Stylized Text-to-Image Generation

Eric Vincent
Department of Computer Science
Stanford University
evincent@stanford.edu

Deepak Chandran
CCRMA
Stanford University
cdeepak@stanford.edu

Abstract

The automatic synthesis of images from text descriptors has practical and creative applications in the fields of computer-aided design, art generation, etc. This project combines this idea of rendering images from text descriptions with neural style transfer to generate stylized versions of these images. Our implementation method involves generating a stylized output image directly from a (stacked) GAN system conditioned on the style image as well as the text description, as opposed to generating the image first and subsequently transferring style without knowledge of the text description. With enough training time, our new model would be a large improvement in running time over the existing baseline process of first generating images and then stylizing the output, because style transfer is a slow iterative process, whereas GANs, once trained, only require a feedforward pass of the generator to produce an output.

1 Introduction

Current methods for generating stylized images from text descriptions (i.e. our baseline) first generate an images from text with a GAN system, then stylize the results with neural style transfer. This takes a long time, due to the use of neural style transfer, and would not be suitable to use in any real-time applications or on low performance computers.

Using a system of only GANs seems like a desirable alternative method to achieve the same goal in much less time. The main obstacle, however, which is likely why there have not been prior attempts to transfer style with GANs, is that there is no existing training data linking images to their stylized alternatives in a variety of styles.

To achieve a GAN-only model without existing training data, we bootstrapped neural style transfer to generate our own training data from unstyled images. This, in combination with a captioning dataset, allows us to get training data with a caption and corresponding styled image. We could then condition the GAN on the image style and image caption and train the system to generate styled images from caption that are hard to distinguish from the neural style transfer generated images.

Another difficulty of this approach is that GANs are notoriously hard to train, and this especially true in our case since we don't have a perfect automatic evaluation criterion.

Unfortunately the system also takes quite a while to train to a satisfactory level, which, in combination with the training difficulties of GANs, made it prohibitively time-consuming for us to execute a full hyperparameter search and optimization process.

2 Related Work

The simplest, original approach to text-to-image generation is a single GAN that takes a text caption embedding vector as input and produces a low resolution output image of the content described in the caption [5].

The text embeddings for these models are produced by a separate neural net. A popular approach is convolutional-recurrent network, where the input characters are passed through some convolutional layers first, then their output is passed through an LSTM, and the final output is the average hidden unit activation over the sequence [6].

Simple GAN systems like these can generate low resolution images which often represent the content of the text descriptions well, but they don't scale well to larger resolutions.

StackGAN managed to generate more realistic, higher resolution images by splitting the problem into two simpler components: Stage-I GAN, which generates low resolution images from the text which vaguely capture the meaning (like the methods described above), and Stage-II GAN which takes Stage-I's output and the text again, and generates a higher resolution version with more detail [9].

Right now, image stylization is separate from image generation, and is achieved by neural style transfer, which iteratively refines an image to minimize an image's "style loss" and "content loss", which measure how similar the produced image is in content to the content image and how similar it is in style to the style image, through an ImageNet-trained CNN model hidden activations and a gram matrix of earlier hidden activations of the same network respectively [2].

3 Approach

3.1 Baseline

Our baseline model first generates images from text using StackGAN, then feeds the outputs into style transfer [2] [1] using a bash and python script under default configurations.

While we call this the "baseline" model, it is only such in terms of execution time and potentially output content accuracy, and is instead more of an oracle in terms of output style accuracy, especially given that we will be using style transfer to help generate our dataset (see below).

3.2 Improvements

In order to combine the two models into one GAN system, we used StackGAN as a starting point, and added the new conditioning to the Stage-II GAN. This means the Stage-I GAN still generates images that capture the basic meaning of the image at low resolution, but the Stage-II GAN now generates higher resolution images in the given style as well.

To train, the loss functions don't need to be changed: only the images being passed compared. This means we train the discriminator network on stylized 256x256 images, instead of the originals.

To handle more than one style, we can condition on either the style's index or an embedding of the style in both networks of Stage-II. In order to do this, we will pass the index/embedding as an additional input to the second stage's generator and discriminator networks. In order to do so, they need to be reshaped to conform to the shape of the down-sampled image volume of each network. This can be done stacking each index/embedding redundantly in each row and column of the volume as new channels, in a similar way that StackGAN inputs text embeddings to these networks. This method enables even small convolutions to have access to the same information at each position in the image, which is desirable. See Figure 1 for a diagram sketch of the architecture.

Conditioning on an embedding would allow the system to generalize to more unseen styles, whereas an index would only allow it to produce styles present in the training set, with the only benefit of conditioning being that knowledge learned from producing one style of image can benefit the production of other styles. While conditioning on the style embeddings seems desirable for this generalization ability, it would require training on a variety of styles, which also requires generating styled training data for these styles: both of which were not a possibility for us in the time we had. For this reason, we decided to condition only on style indices, which unfortunately means the model

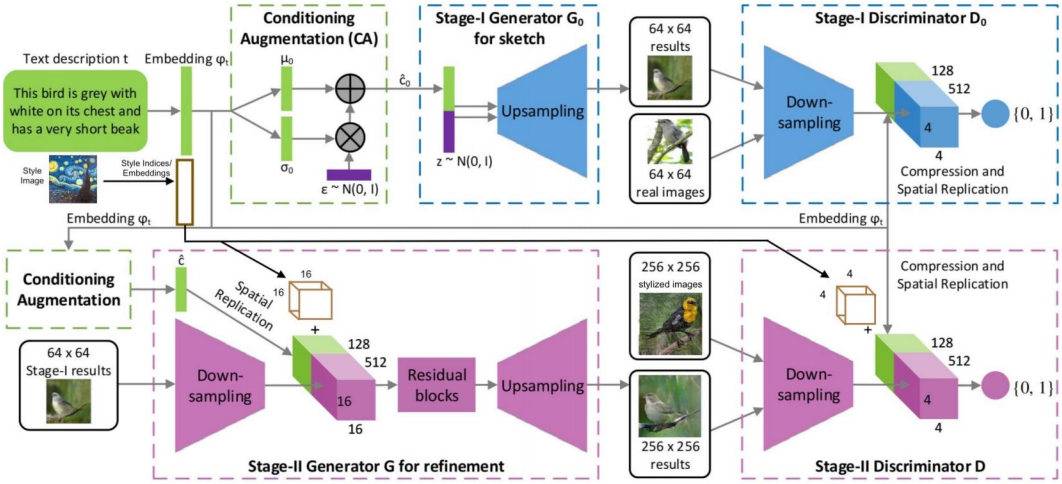

Figure 1: Diagram of the new architecture modified from StackGAN.

has no style generalization ability, and is only marginally better than training on each single style separately.

4 Experiments

4.1 Data

Our task requires data pairs in the form <text description, stylized image>. We weren't aware of any datasets matching this format, so we instead bootstrap an "image captioning" dataset, which have data pairs in the form <text description, image>, by generating the stylized images using neural style transfer. While this will limit our output style accuracy to that of the "baseline", we have been treating the baseline as an oracle in terms of style accuracy, and a baseline in terms of content accuracy and execution time.

There are several image captioning datasets available to choose from, including:

- COCO: over 200,000 photos of a large variety of subject, with 5 captions per image [3]
- CUB-200-2011: 6,033 photos spanning 200 species of birds [8]. Descriptions originally in the form of attribute tags; plain text captions subsequently provided by Reed et. al. [7].

COCO's large variance of subject matter meant that it was necessary to train on a large volume of photos, which in our case, also requires generating stylized versions of all these photos, potentially multiple times, to handle each style. To limit training and data generation time to a more reasonable level, we decided to not only switch to a smaller dataset, CUB, but to limit the dataset to a smaller number of species within the dataset. (we split the data by species, not as a random sample, in order to ensure that each category had enough examples). We generated stylized images in two styles for the first 451 images, which corresponded to 8 species of birds (larger than average categories for the CUB dataset).

Also note that we didn't make a development segment of our dataset, because training and dataset generation collectively take so long that we couldn't get a fully trained model on a full stylized dataset, meaning that any evaluation of our model would be nearly useless.

4.2 Evaluation Method

The inception score is a reasonable metric to help show if the model is producing both diverse and meaningful images. However, this isn't sufficient to judge content and style accuracy, so looking at the images manually is necessary tool. See the "Analysis" section for some qualitative analysis of the produced images.

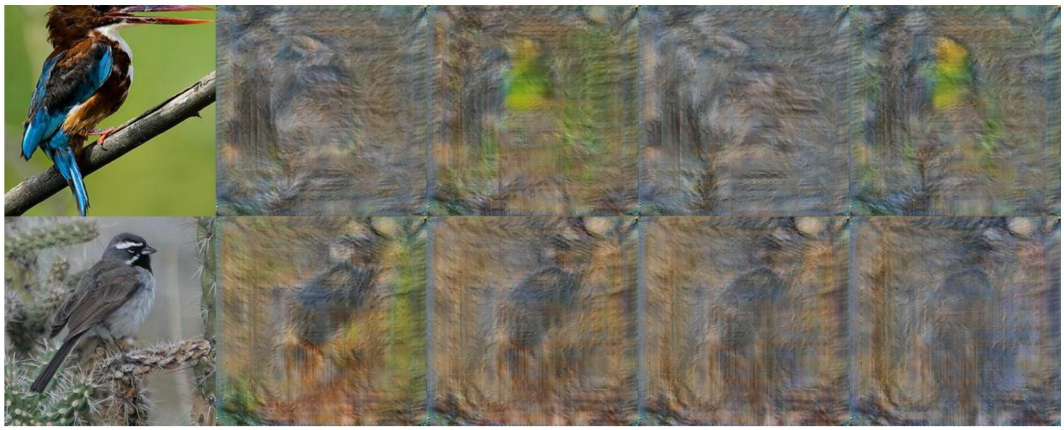

Figure 2: GAN system trained on a single style (Starry Night) on untrained (random noise) Stage-I outputs; unstyled, placeholder test set images shown at left and several outputs are shown in the rest of each row.

Another metric we strived to improve is running time of the model. However, this improvement is a given: generating an image with the GAN takes under a second, while style transfer takes on the order of 2-3 minutes with a reasonable graphics card. For this reason, we didn't analyze runtime, and instead tried to achieve acceptable quality given the fast execution.

4.3 Experimental Details

We wrote code to encapsulate our project into a black box function in order to run constrained Bayesian Optimization [4] for a hyperparameter search, but given that we need to execute each iteration for a large number of epochs before we start to see meaningful results, we decided to just run our code with a few promising hyperparameter settings.

The hyperparameters that are most promising to optimize in the future are:

- DISCRIMINATOR_LR / GENERATOR_LR: learning rates for discriminator and generator
- PRETRAINED_EPOCH / LR_DECAY_EPOCH: delay and frequency of learning rate
- EMBEDDING_DIM: length of text embedding vector

Note that (to train a full model, the authors of StackGAN say that it can take 2-3 days on a TitanX Pascal GPU).

4.4 Results

No meaningful quantitative results were obtained in this process, because as described above, it takes too long to reasonably generate enough stylized training data and to perform a hyperparameter search with enough iterations each time to find the best settings. We do however show some qualitative results in the Analysis section that demonstrate that model can output images that at least show styling similar to the input style image, even if the content component couldn't be seen to any level of detail.

5 Analysis

As a preliminary example demonstrating styled outputs, see figure 2, which shows styled output of Stage-II, when it is trained on untrained Stage-I outputs (random noise). Brush strokes reminiscent of Starry Night, the style image, can be seen especially clearly given the near uniform color and form.

Figure 3 shows the outputs of Stage-II when trained for 2000 iterations(on the left), and when trained for 4000 iterations(on the right). The model starts to converge such that all outputs have a similar pattern. This is evidence that more hyperparameter tuning is necessary (likely the learning rates of the generator and discriminator). Despite this, both show styles still somewhat reminiscent of brush

strokes (less so for the 4000-iteration trained model). Note that StackGAN adds random noise to the text embedding, adding variety to the outputs.

Note that the difficulties in training only get worse with more styles, because more data is needed for each style, and the model needs to learn multiple image distributions; see figure 4 for an example case of divergent generator and discriminator.

Figure 3: GAN system trained on a single style (Starry Night) for 2000 and 4000 updates; real test set images shown at left and several outputs are shown in the rest of each row.

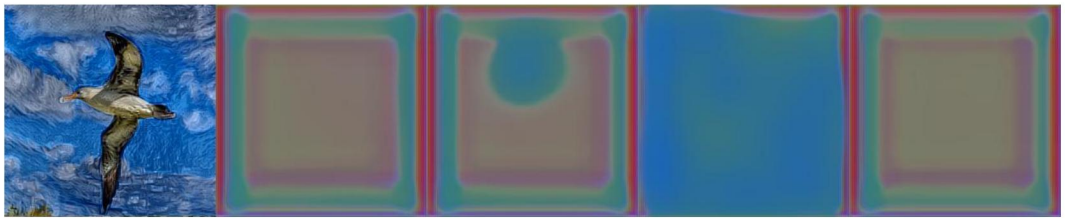

Figure 4: GAN system conditioned on style indices for two styles; results highlight difficulty of finding optimal hyperparameters

6 Conclusions

We have demonstrated that our architecture produces images that resemble the style of the input dataset. Given enough time for stylized dataset generation and model training / hyperparameter search, we are confident that the images produced could also closely represent the content described in the caption and that the system could handle multiple styles.

However, due to the considerable computational cost of generating a stylized image dataset and training the model, we don't recommend attempting to train a system of this architecture unless the time savings are instrumental to the desired application and such training time is available.

6.1 Future work

With more processing power, several more interesting possibilities arise:

First, we anticipate that conditioning the stage-II GAN on image style embedding (such as the gram matrix of the low level hidden activations of VGGNet on the style image) instead of just style ID could yield a model that effectively generalizes to unseen styles.

We also expect that one might be able to achieve better results by also conditioning the text encoder on the image style, such that it produces embeddings that more accurately predict where each element described in the text should be placed in the (styled) output image.

References

[1] Anish Athalye. *Neural Style*. https://github.com/anishathalye/neural-style. commit c1ecf186. 2015.

[2] Leon Gatys, Alexander Ecker, and Matthias Bethge. "A Neural Algorithm of Artistic Style". In: *Journal of Vision* 16.12 (2016), p. 326. DOI: 10.1167/16.12.326.

[3] Tsung-Yi Lin et al. "Microsoft COCO: Common Objects in Context". In: *Computer Vision – ECCV 2014 Lecture Notes in Computer Science* (2014), pp. 740–755. DOI: 10.1007/978-3-319-10602-1_48.

[4] Fernando Nogueira. *Bayesian Optimization*. https://github.com/fmfn/BayesianOptimization. commit 20b2ce49.

[5] Scott E. Reed et al. "Generative Adversarial Text to Image Synthesis". In: *CoRR* abs/1605.05396 (2016). arXiv: 1605.05396. URL: http://arxiv.org/abs/1605.05396.

[6] Scott E. Reed et al. "Learning Deep Representations of Fine-grained Visual Descriptions". In: *CoRR* abs/1605.05395 (2016). arXiv: 1605.05395. URL: http://arxiv.org/abs/1605.05395.

[7] Scott Reed et al. "Learning Deep Representations of Fine-Grained Visual Descriptions". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016). DOI: 10.1109/cvpr.2016.13.

[8] C. Wah et al. *The Caltech-UCSD Birds-200-2011 Dataset*. Tech. rep. CNS-TR-2011-001. California Institute of Technology, 2011.

[9] Han Zhang, Tao Xu, and Hongsheng Li. "StackGAN: Text to Photo-Realistic Image Synthesis with Stacked Generative Adversarial Networks". In: *2017 IEEE International Conference on Computer Vision (ICCV)* (2017). DOI: 10.1109/iccv.2017.629.

7 Appendix: Implementation Notes

One should note the numerous implementation difficulties/annoyances of taking this approach.

For the baseline model:

- The neural-style repository runs in python 3.6 while StackGAN (pytorch or tensorflow versions) runs in python 2.7. There are numerous other conflicts in dependencies, so we set up separate conda environments for each, which have been provided with our code.
- We need to switch conda environments midway through the baseline code, so use a bash wrapper script that calls a python script for each step in the appropriate environment.

Running StackGAN:

- StackGAN runs in python 2.7 and tensorflow 1.0.1 (0.12 listed in StackGAN's readme, but needed to be updated to handle other conflicts), which requires cudnn=5.0 and cudatoolkit=8.0 (these aren't included with current AWS EC2 instances, but can be installed with conda, meaning the conda environment path needs to be added to torch's environment variable LD_LIBARY_PATH)
  - Note that the upgrade to tensorflow 1.0.1 comes with some other changes, all necessary to get StackGAN to a working configuration, which are listed in this github issue thread: https://github.com/torch/distro/issues/239#issuecomment-340136687
- Torch (lua version) needs to be installed in order to run the pretrained CHAR-CNN-RNN embeddings
  - install from http://torch.ch/docs/getting-started.html#_
  - but before running "./install.sh", run these steps: https://github.com/torch/distro/issues/239#issuecomment-340136687
- Note that the tensorflow version of StackGAN should be used, because the pytorch version does not come with a pretrained model for the CUB dataset, and there are small architectural changes between the two versions that prevent reuse of the model weights.

More information about our configuration can be found in our github repository.