Modernizing Generative Adversarial Text to Image Synthesis

Dhruv Saligram
UIUC
dhruvks2@illinois.edu

This project explores the task of modernizing the GAN-CLS architecture introduced in Generative Adversarial Text to Image Synthesis. The original implementation, written 9 years ago in Lua using Torch, is outdated in today's landscape. This is partially due to the language and framework used, but also because of the numerous "GAN tricks" that have been discovered and researched in recent years that help stabilize GAN training and boost output quality.

To address these limitations, this project reimplements the GAN-CLS architecture in PyTorch and systematically evaluates the impact of modern GAN stabilization techniques (including spectral normalization, one-sided label smoothing, and noise injection) and off-the-shelf text encoders on training and image generation quality. Three benchmark models were established by varying encoder complexity and training strategies:

- (1) Basic encoder with classic training
- (2) Basic encoder with updated training techniques
- (3) Fine-tuned encoder with updated training plus additional improvements

After training, these benchmark models produced the following outputs:

(1)

(2)

(3)

The major surprising finding was that the use of more advanced and fine-tuned encoders led to a complete loss of diversity in generations for the same prompt even with different initial noise. This finding was then supplemented by the quantitative evaluation through CLIP, where the basic encoder with updated training outperformed the fine-tuned encoder. These findings suggest that more powerful text encoders may cause overfitting, leading the generator to ignore noise input and produce deterministic outputs.

Ultimately, this project demonstrates the effectiveness of modernizing the original approach – both through language / framework and training techniques.

Introduction

Generative Adversarial Text to Image Synthesis, written by Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, & Honglak Lee in 2016, presented a novel deep learning architecture for utilizing GANs in text-to-image generation. The results were promising and highlighted a powerful potential use case for GANs. Operating on CUB (a dataset of bird images), Flowers 102 (a collection of images of flowers), and COCO (a large scale object detection dataset), the paper offered a comprehensive analysis of their system and showed that it could successfully take in sentence descriptions and produce corresponding images. The architecture used by the authors is illustrated in the following figure:

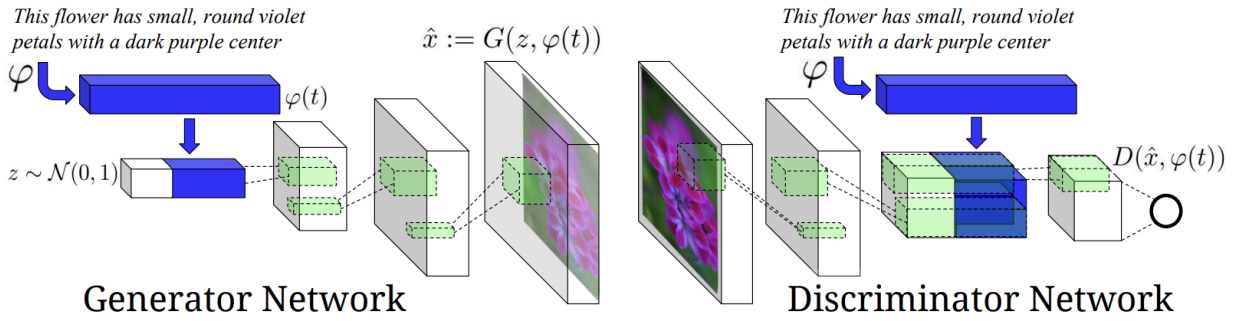


Figure 2. Our text-conditional convolutional GAN architecture. Text encoding $\varphi(t)$ is used by both generator and discriminator. It is projected to a lower-dimensions and depth concatenated with image feature maps for further stages of convolutional processing.

Given the powerful and proven nature of the framework, this project's initial direction was to implement it based on the discussion and pseudocode provided in the paper. However, after further exploration, it was found that the authors' code for implementing their approach was publicly available online and written in Torch with Lua. This adjusted the project's goal to reimplementing the approach in PyTorch using Python.

This "modernization" of the codebase from Lua to Python and Torch to PyTorch was primarily motivated by the emergence of PyTorch as a superior framework over the course of the last decade. PyTorch offers several substantial advantages over Torch, particularly in terms of popularity, ease of use, and integration capabilities, making it a preferred choice for modern machine learning research and development compared to Torch. By adopting Python as its primary language – as opposed to Torch's reliance on Lua – PyTorch experienced new levels of accessibility and popularity that Torch simply could not achieve. Furthermore, PyTorch's dynamic computational graph allows for greater flexibility and ease in debugging, prototyping, and implementing models. This stands in stark contrast to Torch's static graph paradigm. Finally, PyTorch enjoys significantly broader support than Torch which enables even more collaboration and ease of use. PyTorch represents an evolution over Torch in terms of usability, functionality, and keeping up with industry trends, which all contributed to the motivation of modernizing this codebase to use PyTorch.

While the project's new goal was reimplementation in PyTorch, it still felt like a bit of a missed opportunity. The 9 years that have passed since the paper's publication have led to more than just Lua and Torch becoming outdated. New "GAN tricks" have been found, researched, and extensively tested to make GAN training more stable and improve overall output quality. Additionally, PyTorch's ability to connect with other libraries through Python meant that the integration and testing of different text encoders would be exponentially easier in PyTorch than in Torch. As such, this project's definition shifted again to truly embrace "modernizing" the original approach. Motivated by the huge changes in both GAN

and system landscapes since the paper's publication, this project aims to truly modernize the original approach by faithfully reimplementing it in PyTorch and then applying newer techniques to improve output quality.

The approach taken in this project began with a line-by-line reimplementation of the original code base into PyTorch. Extreme care was taken to ensure that, to the fullest extent possible, the exact behavior of the original Torch code was replicated in PyTorch. Once this was completed and images were successfully generated, new techniques for improving GAN training were integrated into the project in an attempt to boost output quality. These techniques included adding spectral normalization for the discriminator, adding noise to the real images used in training for a set number of epochs early on, and using one-sided label smoothing. At the same time, an evaluation of text encoders was conducted in order to determine how powerful off-the-shelf or fine-tuned text encoders could be. The original code relied on a text encoder that was the product of another multi-thousand line code base written in Torch / Lua. By evaluating text encoders in Python, not only was their effectiveness being tested, but the ease of testing and integration was also highlighted. Ultimately, generators created from different training paradigms with different text encoders were qualitatively evaluated against each other and the original approach's output. An extensive analysis was done into the changes made and where they succeeded or failed. Finally, the three best generators were selected for quantitative evaluation using mean CLIP scores.

Details of Approach

The initial phase of this project focused on evaluating and selecting an appropriate text encoder. Three different off-the-shelf SentenceTransformer models were analyzed: multi-qa-mpnet-base-dot-v1, all-MiniLM-L6-v2, and all-mpnet-base-v2. For each model, caption embeddings were generated for all five captions associated with every image in the Oxford Flowers102 dataset. To assess the semantic structure captured by the embeddings, a logistic regression classifier was trained to predict the corresponding flower class based solely on the embedded captions. Further, to improve class separability, each base encoder was fine-tuned using a supervised classification head for multiple epochs.

The evaluation revealed that the multi-qa-mpnet-base-dot-v1 model exhibited the best performance. It achieved a classification accuracy of 37.32% with a logistic regression classifier on the raw embeddings and improved to 42.27% after 10 epochs of fine-tuning. Interestingly, extending fine-tuning to 30 epochs degraded performance, suggesting overfitting. As a result, the 10-epoch fine-tuned multi-qa-mpnet-base-dot-v1 was selected as the "fine-tuned" text encoder for subsequent experiments. The unmodified version of this encoder is referred to as the "advanced" model. In contrast, the lightweight all-MiniLM-L6-v2 encoder performed the worst and was designated as the "basic" model for baseline evaluations. This was done under the belief that the basic and lightweight model would perform the worst, which would allow for an evaluation on how the initial regression loss corresponded to output quality.

After selecting the encoders, the data loading stage involved generating and saving all caption embeddings to disk using the chosen text encoder. Simultaneously, all flower images were resized and saved as PyTorch tensors. This preprocessing step incurred a one-time cost but yielded over a 10× speedup in data loading during training, compared to opening and resizing image files during each epoch's training loop.

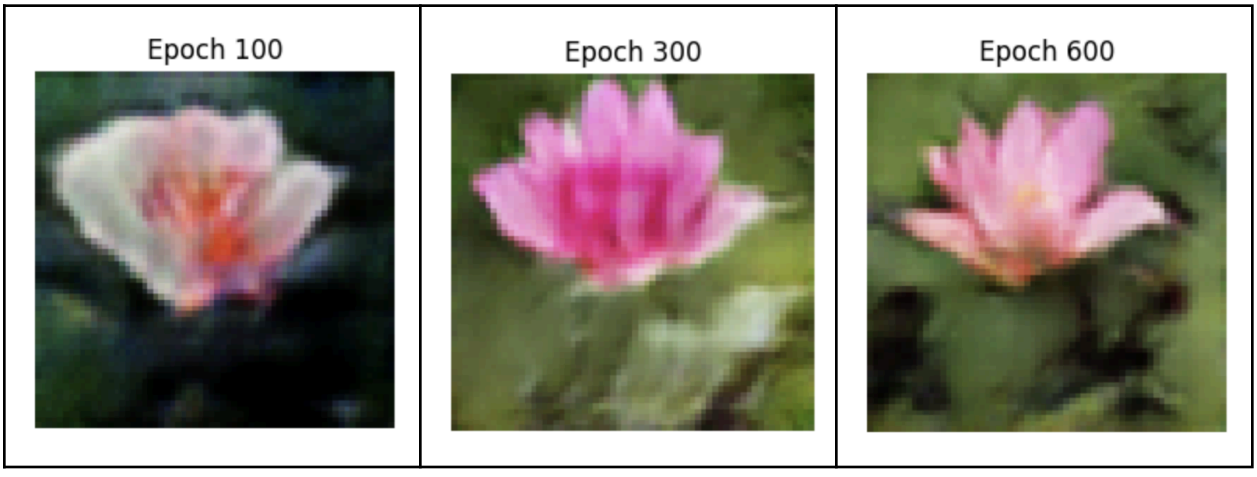
With the data prepared and the encoders selected, the next step was reimplementing the GAN-CLS architecture in order to begin initial training and generation experiments. The original paper provided the following pseudocode which outlined the training algorithm:

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

1: **Input:** minibatch images x, matching text t, mismatching $\hat{t}$, 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**

Upon observing the actual code base, however, it became apparent that the code was using real image and wrong image pairs as opposed to right text and wrong text pairs (as outlined in lines 3 & 4 of the pseudocode). In order to maintain the candid effort to exactly recreate the original architecture, the codebase's example was followed.

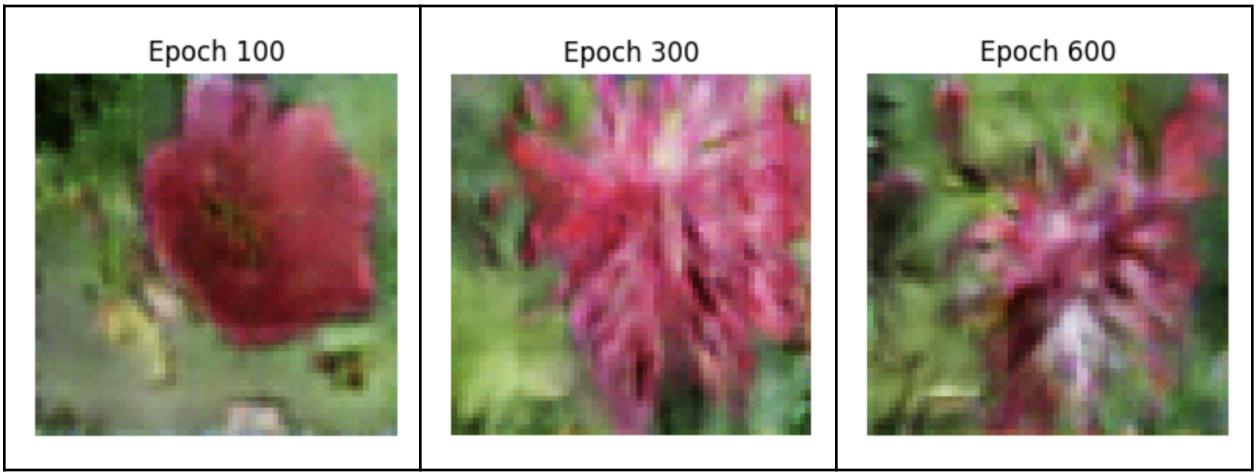
To faithfully replicate the behavior of the original Torch implementation, an almost line-by-line reimplementation was carried out in PyTorch. This included manually recreating specific Lua Torch constructs such as ConcatTable and CAddTable in the generator, and the Replicate module in the discriminator. Once the generator, discriminator, and closure definitions were reimplemented, training was ready to begin. Before training, however, a fixed initial noise and caption embedding were selected to be fed into the generator at the end of each epoch. This was done so that the generator's output on the same exact caption and noise could be visualized each epoch so as to enable direct visual tracking of model progression. The figures below illustrate how the model progressed on the same caption & noise across the training loop.



Ultimately, and as the images show, the reimplementation yielded a functioning PyTorch version of the GAN-CLS architecture that generated visually coherent flower images using the basic text encoder.

After establishing a reliable baseline, the next objective was to improve generation quality through modern GAN techniques, referred to as the “updated” training paradigm. This setup incorporated three key modifications: spectral normalization on the discriminator to stabilize training, the injection of noise into real images during early epochs to improve robustness, and the use of one-sided label smoothing to prevent discriminator overconfidence. These modifications were sourced from established GAN training best practices compiled in online repositories and dedicated research papers. Additional experimental variations were tested, such as performing two generator updates for every discriminator update and employing a lower learning rate for the discriminator. However, these variants consistently degraded performance and were ultimately discarded. This updated training paradigm was also run with the embeddings generated from the fine-tuned encoder in an effort to boost training and data quality.

An interesting feature of the updated training paradigm was slower convergence, which was likely due to the additional input noise during early epochs. While the classic training reached consistently sharp and clear shape outlines around epoch 50, it took the updated training around 90 epochs to achieve similar results. However, the updated training eventually generated high quality outputs.

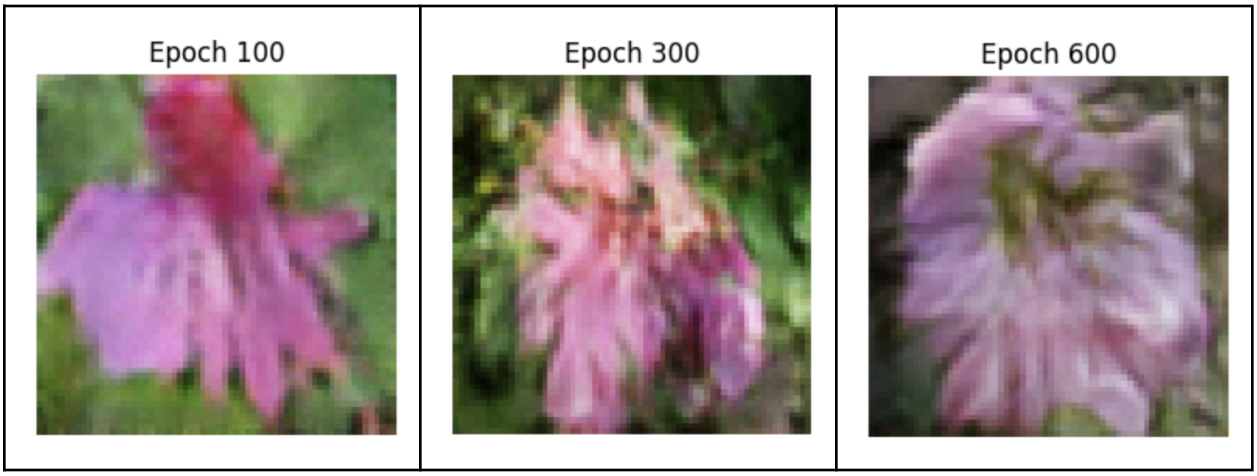


Despite this high quality output during training, test-time image generation revealed a significant shortcoming: the model exhibited very limited diversity, producing nearly identical images regardless of the sampled noise vector. The sample generation for each model would give the model the same prompt X times with different initial noise states each time. This would naturally result in varied outputs that all shared a connection to the base prompt. With this updated training using the fine-tuned encoder, however, the final generations appeared as follows:

This clear lack of diversity was traced back to the fine-tuned text encoder. Because its embeddings were highly class-separable, the generator appeared to over-rely on text embeddings at the expense of the noise vector. This led to overly deterministic outputs that ignored the intended stochasticity

of GANs. To address this, a third variant, termed the “updated plus” training paradigm, was developed. It introduced Gaussian noise into the text embeddings and added an additional diversity loss term to the generator objective. This loss explicitly encouraged output variance for identical text embeddings with different noise vectors, incentivizing the model to better utilize the stochastic input.

The generations during the updated plus training are illustrated below:



The updated plus paradigm produced visually strong outputs, and image fidelity remained high. Interestingly, the overall image quality actually seemed to be much higher and the training generations looked far more photorealistic than any other training setup. However, the diversity issue persisted during generations. While the model could produce compelling flowers from a given caption, it continued to struggle with generating diverse variants of the same caption. This confirmed that, while embedding regularization helped improve overall quality, the highly clustered nature of fine-tuned embeddings continued to constrain diversity. An example output was:

This again reinforced that the fine-tuned encoder was depriving the generator from diversity, but it also highlighted that the updated plus training paradigm could produce very high quality output.

To isolate the effects of encoder sophistication from training methodology and to confirm that the updated training paradigm actually could improve output, it was re-run using the basic text encoder. This allowed a direct comparison across three settings: (1) basic encoder with classic training, (2) basic encoder with updated training, and (3) fine-tuned encoder with updated plus training. This comparison framework enabled a rigorous evaluation of both architectural improvements and encoder sophistication. It also highlighted an important tradeoff: while more advanced encoders improved semantic fidelity, they could inadvertently reduce generative diversity by collapsing variation in the embedding space.

Results

To evaluate the effectiveness of the various training paradigms and text encoders explored in this project, a quantitative analysis was conducted across the three benchmarked models previously mentioned. The goal of this analysis was to determine which combination of encoder and training approach yielded the most successful text-to-image synthesis results.

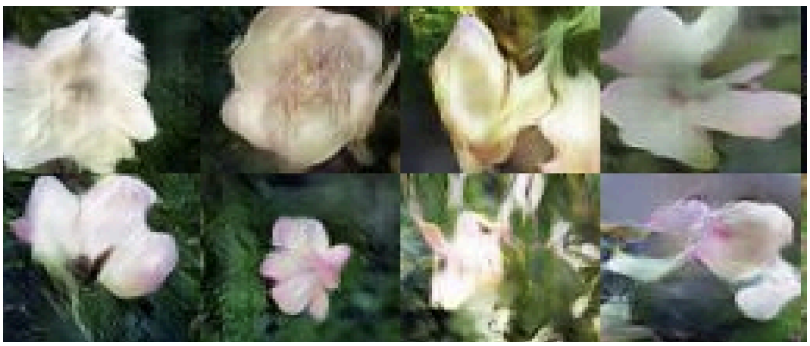
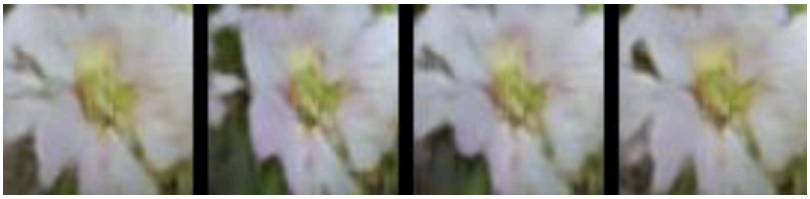
Given that the primary objective of the system is to generate images that are semantically aligned with input text descriptions, CLIP-based scoring was selected as the most appropriate quantitative evaluation metric. The cosine similarity between the embedded caption and generated image was believed to provide a meaningful measure of alignment between captions and generated images, leading to CLIP's use.

For each of the three benchmark models, images were generated for approximately 2,000 captions from the test dataset. The average CLIP score across all generated samples was then computed for each model. The basic encoder with classic training yielded an average CLIP score of 0.2874. When the same encoder was used with the updated training paradigm, this score improved to 0.2882, indicating a clear benefit from the addition of modern GAN training techniques. Interestingly, the fine-tuned encoder combined with the updated plus training paradigm achieved a slightly lower average CLIP score of 0.2869.

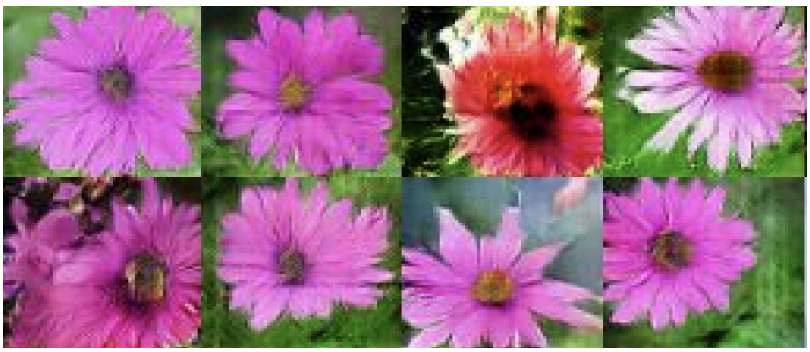
These results suggest a few things. First, using the updated training strategy seems to enhance semantic alignment. Second, increasing encoder complexity does not necessarily lead to better alignment as measured by CLIP. Overall, the basic encoder paired with updated training offered the most favorable balance between fidelity and generalizability, as evidenced by its superior quantitative performance. However, purely based on qualitative evaluation, the fine-tuned updated plus model seems to produce the highest fidelity individual images.

Below are two example outputs from each of the 3 benchmarked models compared to results from the original paper. More can be found in the appendix.

Ground Truth	this flower is white and pink in color, with petals that have veins	

Original GAN-CLS	
Basic text encoder with classic training	
Basic text encoder with updated training	
Fine-tuned text encoder with updated plus training	

Ground Truth	a flower with long pink petals and raised orange stamen	

Original GAN-CLS	
Basic text encoder with classic training	
Basic text encoder with updated training	
Fine-tuned text encoder with updated plus training	

Discussion & Conclusion

One of the more surprising findings was that the fine-tuned encoder combined with the updated plus training paradigm yielded a slightly lower average CLIP score than the basic encoder with classic training. Despite this, it is important to emphasize that all three benchmarked models produced relatively high CLIP scores, indicating that each approach was capable of generating high-quality images that exhibited meaningful alignment with their corresponding textual descriptions.

Among the three models, the combination of the basic encoder with the updated training paradigm achieved the highest average CLIP score. This outcome provides strong empirical support for the central thesis of this work: that modernizing the original GAN-CLS architecture with more recent training strategies and techniques leads to a tangible improvement in output quality. The incorporation of well-established GAN stabilization methods – such as spectral normalization, label smoothing, and noise injection – was instrumental in achieving this gain. Furthermore, reimplementing the original Lua-based Torch architecture in PyTorch introduced a host of practical benefits. These included more efficient data loading, seamless support for off-the-shelf models, and easier extensibility.

Although the fine-tuned text encoder did not lead to improved generative quality in the form of higher CLIP scores, its inclusion still underscored a key strength of the modernized approach. In contrast to the original paper, which relied on a specialized text encoder produced as part of a separate research effort and implemented in a large Torch/Lua codebase, this project was able to evaluate multiple readily available SentenceTransformer-based encoders with minimal overhead. This flexibility not only enabled more comprehensive experimentation but also demonstrated the broader utility and accessibility of the updated codebase.

In summary, the project led to three key conclusions. First, modernizing the original approach – both in terms of architecture (GAN tricks) and framework (Torch to PyTorch) – provided clear advantages in output quality. Second, incorporating newer advances in GAN training enhanced output quality. Finally, careful consideration should be given to the choice and integration of text encoders. Overly strong or fine-tuned encoders seemed to cause the generator to overfit to textual information and ignore noise vectors, thereby removing all output diversity.

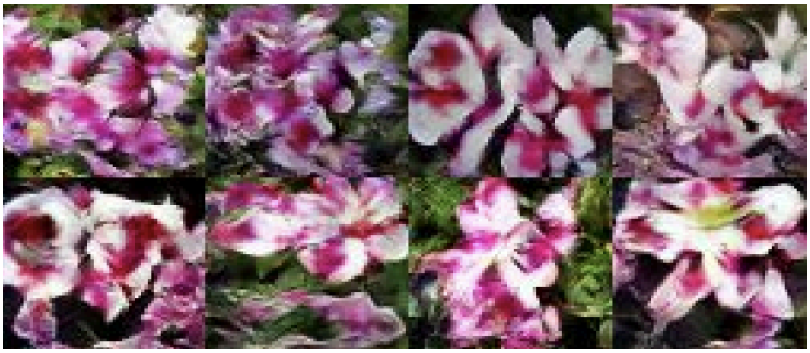
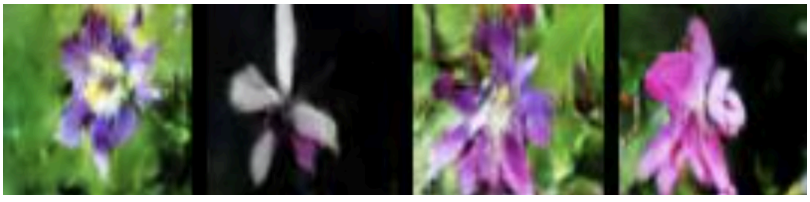
References

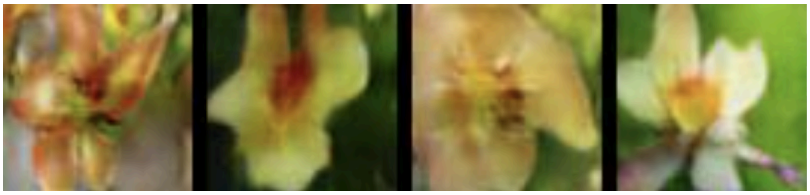
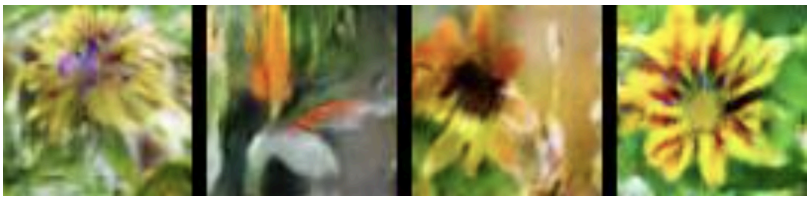
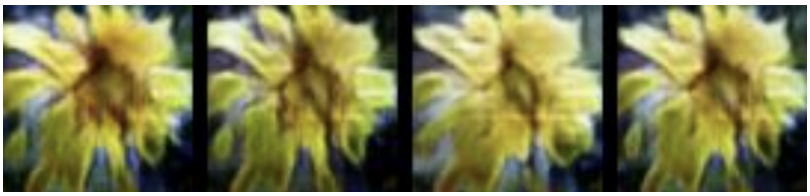
- Original paper
- Original implementation repository
- Flowers102 dataset
- "GAN Hacks" repository
- Improved Techniques for GAN Training

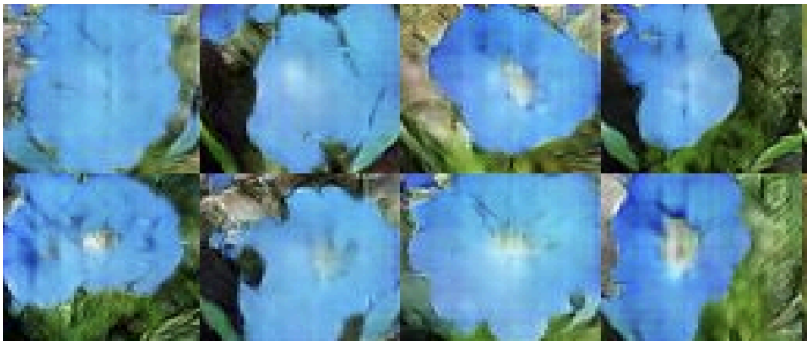
Link to repository containing project source code: https://github.com/dhruvKS7/FlowerTextToImage

Appendix

Additional comparison results between the three benchmarked models and the original paper:

Ground Truth	these flowers have petals that start off white in color and end in a dark purple towards the tips	
Original GAN-CLS		
Basic text encoder with classic training		
Basic text encoder with updated training		
Fine-tuned text encoder with updated plus training		

Ground Truth	bright droopy yellow petals with burgundy streaks, and a yellow stigma	
Original GAN-CLS		
Basic text encoder with classic training		
Basic text encoder with updated training		
Fine-tuned text encoder with updated plus training		

Ground Truth	the flower shown has a blue petals with a white pistil in the center	
Original GAN-CLS		
Basic text encoder with classic training		
Basic text encoder with updated training		
Fine-tuned text encoder with updated plus training