Text2Image with deep convolutional GANs

A. Hermansson, P. Shi, G. Silvestri

KTH Royal Institute of Technology
DD2424 Deep Learning in Data Science
Source Code: https://github.com/yonkshi/text2imageNet

Abstract. In this project, we reproduce the paper Generative Adversarial Text to Image Synthesis. A deep convolutional generative adversarial network is conditioned on text, encoded by a character-level convolutional recurrent neural network, to produce realistic looking images. Even with limited training time, impressive results are still produced by the generative network.

Keywords: Generative models, GANs, Text-to-Image, CNN, RNN

1 Introduction

In this project we try to reproduce the results obtained by Reed et al. in [1]. The goal is to generate realistic images from a human-written description. Ideally, this functionality would allow to synthesize, with only natural language as input, unlimited amount of labeled data that can be used to train deep networks for supervised learning tasks. The description of the images is a single sentence containing visual information, for example “the petals of this flower are red and the anther are yellow”.

Previous approaches were usually capturing visual informations using attribute representations, like for example in [2]. However, these attributes are expensive to obtain and often require domain specific knowledge. In addition, in some cases the differences between different classes are subtle (for example for two species of flower with only slightly different colors), and the number of attributes to account for these details is considerably bigger. On the other hand, the use of natural language is a more intuitive interface that allows for easier visual description of objects. In recent years, the most successful deep architectures have shown great performance on text representation learned directly from words or characters [3]. However, mapping text directly to pixels presents two main challenges: learn a representation of text features that correctly model visual details, and use these features to generate images that look as similar as possible to real ones.

For a given text description there can be many possible matching pixel configuration. Thus, the problem is highly multi-modal, and can be addressed using Generative Adversarial Networks (GANs)[4]. In the GANs setting, a generator network generates synthetic data that look realistic, trying to “fool” a discriminator network, that has to correctly classify the data as synthetic or real.

In this project we first train a hybrid Char-CNN-RNN [5] as text encoder, and then use it as input for the GANs, with the goal of generating realistic images. In particular, the GANs implemented are the GAN-CLS from [1], and are trained using the dataset Oxford-102 Flowers with 10 text captions per image.

1.1 Outline

In the next section we review related work on generative models. In section 3, a background of the text encoder and the generative adversarial models are presented. Furthermore, in section 4 we describe our exact architecture for the Char-CNN-RNN and the GANs. Section 5 is a summary of our experimental results and finally, in section 6 we have our conclusions.

2 Related Work

The use of natural language to generate images requires to learn a shared representation across modalities. This challenge is efficiently solved in [5], where several architectures composed of both convolutional and recurrent neural networks are proposed and evaluated. In particular, the proposed models are trained end-to-end to align with the category specific content of images, acheiving remarkable performances on zero-shot text-based image retrieval.

These results are reused in [1], where the GANs are conditioned on the text embedding to generate realistic images directly from raw text. The authors propose a new method (GAN-CLS) to train the discriminator in order to also learn the right relation between text and image, and introduce a manifold interpolation regularizer for the GAN generator (GAN-INT) that improves the quality of generated samples. By combining the two methods they are able to generate $64{\times}64$ synthetic images that match with the text description, and can barely be distinguished from real images by humans.

The GANs were first introduced in [4] as a new framework for estimating generative models via an adversarial process, corresponding to a minimax two-player game. In [6] the generator and discriminator are standard convolutional decoder/encoder, but a stable architecture is presented, incorporating batch normalization to achieve excellent results in image synthesis.

Some interesting recent applications of GANs are in image super-resolution[7], image-to-image translation [8], and image inpainting [9]. An attempt to generate images from visual description is presented in [10], but the description relies on attributes.

Finally, recent methods have improved the results of the implemented paper, where the models learn generate high quality $128{\times}128$ images from text description [11], [12].

3 Background

To be able to generate images from text, there are multiple challenges. One of them is how the text should be represented to provide a rich enough description, but contain only what is salient. Another issue is the actual process of generating an image, and how to condition it on text. Just like in [5], we try to learn a good text embedding through supervised learning with a dataset $\mathcal{S} = \{(v_n, t_n, y_n)\}$, $n = 1, \ldots, N$ with images $v \in \mathcal{V}$, captions $t \in \mathcal{T}$ and class labels $y \in \mathcal{Y}$.

After doing so, we then follow [1] and build a generative adversarial network with a generator conditioned on the embedded text, and a discriminator that learns to classify images, also conditioned on embedded text.

3.1 Deep Representations of Visual Descriptions

The idea is to learn a mapping from a text to a vector space, such that important features are preserved and so that the dimensions are greatly reduced. It is important that the representation is rich enough so that synthesized images produced by conditioning on the encoded text have correct content.

To learn a good embedding for a text, one idea is to encode that text and a corresponding image into the same vector space and then use a compatibility function to measure the similarity between them.

The text encoder is a mapping $\varphi(t) : \mathcal{T} \to \mathcal{E}$, where $\mathcal{E}$ is the embedding space. The image encoder is similarly a mapping $\theta(v) : \mathcal{V} \to \mathcal{E}$, from image to embedding and the compatibility $F : \mathcal{V} \times \mathcal{T} \to \mathbb{R}$ can be chosen to be the inner product of these two [5]:

$$F(v, t) = \theta(v)^T \varphi(t) \tag{1}$$

Classifiers are defined by looking at the expected compatibility scores for all classes and choosing the one with the highest value:

$$f_v(v) = \arg\max_{y \in \mathcal{Y}} \mathbb{E}_{t \sim \mathcal{T}(y)}[F(v, t)] \tag{2}$$

$$f_t(t) = \arg\max_{y \in \mathcal{Y}} \mathbb{E}_{v \sim \mathcal{V}(y)}[F(v, t)] \tag{3}$$

Assuming that we keep the model for the image encoder fixed, the loss function can be defined just for the text encoder $\varphi(t)$ as:

$$\mathcal{L}_\varphi = \frac{1}{N} \sum_{n=1}^{N} \ell_v(v_n, t_n, y_n) + \ell_t(v_n, t_n, y_n) \tag{4}$$

$$\ell_v(v_n, t_n, y_n) = \sum_{y \neq y_n} \max(0, 1 + \mathbb{E}_{t \sim \mathcal{T}(y)}[F(v_n, t) - F(v_n, t_n)]) \tag{5}$$

$$\ell_t(v_n, t_n, y_n) = \sum_{y \neq y_n} \max(0, 1 + \mathbb{E}_{v \sim \mathcal{V}(y)}[F(v, t_n) - F(v_n, t_n)]) \tag{6}$$

The loss is set up so that the model will prefer to give higher compatibility scores for matching images and captions. It can be seen in equation (5), where the input image is held fixed, that when the score for the matching image and caption is higher than for the same image with captions from other classes, the contribution to the loss is reduced. The same applies in equation (6), but in this case the caption is instead held fixed.

3.2 Generative Adversarial Networks

In the framework of generative adversarial networks, a generative model G tries to fool a discriminative model D by synthesizing data points, and D has to classify them as real or generated. The discriminator D tries to maximize its classification accuracy, while G tries to minimize it by by creating more realistic samples. This minimax game can be described in terms of the value function $V(G,D)$

$$\min_G \max_D V(G,D) = \mathbb{E}_{x\sim p_{data}}[\log D(x)] + \mathbb{E}_{z\sim p_z}[\log[1-D(G(z))]] \tag{7}$$

where x is a real data point and z is noise sampled from a noise prior p_z for the generator to have something to start from. The two models are supposed to learn in unison by playing the above described minimax game,and the distribution of generated images p_g should converge to the real distribution p_{data} [4].

4 Method

In the following, the models and architectures implemented are described. In all the experiments we have used the dataset Oxford-102 Flowers, that contains 102 classes of flowers. The classes are split in 82 for training and validation set and 20 for test set.

4.1 Char-CNN-RNN

For the text embedding we implemented the architecture proposed in [5], where a recurrent network is stacked on top of a mid-level temporal CNN hidden layer. This architecture can take into account temporal dependencies along the input text sequence, while being fast and scalable for long sequences of character strings.

The input of the network is a single sentence describing the corresponding image. The considered alphabet consists of 70 different characters, and each character is represented with a one-hot encoding. The maximum number of characters allowed for each description is 201. Longer captions are cut, while shorter ones are zero-padded.

The convolutional network has three hidden layers. The number of filters is 384, 512 and 256 respectively for each layer. All the filters have width 4, stride 1 and ReLU activation, and the convolutions are always followed by max-pooling with width 3 and stride 3.

The output of the CNN (of size 8×256) is split along the time dimension (in this case 8) and used as input sequence of vectors to the RNN. The hidden layers have ReLu activations. The final encoded feature is the average over the hidden unit activations (size 256) over the sequence, $\varphi(t) = \frac{1}{L}\sum_{i=1}^{L} h_i$, with h_i the i-th hidden unit activation and L the sequence length. Finally $\varphi(t)$ is fed to a dense layer and mapped to a 1024 dimensional representation. An image of the architecture is shown in figure 1.

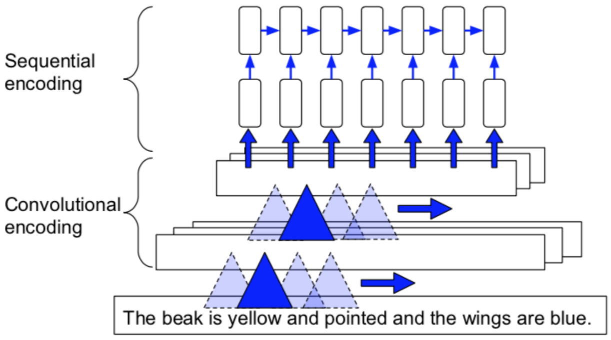


Fig. 1. Char-CNN-RNN architecture for text embedding. The picture is taken from [5].

4.2 Deep convolutional GANs

The GANs architecture is shown in figure 2, and was originally proposed in [6]. The input for the generator is a text description embedded with the pretrained Char-CNN-RNN described in 4.1, compressed using a fully-connected layer followed by leaky-ReLU to a 128 dimensional vector and concatenated to a 100 dimensional noise vector $z \in \mathbb{R}^z \sim \mathcal{N}(0, 1)$. The first layer is a dense layer that maps to $\mathbb{R}^{4\times4\times1024}$. The following three layers are deconvolutional layers with 512, 256 and 128 filters, all of size 4×4 with stride 2 in both width and height. After each layer we apply batch normalization and ReLU activation, apart from the last one where we used tanh activation. The output of the generator is a 64×64×3 image, and is used as input for the discriminator. In the descriminatior there are four convolutional layers with 128, 256, 512 and 1024 filters of size 4×4 with strides 2. Each layer is followed by batch normalization and leaky-ReLU activation. The text embedding $\varphi(t)$ is again compressed to 128 dimensions with a (separate) fully connected layer followed by rectification, and then replicated and concatenated to the 4×4 spatial dimension of the discriminator. An additional layer with 1024 filters 1×1 is used with batch normalization and leaky-ReLU,

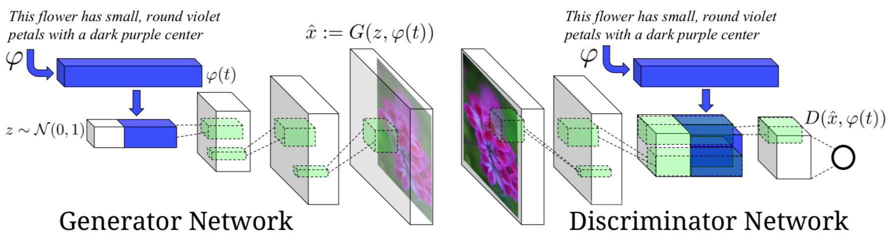

This flower has small, round violet petals with a dark purple center
φ
$\varphi(t)$
$z \sim \mathcal{N}(0,1)$
$\hat{x} := G(z, \varphi(t))$
Generator Network
This flower has small, round violet petals with a dark purple center
φ
$D(\hat{x}, \varphi(t))$
Discriminator Network

Fig. 2. GANs architecture. The picture is taken from [1].

followed by a last layer with single filter 4×4 and sigmoid activation to calculate a probability for a real image with a correct caption.

4.3 GAN-CLS

In the text-to-image scenario the discriminator needs to learn to distinguish real images from synthetic ones, but also if the image represents what is described in the text. Thus, during the training process, the discriminator receives three kind of inputs: fake image with right text, real image with right text, and real image with wrong text. In this way, the discriminator can provide an additional signal to the generator, since is learning to optimize both image realism and image-text matching. The discriminator makes a prediction for each kind of input, returning three probability scores: s_r for real-image/right-text, s_w for real-image/wrong-text, and s_f for fake-image/right-text. Finally, the loss is:

$$\mathcal{L}_D = \log(s_r) + (\log(1 - s_w) + \log(1 - s_f))/2. \tag{8}$$

For the generator the loss is

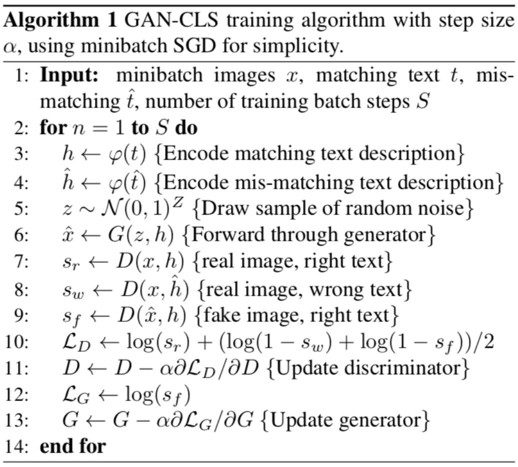
$$\mathcal{L}_G = \log(s_f). \tag{9}$$

The full training algorithm is shown in figure 3.

5 Experiments, training and results

5.1 Part I: Training Char-CNN-RNN against pretrained GoogLeNet

There are a total of four seperated networks in the system divided in two parts. The first part is pre-training the Char-CNN-RNN text encoder against an image encoder. To train the Char-CNN-RNN we extracted 1024 pooling units from a pre-trained GoogLeNet, these act as features in the embedding space $\mathcal{E}$. We feed a batch of images to GoogLeNet and its corresponding captions into the Char-CNN-RNN, the loss is then calculated from equation (4) and gradients are backpropagated to update the text encoder. The experiment was implemented



Algorithm 1 GAN-CLS training algorithm with step size α, using minibatch SGD for simplicity.

```
1: Input: minibatch images $x$, matching text $t$, mis-matching $\hat{t}$, number of training batch steps $S$
2: for $n = 1$ to $S$ do
3:     $h \leftarrow \varphi(t)$ {Encode matching text description}
4:     $\hat{h} \leftarrow \varphi(\hat{t})$ {Encode mis-matching text description}
5:     $z \sim \mathcal{N}(0,1)^Z$ {Draw sample of random noise}
6:     $\hat{x} \leftarrow G(z, h)$ {Forward through generator}
7:     $s_r \leftarrow D(x, h)$ {real image, right text}
8:     $s_w \leftarrow D(x, \hat{h})$ {real image, wrong text}
9:     $s_f \leftarrow D(\hat{x}, h)$ {fake image, right text}
10:    $\mathcal{L}_D \leftarrow \log(s_r) + (\log(1 - s_w) + \log(1 - s_f))/2$
11:    $D \leftarrow D - \alpha \partial \mathcal{L}_D / \partial D$ {Update discriminator}
12:    $\mathcal{L}_G \leftarrow \log(s_f)$
13:    $G \leftarrow G - \alpha \partial \mathcal{L}_G / \partial G$ {Update generator}
14: end for
```

Fig. 3. GAN-CLS training algorithm. The picture is taken from [1].

in Tensorflow, and to monitor the training we relied on the loss and accuracy plots.

After roughly 20,000 update steps, the text encoder converged at around 47% accuracy as seen in figure 4 which is similar to the results from [5].

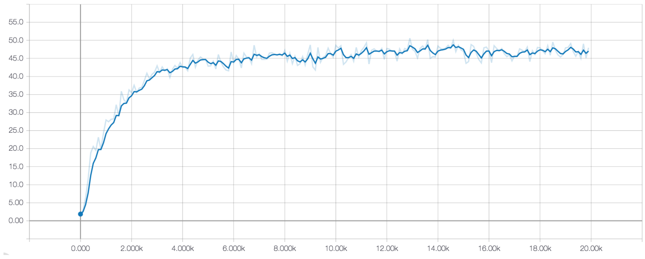


Fig. 4. encoder accuracy was able to converge to ∼47% in 20,000 epochs

5.2 Training GAN

The second part of the experiment is to build the GANs and connect the text encoder with the GANs. The code was implemented in Tensorflow. The original input data and captions were about 8000 images, 10 captions per image. We increase the data size by creating 10 sub images of 64x64 from the original image by cropping, flipping and resizing. In total, we had 800,000 samples to train from. In order to speed up the computation, we designed an efficient pipeline to pre-encode all text and images, and built a flexible multi-GPU implementation.

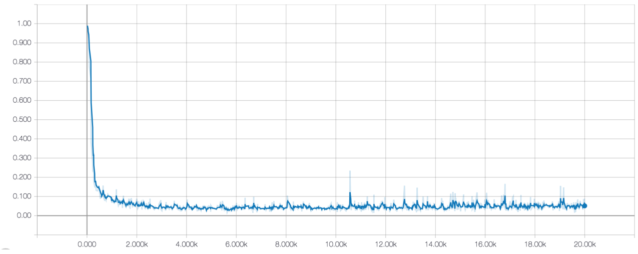


Fig. 5. encoder loss was steadily decreasing for 20,000 epochs.

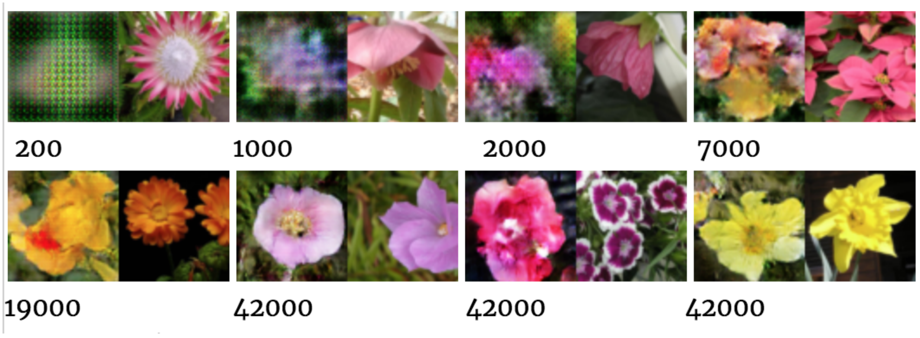


Fig. 6. Progression of 42,000 iterations of training. For each step, the left image is the one generated by the GANs, and the right image is real.

We performed approximately 90 runs, our longest and best experiments ran for 42,000 iterations, with the minibatch size 64, using ADAM optimizer and learning rate 0.0002, same as the original paper's author. The progression of results is visible in 6.

After the training we realized that the pre-trained encoder was not properly linked to the GANs architecture. This could be seen in several results where, even thought the flowers looked realistic, they were not matching the text description. With further investigation we found that the encoder was not working properly, and due to time constraints we could not solve the issue and retrain the network.

6 Conclusions

This project was a massive undertaking in both theoretical and practical knowledge. We thoroughly examined the concepts of text and image embedding, variations of Generative Adversarial Networks and the delicate balance between the generator and discriminator. We implemented three networks and utilized a fourth pre-trained GoogLeNet. We were able to achieve visible results even though the training of the text embedding was difficult to verify. We were impressed by the results of our GAN despite a possibly mistrained text encoder.

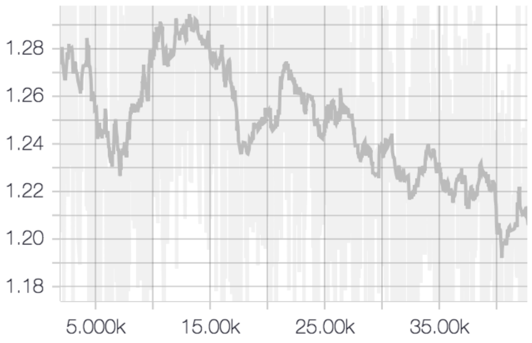


Fig. 7. Discriminator loss

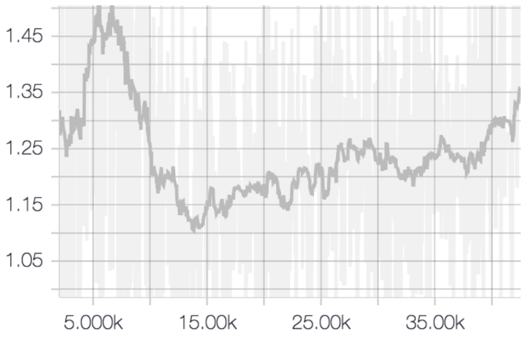


Fig. 8. Generator loss

The images did not always match the text descriptions, but a majority of them were highly realistic.

An obvious extension of this work would be to improve the quality of the text embedding. In addition, the extension GAN-INT proposed in [1] and its combination with the already implemented GAN-CLS can be tried in order to achieve better results. Finally, a study of the approaches presented in [12] and [11] could be done, with the goal to generate realistic images with higher resolution.

References

1. Reed, S., Akata, Z., Yan, X., Logeswaran, L., Schiele, B., Lee, H.: Generative adversarial text to image synthesis. arXiv preprint arXiv:1605.05396 (2016)
2. Akata, Z., Reed, S., Walter, D., Lee, H., Schiele, B.: Evaluation of output embeddings for fine-grained image classification. In: Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on, IEEE (2015) 2927–2936
3. Zhang, X., Zhao, J., LeCun, Y.: Character-level convolutional networks for text classification. In: Advances in neural information processing systems. (2015) 649–657

4. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. In: Advances in neural information processing systems. (2014) 2672–2680
5. Reed, S., Akata, Z., Lee, H., Schiele, B.: Learning deep representations of fine-grained visual descriptions. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. (2016) 49–58
6. Radford, A., Metz, L., Chintala, S.: Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv preprint arXiv:1511.06434 (2015)
7. Ledig, C., Theis, L., Huszár, F., Caballero, J., Cunningham, A., Acosta, A., Aitken, A., Tejani, A., Totz, J., Wang, Z.: Photo-realistic single image super-resolution using a generative adversarial network. arXiv preprint (2016)
8. Zhu, J.Y., Park, T., Isola, P., Efros, A.A.: Unpaired image-to-image translation using cycle-consistent adversarial networks. arXiv preprint arXiv:1703.10593 (2017)
9. Pathak, D., Krahenbuhl, P., Donahue, J., Darrell, T., Efros, A.A.: Context encoders: Feature learning by inpainting. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. (2016) 2536–2544
10. Yan, X., Yang, J., Sohn, K., Lee, H.: Attribute2image: Conditional image generation from visual attributes. In: European Conference on Computer Vision, Springer (2016) 776–791
11. Zhang, H., Xu, T., Li, H., Zhang, S., Huang, X., Wang, X., Metaxas, D.: Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. In: IEEE Int. Conf. Comput. Vision (ICCV). (2017) 5907–5915
12. Reed, S.E., Akata, Z., Mohan, S., Tenka, S., Schiele, B., Lee, H.: Learning what and where to draw. In: Advances in Neural Information Processing Systems. (2016) 217–225