-k}/\tau)} \right).\end{aligned} \tag{15}$$

Generally, the coin flipping mixup loss $\mathcal{L}_{\text{coin}}$ can be formulated as:

$$\mathcal{L}_{\text{coin}} = \begin{cases} \mathcal{L}_{\tilde{I}2T} + \mathcal{L}_{T2\tilde{I}}, & \text{if } \gamma > 0.5 \\ \mathcal{L}_{I2\tilde{T}} + \mathcal{L}_{\tilde{T}2I}, & \text{if } \gamma \leq 0.5. \end{cases} \tag{16}$$

A.4 Discussions on Sampling Strategies

Except for random sampling and debiased sampling mentioned in the manuscript Sec. 3.1, we further explore another strategy, *i.e.*, sequential sampling. As the name suggests, sequential sampling pre-defines the sampling order of multiple datasets and generates batches from the sequence of datasets. Illustrations of three sampling strategies are shown in Figure 1.

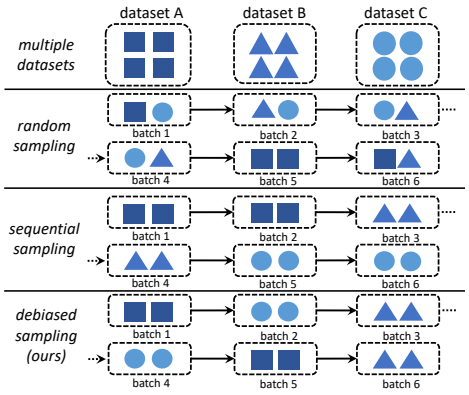

Fig. 1. Comparisons between random, sequential, and debiased sampling strategies.

In the following, we study the effect of different sampling strategies. RSUM scores of zero-shot image-text retrieval task on COCO and F30K datasets are provided in Figure 2, we notice the following phenomena on down-stream tasks:

- Sequential sampling also yields better results on downstream tasks than the random sampling.
- The order of datasets in sequential sampling exerts non-negligible influences on model performances.

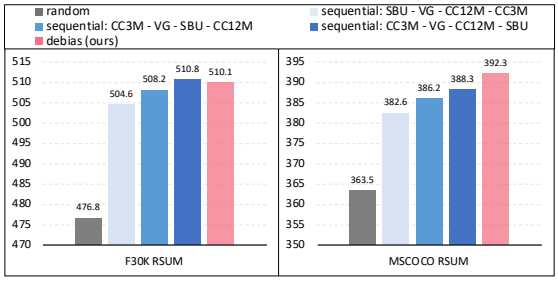

Fig. 2. Comparisons between random and sequential sampling.

Subsequently, we further discuss these observations.
(1) Why sequential sampling works? As proven in Sec. 3 of the manuscript, debiased learning greatly benefits the contrastive vision-language pre-training. We believe that sequential sampling also tackles the dataset bias issue, since it also ensures the samples within a training batch come from one dataset.
(2) Why does the order matter? It is observed that adjusting the order of datasets in sequential sampling exerts non-negligible influences on model performances. We conjecture that the domain relevance between the "last seen" dataset and the downstream dataset is directly proportional to model performances, especially for zero-shot scenarios. For instance, SBU, MSCOCO, and F30K provide images with visually relevant captions, while CC3M and CC12M contain images coupled with noisy or visually irrelevant captions. Results in Figure 2 validate that setting SBU as the last dataset leads to consistently superior results, compared with CC3M or CC12M being the last one.
(3) Drawback of sequential sampling. For sequential sampling, undoubtedly, enumerating the order of collected datasets is not acceptable for real-scenario applications, and the sequence needs further adjustment if new datasets are introduced. Our proposed debiased sampling effectively tackles this problem, and achieves better results.

A.5 Discussions on Dataset Bias

The dataset bias could be generally divided into two types, *i.e.*, semantic bias and context bias. Semantic bias corresponds to that semantics can be totally different between datasets, as mentioned in the manuscript. Context bias corresponds to image and text contexts, *e.g.*, image style, caption lengths and so on. In this part, we demonstrate that debiased sampling works well on solving such context bias.

For avoiding semantic bias, we use only CC3M in the following. We split CC3M into part A with long captions and part B with short captions. Then, we introduce image style bias by adding a red bounding box to each image of part A. We use random sampling to train a CLIP model and conduct illustrations in the Figure 3. Due to context bias, features of A (red) and B (blue) are separated, and gradients are inferior when training with random sampling. Debiased sampling

tackles context bias and improves RSUMs from 211.8/328.0 to 239.5/367.5 on COCO/F30K.

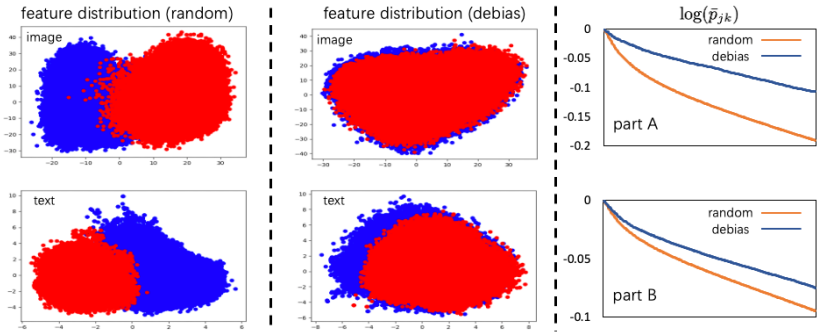

Fig. 3. Effects of debiased sampling on context bias.

Dataset bias and biased data (feature) distribution are different concepts. Both biased feature distributions and inferior gradients are effects, while dataset bias is the cause. Dataset bias is the inherent property of training data, which exists before training. The CLIP model captures such bias, allowing the model to distinguish samples with different biases easily. Figure 3 and 4 in the manuscript both reveal bad effects of dataset bias in CLIP and prove the bias is captured by the model.

A.6 Application of Debiased Sampling on a Single Dataset

Debiased sampling also works well on a single source dataset. Concretely, we extract features of CC12M data, apply the KMeans on features, and produce 100 clusters. Then, we regard 100 clusters as 100 data sources, and apply debiased sampling for training a CLIP model, improving RSUMs on COCO/F30K from 370.8/502.5 to 384.4/511.1.

A.7 More Linear Probing Results

We provide linear probing results on other datasets. Six datasets are included, *i.e.*, CUB-200-2011 [14] (200 categories), Food-101 [1] (101 categories), Oxford Pets [10] (37 categories), FGVC Aircraft [8] (100 categories), iNaturalist-17 [13] (5089 categories), and Places365 [15] (365 categories). Due to the pre-training dataset and linear probing hyper-parameters of CLIP is not accessible (*e.g.*, parameters are obtained by grid search with `sklearn`). We mainly compare ZeroVL with our re-implemented CLIP. Results are reported in Table 1, and ZeroVL consistently outperforms CLIP on all datasets, which further validates the effectiveness of our baseline on linear probing tasks.

	pre-training				linear probing					
	computation device	count	data	input size	CUB	Food101	Pets	Aircraft	iNat17	Places365
CLIP (our impl.)	V100	8	14M	224	61.9	84.3	85.9	50.0	43.7	53.6
CLIP (our impl.)	V100	128	14M	224	75.0	88.5	89.9	51.3	53.4	54.0
ZeroVL (ours)	V100	8	14M	224	**75.7**	**90.9**	**92.0**	**52.1**	**54.3**	**55.8**

Table 1. Linear probing results.

A.8 Details of Pre-Training Datasets

Open-source datasets. Four widely-used image-text pair datasets are selected for pre-training. Details are as followed:

- SBU Captioned Photos (SBU) [9] contains 1M images with associated visually relevant captions.
- Visual Genome (VG) [7] consists of around 100K images and 5M captions, where each image is coupled with 50 captions. For training efficiency, we filter 5 out of 50 captions for each image according to largest areas of bounding box regions.
- Conceptual Captions 3M (CC3M) [12] contains around 3.3M images annotated with captions, collected from web data with an automatic collection pipeline.
- Conceptual 12M (CC12M) [2] is similar to CC3M and the collection pipeline is relaxed. Consequently, the data in CC12M is relatively noisier than CC3M.

A part of download links provided by CC3M and CC12M are lost. Collectively, our visual-linguistic corpus for pre-training is composed of around 14.23M image-text pairs from various domains.

Web data. The web data is mainly collected from an image library community Tuchong [2]. Due to the double-blind review policy, we are not allowed to provide the name of the community. Each image is coupled with a caption created by the image's author. The 100M web data comprises 14M academic data and 86M of web-crawling data (from Tuchong). For applying debiased sampling, we consider the web-crawling data as a prominent source and apply debiased sampling on datasets from five sources.

A.9 Training Details

We elaborate the training details of our strong baseline.

Data preparation. Batches are comprised by applying the debaised sampling strategy on academic pre-training datasets, *i.e.*, SBU, VG, CC3M and CC12M. Each image is randomly cropped to a rectangular region with aspect ratio sampled in $[3/4, 4/3]$ and area sampled in $[60\%, 100\%]$, then resized to 224×224 resolution. Regarding the corresponding text, we set the max length to 25 and

[2] https://www.tuchong.com

use a percentage of 20% input words for processing. For each word, we mask it, replace it with a random word, or delete it with a probability of 50%, 10% and 40%, respectively. We directly apply AutoAugment after crop operation and the policy is search on ImageNet [3]. Coin flipping mixup is also used in the training phase, and the α is set to 0.1 in the coin flipping mixup. During test, images are resized to 256×256 and center cropped to 224×224, while no specific process is applied to texts.

Model architecture. Image and text encoders are ViT-B/16 and BERT-Base, respectively. The image encoder is pre-trained on ImageNet [3] which could be directly obtained from the `timm` [4] library while the text encoder is pre-trained on BookCorpus [6] and English Wikipedia from the `HuggingFace` [5] library. `[CLS]` tokens from image and text encoders are extracted and then projected to 512-dim compact embeddings and ℓ-2 normalized for calculating the contrastive loss.

Training. AdamW optimizer is used for training and the weight decay is 1e-3. Based on our decoupled gradient accumulation, the dual-encoder model is trained for 20 epochs on 8 Nvidia V100 GPUs with a batch size of 16,384. The learning rate is initialized to 1e-4 and follows a cosine decay schedule. Notably, we set a minimum learning rate 1e-5 to avoid over-fitting. The embedding dimension for image and text representations is 512 and the trainable temperature of contrastive loss is initialized to 0.02.

References

1. Bossard, L., Guillaumin, M., Gool, L.V.: Food-101–mining discriminative components with random forests. In: ECCV. pp. 446–461 (2014)
2. Changpinyo, S., Sharma, P., Ding, N., Soricut, R.: Conceptual 12M: Pushing web-scale image-text pre-training to recognize long-tail visual concepts. In: CVPR (2021)
3. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: Imagenet: A large-scale hierarchical image database. In: CVPR (2009)
4. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: BERT: Pre-training of deep bidirectional transformers for language understanding. arXiv:1810.04805 (2018)
5. Jia, C., Yang, Y., Xia, Y., Chen, Y.T., Parekh, Z., Pham, H., Le, Q.V., Sung, Y., Li, Z., Duerig, T.: Scaling up visual and vision-language representation learning with noisy text supervision. In: ICML (2021)
6. Kiros, R., Zhu, Y., Salakhutdinov, R.R., Zemel, R., Urtasun, R., Torralba, A., Fidler, S.: Skip-thought vectors. In: NeurIPS (2015)
7. Krishna, R., Zhu, Y., Groth, O., Johnson, J., Hata, K., Kravitz, J., Chen, S., Kalantidis, Y., Li, L.J., Shamma, D.A., et al.: Visual Genome: Connecting language and vision using crowdsourced dense image annotations. In: IJCV (2017)
8. Maji, S., Rahtu, E., Kannala, J., Blaschko, M., Vedaldi, A.: Fine-grained visual classification of aircraft. arXiv:1306.5151 (2013)

[3] https://github.com/4uiiurz1/pytorch-auto-augment
[4] https://github.com/rwightman/pytorch-image-models
[5] https://huggingface.co

9. Ordonez, V., Kulkarni, G., Berg, T.: Im2text: Describing images using 1 million captioned photographs. In: NeurIPS (2011)
10. Parkhi, O.M., Vedaldi, A., Zisserman, A., Jawahar, C.: Cats and dogs. In: CVPR. pp. 3498–3505 (2012)
11. Radford, A., Kim, J.W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., et al.: Learning transferable visual models from natural language supervision. In: ICML (2021)
12. Sharma, P., Ding, N., Goodman, S., Soricut, R.: Conceptual Captions: A cleaned, hypernymed, image alt-text dataset for automatic image captioning. In: ACL (2018)
13. Van Horn, G., Mac Aodha, O., Song, Y., Cui, Y., Sun, C., Shepard, A., Adam, H., Perona, P., Belongie, S.: The inaturalist species classification and detection dataset. In: CVPR (2018)
14. Wah, C., Branson, S., Welinder, P., Perona, P., Belongie, S.: The caltech-ucsd birds-200-2011 dataset (2011)
15. Zhou, B., Lapedriza, A., Khosla, A., Oliva, A., Torralba, A.: Places: A 10 million image database for scene recognition. TPAMI (2017)