Text-To-Image Synthesis

Jiin Nam, Raghav K. Singhal

Abstract

In recent years, there has been a great deal of work in generating high quality images using Generative Adversarial Networks. This work reproduces results from Zhang et al [3], and Scott et al. [1], which use a conditional Generative Adversarial Netork to condition generation of text on single line sentences to generate images matching the desciption in the text. Our experiments involve using different GAN architectures and their comparison.

1 Introduction

This work is based on a conditional Generative Adversarial Network where we convert short sentence describing an Image setting into an actual Image which resembles the description. This is a problem that has gained interest in the research community and still has a long way to go, the primary problems being:

- Inherent Uncertainity
  Every Text Description has several images corresponding to it, even Image to text suffers from this problem. So a scoring system has not been properly developed and we therefore mostly rely on human scoring.
- GANs
  Some fundamental questions about the dynamics of Generative Adversarial Networks remain unanswered. For instance, if and when does an equilibrium actually exist, and if we reach that equilibrium what does it say about the data/real distribution, $\mathbb{P}_r$m and the generator distribution, $\mathbb{P}_g$.

We do not address the second issue in this report but we attempt to tackle the first issue.

Now, Scott et al,[1], and Zhang et al, [3] attempt to solve the first issue by breaking it into two sub-problems,

1. How to represent the text descriptions on which the GAN is conditioned.

   Following [1], [2], and [3] we use deep convolutional and recurren text encoders that learn a correspondence function with images. These embedding learn important visual details of the images. And based on this approach, we plan to use these text embeddings directly to train a Generative Network conditioned on the text embedding.

2. Image synthesis

   In Zhang et al, [3] and Scott et al,[1], they condition both the generator and discriminator on the text embeddings and by creating a clever training methodlogy, they were able to generate realistic looking images correspongding to text descriptions.

In this project, we follow both Scott et al, [1] and Zhang et al, [3], and compare their methods and apply them on results.

2 Recent Works

In Scott et al,[1], the images generated are not high-resolution (64×64) and do not generalize well, even if we do super-resolution, it does not provide a substantial improvement in the image quality. Thus, a more recent work StackGAN by Zhang et al,[3], has built upon this work which uses a two-stage generative process, where the first stage generates a low-resolution image conditioned on the text embedding similar to [1] but the second stage Generative Adversarial Network is not just a layer doing super-resolution, but it is rather another GAN conditioned on the text-descriptions which generates higher-resolution (256×256) images.

The added advantage of a second conditioned Generative Adversarial Network is that it can add features that were missed in the earlier phase. Mere Super-resolution, by contrast will only marginally increase the quality of the existing image but will not by any chance add any extra-features from the text-description to the image.

StackGAN takes advatage of the two-stage process by letting the first-stage fill in the background details and the second-stage provide fine details from the text. So the image from the first stage is fed as input to the second-stage GAN. There have been other work on this, for instance Nguyen et al, [4], which uses a sampling approach, based on an approximate Langevin sampling method to generate images conditioned on text, but this requires an expensive and 'inefficient' sampling approach. So far, Zhang et al, [3], has provided the simplest and best high-resolution images conditioned on text descriptions.

3 Methods

3.1 GAN-CLS

The first part of the project follows, Scott et al, [1], and uses a DC-GAN as the Generative Adversarial Network architecture. We list the networks employed here:

1. Generative Adversarial Network

   These networks employ a generator, G and a discriminator/critic, D, where the generator learns to generate realistic looking images with random noise,z as input and discriminator tries to distingush between realistic and fake images. This is essentialy a two-player minimax game which is formulated as follows:

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

This game has reaches a Nash equilibrium when $p_{data} = p_g$.

2. Joint Embedding

   We follow Scott et al, [2], where they learn a vector embedding of text descriptions by using convolutional and recurennt text encoders to learn an embedding function with images. Let $\{v_i, t_i, y_i\}_{i=1}^N$ be the training dataset, v_i, t_i are images and text description respectively and y_i are class labels, and Δ is a $0-1$ loss, now we define the classifiers f_v and f_t as follows:

$$f_v(v) = \text{argmax}_y \mathbb{E}_{t\sim\mathcal{T}(y)}[\phi(v)^T\psi(t)]$$
$$f_t(t) = \text{argmax}_y \mathbb{E}_{v\sim\mathcal{V}(y)}[\phi(v)^T\psi(t)]$$

   where ϕ is the image encoder and ψ is the text encoder, we use a Hybrid Character-Level Convolutional-Recurrent Network, and $\mathcal{T}(y)$ and $\mathcal{V}(y)$ are the set of text and image descriptions for class y. The loss we optimze is as follows:

$$\frac{1}{N}\sum_{i=1}^{N}\Delta(y_i, f_v(v_i)) + \Delta(y_i, f_t(t_i))$$

These are the two componenets we employ in [1], and their network architecture and training algorithm are described below.

3.1.1 Model Architecture and Training

Here, we describe the conditional Generative Adversarial Network we use. Let $G : \mathbb{R}^Z \times \mathbb{R}^T \to \mathbb{R}^D$ be the generator, where Z, T, D are the dimensions of the latent space, text embedding and discriminator network respectively. Now, let $D : \mathbb{R}^D \times \mathbb{R}^T \to \{0,1\}$ be the discriminator. You can see the full network architecture in Figure 1:

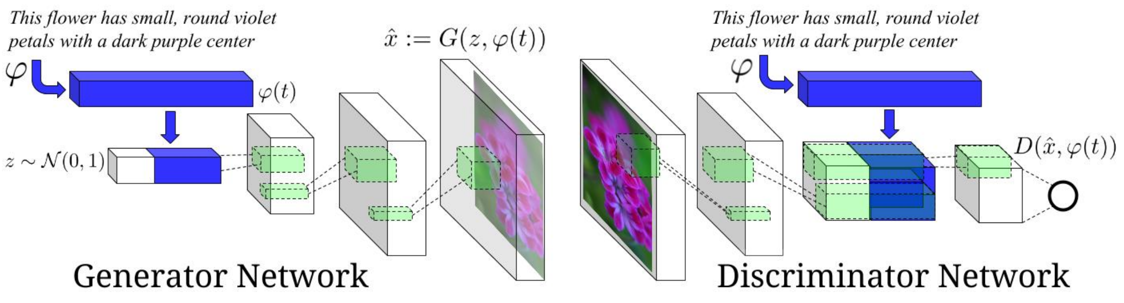

This flower has small, round violet petals with a dark purple center
φ
$\varphi(t)$
$z \sim \mathcal{N}(0,1)$
$\hat{x} := G(z, \varphi(t))$
Generator Network
This flower has small, round violet petals with a dark purple center
φ
$D(\hat{x}, \varphi(t))$
Discriminator Network

Figure 1: f

We then follow the usual procedure, except that we first encode the text using the text encoder ψ, and then compress $\phi(t)$ by passing it through a fully-connected linear layer to a smaller dimension and then a leaky-ReLU activation function. Now, we concatenate this embedding with the noise vector z, and then follow the usual procedure of a deconvolutional network. In the discriminator, we again perform a similar procedure where we compress $\phi(t)$ in a separate fully-connected layer followed by rectification.

Now, the method employed in [1] to train a conditional GAN is to view (text, image) pairs as joint observations and train the discriminator to judge pairs as fake or real.

There are several issues with this kind of an approach,

- Naive Approach
  The discriminator not only has no explicit notion of whether a pair is real or fake but also rejects most of the earlier samples as well as ignoring the conditional information. So the generator has to first learn to generate plausible looking images and then aligning them with the information provided and then the Discriminator must also learn to take the conditional information in to account to distinguish between real and fake pairs.

- Distinguishing plausible images with a mismatch with the Text
  This is a crucial part of this process, once the Generator has learned to generate plausible looking images but has not necessarily taken into account the conditional information, the Discriminator must distinguish this from cases where we have plausible looking images matching the text description.

To address the second problem we add a third kind of input, in addition to real and fake inputs we also add real images with mismatched text descriptions.

The training procedure is as follows:

```
**Data:** minibatch images $x$, matching text $t$, mismatching text $\hat{t}$, number of training batch
          steps $S$
**for** $t$ *in* 1 *to* $S$ *do* **do**
    $h \leftarrow \phi(t)$ ;
    $\hat{h} \leftarrow \phi(\hat{t})$ ;
    $z \sim \mathcal{N}(0, I_Z)$ ;
    $\hat{x} \leftarrow G(z, h)$ ;
    $s_r \leftarrow D(x, h)$ ;
    $s_w \leftarrow D(x, \hat{h})$ ;
    $s_f \leftarrow D(\hat{x}, h)$ ;
    $\mathcal{L}_D \leftarrow \log(s_r) + \frac{1}{2}(\log(1 - s_w) + \log(1 - s_f))$ ;
    $D \leftarrow D - \alpha \frac{\partial \mathcal{L}_D}{\partial D}$ ;
    $L\mathcal{L}_G \leftarrow \log(s_f)$ ;
    $G \leftarrow G - \alpha \frac{\partial \mathcal{L}_G}{\partial G}$
**end**
```

Where s_r denotes the score given to the real image and its corresponding sentence, and s_f

to the fake image with its corresponsing text, and s_w is the score of a real image with a arbitrary text.

3.2 StackGAN

Here, one of the primary differences in conditioning is that, we osberve that the text embedding $\phi(t)$ is usually quite high-dimensional, and due to the high-dimensional nature of the latent manifold concatenated with the text embedding, we require more data due to the curse of dimensionality. So to tackle this problem, Zhang et al, [3] introduce a conditioning augmentation technique. Note that they also use the mismatch technique introduced by Scott et al.,[1].

3.2.1 Conditioning Augmentation

In [3] rather than using only the text embedding $\phi(t) = c$, they produce additional conditioning text variable $\hat{c}$, by randomly sampling $\hat{c}$ from an independent Gaussian Distribution $\mathcal{N}(\mu(\phi(t)), \Sigma(\phi(t)))$, where the mean $\mu(\phi(t))$ and convariance matrix $\Sigma(\phi(t))$ are functions of the text embedding $\phi(t)$. Hence, providing us with more training pairs along with robustness random perturbations along the conditioning manifold. They also add a regularization term to the objective of the Generator in order to enforce smoothness and to avoid over-fitting:

$$D_{KL}(\mathcal{N}(\mu(\phi(t)), \Sigma(\phi(t)))\|\mathcal{N}(0, I))$$

3.2.2 Stage-I GAN

This is the first-stage of the two-stage network they build, in this stage that network supposedly fills in low-resolution details and the rest of the details are filled in by the second network, which takes the output of the first layer as an input.

In Stage-I, the network has the following loss functions associated with it

$$\mathcal{L}_{D_0} = \mathbb{E}_{(v,t)\sim p_{data}}[\log D_0(v, \phi(t))] + \mathbb{E}_{(z\sim\mathcal{N}(0,I), t\sim p_{data}}[\log(1 - D_0(G(z, \hat{c}), \phi(t)))]$$
$$\mathcal{L}_{G_0} = \mathbb{E}_{(z\sim\mathcal{N}(0,I), t\sim p_{data}}[\log(1 - D_0(G(z, \hat{c}), \phi(t)))] + \lambda D_{KL}(\mathcal{N}(\mu(\phi(t)), \Sigma(\phi(t)))\|\mathcal{N}(0, I))$$

Where, they use a trick called the reparametrization trick, so both $\mu(\phi(t)), \Sigma(\phi(t))$ are learned jointly with the network.

Model Architecture

In the Generator Network, we first get $\hat{c}$ be feeding te text embedding $\phi(t)$ into a fully-connected layer to generate μ, Σ and then sampling from a Gaussian Distribution with the above mean and Covariance Matrix. Then just like GAN-CLS we concatenate this with the noise-vector z, and then generate a $W_0 \times H_0$ sized image using a sequence of up-sampling blocks.

For the Discriminator, the text embedding $\phi(t)$ is compressed to N_d dimensions and then spatially replicated to form a $M_d \times M_d \times N_d$ tensor, and the image is also downsampled to $M_d \times M_d$ size. After concatenation, the tensor is fed into a 1×1 convolutional layer so that features can be jointly learned from both the image and the text, and then at the end a fully-connected layer with 1 unit is used to produce the final score.

Here, $W_0 = H_0 = 64$, and $M_d = 4$, and $N_d = 128$. So Stage-II GAN takes a 64×64 size image as input. We provide an Image of the network architecture in Figure 2.

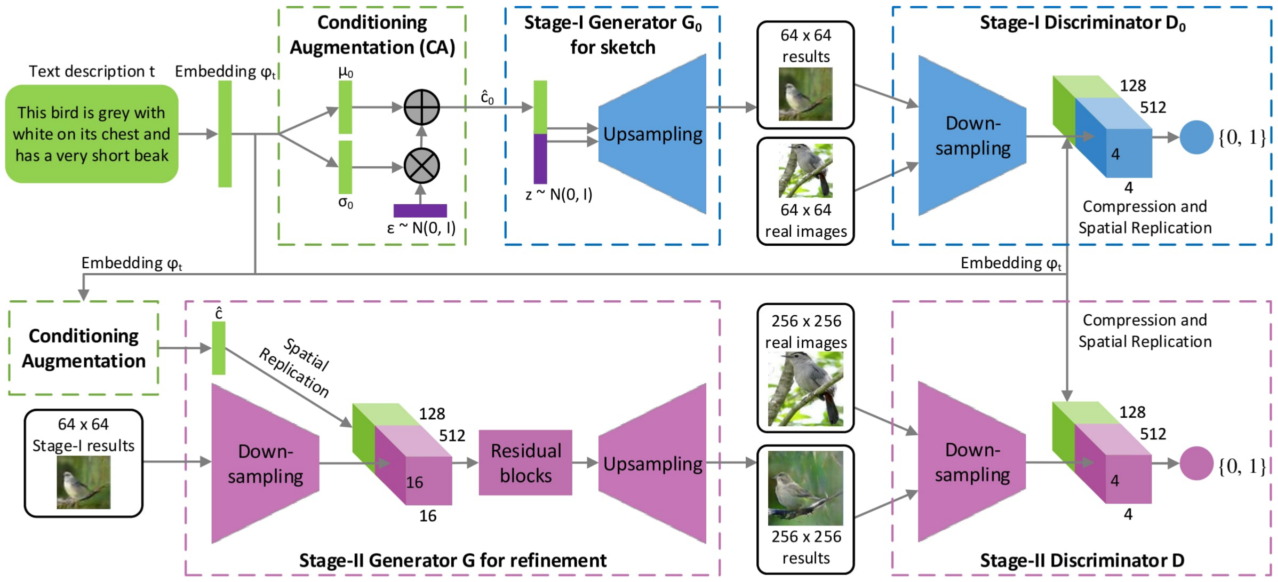


Figure 2:

3.2.3 Stage-II GAN

This is the final stage of the StackGAN process, where we take the low-resolution image from stage-I, which lacks higher order descriptions and might have missed some details from the text, and produce a higher-resolution image with more details added by also adding the conditional text information, which distinguished this from a purely super-resolution procedure.

Another distinction from traditional Generative Adversarial Networks, is that we do not feed noise z into the network, in fact we feed in the image x_0 into the network, as x_0 itself is a function of random noise z. Here we alternatively minimize $\mathcal{L}_{G_1}$ and maximize $\mathcal{L}_{D_1}$, which are as follows:

$$\mathcal{L}_{D_1} = \mathbb{E}_{(v,t)\sim p_{data}}[\log D_1(v, \phi(t))] + \mathbb{E}_{(x_0\sim p_{G_0}, t\sim p_{data}}[\log(1 - D_1(G_1(x_0, \hat{c}), \phi(t)))]$$
$$\mathcal{L}_{G_1} = \mathbb{E}_{(x_0\sim p_{G_0}, t\sim p_{data}}[\log(1 - D_1(G_1(x_0, \hat{c}), \phi(t)))] + \lambda D_{KL}(\mathcal{N}(\mu(\phi(t)), \Sigma(\phi(t)))\|\mathcal{N}(0, I))$$

Note, Zhang et al, [3] designed the Stage-II GAN as an encoder-decoder network.

Model Architecture

As noted above, this layer was designed as an encoder-decoder network and they added Residual Layers as well. Now, just as the previous stage, the text embedding $\phi(t)$ is used to generate a text conditioning vector $\hat{c}$ which is spatially replicated to a tensor of size $M_g \times M_g \times N_g$ and the image x_0 from Stage-I is downsampled till it has size $M_g \times M_g$. We then concatenate the image and text features, and then feed them into Residual Blocks which learn multi-modal features, that is both text and image features.

Finally for the decoding stage, we use up-sampling layers to generate a $W \times H$ sized image. And the Discriminator is trained along the same lines as Stage-I. Here, $W = H = 256$, and $M_g = 4$, and $N_g = 128$. So the final output of Stage-II GAN is a high-resolution 256×256 sized image.

4 Experiments & Results

4.1 GAN-CLS

In this set of experiments, we implemented GAN-CLS based on Scott et al [1], on the Oxford-102, flower dataset [5], which contains 8,189 images with 802 categories. However on top of the architecture mentioned in their paper, we also incorporate a few tricks from the paper Improved training of GANs [6](the paper was referred to us by Professor Rob Fergus).

After training, the network does learn to generate plausible looking images matching the text description, but it does not produce high-quality images and does not capture finer details rather it gets some background details like color correct, as well as understanding the meaning of words like stamen, bright, etc. We present some Images selected at random along with their text descriptions.

bright blue petals that are oval in shape surround numerous white stamen.

bright pink flower with big petals that expose a light pink receptacle.

flower has pink and white petals assembled as a star around a darker purple and white pistil.

layers of bright red oval shaped pedals surround golden yellow circle of stamen with predominant ant

a dull pink flower with green leaves attached to the flower

a flower with large blue petals and a yellow stamen in the middle

a bright pink heart shaped petaled flower set in a circular pattern with a bright yellow center.

We provide more images with their text descriptions as the name of the file in a separate folder.

4.2 MNIST

1. Fully connected layers
   The first GAN we present here consists of 4 sequential fully connected layers. Exponential Linear Unit(elu) used as a non-linear function for both G and D except tanh for last layer for G. For optimizer we chose Adam for both with learning rate 0.0002. Batch normalization and dropout has applied for G not D and the reason why will be explained the following subsection. The result looks like real numbers as shown in figure 4.

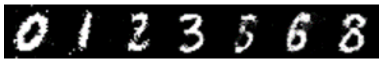


Figure 3: Generated images for MNIST

Layer	Type	# maps & neurons	kernel
0	input	100 neurons	
1	fully connected	256 neurons	1x1
2	fully connected	512 neurons	1x1
3	fully connected	1024 neurons	1x1
4	fully connected	784 neurons	1x1

Table 1: Network Architecture for Generator.

Layer	Type	# maps & neurons	kernel
0	input	784 neurons	
1	fully connected	1024 neurons	1x1
2	fully connected	512 neurons	1x1
3	fully connected	256 neurons	1x1
4	fully connected	1 neurons	1x1

Table 2: Network Architecture for Discriminator.

Without batch normalization, both G and D could not learn that well due to banish gradient problem. We first applied the techniques for G and it gave much better results as shown in figure 1. As a next step, we also applied this for D and from the certain epochs around 60s D's loss for generated data from G becomes 0 and loss for G became full and both stop learning. We realized that finding balanced ways for both to learn is important and G should be little better to make both to work. For the later work, we chose to use batch normalization for both and change optimizer for D to SGD to learn slowly than G.

4.3 CUB

Since we got reasonable result for MNIST data and get some intuition, we moved on to next step using CUB.

1. Fully connected layers
   We firstly applied the same network architecture as MNIST one and switch an optimizer to SGD for D with learning rate 0.01. It worked purely even after 800 epochs as shown in figure 4 since birds are much more complex than numbers.

2. Convolutional GAN
   Previous network architecture was not sufficient to learn birds as we expected. We switched it to Deep Constitutional Neural Network as described in detail in table 3 and 4. We referred to some comments from the StackGAN paper authors saying that LeakyRELU works good for both D and G and all experiments from now on we used LeakyRELU for non-linear layer. We added dropout for G to see if it improves performance, but G has tendency to generate clear images without dropout so we decided not to apply it.

   While doing research, we realized that there are some argue about ordering of batch normalization layer. We tested both and one applying batch normalization first generate slightly better images.

   We did experiments two different network architectures for birds and flowers. For birds, 2D nearest neighbor upsampling has applied and 2D transposed convolution operator has applied for flower to see difference. The conclusion for this is that upsampling version gives little more clearly images while convolution operator version gives grid shaped images but much faster.

   The final result of this is shown in figure 2.

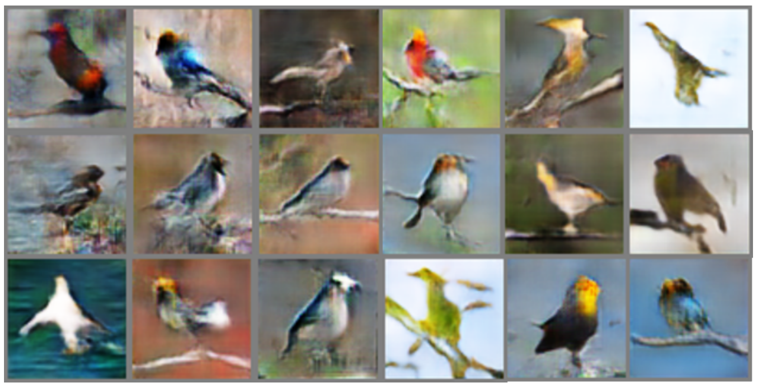


Figure 4: Generated images for CUB

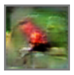
Layer	Type	# maps & neurons	kernel	padding
0	input	100 neurons		
1	deconvolutional	512 maps of 4x4 neurons	4x4	
2	upsampling	256 maps of 8x8 neurons		
3	convolutional	128 maps of 8x8 neurons	3x3	1
4	upsampling	128 maps of 16x16 neurons		
5	convolutional	128 maps of 16x16 neurons	3x3	1
6	upsampling	128 maps of 32x32 neurons		
7	convolutional	128 maps of 32x32 neurons	3x3	1
8	upsampling	128 maps of 64x64 neurons		
9	convolutional	128 maps of 64x64 neurons	3x3	1
10	convolutional	3 maps of 64x64 neurons		

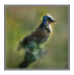
Table 3: Convolutional GAN Architecture for Generator.

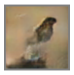
Layer	Type	# maps & neurons	kernel	stride	padding
0	input	3 maps of 64x64 neurons			
1	convolutional	64 maps of 32x32 neurons	4x4	2	1
2	convolutional	128 maps of 16x16 neurons	4x4	2	1
3	convolutional	256 maps of 8x8 neurons	4x4	2	1
4	convolutional	512 maps of 4x4 neurons	4x4	2	1
5	convolutional	1 neuron	4x4	2	

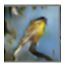
Table 4: Convolutional GAN Architecture for Discriminator.

3. Conditional GAN
Finally, we applied Conditional GAN to achieve our optimal goal to generate image from text. It is using the same network architecture except additional word embedding layers for bird descriptions described in table 5. It concatenated with 100 random values and used as an input for G and for D it concatenated with the output of 4th convolutional layer and added just one more convolutional layer than the previous one. The result looks like worse than previous one but it learns the descriptions and seems the G was trying to follow the descriptions. There are two things we have not tried. The first thing is to train wrong descriptions for D to make robust Discriminator. Secondly, we have only tried the first layer of StackGAN so generated only 64 by 64 imperfect images. In order to get clear and larger image, those we missed should be implemented.

 This bird is black with red on its wing and has a very short beak.

 A small bird with a red belly, blue cheeks and crown, green back, orange-rimmed black eyes, and a blunt, blue-green beak.

 Various brown tones from head to tail across the top of this bird, a white chest, short thighs with wide feet.

 A small bird that is brown and white with a vivid yellow throat,breast and belly.

Layer	Type	# maps & neurons	kernel
0	input	1024 neurons	
1	fully connected	512 neurons	1x1
2	fully connected	256 neurons	1x1

Table 5: Word embedding layers.

4.4 Future work

There are two things we have not tried. The first thing is to train wrong descriptions for D to make robust Discriminator. Secondly, we have only tried the first layer of StackGAN so generated only 64 by 64 imperfect images. In order to get clear and larger image, those we missed should be implemented.

5 Conclusion & Future Work

In this report, we try two different methods to generate Images from Text and can verify results mentioned in both papers as well a slight improvement from [1], as stated in [3], Stage-I GAN only generates 64×64 looking images with primitive features such as color and rudimentary shapes. However, generating images of flowers was much simpler compared to generating images of birds, we suspect it is because birds have finer visual features than flowers, as well as the text descriptions provided for birds were much longer than those for flowers, however it remains to be seen whether the later has any effect on the generation of images.

6 Acknowledgement

We would like to thank Professor Rob Fergus, for his advice and insights throughout the class and specifically for the project. We would also like to cite, the original paper repositories [7] and [8], as basic code which we expanded upon.

References

[1] Reed, Scott, et al. *Generative adversarial text to image synthesis.* arXiv preprint arXiv:1605.05396 (2016).

[2] Reed, Scott, et al. *Learning deep representations of fine-grained visual descriptions*. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2016.

[3] Zhang, Han, et al. *Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks.* arXiv preprint arXiv:1612.03242 (2016).

[4] Nguyen, Anh, et al. *Plug & play generative networks: Conditional iterative generation of images in latent space.* arXiv preprint arXiv:1612.00005 (2016).

[5] M.-E. Nilsback and A. Zisserman. *Automated flower classification over a large number of classes.* In ICCVGIP, 2008

[6] Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A. and Chen, X., 2016. *Improved techniques for training gans.* In Advances in Neural Information Processing Systems (pp. 2234-2242). Vancouver

[7] Original GAN-CLS Code .https://github.com/reedscot/icml2016

[8] Original StackGAN Code. https://github.com/hanzhanggit/StackGAN