Learning-based Image Synthesis by Utilizing Disentangled Feature Representations

by

Minh-Duc VO

Dissertation

submitted to the Department of Informatics

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

The Graduate University for Advanced Studies, SOKENDAI

September 2020

Abstract

Image synthesis is to render a novel image from given inputs. Besides having a wide range of applications, it plays an important role to further step in computer vision research in which many work effort to move from visual-level understanding to reasoning-level understanding. Thanks to deep learning, learning-based image synthesis has been established where a deep network is used to first learn feature space from given inputs and then map the learned feature space to the image domain. Though learning-based models obtain remarkable results, they still limit in generating faithful and realistic images. Their shortcomings come from the fact that the inputs of image synthesis are themselves diverse, leading multiple descriptive feature representations can be obtained in the feature space along with the depth of the network. Consequently, simply mapping all the feature representations (at the same layer in the network) is unable to elaborate the contribution of each feature representation in the generated image as our expectation. Based on the fact that the more helpful feature space is attained, the more chance to generate better images, faithfully understanding and effectively utilizing the feature representations thus is crucial in robustly elevating the performance of image synthesis.

Needless to say, the feature representations are disentangled and have different roles in generating image. This dissertation, therefore, addresses learning-based image synthesis by an introduction to a novel approach that fully takes into account the feature space. Our approach first selects disentangled feature representations depending on the role to obtain descriptive information. It then combines the disentangled feature representations using an appropriate mapping process to generate images faithfully and realistically. Generally, our approach is potential to deal with a wide range of image synthesis tasks. We therefore apply our proposed way on three interestingly challenging tasks including (i) rendering image contents in different styles (i.e., style transfer), (ii) image manipulation with text and (iii) text-to-image synthesis tasks. The comprehensive experiments manifest that our proposed approach is sufficient and flexible to handle many tasks in image synthesis.

The first task, rendering image contents in different styles, is to render given image contents in given styles. This task requires to preserve contents and to faithfully render of styles. We thus propose a feed-forward network having two distinct streams

to learn disentangled content and style features. These features are then combined using our proposed adaptive feature injection and concatenation which fully take into account contribution of the content and the style features in stylized images. In order to train our proposed network, we employ a loss network, the pre-trained VGG-16, to compute content loss and style loss, both of which are efficiently used for the feature injection as well as the feature concatenation.

The second task, image manipulation with text, on the other hand, is to render foreground (object) given as a text description into a given source image. It requires to disentangle the semantics contained in (source) image and text information and then combine the disentangled semantics to synthesize realistic images. We propose a generative adversarial network (GAN) where the network possesses one generator and a pair of discriminators with different architecture, called *Paired-D GAN*. The generator has encoder–decoder architecture with skip-connections and synthesizes an image matching the given text description while preserving other parts of the source image. Two discriminators, on the other hand, judge foreground and background of the generated image separately to meet an input text description and a source image. We also propose a three-player adversarial learning process to simultaneously and effectively train our *Paired-D GAN*.

The third task, text-to-image synthesis, is to render a novel image that is consistent with a given text description. This task requires not only entity information (i.e., object type, attribute, shape...) but also relations among entities (i.e., position, interaction...). Since the gap between text description and image is large, we thus follow two-step approach where inference of the scene layout as an intermediate representation between text and image is followed by using the layout to generate images. The layout in previous work is constructed through only the comprehensive usage of relation among entities for bounding-boxes' localization, resulting generated images may have poor scene structure as a whole even if each entity is realistically rendered. We step further in predicting visual-relation layout by employing not only all available relations together but also individual relation separately. More precise, we first comprehensively use all available relations together to localize initial bounding-boxes of all the entities. Next, we use individual relation separately to predict from the initial bounding-boxes relation-units for all the relations in the input text. Since each entity may involve in multiple relations, we then unify all the relation-units to produce the visual-relation

layout. Finally, our visual-relation layout is conditioned on a stack of three GAN, namely stacking-GAN, to generate images that consistently capture the scene structure.

We evaluate our approach on publicly available dataset. More precise, for rendering image contents in different styles, we use images in the MS-COCO 2014 dataset as our content images, and six famous paintings widely used in style transfer as our style images. For image manipulation with text, we conduct experiments on the Caltech-200 bird dataset and the Oxford-102 flower dataset. For text-to-image synthesis, we verify our method on challenging 2017 COCO-stuff dataset and Visual GENOME dataset. The intensive and extensive experiments show outperformances of our approach against state-of-the-arts.

Acknowledgement

I would first like to express my sincere gratitude to my advisor Prof. Akihiro Sugimoto for his contribution, motivation, patience, and immense knowledge. He always convincingly guided and encouraged me to do the right thing even the research progress got tough. I have been lucky to be a member of his research group where my research skill becomes professional. Without his persistent help, the goal of this dissertation would not have been completed.

Besides my advisor, I wish to acknowledge the rest of my Ph.D. committee: Prof. Miyao Yusuke, Prof. Imari Sato, Prof. Hiroshi Ishikawa, and Prof. Satoshi Ikehata, for their encouragement, and insightful comments.

I thank my fellow labmates, for their helpful discussions and sharing unforgettable memories together.

I am also thankful to all people at SOKENDAI and NII for their kind helps during my stay in Japan and study in NII.

Last but not the least, I would like to thank my family, my friends for continuously supporting me spiritually throughout my journey.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Background

Human has two ways of thinking: one is unconscious and impulsive while the other is very conscious, aware and considerate [24]. With respect to two ways of thinking, our visual system functions two kinds of understanding: (i) visual-level understanding and (ii) reasoning-level understanding [25]. With one glance at a scene, we immediately recognize objects appearing in that scene (visual-level understanding) while it takes time to infer further information such as relations among objects (reasoning-level understanding). The visual-level understanding is to acquire and improve knowledge from the outside environment through experiences by remembering visual features such as color, shape, or viewpoint. The reasoning-level understanding, on the other hand, is to infer new knowledge from previously acquired knowledge for higher-level understanding where visual features are considered in different aspects such as meaning, role, or relations among visual features. As a result, higher-level understanding allows us to describe the scene in various forms such as a painting, an image, or a sentence.

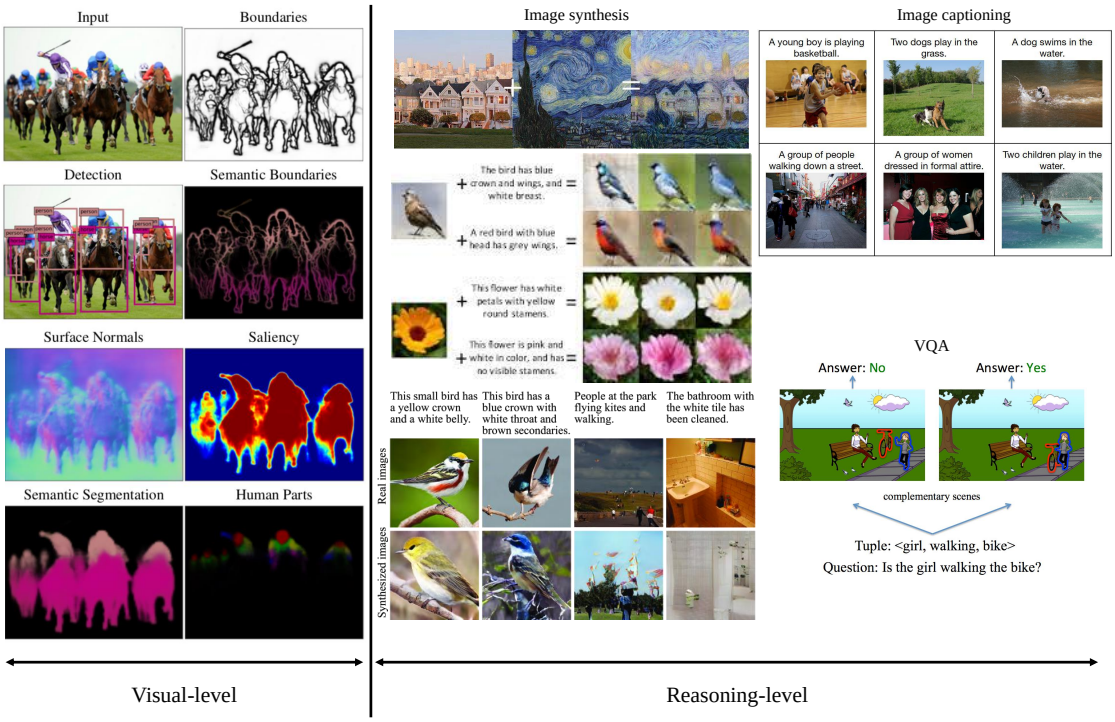


Figure 1.1: Examples of computer vision problems.

To simulate our visual system, computer vision research has been developed for years. "At an abstract level, the goal of computer vision problems is to use the observed image data to infer something about the world" [26]. Fig. 1.1 illustrates some popular problems in computer vision.

With the progress of computer vision, a wide variety of problems have been launched aiming to solve the visual-level understanding (Fig. 1.1, left). With either hand-crafted features (photometric stereo [27], SIFT [28], for examples) or auto-learned features from deep learning models (AlexNet [29], VGG [30], ResNet [31], for examples), state-of-the-art vision systems can reliably handle a wide range of long-standing problems such as object detection [32], object recognition [29–31], semantic segmentation [33]. To some extent, computer vision technologies perform even better than human ability [34–36].

To cope the reasoning-level understanding in computer vision, some novel problems have been launched recently such as image synthesis [7–9, 11, 13, 17–20, 22, 37–48], image captioning [49–52], visual question answering [53, 54] (Fig. 1.1, right). In

contrast to human, current computer vision technologies still have difficulty in solving these new problems. Moving towards reasoning-level understanding in computer vision is thus desired [55].

Among the reasoning-level understanding problems, image captioning and visual question answering achieve reasonable results with help from state-of-the-arts in visual-level understanding such as object detection and object recognition. On the contrary, the current stage of image synthesis is still far from its goal because of the diversity of given inputs. In practice, image synthesis has a wide range of applications such as intelligent image manipulation, game industry, mobile application, medical research, or image search engines. In addition, it also potentially supplement a massive amount of data which benefits the training in deep learning. To elevate the next step in computer vision, this dissertation therefore investigates image synthesis problem.

1.2 Motivation

Image synthesis is a process that creates a novel image from given inputs. The typical given inputs are attributes [38], textures [39], images [7–9, 11, 37, 40–44], and texts [13, 17–20, 22, 45–48]. The diversity of given inputs raises some sort of image synthesis tasks such as constrained image synthesis [38, 39], image-to-image translation [37, 40, 41], style transfer [7–9, 11, 42–44], image manipulation with text [13, 56, 57], or text-to-image [17–20, 22, 45–48].

Early work on image synthesis was reported in the context of texture synthesis by using hand-crafted features [39, 58, 59]. These methods had limited results because they were not able to deal with various kinds of inputs. Image synthesis therefore stepped slowly during an early stage. Thanks to deep learning, image synthesis has been relaunched recently. Gatys et al. [9] first employed convolutional neural network (CNN) for style transfer task. At the same time, Reed at al. [45] used text features extracted from a pre-trained text encoder to generate a novel image from text description. Dong et al. [13] proposed image manipulation with text description by generating an image that matches the semantic meaning of the input text description while maintaining other parts of a source image. These work broke through previous methods relying on hand-crafted features, and established new benchmarks. Following the success of first attempts that employ deep learning in image synthesis [9, 13, 45],

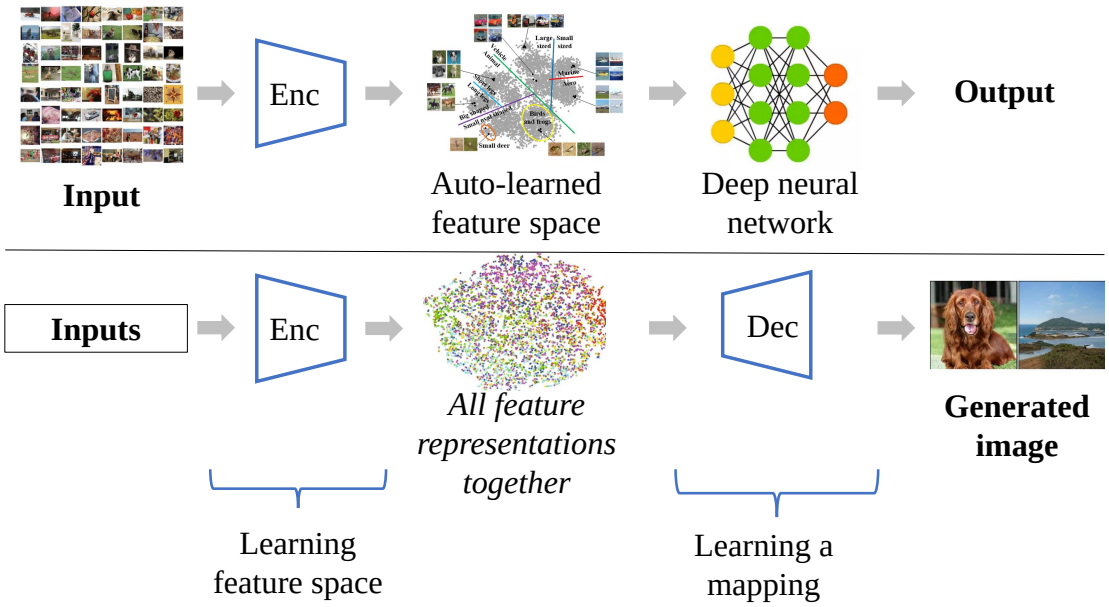


Figure 1.2: Deep learning pipelines used in basic computer vision methods (first row), and previous learning-based image synthesis methods (second row).

many follow-up work such as [7, 8, 17, 19, 56, 60] proposed various deep learning architectures to generate better images. Using deep learning, therefore, plays an important role in generating images and establishes the learning-based image synthesis approach.

Although learning-based image synthesis methods obtain remarkable results, they still struggle to faithfully and realistically generate images. The image synthesis problem remains unsolved. This is because the way of using the deep learning approach is not effective. As shown in Fig. 1.2, similar to other tasks in computer vision using deep learning models (for examples, object detection [32], object recognition [29–31]), learning-based image synthesis models can be decomposed into two stages: (i) learning feature space from input(s) and (ii) learning a mapping from feature space to image domain. The former stage exploits characteristics of input(s) to extract feature representations while the latter one transforms the extracted feature representations to create images. These two stages together bring gains on performance. Both the stages have been considered in previous work. However, prior work paid more attention to one stage while alleviating the other stage. For example, Dong et al. [13] (image manipulation with text) focused on learning image and text feature representations

from inputs without consideration on learning a mapping (i.e., their mapping process was not good). Gatys et al. [9] (style transfer), on the other hand, learned a mapping by minimizing content and style losses in stylized images. Their method, however, captured content and style feature representations from the same layer of a deep neural network, leading feature space cannot retain useful information from both the content and the style. Above examples show that either a good feature space or a good mapping is not necessarily able to handle image synthesis task properly. These shortcomings limit the ability of learning-based image synthesis in terms of image quality. Therefore, focusing on both the stages at the same time is crucial to improve the quality of generated images.

Previous methods relied on the network to extract features from given input(s) and then use these feature representations together to produce new images (Fig. 1.2, second row). Along this line, previous work often improved network architecture by adding more and more layers [18, 41, 46, 61] or employing a so-called attention mechanism [19]. Such improvements significantly bring gains on performance of image synthesis but raise difficulty in training and demand on memory resource. For example, although BigGAN [61] introduced a set of tricks that lead a new benchmark for generating high resolution images, it requires a super-powerful computer to train. Therefore, exploring such network structures is not promising and limits applications of image synthesis at a consumer level. Instead of increasing capability of networks (i.e., network parameters), investigating the input would be a better choice. This is because input itself is universe and has different roles in generated image. Consequently, exploiting feature space is promising to improve image synthesis. Yet such investigation is not well explored in the literature.

The given inputs bring multiple levels of descriptive information. Consequently, as pointed out in previous work [9, 62, 63], the feature space can be separated into disentangled feature representations where each of them takes different role. Understanding disentangled feature representations naturally enables us to bring desired information from inputs to generated images, resulting in generating better images. Accordingly, image synthesis models need to adequately exploit feature space in a deep neural network. Unfortunately, the current usage of feature space is insufficient. Previous work usually made use of feature representations at the same level (i.e., extracted from the same layer in a deep neural network) together, resulting in

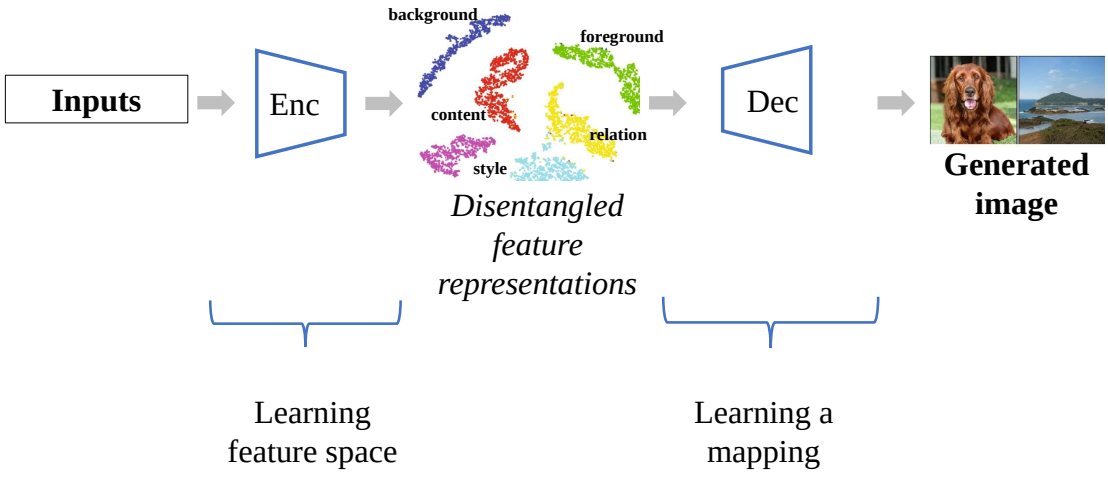


Figure 1.3: Overview of our approach.

uncontrollable contributions of feature representations in generated image. As a result, the generated image are less realistic. On the other hand, employing disentangled feature representations makes sure that we are able to handle the generated images as our expectation. Needless to say, a (simple) deep network is capable of capturing multiple informative feature representations along with its depth. We thus select useful representations and utilize them to generate images depending on the task. For example, let us consider image manipulation with text where its goal is to render foreground (object) given as a text description into a given source image. If we use all feature representations extracted at a same layer, the background and foreground information may be mixed, leading we cannot successfully handle the task. On the contrary, based on the fact that the network weights background representations at early layers and foreground representations at latter layers [9, 62, 63], we can select these disentangled feature representations to ensure that only descriptive information (i.e., background and foreground information) are used to properly form the generated images. More precisely, the network first edits the foreground representations to match the text whereas the background representations are retained. Then, it combines the (new) foreground representations and the (original) background representations to produce a new image that satisfies the requirement of the task. Inductively, the idea of using disentangled representations is able to apply for other image synthesis tasks. Therefore, employing disentangled feature representations is crucial.

Besides learning feature space from input, learning a mapping from feature space to image domain also advances the quality of generated images as discussed above. This stage involves two steps: (i) combining feature representations and (ii) training a network to achieve the goal. In the first step, previous work [7, 9, 45] often simply combine all the feature representations with the same ratio (in other words, naively concatenating feature representations before feeding them into the image decoder). However, since each feature representation has its own role, the simple concatenation operation is unable to bring the contributions of all the feature representations in generated images as expected. For instance, in style transfer task, the style representation (in Gatys et al. [9]) is dominant while the work by Johnson et al. [7] advocates the content representation. Consequently, neither Gatys et al. [9] nor Johnson et al. [7] is capable of controlling the appearance of content and style as expected. To tackle the above mentioned issue, some work [19, 56] employs attention mechanism to automatically weight the contribution of feature representations, leading better generated images. However, the attention layer often overweights the features that appear frequently. This is because the attention mechanism does not indeed understand the role of features and thus tends to learn the attention weights based on the number of occurrences of the features (patterns) during training. As a result, more frequently appearing features receive more attention. For example, the results by [19] tend to focus on the details of objects described in the given text while alleviating other information such as background, relations among objects. Therefore, how to effectively combine the feature representations extracted from feature space should be carefully considered.

The second step in learning a mapping (i.e., training a network) is nontrivial due to lack of well-annotated ground-truth data. To train a model for image synthesis, we need to design an objective and appropriate loss function. For examples, Gatys et al. [9], Johnson et al. [7] proposed a perceptual loss which is a summation of content and style loss. Reed et al. [45], on the other hand, employed an adversarial loss [64] to enforce the network to generate realistic images. Though loss functions introduced in previous work can be used to optimize the network, they do not always converge at expected point. For instance, in Gatys et al. [9], when the ratio of content and style in the loss function is significantly changed, the convergence point does not change so much. Another example is the loss function in Dong et al. [13] (image manipulation

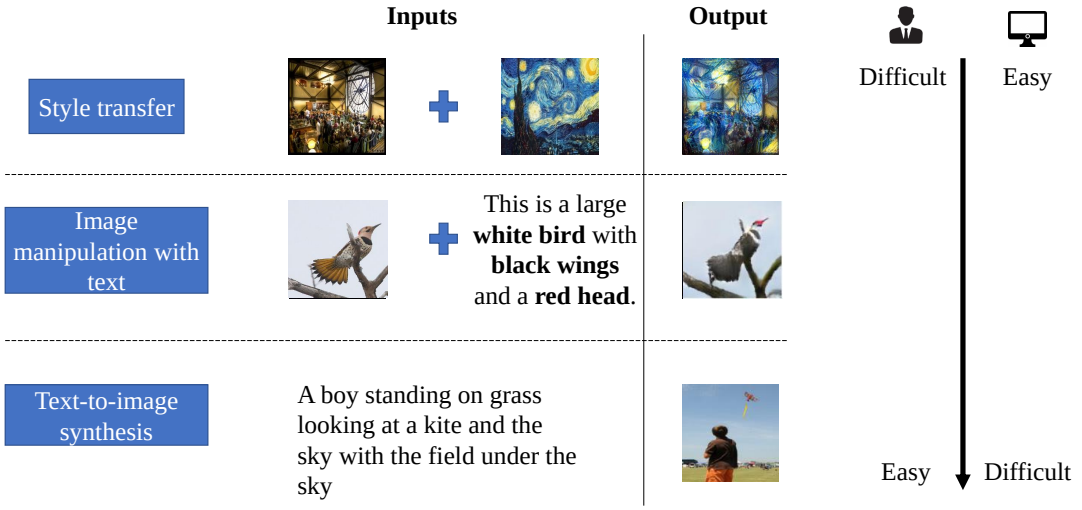


Figure 1.4: Inputs and output of rendering image contents in different styles (style transfer), image manipulation with text and text-to-image synthesis tasks.

with text task) where they use one discriminator (e.g., one adversarial loss function) to simultaneously deal with both foreground and background. Because the foreground and background feature representations are different, using one discriminator is unable to give strong feedback to the generator; generated images by Dong et al. [13] cannot successfully render the foreground according to a given text while retaining the background in a source image. Aforementioned examples suggest that designing a good training strategy is also a key to improve the quality of generated images.

Inspired by aforementioned discussions, this dissertation investigates learning-based image synthesis by utilizing disentangled feature representations depending on the role in the task (Fig. 1.3). In particular, we address three challenging tasks: (i) rendering image contents in different styles (style transfer), (ii) image manipulation with text and (iii) text-to-image synthesis. More precise, we employ disentangled feature representations (in feature space) and design an appropriate mapping process depending on specific task. More details of our research statement and methodology will be described in next section.

1.3 Research Statement and Methodology

Though image synthesis can handle a wide range of inputs, only either image or text description is friendly and understandable to us. Using these kinds of inputs in image synthesis is thus interesting and has received a lot of attentions. The different combinations of these image and text descriptions promote different tasks including rendering image contents in different styles (style transfer), image manipulation with text and text-to-image synthesis. The definition of each task is follows. (i) Rendering image contents in different styles is to render given image contents in given styles. (ii) Image manipulation with text is to render foreground (object) given as a text description into a given source image. (iii) Text-to-image synthesis is to render a novel image that is consistent with input text description. Fig. 1.4 illustrates the inputs and output of each task. From (i) to (iii), the tasks gradually become easy to human, but more challenging to the computer. Since either image or text description conveys different meaningful roles in each task as discussed above, understanding and utilizing appropriate feature representations depending on the role is important to faithfully generate images. This question still opens and remains challenging. Therefore, this dissertation addresses this challenge by exploiting the three tasks in a row.

Instead of improving deep learning architectures and learning the improved model as aforementioned work, we introduce a novel approach in image synthesis that fully takes into account feature representations in a deep neural network. More precisely, our approach first selects disentangled feature representations with respect to their roles in a specific task to obtain useful information. It then utilizes the disentangled feature representations to generate images. By utilizing disentangled feature representations, we are able to decide what to bring from input to generated images. Therefore, our approach is sufficient and flexible to apply to various tasks in image synthesis. In particular, we investigate our approach on three interestingly challenging tasks including rendering image contents in different styles, image manipulation with text, and text-to-image.

For rendering image contents in different styles, we experimentally found that the deep network extracts the style representation at early layers while doing the content representation at latter layers. We, therefore, prepare two distinct networks to deal with the content and the style representations separately. More precisely, a deep network

(in terms of number of hidden layers) is used to obtain the content representation while a shallow one extracts the style representation. The two representations are then utilized in an adaptive manner where our model takes into account the contribution of each representation through the ratio of content and style loss in (total) loss value.

For image manipulation with text, based on the observation that the background representation is weighted at early layers in the network while the foreground representation is obtained at latter layers, we first extract the background and the foreground representations at different layers in the network. Next, we match the foreground representation with the text feature obtained from the pre-trained text encoder (i.e., concatenating foreground representation and text feature). Finally, the manipulated foreground representation and the background representation are utilized to produce a new image that matches the given text description while retaining other irrelevant information in the source image. However, simply matching the foreground representation and the text feature does not ensure that the generated images satisfy the requirement of the task. This is because the (image) foreground representation is totally different from the text feature. For robust foreground-text matching, we therefore design a novel mapping process that fully makes use of the background and the foreground representations. More precisely, the background representation is used to verify whether the background in generated image is the same as that of source image. The foreground representation, on the other hand, is employed to enhance the foreground-text matching.

For text-to-image synthesis, we design a two-step model that first infers an image layout from text description and then converts the layout to image. Based on the observation that the comprehensive usage of relations in previous work loses the individual relation information in predicting a layout, we propose to use relations comprehensively and individually to infer the layout. Our model first aggregates all available relations to obtain the comprehensive relation representation and select each relation one by one for the individual relation representation. It then uses the comprehensive relation representation to infer initial bounding-boxes. Next, our model uses the individual relation representations to adjust the initial bounding-boxes for producing the layout. Finally, the layout is converted to the image.

1.4 Main Contributions

Many existing work with complex network architectures are struggling to deal with image synthesis because of diversity of inputs. More precisely, the role of inputs is different according to specific task. For rendering image contents in different styles, when we stylize images, the content gives us what exist (object shapes and locations) in the rendered image and the style gives us the impression of the rendered image. For image manipulation with text, the source image gives us background information while the given text describes foreground information. For text-to-image, the text description not only describes object information but also presents relations among objects. These mean that we cannot obtain all those features at the same level (layer) in a deep neural network. We instead address learning-based image synthesis by utilizing disentangled feature representations depending on the role. Since the feature space contains multiple levels of feature representations, our work therefore is effective to handle the shortcomings in current image synthesis methods. Comprehensive results on addressing tasks show effectiveness of our proposed approach. In the direction of our research, the main contributions of this dissertation are follows:

- For rendering image contents in different styles, we propose an end-to-end two-stream Fully Convolutional Networks (FCNs) aiming at balancing the contributions of the content and the style in rendered images. Our proposed network consists of the encoder and decoder parts. The encoder part utilizes a FCN for content and a FCN for style where the two FCNs have feature injections and are independently trained to preserve the semantic content and to learn the faithful style representation in each. The semantic content feature and the style representation feature are then concatenated adaptively and fed into the decoder to generate style-transferred (stylized) images. In order to train our proposed network, we employ a loss network, the pre-trained VGG-16, to compute content loss and style loss, both of which are efficiently used for the feature injection as well as the feature concatenation. Our intensive experiments show that our proposed model generates more balanced stylized images in content and style than state-of-the-art methods. Moreover, our proposed network achieves efficiency in speed.

- For image manipulation with text, we propose a generative adversarial network having a pair of discriminators with different architectures, called *Paired-D GAN*, where the two discriminators make different judgments: one for foreground synthesis and the other for background synthesis. The generator of paired-D GAN has the encoder-decoder architecture with skip-connections and synthesizes an image matching the given text description while preserving other parts of the source image. The two discriminators judge foreground and background of the synthesized image separately to meet an input text description and a source image. The paired-D GAN is trained using the effective adversarial learning process in a simultaneous three-player minimax game. Experimental results on the Caltech-200 bird dataset and the Oxford-102 flower dataset show that Paired-GAN is capable of semantically synthesizing images to match an input text description while retaining the background in a source image against the state-of-the-art methods.

- For text-to-image synthesis, we propose an end-to-end network for image generation from given structured-text that consists of the visual-relation layout module and the pyramid of GANs, namely stacking-GANs. Our visual-relation layout module uses relations among entities in the structured-text in two ways: comprehensive usage and individual usage. We comprehensively use all available relations together to localize initial bounding-boxes of all the entities. We also use individual relation separately to predict from the initial bounding-boxes relation-units for all the relations in the input text. We then unify all the relation-units to produce the visual-relation layout, i.e., bounding-boxes for all the entities so that each of them uniquely corresponds to each entity while keeping its involved relations. Our visual-relation layout reflects the scene structure given in the input text. The stacking-GANs is the stack of three GANs conditioned on the visual-relation layout and the output of previous GAN, consistently capturing the scene structure. Our network realistically renders entities' details in high resolution while keeping the scene structure. Experimental results on two public datasets show outperformances of our method against state-of-the-art methods.

1.5 Organization of the Dissertation

The rest of this dissertation is organized as follows.

- Chapter 2 recalls some basic concepts which are related to image synthesis. It also reviews current approaches in image synthesis and analyzes the advantages and the disadvantages of these approaches. Finally, we present overview of our approach for each task.

- Chapter 3 presents our two-stream FCNs network for rendering image contents in different styles. Our network employs disentangled content and style feature representations in an adaptive manner to take into account the contribution of the content and the style feature representations in stylized images.

- Chapter 4 introduces our *Paired-D GAN* which consists of one generator and a pair of discriminators for image manipulation with text. To train our *Paired-D GAN*, this chapter presents a three-player adversarial training which is helpful to force the generated image keeps background as in the source image while its foreground meets the input text.

- Chapter 5 presents a two-step approach for text-to-image synthesis where inference layout is followed by layout-to-image step. Our focus is introduction to construct visual-relation layout by using subject–predicate–object relations between entities extracted from an input structured-text not only comprehensively but also individually.

- Chapter 6 summarizes our contributions in this dissertation. We also discuss direction of our future work.

2

Literature Review

Image synthesis is a long-standing and fundamental research in computer vision. Early work on image synthesis was closely related to texture synthesis and color transfer [39, 59]. Some methods there used histogram matching [58] and/or non-parametric sampling [39, 59]. These methods had limited results because they relied on hand-crafted low-level features and often failed in capturing image structures effectively.

Inspired by the impressive progress of various tasks in computer vision using deep neural networks, image synthesis has recently relaunched in both academy and industry and has achieved remarkable results.

In this chapter, this dissertation first recalls some basic concepts regarding to our work. It next discusses two main approaches in image synthesis. Finally, we present overview of our approach.

2.1 Preliminary Background

2.1.1 Deep learning

Deep learning is a subset of machine learning which is designed following neural network in human brain. The idea of deep learning has been launched for a couple of years but stepped slowly in early stage due to the limitations of computing resources. With the rapid development of powerful computers, deep learning is getting more attentions nowadays. In literature, the definition of deep learning is: "Deep learning is a particular kind of machine learning that achieves great power and flexibility by learning to represent the world as nested hierarchy of concepts, with each concept defined in relation to simpler concepts, and more abstract representations computed in terms of less abstract ones." [65]. The differences of traditional algorithms in machine learning and deep learning can be summarized as follows:

- Training dataset: the performance of deep learning depends on amount of training dataset. When the dataset is small, a deep learning model does not work well. If we increase the size of the dataset, the performance of deep learning exponentially becomes better and better. This is because with a large-scale dataset, deep learning model is able to capture the underlying distribution of data perfectly. On the other hand, the traditional algorithms do not suffer from this issue because they use well-designed handcrafted features. However, developing a good handcrafted feature extractor which is able to handle the diversity of data in real-world scenario is not easy. Deep learning thus is dominant in such scenarios.

- Hardware: deep learning relies on high-end machines while traditional algorithms are able to work on low-end machines.

- Feature selection: deep learning model automatically learns low-level to high-level features from input data. Moreover, deep learning model is capable of extracting features from any data. This is a major advance of deep learning comparing to traditional algorithms which needs to develop new feature extractor depending on every task.

- Training and Testing time: deep learning and traditional algorithms have opposite behavior in training and testing time. More precisely, deep learning requires long time in training while taking (nearly) real-time in testing. On the other hand, traditional algorithms usually take less time for training and long time for testing.

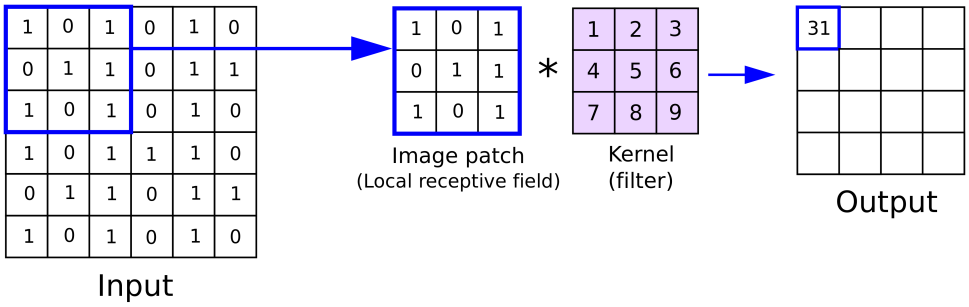
- Problem solving approach: deep learning model tends to solve problem in an end-to-end manner. In contrast, the traditional algorithms advocate to break a problem into sub-problems, then solve sub-problems separately and finally combines them to obtain final result.

A deep learning model often comprises of an input layer, multiple hidden layers, and an output layer. The input layer receives input variables with various forms such as image, text, sound signal. Next, the hidden layers model and process non-linear relationships of input variables. Finally, the output layer predicts/produces output variable corresponding to the problem. In general, a deep learning model learns a mapping from input variables to output variable. To this end, we use training dataset to iteratively optimize the weights of the model. This optimization process not only requires to explore the underlying distribution of training dataset but also needs to ensure the capability of the trained model in processing new data (in testing phase). Therefore, optimizing a deep learning model is nontrivial because the structure of training dataset is complex and non-linear. The widely used optimization algorithms in deep learning are Stochastic gradient descent (SGD) [66] and Adaptive Moment Estimation (Adam) [67].

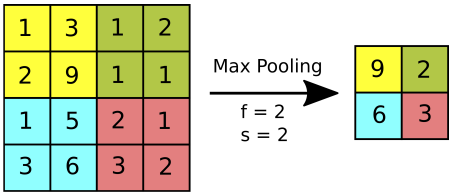
In the context of computer vision, deep learning is applied to many tasks (Fig. 1.4, Chapter 1). As we mentioned above, current deep learning models in computer vision are able to handle many tasks such as object detection [32], object recognition [29–31], semantic segmentation [33]. However, deep learning still struggle to solve image synthesis.

2.1.2 Convolutional neural network

In computer vision, a deep learning model often consists of multiple hidden layers which learns the underlying distribution of input images. In an early stage of deep



(a) Convolution operator.



(b) Max pooling operator.

Figure 2.1: Examples of convolution operator and max pooling operator (image credits: [1]).

learning, these hidden layers were built upon fully connected layers. However, the fully connected layers do not work well when the size of image quickly increases because of the explosion of the number of parameters in the network. To overcome this problem, convolutional neural network (CNN) [68] has been proposed. By using filters which stride over image, CNN is capable of successfully capturing the spatial and the temporal information within image. Consequently, the feature representations by CNN are better than those by fully connected layers, resulting in better performance. CNN, therefore, becomes major architecture in computer vision. Nowadays, most of deep learning model in computer vision employ CNN. This dissertation also uses CNN in designing network architecture. We here briefly review basic concepts in CNN.

Convolution layer. The core part of convolution layer is kernel/filter. It is a small learnable matrix with the size of $M \times N$ which traverses entire image (or output from previous layer) to extract feature representations. Each value of feature representations is computed by performing convolution operator between the kernel and the image patch (local receptive field) corresponding to traversing step. We remark that the size

of local receptive field is the same with that of kernel. Fig. 2.1a illustrates an example of convolution operator. At each step, the kernel moves to right with a pre-set *stride* to reach the width of image. It then hops down to the beginning of image with the same *stride* and repeats the process until completion of traversing.

Depending on the size of kernel, the kernel may lose information from several (of the last) columns. This is because the kernel cannot stride outside the size of image. To avoid such scenario, padding the border of image may be used. More precisely, every direction of the image is added *padding* extra layer(s) of zero.

Using the kernel with the size of $M \times N$, the stride step *stride*, and the padding size *padding*, the size of output (feature representation) is computed as follows:

$$\left\lfloor H_{out} = \frac{H_{in} + 2 \times padding - M}{stride} + 1 \right\rfloor \tag{2.1}$$

$$\left\lfloor W_{out} = \frac{W_{in} + 2 \times padding - N}{stride} + 1 \right\rfloor \tag{2.2}$$

where H_{in}, W_{in} are height and width of input while H_{out}, W_{out} represent those of output, respectively.

Activation layer. The activation layer is used after convolution layer. This will apply element wise activation function to the output of convolution layer. Several common activation layers are ReLU [69], Sigmoid, Tanh, and LeakyReLU [70].

Pooling layer. This layer is used to reduce the size of feature representations before feeding those feature representations into the convolution layer. This layer is helpful to decrease computational cost and to prevent from overfitting. The most widely used pooling layer is max-pooling (Fig. 2.1b). We remark that using pooling layer is optional.

Combining aforementioned layers in an appropriate order creates a complete convolutional neural network which has capability of extracting desired feature representations. There is no common way to design such CNN. In general, we can use a combination of convolution layer and activation layer following by a pooling layer. In addition, a normalization layer (Batch normalization [71], Instance normalization [72]) is used to accelerate the speed, performance, and stability of CNN. Fig. 2.2 show the architecture of VGG-16 [30] which we use as our backbone network in this dissertation (see Chapters 3, 4, and 5).

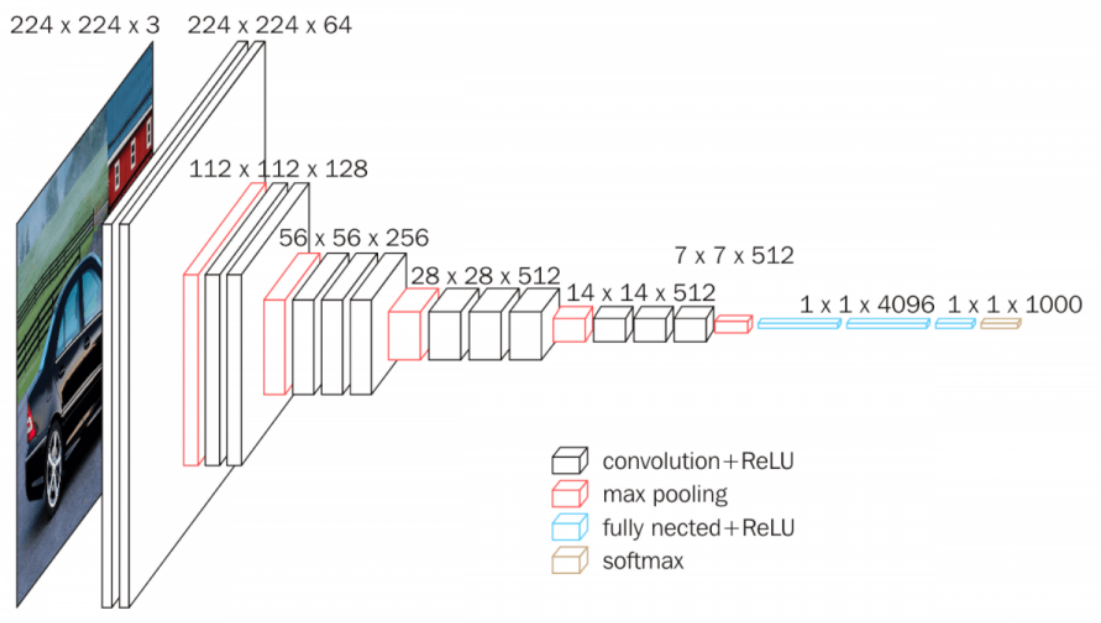


Figure 2.2: Architecture of VGG-16 (image credits: [2]).

2.1.3 Generative models

Generally, deep learning models fall into two categories: discriminative models and generative models. The discriminative models learn the boundaries between data instances so that they are able to discriminate between different kinds of data instances. The generative models, on the other hand, capture the underlying distribution of data so that they are capable of generating new data instance. More precisely, given a set of data instances X and a set of labels Y:

- The discriminative models capture the conditional probability $p(X|Y)$.
- The generative models capture the joint probability $p(X, Y)$ or $p(X)$ (if there is no label)

Since image synthesis attempts to produce a novel image from given inputs, it therefore belongs to generative models family. We then briefly review generative models in this section. A generative model is generally optimized by using Maximum Likelihood Estimation where the model seeks a set of parameters θ that maximize the objective function:

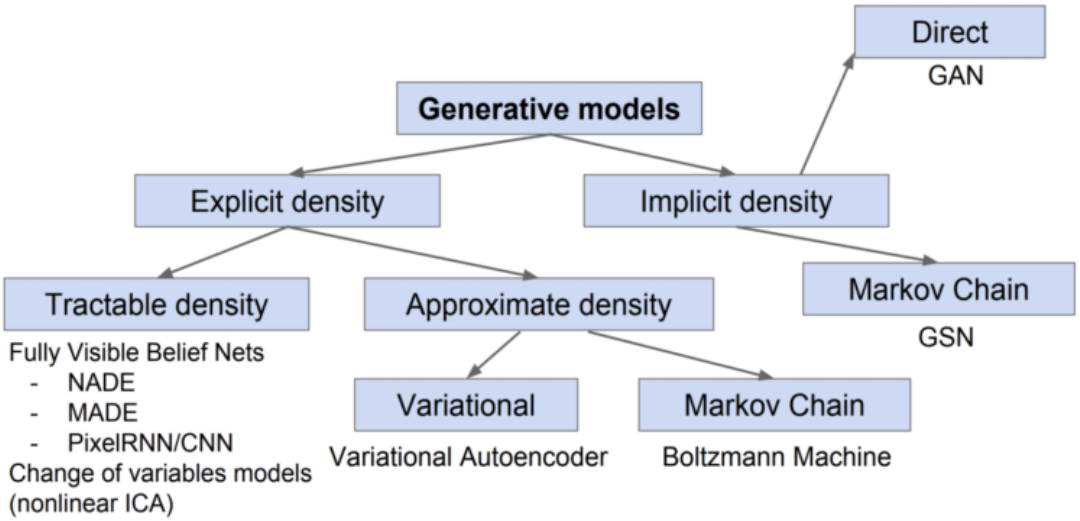


Figure 2.3: Taxonomy of Generative Models (image credit: [3]).

$$\theta = \max_{\theta} p(X|\theta) \tag{2.3}$$

where X is a set of data instances. We assume that X contains N data instances, the above equation is equivalent to:

$$\theta = \max_{\theta} p(x_1, ..., x_N|\theta) \tag{2.4}$$

Since the probability of all data instance $x_1, ...x_N$ happen at the same time is joint probability, optimizing Eq. 2.4 becomes maximum likelihood:

$$\theta = \max_{\theta} p(x_1, ..., x_N|\theta) \tag{2.5}$$

We can approximate the term $p(x_1, ..., x_N|\theta)$ in Eq. 2.5 as follows:

$$p(x_1, ..., x_N|\theta) \approx \prod_{n=1}^{N} p(x_n|\theta) \tag{2.6}$$

Using the approximation of $p(x_1, ..., x_N|\theta)$, Eq. 2.5 is rewrote:

$$\theta = \max_{\theta} \prod_{n=1}^{N} p(x_n|\theta) \tag{2.7}$$

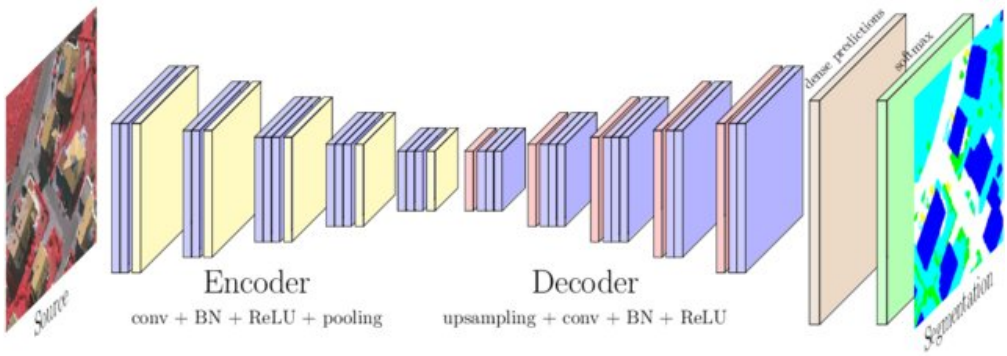


Figure 2.4: Example of encoder–decoder architecture (image credits: [4]).

In practice, optimizing Eq. 2.7, however, is difficult since it is a production of probabilities. We thus transform it to a simpler form which is a summation of log functions:

$$\theta = \max_{\theta} \sum_{n=1}^{N} \log(p(x_n|\theta)) \tag{2.8}$$

Eq. 2.8 is a basic objective function of generative models. Since generative models are diverse (Fig. 2.3) such as PixelRNN [73], variational Autoencoder [74] and generative adversarial network (GAN) [64], Eq. 2.8 will be different depending on the type of model. For example, the objective function of GAN is a minimax game between generator and discriminator $\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{\text{data}}}[\log D(x)] + \mathbb{E}_{z\sim p_z}[\log(1 - D(G(z))]$. We remark that our work heavily employs GAN. We thus review GAN in more details later.

2.2 Encoder–Decoder Approach

2.2.1 Overview of encoder and decoder

Encoder–decoder architecture [75] aims to transfer inputs into outputs without much data distortion which are popularly used in image processing, machine translation. The encoder–decoder architecture consists of two neural networks, namely encoder part and decoder part. The encoder part learns to map the input into feature representations. The decoder part, on the other hand, takes the feature representations as its input and processes to produce results. Note that the results obtained by the decoder are variety

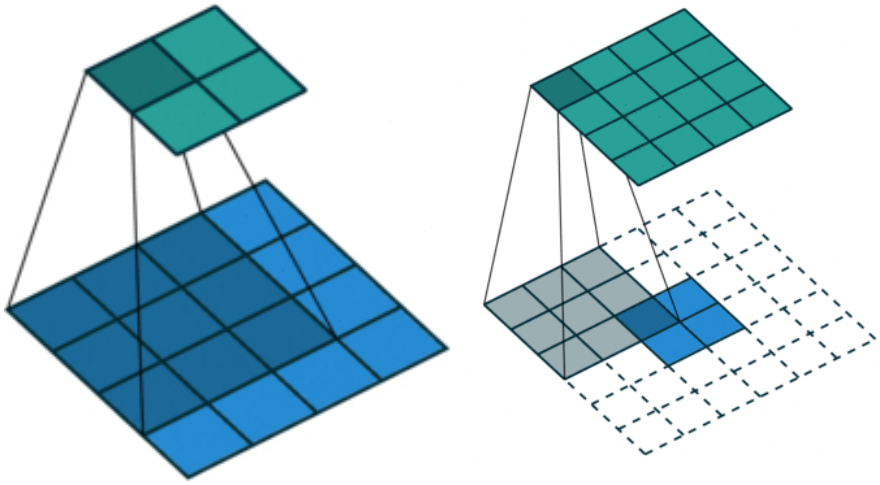


Figure 2.5: Examples of convolution layer used in Encoder (left) and Decoder (right). Blue maps are inputs, and cyan maps are outputs (image credits: [5]).

depending on the task. For example, the encoder–decoder network can be used for semantic segmentation [4], style transfer [7, 9], text-to-image synthesis [76].

Theoretically, the encoder and the decoder can use different network architecture. For example, both encoder and decoder can be designed using CNN [68] or encoder is RNN [75] while decoder is CNN [68]. In our work, however, we employ CNN [68] for both the encoder and the decoder.

Fig. 2.4 illustrates a typical encoder–decoder network which uses CNN [68] in designing both the encoder and the decoder. Though both the encoder and the decoder employ convolution layer, they use different sets of kernel size, *stride*, and *padding* (Fig. 2.5) (see Eqs. 2.1, 2.2). As a result, the encoder reduces the size of feature representations along with its depth. On the other hand, the decoder gradually increases the size of feature representations. The encoder and the decoder are jointly trained to optimize the loss function. To this end, we iteratively feed the training data to the network and compare the obtained results with the initial data using reconstruction loss. We then update the parameters of the network by backpropagating the error through the network architecture (i.e., both the encoder and the decoder).

2.2.2 Image synthesis using encoder–decoder

Though the encoder–decoder network [75] is able to reconstruct image, but it still lacks of ability of generating a novel image. This is because the encoder–decoder network encodes each data instance independently, leading feature space is not regular enough. Obviously, if the feature space is regular enough, any random feature representation can be decoded to create a new image. To overcome this issue, variational auto-encoder [74] has been proposed. Basically, the architecture of both the networks (i.e., encoder–decoder [75] and variational auto-encoder [74]) are the same. However, instead of encoding each data instance individually, the encoder in variational auto-encoder encodes all data instances as distribution over the feature space. Next, the encoder samples feature representations from the encoded distribution. Then, the sampled feature representations are feed into the decoder to generate a new image. It is worth to note that the learned distribution from the encoder is forced to be close to the standard distribution by regularizing the KL-divergence between the two distributions. This is because the standard distribution is continuity (i.e., two close feature representations in feature space should not give two completely different images) and completeness (i.e., any feature representation in feature space should give reasonable image).

Following the success of variational auto-encoders [74], many work [76–79] have been proposed for image synthesis. Among these models, Mansimov et al. [76] introduced text-to-image model that generates images from natural language descriptions. The model in [76] inherits DRAW mechanism [79] to iteratively draw patches on a canvas while attending to relevant words in description. Although these models [74, 76–79] showed better performances than methods employing hand-crafted features, they were unable to achieve highly realistic images.

Another major approach to improve encoder–decoder network [75] is to guide the network with "meaningful" feature representations. In this approach, instead of minimizing the reconstruction loss, we minimize the difference of meaningful feature representations. More precisely, this approach employs a pre-trained CNN network to extract the meaningful feature representations from input data and generated images depending on the task. Then, the difference of those extracted features is used to guide the network to update its parameters. The most famous work of this

approach is proposed by Gatys et al. [9]. They found that the pre-train VGG-16 [30] on ImageNet [80] is capable of capturing *content* and *style* along with its depth. The *content* is the feature representations at higher layers in VGG-16. The *style*, on the other hand, is Gram matrix [81] (i.e., a matrix of inner products) of feature representations. They then apply these characteristics of feature representations extracted from VGG-16 in rendering image contents in different styles. To this end, they [9] start from a noise image sampled from standard distribution (we may regard this noise image as output of encoder) and iteratively update the image to produce an image satisfying the semantic distribution of the content image and appearance statistics of the style. During the iteration, the weighted sum of style loss and content loss is minimized:

$$\mathcal{L}(\hat{y}, y_c, y_s) = \alpha \mathcal{L}_{\text{content}}(\hat{y}, y_c) + \beta \mathcal{L}_{\text{style}}(\hat{y}, y_s), \tag{2.9}$$

where y_c, y_s, and $\hat{y}$ denote the content image, the style, and the stylized image, respectively. α and β are the weighting factors for content and style reconstruction. $\mathcal{L}_{\text{content}}$ and $\mathcal{L}_{\text{style}}$ are content and style loss respective. The content loss is defined as follows:

$$\mathcal{L}_{\text{content}}(\hat{y}, y_c) = \frac{1}{M} \sum_{k \in M} \frac{1}{C_k \times H_k \times W_k} \|\Phi_k(\hat{y}) - \Phi_k(y_c)\|_2, \tag{2.10}$$

where $\Phi_k(\cdot)$ denotes the normalized feature map at the k-th layer, which has $C_k \times H_k \times W_k$ elements.

The style loss is computed at N layers as follows:

$$\mathcal{L}_{\text{style}}(\hat{y}, y_s) = \frac{1}{N} \sum_{k \in N} \|G(\Phi_k(\hat{y})) - G(\Phi_k(y_s))\|_F, \tag{2.11}$$

where $\|\cdot\|_F$ denotes the Frobenius norm [81]. $G(\Phi_k(\cdot))$ is the Gram matrix [81] of the normalized feature map at the k-th layer. The Gram matrix $G_{C_k \times C_k}$ has elements $G_{ij} = \langle v_i, v_j \rangle$ where v_i, v_j are features at the i-th and the j-th channels respectively of the feature map $\Phi_k(\cdot)$.

The work by Gatys et al. [9] showed remarkable results and opened up a new trend in style transfer. As follow-up work of [9], [42] proposed a structure preservation method using Matting Laplacian for photo-realistic style transfer. [43] utilized the

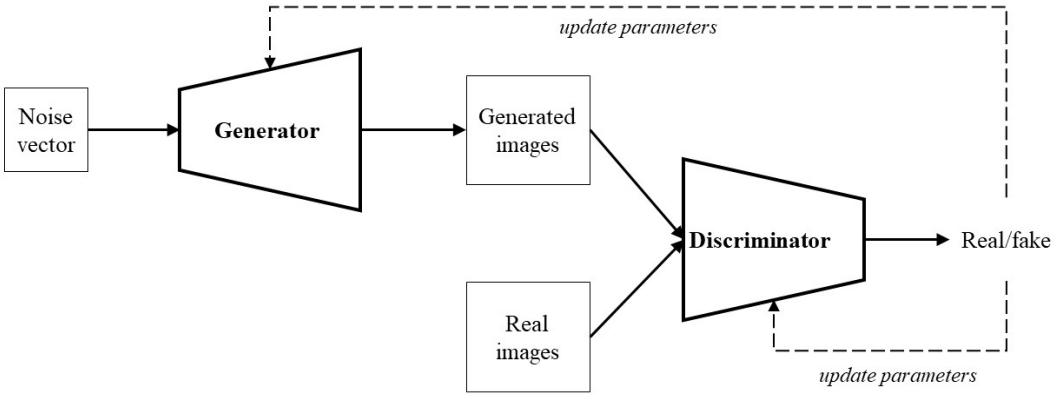


Figure 2.6: Overview of Generative Adversarial Network.

screened Poisson equation to make a stylized image more photo-realistic. [82] proposed a Laplacian loss that computes the Euclidean distance between the Laplacian filters responding to a content image and a stylized image in order to keep a fine structure of the content image.

Johnson et al. [7] and Ulyanove et al. [83], on the other hand, proposed a feed-forward CNN and used the perceptual loss function for gradient-based optimization. The perceptual loss used there is similar to the content and the style losses used in [9]. Their models have only to pass the content image to a single forward network to produce a stylized image, which is fast. Methods related to [7] were proposed [8, 10–12, 44, 84] where most of them improved network architecture to extract *content* and *style* features, resulting in the explosion of network parameters.

2.3 GAN-based Approach

2.3.1 Overview of Generative adversarial network

Generative Adversarial Network (GAN) is first proposed by Goodfellow et al. [64]. Since GAN aims to learn the underlying distribution of data, it thus also falls into the family of generative models (see Fig. 2.3). Different from other generative models, GAN composes of two independently different network, namely Generator G and Discriminator D (Fig. 2.6). The generator receives a noise vector z sampled from a normal distribution p_z and outputs a novel image $G(z)$. The discriminator, on the other

hand, distinguishes whether its input x is real (i.e., image in training data p_{data}) or fake (i.e., image $G(z)$ obtained by generator). The two models are simultaneously trained where the network first updates the parameters of G with fixing the parameters of D and then updates the parameters of D with fixing the parameters of G. The training process is summarized as follows:

- First, with fixing the parameters of D, the generator G generates an fake image $G(z)$ from a sampled noise z. Then, $G(z)$ is fed into D to obtain classification error which is used to update the parameters of G through backpropagation. Note that, $G(z)$ is expected to be classified as real label.
- Next, with fixing the parameters of G, the discriminator D takes a real image x and a fake image $G(z)$ to update its parameters.
- The above two steps are repeated until the network is optimized.

In formally, training GAN is a minimax game between G and D:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{\text{data}}}[\log D(x)] + \mathbb{E}_{z\sim p_z}[\log(1 - D(G(z))] \tag{2.12}$$

where p_{data} denotes all training data, p_z denotes sample distribution and $\mathbb{E}$ means expectation over data.

The idea of GAN is simple but effective. However, training a GAN model is hard with some major problems:

- Non-convergence: the training of GAN is a minimax game between the generator G and the discriminator D (Eq. 2.15). In other word, if one wins the other loses. However, depending on the learning rate and the initial parameters of each model, the convergence may be never happened. Intuitively, either the generator or the discriminator always countermeasures against each other. This leads the network is hard to converge.
- Mode collapse: the real data distributions are diverse and multimodal. For example, the numeric system has 10 modes from digit "0" to digit "9". However, if a GAN model is able to generate digit "1" only, it suffers from a so-called "mode

collapse". This issue happens when the generator (accidentally) finds a good sample that always fools the discriminator. As a result, the generator tends to generate that good sample independently to sampled vector. The GAN network, therefore, produces images less diverse.

- Diminished gradient: In generally, training a classification (i.e., training the discriminator) is easier than training the generator. When the discriminator becomes very good at its task, the gradient of the generator is saturated, leading that the parameters of the generator are not updated.

- Training GAN is highly sensitive to hyperparamter selections.

To stabilize the training of GAN, many work [85–88] has been proposed. Since this dissertation does not aim at improving GAN, we thus refer previous work [85–88] for more details.

GAN evaluation. In order to evaluate the performance of GAN, we need to measure two criteria: (i) the overall quality of generated images, and (ii) the diversity of generated images. To this end, two metrics have been proposed, namely *Inception score* [89] and *Fréchet inception distance* [90].

Inception score (*IS*). The idea behind of *IS* comes from the *entropy* value of a random variable. Intuitively, when an image is highly predictable, its entropy is low. In contrast, entropy is high if an image is highly unpredictable. Given a generated image $\hat{x}$, we expect that $\hat{x}$ is easily classified. Therefore, the conditional probability $p(y|\hat{x})$ (where y is class label) should be low. The $p(y|\hat{x})$ evaluates the quality of generated image (criterion (i)). On the other hand, if the generated images are diverse, the distribution of each class label y should be uniform (criterion (ii)). To this end, we compute the empirical marginal-class distribution for each class label y: $\hat{p}(y)$. To combine two criteria, we compute the KL-divergence between $p(y|\hat{x})$ and $\hat{p}(y)$. We remark that higher inception score is better. In practice, inception score is computed through the output of the Inception-v3 network [91]:

$$IS(G) \approx \exp(\frac{1}{N}\sum_{i=1}^{N} D_{\mathrm{KL}}(p(y|\hat{x}^{(i)}||\hat{p}(y)))) \tag{2.13}$$

where $\hat{x}$ is a synthesized image by the generator G, N is the number of generated im-

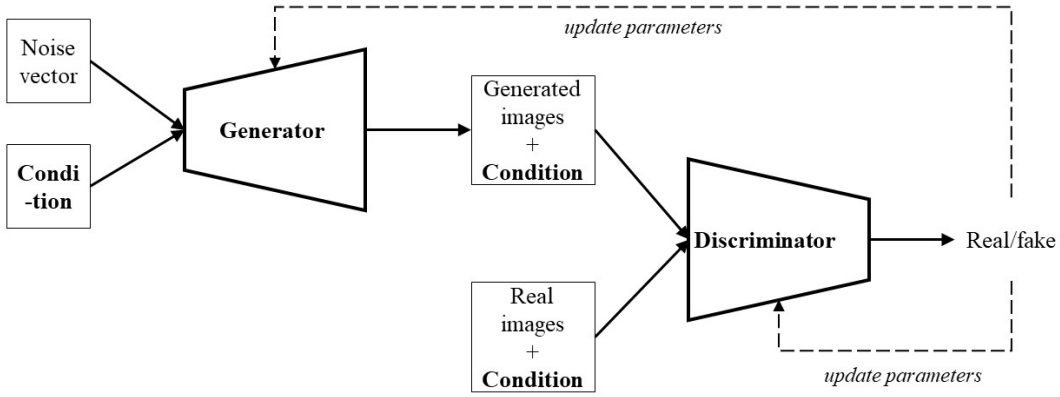


Figure 2.7: Overview of Conditional GAN.

ages, D_{KL} is the Kullback–Leibler divergence, y indicates an instance of all classes given in the dataset, $p(y|\hat{x})$ is the conditional class distribution, and $\hat{p}(y) = \frac{1}{N}\sum_{i=1}^{N} p(y|\hat{x}^{(i)})$ is the empirical marginal-class distribution.

Fréchet inception distance (*FID*). This metric evaluate how close between generated images and real images. To this end, *FID* measures the similarity of real and generated data using the Fréchet distance [92] between their activation distributions extracted from the *pool*3 layer of the Inception-v3 network [91]:

$$FID = \left\|\mu_{\text{real}} - \mu_{\text{gen}}\right\|^2 + \text{tr}(\Sigma_{\text{real}} + \Sigma_{\text{gen}} - 2(\Sigma_{\text{real}}\Sigma_{\text{gen}})^{1/2}) \tag{2.14}$$

where μ_{real}, μ_{gen} Σ_{real}, Σ_{gen} are means and covariance matrices of the activation distributions of real and generated data, and $\text{tr}(\cdot)$ is the trace.

2.3.2 Conditional GAN and DCGAN

Conditional GAN [87] and Deep convolutional GAN (DCGAN) [6] are two of the popular and successful network design of GAN. Moreover, they are fundamental ideas of GAN-based image synthesis model.

Conditional GAN. As we mentioned above, training a GAN model is nontrivial. This is because GAN automatically learns from the noise vector resulting in uncontrollable results. To better control the results of GAN, we can add an extra information to guide the learning process. Conditional GAN [87] comes with this idea. More precisely, conditional GAN [87] adds an additional condition to both the generator and the

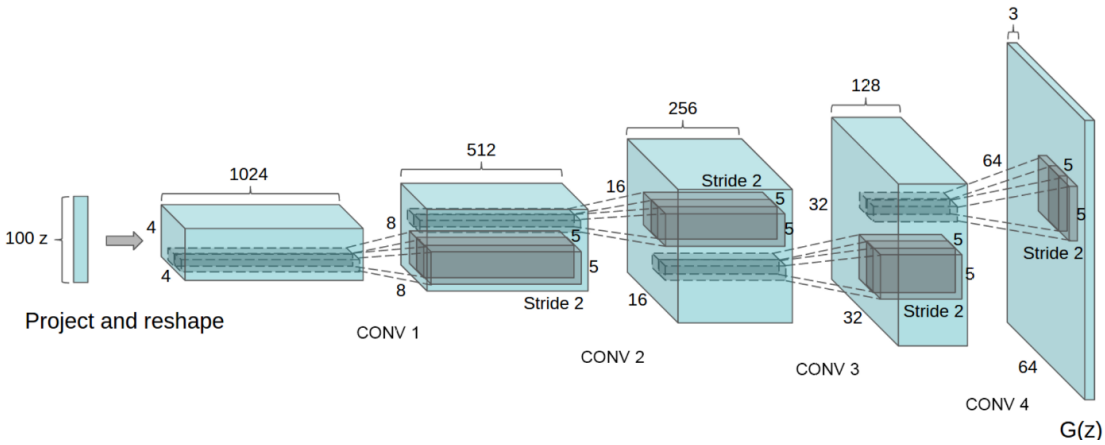


Figure 2.8: Generator in DCGAN (image credits: [6]).

discriminator to not only to generate plausible images but also to meet the conditions (Fig. 2.7). By adding class label, the loss function in GAN (Eq. 2.15) becomes:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{\text{data}}}[\log D(x|y)] + \mathbb{E}_{z\sim p_z}[\log(1 - D(G(z)|y)] \tag{2.15}$$

where y is given condition.

Interestingly, the additional condition in conditional GAN is variety such as class label, segmentation map, text, or image. Nowadays, the idea of conditional GAN is widely used in GAN-based models.

Deep convolutional GAN (DCGAN). This architecture is proposed by Radford et al. [6]. DCGAN is mainly built upon convolution layers without max-pooling layer or fully connected layer. Fig. 2.8 illustrates the generator in DCGAN. The discriminator in DCGAN (mostly) mirrors to the generator. In summary, some important points in DCGAN are below:

- Replace max-pooling layer by using the *stride* in convolution layer (Eqs. 2.1, 2.2).
- Remove all full-connected layers.
- Use transposed convolution layer to upsample feature maps.
- Use Batch Normalization [71] after convolution layer. Note that the output layer

of generator and the input layer of discriminator are not followed by Batch Normalization.

- In generator, use ReLU [69] except for the last convolution layer uses *tanh*.
- In discriminator, use LeakyReLU [70].

2.3.3 Image synthesis using GAN

As we mentioned above, combination of conditional GAN and DCGAN is a basic architecture for GAN-based image synthesis. Some work conditions GANs on the attribute label [38, 86] or images [37, 93–96] for domain transfer [37, 93], photo editing [94], image super-resolution [95].

Several GAN-based models for style transfer are also proposed [96–99]. These models also optimize the network with a large number of content images during the training step. Though GAN-based models bring a promising approach to improve the quality of stylized images, their results, at this time, still are less impressive [100]. Furthermore, as in common with other GAN-based approaches, their training processes are also unstable.

Dong et al. [13] proposed image manipulation with text where they condition text and source image on GAN. Though it generates images that match the semantic meaning of the input text description while maintaining other parts of a source image, it does not preserve background precisely because the discriminator is used only for foreground; synthesized images are less realistic images. To enhance the matching between text description and the foreground, Nam et al. [56] proposed a text-adaptive discriminator. Their discriminator splits a text description into word-level so that the discriminator is able to match each word to each visual attribute more precisely. Li et al. [57], on the other hand, attempted to generate attributes matching text description and to reconstruct text-irrelevant contents of the source image at the same time. They thus proposed text-image affine combination module (ACM) and detail correction module (DCM). The ACM seeks the text-relevant regions in source image to generate new attributes matching given text descriptions while the DCM rectifies text-irrelevant regions and completes missing contents. Addressing the background problem in image synthesis, Yang et al. [62] proposed to decompose the image synthesis into two phases

using foreground and background generators. However, using only one discriminator is not able to judge foreground and background well. Nonetheless, their method illustrated the capability of GAN in separating feature space.

Reed et al. [45] proposed an end-to-end GAN using the text condition. They employed a pre-trained text encoder [47] to extract text features from an input text, and then combined text features with a vector representing random noise to produce the input of the generator. They also employed the combination of text features and image features in the discriminator to discriminate real images and generated images. Their proposed model [45] became the baseline of the GAN framework for generating images from text descriptions. As an extension, a model conditioned on texts and location information was proposed [101]. Models with two stages of GAN, Stack-GAN [18] (and Stack-GAN++ [46]), were also proposed, showing successfully generated higher resolution images (256×256), compared to [45] (64×64). Xu et al. [19] proposed an attention mechanism to fine-grained text-to-image generation. Their model generated image details by paying attention to the relevant words in text description. These models [18, 19, 45, 46, 62] condition on GAN only texts or a pair of texts and location information [101]. However, directly generating images from text description is difficult because of the large gap between text and image domains. To overcome the limitation of GANs conditioned on text descriptions, a two-step approach was proposed where inference of the scene layout as an intermediate representation between text and image is followed by using the layout to generate images [17, 20, 22, 48]. Since the gap between the intermediate representation and image is smaller than that of text and image, this approach generates more realistic images.

2.4 Our Approach

As we mentioned above, a deep neural network is capable of capturing from low-level to high-level feature representations. Moreover, input data of a network is diverse and multimodal, leading the feature representations have different meaning depending on their role. Unfortunately, either the encoder–decoder models or the GAN-based models currently employ all the feature representations together, resulting in lack of ability of controlling generated images. This dissertation, on the other hand, aims to propose a novel methodology in image synthesis which employs disentangled feature

Table 2.1: Feature representations used in state-of-the-arts and our research.

Task	**Style transfer**		**Image manipulation with text**		**Text-to-image synthesis**		
Feature space	*Content*	*Style*	*Foreground*	*Background*	*Object*	*All relations*	*Individual relation*
Gatys+ [9]	–	✓	–	–	–	–	–
Sheng+ [10]	–	✓	–	–	–	–	–
Li+ [12]	–	✓	–	–	–	–	–
Johnson+ [7]	✓	–	–	–	–	–	–
Huang+ [8]	✓	–	–	–	–	–	–
Chen+ [11]	✓	–	–	–	–	–	–
Dong+ [13]	–	–	✓	–	–	–	–
Yang+ [62]	–	–	✓	–	–	–	–
Nam+ [56]	–	–	✓	–	–	–	–
Li+ [57]	–	–	✓	–	–	–	–
Zhang+ [18]	–	–	–	–	✓	–	–
Xu+ [19]	–	–	–	–	✓	–	–
Johnson+ [17]	–	–	–	–	–	✓	–
Hong+ [22]	–	–	–	–	–	✓	–
Li+ [48]	–	–	–	–	–	✓	–
Ashual+ [20]	–	–	–	–	–	✓	–
Our research	✓	✓	✓	✓	–	✓	✓

representations. More precisely, we choose feature representations depending on their role in the task and then combine these disentangled feature representations to create image. Table 2.1 summarizes feature representations used in state-of-the-arts and our research.

In order to investigate our proposed methodology, we take both basic approaches in image synthesis (i.e., encoder–decoder and GAN-based models) in our research.

Rendering content images in different styles (style transfer). In this task, we follow the encoder–decoder architecture because it is more promising in style transfer task. Different to other style transfer models, we extract *content* and *style* feature representations from inputs and then combine these disentangled feature representations to stylize images. Fig. 2.9 intuitively illustrates our idea. More precisely, we propose a two-stream network where a deep stream extracts content feature representations while a shallow stream does style feature representations. Then, the network combines these disentangled representations in an adaptive manner to produce stylized images. By using disentangled content and style representations, our network is able to control

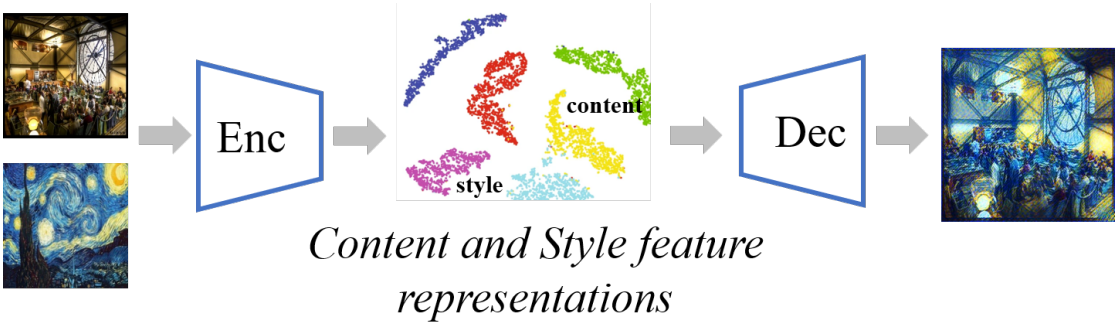


Figure 2.9: Overview of our idea for rendering content images in different style. More details can be found in Chapter 3.

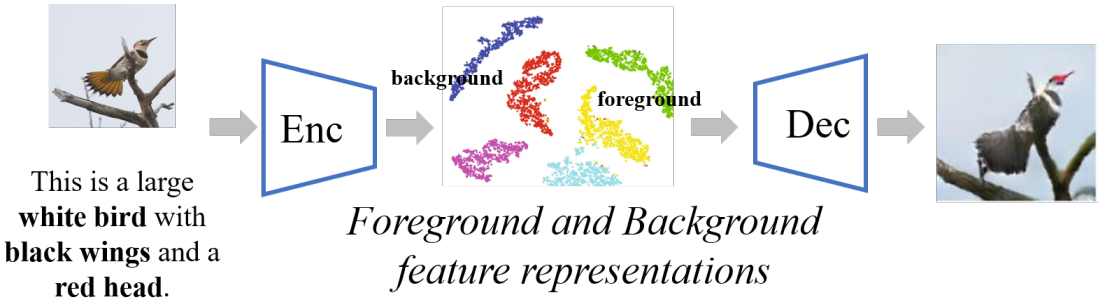


Figure 2.10: Overview of our idea for image manipulation with text. More details can be found in Chapter 4.

the contribution of content and style images in the final results. This makes our work distinct to previous work. We refer Chapter 3 for more details.

Image manipulation with text. This task conditions on two (very) different conditions, namely image and text. We thus take GAN-based model to deal with this task because GAN-based model has been known better in controlling such different conditions. We observe that the foreground and background feature representations in an image are obtained at different layers in the network. We thus fully take into account the disentangled foreground and background feature representations in this task. Overview of our idea is shown in Fig. 2.10. In particular, we propose a GAN-based model having two discriminators, one for foreground and one for background. The two discriminators separately judge foreground and background of the synthesized image to meet an input text description and a source image. The use of disentangled foreground and background representations brings us two benefits. First, the matching

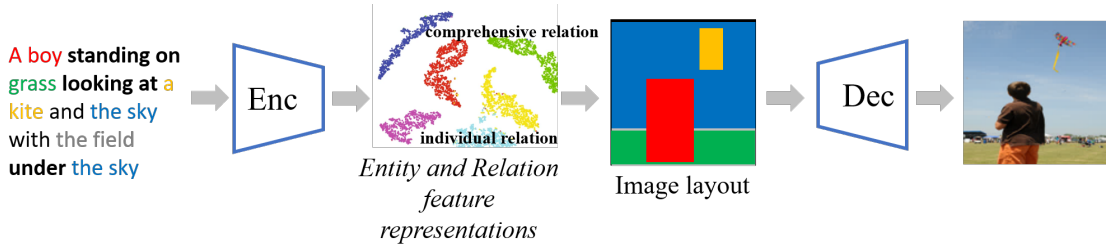


Figure 2.11: Overview of our idea for text-to-image synthesis. More details can be found in Chapter 5.

between foreground and text description is more precise because the network does not distract by irrelevant information (i.e., background). Second, the network is able to retain background information successfully because it brings background representations in source image to generated image directly. Chapter 4 discusses our proposed method for this task in more details.

Text-to-image synthesis. As we discussed above, GAN-based models obtain impressive results in text-to-image synthesis. However, they still struggle to generate realistic images that are consistent with text description. We thus apply our methodology on GAN-based text-to-image synthesis model to tackle the current issues. Due to large gap between text description and image, we first predict an intermediate layer (i.e., image layout) from text and then convert the layout to the image. The image layout reflects visual relations among objects in the text description. When localizing each object in the layout, we necessarily keep not only the whole structure (i.e., all relations together) but also local structure (i.e., each relation individually). To ensure such conditions, we therefore employ two kinds of relations in predicting layout: (i) comprehensive relation, and (ii) individual relation (Fig. 2.11). The two kinds of relations enable us to first initialize the whole structure of image and then to adjust objects' locations to preserve local structure. Consequently, our predicted layout is consistent to the text description which allows better generating images. Chapter 5 presents our proposed method for this task.

3

Learning Content and Style Feature Representations Adaptively for Style Transfer

3.1 Introduction

How New York looks like in "The Starry Night" by Vincent van Gogh is an interesting question and, at the same time, difficult to answer. In practice, re-painting a famous fine-art style takes much time and requires well-trained artists. Answering this question can be stated as the problem of rendering semantic content of one image to different styles, and it is called style transfer.

[9] showed that the image representation derived from a Convolutional Neural Network (CNN) can be used to represent the semantic content of an image and the style, which opened up a new trend of CNN-based style transfer. CNN-based approaches in style transfer fall into two categories [100]: Image-Optimisation-Based Online

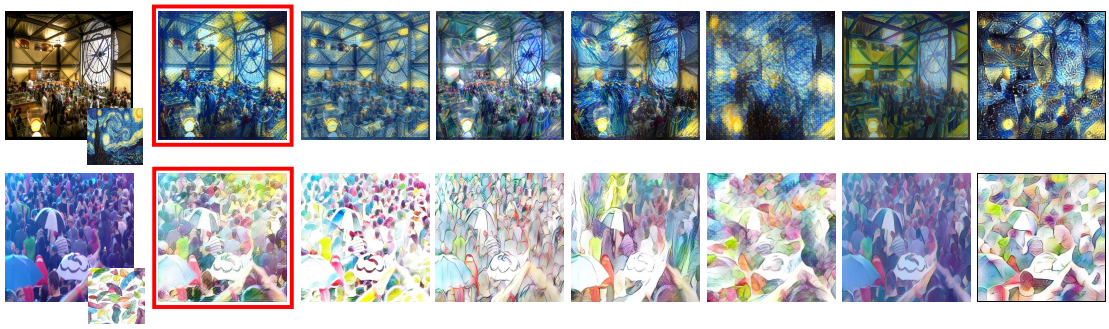


Figure 3.1: Example of stylized results. Left-most column: content image (large) and style image (small). From left to right: the stylized image by our method, Johnson+ [7], Huang+ [8], and Gatys+ [9], Sheng+ [10], Chen+ [11], and Li+ [12]. Our results surrounded with red rectangles are more balanced in content and style than the others.

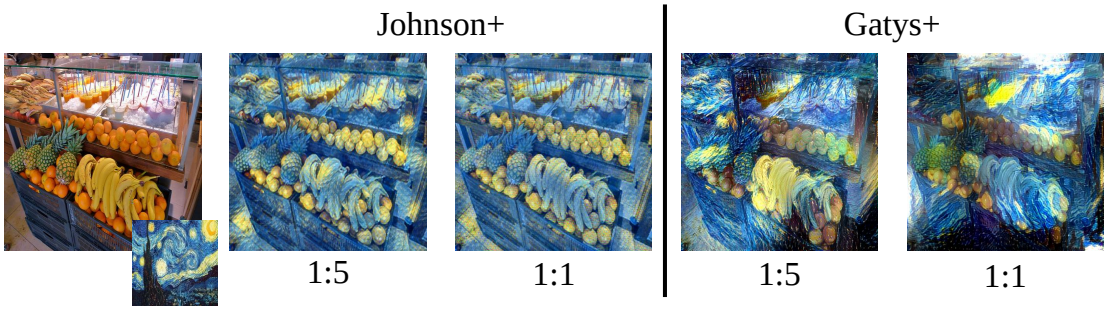


Figure 3.2: Example of stylized results obtained by Johnson+ [7] and Gatys+ [9] by changing the ratio of content and style from 1:5 to 1:1. Left-most column: content image (large) and style image (small). In each block, from left to right: the stylized image with various ratio of content and style.

Neural Methods (IOB-NST) and Model-Optimisation-Based Offline Neural Methods (MOB-NST). The key idea of IOB-NST is to synthesis a stylized image by directly updating pixels in the image iteratively through the back-propagation. The IOB-NST such as [9, 42, 43] starts with a noise image and iteratively updates the image by changing the distribution of noise along with the statistics of content and style until the defined loss function is minimized. MOB-NST such as [7, 8, 10–12, 44, 98, 99, 102], on the other hand, first optimizes a generative model through iterations, and then renders the stylized image using a forward pass. In order to optimize the generative model, MOB-NST trains each feed-forward model for each specific style by using the gradient descent over a large dataset. IOB-NST is known to produce better stylized

results in quality than MOB-NST [100], while MOB-NST has more efficiency in speed.

IOB-NST and MOB-NST have the capability of controlling the balance between the content and the style. Namely, they allow to manually change the ratio of content and style. However, changing the ratio do not guarantee that network parameters for stylized images changes as expected, meaning that the contributions of the content and the style in a stylized image are uncontrollable in reality. Fig. 3.2 shows examples obtained by IOB-NST (Gatys+ [9]) and MOB-NST (Johnson+ [7]) with various settings of contributions of the content and the style. We can see although the ratio of content and style is significantly changed, the results do not change much. How to control such contributions is important. To this end, the network need to disentangle content and style feature space. This allows us to explicitly control the ratio of content and style in stylized images. Since the balance between the content and the style in style transfer is required in many applications; for instance, font transfer [103], realistic photo transfer [43, 104], we thus address the balancing content and style in stylized images in this work.

Although existing methods [7–12, 42–44, 97–99, 102] show the capability of rendering image contents in different styles, generated stylized images are not always well balanced in content and style. Such methods take care of either the content or the style, but not both, producing unbalanced stylized images. IOB-NST is good at faithfully rendering the style while it tends to lose the content. MOB-NST, on the other hand, preserves more semantic content than the style. How to keep the balance between the content and the style in style transfer is a crucial issue to improve the quality of stylized images.

Another important issue to address is the computational speed. Although MOB-NST such as [7, 8, 10–12, 44, 97–99, 102] are able to produce stylized images fast, they rely on a strong computational power. Therefore, either IOB-NST or MOB-NST is hard to apply to real-time applications.

We propose an end-to-end two-stream network for balancing the content and style in stylized images where contributions of the content and the style are adaptively taken into account. The encoder part of our network consists of the content stream and the style stream where the streams have different architectures. The two streams are connected by adaptive feature injection and independently trained to learn the semantic content or the style representation. The content features and the style

features are then combined in our proposed adaptive concatenation to ensure the balanced contribution of each stream. As the decoder part of our network, we use the feed-forward model to reduce the rendering time while we spend much time on learning like [7, 8, 10–12, 44]. Unlike other methods that train a new model from the scratch for a yet unknown style, we fine-tune parameters from an existing model, allowing our network not only to accommodate fast training but also to easily adapt new styles. Our experiments demonstrate that our method produces more balanced stylized images in both content and style than the state-of-the-art methods (Fig. 3.1). They also show that our method runs about 22× faster than the state-of-the-art methods. We remark that our proposed model is trained for one style only, but it is easy to be fine-tuned to other styles incrementally with a low cost.

3.2 Related Work

Gatys et al.[9] for the first time proposed a method using CNNs and showed remarkable results. Their method trains CNNs to learn the semantic information from content images and matched it with the distribution of the style. It starts from a randomly distributed noise image and iteratively updates the image to produce an image satisfying the semantic distribution of the content image and appearance statistics of the style. During the iteration, the weighted sum of style loss and content loss is minimized. As follow-up work of [9], [42] proposed a structure preservation method using Matting Laplacian for photo-realistic style transfer. [43] utilized the screened Poisson equation to make a stylized image more photo-realistic. [82] proposed a Laplacian loss that computes the Euclidean distance between the Laplacian filters responding to a content image and a stylized image in order to keep a fine structure of the content image. These approaches fall into the IOB-NST category, and all face with the computational speed problem.

[7] and [83], on the other hand, took MOB-NST, proposing a feed-forward CNN and used the perceptual loss function for gradient-based optimization. The perceptual loss used there is similar to content and style loss in [9]. Their models have only to pass the content image to a single forward network to produce a stylized image, which is fast. Their two models are different only in the network architecture. [7] follows the design of [6] with their modification of using residual blocks and fractionally strided

convolutions while [83] uses a multi-scale in their generator. [44] also utilized the feed-forward network, and they used multiple-generator to improve the quality of results. These methods are fast in generating stylized images, but they are capable of dealing with a single style only.

[84] proposed a multi-style network that introduces shared-computation in many style images where they used instance normalization (IN) [72] for balancing features from the content and from the style. They also proposed an improvement of IN to learn a different set of affine parameters for multi-styles in the batch way. However, their model can train a limited number of styles because the network capability is limited, meaning that the number of styles to handle is limited. [11] proposed a method that overcomes the limitation of the number of styles by using a patch-based method. Their method first extracts a set of patches from the content and style each, and then, for each content patch, the method finds its closest style patch and swaps their activation. In this way, their method transfers an unlimited number of styles; however, the cost for patch extraction and swapping increases the computational time significantly. [12] also proposed a method for multi-style transfer using feature transformations. They first employ pre-trained VGG-19 as their encoder to train an decoder for image reconstruction. Then, with fixing both encoder (VGG-19) and decoder, their model performs the style transfer through whitening and coloring transforms on a given content image and a style image. Though their method successfully solves the multi-style transfer, it still suffers from the computational cost and loses the content due to the feature transformations.

[8] and [10] proposed multi-style transfer models consisting of two CNN streams for content and style. [8] employed the pre-trained VGG-16 to extract content and style features and introduced Adaptive Instance Normalization (AIN) to make the mean and the variance of content features similar to those of style features. [10], on the other hand, proposed AvatarNet which employed the pre-trained VGG-19 to extract the content and style features. These features are matched by using style-swap [11] or AIN [8] before being fed into the decoder. Different from [8], their models have skip-connections from the style encoder to the decoder. [8] and [10], however, used the same architecture for the content CNN and for the style CNN. Having the same CNN architecture for the content and the style causes unavoidable unbalance between the content and the style because semantic levels extracted from the content and

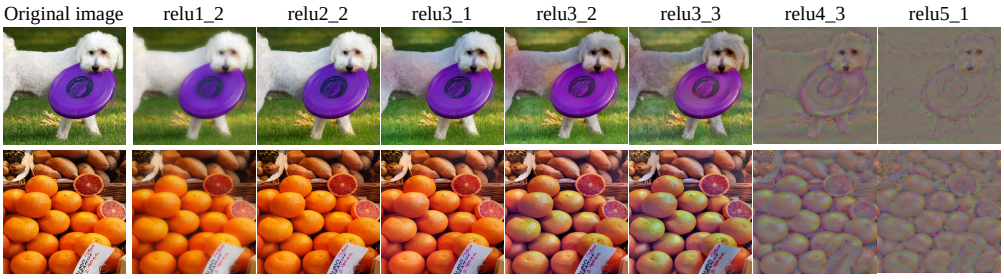


Figure 3.3: Examples of the feature reconstruction for several layers from the VGG-16 pre-trained network.

the style should not be the same in style transfer. Those models require expensive computational cost as well. Furthermore, AIN [8] assumes the standard distribution on pixel values of images, which is not always ensured in styles when normalizing data. In deed, AIN [8] tends to produce a lot of artifacts; especially they are visible on flat surfaces [98]. We remark that the skip-connection in AvatarNet [10] weights the style contribution more, causing unbalance in stylized images.

Different from the methods above, we take into account the contributions of the content and the style through a two-stream feed-forward network to balance the content and the style in stylized images. In particular, our proposed two-stream network is different from [8, 10] in that our network has different depths in layer for the content and the style encoders to extract different semantic levels of the content and the style. In addition, separating content and style enables our method easy to fine-tune to other styles with a cheaper computational cost (re-training time, required numbers of training images) than other models possessing only one encoder [7, 11, 44]. As a result, our method is able to easily deal with multi-styles.

3.3 Semantic Levels of Image Features for Content and Style

Along with the depth, CNN is known to extract different semantic levels of image features in layers. As demonstrated in [7, 9], features in early layers reflect colors, textures, and common patterns of images while those in latter layers preserve content and spatial structure of images. We, therefore, expect that the features in lower

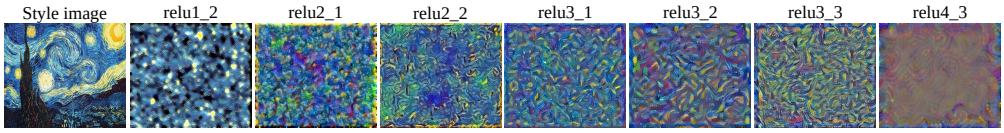


Figure 3.4: Examples of style image reconstruction for several layers from the VGG-16 pre-trained network.

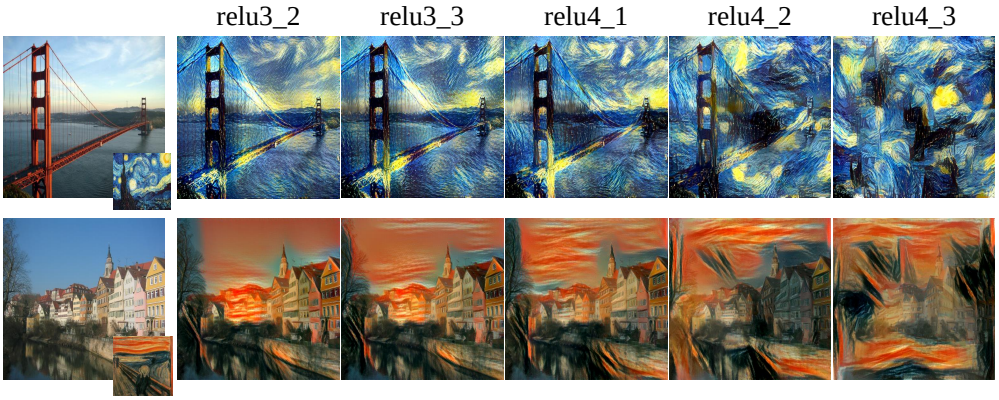


Figure 3.5: Examples of combination of content and style images from *relu*3_2 to *relu*4_3. Left-most column: content image (large) and style image (small), From left to right: the stylized images at different combination levels by Gatys+ [9] where the ratio of contributions of content and style is 1:1.

layers work as style features and those in higher layers do as content features. Using appropriate semantic levels of image features in style transfer is crucial. We thus experimentally exploit the semantic levels of image features in VGG-16 [30] to design suitable numbers of layers in designing our network to extract content and style features. We remark that we refer [7, 9] in which image reconstruction is learned using hidden features in CNN layers.

For the content image reconstruction, we randomly prepare 100 images. We then feed each of the 100 images into the VGG-16 [30] pre-trained on object recognition using ImageNet dataset [80] without any fine-tuning and extract the features at each Rectified Linear Unit (ReLU) [105]. These features are employed to reconstruct original images using inverting technique [106]. Hereafter, we use *reluX_Y* to mention a specific ReLU layer; see the definition of VGG-16 [30] architecture for details. Fig. 3.3 shows some examples of image reconstruction at several layers. We see that at low levels, i.e., from the 2nd layer (*relu*1_2) to the 5th layer (*relu*3_1), the reconstructed

images are similar to the original image, meaning that these layers successfully keep colors, textures, and common patterns of images. At higher levels, i.e., from the 6th layer ($relu3\_2$) to the 10th layer ($relu4\_3$), the reconstructed images preserve the content and spatial structure. At even higher layers that start from the 11th layer ($relu5\_1$), semantic features are gradually learned; the exact shape, on the other hand, is not preserved.

For the style image reconstruction, we use Adam optimization [107] to find an image that minimizes the style reconstruction loss (proposed in [9]). To obtain style reconstructed images, we start from a noise image and optimize the style loss as [9] using the VGG-16 pre-trained on ImageNet. Fig. 3.4 shows an example of the style image reconstruction. We see that the style of image can be obtained until the 7th layer ($relu3\_3$)

The above observation holds true for the images and the styles that we evaluated. Combining the insight given by [7, 9], we may thus conclude that the low-level layers reflect the style of the image while the high-level layers capture the content of the image. More precisely, from the 6th layer ($relu3\_2$) to the 10th layer ($relu4\_3$), the network is capable of appropriately capturing content information in the images. The style information, on the other hand, can be obtained from the 2nd ($relu1\_2$) to the 7th ($relu3\_3$) layers.

[9] pointed out that image content and style cannot be completely disentangled. This indicates that depending on the objective, we have to appropriately design the layer levels of content and style features for their combination. We thus further analyze effectiveness of the layers from the 6th ($relu3\_2$) to the 10th ($relu4\_3$) for content matching to determine the best one for combination. We follow [9] to synthesize the stylized images where we set the contributions of content and style to be equal with each other. To this end, we fix the style matching from the 2nd ($relu1\_2$) to the 7th ($relu3\_3$) layers, while performing the content matching at every single layer from the 6th ($relu3\_2$) to the 10th layers ($relu4\_3$). Fig. 3.5 shows examples of stylized images having different layers in combination. We see that the content matching at the 6th and the 7th layers ($relu3\_2$ and $relu3\_3$) is most reasonable to keep the balance of content and style in stylized images.

Using above observation, we design our network to fully exploit the characteristics of image features. We choose the 6th layer for content because it has a smaller number

of parameters than the 7th layer (it is faster to learn). We choose the 4th layer for style because it is neither too early in layer nor marginally different from the layer used for content. In conclusion, we use the features at the 6th layer ($relu3\_2$) for content and those at the 4th layer ($relu2\_2$) for style.

3.4 Proposed Method

3.4.1 Network design

Our network follows end-to-end encoder-decoder architecture for rendering of the content in a given style [7, 11, 44]. The network in [7, 11, 44] possesses only one encoder to extract the semantic content and style. This means that the extracted semantic level of the content and that of the style are the same. When we stylize images, the role of the content should be different from that of the style because the content gives us what exist (object shapes and locations) in the rendered image and the style gives us the impression of the rendered image. Accordingly, the semantic level used for the rending should be different depending on the content or the style. Otherwise, unbalance between the content and style remains in stylized images. We thus design a network having two encoders in which their architectures are different from each other to extract different semantic levels of the content and the style. With the two encoders, our model treats the content and the style in different ways, allowing the network to be able to balance the roles of the content and the style better than the model having only one encoder.

Ideally, the network should be able to retain the semantics of the content as well as the statistics of the style as much as possible. The semantic content and style of an image are captured at different layers in the network (see [7, 9] and Section 3.3): the network obtains the style at low-level layers in depth while high-level layers become more sensitive to the actual content of the image. We thus design the encoders with different depths to retain useful information from both the content and the style. Namely, we design a deep encoder for the content and a shallow encoder for the style. Moreover, in order to reflect features extracted from the style at low-level to those from the content, we employ the feature injection via the skip-connection technique from the shallow encoder to the deep one. Because the content feature and the style feature

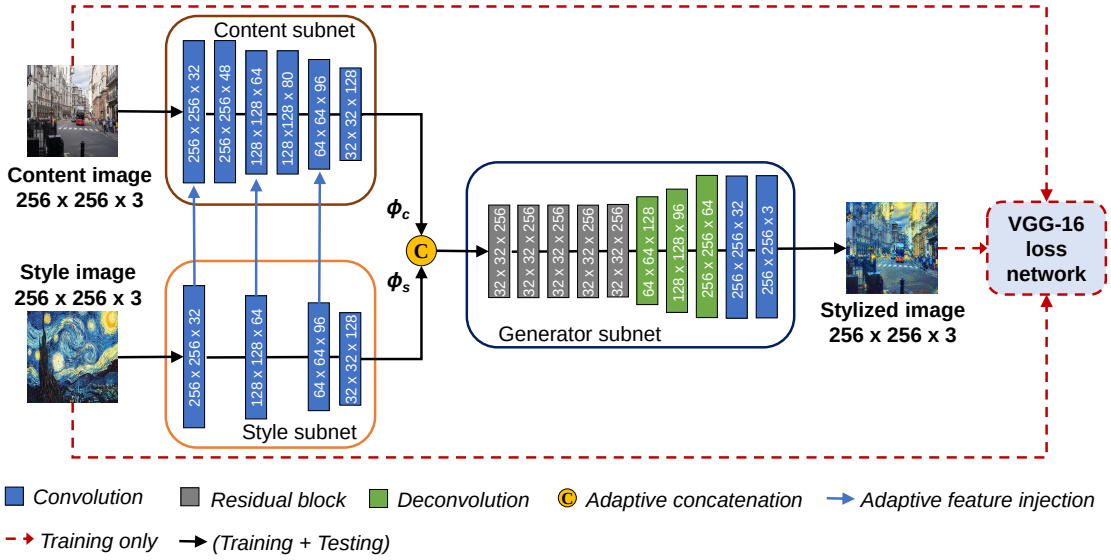


Figure 3.6: Framework of our proposed method. Our network consists of two encoders having different architectures and one decoder. The loss network is used to train the encoders and the decoder.

are extracted at different levels in the network, they have different characteristics. We thus introduce an effective concatenation to enhance the contribution of these features for good performances instead of implementing their simple ones.

3.4.2 Network architecture

Our proposed network consists of three Fully Convolutional Network (FCNs): two encoders and one decoder (Fig. 5.2). The two encoders are a deep network, the content subnet, to extract content feature ϕ_c from a content image, and a shallow network, the style subnet, to extract style feature ϕ_s from a style image. The feature injection is employed between the content subnet and the style subnet using the balance weight (cf. Section 3.4.4). This balance weight is also used to adaptively concatenate the features ϕ_c and ϕ_s at the top of content and style subnet before being fed into a deep network, the generator subnet, to produce a stylized image. We employ the VGG-16 model [30] as the loss network in the training phase.

Our network receives the content and style images where each image is with the size of $n \times n \times 3$ (n is the size of image, 3 are for RGB channels), and synthesizes an

stylized image of $n \times n \times 3$. In the training phase, we use the images of $256 \times 256 \times 3$ ($n = 256$). Although we train the network on images with the size of $256 \times 256 \times 3$, the network can accept any size of images in testing (n can be 64, 128, 256, or 512). We remark that the size of the content image and that of the style image have to be the same to ensure the consistency of the feature size when injecting and concatenating the content and the style features.

Content subnet

The content subnet is a stack of six convolution layers with the filter size of 3×3, and the padding size of 1×1. We use the stride of 2×2 at the third, the fifth, and the sixth layers to reduce the size of feature maps and the stride of 1×1 at the other layers. The numbers of the output channels are 32, 48, 64, 80, 96, and 128, respectively. Each convolution layer is followed by a spatial instance normalization (IN) layer [72] and a Rectified Linear Unit (ReLU) layer [105]. In order to avoid the border artifacts caused by convolution, the reflection-padding is used instead of the zero-padding similarly to [84].

Style subnet

The style subnet, which has four convolution layers, is shallow network (more precisely, shallower than the content subnet). All convolution layers have the filter size of 3×3, the reflection-padding of 1×1, and the stride of 2×2, except for the first layer that employs the stride of 1×1. The numbers of the output channels are 32, 64, 96, and 128, respectively. Similarly to the content subnet, each convolution layer is also followed by an IN layer [72] and a ReLU layer [105].

We employ feature injection from the feature ϕ_s^q at the q-th layer in the style subnet to those ϕ_c^p at the p-th layer in the content subnet, the size of whose feature map is the same (Table 3.1). To take into account the contributions of ϕ_s^q and ϕ_c^p, we introduce the adaptive feature injection with the balance weight (cf. Section 3.4.4).

Generator subnet

The generator subnet consists of five residual blocks, three deconvolution layers, and two convolution layers in this order.

Table 3.1: Architecture of our encoders. The arrow (←) indicates the adaptive feature injection.

Content subnet				Style subnet		
No	**Layer**	**Output channel**		**No**	**Layer**	**Output channel**
0	Content image	3		0	Style image	3
1	Convolution	32		1	Convolution	32
2	Instance normalization	32		2	Instance normalization	32
3	ReLU	32	←	3	ReLU	32
4	Convolution	48				
5	Instance normalization	48				
6	ReLU	48				
7	Convolution	64		4	Convolution	64
8	Instance normalization	64		5	Instance normalization	64
9	ReLU	64	←	6	ReLU	64
10	Convolution	80				
11	Instance normalization	80				
12	ReLU	80				
13	Convolution	96		7	Convolution	96
14	Instance normalization	96		8	Instance normalization	96
15	ReLU	96	←	9	ReLU	96
16	Convolution	128		10	Convolution	128
17	Instance normalization	128		11	Instance normalization	128
18	ReLU	128		12	ReLU	128

[7] argues that the residual block can enrich the information involved in the input feature. We, therefore, use residual blocks to increase the impact of the balance weight in the concatenated feature. Similarly to [7], we use five residual blocks outputting 256 channels, where each of them has two convolution layers with the filter size of 3×3, the reflection-padding of 1×1, the stride of 1×1, and a summation layer as in [31]. All convolution layers are followed by an IN layer [72] (we use it to replace the batch normalization [71] in the original architecture [31]) and a ReLU layer [105].

To upscale the feature map, we employ three deconvolution layers with the same filter size of 3×3, the reflection-padding of 1×1, and the stride of 2×2, outputting 128, 96, and 64 channels, respectively.

In order to eliminate the affect of the convolution stride, we use two convolution layers which have the filter size of 1×1, the padding of 0×0, and the stride of 1×1, outputting 32 and 3 channels. All deconvolution layers and convolution layers are followed by an IN layer [72] and a ReLU layer [105], except for the last convolution

layer that uses the tanh activation to guarantee that the range of the output can be normalized to be $[0, 255]$.

3.4.3 Loss function

We employ two loss functions for content loss and style loss, which are computed from layers of the loss network. The content loss $\mathcal{L}_c$ computes the similarity of high-level features between the content image and the stylized image. The style loss $\mathcal{L}_s$, on the other hand, computes the similarity of low-level features between the style image and the stylized image.

The overall loss is a weighted sum of the content loss and the style loss:

$$\mathcal{L}(\hat{y}, y_c, y_s) = \alpha \mathcal{L}_c(\hat{y}, y_c) + (1 - \alpha)\mathcal{L}_s(\hat{y}, y_s), \tag{3.1}$$

where y_c, y_s, and $\hat{y}$ denote the content image, the style, and the stylized image, respectively. α is the combination weight (we set $\alpha = 0.5$ in our experiments to equally weight these two loss functions).

We obtain the content loss at M layers as follows:

$$\mathcal{L}_c(\hat{y}, y_c) = \frac{1}{M} \sum_{k \in M} \frac{1}{C_k \times H_k \times W_k} \|\Phi_k(\hat{y}) - \Phi_k(y_c)\|_2, \tag{3.2}$$

where $\Phi_k(\cdot)$ denotes the normalized feature map at the k-th layer, which has $C_k \times H_k \times W_k$ elements. The range of $\mathcal{L}_c$ is $[0, 1]$.

The style loss is computed at N layers as follows:

$$\mathcal{L}_s(\hat{y}, y_s) = \frac{1}{N} \sum_{k \in N} \|G(\Phi_k(\hat{y})) - G(\Phi_k(y_s))\|_F, \tag{3.3}$$

where $\|\cdot\|_F$ denotes the Frobenius norm [81]. $G(\Phi_k(\cdot))$ is the Gram matrix [81] of the normalized feature map at the k-th layer. The Gram matrix $G_{C_k \times C_k}$ has elements $G_{ij} = \langle v_i, v_j \rangle$ where v_i, v_j are features at the i-th and the j-th channels respectively of the feature map $\Phi_k(\cdot)$. The range of $\mathcal{L}_s$ is $[0, 1]$.

3.4.4 Adaptive feature injection and concatenation

In our network, we employ the feature injection between the content features and the style features. We also concatenate them to feed into the generator subnet. To weight the contributions of the content features and the style features, we introduce the balance weight γ. This balance weight is adaptively updated during the training so that it retains the balance between the content and the style in stylized images.

At the t-th iteration in training phase, γ_t is computed as follows:

$$\gamma_t = \frac{\mathcal{L}_s(t)}{\mathcal{L}_s(t) + \mathcal{L}_c(t)}, \tag{3.4}$$

where $\mathcal{L}_s(t)$ and $\mathcal{L}_c(t)$ are the style loss and the content loss at the t-th iteration in the training phase. To restrict the fluctuation of the balance weight, we compute γ at every non-overlapping T iterations and use it for the next T iterations:

$$\gamma = \frac{1}{T}\sum_{t=1}^{T}\gamma_t. \tag{3.5}$$

Using γ, we sum up the content feature at the p-th layer ϕ_c^p and the style feature at the q-th layer ϕ_s^q for the feature in adaptive feature injection as follows:

$$\phi^{pq} = (\gamma \times \phi_c^p) + ((1-\gamma) \times \phi_s^q). \tag{3.6}$$

Similarly, we concatenate the content feature ϕ_c and the style feature ϕ_s in the adaptive concatenation as follows:

$$\phi = (\gamma \times \phi_c) \oplus ((1-\gamma) \times \phi_s). \tag{3.7}$$

The learned balance weight γ ensures the balance of the contributions of the content feature and the style feature in both feature injection and concatenation layers. For example, when $\mathcal{L}_s$ is smaller than $\mathcal{L}_c$ (meaning $\gamma \leq 0.5$ in Eq. (3.5)), the contribution of style feature is increased in the next iterations, and vice verse. Moreover, the learned balance weight γ is more advantageous than the fixed balance weight that does not concern the balance of losses.

In order to explicitly control the contribution ratio of the content and the style, we

Figure 3.7: Styles used in experiments. From left to right: Starry Night, Mosaic, Composition VII, La Muse, The Wave, and Feathers.

manually set the expected contribution ratio in the loss function, and then introduce the learnable weight that allows us to change stylized images as we expect. The combination weight α takes the former role while the learnable balance weight γ does the latter role. In other words, in our method, α sets an expected contribution ratio of content and style in stylized images through the loss function while γ controls the learning direction of the network during the training to achieve the contribution ratio specified by α. In our experiments where we set $\alpha = 0.5$, we see that γ works for the equal contribution ratio of the content and the style as expected (see Section 3.6.1 for details). We remark that α and γ together play the role of the indicator for how much the content and the style are emphasized in obtained stylized images.

3.5 Experimental Settings

3.5.1 Dataset and compared methods

Dataset

We used in our experiments, images in the MS-COCO 2014 dataset [108] as our content images, and six famous paintings widely used in style transfer [7–9], as our style images (cf. Fig. 3.7).

We used the MS-COCO 2014 training set for our training, and we randomly selected 20 images from the MS-COCO 2014 validation set for our validation. In the testing phase, on the other hand, we randomly selected 50 images from MS-COCO 2014 validation (different ones from the 20 images used in our validation).

Compared methods

We compared our method with SOTA methods: Gatys+ [9], Johnson+ [7], Huang+ [8], Sheng+ [10], Chen+ [11], and Li+ [12]. We note that Gatys+ is based on IOB-NST and the others are on MOB-NST. For Gatys+, we used the re-implementation version by J. Johnson[1]. For the others, we used publicly available source codes with parameters recommended by the authors (Johnson+[2], Huang+[3], Sheng+[4], Chen+[5], Li+[6]). We remark that we set 1000 iterations for Gatys+.

3.5.2 Evaluation metrics

In order to evaluate the quality of synthesized images, most previous work employed user studies although they are subjective and have ambiguity in evaluation. We, on the other hand, evaluate stylized images by quantifying the content and style losses. Intuitively, when the total loss is sufficiently small, we may say that the overall quality of stylized images is good. Furthermore, the quality of stylized images also depends on how the content and the style are reflected in them. We have to consider these two factors in evaluating the quality of stylized images. Since the contributions of the content and the style are controlled by α (set in advance) in our method, we may see if a synthesized image is good in quality by evaluating (i) whether its total loss is sufficiently small, and (ii) whether the ratio between its content and style losses consistently agrees with the pre-set contribution ratio (i.e., combination weight α) between the content and the style. We thus introduce a metric to evaluate the quality of synthesized images using these two criteria. We remind that we set $\alpha = 0.5$ (for simplicity) in our experiments to see the content and style losses converge to almost the same values.

For each pair of content image c and style image s, we compute content loss $\mathcal{L}_c$ and style loss $\mathcal{L}_s$. In the 2D plane whose coordinate system is defined by content loss and style loss, the criterion (i) can be measured using the distance between the origin and

[1]https://github.com/jcjohnson/neural-style
[2]https://github.com/jcjohnson/fast-neural-style
[3]https://github.com/xunhuang1995/AdaIN-style
[4]https://github.com/LucasSheng/avatar-net
[5]https://github.com/rtqichen/style-swap
[6]https://github.com/Yijunmaverick/UniversalStyleTransfer

$(\mathcal{L}_c, \mathcal{L}_s)$. The criterion (ii), on the other hand, can be measured by evaluating how close $(\mathcal{L}_c, \mathcal{L}_s)$ is to the line of "content loss"="style loss" (called the balanced axis hereafter).

We assume that we have K stylized images. We normalize content loss and style loss for each stylized image over K images:

$$\widetilde{\mathcal{L}_c} = \frac{1}{1 + \exp\left(\frac{\mathcal{L}_c - \overline{\mathcal{L}_c}}{\sigma_c}\right)}, \quad \widetilde{\mathcal{L}_s} = \frac{1}{1 + \exp\left(\frac{\mathcal{L}_s - \overline{\mathcal{L}_s}}{\sigma_s}\right)}, \tag{3.8}$$

where $\overline{\mathcal{L}_c}$, σ_c, $\overline{\mathcal{L}_s}$, and σ_s are the mean and the standard deviation of content loss and style loss over K stylized images, respectively.

The quality of stylized images with respect to the criterion (i) is measured using

$$length = \sqrt{\widetilde{\mathcal{L}_c}^2 + \widetilde{\mathcal{L}_s}^2}. \tag{3.9}$$

Let ω ($\in [0, \frac{\pi}{4}]$) denote the angle between the line going through the origin and $(\widetilde{\mathcal{L}_c}, \widetilde{\mathcal{L}_s})$ and the content loss axis or the style loss axis (the smaller angle is selected):

$$\omega = \begin{cases} \tan^{-1} \frac{\widetilde{\mathcal{L}_s}}{\widetilde{\mathcal{L}_c}} & \text{if } \widetilde{\mathcal{L}_c} \geq \widetilde{\mathcal{L}_s} \\ \pi/2 - \tan^{-1} \frac{\widetilde{\mathcal{L}_s}}{\widetilde{\mathcal{L}_c}} & \text{otherwise} \end{cases}. \tag{3.10}$$

Larger ω indicates that $(\widetilde{\mathcal{L}_c}, \widetilde{\mathcal{L}_s})$ is closer to the balanced axis, meaning that the stylized image is more balanced in content and style. This reflects the criterion (ii).

Using *length* and ω above, we define our metric *balance*:

$$balance = \frac{\tan(\omega)}{length}. \tag{3.11}$$

balance concerns both the two criteria (i) and (ii). Therefore it is a useful metric for evaluating stylized images. We note that larger *balance* is better because $\tan(\omega)$ should be larger and *length* should be smaller for better stylized images.

3.5.3 Implementation and training details

Implementation setup

We implemented our method in PyTorch[7]. We used the instance incremental learning strategy for dealing with multiple styles. We conducted all experiments using a PC with CPU core i7 3.7 GHz, 12 GB of RAM, and GTX 770 GPU (4 GB of VRAM).

We performed the adaptive feature injection from layers $q = 3, 6, 9$ in the style subnet to layers $p = 3, 9, 15$ in the content subnet, respectively (Table 3.1). We adopted the VGG-16 model [30] pre-trained on the ImageNet [80] as the loss network without any fine-tuning. All layers after $relu4\_3$ layer were dropped. We obtained the content loss at $M = 1$ layer, e.g., $relu4\_3$, and the style loss at $N = 3$ layers, e.g., $relu1\_2$, $relu2\_2$, and $relu3\_3$ (M and N are defined in Section 5.4.3).

Training the model

Our method addresses a one-style model to reduce computational time. For training a new yet unknown style, we fine-tune parameters from an existing model. With this learning strategy, our method can easily adapt a new style with a lower cost than existing work [7–9, 44]. Moreover, the fine-tuning learning enables our method to deal with an unlimited number of styles fast unlike existing methods such as [11, 84].

We first trained an initial model on the Starry Night style and then incrementally fine-tuned on the other styles one by one. We trained the network on the Starry Night style with a batch size of 2 for 80k iterations corresponding to 2 epochs. The balance weight γ in Eq. (3.5) is re-computed at every $T = 500$ iterations. All the training and validation images are resized to 256×256. To train the model, we used the Adam optimizer [107] with the learning rate of 10^{-3}, the moments $\beta_1 = 0.9$ and $\beta_2 = 0.999$, and the division from zero parameter $\epsilon = 10^{-8}$. We did not use the learning rate decay and the weight decay.

For the initial model, we trained all subnets simultaneously with independently updating the weight of each subnet. Validation was performed at every 100 iterations during the training process. When observing the content loss and the style loss on the validation set, if any loss function raises the overfitting problem, we stopped updating

[7] https://pytorch.org/

the weight of the corresponding subnet.

We incrementally fine-tuned the initial model to the other styles one by one. 2000 images in the MS-COCO 2014 training set [108] were randomly selected as content images for training. The network was trained for 1000 iterations with the batch size of 2. The Adam optimizer [107] was also used with the same parameters as the training of the initial model. The balance weight γ in Eq. (3.5) was re-computed at every $T = 50$ iterations. The loss-based training technique was also applied to avoid overfitting, where the validation was performed at every 50 iterations.

3.6 Experimental Results

3.6.1 Comparison with state-of-the-arts

Qualitative evaluation.

Figures 3.8 and 3.9 show examples of the obtained results, showing that the stylized images obtained by our method are more balanced in content and style. We also see that overall the results obtained by Gatys+ [9], Sheng+ [10], and Li+ [12] reflect the style well, but they mostly lose content (we cannot understand the content of stylized results using La Muse and Feathers styles). In some styles (Starry Night, Composition VII, and The Wave), we see that Johnson+ [7] seems to randomly select a patch in the style and paste it into the content image. Huang+ [8] also loses the content and suffers from a so-called checkerboard effect. We also see that Chen+ [11] loses almost style and tends to keep the original content images.

To objectively compare the obtained results, we conducted three user studies, including overall quality, content preserving and style look-like. From the visual comparison in Figs. 3.8 and 3.9, we see that evaluating all stylized results among compared methods is pretty difficult. We thus picked up three methods only for our user studies. To this end, we investigated the quantitative comparison (Section 3.6.1). As Gatys+ [9] is known to keep styles most while Johnson+ [7] retain the content most, these methods are appropriate to choose for our user studies. Among the remaining compared methods, we see that Huang+ [8] is most balanced (the loss distributions of Huang+ [8] appear near balanced axis (Fig. 3.11)). We, therefore, chose Gatys+ [9], Johnson+ [7] and Huang+ [8] for our user studies.

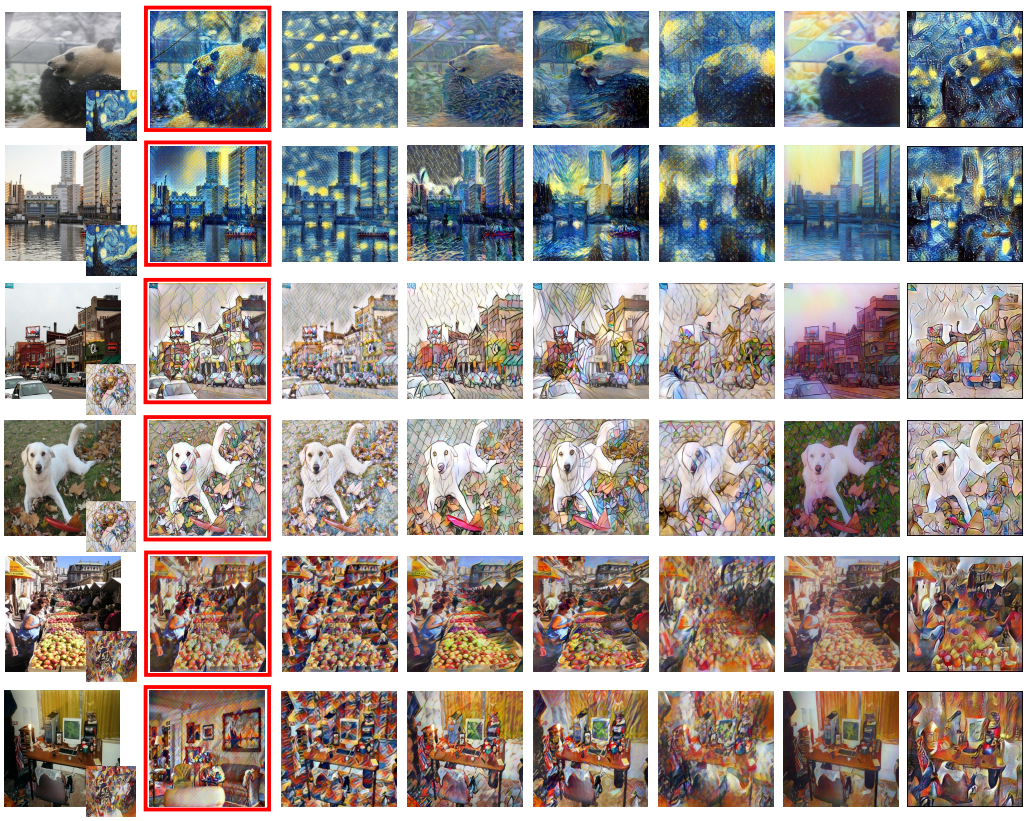


Figure 3.8: Visual comparison of our method against the state-of-the-art methods. Left-most column: content image (large) and style image (small). From left to right: the stylized image by our method, Johnson+ [7], Huang+ [8], and Gatys+ [9], Sheng+ [10], Chen+ [11], and Li+ [12]. Our results surrounded with red rectangles are more balanced in content and style than the others. Note that all stylized images are with the size of 512×512.

For our user studies, we randomly selected 20 images from the 50 testing images as content images and chose 5 styles by excluding The Wave style because it is simpler than the other styles (Fig. 3.7). We remark that the combination of 20 content images and 5 styles results in 100 stylized images by each method. In each user study, we presented 100 sets of images to 31 subjects where each set consists of a content image, a style image, and four output images obtained by our method and the three comparison methods [7–9]. We then asked the subjects to rank the four output stylized images at each set (1st is best, and 4th is worst). For the overall quality study, the subjects were asked to give the ranking based on the overall quality at each set. For the content

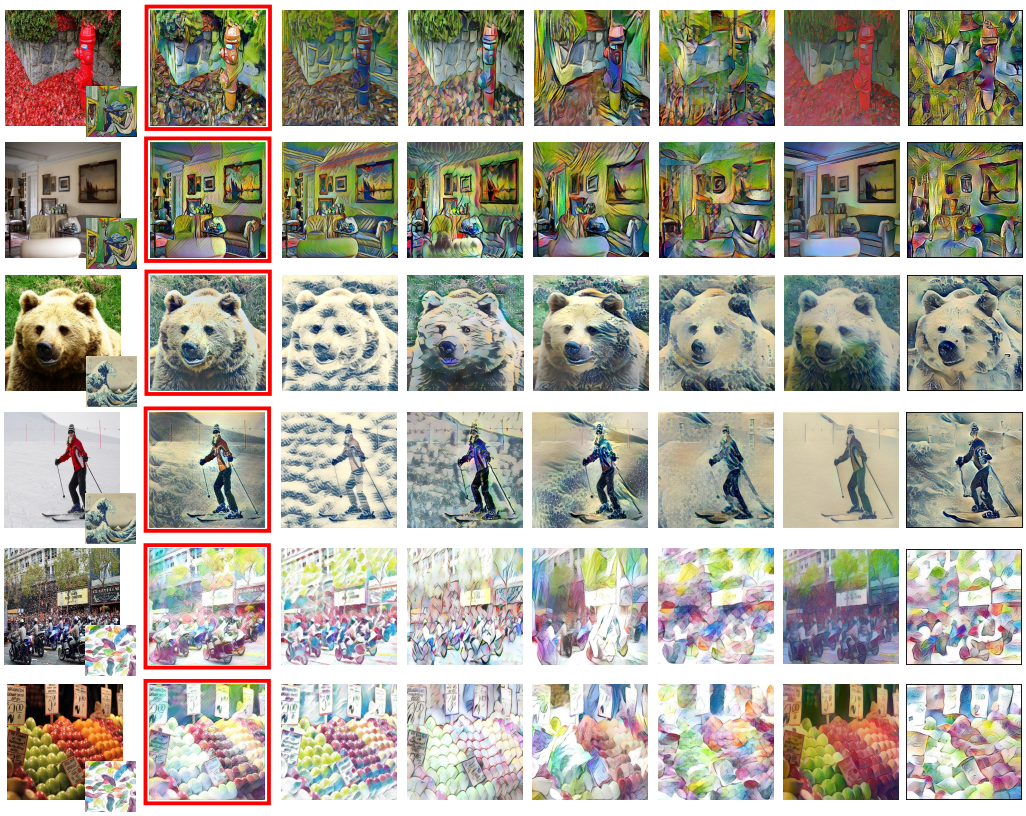


Figure 3.9: Visual comparison of our method against the state-of-the-art methods. Left-most column: content image (large) and style image (small). From left to right: the stylized image by our method, Johnson+ [7], Huang+ [8], and Gatys+ [9], Sheng+ [10], Chen+ [11], and Li+ [12]. Our results surrounded with red rectangles are more balanced in content and style than the others. Note that all stylized images are with the size of 512×512.

preserving study, the subjects were asked to rank output images in each set based on how faithfully the images preserve the content in content images. For the style look-like study, on the other hand, the subjects ranked output images in each set based on how the images look like the style in style images. We note that four output images are aligned in the random order in each set and that each set was displayed for 6 seconds.

Table 3.2, Table 3.3, and Table 3.4 show the average of rankings over the 100 sets for the overall quality, the content preserving, and the style look-like studies, respectively. We also computed the average of rankings in each style, which is also illustrated in

Table 3.2, Table 3.3, and Table 3.4.

We see that our method takes the best ranking among the four methods in overall quality (Table 3.2). Looking into the results in more detail, we see that our method is ranked in the first place at the Mosaic style, and in the second place at others (except for Composition VII style). This indicates that our method performs stably well in overall quality in accordance with human cognition. We remark that the Composition VII style is rather complex (Fig. 3.7) and, the results for this style are difficult to evaluate. We also remark that the single-style models (ours, Gatys+[9], and Johnson+[7]) performed better than the multi-style model (Huang+[8]).

For the content preserving (Table 3.3) and the style look-like (Table 3.4) studies, our method takes the second best ranking. Note that the scores in these studies more largely distributed than those in the overall quality study. As MOB-NST is known to perform better in content preserving than IOB-NST [100]; Johnson+ [7], which is MOB-NST, takes the best ranking in the content preserving study. Gatys+ [9], on the other hand, which is IOB-NST, takes the best ranking in the style look-like study. In contrast, our method is ranked in the second place for all styles in the content preserving study (except for Composition VII style) (Table 3.3) and in the style look-like study (except for Feathers style) (Table 3.4). These indicate that our method stably produces stylized images balanced in content and style for almost all the styles. We remark that in the case of the Feathers style, the two best methods for the look-like study follow the MOB-NST approach. As MOB-NST is known not to keep styles well [100], this suggests that the Feathers style is a difficult style for users to evaluate stylized images.

Quantitative evaluation.

In order to quantitatively evaluate the obtained results, we computed the averages of *length*'s and *balance*'s over 300 (= 50 contents $\times$ 6 styles) sets for each method (Table 3.5). We see that our method performs best both in *length* and *balance*. We also computed the averages of *length*'s and *balance*'s in each style, which is illustrated in Fig. 3.10. Fig. 3.10 shows that our method performs best in *length* and best in *balance* for all the styles.

To look into the results in more detail, we show the loss distribution of 50 stylized images in each style (Fig. 3.11). We see that (1) the content loss and the style loss (for each stylized result) in our method are similar with each other and that (2) loss

Table 3.2: Average of rankings in the overall quality study. The best and the second best results are given in **red** and **blue**, respectively.

Style	Ours	Johnson+ [7]	Huang+ [8]	Gatys+ [9]
Starry Night	**2.12**	2.72	3.14	**2.01**
Mosaic	**2.21**	**2.25**	2.91	2.63
Composition VII	2.47	2.95	**2.4**	**2.18**
La Muse	**2.38**	**2.28**	2.82	2.51
Feathers	**2.15**	**1.82**	3.28	2.74
All together	**2.27**	**2.40**	2.91	2.41

distributions in our method appear densely near the balanced axis for all the styles while those in the other methods do not.

Computational speed.

We measured the running time for generating 300 stylized images with the sizes of 256×256 and 512×512 by each method and compared the average for generating one stylized image by each method.

Table 3.6 illustrates the average of the running time in generating one stylized image. As we see, our method is the fastest and speeds up 22 times for the image size of 256×256 and 21 times for that of 512×512 when compared with the fastest state-of-the-arts [7]. We can thus conclude that our method is promising for real-time applications.

3.6.2 Ablation studies

Behavior of balance weight γ during the training

We investigate the behavior of balance weight γ to verify that γ is adaptively updated to converge to an expected value.

Figure 3.12 illustrates how balance weight γ changes during the training on the Starry Night style. We see that γ is adaptively updated corresponding to the content and style losses. We remark that since we set $\alpha = 0.5$ in the loss function, γ is expected

Table 3.3: Average of rankings in the content preserving study. The best and the second best results are given in **red** and **blue**, respectively.

Style	Ours	Johnson+ [7]	Huang+ [8]	Gatys+ [9]
Starry Night	**2.53**	**1.96**	2.67	2.84
Mosaic	**2.13**	**1.60**	3.05	3.22
Composition VII	3.02	**1.81**	**2.50**	2.67
La Muse	**1.99**	**1.82**	3.06	3.13
Feathers	**2.02**	**1.81**	2.50	2.67
All together	**2.34**	**1.80**	2.87	2.99

to be close to 0.5 after the training. At the beginning of training, the style loss $\mathcal{L}_s$ is far larger than the content loss $\mathcal{L}_c$, resulting in γ far larger than 0.5 (close to 1.0). As the training proceeds, the network is gradually optimized, resulting γ close to 0.5 in the end of the training.

We observe that γ quickly decreases after one epoch (about 40k iterations). This can be explained as follows. After one epoch, the overfitting problem on the style image occurs since our network is trained using a single style image. Hence, the style loss quickly drops. As a result, the behavior of γ becomes different. Indeed, we observed that the style loss raised the overfitting problem through the validation phase. We thus stopped the training of the style subnet while kept updating the weights of the other subnets. As a result, the content loss decreased more quickly than the style loss. Then, γ was gradually recovered; its value became close to 0.5 in the end of training.

This evaluation confirms that γ gradually adapts to achieve the equal contributions of content and style in stylized images during the training thanks to our adaptive feature injection and concatenation. We remark that we observed similar behaviors of γ for other styles.

Effectiveness of feature injection

In this section, we evaluate the effectiveness of the introduction to the adaptive feature injection between the content subnet and the style subnet.

Table 3.4: Average of rankings in the style look-like study. The best and the second best results are given in **red** and **blue**, respectively.

Style	Ours	Johnson+ [7]	Huang+ [8]	Gatys+ [9]
Starry Night	**2.27**	2.77	3.34	**1.61**
Mosaic	**2.26**	2.66	2.94	**2.13**
Composition VII	**2.49**	2.96	2.65	**1.90**
La Muse	**2.71**	2.81	2.81	**1.67**
Feathers	**1.69**	**2.34**	3.42	2.55
All together	**2.28**	2.71	3.03	**1.97**

We compared our complete model with the model w/o feature injection (i.e., the model that disabled only the adaptive feature injection), which is shown in Fig. 3.13. Fig. 3.13 shows that the stylized images obtained by the complete model are in general more balanced in content and style than those by the model w/o feature injection. However, we can see roughly global structure appearing in the synthesized images in Fig. 3.13 upper set (in particular, the leftmost which is with Starry Night style). This can be explained as follows. In general, the model w/o feature injection tends to preserve more content than style while the complete model does more style than content. This is because the feature injection from the style subnet to the content subnet tries to reduce the style loss (see below). The feature injection at multiple layers employed in the content and style subnets helps to keep both global and local structure in rendering. As a result, global structure in stylized images such as the stroke in the Starry Night may sometimes become impressive.

We also compared the *length* and *balance* of stylized images (Table 3.7). We see that the complete model performs better both in *length* and *balance* than the model w/o feature injection. Table 3.7 also shows that employing adaptive feature injection improves both *length* and *balance* for each style (except for La Muse style). This indicates that adaptive feature injection is effective to improve not only the quality but also the balance in content and style of stylized images. With respect to the La Muse style, *length* of the complete model is comparable to that of model w/o feature

Table 3.5: Averages of *length* (smaller is better) and *balance* (larger is better).

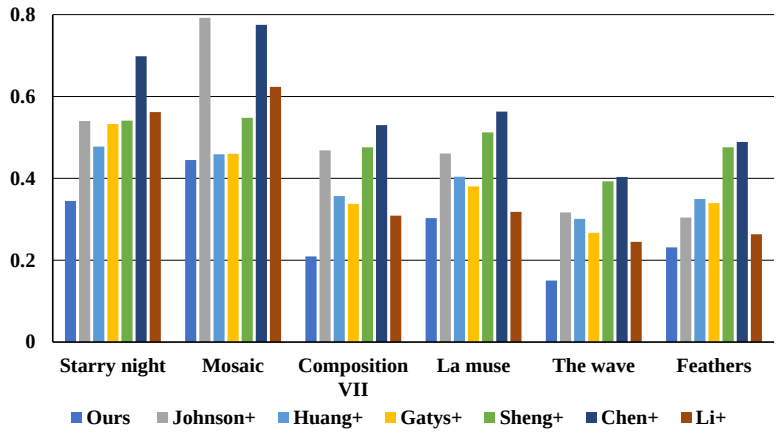
Method	*length* (⇓)	*balance* (⇑)
Ours	**0.37**	**2.95**
Johnson+ [7]	0.54	1.60
Huang+ [8]	0.45	1.23
Gatys+ [9]	0.45	1.36
Sheng+ [10]	0.52	1.21
Chen+ [11]	0.59	0.72
Li+ [12]	0.49	1.40

Table 3.6: The average wall-clock time in second for producing one stylized image.

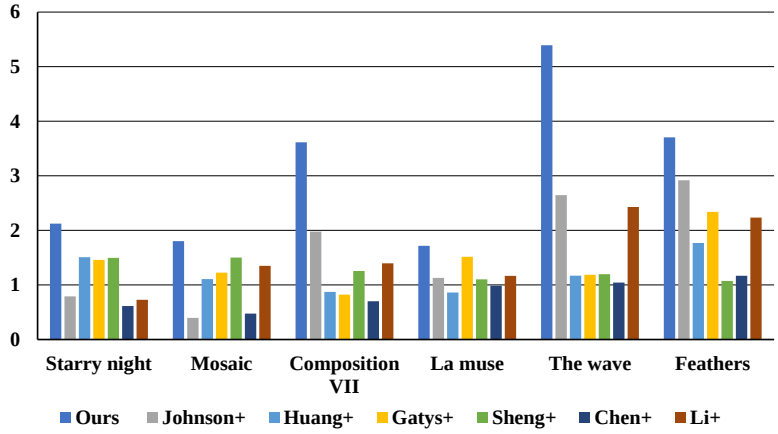
Method	**Image size**		**Implemented framework**
	256×256	512×512	
Ours	**0.05**	**0.18**	PyTorch
Johnson+ [7]	1.12	3.79	Torch
Huang+ [8]	1.98	6.78	Torch
Gatys+ [9]	74.12	269.74	Torch
Sheng+ [10]	3.04	10.67	TensorFlow
Chen+ [11]	2.74	9.33	Torch
Li+ [12]	3.53	9.42	Torch

injection, however *balance* is not the case. This can be explained as follows. The La Muse style follows Cubism and thus it is very unique. Because of this, the adaptive feature injection tends to keep more style to reflect the impression of this style.

Finally, we compare the loss distributions of 50 stylized images in each style (Fig. 3.14). We see that for all styles (except for the La Muse style) the loss distributions of the complete model appears more densely near the balanced axis and is closer to the origin than those of the model w/o feature injection for all styles. In the case of the Starry Night style (Fig. 3.14a), we see that the model w/o feature injection preserves much more content than the style because the loss distribution appears far above the balanced axis. This observation also holds true for the Mosaic style (Fig. 3.14b), the



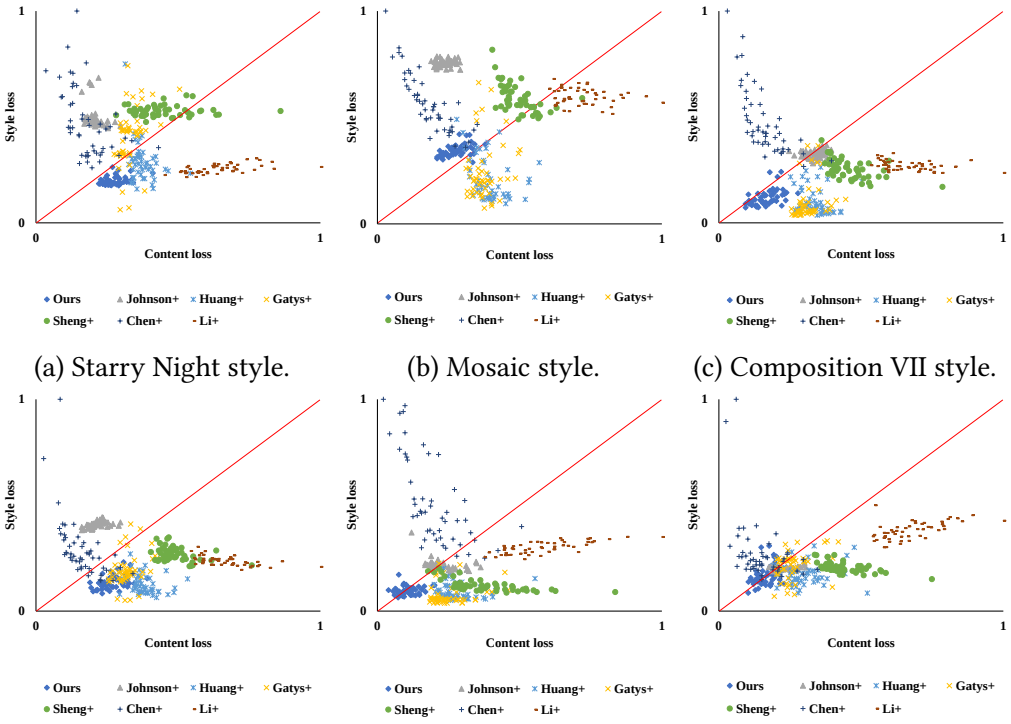
(a) *length* (⇓).



(b) *balance* (⇑).

Figure 3.10: Averages of *length* and *balance* in each style.

Composition VII (Fig. 3.14c), and the La Muse (Fig. 3.14d). By using adaptive feature injection, the complete model is able to reduce the style loss in stylized images (e.g., the Starry Night, the Mosaic, the Composition VII, the La Muse styles), compared to the model w/o feature injection. These observations indicate that the adaptive feature injection effectively improves to keep the balance in content and style of stylized images.



(a) Starry Night style. (b) Mosaic style. (c) Composition VII style.

(d) La Muse style. (e) The Wave style. (f) Feathers style.

Figure 3.11: Loss distribution in each style. Red lines denote the balanced axis. Our method has the distributions nearer the balanced axis than the other methods.

Effectiveness of combination weight α and balance weight γ

Here, we evaluate the necessity of combination weight α in Eq. (3.1) and balance weight γ in Eq. (3.4). In particular, we evaluate whether α plays the role of explicitly controlling the contribution ratio of the content and the style.

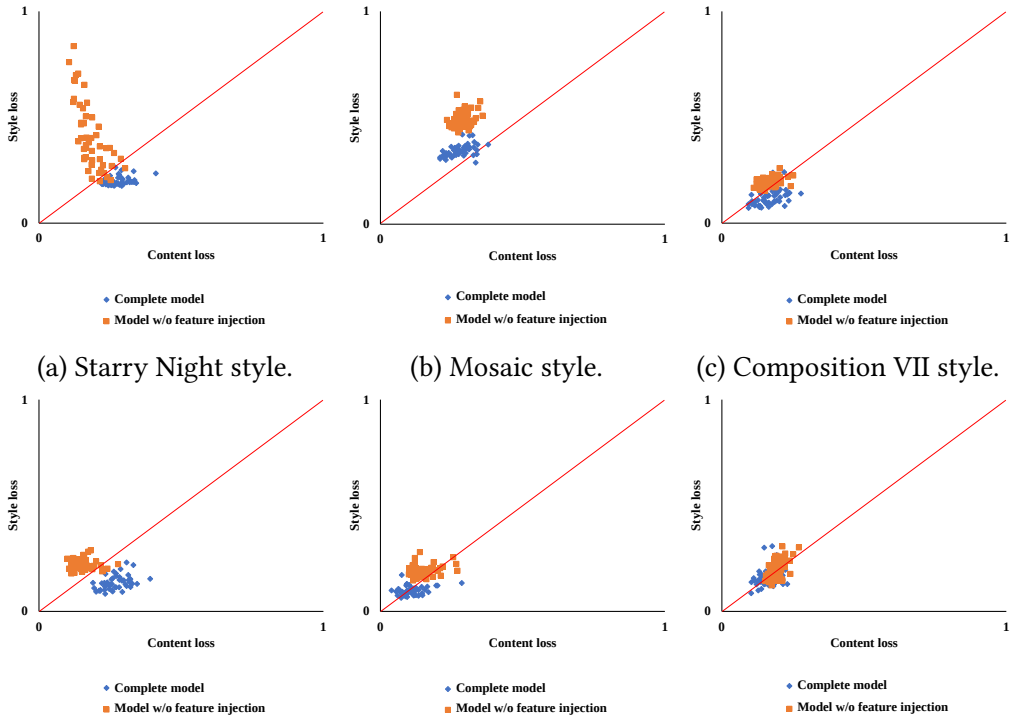
We generated stylized images using different values of α: $\alpha = 0.1, 0.3, 0.7, 0.9$. The results are illustrated in Fig. 3.15 where the complete model denotes the model using α and γ together while the model w/o γ denotes the model using α only (i.e., γ is disabled). Ideally, for smaller α, the style is more emphasized and results become more similar to those by Gatys+ [9]. For larger α, on the other hand, the content is more emphasized and results become more similar to those by Johnson+ [7]. We observe these in Fig. 3.15 and see that α of the complete model indeed controls the contribution ratio of the content and the style as we expected. However, we see that the model w/o γ is not the case. This observation suggests the necessity of both α and γ.

Figure 3.12: Behavior of γ during the training on the Starry Night style.

Figure 3.13: Visual comparison of the complete model and the model w/o feature injection. In each block, from left to right, a content image (large one) with a style (small one) is followed by outputs by the complete model and the model w/o feature injection. Note that all stylized images are with the size of 512×512.

3.7 Conclusion

We presented an end-to-end two-stream network for balancing the content and style in stylized images. Our proposed method utilizes a deep FCN to preserve the semantic content and a shallow FCN to faithfully learn the style representation, whose outputs are adaptively feature injected and concatenated using the balance weight and fed into the decoder to generate stylized images. Our intensive experiments using six famous styles widely used in style transfer demonstrate the effectiveness of our proposed method against state-of-the-art methods in terms of balancing content and style. Furthermore, our proposed method outperforms the state-of-the-art methods in speed.



(a) Starry Night style. (b) Mosaic style. (c) Composition VII style.

(d) La Muse style. (e) The Wave style. (f) Feathers style.

Figure 3.14: Loss distribution in each style obtained by the complete model and the model w/o feature injection. Red lines denote the balanced axis.

Table 3.7: Averages of *length* (smaller is better) and *balance* (larger is better) in the complete model (denoted by **complete**) and the model w/o feature injection (denoted by **w/o injection**).

Style	*length* (⇓)		*balance* (⇑)	
	complete	**w/o injection**	**complete**	**w/o injection**
Starry Night	**0.34**	0.46	**2.12**	1.35
Mosaic	**0.45**	0.57	**1.80**	1.03
Composition VII	**0.21**	0.26	**3.61**	3.21
La Muse	0.30	**0.27**	1.72	**2.54**
The Wave	**0.15**	0.24	**5.39**	3.15
Feathers	**0.23**	0.28	**3.71**	3.20
All together	**0.28**	0.35	**3.06**	2.41

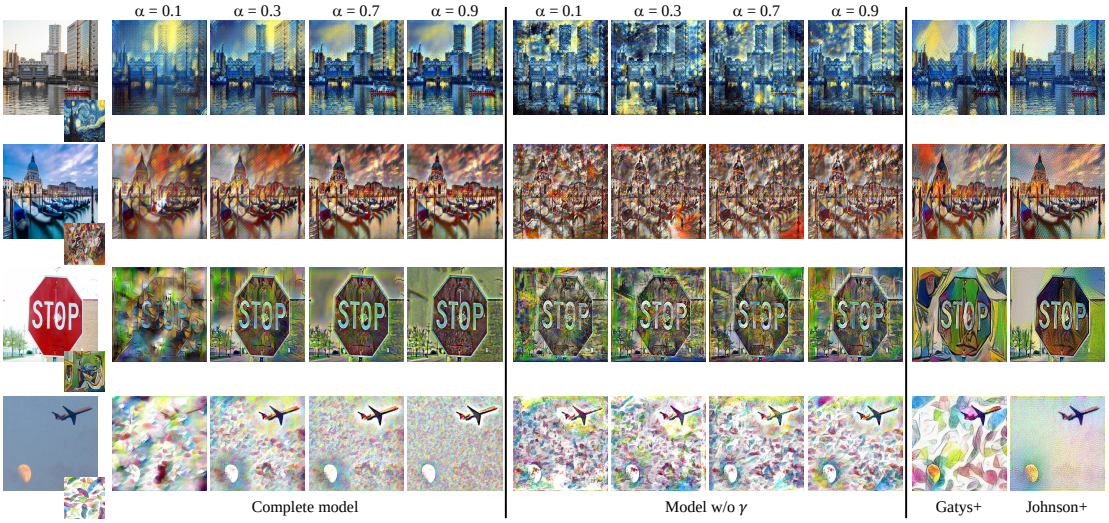


Figure 3.15: Example of stylized images by changing α from 0.1 to 0.9. Left-most column: the content image (large) and the style image (small). From left to right: the stylized image using various α. The last column shows results obtained by Gatys+ [9] and Johnson+ [7] for the reference.

4

Learning Background and Foreground Adversarially for Image Manipulation with Text

4.1 Introduction

Image manipulation with text [13] is to manipulate a given source image semantically with given text descriptions, while still maintain features that are irrelevant to what text descriptions. Text descriptions are usually on foreground (objects), and thus the task is to render foreground given as a text description into a given source image.

Image manipulation with text requires to disentangle the semantics contained in image and text information and then combine the disentangled semantics to synthesize realistic images. This suggests to separately deal with text descriptions and images with different semantic levels. We thus design a GAN with a pair of discriminators, called *Paired-D GAN*, to separately condition text descriptions and images. Indeed, dual

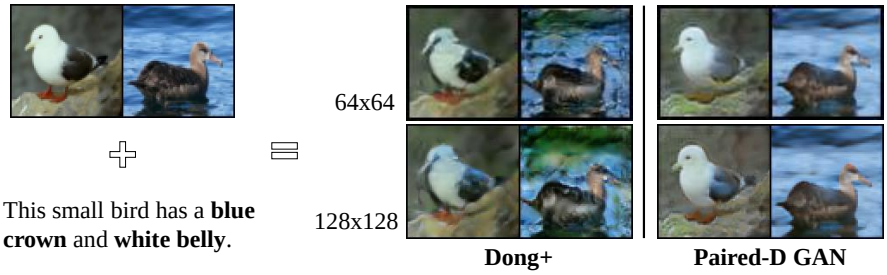


Figure 4.1: Examples of synthesized images. Our results match the text description more precisely than [13] while successfully retaining background of the source image. The performance of our method does not change for different sizes of images (64×64 and 128×128 images).

discriminator GAN [109] showed that having two discriminators is more effective than GANs with one discriminator for image synthesis. Different from dual discriminator GAN, we design different architectures for two discriminators to deal with different levels of semantics of text descriptions and images. The two discriminators separately judge foreground and background of the synthesized image to meet an input text description and a source image. Furthermore, we employ the skip-connection in the generator to more precisely retain background information in the source image. We also introduce a training process for adversarial learning in the three-player minimax game of the generator and two discriminators. In this way, Paired-D GAN improves the quality of synthesized images. Experiments on the Caltech-200 bird dataset [14] and the Oxford-102 flower dataset [15] demonstrate outperformances of Paired-D GAN against [13, 62]. Fig. 5.1 shows an example of our results.

4.2 Related Work

Generative Adversarial Network (GAN) [64] can be constrained on various conditions not only to generate plausible images but also to meet the conditions. Among various conditions on GAN, text descriptions make image synthesis easier and more friendly to us. Reed et al. [45] proposed an end-to-end GAN using the text condition. They employed a pre-trained text encoder [47] to extract text features from an input text, and then combined text features with a vector representing random noise to produce the input of the generator. They also employed the combination of text features and image features in the discriminator to discriminate real images and generated images.

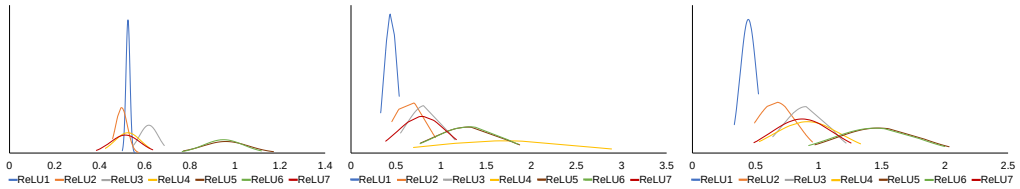
Their proposed model [45] became the baseline of the GAN framework for generating images from text descriptions.

As an extension, a model conditioned on texts and location information was proposed [101]. Models with two stages of GAN, Stack-GAN [18] (and Stack-GAN++ [46]), were also proposed, showing successfully generated higher resolution images (256×256), compared to [45] (64 × 64). These models [18, 45, 46, 62] condition on GAN only texts or a pair of texts and location information [101].

Addressing the background problem in image synthesis, Yang et al. [62] proposed to decompose the image synthesis into two phases using foreground and background generators. They fed random noise vectors to a long short-term memory (LSTM) network to obtain hidden states for the foreground generator and used the first hidden state to generate background. They then combined foreground and background by a compositor operator. However, decomposing foreground and background may cause less realistic images.

The model proposed by Dong et al. [13] is most related with ours. It also conditions text and source image on GAN. The architecture of the model is, however, similar to [45] and has a single discriminator: the noise vector in [45] is replaced by image features from the image encoder. Though it generates images that match the semantic meaning of the input text description while maintaining other parts of a source image, it does not preserve background precisely because the discriminator is used only for foreground; synthesized images are less realistic images.

Different from the above mentioned models, we fully take into account each role of foreground and background in synthesized images. More precisely, our proposed Paired-D GAN is conditioned on both text descriptions and images, has skip-connections in its generator to preserve background information as much as possible, and has two discriminators with different architectures for synthesizing realistic images. Paired-D GAN generates simultaneously foreground and background.



(a) Same background (with different foregrounds). (b) Different backgrounds (with one foreground). (c) Different backgrounds (with another foreground).

Figure 4.2: Distribution of the mean values of the first 7 ReLU layers in VGG-16.

4.3 Semantic Levels of Image Features for Foreground/Background

Convolution Neural Network (CNN) has proved the effectiveness in many tasks. Along with the depth, CNN extracts different semantic levels of image features in layers. Gatys et al. [9] pointed out that features in early layers reflect color or texture of images while features in latter layers convey foreground information. The work [63] also found that features in early layers address background while foreground is obtained in latter layers. As [110] learned the statistic of image features, we experimentally exploit semantic levels of image features in VGG-16 [30].

We randomly prepare 10 foreground images and 8 background ones. We then generated 100 images for each pair of foreground-background images with randomly localizing foreground (we have 8000 images in total). We feed these generated images into the VGG-16 [30] pre-trained on ImageNet dataset [80] without any fine-tuning to compute the mean activation at each Rectified Linear Unit (ReLU) layer [105].

The distribution of the mean activations in all 13 ReLU layers shows that the first 7 ReLU layers are more sensitive to background and foreground than the other ReLU layers. In the case where background is the same (Fig. 4.2a), the distribution of the mean values is small at the 1st – 3rd ReLU layers and becomes larger from the 4th ReLU layer. This suggests that VGG-16 recognizes the similarity of images at the 1st – 3rd ReLU layers and starts to learn differences of images from the 4th ReLU layer (though not strictly clear at the 4th layer).

On the other hand, in the case where backgrounds are different (Fig. 4.2b, Fig. 4.2c), the values at the 1st – 3rd ReLU layers are larger and similar with each other even if foregrounds are different (compared to the same background case). This observation is

in good harmony with the same background case. If we change foreground (Fig. 4.2b, Fig. 4.2c), the distribution of mean values is completely different from the layer 4th to the 7th layer (at each layer respectively).

Combining insights given by [9, 63], we may thus conclude that VGG-16 weights background in early layers and foreground at latter layers. More precisely, from the 1st to the 3rd ReLU layers capture background while from the 5th to the 7th ReLU layers do foreground, and the 4th ReLU layer seems to be in-between as a transition.

Using appropriate semantic levels of image features for discriminators is crucial. We use above observation for employing appropriate semantic levels of image features for foreground and background. Namely, we use features from the 1st to the 3rd ReLU layers for background and those from the 5th to the 7th ReLU layers for foreground. We remark that more deeply exploring background-foreground relation is preferable.

4.4 Proposed Method

4.4.1 Network design

Our network follows the GAN architecture [64] for image synthesis [13, 18, 45, 46]. Like [13], we condition GAN on both text descriptions and a source image. As seen in Section 4.3, we use different semantic levels of features depending on foreground and background. Namely, we design the network in which a text description on foreground matches features in latter layers while features of a source image in early layers are preserved as much background information as possible. This appropriate-level selection allows our model to synthesize realistic images that meet both a text description and a source image.

Nguyen et al. [109] argued that dual discriminators in GAN generate better images in quality than a single discriminator, though the two discriminators has the same architecture. To deal with foreground and background separately and more precisely, we employ a pair of discriminators where each of them independently judges foreground/background of synthesized images. For different semantic levels of foreground and background, we design our discriminators with different architectures and make each play a different role. Namely, we design one discriminator to evaluate matching foreground between a text description and a synthesized image following [13,

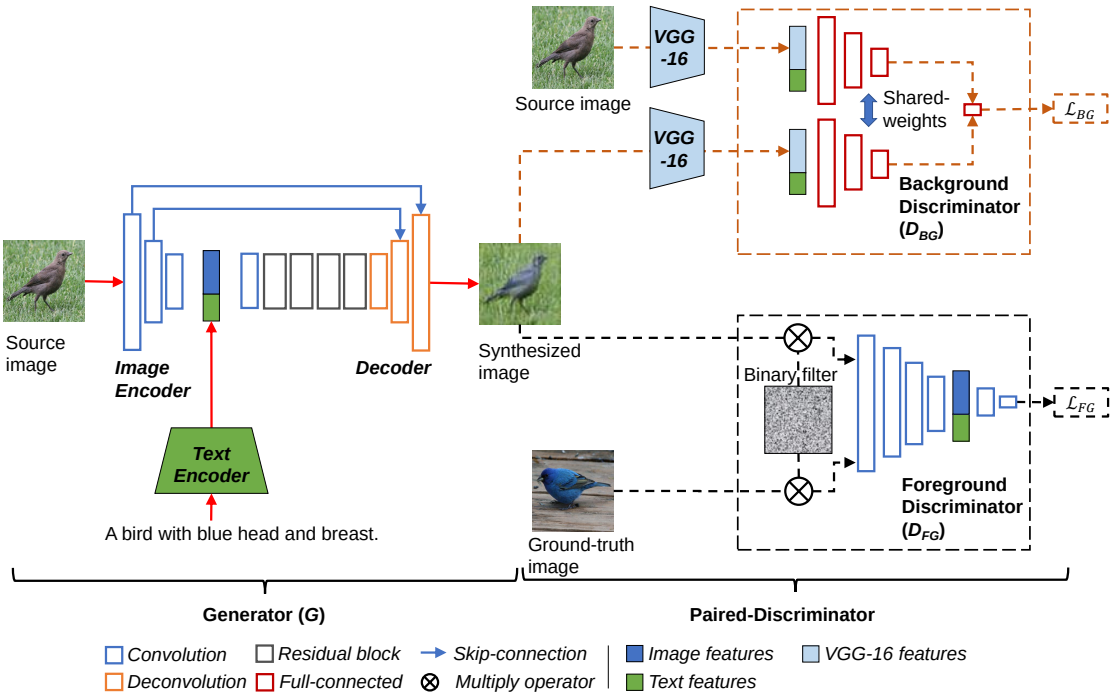


Figure 4.3: Framework of our proposed Paired-D GAN.

18, 45] and the other discriminator to evaluate whether background of a source image is retained in the synthesized image. We also introduce an effective training strategy for adversarial learning in a three-player minimax game.

4.4.2 Network architecture

We build our network, called Paired-D GAN, upon the GAN architecture with one generator G and a pair of discriminators, foreground discriminator D_{FG} and background discriminator D_{BG} (Fig. 5.2). We employ the end-to-end encoder-decoder architecture for our generator G following [13]. The generator G receives a source image and a text description where the source image is with the size of $n \times n \times 3$ (n can be 64, 128 or 256; 3 are for RGB channels) and the text description is with maximum of 50 words. G synthesizes an image of $n \times n \times 3$ that adaptively changes foreground to match the text description while retaining background of the source image.

Two discriminators D_{FG}, D_{BG} evaluate whether the synthesized image is real or generated. D_{FG} receives the generated image and the ground-truth foreground

image with the text feature extracted from the text description to focus on foreground evaluation. D_{BG}, on the other hand, receives the image feature extracted from the source image and the text feature extracted from the text description to focus on background evaluation. We use the pre-trained VGG-16 to extract image features from input images for D_{BG} as mentioned in Section 4.3. We remark that the two discriminators do not share their parameters.

We train G, D_{FG}, and D_{BG} simultaneously in a three-player minimax game using adaptive loss functions. This adversarial learning process enables our generator G to generate plausible images that mach text descriptions while preserving background information of the source image.

Generator

Our generator G consists of an image encoder, a text encoder, and a decoder.

The image encoder is a stack of three convolution layers that receives the source image size of $n \times n \times 3$ to produce an image feature with the size of $m \times m \times 512$ (m can be 16, 32 or 64 depending on n) at the top. We adopt the pre-trained text encoder [47] for our text encoder and use the text embedding augmentation [18] to produce a text feature with the size of 1×128. The channel of the text feature is duplicated to the size of $m \times m \times 128$ to be consistent with that of the image feature.

The image feature and the text feature are then concatenated to produce an image-text feature as the input of the decoder.

The decoder in our generator consists of one convolution layer, four residual blocks [31], and two deconvolution layers. The convolution layer reduces the channel of the image-text feature, and the four residual blocks enrich feature maps. The two deconvolution layers, on the other hand, upscale the feature maps.

We remark that each of the convolution and deconvolution layers in the image encoder and the decoder is followed by a batch normalization (BN) layer [71] and a ReLU layer. The only exception is the last deconvolution layer in the decoder where it uses the tanh activation to guarantee that the range of the output can be normalized to be $[0, 255]$ (in the test step). We remark that we use images with the range $[-1, 1]$ in the training step.

To reflect the features at early layers weighting background information into a

synthesized image, we employ the skip-connection from the image encoder to the decoder. More precisely, the first layer in the image encoder is connected to the last layer in the decoder while the second layer in the image encoder is paired with the second last layer in the decoder.

Foreground discriminator

The foreground discriminator should be able to discriminate foreground of real images and that of generated images. We employ the foreground-text matching in the foreground discriminator. Following previous work [13, 18, 45, 46], we design our foreground discriminator D_{FG} as a classification task that rewards high probability scores to real images and low ones to generated images in the adversarial learning phase.

Our D_{FG} is a stack of six convolution layers.

Each of the first four convolution layers uses the filter size of 4×4, the reflection-padding size of 1×1, and the stride size of 2×2, producing 64, 128, 256, 512 output channels, respectively. These convolution layers encode an input to produce the high-level semantic image features containing mostly foreground information (cf. Section 4.3). These image features are then concatenated with the text feature obtained from the input text description using the text encoder to produce a image-text feature.

Next, the image-text feature is fed into the last two convolution layers, each of which is with the filter size of 1×1, and 4×4, respectively, no padding, the stride size of 1×1, outputting 512, 4 channels respectively. The output of the last convolution layer indicates how realistic the image input to D_{FG} is using the similarity probability.

We remark that each of the all convolution layers except for the last one is followed by a BN layer and a ReLU layer. We follow Reed et al. [45] to train D_{FG} (Eq. 4.1).

D_{FG} need not access all image information but focuses on foreground image information. To enhance the performance of D_{FG}, we introduce a processing before feeding an input image to D_{FG}. Namely, we create a binary filter where 0 at each pixel is generated with the probability of p. We then apply the binary filter to the image input to D_{FG}, and feed the filtered image to D_{FG}. This processing brings two benefits: (1) D_{FG} has more chance to focus on only foreground information, helping to extract semantic image features of foreground, and (2) this operation prevents quick

convergence [111].

Background discriminator

The background discriminator evaluates how real and generated images are different in background. We therefore design the background discriminator as a verification task with the limited number of samples in each category. This is because each image in a dataset has different background in general, and the number of samples with the same (very similar) background is limited. To this end, we follow the idea of the Siamese network [112] because it shows the effectiveness for the verification task.

Our D_{BG} consists of four fully-connected layers in which the first three layers are two shared-parameter layers and the last one is the joint layer, producing 512, 100, 10, 1 outputs, respectively. D_{BG} receives two input features (one from the source image with the text description and the other from the generated image with the text description) and passes them to the two shared-parameter layers separately before being jointly trained at the last layer.

In order to create the input of D_{BG}, we feed the input image into the pre-trained VGG-16 to compute the mean and variance at the first four ReLU layers (cf. Section 4.3), and then concatenate them with the text feature extracted from the input text description using the pre-trained text encoder [47] (without using the text embedding augmentation [18]). The text feature is useful to disentangle background and foreground information (e.g. images with the same background and different foreground information can be positive samples for the background verification task). We remark that the size of the input is 1×1068 where the image feature is with the size of 1×768 and the text feature is with the size of 1×300.

We propose a new training strategy for D_{BG}, which is based on the contrastive loss function [112] that fully uses a source image and a text description.

4.4.3 Adversarial learning for Paired-GAN

Training the generator G, and a pair of discriminators D_{FG} and D_{BG} becomes a three-player minimax game conditioned on images and text descriptions. Using positive/negative training samples, we first update the parameters of D_{FG} with fixing the parameters of D_{BG} and G, and then update the parameters of D_{BG} with fixing

Table 4.1: Types of input pairs used in the adversarial leaning process.

	D_{FG}	D_{BG}
Positive	$\{g, \varphi(t)\}$	$\{(s, t), (s, \bar{t})\}$
Negative	$\{g, \varphi(\bar{t})\}$, $\{G(s, \varphi(t)), \varphi(t)\}$	$\{(G(s, \varphi(t)), t), (G(\bar{s}, \varphi(t)), t)\}$

the parameters of D_{FG} and G, and finally update the parameters of G with fixing the parameters of the two discriminators. We iterate this adversarial training to minimize each loss function separately.

For the adversarial training for Paired-D GAN, we use positive and negative samples whose definitions depend on D_{FG} and D_{BG}. A positive sample of D_{FG} is a sample in which foreground is the ground-truth and its text description is matching. A sample is negative if (1) foreground is the ground-truth but its text description is mismatching or (2) foreground is generated even if its text description is matching. A positive sample of D_{BG}, on the other hand, is the one where the background of the source image used in training the generator and discriminators for each iteration is the same regardless of whether text descriptions are matching or mismatching. A sample is negative if background is generated even if the text descriptions match foreground.

Let s be an image in a dataset and t be a text description. Then, we let g be an image in the dataset whose foreground is the ground-truth to t (t is thus a matching text description to g). We denote by $\bar{s}$ a randomly selected image (from the dataset) having different background from s, and by $\bar{t}$ a different text description from t (a mismatching text description to g). We also denote by $\varphi(\cdot)$ the text embedding augmentation [18]. Then, positive/negative samples of D_{FG} and D_{BG} can be classified as in Table 4.1.

Let $D(\cdot)$ denote the discriminators (D_{FG} and D_{BG}). At each iteration in training $D(\cdot)$, we randomly select all the types of samples in Table 4.1 from the training dataset, and feed them one by one to $D(\cdot)$ to obtain the probability whether the sample is positive or negative. We train the two discriminators to reward a high score to a positive sample and a low score to a negative sample. Through the training, we maximize the ability of $D(\cdot)$ to assign relevant scores to the samples. The loss functions

for $D(\cdot)$ are defined as follows:

$$\begin{aligned}\mathcal{L}_{\text{FG}} &= \mathbb{E}_{(g,t)\sim p_{\text{data}}}[\log D_{\text{FG}}(g,\varphi(t))] + \frac{1}{2}\mathbb{E}_{(g,\bar{t})\sim p_{\text{data}}}[\log(1 - D_{\text{FG}}(g,\varphi(\bar{t})))] \\ &\quad + \frac{1}{2}\mathbb{E}_{(s,t)\sim p_{\text{data}}}[\log(1 - D_{\text{FG}}(G(s,\varphi(t)),\varphi(t)))]. \qquad (4.1)\\ \mathcal{L}_{\text{BG}} &= \mathbb{E}_{(s,t,s,\bar{t})\sim p_{\text{data}}}[\log D_{\text{BG}}((s,t),(s,\bar{t}))] \\ &\quad + \mathbb{E}_{(s,t,\bar{s},t)\sim p_{\text{data}}}[\log(1 - D_{\text{BG}}((G(s,\varphi(t)),t),(G(\bar{s},\varphi(t)),t)))], \qquad (4.2)\end{aligned}$$

where p_{data} denotes the all the training data and $\mathbb{E}_{(\cdot)\sim p_{\text{data}}}$ means the expectation over p_{data}. Each term in Eqs. 4.1 and 4.2 corresponds to the type of samples: $\log(D(\cdot))$ for positive samples and $\log(1 - D(\cdot))$ for negative samples. Note that Eq. 4.1 follows [45].

Since our adversarial learning process is a three-player minimax game, we also train the generator G in which we minimize the terms of $\log(1 - D(\cdot))$ in Eqs. 4.1 and 4.2. In practice, however, maximizing $\log(D(\cdot))$ is known to be better than minimizing $\log(1 - D(\cdot))$ in training G [64]. We also introduce the reconstruction loss to keep the structure of the input source image. Now the loss function for G is:

$$\begin{aligned}\mathcal{L}_{\text{G}} &= \mathbb{E}_{(s,t)\sim p_{\text{data}}}[\log(D_{\text{FG}}(G(s,\varphi(t)),\varphi(t))] \\ &\quad + \mathbb{E}_{(s,t,s,\bar{t})\sim p_{\text{data}}}[\log(D_{\text{BG}}((G(s,\varphi(t)),t)),(G(s,\varphi(\bar{t})),\bar{t})))] \\ &\quad + \lambda\mathbb{E}_{(s,t)\sim p_{\text{data}}}\left\|s - G(s,\varphi(t))\right\|_2, \qquad (4.3)\end{aligned}$$

where λ is the hyperparameter, and $\|.\|_2$ is the Euclidean distance. To train G, we randomly select an image s, and two text descriptions t and $\bar{t}$ to generate the synthesized images. We then feed them to the D_{FG} and D_{BG} to receive feedback signals for updating parameters of G. We remark that since our aim is not to reconstruct the source image, λ can be small (we set $\lambda = 0.0001$ in our experiments).

As discussed in [45], training D_{FG} with match and mismatching text descriptions enables D_{FG} to feedback stronger image-text matching signals, allowing G to generate plausible images that match text descriptions. Our usage of a pair of image and a text description in training D_{BG}, on the other hand, enables D_{BG} to generate stronger signals as well, leading to the capability of G of retaining background information (though at the beginning, D_{BG} spends more time to verify background, D_{BG} gradually need not concern foreground thanks to text descriptions, and has ability of easily

judging whether the image is real or generated). Accordingly, the above adversarial learning brings to Paired-D GAN the capability of generating realistic images that match text descriptions in foreground and precisely retain background of source images.

4.5 Experimental Settings

4.5.1 Dataset and compared methods

Dataset. We used the Caltech-200 bird dataset [14] and the Oxford-102 flower dataset [15]. The Caltech-200 bird dataset contains 11,788 images belonging to one of 200 different bird classes. The Oxford-102 flower dataset has 8,189 images with 102 classes of the flower. Each image in the datasets has 10 captions collected by Reed et al. [47]. Following previous work [13, 45], we split the Caltech-200 dataset into 150 training classes and 50 testing classes, and the Oxford-102 dataset into 82 training classes and 20 testing classes. We remark that we resized the images used in our experiments to ones with 64×64.

Compared methods. We employed the model proposed by Dong+ [13] as the baseline. We also compared our method with Yang+ [62] that generates image foreground and background separately and recursively from input text descriptions (we chose this though the task is different because it generates realistic images). For Dong+ [13], we used the re-implementation by Seonghyeon [113] (as recommended by the authors of Dong+ [13]). For Yang+ [62], we used the publicly available source codes with the parameters recommended by the authors [114]. We remark that we used the combination of a noise vector and a text feature [45] as an input for Yang+ [62].

4.5.2 Evaluation metrics

We use the inception score (*IS*) [89] to evaluate the overall quality of synthesized images. We also use two metrics, foreground score (*FGS*) and background score (*BGS*) for evaluating foreground and background of synthesized images separately.

IS is widely used for the generative model evaluation through the output of the Inception-v3 network [91]: $IS(G) \approx \exp(\frac{1}{N}\sum_{i=1}^{N} D_{\mathrm{KL}}(p(y|\hat{x}^{(i)})||\hat{p}(y)))$, where $\hat{x}$ is a

synthesized image by the generator G, N is the number of generated images, D_{KL} is the Kullback–Leibler divergence, y indicates an instance of all classes given in the dataset, $p(y|\hat{x})$ is the conditional class distribution, and $\hat{p}(y) = \frac{1}{N}\sum_{i=1}^{N} p(y|\hat{x}^{(i)})$ is the empirical marginal-class distribution.

We employ the visual-text shared-space [47] and compute the matching between text descriptions and foreground for the foreground evaluation: $FGS = \left\| f_{\mathrm{img}} - f_{\mathrm{text}} \right\|_2$ where f_{img} and f_{text} are the features from the image encoder and the text encoder.

For background evaluation, we use $BGS = \left\| \hat{x} \odot \overline{x_{\mathrm{seg}}} - x \odot \overline{x_{\mathrm{seg}}} \right\|_2$ where x is the source image, and $\odot$ is the element-wise multiplication. $\overline{x_{\mathrm{seg}}}$ is the inverse map of x_{seg} where x_{seg} is the binary segmentation map of x provided from the dataset. We use $\overline{x_{\mathrm{seg}}}$ to mask foreground for x and $\hat{x}$.

4.5.3 Implementation and training details

We implemented our model in PyTorch. We adopted the pre-trained text encoder [47] without any fine-tuning. To extract image features for the background discriminator input, we employed the VGG-16 [30] pre-trained on ImageNet dataset [80] without any fine-tuning. Like [13], we also used the image augmentation technique (e.g., flipping, rotating, zooming and cropping). We conducted all the experiments using a PC with CPU 6-cores Xeon 3.7GHz, 64GB of RAM, and GTX1080 Titan GPU (11GB of VRAM).

We optimized the adaptive loss functions (Section 4.4.3) using Adam optimizer [67] with the learning rate of 2×10^{-3}, the learning rate decay of 0.5 performed every 100 epochs, the momentum $\beta_1 = 0.9$ and $\beta_2 = 0.999$, and the division from zero parameter $\epsilon = 10^{-8}$. We did not use the weight decay. We trained our model with the batch size of 48 for 600 epochs.

4.6 Experimental Results

4.6.1 Comparison with state-of-the-arts

Qualitative evaluation

Figures 4.4 and 4.5 illustrate some examples of the results obtained by our method (with $p = 0.8$) and Dong+ [13]. They show that the synthesized images by our method

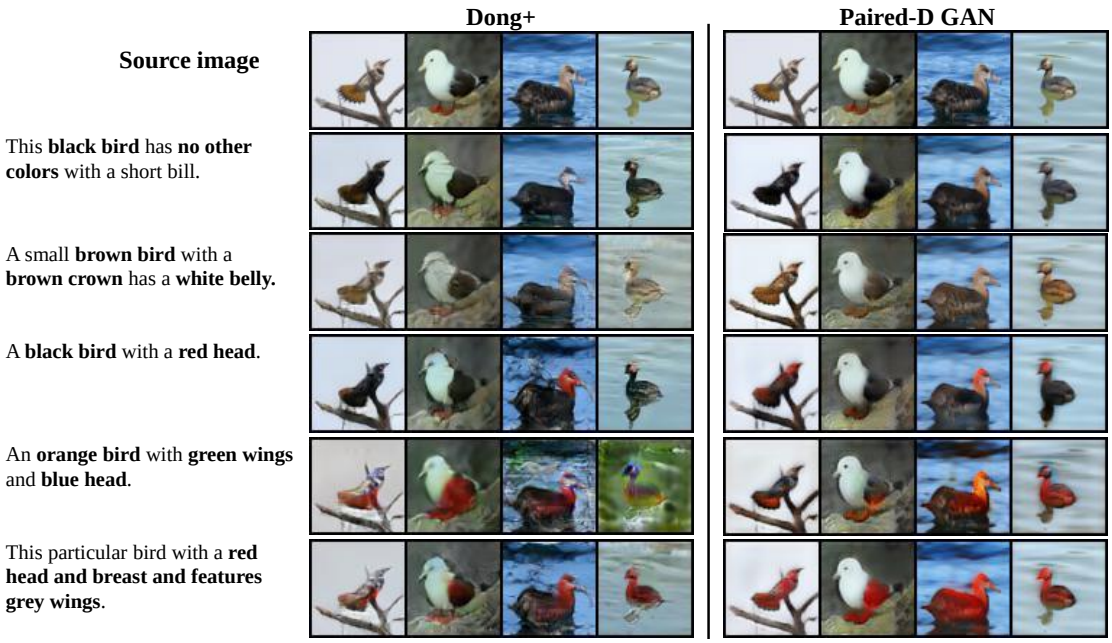


Figure 4.4: Visual comparison of our method against Dong+ [13] on the Caltech-200 bird dataset [14]. First row: source images, most left column: text descriptions. Each image is generated using a source image and a text description.

match the text descriptions more precisely than Dong+ [13] while successfully retaining background of the source images.

On the Caltech-200 dataset, we see that the results by our method are clearer in foreground and background with less noise than Dong+ [13] (Fig. 4.4). Though foreground of the results by Dong+ [13] also matches the text descriptions (not always though), we observe that background is not preserved well.

On the Oxford-102 dataset, on the other hand, we see that our method and Dong+ [13] both have some failures in synthesizing images (red rectangles in Fig. 4.5). This is because images in the dataset are too complex; for example, the detail of flowers such as a stamen is too small. Nevertheless, we still observe that our method outperforms Dong+ [13]. We note that Dong+ [13] generated different flowers from the source images (blue rectangles in Fig. 4.5).

Quantitative evaluation

For the quantitative evaluation, we computed *IS*, *FGS*, and *BGS* of the synthesized images, which are shown in Table 4.2. To compute *IS*, we iterated 10 times the

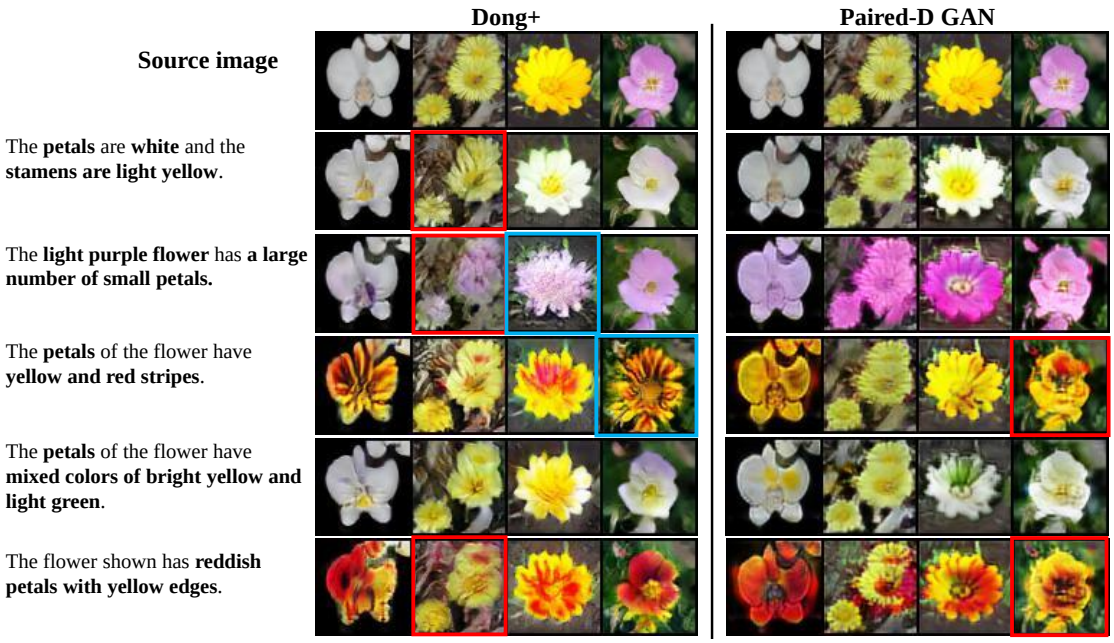


Figure 4.5: Visual comparison of our method against Dong+ [13] on Oxford-102 flower dataset [15]. First row: source image, most left column: text descriptions. Each image is generated using its source image and text. The red rectangles indicate the failure synthesized images in both Dong+ [13] and ours. The blue rectangles indicate the generated images different from their source images.

experiment that we synthesize 8000 images, and computed the average and the standard deviation of the resulting scores, as recommended in [89]. For *FGS* and *BGS*, we iterated 5 times the experiment that we synthesize 600 images, and computed the average and the standard deviation of the resulting scores. Note that we cannot compute *BGS* for Yang+ [62] because no ground-truths of background images exit for Yang+ [62]. We also remark that we used the visual-text shared-space model [47] pre-trained on the Caltech-200 (or Oxford-102) dataset to compute features for *FGS*.

Table 4.2 shows that our method achieves the best performances in all the metrics, meaning that the images synthesized by our method are superior not only in the overall quality (*IS*) but also in foreground-text matching (*FGS*) and in background preservation (*BGS*). The outperformance of our method against Dong+ [13] in all the metrics confirms that evaluating foreground and background separately in the training phase is effective. Compared to Yang+ [62], we see that our method and Dong+ [13] generate more realistic image, suggesting that for semantic image synthesis, generating

Table 4.2: Quantitative comparison using *IS* (larger is better), *FGS*, and *BGS* (smaller is better). The best results are given in blue.

Dataset	**Caltech-200** [14]			**Oxford-102** [15]		
Metric	*IS* ⇑	*FGS* ⇓	*BGS* ⇓	*IS* ⇑	*FGS* ⇓	*BGS* ⇓
Paired-D GAN	6.39±0.18	17.26±0.21	9.03±0.06	4.41±0.08	8.81±0.08	8.87±0.04
Dong+ [13]	5.56±0.14	18.60±0.09	11.83±0.06	4.03±0.11	9.71±0.11	9.47±0.14
Yang+ [62]	5.92±1.04	18.34±0.14	–	3.49±0.04	10.32±0.09	–

Table 4.3: Evaluation on the effectiveness of employing D_{BG}.

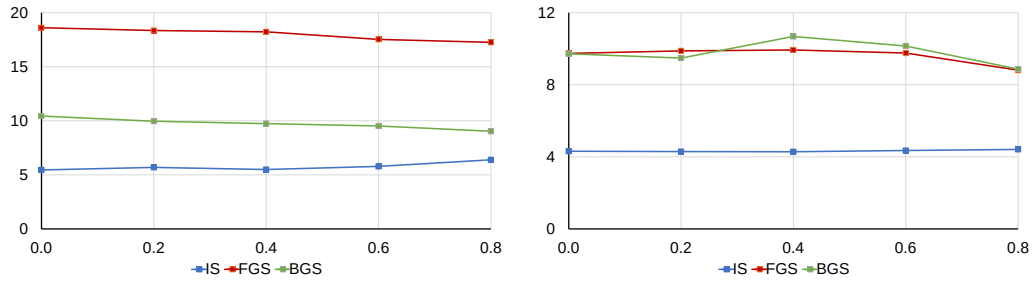
Dataset	**Caltech-200**			**Oxford-102**		
Metric	*IS* ⇑	*FGS* ⇓	*BGS* ⇓	*IS* ⇑	*FGS* ⇓	*BGS* ⇓
Complete model (D_{FG} + D_{BG})	6.39±0.18	17.26±0.21	9.03±0.06	4.41±0.08	8.81±0.08	8.87±0.04
Model with D_{FG} only	5.83±0.19	16.74±0.12	11.89±0.08	4.21±0.07	8.52±0.13	10.02±0.08
Model with D_{BG} only	6.02±0.15	20.33±0.11	7.63±0.08	4.24±0.10	10.68±0.14	8.32±0.15

foreground and background at the same time is better than separately and recursively generating foreground and background.

4.6.2 Ablation studies

First of all, we evaluated the effectiveness of employing D_{BG} through comparing our complete model with models using D_{FG} only or D_{BG} only. As shown in Table 4.3, the method using D_{FG} only achieves *FGS* best, and the method using D_{BG} only achieves *BGS* best. This means that the method using D_{FG} is correctly tuned to the foreground while the method using D_{BG} is correctly tuned to the background, and that D_{FG} and D_{BG} properly work for foreground and background each. Our completed method, on the other hand, balances foreground and background well as it achieves *IS* best.

Then, we evaluated the impact on the results by different p's (the probability of generating zero at each pixel) used in creating the binary filter for the foreground discriminator D_{FG}. We changed p by 0.2 from 0.0 (no mask) to 0.8 and computed *IS*, *BGS*, *FGS* at each p (Fig. 4.6). Visual comparison with different p's are illustrated in Fig. 4.7 (two examples with simple/complex background). Fig. 4.6 indicates that all the metrics become better at $p = 0.8$ (80% in probability of a source image are masked to focus on foreground). The explanation for this can be as follows, which is also



(a) Caltech-200 dataset. (b) Oxford-102 dataset.

Figure 4.6: Quantitative comparison by changing p by 0.2 from 0.0 to 0.8.

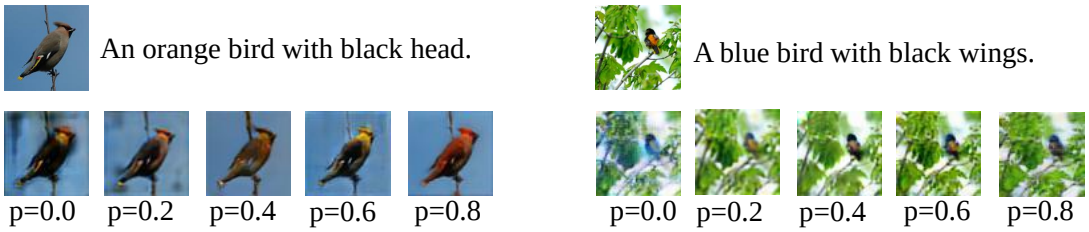


Figure 4.7: Zero-shot results by changing p by 0.2 from 0.0 to 0.8. The source image has simple background (right) or complex background (left).

supported by Fig. 4.7. When $p = 0.0$ (no mask), D_{FG} accesses the whole source image in the training phase, affecting background of generated images. By increasing p, D_{FG} is likely to focus on only foreground, leading to improving the quality of generated images. We note that background discriminator D_{BG} also succeeds in maintaining background of the source image (we can see that the background is kept well in most cases).

We next demonstrated the smooth interpolation between the source image and the

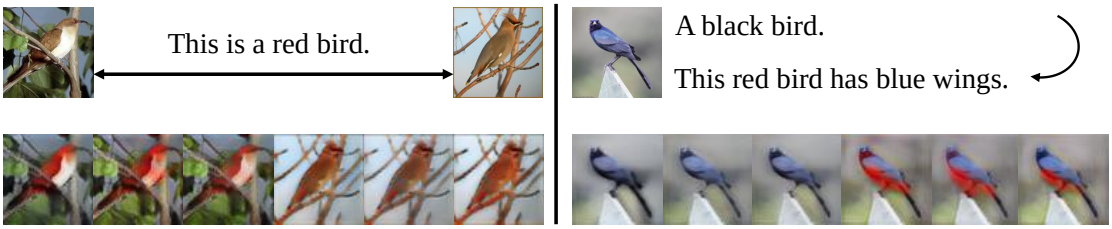


Figure 4.8: Zero-shot results of interpolation. Left: interpolation between two source images with the same target text description. Right: interpolation between two target text descriptions for the same source image.

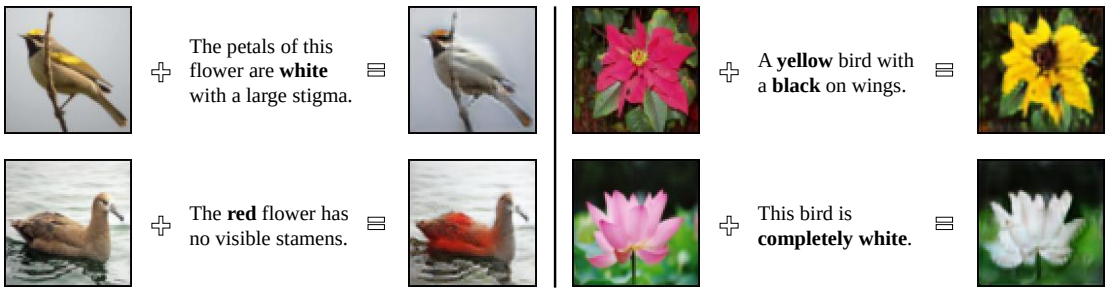


Figure 4.9: Zero-shot results from a source image and text descriptions that are not related to each other, showing the effectiveness of foreground and background discriminators.

Figure 4.10: Zero-shot results from the same source image and text descriptions, showing variety of foregrounds.

target image. Fig. 4.8 show synthesized images obtained by the linear interpolation between the source and the target images. In Fig. 4.8 (left), we interpolated two source images with a fixed text description. In contrast, we keep the source image fixed while changing text descriptions in Fig. 4.8 (right). These results indicate that our method is capable of independently interpolating between source images and text descriptions. We remark that our method preserve background well regardless of interpolation.

Figure 4.9 shows the generated images obtained using source images from the Caltech-200 [14] dataset with text descriptions from the Oxford-102 [15] dataset (not used in training phase), and vice verse. Fig. 4.9 shows that our model retains background of source images and changes only foreground to match text descriptions (e.g. color) even if they are not used in the training (regardless of untrained text descriptions). This illustrates the flexible capability of our model to disentangle foreground and background.

We also show in Fig. 4.10 the effectiveness of text embedding augmentation [18] in our method to synthesize various images using the same source image and text

descriptions.

4.7 Conclusion

We proposed Paired-D GAN conditioned on both text descriptions and images for image manipulation with text. Paired-D GAN consists of one generator and two discriminators with different architectures where one discriminator is used for judging foreground and the other is for judging background. Our method is able to synthesize a realistic image where an input text description matches its corresponding part (foreground) of the image while preserving background of a given source image. Experimental results on the Caltech-200 and the Oxford-102 datasets demonstrate the effectiveness of our method.

5 Comprehensive and Individual Usage of Relations for Text-to-Image Synthesis

5.1 Introduction

Generating photo-realistic images from text descriptions is one of the major problems in computer vision. Besides having a wide range of applications such as intelligent image manipulation, it drives research progress in multimodal learning and inference across vision and language [17, 115, 116].

The GANs [64] conditioned on unstructured text description [13, 18, 19, 45, 47] show remarkable results in text-to-image generation. Stacking such conditional GANs has shown even more ability of progressively rendering a more and more detailed entity in high resolution [18, 19]. However, in more complex scenarios where sentences are with many entities and relations, their performance is degraded. This is because they use only entity information in given text descriptions for rendering a specific entity, leading to a poor layout of multiple entities in generated images.

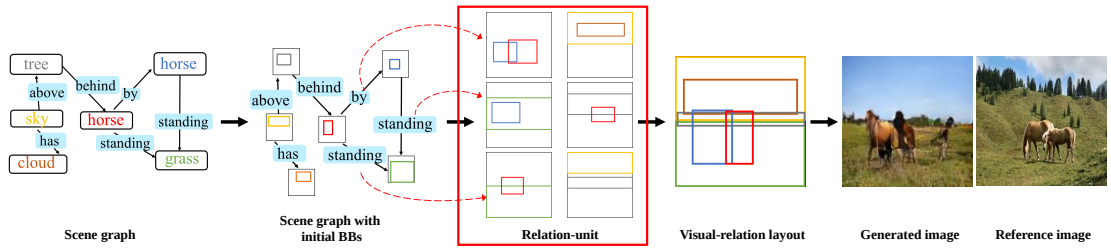


Figure 5.1: Overall framework of our proposed method. Given a structured-text (scene graph), our method firstly predicts initial bounding-boxes for entities using all available relations together, next takes individual relation one by one to infer a relation-unit for the relation, then unifies all the relation-units to produce visual-relation layout. Finally, the visual-relation layout is converted to image (256×256). Color of each entity bounding-box corresponds to that in the scene graph. Dotted arrow in red illustrates the individual usage of relations.

In the presence of multiple entities, besides the details of each entity, how to localize all the entities so that they reflect given relations becomes crucial for better image generation. Indeed, recent work [17, 20, 22, 48] show the effectiveness of inferring the scene layout first from given text descriptions. Johnson+[17], Li+[48], and Ashual+[20] use structured-text, i.e., scene graphs [115], first to construct a scene layout by predicting bounding boxes and segmentation masks for all entities, then convert it to an image. Hong+[22] constructs a semantic layout, a scene structure based on object instances, from input text descriptions and converts the layout into an image. However, those mentioned methods [17, 20, 22, 48] aggregate all relations in which each entity is involved, and then localize all entities' bounding-boxes at the same time. As a result, the predicted bounding-boxes do not preserve the relations among entities well. Localizing entities faithfully by preserving their relations given in text descriptions is desired.

We leverage advantages of the pyramid of GANs and inferring the scene layout, proposing a GAN-based model for text-to-image generation where our network steps further in relation usage by employing not only all available relations together but also individual relation separately. We refer the former usage of relations as *comprehensive* while the latter as *individual*. Our network has two steps: (1) inferring from input the *visual-relation layout*, i.e., localized bounding-boxes for all the entities so that each of which uniquely corresponds to each entity and faithfully preserves relations

between the entities, and (2) progressively generating coarse-to-fine images with the pyramid of GANs, namely stacking-GANs, conditioned on the visual-relation layout. The first step takes the comprehensive usage of relations first to generate initial bounding-boxes for entities as in [17, 20, 22, 48], and then takes the individual usage to predict a relation-unit for each *subject–predicate–object* relation where all the relations in the input are extracted through its scene graph [115]. Each relation-unit consists of *two* bounding-boxes that participate in the relation: one for a "subject" entity and one for an "object" entity. Since one entity may participate in multiple relations, we then unify all the relation-units into refined (entity) bounding-boxes (including their locations and sizes) so that each of them uniquely corresponds to one entity while keeping their relations in the input text. Aggregating the refined bounding-boxes allows us to infer the visual-relation layout reflecting the scene structure given in the text. In the second step, three GANs progressively generate images where entities are rendered in more and more details while preserving the scene structure. At each level, a GAN is conditioned on the visual-relation layout and the output of previous GAN. Our network is trained in a fully end-to-end fashion.

The main contribution of our proposed method is our introduction to the individual usage of *subject–predicate–object* relations for localizing entity bounding-boxes, so that our proposed *visual-relation layout* surely preserves the visual relations among entities. In addition, we stack and condition GANs on the visual-relation layout to progressively render realistic detailed entities that keep their relations even from complex text descriptions. Experimental results on two public datasets (COCO-stuff [16] and GENOME [21]) demonstrate outperformances of our method against state-of-the-arts. Fig. 5.1 shows the overall framework of our proposed method.

5.2 Related Work

Recent conditional GAN-based methods have shown promising results on text-to-image generation [13, 17–19, 22, 45, 101]. They, however, struggle to faithfully reproduce complex sentences with many entities and relations because of the gap between text and image representations.

To overcome the limitation of GANs conditioned on text descriptions, a two-step approach was proposed where inference of the scene layout as an intermediate

representation between text and image is followed by using the layout to generate images [17, 20, 22, 48]. Since the gap between the intermediate representation and image is smaller than that of text and image, this approach generates more realistic images. Zhao+[117] and Sun+[118] propose a combination of ground-truth layout and entity embeddings to generate images. Hong+[22] infers a scene layout by feeding text descriptions into a LSTM. More precisely, they use a LSTM to predict bounding-boxes for all entities independently, then employ a bi-directional conv-LSTM to generate entity shapes from each predicted bounding-box without using any relation. The function of the bi-directional conv-LSTM used here is just the putting-together. They then combine the layout with text embeddings obtained from the pre-trained text encoder [47], and use a cascade refinement network (CRN) [41] for generating images.

Johnson+[17], Li+[48], and Ashual+[20] employ a scene graph [115] to predict a scene layout and then condition CRN [41] on the layout. The graph convolution network (GCN) used in these methods aggregates available relations of all the entities together along the edges of the scene graph. Namely, only the comprehensive usage of relations is employed. As a result, individual relation information is lost at the end of GCN because of the averaging operation on entity embeddings. Averaging entity embeddings means mixing different relations in which a single entity is involved, resulting in failure of retaining individual relation information. Different from [17], [48] retrieves entity appearances from a pre-defined tank while [20] adds entity appearances to the scene layout before feeding it to the generation part. The layout in [17, 20, 22, 48] is constructed through only the comprehensive usage of relation among entities for bounding-boxes' localization. As a result, their generated images may have poor scene structure as a whole even if each entity is realistically rendered.

Our main difference from the aforementioned methods is to construct the visual-relation layout using *subject–predicate–object* relations between entities extracted from an input structured-text not only comprehensively but also individually. Recursively conditioning stacking-GANs on our constructed visual-relation layout enables us to progressively generate coarse-to-fine images that consistently preserve the scene structure given in texts.

Table 5.1: Scene structure preservation in image layout by different usage of relations on COCO-stuff [16] using $R@\tau$, $rIoU$, and RS (larger is better).

	$R@0.3$	$rIoU$	RS
Comprehensive usage	51.20	0.2532	59.75
Individual usage	30.47	0.1109	29.18
Comprehensive usage + Individual usage	75.79	0.2918	67.48
Individual usage + Comprehensive usage	41.38	0.1837	48.17

5.3 Scene Structure Preservation by Different Usage of Relations

The two ways usage of relations leads to different ways to predict image layout. We therefore experimentally exploit the degree of retaining relations in the constructed layout using different usage of relations. More precisely, we investigate four ways: (i) comprehensive usage only, (ii) individual usage only, (iii) combination of comprehensive and individual usage, and (iv) combination of individual and comprehensive usage.

For comprehensive usage only, we employ the graph convolution network used in [17] to predict a bounding-box for each entity in the scene graph.

For individual usage only, we first train a bounding-box regression using ground-truth bounding-boxes. We build the regression upon two fully-connected layers followed by a ReLU layer [69] outputting 512 and 8 outputs. For each subject–predicate–object relation of scene graph, we take its corresponding subject, predicate, object embeddings and two random initial bounding-boxes with the size of $1\times(2\times|C|+|\mathcal{R}|+8)$. We note that either subject or object embedding is with the size of $1 \times |C|$, predicate embedding is with the size $1 \times |\mathcal{R}|$ whereas each initial bounding-box is with the size of 1×4. We also remark that C and $\mathcal{R}$ are defined as the set of categories and the set of relations given in a dataset, and a learned embedding layer is used to produce subject/predicate/object embeddings. We refer Section 5.4 for more details. We then feed the combination of embeddings and initial bounding-boxes into the bounding-box regression to predict two bounding-boxes for entities (e.g., subject and object) that involve in the relation. In testing time, since each entity may participate

multiple relations, we further unify its corresponding bounding-boxes into a single bounding-box by averaging all the bounding-boxes correspond to one entity. By this way, we are able to predict bounding-box for each entity using individual usage of relations only.

For combination of comprehensive and individual usage, we first employ graph convolution network to aggregate all the relations in scene graph to infer (initial) bounding-box for each entity. We then use each relation separately to modify the location of initial bounding-boxes. To this end, for each relation, we identify the two initial bounding-boxes that involve in the relation. If they do not satisfy the relation, we fix the location of the "subject" initial bounding-box and then move the location of the "object" initial bounding-box to meet the relation. After modifying all the initial bounding-boxes according to all the relations, we also unify all the bounding-boxes corresponding to a single entity.

For combination of individual and comprehensive usage, we first use pre-trained bounding-box regression (i.e., individual usage only) to predict entities' bounding-boxes. Then, all the bounding-boxes along with entity/predicate embeddings are fed into the graph convolution network to produce final bounding-boxes.

We conducted four different ways to predict image layout on COCO-stuff dataset [16]. This is because the dataset has geometrical relations only which is easy to modify location of bounding-box when we use individual usage of relation. Table 5.1 shows the scene structure preservation by different usage of relations. We note that the details of evaluation metrics are presented in Section 5.5.2. We see that the combination of comprehensive and individual usage preserves the scene structure much better than others. We may conclude that the comprehensive usage followed by individual usage is promising because it is reasonable to initialize all entities' locations first and then to adjust them to meet their relations. These observations lead to our idea to design a sequential network to predict image layout. In particular, We first use comprehensive usage to initialize locations for all entities appearing in scene graph, and then use individual usage to adjust the initial locations to meet all relations. By this way, our network can predict image layout more precisely, resulting in better generated image. Next section will describe our proposed method in more details.

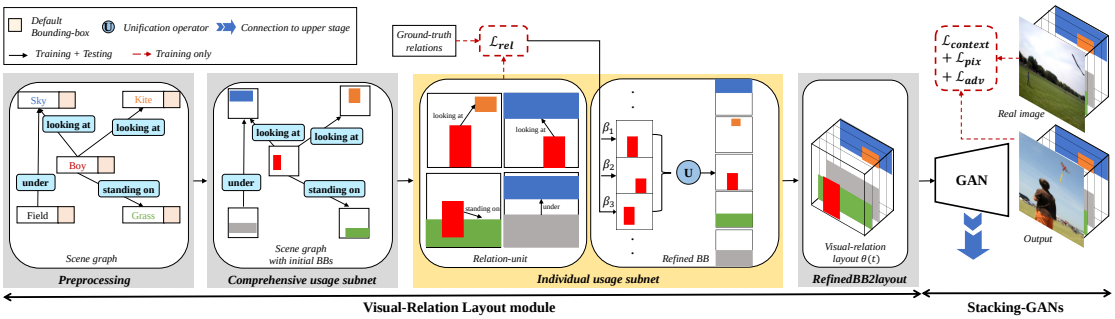


Figure 5.2: Our proposed network model consisting of the Visual-relation layout module and the Stacking-GANs.

5.4 Proposed Method

Our method is decomposed into two steps: (1) inferring the visual-relation layout $\theta(t)$ from structured-text description t, and (2) generating an image from the visual-relation layout, namely $\hat{I} = G(\theta(t))$. To this end, we design an end-to-end network with two modules: the visual-relation layout module and the stacking-GANs (Fig. 5.2). We train the network in a fully end-to-end manner.

5.4.1 Visual-relation layout module

The visual-relation layout module constructs the visual-relation layout $\theta(t)$ from a given structured-text description t (Fig. 5.3) where t is assumed to be converted into a scene graph [115], i.e., the collection of *subject–predicate–object*'s. After the pre-processing on converting t to its scene graph, the comprehensive usage subnet in this module predicts initial bounding-boxes (BBs) for all the entities in t by aggregating all available relations together through GCN ("comprehensive usage"). The individual usage subnet takes each *subject–predicate–object* relation from the scene graph one by one and select the pair of initial BBs involved in the relation (predicate): one for "subject" entity and one for "object" entity. The subnet then adjusts the location and size of the pair of initial BBs using the relation ("individual usage") to have a relation-unit for the relation. Since one entity may participate in multiple relations, it next unifies relation-units so that each entity uniquely has a single BB (called refined BB) that is further adjusted in location and size using weights learned from all the participating relations. The RefinedBB2layout subnet constructs the visual-relation

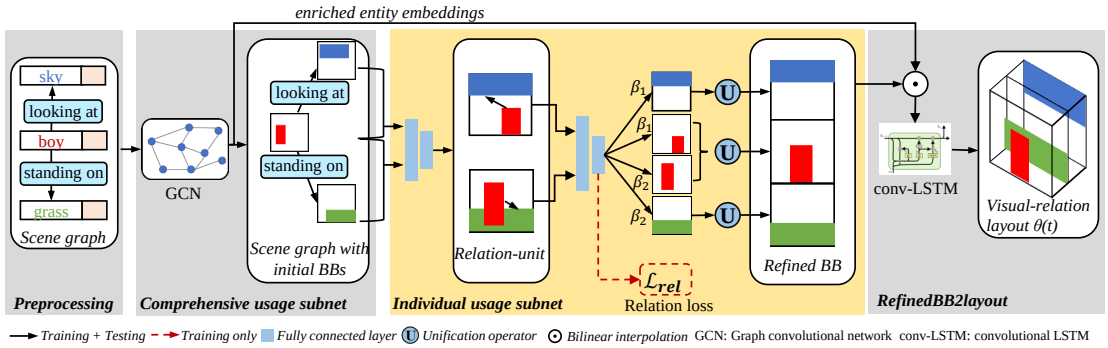


Figure 5.3: Details of visual-relation layout module. This figure illustrates the prediction for two *subject–predicate–object* relations.

layout by aggregating all the refined BBs together using conv-LSTM.

Preprocessing. Similar to [17], we convert the structured-text t to its scene graph (E, P) where $E \subseteq C$ and $P \subseteq C \times \mathcal{R} \times C$. C and $\mathcal{R}$ are the set of categories and the set of relations given in a dataset. An edge of (E, P) is associated with one *subject–predicate–object*. It is directed and represented by $(e^{\mathrm{s}}, p, e^{\mathrm{o}})$ with entities $e^{\mathrm{s}}, e^{\mathrm{o}} \in E$ and predicate $p \in \mathcal{R}$ (s and o indicate subject and object).

Like [17], we employ a learned embedding layer to produce the entity embedding with the size of $1 \times |C|$ and the predicate embedding with the size of $1 \times |\mathcal{R}|$ for any of all the entities and predicates appearing in the scene graph (E, P). Any entity embedding is associated with a single default BB presented by $[x, y, w, h] \in [0, 1]^4$ where x is the *left coordinate*, y is the *top coordinate*, w is the *width*, and h is the *height*. We set $x = y = 0$ and $w = h = 1$ as default. This process ensures that all the entities appear in the image. In the implementation, we concatenate the default BB and its associated entity embedding to produce the vector with the size of $1 \times (|C| + 4)$.

Comprehensive usage subnet. This subnet applies the comprehensive usage to predict a single initial BB for each entity appearing in t as in [17, 20, 22, 48]. This subnet gives us initial locations and sizes of entities and they do not necessarily satisfy the relations given in t.

In order to aggregate all information along the edges in the scene graph, we employ GCN [17]. Our GCN composed of five graph convolution layers is designed following the idea in [17] with a modification that produces 388 outputs instead of 384 not only to enrich entity/predicate embeddings as in [17, 20, 48] but also to infer initial BBs.

We do not use the average pooling layer on top of GCN to retain individual relation information.

For each edge k of scene graph (E, P), the triplet of its embeddings $(\boldsymbol{e}_k^{\mathrm{s}}, \boldsymbol{p}_k, \boldsymbol{e}_k^{\mathrm{o}})$ and two default BBs with the size of $1 \times (|C| + |\mathcal{R}| + |C| + 8)$ are processed to give enriched subject $\boldsymbol{e}_k'^{\mathrm{s}}$, predicate $\boldsymbol{p}_k'$, and object $\boldsymbol{e}_k'^{\mathrm{o}}$ embeddings with the size of 1×128 each, separately, and a pair of initial BBs (one for "subject" and one for "object") with the size of 1×4 each.

Individual usage subnet. Since the initial BBs of the entites do not always satisfy the relations given in t, we adjust their locations and sizes using each relation separately. For each relation, we select a pair of initial BBs corresponding to the "subject" and "object" involved in the relation, and adjust the locations and sizes of the pair of BBs using the relation to have a relation-unit for the relation where a relation-unit consists of *two* BBs for "subject" and "object" entities in the relation. This process causes the situation where multiple BBs correspond to the same entity, as different relations may involve same entities in common. We thus move to focus on each entity to unify its corresponding BBs into a single BB (called refined BB) where we use weights learned to retain all the relations. Accordingly, the function of this subnet is two-folds: relation-unit prediction using individual relation separately and unification of multiple BBs corresponding to the same entity into a single refined BB. The subnet is built upon two fully-connected layers followed by a ReLU layer [69] producing 512 and 8 outputs.

For each edge k of scene graph (E, P), its enriched embeddings and its corresponding pair of initial BBs with the size of $1 \times 392 (= 128 + 4 + 128 + 128 + 4)$ are fed into this subnet to infer relation-unit $(\boldsymbol{b}_k^{\mathrm{s}}, \boldsymbol{b}_k^{\mathrm{o}})$ with the size of 1×8. Each BB ($\boldsymbol{b}_k^{\mathrm{s}}$ or $\boldsymbol{b}_k^{\mathrm{o}}$) in the relation-unit is associated with enriched embedding either $\boldsymbol{e}_k'^{\mathrm{s}}$ or $\boldsymbol{e}_k'^{\mathrm{o}}$, respectively for "subject" or "object". Since the number of relation-units is $|P|$, the total number of obtained BBs is $|\{\boldsymbol{b}_k^{\mathrm{s}}, \boldsymbol{b}_k^{\mathrm{o}}\}| = 2 \times |P|$, which is in general larger than $|E|$.

To encourage the refined BB of each entity to keep its involved relations, we use the relation loss $\mathcal{L}_{rel}$ (see Section 5.4.3 for details) in a supervised manner. This is because the relation loss indicates the degree of retaining the involved relations in terms of relation-unit.

For entity $e_i \in E$, let $\boldsymbol{B}_i = \{\boldsymbol{B}_{iv}\}$ denote the set of its corresponding BBs (appearing in different relation-units) and $\beta_i = \{\beta_{iv}\}$ be the set of their weights. We define the

refined BB $\hat{\boldsymbol{B}}_i$ of entity e_i as the weighted sum:

$$\hat{\boldsymbol{B}}_i = \frac{\sum_{v=1}^{|\boldsymbol{B}_i|}\{(1+\beta_{iv}) \times \boldsymbol{B}_{iv}\}}{\sum_{v=1}^{|\boldsymbol{B}_i|}(1+\beta_{iv})} \tag{5.1}$$

Each weight in β_i is obtained from the outputs of the softmax function in the relation auxiliary classifier using the relation loss $\mathcal{L}_{\text{rel}}$.

At the beginning of training, relation-units cannot exactly reproduce their involved relations. Their weights thus tend to be close to *zero*, leading Eq. (5.1) almost similar to the simple average. Our refined BBs may be close to those of [17, 20, 48] at the beginning of training yet they keep their relations thanks to their weights.

As training proceeds, the contribution of the relation-units retaining relations consistent with text t to the refined BB gradually increases. As a result, the location and size of the refined BB are continuously altered to keep relations consistent with t.

For entity e_i, its embeddings that are associated with $\{\boldsymbol{B}_{iv}\}$'s over v are averaged. In this way, we obtain the set of refined BBs $\{\hat{\boldsymbol{B}}_i\}$ and their associated embeddings for all the entities in E. We remark that $|\{\hat{\boldsymbol{B}}_i\}| = |E|$.

If all the initial BBs completely keep their relations, the individual usage subnet works as the averaging operator as in [17, 20, 48] and our visual-relation layout is similar to the layout by [17, 20, 48]. In practice, however, the comprehensive usage of relations cannot guarantee to completely keep the relations. Our individual usage subnet plays the role of adjusting all the BBs in location and size to keep their relations as much as possible using each relation separately.

RefinedBB2layout subnet. In order to construct the visual-relation layout, we aggregate all the refined BBs and transfer them from the bounding-box domain to the image domain. This process should meet two requirements: (i) each entity in the image should be localized and resized to match its individual refined BB, and (ii) each entity should appear in the image even if some refined BBs overlap with each other. To this end, we design *refinedBB2layout* subnet as a learnable network rather than the putting-together operation. We build this subnet using a conv-LSTM [119] with the 5 hidden states each outputting 128 channels.

For $\hat{\boldsymbol{B}}_i$ of entity e_i, we first convert it to the binary mask with the size of $64 \times 64 \times 128$ whose element is 1 if and only if it is contained in $\hat{\boldsymbol{B}}_i$, 0 otherwise. Then, we reshape its

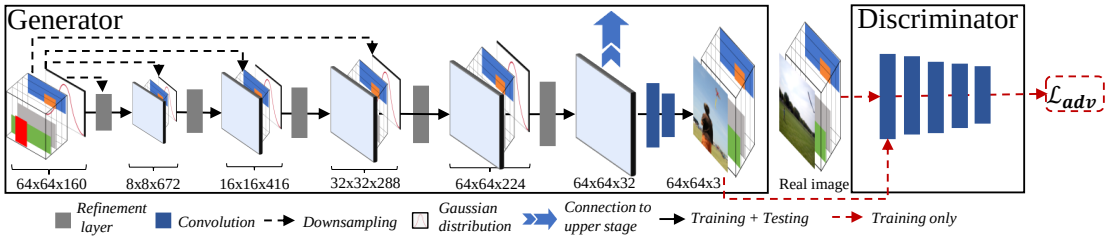


Figure 5.4: Details of one GAN in the stacking-GANs. Illustrated is the first GAN which receives the visual-relation layout ($64 \times 64 \times 128$) and Gaussian distribution noise ($64 \times 64 \times 32$) as its inputs. The second (the third) GAN receives the upsampled visual-relation layout and the hidden features of previous GAN.

associated embedding from 1×128 to $1 \times 1 \times 128$. Finally, the reshaped embedding is wraped to $\hat{\boldsymbol{B}}_i$ using the bilinear interpolation [120] for the layout of entity e_i ($64 \times 64 \times 128$). To produce $\theta(t)$, we feed the sequence of entity layouts into the *refinedBB2layout* subnet. The size of $\theta(t)$ is $64 \times 64 \times 128$.

5.4.2 Stacking-GANs

We employ stacking-GANs to progressively generate coarse-to-fine images. It consists of three GANs, each of which is identical to CRN [41] to generate images with the size of $n \times n \times 3$ (n = 64, 128, 256) (Fig. 5.4). Parameters are not shared by any GANs.

The first GAN generator receives the visual-relation layout $\theta(t)$ and a standard Gaussian distribution noise as input while the others receive the bilinear upsampled [120] layout $\theta(t)$ and the output of the last refinement layer from the previous GAN. The discriminators receive an image-layout pair as their inputs. Each pair is either a real sample or a fake sample. A real sample consists of a real image and a real layout while a fake sample consists of a predicted layout and a generated or real image. These samples not only encourage the GAN to improve the quality of generated images but also give the helpful feedback to the visual-relation layout module.

Generator. Our generator consists of five refinement layers [41] producing 512, 256, 128, 64, 32 outputs and two convolution layers outputting 32 and 3 channels.

We concatenate $\theta(t)$ and Gaussian noise (for the first generator) or the output of the last refinement layer of the previous generator (for the second and the third generators) to produce the input of $l \times l \times 160$ (l = 64, 128, 256). The size of the input is

upsampled using the bilinear interpolation [120] to be consistent with that of the generator input. At each level of the refinement layers of each generator, the generator input is downsampled and concatenated with the output of the previous refinement layer (upsampled using the bilinear interpolation) to produce the input (except for the first refinement layer that receives the generator input only).

Discriminator. Following [18], we design our discriminator as the classification task rewarding high probability for real images and low one for generated images. Our discriminator consists of five convolution layers, outputting 64, 128, 256, 512, and 4 channels, respectively.

5.4.3 Loss function

We jointly train our network in an end-to-end manner to minimize the weighted sum of four losses.

Relation loss $\mathcal{L}_{\text{rel}}$ is computed using cross entropy between relation-units and their ground-truth relations that is obtained through the relation auxiliary classifier which is built upon two fully-connected layers producing 512 and $|\mathcal{R}|$ outputs. The first fully-connected layer is followed by a ReLU layer while the second one ends with the $softmax$ function.

For each edge k of (E, P), its relation-unit and involved embeddings, i.e., $\boldsymbol{e}'^{\text{s}}_k$, $\boldsymbol{b}^{\text{s}}_k$, $\boldsymbol{e}'^{\text{o}}_k$, and $\boldsymbol{b}^{\text{o}}_k$, are concatenated in this order to have an input vector of 1×264. We then feed this vector into the relation auxiliary classifier to obtain the probability distribution $\boldsymbol{w}_k$ of the relations over $\mathcal{R}$. $\boldsymbol{w}_k$ is a vector of $1 \times |\mathcal{R}|$ and contains all the predicates $p_k \in \mathcal{R}$. We first obtain the *index* of predicate $p_k \in \mathcal{R}$. Since the order of predicates in $\boldsymbol{w}_k$ is the same as that in $\mathcal{R}$, the value at *index* in $\boldsymbol{w}_k$ is the weight of p_k, which is used as the weight of the relation-unit $(\boldsymbol{b}^{\text{s}}_k, \boldsymbol{b}^{\text{o}}_k)$ in the individual usage subnet. Note that the weight of a relation-unit is used for the weight of both $\boldsymbol{b}^{\text{s}}_k$ and $\boldsymbol{b}^{\text{o}}_k$ involved in the relation-unit.

The relation loss is defined as: $\mathcal{L}_{\text{rel}} = -\sum_{k=1}^{|P|}\sum_{\nu'=1}^{|\mathcal{R}|} \boldsymbol{p}_k[\nu'] \log(\boldsymbol{w}_k[\nu'])$. Minimizing the relation loss encourages relation-units to adjust their locations and sizes to meet the "predicate" relation. This is because the relation reflects the relative spatial locations among its associated relation-units.

Pixel loss: $\mathcal{L}_{\text{pix}} = ||I - \hat{I}||_2$, where I is the ground-truth image and $\hat{I}$ is a generated image. The $\mathcal{L}_{\text{pix}}$ is useful for keeping the quality of generated images.

Contextual loss [121]: $\mathcal{L}_{\text{context}} = -\log(CX(\Phi^l(I), \Phi^l(\hat{I})))$, where $\Phi^l(\cdot)$ denotes the feature map extracted from layer l of perceptual network Φ, and $CX(\cdot)$ is the function that computes the similarity between image features. $\mathcal{L}_{\text{context}}$ is used to learn the context of an image since refined BBs may lose the context such as missing pixel information or the size of entity.

Adversarial loss [64] $\mathcal{L}_{\text{adv}}$ is used to encourage the stacking-GANs to generate realistic images. Since the discriminator also receives the real/predicted visual-relation layout as its input, the $\mathcal{L}_{\text{adv}}$ is helpful in training the visual-relation layout module as well.

In summary, our loss function is defined as

$$\mathcal{L} = \lambda_1 \mathcal{L}_{\text{rel}} + \lambda_2 \mathcal{L}_{\text{pix}} + \lambda_3 \mathcal{L}_{\text{context}} + \sum_{i=1}^{3} \lambda_4 \mathcal{L}_{\text{adv}i}, \tag{5.2}$$

where λ_i are hyper-parameters. Note that we compute $\mathcal{L}_{\text{adv}}$ at each level in the stacking-GANs, while $\mathcal{L}_{\text{pix}}$ and $\mathcal{L}_{\text{context}}$ are computed at the last level in the stacking-GANs.

5.5 Experimental Settings

5.5.1 Dataset and compared methods

Dataset. We evaluated our method on challenging 2017 COCO-stuff dataset [16] and Visual GENOME dataset [21], which have complex descriptions with many entities and relations in diverse context. Note that we followed [17] to pre-process all the datasets: $|C| = 171$ and $|\mathcal{R}| = 6$ on COCO-stuff dataset [16], and $|C| = 178$ and $|\mathcal{R}| = 45$ on GENOME dataset [21].

Compared methods. We employed Johnson+[17] as the baseline (64×64). To factor out the influence of image generator, we replaced the CRN in [17] by our stacking-GANs to produce higher resolution images (128×128 and 256×256). We also compared our method with Hong+[22], Zhang+[18], Xu+[19], Li+[48], Ashual+[20], Zhao+[117], and Sun+[118]. We reported the results in the original papers whenever possible. For the methods that released at least one reference pre-trained model ([60] and [122]), we trained authors' provided codes (Zhang+[18] and Xu+[19]) on GENOME dataset.

5.5.2 Evaluation metrics

We use the inception score (IS) [89] and Fréchet inception distance (FID) [90] to evaluate the overall quality of generated images (we used the implementation in [123, 124]). We also use four metrics to evaluate the visual-relation layout: the entity recall at IoU threshold ($R@\tau$), the relation IoU ($rIoU$), the relation score (RS), and the BB coverage. Furthermore, to evaluate the relevance of generated images and input text descriptions, we use the image caption metrics: *BLEU* [125], *METEOR* [126], and *CIDEr* [127]. In addition, to evaluate the diversity of generated images, we use the diversity score [23] (implemented in [128]).

To evaluate how much the predicted layout is consistent with the ground-truth, we measure the agreement in size and location between predicted (i.e., refined) and ground-truth BBs using the entity recall at IoU threshold: $R@\tau = |\{i \mid IoU(\hat{\boldsymbol{B}}_i, \boldsymbol{GT}_i) \geq \tau\}|/N$, where $\hat{\boldsymbol{B}}_i$ and $\boldsymbol{GT}_i$ are predicted and ground-truth BBs for entity e_i, $N = \min(|\{\hat{\boldsymbol{B}}_i\}|, |\{\boldsymbol{GT}_i\}|)$ (we always observed $|\{\hat{\boldsymbol{B}}_i\}| = |\{\boldsymbol{GT}_i\}|$), τ is a IoU threshold, and $IoU(\cdot)$ denotes Intersection-over-Union metric. Note that we used only the BBs that exist in both $\{\hat{\boldsymbol{B}}_i\}$ and $\{\boldsymbol{GT}_i\}$ to compute $R@\tau$.

We also evaluate the predicted layout using *subject–predicate–object* relations. For each *subject–predicate–object* relation, we computed the IoU of the predicted "subject" BB and its corresponding ground-truth, and that for the "object". We then multiplied the two IoUs to obtain the IoU for the relation. $rIoU$ is the average over all the *subject–predicate–object* relations.

We use the relation score (RS) [129] for COCO-stuff to evaluate the compliance of geometrical relation between predicted BBs. For each edge k of scene graph (E, P), we define $score(\hat{\boldsymbol{B}}^{\mathrm{s}}_k, \hat{\boldsymbol{B}}^{\mathrm{o}}_k) = 1$ if and only if the relative location between $\hat{\boldsymbol{B}}^{\mathrm{s}}_k$ and $\hat{\boldsymbol{B}}^{\mathrm{o}}_k$ satisfies the relation p_k, 0 otherwise. $RS = \sum_{k=1}^{|P|} score(\hat{\boldsymbol{B}}^{\mathrm{s}}_k, \hat{\boldsymbol{B}}^{\mathrm{o}}_k)/|P|$.

To evaluate how much BBs cover the area of the whole image, we compute the coverage of predicted BBs over the image area: $coverage = \bigcup_{i=1}^{|E|} \hat{\boldsymbol{B}}_i/(\text{image area})$.

We note that $R@\tau$ and $rIoU$ consider the consistency between predicted BBs and GTs, and RS and $coverage$ are independent of GTs. On other words, $R@\tau$ and $rIoU$ evaluate absolute locations of BBs while RS (and $coverage$ as well to some extent) does semantic relations. Therefore, they together effectively evaluate the layout in a wide range of aspects.

5.5.3 Implementation and training details

We implemented our model in PyTorch [130] and optimized it on a PC with GTX1080Ti ×2 using the Adam optimizer with the recommended parameters [74] and the batch size of 16 for 500 epochs. We used VGG-19 [30] pre-trained on ImageNet without any fine-tuning as Φ, and $l = conv4\_2$ to compute $\mathcal{L}_{\text{context}}$. The total training time for each model was about one week on a PC with GTX1080Ti × 2 while testing time was less than 0.5 second per structured-text input.

We trained the model except for the pre-processing in the end-to-end manner where we set $\lambda_1 = \lambda_2 = \lambda_3 = \lambda_4 = 1$, and do not pre-train each individual subset, meaning that we do not use any ground-truth BBs to train the visual-relation layout. The layout predictor receives signals not only directly from the relation loss but also from the other losses. In an early stage of the training, the rendering part cannot generate reasonable images because the quality of BBs is poor. This means the signals from losses are strong, leading to quick convergence of the layout predictor. As the training proceeds, the layout predictor properly works, and the rendering part gradually becomes better. $\mathcal{L}_{\text{rel}}$, at that time, keeps the layout predictor stable and more accurate.

5.6 Experimental Results

5.6.1 Comparison with state-of-the-arts

Qualitative evaluation. Figs. 5.5 and 5.6 show examples of the results obtained by our method and SOTAs [17–20] on COCO-stuff [16] and GENOME [21] datasets. They show that the generated images by our method successfully preserve the scene structure given in text descriptions, indicating that our proposed visual-relation layouts are highly consistent with those of ground-truths. We see that the results by Johnson+[17] have reasonable layouts, however, their layouts failed to keep all relations well and the visual impression of their results is not good. The results by Zhang+[18] and Xu+[19] are clear in (entities) details but they lose the scene structure (some entities disappear). The results by Ashual+ [20] (COCO-stuff only) are more impressive than ours to some extent, however, they use GT layout and pre-defined

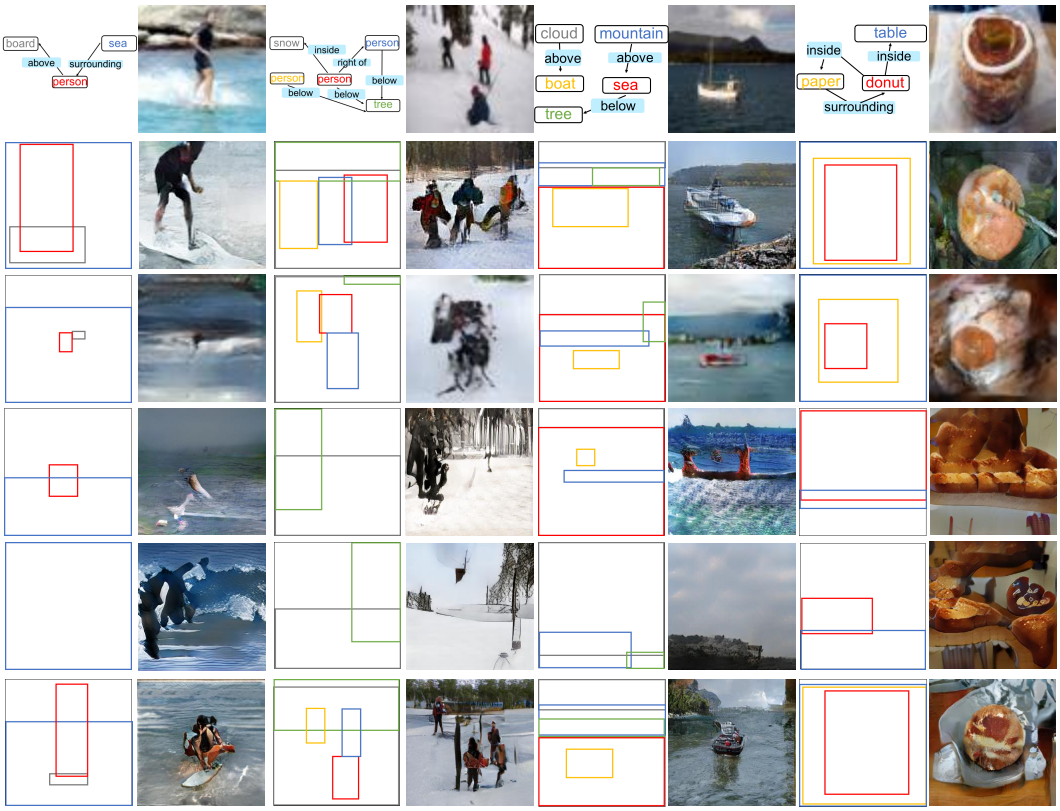


Figure 5.5: Visual comparison on COCO-stuff [16]. For each example, we show the scene graph and reference image at the first row. From second to the last rows, we show the layouts and images generated by our method (256 × 256), Johnson+[17] (64 × 64), Zhang+[18] (256 × 256), Xu+[19] (256 × 256), and Ashual+ [20] (256 × 256). The color of each entity BB corresponds to that in the scene graph. Scene graphs and layouts are enlarged for best views. Note that the layouts of Ashual+ [20] are the ground-truth ones.

entities' appearances.

Quantitative evaluation. We classify all the compared methods into three: (A) Johnson+ [17], Hong+[22], Li+[48], and Ashual+[20] (which firstly infer a layout and then convert it to an image), (B) Zhang+[18] and Xu+[19] (which are directly conditioned on texts), and (C) Zhao+[117] and Sun+[118] (which are directly conditioned on ground-truth layouts).

Table 5.2 shows that our method (almost) outperforms (A) in *IS* and *FID* on both COCO-stuff and GENOME. In comparison with (B), our method achieves the best in *FID* on both the datasets, the best on GENOME and the second best on COCO-stuff in

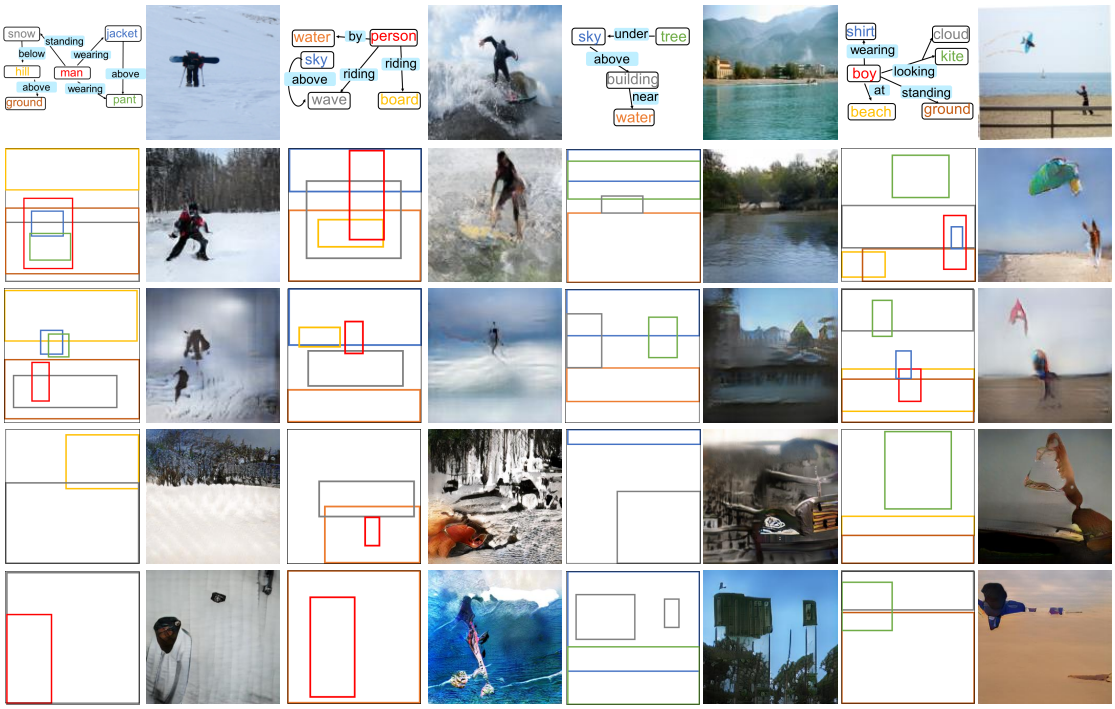


Figure 5.6: Visual comparison on GENOME [21]. For each example, we show the scene graph and reference image at the first row. From second to the last rows, we show the layouts and images generated by our method (256 × 256), Johnson+[17] (64 × 64), Zhang+[18] (256 × 256), and Xu+[19] (256 × 256). The color of each entity BB corresponds to that in the scene graph. Scene graphs and layouts are enlarged for best views.

IS. Xu+[19] achieved better *IS* on COCO-stuff than us because (i) Xu+[19] focuses on generating images in good human perception based on entity information and (ii) COCO-stuff has less complex relations, in other words, layouts may be less important. On GENOME, however, text descriptions are more complex with many entities and relations, and their results are degraded due to poor layouts as seen later in Table 5.3. Table 5.2 also shows that the scores of our completed model are comparable to those of (C), meaning that our (predicted) visual-relation layout is close to the ground-truth layout. When replacing the predicted layout by the ground-truth (the 17th row), our results achieve the same level with (C). Note that the results in the 17th row are different from simply conditioning a single CRN [41] on GT layout. We attribute these results (the 17th row) to the stacking-GANs through training. We may thus conclude that our method is more effective and promising than the others.

Table 5.2: Comparison of the overall quality using *IS* (larger is better) and *FID* (smaller is better). From the 4th to the 16th rows: the methods conditioned on structured/unstructured texts (the best in blue; the second best in red). From the 17th to the 19th rows: the methods conditioned on ground-truth layouts (**bold** indicates the best). Scores inside the parentheses indicate those reported in the original papers.

	IS						FID					
Dataset	**COCO-stuff** [16]			**GENOME** [21]			**COCO-stuff** [16]			**GENOME** [21]		
Image size	64 × 64	128 × 128	256 × 256	64 × 64	128 × 128	256 × 256	64 × 64	128 × 128	256 × 256	64 × 64	128 × 128	256 × 256
Ours w/o individual usage	7.02±0.19	8.12±0.41	9.95±0.31	5.48±0.16	5.66±0.26	5.91±0.41	63.28	59.52	55.21	72.42	72.02	71.49
Ours w/o weighted unification	7.10±0.27	8.64±0.37	10.49±0.41	5.99±0.22	6.61±0.31	7.32±0.37	61.89	57.20	49.16	69.37	60.89	57.18
Ours w/o refinedBB2layout	7.23±0.20	8.70±0.35	10.50±0.37	6.11±0.25	6.93±0.29	7.87±0.33	57.68	53.81	46.55	67.65	58.54	54.45
Ours w/o $\mathcal{L}_{\text{pix}}$	7.29±0.17	9.26±0.31	11.36±0.40	6.05±0.15	8.26±0.27	8.66±0.36	56.81	51.02	43.18	70.18	60.02	58.63
Ours w/o $\mathcal{L}_{\text{context}}$	7.56±0.11	9.68±0.33	11.47±0.42	6.37±0.16	8.41±0.22	8.97±0.31	50.89	47.22	40.10	68.20	56.39	53.75
Ours w/o $\mathcal{L}_{\text{adv}}$	7.31±0.19	9.47±0.34	11.41±0.47	6.30±0.19	8.39±0.20	8.96±0.39	56.24	50.87	41.05	68.34	57.23	53.86
Ours (completed model)	9.20±0.32	12.01±0.40	14.20±0.45	7.97±0.30	9.24±0.41	11.75±0.43	35.12	29.12	27.39	58.37	50.19	36.79
Johnson+ [17]	(6.70±0.10)	7.13±0.24	7.25±0.47	(5.50±0.10)	5.72±0.33	5.81±0.37	67.99	65.23	64.19	73.39	69.48	68.42
Hong+ [22]	–	(11.46±0.09)	–	–	–	–	–	–	–	–	–	–
Li+ [48]	(9.40±0.20)	–	–	(7.30±0.20)	–	–	–	–	–	–	–	–
Ashual+ [20]	(7.90±0.20)	(10.40±0.40)	(14.50±0.70)	–	–	–	(65.30)	(75.40)	(81.00)	–	–	–
Zhang+ [18]	7.79±0.32	8.49±0.52	(10.62±0.19)	6.35±0.16	6.44±0.25	7.39±0.38	87.21	85.37	78.19	108.68	86.17	77.95
Xu+ [19]	11.78±0.14	19.11±0.28	(25.89±0.47)	6.38±0.22	6.88±0.32	8.20±0.35	50.06	43.98	34.48	96.40	83.39	72.11
Ours with GT layout	**10.36±0.41**	13.73±0.59	**14.78±0.65**	**8.87±0.57**	10.04±0.45	**12.03±0.37**	**30.98**	**27.74**	**26.32**	45.63	40.96	**27.33**
Zhao+ [117] (GT layout)	(9.10±0.10)	–	–	(8.10±0.10)	–	–	–	–	–	–	–	–
Sun+ [118] (GT layout)	(9.80±0.20)	**(13.80±0.40)**	–	(8.70±0.40)	**(11.10±0.60)**	–	(34.31)	(29.65)	–	**(34.75)**	**(29.36)**	–
Ground-truth	16.25±0.38	25.89±0.47	32.61±0.69	13.92±0.42	21.43±1.03	31.22±0.65	–	–	–	–	–	–

Next, we evaluated how the scene structure given in input text was preserved in generated images using $R@\tau$ (we changed τ from 0.3 to 0.9 by 0.2), *rIoU*, *RS*, and *coverage*, see Table 5.3. We remark that we computed *RS* only for COCO-stuff because COCO-stuff has geometrical relations only. For Zhang+[18] and Xu+[19], we employed Faster-RCNN [32] to estimate their predicted BBs of entities where we set the number of generated BBs to be the number of entities in an image. We note that the number of predicted BBs by ours or Johnson+[17] was always the same with the number of entities in an image.

Table 5.3: Comparison of the scene structure using $R@\tau$, *rIoU*, *RS*, and *coverage* (larger is better; the best in **bold**).

Dataset	**COCO-stuff** [16]							**GENOME** [115]					
Metric	$R@\tau$				*rIoU*	*RS*	*coverage*	$R@\tau$				*rIoU*	*coverage*
	0.3	0.5	0.7	0.9			GT=98.24	0.3	0.5	0.7	0.9		GT=77.10
Ours w/o individual usage	61.45	43.22	29.71	20.05	0.2652	53.48	94.96	26.48	14.29	11.90	9.81	0.1264	50.07
Ours w/o weighted unification	61.76	45.28	30.22	20.51	0.2795	56.27	95.07	29.57	18.22	13.76	10.80	0.1501	56.77
Ours (completed model)	**65.34**	**49.01**	**35.87**	**23.61**	**0.3186**	**68.23**	**97.19**	**35.00**	**23.12**	**16.34**	**13.40**	**0.1847**	**71.13**
Johnson+ [17]	59.75	42.53	29.23	19.89	0.2532	51.20	94.82	28.13	17.17	12.30	10.47	0.1485	52.28
Zhang+ [18]	37.81	20.50	10.64	7.76	0.0824	30.72	60.15	18.38	10.84	8.11	5.82	0.0643	40.07
Xu+ [19]	21.39	10.71	8.15	5.83	0.0671	31.97	52.76	16.02	9.33	7.66	5.15	0.0579	36.82

Table 5.4: Comparison using caption generation metrics on COCO-stuff (larger is better; the best in **bold**). Scores inside the parentheses indicate those reported in [22].

Method	*BLEU* − 1	*BLEU* − 2	*BLEU* − 3	*BLEU* − 4	*METEOR*	*CIDEr*
Ours	**0.561**	**0.352**	**0.217**	**0.139**	**0.157**	0.325
Johnson+ [17]	0.531	0.321	0.183	0.107	0.141	0.238
Hong+ [22]	(0.541)	(0.332)	(0.199)	(0.122)	(0.154)	(**0.367**)
Zhang+ [18]	0.417	0.214	0.111	0.062	0.095	0.078
Xu+ [19]	0.450	0.251	0.157	0.087	0.105	0.251
Ground-truth	0.627	0.434	0.287	0.191	0.191	0.367
	(0.678)	(0.496)	(0.349)	(0.243)	(0.228)	(0.802)

Table 5.3 shows that our method performs best, indicating that our predicted BBs more precisely agree with those in relation (location and size) of entities given in texts than the compared methods. To be more specific, *rIoU*'s in Table 5.3 show that our predicted BBs more successfully retain the relations of entities than the other methods. This observation is also supported by *RS* on COCO-stuff. Moreover, our method outperforms the others in *coverage* and achieves comparable levels with the ground-truth BBs. These indicate that our visual-relation layout is well-structured. Our method thus has even better ability of rendering more realistic images with multiple entities since the faithful scene structure and more BB coverage (i.e., entity information) are achieved. Note that the observation that the *coverage*'s on COCO-stuff are better than those on GENOME explains the reason why generated images on COCO-stuff are better in *IS* and *FID* than those on GENOME.

Next, we use the image caption task to evaluate how the generated image is relevant to its input text. We follow [22] to report scores on COCO-stuff [16], see Table 5.4. Note that we evaluated on COCO-stuff only since the pre-trained image caption model on GENOME is not available. We also note that all the scores on the ground-truth dataset in [22] are higher than our re-computation. Table 5.4 shows that our method outperforms the others [17–19, 22] on *BLEU*, *METEOR* and comparable to [22] on *CIDEr*. We thus conclude that our method performs more consistently with input texts than the others.

Finally, we show the diversity score of generated images in Table 5.5. Overall,

Table 5.5: Comparison using diversity score [23] (the best in blue; the second best in red). Original scores are inside the parentheses.

Method	**COCO-stuff** [16]	**GENOME** [115]
Ours (64 × 64)	0.36±0.10	0.39±0.09
Ours (128 × 128)	0.45±0.12	0.49±0.07
Ours (256 × 256)	0.52±0.09	0.56±0.06
Johnson+ [17]	0.29±0.10	0.31±0.08
Ashual+ [20]	(0.67±0.05)	–
Zhao+ [117]	(0.15±0.06)	(0.17±0.09)
Sun+ [118]	(0.40±0.09)	(0.43±0.09)

our scores are higher than Johnson+ [17], Zhao+ [117], and Sun+ [118] on both COCO-stuff and GENOME datasets and comparable to Ashual+ [20] on COCO-stuff dataset. Moreover, along with our stacking-GANs, our scores become better and better. These scores also support the efficacy of our method.

We quantitatively observed that the number of (trainable) parameters in our model (41M) is comparable with Johnson+ [17] (28M), Zhang+ [18] (57M), Xu+ [19] (23M), and is significantly smaller than Ashual [20] (191M). We thus conclude that our method is not so complex and more sufficient than others.

5.6.2 Ablation studies

We evaluated the plausibility of employing the visual-relation layout module, see the first block of Tables 5.2 and 5.3: ours w/o individual usage denotes the model dropping the individual usage subnet; ours w/o weighted unification denotes the replacement of Eq. (5.1) with just averaging in the individual usage subnet; ours w/o refinedBB2layout denotes the replacement by just putting all entity layouts together in constructing the visual-relation layout. Fig. 5.7 illustrates a typical output example of the ablation models. We remark that model w/o comprehensive usage is not applicable since all the other subnets in our visual-relation layout module need the output by the

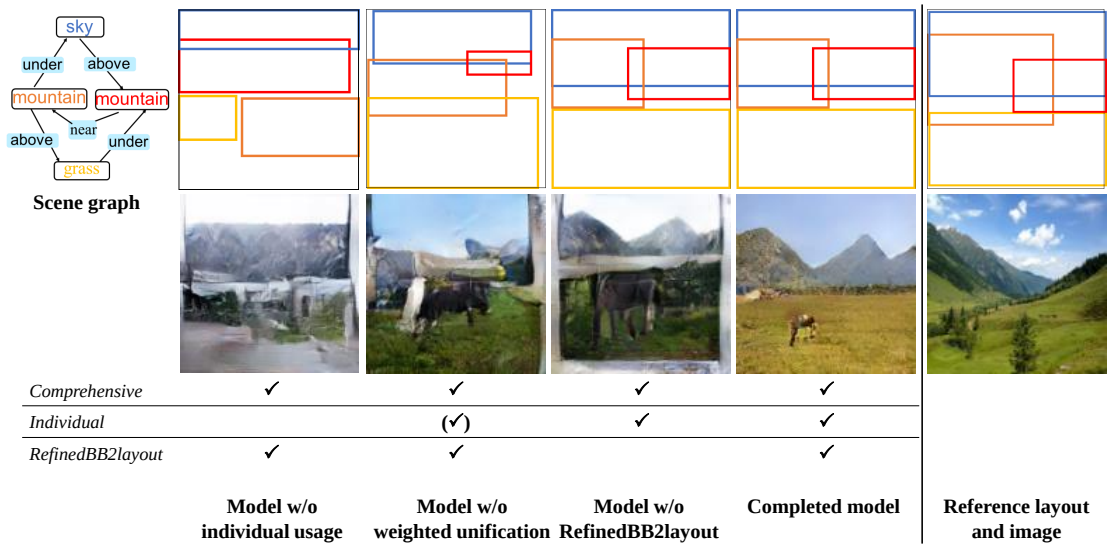


Figure 5.7: Example of layouts and generated images by the ablation models. For each model, the 1st row shows the layout, the 2nd row shows the generated image. All images are at 256×256 resolution.

comprehensive usage subnet.

The 4th and 5th rows of Tables 5.2 and 5.3 confirm the importance of the individual usage subnet. We also see the necessity of our learnable weights in Eq. (5.1) because model w/o weighted unification performs better than model w/o individual usage. We may conclude that the relation-unit prediction and the weighted unification together bring gain on our performance.

From Fig. 5.7, we visually observe that the layout by the model w/o individual usage does not successfully reflect relations. This observation is applicable to the model w/o weighted unification as well. As a result, both the models generated images in poorer quality than our complete model. The relation-units are in diversity: entity BBs can be various in size and location because of multiple relations (see Fig. 5.8, for example), and thus simply averaging BBs corresponding to the same entity does not successfully retain the relations among entities. Therefore, the individual usage of relations in addition to the comprehensive usage is important for more consistent layout with input text.

The 6th row in Table 5.2 shows the significance of the refinedBB2layout. Complex descriptions with many entities and relations tend to produce overlapped BBs. The

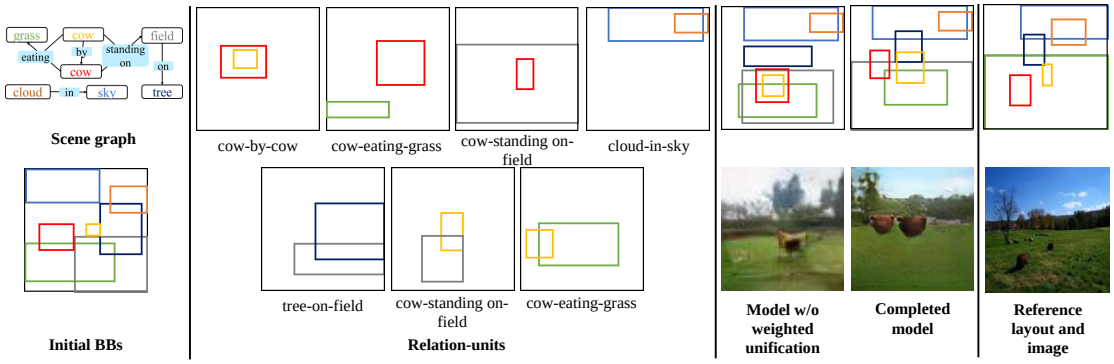


Figure 5.8: Example of relation-units in the individual usage subnet; layouts and generated images by model w/o weighted unification and completed model.

Figure 5.9: Example of output along with the stacking-GANs. From left to right, scene graph, visual-relation layout, the outputs at 64×64, 128×128, 256×256 resolutions, and the reference image.

model w/o refinedBB2layout cannot necessarily produce all the entities in the layout, generating poor images.

We also evaluated the necessity of each term of the loss function through comparing our completed model with models dropping one term each: model w/o $\mathcal{L}_{\text{pix}}$, model w/o $\mathcal{L}_{\text{context}}$, and model w/o $\mathcal{L}_{\text{adv}}$ (we dropped each term in the loss function (Eq. (5.2)) except for stacking-GANs). From the 2nd block of Table 5.2, we see that the absence of any term degrades the quality of generated images. This indicates that all the loss terms indeed contribute to performance.

Finally, we see that along with the stacking of GANs, our method progressively generates better images in terms of *IS* and *FID* (Table 5.2). We observe that at 64×64 resolution, generated images tend to be blurred and lose some details while the details of images are improved as the resolution becomes higher (the best result is obtained at 256×256 resolution) (see Fig. 5.9 as an example). We also confirmed that the visual-relation layouts of generated images at any resolutions are the same and highly

consistent with texts.

When we replaced CRN in [17] with our stacking-GANs for 128×128 and 256×256 resolutions to factor out the influence of image generators, we see that the improvement of [17] on *IS* and *FID* along the resolution is worse than that of our model (the 10th and the 11th rows of Table 5.2). This indicates that better layout significantly improves the performance of the final image generation and also confirms clearer contribution of the proposed visual-relation layout module. We remark that the stacking-GANs enables us to generate coarse-to-fine images while keeping the scene structure and the quality of generated images significantly depends on layout because with more precise BBs, more realistic images can be generated.

5.7 Conclusion

We proposed a GAN-based end-to-end network for text-to-image generation where relations between entities are comprehensively and individually used to infer a visual-relation layout. We also conditioned the stacking-GANs on the visual-relation layout to generate high-resolution images. Our layout preserves the scene structure more precisely than the layout by SOTAs. Experimental results on two public datasets demonstrate the effectiveness of our method.

6 Conclusion and Future Work

6.1 Conclusion

This dissertation introduced a novel approach for learning-based image synthesis. Our approach first select disentangled feature representations according to specific task and then combined the disentangled feature representations with an appropriate learning process to generate images. Since the feature representations themselves contain different meaningful information, our approach therefore is promising and is able to apply to a wide range of image synthesis. We investigate effectiveness of our approach on three interestingly challenging tasks. They are (i) rendering image contents in different styles, (ii) image manipulation with text, and (iii) text-to-image synthesis.

For rendering image contents in different styles, we extract content feature and style feature in different layers in network. We thus design the encoders with different depths to retain useful information from both the content and the style. Namely, we design a deep encoder for the content and a shallow encoder for the style. These features are combined in an adaptive way by using our proposed adaptive weight

computed from the content and the style losses during training. In order to train the network, we employ the pre-trained VGG-16, to compute content loss and style loss, both of which are efficiently used for the feature injection as well as the feature concatenation. By employing disentangled content and style features, our method therefore is able to control contribution of each feature in stylized images.

For image manipulation with text, we introduce *Paired-D GAN* where the network consists of one generator and two discriminators. The generator captures background feature in source image from lower-layer. It then combines background feature and text (foreground) feature extracted from pre-trained text encoder to generate images. Two discriminators, on the other hand, judge foreground and background of the synthesized image separately to meet an input text description and a source image. We also introduce a three-player adversarial learning process to simultaneously train one generator and two discriminators. By using disentangled foreground and background features, our method is able to synthesize a realistic image where an input text description matches its corresponding part (foreground) of the image while preserving background of a given source image.

For text-to-image synthesis, we comprehensively use all available relations in text description together and individual relation separately to predict visual-relation layout, i.e., localized bounding-boxes for all the entities so that each of which uniquely corresponds to each entity and faithfully preserves relations between the entities. We then condition the layout on a stack of three GANs, namely stacking-GAN, to generate image that consistently captures the scene structure. By using relations among entities in two ways: comprehensive usage and individual usage, our method can predict scene layout precisely, leading better image can be generated.

Our comprehensive experiments on public dataset verify the effectiveness of our approach. Moreover, they also confirm that our approach is capable of handling a wide range of image synthesis tasks.

6.2 Future Work

In this section, this dissertation discusses our future direction to tackle our current shortcomings. We thus sequentially present our plan for each task as belows.

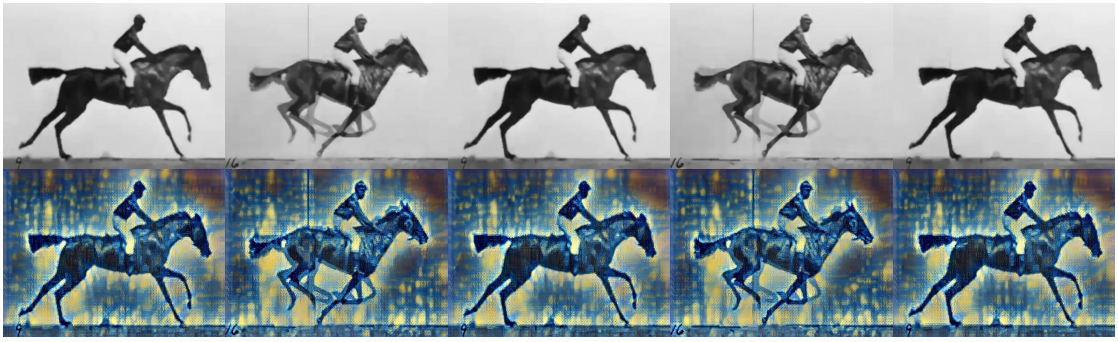


Figure 6.1: Examples of stylized video in real-time using the "Starry night" style. We use the video of Eadweard Muybridge "The horse in motion" (1878) as the content input. Our model processes every frame independently without any post-processing. Video resolution is 480×640 at 30 FPS.

6.2.1 Rendering image contents in different styles

Our proposed method requires fine-tuning of parameters from an existing model to deal with different styles. This limits the applicability of our proposed method to multi-style transfer. Extending our proposed method so that it can deal with a large style dataset such as Wikiart or unseen styles is left for future work.

As an extension of rendering image contents in different styles, the real-time video stylization methods are currently proposed [131–133]. Since our proposed method runs fast, we believe that it can be useful for real-time video stylization. Though video stylization is out of the scope of this dissertation, we applied our method in the frame-by-frame manner to several videos for video stylization demonstration. Fig. 6.1 shows some examples of stylized frames from a video. Our approach was able to stylize videos in real-time with the resolution 480×640 at 30 FPS or more. As we see, our method produces reasonable results for consecutive frames with varying appearance, meaning that the usage of our method for real-time video stylization is promising. We remark that we did not use either temporal regularization or post-processing. Different from image style transfer, real-time video stylization needs to pay attentions to the temporal consistency among adjacent video frames. Incorporating the temporal consistency into our method for real-time video stylization is left for our future work.

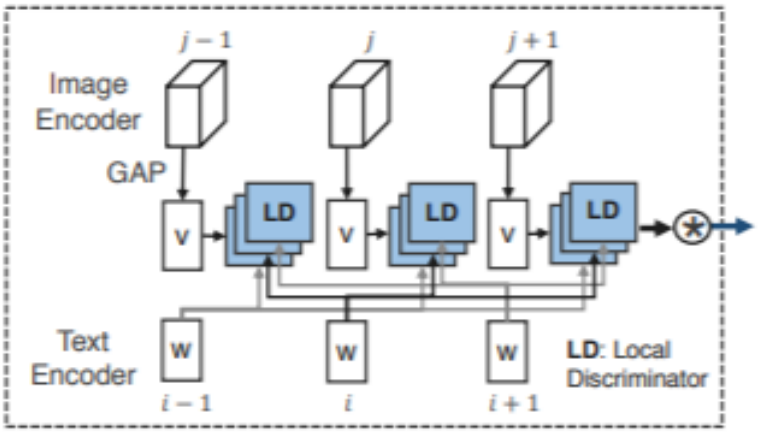


Figure 6.2: Word-level discriminator.

6.2.2 Image manipulation with text

Our *Paired-D GAN* is able to generate an image where an input text description matches its corresponding part (foreground) of the image while preserving background of a given source image. However, as illustrated in Fig. 4.5, our proposed method failed in matching foreground well in case the details of foreground are too small. Taking into account the details of foreground is thus crucial to improve quality of generated images. Such case may be handled by employing an attention mechanism [19] which helps network focuses on significant semantic information and ignores some unimportant words. Indeed, Nam et al. [56] recently proposed employing attention mechanism [19] in the discriminator for this task (Fig. 6.2). Their method really generate details of image that match the text description. However, they still struggle to retain background well. We thus believe that incorporating attention mechanism in our current method is helpful. Investigating such combination is left for our future work.

Moreover, our method only handles one foreground object which limits our application. In reality, one image may contain more than one foreground. Manipulating multiple (foreground) objects with text is more challenging than our current setting. To deal with it, the model should have ability of matching object in image with its corresponding words in text description. More exploiting in this direction is also left for our future.

6.2.3 Text-to-Image synthesis

Currently, our visual-relation layout covers a large area of the whole image leading our method has even better ability of rendering more realistic images. Recent work [134] show that well-annotated instance maps are helpful in realistically generating images. Our layout is currently constructed by bounding-boxes. It thus need a further step to transform from bounding-boxes to instance segmentation maps. Incorporating a novel step in our method is left for our future work.

Publication List

Journal

1. Duc Minh Vo, Akihiro Sugimoto, "Two-Stream FCNs to Balance Content and Style for Style Transfer", In Machine Vision and Applications (MVAP), 2020.

Conference

1. Duc Minh Vo, Trung-Nghia Le, and Akihiro Sugimoto, "Balancing Content and Style with Two-Stream FCNs for Style Transfer", In IEEE Winter Conference on Applications of Computer Vision (WACV), 2018.

2. Duc Minh Vo, Akihiro Sugimoto, "Paired-D GAN for Semantic Image Synthesis", In Asian Conference on Computer Vision (ACCV), 2018.

3. Duc Minh Vo, Akihiro Sugimoto, "Visual-Relation Conscious Image Generation from Structured-Text", In European Conference on Computer Vision (ECCV), 2020.

4. Tzu-Ting Fang, Duc Minh Vo, Akihiro Sugimoto, Shang-Hong Lai, "Stylized-Colorization for Line Arts", In International Conference on Pattern Recognition (ICPR), 2020.

Bibliography

[1] https://anhreynolds.com/blogs/cnn.html.

[2] https://neurohive.io/en/popular-networks/vgg16/.

[3] Ian Goodfellow. Nips 2016 tutorial: Generative adversarial networks, 2016.

[4] Nicolas Audebert, Bertrand Le Saux, and Sébastien Lefèvre. Semantic segmentation of earth observation data using multimodal and multi-scale deep networks. In *ACCV*, 2016.

[5] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. *ArXiv e-prints*, mar 2016.

[6] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In *ICLR*, 2016.

[7] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *ECCV*, 2016.

[8] Xun Huang and Serge Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. In *ICCV*, 2017.

[9] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *CVPR*, 2016.

[10] Lu Sheng, Ziyi Lin, Jing Shao, and Xiaogang Wang. Avatar-net: Multi-scale zero-shot style transfer by feature decoration. In *CVPR*, 2018.

[11] Tian Qi Chen and Mark Schmidt. Fast patch-based style transfer of arbitrary style. In *NIPS*, 2016.

[12] Yijun Li, Chen Fang, Jimei Yang, Zhaowen Wang, Xin Lu, and Ming-Hsuan Yang. Universal style transfer via feature transforms. In *NIPS*, 2017.

[13] H. Dong, S. Yu, C. Wu, and Y. Guo. Semantic image synthesis via adversarial learning. In *ICCV*, 2017.

[14] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The Caltech-UCSD Birds-200-2011 Dataset. Technical Report CNS-TR-2011-001, California Institute of Technology, 2011.

[15] M-E. Nilsback and A. Zisserman. Automated flower classification over a large number of classes. In *Proceedings of the Indian Conference on Computer Vision, Graphics and Image Processing*, 2008.

[16] Holger Caesar, Jasper Uijlings, and Vittorio Ferrari. Coco-stuff: Thing and stuff classes in context. In *CVPR*, 2018.

[17] Justin Johnson, Agrim Gupta, and Li Fei-Fei. Image generation from scene graphs. In *CVPR*, 2018.

[18] Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaogang Wang, Xiaolei Huang, and Dimitris Metaxas. Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. In *ICCV*, 2017.

[19] Tao Xu, Pengchuan Zhang, Qiuyuan Huang, Han Zhang, Zhe Gan, Xiaolei Huang, and Xiaodong He. Attngan: Fine-grained text to image generation with attentional generative adversarial networks. In *CVPR*, 2018.

[20] Oron Ashual and Lior Wolf. Specifying object attributes and relations in interactive scene generation. In *ICCV*, 2019.

[21] Ranjay Krishna, Yuke Zhu, Oliver Groth, Justin Johnson, Kenji Hata, Joshua Kravitz, Stephanie Chen, Yannis Kalantidis, Li-Jia Li, David A Shamma, Michael Bernstein, and Li Fei-Fei. Visual genome: Connecting language and vision using crowdsourced dense image annotations. In *IJCV*, 2017.

[22] Seunghoon Hong, Dingdong Yang, Jongwook Choi, and Honglak Lee. Inferring semantic layout for hierarchical text-to-image synthesis. In *CVPR*, 2018.

[23] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *CVPR*, 2018.

[24] Daniel Kahneman. *Thinking fast and slow*. 2011.

[25] Ernest Davis and Gary Marcus. Commonsense reasoning and commonsense knowledge in artificial intelligence. *Communications of the ACM*, 2015.

[26] Simon J. D. Prince. *Computer Vision: Models, Learning, and Inference*. Cambridge University Press, New York, NY, USA, 2012.

[27] Robert J. Woodham. Photometric Method For Determining Surface Orientation From Multiple Images. *Optical Engineering*, 19:139 – 144, 1980.

[28] David G. Lowe. Object recognition from local scale-invariant features. In *ICCV*, 1999.

[29] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pages 1097–1105. 2012.

[30] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

[31] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[32] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *NIPS*, 2015.

[33] Iasonas Kokkinos. Ubernet: Training a universal convolutional neural network for low-, mid-, and high-level vision using diverse datasets and limited memory. In *CVPR*, 2017.

[34] F. Schroff, D. Kalenichenko, and J. Philbin. Facenet: A unified embedding for face recognition and clustering. In *CVPR*, pages 815–823, 2015.

[35] Qizhe Xie, Eduard Hovy, Minh-Thang Luong, and Quoc V. Le. Self-training with noisy student improves imagenet classification, 2019.

[36] Hugo Touvron, Andrea Vedaldi, Matthijs Douze, and Hervé Jégou. Fixing the train-test resolution discrepancy: Fixefficientnet, 2019.

[37] Jun-Yan Zhu, Richard Zhang, Deepak Pathak, Trevor Darrell, Alexei A Efros, Oliver Wang, and Eli Shechtman. Toward multimodal image-to-image translation. In *NIPS*. 2017.

[38] Xinchen Yan, Jimei Yang, Kihyuk Sohn, and Honglak Lee. Attribute2image: Conditional image generation from visual attributes. *ECCV*, 2016.

[39] Alexei A. Efros and William T. Freeman. Image quilting for texture synthesis and transfer. In *SIGGRAPH*, 2001.

[40] P. Isola, J. Y. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. In *CVPR*, 2017.

[41] Qifeng Chen and Vladlen Koltun. Photographic image synthesis with cascaded refinement networks. In *ICCV*, 2017.

[42] Fujun Luan, Sylvain Paris, Eli Shechtman, and Kavita Bala. Deep photo style transfer. In *CVPR*, 2017.

[43] Roey Mechrez, Eli Shechtman, and Lihi Zelnik-Manor. Photorealistic style transfer with screened poisson equation. In *BMVC*, 2017.

[44] Xin Wang, Geoffrey Oxholm, Da Zhang, and Yuan-Fang Wang. Multimodal transfer: A hierarchical deep convolutional neural network for fast artistic style transfer. In *CVPR*, 2017.

[45] Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee. Generative adversarial text-to-image synthesis. In *ICML*, 2016.

[46] Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaogang Wang, Xiaolei Huang, and Dimitris Metaxas. Stackgan++: Realistic image synthesis with stacked generative adversarial networks. *arXiv: 1710.10916, 2017*, (*IEEE TPAMI*, to appear).

[47] S. Reed, Z. Akata, H. Lee, and B. Schiele. Learning deep representations of fine-grained visual descriptions. In *CVPR*, 2016.

[48] Yikang Li, Tao Ma, Yeqi Bai, Nan Duan, Sining Wei, and Xiaogang Wang. Pastegan: A semi-parametric method to generate image from scene graph. In *CVPR*, 2019.

[49] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 2048–2057, 2015.

[50] Justin Johnson, Andrej Karpathy, and Li Fei-Fei. Densecap: Fully convolutional localization networks for dense captioning. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 4565–4574, 2016.

[51] Jiasen Lu, Jianwei Yang, Dhruv Batra, and Devi Parikh. Neural baby talk. In *CVPR*, 2018.

[52] Xu Yang, Kaihua Tang, Hanwang Zhang, and Jianfei Cai. Auto-encoding scene graphs for image captioning. In *CVPR*, 2019.

[53] S. Antol, A. Agrawal, J. Lu, M. Mitchell, D. Batra, C. L. Zitnick, and D. Parikh. Vqa: Visual question answering. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 2425–2433, 2015.

[54] Peter Anderson, Xiaodong He, Chris Buehler, Damien Teney, Mark Johnson, Stephen Gould, and Lei Zhang. Bottom-up and top-down attention for image captioning and visual question answering. In *CVPR*, 2018.

[55] Léon Bottou. From machine learning to machine reasoning. *Machine learning*, 2011.

[56] Seonghyeon Nam, Yunji Kim, and Seon Joo Kim. Text-adaptive generative adversarial networks: Manipulating images with natural language. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[57] Bowen Li, Xiaojuan Qi, Thomas Lukasiewicz, and Philip HS Torr. Manigan: Text-guided image manipulation. In *CVPR*, 2020.

[58] David J. Heeger and James R. Bergen. Pyramid-based texture analysis/synthesis. In *SIGGRAPH*, 1995.

[59] Michael Ashikhmin. Synthesizing natural textures. In *Symposium on Interactive 3D Graphics*, 2001.

[60] https://github.com/hanzhanggit/StackGAN-Pytorch.

[61] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale GAN training for high fidelity natural image synthesis. In *ICLR*, 2019.

[62] Jianwei Yang, Anitha Kannan, Dhruv Batra, and Devi Parikh. Lr-gan: Layered recursive generative adversarial networks for image generation. *ICLR*, 2017.

[63] M. L. Ha, G. Franchi, M. Moller, A. Kolb, and V. Blanz. Segmentation and shape extraction from convolutional neural networks. In *WACV*, 2018.

[64] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *NIPS*. 2014.

[65] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. The MIT Press, 2016.

[66] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In Yves Lechevallier and Gilbert Saporta, editors, *Proceedings of COMPSTAT'2010*, pages 177–186, 2010.

[67] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[68] Yann Lecun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998.

[69] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010.

[70] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical Evaluation of Rectified Activations in Convolutional Network, 2015.

[71] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

[72] Dmitry Ulyanov, Andrea Vedaldi, and Victor S. Lempitsky. Instance normalization: The missing ingredient for fast stylization. In *ICML*, 2016.

[73] Aäron Van Den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. In *ICML*, 2016.

[74] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. *ICLR*, 2014.

[75] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning internal representations by error propagation. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations*, pages 318–362. MIT Press, 1986.

[76] Elman Mansimov, Emilio Parisotto, Jimmy Ba, and Ruslan Salakhutdinov. Generating images from captions with attention. In *ICLR*, 2016.

[77] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *ICML*, 2014.

[78] Tim Salimans, Diederik Kingma, and Max Welling. Markov chain monte carlo and variational inference: Bridging the gap. In *ICML*, pages 1218–1226, 2015.

[79] Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Jimenez Rezende, and Daan Wierstra. Draw: A recurrent neural network for image generation. In *ICML*, 2016.

[80] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Fei-Fei Li. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 2015.

[81] Roger A. Horn and Charles R. Johnson. *Matrix Analysis*. Cambridge University Press, New York, NY, USA, 2nd edition, 2012.

[82] Shaohua Li, Xinxing Xu, Liqiang Nie, and Tat-Seng Chua. Laplacian-steered neural style transfer. In *ACM-MM*, 2017.

[83] Dmitry Ulyanov, Vadim Lebedev, Andrea Vedaldi, and Victor Lempitsky. Texture networks: Feed-forward synthesis of textures and stylized images. In *ICML*, 2016.

[84] Vincent Dumoulin, Jonathon Shlens, and Manjunath Kudlur. A learned representation for artistic style. In *ICLR*, 2017.

[85] Martin Arjovsky, Soumith Chintala, and L'eon Bottou. Wasserstein generative adversarial networks. In *ICML*, 2017.

[86] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *NIPS*, 2016.

[87] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets, 2014.

[88] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. In *ICLR*, 2018.

[89] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, Xi Chen, and Xi Chen. Improved techniques for training gans. In *NIPS*. 2016.

[90] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *NIPS*, 2017.

[91] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *CVPR*, 2016.

[92] D.C Dowson and B.V Landau. The fréchet distance between multivariate normal distributions. *Journal of Multivariate Analysis*, 1982.

[93] Yaniv Taigman, Adam Polyak, and Lior Wolf. Unsupervised cross-domain image generation. *ICLR*, 2017.

[94] Guim Perarnau, Joost van de Weijer, Bogdan Raducanu, and Jose M. Álvarez. Invertible Conditional GANs for image editing. In *NIPS Workshop on Adversarial Training*, 2016.

[95] Christian Ledig, Lucas Theis, Ferenc Huszar, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew P. Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, and Wenzhe Shi. Photo-realistic single image super-resolution using a generative adversarial network. In *CVPR*, 2017.

[96] Chuan Li and Michael Wand. Precomputed real-time texture synthesis with markovian generative adversarial networks. In *ECCV*, 2016.

[97] Samaneh Azadi, Matthew Fisher, Vladimir Kim, Zhaowen Wang, Eli Shechtman, and Trevor Darrell. Multi-content gan for few-shot font style transfer. In *CVPR*, 2018.

[98] Artsiom Sanakoyeu, Dmytro Kotovenko, Sabine Lang, and Björn Ommer. A style-aware content loss for real-time hd style transfer. In *ECCV*, 2018.

[99] Dmytro Kotovenko, Artsiom Sanakoyeu, Pingchuan Ma, Sabine Lang, and Björn Ommer. A content transformation block for image style transfer. In *CVPR*, 2019.

[100] Yongcheng Jing, Yezhou Yang, Zunlei Feng, Jingwen Ye, Yizhou Yu, and Mingli Song. Neural style transfer: A review. *IEEE Transactions on Visualization and Computer Graphics*, 2019.

[101] S. Reed, Z. Akata, S. Mohan, S. Tenka, B. Schiele, and H. Lee. Learning what and where to draw. In *NIPS*, 2016.

[102] Dae Young Park and Kwang Hee Lee. Arbitrary style transfer with style-attentional networks. In *CVPR*, 2019.

[103] Yexun Zhang, Ya Zhang, and Wenbin Cai. Separating style and content for generalized style transfer. In *CVPR*, 2018.

[104] Yijun Li, Ming-Yu Liu, Xueting Li, Ming-Hsuan Yang, and Jan Kautz. A closed-form solution to photorealistic image stylization. In *ECCV*, 2018.

[105] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010.

[106] Aravindh Mahendran and Andrea Vedaldi. Understanding deep image representations by inverting them. In *CVPR*, 2015.

[107] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[108] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft coco: Common objects in context. In *ECCV*, 2014.

[109] Tu Nguyen, Trung Le, Hung Vu, and Dinh Phung. Dual discriminator generative adversarial nets. In *NIPS*. 2017.

[110] R. Wu, X. Li, and B. Yang. Identifying computer generated graphics via histogram features. In *ICIP*, 2011.

[111] Alexandros Dimakis Ashish Bora, Eric Price. Ambientgan: Generative models from lossy measurements. In *ICLR*, 2018.

[112] S. Chopra, R. Hadsell, and Y. LeCun. Learning a similarity metric discriminatively, with application to face verification. In *CVPR*, 2005.

[113] https://github.com/woozzu/dong_iccv_2017. Accessed: 2018-06-01.

[114] https://github.com/jwyang/lr-gan.pytorch. Accessed: 2018-06-01.

[115] Justin Johnson, Ranjay Krishna, Michael Stark, Jia Li, Michael Bernstein, and Li Fei-Fei. Image retrieval using scene graphs. In *CVPR*, 2015.

[116] Yikang Li, Wanli Ouyang, Bolei Zhou, Kun Wang, and Xiaogang Wang. Scene graph generation from objects, phrases and region captions. In *ICCV*, 2017.

[117] Bo Zhao, Lili Meng, Weidong Yin, and Leonid Sigal. Image generation from layout. In *CVPR*, 2019.

[118] Sun Wei and Wu Tianfu. Image synthesis from reconfigurable layout and style. In *ICCV*, 2019.

[119] Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Waikin Wong, and Wangchun Woo. Convolutional lstm network: A machine learning approach for precipitation nowcasting. In *NIPS*, 2015.

[120] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and Koray Kavukcuoglu. Spatial transformer networks. In *NIPS*, 2015.

[121] Roey Mechrez, Itamar Talmi, and Lihi Zelnik-Manor. The contextual loss for image transformation with non-aligned data. In *ECCV*, 2018.

[122] https://github.com/taoxugit/AttnGAN.

[123] https://github.com/openai/improved-gan/tree/master/inception_score.

[124] https://github.com/bioinf-jku/TTUR.

[125] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: A method for automatic evaluation of machine translation. In *ACL*, 2002.

[126] Alon Lavie and Abhaya Agarwal. Meteor: An automatic metric for mt evaluation with improved correlation with human judgments. In *ACL*, 2005.

[127] Ramakrishna Vedantam, C. Lawrence Zitnick, and Devi Parikh. Cider: Consensus-based image description evaluation. In *CVPR*, 2015.

[128] https://github.com/richzhang/PerceptualSimilarity.

[129] Subarna Tripathi, Anahita Bhiwandiwalla, Alexei Bastidas, and Hanlin Tang. Using scene graph context to improve image generation. In *CVPRW (WiCV)*, 2019.

[130] https://pytorch.org/.

[131] Chang Gao, Derun Gu, Fangjun Zhang, and Yizhou Yu. Reconet: Real-time coherent video style transfer network. In *ACCV*, 2018.

[132] H. Huang, H. Wang, W. Luo, L. Ma, W. Jiang, X. Zhu, Z. Li, and W. Liu. Real-time neural style transfer for videos. In *CVPR*, 2017.

[133] Wenbo Li, Longyin Wen, Xiao Bian, and Siwei Lyu. Evolvement constrained adversarial learning for video style transfer. In *ACCV*, 2018.

[134] Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. Semantic image synthesis with spatially-adaptive normalization. In *CVPR*, 2019.