

Text to Image Synthesis using Generative Adversarial Networks

Chockalingam Ravi Sundaram

cravis@umich.edu

Yijia Zhang

yijiaz@umich.edu

Fang Zhou

fzho@umich.edu

Abstract

The objective of our project is to train and optimize a Text to Image synthesizer model using Generative Adversarial Networks (GANs). Automatic synthesis of realistic images from text is an important and a challenging task as it involves translating visual concepts from characters to image pixels. Our project is an implementation of the paper "Generative Adversarial Text to Image Synthesis" by Reed et al. (Reed et al., 2016b). Our model involves a combination of RNN sequential model and Generative Adversarial Network (GAN) to map the text input to realistic images.

1 Introduction

Automatic synthesis of realistic images from text have become popular with deep convolutional and recurrent neural network architectures to aid in learning discriminative text feature representations. Recently, deep convolutional and recurrent networks for text have yielded highly discriminative and generalizable text representations learned automatically from words and characters.

In this project, we are interested in translating text in the form of a single sentence human written descriptions directly into image pixels. Examples of this single sentence human written descriptions include "a big bird with a white underbelly and black wings", "a bird shaped flower with large orange and purple petals protruding from it's light green and red petals", "a downward facing ring of pink petals surrounds a orange stamen in this flower" etc.

The problem of mapping a text to an image can be broadly classified into two tasks:

1. **Training a model that effectively captures the important information within a text:** Recent advances in sequential models show

that this task can be accomplished by training a RNN based language model. RNNs can effectively learn a text feature representation that captures the important visual details.

2. **Translating the text information to image:** This will be accomplished by a Conditional Generative Adversarial Neural Network (Goodfellow et al., 2014). Here the GAN is conditioned on the text descriptions instead of the class labels.

We have used the text embeddings used by the authors which are available at <https://github.com/reedscot/icml2016> Our implementation of the project differs from the paper (Reed et al., 2016b) in the following aspects:

- The paper does not specify the exact Generator architecture the authors used(like the number of conv transpose layers, stride, number of filters, kernel size, BatchNorm etc). So we experimented with these parameters resulting in different Generator architectures. We have shown the best qualitative results in this report from various Generator architectures.
- The paper does not specify the exact Discriminator architecture the authors used(like the number of conv layers, stride, number of filters, kernel size, BatchNorm etc). So we experimented with these parameters resulting in different Discriminator architectures. We have shown the best qualitative results in this report from various Discriminator architectures.

Despite these differences in the Generator and Discriminator architectures we believe that our resultant images are similar in quality to the resultant images in the paper (Reed et al., 2016b).

2 Related Work

Work on generative models of images typically addresses the problem of unsupervised learning of a data model which can generate samples from a latent representation. Prominent examples from this line of work are Restricted Boltzmann Machines (RBMs) (Hinton and Salakhutdinov, 2006) and Deep Boltzmann Machines (DBMs) (Salakhutdinov and Hinton, 2009).

With the advent of Convolutional Neural Networks (CNNs), many researchers exploited the capability of using deep convolutional decoder networks to generate realistic images. Dosovitskiy et al. (Dosovitskiy et al., 2015) for instance, trained a deconvolutional network to generate 3D chair renderings conditioned on shape, position and lighting. The Generative Adversarial Networks (GANs) presented by Goodfellow et al. (Goodfellow et al., 2014) also uses a deep convolutional decoder generator architecture except that it also has a Discriminator to determine whether the image generated by the Generator is real or fake.

Over the past few years, there has been a breakthrough in using recurrent neural network decoders to generate text descriptions conditioned on images. A classic example of this is the "Image Captioning Task" wherein the model reads in an image input and generates a text output depicting the image. "Show and Tell: A Neural Image Caption Generator" by Vinyals et al. (Vinyals et al., 2015) is a prominent paper on this field. Most of the Image Captioning papers typically use a Long Short-Term-Memory (LSTM) (Hochreiter and Schmidhuber, 1997) on the top-layer features of a deep convolutional network to generate captions. Our work is an exact opposite of this. We use a deep convolutional recurrent text encoders to condition the GAN to generate images.

3 Generative Adversarial Networks

Generative adversarial networks (GANs) consist of a generator G and a discriminator D that compete in a two- player minimax game: The discriminator tries to distinguish real training data from synthetic images, and the generator tries to fool the discriminator. Concretely, D and G play the following game on $V(D, G)$:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{data}(x)} [\log D(x)] + \mathbb{E}_{x \sim p_z(z)} [\log(1 - D(G(z)))]$$

(Goodfellow et al., 2014) proved that this minimax game has a global optimum precisely when $p_g = p_{data}$, and that under mild conditions p_g converges to p_{data} . Usually, samples from G are low in quality and are easily rejected by D (D is more dominant by G). It has been found to work better in practice for the Generator to maximize $\log(D(G(z)))$ than minimize $\log(1 - D(G(z)))$.

4 Text Embedding

To obtain visually-discriminative vector representation of text descriptions, we follow the approach by Reed et al. (Reed et al., 2016a) by using deep convolutional and recurrent text encoders that learn a correspondence function with images.

Reed et al. (Reed et al., 2016a) uses a RNN sequential model to encode the text description of an image. Typically character level RNN worked better as text encoders for this task as they were more robust. We have also used a character level RNN to model the text encoder. Let ψ denote the text encoder(char level RNN in our case). Let ϕ denote the image encoder(using a deep CNN). The paper uses a combination of these encoders to learn both the image encoders and text encoders using the following loss function:

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

where $\{(v_n, t_n, y_n) : n = 1, 2, \dots, N\}$ is the training dataset with Δ being the 0-1 loss. Here v_n are the images, t_n are the corresponding text descriptions, and y_n are the class labels. f_v and f_t are classifiers which are defined as follows:

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

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

The model is trained by backpropagating the loss and using the resultant gradients to learn ψ which can make it a discriminative text encoder.

5 Method

In this project, we used a hybrid character level convolutional-recurrent neural network to encode the text description as mentioned in (Reed et al., 2016a). We used these encoded text as a conditioning to the Deep Convolutional Generative Adversarial Networks (DC-GAN) (instead of class labels) to generate realistic images corresponding to the

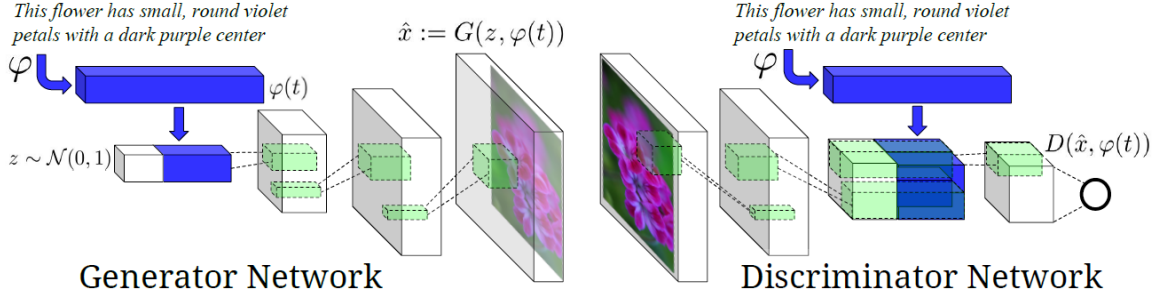


Figure 1: Our Model Architecture

text descriptions.

5.1 Model Architecture

The Neural Network architecture for our model is shown in Fig 1. We will be using the following notation. The Generator network is denoted by $G: \mathbb{R}^Z \times \mathbb{R}^T \rightarrow \mathbb{R}^D$. The Discriminator network is denoted by $D: \mathbb{R}^D \times \mathbb{R}^T \rightarrow \{0, 1\}$. Here T is the dimension of the text description embedding, D is the dimension of the image, and Z is the dimension of the noise input to G .

The Generator takes the text embedding and the random noise vector as an input and generates an image corresponding to the text information. The Discriminator determines whether the image generated by the Generator is real or fake.

As previously mentioned in Section 1, both the Generator and Discriminator architectures were not given by the authors of the paper (Reed et al., 2016b). We experimented on different set of Generator and Discriminator architectures by varying the number of conv layers, stride length, number of filters, kernel size, activation functions, adding BatchNorm layers, etc. The following architecture of the Generator and Discriminator was finalized and it produced images of similar quality as in (Reed et al., 2016b).

5.1.1 Generator Architecture

- **Input:** First, we sample a random Gaussian distributed noise $z \sim N(0, 1)$ and we encode the text query t using the text encoder ψ . This text encoder is a hybrid character level Convolutional-Recurrent Neural Network (mentioned in Section 4). Now the text embedding i.e. $\psi(t)$ is first compressed using a fully connected layer to a small dimension (we used 128) followed by a leaky-ReLU. This 128-dimensional compressed text embedding is concatenated with a 100-dimensional

noise vector to get a 228-dimensional input vector which is fed to the Generator model.

• Generator Model:

- **Layer 1:** Transpose Convolutional Layer(in_channels = 228, out_channels = 512, kernel_size = 4, stride = 1, padding = 0), followed by BatchNorm layer and ReLU activation
- **Layer 2:** Transpose Convolutional Layer(in_channels = 512, out_channels = 256, kernel_size = 4, stride = 2, padding = 1), followed by BatchNorm layer and ReLU activation
- **Layer 3:** Transpose Convolutional Layer(in_channels = 256, out_channels = 128, kernel_size = 4, stride = 2, padding = 1), followed by BatchNorm layer and ReLU activation
- **Layer 4:** Transpose Convolutional Layer(in_channels = 128, out_channels = 64, kernel_size = 4, stride = 2, padding = 1), followed by BatchNorm layer and ReLU activation
- **Layer 5:** Transpose Convolutional Layer(in_channels = 64, out_channels = 3, kernel_size = 4, stride = 2, padding = 1), followed by Tanh activation

- **Output:** A 64 x 64 color image

5.1.2 Discriminator Architecture

- **Input:** We encode the text query t using the text encoder ψ . This text encoder is a hybrid character level Convolutional-Recurrent Neural Network (mentioned in Section 4). Now the text embedding i.e. $\psi(t)$ is first compressed using a fully connected layer to a small dimension (we used 128) followed by a

leaky-ReLU. Apart from this, we also have a 64 x 64 x 3 image from the generator.

- **Discriminator Model:**

- **Discriminator Architecture 1:**

- * **Layer 1:** Convolutional Layer(in_channels = 3, out_channels = 64, kernel_size = 4, stride = 2, padding = 1), followed by Leaky ReLU activation(p = 0.2)

- * **Layer 2:** Convolutional Layer(in_channels = 64, out_channels = 128, kernel_size = 4, stride = 2, padding = 1), followed by BatchNorm layer Leaky ReLU activation(p = 0.2)

- * **Layer 3:** Convolutional Layer(in_channels = 128, out_channels = 256, kernel_size = 4, stride = 2, padding = 1), followed by BatchNorm layer Leaky ReLU activation(p = 0.2)

- * **Layer 2:** Convolutional Layer(in_channels = 256, out_channels = 512, kernel_size = 4, stride = 2, padding = 1), followed by BatchNorm layer Leaky ReLU activation(p = 0.2)

- The generated image from the Generator is passed through the Discriminator Architecture 1 and the output of this model is concatenated with the 128-dimensional compressed text embedding. This is then passed through Discriminator Architecture 2.

- **Discriminator Architecture 2:**

- * **Layer 1:** Convolutional Layer(in_channels = 640 (512+128), out_channels = 1, kernel_size = 4, stride = 1, padding = 0), followed by Sigmoid activation

- **Output:** A scalar value between 0(is fake) and 1(is real) indicating whether the input image to the Discriminator is real or fake.

5.2 GAN algorithm

This is a naive algorithm where a conditional GAN (c-GAN) is trained by viewing (text, image) pairs as joint observations and training the discriminator to judge pairs as real or fake.

In naive GAN, the discriminator observes two kinds of inputs: real images with matching text, and synthetic images with arbitrary text. Therefore, it must implicitly separate two sources of error: unrealistic images (for *any* text), and realistic images of the wrong class that mismatch the conditioning information.

Algorithm 1 Naive GAN training algorithm with step size α

```

1: Input: minibatch images  $x$ , matching text  $t$ ,
   number of training batch steps  $S$ 
2: for  $n = 1$  to  $S$  do
3:    $h \leftarrow \psi(t)$  ( Encode matching text)
4:    $z \sim N(0, 1)$  (Gaussian noise)
5:    $\hat{x} \leftarrow G(z, h)$  ( Generator Feed forward)
6:    $s_r \leftarrow D(x, h)$  ( real image, right text)
7:    $s_f \leftarrow D(\hat{x}, h)$  ( fake image, right text)
8:    $L_D \leftarrow \log(s_r) + \log(1 - s_f)$  ( Discriminator
   Loss)
9:    $D \leftarrow D - \alpha \partial L_D / \partial D$  (Update Discrimina-
   tor)
10:   $L_G \leftarrow \log(s_f)$  ( Generator Loss)
11:   $G \leftarrow G - \alpha \partial L_G / \partial G$  (Update Generator)
12: end for
```

Algorithm 1 summarizes the training procedure for naive GAN algorithm. After encoding the text, image and noise (lines 3-4), we generate the fake image (, line 5). s_r indicates the Discriminator score associated with a real image and right text(line 6), and s_f indicates the Discriminator score associated with a fake image and right text(line 7). We use $\partial L_G / \partial G$ and $\partial L_D / \partial D$ to denote the gradients of the objective functions of the Generator and Discriminator respectively. Lines 9 and 11 indicate the update step for the Generator and Discriminator.

5.3 GAN-CLS algorithm

In this project, we have implemented both naive GAN and GAN-CLS algorithm and have provided a comparison of the results of both the algorithm.

The novelty of GAN-CLS algorithm in comparison with the naive GAN is that this algorithm includes mismatched pairs(realistic images of the wrong class that mismatch the conditioning information) as a part of the training set. In GAN-CLS algorithm, we modify the GAN training algorithm to separate the unrealistic images and mismatched pairs. Thus, in addition to the real or fake inputs to the discriminator during training, we add a third

type of input consisting of real images with mismatched text, which the discriminator must learn to classify as fake. The advantage of the GAN-CLS algorithm is that this type of training enables the discriminator to optimize the image-text matching and this information is fed back to the generator through backpropagation.

Algorithm 2 GAN-CLS training algorithm with step size α

```

1: Input: minibatch images  $x$ , matching text  $t$ ,
   mis-matching text  $\hat{t}$ , number of training batch
   steps  $S$ 
2: for  $n = 1$  to  $S$  do
3:    $h \leftarrow \psi(t)$  ( Encode matching text)
4:    $\hat{h} \leftarrow \psi(\hat{t})$  ( Encode mis-matching text)
5:    $z \sim N(0, 1)$  (Gaussian noise)
6:    $\hat{x} \leftarrow G(z, h)$  ( Generator Feed forward)
7:    $s_r \leftarrow D(x, h)$  ( real image, right text)
8:    $s_f \leftarrow D(\hat{x}, h)$  ( fake image, right text)
9:    $s_w \leftarrow D(x, \hat{h})$  ( real image, wrong text)
10:   $L_D \leftarrow \log(s_r) + \log(1 - s_f) + \log(1 - s_w)$ 
    ( Discriminator Loss)
11:   $D \leftarrow D - \alpha \partial L_D / \partial D$  (Update Discriminator)
12:   $L_G \leftarrow \log(s_f)$  ( Generator Loss)
13:   $G \leftarrow G - \alpha \partial L_G / \partial G$  (Update Generator)
14: end for
```

Algorithm 2 summarizes the training procedure for GAN-CLS algorithm. After encoding the text, image and noise (lines 3-5), we generate the fake image (, line 6). s_r indicates the Discriminator score associated with a real image and right text(line 7), s_f indicates the Discriminator score associated with a fake image and right text(line 8), and s_w indicates the Discriminator score associated with a real image and wrong text(line 9). We use $\partial L_G / \partial G$ and $\partial L_D / \partial D$ to denote the gradients of the objective functions of the Generator and Discriminator respectively. Lines 11 and 13 indicate the update step for the Generator and Discriminator.

6 Experiments

6.1 Datasets

Our project was implemented using the following datasets:

6.1.1 Oxford-102

Oxford-102 Flowers dataset (Nilsback and Zisserman, 2008) consists of images from 102 flower categories. Each class consists of between 40 and 258 images. Each image has one associated label. The images have large scale, pose and light variations. In addition, there are categories that have large variations within the category and several very similar categories.

We split this dataset into class-disjoint training and test sets. Oxford-102 has 82 train+val sets and 20 test classes.

6.1.2 Caltech-UCSD Birds 200

Caltech-UCSD Birds 200 (CUB-200) (Welinder et al., 2010) dataset contains photos of about 200 bird species with the following information:

- Number of categories: 200
- Number of images: 6,033
- Annotations: Bounding Box, Rough Segmentation, Attributes

We split this dataset into class-disjoint training and test sets. CUB-200 has 150 train+val sets and 50 test classes.

6.2 Training and Implementation details

6.2.1 Text Encoder

For text features, we use the text embeddings provided by the authors which are available at <https://github.com/reedscot/icml2016>. For both CUB-200 and Oxford-102, a hybrid character level ConvNet with a recurrent neural network (CNN-RNN) as described in (Reed et al., 2016a) was used. The text encoder produces a 1024-dimensional embeddings that is projected to 128 dimensions in both the generator and discriminator.

6.2.2 GAN Model

For both naive GAN and GAN-CLS algorithms, we used the same Generator (refer 5.1.1) and Discriminator architectures (refer 5.1.2).

We trained the model using the Adam optimizer with a learning rate of 0.0002. The Loss function used was the Binary Cross Entropy Loss. The model was trained for 200 epochs with a batch size of 64.


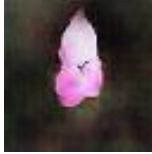
















Text Input	Naive GAN Output	GAN-CLS Output
a bird shaped flower with large orange and purple petals protruding from it's light green and red petals		
a cluster of flowers with pink petals and no visible stamens		
a downward facing ring of pink petals surrounds a orange stamen in this flower		
a flower with flat pink petals and central orange stamen cluster		
a flower with orange petals with black spots accompanied by black anther filaments		
the petals of the flower are long and skinny and have a center that is yellow in color		
the petals on this flower are purple with purple stamens		
this is a flower with yellow petals and brown anthers		
a flower with long and narrow petal that are pink		

Table 1: Comparison of the Generated images using GAN (refer Algo 1) and GAN-CLS (refer Algo 2) algorithms on Oxford-102 Flowers dataset (Nilsback and Zisserman, 2008)

Text Input	Naive GAN Output	GAN-CLS Output
a bird has bright white eyes, a short stubby bill, and a black crown		
a bird with a bright yellow belly and breast that have dark green lines going down them		
a brown bird with black eyes, a white belly, and a short beak		
the bird is brown from the rump to the ends of it's wingbars and the neck is solid white		
the bird is brown with a yellow flank and a black eyebrow and crooked bill		
the bird is small with a pointed bill, and the wings are brown		
the small bird has a striped chest, small pointed bill, and a black stripe across the eye area		
very colorful bird with black and yellow body with yellow feathers with black strips and crown black		
this woodpecker has a strong beak, black head, white throat, black wings, and its black body is spot		

Table 2: Comparison of the Generated images using GAN (refer Algo 1) and GAN-CLS (refer Algo 2) algorithms on CUB-200 Birds dataset (Welinder et al., 2010)

7 Results

Qualitative results can be seen in Table 1 and Table 2. From Table 1 and 2, it is clear that GAN-CLS algorithm provides better qualitative results than naive GAN algorithm.

Table 1 and Table 2 has three columns: text description of the image to be generated which is given as the input to the model, Image generated by the naive GAN model, and Image generated by the GAN-CLS model.

In the results on the Oxford-102 flowers dataset both naive GAN and GAN-CLS gets the color information right. But the GAN-CLS provides better quality images than naive GAN. The naive GAN tends to have the most variety in flower morphology, while the GAN-CLS tends to generate more class consistent images. This is exactly what is observed in CUB-200 birds dataset. Although both the naive GAN and GAN-CLS model gets the color information right, GAN-CLS produces better quality images.

8 Conclusion

In this project, we developed a simple and an effective model for generating images based on detailed visual descriptions. The text description in this dataset were complex and sometimes long. But our model was still able to extract the important descriptors from the text input (like color, shape etc) and produce images. We believe that with greater computational power and deeper CNN architectures, we can produce more realistic images with higher resolution.

Acknowledgments

We would like to acknowledge our course instructor Prof. Joyce Chai and the support of our GSI Cristian Paul Bara for their continued support. We would also like to thank the developers of PyTorch framework. We also like to thank Google Colab services for providing us with free GPU credits to run our codes.

References

Alexey Dosovitskiy, Jost Tobias Springenberg, and Thomas Brox. 2015. Learning to generate chairs with convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1538–1546.

- Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.
- Geoffrey E Hinton and Ruslan R Salakhutdinov. 2006. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507.
- Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.
- Maria-Elena Nilsback and Andrew Zisserman. 2008. Automated flower classification over a large number of classes. In *Indian Conference on Computer Vision, Graphics and Image Processing*.
- Scott Reed, Zeynep Akata, Honglak Lee, and Bernt Schiele. 2016a. Learning deep representations of fine-grained visual descriptions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 49–58.
- Scott Reed, Zeynep Akata, Xincheng Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee. 2016b. *Generative adversarial text to image synthesis*. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48, ICML’16*, pages 1060–1069. JMLR.org.
- Ruslan Salakhutdinov and Geoffrey Hinton. 2009. Deep boltzmann machines. In *Artificial intelligence and statistics*, pages 448–455.
- Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. 2015. Show and tell: A neural image caption generator. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3156–3164.
- P. Welinder, S. Branson, T. Mita, C. Wah, F. Schroff, S. Belongie, and P. Perona. 2010. Caltech-UCSD Birds 200. Technical Report CNS-TR-2010-001, California Institute of Technology.