Aligning Information Capacity Between Vision and Language via Dense-to-Sparse Feature Distillation for Image-Text Matching

Supplementary Material

Dataset	Images	Texts	Tokens per Text
Flickr30K original	29000	145000	13.41
Flickr30K generated	29000	29000	103.24
MS-COCO original	113287	566435	11.30
MS-COCO generated	113287	113287	104.34

Table 6. Comparison between the generated data with the original datasets.

6. Additional Implementation Details

Details of Dense Text Generation. For dense text generation, we use LLaVa [23] to generate dense textual descriptions. The hyperparameters for LLaVa are set as follows: Top-P $=$ 0.9, temperature$=$ 0.2, and max-new-tokens$=$ 500. We generate a dense text description for each image in the MS-COCO and Flickr30K datasets, with each dense text sentence corresponding to $\frac{1}{5}$ of the sparse text descriptions. As shown in Table 6, we compare the generated image-dense text dataset with the original dataset. **Note that, we only generate the dense texts for training dataset.**

Model settings. The dimensionality d of the joint embedding space is 512 for all experiments. For the image encoder, we utilize different backbones to extract image features and employ a feature aggregator to obtain a holistic embedding. Specifically, we adopt GPO [3] as the feature aggregator. Since different backbones produce features with varying dimensionalities, we introduce an MLP before GPO to standardize the feature dimensions. Similarly, for the text encoder, we use different backbones along with GPO for feature aggregation. In addition, we follow [3] by using a higher image resolution.

For the text decoder, we adopt a Transformer-based architecture with a depth of 4 and 4 attention heads per layer. Additionally, we introduce 100 learnable tokens to predict the context of sparse text embeddings. Notably, the image encoder and text encoder maintain the same architecture across both the pre-training and fine-tuning stages.

Training details. For dense text pretraining, the VSE model is optimized using AdamW with a weight decay factor of 1e-4 The batch size is set to 96 for a Swin-based backbone and 128 for other architectures. The margin α of the triplet loss is 0.2. The initial learning rate is set to 0.0005 for the first few epochs and is progressively reduced by a factor of 10 at the 9th, 15th, 20th, and 25th epochs. Additionally, a warm-up strategy is applied at the beginning of training: during the first epoch, the triplet loss considers all negative examples in the batch instead of selecting only the hardest negative example. For sparse text fine-tuning, we initialize the image and text encoders with weights from the pretraining phase. The objective of fine-tuning is to reconstruct the dense text embedding from the sparse text embedding. The training strategy remains consistent with the pretraining phase.

The pseudo-code for the fine-tuning phase is presented in Algorithm 1.

Algorithm 1: PyTorch Pseudocode of Finetune Stage

```
# n: batch_size
# d: dimensionality of the embeddings
# m: number of the words in sentences
# i_e: image embedding
# st_e: sparse text embedding
# st_w: sparse text word embedding
# dt_e: dense text embedding

# compute embeddings
i_e = image_encoder(I) #[n, d]
st_e, st_w  = sparse_text_encoder(T) # [n
    , d] [n,m,d]
st_e = decoder(st_w) + st_e

with torch.no_grad():
    dt_e, _ = dense_text_encoder(T) # [n,
        d]

# normalize
i_e_norm = l2_normalize(i_e, dim=1) # [n,
    d]
st_e_norm = l2_normalize(st_e, dim=1) # [
    n,d]
dt_e_norm = l2_normalize(dt_e, dim=1) # [
    n,d]

# contrastive learning
sims = st_e_norm.mm(i_e_norm.t())
loss_align = triplet(sims)

# d2s distill
loss_distill = distillation_loss(
    dt_e_norm, st_e_norm)

loss = loss_align + loss_distill
```