Text to Artistic Image Generation using GANs

Yuxing Chen
Symbolic Systems Program
yxchen28@stanford.edu

Zhefan Wang
Department of Electrical Engineering
zwang141@stanford.edu

Abstract

Given an input sentence and a desired style, we build a system which can generate both a photo-like image and a synthetic image whose content matches the sentence description and style matches the target style. Considering the limited time and GPU resource, we believe we achieved decently well results.

1. Introduction

Generating images from texts has been a trending research topic in computer vision. Style transferring between photos and artwork is also a popular subfield. We build an application that combines these two, which allows the user to not only generate ordinary photo-like images from sentences, but also get the certain artistic style of images specified by the user.

To be more specific, our algorithm needs two inputs. The first input is a sentence that describes the scene we want to generate. The second input is an artistic style to be applied. We first feed the input sentence to a two-stage GAN that generates a realistic photo-like image. Then, we feed this photo-like image to the image transfer network to get the desired stylized output image. Note that the image transfer network is trained per style images. Therefore, user should only select a style image in the given list of choices.

2. Related Work

Much work has been done in text-to-image generation. In [9], Reed *et al*. discussed different approaches to embedding images and fine-grained visual descriptions jointly. Reed *et al*. proposed a novel deep architecture and GAN formulation to generate plausible images from detailed text descriptions [10]. Zhang *et al*. improved the quality of the synthesized photo-realistic images using Stacked Generative Adversarial Networks (StackGAN) [19]. Xu *et al*. built an Attention Generative Adversarial Network (AttnGAN) to address the problem that images generated using GANs conditioning on the global sentence vector lack essential fine-grained word-level information [17].

Image transformation is also a heavily researched area in recent years. Gatys *et al*. [4] performs image style transfer by jointly minimizing the feature reconstruction loss of [8] and the style reconstruction loss of [2]. Their approach produces high-quality results, but suffer from very high computational cost since inference requires solving an optimization problem. Johnson *et al*. proposed an approach to solve the optimization problem in real-time[5]. They trained a feed-forward network that generates qualitatively similar results to that of Gatys in three orders of magnitude shorter time. Ulyanov *et al*. shows in [16] that substitute batch normalization with instance normalization both at training and testing times in the stylization architecture results in a significant qualitative improvement in the generated images. One potential shortcoming of all these approaches is that these algorithms alter the colors of the original painting. Specifically, as a byproduct of style transfer, colors of the stylized images look alike the colors of the style image, which may change the appearance of the scene in undesirable ways. Gatyes *et al*. presents two linear methods for transferring style while preserving colors in [3].

3. Methods

The system can be decomposed into two parts: a text-to-image generator and a style transferrer. User input is fed into the text-to-image generator. The generator produces an intermediate photo-like image result. The style transferrer then takes in the intermediate result and produces the final synthetic stylish image.

3.1. Text to Image Generation

Text Encoder

The first challenge is to correctly connect the content of images and the natural language concepts in the corresponding text descriptions. As mentioned in the previous section, a widely-applied way of encoding text descriptions is to use deep convolution and recurrent text encoder (i.e. char-CNN-RNN model) which learn the correspondence function with images [9, 10]. The idea of this approach is that

an recurrent neural network is stacked on top of a temporal convolutional neural network hidden layer. This helps prevent the drawback of lacking temporal dependencies along the input word sequence when only using CNN, preserving the advantage that low-level temporal features can be learned fast and efficiently. Figure 1 visualizes the architecture of this text encoder.

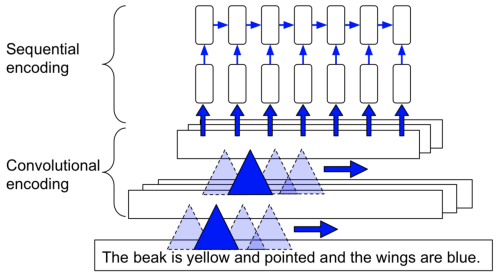

Figure 1. The architecture of char-CNN-RNN text encoder [9]

The objective function that we want to optimize is defined as follows [9, 10]:

$$\frac{1}{N}\sum_{n=1}^{N}\Delta(y_n, f_v(v_n)) + \Delta(y_n, f_t(t_n))$$

, where $v_n \in \mathcal{V}$ is the visual information in the dataset, $t_n \in \mathcal{T}$ is the text description, and $y_n \in \mathcal{Y}$ is the class label. Δ is the 0-1 loss. f_v, f_t are image and text classifiers, respectively, which are formulated as:

$$f_v(v) = \underset{y\in\mathcal{Y}}{\operatorname{argmax}}\, \mathbb{E}_{t\sim\mathcal{T}(y)}[\theta(v)^T\psi(t)]$$

$$f_t(t) = \underset{v\in\mathcal{V}}{\operatorname{argmax}}\, \mathbb{E}_{v\sim\mathcal{T}(y)}[\theta(v)^T\psi(t)]$$

Here, $\theta(v)$ is the image encoder, and $\psi(t)$ is the text encoder.

In the original paper [9], Reed *et al.* compared several different encoding models, including word2vec, BOW (bag-of-words), char-CNN-RNN model, word-CNN-RNN model, and etc. Here we choose to use char-CNN-RNN model, the same encoder as used in the Generative Adversarial Text-to-Image Synthesis project [10].

Conditioning Augmentation

Conditioning augmentation technique is designed to deal with the discontinuity in the latent data manifold when the amount of data is limited [19]. This discontinuity comes from the fact that the latent space for text embeddings is high-dimensional, while we transform text embeddings nonlinearly to get the conditioning latent variables (which are the inputs of the generator). By using additional conditioning variables $\hat{c}$, we introduce randomness to the model. $\hat{c}$ are sampled from an independent Gaussian distribution $\mathcal{N}(\mu(\phi_t), \Sigma(\phi_t))$, where ϕ_t represents the text embedding. We also use the Kullback-Leibler divergence to preserve smoothness and avoid the overfitting problem [1, 6].

StackGAN

The network has two stages. Stage-I GAN sketches the primitive shape and colors of the object based on the text description, yielding low-resolution images. Stage-II GAN takes Stage-I results and text descriptions, and generates high-resolution images with more realistic details.

Let t be the text description for the real image I_0, ϕ_t be the text embeddings of the given description and z be a noise vector randomly sampled from a given distribution p_{data}. The Gaussian conditioning variables $\hat{c}_0$ for text embeddings are sampled from $\mathcal{N}(\mu_0(\phi_t), \Sigma_0(\phi_t))$. Denote the Stage-I GAN discriminator as D_0 and the Stage-I GAN generator as G_0. We train D_0 and G_0 by maximizing $\mathcal{L}_{D_0}$ and minimizing $\mathcal{L}_{G_0}$, alternatively:

$$\begin{aligned}\mathcal{L}_{D_0} &= \mathbb{E}_{(I_0,t)\sim p_{data}}[\log D_0(I_0, \phi_t)] \\ &\quad + \mathbb{E}_{z\sim p_z, t\sim p_{data}}[\log(1 - D_0(G_0(z, \hat{c}_0), \phi_t))],\end{aligned}$$

$$\begin{aligned}\mathcal{L}_{G_0} &= \mathbb{E}_{z\sim p_z, t\sim p_{data}}[\log(1 - D_0(G_0(z, \hat{c}_0), \phi_t))] \\ &\quad + \lambda D_{KL}(\mathcal{N}(\mu_0(\phi_t), \Sigma_0(\phi_t))||\mathcal{N}(0, I)),\end{aligned}$$

where D_{KL} is the Kullback-Leibler divergence and λ is a regularization parameter.

Stage-II GAN is built upon the results of Stage-I GAN. Let $s_0 = G_0(z, \hat{c}_0)$ be the low-resolution result from Stage-I, $\hat{c}$ be the Gaussian latent variables, D be the Stage-II discriminator and G be the Stage-II generator. Then, D and G are trained by maximizing $\mathcal{L}_D$ and minimizing $\mathcal{L}_G$, alternatively:

$$\begin{aligned}\mathcal{L}_D &= \mathbb{E}_{(I,t)\sim p_{data}}[\log D(I, \phi_t)] \\ &\quad + \mathbb{E}_{s_0\sim p_{G_0}, t\sim p_{data}}[\log(1 - D(G(s_0, \hat{c}), \phi_t))],\end{aligned}$$

$$\begin{aligned}\mathcal{L}_G &= \mathbb{E}_{s_0\sim p_{G_0}, t\sim p_{data}}[\log(1 - D(G(s_0, \hat{c}), \phi_t))] \\ &\quad + \lambda D_{KL}(\mathcal{N}(\mu(\phi_t), \Sigma(\phi_t))||\mathcal{N}(0, I)).\end{aligned}$$

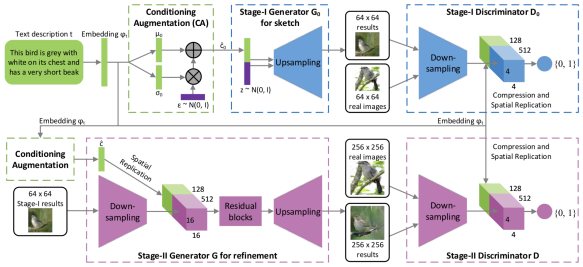

Figure 2. The architecture of the proposed StackGAN [19]

The overall architecture is shown in figure 2. Each up-sampling block contains the nearest-neighbor up-sampling

followed by a 3×3 stride 1 `conv2d` layer, Batch normalization and `ReLU` activation. Residual block consists of a 3×3 stride 1 `conv2d` layer followed by Batch normalization, `ReLU` activation, another 3×3 stride 1 `conv2d` layer, Batch normalization and `ReLU` activation. Each down-sampling block has a 4×4 stride 2 `conv2d` layer, followed by Batch normalization and `LeakyReLU`, except that the first down-sampling block does not contain Batch normalization.

3.2. Style Transfer

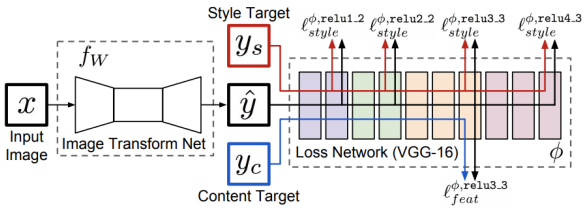

Figure 3. Style Transfer System

As discussed in [5], one common approach to solve image transformation tasks is to train a feed-forward CNN using per-pixel loss function that measures the difference between output and ground-truth images. Such approach is fast to run at test-time but the per-pixel loss cannot capture perceptual differences between output and ground-truth images. In parallel, recent work such as [4] shows that images can be generated using perceptual loss functions based on differences between high-level image feature representations extracted from pretrained CNNs. Such approach can produce high-quality images, but is very slow due to inference requiring solving an optimization problem.

To combine the benefits of both methods and avoid the mentioned disadvantages, we followed the work [5] and trained a style transfer system as shown in 3. The system consists of two parts: an image transformation network and a lost network. Instead of using a per-pixel loss function or solving an inference problem, a perceptual loss function that depends on high-level features from a pretrained loss network (VGG-16) is defined. At test-time, the image transform net runs in real-time to generate stylized output images.

- Image transformation network: The exact architecture is shown in figure 4. The network essentially down-samples and then upsamples the input image. All convolution layers are immediately followed by batch normalization and `ReLU` activation. Except that the output layer uses a scaled `tanh` activation function such that the output values are in the range $[0, 255]$.

- Loss network: This network is the 16-layer pretrained VGG network. It remains fixed during training, ie. weights are not updated upon back propagation. It examines the perceptual differences in content and style between images. Denote the loss network as ϕ, input images as x and output images as $\hat{y}$ via the mapping $\hat{y} = f_W(x)$. Each ℓ_i calculates the difference between output image $\hat{y}$ and target image y_i (y_i is either the input content image or the input style image). The loss function that captures both style loss and feature (ie.content) loss is define as:

$$\mathcal{W}^* = \underset{W}{\operatorname{argmin}}\, \mathbb{E}_{x,y_i}\Big[\sum_i \lambda_i \ell_i(f_W(x), y_i)\Big]$$

To be more specific, define the j-th feature map in the loss net as ϕ_j of shape $C_j \times H_j \times W_j$. The feature loss is defined as:

$$\ell^{\phi,j}_{feat}(\hat{y}, y) = \frac{1}{C_j H_j W_j}\|\phi_j(\hat{y}) - \phi_j(y)\|^2$$

As for the style loss, Gram matrices is used to measure the difference between target style and the predicted images:

$$\ell^{\phi,j}_{style}(\hat{y}, y) = \|G^\phi_j(\hat{y}) - G^\phi_j(y)\|^2_F$$

Here G^ϕ_j is a $C_j \times C_j$ Gram matrix whose elements are given by:

$$G^\phi_j(x)_{c,c'} = \frac{1}{C_j H_j W_j}\sum_{h=1}^{H_j}\sum_{w=1}^{W_j}\phi_j(x)_{h,w,c}\phi_j(x)_{h,w,c'}$$

As shown in figure 3, the overall loss function used by [5] is:

$$\begin{aligned} Loss = \lambda_c \cdot \ell^{\phi,relue3\_3}_{feat}(\hat{y}, y_c) + \lambda_s \cdot (\ell^{\phi,relue1\_2}_{style}(\hat{y}, y_s) \\ + \ell^{\phi,relue2\_2}_{style}(\hat{y}, y_s) + \ell^{\phi,relue3\_3}_{style}(\hat{y}, y_s) \\ + \ell^{\phi,relue4\_3}_{style}(\hat{y}, y_s)) \end{aligned}$$

Layer	Activation size
Input	$3 \times 256 \times 256$
$32 \times 9 \times 9$ conv, stride 1	$32 \times 256 \times 256$
$64 \times 3 \times 3$ conv, stride 2	$64 \times 128 \times 128$
$128 \times 3 \times 3$ conv, stride 2	$128 \times 64 \times 64$
Residual block, 128 filters	$128 \times 64 \times 64$
Residual block, 128 filters	$128 \times 64 \times 64$
Residual block, 128 filters	$128 \times 64 \times 64$
Residual block, 128 filters	$128 \times 64 \times 64$
Residual block, 128 filters	$128 \times 64 \times 64$
$64 \times 3 \times 3$ conv, stride 1/2	$64 \times 128 \times 128$
$32 \times 3 \times 3$ conv, stride 1/2	$32 \times 256 \times 256$
$3 \times 9 \times 9$ conv, stride 1	$3 \times 256 \times 256$

Figure 4. Architecture used for style transfer network

This is our prototype model. We then did two optimizations on top of it.

- Replace all batch normalization with instance normalization both at training and testing times in the image transform net. This leads to a huge qualitative improvement in the generated images.

- Optionally preserve the colors of the original image:

  As a byproduct of style transfer, colors of the stylized images look alike the colors of the style image, which very often changes the appearance of the scene in undesirable ways. Luckily, we come up a quick fix. Visual perception is far more sensitive to changes in luminance than in colors. We decomposed the style transfered image $\hat{y}$ and the content image y_c into three separate channels in `HUV` space. Denote them as $\hat{c}_1$, $\hat{c}_2$, $\hat{c}_3$ and c_{cont1}, c_{cont2}, c_{cont3}. Then combining $\hat{c}_1$, c_{cont2}, and c_{cont3} gives a stylized image that preserves the original color.

4. Dataset and Features

We perform experiments on Microsoft COCO dataset. It contains 328,000 images with 5 captions per image and 91 object categories [7]. For our project, we use the 2014 dataset with an 80K/40K train/val split.

4.1. Text to Image Generation

Due to speed and memory limit of Google Cloud service, we decide to select 40,000 out of 82,783 examples from the 2014 training set. The size of the validation set is adjusted accordingly to match our new training set. Figure 5 shows one pair of training example:

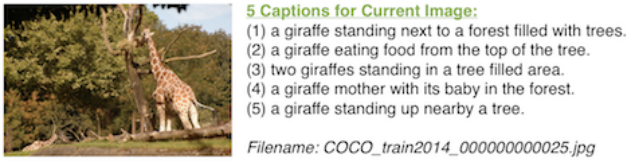

Figure 5. Microsoft COCO dataset example

Before feeding image data into the image generator, we preprocess each image by cropping the given image at a random location with output size 64×64 and horizontally flipping it randomly with fixed probability 0.5. Then we convert it into a $C \times H \times W$ `torch.FloatTensor` in the range $[0, 1]$ and normalize it using `Normalize((0.5,0.5,0.5),(0.5,0.5,0.5))` in the `torchvision.transforms` package.

4.2. Style Transfer

- As shown in 3, the style transfer system uses the pre-trained VGG16 model provided in the Pytorch model zoo. VGG16 is a 16-layer convolutional network for large-scale image recognition and was used in the ILSVRC-2014 competition [13].

- The style transfer system uses a subset of the 2014 train/val dataset. Specifically, the training set contains 8,000 images and the validation set contains 3,000 images. Annotation data is not needed.

  All pre-trained models in Pytorch model zoo expect input images normalized to mini-batches of 3-channel RGB images of shape (3 x H x W), where H and W are expected to be at least 224. The images have to be loaded in to a range of [0, 1] and then normalized using mean=[0.485, 0.456, 0.406] and std=[0.229, 0.224, 0.225]. Thus we preprocess each image by resizing and cropping the images so that images fed into the network are 256×256 large. We then convert all images into $C \times H \times W$ `torch.FloatTensor` in the range $[0, 1]$ and normalize it using `Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])` in the `torchvision.transforms` package.

5. Metrics, Experiments and Results

5.1. Expected Results

Given an arbitrary input sentence and a desired style image, the system is expected to generate a synthetic image whose content matches the sentence description and style matches the target style.

5.2. Evaluation Metrics

1. **Text to Image Generation:** Work [11] proposed a numerical assessment approach called "inception score". We plan to evaluate this metric on a large number of random samples by directly use the pre-trained Inception model for COCO dataset. Inception score quantitatively evaluates the performance of our model:

$$I = \exp(\mathbb{E}_x D_{KL}(p(y|x)||p(y))),$$

   where x is a generated sample, and y is the label predicts by the Inception model [15, 19].

2. **Style Transfer:** Quantitative evaluation of artistic style transfer is difficult. The loss contains many gram matrices. This implies that the measurement and reductions in loss is very much subjective to that particular image [14]. We examined many papers and none of them proposed any quantitative measure that determines the overall effectiveness of the model. Therefore, out primary evaluation of the style transfer system will be qualitative. The ideal result is that the style is adapted onto the content without overwhelming it.

5.3. Experiments on StackGAN

Baseline

We first build our model upon the original architecture specified in the work of Zhang *et al*. [19]. Instead of using the whole COCO 2014 train images as the training dataset, we first select 2,000 images as our toy training dataset. The StackGAN model is trained on the toy COCO dataset using preprocessed COCO char-CNN-RNN embeddings provided by Han Zhang [18]. We train the Stage-I GAN and the Stage-II GAN for 50 epochs and 15 epochs, respectively. The pattern of different losses [fig.6, fig.7] we got seems reasonable:

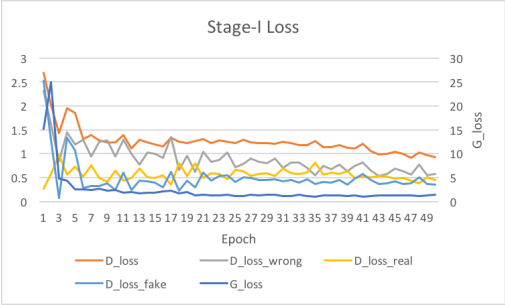

Figure 6. Stage-I loss for toy set

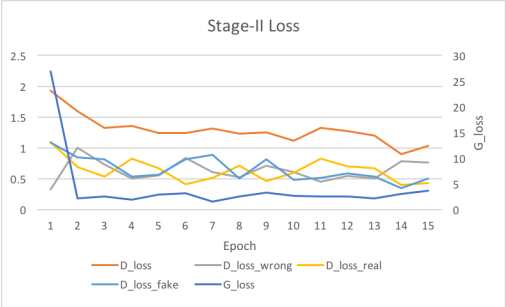

Figure 7. Stage-II loss for toy set

However, samples generated from captions in COCO validation set are really bad, as we can see in figure 8. We find that a training set with only 2,000 examples is way too small. We can hardly tell anything from the images generated by the model using our toy dataset, which motivates us to increase the size of the training dataset to have 40,000 images and do experiments on it.

5.3.1 Hyper-parameters Choosing

We first fix Stage-II and train Stage-I GAN with Conditioning Augmentation for 40 epochs. Then we train Stage-II GAN for another x epochs with fixed Stage-I GAN.

- Shared hyper-parameters between two stages: We set the input text embedding dimension obtained from the text encoder to be 1024, and we set the dimension for random noise vector as 100. The coefficient for Kullback-Leibler divergence is 2.

- Hyper-parameters for Stage-I GAN: To begin with, we set the learning rates for both the discriminator D_0 and generator G_0 to be 0.2×10^{-3}, which is the value suggested by Zhang *et al*. in their work [19]. During the training process, we adjust each learning rate every 10 epochs by multiplying 0.5. As we know, although high learning rate makes the loss decrease faster, it may also make our model get stuck at some point. In order to avoid this issue, we choose to use decreasing learning rates. The batch size is set to be 64. In [19], they use 128 as the batch size. However, due to the consideration of memory limit and smaller dataset, we use 64 instead. The optimizer we use here is Adam with `betas=(0.5, 0.999)`. Since the purpose of Stage-I GAN is to learn the rough shape and basic colors conditioned on text descriptions, there is no need to generate high-resolution images. Therefore, the size of the image generated by Stage-I GAN is set to be 64×64.

- Hyper-parameters for Stage-II GAN: Similar to what we have in Stage-I GAN, we also use decreasing learning rates for both the discriminator D and the generator G for the same reason. The initial value of each is 0.2×10^{-3}. After two epochs, the new learning rates will be 0.15×10^{-3}, and so on. We also use Adam optimizer with `betas=(0.5, 0.999)`. The batch size is set to be 32. Since Stage-II GAN is much larger and harder to train, we decide to use this value instead of 40 as suggested in the code written by the original authors of this paper [19].

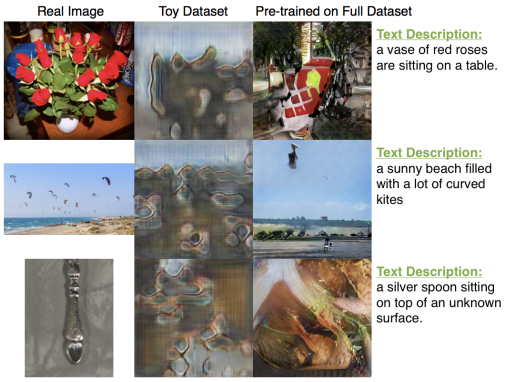

Figure 8. Samples Generated from Captions in Validation Set

5.3.2 Results

Figure 9 are some example results of different models.

- Model I: 50 epochs on Stage-I GAN training and 15 epochs on Stage-II GAN training. The toy dataset with 2,000 images is applied.
- Model II: 40 epochs only on Stage-I GAN training. The training dataset contains 40,000 examples selected from 2014 COCO training set.
- Model III: 40 epochs on Stage-I GAN training and 16 epochs on Stage-II GAN training. The training dataset is the same as we used in Model II.
- Model IV: Pretrained model provided by the author of StackGAN [19] with 90 epochs. The model is trained on the full 2014 COCO training set.

Figure 9. Image generated from the same text description using different models

As we can see from the figure above, there is an significant improvement between the first two model, which implies that in general GANs requires large dataset to train on. The image generated by Model III (Stage-I + Stage-II GANs) has higher resolution and more details (i.e. water taps over the sink, the cabinet under the sink) than the image generated by Model II (only Stage-I GAN). The image generated by Model II seems to give better shape of the objects indicated in the text description, so people can easily recognize that it might be a photo taken in the bathroom.

The table 1 below shows the inception score for each of the models we trained.

Table 1. Inception Scores for Different Models

Model Name	Inception Score (mean $\pm$ std)
Model I	1.7994797 ± 0.15638991
Model II	3.9063458 ± 0.37369365
Model III	3.5610385 ± 0.31590825
Model IV	N/A

After analyzing the generated results, we find that our current model did well in plotting the "background" indicated in the text descriptions, while it often failed to "fill in" the details of the human-like objects, as shown in figure 10. The top four are the real images, and the bottom four are the fake image generated using Model III. For example, the text description for the second image is "a man riding on the back of a brown horse down the street." We can easily tell that the fake image contains the "street" component, and there is a shape of a certain type of animals like a cow or a horse. However, "the man" and "the horse" are missing from the image, and it seems that they should fill in the blank within that shape. The problem described above may results from the fact that we performed the limited number of epochs on training Stage-II GAN, but it still remains to be verified.

Figure 10. Well-generated background, but losing the main characters

5.4. Experiments on Style Transfer

5.4.1 Hyper-parameters

- $lambda\_c = 1$, $lambda\_s$ varies from 1.9×10^5 to 4.8×10^6: This ratio is the most important hyper parameter. It has a heavy influence on the stylized images. $loss = lambda\_c \times content\_loss + lambda\_s \times style\_loss$. Fix $lambda\_c = 1$, the larger $lambda\_s$ is, the stronger the influence of the style image will be. To make the style noticeable on the generated image, different style images use very different ratios.
- Learning rate: 0.5×10^{-3}. Training at this rate results in good outputs after decent number of ephocs. When the learning rate is very large, loss explodes. When the learning rate is too small, learning progress is slow.
- Optimizer: Adam appears to be good enough.
- Batch size: 4. Batch size of 8 results in out of memory issue.

5.4.2 Results

Figures 11 shows the training losses of several styles. Figures 12,13 and 14 show some example results of the style transfer system. The first row is input images from the validation set. The second row is the style transferred images with original color preserved. The third row is the style transferred images without color preservation. The last row is the style images.

We see that the image transfer net successfully adapted the target style onto the content images. By tweaking the ratio of $lambda\_c$ to $lambda\_s$, we can have weaker or stronger effects. We observe that the color of images on the second row is very similar to the color of the first row (ie. the content images), whereas the color of images on the third row is very similar to the color of the bottom row (ie. the style image). This proves that our color preservation optimization technique works well in most cases. But there are some images where color preservation doesn't work that well. For example, in figure 13, the images on the second row are all darker than the content images. There are also some white artifacts around some strokes. In future, we plan to explore if there are better approaches to preserve color. Specifically, we will try the different linear color transformation techniques proposed by Gatys *et al*. [3].

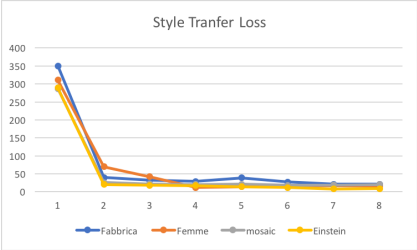

Figure 11. Example Style Transfer Loss

Figure 12. Top to bottom: validation images, transferred image(color preserved vs not preserved) and style (Weeping Woman by Pablo Picasso)

5.5. Overall Results

Now we combine the text generator and the style transferrer together. Figure 15 shows some validation set text descriptions. Figures 16, 17 and 18 show the corresponding synthetic photo-like images and various style transferred images.

Figure 13. Top to bottom: validation images, transferred image(color preserved vs not preserved) and style (Femme nue assise by Pablo Picasso)

Figure 14. Top to bottom: validation images, transferred image(color preserved vs not preserved) and style (Sketch on Google of Einstein)

a group of people walking across a beach flying kites near the ocean.
a bathroom with a large white sink and a door.
an old van has been given a mirrored finish
a plate of cheese bread next to bread sticks and wine.
a snowboarder sitting in the snow with his hand up

Figure 15. Some text description from validation set

6. Conclusion and Future Work

Given various input sentences and desired styles, we generated pretty decent photo-like images and artistic images. However, there are still some problems, as mentioned in the previous part 5.3.2. In our future work, we want to compare our current models with other creative modesl such as AttnGANs [17]. In recent works, some researchers demonstrate improvements on StackGAN by using VisDial dialogues along with MS COCO captions to generate images, as stated in the paper [12]. We may also run more experiments to find better hyper-parameters for our models. As for style transfer, we improved the performance of image transform net by substituting batch normalization with

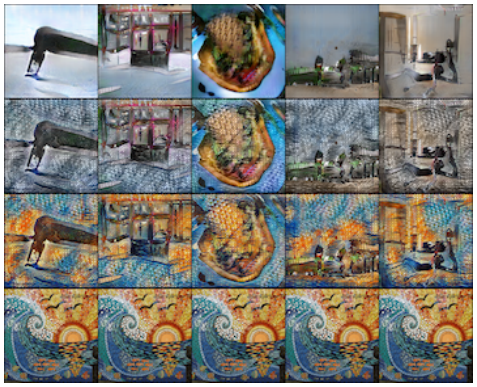

Figure 16. Top to bottom: validation images, transferred image(color preserved vs not preserved) and style (Mosaic painting on Google)

Figure 17. Top to bottom: validation images, transferred image(color preserved vs not preserved) and style (Fabbrica a Horta by Pablo Picasso)

Figure 18. Top to bottom: validation images, transferred image(color preserved vs not preserved) and style (Sketch on Google of Einstein)

instance normalization and we also provided a solution to preserving the original colors. As analyzed in section 5.4.2, as part of our future work, we want to find some even better approaches that not only preserve color but also solve the artifacts issue.

7. Contributions and Acknowledgements

7.1. Contributions

Yuxing Chen: Text-to-image generation using GANs
Zhefan Wang: Style transfer

- Collaborators: None
- Starter code:
  - Some of the checkpoint store/reload code are taken from Pytorch tutorial at `https://github.com/pytorch/examples/blob/master/imagenet/main.py`.
  - **No** code taken from assignment. Even for the style transfer system, we already implemented the prototype model one week before style transfer was discussed in class.
  - Code partially taken from Open-source projects (similar architecture): `https://github.com/hanzhanggit/StackGAN-Pytorch`.
  - The code for computing inception scores is partially (some python functions) adapted from: `https://github.com/tsc2017/inception-score`
  - Open-source projects that we've checked out: `https://github.com/cysmith/neural-style-tf`, `https://github.com/eveningglow/fast-style-transfer-pytorch`,

References

[1] C. Doersch. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*, 2016.

[2] L. Gatys, A. S. Ecker, and M. Bethge. Texture synthesis using convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 262–270, 2015.

[3] L. A. Gatys, M. Bethge, A. Hertzmann, and E. Shechtman. Preserving color in neural artistic style transfer. *arXiv preprint arXiv:1606.05897*, 2016.

[4] L. A. Gatys, A. S. Ecker, and M. Bethge. Image style transfer using convolutional neural networks. In *Computer Vision and Pattern Recognition (CVPR), 2016 IEEE Conference on*, pages 2414–2423. IEEE, 2016.

[5] J. Johnson, A. Alahi, and L. Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European Conference on Computer Vision*, 2016.

[6] A. B. L. Larsen, S. K. Sønderby, H. Larochelle, and O. Winther. Autoencoding beyond pixels using a learned similarity metric. *arXiv preprint arXiv:1512.09300*, 2015.

[7] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[8] A. Mahendran and A. Vedaldi. Understanding deep image representations by inverting them. 2015.

[9] S. Reed, Z. Akata, H. Lee, and B. Schiele. Learning deep representations of fine-grained visual descriptions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 49–58, 2016.

[10] S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee. Generative adversarial text to image synthesis. *arXiv preprint arXiv:1605.05396*, 2016.

[11] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved techniques for training gans. In *Advances in Neural Information Processing Systems*, pages 2234–2242, 2016.

[12] S. Sharma, D. Suhubdy, V. Michalski, S. E. Kahou, and Y. Bengio. Chatpainter: Improving text to image generation using dialogue. *arXiv preprint arXiv:1802.08216*, 2018.

[13] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[14] G. J. Somshubra Majumdar, Amlaan Bhoi. A comprehensive comparison between neural style transfer and universal style transfer. 2018.

[15] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826, 2016.

[16] D. Ulyanov, A. Vedaldi, and V. S. Lempitsky. Instance normalization: The missing ingredient for fast stylization. *CoRR*, abs/1607.08022, 2016.

[17] T. Xu, P. Zhang, Q. Huang, H. Zhang, Z. Gan, X. Huang, and X. He. Attngan: Fine-grained text to image generation with attentional generative adversarial networks. *arXiv preprint arXiv:1711.10485*, 2017.

[18] H. Zhang. Stackgan-pytorch. `https://github.com/hanzhanggit/StackGAN-Pytorch`, 2017.

[19] H. Zhang, T. Xu, H. Li, S. Zhang, X. Huang, X. Wang, and D. Metaxas. Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. In *IEEE Int. Conf. Comput. Vision (ICCV)*, pages 5907–5915, 2017.