Text to Image: Generating Images from text using a DC-GAN

M. Adeel Hassan
University of Massachusetts, Amherst
muhammadadee@umass.edu

Abstract

We tackle here the problem of synthesizing realistic looking images from textual descriptions using a Deeply Convolutional Generative Adversarial Network (DC-GAN). We use a dataset consisting of captioned images of flowers and an off-the-shelf sentence encoder. We achieve a generative model that is able to generate images that correctly render the color of the flower provided in the description, but not its other visual and structural features.

1. Introduction

The ability to convert images to text and vice versa comes naturally to humans. Presumably, we are able to encode signals from one representation into an abstract space and then render them in the form of a different representation. The process of converting images to text is known as captioning and is a well researched domain. The reverse process, converting text to images, is a much harder problem.

We explore here an attempt at this problem using the Oxford-102 dataset, consisting of pictures of flowers accompanied by captions, an off-the-shelf sentence encoder, and a Deeply Convolutional Generative Adversarial Network.

2. Related work

There have been impressive results in the domain of synthesizing images from text using Generative Adversarial Networks, starting with [11]. We understand StackGANs, introduced in [14], to be the state-of-the-art.

Our approach here is similar to [11] but simpler in many respects, and we achieve worse results.

3. Approach

3.1. Dataset

The dataset used is the Oxford-102 dataset[1], consisting of about 6000 pictures of various flowers, with 10 captions per image [10].

3.2. Encoder

The encoder we use for this project is the pre-trained encoder called InferSent, by Facebook Research [3] [12]. It takes any English sentence and embeds it into a 4096-length real vector.

3.3. Squeezenet

Squeezenet is a CNN that achieves AlexNet-level accuracy using fewer parameters [5].

We have built our Discriminator on top of a pre-trained Squeezenet. More specifically, our Discriminator takes as its input the output of the 3rd layer and the 6th layer of Squeezenet, instead of the raw image itself. This way, we avoid having to train our own initial convolutional layers.

3.4. Architecture

The overall GAN architecture is shown in Figure 1.

The learning pipeline goes like this:

1. the caption is encoded into an embedding
2. A noise vector of length 64 with each value sampled uniformly at random from $[-1, 1]$ is concatenated with embedding
3. This is fed into the generator, which outputs a fake image
4. The fake image is passed through the first 6 layers of Squeezenet [5]
5. The output of the 3rd layer and the 6th layer of Squeezenet is concatenated together
6. The above is then concatenated with the input embedding
7. This is fed into the discriminator which assigns a score to indicate whether the image is real or fake

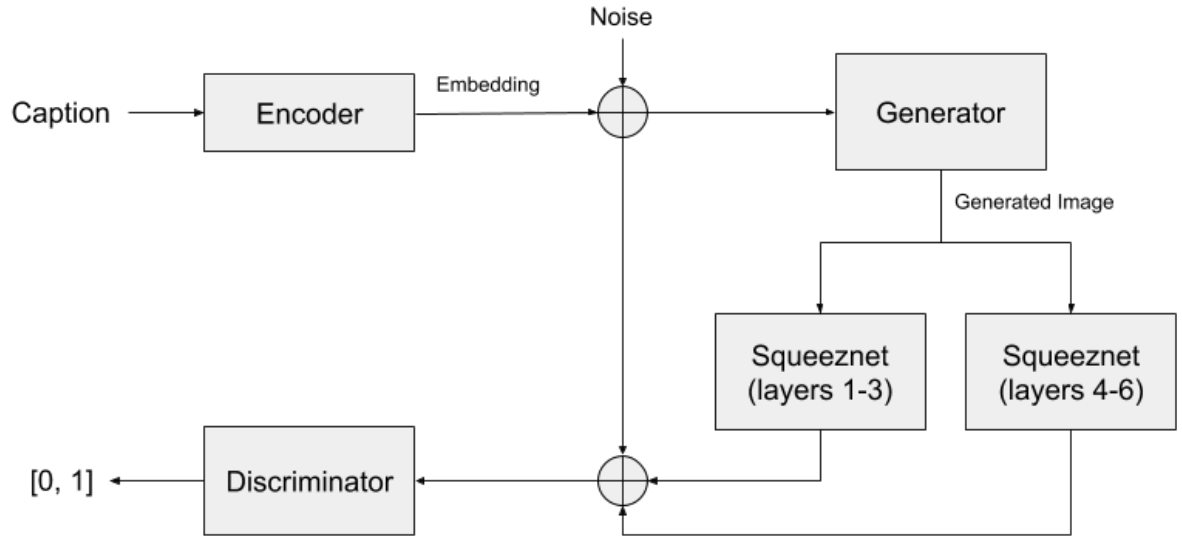

Figure 1. Overall GAN architecture.

3.4.1 Generator

The Generator architecture was as follows. (Some arguments omitted for clarity.)

```
Sequential(
    Linear(4096 + 64, 32*16*16),
    LeakyReLU(),
    BatchNorm1d(32*16*16),
    Unflatten((32, 16, 16)),
    Conv2d(32, 32, 5),
    LeakyReLU(),
    BatchNorm2d(32),
    ConvTranspose2d(32, 32, 4),
    LeakyReLU(),
    BatchNorm2d(32),
    Conv2d(32, 32, 5),
    LeakyReLU(),
    BatchNorm2d(32),
    Conv2d(32, 3, 5)
)
```

3.4.2 Discriminator

The Discriminator architecture was as follows. (Some arguments omitted for clarity.)

```
Sequential(
    Linear(3136+1152+4096+64, 1024),
    LeakyReLU(),
    BatchNorm1d(1024),
    Linear(1024, 128),
    LeakyReLU(),
    BatchNorm1d(128),
    Linear(128, 1)
)
```

3.4.3 Activation functions and batch-normalization

We use a Leaky ReLU activation followed by a Batch-normalization [6] layer after each linear or convolutional layer, except the ones at the end.

3.5. Loss function

We use the standard least squares loss [8] for the discriminator.

$$\ell_D = \frac{1}{2}\mathbb{E}_{x\sim P_{Data}}[(D(x)-1)^2] + \frac{1}{2}\mathbb{E}_{z\sim P(z)}[(D(G(z)))^2] \tag{1}$$

For the Generator, we also use the least squares loss, but with one modification: we add a new term which measures the difference in feature activations across the first 6 layers of Squeezenet for the fake image vs the same for the corresponding real image. This is similar to how style layer activations are compared in Style Transfer.

$$\ell_G = \frac{1}{2}\mathbb{E}_{z\sim P(z)}[(D(G(z))-1)^2] + F(M, G(z), y) \tag{2}$$

where y is the real image and $F(M, G(z), y)$ is sum of the mean squared distances between layer activations for input $G(z)$ and for y, for each layer f in model M.

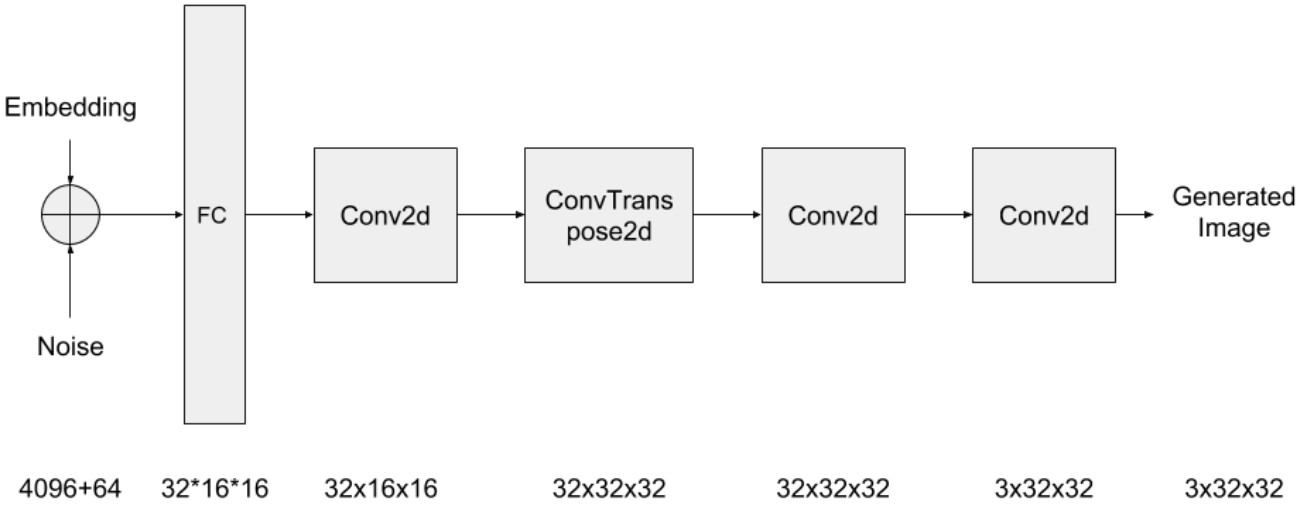

Figure 2. Generator architecture.

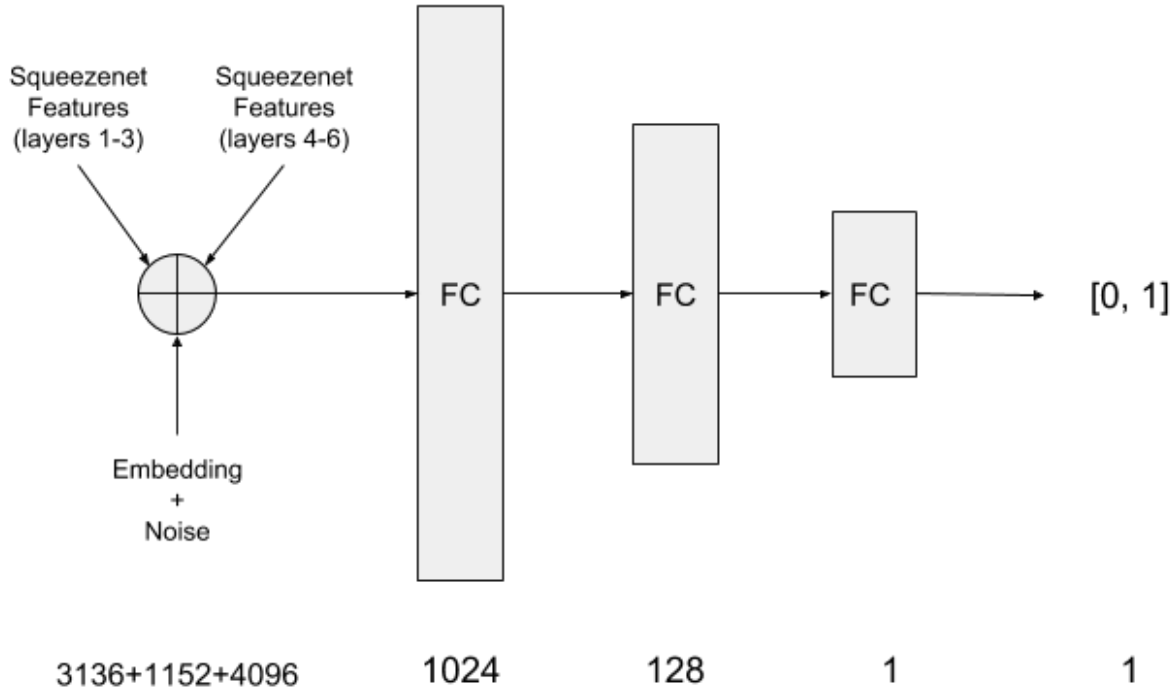

Figure 3. Discriminator architecture.

$$\sum_{f \in M} \mathbb{E}_{z \sim P(z)}[(f(G(z)) - f(y))^2] \qquad (3)$$

This is similar to one of the techniques suggested in [13].

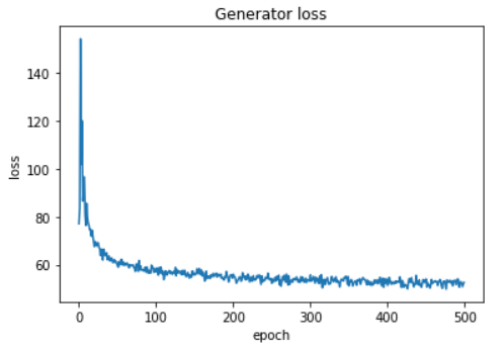

Figure 4. Generator loss during training.

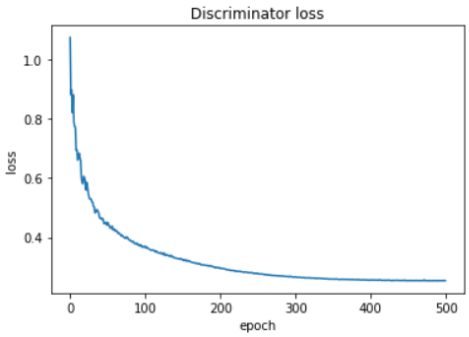

Figure 5. Discriminator loss during training.

3.6. Training

3.6.1 Weight initialization

We use the Xavier initialization [4] for all layers in the Generator and Discriminator. The squeezenet layers are not touched, of course, since they are pre-trained and are not affected during.

3.6.2 Optimizers

We use the Adam optimizer [7] with default values for beta, for both the Generator and the Discriminator.

3.6.3 Hyper-parameters

- Batch-size: 250
- Generator learning rate: 1e-2
- Generator L2 regularization weight: 0
- Discriminator learning rate: 1e-4
- Discriminator L2 regularization weight: 1e-6

We run the training for 500 epochs. The loss trends are shown Figures 4 and 5.

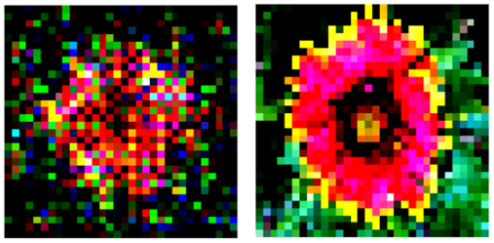

Figure 6. Input: "this flower is pink and yellow in color, and has petals that are yellow on the tips". Left: Generated. Right: Real.

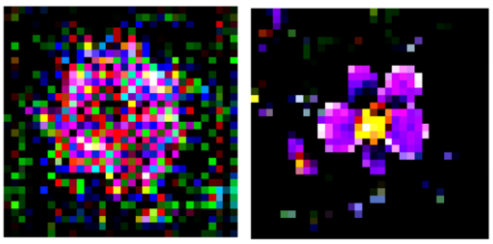

Figure 7. Input: "this flower has petals that are purple and has yellow stamen". Left: Generated. Right: Real.

4. Experiments

We tried a number of variations of our approach. Here we list some of them and their results:

- **Generator loss modification**. We found this to be crucial. Without it, the generator loss does not converge and we end up with output images which are basically just noise.
- **Adding noise**. It is standard for both GANs and conditional GANs [9] to take a noise vector as one of the inputs. We found this had little effect in our case. There was no appreciable difference in the outputs with or without noise.

5. Evaluation

Qualitatively, the generated images are not very impressive and are only faithful to the input texts to the extent of the color of the flower.

Some example outputs are shown in Figures 6, 7, and 8.

5.0.1 Inception score

Inception score is a popular (though imperfect [2]) method for measuring the quality of images generated by GANs.

The output images achieved an inception score of about 1.22. This was measured using a random sample of 250 generated images. This is much lower than the scores achieved by state-of-the-art solutions such as StackGANs [14], which achieve 3.20 on the same dataset.

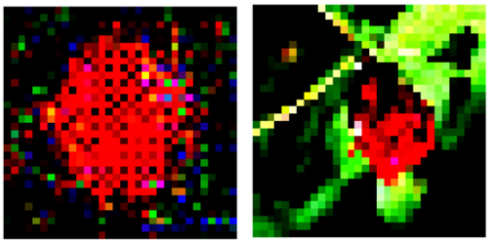

Figure 8. Input: "this flower is red in color, and has petals that are closely wrapped around the center". Left: Generated. Right: Real.

6. Conclusions

Looking at the generated images, it is clear that the model has only learned to correctly parse color from text and not anything else, such as structure or color patterns.

For every input text, the model outputs an image with a vaguely flower-like blob at the center, with approximately the right color.

These results are well below par, compared to the state-of-the-art for this domain.

We propose the following as the likely reasons for this:

- **The sentence embeddings**. While InferSent might be good at embedding sentences in general, it might not be good at taking sentences that are very similar (such as "this flower is red with short petals" and "this flower is red with long petals") and embedding them in sufficiently distant vectors. One significant finding of this project, then, is the unsuitability of InferSent as a sentence encoder for this task.

- **Image resizing**. All images were resized to 32x32 for training due to computational limitations. This would have resulted in a significant loss of detail.

References

[1] 102 category flower dataset.

[2] S. Barratt and R. Sharma. A Note on the Inception Score. *arXiv e-prints*, page arXiv:1801.01973, Jan. 2018.

[3] A. Conneau, D. Kiela, H. Schwenk, L. Barrault, and A. Bordes. Supervised learning of universal sentence representations from natural language inference data. In *EMNLP*, 2017.

[4] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[5] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and ¡1mb model size. *CoRR*, abs/1602.07360, 2016.

[6] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

[7] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[8] X. Mao, Q. Li, H. Xie, R. Y. K. Lau, Z. Wang, and S. P. Smolley. Least squares generative adversarial networks. *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2813–2821, 2017.

[9] M. Mirza and S. Osindero. Conditional generative adversarial nets. *CoRR*, abs/1411.1784, 2014.

[10] S. E. Reed, Z. Akata, B. Schiele, and H. Lee. Learning deep representations of fine-grained visual descriptions. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 49–58, 2016.

[11] S. E. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee. Generative adversarial text to image synthesis. In *ICML*, 2016.

[12] F. Research. Infersent.

[13] T. Salimans, I. J. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved techniques for training gans. In *NIPS*, 2016.

[14] H. Zhang, T. Xu, H. Li, S. Zhang, X. Huang, X. Wang, and D. N. Metaxas. Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 5908–5916, 2017.