Supplementary Material for GRIT-VLP: Grouped Mini-batch Sampling for Efficient Vision and Language Pre-training

Jaeseok Byun[1⋆], Taebaek Hwang[2⋆], Jianlong Fu[3], and Taesup Moon[1⋆⋆]

[1] Department of ECE/ASRI/IPAI, Seoul National University
[2] Department of ECE, Sungkyunkwan University
[3] Microsoft Research Asia
wotjr3868@snu.ac.kr, gxq9106@gmail.com, jianf@microsoft.com, tsmoon@snu.ac.kr

1 Data and Implementation Details

Here we describe the details of software platform and dataset. All experiments are conducted with four NVIDIA A100 GPUs. We use Python 3.7 and Pytorch [4] with CUDA 11.1 to implement GRIT-VLP. Table 1 summarizes the statistics of pre-training dataset.

2 Details of Pre-training

2.1 Computation comparison

We note that GRIT-VLP does not use the momentum encoder and momentum distillation in the pre-training phase. Table 2 shows the computational costs when $N = 96$ and $M = 960$. As can be verified in Table 2, the model parameters, time per training one epoch, and queue size of GRIT-VLP are much smaller than the ALBEF, which clearly shows the efficiency of GRIT-VLP.

Table 1. Statistics of dataset

	COCO	VG	SBU	CC
# images	113K	100K	851K	2.82M
# texts	567K	769K	851K	2.82M

⋆ Equal contribution. This work was performed when Jaeseok Byun did an internship at Microsoft Research Asia.
⋆⋆ Corresponding author (tsmoon@snu.ac.kr)

Table 2. Computational costs

Model	Time per epoch	Parameters	Queue
ALBEF	3h 10m	420M	65536
GRIT-VLP	2h 30m	210M	48000

2.2 Overall Process of GRIT-VLP

Here, we provide a pseudo-code of the overall process of GRIT-VLP, mainly focusing on the GRIT we proposed. For simplicity, we omit the explanation about functions that are not related to the GRIT when we describe Algorithms (1,2). In Algorithm 1, note that in the example-level shuffling (*Phase 2*), all three queues containing a queue for image features, a queue for text features, and a queue for indices are shuffled in the same order to keep image-text pairs and their corresponding indices unmixed. Likewise, the split procedure (*Phase 2*) applies equally to all three queues. As described in Algorithm 2, in grouping phase (*Phase 3*), we exclude previously visited indices to avoid including duplicate examples. After grouping phases are done for all the small sub-queues split from original queue, we make the original queue empty. Thus, each example in the dataset goes through the grouping phase only once per epoch.

2.3 First training epoch with GRIT

Since GRIT-VLP obtains the indices for the grouped mini-batches from the previous epoch, randomized indices are given to the model at the first training epoch, as expected. But, if we assume another pre-trained network for calculating similarity is available, we can generate indices considering the grouped mini-batches. Then, we can use indices for the grouped mini-batches at the first epoch of the training. In our experiments, all GRIT-variant models use the generated indices from $\text{ALBEF}_{\text{Base}_{50}}$ (pre-trained for one single epoch) at the first training epoch. Note that the effect of using this simple indices generation procedure at the first epoch is marginal in our experimental setting. But, we believe that the impact of this simple trick can be large when the training epoch is set *very* small (*e.g.*, one or two epoch).

3 Details of Downstream Tasks

As described in Section 5 (manuscript), we mainly follow the implementation details of ALBEF to fine-tune the pre-trained model. Unlike pre-training, we use randomly cropped image of resolution 384×384 in fine-tuning, and resize the images without cropping in inference. The same RandAugment, optimizer, cosine learning rate decay, and weight decay are applied to all downstream tasks. However, unlike ALBEF, since we do not use a momentum encoder in the pre-training phase, the momentum distillation (MD) is not utilized in all downstream tasks except for the Table 6 (manuscript) to show the model-agnostic property

Algorithm 1 Entire process: Pseudocode, Pytorch-like

```
# L_Q_v, L_Q_t - queues size of (L × d) for saving feature representations
# L_Q_idx - queues size of L for saving unique indices
# f_v, f_t - image encoder and text encoder
# g_v, g_t - linear embbedings for image features and text features
# I - index queue size of M
# G - output index array size of D; (D = #(total examples))

# example_shuffle(L_Q_v, L_Q_t, L_Q_idx) - a function that shuffles across the examples
# divide(L_Q_v, L_Q_t, L_Q_idx) - a function for dividing each input queues
# grouping(s_Q_v, s_Q_t, s_Q_idx) - a function for grouping (described in Algorithm 2)
# cal_itc_cons_loss(v_feats, t_feats) - a function for calculating ITC and consistency loss
# mini_batch_shuffle(G) - a function that shuffles across the mini-batches

for e in epoch:
    loader ← initialize(G) # Initialize indices of loader with idxSet_e
    for batch_i, (idx,V,T) in enumerate(loader): # idx denotes the unique index of example
        # ITC and contrastive loss
        # Forwarding for uni-modal encoders
        v_embeds = f_v(V)
        t_embeds = f_t(T)

        # Calculate [CLS] tokens for both image and text
        v_feats = l2_normalize(g_v(v_embeds[0]))  # dim(v_feats)= N × d
        t_feats = l2_normalize(g_t(t_embeds[0]))  # dim(t_feats)= N × d
        loss_itc_cons = cal_itc_cons_loss(v_feats, t_feats) # using (6) in manuscript

        with torch.no_grad():
            # Phase 1: Collection phase
            L_Q_v.enqueue(v_feats)
            L_Q_t.enqueue(t_feats)
            L_Q_idx.enqueue(idx)

            if is_full(L_Q_idx):
                # L features (indices) stored in queues
                # Phase 2: Example-level shuffle
                example_shuffle(L_Q_v, L_Q_t, L_Q_idx)
                # Divide queues
                div_Q_v, div_Q_t, div_Q_idx = divide(L_Q_v, L_Q_t, L_Q_idx)

                # Phase 3: Grouping phase (Repeat for (L//M) times)
                for s_Q_v, s_Q_t, s_Q_idx in zip(div_Q_v, div_Q_t, div_Q_idx):
                    # M features (indices) stored in small queues
                    I = grouping(s_Q_v, s_Q_t, s_Q_idx)
                    G.append(I)

                # Make all three queues empty
                clear(L_Q_v, L_Q_t, L_Q_idx)

        # Calculate ITM and MLM loss (itm_loss, mlm_loss)
        # We omit the detailed procedures of ITM and MLM for simplicity
        # Note both losses require entire sequences of representations (v_embeds, t_embeds)

        loss=loss_itc_cons+loss_itm+loss_mlm
        loss.backward()

    # Phase 4: Mini-batch level shuffle
    mini_batch_shuffle(G)
```

Algorithm 2 *Phase 3:* Grouping phase: Pseudocode, Pytorch-like

```
# s_Q_v, s_Q_t - queues size of (M × d) for saving feature representations
# s_Q_idx - queues size of M for saving unique indices
# I - output index queue size of M
# cur_index, pre_index - index of current step, index of previous step

# exclude_index(matrix,index) - a function that sets both row and column of matrix
corresponding to the index as 0

def grouping (s_Q_v, s_Q_t, s_Q_idx):

    # Calculate contrastive similarity
    similarity = s_Q_v @ s_Q_t  #  dim(similarity)= M × M
    P_v2t = softmax(similarity, dim=1)  #  dim(P_v2t)= M × M
    P_t2v = softmax(similarity.t(), dim=1)  #  dim(P_t2v)= M × M

    # Randomly sample one example
    M = len(s_Q_idx)
    pre_index = randint(0,M-1)
    I.enqueue(s_Q_idx[pre_index])
    use_v2t = True  # For simplicity, we started with (image → text) direction

    # Iteratively find index
    for i in range(M-1):
        # Use P_v2t, P_t2v alternatively
        if (use_v2t):
            cur_index=argmax(P_v2t[pre_index]) # dim(P_v2t[pre_index])= M
        else:
            cur_index=argmax(P_t2v[pre_index]) # dim(P_t2v[pre_index])= M

         # Exclude pre_index for both P_v2t and P_t2v whether use_v2t is True or not.
        exclude_index(P_v2t,pre_index)
        exclude_index(P_t2v,pre_index)

        I.enqueue(s_Q_idx[cur_index])
        pre_index = cur_index
        use_v2t = not use_v2t

    return I
```

of our method. The total mini-batch size in S.M refers to the overall mini-batch size. Namely, it denotes "number of GPUs × mini-batch size per GPU" ($4 \times N$).

[Image-Text Retrieval (IRTR)] IRTR finds the most similar text to a given image in a set of texts, or vice versa. In IRTR, with an updated momentum encoder during the pre-training phase, ALBEF [3] fine-tunes the pre-trained model using queue-based ITC, MD for ITC, and ITM_{hard}. To fairly evaluate the effectiveness of our method on the pre-training only, we mostly follow the fine-tuning phase of ALBEF by adopting the queue-based ITC (queue size: 65280) and ITM_{hard} (but, not adopting the MD for ITC). Since we do not have an updated momentum encoder during the pre-training, we use the initial model (initialized with original BERT-base and ViT-B/16 pre-trained on ImageNet-1k) for initializing the momentum encoder. As another option, the momentum encoder can also be initialized with weights from the pre-trained model. We empirically verify that the performance of these two variants is almost similar. Note that we use the momentum encoder and the additional queue only for constructing negative sets of queue-based ITC objective, not for the momentum distillation. Although our method (GRIT and consistency loss) can also be incorporated into

fine-tuning step for IRTR, we do not include them for a fair comparison. Unless otherwise noted, this fine-tuning setting for IRTR is applied in all experiments.

The train/validation/test set consists of 113k/5k/5k and 29k/1k/1k for COCO and F30K, respectively. In fine-tuning, we set the total batch size as 256 and the initial learning rate as $1e-5$ for both datasets, and fine-tune for 5 epochs for COCO, and 10 epochs for F30K. In evaluation, we first get the top-k candidates based on image-text contrastive similarity. Then, we re-rank these calculated top-k candidates using ITM scores. k is set as 256 for COCO and 128 for F30K.

[Visual Reasoning (NLVR2)] NLVR classifies whether a textual description is true based on two images. The multi-modal encoder is consecutively duplicated to infer two images like [3]. Since the model architecture has changed, one more pre-training is performed. Then, the pre-trained model is fine-tuned and evaluated on the NLVR2 dataset.

We use the original train/val/test split of NLVR2[5] for evaluating visual reasoning. Since NLVR tries to classify whether a textual description is true based on two images, the multi-modal encoder is duplicated to consider two images. As we mentioned above, since the overall structure of the model is changed, one more pre-training is performed with text-assignment task like ALBEF [3]. Text-assignment task assigns the text to one of three choices: the first image, the second image, or none of them (three-way classification task). For an additional 1 epoch NLVR pre-training step, we use images of size 256×256 on the 4M dataset with a total batch size of 256 and a learning rate of $2e-5$. After that, we fine-tune for 10 epochs with a total batch size of 64 and an initial learning rate of $2e-5$. We measure the performance on dev and test-P splits.

[Visual Question Answering (VQA)] VQA aims to derive an answer given an image and a relevant question. Following [3], an auto-regressive transformer decoder is added to generate answers. The decoder is initialized with weights of the pre-trained multi-modal encoder and fine-tuned with conditional language modeling loss. For VQA, we conduct experiment on the VQA2.0 dataset [1], where train/val/test split set is composed of 83k/41k/81k. Both training and validation sets are used for training, and also include additional question-answer pairs from Visual Genome, following previous works [6,3]. We fine-tune for 8 epochs with a total batch size of 128 and an initial learning rate of $2e-5$. During inference, the decoder is constrained to only generate from the 3192 candidate answers [2] for a fair comparison. We measure performance on the test-dev (t-dev) and test-std (t-std) splits.

4 Detailed explanations on Section 5.5 (manuscript)

4.1 Small model, ITC

Note that all the models in Table 5 (manustript) are trained with dual uni-modal encoders. Thus, other losses like MLM and ITM can not be used. For the Queue-based ITC, queue size for storing extra negatives is set as 65280. All other models

are trained with *in-batch* ITC. We verified that adding ITC_{cons} and GRIT has considerable gain. We believe these results demonstrate the effectiveness of our method.

4.2 Large model, more objectives

We also further measure the gains when another type of VLP model, TCL [7], is combined with our method. TCL extends ALBEF with two complementary objectives in the pre-alignment step. They utilize the momentum encoder to design a informative pseudo-target of loss functions in the pre-training. Thus, TCL has a larger network architecture than ours (which do not contain momentum encoders in the pre-training) and is pre-trained with additional objectives. Despite these differences, we verify that the combination of our method and TCL again brings significant gains in Table 6 (manuscript).

For a fair comparison, we mostly follow the pre-training and fine-tuning settings of TCL. Namely, we maintain the original architecture and objectives of TCL, and then additionally apply our grouped mini-batch sampling strategy and enlarged masking probability. Note that we do not employ the consistency loss, since TCL already has objectives for elaborating contrastive learning. Since TCL uses a momentum encoder and distillation in the pre-training, for fine-tuning, we use an updated momentum encoder in the pre-training phase and MD in this experiment.

5 Detailed Experimental Results

Here we report additional results about analysis on sampling for VLP in Section 3 (manuscript). We include VQA results in both Table 3 and Table 4. Moreover, we report models trained with various masking probabilities $(15\%, 35\%, 50\%, 75\%)$ in Table 4.

Table 3. Ablation study on pre-training objectives of ALBEF on two V+L downstream tasks. MLM: masked language modeling with 15% masking probability.

Epochs	Training tasks	TR (COCO)			IR			NLVR		VQA	
		R@1	R@5	R@10	R@1	R@5	R@10	(dev)	(test-P)	(t-dev)	(t-std)
10	MLM + ITM_{rand}	61.6	86.1	92.5	47.8	75.4	84.8	77.02	78.44	72.82	73.11
	MLM + ITM_{rand} + ITC	66.8	88.8	94.5	51.1	78.4	86.8	76.59	78.69	73.24	73.44
	MLM + ITM_{hard}	68.6	89.4	94.9	52.1	79.0	87.1	79.18	79.32	73.55	73.67
	$\text{ALBEF}_{\text{Base}}$	**72.3**	**91.3**	**96.0**	**55.1**	**81.0**	**88.5**	**79.21**	**79.78**	**73.97**	**74.10**
20	MLM + ITM_{rand}	66.5	88.3	94.0	51.3	78.3	86.5	78.02	79.43	73.59	73.80
	MLM + ITM_{rand} + ITC	69.6	90.9	95.3	53.8	80.0	87.8	77.61	79.43	73.68	73.98
	MLM + ITM_{hard}	72.0	91.5	**96.6**	57.5	81.2	88.4	**80.44**	**80.83**	74.19	74.43
	$\text{ALBEF}_{\text{Base}}$	**73.8**	**92.3**	96.5	**57.7**	**82.5**	**89.6**	79.22	80.37	**74.62**	**74.70**

5.1 Analysis on hard negative sampling

Table 3 shows three downstream task performances (including VQA) of models with and without each objective of $\text{ALBEF}_{\text{Base}}$. As mentioned in Section 3 (manuscript), the bottom two rows of each epoch in the Table 3 reaffirm that the ITM_{hard} is the most essential component for achieving great performance quickly in all downstream tasks including VQA.

5.2 Analysis on masking probabilities

Table 4. Evaluation of the various masking probability methods on two downstream V+L tasks. Base_n: model with n% masking probability.

Epochs	Training tasks	TR (COCO)			IR			NLVR		VQA	
		R@1	R@5	R@10	R@1	R@5	R@10	(dev)	(test-P)	(t-dev)	(t-std)
10	$\text{ALBEF}_{\text{Base}}$	72.3	91.3	96.0	55.1	81.0	88.5	79.21	79.78	73.97	74.10
	$\text{ALBEF}_{\text{Base}_{35}}$	73.1	91.6	96.2	56.7	81.9	89.2	**79.42**	79.78	**74.42**	74.42
	$\text{ALBEF}_{\text{Base}_{50}}$	**73.4**	**92.5**	**96.4**	**57.2**	**82.3**	**89.4**	**79.42**	**79.87**	**74.42**	**74.61**
	$\text{ALBEF}_{\text{Base}_{75}}$	72.6	91.5	96.1	56.3	81.4	89.0	79.24	79.33	74.38	74.42
20	$\text{ALBEF}_{\text{Base}}$	73.8	92.3	96.5	57.7	82.5	89.6	79.22	80.37	74.62	74.70
	$\text{ALBEF}_{\text{Base}_{35}}$	75.2	93.1	96.6	58.7	82.9	89.8	**80.56**	80.47	74.79	75.00
	$\text{ALBEF}_{\text{Base}_{50}}$	**75.6**	**93.2**	**96.7**	**58.8**	**83.2**	**90.1**	80.41	**80.54**	**74.94**	**75.07**
	$\text{ALBEF}_{\text{Base}_{75}}$	75.1	92.9	96.8	58.1	82.7	89.6	79.30	79.69	74.86	74.85

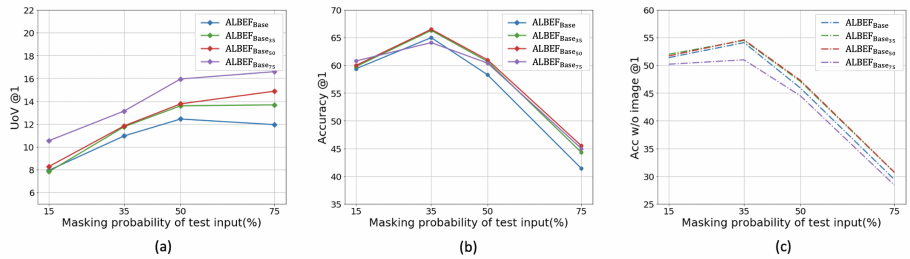

Fig. 1. Results of the four pre-trained models: (a) UoV (*Usage of Vision*), (b) Accuracy and (c) Accuracy w/o image.

Table 4 shows the results of models trained with various masking probabilities (15%, 35%, 50%, 75%) on three downstream tasks including VQA. We verify that moderately enlarging the masking probability (50%) enhances the final performance with a considerable gain. However, in Table 4, we observe that if the masking probability is too high (75%), the performance is rather degraded. In this section, we scrutinize the reasons behind theses results with the MLM accuracy and UoV (*Usage of Vision*).

From the Fig. 1(a), we observe that the UoV of the model trained with 75% masking probability is significantly higher than that of other models. It means that the $\text{ALBEF}_{\text{Base}_{75}}$ is the model that most reflects image information for prediction. However, as shown in the Table 4, the final performance of $\text{ALBEF}_{\text{Base}_{75}}$ is lower than the $\text{ALBEF}_{\text{Base}_{50}}$. This result suggests that simply continuously increasing the usage of image information does not always lead to final performance improvement. To properly explain about this somewhat counter-intuitive result, it is necessary to consider the *Accuracy w/o image* metric which indicates the MLM accuracy based only on the text information without image (Fig. 1(c)), as well as the UoV metric (Fig. 1(a)).

Since the *Accuracy w/o image* measures the MLM accuracy by using only text (without image) as input to the pre-trained model, it shows the ability of the pre-trained model to derive the correct answer for the masked word from the only textual information. In Fig. 1(a) and Fig. 1(c), we observe that the model pre-trained with sparse sentence (75%) highly use the visual information, but its *Accuracy w/o image* is relatively low compared to other models, which means that $\text{ALBEF}_{\text{Base}_{75}}$ lacks the ability to leverage textual information. We can observe more clearly that $\text{ALBEF}_{\text{Base}_{75}}$ lacks the ability to utilize textual information in the NLVR2 task, which requires complex reasoning over sentences. As shown in the Table 4, the results of the $\text{ALBEF}_{\text{Base}_{75}}$ on NLVR2 is even lower than the $\text{ALBEF}_{\text{Base}}$. In conclusion, we verify that the moderately enlarged masking probability (50%), which can utilize both visual and textual information in a balanced way, is more suitable for vision and language pre-training.

Table 5. Ablation studies on mini-batch size and search space

		TR (COCO)			IR			NLVR	
N	M	R@1	R@5	R@10	R@1	R@5	R@10	(dev)	(test-P)
48	480	76.4	93.5	96.7	59.6	83.4	**90.1**	80.28	80.50
48	960	76.9	**94.0**	**97.1**	59.7	**83.6**	**90.1**	81.23	81.07
48	1920	76.8	93.7	96.9	**60.0**	83.3	89.9	81.29	81.34
96	480	76.5	93.9	97.0	59.4	83.4	90.0	80.68	80.68
96	960	**77.1**	93.8	97.0	59.5	83.4	90.0	**81.30**	81.43
96	1920	**77.1**	93.5	96.9	59.6	83.5	90.0	80.53	81.01
128	480	76.3	93.7	96.9	59.1	83.1	89.7	80.73	80.88
128	960	77.0	93.5	**97.1**	59.6	83.3	90.0	80.54	81.47
128	1920	**77.1**	93.6	96.7	59.5	83.3	89.9	80.73	**81.60**

5.3 Ablation study on hyper-parameters

Regarding the batch size (N) and search space size (M), we carry out an additional ablation study in Table 5 (with other hyper-parameters fixed). We experiment

with three mini-batch sizes (48, 96, 128) and search space sizes $(480, 960, 1920)$. In all search space sizes, we verify that our method is generally robust with batch size. However, if the search space is small (480), the performance is degraded compared to the other search space size. If the search space is sufficiently large (more than 960), the performance gap is marginal. Namely, N does not have a crucial impact on performance but M does. In Table 3 (manuscript), we select $N = 128$ and $M = 1920$ since the total batch size ($128 \times 4 = 512$) is same as ALBEF ($64 \times 8 = 512$) and it shows consistently great performance on all downstream tasks.

Table 6. Fine-tuned results image-text retrieval on Flickr30K and MSCOCO datasets

Method	#Pre-train Images	MSCOCO (5K test set)						Flickr30K (1K test set)					
		TR			IR			TR			IR		
		R@1	R@5	R@10	R@1	R@5	R@10	R@1	R@5	R@10	R@1	R@5	R@10
UNITER	4M	65.7	88.6	93.8	52.9	79.9	88.0	87.3	98.0	99.2	75.6	94.1	96.8
VILLA	4M	-	-	-	-	-	-	87.9	97.5	98.8	76.3	94.2	96.8
OSCAR	4M	70.0	91.1	95.5	54.0	80.8	88.5	-	-	-	-	-	-
ALBEF	4M	73.1	91.4	96.0	56.8	81.5	89.2	94.3	99.4	99.8	82.8	**96.7**	**98.4**
GRIT-VLP$_{\text{E-10}}$	4M	74.9	93.0	**97.0**	58.1	82.7	89.6	94.7	**99.6**	**99.9**	82.0	95.3	97.7
GRIT-VLP	4M	**77.1**	**93.6**	96.7	**59.5**	**83.3**	**89.9**	**96.0**	**99.6**	**99.9**	**83.8**	96.2	97.8
ALBEF	14M	77.6	94.3	97.2	60.7	84.3	90.5	95.9	99.8	100.0	85.6	97.5	98.9
ALIGN	1.8B	77.0	93.5	96.9	59.9	83.3	89.8	95.3	99.8	100.0	84.9	97.4	98.6

5.4 Additional results on IRTR

Table 6 shows the full IRTR results including R@5 and R@10 accuracy on both COCO and Flickr30K dataset. As we mentioned in the Section 5 (manuscript), GRIT-VLP (4M) outperforms other methods including ALBEF by a considerable gain, except for the IR results (R@5 and R@10) that are slightly lower than ALBEF in the Flickr30K dataset. In particular, in the COCO dataset, we verify that our GRIT-VLP (4M) shows competitive results with ALBEF (14M) and ALIGN (1.8B) trained on much larger datasets. Moreover, when using the exact same dataset (4M), we observe that "GRIT-VLP$_{\text{E-10}}$" (4M) trained with only 10 epochs shows superior performance to the ALBEF (4M) trained with 30 epochs, in the COCO dataset. Note that our GRIT-VLP also obtains faster training time per epoch and smaller model parameters in the pre-training compared to ALBEF.

References

1. Goyal, Y., Khot, T., Summers-Stay, D., Batra, D., Parikh, D.: Making the v in vqa matter: Elevating the role of image understanding in visual question answering. In: CVPR (2017)
2. Kim, J.H., Jun, J., Zhang, B.T.: Bilinear attention networks. In: NeurIPS (2018)
3. Li, J., Selvaraju, R., Gotmare, A., Joty, S., Xiong, C., Hoi, S.C.H.: Align before fuse: Vision and language representation learning with momentum distillation. In: NeurIPS (2021)
4. Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., Lerer, A.: Automatic differentiation in pytorch. In: NeurIPS Workshops (2017)
5. Suhr, A., Zhou, S., Zhang, A., Zhang, I., Bai, H., Artzi, Y.: A corpus for reasoning about natural language grounded in photographs. arXiv preprint arXiv:1811.00491 (2018)
6. Tan, H., Bansal, M.: Lxmert: Learning cross-modality encoder representations from transformers. arXiv preprint arXiv:1908.07490 (2019)
7. Yang, J., Duan, J., Tran, S., Xu, Y., Chanda, S., Chen, L., Zeng, B., Chilimbi, T., Huang, J.: Vision-language pre-training with triple contrastive learning. In: CVPR (2022)