RetrieveGAN: Image Synthesis via Differentiable Patch Retrieval Supplementary Materials

Hung-Yu Tseng*[2], Hsin-Ying Lee*[2], Lu Jiang[1], Ming-Hsuan Yang[1,2,3], Weilong Yang[1]

[1]Google Research [2]University of California, Merced [3]Yonsei University

1 Overview

In this supplementary material, we first show the critical steps in the weighted reservoir sampling approach. We then present the implementation details of the proposed framework. Finally, we supplement the training and evaluation details.

2 Weighted Reservoir Sampling

Our method is inspired by the weighted reservoir sampling approach in [8]. The classical reservoir sampling [7] is designed to sample k items from a collection of n items [7]. [8] shows this classical subset sampling procedure can be relaxed to a differentiable process by introducing 1) the Gumbel-max trick and 2) the relaxed top-k function (Iterative Softmax). In Algorithm 1, we list the key steps using the notation defined in the main paper. IterSoftmax is a relaxed top-k function that iteratively applies the Softmax operation to obtain the selection variable s.

Algorithm 1: Subset Selection using Weighted Reservoir Sampling.

Input : Patch bank M; subset size n; weights $\pi = [\pi_1, \ldots, \pi_{|M|}]$

Output: relaxed n-hot vector s where $\sum_{i=1}^{|M|} s_i = n$ and $0 \leq s_i \leq 1$.

Initialize $\hat{\pi}$ as a zero vector of length $|M|$;
for $i = 1, \ldots, |M|$ **do**
    $u_i \leftarrow \text{Uniform}(0, 1)$;
    $\hat{\pi}_i \leftarrow -\log(-\log(u_i)) + \log(\pi_i)$;
end
$s \leftarrow \text{IterSoftmax}(\hat{\pi}, n)$;
return the relaxed n-hot vector s

Our method is conceptually similar to Algorithm 1. But to make it work in our problem setting, we discuss two modifications in the main paper, including

* Equal contribution. Work done during their internships at Google Research.

Table 1: **Statistics of datasets.**

Dataset	Train	Val	Test	# Category	# Patches
COCO-Stuff	74121	1024	2048	171	411682
Visual Genome	62565	5506	5088	178	606319

the group-wise sampling and the greedy selection strategy. The greedy selection strategy requires our model to update the query iteratively. Since the query is used to compute the weight $\hat{\pi}$, our algorithm presented in the main paper merges the iterative softmax with the Gumbel-max sampling step in Algorithm 1.

3 Implementation Details

Scene graph encoder. We adopt the graph convolutional layers in sg2im [3] for processing the input scene graph. Specifically, given an edge $e = (o_j, r_k, o_t)$ in the scene graph, we compute the corresponding output vectors o'_j, r'_k, and o'_t via a fully-connected layer. The vectors o_j and o_t represent the object, while r_k indicates the relation between o_j and o_t. However, since a single object o_j may appear in multiple edges (i.e., participate in many relationships), we use the average pooling to fuse all the output vectors o'_j computed from all the edges involving o_j. We show an example of the single graph convolutional layer in Figure 1. In practice, we use 5 layers for our scene graph encoder E.

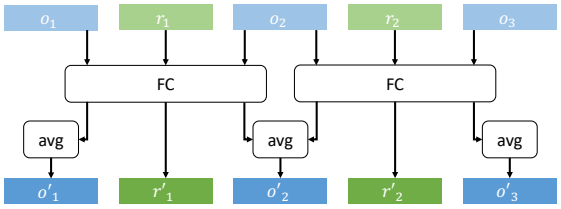

Fig. 1: **Example of single graph convolutional layer.** Given an edge $e = (o_j, r_k, o_t)$, we first use a fully-connected layer to compute the output o'_j, r'_k, o'_t. We then apply the average pooling to fuse all the output vectors o'_j computed from all the edges involving o_j (e.g., o_2 in this example).

Patch embedding function. The patch embedding function aims to compute the embedding of the candidate patches for the retrieval process. We first use the pre-trained ResNet model [2] to extract the ImageNet features of all the patches in the patch memory bank. We operate this process offline. Then our patch embedding function is a series of fully-connected layers that maps the ImageNet feature space to the patch embedding space.

Fig. 2: **Additional qualitative results.** For each sample, we show the retrieved patches which are used to guide the following image generation process. We also show the original image of each selected patch for clearer visualization.

Image Generation We use four modules to generate an image from a set of selected patches: crop encoder, object^2 refiner, object-image fuser, and decoder, described as follows:

Crop encoder. The crop encoder extracts crop features $\{c_i\}$ from the selected patches. We adopt a series of convolutional and down-sampling layers to build the crop encoder.

Object^2 refiner. The object^2 refiner aims to associate the crop features with the relationships $\{r_k\}$ defined in the input scene graph. Specifically, we replace the objects $\{o_i\}$ with the crop features $\{c_i\}$ in the edges defined in the input scene graph. Similarly to the scene graph encoder, we use the graph convolutional layer to compute the output vectors (c'_j, r'_k, c'_t) given the input edge (c_j, r_k, c_t).

The average pooling function is then applied to combine the output vectors c'_j computed from all the edges containing c_j. Unlike the scene graph encoder, we use the 2D convolutional layer to build the graph convolutional layer since the input crop feature is of the dimension $D_c \times h \times w$. We use 2 layers in practice for the object[2] refiner.

Object-image fuser. Given the refined object and predicate features, we use an object-image fuser to encode all features into a latent canvas L. For each object, we first concatenate its refined crop features c'_i and the original object feature o_i. We then expand the concatenated feature to the shape of the corresponding predicted bounding box to get u_i with dimension $D \times W \times H$. Then we measure the attention map of each object by

$$a_i = \frac{\exp(s_i)}{\sum_{j=i}^{N} \exp(s_j)}, \tag{1}$$

where $s_i = f(u_i)h(r_{p_i})$, f and h are learned mapping function, and r_{p_i} is the relation feature for the relationship between the object and the image. We can then obtain the final attention maps by summing up all object attention maps: $a = \sum_{i=1}^{N} a_i g(u_i)$, where g is a learned mapping function. Finally, we aggregate the attention maps to form the scene canvas with the 'image' object features:

$$L = \lambda_a a + u_{img}, \tag{2}$$

where λ_a is the weight.

Decoder. We use a series of convolutional and up-sampling layers to synthesize the final image from the scene canvas created by the object-image fuser.

4 Training and Evaluation

Training details. We implement with PyTorch [6] and train our model with 90 epochs on both the COCO-stuff [1] and visual genome [5] datasets. We use the Adam optimizer [4] with a batch size of 16. The learning rates for the generator and discriminator are respectively 0.00025 and 0.001, and the exponential decay rates (β_1, β_2) are set to be $(0, 0.9)$. We set the hyper-parameters as follows: $\lambda_{\text{gt}}^{\text{sel}} = 0.1$, $\lambda_{\text{occur}}^{\text{sel}} = 0.001$, $\lambda_{\text{adv}}^{\text{img}} = 0.01$, $\lambda_{\text{recon}}^{\text{img}} = 1$, $\lambda_{\text{p}}^{\text{img}} = 1$, $\lambda_{\text{adv}}^{\text{obj}} = 0.01$, $\lambda_{\text{ac}}^{\text{obj}} = 0.1$, $\lambda_{\text{p}}^{\text{obj}} = 0.5$, and $\lambda_{\text{bbx}} L_{\text{bbx}} = 10$.

Dataset summary. We present the summary of the datasets used for the training and evaluation in Table 1.

5 Additional Results

We show more qualitative results of our approach in Figure 2.

References

1. Caesar, H., Uijlings, J., Ferrari, V.: Coco-stuff: Thing and stuff classes in context. In: CVPR (2018) 4
2. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: CVPR (2016) 2
3. Johnson, J., Gupta, A., Fei-Fei, L.: Image generation from scene graphs. In: CVPR (2018) 2
4. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: ICLR (2015) 4
5. 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. IJCV **123**(1), 32–73 (2017) 4
6. 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 workshop (2017) 4
7. Vitter, J.S.: Random sampling with a reservoir. ACM Transactions on Mathematical Software (TOMS) **11**(1), 37–57 (1985) 1
8. Xie, S.M., Ermon, S.: Reparameterizable subset sampling via continuous relaxations. In: IJCAI (2019) 1