Text2LIVE: Text-Driven Layered Image and Video Editing Supplementary Material (SM)

ECCV 2022 Submission ID:3228

We provide implementation details for our architecture and training regime.

1 Generator Network Architecture

We base our generator G_θ network on the `U-Net` architecture [6], with a 7-layer encoder and a symmetrical decoder. All layers comprise 3×3 Convolutional layers, followed by `BatchNorm`, and `LeakyReLU` activation. The intermediate channels dimensions is 128. In each level of the encoder, we add an additional 1×1 Convolutional layer and concatenate the output features to the corresponding level of the decoder. Lastly, we add a 1×1 Convolutional layer followed by `Sigmoid` activation to get the final RGB output.

2 Internal Dataset (Sec. 3.1)

We apply data augmentations to the source image and target text (I_s, T) to create multiple *internal examples* $\{(I_s^i, T^i)\}_{i=1}^N$. Specifically, at each training step, we apply a random set of image augmentations to I_s, and augment T using a pre-defined set of text templates, as follows:

2.1 Image augmentations

- Random spatial crops: 0.85 and 0.95 of the image size in our image and video frameworks respectively.
- Random scaling: aspect ratio preserved scaling, of both spatial dimensions by a random factor, sampled uniformly from the range $[0.8, 1.2]$.
- Random horizontal-flipping is applied with probability p=0.5.
- Random color jittering: we jitter the global brightness, contrast, saturation and hue of the image.

2.2 Text augmentations and the target text prompt T

We compose T with a random text template, sampled from of a pre-defined list of 14 templates. We designed our text-templates that does not change the semantics of the prompt, yet provide variability in the resulting CLIP embedding e.g.:

- ”photo of {}.”
- ”high quality photo of {}.”
- ”a photo of {}.”
- ”the photo of {}.”
- ”image of {}.”
- ”an image of {}.”
- ”high quality image of {}.”
- ”a high quality image of {}.”
- ”the {}.”

- ”a {}.”
- ”{}.”
- ”{}”
- ”{}!”
- ”{}...”

At each step, one of the above templates is chosen at random and the target text prompt T is plugged in to it and forms our augmented text. By default, our framework uses a single text prompt T, but can also support multiple input text prompts describing the same edit, which effectively serve as additional text augmentations (e.g., “crochet swan”, and “knitted swan” can both be used to describe the same edit).

3 Training Details

We implement our framework in PyTorch [4] (code will be made available). As described in Sec. 3, we leverage a pre-trained CLIP model [5] to establish our losses. We use the ViT-B/32 pretrained model (12 layers, 32x32 patches), downloaded from the official implementation at GitHub. We optimize our full objective (Eq. 2, Sec. 3.1), with relative weights: $\lambda_g = 1$, $\lambda_s = 2$ (3 for videos), $\lambda_r = 5 \cdot 10^{-2}$, ($5 \cdot 10^{-4}$ for videos) and $\gamma = 2$. For bootstrapping, we set the relative weight to be 10, and for the image framework we anneal it linearly throughout the training. We use the MADGRAD optimizer [2] with an initial learning rate of $2.5 \cdot 10^{-3}$, weight decay of 0.01 and momentum 0.9. We decay the learning rate with an exponential learning rate scheduler with gamma $= 0.99$ (gamma $= 0.999$ for videos), limiting the learning rate to be no less than 10^{-5}. Each batch contains (I_s^i, T^i) (see Sec. 3.1), the augmented source image and target text respectively. Every 75 iterations, we add $\{I_s, T\}$ to the batch (i.e., do not apply augmentations). The output of G_θ is then resized down to 224[px] maintaining aspect ratio and augmented (e.g., geometrical augmentations) before extracting CLIP features for establishing the losses. We enable feeding to CLIP arbitrary resolution images (i.e., non-square images) by interpolating the position embeddings (to match the size of spatial tokens of a the given image) using bicubic interpolation, similarly to [1].

Training on an input image of size 512×512 takes ∼9 minutes to train on a single GPU (NVIDIA RTX 6000) for a total of 1000 iterations. Training on one video layer (foreground/background) of 70 frames with resolution 432×768 takes ∼60 minutes on a single GPU (NVIDIA RTX 8000) for a total of 3000 iterations.

3.1 Video Framework

We further elaborate on the framework’s details described in Sec. 3.2 of the paper.

Atlas Pre-processing. Our framework works on a discretized atlas, which we obtain by rendering the atlas to a resolution of 2000×2000 px. This is done as in [3], by querying the pre-trained atlas network in uniformly sampled UV locations. The neural atlas representation is defined within the [-1,1] continuous space, yet the video content may not occupy the entire space. To focus only on the used atlas regions, we crop the atlas prior to training, by mapping all video locations to the atlas and taking their bounding box. Note that for foreground atlas, we map only the foreground pixels in each frame, i.e., pixels for which the foreground opacity is above 0.95; the foreground/background opacity is estimated by the pre-trained neural atlas representation.

Training. As discussed in Sec. 3.2 in the paper, our generator is trained on atlas crops, yet our losses are applied to the resulting edited frames. In each iteration, we crop the atlas by first sampling a video segment of 3 frames and mapping it to the atlas. Formally, we sample a random frame t and a random spatial crop size (W, H) where its top left coordinate is at (x, y). As a result we get a set of cropped (spatially and temporally) video locations:

$$\mathcal{V} = \{p = (x + j, y + i, t + m) \qquad \text{s.t.} \qquad 0 \leq j < W, \quad 0 \leq i < H, \quad m \in \{-k, 0, k\}\} \tag{1}$$

where $k = 2$ is the offset between frames.

The video locations set $\mathcal{V}$ is then mapped to its corresponding UV atlas locations: $\mathcal{S}_\mathcal{V} = \mathbb{M}(\mathcal{V})$, where $\mathbb{M}$ is a pre-trained mapping network. We define the atlas crop I_{Ac} as the minimal crop in the atlas space that contains all the mapped UV locations:

$$I_{Ac} = \begin{Bmatrix} I_A[u, v] & s.t. & \text{min}(\mathcal{S}_\mathcal{V}.u) \leq u \leq \text{max}(\mathcal{S}_\mathcal{V}.u) \\ & & \text{min}(\mathcal{S}_\mathcal{V}.v) \leq v \leq \text{max}(\mathcal{S}_\mathcal{V}.v), \end{Bmatrix} \tag{2}$$

We augment the atlas crop I_{Ac} as well as the target text T, as described in Sec. 2 herein to generate an internal training dataset. To apply our losses, we map back the atlas edit layer to the original video segment and process the edited frames the same way as in the image framework: resizing, applying CLIP augmentations, and applying the final loss function of Eq. 2 in Sec. 3.1 in the paper. To enrich the data, we also include one of the sampled frame crops as a direct input to G and apply the losses directly on the output (as in the image case). Similarly to the image framework, every 75 iterations we additionally pass the pair $\{I_A, T\}$, where I_A is the entire atlas (without augmentations, and without mapping back to frames). For the background atlas, we first downscale it by three due to memory limitations.

Inference. As described in Sec. 3.2, at inference time, the entire atlas I_A is fed into G_θ results in $\mathcal{E}_A$. The edit is mapped and combined with the original frames using the process that is described in [3](Sec. 3.4, Eq. (15),(16)). Note that our generator operates on a single atlas. To produce foreground and background edits, we train two separate generators for each atlas.

References

[1] Caron, M., Touvron, H., Misra, I., Jegou, H., Mairal, J., Bojanowski, P., Joulin, A.: Emerging properties in self-supervised vision transformers. ICCV (2021) 2

[2] Defazio, A., Jelassi, S.: Adaptivity without compromise: a momentumized, adaptive, dual averaged gradient method for stochastic optimization. arXiv preprint arXiv:2101.11075 (2021) 2

[3] Kasten, Y., Ofri, D., Wang, O., Dekel, T.: Layered neural atlases for consistent video editing. ACM Transactions on Graphics (TOG) **40**(6), 1–12 (2021) 2, 3

[4] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: Pytorch: An imperative style, high-performance deep learning library. In: Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., Garnett, R. (eds.) Advances in Neural Information Processing Systems 32, pp. 8024–8035. Curran Associates, Inc. (2019), http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf 2

[5] 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: International Conference on Machine Learning. pp. 8748–8763. PMLR (2021) 2

[6] Ronneberger, O., Fischer, P., Brox, T.: U-net: Convolutional networks for biomedical image segmentation. In: International Conference on Medical image computing and computer-assisted intervention. Springer (2015) 1