ESCOLA POLITÉCNICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
MESTRADO EM CIÊNCIA DA COMPUTAÇÃO

MARTIN DUARTE MÓRE

STYLE TRANSFER FOR TEXT-BASED IMAGE MANIPULATION

Porto Alegre
2019

PONTIFICAL CATHOLIC UNIVERSITY OF RIO GRANDE DO SUL
SCHOOL OF TECHNOLOGY
COMPUTER SCIENCE GRADUATE PROGRAM

STYLE TRANSFER FOR TEXT-BASED IMAGE MANIPULATION

MARTIN DUARTE MÓRE

Dissertation submitted to the Pontifical Catholic University of Rio Grande do Sul in partial fullfillment of the requirements for the degree of Master in Computer Science.

Advisor: Prof. Rodrigo Coelho Barros

Porto Alegre
2019

Ficha Catalográfica

M999s Móre, Martin Duarte

Style transfer for text-based image manipulation / Martin Duarte Móre . – 2019.
108 p.
Dissertação (Mestrado) – Programa de Pós-Graduação em Ciência da Computação, PUCRS.

Orientador: Prof. Dr. Rodrigo Coelho Barros.

1. Image manipulation. 2. Natural language. 3. Style transfer. 4. GANs. 5. Adversarial training. I. Barros, Rodrigo Coelho. II. Título.

Elaborada pelo Sistema de Geração Automática de Ficha Catalográfica da PUCRS
com os dados fornecidos pelo(a) autor(a).
Bibliotecária responsável: Salete Maria Sartori CRB-10/1363

Martin Duarte Móre

Style Transfer for Text-based Image Manipulation

This Dissertation has been submitted in partial fulfillment of the requirements for the degree of Master of Computer Science, of the Graduate Program in Computer Science, School of Technology of the Pontifícia Universidade Católica do Rio Grande do Sul.

Sanctioned on March 28th, 2019.

COMMITTEE MEMBERS:

Prof. Dr. Cláudio Rosito Jung (PPGC/UFRGS)

Prof. Dr. Soraia Raupp Musse (PPGCC/PUCRS)

Prof. Dr. Rodrigo Coelho Barros (PPGCC/PUCRS - Advisor)

TRANSFERÊNCIA DE ESTILO PARA MANIPULAÇÃO DE IMAGENS BASEADA EM TEXTO

RESUMO

Grande parte dos dados que produzimos atualmente estão na forma de fotografias digitais, o que aumenta a demanda por aplicações de edição de imagens. Contudo, manipulação de imagens contém uma curva de aprendizado íngreme; desta forma, seria extremamente valioso automatizar ou simplificar este processo artístico para torná-lo mais acessível. Neste estudo, nós investigamos o uso de um subconjunto de linguagem natural (mais específicamente, descrições textuais de objetos) como entrada para automatizar a manipulação de imagens. Nós propomos uma abordagem baseada em aprendizado produnfo para a tarefa de manipulação de imagens baseada em texto que combina treinamento adversário e conceitos de transferência de estilo. Nós avaliamos nosso método, comparamos com abordagens referência e concluímos que nossos resultados possuem qualidade competitiva quando comparados com o estado-da-arte.

Palavras-Chave: Manipulação de Imagens, Linguagem Natural, Transferência de Estilo, GANs, Treinamento Adversário.

STYLE TRANSFER FOR TEXT-BASED IMAGE MANIPULATION

ABSTRACT

A large amount of the data we produce nowadays is in the form of digital photographs, which increases the demand for photo editing applications. However, image manipulation has a steep learning curve; as such, it would be invaluable to automate or simplify this artistic process to make it more accessible. In this study, we investigate the use of a subset of natural language (more specifically, textual descriptions of objects) as input to automatize image manipulation. We propose a deep learning approach for the task of text-based image manipulation that combines adversarial learning and style transfer concepts. We evaluate our method, compare it to baseline approaches, and conclude that our results have competitive quality when compared to the current state-of-the-art.

Keywords: Image Manipulation, Natural Language, Style Transfer, GANs, Adversarial Training.

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

ADAIN – Adaptive Instance Normalization

AI – Artificial Intelligence

ANN – Artificial Neural Network

ASPM – Arbitrary-Style-Per-Model

BN – Batch Normalization

BPTT – Backpropagation Through Time

CGAN – Conditional GAN

CNN – Convolutional Neural Network

DCGAN – Deep Convolutional GAN

FID – Fréchet Inception Distance

GANS – Generative Adversarial Networks

IN – Instance Normalization

IOB-NST – Instance-Optimization Based Online Neural Style Transfer

IS – Inception Score

LSGAN – Least-Squares Generative Adversarial Network

ML – Machine Learning

MLP – Multilayer Perceptron

MOB-NST – Model-Optimization Based Online Neural Style Transfer

MSPM – Multiple-Style-Per-Model

NST – Neural Style Transfer

PSPM – Per-Style-Per-Model

RELU – Rectified Linear Unit

RNN – Recurrent Neural Network

SISGAN – Semantic Image Synthesis GAN

SGD – Stochastic Gradient Descent

ST – Style Transfer

TAGAN – Text-Adaptive GAN

VAE – Variational Autoencoder

CONTENTS

1. INTRODUCTION

With the ever increasing digitization of society, more and more of what we do is recorded. This phenomenon increased even more with the appearance of mobile computing devices (such as smartphones), the advent of social media, and the Internet of Things (IoT). It is estimated that a total of 1.2 trillion digital photos were taken worldwide in 2017, and 85% of them were taken on smartphones[1]. Since a large amount of the data we produce now is in the form of digital photographs, the demand for photo editing is growing. Some of these photo editing operations, such as applying filters, augmenting the contrast of an image, converting to grayscale, or performing small touch-ups, are relatively trivial and are already available to the general public. However, some non-trivial operations, such as making montages or doing complex object manipulations (e.g., modifying facial attributes), are not; these tasks require a deeper scene and object comprehension. In this case, a skilled human and a specialized image editing software are required to obtain realistic-looking results.

Image manipulation has a steep learning curve: it requires users to understand specialized language, understand the effects of image editing operations, and use specialized software to combine this knowledge and operations. Additionally, motivated by the pervasiveness and ease of usage, consumers are increasingly relying on portable embedded devices such as smartphones and tablets for their everyday activities. These devices tend to have small screens and limited interfaces that preclude fine-grained spatial controls. Thus, it would be extremely valuable to make image manipulation more accessible or automated to empower the general public. This automatization could have ramifications in several different areas, such as art, fashion, design, retail, and many more.

Some studies have already attempted to automate or simplify image manipulation processes, such as image colorization [136], style transfer [35, 59, 120, 121, 27, 53], image super-resolution [59], object replacement [72], draw-based editing [15, 137], and many more. The interfaces these studies provide for these tasks are simpler and more intuitive, but are still different from task to task.

One interesting input modality that can be used to manipulate images is natural language. Language offers a general and flexible interface for describing objects in any space of visual categories and is not tied to any specific domain of application. Using language would give more freedom of expression to the user, and has the potential to work across many device types, even emerging ones such as Virtual Reality (VR) and Augmented Reality (AR) glasses. This freedom of expression, however, comes at a cost: using language requires solving numerous sub-problems, many of which did not exist (or were delegated to humans) in traditional image manipulation software. Ultimately, this occurs because language is a weaker source of supervision. A language-based image manipulation system

[1]https://www.statista.com/chart/10913/number-of-photos-taken-worldwide/

would have to be able to extract information from text, parse the input image, and combine both to select which aspects of the image must change (and remain the same) to generate a manipulated image that conforms to the input specification.

One possible way of tackling text-based image manipulation is using deep learning, a subfield of Machine Learning that has increased rapidly in popularity and pervasiveness during the last decade due to recent theoretical advances, the development of ever increasing datasets, and the improvement of computational resources. Deep learning is a particular kind of machine learning that attempts to solve problems by allowing the computer to build knowledge of complex concepts out of simpler concepts in a nested hierarchy of concepts. This presents a shift of paradigm towards automated feature extraction and away from handcrafted feature extraction systems, which used to be the norm when creating pattern recognition systems. This is beneficial because learned representations often result in much better performance and more general systems that can be applied in a multitude of tasks with fewer modifications. Many of these deep learning systems show promising results, often surpassing human-level performance in several tasks, such as image recognition [46] and game-playing [111]. In fact, all the aforementioned studies of automatic image manipulation involve the use of deep learning [136, 35, 59, 120, 121, 27, 53, 59, 72, 15, 137].

Up until recently, most of the striking successes in deep learning have involved deep discriminative models that rely on a supervisory signal from large-scale databases of hand-labeled data, ignoring much of the useful information present in the structure of the data itself. Deep generative models have had less of an impact, due to the difficulty of approximating many intractable probabilistic computations that arise in maximum likelihood estimation and related strategies. However, variational autoencoders (VAEs) [64, 98] and generative adversarial networks (GANs) [39] recently emerged, providing powerful frameworks for learning generative models of arbitrarily complex data distributions. This gave generative models the potential to enhance or automate the process of artistic creation for non-trivial tasks, such as image synthesis [39, 64, 95], text-to-image synthesis [97, 135, 21, 133], image-to-image translation [56, 137], and many more.

In this study, we investigate how generative models have been used on the task of text-based image manipulation, which is characterized by transforming existing images based on natural language descriptions of the desired modifications. We propose a novel training procedure that combines *adversarial training* and methods from *style transfer* literature; the key idea is to treat points in sentence embedding space as containing aggregate style information, characterized by the combination of individual styles for each specific characteristic of the object being described in text form. We use the text embeddings to modulate the features of an existing image on a generator network. This network is optimized based on the difference in styles using a style representation extracted from a pre-trained network. We present the strengths and shortcomings of our method based on experimental anaylsis done on a dataset commonly used for this task. To the best of our knowledge, this is the first time that style transfer has been used in a multimodal scenario with text and image, and applied to the task of text-based image manipulation.

The remainer of this document is structured as follows. First, in Chapter 2, we discuss the theoretical background through which we ground our proposed solution. Next, in Chapter 3, we provide an overview of prior studies that are related to text-based image editing. In Chapter 4, we present the proposed solutions and, following that, provide experimental results in Chapter 5. Finally, in Chapter 6, we present a summary of our findings during the course of this study.

2. BACKGROUND

2.1 Artificial Intelligence

Ever since programmable computers were conceived, we have wondered whether they might be capable of thinking and learning. This interest is further evidenced by the proposal of tasks such as the Turing Test [118] and the Chinese Room Experiment [107], which attempt to define intelligence and determine whether a machine is capable of exhibiting intelligent behavior. We have also witnessed numerous attemps at replicating intelligent behavior inside machines. One such example is the work of McCulloch and Pitts [77], who proposed a model of artificial neurons that, when combined into simple network structures, could theoretically compute any computable function. This curiosity paved the way for the creation of the field of research known as Artificial Intelligence (AI), which attempts to understand and build intelligent entities [103].

Initially, the field of AI tackled and solved many problems that were deemed intellectually difficult by humans, such as playing chess. These problems usually could be described by a list of hard and formal mathematical rules. However, in almost all cases, these early systems turned out to fail miserably when tested on wider selections of problems and on more difficult problems. This set of difficult problems, such as recognizing spoken words or faces in images, had an interesting characteristic: they were easy for humans to intuitively solve, but hard for humans to formally describe and manually write computer programs to solve [38, 1]. This happened because early programs knew next to nothing about their subject matter, succeeding by means of simple syntactic manipulations, and because these systems solved problems by trying out different combinations of steps until the solution was found, which usually becomes intractable for complex problems [103].

This failure proved that solving complex problems using hard, fixed rules is cumbersome and difficult – if not impossible –, and that efforts should also be directed towards creating systems that adapt to new circumstances and detect and extrapolate patterns through experience, effectively creating their own "formal descriptions" for their domain of application. This can be interpreted as the existence of a self-improving mechanism that could effectively allow a machine to learn. Mitchell provides the following formal definition of learning: "A computer program is said to **learn** from experience *E* with respect to some class of tasks *T* and performance measure *P*, if its performance at tasks in *T*, as measured by *P*, improves with experience *E*" [80].

2.2 Machine Learning

Machine Learning (ML) is a subfield of AI that addresses the question of how to build computer programs that are capable of learning through experience. Systems that use machine learning focus on learning using a data-oriented approach, obtaining knowledge about their tasks by interpreting and extracting patterns from data and later applying that knowledge on unseen data for decision-making and predicting results [38]. Breaking down Mitchell's definition of learning from a Machine Learning perspective, we have the following components:

Task (*T*): the actual problem that the system is trying to solve. There are many types of tasks that ML can tackle, such as classification, regression, clustering, density estimation, sampling, etc. ML offers many approaches (algorithms) for problem solving, each with varying complexities and capacities. These algorithms can sometimes be used to solve multiple types of problems. Hence, in most cases, it is up to the ML practitioner to determine which approach is more suitable to solve the proposed problem.

Experience (*E*): the type of training experience that the system will be exposed to and try to learn from. The two most common categories that experiences fall into are **supervised learning** and **unsupervised learning**, which will be discussed on Sec. 2.2.1 and Sec. 2.2.2, respectively. In both cases, the experience is comprised of data (observations) and learning is performed by extracting patterns (knowledge) that emerge from observing the *features* (characteristics) of the data.

Performance measure (*P*): a quantitative measure used to evaluate the algorithm (or generated *model*). The performance measure is specific to the task being carried out by the system and must be carefully chosen to correspond well to the desired behavior for the system; if not, the system may fail to learn. Sometimes, we know what quantity we would like to measure, but measuring it may be impractical or intractable. In those cases, an approximation must be used to compute the performance of the system.

The algorithms (and models) that ML provides fall into two categories: **parametric** and **non-parametric**. Examples of parametric and non-parametric models include neural networks and k-nearest neighbor classifiers, respectively. One of the main differences between these categories is that the former contains a fixed amount of parameters, regardless of the dataset size, and the latter varies its number of parameters depending on dataset size [84]. Parametric models are usually faster to use, but make stronger assumptions about the nature of data distributions (e.g., when using a linear regression model, we assume the real function can be described linearly). Non-parametric models are more flexible and make less assumptions (the main assumption being that similar inputs have similar outputs), but are also often computationaly intractable [84, 57, 1]. Each ML algorithm will make certain assumptions and contain inductive bias, which helps with learning and generalization [80]. Trade-offs between overfitting, underfitting, model complexity, and dataset size must be analyzed on a case-by-case basis.

Many theoretical problems arise when learning from data. During training, we expose an ML algorithm to a *finite* amount of samples (which may even be noisy) and expect it to generalize well. Logically speaking, we can not infer general rules from a limited set of examples (inductive reasoning). It is also possible that our data does not capture all aspects necessary to determine the correct output for all future instances. Additionally, most of the current theory of machine learning relies on the crucial assumption that the distribution of training examples is identical to the distribution of future examples, which may not be the case in practice [80]. Machine Learning avoids these pitfalls, introducing uncertainty and approximations by borrowing concepts from statistics and probability theory to train and evaluate results [12]. If we assume that the distributions of the training examples and future observations are identical and that examples are independent from each other, we can expect our models to perform well. These assumptions are known collectively as the *i.i.d. assumptions* [38].

2.2.1 Supervised Learning

In supervised learning, the goal is to learn a mapping from some input $\mathbf{x}$ to some output y, given that the learning algorithm experiences a set of *training samples* $X = \{\mathbf{x}^i, y^i\}_{i=1}^N$ from a dataset X that contains N samples, called the *training set*. Here, y^i represents the *output* expected from the algorithm when provided $\mathbf{x}^i$, which contains the features of a given sample [84]. The term supervised learning comes from the interpretation that y acts as a "supervisor" to the learning algorithm [38].

If some relationship between $\mathbf{x}$ and y exists, we can formalize the problem of supervised learning as a **function approximation** problem. We assume $y = f(\mathbf{x}) + \epsilon$ for some unknown function f and the goal of learning is to estimate the function $\hat{f}$ given a labeled training set and make predictions using $\hat{y} = \hat{f}(\mathbf{x})$ [84]. The $f(\cdot)$ term represents the *reducible error* (can be minimized by learning using X) and $\epsilon \geq 0$ represents the *irreducible error* (independent of X and, therefore, not minimizable) [57]. The algorithm searches for possible solutions inside a *hypothesis space*, denominated $\mathcal{H}$, searching for the function that best fits the observed data and any prior knowledge held by the learner and also generalizes well to unseen data. This search is conducted by optimizing an *error function* J as to minimize the difference between the desired output $f(\cdot)$ and the approximation $\hat{f}(\cdot)$ [1].

Many supervised learning algorithms – such as logistic regression for classification – learn by estimating probability distributions, which allows them to give results using a probabilistic perspective instead of hard boundaries or binary decisions [84]. Parametric algorithms may, for instance, use a frequentist approach and use maximum likelihood estimation to solve the problem of estimating a conditional probability distribution $\hat{f}(\mathbf{x}) = p(y|\mathbf{x}, \theta)$, where θ represents the parameters of the model [1, 38].

2.2.2 Unsupervised Learning

The goal of unsupervised learning algorithms is finding patterns in data without supervision (i.e., not being given the desired outputs for each input) [84]. The process of **density estimation** in statistics is one of the central cases of unsupervised learning [1]. In order to do that, the algorithm experiences a set of *samples* $X = \{\mathbf{x}^i\}_{i=1}^N$ from a dataset X that contains N samples. Instead of performing conditional density estimation, as in supervised learning, we now are concerned with the task of unconditional density estimation, or $p(\mathbf{x}|\theta)$.

Clustering and dimensionality reduction are also classic examples of unsupervised learning tasks [84]. In the context of deep learning, unsupervised learning algorithms usually attempt to learn the probability distribution that governs the generation of samples of a dataset, whether explicitly or implicitly [38]. Unsupervised learning and supervised learning are not formally defined terms, which makes the boundaries between them not clear-cut, and many machine learning algorithms can be used to perform both tasks [38, 57]. Density estimation in support of other tasks is usually considered unsupervised learning [38].

2.3 Deep Learning

Traditional ML algorithms work very well on a wide variety of important problems. However, some of the central problems in AI, such as speech recognition and object recognition, do not have a satisfying solution using these traditional approaches [38].

One of the main problems that undermine the performance of traditional approaches is the *curse of dimensionality*. This phenomenon appears in high-dimensional spaces (data points described using many features) and occurs because the number of possible configurations of $\mathbf{x}$ becomes much larger than the number of training examples available in X [84]. Many machine learning algorithms examine neighboring data points to determine the most probable behavior of an unseen data point, but the concept of proximity becomes blurry in sparse high dimensions since most configurations will have no training example associated with it [1, 6, 7].

The process of examining neighboring data points to infer behavior about unseen data points is called the *smoothness prior* [6]. Formally, this prior states that the algorithm should learn a function f^* that satisfies the condition $f^*(\mathbf{x}) \approx f^*(\mathbf{x} + \epsilon)$, i.e., for small variations in $\mathbf{x}$, the output should be similar [38]. This is another crucial problem for traditional algorithms, which may sometimes rely exclusively on this concept of local template matching to generalize [8].

This failure is what, in part, motivated the development of *deep learning*. This field of research appears to have been recently created, however it dates back to the 1940s [38]. The most recent increase in popularity can be explained by several factors, some of them being:

- **Increasing Dataset Sizes:** with the ever increasing digitization of society, more and more of what we do is recorded. This process moves us towards the era of **Big Data**, which gives us access to ever increasing datasets for a wide variety of tasks. This has made using machine learning algorithms much easier because the key burden of statistical estimation – generalizing well to new data after observing only a small amount of data – has been considerably lightened [1, 84, 38].
- **Increasing Model Sizes:** compared to the early days of deep learning, we now have access to computational resources that are orders of magnitude better. Armed with faster computers, larger memory sizes, and processing units that are extremly parallel in nature (GPUs), deep learning practicioners can build and train models that are much bigger and more complex than before. These new models are able to achieve higher accuracy on more complex tasks [38].

Another important factor to consider is that the performance of some machine learning algorithms depend heavily on the representation (features) of the data that are provided. However, for many tasks, it is difficult to know what features should be extracted. Additionally, manually designing features for a complex task requires a great deal of human time and effort; it can take decades for an entire community of researchers. One solution to this problem is *representation learning*, which focuses on discovering not only the mapping from representation to output but also the representation itself [7]. Learned representations often result in much better performance than can be obtained with hand-crafted representations [38].

Deep learning is a particular kind of machine learning that attempts to solve this representation learning problem by allowing the computer to build knowledge of complex concepts out of simpler concepts in a nested hierarchy of concepts [38]. One of the core technologies that allow deep learning to perform representation learning are artificial neural networks, which will be discussed in Sec. 2.4. The term “deep” comes from the fact that the models used involve a greater amount of composition of learned functions than traditional machine learning does. Fig. 2.1 expresses the relationship between artificial intelligence, machine learning, representation learning, and deep learning.

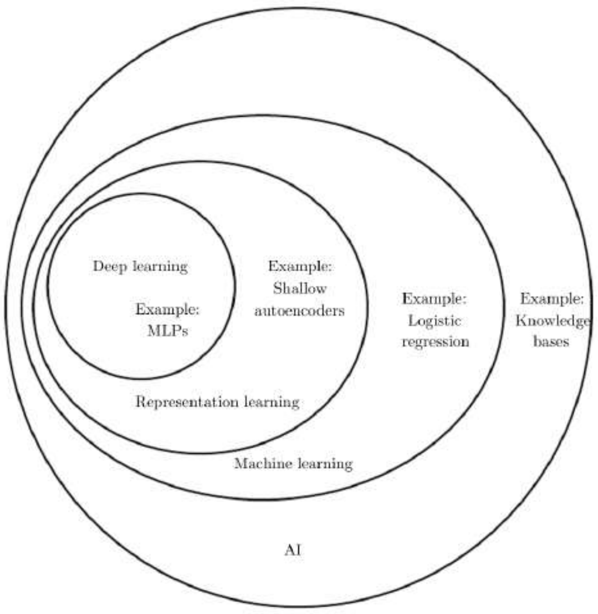


Figure 2.1: A Venn diagram showing the relationship between artificial intelligence, machine learning, representation learning, and deep learning. Source: [38].

2.4 Artificial Neural Networks

An artificial neural network (ANN) is a parametric machine learning algorithm that is capable of learning real-valued, discrete-valued, and vector-valued functions from examples. The term "neural network" refers to the fact that ANNs are an attempt to find a mathematical representation of information processing in biological systems, which are made of very complex webs of interconnected neurons [80, 12]. The most successful ANN is the feed-forward neural network, also known as the *multilayer perceptron* (MLP). This type of network appeared when researchers discovered the theoretical limitations that many previous, single-layered models of ANNs had, such as only being capable of approximating functions that were linearly separable [1].

The goal of the multilayer perceptron is to approximate some function $y = f^*(\mathbf{x})$ by defining a mapping of the form $y = \hat{f}(x;\theta)$, learning the values of the parameters θ (which are randomly initialized) given a cost function $J(\theta)$ [38]. This model is organized in a series of *layers* comprised of computing units called *neurons* (depicted in Figure 2.3). Each layer is connected to the following layer in a feedforward fashion by connections that are regulated by *weights* – which are the learnable parameters of the model. The first layer, connected to the input, is called the *input layer* and the final layer, which provides the network output, is called the *output layer*. All remaining intermediate layers are called *hidden layers*. Figure 2.2 illustrates a possible neural network architecture.

Formally, the output of unit j in layer l is given by a functional transformation calculated via a linear combination [11]:

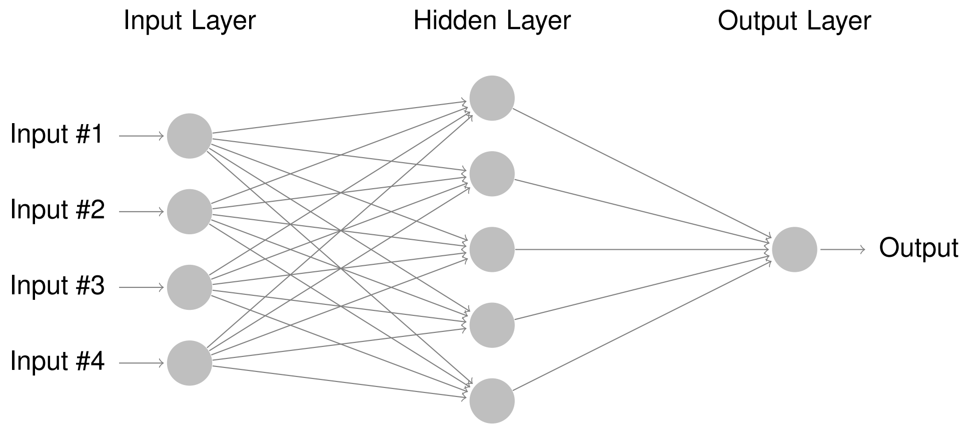


Figure 2.2: Illustration of a possible topology of a feedforward neural network.

$$a_j^l = b_j^l + \sum_k w_{jk}^l x_k^{l-1} \tag{2.1}$$

where x can be given by the input or the result of previous layers and b_j^l is the bias term for that neuron. Then, the activation of the same unit is given by applying a differentiable, nonlinear activation function on the previous result:

$$z_j = h(a_j) \tag{2.2}$$

where h is any differentiable, nonlinear activation function, such as *sigmoid*, *hyperbolic tangent* (*tanh*), or *rectified linear unit* (*ReLU*). A nonlinearity is needed because multiple layers of cascaded linear units still produce only linear functions [80]. By combining the results obtained on a layer and using them on sequential ones, we are able to compute complex, nonlinear functions. For instance, the function computed by a 3-layered MLP is given by $f(\mathbf{x}) = f^{(3)}(f^{(2)}(f^{(1)}(\mathbf{x})))$. The process of evaluating the final function of a neural network is called *forward propagation*, or the *forward pass*. An MLP is shown to be a **universal approximator**, meaning it can model any suitably smooth function, given enough hidden units, to any desired level of accuracy [51].

Unlike single-layered networks, MLPs are non-convex optimization problems due to the added nonlinearities, which means that the function represented by the network can have several local minima. For this reason, this type of network cannot be optimized analytically and a learning method based on an iterative numerical procedure, usually based on computing the *gradient* of a loss function with respect to the parameters θ of the neural network, must be used. These gradients can be computed efficiently in directed graphs of computations, such as neural networks, using a process called *error backpropagation* [101]. This process is also referred to as the *backward pass*, and involves computing the gradients of an expression (the loss function) through the layers of the network via the repeated appli-

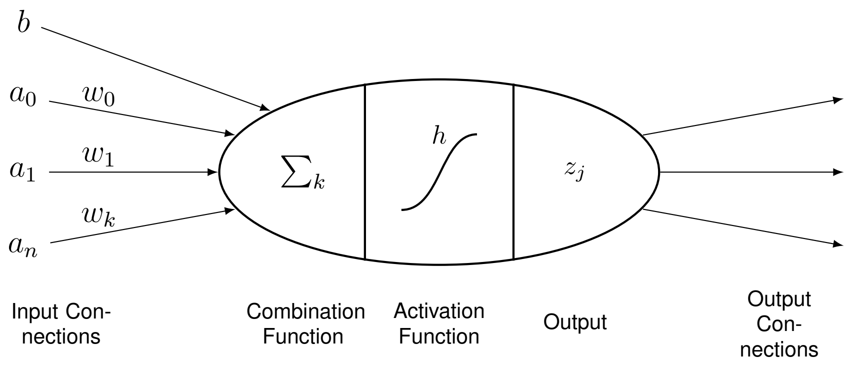


Figure 2.3: Illustration of a neuron, the processing unit in a neural network. Note that the operations performed by the neurons in a layer tipically occur in parallel as a vectorized operation.

cation of the chain rule for partial derivatives, until the network parameters are reached. To use backpropagation, the computational graph must be differentiable.

Gradient descent is one of the most popular gradient-based optimization algorithms to train neural networks. The gradient informs us the direction of biggest positive change in the slope of a function and, as mentioned before, can be computed via backpropagation. Since we wish to *minimize* an objective function $J(\theta)$, we update the parameters θ in the *opposite* direction of the gradient of the objective function $\nabla_\theta J(\theta)$ w.r.t θ [12]. Using the opposite direction of the gradient tells us in which direction we should travel, but it does not tell us how much we should move. The final rate of change in θ is governed by a hyperparameter η called *learning rate*, which multiplies the resulting gradient values:

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta) \tag{2.3}$$

Vanilla (batch) gradient descent, in its original form, computes the gradient using the entire dataset. Depending on the dataset size, this can be very slow and even become intractable, since our data may not fit in memory. There are two commonly used variants of gradient descent, namely Stochastic Gradient Descent (SGD) and Mini-batch Gradient Descent [100]. Instead of using the entire dataset, SGD calculates the gradient based only on one instance at a time. Mini-batch gradient descent is a variation of SGD that uses n instances to calulate the gradient. These variations are much faster than the vanilla version, but can also be more unstable, since there can be a lot of variance between samples. As such, the practicioner must identify this trade-off when optimizing neural networks. There are many studies that propose improvements to the SGD algorithm, such as the use of momentum [94, 86] and adaptive learning rates [26, 134, 63].

When using iterative training methods, such as SGD, we require an initial point from which to begin iterating from. Selecting this initial point has proven to be a non-trivial task [38, 66], and may determine the convergence or not of the algorithm (especially for deep networks). Most modern weight initialization strategies are based on heuristics, perhaps the only truly known necessity is that the initial parameters need to “break simmetry” betwen different units [38]. One common initialization strategy is to simply draw randomly from a Gaussian or uniform distribution. Other common initialization choices include the *Xavier* initialization [37] and the *He* initialization [45].

Another commonly used mechanism to help during optimization is the usage of *normalization layers*, which reparameterize deep networks, which significantly reduces the problem of coordinating updates across the many layers of deep networks. Some layer normalization methods include *Batch Normalization* (BN) [55], *Instance Normalization* (IN) [121], *Layer Normalization* [4], *Group Normalization* [132], and many more. Batch Normalization is, by far, the most widely used normalization technique, while the others are more appropriate for specific use cases.

2.4.1 Convolutional Neural Networks for Image Processing

In some applications, the input may contain local structure. For instance, in images, nearby pixels are correlated and objects can be interpreted as the combination of primitives, like edges and corners, into higher-order features [1]. In text, the semantic meaning of a word may vary according to its neighboring words. Many of the modern approaches to computer vision (and natural language processing) attempt to exploit this property of local structure by extracting features that depend only on small subregions of the input [12].

A *Convolutional Neural Network* (CNN) is a specialized kind of neural network for processing data that has a known, grid-like topology, such as images, audio, text, and video [68]. CNNs are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers [38]. The name indicates that the network employs a mathematical operation called *convolution*, denoted by the symbol $*$, which is a special kind of linear operation. In the specific case of a two-dimensional image I (discrete domain) and a two-dimensional kernel K, a single two-dimensional convolution operation S for the point (i, j) is given by applying the kernel filter on top on an (m, n) overlap segment of the image, which is expressed by the formula:

$$S(i,j) = (I * K)(i,j) = \sum_{m}\sum_{n} I(m,n)K(i-m, j-n) \tag{2.4}$$

which is a computation of the inner product between vectorized kernel and vectorized segment of the image[1]. Instead of implementing a convolution, many neural network libraries use a related function called *cross-correlation* because is it is simpler to implement. While there is a difference between these operations in signal processing literature, in the con-

[1]Naturally, this process can be generalized for three-dimensional inputs, which is the case of RGB-encoded images.

text of machine learning they are interchangeable: since the kernels are initialized randomly and learned using the same procedure, when we use cross-correlation we end up with the flipped version of the kernels learned using convolution [38, 28].

The convolutional filter is then slided on top of the whole image. The size of the step (how much the filter moves) is given by a hyperparameter called *stride*. It is common to apply several convolutional filters on the same layer of the network. By doing so, each filter can specialize in identifying a specific feature (characteristic), and each layer can be understood as a collection of image filters. Thus, the output of a convolution layer consists of so-called *feature maps*: differently filtered versions of the input [38]; by doing so, each filter can specialize in identifying a specific characteristic.

When compared to MLPs, there are several advantages to using a convolutional layer to process inputs with local structure. Convolutional layers tend to have fewer parameters than a traditional MLP, because all units in a feature map are constrained to share the same weight values. This means that CNNs scale better for large inputs than MLPs. Additionally, since we slide the filters throughout the entire image, CNNs are able to detect the same patterns if the image is shifted, a property called *translation equivariance* [12, 38]. This process becomes harder for MLPs, since a neuron is connected to all neurons of the previous layer.

In a practical architecture, convolutional layers may be followed by *subsampling* operations (such as average pooling) and additional convolutional layers. Convolution layers operate on a restricted area of the previous layer, which is called its *receptive field*. As we stack convolutional layers, their receptive fields progressively grow. This process allows CNNs to operate as a hierarchical cone and compute increasingly complex features [1]. Since the convolution operation is differentiable, it is possible to train CNNs using backpropagation [12]. CNNs are currently one of the fundamental building blocks for deep learning, and have been applied to several tasks and numerous input types, such as images [46], text [62, 22], and many more.

2.4.2 Recurrent Neural Networks for Sequence Modelling

There are many problems that requires us to process or generate sequential data types, such as text, audio, or video. Some examples of problems are speech recognition, machine translation, text summarization, and image captioning. In this context, it would be helpful if we could somehow keep a *history* of what we have processed so far. It quickly becomes impractical to use traditional MLPs for this task due to its fixed-length input and output and generalization problems [38, 40]. Thus, a new family of neural networks called *recurrent neural networks* (RNNs) was invented for processing sequential data; this new structure relies on *parameter sharing*, which means it uses the same set of weights across several time steps. An RNN is a neural network that consists of a hidden state h and operates on a variable length sequence $x = (x_1, \dots, x_t)$. At each time step t, the hidden state h_t is updated by

$$h_t = F(h_{t-1}, x_t; \theta) \tag{2.5}$$

where θ represents the model weights and F is the transition (or update) function, which varies depending on the type of RNN we are using. These hidden states can be used to produce one or multiple outputs y[2]. As an example, the transition function of a vanilla RNN (also known as Elman RNN [31]) is:

$$h_t = tanh(W_{xh}x_t + W_{hh}h_{t-1} + b_{hh}), \tag{2.6}$$

$$y_t = W_y h_t + b_y \tag{2.7}$$

where W_{xh}, W_{hh}, and W_y represent weight matrices and b_{hh} and b_y denote bias vectors. RNNs can also be optimized using gradient-based methods. Particularly, the most commonly used optimization procedure is *backpropagation through time* (BPTT) [131], which can be computed just like standard backpropagation if we *unroll* or *unfold* the RNN to become a feedforward model.

Sometimes, it is useful to have access to future inputs as well as past inputs. For instance, the meaning of some words can only be identified by inspecting the words that follow (and precede) it. Bidirectional RNNs [106] do just that by combining an RNN that starts at the beginning of the sequence and moves forward and another RNN that starts from the end of the sequence and moves backward. This provides the output layer with complete past and future context for every point in the input sequence. Although this breaks causality (since sometimes we cannot see into the future), this is acceptable on several tasks in which we already have the entire input to process (i.e., the input is spatial but not temporal) or when the network outputs are only needed at the end of some input segment [40].

There are two main problems when training traditional RNNs: gradients propagated through many stages may either vanish (common occurrence) or explode (rare). This happens because gradient multiplications can become exponentially smaller or bigger as we go back through the computational graph [9]. There are several proposed solutions for this problem, such as different optimization procedures, adding skip connections through time, gradient clipping, and many more [38].

One interesting and effective solution to handle long-term dependencies and mitigate the vanishing gradients problems are *gating mechanisms*, which augment the traditional

[2]The number of inputs/outputs of RNNs determine their categorization, which are (followed by examples): one-to-many (image captioning), many-to-one (sentiment analysis), and many-to-many (machine-translation)

RNN cell (the transition function F) by creating paths through time that have derivatives that neither vanish nor explode. Two of the most famous gated RNN architectures are the *Long Short-Term Memory* (LSTM) [50] and the *Gated Recurrent Unit* [20]. These architectures are currently the most effective sequence models used in practical application [38].

2.5 Deep Generative Models

In order to extract patterns from data, Machine Learning relies on probability theory to quantify and manipulate uncertainty in observations. Combined with decision theory, this provides ML algorithms with a consistent mathematical framework to perform predictions. Consider the task of classification, where we are given an input vector $\mathbf{x}$ with corresponding output vector $\mathbf{y}$ and our goal is to predict y given new values for $\mathbf{x}$. Since this task corresponds to discovering the *conditional probability distribution* $p(y|\mathbf{x})$, one possible approach for solving this problem is to use a **discriminative model** to learn the conditional probabilities directly. However, another valid approach would be using a **generative model** to learn the *joint probability distribution* $p(\mathbf{x}, y)$ that describes the behavior of the data and use Bayes' theorem to infer $p(y|\mathbf{x})$ [12].

If the goal is to simply make classification decisions, a discriminative model is probably the best choice, given that they are known to give better results and the problem is simpler to solve [87, 12]. However, one major advantage of using a generative model is that, by learning the joint probability distribution $p(\mathbf{x}, y)$, they acquire a much deeper understanding of the structural characteristics of the input, such as the data generating distribution that governs the dataset. This can be useful for several kinds of tasks [38], such as:

- **Density estimation:** given an input $\mathbf{x}$, return an estimate of the true density $p(\mathbf{x})$ under the data generating distribution.
- **Denoising:** given a damaged or incorrectly observed input $\tilde{\mathbf{x}}$, return an estimate of the original or correct $\mathbf{x}$.
- **Missing value imputation:** given some observed elements of $\mathbf{x}$, return estimates of or a probability distribution over some or all of the unobserved elements of $\mathbf{x}$.
- **Sampling:** generate new samples from the distribution $p(\mathbf{x})$.

One way to efficiently design a generative model to describe a probability distribution is using a graph to represent interactions between random variables. In this *graphical model*, nodes represent random variables and edges represent direct interactions. This graph then captures how the joint distribution over all variables can be decomposed into a product of factors that depend only on a subset of the variables [12]. Graphical models can be *directed* or *undirected*; this indicates, respectively, if the links of the graph have a particular directionality or not.

There exist many possible ways to evaluate generative models. For density estimation and related tasks, log-likelihood (or equivalently the Kullback-Leibler divergence) has been the de-facto standard for training and evaluating generative models [117]. However, it is important to emphasize that the likelihood of many interesting models, such as GANs, is computationally intractable.

Hence, for computational reasons, generative models are also often compared in terms of properties more readily accessible than likelihood, even when the task is density estimation. Some examples of comparison metrics are: visualizations of model samples, nearest neighbor, interpretations of model parameters, Parzen window estimates of the model's log-likelihood, and evaluations of model performance in surrogate tasks such as denoising or missing value imputation.

However, Theis et al. [117] show that good or bad performance with respect to one metric is no guarantee of good or bad performance with respect to the other metrics and that the quality of generated samples is generally uninformative about the likelihood and vice versa. There is no single metric that is capable of evaluating different generative models for different tasks, and proper assessment of model performance is only possible in the context of an application. Nevertheless, for image synthesis, a subjective evaluation based on visual fidelity of samples is appropriate [117].

There are many different approaches for constructing generative models. Among them, there are two promising ones that are recently pushing the state-of-the-art with highly plausible generated images using neural networks: variational autoencoders (VAEs) [64] and generative adversarial networks (GANs) [39].

2.5.1 Generator Networks

Generative models give us the ability to generate samples from distributions. However, complex distributions may be difficult to specify directly or even be computationally intractable [49, 104]. In the context of deep learning, many generative models are based on a differentiable *generator network*, whose goal is to transform samples of latent variables $\mathbf{z}$ to samples $\mathbf{x}$ or to distributions over samples $\mathbf{x}$ using a differentiable function $g(\mathbf{z};\theta^{(g)})$. This is tipically achieved by using artificial neural networks; the architecture of the network provides the family of possible distributions to sample from and the parameters select a distribution from within that family [38].

Since a neural network is differentiable, we can infer the parameters by using training data and traditional optimization algorithms. However, this learning process requires optimizing intractable criteria, because the training data does not specifiy both input $\mathbf{z}$ and output $\mathbf{x}$ [38].

2.5.2 Generative Adversarial Networks

Generative Adversarial Networks (GANs) is a framework for estimating generative models via an adversarial process [39]. This framework is comprised of two models that are trained simultaneously in an adversarial fashion: a generative model G that attempts to capture the data generating distribution and produce samples via a differentiable generatork network, and a discriminative model D who computes the probability that a sample came from the training data (true data distribution) rather than generated by G. Figure 2.4 illustrates the architecture of a the traditional GAN framework.

Formally, to learn the generator's distribution p_g over data $\mathbf{x}$, we define a prior on input noise variables $p_z(\mathbf{z})$, which are drawn from a simpler, known distribution (such as uniform or Gaussian), and then represent a mapping to data space as a differentiable function $G(\mathbf{z}; \theta^{(g)})$ represented by a multilayer perceptron with parameters $\theta^{(g)}$. Then, we define a second multilayer perceptron $D(\mathbf{x}; \theta^{(d)})$ that outputs a single scalar value that represents the probability that $\mathbf{x}$ came from the data distribution p_{data} rather than p_g.

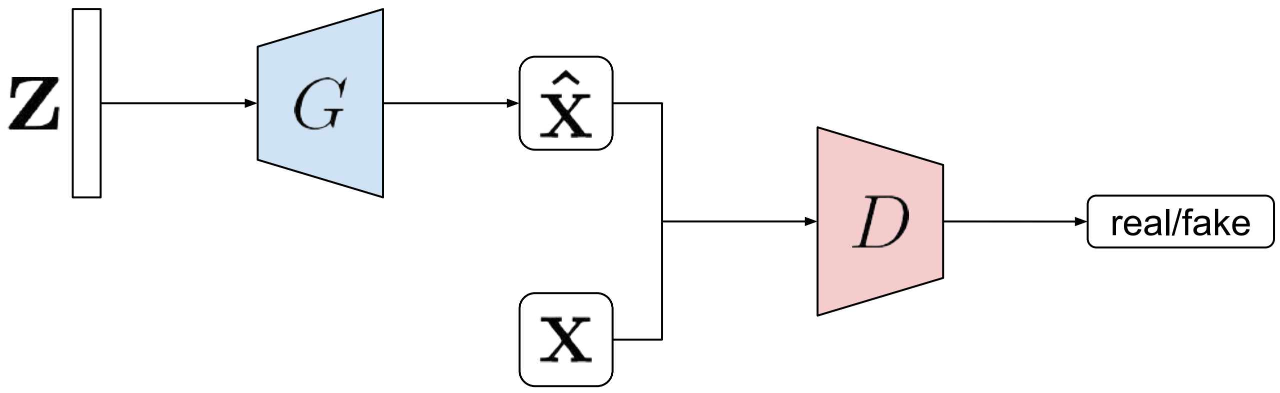


Figure 2.4: Diagram depicting the architecture of a vanilla GAN, where trapezoids represent processing via a neural network.

In the adversarial networks framework, D is trained to maximize the probability of assigning the correct label to both training examples, given by $D(\mathbf{x})$, and samples produced by G, given by $(1 - D(G(\mathbf{z}))$. Then, G is simultaneously trained to maximize the probability of D making classification mistakes, which is given by minimizing $log(1 - D(G(\mathbf{z})))$. This adversarial training procedure means that D and G play a two-player, zero-sum game with a minimax value function of

$$\min_{G}\max_{D} V(D,G) = \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x})}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}[\log(1-D(G(\mathbf{z})))]. \tag{2.8}$$

where the global optimum for this minimax game occurs when $p_g = p_{\text{data}}$. The name minimax stems from the fact that each player minimizes the maximum payoff possible for the other; since the game is zero-sum, they also minimize their own maximum loss. In a zero-sum game, the minimax solution is the same as the *Nash Equilibrium*, a situation where no player can benefit by changing strategies while the other players keep theirs unchanged [70]. This occurs when the discriminator is unable to differentiate between the two distributions, i.e. $D(x) = \frac{1}{2}$ for all x (real and fake).

The game defined in Equation 2.8 is implemented using an iterative, numerical approach, and the error function used for optimization can be easily derived from the value function. Since both D and G are defined by multilayer perceptrons, this framework can be trained with backpropagation. In practice, Equation 2.8 may provide poor gradients for G to learn well. This happens because, early in learning, G has a much harder task (generation) than D (discrimination), and D can learn quickly to reject samples with high confidence ($D(G(z)) \approx 0$) because they are clearly different from the training data. In this case, $log(1 - D(G(z)))$ saturates (i.e., the gradient has a very small value). A common modification to alleviate this issue is to train G to maximize $log(D(G(z))$. While no longer a true minimax game, this objective function results in the same optimization dynamic and provides much stronger gradients early in learning.

There are advantages and disadvantages to using the GANs framework. The main disadvantages are: there is no explicit representation of $p_g(\mathbf{x})$ and updates to D and G must be well synchronized during training to prevent mode collapse (a problem where G collapses many samples to a single point, reducing variability). The advantages of this framework are: Markov chains are not needed, only backpropagation is used to obtain gradients, no inference is needed during learning, a wide variety of functions can be incorporated into the model, the generator is not updated directly with data examples, and the ability to represent very sharp and degenerate distributions.

The traditional GAN framework is *unconditioned*, which means that there is no control over the modality of the data being generated. It is possible, however, to create a Conditional Generative Adversarial Network (CGAN) if we condition the models on additional information y, such as class labels or data from other modalities, to direct the data generation process (see Figure 2.5) [78].

The generator G implicitly defines a conditional density model $p_g(\mathbf{x}|y)$, which can be combined with an existing conditional density $p_y(y)$ (e.g., prior distribution over classes) to yield the joint model $p(\mathbf{x}, y)$. The exact task we need to perform is to parameterize G such

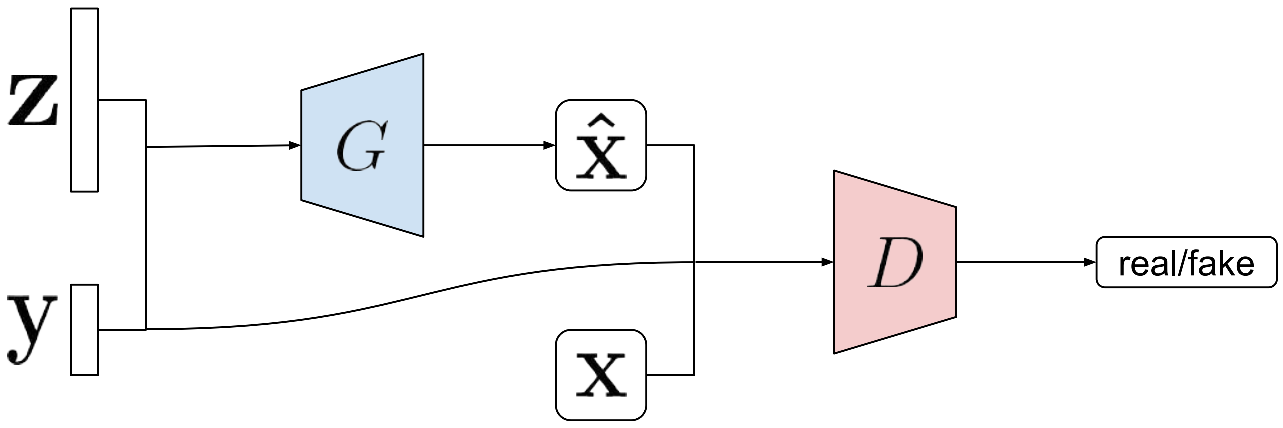


Figure 2.5: Diagram depicting the architecture of a conditional GAN (CGAN), where trapezoids represent processing via a neural network.

that it replicates the empirical density model $p_{\text{data}}(\mathbf{x}, y)$. We modify the original GANs value function described in Eq. 2.8 to be derived from expectations over distributions p_{data}, p_y, and $p_{\mathbf{z}}$:

$$\begin{aligned}\min_G \max_D V(D,G) = \mathbb{E}_{\mathbf{x},y\sim p_{\text{data}}(\mathbf{x},y)}[\log D(\mathbf{x},y)] \\ + \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z}),y\sim p_y(y)}[\log(1-D(G(\mathbf{z},y),y))].\end{aligned} \tag{2.9}$$

A crucial design decision is exactly where to insert the conditional information y. There are many possible architectures to incorporate the conditional information, one example being to concatenate y to the inputs of both the generator network and the discriminator network. Additionally, using such conditional data as "priors" on generation usually makes the generator model perform better, since it is now able to better navigate the space of possible outputs [90].

Even though convolutional neural networks were widely used for supervised learning, they received way less attention for unsupervised learning tasks, such as those tackled by generative models. Neural networks in general are heavily criticized for their lack of interpretability, and a great attractive of CNNs is that there are many studies that attempt to understand exactly what each convolution filter is processing [134, 82].

One of the first attempts to bridge this gap was the study conducted by Dosovitskiy et al. [25], who used supervised training to train a generative convolutional neural network to generate chairs given high-level information about the desired chair. For GANs, specifically, initial attempts to scale up using CNNs to model images had been unsuccessful. This motivated the authors of LAPGAN [23] to develop an alternative approach to iteratively upscale low resolution generated images with laplacian pyramids, which can be modeled more reliably. However, they still suffered from the objects looking wobbly because of noise introduced in chaining multiple models.

Radford et al. [95] conducted an extensive model exploration study and proposed a set of constraints on the architectural topology of Deep Convolutional GANs (DCGANs) that make them stable to train. These constraints include using all convolutional nets [113], eliminating fully connected layers on top of convolutional filters, using Batch Normalization [55], and using a specific combination of activation functions for the generator (ReLU and Tanh) and the discriminator (LeakyReLU). They also showed that, using convolutional layers in GANs makes it easier to visualize the filters learnt. Following this study, other researchers have also proposed guidelines [105, 2, 43] and new, improved techniques [3, 10, 76] for training GANs.

Although evaluating GANs is difficult, numerous metrics have been proposed [14]. The most famous and commonly used metrics for quantitative evaluation are the Inception Score (IS) [105] and Fréchet Inception Distance (FID) [48]; for qualitative evaluation, the most commonly used metrics are human evaluation (preference-based and real-fake distinguishment) and nearest neighbor inspection (generated samples are shown next to their nearest neighbors in the training set).

3. RELATED WORK

In this chapter, we perform a literature review of studies regarding the task of text-based image manipulation. First, in Sections 3.1, 3.2, 3.3, 3.4, and 3.5, we detail state-of-the-art approaches for performing text-based image manipulation on the flowers and birds domains. Next, in Section 3.6, we briefly discuss the same task, but on the fashion domain. After that, in Section 3.7, we list some studies that also manipulate images using natural language, but either solve different tasks, propose non-deep learning approaches, or use alternative text formats. Finally, in Section 3.8, we provide an overview and compare the current solutions for text-based image manipulation task, more specifically regarding the flowers and birds domains.

3.1 Generative Adversarial Text to Image Synthesis

In [97], the authors propose a deep learning approach for performing text-to-image synthesis, where the goal is to generate images from human-written visual descriptions (usually single-sentence). This task involves solving two sub-problems: learning a text feature representation that captures the relevant visual details and using these features to generate an image with the corresponding visual characteristics.

Since the paper was mainly focused on the generation task, the text feature representations were obtained via pre-trained hybrid character-level convolutional recurrent neural networks (char-CNN-RNN), denoted by φ, following the approach described in [96]. To generate images, the authors trained a GAN architecture named GAN-INT-CLS, depicted in Figure 3.1, conditioned on the aforementioned textual features, with a resolution of 64×64.

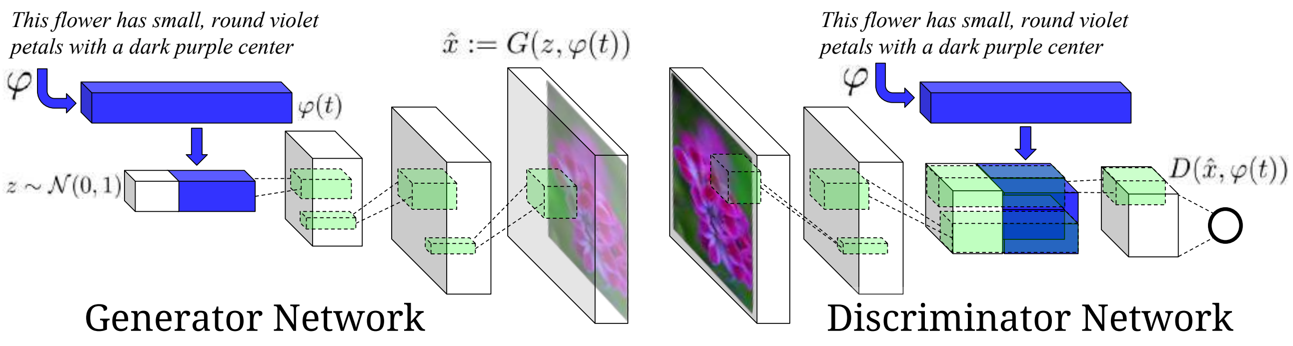

This flower has small, round violet petals with a dark purple center
φ
$\varphi(t)$
$z \sim \mathcal{N}(0,1)$
$\hat{x} := G(z, \varphi(t))$
This flower has small, round violet petals with a dark purple center
φ
$D(\hat{x}, \varphi(t))$
Generator Network
Discriminator Network

Figure 3.1: The text-conditional convolutional GAN architecture proposed by [97] for the text-to-image synthesis task. Source: [97]

The generator G synthesizes a fake image $G(z, \varphi(t)) \to \hat{x}$, where $z \in \mathbb{R}^Z \sim \mathcal{N}(0,1)$ is a sample from the noise prior and $t \in \mathbb{R}^T$ is a text query that is encoded by φ. The discriminator D generates a probabilistic output $D(x, \varphi(t)) \to [0,1]$. The image is processed via several layers of stride-2 convolution (until the spatial resolution of 4×4) and the resulting feature map is depth-concatenated with a dimensionality-reduced version of the description embedding. The concatenated features are processed via convolution to compute the final score. The authors propose a modification to the traditional GAN training to improve learning: in addition to the real/fake input pairs, a third type of input pair consisting of a real image with mismatched text $\hat{t}$ was added, which D must learn to score as fake.

Following the text to image experiments, the authors concluded that, if the text encoding $\varphi(t)$ captures the desired image *content* (e.g., object shape, texture, and color), then G must learn to use the noise sample z to encode the *style* factors (such as object pose, location, and background) and combine both to generate an image. Thus, the authors propose a second task with the goal of transfering the style of a query image onto the content of a particular text description. To do so, they trained a style encoder network S to invert G, regressing from generated samples back onto their latent representation. This network is trained using a simple mean square loss:

$$\mathcal{L}_{style} = \mathbb{E}_{t \sim p_{\text{data}}, z \sim p_z} ||z - S(G(z, \varphi(t)))||_2^2. \tag{3.1}$$

When both networks are trained, the style transfer task from a query image x onto text t proceeds by encoding the source image into it's latent style representation $S(x) \rightarrow s$ and then generating an image with the content contained in t, given by $G(s, \varphi(t)) \rightarrow \hat{x}$. The authors conducted experiments for the style transfer task on the Caltech-UCSD Birds 200-2011 dataset [125] and the textual descriptions were provided by [96]; the results can be visualized in Figure 3.2. Note that these results are not zero-shot (i.e., the network was optimized using the selected images).

Since the style transfer experiment was not the main topic of the paper, the authors do not evaluate the quality of the text-manipulated images directly. They do, however, evaluate the degree of disentanglement between style and content that the model can achieve. They set up two experiments using noise z as input: pose verification and background color verification (using inverted generators S). The intuition behind this is as follows: if the model has disentangled style (z) from content (text), the similarity between images of the same style (e.g. similar pose) should be higher than that of different styles (e.g. different pose). After executing the experiments, they report that – as expected – captions alone are not informative enough for style prediction, but the models are good at predicting background characteristics. This provides evidence that there is significant disentanglement and that z contains style information, since the captions normally do not describe the pose of the bird or the background.

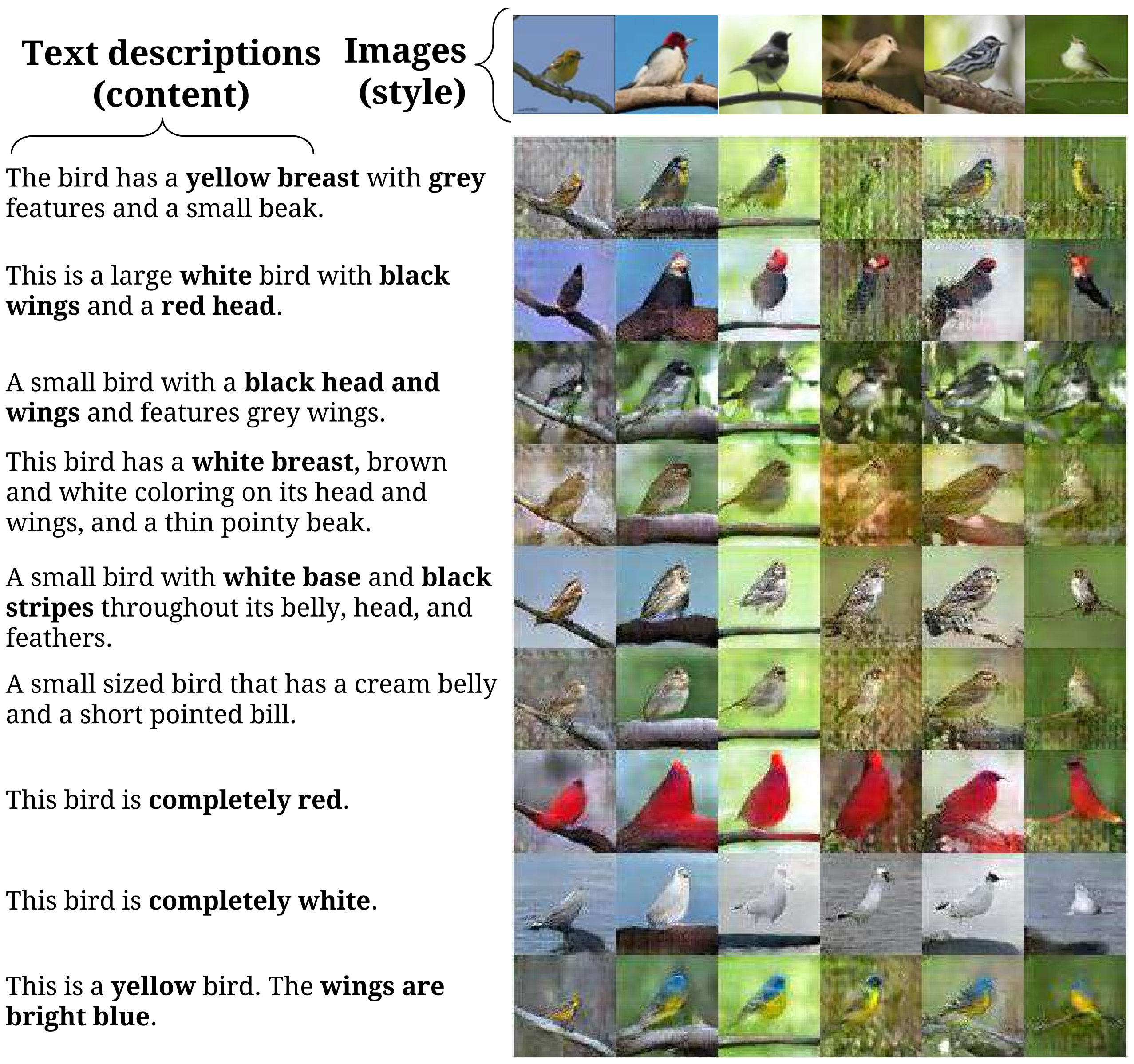

Text descriptions (content)
Images (style)
The bird has a **yellow breast** with **grey** features and a small beak.
This is a large **white** bird with **black wings** and a **red head**.
A small bird with a **black head and wings** and features grey wings.
This bird has a **white breast**, brown and white coloring on its head and wings, and a thin pointy beak.
A small bird with **white base** and **black stripes** throughout its belly, head, and feathers.
A small sized bird that has a cream belly and a short pointed bill.
This bird is **completely red**.
This bird is **completely white**.
This is a **yellow** bird. The **wings are bright blue**.

Figure 3.2: Results for the style transfer task. Source: [97]

3.2 Semantic Image Synthesis via Adversarial Learning

Dong et al. [24] argue that the method proposed in [97] for the image manipulation task has two main drawbacks. First, it is a two-step method, since the style encoder network S must be optimized using a pre-trained image synthesis network G. The second drawback is that S is never exposed to real images; rather, it is trained using images generated by G. Thus, the quality of the features learned by S depends not only on its optimization procedure, but also on how well G approximates the true data distribution. While this does not pose an issue for GANs in a theoretical optimum, where $p_g = p_{\text{data}}$, the generator may not achieve this result for highly complex data distributions. This means that S may generate poor feature representations when encoding real images.

With that in mind, the authors take inspiration from image-to-image translation literature to propose a new conditional GAN architecture (SISGAN), depicted in Figure 3.3, and changes to the adversarial training to improve results on the task of text-based style transfer. The resolution fo the images generated by this approach is 64×64.

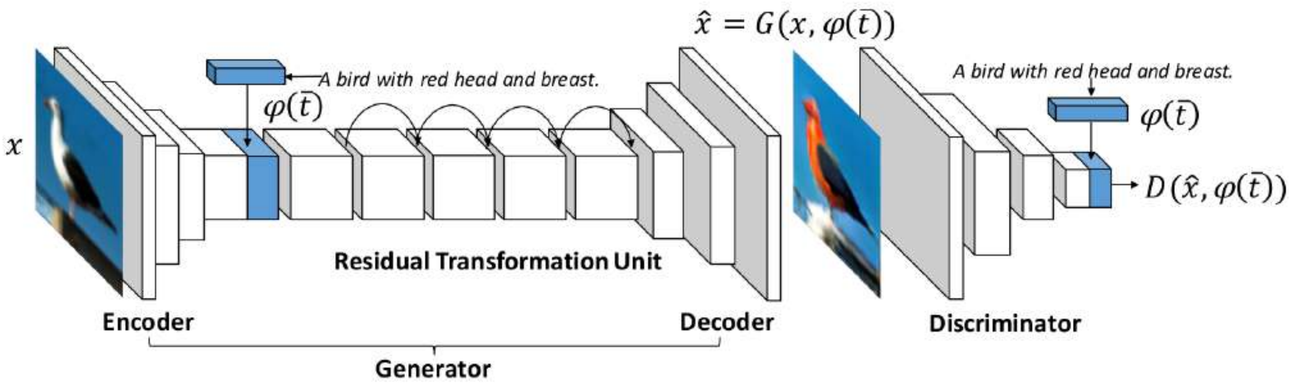

$\hat{x} = G(x, \varphi(\bar{t}))$
x
A bird with red head and breast.
$\varphi(\bar{t})$
Residual Transformation Unit
Encoder
Decoder
Generator
A bird with red head and breast.
$\varphi(\bar{t})$
$D(\hat{x}, \varphi(\bar{t}))$
Discriminator

Figure 3.3: The architecture for the method proposed by Dong et al [24]. Source: [24]

To obtain visually-descriptive textual feature representations, the authors follow the approach described in [65] to pre-train a text encoder φ using a multimodal learning approach, bringing pairs of textual descriptions and image features into the same point in an embedded space. Specifically, given a pair of image x and textual description t, they use a convolutional network ϕ and a recurrent network φ to encode image and text, respectively. Then, the following pairwise ranking loss is minimized:

$$\begin{aligned}\mathcal{L}_r = &\sum_x \sum_k \text{max}\{0, \alpha - s(\phi(x), \varphi(t)) + s(\phi(x), \varphi(t_k))\} \\ &+ \sum_t \sum_k \text{max}\{0, \alpha - s(\varphi(t), \phi(x)) + s(\varphi(t), \phi(x_k))\},\end{aligned} \tag{3.2}$$

where s denotes the cosine similarity of two embeddings and x_k and t_k represent contrastive (mismatching) examples. They further apply a text embedding augmentation method on $\varphi(t)$ to reduce dimensionality and sparsity. Originally proposed by [135], this augmentation method helps to generate a large number of additional text embeddings for the same textual description, which adds noise to the model. The authors use FastText vector representations for individual words using pre-trained word embedding models [13].

The architecture of the generator netork G is comprised of an encoder, a residual transformation unit, and a decoder. The encoder is a CNN whose input is an image x, which is processed into a spatial feature representation, retaining the convolutional features of the source image (the authors also experimented with using a pre-trained VGG [112]). Then, the text embedding t is processed by φ, processed by the conditioning augmentation subnetwork, and is spatially replicated to match the size of the convolutional features and is concatenated with them. Next, the features from both image and text are processed through the residual transformation unit, which is comprised of several residual blocks [46]. These blocks provide the model with a deeper encoding process at a reduced computational cost and allow the network to learn the identity function more quickly – which is a desirable property, since it allows the network to retain the features of the input image that are irrelevant to the target text descriptions. Finally, the decoder consists of several upsampling layers that transform the latent feature representations into an ouput image.

For the discriminator, D, the input image is processed through convolutional layers, concatenated with the (spatially-replicated) text embedding features and, finally, the concatenation is processed through more convolutional layers to produce the final probabilistic output. To train the networks, the authors propose the following adversarial loss:

$$\begin{aligned}\mathcal{L}_D &= \mathbb{E}_{(x,t)\sim p_{\text{data}}}[\log D(x,\varphi(t))] \\ &\quad + \mathbb{E}_{(x,\hat{t})\sim p_{\text{data}}}[\log(1-D(x,\varphi(\hat{t})))] \\ &\quad + \mathbb{E}_{(x,\bar{t})\sim p_{\text{data}}}[\log(1-D(G(x,\varphi(\bar{t})),\varphi(\bar{t})))], \\ \mathcal{L}_G &= \mathbb{E}_{(x,\bar{t})\sim p_{\text{data}}}[\log(D(G(x,\varphi(\bar{t})),\varphi(\bar{t})))]\end{aligned} \tag{3.3}$$

where where t is matching text, $\hat{t}$ is mismatching text and $\bar{t}$ is semantically relevant text (i.e., other texts describing objects of the same class). The conditioning augmentation subnetwork is optimized using the Kullback-Liebler Divergence loss, based on a normal distribution: $\mathcal{KL}\left[\mathcal{N}(\mu(\varphi(t)),\Sigma(\varphi(t)))||\mathcal{N}(0,I)\right]$. Unlike in the traditional GAN setup, the generator does not receive noise as input; the stochasticity/diversity in this method relies solely on the conditioning augmentation procedure. The method was evaluated by conducting experiments on the Caltech-UCSD Birds 200-2011 dataset [125] and the Oxford 102 Category Flowers dataset [88]; Figures 3.4 and 3.5 contain sample results.

For evaluating the models, the authors qualitatively and quantitatively compare their approaches (trained encoder and VGG encoder) with the baseline of [97]. The qualitative experiment was an empirical comparison, done by the authors, between the outputs for each method. The quantitative experiment was done using human evaluation using ten subjects, which were asked to rank the quality of zero-shot generated images using each method (the order was randomized and the method name was hidden from the participants). The participants were asked to rank each method from best to worst based on the following criteria: the synthesized image keeps the original pose of the object; the synthesized image keeps the original background, ad the synthesized image matches the text description while being realistic. The scores were averaged among participants. Analyzing the data, the authors concluded that their approaches were preferred over the baseline.

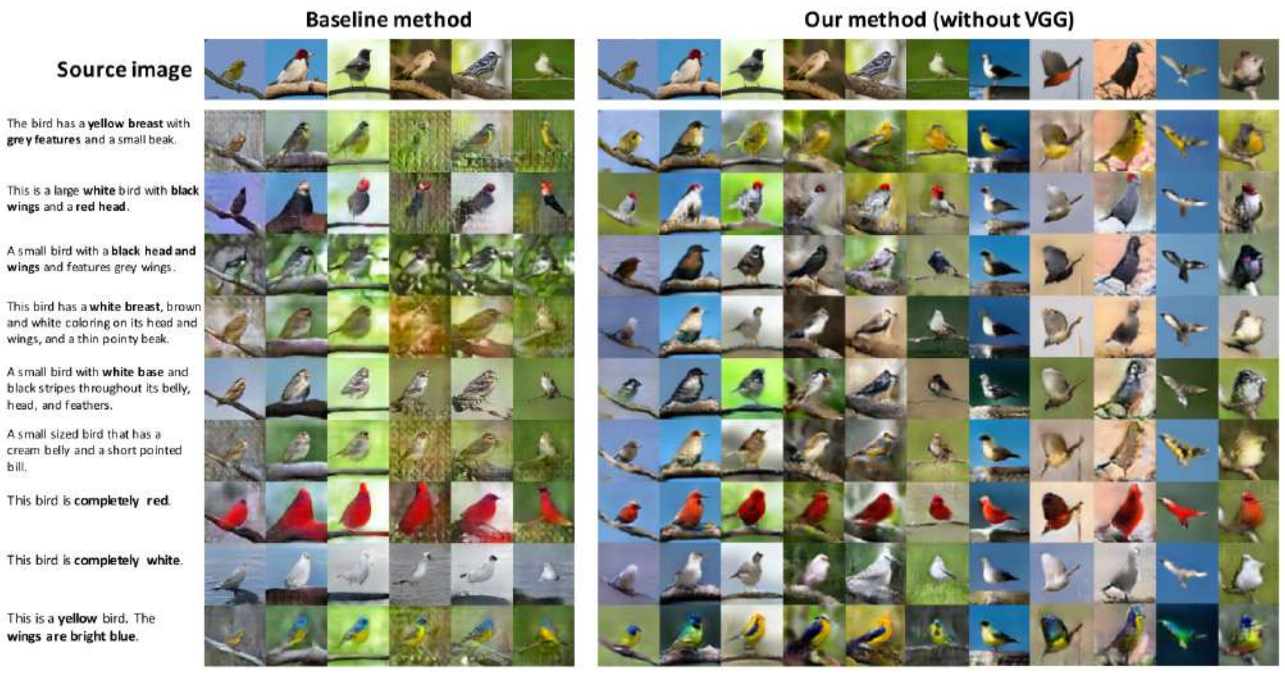

Baseline method
Our method (without VGG)
Source image
The bird has a yellow breast with grey features and a small beak.
This is a large white bird with black wings and a red head.
A small bird with a black head and wings and features grey wings.
This bird has a white breast, brown and white coloring on its head and wings, and a thin pointy beak.
A small bird with white base and black stripes throughout its belly, head, and feathers.
A small sized bird that has a cream belly and a short pointed bill.
This bird is completely red.
This bird is completely white.
This is a yellow bird. The wings are bright blue.

Figure 3.4: Zero-shot results for the method proposed by by Dong et al [24] on the birds dataset. The baseline is the method in [97]. Source: [24]

3.3 MC-GAN: Multi-conditional Generative Adversarial Network for Image Synthesis

Park et al. [91] propose an extension of the text-to-image synthesis task; instead of generating images based only on textual descriptions, the authors aim to synthesize images of objects using a base image as background, the textual description of the object, and its desired location on the base image. To do so, they propose a multi-conditional GAN architecture comprised of synthesis blocks, which act like pixel-wise gating functions controlling the amount of information from the base background image with the help of the text description for a foreground object.

The text embeddings used in this study are provided by pre-trained models using the same method as in [24], denoted by φ. The conditioning augmentation method described in [135] is also used. Besides this new extension to the text-to-image task, the authors also analyze how well the proposed architecture can deal with the style transfer task described in [24].

A synthesis block, depicted in Figure 3.6, receives as input both the foreground and the background feature maps, which have the same spatial and depth sizes. The foreground features are processed via convolutional layers, which also double the depth of the feature map. Half of these features are passed through a sigmoid function, which then act as a switch tha control the presence of the background features via an element-wise multiplication. Then, the other half of the foreground features are element-wise added to the

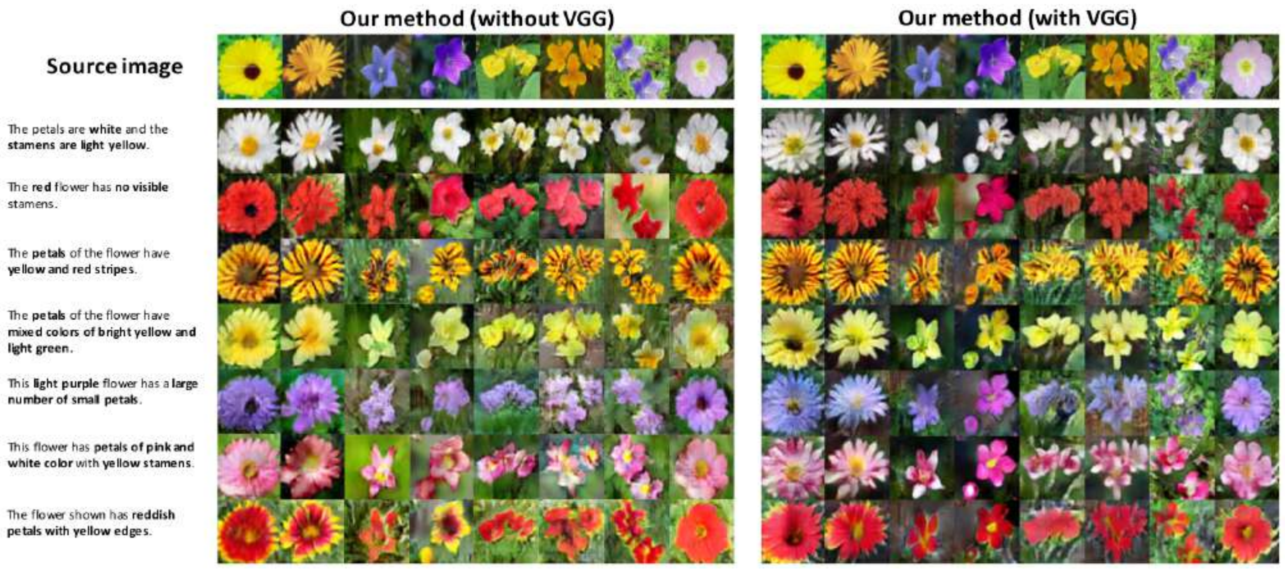

Our method (without VGG)
Our method (with VGG)
Source image
The petals are white and the stamens are light yellow.
The red flower has no visible stamens.
The petals of the flower have yellow and red stripes.
The petals of the flower have mixed colors of bright yellow and light green.
This light purple flower has a large number of small petals.
This flower has petals of pink and white color with yellow stamens.
The flower shown has reddish petals with yellow edges.

Figure 3.5: Zero-shot results for the method proposed by by Dong et al [24] on the flowers dataset. Source: [24]

modulated background features. Finally, the features are upsampled using nearest neighbor interpolation, which is the output of the synthesis block.

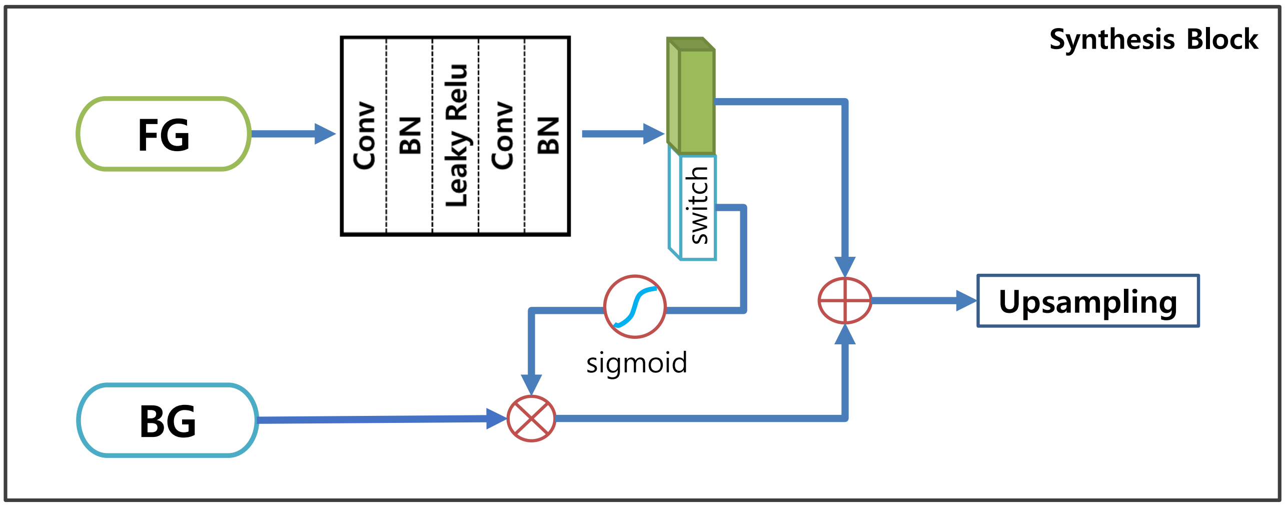


Figure 3.6: Synthesis blocks proposed in [91]. Source: [91]

The style transfer task using MC-GAN occurs as follows. First, the generator encodes the textual input t using the pre-trained text network φ. The text embedding is concatenated with a noise vector z and both are processed via fully-connected layers; the result is the seed feature map. Following that, they apply a series of synthesis blocks, whose inputs are the seed feature map and the image features from the background image. To train this network, the authors use the same loss function as in [24], but instead of using the traditional binary cross-entropy loss, they use the Least-Squares loss [76].

In order to compare the results with the baseline [24], they conducted experiments on the Caltech-UCSD Birds 200-2011 dataset [125] and the Oxford 102 Category Flowers dataset [88], using 4 synthesis blocks and synthesizing images of size 64×64. Figures 3.7 and 3.8 show the results of MC-GAN on the birds and flowers datasets, respectively. For evaluation, the authors qualitatively compared their results with the baseline of Dong et al. [24], and concluded that their method was able to better maintain the background information and the object's shape.

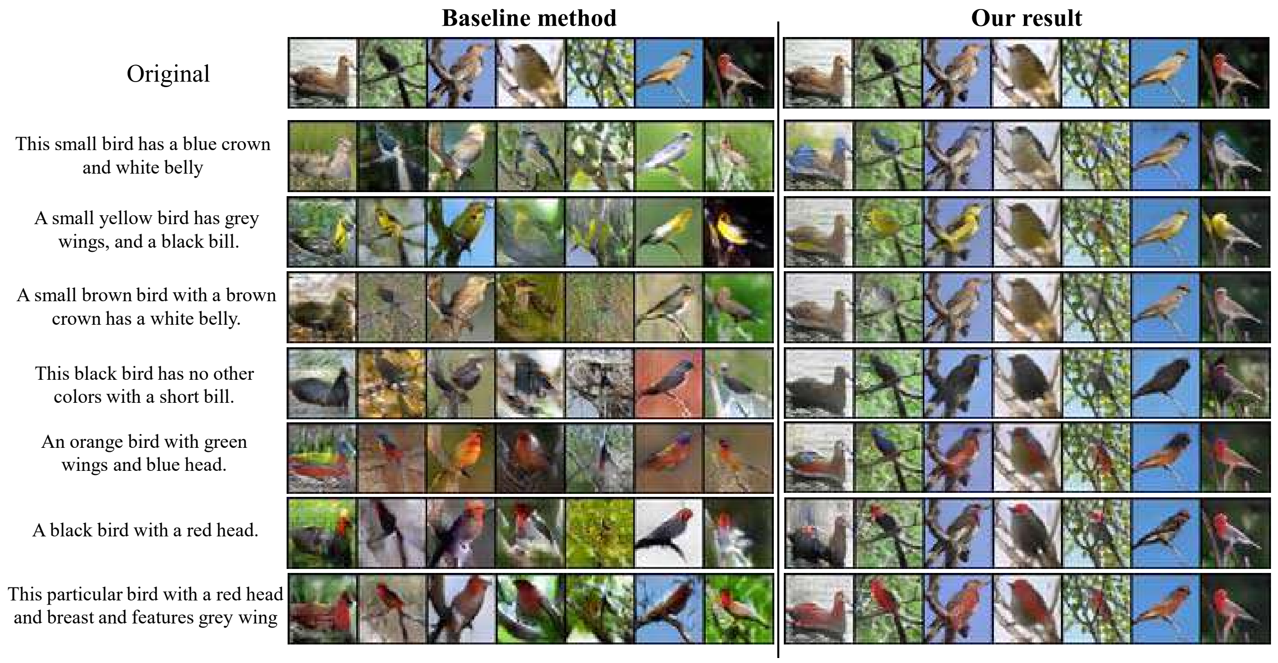

Baseline method
Our result
Original
This small bird has a blue crown and white belly
A small yellow bird has grey wings, and a black bill.
A small brown bird with a brown crown has a white belly.
This black bird has no other colors with a short bill.
An orange bird with green wings and blue head.
A black bird with a red head.
This particular bird with a red head and breast and features grey wing

Figure 3.7: Results for the method proposed by by Park et al [24] on the birds dataset. Source: [91]

3.4 Semantic Image Synthesis via Conditional Cycle-Generative Adversarial Networks

Liu et al. [74] argue that other approaches for the task of semantic image synthesis focus on making sure that the characteristics that are present in the text descriptions are preserved in generated images, while ignoring the structural information in the original images. Hence, even modifications in attributes that are not specifically called for in the descriptions may suffer alterations (e.g., background, shape, pose, or style). To prevent undesired structural and attribute modifications on the generated images, the authors propose a novel training procedure for the task of semantic image synthesis model called Conditional Cycle-Generative Adversarial Network (CCGAN), depicted in Figure 3.9.

CCGAN draws inspiration from unpaired image-to-image translation literature [138] and attempts to solve the task by learning two mapping functions (neural networks) between two domains, X and Y: $G : X \rightarrow Y$ and $F : Y \rightarrow X$. These networks adopt the same architecture but do not share weights, and each of them is paired with an adversarial dis-

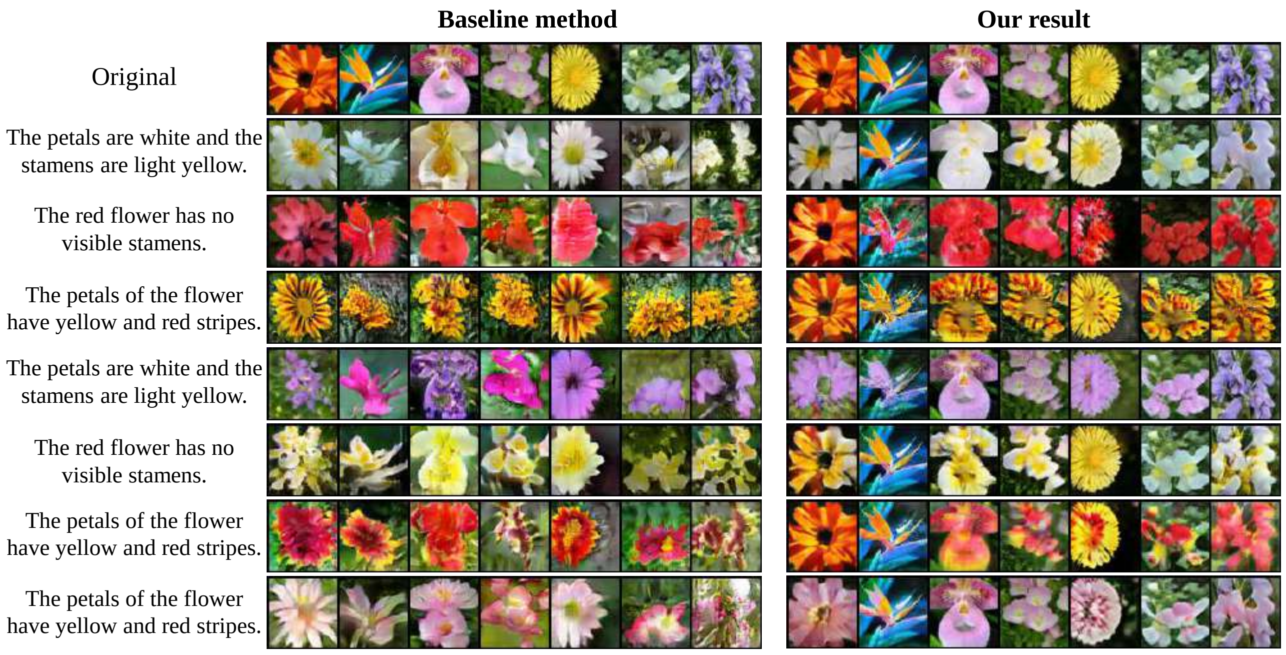

Baseline method
Our result
Original
The petals are white and the stamens are light yellow.
The red flower has no visible stamens.
The petals of the flower have yellow and red stripes.
The petals are white and the stamens are light yellow.
The red flower has no visible stamens.
The petals of the flower have yellow and red stripes.
The petals of the flower have yellow and red stripes.

Figure 3.8: Results for the method proposed by by Park et al [24] on the flowers dataset. Source: [91]

criminator, D_X and D_Y, which are also conditioned on the text descriptions. The adversarial losses can, in theory, learn mappins G and F that produce outputs for the target domains X and Y, respectively. However, it is not easy to guarantee that the networks will map an individual input x_i to a desired output y_i. Thus, to further reduce the space of possible mapping functions, one can add a cycle-consistency constraint to the optimization procedure. Cycle-consistency states that, for each image x from domain X, the image translation cycle should be able to bring x back to the original image, i.e., $x \to G(x) \to F(G(x)) \approx x$. This is called a *forward cycle consistency*. A *backward cycle consistency*, as expected, states that $y \to F(y) \to G(F(y)) \approx y$. On a conditional setting, this behavior is incentivized using a conditional cycle-consistency loss:

$$\begin{aligned}\mathcal{L}_{cyc}(G, F) = \mathbb{E}_{(x,\bar{t}_y,t_x)\sim p_{\text{data}}}[||F(G(x, \varphi(\bar{t}_y)), \varphi(t_x)) - x||_1] \\ + \mathbb{E}_{(x,\bar{t}_x,t_y)\sim p_{\text{data}}}[||G(F(x, \varphi(\bar{t}_x)), \varphi(t_y)) - y||_1]\end{aligned} \tag{3.4}$$

The adversarial loss for both G and F in CCGAN are the same as in [24]; each network has its own discriminator. The final loss optimized is, thus, the combination of two adversarial losses and a cycle-consistency loss:

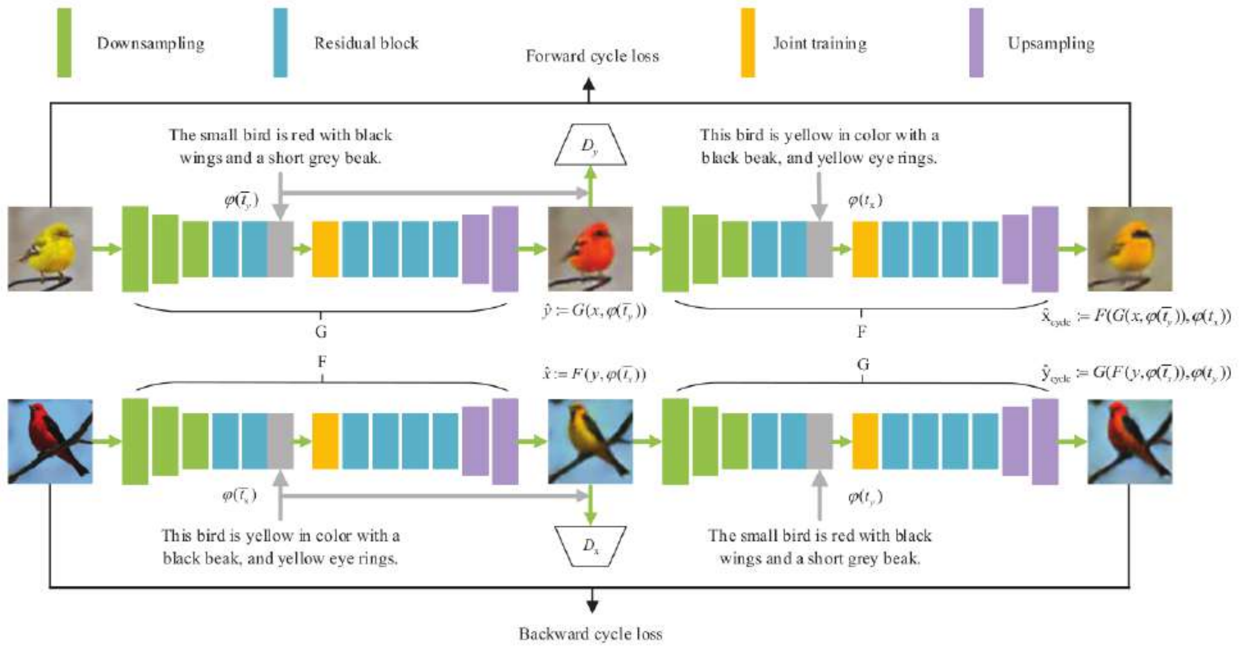

Downsampling
Residual block
Forward cycle loss
Joint training
Upsampling
The small bird is red with black wings and a short grey beak.
This bird is yellow in color with a black beak, and yellow eye rings.
Backward cycle loss

Figure 3.9: Conditional Cycle-Generative Adversarial Network (CCGAN) training, as proposed in [74]. Source: [74]

$$
\begin{aligned}
\min_{G,F} \max_{D_X,D_Y} V(G,F,D_X,D_Y) = {} & \mathcal{L}_{GAN}(G,D_Y,X,Y) \\
& + \mathcal{L}_{GAN}(F,D_X,X,Y) \\
& + \lambda \mathcal{L}_{cyc}(G,F)
\end{aligned}
\tag{3.5}
$$

where λ regulates the weight of the cycle-consistency loss. The text embeddings and architectures for generators and discriminators in the experimental setup are identical to [24]. Figures 3.10 and 3.11 show the results for CCGAN in the birds and flowers datasets, respectively.

The authors performed both quantitative and qualitative evaluations, comparing their approach with the approaches in [97] and [24]. The qualitative evaluation was a visual inspection, while the quantitative evaluation was done via human evaluation. They manipulated 20 images with 20 textual descriptions using all methods and recruited 20 participants, who were asked to rate the images on a scale from 1 to 5 on three categories: (i) whether the features (e.g., style, shape, pose, background) of the original images are retained; (ii) whether the generated image is consistent with the semantic information of the text description, and (iii) the quality/realism of the generated image. On both evaluations, the authors report that their method is superior to the baselines.

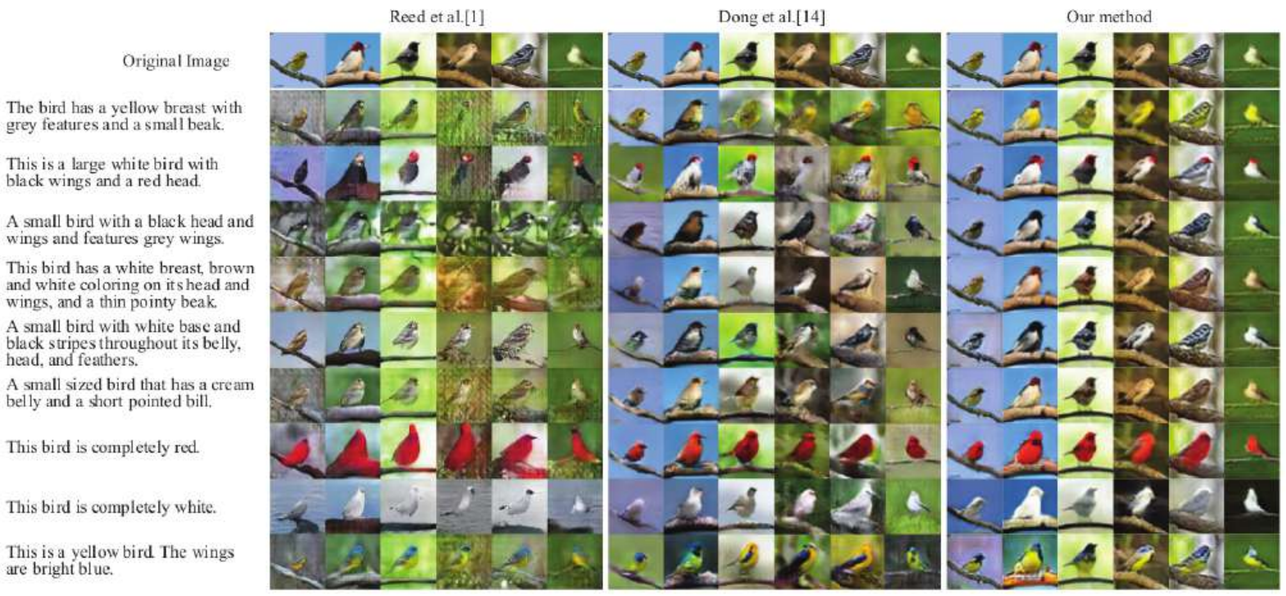

Reed et al.[1]
Dong et al.[14]
Our method
Original Image
The bird has a yellow breast with grey features and a small beak.
This is a large white bird with black wings and a red head.
A small bird with a black head and wings and features grey wings.
This bird has a white breast, brown and white coloring on its head and wings, and a thin pointy beak.
A small bird with white base and black stripes throughout its belly, head, and feathers.
A small sized bird that has a cream belly and a short pointed bill.
This bird is completely red.
This bird is completely white.
This is a yellow bird. The wings are bright blue.

Figure 3.10: Results for the method proposed by by Liu et al [74] on the birds dataset. Source: [74]

3.5 Text-Adaptive Generative Adversarial Networks: Manipulating Images with Natural Language

Nam et al. [85] make the following observation: in previous approaches [97, 24] for text-based image manipulation, the sentence encoder is pre-trained on the task of visual-semantic similarity, which approximates pairs of images and sentences to the same point in a multidimensional space. Although these approaches are moderately succesful at solving the task, they are likely to generate a new image conditioned on the pose of the original image, instead of modifying only the parts that are described in text. The authors postulate that this behavior occurs mainly because, during the adversarial training, the discriminator is trained conditioned on the entire sentence; the result is that it is only capable of providing coarse feedback to the generator.

To overcome the limitation imposed by the sentence-level textual features, the authors propose a new method called Text-Adaptive Generative Adversarial Network (TAGAN), depicted in Figure 3.12a. Instead of using a single sentence-level discriminator, they introduce the text-adaptive discriminator, which consist of word-level local discriminators. This new discriminator architecture classifies an image as real of fake by aggregating the scores of all word-level outputs. With this new approach, the discriminator can provide fine-grained training feedback to the generator, which can now learn to change only parts of the image while preserving other contents.

The generator architecture, much similar to previous approaches [24, 74], follows an encoder-decoder format, with residual blocks with skip-connections in the middle. The image is encoded, concatenated with the text features, and processed through the resid-

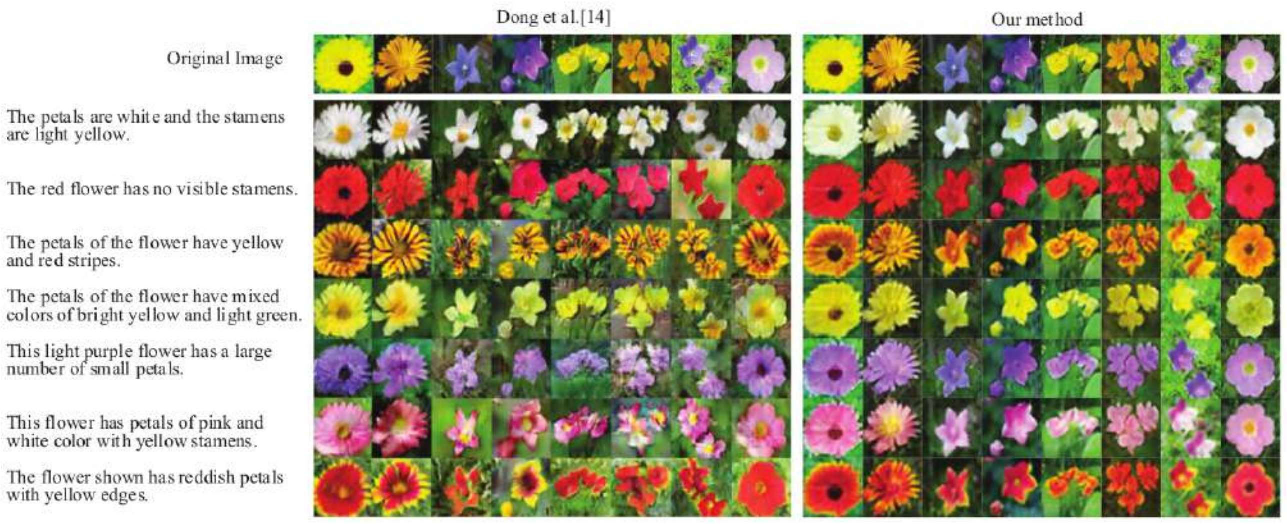

Dong et al.[14]
Our method
Original Image
The petals are white and the stamens are light yellow.
The red flower has no visible stamens.
The petals of the flower have yellow and red stripes.
The petals of the flower have mixed colors of bright yellow and light green.
This light purple flower has a large number of small petals.
This flower has petals of pink and white color with yellow stamens.
The flower shown has reddish petals with yellow edges.

Figure 3.11: Results for the method proposed by by Liu et al [74] on the flowers dataset. Source: [74]

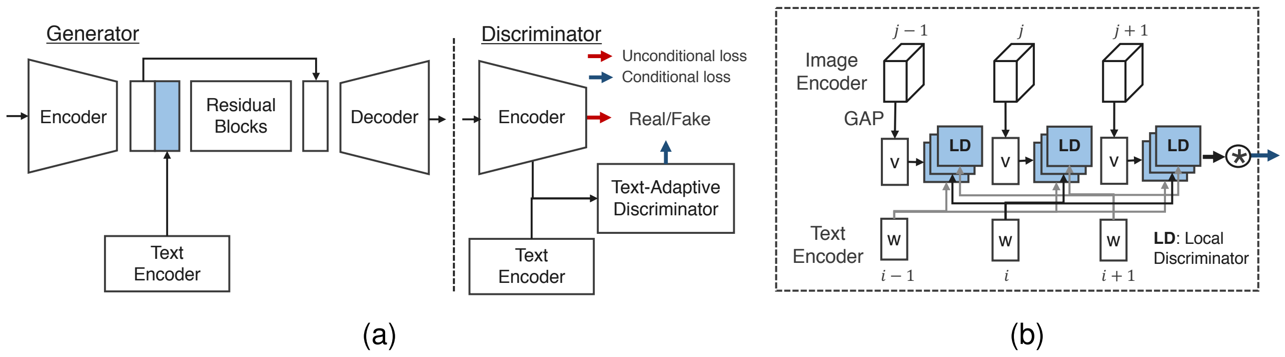


Figure 3.12: The GAN structure proposed in [85]. (a) shows the overall GAN architecture and (b) depicts the text-adaptive discriminator. In (b), the attention and the layer-wise weight are omitted for simplicity. Source: [85].

ual blocks and decoder. For the text representation in the generator, the authors use a bidirectional RNN to encode the whole text, trained from scratch with no pre-training. They also adopt the conditioning augmentation method [135], to add diversity and improve feature quality.

The text-adaptive discriminator, illustrated in Figure 3.12b, is also trained with its own text encoder. For each vector $\mathbf{w}_i$, which is the i-th output of the text encoder, we create a 1D sigmoid local discriminator $f_{\mathbf{w}_i}$, which determines whether a visual attribute related to $\mathbf{w}_i$ exists in the image. Formally, $f_{\mathbf{w}_i}$ is described as:

$$f_{\mathbf{w}_i}(\mathbf{v}) = \sigma(\mathbf{W}(\mathbf{w}_i) \cdot \mathbf{v} + \mathbf{b}(\mathbf{w}_i)), \tag{3.6}$$

where $\mathbf{W}(\mathbf{w}_i)$ and $\mathbf{b}(\mathbf{w}_i)$ are the weight and the bias dependent on $\mathbf{w}_i$, while $\mathbf{v}$ is an 1D image vector computed by applying global average pooling to the feature map of the image encoder. The final classification score is computed by adding word-level attentions, to reduce the impact of less important words to the final score. The attention consists of softmax values across T words, which is computed by:

$$\alpha_i = \frac{\exp(\mathbf{u}^T\mathbf{w}_i)}{\sum_i \exp(\mathbf{u}^T\mathbf{w}_i)}, \tag{3.7}$$

where $\mathbf{u}$ is a temporal average of $\mathbf{w}_i$. The final score is computed according to the following formulation:

$$D(\mathbf{x},\mathbf{t}) = \prod_{i=1}^{T}[f_{\mathbf{w}_i}(\mathbf{v})]^{\alpha_i}. \tag{3.8}$$

Instead of learning sentence-level correspondence, the discriminator is trained to first identify individual attributes in a sentence, and then to find the existance of each attribute in an image. The authors additionally consider multi-scale image features to make some attribute detectors focus on small-scale features and other focus on large-scale features. Therefore, the final output for the text-adaptive discriminator is written as:

$$D(\mathbf{x},\mathbf{t}) = \prod_{i=1}^{T}[\sum_j \beta_{ij} f_{\mathbf{w}_i,j}(\mathbf{v}_j)]^{\alpha_i}, \tag{3.9}$$

where $\mathbf{v}_j$ is the image vector of j-th layer, and β_{ij} is a softmax weight that determines the importance of the layer j for each word $\mathbf{w}_i$. The discriminator computes both an unconditional output $D(\mathbf{x})$ and conditional output $D(\mathbf{x},\hat{\mathbf{x}})$. The network is trained by alternatively minimizing both the discriminator and the generator objectives described as:

$$\begin{aligned} L_D = {} & \mathbb{E}_{\mathbf{x},\mathbf{t},\hat{\mathbf{t}}\sim p_{data}}[\log D(\mathbf{x}) + \lambda_1(\log D(\mathbf{x},\mathbf{t}) + \log(1 - D(\mathbf{x},\hat{\mathbf{t}})))] \\ & + \mathbb{E}_{\mathbf{x},\hat{\mathbf{t}}\sim p_{data}}[\log(1 - D(G(\mathbf{x},\hat{\mathbf{t}})))], \end{aligned} \tag{3.10}$$

$$L_G = \mathbb{E}_{\mathbf{x},\hat{\mathbf{t}}\sim p_{data}}[\log D(\mathbf{x}) + \lambda_1 \log D(G(\mathbf{x},\hat{\mathbf{t}}),\hat{\mathbf{t}})] + \lambda_2 L_{rec}, \tag{3.11}$$

where λ_1 and λ_2 control the importance of additional losses, and $\hat{\mathbf{t}}$ is randomly sampled from a dataset regardless of $\mathbf{x}$. L_{rec} is an additional reconstruction loss when a positive text is given, which enforces the generator to reconstruct the text-irrelevant contents from the input image instead of generating new contents: $L_{rec} = ||\mathbf{x} - G(\mathbf{x},\mathbf{t})||$.

The resolution of the generated images was 128×128, and the authors conducted experiments on both the Caltech-UCSD Birds 200-2011 dataset [125] and the Oxford 102 Category Flowers dataset [88]. The quantitative analysis involved human evaluation via crowdsourcing (20 workers). They randomly selected 10 images and 10 captions from the test set and produced 200 outputs for each dataset and for each method (since the images in [24] were 64×64, all images were resized to that resolution). Workers were asked to rank three results after looking at both input image/text and outputs, based on the following criteria: (i) whether the visual attributes (colors, textures) of the manipulated image match the text, and the background irrelevant to the text is preserved, and (ii) whether the manipulated image looks natural, and visually pleasing. They additionally compared the quality of content preservation between methods by computing an L_2 reconstruction error by forwarding images with matching texts. Analyzing the results, they identify that TAGAN obtained superior results. Figure 3.13 shows sample images generated by TAGAN for both the birds and flowers datasets.

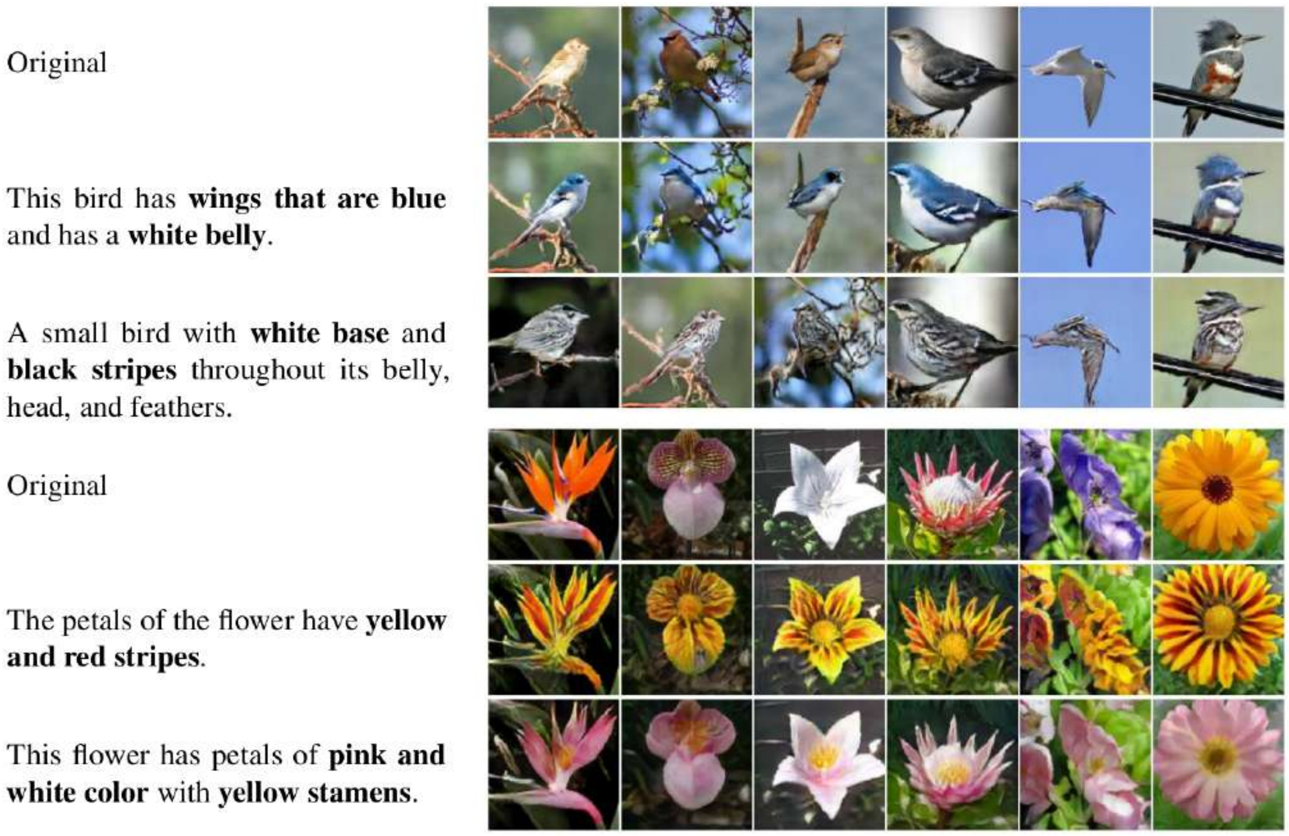

Original
This bird has **wings that are blue** and has a **white belly**.
A small bird with **white base** and **black stripes** throughout its belly, head, and feathers.
Original
The petals of the flower have **yellow and red stripes**.
This flower has petals of **pink and white color** with **yellow stamens**.

Figure 3.13: Results for the method proposed by by Nam et al [85] on the birds and flowers dataset. Source: [85]

3.6 Image Manipulation on the Fashion Domain

In this section, we mention studies that also propose solutions to the text-based image manipulation task, but within another domain: fashion images. Figure 3.14 illustrates an example. Given that the datasets are very different, some problems are alleviated while others are intensified. Nevertheless, it is important to cite them as to better contextualize the state-of-the-art and inspect the landscape of the text-based image manipulation task.

Zhu et al. [139] construct a GAN-based, two-step training method for generating new clothing on an image of a person, based on textual descriptions. To do so, the authors extend a subset of approximately 79000 images from a fashion image dataset by collecting textual descriptions and segmentation maps. To capture further information about the wearer, they extract a vector of binary attributes from the person's face, body, and other physical characteristics (e.g., gender, long/short hair, wearing/not wearing sunglasses). Then, they generate a design coding vector comprised of the binary atttributes and the textual description. On the first stage, a network generates a semantic segmentation map for the original image based on a noise vector, a spatially constrained version of the ground-truth segmentation map, and the design coding, separating the wearer's body parts (e.g., arms, head, torso) and upper-body garment. On the second stage, a network renders the new image based on a noise vector, the previously generated segmentation map, and the design coding. Each stage's network is trained separately.

Gunel et al. [44] also explore the text-based image manipulation task on the fashion domain. They believe that using segmentation maps as an auxiliary guidance to the model, as in [139], may introduce visual inconsistencies between the generated output and the input image, since the output is not generated in a holistic manner. Thus, they propose a new model called FiLMedGAN, which incorporates Feature-wise Linear Modulation (FiLM) [93] to better guide the manipulation process without the use of segmentation maps. The architecture of the generator is a slight modification of the architecture in [24]; instead of depth-concatenating the text features to the image feature map, they modulate the image features using FiLM layers.

Similarly to [139], Rostamzadeh et al. [99] introduce a fashion dataset of 293,008 high-definition fashion images paired with item descriptions (provided by professional stylists). Each item on the dataset is photographet from a variety of angles against a standardized background under consistent lighting conditions. This dataset was introduced for a competition on text-to-image synthesis, but could be used for text-based image manipulation in the future.

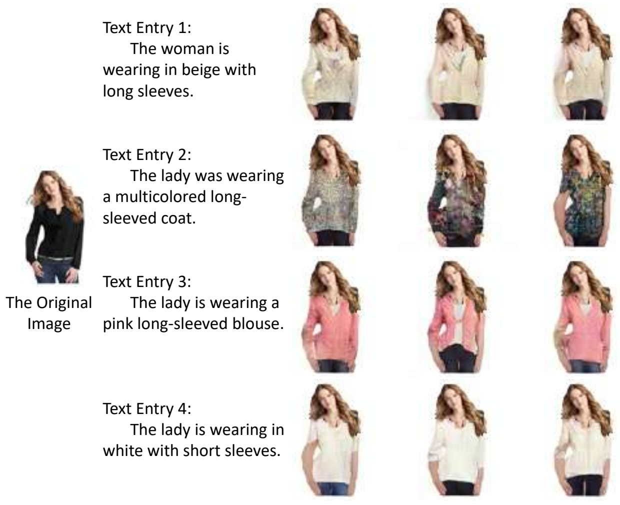

Text Entry 1:
The woman is wearing in beige with long sleeves.
Text Entry 2:
The lady was wearing a multicolored long-sleeved coat.
Text Entry 3:
The lady is wearing a pink long-sleeved blouse.
Text Entry 4:
The lady is wearing in white with short sleeves.
The Original Image

Figure 3.14: Example of the text-based image manipulation task on the fashion domain. Source: [139]

3.7 Image Manipulation: Miscellaneous

In this section, we list some studies that also manipulate images using natural language, but solve different tasks (e.g., image colorization and global image editing), propose non-deep learning approaches (i.e., traditional NLP), or use alternative text formats, such as texts describing the difference between two images or dialogue-based systems. Although not directly related, these studies provide insights on how related tasks are tackled and present new perspectives on how to deal with the difficult aspects of the task at hand.

Zou et al. [140] propose LUCSS, a GAN-based framework for language-based system for colorizing image sketches. This framework consists of three modules: instance segmentation, captioning, and colorization. First, the input sketch is passed through the instance segmentation module to extract object instances from the sketch. Next, the captioning module takes the output of the segmentation module and automatically generates a caption describing the input scene sketch. This caption can then be altered to include color information. Finally, the colorization module produces a colorized image using the input image and the generated caption. The colorization is decomposed into two instance and background colorization to alleviate the complexity of the task.

Wang et al. [127] present an end-to-end, GAN-based, textual image editing model to perform global image editing tasks, such as changing contrast, brightness, and so on. They construct a dataset using crowdsourcing by asking workers to textually describe the modification between an original image and a filter being applied to it. They propose three different ways to parse the textual information: a hand-crafted, bucket-based model, an end-to-end model, and a filter-bank model.

Chen et al. [17] investigate the problem of Language-Based Image Editing, an umbrella term (hypernym) for tasks where, given a source image and a natural language description, the goal is to generate a target image by editing the source image based on the description. They explore two sub-tasks of LBIE, language-based image segmentation and image colorization, using GANs and recurrent attentive models.

Manjunatha et al. [75] develop two fully-convolutional neural architectures to tackle the task of colorization using natural language. They treat the colorization as a classification problem in CIELAB color space: given only the lightness channel of an image (grayscale), the task is to predict values for the two color channels. The first architecture is a simple concatenation of text features to the image features; while the second one uses FiLM [93]. The results between both architectures are similar.

Laput et al. [67] state that speech interfaces can make complex image editing tasks more accessible to amateur editors because they allow users to state goals instead of learning the program interface. However, image editing is hard to perform with speech alone, because it is hard to communicate spatial locations accurately and consistently. Given that scenario, they develop a multimodal photo editing application, called PixelTone, that allows users to use natural language and sketching to define and localize changes. The natural language interpreter uses traditional NLP techniques and offers users a graceful fallback when it fails to parse the request. The NLP module processes speech and translates it directly into image editing operations, which are applied to the regions defined by the user.

Cheng et al. [18] propose ImageSpirit, a system that allows users to refine images using language by parsing images into regions with semantic labels. Typical verbal commands include correcting an object label and refining a specific label. Based on the initial image parsing result, ImageSpirit updates the local relationship between different objects in a Conditional Random Field (CRF) in response to the language input, resulting in the enhanced result. Similarly to [67], the language commands are predefined.

Mohapatra [81] argues that, although deep learning has proven to be very succesful in some tasks, there is still progress to be made before its application in fine-grained image editing. Thus, he proposes a framework to fo from free-form natural language commands to performing fine-grained image edits using traditional natural language processing (NLP) techniques. The input query is parsed to identify the entities, attributes, and relationships to generate a command entity representation. Next, the semantic command entity representations are mapped into lower-level operations via an intermediate programming language to carry out the intended execution.

Shinagawa et al. [110] propose an interactive image-manipulation system with natural language instruction, which can generate a target image from a source image and an instruction that describes the difference between the source and the target image. The manipulation is performed by encoding image and textual instructions into latent space and obtaining a latent representation of the result via the additive property, i.e., *source image* + *instruction* = *target image*. They constructed a synthetic dataset based on MNIST [69] by manually creating manipulated versions of images based on a fixed instruction verb, position, and direciton sets. As such, this study can be viewed as a transition between traditional NLP and deep learning. On a folowup study [109], the authors propose to manipulate avatar images and propose a Source Image Masking (SIM) system to prevent unwanted changes to the original image (i.e., changes that were not requested via text). The avatar dataset was collected in a similar fashion, but this time the text modifications were acquired via crowdsourcing instead of automatically generated based on fixed rules.

Cheng et al. [19] explore text-based image editing via conversational language, where an user can guide an agent to edit images via multi-turn dialogue in natural language. At each step, the system takes as input a source image and a target modification described in textual form, outputting an image with the desired modification. They collect two datasets to test their system, one for footwear and the other for clothing, using crowdsourcing. Workers were presented with two images and were tasked with describing the difference from one image with respect to the other. Thus, the model is trained to transform an image onto another one based on the differences described. The models are based on GANs and attention. Sometimes, the text describing the modification may cover only the most important differences between two images, which may result in undesired changes on the manipulated image.

3.8 Overview

In this section, we will present a summarization of studies that explore the task of text-based image manipulation. We briefly discuss how they deal with text representations, which datasets are commonly used, common architectural trends, how they assess the quality of images, and the specific advantages and disadvantages of the core methods.

3.8.1 Text Representation

Regarding textual representations, there are different methods for representing the modifications. Some studies pose the task differently, by having the text describe the difference between images [110, 109, 19, 127, 67, 18, 81], or by breaking the modification into separate steps using a dialogue-based system [19]. However, the common trend is to use descriptions that list the characteristics of the entire image for modifications [97, 24, 91, 74, 85, 139, 44, 140, 17]. We believe that two of the main reasons for this is that it makes it easier to collect the dataset and use it during training (if we have captions for all images, we are not linked to specific pairs of images, which means we can pair any two images on the dataset).

The textual description representations are usually trained using a multimodal approach [65, 96], which helps the network link the concepts described in text with the content of images. Among the main studies, TAGAN [85] is the only work that trains their text embeddings from scratch. These text descriptions are usually combined with the image feature map via depth-concatenation. The exceptions are GAN-INT-CLS [97] (whose network was designed initially for text-to-image synthesis) and MC-GAN [91] (which proposes a different way of manipulating images via synthesis blocks). SISGAN, MC-GAN, and TAGAN use conditioning augmentation [135] to improve the text embeddings by smoothing the embedded space and adding diversity to samples.

3.8.2 Datasets Used

Regarding datasets, several authors have proposed the creation of new datasets for the task. However, the predominant datasets for text-based image manipulation are the Oxford 102 Category Flowers dataset [88] and the Caltech-UCSD Birds 200-2011 dataset [125]. The fashion datasets are also commonly used, but there are no studies that experiment with both birds/flowers and fashion datasets. The MS-COCO [73] dataset also contains descriptions for its images, but it has not been explored for text-based image manipulation so far. A possible reason is that the dataset is much more complex: sometimes there are several objects per image, the classes of objects vary a lot, and the captions do describe object's details that much.

3.8.3 Architecture

Next, we analyize the similarities between the architectures of the methods for the birds and flowers datasets. Dong et al. [24] were the first to focus on the task of text-based image manipulation. The discriminator architecture follows the guidelines proposed in [95] and is relatively straightfoward: first, they apply several stride-2 convolutions followed by batch normalization and leaky-ReLU activation. Then, the text features are concatenated and processed using the same pipeline as above (normalization is not applied on the first and last layers, and the output layer has no activation).

They draw inspiration from [59] and [135] to design their generator architecture, which comprises of an encoder, a decoder, an a residual transformation unit. The encoder is a fully-convolutional network that encodes the image to a spatial representation, and follows the design recommendations of [95]. Each layer consists of a stride-2 convolution operation followed by batch normalization and ReLU activation (except for the first layer, which has no normalization). The residual transformation unit consists of several residual blocks [46] and processes the concatenation of text features and the spatial features of the image. Finally, the decoder is a series of nearest-neighbor upsampling layers followed by convolution, batch normalization, and ReLU activation (except for the last layer, which does not use normalization and the activation function is $tanh$). This architecture has several benefits:

- By processing the image using an encoder and keeping a spatial feature map, we can somewhat retain the strucure of the original image. This is an appealing property since in most cases the output image should share structure with the input image.

- The residual transformation unit makes the task of learning the identity function (i.e., to not change an image) much easier [46].
- The “bottleneck” on the encoder-decoder architecture reduces computational cost (which allows us to use a larger network) and increases the receptive field size to perform modifications on the residual unit.

The generator architecture proposed by [24] has been used in some of the other approaches [74, 85, 44], and, due to the properties listed above, appears to be a very efficient architecture for this task. GANs are capable of generating high-resolution images (2048×1024), such as in [128]. However, the resolution targeted for text-based image manipulation is usually 64×64 or 128×128, which indicates that the goal so far is improving the understanding of the task, and not high-definition. The most commonly used adversarial loss is the original proposed by [39]; MC-GAN is the only one that deviates and uses the LSGAN [76] loss instead.

3.8.4 Evaluation

Current datsets used by text-based image manipulation methods usually do not contain ground-truth information of which transformation is considered correct. This makes evaluating and comparing approaches a difficult task. Qualitative evaluation is normally done by visually inspecting the results and comparing them with other approaches. This includes a simple visual inspection, interpolation of results to check for smoothness between samples, and sample variety check (generate different results given the same text).

Quantitative evaluation is usually done using human judgement (ranging from 10 to 20 subjects) by asking them to analyze synthesized images based on several criteria. The criteria used is established by the authors and is somewhat standardized. The assessment can be comparative (by ranking approaches from best to worst) or based on score (e.g., from 1 to 5), and includes: (i) whether the synthesized image is visually pleasing; (ii) whether the aspects not present in text are retained, and (iii) whether the synthesized image matches the text description provided. TAGAN also evaluates the quality of content preservation by computing an L_2 reconstruction error by forwarding images with positive text (i.e., an image should remain the same using the original description).

3.8.5 Comparison

In this section, we will compare the core work of text-based image manipulation while restraining ourselves to the birds and flowers domains. We qualitatively assess the results of the different methods proposed in literature, highlighting their pros and cons and comparing them amongst each other.

Reed et al. [97] were the first to succesfully perform the task of text-based image manipulation. To do so, their architecture requires two steps: encoding the image into latent space and then passing it through the generator network with the desired text modification. While this approach can produce good results, most of the images suffer significant alteratioons in background and object pose. Since they encode the original image into an embedding space that is small, it is hard to keep all spatial and background information intact. Thus, it is hard for the generator to accurately reconstruct all of the aspects of the original image. Besides, many of the images contain distortions, such as checkerboard effects and an overall blurriness.

SISGAN [24] attempts to fix the problems of the original approach via a one-step pipeline that processes the original image while keeping some of its spatial features intact. Additionally, it uses nearest-neighbor upsampling instead of transposed convolutions on the decoder; this small switch is known for helping reduce the checkerboard artifacts caused by the transposed convolution [89]. The results are significantly improved in terms of detail, but there is still some blurriness and background modification. Additionally, if the background is too complex, the method may fail to generate a plausible image. Due to the way the network is trained, the shape of some objects are unintentionally modified: this method is likely to generate a new image conditioned on the pose of the original image instead of modifying only the parts that are described in the text. In some cases, the method generates a similar image because the sentence is highly correlated to a particular class of object. Thus, the object may be modified to have the appearance of the closest class that is described by the modification description.

MC-GAN [91] uses the same training procedure as [24] but switches the architecture to use synthesis blocks. The results for flowers are slightly better at keeping the original object's pose/class and have less background variation. On the birds domain, the method keeps background information and original object pose much more consistently, but the details are much less vivid (all birds have a brownish tone and very few colored parts).

CCGAN [74] follows the opposite route of MC-GAN: it uses the same architecture as [24] and attempts to fix the unintentional object modifications by modifying the training procedure. For birds, the results are much sharper and very frequently contain fine details (e.g., colorizing multiple parts differently at the same time), which is a significant improved when compared to SISGAN and MC-GAN. The results for flowers are better at keeping the original object's pose/class and have less background variation, but the modifications are much more focused on overall color and less on fine details and textures.

TAGAN [85] uses the same generator architecture as [24] but is the first to synthesize images on a resolution of 128×128. It uses a multi-step discrminator that processes words individually instead of using complete sentences. This allows the discriminator to provide fine-grained feedback to the generator and focus on individual visual attributes instead of a combination of them. This approach has the best results so far in terms of colors, textures, and shape preservation, on both the birds and flowers datasets. However, it still fails to generate the exact shape described in the text. This can be explained by inspecting the loss function: it presents a trade-off between generating new content (adversarial loss) and preserving original content (reconstruction loss). While this can be controlled by tuning loss multipliers, this may compromise the overall quality of the method.

4. TEXT-BASED IMAGE MANIPULATION USING STYLE TRANSFER

We are concerned with exploring natural language as an input modality for an automatic image manipulation task. Specifically, we wish to alter an image according to a specification provided in natural language form. This task, as we have seen in Chapter 3, has been referred to using several names, such as style transfer[1], semantic image manipulation, semantic photo editing, language-based image editing (LBIE), and text-based image manipulation. In this chapter, we present the approach we developed for this task. We start by describing which types of natural language input our approach will deal with, and then we provide a brief introduction to *Neural Style Transfer* (NST) literature. After that, we provide a detailed description of the workings of our method.

4.1 Natural Language Input

Natural language offers a general and flexible interface for describing objects and modifications in any space of visual categories. This adaptability makes it easier to use language across different domains and tasks. However, this freedom of expression comes at a cost; language can be ambiguous and is a weaker source of supervision, which means using it requires solving numerous sub-problems that are absent (or are much easier to solve) on other automatic image editing input types. We present a non-exhaustive list of potential sub-problems that must be solved to perform text-based image manipulation:

- **Text Understanding**: we must parse the textual input to understand what are the objects and modifications that are being referred to in the text. Natural Language Processing is a hard problem on its own, with an entire subfield dedicated to it.
- **Localization**: we must locate the objects (or parts) we wish to modify in the original image provided by the user. This should preferably happen at a pixel level. The computer vision and deep learning research communities have already produced several studies regarding this specific topic on its own.
- **Image Generation**: we must combine textual and image information to understand which aspects must change (or remain the same) to generate a manipulated image that conforms to the specification and is also realistic. Image generation is also a heavily studied task in deep learning.
- **Unwanted Modifications**: there can be multiple ways to describe a modification, some more verbose and precise than others. The sentence provided can be straight to the

[1]This was the name given in [97], when the task was originally conceived. However, this name can be ambiguous because there is another task that goes by the same name for style transfer between images.

point and describe only the modification, or it can describe the entire object and the parts that the user wishes to change. This variability can cause ambiguity and result in unwanted modifications.

- **Multiple Modifications/Sentences**: the user may request several modifications on the same sentence, or provide multiple sentences, each with its own modification. Thus, we must detect all changes and either combine them or apply them sequentially.
- **Unmodeled Modifications**: sometimes, the desired modification is not modeled by the data that we are using to train the models (e.g., a color not present in the dataset). Preferably, the system should detect that this is the case, to either alert the user or fail gracefully.

Given all these potential problems, using unbounded natural language as input for text-based image manipulation becomes extremely hard. Thus, it is wise to impose some form of limitation on the form of the manipulation sentences we use. As we have seen in Chapter 3, researchers have posed the problem in several ways, using different datasets and approaches. Regarding the textual input, the vast majority of studies use datasets with sentences that describe the characteristics of the objects in the scene to train their models. These sentences usually contain multiple modifications, which makes it more difficult to detect and combine them; this often results in unwanted modifications and some modifications not being performed at all. However, several datasets [88, 125, 73, 139] contain pairs of real images and sentences of this type, and the alternative datasets [110, 109, 17] are either comprised of synthetic images (which limits applicability) and/or contain few instances (which hinders training). Thus, we will focus on devising approaches using datasets with sentences describing the whole image.

4.2 Style Transfer

In fine art, a *pastiche* is an artistic work that imitates the style or character of the work of other artists. In painting, specifically, people have been creating pastiches for hundreds of years, to celebrate renowned artists and to compose new paintings based on a specific style. Recently, there has been a growing interest in computer vision and machine learning research to automate this creative process, called *style transfer*, where the goal is to synthesize an image that combines the content of one image with the style of another. Figure 4.1 shows an example of style transfer for paintings.

Understanding the algorithmic basis of this process, however, is a hard problem. The style transfer task is inherently ill-posed: the concepts of image content and style are vaguely defined, and thus are hard to completely disentangle. Additionally, there are no clear answers to questions such as [30]:

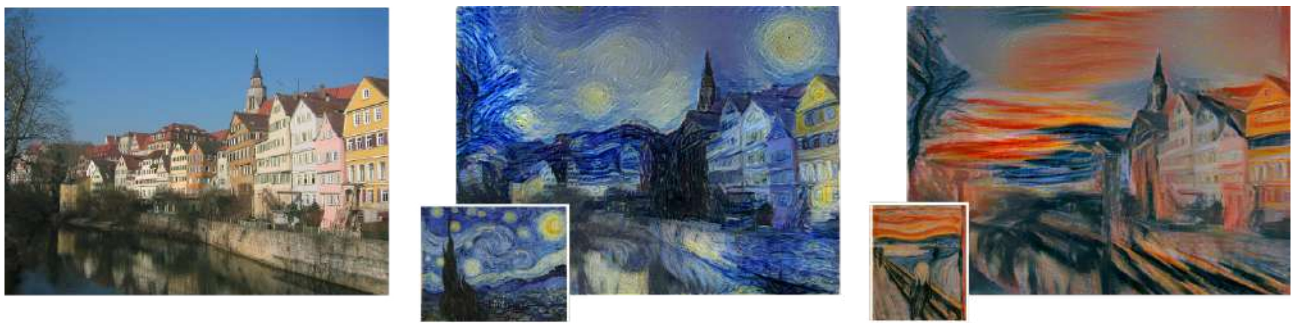


Figure 4.1: Example of style transfer for paintings. The image on the left is the content image and the following pairs of images are the style image used (bottom left) and the style transfer result. Adapted from [35].

- Which parts of the content should be preserved, discarded, and modified?
- Should we modify the content image's contrast as part of the transfer?
- Should edges and other elements in the content image be allowed to shift (and how)?
- Which color palette should the output adopt?
- Which parts of the style image should be used as style?
- How do we differentiate between copying and hallucination of style?[2]

This ultimately leads us to the conclusion that, for now, there is no single correct output for this task. Thus, the style transfer task can be interpreted, addressed, and have its success determined in various ways.

In computer vision, style transfer also goes by the name of texture transfer, and is closely related to the texture synthesis task. While the latter tries to understand the statistical relationship between the pixels of a source image to create textures, the former does so while also preserving some notion of content from another image. There are several non-deep learning studies tackling style transfer [29, 47, 30]. However, they tipically have limitations in flexibility, style diversity, and effective image structure extractions [58].

Recently, deep learning has emerged as an alternative path to solve style transfer (and texture synthesis), presenting incredible results. Typical NST approaches combine texture synthesis methods and the knowledge contained in trained neural networks to extract the relevant information required for the task. Solving style transfer using a neural approach presents many advantages over traditional solutions, such as the ability to transfer and combine increasingly complex styles while not requiring hand-crafted rules, feature specification, or ground-truth data. Hence, we will focus on exploring deep learning-based methods for style transfer to perform text-based image manipulation.

[2]In this context, copying can be interpreted as replicating "patches" of style while hallucinating refers to a deeper understanding of style (i.e., knowing how to apply styles in previously unseen situations).

We present a brief evolution of NST while following the taxonomy proposed in [58]. We will center on methods that model the style of an image (Visual Style Modelling) by relating the concept of style to a texture (Visual Texture Modelling), using *Parametric Texture Modelling with Summary Statistics*. We will explore two categories of parametric NST: *Instance-Optimization Based Online Neural Methods* (IOB-NST), and *Model-Optimization Based Offline Neural Methods* (MOB-NST). The first category transfers the style by iteratively optimizing an image, while the second optimizes a model offline and produces the stylized image with a single forward pass.

After training, CNNs develop internal representations where features tend to correspond to higher levels of abstraction as we advance in the processing hierarchy of the network [1]. The authors in [34] show that these trained networks can be exploited to compute summary statistics over the features of their intermediate layers to obtain stationary, multi-scale descriptions that capture texture information while discarding the global arrangement. They develop a parametric texture modelling approach by using a Gram-based representation, which represents the correlations between filter responses in different layers of the network. This representation can be obtained by computing the *Gram matrix*. Let $\phi_j(x)$ be a feature map of size $C_j \times H_j \times W_j$ for a sample image x at layer j of a deep network ϕ, and $\phi_j(x)'$ be a reshaped version of $\phi_j(x)$ with size $C_j \times (H_j \times W_j)$ i.e., a vectorized feature map. Then, the Gram matrix $\mathcal{G}_j^{\phi}(x)$ of size $C_j \times C_j$ is given by:

$$\mathcal{G}_j^{\phi}(x) = [\phi_j(x)'][\phi_j(x)']^T. \tag{4.1}$$

Using this knowledge, the seminal work of Gatys et al. [35] introduced the study of neural style transfer by presenting the first IOB-NST method. The authors state that deep neural networks encode not only the *content* but also the *style* representation of an image. Furthermore, they show that content and style are somewhat separable, which can be used to perform style transfer between images. Given a content image c and a style image s, they iteratively optimize an image p (as in pastiche) to seek a stylized image p^* that jointly minimizes content and style loss functions:

$$\begin{aligned} p^* &= \arg\min_p \mathcal{L}_{st}(p, c, s) \\ &= \arg\min_p \lambda_c \mathcal{L}_c(p, c) + \lambda_s \mathcal{L}_s(p, s), \end{aligned} \tag{4.2}$$

where $\mathcal{L}_c$ compares content representations of p and c, $\mathcal{L}_s$ compares style representations of p and s, and λ_c and λ_s are used to balance the content and style components in the final style transfer loss, $\mathcal{L}_{st}$. Both losses are based on the representations from the feature responses of a *loss network* ϕ, which is a VGG Network [112] trained on image classification[3].

Rather than requiring an exact pixel match, the *content reconstruction loss* $\mathcal{L}_c$ encourages p to have a content similar to c by having similar feature representations; it is computed as the (squared, normalized) Euclidean distance between feature representations:

$$\mathcal{L}_c(p, c) = \sum_{j \in \mathcal{C}} w_j \left(\frac{1}{C_j H_j W_j} \|\phi_j(p) - \phi_j(c)\|_2^2 \right), \tag{4.3}$$

where $\mathcal{C}$ denotes the set of layers in ϕ used for computing the content loss and w_j indicates the weight (contribution) of layer j. Next, the *style reconstruction loss* $\mathcal{L}_s$, similarly to $\mathcal{L}_c$, encourages p to have a style similar to s by having similar feature representations. The style representation for each layer is computed as the Gram matrix $\mathcal{G}_j^{\phi}(x)$ (Equation 4.1). Then, the style reconstruction loss is calculated as the (squared, normalized) Frobenius norm[4] of the difference between the Gram matrices of the output and the target style:

$$\mathcal{L}_s(p, s) = \sum_{j \in \mathcal{S}} w_j \left(\frac{1}{C_j H_j W_j} \|\mathcal{G}_j^{\phi}(p) - \mathcal{G}_j^{\phi}(s)\|_F^2 \right), \tag{4.4}$$

where $\mathcal{S}$ denotes the set of layers in ϕ used for computing the style loss and w_j indicates the weight (contribution) of layer j.

The algorithm for neural style transfer then proceeds as follows: starting from some initialization of p (e.g., a random initialization), we adapt p to minimize $\mathcal{L}_{st}(p, c, s)$ via an iterative optimization based on gradient descent, given a content image c and a style image s. Thus, at each step, p is slightly modified to conform to the desired emphasis of content and style. A strong emphasis on style (λ_s) will result in an image that matches the appearance of s, effectively giving a texturized version of it, while hardly showing any of the content in c. Similarly, when placing strong emphasis on content (λ_c), one can clearly identify the content in c, but the style of s is not as well-matched. Note that both $\mathcal{L}_c$ and $\mathcal{L}_s$ are normalized by the size of the feature map, which allows us to better balance the importance of each layer independent of sizes when using multiple layers.

IOB-NST methods produce high-quality results and work across many styles, but are notorious for their computational cost: each time we wish to stylize an image, we must execute several optimization steps that require forward and backward passes through a network. The second category of methods, MOB-NST, attempts to address these speed and computational cost issues by using a style transfer network T to stylize images. To train this network, the authors use the same objective function as in [35], but instead of finding the best image, they search for the best set of weights to parameterize T:

[3]This network is trained on the ImageNet [102] dataset and, after that, it is freezed, meaning it does not update its weights during the style transfer task

[4]The Frobenius norm is equivalent to the Euclidean norm, generalized to matrix space.

$$\theta^* = \arg\min_\theta \mathcal{L}_{st}(T_\theta(c), c, s) \tag{4.5}$$

where θ represents the network weights. Depending on the number of styles T can produce, MOB-NST algorithms are further divided into *Per-Style-Per-Model* (PSPM), *Multiple-Style-Per-Model* (MSPM), and *Arbirary-Style-Per-Model* (ASPM) methods.

The first two PSPM-MOB-NST methods were proposed by Johnson et al. [59] and Ulyanov et al. [120]. These two methods share the same core idea, which is to train a style-specific feedforward network and produce a stylized result with a single forward pass at testing time. They only differ in network architecture, and the architecture proposed by Johnson et al. [59] is one of the most used ones up to date. The network contains an encoder-decoder architecture with fractionally-strided convolutions and residual blocks in the middle. Soon after, Ulyanov et al. [121] discovered that applying normalization to images separately rather than a batch of images (BN) leads to a significant improvement in stylization quality. They named this single image normalization as *Instance Normalization* (IN). One interpretation is that IN is a form of style normalization and can directly normalize the style of input images to the desired style [53]. Therefore, the objective is easier to learn as the rest of the network only needs to take care of the content loss.

There is, however, a clear trade-off between the IOB-NST and PSPM-MOB-NST methods. The significant gain in speed comes at the cost of flexibility, since the style transfer network is tied to a single style. But many paintings share similar paint strokes and only differ in color palettes, so it may be redundant to train separate networks for each style. MSPM-MOB-NST methods attempt to improve the flexibility of PSPM by incorporating multiple styles into a single model. Inspired by [121], Dumoulin et al. [27] find that using the same convolutional parameters while scaling and shifting parameters in IN layers is sufficient to model different styles in the same network. They propose to train a conditional style transfer network based on *Conditional Instance Normalization* (CIN), defined as:

$$\text{CIN}(\phi_j(x), s) = \gamma_s \left(\frac{\phi_j(x) - \mu(\phi_j(x))}{\sigma(\phi_j(x))} \right) + \beta_s \tag{4.6}$$

where μ and σ are the mean and standard deviation taken across spatial axes and γ_s and β_s are obtained by selecting the row corresponding to s in the γ and β matrices which are $N \times C$ in size (N is the number of styles modeled and C is the number of output feature maps). The third category, ASPM-MOB-NST, aims at achieving one model to transfer an arbitrary number of artistic styles. Huang et al. [53] propose a modification on IN named *Adaptive Instance Normalization* (AdaIN), which adjusts the mean and variance of the content input to match those of the style input:

$$\text{AdaIN}(\phi_j(x), \phi_j(s)) = \sigma(\phi_j(s)) \left(\frac{\phi_j(x) - \mu(\phi_j(x))}{\sigma(\phi_j(x))} \right) + \mu(s) \tag{4.7}$$

They use an encoder-decoder architecture, in which the encoder f is fixed to the first few layers of a pre-trained network. After encoding the content and style images using f, they feed both feature maps to an AdaIN layer that produces the target feature maps $t = \text{AdaIN}(f(c), f(s))$. The decoder then transforms the processed feature maps t back into image space, effectively transfering the style from one image to another.

Evaluation of style transfer algorithms remain an open and important problem [58]. Possible explanations for the evaluation problem include the lack of training and evaluation datasets (including benchmarks and automatic metrics), the unavailability of paired data (it is hard to get stylized versions of thousands of images) and the difficulty in separating the concepts of content and style. For now, qualitative evaluation relies on the aesthetic judgement of human observers, while quantitative evaluation is usually done based on inference time, model complexity, number of styles the model can support, etc.

4.3 Method

Even though Neural Style Transfer (NST) was originally conceived for stylizing image data, some studies have taken concepts from this task and applied them to data from other modalities. Some examples are the 2D-To-3D Style Transfer [61, 16], Audio Style Transfer [41, 123, 79, 119], and Text Style Transfer [52, 108, 33] tasks. So far, however, no one has explored style transfer in a multimodal scenario, such as transfering a style described in a text onto an image. In this approach, we aim to fuse style transfer concepts in a multimodal scenario with adversarial training to perform text-based image manipulation. The major contribution of this approach is that we developed a training procedure for text-based image manipulation using a combination of style transfer losses to alter images and adversarial losses to help keep them realistic-looking. This is the first method using style transfer concepts for this task.

4.3.1 Text Representation

The first challenge for text-based image manipulation is textual understanding. Since our modifications are described in text, we must parse it to understand what exactly must be changed. In our case, the text describes some of the characteristics of the object in the image (e.g., color, textures, and shape of specific parts), thus the semantic meaning of the images and textual descriptions are closely matched. However, it is hard to combine data from different modalities, since each modality has its own statistical properties. One interesting way of tackling this multimodal learning scenario using deep learning is to process unimodal signals separately but enforce certain similarity constraints on the results to bring them closer in a shared representation space [5]. This representation space is learned from data using neural architectures such as CNNs and RNNs, which map data into low-dimensional vectors called *embeddings*. Some examples of tasks that are solved using multimodal learning are image captioning [65, 124, 60] and multimodal retrieval [32, 65, 122, 130, 129].

As we have seen in Chapter 3, many methods use such multimodal learning approaches to parse the textual input that describes the modification [65, 96]. We will also take advantage of these well-tested solutions to represent text. Using such an approach, we can achieve textual representations in a latent space that matches the semantic meaning of images (i.e., the points are close in space). This is mainly done for convenience reasons; we could train a text encoder from scratch, but this would increase training times. This will also make comparing our solution with different approaches that use the same text embeddings. We will use the Char-CNN-RNN [96] text embedding network, denoted by φ, and the precomputed embeddings for the Oxford 102 Category Flowers dataset [88], which are available for download [5].

The architecture of the Char-CNN-RNN model is shown in Figure 4.2. The model is comprised of a temporal CNN and an RNN stacked on top of it, which allows it to get the benefits of both CNNs (speed and paralellism) and RNNs (strong long-term temporal dependencies) for text processing. The text-based CNN can be viewed as a standard CNN for images, except that the image width is 1 pixel and the number of channels is equal to the alphabet size. The maximum input length for sequences is constrained by the network architecture (201 characters), while variable-length sequences beneath this limit are handled by zero-padding the input past the final input character. Each convolution layer is followed by a ReLU activation and temporal max pooling. The CNN hidden activations are split along the time dimension and treated as an input sequence of vectors, which is processed by the RNN. The final encoded feature is the average hidden unit activation over the sequence.

Since the text embeddings produced by Char-CNN-RNN are trained to be similar to their image counterparts in an embedded space, we make the assumption that this space is indirectly modelling the style of an image by combining the individual stylistic characteristics contained in a text description. For instance, the sentence *“this flower has thin yellow petals and a black stamen”* contains the aggregation of three major styles: thin petals, predominant yellow color, and a black stamen. We refer to this style aggregation as **style code**. The style code s for a text description t is computed using the text embedding network φ in the following manner: $s = \varphi(t)$.

[5]https://github.com/reedscot/icml2016

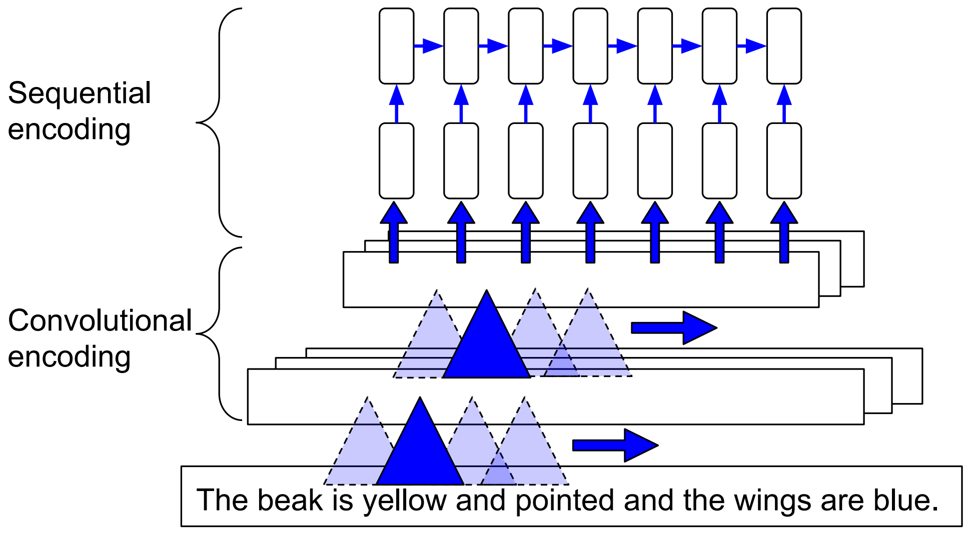


Figure 4.2: The architecture for the Char-CNN-RNN model used to extract text feature representations. Source: [96]

4.3.2 Architecture

Our task now is to design a generator network architecture that can combine style codes and images. Similarly to previous approaches [24, 74, 85], we will use an encoder-decoder architecture with a residual transformation unit in the middle. As we have seen in Section 3.8, this architecture has several benefits, such as maintaining spatial features for the source image, being computationally efficient, and making the task of learning the identity function easier. Our **generator network** G, depicted in Figure 4.3, consists of two components: an encoder network and a decoder network.

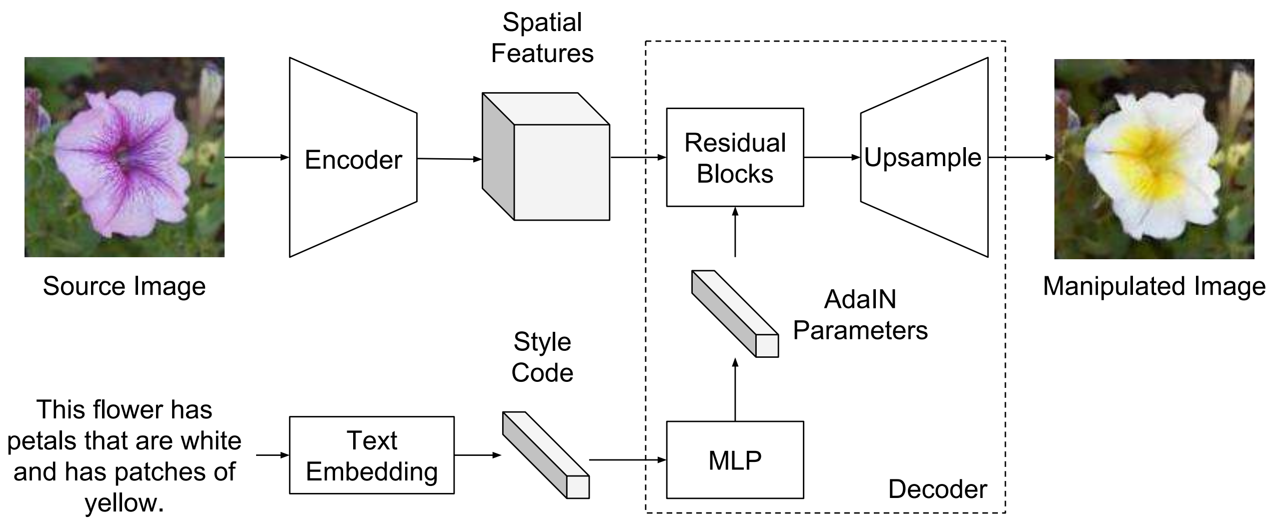


Figure 4.3: Our proposed generator architecture, which is comprised of an encoder that processes a source image and a decoder that uses the spatial features from the source image and a style code to synthesize new images.

The encoder exists for two purposes. The first purpose is to downsample a source image x into spatial features, which is done using several stride-2 convolutional layers. We downsample the image to get a spatial feature representation of the original image, which reduces computational cost for future convolution operations and allows us to retain the structure of the original image. The second purpose is to remove the style information from the original image. This is done using Instance Normalization (IN) [121] after each convolutional operation. Studies show that the convolutional feature statistics of DNNs can capture the style information of an image [34, 35], and that IN performs a form of style normalization by normalizing feature statistics, namely the mean and variance [53]. Thus, after processing the original image using the encoder, we have a "style-free" spatial feature representation of the original image.

Our decoder is responsible for synthesizing an image based on the spatial features of a source image and a style code. It processes the spatial features of the source image through a residual transformation unit followed by several upsampling and convolutional layers to return to the original image size. The residual transformation unit is composed of several residual blocks, which follow the guidelines proposed in [42], but using IN instead of BN. Inspired by recent style transfer work [27, 53, 126, 36], we use Adaptive Instance Normalization (AdaIN) [53] to dynamically change the parameters in the normalization layers of the residual transformation unit. AdaIN usually processes style images through a pre-trained network to extract statistics that are used to modulate the content features. In our case, however, we do not wish to use a style image; we use a style code that comes from textual features. Inspired by another recent study [54], we generate the modulation parameters using an MLP that processes style codes:

$$\text{AdaIN}(z, \gamma, \beta) = \gamma \left(\frac{z - \mu(z)}{\sigma(z)} \right) + \beta \tag{4.8}$$

where z are the features of the previous layer, μ and σ are the channel-wise mean and standard deviation, and γ and β are the parameters generated by the MLP for that layer. Since we wish to keep style information after the residual transformation unit, we use Batch Normalization after convolutional layers on the upsampling segment of the decoder.

Since we also use adversarial training to optimize the generator, we also define a **discriminator network** D, which consists of several stride-2 convolutions followed by Batch Normalization (the first and last layers have no normalization and there is no non-linearity on the last layer). This discriminator is not conditioned using text information; thus, it only provides unconditional loss to the generator, which helps in producing realistic images.

We also define an additional **loss network**, denoted as ϕ, which will serve as basis for our style transfer losses. The architecture for this network, depicted in Figure 4.4, is the VGG-16 [112] network minus the classifier segment, which corresponds to the last few layers. This network is pre-trained using the ImageNet [102] dataset. We will use this network to extract content and style representations. While it is possible to use other architectures to extract representations, such as Inception networks [114] or Residual Networks [46], most succesful implementation of NST use variants of the VGG architecture. There are several reports on the Internet of people attempting to use other architectures[6][7] and, while it is possible to make them work, it requires significant effort. Some possible explanations for the superiority of VGG architectures are [83]:

- The VGG architecture is significantly larger in number of parameters than other recent architectures (they are usually deeper and more "optimized"). More parameters may not help in classification (verified by its inferior performance), but may allow the model to capture information that is useful for style transfer that other architectures discard.
- Other models downsample more aggresively than VGG, losing valuable spatial information.
- Most modern vision models have checkerboard artifacts in their gradient computations due to their architectural choices.
- VGG is shallow when compared to more modern architectures. On deeper architectures the features spread wildly between distant layers. This effect is increased when residual connections are involved.

For these reasons, we will stick to the VGG architecture.

4.3.3 Training Procedure

Let A_I and B_I represent two different images with corresponding text descriptions A_T and B_T. Our goal is to modify image A_I with the style code extracted from B_T using φ, generating a new image S. This is done using our generator network, i.e., $G(A_I, \varphi(B_T)) = S$. From now on, we will omit φ from equations for brevity. The generator is optimized for this purpose using several losses (Figure 4.5), which we now describe.

The first two losses that we employ are based on the style transfer loss proposed in [35]. For the content reconstruction loss, we employ the Euclidean distance loss (Equation 4.3) between feature representations of original images and images synthesized by G, i.e., $\mathcal{L}_c(G(A_I, B_T), A_I)$. This will force the generator to ouput an image that has a structural appearance similar to the original image. For the style reconstruction loss, we use the Frobenius norm loss (Equation 4.4) of the difference between style representations of ground-truth

[6]https://medium.com/mlreview/getting-inception-architectures-to-work-with-style-transfer-767d53475bf8

[7]https://www.reddit.com/r/MachineLearning/comments/7rrrk3/d_eat_your_vggtables_or_why_does_neural_style/

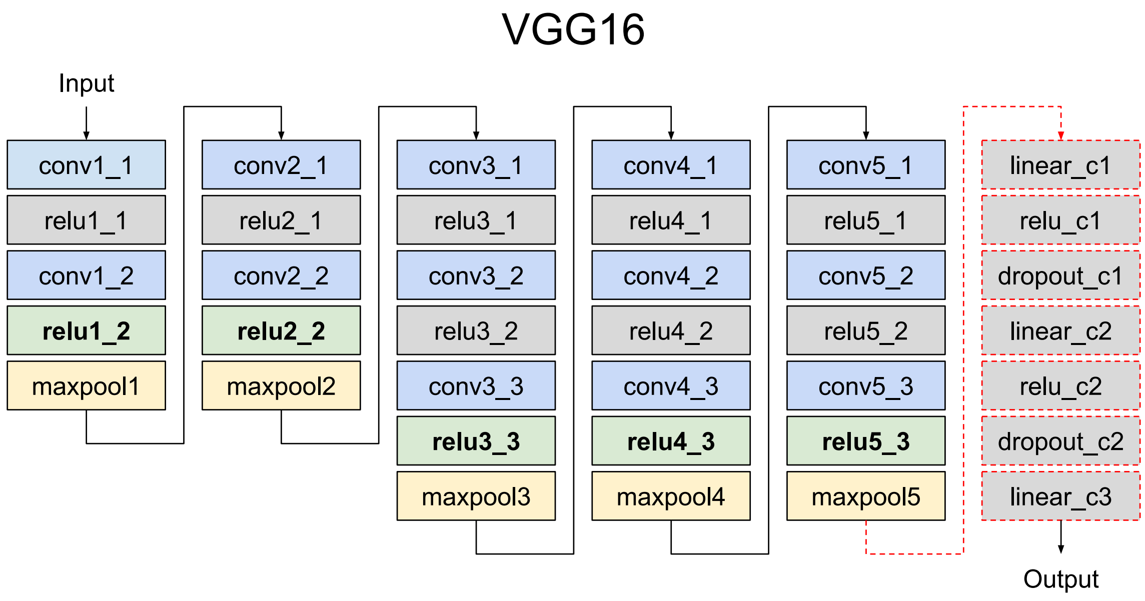
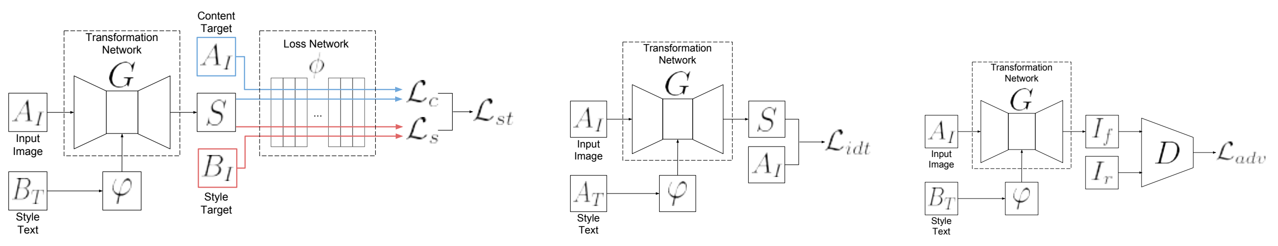


Figure 4.4: The architecture of the loss network used to extract content and style representations of images. Green boxes indicate layers tipically used to extract representations and red dashed boxes indicate the classifier segment of the network, which is removed.

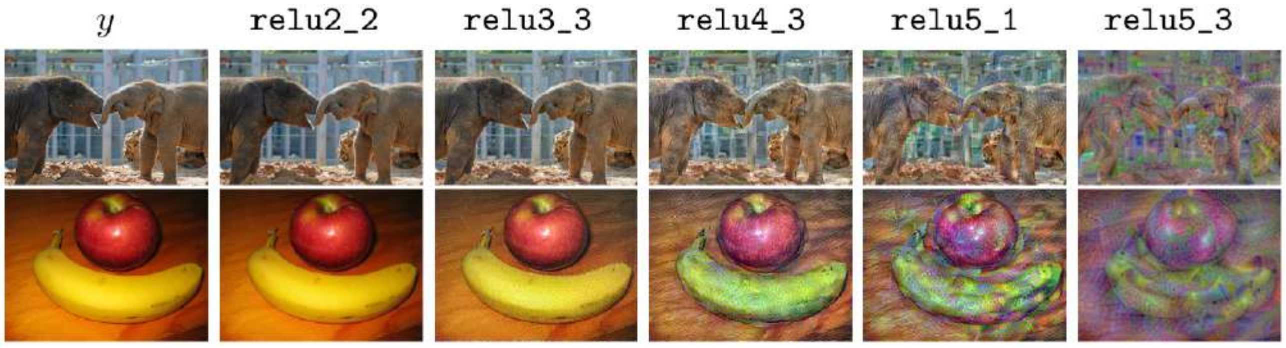


(a) Style transfer loss. (b) Reconstruction loss. (c) Adversarial loss.

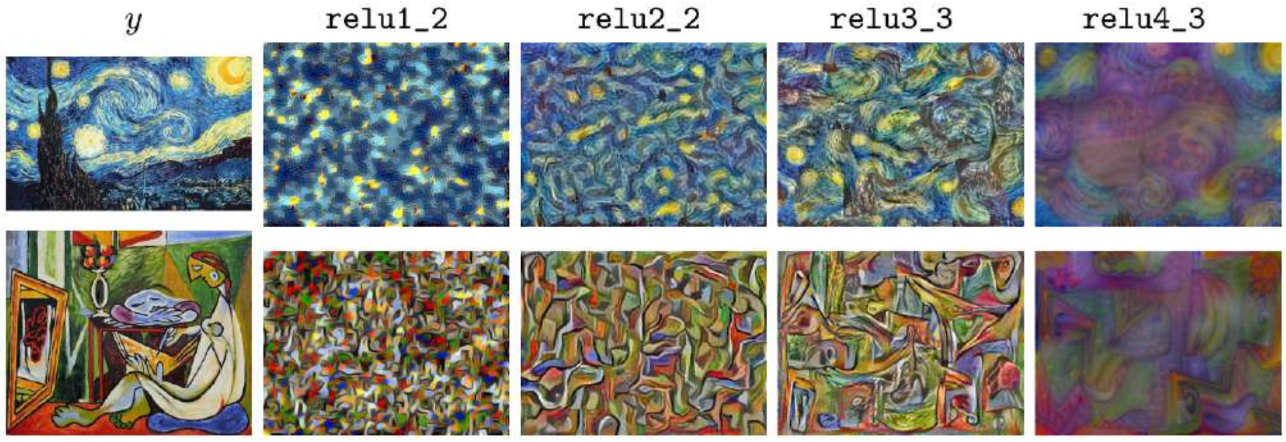
Figure 4.5: Illustration of the loss functions used for optimizing our generator network.

images B_I for text representations B_T and images synthesized by G, i.e., $\mathcal{L}_s(G(A_I, B_T), B_I)$. The style representations of images are calculated using Gram matrices (Equation 4.1). The style loss will force the generator to use the original image and the style code to output an image that has a style appearance similar to the style image. It is important to note that, even though we use both text representation B_T as well as its image counterpart B_I for training, during test time we only need B_T to obtain style information.

Several studies [34, 35, 59] show that content and style features follow a hierarchy, going from low-level (early layers) to high-level (final layers) abstractions (Figure 4.6). For content, using early layers tend to produce images that are visually indistinguishable from the source image. As we move to higher layers, the image content and overall spatial structure are preserved, but color, texture, and exact shape are not [59]. As for style, reconstructing from lower layers transfers small-scale and simple structures and higher layers transfer larger-scale and more complex structures from the target image. There is no formula for picking which layers to use, and most of the time layers are picked by doing empirical reconstruction experiments to see which combination leads to better results. We describe which layers we use to compute $\mathcal{L}_c$ and $\mathcal{L}_s$ an its implications in the experimental section on Chapter 5. The style transfer losses we use are illustrated in Figure 4.5a.



(a) Content reconstruction loss.



(b) Style reconstruction loss.

Figure 4.6: Effect of choosing different layers of a VGG-16 model to optimize and find images that minimize content/style reconstruction losses. Source: [59]

The process of manipulating an image may generate a new background and other contents that are not described in text. If we observe the results of traditional style transfer methods, the structure of the original image is maintained, but the new style is applied to the image on a global level. This behavior is not desirable, since we wish to modify only the object. Thus, to enforce G to modify the object while maintaining the text-irrelevant contents of the original image, we use an L_1 reconstruction loss (depicted in Figure 4.5b) between an image A_I and a modification synthesized by G using the original image's text, A_T:

$$\mathcal{L}_{idt}(G(A_I, A_T), A_I) = \|G(A_I, A_T) - A_I\|_1 \tag{4.9}$$

Traditional style transfer methods usually focus on non-photorealistic results, which may significantly distort original images. Since we wish modified images to look realistic, we additionally use adversarial training between G and D to enforce G to produce realistic-looking samples (Figure 4.5c). This is done using the LSGAN [76] adversarial loss:

$$\mathcal{L}_{adv}(D) = \frac{1}{2}\mathbb{E}_{A_I \sim p_{\text{data}}}[(D(A_I) - 1)^2] + \frac{1}{2}\mathbb{E}_{A_I, B_T \sim p_{\text{data}}}[(D(G(A_I, B_T))^2] \quad (4.10)$$

$$\mathcal{L}_{adv}(G) = \frac{1}{2}\mathbb{E}_{A_I, B_T \sim p_{\text{data}}}[(D(G(A_I, B_T) - 1)^2] \quad (4.11)$$

where $\mathcal{L}_{adv}(G)$ and $\mathcal{L}_{adv}(D)$ are the losses optimized by G and D, respectively. We choose the LSGAN loss because it tends to be more stable than the traditional adversarial loss since it alleviates the problem of vanishing gradients for fake samples that lie far from real data on the manifold [76].

We then combine the style transfer, reconstruction, and adversarial losses to get the total loss optimized by G:

$$\begin{aligned}\mathcal{L}_G = {} & \lambda_c \mathcal{L}_c(G(A_I, B_T), A_I) \\ & + \lambda_s \mathcal{L}_s(G(A_I, B_T), B_I) \\ & + \lambda_{idt} \mathcal{L}_{idt}(G(A_I, A_T), A_I) \\ & + \lambda_{adv} \mathcal{L}_{adv}(G)(A_I, B_T)\end{aligned} \quad (4.12)$$

5. EXPERIMENTAL ANALYSIS

In this chapter, we will describe the experimental analysis we conducted to assess the performance of our method. We conducted experiments on the Oxford 102 Category Flowers dataset [88], which is comprised of 8,189 images of flowers divided into 102 categories, each consisting of around 40 to 250 images of a specific class (species). Similarly to [24], we split the flowers dataset into 82 training classes and 20 testing classes for evaluation purposes. Each image is accompanied by 10 textual descriptions that were collected by [96], which mainly describe the colors, textures, and some overall appearance information for different parts of the object. Using multiple datasets for evaluation is a good way to test for generalizability of the approach, but it considerably slows down development speed due to the added variability and hyperparameter tuning that follows for each dataset. Thus, we decided to focus our experimental analysis on only the flowers dataset because this would ease the development of our method.

For data augmentation, we used random cropping, horizontal flipping, and rotation (in that order). For a final output size of D^2 (same width and height), we first resized all images to $(D + \lceil D \cdot 0.1 \rceil) \times (D + \lceil D \cdot 0.1 \rceil)$. For instance, when generating images with size 128^2, we resize the original image to 141^2. This guarantees that the random crop captures the object. The flip chance was set to 50% and rotations ranged from -10 to 10 degrees.

We implemented our method using PyTorch [92]. All of our models use the same architecture for residual blocks, G, and D; the detailed architectures are shown in Appendix A. We train the networks for 60 epochs (same as our comparison baselines) with a batch size of 64, image size of 128, text embedding size of 1024, and using all 10 captions for all images. We use the Adam [63] optimizer to train both G and D simultaneously, with an initial learning rate of 0.0002, $\beta_1 = 0.5$, $\beta_2 = 0.999$, and decay the learning rate dividing it by 2 every 10 epochs. We initialize the weights of convolutional layers with a normal distribution of mean 0 and standard deviation 0.02 and batch normalization layers with a normal distribution of mean 1 and standard deviation of 0.02 (bias are initialized as 0). For the text encoder φ, we use a pre-trained Char-CNN-RNN [96] model and initialize its weights to the pre-trained model provided in [97]. For the loss network ϕ, we use a VGG-16 architecture and the pre-trained weights available in PyTorch.

The remainder of this chapter is as follows. First, in Section 5.1, we define a default configuration for the loss multipliers λ_c, λ_s, λ_{idt}, and λ_{adv}, and the layers we use to compute $\mathcal{L}_c$ and $\mathcal{L}_s$ from. Then, in Section 5.2, we perform ablation studies do determine the impact of the style losses on our results. Next, in Section 5.3, we perform both quantitative and qualitative analysis comparing our method with two baseline studies.

5.1 Default Configuration

In order to evaluate the performance of our model, we first selected a default configuration of hyperparameters to experiment upon. We first selected which layers to use to compute the content and style reconstruction losses. We compute the content reconstruction loss using the `relu2_2` layer, and the style reconstruction loss is computed using the layers [`relu1_2`, `relu2_2`, `relu3_3`, `relu4_3`, `relu5_3`], with equal weight between them (0.2 for each one, summing up to 1). Different combinations of content and style layers can produce very different results and, since it is hard to quantify style transfer results, there is no "correct" combination. The basis for picking these layers is an empirical test on the style transfer task, where they provide good results and work consistently across many images.

The next step is to pick values for the remaining hyperparameters, which are used to weight the different losses used to optimize the model. We searched for a good set of the remaining hyperparameters via trial-and-error, which are: $\lambda_c = 1$, $\lambda_s = 10$, $\lambda_{idt} = 1$, and $\lambda_{adv} = 1$. Figure 5.1 shows a few samples of the results for this configuration. This will be the hyperparameter configuration we will use when performing ablation studies and comparing to other methods.

5.2 Ablation Studies

In the context of deep learning, an *ablation study* refers to the process of removing some "feature" or component of the model and checking how that affects performance. This gives the researcher the ability to gain a better understanding of the model's behavior, detect the role of each component, and, most importantly, determine which aspects of the model are crucial (or superfluous) for its functioning. We conduct ablation studies to check the impact of the style transfer losses used to optimize our model. These ablation experiments demonstrate that the combination of both content and style losses is needed to perform text-based image manipulation using our approach.

5.2.1 Removing the Content Loss

First, we check the effect of the content reconstruction loss $\mathcal{L}_c$ by removing it from the optimization. The role of the content loss in the optimization process is to force the generator to synthesize an image that has a similar structural appearance to the source image. Our hypothesis is that, without the content loss, the generator will not have enough incentive to reconstruct the original image; this would make our manipulation results considerably

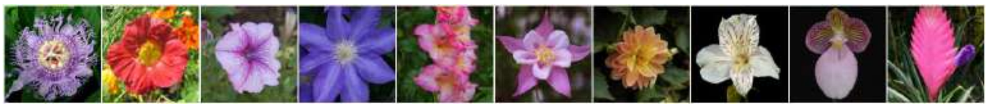
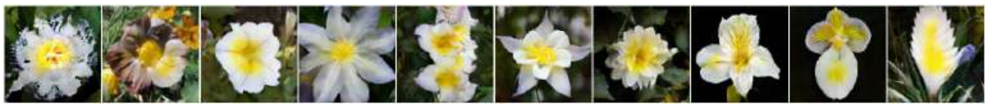
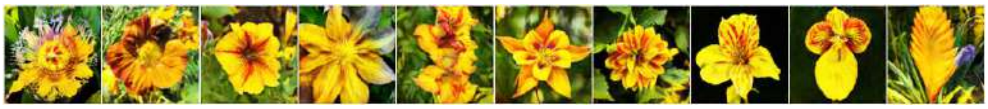
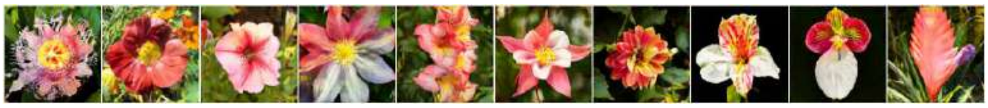
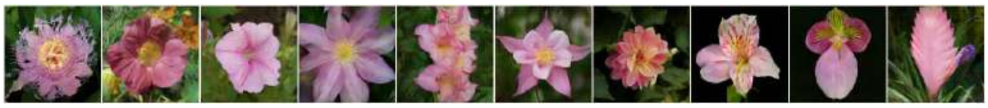
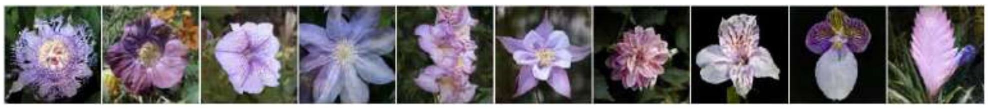
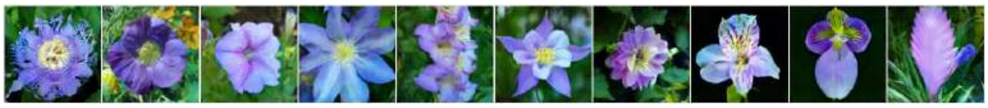
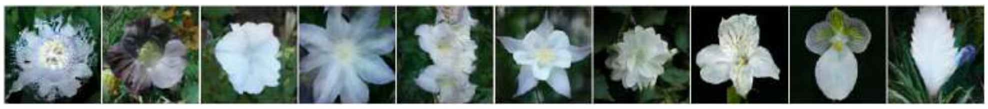
Caption	Results
Original	
“This flower has petals that are white and has patches of yellow.”	
“The petals of the flower have yellow and red stripes.”	
“This flower has petals that are red and has yellow tips.”	
“A light pink flower with pointed petals and a yellow circle.”	
“The light purple flower has a large number of small petals.”	
“This flower has petals that are blue and has yellow tips.”	
“This flower is completely white.”	

Figure 5.1: Manipulation results for our approach on the Flowers dataset. The first row contains the original instances and each subsequent row contains the results for a specific textual description.

worse, since they may not have any relation to the original image. Additionally, the style reconstruction loss would predominate during the optimization process, which would make generated images look like texturized versions of the style described in text.

We trained the model using the default configuration (Section 5.1) but changing $\mathcal{L}_c$ to 0. We now analyze the effect of a specific instance depicted in Figure 5.2. We can clearly see that the content of the original image was completely discarded by the generator. Interestingly, if we compare the modification text and its corresponding image with the synthesized result, we can see that the generator has learned to create images that broadly resemble the style described in text form. In this case, the resulting flower has yellow and red stripes and a stamen that is characteristic for that type of flower. This experiment allows us to observe that the style loss is, indeed, helping the network learn what each combination of characteristics (features) looks like. We present more results for this ablation on Figure 5.3.

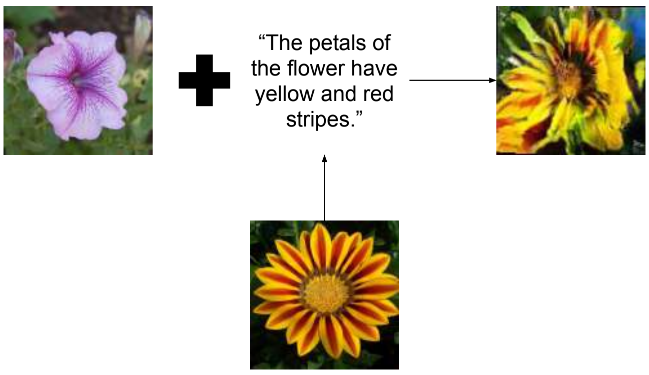


Figure 5.2: Inspecting the effect of removing the content loss on specific instances of the flowers dataset. The figure on the left is the content image, the figure and text on the middle is the style image, and the figure on the right is the result of the generator.

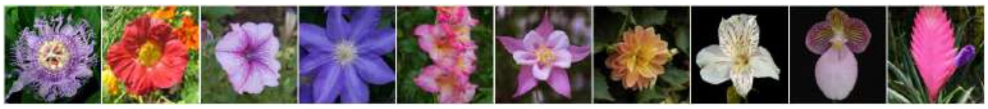
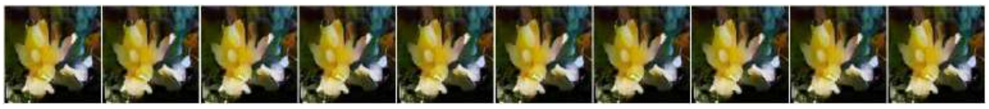
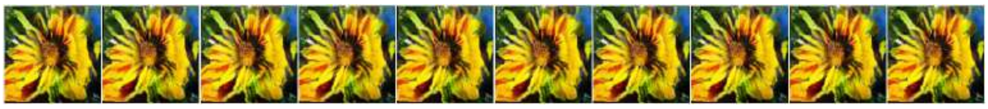
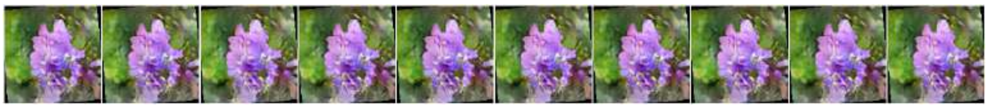
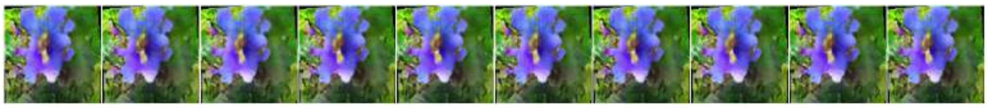
Caption	Results
Original	
"This flower has petals that are white and has patches of yellow."	
"The petals of the flower have yellow and red stripes."	
"The light purple flower has a large number of small petals."	
"This flower has petals that are blue and has yellow tips."	

Figure 5.3: Ablation experiment: removing the content loss makes our generator learn "texturized" versions of the style.

5.2.2 Removing the Style Loss

Next, we check the effect of the style reconstruction loss $\mathcal{L}_s$ by removing it from the optimization. The role of the style loss in the optimization process is to force the generator to synthesize an image that has a similar stylistic characteristics to the style described in the style code. Our hypothesis is that, without the style loss, the generator will completely disregard the text description and focus on learning the identity function, which would improve its performance on all other loss functions.

We trained the model using the default configuration (Section 5.1) but changing $\mathcal{L}_s$ to 0. We now analyze the effect of a specific instance depicted in Figure 5.4. Observing the result in the image, one can see that the original image was completely copied by the generator. This behavior occurs for all instances, no matter the input image and text (for more results, see Figure 5.5). This experiment also shows us that the content loss is doing its job, since the features of the original image were preserved.

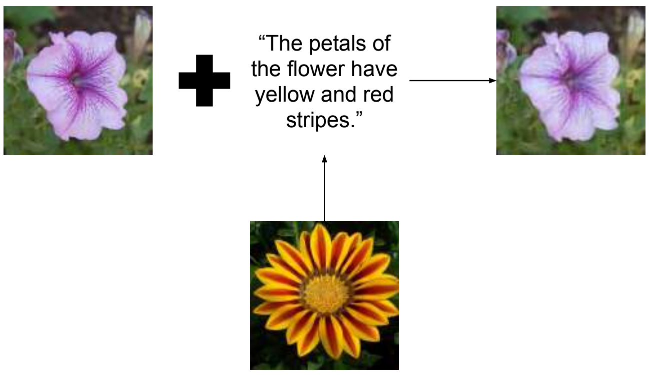
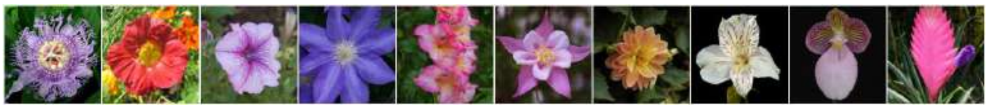
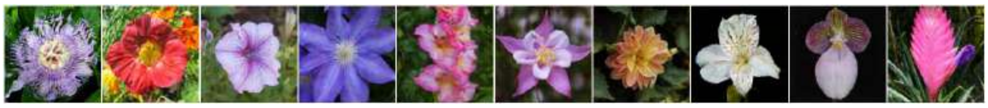
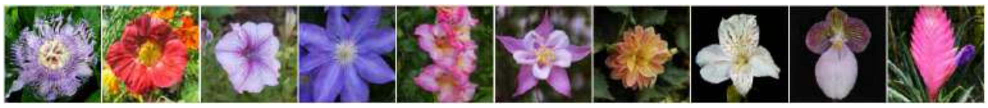


Figure 5.4: Inspecting the effect of removing the style loss on specific instances of the flowers dataset. The figure on the left is the content image, the figure and text on the middle is the style image, and the figure on the right is the result of the generator.

Figure 5.5: Ablation experiment: removing the style loss makes our generator learn the identity function.

Caption	Results
Original	
"This flower has petals that are white and has patches of yellow."	
"The petals of the flower have yellow and red stripes."	

5.3 Evaluation

Evaluating the performance of generative models is a hard problem. While several measures have been introduced recently, there is no consensus as to which measure best captures the strengths and limitations of models and should be used for fair model comparison [14]. Regarding the task of text-based image manipulation, evaluation becomes an even harder problem since we do not have ground-truth data. As we have seen in Section 3.8, most studies use a combination of quantitative evaluation via human judgement (usually collected on Amazon Mechanical Turk[1]) [24, 74, 85], and a qualitative assessment by manually inspecting the results [97, 24, 91, 74, 85] to bypass these evaluation problems. The main criteria used to evaluate the results on this task are:

- **Image Quality**: related to realism and similarity to dataset instances
- **Identity Robustness**: retainment of aspects not present in text
- **Manipulation Quality**: synthesized images match the text descriptions provided

We compare the performance of our method using two baselines: SISGAN [24] and TAGAN [85]. We picked SISGAN because it was the first one-stage method for the task, and TAGAN because it is currently the approach with the best results in literature. The other methods either do not open source their code [97, 74] (which makes reproducing their work harder) or have inferior results when compared to TAGAN [91]. It is important to note that our method, SISGAN, and TAGAN use different approaches for computing text embeddings. One could argue that this could influence the results when comparing approaches and, for an even fairer comparison, we should implement all methods using the same text encoder. However, in the case of TAGAN, the difference in text embedding computation is one of the core changes of the method. For this reason, we use each method as-is regarding the text embedding approach. In the following sections, we compare the results of our method regarding the three criteria described above.

5.3.1 Image Quality

The first criterion we will analyze is image quality, or realism. There are several metrics that can be used to check the overall quality of synthesized images in GANs. Two of the most commonly used quantitative evaluation measures are the Inception Score (IS) [105] and the Fréchet Inception Distance (FID) [48]. Since FID is a proposed improvement of the IS, we will employ the **FID Score** as one of our quantitative evaluation measures for image quality assessment. FID embeds a set of samples into a feature space using a specific layer of an Inception model [115] (or any CNN), which contains vision-relevant features. We assume that the embedding layer follows a multidimensional Gaussian and calculate the first two moments, mean μ and covariance Σ, for generated and real samples. The Fréchet distance between these two Gaussians is used to quantify the quality of generated samples:

$$\text{FID}(r, g) = \|\mu_r - \mu_g\|_2^2 + Tr\Big(\Sigma_r + \Sigma_g - 2(\Sigma_r\Sigma_g)^{\frac{1}{2}}\Big), \tag{5.1}$$

[1]https://www.mturk.com/

where (μ_r, Σ_r) and (μ_g, Σ_g) are the mean and covariance of the real and generated distributions, respectively, and Tr is the trace operation. These statistics are extracted from the 2048-dimensional activations of the last pooling layer (`pool_3`) of a pre-trained Inception-v3 model [116]. Since we are measuring the distance between the two distributions, lower scores signify better results (0 is the lowest possible score). Heusel et al. [48] show that the FID score is consistent with human judgements and is more robust to disturbances than IS.

It is important to highlight that, with this metric, we are evaluating the quality of images manipulated using **mismatched texts**, i.e., $G(A_I, B_T)$. We generate 10,000 samples for all three methods and compare them to the precalculated dataset statistics on 64×64 and 128×128. For a fair comparison with SISGAN (which generates 64^2 images), we rescale the images generated by our method and TAGAN to the same resolution as SISGAN, while also comparing both methods at the original resolution.

We report the FID Score results in Table 5.1. The first thing that we can observe is that the FID Score for SISGAN is, by far, the highest. This occurs because SISGAN frequently produces images that are blurred, distorted or are nonsensical (do not look like flowers at all). This matches the expected FID Score result. TAGAN has the second best results and is better than SISGAN by over an order of magnitude. This signifies an enormous leap in image quality between the two methods. Our method has the best FID Score results. This could be explained by the fact that our approach can keep more fine-grained details when compared to TAGAN, which could explain the small difference between approaches.

5.3.2 Identity Robustness

The second criterion we analyze is identity robustness, or how well our method can maintain the irrelevant aspects of the image (e.g., background and characteristics that are not mentioned in text). This test is important because it allows us to test if the network understands the relationship between object parts and modifications described in text and object parts and background information present in the image. This knowledge is crucial when manipulating images because one would expect that the irrelevant aspects of the image remain similar to the original image. Nam et al. [85] suggested the use of a **reconstruction score** to evaluate the quality of content preservation. This can be done by computing an L_2 reconstruction error by forwarding an image with its **matching text**:

$$\text{RS} = \|G(A_I, A_T) - A_I\|_2 \tag{5.2}$$

The score results are normalized between $[0, 1]$, where low values indicate that the method can preserve the contents of the original image that should not be modified. We will use this reconstruction score as another quantitative evaluation measure. To compute it,

Table 5.1: Quantitative evaluation of our method using the FID Score and the Reconstruction Score. For both measures, lower is better.

Method	Rec. Score	FID Score	
		64×64	128×128
SISGAN	0.1709	383.1528	-
TAGAN	0.0483	35.4931	18.6384
Ours	**0.0339**	**31.8682**	**14.1197**

we forward all pairs of images and text in the dataset through the network and compute the mean reconstruction error over all samples on the training set; Table 5.1 shows the results.

Inspecting the table, we can right away identify that the reconstruction score for SISGAN is the highest. According to the results, SISGAN modifies approximately 17% of the pixel values in images when nothing should be changed. This happens because, as explained previously in Section 3.8, the method is likely to generate a new image conditioned on the pose of the original image instead of modifying only the parts that are described in the text. In some cases, the method generates a similar image merely because the sentence is highly correlated to a particular class of object. Additionally, there is frequent background modification, regardless of the current image. Similarly to the FID Score, TAGAN presents results that are one order of magnitude better than SISGAN. This means that the method is much more likely to understand what is the relationship between the text description and the current image. Our method has the best reconstruction score, which could be explained by the fact that the content loss helps reconstruct the original image to the finest of details. This means that TAGAN and our approach can understand that an image already has a certain style, and that nothing needs to be updated.

5.3.3 Manipulation Quality

The third and last criterion we wish to analyze is manipulation quality. We will focus on using qualitative analysis and visually compare our results with the baseline approaches. A comparison with several samples is depicted in Figure 5.10 and Figure 5.11 (note that SISGAN's results are scaled to 128^2). We tried to select source images that contained a wide variety of different shapes, colors, and textures. We now present specific comparison scenarios to highlight the main advantages and disadvantages between methods.

The first scenario is depicted in Figre 5.6. In this comparison, we want to focus on the capability of the methods to reconstruct fine-grained content details from the original image. This flower has lots of small and curly petals, which makes it harder to faithfully reconstruct it. Analyzing the details, we can see that the approach we developed it the best at reconstructing fine-grained details. This can be explained by the fact that the content loss is very good at forcing the network to keep such details in its internal representation. TAGAN has the second best result: although the overall structure of the flower is intact, the finer details (such as the small, curly petals) are lost during manipulation. SISGAN has the worst result, since it fails to keep even coarse structural information from the source image and produces a blurred and distorted result.

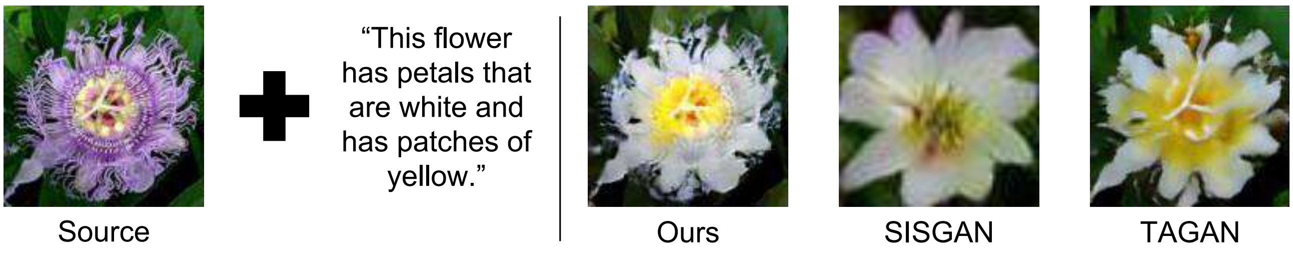

Source
“This flower has petals that are white and has patches of yellow.”
Ours
SISGAN
TAGAN

Figure 5.6: First comparison scenario between our approach, SISGAN, and TAGAN.

On the second scenario, illustrated in Figure 5.7, we compare methods regarding the ability to perform small modifications that should not require major structural perturbations. Here, we can see that our method does produce red stripes on the output, but it only does so in certain parts of the image. We postulate that this happens because the content loss forces a faithful reconstruction of the original image even on a contrast level. Therefore, it focuses on applying the modifications where they do not influence on the reconstruction of contrast later on. Although TAGAN can produce better stripes when compared to our method, it also heavily modifies the central part of the flower. This happens because red and yellow stripes are linked to a specific species of flower, and TAGAN is also slightly biased towards applying unecessary structural changes to conform to a specific class. SISGAN has the same behavior as TAGAN, but the modifications are even less subtle: the flower is deformed and barely resembles the original image.

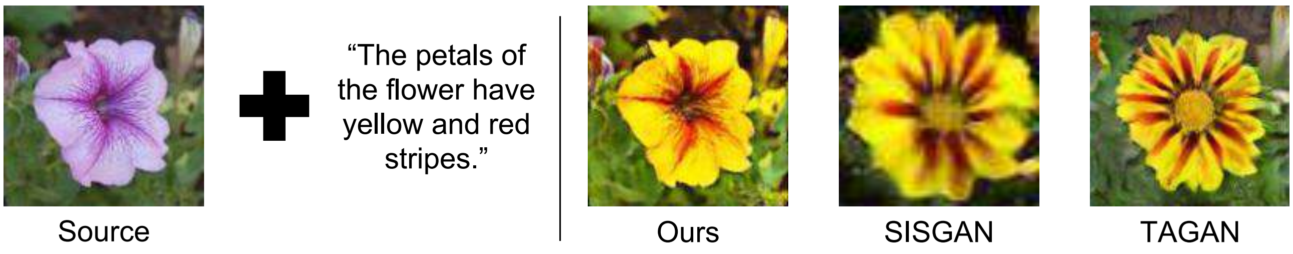


Figure 5.7: Second comparison scenario between our approach, SISGAN, and TAGAN.

Since both TAGAN and SISGAN frequently modify the source image to resemble the class of flower described in text form, we analyze a third scenario, depicted in Figure 5.8, where the methods receive a text that describes a flower that does not exist on the dataset. This text was invented by us to test how each approach behaves when facing text descriptions it has never seen before. In this scenario, we can see that our approach and TAGAN can somewhat deal with this description: the flower changes its main color to blue, but the tips are not painted yellow; instead both methods change the central part of the flower to yellow. On our method, this behavior can be explained using the same argument we used on Figure 5.7, we attempt to modify images only when it does not influence the overall contrast reconstruction. For TAGAN, this happens possibly because it also has never seen flowers that look like the description, so it makes a modification to include the yellow color that does not match exactly the description location. SISGAN fails to generate a reasonable result for this description, which explained by the fact that it frequently changes the source image to match the overall class of flowers described by the caption. However, since it has never seen any flower with that aggregation of characteristics, it does not know what to do.

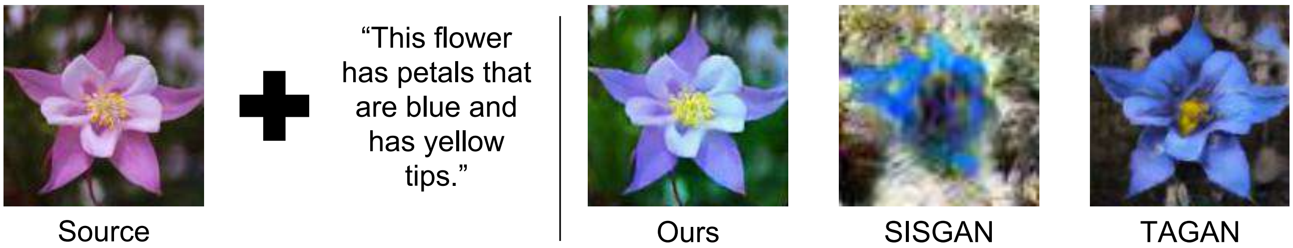


Figure 5.8: Third comparison scenario between our approach, SISGAN, and TAGAN.

On the last case, illustrated in Figure 5.9, we analyze a scenario where structural modifications are needed to conform to the desired textual specification. The flower in the source image has large petals, and the text requests them to be changed to become small (and plenty). Both our approach and TAGAN focus more on changing textures and colors, so they are not capable of drastically changing the structure of the original image. SISGAN, on the other hand, has no such problem: it changes the overall structure of the source image to become more like the class of flower described in text. Even though the flower does not resemble the original one, this may be just what the user desires.

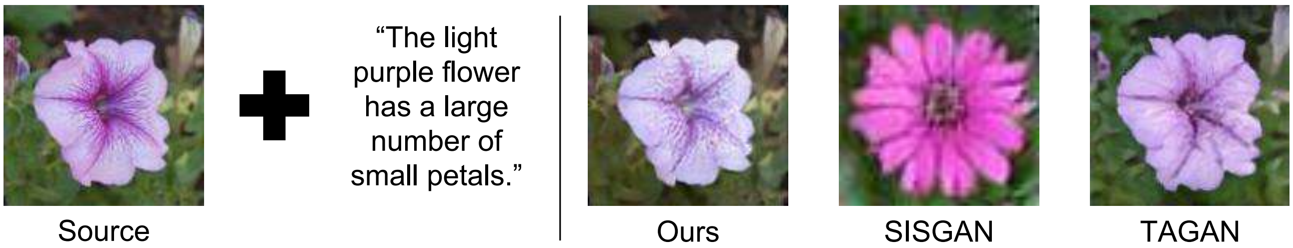

"The light purple flower has a large number of small petals."
Source
Ours
SISGAN
TAGAN

Figure 5.9: Fourth comparison scenario between our approach, SISGAN, and TAGAN.

We now make a broad analysis by inspecting the results in Figure 5.10 and Figure 5.11. We can see that our approach maintains the structure and fine-grained details (e.g., textures and contrast) of the original image the most, followed by TAGAN and SISGAN. This matches our findings when analyzing the quantitative results for the FID Score and Reconstruction Score. TAGAN usually removes (or disregards) fine-grained details of the original image (especially textures and contrast) and synthesizes another image using the “carcass” of the source image. SISGAN goes to the extreme and sometimes appears to be creating a new image using the spatial features of the original image as conditioning.

Being so rigid with keeping the original content can be undesirable sometimes: our method cannot get rid of fine-grained details (such as spots and patterns) when generating a new image. Additionally, our method follows the contrast of the original image too much. This leads to our method failing to modify images on specific places. These problems can be linked to the content loss: we are using an early layer (`relu2_2`) for the content reconstruction loss, and this layer perhaps still contains too much detailed information about the image. Thus, finding another set of content layers may improve our method.

Overall, we can see that the method we developed using style transfer concepts is successful in a lot of cases, but fails to manipulate images when the request involves small details on precise locations. TAGAN is better at performing small modifications, but it is not as good as our approach at maintaining fine-grained details. SISGAN is the best method for changing structure, but has several drawbacks, such as blurred and distorted images and the inability to change images when the text description is novel. We can see a clear trade-off between maintaining structure and applying modifications on all methods. Our method is more rigid at keeping the structure of the overall image, TAGAN is a middle ground that can perform more detailed manipulations by compromising some structural integrity, and SISGAN radically changes images even when there is no need. We conclude our qualitative analysis by stating that our results are competitive with the state-of-the-art, and that style transfer concepts can lead us to find even better models for the task of text-based image manipulation.

Figure 5.10: Qualitative comparison of the results of our method, SISGAN, and TAGAN. Set 1 of 2.

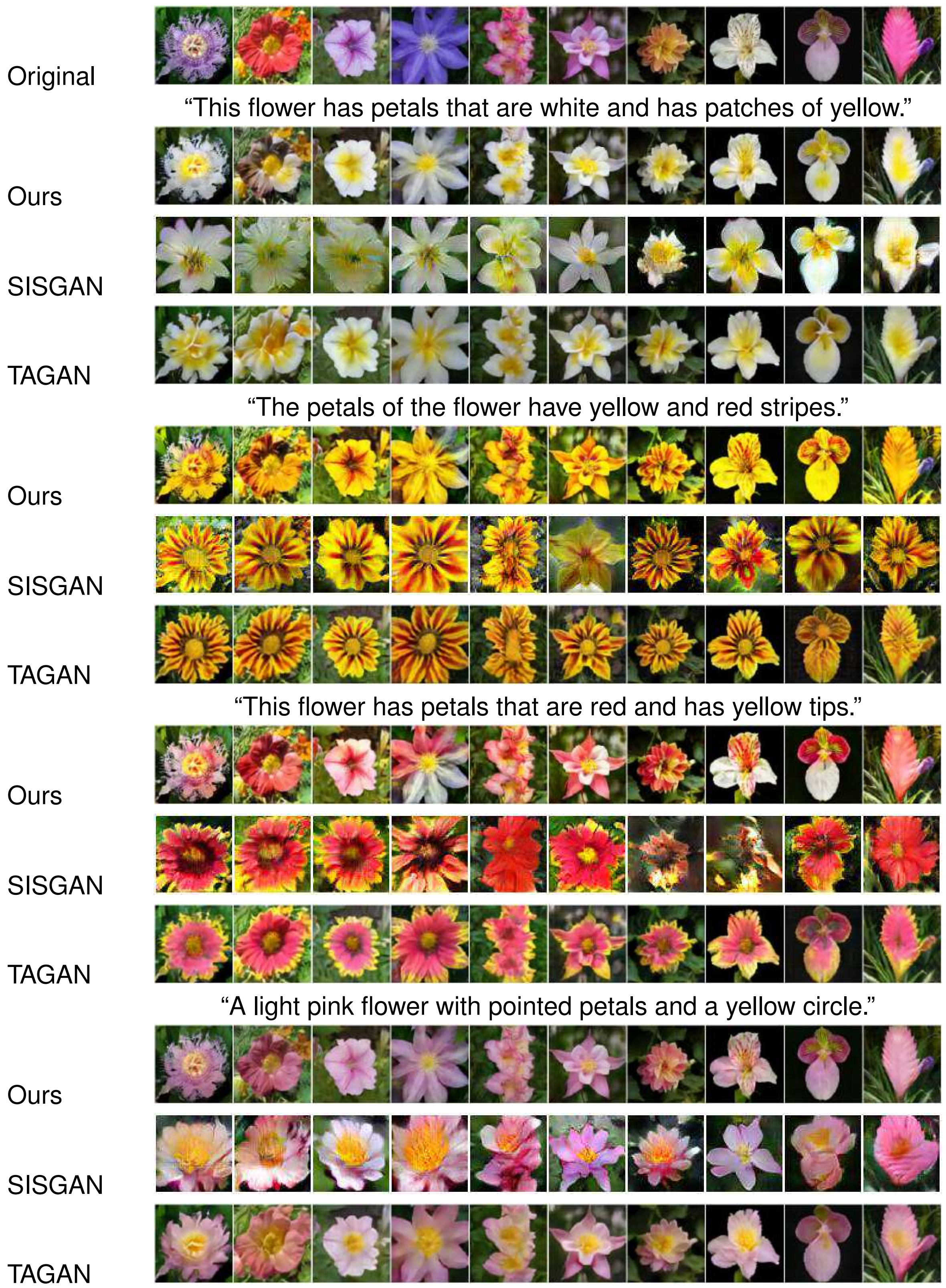
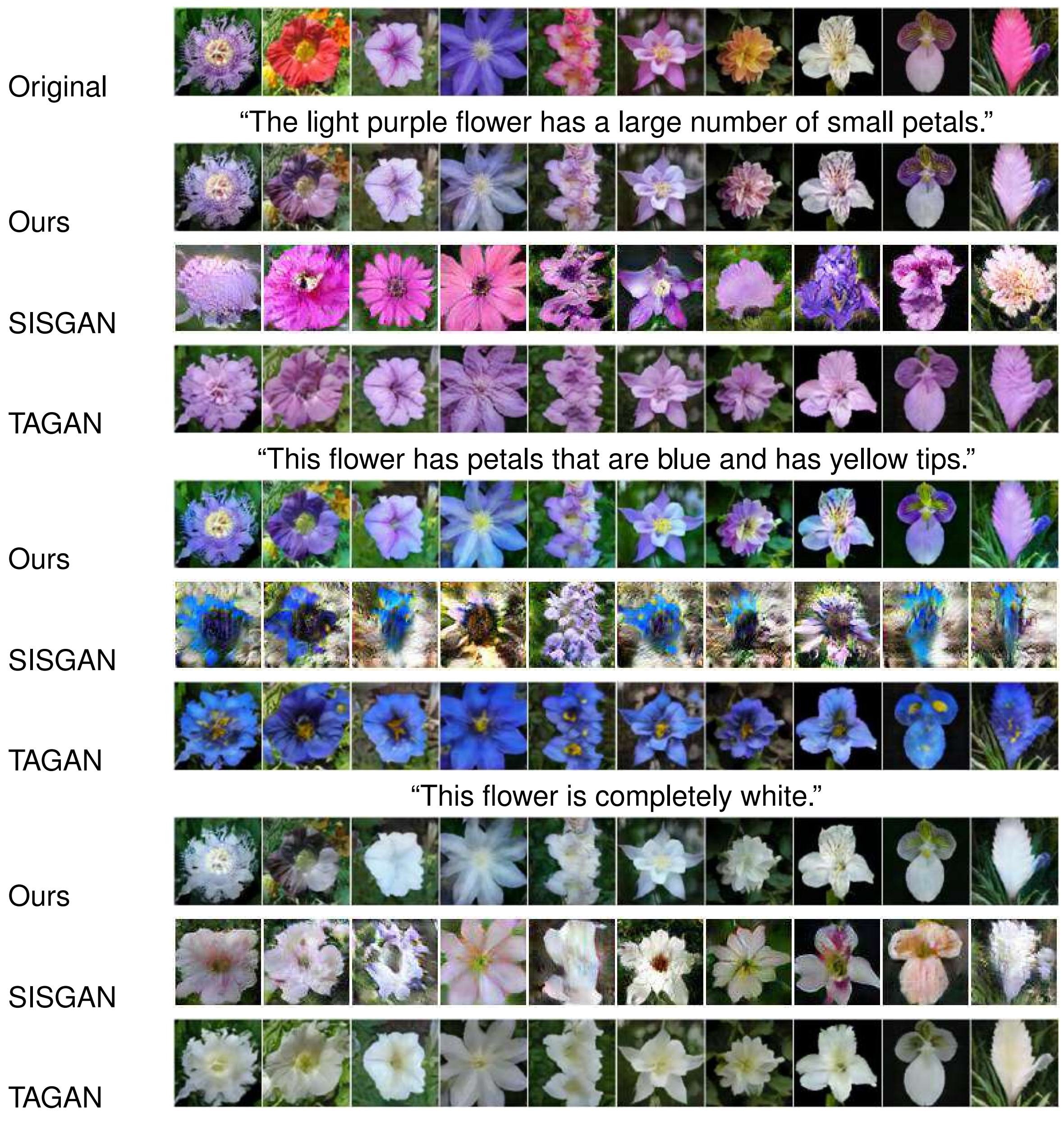

Original
"This flower has petals that are white and has patches of yellow."
Ours
SISGAN
TAGAN
"The petals of the flower have yellow and red stripes."
Ours
SISGAN
TAGAN
"This flower has petals that are red and has yellow tips."
Ours
SISGAN
TAGAN
"A light pink flower with pointed petals and a yellow circle."
Ours
SISGAN
TAGAN

Figure 5.11: Qualitative comparison of the results of our method, SISGAN, and TAGAN. Set 2 of 2.


Original
"The light purple flower has a large number of small petals."
Ours
SISGAN
TAGAN
"This flower has petals that are blue and has yellow tips."
Ours
SISGAN
TAGAN
"This flower is completely white."
Ours
SISGAN
TAGAN

6. FINAL CONSIDERATIONS

A traditional image manipulation software provides a wide variety of tools that allows proficient humans to manipulate images an generate novel artistic creations. Automatic image manipulation is a reserach-worthy topic of study because it empowers the average person by giving the ability to apply complex modifications without the expertise normally needed. In this study, we explored automatic image manipulation using language as input signal for modification, which we referred to as text-based image manipulation. Language offers a general and flexible interface for describing objects in any space of visual categories and is not tied to any specific task; therefore, it has enormous potential to automate several artistic processes.

We proposed a novel deep learning model that combines adversarial training and style transfer concepts. We treat points in sentence embedding space as containing aggregate style information (which we refer to as style code), which is characterized by the combination of individual styles for each characteristic of the object. We use the text embeddings to modulate the features of an existing image on a generator network. This network is then optimized using content and style representations extracted from a loss network and two additional losses to enforce realism and keeping irrelevant aspects intact.

We presented the strengths and shortcomings of our method based on experimental anaylsis, where we concluded that our approach is moderately succesful: it is good at changing colors and some minor details on the original image while keeping the majority of the content intact. However, localized or specialized modifications are harder to perform because our method focuses too much on maintaining the structure and fine-grained details of the original image. Nevertheless, our method has comparable results to the state-of-the-art for the task. We argue that our approach can be further improved to achieve substantially better results. Insights provided by style transfer literature may also allow us to considerably improve text-based image manipulation theory. We also plan on improving our evaluation procedures by including quantitative human judgement and by testing on the birds dataset [125], which we were unable to do due to time constraints.

We now highlight some of the potential future work for improving our approach. The first improvement is related to the loss network: we plan on exploring a better combination of content and style layers and investigate the effect of image size on which characteristics are captured by the network. Since recent studies [53, 71] show that it is possible to use other computed statistics to represent style information, we also plan on experimenting with replacing the Gram Matrix. This could prove benefitial because the Gram Matrix was originally conceived for non-photorealistic results. Another possible improvement is to add more controls to where the style information is extracted from and applied at. One could use segmentation masks or attention mechanisms to allow the network to focus on specific parts of the image when dealing with style. Some studies also address Semantic Style Transfer, in which given a pair of style and content images which are similar in content, the goal is to build a semantic correspondence between both images and map style regions to a semantically similar content region [58]. The last improvement is to improve the adversarial aspect of our training. As of now, the adversarial loss is used mainly to force the generator to synthesize realistic-looking images, and the discriminator is trained using only images. Thus, one could add conditional information to the discriminator to check if this improves even more the performance.

As for text-based image manipulation in general, the task is in desperate need of improved evaluation metrics for measuring modification quality, since the go-to quantitative measure for now is human judgement based on crowdsourcing. This evaluation measure, unfortunately, has several drawbacks [14]. Evaluating the quality of generated images with human vision is expensive and cumbersome (scales poorly and cannot be iteratively tested with ease), can be biased depending on the evaluation setup presented to evaluators, is difficult to reproduce, does not fully reflect the capacity of models (since we only use a small batch of images), and human inspectors may have high variance, which makes it necessary to average over a large number of subjects. A standardized evaluation measure that could capture the quality of manipulations (and even maybe a specialized dataset or benchmark) would be extremely beneficial for the further advancement of methods.

There is currently a gap between dataset usage: some researchers focus on the Birds [125] and Flowers [88] datasets while others use the Fashion [139] dataset. Thus, researchers should focus on testing their approaches on all three datasets when possible. Analyzing the state-of-the-art as of this writing, the next challenge on this task is to incorporate an explicit control variable to decide the trade-off between maintaining irrelevant aspects and structural modification on an example-by-example basis. Currently, this can only be done at training time and, after that, the model is “locked” to a specific configuration.

REFERENCES

[1] Alpaydin, E. “Introduction to Machine Learning”. The MIT Press, 2014, 640p.

[2] Arjovsky, M.; Bottou, L. “Towards Principled Methods for Training Generative Adversarial Networks”. In: International Conference on Learning Representations, 2017, pp. 17.

[3] Arjovsky, M.; Chintala, S.; Bottou, L. “Wasserstein Generative Adversarial Networks”. In: International Conference on Machine Learning, 2017, pp. 10.

[4] Ba, J. L.; Kiros, J. R.; Hinton, G. E. “Layer Normalization”, *ArXiv e-prints*, vol. 1607.06450, Jul 2016, pp. 14.

[5] Baltrušaitis, T.; Ahuja, C.; Morency, L. “Multimodal Machine Learning: A Survey and Taxonomy”, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 41–2, Feb 2019, pp. 423–443.

[6] Bengio, Y. “Learning Deep Architectures for AI”, *Foundations and Trends in Machine Learning*, vol. 2–1, Nov 2009, pp. 1–127.

[7] Bengio, Y.; Courville, A.; Vincent, P. “Representation Learning: A Review and New Perspectives”, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35–8, Aug 2013, pp. 1798–1828.

[8] Bengio, Y.; Delalleau, O.; Roux, N. L. “The Curse of Highly Variable Functions for Local Kernel Machines”. In: Advances in Neural Information Processing Systems, 2006, pp. 8.

[9] Bengio, Y.; Simard, P.; Frasconi, P. “Learning Long-term Dependencies with Gradient Descent is Difficult”, *IEEE Transactions on Neural Networks*, vol. 5–2, Mar 1994, pp. 157–166.

[10] Berthelot, D.; Schumm, T.; Metz, L. “BEGAN: Boundary Equilibrium Generative Adversarial Networks”, *ArXiv e-prints*, vol. 1703.10717, Mar 2017, pp. 10.

[11] Bishop, C. M. “Neural Networks for Pattern Recognition”. Oxford University Press, Inc., 1995, 502p.

[12] Bishop, C. M. “Pattern Recognition and Machine Learning”. Springer, 2006, 738p.

[13] Bojanowski, P.; Grave, E.; Joulin, A.; Mikolov, T. “Enriching Word Vectors with Subword Information”, *Transactions of the Association for Computational Linguistics*, vol. 5–1, Jun 2017, pp. 135–146.

[14] Borji, A. “Pros and Cons of GAN Evaluation Measures”, *Computer Vision and Image Understanding*, vol. 179–1, Feb 2019, pp. 41–65.

[15] Brock, A.; Lim, T.; Ritchie, J. M.; Weston, N. “Neural Photo Editing with Introspective Adversarial Networks”. In: International Conference on Learning Representations, 2017, pp. 15.

[16] Chen, D.; Yuan, L.; Liao, J.; Yu, N.; Hua, G. “Stereoscopic Neural Style Transfer”. In: IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 16.

[17] Chen, J.; Shen, Y.; Gao, J.; Liu, J.; Liu, X. “Language-based Image Editing with Recurrent Attentive Models”. In: IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 9.

[18] Cheng, M.-M.; Zheng, S.; Lin, W.-Y.; Vineet, V.; Sturgess, P.; Crook, N.; Mitra, N. J.; Torr, P. “ImageSpirit: Verbal Guided Image Parsing”, *ACM Transactions on Graphics*, vol. 34–1, Nov 2014, pp. 3:1–3:11.

[19] Cheng, Y.; Gan, Z.; Li, Y.; Liu, J.; Gao, J. “Sequential Attention GAN for Interactive Image Editing via Dialogue”, *ArXiv e-prints*, vol. 1812.08352, Dec 2018, pp. 10.

[20] Cho, K.; van Merrienboer, B.; Gulcehre, C.; Bahdanau, D.; Bougares, F.; Schwenk, H.; Bengio, Y. “Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation”. In: Conference on Empirical Methods in Natural Language Processing, 2014, pp. 11.

[21] Dash, A.; Gamboa, J. C. B.; Ahmed, S.; Afzal, M. Z.; Liwicki, M. “TAC-GAN – Text Conditioned Auxiliary Classifier Generative Adversarial Network”, *ArXiv e-prints*, vol. 1703.06412, Mar 2017, pp. 9.

[22] Dauphin, Y. N.; Fan, A.; Auli, M.; Grangier, D. “Language Modeling with Gated Convolutional Networks”. In: International Conference on Machine Learning, 2017, pp. 9.

[23] Denton, E. L.; Chintala, S.; Fergus, R.; et al.. “Deep Generative Image Models using a Laplacian Pyramid of Adversarial Networks”. In: Advances in Neural Information Processing Systems, 2015, pp. 9.

[24] Dong, H.; Yu, S.; Wu, C.; Guo, Y. “Semantic Image Synthesis via Adversarial Learning”. In: IEEE International Conference on Computer Vision, 2017, pp. 9.

[25] Dosovitskiy, A.; Tobias Springenberg, J.; Brox, T. “Learning to Generate Chairs with Convolutional Neural Networks”. In: IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 9.

[26] Duchi, J.; Hazan, E.; Singer, Y. “Adaptive Subgradient Methods for Online Learning and Stochastic Optimization”, *Journal of Machine Learning Research*, vol. 12–1, Jul 2011, pp. 2121–2159.

[27] Dumoulin, V.; Shlens, J.; Kudlur, M. “A Learned Representation for Artistic Style”. In: International Conference on Learning Representations, 2017, pp. 26.

[28] Dumoulin, V.; Visin, F. “A Guide to Convolution Arithmetic for Deep Learning”, *ArXiv e-prints*, vol. 1603.07285, Mar 2016, pp. 31.

[29] Efros, A. A.; Freeman, W. T. “Image Quilting for Texture Synthesis and Transfer”. In: SIGGRAPH International Conference on Computer Graphics and Interactive Techniques, 2001, pp. 6.

[30] Elad, M.; Milanfar, P. “Style Transfer via Texture Synthesis”, *IEEE Transactions on Image Processing*, vol. 26–5, May 2017, pp. 2338–2351.

[31] Elman, J. L. “Finding Structure in Time”, *Cognitive Science*, vol. 14–2, Mar 1990, pp. 179–211.

[32] Frome, A.; Corrado, G. S.; Shlens, J.; Bengio, S.; Dean, J.; Mikolov, T.; et al.. “DeViSE: A Deep Visual-semantic Embedding Model”. In: Advances in Neural Information Processing Systems, 2013, pp. 11.

[33] Fu, Z.; Tan, X.; Peng, N.; Zhao, D.; Yan, R. “Style Transfer in Text: Exploration and Evaluation”. In: AAAI Conference on Artificial Intelligence, 2018, pp. 8.

[34] Gatys, L.; Ecker, A. S.; Bethge, M. “Texture Synthesis using Convolutional Neural Networks”. In: Advances in Neural Information Processing Systems, 2015, pp. 9.

[35] Gatys, L. A.; Ecker, A. S.; Bethge, M. “A Neural Algorithm of Artistic Style”, *ArXiv e-prints*, vol. 1508.06576, Aug 2015, pp. 16.

[36] Ghiasi, G.; Lee, H.; Kudlur, M.; Dumoulin, V.; Shlens, J. “Exploring the Structure of a Real-time, Arbitrary Neural Artistic Stylization Network”. In: British Machine Vision Conference, 2017, pp. 27.

[37] Glorot, X.; Bengio, Y. “Understanding the Difficulty of Training Deep Feedforward Neural Networks”. In: International Conference on Artificial Intelligence and Statistics, 2010, pp. 8.

[38] Goodfellow, I.; Bengio, Y.; Courville, A. “Deep Learning”. The MIT Press, 2016, 800p.

[39] 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, pp. 9.

[40] Graves, A. “Supervised Sequence Labelling with Recurrent Neural Networks”. Springer, 2012, 137p.

[41] Grinstein, E.; Duong, N. Q. K.; Ozerov, A.; Perez, P. “Audio Style Transfer”. In: IEEE International Conference on Acoustics, Speech and Signal Processing, 2018, pp. 5.

[42] Gross, S.; Wilber, M. “Training and Investigating Residual Nets”. Source: http://torch.ch/blog/2016/02/04/resnets.html, Mar 2019.

[43] Gulrajani, I.; Ahmed, F.; Arjovsky, M.; Dumoulin, V.; Courville, A. “Improved Training of Wasserstein GANs”. In: Advances in Neural Information Processing Systems, 2017, pp. 11.

[44] Günel, M.; Erdem, E.; Erdem, A. “Language Guided Fashion Image Manipulation with Feature-wise Transformations”, *ArXiv e-prints*, vol. 1808.04000, Aug 2018, pp. 11.

[45] He, K.; Zhang, X.; Ren, S.; Sun, J. “Delving Deep into Rectifiers: Surpassing Human-level Performance on ImageNet Classification”. In: IEEE International Conference on Computer Vision, 2015, pp. 9.

[46] He, K.; Zhang, X.; Ren, S.; Sun, J. “Deep Residual Learning for Image Recognition”. In: IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 9.

[47] Hertzmann, A.; Jacobs, C. E.; Oliver, N.; Curless, B.; Salesin, D. H. “Image Analogies”. In: SIGGRAPH Annual Conference on Computer Graphics and Interactive Techniques, 2001, pp. 14.

[48] Heusel, M.; Ramsauer, H.; Unterthiner, T.; Nessler, B.; Hochreiter, S. “GANs Trained by a Two Time-scale Update Rule Converge to a Local Nash Equilibrium”. In: Advances in Neural Information Processing Systems, 2017, pp. 12.

[49] Hinton, G. E.; Osindero, S.; Teh, Y.-W. “A Fast Learning Algorithm for Deep Belief Nets”, *Neural Computation*, vol. 18–7, Jul 2006, pp. 1527–1554.

[50] Hochreiter, S.; Schmidhuber, J. “Long Short-term Memory”, *Neural Computation*, vol. 9–8, Nov 1997, pp. 1735–1780.

[51] Hornik, K. “Approximation Capabilities of Multilayer Feedforward Networks”, *Neural Networks*, vol. 4–2, Mar-Apr 1991, pp. 251–257.

[52] Hu, Z.; Yang, Z.; Liang, X.; Salakhutdinov, R.; Xing, E. P. “Toward Controlled Generation of Text”. In: International Conference on Machine Learning, 2017, pp. 10.

[53] Huang, X.; Belongie, S. “Arbitrary Style Transfer in Real-Time with Adaptive Instance Normalization”. In: International Conference on Computer Vision, 2017, pp. 10.

[54] Huang, X.; Liu, M.-Y.; Belongie, S.; Kautz, J. “Multimodal Unsupervised Image-to-image Translation”. In: European Conference on Computer Vision, 2018, pp. 18.

[55] Ioffe, S.; Szegedy, C. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: International Conference on Machine Learning, 2015, pp. 9.

[56] Isola, P.; Zhu, J.-Y.; Zhou, T.; Efros, A. A. “Image-to-image Translation with Conditional Adversarial Networks”. In: IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 10.

[57] James, G.; Witten, D.; Hastie, T.; Tibshirani, R. “An Introduction to Statistical Learning – With Applications in R”. Springer, 2013, 436p.

[58] Jing, Y.; Yang, Y.; Feng, Z.; Ye, J.; Yu, Y.; Song, M. “Neural Style Transfer: A Review”, *ArXiv e-prints*, vol. 1705.04058, May 2017, pp. 25.

[59] Johnson, J.; Alahi, A.; Fei-Fei, L. “Perceptual Losses for Real-Time Style Transfer and Super-Resolution”. In: European Conference on Computer Vision, 2016, pp. 18.

[60] Karpathy, A.; Fei-Fei, L. “Deep Visual-semantic Alignments for Generating Image Descriptions”. In: IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 10.

[61] Kato, H.; Ushiku, Y.; Harada, T. “Neural 3D Mesh Renderer”. In: IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 10.

[62] Kim, Y. “Convolutional Neural Networks for Sentence Classification”. In: Conference on Empirical Methods in Natural Language Processing, 2014, pp. 6.

[63] Kingma, D. P.; Ba, J. “Adam: A Method for Stochastic Optimization”. In: International Conference on Learning Representations, 2015, pp. 15.

[64] Kingma, D. P.; Welling, M. “Auto-encoding Variational Bayes”. In: International Conference on Learning Representations, 2014, pp. 14.

[65] Kiros, R.; Salakhutdinov, R.; Zemel, R. “Unifying Visual-Semantic Embeddings with Multimodal Neural Language Models”, *ArXiv e-prints*, vol. 1411.2539, Nov 2014, pp. 13.

[66] Kumar, S. K. “On Weight Initialization in Deep Neural Networks”, *ArXiv e-prints*, vol. 1704.08863, Apr 2017, pp. 9.

[67] Laput, G. P.; Dontcheva, M.; Wilensky, G.; Chang, W.; Agarwala, A.; Linder, J.; Adar, E. “PixelTone: A Multimodal Interface for Image Editing”. In: SIGCHI Conference on Human Factors in Computing Systems, 2013, pp. 10.

[68] LeCun, Y.; Boser, B.; Denker, J. S.; Henderson, D.; Howard, R. E.; Hubbard, W.; Jackel, L. D. “Backpropagation Applied to Handwritten Zip Code Recognition”, *Neural Computation*, vol. 1–4, Dec 1989, pp. 541–551.

[69] LeCun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. “Gradient-based Learning Applied to Document Recognition”, *Proceedings of the IEEE*, vol. 86–11, Nov 1998, pp. 2278–2324.

[70] Leyton-Brown, K.; Shoham, Y. “Essentials of Game Theory: A Concise Multidisciplinary Introduction”. Morgan & Claypool Publishers, 2008, 88p.

[71] Li, Y.; Wang, N.; Liu, J.; Hou, X. “Demystifying Neural Style Transfer”. In: International Joint Conference on Artificial Intelligence, 2017, pp. 7.

[72] Liang, X.; Zhang, H.; Xing, E. P. “Generative Semantic Manipulation with Contrasting GAN”. In: European Conference on Computer Vision, 2018, pp. 17.

[73] Lin, T.-Y.; Maire, M.; Belongie, S.; Hays, J.; Perona, P.; Ramanan, D.; Dollár, P.; Zitnick, C. L. “Microsoft COCO: Common Objects in Context”. In: European Conference on Computer Vision, 2014, pp. 16.

[74] Liu, X.; Meng, G.; Xiang, S.; Pan, C. “Semantic Image Synthesis via Conditional Cycle-Generative Adversarial Networks”. In: International Conference on Pattern Recognition, 2018, pp. 6.

[75] Manjunatha, V.; Iyyer, M.; Boyd-Graber, J.; Davis, L. “Learning to Color from Language”. In: Conference of the North American Chapter of the Association for Computational Linguistics, 2018, pp. 6.

[76] Mao, X.; Li, Q.; Xie, H.; Lau, R. Y.; Wang, Z.; Paul Smolley, S. “Least Squares Generative Adversarial Networks”. In: IEEE International Conference on Computer Vision, 2017, pp. 9.

[77] McCulloch, W. S.; Pitts, W. “A Logical Calculus of the Ideas Immanent in Nervous Activity”, *The Bulletin of Mathematical Biophysics*, vol. 5–4, Dec 1943, pp. 115–133.

[78] Mirza, M.; Osindero, S. “Conditional Generative Adversarial Nets”, *ArXiv e-prints*, vol. 1411.1784, Nov 2014, pp. 7.

[79] Mital, P. K. “Time Domain Neural Audio Style Transfer”, *ArXiv e-prints*, vol. 1711.11160, Nov 2017, pp. 3.

[80] Mitchell, T. M. “Machine Learning”. McGraw-Hill, Inc., 1997, 432p.

[81] Mohapatra, A. “Natural Language Driven Image Edits using a Semantic Image Manipulation Language”, Master’s Thesis, Virginia Tech, 2018, 49p.

[82] Mordvintsev, A.; Olah, C.; Tyka, M. “Inceptionism: Going Deeper into Neural Networks”. Source: https://ai.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html, Mar 2019.

[83] Mordvintsev, A.; Pezzotti, N.; Schubert, L.; Olah, C. “Differentiable Image Parameterizations”. Source: https://distill.pub/2018/differentiable-parameterizations, Mar 2019.

[84] Murphy, K. P. “Machine Learning: A Probabilistic Perspective”. The MIT Press, 2012, 1104p.

[85] Nam, S.; Kim, Y.; Kim, S. J. “Text-Adaptive Generative Adversarial Networks: Manipulating Images with Natural Language”. In: Advances in Neural Information Processing Systems, 2018, pp. 10.

[86] Nesterov, Y. “A Method for Unconstrained Convex Minimization Problem with the Rate of Convergence O $(1/k^2)$”, *Soviet Mathematics–Doklady*, vol. 27–2, Jan 1983, pp. 372–376.

[87] Ng, A. Y.; Jordan, M. I. “On Discriminative vs. Generative Classifiers: A Comparison of Logistic Regression and Naive Bayes”. In: Advances in Neural Information Processing Systems, 2001, pp. 8.

[88] Nilsback, M.-E.; Zisserman, A. “Automated Flower Classification Over a Large Number of Classes”. In: Indian Conference on Computer Vision, Graphics & Image Processing, 2008, pp. 8.

[89] Odena, A.; Dumoulin, V.; Olah, C. “Deconvolution and Checkerboard Artifacts”. Source: http://distill.pub/2016/deconv-checkerboard, Mar 2019.

[90] Odena, A.; Olah, C.; Shlens, J. “Conditional Image Synthesis with Auxiliary Classifier GANs”. In: International Conference on Machine Learning, 2016, pp. 10.

[91] Park, H.; Yoo, Y.; Kwak, N. “MC-GAN: Multi-conditional Generative Adversarial Network for Image Synthesis”. In: British Machine Vision Conference, 2018, pp. 76.

[92] Paszke, A.; Gross, S.; Chintala, S.; Chanan, G.; Yang, E.; DeVito, Z.; Lin, Z.; Desmaison, A.; Antiga, L.; Lerer, A. “Automatic Differentiation in PyTorch”. In: Advances in Neural Information Processing Systems, 2017, pp. 4.

[93] Perez, E.; Strub, F.; De Vries, H.; Dumoulin, V.; Courville, A. “FiLM: Visual Reasoning with a General Conditioning Layer”. In: AAAI Conference on Artificial Intelligence, 2018, pp. 10.

[94] Qian, N. “On the Momentum Term in Gradient Descent Learning Algorithms”, *Neural Networks*, vol. 12–1, Jan 1999, pp. 145–151.

[95] Radford, A.; Metz, L.; Chintala, S. “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks”. In: International Conference on Learning Representations, 2016, pp. 16.

[96] Reed, S.; Akata, Z.; Lee, H.; Schiele, B. “Learning Deep Representations of Fine-grained Visual Descriptions”. In: IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 49–58.

[97] Reed, S.; Akata, Z.; Yan, X.; Logeswaran, L.; Schiele, B.; Lee, H. “Generative Adversarial Text to Image Synthesis”. In: International Conference on Machine Learning, 2016, pp. 10.

[98] Rezende, D. J.; Mohamed, S.; Wierstra, D. “Stochastic Backpropagation and Approximate Inference in Deep Generative Models”. In: International Conference on Machine Learning, 2014, pp. 9.

[99] Rostamzadeh, N.; Hosseini, S.; Boquet, T.; Stokowiec, W.; Zhang, Y.; Jauvin, C.; Pal, C. “Fashion-Gen: The Generative Fashion Dataset and Challenge”, *ArXiv e-prints*, vol. 1806.08317, Jun 2018, pp. 10.

[100] Ruder, S. “An Overview of Gradient Descent Optimization Algorithms”, *ArXiv e-prints*, vol. 1609.04747, Sep 2016, pp. 14.

[101] Rumelhart, D. E.; Hinton, G. E.; Williams, R. J. “Learning Representations by Back-propagating Errors”, *Nature*, vol. 323–6088, Oct 1986, pp. 533–536.

[102] Russakovsky, O.; Deng, J.; Su, H.; Krause, J.; Satheesh, S.; Ma, S.; Huang, Z.; Karpathy, A.; Khosla, A.; Bernstein, M.; et al.. “ImageNet Large Scale Visual Recognition Challenge”, *International Journal of Computer Vision*, vol. 115–3, Dec 2015, pp. 211–252.

[103] Russell, S.; Norvig, P. “Artificial Intelligence: A Modern Approach (3rd Edition)”. Pearson Education, 2009, 1152p.

[104] Salakhutdinov, R.; Hinton, G. “Deep Boltzmann Machines”. In: International Conference on Artificial Intelligence and Statistics, 2009, pp. 8.

[105] Salimans, T.; Goodfellow, I.; Zaremba, W.; Cheung, V.; Radford, A.; Chen, X. “Improved Techniques for Training GANs”. In: Advances in Neural Information Processing Systems, 2016, pp. 9.

[106] Schuster, M.; Paliwal, K. K. “Bidirectional Recurrent Neural Networks”, *IEEE Transactions on Signal Processing*, vol. 45–11, Nov 1997, pp. 2673–2681.

[107] Searle, J. R. “Minds, Brains, and Programs”, *Behavioral and Brain Sciences*, vol. 3–3, Sep 1980, pp. 417–424.

[108] Shen, T.; Lei, T.; Barzilay, R.; Jaakkola, T. “Style Transfer from Non-parallel Text by Cross-alignment”. In: Advances in Neural Information Processing Systems, 2017, pp. 12.

[109] Shinagawa, S.; Yoshino, K.; Sakti, S.; Suzuki, Y.; Nakamura, S. “Interactive Avatar Image Manipulation with Unconstrained Natural Language Instruction using Source Image Masking”. In: Meeting on Image Recognition and Understanding, 2018, pp. 4.

[110] Shinagawa, S.; Yoshino, K.; Sakti, S.; Suzuki, Y.; Nakamura, S. “Interactive Image Manipulation with Natural Language Instruction Commands”, *ArXiv e-prints*, vol. 1802.08645, Feb 2018, pp. 11.

[111] Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al.. “Mastering the Game of Go with Deep Neural Networks and Tree Search”, *Nature*, vol. 529–7587, Jan 2016, pp. 484–489.

[112] Simonyan, K.; Zisserman, A. “Very Deep Convolutional Networks for Large-scale Image Recognition”. In: International Conference on Learning Representations, 2015, pp. 14.

[113] Springenberg, J. T.; Dosovitskiy, A.; Brox, T.; Riedmiller, M. “Striving for Simplicity: The All Convolutional Net”. In: International Conference on Learning Representations, 2014, pp. 14.

[114] Szegedy, C.; Ioffe, S.; Vanhoucke, V.; Alemi, A. A. “Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning”. In: AAAI Conference on Artificial Intelligence, 2017, pp. 6.

[115] Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; Rabinovich, A. “Going Deeper with Convolutions”. In: IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 9.

[116] Szegedy, C.; Vanhoucke, V.; Ioffe, S.; Shlens, J.; Wojna, Z. “Rethinking the Inception Architecture for Computer Vision”. In: IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 9.

[117] Theis, L.; Oord, A. v. d.; Bethge, M. “A Note on The Evaluation of Generative Models”. In: International Conference on Learning Representations, 2016, pp. 10.

[118] Turing, A. M. “Computing Machinery and Intelligence”, *Mind*, vol. 59–236, Oct 1950, pp. 433–460.

[119] Ulyanov, D.; Lebedev, V. “Audio Texture Synthesis and Style Transfer”. Source: https://dmitryulyanov.github.io/audio-texture-synthesis-and-style-transfer, Mar 2019.

[120] Ulyanov, D.; Lebedev, V.; Vedaldi, A.; Lempitsky, V. S. “Texture Networks: Feed-forward Synthesis of Textures and Stylized Images”. In: International Conference on Machine Learning, 2016, pp. 9.

[121] Ulyanov, D.; Vedaldi, A.; Lempitsky, V. “Instance Normalization: The Missing Ingredient for Fast Stylization”, *ArXiv e-prints*, vol. 1607.08022, Jul 2016, pp. 6.

[122] Vendrov, I.; Kiros, R.; Fidler, S.; Urtasun, R. “Order-Embeddings of Images and Language”. In: International Conference on Learning Representations, 2016, pp. 12.

[123] Verma, P.; Smith, J. O. “Neural Style transfer for Audio Spectrograms”, *ArXiv e-prints*, vol. 1801.01589, Jan 2018, pp. 3.

[124] Vinyals, O.; Toshev, A.; Bengio, S.; Erhan, D. “Show and Tell: A Neural Image Caption Generator”. In: IEEE conference on Computer Vision and Pattern Recognition, 2015, pp. 9.

[125] Wah, C.; Branson, S.; Welinder, P.; Perona, P.; Belongie, S. “The Caltech-UCSD Birds-200-2011 Dataset”, Technical Report, California Institute of Technology, 2011, 8p.

[126] Wang, H.; Liang, X.; Zhang, H.; Yeung, D.-Y.; Xing, E. P. “ZM-Net: Real-time Zero-shot Image Manipulation Network”, *ArXiv e-prints*, vol. 1703.07255, Mar 2017, pp. 9.

[127] Wang, H.; Williams, J. D.; Kang, S. “Learning to Globally Edit Images with Textual Description”, *ArXiv e-prints*, vol. 1810.05786, Oct 2018, pp. 21.

[128] Wang, T.-C.; Liu, M.-Y.; Zhu, J.-Y.; Tao, A.; Kautz, J.; Catanzaro, B. “High-resolution Image Synthesis and Semantic Manipulation with Conditional GANs”. In: IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 10.

[129] Wehrmann, J.; Barros, R. C. “Bidirectional Retrieval Made Simple”. In: IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 9.

[130] Wehrmann, J.; Lopes, M. A.; More, M. D.; Barros, R. C. “Fast Self-attentive Multimodal Retrieval”. In: IEEE Winter Conference on Applications of Computer Vision, 2018, pp. 8.

[131] Werbos, P. J.; et al.. “Backpropagation Through Time: What it Does and How to Do it”, *Proceedings of the IEEE*, vol. 78–10, Oct 1990, pp. 1550–1560.

[132] Wu, Y.; He, K. “Group Normalization”. In: European Conference on Computer Vision, 2018, pp. 17.

[133] Xu, T.; Zhang, P.; Huang, Q.; Zhang, H.; Gan, Z.; Huang, X.; He, X. “AttnGAN: Fine-Grained Text to Image Generation with Attentional Generative Adversarial Networks”. In: IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 9.

[134] Zeiler, M. D.; Fergus, R. “Visualizing and Understanding Convolutional Networks”. In: European Conference on Computer Vision, 2014, pp. 16.

[135] 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 International Conference on Computer Vision, 2016, pp. 9.

[136] Zhang, R.; Isola, P.; Efros, A. A. “Colorful Image Colorization”. In: European Conference on Computer Vision, 2016, pp. 18.

[137] Zhu, J.-Y.; Krähenbühl, P.; Shechtman, E.; Efros, A. A. “Generative Visual Manipulation on the Natural Image Manifold”. In: European Conference on Computer Vision, 2016, pp. 17.

[138] Zhu, J.-Y.; Park, T.; Isola, P.; Efros, A. A. “Unpaired Image-to-image Translation Using Cycle-consistent Adversarial Networks”. In: IEEE International Conference on Computer Vision, 2017, pp. 10.

[139] Zhu, S.; Urtasun, R.; Fidler, S.; Lin, D.; Change Loy, C. “Be Your Own Prada: Fashion Synthesis with Structural Coherence”. In: IEEE International Conference on Computer Vision, 2017, pp. 1689–1697.

[140] Zou, C.; Mo, H.; Du, R.; Wu, X.; Gao, C.; Fu, H. “LUCSS: Language-based User-customized Colourization of Scene Sketches”, *ArXiv e-prints*, vol. 1808.10544, Aug 2018, pp. 14.

APPENDIX A – DETAILED NETWORK ARCHITECTURES

Table A.1: Generator architecture.

Layer	Filter Size	Stride	Padding	Tensor Shape
Input: x	-	-	-	$128 \times 128 \times 3$
Conv. 2D	4×4	2	1	$64 \times 64 \times 64$
ReLU	-	-	-	$64 \times 64 \times 64$
Conv. 2D	4×4	2	1	$32 \times 32 \times 128$
IN + ReLU	-	-	-	$32 \times 32 \times 128$
Conv. 2D	4×4	2	1	$16 \times 16 \times 256$
IN + ReLU	-	-	-	$16 \times 16 \times 256$
Conv. 2D	4×4	2	1	$8 \times 8 \times 512$
IN + ReLU	-	-	-	$8 \times 8 \times 512$
Output: $E(x)$	-	-	-	$8 \times 8 \times 512$
Input: $E(x)$	-	-	-	$8 \times 8 \times 512$
Input: $MLP(s)$	-	-	-	-
ResBlock	-	-	-	$8 \times 8 \times 512$
ResBlock	-	-	-	$8 \times 8 \times 512$
ResBlock	-	-	-	$8 \times 8 \times 512$
ResBlock	-	-	-	$8 \times 8 \times 512$
$E(x) + ResBlocks(x, MLP(s))$	-	-	-	$8 \times 8 \times 512$
Output: $Res(E(x), MLP(s))$	-	-	-	$8 \times 8 \times 512$
Input: $Res(E(x), MLP(s))$	-	-	-	$8 \times 8 \times 512$
Upsample (NN)	-	-	-	$16 \times 16 \times 512$
Conv. 2D	3×3	1	1	$16 \times 16 \times 256$
BN + ReLU	-	-	-	$16 \times 16 \times 256$
Upsample (NN)	-	-	-	$32 \times 32 \times 256$
Conv. 2D	3×3	1	1	$32 \times 32 \times 128$
BN + ReLU	-	-	-	$32 \times 32 \times 128$
Upsample (NN)	-	-	-	$64 \times 64 \times 128$
Conv. 2D	3×3	1	1	$64 \times 64 \times 64$
BN + ReLU	-	-	-	$64 \times 64 \times 64$
Conv. 2D	3×3	1	1	$128 \times 128 \times 3$
Tanh	-	-	-	$128 \times 128 \times 3$

Table A.2: Discriminator architecture.

Layer	Filter Size	Stride	Padding	Tensor Shape
Input: x	-	-	-	$128 \times 128 \times 3$
Conv. 2D	4×4	2	1	$64 \times 64 \times 64$
LeakyReLU	-	-	-	$64 \times 64 \times 64$
Conv. 2D	4×4	2	1	$32 \times 32 \times 128$
BN + LeakyReLU	-	-	-	$32 \times 32 \times 128$
Conv. 2D	4×4	2	1	$16 \times 16 \times 256$
BN + LeakyReLU	-	-	-	$16 \times 16 \times 256$
Conv. 2D	4×4	2	1	$8 \times 8 \times 512$
BN + LeakyReLU	-	-	-	$8 \times 8 \times 512$
Conv. 2D	4×4	2	1	$4 \times 4 \times 512$
BN + LeakyReLU	-	-	-	$4 \times 4 \times 512$
Conv. 2D	4×4	2	1	$1 \times 1 \times 1$

Table A.3: Residual Block architecture.

Layer	Filter Size	Stride	Padding	Tensor Shape
Input: z	-	-	-	$W \times H \times C$
Input: $MLP(s)$	-	-	-	-
Conv. 2D	3×3	1	1	$W \times H \times C$
AdaIN	-	-	-	$W \times H \times C$
ReLU	-	-	-	$W \times H \times C$
Conv. 2D	3×3	1	1	$W \times H \times C$
AdaIN	-	-	-	$W \times H \times C$
$z + ResBlock(z, MLP(s))$	-	-	-	$W \times H \times C$

MARISTA PUCRS

Pontifícia Universidade Católica do Rio Grande do Sul
Pró-Reitoria de Graduação
Av. Ipiranga, 6681 - Prédio 1 - 3º. andar
Porto Alegre - RS - Brasil
Fone: (51) 3320-3500 - Fax: (51) 3339-1564
E-mail: prograd@pucrs.br
Site: www.pucrs.br