Text-to-Image Synthesis Using GANs

Kaiyuan Wang
k5wang@ucsd.edu

Merve Kilic
mkilic@ucsd.edu

Utkarsh Jain
utjain@ucsd.edu

Marialena Sfyraki
msfyraki@ucsd.edu

Abstract

This paper investigates the task of text-to-image synthesis using Generative Adversarial Networks (GANs). We first implement a baseline Deep Convolutional GAN (DCGAN) model as a baseline that generates new, realistic images based only on training images. Then, we implement GAN Conditional Latent Space (GAN-CLS) that incorporates text descriptions as input so that the model can generate images from text. However, we also train the GAN-CLS model on a modified loss function that helps the model optimize image/text matching in addition to generating the images. Our experiments show that although DCGAN's generated images are not realistic enough to fool a human into believing they are real, they are higher quality and less blurry than GAN-CLS images. Furthermore, we notice that GAN-CLS models take significantly more time to train and do not converge as efficiently as the DCGAN model but still achieve the desired results of generating images that follow the text. We also observe that the GAN-CLS model trained using the modified loss function produces slightly better images with sharper features.

1 Introduction

Text-to-image synthesis is a task in which a machine learning model generates an image from a textual description. The goal is to produce an image that is most accurate to the meaning of the description. This is an important research area in artificial intelligence, machine learning, and computer vision because it has the potential to revolutionize many fields. One of its many applications is for creative projects in art, design, and animation. Artists and designers can quickly create new and innovative visual content by generating images from text descriptions. Another application is for creating visualizations that can depict complex data or information in a more comprehensible manner. This may help visual learners understand concepts or scientists trying to convey scientific data. A third application of text-to-image visualization is data augmentation. Existing image datasets can be expanded with new, synthetic images. This can help improve the performance of machine learning models that rely on large amounts of data for training. Overall, text-to-image synthesis is an exciting task with many downstream applications that would revolutionize many fields.

However, text-to-image synthesis is a challenging problem. It requires the model to understand the semantics of the text and translate it into a visual representation. This involves not only generating an image that contains the objects, scenes, and attributes described in the text but also ensuring that the image is coherent and realistic.

Some approaches to this task include deep learning methods such as GANs. GANs consist of a generator and discriminator network in which the generator tries to produce images that the discriminator cannot determine whether they are true or generated images. The discriminator is trained to identify which images are real and which are fake. Some challenges GANs face include generating

diverse and high-quality images, handling complex textual descriptions, and avoiding mode collapse (where the generator produces limited variations of images).

In this project, we implement GAN models to synthesize images from text. First, we implement a DCGAN as a baseline because it does not use any text descriptions. DCGANs use Convolutional Neural Networks (CNNs) for both the generator and discriminator networks. This allows them to generate high-quality realistic images because the convolutional and deconvolutional layers in the generator gradually upscale the input noise into a realistic image. In the discriminator, the convolutional layers allow spatial features of the image to be captured, improving the discriminator's ability to distinguish between real and fake images.

Next, we implement GAN-CLS (based on [7]) which uses text descriptions to generate images. This differs from a basic GAN because it incorporates a classifier into the discriminator network so that the discriminator not only distinguishes between real and fake images but also classifies the generated images into different classes. Thus, the model's goal is to not only generate realistic images but to also have them belong to a specific class. This applies well to text-to-image generation where the textual description serves as a class for the image to belong to. Thus, the generated images will not only seem realistic but also match a given textual description.

We train our models with the Oxford-102 Flowers [5] dataset because it was used in the paper [7] we are basing our GAN-CLS model on. This dataset contains images of flowers of 102 different categories. Each category has between 40 and 258 images. This is a great fit for our task because we can focus on generating images of flowers rather than more diverse topics.

2 Related Work

Oxford-102 Flowers (102 Category Flower Dataset) was first introduced by Nilsback et al. in [5] for the task of classifying flowers from a large number of classes. The proposed system addresses the issue of recognizing classes from a dataset with high degree of intra-class variability and inter-class similarity. The Oxford-102 Flowers dataset is widely used in computer vision and machine learning research for benchmarking flower classification algorithms. Other similar datasets include Flower-17 dataset, Flower-102 dataset, and SUN Flower dataset.

More recently, [1] used the Oxford 102 Flower dataset in order to learn a search algorithm based on a convolutional neural network (CNN) that learns a sequence of augmentation operations, such as rotation, flipping, and color transformations, and their associated parameters. The authors demonstrate how the proposed algorithm achieves state-of-the-art performance using optimal augmentation policies in several image classification benchmarks. Furthermore, [2] used the Flowers-102 dataset to train a deep neural network model for the flower species classification task. The authors proposed an evolutionary approach to incorporate new tasks dynamically into large-scale multitask learning systems. They developed an algorithm, which optimizes the selection and order of tasks in a multitask learning system based on their performance and compatibility with the existing tasks.

GANs were first designed by Ian Goodfellow in 2014 and introduced in the paper [3]. The authors proposed a new approach to generative modeling by training a two-part neural network - consisting of a generator and a discriminator - in an adversarial setting. The discriminator tries to learn a binary classification indicating whether a sample is from the model distribution or from the data distribution, while the generator is trying to fool the discriminator by producing synthetic images from a noisy input that mimics the distribution of the training data. The paper demonstrates the computational advantages of the proposed adversarial modeling framework, which only requires backpropagation to obtain gradients, while inference is not needed during learning.

The task of generating images from visual descriptions has been addressed recently by Yan et al. in [10]. The authors proposed a conditional generative adversarial network (cGAN) architecture in order to learn the mapping of the input textual attributes to the corresponding image output. The proposed "Attribute2Image" model uses a text encoder to convert the input visual descriptions into fixed-length vector representations. Each attribute is mapped into a separate embedding space. The representations are then concatenated and used as input to the generator which transforms them into a corresponding image. The paper demonstrates how this new method produces images that look more realistic than those produced by other state-of-the-art methods.

Since the datasets with captioned images often contain more than one caption per image, the use of the additional information could be utilized to produce images that correspond better to the descriptions. The authors in [4] used a cycle-consistent adversarial network that incorporates cross-caption consistency constraints to ensure that the generated image is consistent with a set of captions that describe it. The proposed model, C4Synth, consists of two generators, one for generating images from textual descriptions and another for generating textual descriptions from images, and two discriminators that provide feedback to the generators. This model architecture ensures that the generated image matches the textual description that was used to generate it, and that the generated textual description matches the image that was used to generate it. The model is shown to outperform several state-of-the-art methods in experiments, indicating its potential for practical use in text-to-image synthesis applications.

3 Methods

For this project, our main goal was to understand how GANs work and explore their use in the text-to-image synthesis task. In that regard, we implemented a Vanilla DCGAN model that tries to learn the distribution of the training dataset and generate images when given a random Gaussian noise. Next, we implement the GAN-CLS model that conditions the image generation on some text.

3.1 Deep Convolutional GAN

Deep Convolutional GAN (DCGAN) was introduced in [6] and is a direct extension of GANs, but it differs in three major ways. First, DCGAN does not use any spatial pooling layers (for e.g., maxpool layer) and replaces them with strided convolutions which allows the model to learn its own spatial downsampling. They use this technique in the discriminator. In the generator, they use the same concept but with convolutional-transpose layers which allows the network to learn its own spatial upsampling. Second, they remove the fully connected layers on top of convolutional layers but instead directly connect the output of the generator to the input of the discriminator. Finally, they employ batch normalization to stabilize learning and prevent the generator from falling into the mode collapse problem.

A DCGAN model has two networks - Generator and Discriminator. The generator model is comprised of convolutional-transpose layers, batch normalization layers, and the Relu activation layers. It takes as input a noise vector sampled from a standard Gaussian distribution and outputs an image of shape $3 \times 64 \times 64$. The discriminator accepts an image of a shape $3 \times 64 \times 64$ and passes it through its convolutional, batch normalization, and Leaky Relu activation layers and outputs the probability of the image belonging to the real data distribution.

Generator

The generator network is designed to map the noise vector into a space of the same size as the images in the training dataset (i.e $3 \times 64 \times 64$). This is achieved by using 4 blocks of strided convolutional-transpose layers coupled with a 2-dimensional batch normalization layer and Relu activation layer. Finally, the output passes through a single strided convolutional-transpose layer with a Tanh activation function to map the final output in a $[-1, 1]$ range. Figure 1 illustrates the different layers of the DCGAN generator model.

Discriminator

The discriminator network acts like a binary classification model which predicts the probability of an image being real. This network accepts an input image of size $3 \times 64 \times 64$ and outputs a scalar probability. This is achieved by using 4 blocks of strided convolutional layers coupled with a 2-dimensional batch normalization layer (except for the first block) and leaky-Relu activation layer. Finally, the output passes through a single convolutional layer and a sigmoid activation function to output the probability.

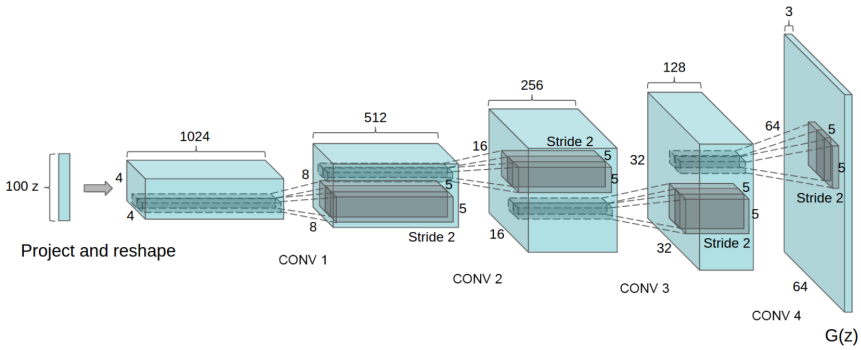


Figure 1: DCGAN generator architecture as proposed in [6]

Training

For training our DCGAN model, we employ the modified loss function taught in the lectures which looks like this:

$$\max_{\theta_d} \left[\mathbb{E}_{\mathbf{x} \sim p(data)} \left(log D_{\theta_d} \left(\mathbf{x} \right) \right) + \mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})} \left(1 - log D_{\theta_d} \left(G_{\theta_g} \left(\mathbf{z} \right) \right) \right) \right], \tag{1}$$

$$\max_{\theta_g} \left[\mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})} \left(log D_{\theta_d} \left(G_{\theta_g} \left(\mathbf{z} \right) \right) \right) \right], \tag{2}$$

where $\mathbf{x}$ denotes the data, $p(data)$ denotes the distribution of the data, $\mathbf{z}$ denotes the input noise, $p(\mathbf{z})$ denotes the distribution of the noise, θ_d stands for the parameters of the discriminator, θ_g stands for the parameters of the generator, D_{θ_d} stands for the value of the discriminator objective, G_{θ_g} stands for the value of the generator objective.

The goal of the discriminator is to maximize its objective function (1). We do this in two steps. First, we randomly sample a batch of real images from our dataset and process it through the discriminator. We then compute the loss $log(D(x))$ and the gradients. Second, we construct a batch of noise samples and pass it through the generator which generates a corresponding batch of fake generated images. We then pass these generated images through the discriminator and compute the loss $log(1 - D(G(x)))$ and the gradients. The gradients from both steps are accumulated and we optimize the discriminator by keeping the generator frozen.

Once we have trained the discriminator network on a batch of real and fake images, we now train the generator network to maximize its loss function (2). For this, we reuse the output from the discriminator network above on the batch of noise samples and compute the loss and gradients using real labels as ground truth. This way we can use PyTorch's nn.BCELoss function to compute both the generator and discriminator losses. Finally, we optimize the generator by keeping the discriminator frozen.

3.2 GAN Conditional Latent Space (GAN-CLS)

The GAN-CLS differs from the Vanilla DCGAN model in that GAN-CLS assumes a multimodal embedding space that encodes both text and image information. With this core difference in mind, the model architecture and loss functions for GAN-CLS are slightly different than that used for DCGAN.

Generator and Discriminator

We refer to the GAN-CLS architecture proposed in [7]. An illustration of said architecture is shown in 2. The multimodal embedding used by the generator network is constructed by concatenating a noise vector z sampled from a multi-dimensional Gaussian distribution with the text embedding. The multimodal embedding used by the discriminator network is constructed by concatenating the feature map after the second last convolution layer with the text embedding replicated in the height and width dimensions.

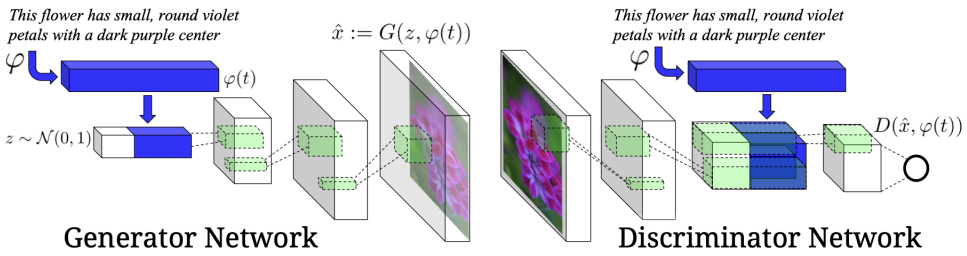


Figure 2: GAN-CLS architecture as proposed in [7]

Text Encoder

We thought of using an RNN-based text encoder and training it end-to-end with our GAN-CLS model. However, there were some conceptual problems with this approach. First, an RNN-based encoder would not pay any special attention to the adjective and visually descriptive words. For eg. in the sentence "this flower has orange petals with black dots and purple anther", we ideally want our encoder to pay more attention to words like "flowers", "orange petals", etc, which would not be possible with the basic RNN-based encoder. Second, we thought of using a Transformer-based approach to encode the text but that would have increased the overall training time. Finally, we were not sure of the loss function we could use to make our encoder produce embeddings that align with the fine-grained and category-specific content of the images. Due to these reasons, we opted to go for the pre-trained text encoder introduced in [8] which was trained in a zero-shot visual recognition setting and achieved state-of-the-art results in the same.

The main idea behind their work is to jointly embed images and fine-grained visual descriptions by learning a compatibility function of images and text such that the compatibility of a description is maximized with a matching image and minimized with a non-matching image. Given a dataset $S = \{(v_n, t_n, y_n) \; | 1 \leq n \leq N\}$, they seek to learn functions f_v and f_t such that equation3 is minimized:

$$\frac{1}{N} \sum_{n=1}^{N} L(y_n, f_v(v_n)) + L(y_n, f_t(t_n)), \tag{3}$$

where L is the 0-1 loss, v_n denotes the images, t_n denotes the text descriptions, and y_n denotes the class labels. The classifier f_v and f_t are parametrized as follows:

$$f_v = argmax_y E_t[\phi(v)^T \omega(t)], \tag{4}$$

$$f_t = argmax_y E_v[\phi(v)^T \omega(t)], \tag{5}$$

where $\phi(v)$ is the image encoder and $\omega(t)$ is the text encoder (hybrid character-level CNN LSTM network; used in our GAN-CLS model). A visualization of the model is given in Figure 3

Training using DCGAN Loss Function

Here, we describe an extension of the DCGAN by adding a conditioning ability to the network. This means that we can set a specific textual condition for the generator's output and the discriminator's input. This condition will limit the generator's output and the discriminator's expected input, making them work together in a particular mode of generation or prediction. The network is trained with text-image pairs. During the early stages of training the discriminator will easily identify the images generated by the discriminator as implausible, ignoring the conditional information. However, as the generator learns to produce more believable images, it also needs to align them with the corresponding textual information. The discriminator is trained to predict under the textual condition if the image came from the data distribution rather than the generator model. Thus, the discriminator

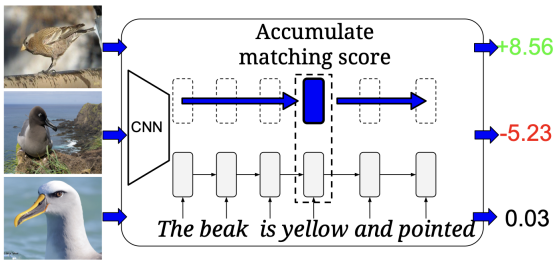


Figure 3: The text encoder $\omega(t)$ learns a scoring function between the images and text descriptions

learns to distinguish two sources of error: (a) unrealistic images with any conditioning information and (b) realistic images that do not match the corresponding textual description.

The objective function of the discriminator combines loss on images sampled from the training data (term 1) with loss on images sampled from the generator under the textual conditions h (term 2):

$$\max_{\theta_d} \Bigg[\underbrace{\mathbb{E}_{(\mathbf{x},h)\sim p_{data}(\mathbf{x},h)} \left(\log D_{\theta_d}(\mathbf{x},h)\right)}_{\text{term1}} + \underbrace{\frac{1}{2}\mathbb{E}_{\mathbf{z}\sim p(\mathbf{z}),h\sim p(h)} \left(1 - \log D_{\theta_d}\left(G_{\theta_g}(\mathbf{z},h),h\right)\right)}_{\text{term2}} \Bigg], \quad (6)$$

where h denotes the textual embedding corresponding to image $\mathbf{x}$, $p_{data}(\mathbf{x},h)$ denotes the distribution of the images in the training data and their associated textual embeddings, and $p(y)$ denotes the density model of the conditional information provided in the training data.

The objective function of the generator is obtained by:

$$\max_{\theta_g} \left[\mathbb{E}_{\mathbf{z}\sim p(\mathbf{z}),h\sim p(h)} \left(logD_{\theta_d}\left(G_{\theta_g}\left(\mathbf{z},h\right),h\right)\right)\right]. \quad (7)$$

Training with Modified Loss Function

In order to help the model discriminate between the different sources of errors, we introduce a third type of input that includes real images with incompatible text. The discriminator has to learn to recognize these inputs as fake.

The generator objective function is obtained by the same formulation mentioned in equation 7. The discriminator objective function is obtained through the following formulation:

$$\begin{aligned}\max_{\theta_d} \Bigg[&\underbrace{\mathbb{E}_{(\mathbf{x},h)\sim p_{data}(\mathbf{x},h)} \left(\log D_{\theta_d}(\mathbf{x},h)\right)}_{\text{term1}} + \underbrace{\frac{1}{2}\mathbb{E}_{\mathbf{z}\sim p(\mathbf{z}),h\sim p(h)} \left(1 - \log D_{\theta_d}\left(G_{\theta_g}(\mathbf{z},h),h\right)\right)}_{\text{term2}} \\ &+ \underbrace{\frac{1}{2}\mathbb{E}_{\mathbf{z}\sim p(\mathbf{z}),\hat{h}\sim p(\hat{h})} \left(1 - \log D_{\theta_d}\left(G_{\theta_g}(\mathbf{z},\hat{h}),\hat{h}\right)\right)}_{\text{term3}} \Bigg].\end{aligned} \quad (8)$$

Here we introduce two additional variables h and $\hat{h}$. h is the "right text" embedding corresponding to image $\mathbf{x}$, that is, the embedding of one of the ground-truth captions of image $[x]$. $\hat{h}$ is the "wrong text" embedding corresponding to an image irrelevant to $\mathbf{x}$. The intuition is that we wish the discriminator to be able to separate two sources of errors that render a generated image implausible: 1) unrealistic images for any text, which correspond to term1 and term2, and 2) realistic images that mismatch the provided text description, which correspond to term3.

4 Results

In this project, we use the Oxford-102 Flowers dataset which contains 8,189 images of flowers from 102 different categories. For both the DCGAN and the GAN-CLS model, we use all the images during training.

We use 1024-dimensional text embeddings produced by a character-level recurrent convolutional neural network (char-CNN-RNN) proposed by [8]. As the main point of this project is to learn and investigate GANs, we did not train said char-CNN-RNN from scratch. Instead, we used the text embedding provided by [8]. As a disclaimer, we implemented every other aspect of this project from scratch and trained it end-to-end.

For all our experiments, the image size was set to $3 \times 64 \times 64$ with a batch size of 64. The text encoder produces an embedding of size 1024 which is projected to a vector of size 128 using two different linear layers before both the generator and discriminator networks. We follow the commonly used method of alternatively training the two networks with Adam optimizer with a learning rate of 0.0002 and momentum of 0.5. The noise for the generator network is sampled from a 100-dimensional Gaussian distribution.

Deep Convolutional GAN

Figure 6 shows the loss of the generator and discriminator network over 30 epochs.

Figure 7 illustrates three different instances of the images generated by the generator during the training of the DCGAN model.

GAN-CLS with DCGAN Loss Function

Figure 8 shows the loss of the generator and discriminator network over 100 epochs.

Figure 9 shows the real vs. reconstructed pairs of images for the GAN-CLS model trained on the DCGAN loss function, at different training epochs. The first column begins as real images and the second column shows their reconstructed pairs. The other columns follow the same pattern so that the odd columns are the real images, and the following even columns are their corresponding generated image.

GAN-CLS with modified Loss Function

Figure 10 shows the loss of the generator and discriminator network over 100 epochs.

Figure 11 shows the real vs. reconstructed pairs of images for the GAN-CLS model trained on the modified loss function, at different training epochs. The first column begins as real images and the second column shows their reconstructed pairs. The other columns follow the same pattern so that the odd columns are the real images, and the following even columns are their corresponding generated image.

5 Discussion

5.1 Deep Convolutional GAN

GANs are challenging to train because of the inherently unstable learning due to the simultaneous training of two competing agents. Since we were completing new to GANs at the beginning, we did not explore different architectural settings and chose to use the best parameters from earlier works. As we can see from Figure 6, the training loss for both the discriminator and generator is very unstable, unlike our earlier assignments where the training loss usually went down as the epochs proceeded. However, there is a general trend of decreasing loss in Figure 6 which stabilizes after around 2000 iterations.

We trained our DCGAN model for 30 epochs and saved the generated images in between the training which can be seen in Figure 7. We can clearly see that the generator model keeps on improving over time and generating better results. One thing we noticed was that the generated images from early epochs had holes and grid pattern lines on them, which slowly disappeared as we kept on training the model. Although the images look plausible, we observe that they are not as good as the images generated from state-of-the-art GAN models.

Now, we move our discussion toward exploring the effect of changing the architecture of our model. These experiments were run on a subset of the original training dataset to save time.

Removing BatchNorm Layers

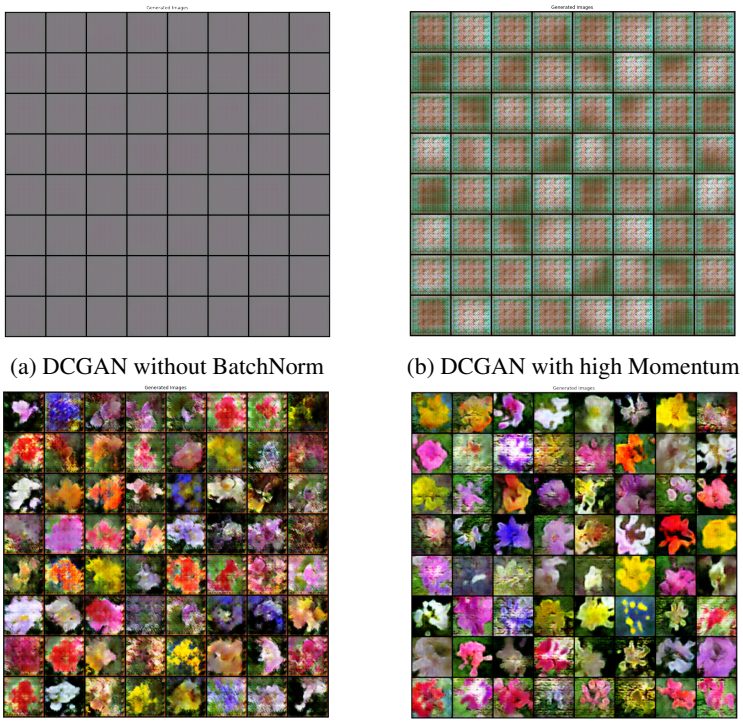
Batch normalization plays a vital role in training neural networks and stabilizes training by normalizing the input to every unit to have zero mean and unit variance. This proved to be critical in training our DCGAN model and prevented the generator from falling into the mode collapse problem where it collapses all samples to a single point. In one of our experiments, we removed the BatchNorm layers and face this issue firsthand. This can be visualized in Figure 4(a).

Effect of Momentum

We next experiment with the momentum of the adam optimizer we use while training. Unlike previous assignments where a momentum of 0.9 gave us the optimal results, we found out that using a momentum of 0.9 in our DCGAN model resulted in suboptimal training and did not let the discriminator loss "converge". We also notice that the generator loss did not change too much and stayed flat for most of the training. As expected the generated images suffered, but not as much as removing the BatchNorm layers where the generator produced noise. This can be visualized in Figure 4(b).

Using Max Pooling

We next experiment by using max pooling layers in our architecture. It is common to use spatial pooling layers in Convolutional Neural Networks for downsampling. Hence, we tried incorporating them into our DCGAN model to see what effect they might have on the overall training procedure and image generation. Although we expected the model to generate random noisy images as we saw in the last two experiments, we actually see that the model learns some of the features from the training dataset and generates images that somewhat look like flowers. However, we observe that using strided convolutions works better than max-pooling layers as a downsampling technique. This can be seen in Figure 4(c). Furthermore, we observe a sharp difference in the training loss curve where we noticed the discriminator and generator losses oscillate over time.



(a) DCGAN without BatchNorm

(b) DCGAN with high Momentum

(c) DCGAN with Max Pooling

(d) DCGAN

Figure 4: Images generated by different architecture configurations of DCGAN model

5.2 GAN-CLS

The main motive of our project was to explore how we can generate images from text and as we saw in section 4, our GAN-CLS model successfully achieves that goal. In this section, we discuss the effect of using the modified loss function and compare the images generated by the DCGAN and GAN-CLS models.

Effect of modified Loss Function

We trained our GAN-CLS model using two loss functions - DCGAN loss function, and a modified loss function. Here we investigate how the auxiliary loss term 3 in equation 8 improves generator training. As a recap, equation 8 is intended to regularize the discriminator such that it recognizes realistic generated images with an irrelevant text description. From Figure 8 and 10 we can see that the model with auxiliary loss term 3 actually converges faster as compared to the other. As shown in Figure 5, the generator trained with the scheme using auxiliary loss term 3 learns to generate finer-grained details at an earlier stage in the training process, as compared to the one trained without the auxiliary loss scheme. The point to note, however, is that both models are able to generate images that follow the text and we believe that using recent advancements in GANs would lead to improved results.

DCGAN vs. GAN-CLS

From Figures 7, 9, and 11 we can clearly see that DCGAN produces much more realistic images as compared to the GAN-CLS model which tends to produce blurry images. From the loss curves of the three models, we notice that DCGAN converges quickly and GAN-CLS never really converges even after being trained for 4 times longer. However, this behavior is expected because the GAN-CLS model is learning a more complex task than the DCGAN model.

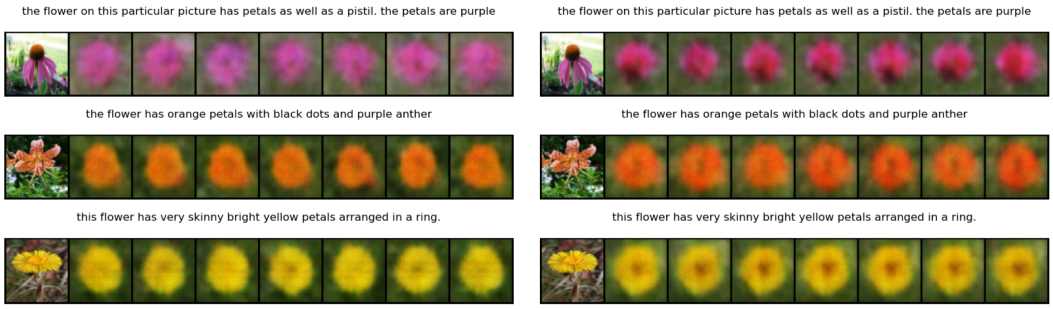


Figure 5: Generator (trained using 40 epochs) outputs. Left: Discriminator trained without auxiliary loss term 3. Right: Discriminator trained with auxiliary loss term 3.

6 Conclusions and Future Work

In this project, we saw how we can use GANs to generate images. First, we implemented a DCGAN model which learns the distribution of the training dataset and generates quite realistic flower images. We then moved our focus on experimenting with different architectural choices to see how it affects the overall GAN training procedure and concluded that GANs are hard to train and only a small number of configurations result in reliable training. Next, we took upon a more challenging task of text-to-image synthesis in which we aimed to generate images given some textual description. To achieve this, we first made some architectural changes in our DCGAN model to incorporate text in the training process. We called this model the GAN-CLS and saw how it successfully achieved our goal. Next, we saw the problem with the loss function and modified it to introduce a new source of error to help the model generate even better images that closely follow the textual description. We observed how this modified loss actually helped the model generate slightly better images with finer details.

In the future, we would like to try out a bigger version of our GAN-CLS model (with more layers and parameters) and train it on a bigger dataset to see how it affects the overall performance. We believe that with more capacity and a bigger dataset, we can achieve better results with some simple architectural changes. Furthermore, we would like to investigate how the quality of text embedding affects the quality of the generated images. More specifically, we would like to experiment with transformer-based encoders such as a BERT to obtain text embeddings. Recent research on Contrastive Language-Image Pretraining (CLIP) [9] has found that CLIP does a better job of phrase understanding, and it is something we would like to explore for our task of text-to-image generation. Moreover, current image generation models are heavily dependent on GANs which still deal with a lot of open problems. Hence, one possible research direction would be to explore other image-generative models like Variational Autoencoders (VAEs), Autoregressive Models, etc, and compare their performance with GANs. We would also like to explore more about how our model performs using images with higher resolution and how it can be altered (possibly with recent advancements in GANs) to produce images that better align with the semantics of the input text.

Team Contributions

- **Kaiyuan Wang**: Conducted literature review about DCGAN, GAN-CLS, textual-visual feature alignment, and text encoder. Outlined data pipeline, including retrieving and organizing the Flower102 dataset (more details in Marialena's contribution). Implemented GAN-CLS model and framed ablation studies about the improved loss function.
- **Utkarsh Jain**: Researched how GANs and DCGANs work and implemented the baseline DCGAN model. Ran experiments on the same and worked on the report with others. Worked with Marialena on understanding the unstable learning in GANs. Conducted a literature review on the text-to-image synthesis task and researched into how the text encoder used in GAN-CLS works.
- **Merve Kilic**: Conducted literature review about text-to-image synthesis model. Collaborated with Kyle on implementing the GAN-CLS model and the modified objective function. Conducted experiments on GAN-CLS, including hyperparameter tuning and ablation study, and worked on writing the report.
- **Marialena Sfyraki**: Worked on understanding how GANs work and researched why they suffer from unstable training. Explored what architectural changes lead to unstable learning, implemented the DCGAN architecture alternatives, and ran the experiments. Collaborated with Kyle on data pipeline by acquiring text embeddings for image captions (authors of [7] published a version of Flowers102 dataset that has pre-trained text embeddings in outdated `.h5py` format, which was reformatted into modern PyTorch data loader). Worked on writing the report.

References

[1] Ekin D. Cubuk et al. "AutoAugment: Learning Augmentation Strategies From Data". In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, pp. 113–123. DOI: 10.1109/CVPR.2019.00020.

[2] Andrea Gesmundo and Jeff Dean. *An Evolutionary Approach to Dynamic Introduction of Tasks in Large-scale Multitask Learning Systems*. 2022. arXiv: 2205.12755 [cs.LG].

[3] Ian Goodfellow et al. "Generative adversarial nets". In: *Advances in neural information processing systems*. 2014, pp. 2672–2680.

[4] K. J. Joseph et al. "C4Synth: Cross-Caption Cycle-Consistent Text-to-Image Synthesis". In: *CoRR* abs/1809.10238 (2018). arXiv: 1809.10238. URL: http://arxiv.org/abs/1809.10238.

[5] Maria-Elena Nilsback and Andrew Zisserman. "Automated Flower Classification over a Large Number of Classes". In: *Indian Conference on Computer Vision, Graphics and Image Processing*. Dec. 2008.

[6] Alec Radford, Luke Metz, and Soumith Chintala. *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*. 2016. arXiv: 1511.06434 [cs.LG].

[7] Scott Reed et al. *Generative Adversarial Text to Image Synthesis*. 2016. DOI: 10.48550/ARXIV.1605.05396. URL: https://arxiv.org/abs/1605.05396.

[8] Scott Reed et al. *Learning Deep Representations of Fine-grained Visual Descriptions*. 2016. arXiv: 1605.05395 [cs.CV].

[9] An Yan et al. *CLIP also Understands Text: Prompting CLIP for Phrase Understanding*. 2022. arXiv: 2210.05836 [cs.CL].

[10] Xinchen Yan et al. *Attribute2Image: Conditional Image Generation from Visual Attributes*. 2016. arXiv: 1512.00570 [cs.LG].

Appendix A: Deep Convolutional GAN

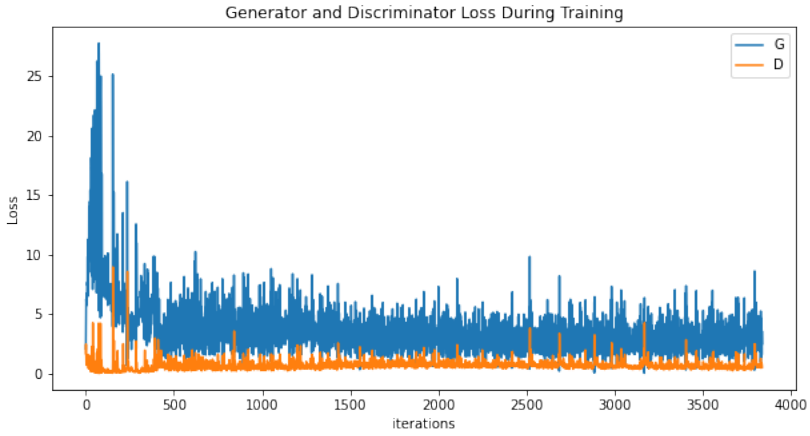
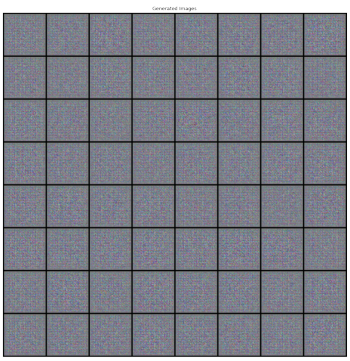
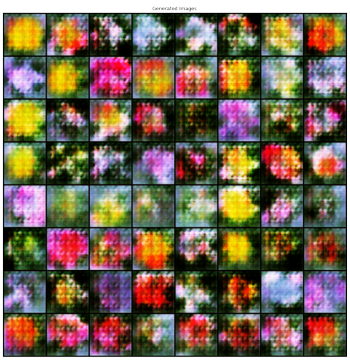


Figure 6: DCGAN loss

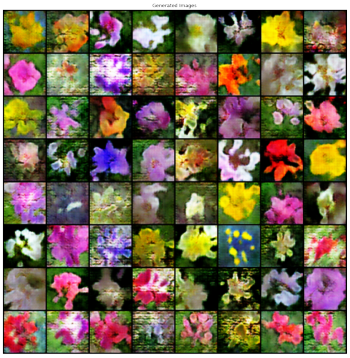


(a) Initial noise vectors



(b) Epoch 10

(c) Epoch 20



(d) Epoch 30

Figure 7: DCGAN generated images at different training epochs

Appendix B: GAN-CLS with DCGAN Loss Function

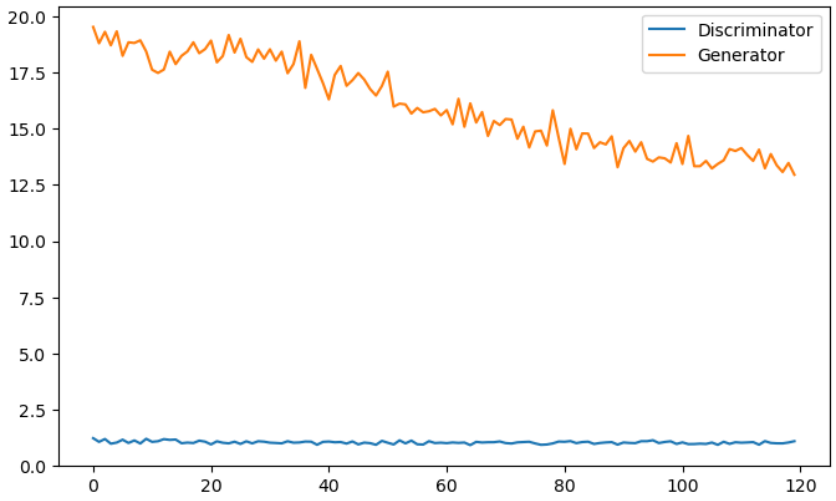


Figure 8: GAN-CLS (with DCGAN loss function) training loss across 120 epochs

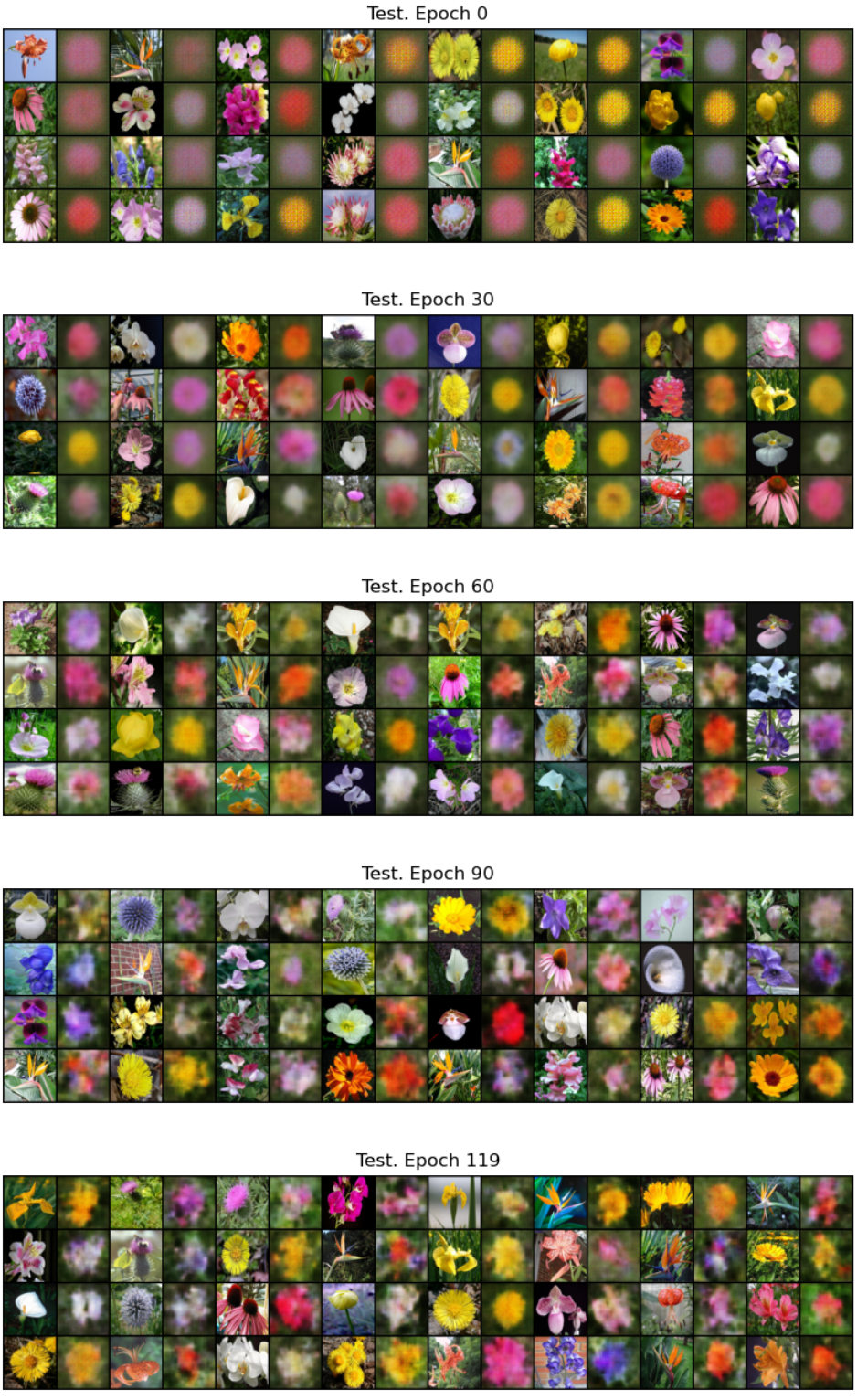


Figure 9: Real vs. reconstructed pairs of images for the GAN-CLS model (DCGAN loss function)

Appendix C: GAN-CLS with modified Loss Function

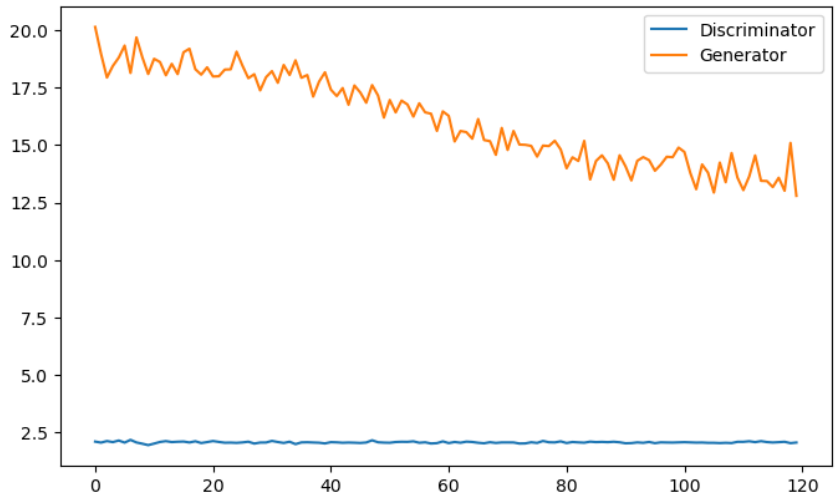


Figure 10: GAN-CLS (with modified loss function) training loss across 120 epochs

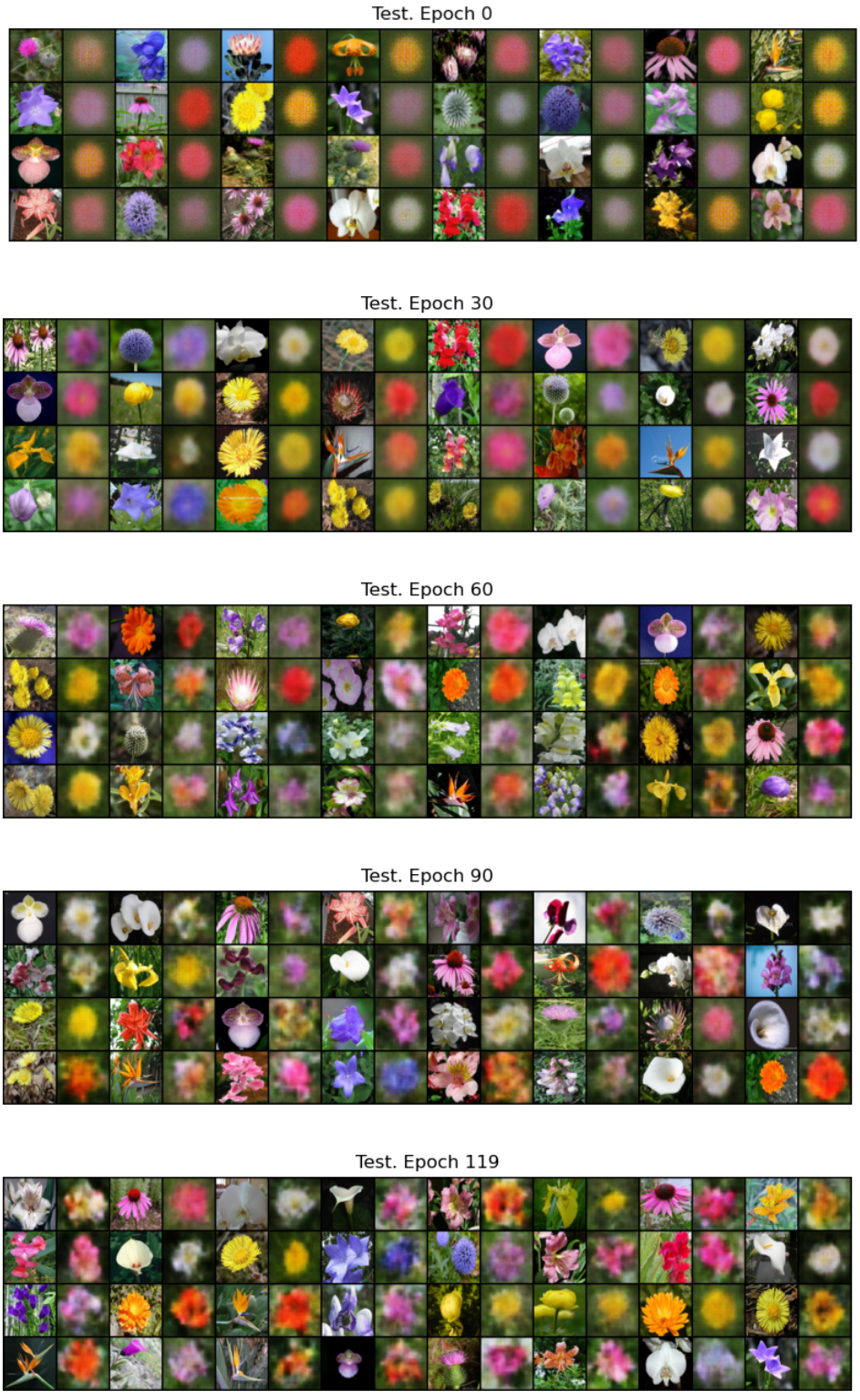


Figure 11: Real vs. reconstructed pairs of images for the GAN-CLS model (modified loss function)