Stuff segmentation and non-photorealistic rendering with generative adversarial networks

Choy, Jin Xiang

2019

Choy, J. X. (2019). Stuff segmentation and non-photorealistic rendering with generative adversarial networks. Master's thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/137213>

<https://doi.org/10.32657/10356/137213>

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License (CC BY-NC 4.0).

Downloaded on 26 Apr 2023 19:05:04 SGT

**NANYANG
TECHNOLOGICAL
UNIVERSITY**

SINGAPORE

STUFF SEGMENTATION AND
NON-PHOTOREALISTIC RENDERING WITH
GENERATIVE ADVERSARIAL NETWORKS

A thesis submitted to the
School of Computer Science and Engineering
of the Nanyang Technological University

by

CHOY JIN XIANG

Supervisor:

PROF ONG YEW SOON

in partial fulfilment of the requirement for the degree of
Master of Engineering

2019

Statement of Originality

I hereby certify that the work embodied in this thesis is the result of original research, is free of plagiarised materials, and has not been submitted for a higher degree to any other University or Institution.

29/10/2019

Date

Choy Jin Xiang

Supervisor Declaration Statement

I have reviewed the content and presentation style of this thesis and declare it is free of plagiarism and of sufficient grammatical clarity to be examined. To the best of my knowledge, the research and writing are those of the candidate except as acknowledged in the Author Attribution Statement. I confirm that the investigations were conducted in accord with the ethics policies and integrity standards of Nanyang Technological University and that the research data are presented honestly and without prejudice.

29/10/2019

Date

Prof. Ong Yew Soon

Authorship Attribution Statement

This thesis contains material from 1 paper published in the following peer-reviewed journal(s) / from papers accepted at conferences in which I am listed as an author.

Chapter 5 is submitted for publication as Choy Jin Xiang, Iti Chaturvedi, Yew-Soon Ong, 'Background segmentation for Deep Style', submitted to AAAI 2020.

The contributions of the co-authors are as follows:

- Prof Ong provided the initial project direction.
- I gathered and generated training data for the experiments. I also performed the experiments for introducing a semantic segmentation discriminator. I also analyzed the experiment data.
- I prepared the manuscript drafts. The manuscript was revised by Dr. Iti.
- Dr. Iti provided guidance in further reducing stuff painting error via Gaussian Failure Correction. She also provided guidance for evaluation metrics.

29/10/2019

Date

Choy Jin Xiang

Acknowledgements

I thank my supervisor Prof. Ong Yew Soon for his encouragement, guidance, and immense patience. I faced many difficulties on this long research journey, and without his constant encouragement and patience, my research would not be finished. I also thank Dr. Iti Chaturvedi for her many ideas and feedback. The many discussions provided much needed ideas for this research.

Lastly, I thank my family for their unwavering support of my graduate studies. Their understanding of the time I spent in my graduate studies gave me much needed motivation.

Contents

1	Introduction	1
1.1	Motivation	1
1.2	Background	7
1.2.1	Neural Style Transfer	7
1.2.2	GANs In Semantic Image Segmentation	9
1.3	Thesis Organization	10
1.4	Contributions	11
2	Related Work	12
2.1	Artificial Neural Networks (ANN)	12
2.1.1	Artificial Neurons	12
2.1.2	Activation Functions	13
2.1.3	Feed-Forward Neural Network	15
2.2	Deep Convolutional Neural Networks	17
2.2.1	Multilayer Perceptrons	17
2.2.2	Backpropagation	18
2.2.3	Common Layers In Neural Networks	19
2.3	Generative Adversarial Nets	28
2.3.1	Adversarial Nets	30
2.3.2	Conditional Generative Adversarial Networks (cGAN) .	32
2.3.3	Deep Convolutional Generative Adversarial Networks (DCGANs)	33

CONTENTS

2.3.4	Pixel Level Domain Transfer	35
2.3.5	Pix2Pix	36
2.3.6	CycleGAN	38
2.4	Summary	40
3	Experiment Data Preparation	42
3.1	Data Preparation Process	42
3.1.1	COCO-Stuff Dataset	42
3.1.2	Data Generation Pipeline	43
3.2	NPR Software	47
3.2.1	Painterly: Painting Generators	48
3.2.2	CartoonGAN	48
3.3	Summary	50
4	<i>Stuff</i> painting with GANs	52
4.1	Experiment Setup	52
4.1.1	Network Architecture	52
4.1.2	Training Data	54
4.1.3	Evaluation Methods	56
4.1.4	Training details	64
4.2	Results And Discussion	65
4.2.1	Comparison Baselines	65
4.2.2	Success Cases	66
4.2.3	Evaluation Metrics	70
4.2.4	Texture Learning	73
4.2.5	Advantage Of U-Nets	77
4.2.6	Analysis Of Loss Functions	81
4.2.7	Limitations	84
4.3	Summary	87

5	Stuff Painting GAN(SPGAN): Reducing <i>Stuff</i> Painting Error	89
5.1	Semantic Segmentation Discriminator	90
5.2	Gaussian Failure Correction	92
5.3	Experiments	94
5.3.1	Training Data	95
5.4	Results And Discussion	96
5.4.1	Improved <i>Stuff</i> Painting	96
5.4.2	Effect Of Second Discriminator	99
5.4.3	Impact Of Receptive Field Sizes	100
5.4.4	Gaussian Failure Correction	105
5.4.5	Evaluation Metrics	105
5.5	Summary	109
6	Conclusion And Future Work	110
6.1	Conclusion	110
6.2	Future Work	112

Abstract

Generative Adversarial Networks (GANs) have shown impressive results in a variety of image generation tasks in recent years, including rendering photorealistic images with artistic styles. However, current work in transforming images have mostly focused on either transforming the whole image, or on the *thing* classes. There have been little attention on the artistic rendering of only the *stuff* classes of images. Current possible methods of performing painting of specific image regions also result in unnatural boundaries between painted and non-painted regions. Therefore, we aim to develop an end-to-end model for the novel task of Non-Photorealistically Rendering the *stuff* of images.

In order to train a model capable of doing so, we first require images with partially painted *stuff* classes for training. However, due to a lack of such images, we propose a flexible and extensible partially painted image generation pipeline that uses an image segmentation dataset to generate partially painted image datasets.

We use these datasets with a GAN framework based on the Pix2Pix model for training. We find that the trained model can perform image *stuff* painting acceptably well, and it generated results that have more natural boundaries between painted and non-painted image regions. We then provide an analysis of the effects of the Pix2Pix architecture on the training task. We find that satisfactory results can also be obtained. We then discuss the limitations for this context and compare the performance to that of image *stuff* painting.

We then introduce the Stuff-Painting GAN (SPGAN) for reducing errors in identifying image regions for painting by introducing segmentation masks into the training process. An additional discriminator that takes in segmentation masks as input is introduced to the architecture. We then show that it performs on par or better than the baseline GAN framework in identifying

image regions for painting. To improve the segmentation learning we also introduce a Gaussian error correction in each training iteration.

Chapter 1

Introduction

1.1 Motivation

Throughout the history of humanity, images have played diverse and crucial roles. From recording important moments in history to capturing the wonders of human imagination, images have reminded people of the past and sparked inspiration for the future. As the saying goes, "A picture is worth a thousand words." Images usually contain a large amount of information, from details such as the fur of a dog to nuances such as the posture of a dog that shows that it is running. Thus, it is no wonder that the task of generating realistic images is not an easy one. Master painters are famed for their ability to generate life-like images, and famous photographers are known for their ability to capture beautiful photographs that accentuate the desired details of their subjects. It is an even harder task for a machine to generate realistic pictures, as small inaccurate details often betray the realism of the picture.

The challenges involved in having machine generate paintings have inspired the field of Non-Photorealistic Rendering (NPR). NPR is concerned about the generation of images that are composed of artistic styles such as painting and drawing [1]. It is the application of computational methods

to artistic rendering. The most common form of NPR is in the application of artistic styles to realistic images, thereby transforming inputs of realistic images into artistic versions of the same image.

Traditionally, NPR was performed with non-automatic methods where human artists would use digital tools to digitally transform an image. However, with the advent of machine learning, automatic NPR, where the machine artistically transforms an image automatically, is becoming a viable alternative method. One popular method for this was style transfer [2, 3], where an AI model is given an image and is tasked to apply a certain painting style to the image, which may be specified by giving the model a reference painting or a style that the model has learnt to apply through training.

Generative Adversarial Networks(GANs) [4] gained widespread popular attention due to its ability in generating surprisingly realistic images. Previous attempts at automatic image generation usually only produced very blurry images and were certainly not realistic. Other than realistic image generation, GANs have also shown great promise in many other image processing tasks in recent years. They have demonstrated spectacular success in any tasks that involve generating images, from generating new cartoon characters to image inpainting, there has been a surge of applications of GANs.

One area where GANs have shown huge successes is in the area of image-to-image translation. This task is defined as the transformation of a certain representation of a scene into another representation of the same scene, i.e. the translation from one image domain to another. For example, from a normal image to an edge map [5]. This is analogous to that of language translation, where the task is to translate a concept from one language to another. It is useful in a variety of applications, such as Deep Style that is capable of interpreting a painting style and transfer it to an uploaded image¹.

¹<https://deepdreamgenerator.com/>

Figure 1.1: Examples of cartoons where the main characters and the background have distinctly different styles.

Style transfer in NPR can be seen as a special case of an image-to-image translation task, where the input image domain consists of photorealistic images and the output domain consists of painted images of similar scenes as the input domain. Recently, in [6] the authors consider realistic geometric corrections so that a foreground object appears natural when superimposed into a new background image. In [7], the authors consider background-foreground segmentation to introduce shallow depth-of-field in photographs. In style transfer, applications usually focus on applying styles to the entire image. However, there are use cases where different styles are desired for different parts of the image. For example, in cartoons, the background often takes on a more painterly appearance as compared to the characters, as shown in Figure 1.1. Even though this style originated from cost-saving measures where animators reused painted backgrounds across frames, this has become an iconic style in cartoons, where artistic choices are made to deliberately render the background in a style that is different from the main characters.

The background of images are generally composed of *stuff* objects. *Stuff* has no specific or definite parts [8] and is defined by its texture or patterns [9]. Some examples of stuff would be sky, grass, and sea. In contrast, *things* are defined as objects that have a certain definite shape and size [9, 10, 11], and may also contain certain distinct identifiable parts (e.g. a plane has wings) [8].

With this in mind, we seek an automatic method of applying painterly

styles to only the *stuff* classes of images, which is analogous to the background of images, while leaving the *thing* classes unchanged. We term this process as **partial painting**. Given the impressive results that GANs have shown in image-to-image translation tasks, we explore the use of GANs for this task. In this thesis, we present a conditional GAN framework for the partial painting of *stuff* classes in images.

Conventionally, the painting of only certain regions in images can be achieved by combining an image with its painted version. A partially painted image can be generated by combining photorealistic and painted images in such a way that non-painted regions are transferred from the photorealistic image and painted regions are transferred from the painted images. Determining which image regions to be taken from which image can be done via annotations called segmentation maps (also referred to as segmentation masks). These segmentation maps can be generated either via manual annotation or via machine learning methods. We term this process as **two-step partial painting**, as illustrated in Figure 1.2. Generating these maps manually is a costly and time consuming effort, but produces the best segmentation maps. Segmentation maps generated via manual annotation produces partially painted images that are the best, with natural and meaningful boundaries between painted and non-painted image regions. Generating segmentation maps via machine learning requires significantly less time and effort in annotation but the segmentation results are less than ideal, with the resultant segmentation masks often having abrupt and unnatural boundaries. This results in images where the painted and non-painted image regions have abrupt and unnatural boundaries. An example of this is shown in Figure 1.3. Therefore, a one step machine learning process that implicitly performs these two actions together would be preferred, as we can obtain the cost and time savings provided by machine learning models while the resultant images should have a more natural-looking boundary between the painted and non-painted regions.

Figure 1.2: Two-step partial painting.

We need a dataset of images where only the stuff class in images is painted for training our model. As far as we know, there is no such dataset available. Therefore, we introduce a process for generating partially painted images from semantic segmentation datasets, which are datasets meant for training semantic segmentation models. In these datasets, each image has a corresponding segmentation mask. We make use of these image/mask pairs and image painting software to generate partially painted datasets.

The task of image partial painting is a combination of a segmentation task and an image translation task, where the generator has to segment the

Figure 1.3: Comparison between (b) human-annotated segmentation results and (a) VGG-Unet[12, 13] generated segmentation results.

image *stuff* and *thing* classes, before performing image translation on the *stuff* regions of the image. To improve the segmentation accuracy of the generator, we introduce a **semantic segmentation discriminator** to the GAN architecture, in addition to the usual real/fake discriminator. This additional discriminator takes in a real or fake partially painted image and its corresponding segmentation mask, and learns to discriminate whether a partially painted image fits a segmentation mask. I.e. whether the painted

portions are *stuff* classes in the mask. This Stuff Painting GAN (SPGAN) architecture is then trained with the generated partially painted image dataset.

1.2 Background

1.2.1 Neural Style Transfer

Neural Style Transfer (NST) is the rendering of the content of an image in a different style using neural networks [14]. With advances in deep CNNs for image processing tasks, computer vision systems are able to model high-level semantic features in images. This has enabled the automatic application of artistic styles to images in meaningful ways, resulting in the field of neural style transfer.

Style transfer is performed when a style from a reference image is transferred to a source content image, which results in an image that is an artistic rendering of the source content image in the style of the reference image. This can also be considered as a texture transfer problem [15]. This is a problem where the goal is to produce an output image given two input images: a content image and a style image. The output image is to have the semantic content of the content image and the style of the style image.

A neural network approach to style transfer that attempts to preserve the semantic content of the source image was first proposed by Gatys et al. [15]. In their approach, a CNN pre-trained for object recognition was used to obtain the *content representation* of the content image and a *style representation* of the style image. The *content representation* was obtained from the upper layer activations of the pre-trained CNN for the source image. The *style representation* was represented as a Gram matrix derived from the layer activations for the style image. Then, the network was trained to jointly minimize the distance between the feature representations of the white noise image and both the content and style representations. Thus, an image with features of both representations is produced.

The task of *stuff* painting is related to semantic style transfer, where the goal is to transfer the style of semantic regions in style images to semantically similar regions in the content image (e.g. the style of the sky in the style image is applied to the sky in the content image) [14]. Current work in semantic style transfer build upon NST methods by introducing semantic segmentation maps to guide the targeted transfer of style between semantically similar regions [16, 17, 18]. One drawback of this approach is that a semantic segmentation mask of the content image is required for semantic style transfer. We seek an approach where after training, one only needs to provide the content image itself and without any additional annotation for style transfer.

Stuff painting is also similar to the targeted style transfer problem explored by Castillo et al. [19]. In their work, they introduced a pipeline for transferring style to only specific objects of images. For each pair of content and style images, NST is first performed to generate a stylized content image, after which a semantic segmentation mask is used to identify the object in the content image for which NST is desired. Finally, Markov Random Field based blending is used to merge and blend the original content image and the stylized object image together, with the semantic segmentation mask as a guide. Our work is different in that while they focus on targeted style transfer for *thing* instances, we focus on targeted style transfer for *stuff* regions. We also present a framework that is end-to-end trained, and that does not require semantic segmentation masks for new images after training.

There are also previous works on NST in a GAN setting. [20, 21] focused on style transfer in a non-targeted setting. There have also been works for tasks that are formulated as targeted style transfer problems in a GAN setting, such as make-up style transfer [22] and face aging [23]. Both of these focused on targeting human faces for style transfer, whereas we are interested in style transfer for *stuff* regions. Also, human faces are usually pre-processed such that all the faces are in the same posture, position, and

have the same size. Such pre-processing cannot be done in style transfer for *stuff* regions as we work on the entire image at once.

1.2.2 GANs In Semantic Image Segmentation

The key to *stuff* painting lies in the ability of the model to segment out the desired image regions for the transfer of painting styles. This can be seen as a form of implicit semantic segmentation.

Image segmentation is the task of dividing an image into parts. It has wide applications in many fields, ranging from medical imaging, object recognition, to face recognition [24]. Semantic segmentation, also known as object segmentation, is an area of image segmentation that seeks to segment images into regions that completely and only encapsulates objects of interest [25]. It is represented as a dense labeling task, where the aim is to predict a class label for every pixel in a given image input [26].

The ability of GANs in performing semantic segmentation was first demonstrated by Luc et al. [26]. They noted that the use of an adversarial network with a field of view over the entire image can assess the joint configuration of multiple label varieties, which lets the model enforce higher-order consistency in a form that cannot be enforced with other forms of loss functions. In addition to the adversarial loss typical of GANs, they also used a multi-class cross-entropy loss that is commonly used in semantic segmentation models [26, 27, 28, 29, 30].

A commonly seen application of semantic segmentation with GANs is in segmenting medical images, such as brain MRI [31], spinal structures [32], anatomy imaging [33], and brain tumour segmentation [34, 35]. Other applications include segmentation of satellite images [36] and cityscape images [37].

There have been a variety of GAN architectures proposed for semantic segmentation. Many of them utilize a combination of adversarial loss and an additional loss function, such as L1 loss [36], mean cross-entropy [31], and

multi-class cross-entropy [26]. [37] and [38] replaced the usual discriminator that produces just a binary output with fully convolutional discriminators that outputs segmentation maps for training. Joyce et al. [33] proposed a multi-adversarial architecture for semantic segmentation, where more than one discriminator is used in training. One of the discriminator played the usual role of discriminating ground truth data from fake generated data, while the other discriminator introduced a reconstruction loss to encourage correspondence between the input and output images. An interesting application of GAN in semantic segmentation is introduced in [39], where a GAN is pre-trained to translate ground truth segmentation masks to their original content image. This pre-trained GAN then acts as a form of loss function to a segmentation network, where the segmentation network is encouraged to produce segmentation maps that the pre-trained GAN can use to translate effectively back into the original content image.

1.3 Thesis Organization

In this thesis, we first present a literature review on neural networks and GANs in chapter 2 to familiarize the reader on the GAN frameworks that this thesis is based on. Following that, we note that there is a lack of available training data for partial painting of *stuff* classes and so we present a data generation pipeline for such training data in chapter 3. After generating the training data, we can start our experiments. In chapter 4, we present the results of training the popular Pix2Pix GAN framework to perform partial painting of *stuff* classes. Afterwards, in chapter 5, we introduce SPGAN, which is an improved GAN framework for performing partial painting of *stuff* classes. Finally, we conclude this thesis and briefly discuss possible future work in chapter 6. The objectives of each chapter are listed in Table 1.1.

Table 1.1: Objective of respective chapters

Chapter	Objective
1	Introduction
2	Literature review
3	Data generation pipeline
4	Stuff painting with Pix2Pix
5	Stuff painting with SPGAN
6	Conclusion and future work

1.4 Contributions

This master’s thesis introduces the following contributions:

1. In chapter 2, we provide a literature review on the GAN framework and the state of the art of GANs in the field of image translation.
2. In chapter 3, we introduce a flexible pipeline through which a dataset of partially painted images can be created. We also describe the dataset and software that was used in the pipeline.
3. In chapter 4, we present the results of our experimentation on automatically painting the *stuff* classes in images with GANs. We also provide an analysis on the effects of the chosen architecture on our training results.
4. In chapter 5, we present a GAN framework for the partial painting of *stuff* classes, which incorporates an additional semantic segmentation discriminator for improving segmentation accuracy of the generator network. We also introduce a Gaussian failure correction to improve segmentation learning. We also demonstrate that our model outperforms the quality of the partially painted images as compared to a baseline GAN.

Chapter 2

Related Work

2.1 Artificial Neural Networks (ANN)

Drawing inspiration from the structure and mechanics of the neural networks found in biological brains, Artificial Neural Networks (ANN) is a computational model designed to replicate and mimic this architecture for processing information.

A lot of excitement and interest was generated in Machine Learning (ML) research with the onset of ANNs. Breakthroughs in areas such as Computer Vision, Text Processing and Speech Recognition led to wide acceptance of ANNs, helping it gain widespread use in industrial applications.

2.1.1 Artificial Neurons

The simplest and most fundamental unit of computation in an ANN is a Neuron, sometimes also referred to as a unit or node. It is inspired by the mechanics of a biological neuron which works by deciding whether to fire off an output (to activate), given its current input. Mathematically, an artificial neuron is a function that maps a series of inputs to a single output, as follows:

$$\text{Output } Y = \varphi \left(\sum_{i=0}^n w_i x_i \right)$$

It calculates an output indicating its activation, Y , by accepting inputs from either other nodes or from other sources such as external inputs given by a user. Together with each source of input $X = \{x_0, \dots, x_n\}$ is an associated weight $W = \{w_0, \dots, w_n\}$ which indicates the significance of that particular input to this neuron. The weights are usually network parameters that will be learned when the network is being trained. The function φ is called the *activation function*, and is applied on the weighted sum of the inputs to generate the output.

Usually, another input with a constant value of 1 and a learnable weight of b , called the *bias input* will also be present. The *bias input* allows for the activation threshold of the neuron to be adjusted as needed during the training process. An example is shown in Figure 2.1. In the example, a bias of -1 shifts the activation graph to the left, which means that lower values of the inputs are needed for the neuron to activate. Likewise, a bias of 0 leaves the graph unshifted, where any positive input will cause the function to activate, and a bias of 1 shifts the activation graph to the right, which means that higher values of the inputs are needed for the neuron to activate.

2.1.2 Activation Functions

The activation function φ is non-linear and introduces non-linearity into the network outputs. Real-world data, such as the human speech patterns, are often complex and non-linear, which implies that neural networks that model complex real-world data are also non-linear. Hence, by stacking together layers of activation functions in neural networks, the goal is that these non-linear activations can be trained to jointly model any distribution of complex real-world data.

Each activation function takes in a single value as an input, which is the

Figure 2.1: Adjusting the bias will affect the activation threshold of the activation function.

weighted sum of the inputs together with the bias term which is then used to calculate an output. Some of the activation functions that are commonly used in practice are Tanh, Sigmoidal, ReLU, Leaky ReLU. Their mathematical definitions are as follows:

$$\text{Tanh} : \tanh(U) = \frac{1 - e^{-2U}}{1 + e^{-2U}}$$

$$\text{Sigmoid} : \sigma(U) = \frac{1}{1 + e^{-U}}$$

$$\text{ReLU} : f(U) = \max(0, x)$$

$$\text{LeakyReLU} : f(U) = \begin{cases} x & \text{if } x > 0 \\ 0.01x & \text{otherwise} \end{cases}$$

2.1.3 Feed-Forward Neural Network

One of the earliest and simplest type of neural network architecture was the Feed-Forward Neural Network (FFNN). This network architecture is divided into layers, with each layer comprising multiple neurons. These layers are stacked on top of each other, with each layer taking as inputs the outputs of the previous layer.

There are three types of layers in an FFNN architecture: the Input Layer, the Hidden Layer and the Output Layer.

The Input and Output layers have straightforward roles. The Input layer serves as nodes that provide input to the network. Neurons in the input layer take in data from an external source and forwards it to the next layer in the network. There are typically as many input nodes as the dimensions of the input data, with each neuron responsible for passing a single dimension of data. The Output layer produces the outputs of the network. This layer is responsible for producing the desired outputs of the network, such as a 0 or 1 for a network meant for performing binary classification.

The Hidden Layer is so named as they are “hidden” from external interactions i.e. they only interact with other layers in the network. Hidden layers are primarily responsible for the computation of outputs based on their own inputs. By learning suitable weights for their inputs and outputs, hidden layers provide the ability of the model to approximate a desired distribution.

Figure 2.2: Feedforward network consisting of an input layer, one hidden layer, and one output layer.

There can be multiple hidden layers between the input and output layers. Generally, having more hidden layers increases the ability of the network to approximate any distribution, but this increases the size and training time of the network. It might also result in overfitting, where the model learns to perfectly model the distribution of the training data but cannot generalize the model to unseen data.

The most basic FFNN is known as the Single-Layer Perceptron. It has no hidden layers, and the ability of the network to model distributions comes from the weights associated with the connections between the Input and Output Layers. A more commonly seen network architecture is the Multi-Layer Perceptron (MLP), which contains at least one hidden layer, as shown in Figure 2.2. This gives it a stronger ability to model data distributions, which makes it a better fit in practical applications.

2.2 Deep Convolutional Neural Networks

2.2.1 Multilayer Perceptrons

The concept of a deep neural network stems from the architecture of an MLP. As more hidden layers are added to an MLP, the network is said to get "deeper", hence the terms "deep neural network" and "deep learning".

By adding layers, the network is able to learn a hierarchy of features across various scales and resolutions. For example, in a network for facial recognition, the first few layers would be trained to recognize lower level features such as various lines and basic shapes. The upper layers would then combine these features into higher level features, such as recognizing the shape of a person's nose, which would then provide enough information for the last layers to perform facial recognition.

In essence, neural networks can be viewed as models that define an arbitrarily complex function g , where

$$Y = g(X)$$

where X is the input vector to the neural network, and Y stands for the desired output of the network given an input X .

The process of training an MLP consists of repeated cycles that comprise a forward and a backward pass. Intuitively, this is akin to letting the network guess an output when given an input, then telling the network how far it was from the right answer and letting it learn. The forward pass is the guessing phase, and the backward pass is the learning phase.

Before the training begins, all weights in the network are stochastically initialized. During the forward pass phase (also known as the forward propagation phase), the network is given an input, and from the input layer onwards, each layer takes in the input of the previous layer and calculates an output, which is passed to the next layer. In this way, the input is propagated

through the layers until an output is produced from the output layer.

2.2.2 Backpropagation

The backward pass phase, better known as the backpropagation phase, is when learning takes place in the neural network. The backpropagation algorithm [40] can be said to be the key to the success of ANNs. This phase is when the network learns from its guess during the preceding forward pass.

The first step in learning is to determine how "wrong" the guess of the network is. For this purpose, a cost function is used to calculate the error that the network is making. The aim of training is to find a set of parameters (weights) that will minimize this cost function. A common use of neural networks is in classification, where the network will attempt to label a given example with a certain class. In classification tasks, a common cost function is the categorical cross-entropy,

$$\text{Categorical Cross Entropy} = J(W, b) = - \sum_{n=1}^N \left(\sum_{c=1}^C y_{n,c} \log(p_{n,c}) \right),$$

where N is the number of training data instances, C is the number of class labels, $y_{n,c}$ is a binary indicator (0 or 1) which indicates whether the class c is the correct class for data instance n , and $p_{n,c}$ is the class that the network predicts for data instance n .

Minimizing the cost function is done through adjusting the network weights, which is performed using an algorithm called gradient descent. During gradient descent, the gradient of each weight against the cost function is calculated, and the weight is adjusted such that it takes a step down the direction of the steepest gradient, i.e. the weight is changed such that the error is reduced for that particular training example. The backward and forward passes usually repeat until the error reaches a minimum through this approach. The

error has converged.

By differentiating the cost function by each weight, the desired gradient can be found. The weights are then updated by the value of the gradient, as follows:

$$W_k = W_k - \alpha \frac{\partial J(W, b)}{\partial w_k},$$

where α denotes the learning rate, which determines the size of the step to take. A learning rate that is too small will result in the training taking too much time, whereas a learning rate that is too big might cause the error to oscillate around the minimum and never converge. A common approach is to start with a big learning rate, then reduce it as learning proceeds to gain the advantages of both big and small learning rates.

2.2.3 Common Layers In Neural Networks

There are many different types of layers used in Neural Networks, each with their own advantages, disadvantages, and useful applications. This section will introduce some of the more commonly used types of layers used in image processing with Neural Networks.

Fully Connected Layers

The fully connected layer, also known as the Dense Layer, is a layer where every node in the layer is connected to every node in the previous layer, hence the names. Therefore, every node in this layer can potentially have a full view of all the features calculated in the previous layer, since each node in this layer receives input from every node in the previous layer.

However, due to the fact that the weights associated with each connection is likely different, each node in the layer will receive a different combination of the outputs of the previous layer, as illustrated in Figure 2.3. This helps to enable each node to specialise in detecting specific combinations of features.

The fully connected layer is often the last layer of neural networks for

Figure 2.3: Fully connected layer. Different thickness of each connection denotes the relative magnitude of the weights associated with each connection.

classification, as this layer can look at the entire high level feature set calculated by the previous layers and make decisions related to the classification of the data instance. For example, in a convolutional neural network, the convolutional layers would serve to detect features present in the image, and based on the detected presence or absence of these features as reported by the convolutional layers, the fully connected layer can then determine if a certain object is present in the image.

Fully connected layers can also be present between other layers in a neural network for the purpose of introducing non-linearity into the network. This is an important feature due to the fact that adding any combination of linear functions will still result in a linear function. This means that having many layers of linear layers (such as convolutional layers, as explained in a

later section) will only produce results that are achievable by a single linear layer, which is insufficient to model complex real world data. Introducing non-linearity between these layers then gives the model the ability to model increasingly complex features across layers.

Convolutional Layers

The convolutional layer is the core of Convolutional Neural Networks(CNNs). Most, if not all, neural networks that are used in image processing tasks are CNNs, and the convolutional layer is the key behind the CNNs ability to work with images.

The convolutional layer differs from other layers in that it is based on the convolution operation. The convolution operation consists of applying a convolution kernel (also known as filters), across a matrix of values. The convolution kernel is itself a matrix of values called *weights*. As illustrated in Figure 2.4, applying a single convolution is to compute the dot product between the convolution kernel and a portion of the target matrix, equal to the size of the convolution kernel. In a full convolution operation with an image representing as a matrix of pixel values, the kernel will slide across the image, computing a convolution at each step of the slide. The slide is controlled by a step size, which controls how many spaces to move at each step of the slide.

The result of a full convolution is another matrix, with each value of the matrix indicating the response of the kernel at a corresponding spatial location in the image. This matrix is known as the activation map. With this activation map, the next layer can see which part of the image contains the particular feature that the particular kernel has learnt to detect. As seen in Figure 2.4, the output of a convolution is usually of a reduced size as compared to the input, with the degree of reduction determined by the step size. If it is desired that the size of the input is preserved, one can pad the images such that the resultant output is of the same size of the input.

Figure 2.4: A 2×2 convolution kernel is applied across a sample 3×3 , image with step size 1, to generate an output. The resulting activation map is of size 2×2 .

Mathematically, we can express the convolution operation as

$$g(x, y) = w * f(x, y) = \sum_{s=-a}^a \sum_{t=-b}^b w(s, t) f(x - s, y - t)$$

where $g(x, y)$ denotes the resultant filtered matrix, $f(x, y)$ denotes the input image, and w denotes the filter kernel. s and t denote the coordinates of each element in the convolution kernel and $-a \leq s \leq a$, $-b \leq t \leq b$.

Each convolution layer typically consists of many kernels. The values of each kernel matrix, also known as *weights*, are learnt during backpropagation, therefore after learning, each kernel in the layer will learn to detect different features in the image. A CNN typically contains many convolution layers. By stacking convolution layers, a CNN is able to detect increasingly complex features. For example, the lower layers will learn to detect the locations of various lines and edges in the image, whereas the following layers will be to look at the presence/absence of the lines and edges to detect increasingly identifiable shapes and patterns such as circles, a human nose, or alphabets.

In Figure 2.5, we can see an example of how stacking convolution layers enable CNNs to detect complex features such as human faces. From layer

Figure 2.5: A visualization of activation maps in a CNN for detecting human faces. Images are obtained from [41].

1 to layer 3, we can see that increasingly recognizable shapes emerge in the activation maps.

Pooling Layers

Convolutional layers are computationally expensive and also have a high spatial demand due to the large number of weights that need to be learnt in each layer. One way to mitigate the costs of convolution layers is to introduce pooling layers after convolution layers.

Pooling layers are layers that aggregate a matrix of inputs into a single value. Similarly, to convolution layers, pooling layers also consists of filters. However, these filters do not perform convolutions. Instead, these filters slide across input matrices and aggregates the values of the input matrices into a

Figure 2.6: An example of max pooling. The kernel has dimensions 2×2 . At each step of the slide, the largest value under the kernel is chosen as output for the kernel.

single value according to certain rules. The main role of the pooling layer is to reduce the size of the intermittent representations in a network. This would then reduce the number of weights and computations in the network.

There are a few common variations of pooling layers, each differing in *how* its filters aggregate values, such as L2-Norm pooling, Average Pooling, and Max Pooling. Regardless of the method of pooling, the intuition behind the pooling layer is that the *exact* spatial location of a feature is not as important as its spatial location *relative* to other features.

The most commonly used type of pooling is Max Pooling. In max pooling, the **largest** value under a filter is selected as output, as illustrated in Figure 2.6. Also, notice how the pooling operation results in a matrix with reduced size. Even though it seems that information is lost during pooling operations, in practice it has been shown that networks with pooling layers can still produce accurate results while reducing their computational and spatial requirements.

On the other hand, in recent times researchers have raised opinions against using pooling layers. They argue that the problem can be solved by simply

replacing the convolutions with larger strides in the network (sometimes also referred to as strided convolutions). Another advantage that using strided convolutions might have against pooling layers is that strided convolutions allow the layer to learn its own pooling function, as opposed to the fixed function (E.g. Max pooling) that defining a pooling layer would entail.

Transposed Convolution Layers

In neural networks, the transposed convolution layer essentially acts like the reverse of a convolution layer, where it performs a convolution but reverses the spatial transformation effect of a convolution layer.

In CNNs, convolution layers are usually stacked together in order to detect increasingly sophisticated features in images. This allows the CNN to learn a concise and abstract representation of the input image that is of a much smaller dimension than the original image. In other words, the CNN *compresses* the input image down to a concise abstract representation. So what can we do with this abstract representation? CNNs usually add dense layers on top of the stack of convolution layers to perform some form of classification or labeling. Another possible use of this abstract representation is to generate a new image based on this representation. To do so, a key operation would be to somehow *decompress* the abstract representation into our desired image. One popular way of doing this is to pass this abstract input through a series of Transposed Convolution Layers.

An intuitive way to think about a transposed convolution on an input is to imagine the input as resulting from a direct convolution that was applied on an initial feature image [42]. The transposed convolution can then be thought of as the operation that allows the recovery of the shape of this initial feature image. For example, we look at the convolution shown in Figure 2.4, which is a convolution with a 2×2 kernel across a 3×3 input with a stride size of 1. This convolution produces a 2×2 output. If we treat this output as an input to the transpose of this convolution, the resultant

Figure 2.7: An example of applying zero padding on an image to increase the size of the output. The padding is applied both *around* and *between* the pixels of the image. Image is obtained from [42].

output would then be of size 3×3 .

Similar to how we can control the output size of a convolution by applying padding, we can also think of a transposed convolution as a convolution over a padded input, which would result in an output of our desired size. However, in this case, the padding is often applied both *around* and *between* the input values of the input image. An example of this can be seen in Figure 2.7.

The intuition behind this specific way of padding is that the corresponding portions of the output should only be affected by the corresponding input pixel. For example, when applying the kernel on the top-left of the padded image, the kernel should only touch the top-left pixel of the input image. By applying the padding between the pixels, we can maintain this desired connectivity pattern of the resultant convolution.

A common problem with using transposed convolutions are checkerboard artifacts. This artifact results from some parts of the output feature map receiving more input than others, which is mainly due to the uneven overlap of filter kernels. This uneven overlap appears when the kernel size cannot be divided by the stride (e.g. 5×5 filter with a stride 2). One possible way to alleviate this problem is to use kernel sizes and strides that result in an even overlap. However, since deconvolution kernel weights are learnt, it is also possible that kernels with even overlap will end up learning weights that

also result in checkerboard artifacts [43].

A suggested solution to this problem is to replace transposed convolutions with a two-step process that emulates the effects of a transposed convolution, while also avoiding the problem of checkerboard artifacts. Also known as *resize-convolution*, this approach consists of an upsampling step (using methods such as nearest-neighbour interpolation) followed by a normal convolution. Replacing transposed convolution layers have been shown to effectively reduce checkerboard artifacts [43].

Residual Blocks

Despite the breakthroughs in deep learning with deeper and deeper networks, there exist issues and limitations that come with stacking more layers, such as the problem of vanishing/exploding gradients [44, 45], and performance degradation [46, 47, 48]. He et al. [48] proposed the "deep residual learning" framework to address the issues that come with increasing the number of layers in a neural network.

Figure 2.8: A diagram of a residual block. Figure is obtained from [48].

The key building block in a deep residual network is the *residual block*, a diagram of which is shown in Figure 2.8. A residual block is elegantly simple in design. It is essentially a normal activation layer with a skip connection from the input straight to the output. Mathematically, given an input to an activation layer as x and we denote the true distribution that the layer is trying to learn as $H(x)$, the residual is defined as follows:

$$\text{Residual} = F(x) = H(x) - x$$

which can then be rearranged to:

$$H(x) = F(x) + x$$

This means that with the goal of learning the true distribution $H(x)$, the residual block is actually learning the residual $F(x)$, as opposed to conventional layers which attempts to learn $H(x)$. It has been demonstrated [48] that it is easier to learn a residual mapping than the true mapping directly. Also, by adding skip connections, the vanishing gradient problem can be mitigated. This problem stems from the repeated multiplication of gradient values as they flow through the layers, which has a tendency to rapidly approach zero. This causes earlier layers to learn at a much slower rate as they receive smaller gradients. The skip connections allow for larger gradients to be propagated to earlier layers, which helps these early layers to learn much faster and enable the training of deeper networks.

2.3 Generative Adversarial Nets

In recent years, Generative Adversarial Networks (GANs) have exploded in popularity. The concept of a GAN originated from a landmark paper by Goodfellow et al. [4], which described the GAN framework and training methods. The authors were motivated by the difficulty that current deep

models have in the generative context. This is due to the highly complex nature of real images, which results in a low success rate when conventional deep models attempt to generate realistic looking images. The maximum likelihood estimation and related methods that conventional deep generative models usually adopt face difficulties in approximating the probability distribution required to calculate the pixel values of the generated image [4]. The GAN framework proposed is a method that can avoid these difficulties.

Under the GAN framework, models have demonstrated an astonishing ability to generate extremely realistic looking images. Initially, GANs were trained to generate images from random vectors [4], and early research was focused on improving the GAN framework in this area [49, 50]. Despite the huge improvement in image generation ability, there was still a lack of control over the mode of images being produced. Users of the models can only ensure that the images produced are in a certain domain (e.g. images of rooms, dogs, cats) as determined by the training dataset, but is unable to control what *class* of images are produced (e.g. bedroom vs living room, the colour of the dog/cat).

To address this issue, Mirza et al. [51] proposed an extension to the GAN framework, called Conditional GAN (cGAN), where in addition to the usual training data, additional information such as labels are also shown to the GAN during training. This allowed a user to use the additional information to control the outputs of the GAN, and would go on to become the base framework of many other GANs. Alternatively, Chen et al. [52] proposed a way for the GAN to learn a latent code that comprises meaningful components that can be used to effect meaningful changes in the generated images, called InfoGAN. For example, in a GAN trained to output handwritten digits, some latent code components can be a variable that determines which digit is generated, whereas another variable determines the stroke thickness.

With the advent of cGANs, researchers have explored many different methods of conditioning the output of the GAN with a variety of conditioning

Figure 2.9: Brief timeline of notable GAN applications and architectures for image processing tasks.

inputs. Apart from class labels and latent codes, Reed et al. [53] described an architecture where the input was conditioned with text, which enabled users to *describe in text* the desired image. Given that GANs are almost exclusively used for image generation tasks, it is perhaps fitting that the most popular conditioning input can be said to be images.

Using images as conditioning inputs in GANs has led to impressive results in many image processing tasks. Besides image-to-image translation, GANs have also been applied in image processing tasks such as image inpainting [54] and image super-resolution [56]. A brief timeline of notable GAN applications and architectures in image processing tasks is shown in Figure 2.9. A brief summary of some notable GAN architectures in image processing tasks are listed in Table 2.1.

The following sections will provide a review of some notable GAN architectures for the task of image-to-image translation.

2.3.1 Adversarial Nets

Adversarial nets consists of two networks: a generator and a discriminator. The generator aims to produce a realistic image from a random input vector $z \in Z$. by learning a mapping from the input vector space Z to a pixel-level image space. The discriminator is used to guide the generator in achieving

CHAPTER 2. RELATED WORK

Table 2.1: Brief summary of notable GAN architectures and applications.

Architecture	Application	Reference
Adversarial Nets	Unconditional Image Generation	[4]
cGAN	Conditional Image Generation with Labels	[51]
DCGAN	Modifying CNNs for use in GANs	[50]
InfoGAN	Conditional Image Generation with Latent Codes	[52]
Generative Text to Image Synthesis	Conditional Image Generation with Text	[53]
Context Encoders	Image Inpainting	[54]
SRGAN	Image Super-Resolution	[56]
Pixel Level Domain Transfer	Image-to-Image Translation	[58]
Pix2Pix	Generalized Image-to-Image Translation	[5]
CycleGAN	Image-to-Image Translation with Cycle Consistency	[59]

this aim. The aim of the discriminator is to learn to distinguish between real images and fake images produced by the generator. In this manner, the two networks have adversarial aims, with the generator aiming to produce images that the discriminator cannot differentiate from real images and the discriminator aiming to perfectly differentiate between generated fake images and real images. This is analogous to a competition between counterfeiters (generator) and the police (discriminators), where the counterfeiters aim to improve their counterfeiting techniques and the police aim to detect the increasingly realistic counterfeits.

In the training phase, the generator and discriminator are first initialized, denoted as G^0 and D_R^0 respectively. In each training iteration, the discriminator is first trained to distinguish between real images, $\{I^i\}$, and fake images generated by the generator, $\{\hat{I}^j = G^0(z^j)\}$. Afterwards, training

for the updated discriminator D_R^1 is paused and the generator G^0 is trained to produce images that are more realistic and hence, cause the discriminator to wrongly classify as real images. This adversarial process is repeated until convergence. The objective function for an adversarial network can be represented as a minimax objective as follows:

$$\min_{\Theta^G} \max_{\Theta_R^D} \mathbb{E}_{I \sim p_{data}(\mathbf{I})} [\log(D_R(I))] + \mathbb{E}_{z \sim p_{noise}(\mathbf{z})} [\log(1 - D_R(\hat{I}))]$$

where Θ^G and Θ_R^D indicate the weights of the generator and the discriminator respectively. The discriminator outputs a probability that its input is a real image; the output is high when the image is real and low when it is not. The discriminator loss function $\mathcal{L}_R^D$ is defined as the binary cross entropy,

$$\mathcal{L}_R^D(I) = -t \cdot \log[D_R(I)] + (t - 1) \cdot \log[1 - D_R(I)], \text{ s.t. } t = \begin{cases} 1 & \text{if } I \in \{I^i\} \\ 0 & \text{if } I \in \{\hat{I}^j\} \end{cases}$$

Using this framework, a GAN can be trained with a very low level of supervision. In fact, an unlabeled dataset of real images is sufficient to train a GAN to produce realistic images.

2.3.2 Conditional Generative Adversarial Networks (cGAN)

One problem that vanilla GANs face is that it is not possible to determine the *mode* of the image being generated. Therefore, Mirza et al. [51] proposed an extension of the GAN framework. By extending GANs to a conditional mode, it is possible to direct the process of image generation, such as class labels.

To extend an unconditioned GAN to a conditional model, a conditioning input is added to the network. The conditioning input is usually some form of auxiliary information about the desired image, such as the class label of

Figure 2.10: Architecture of a cGAN. Both the discriminator and generator receive auxiliary information as input.

the image. This auxiliary is shown to the generator along with the usual random vector input. During training, the discriminator is also shown the auxiliary information, which allows it to also determine whether the image generated fits the auxiliary information, in addition to the usual real/fake image discrimination. An illustration of this architecture is shown in Figure 2.10. In this manner, the generator is usually a form of autoencoder.

With the addition of conditioning inputs to the network, it is then possible to produce desired outputs by varying the conditioning inputs.

2.3.3 Deep Convolutional Generative Adversarial Networks (DCGANs)

Previously, attempts to scale up the outputs of GANs using CNNs to model images have been unsuccessful, with generated images generally being blurry, noisy, and even incomprehensible. DCGANs [50] are a family of Convolutional GANs with certain architectural constraints that overcome these problems. DCGANs adopted and modified three changes to CNN architectures that resulted in improved training stability across various datasets and allowed for deeper generative models with higher resolutions.

The first change is the elimination of pooling layers that have traditionally been a mainstay of CNNs. The downsampling performed by the pooling layers is instead done by increasing the stride of the convolutional layers,

which results in the same effect. This has the advantage of allowing the network to learn its own spatial downsampling function during training, as compared to using pooling layers where the pooling function is predetermined when designing the network.

The second change is to remove the fully connected layers that usually sit on top of convolutional features. The authors noted previously shown results that indicated that using other layers to replace the fully connected layers, such as global average pooling, have resulted in better performance. Therefore, they adopted a compromise between both approaches where the fully connected layers are removed but not replaced with other layers. Instead, the outputs of the highest convolutional layer are directly connected to the input and output of the generatory and discriminator, respectively.

The third change is to apply Batch Normalization [57] (also known as batchnorm) to the inputs of layers. This normalizes the layer inputs to have zero mean and unit variance. This was introduced to mitigate the effects of poor initialization and also to facilitate gradient flow. This was shown to be an important effect to help the generator start learning and prevent mode collapse, which is a failure mode commonly seen in GANs where the generators only produce a single image. However, batchnorm was not applied to the generator input and discriminator output layer as it was shown to result in model instability.

In addition to these three main changes, the authors also used ReLU activation layers in the generator except for the output layer, which used the Tanh activation function. This was found to allow the model to learn quicker over the colour space of the training distribution. Leaky ReLU activation functions were used in the discriminator, which is different from that used in the GAN proposed by Goodfellow et al. [4]. The authors found this to work well, especially when modeling images with higher resolutions.

Figure 2.11: Example of an image translation task. There are potentially infinitely many output modes, as a piece of clothing can be presented in many angles and poses. Image is obtained from [58].

2.3.4 Pixel Level Domain Transfer

Yoo et al. [58] introduced one of the first applications of GANs to image-to-image translation. They framed this as the problem of *image-conditioned image generation*, where the task is to generate an image belonging to a target domain given an input image lying in a source domain, assuming that the two domains by some form of semantic relation. They were seeking a way of performing image domain transfer on the pixel level, as compared to previous works in image domain transfer that performed the image adaptation on the feature space level. They demonstrated this with the following task: where given an image of a human wearing a certain piece of clothing, the model is to generate an image of just the piece of clothing that the model is wearing, as demonstrated in Figure 2.11.

Another novel point about this GAN framework is the use of an additional discriminator as a way of introducing an additional objective to the training. As seen in Figure 2.11, there are many possible target images for a given input. To solve this problem, the authors introduced an additional discriminator on top of the usual "real/fake" discriminator that encourages the generator to produce realistic looking images. This additional discriminator determines whether the output produced by the generator is semantically

associated with the input image i.e. whether the clothing drawn by the generator is the clothing worn by the human in the input image. By jointly optimizing these two discriminators during adversarial training, they are able to produce images that satisfies both the conditions of being realistic looking and being associated with the input image.

2.3.5 Pix2Pix

In their paper, Isola et al. [5] demonstrated that cGANs can serve as a general solution to various image-to-image translation tasks. A testament of this can be found in the fact that the Pix2Pix software has been used by many users for their own applications without having to tune the model.

The authors were motivated by two points. Firstly, GANs learn a mapping from input image to output image *and* a loss function for the current task. Since different problems can require different loss functions, being able to *learn* a loss function is a very desirable trait for generalization. Secondly, the output image of a cGAN is *conditioned* on the input image, as opposed to an unconditional GAN, where the input is often just a random vector. This is the key feature that lets a GAN perform image-to-image translation tasks. This is implemented by also letting the discriminator observe the input image, as shown in Figure 2.12. This allows the discriminator to also learn the association between input and output images, which in turn forces the generator to also learn an association between input and output image.

Noting that previous applications of cGANs utilized methods specific to the task being performed, the authors employed a general framework that did not optimize for any specific task.

The authors also introduced a novel discriminator architecture that they dubbed the PatchGAN. The authors noted that traditional L1 losses can encourage the generator to accurately model low-frequency structure of the images, but cannot encourage the generator to model high-frequency details of the images, which results in blurry structure. Therefore, the authors used

Figure 2.12: Conditioning an output image of the GAN based on an input image. Image is obtained from [5].

a discriminator that focused on encouraging the accurate modelling of high-frequency details, while relying on L1 loss to encourage the modelling of low-frequency structure. The novel discriminator only tries to predict whether an $N \times N$ patch of its input image is real or fake. This patch level discriminator is then applied convolutionally across the whole input image, with the final output being an average of all the individual patch level predictions. The authors have demonstrated that high quality results can be generated even if N is much smaller than the input image size.

For the generator, the authors utilized a “U-Net”[12] architecture. This is essentially an encoder-decoder network that also includes skip connections between corresponding layers in the encoder and decoder parts of the network, as shown in Figure 2.13. In a typical encoder-decoder network, as the input is downsampled through the encoder side of the network until it passes through a bottleneck layer to the decoder part of the network, low-level information such as the location of certain kinds of edges in the original image is usually lost, and only high level information is retained. Therefore, skip connections that allow low level information to be passed is introduced to let the network make use of both high level and low level information when generating images.

Another unique point about the Pix2Pix model is that during the in-

Figure 2.13: Illustration of an Encoder-decoder architecture(left) vs a U-Net architecture(right). Image is obtained from [5].

ference phase, the generator is executed in the same manner as during the training phase, i.e. dropout is still applied, and batch normalization is performed using test batch statistics instead of the training batch aggregated statistics. The authors note that this approach has been previously demonstrated to be effective for image generation tasks.

2.3.6 CycleGAN

The CycleGAN attempts to tackle the problem of Image-to-Image translation while not having paired training data. It is often an expensive and difficult task to obtain paired training data, where every input has a labeled corresponding output. Zhu et al. [59] proposed a method of training a GAN for image-to-image translation between two image domains without the need for specifically paired data. The authors instead seek to learn a mapping between images in two different but related domains only via supervision on the level of sets, i.e. only two sets of images are needed, one for the source domain X and one for the target domain Y .

Supervision only on the level of image sets can lead to cases where the output, $\hat{y}$, that is generated from an input x , $x \in X$, belongs to Y , but $\hat{y}$ holds no meaningful relation with x . To overcome this, the authors added a

Figure 2.14: Results exhibiting cycle-consistency in CycleGAN. (a) The original source input image x . (b) The output image $G(x)$. (c) The reconstructed source image that is constructed from the output image $F(G(x))$. Image is obtained from [59].

constraint deriving from the fact that translations should be *cycle-consistent*. Intuitively, this is similar to language translation, where translating a sentence from English to Chinese, and then back into English, the original sentence should be obtained. Mathematically, given two translation functions $G : X \rightarrow Y$ and $F : Y \rightarrow X$, then it should follow that $G(F(y)) = y$ and $F(G(x)) = x$. An example of the aim of cycle-consistency can be seen in Figure 2.14. This means that the CycleGAN actually contains two generator networks, one for each translation function, and two discriminator networks for adversarial training of each translation function, as shown in Figure 2.15.

The authors enforce *cycle-consistency* in the CycleGAN model by adding the following loss term, called the *cycle-consistency loss*:

$$\begin{aligned} \mathcal{L}_{cyc}(G, F) = & \mathbb{E}_x \sim p_{data}(x) [\| F(G(x)) - x \|_1] \\ & + \mathbb{E}_y \sim p_{data}(y) [\| G(F(y)) - y \|_1] \end{aligned}$$

The *cycle-consistency loss* is combined with the adversarial losses for both

Figure 2.15: (a) Framework of CycleGAN, where given two image domains, source domain X and target domain Y , a GAN is trained for translation in each direction. (b) Forward cycle-consistency loss. (c) Backward cycle-consistency loss. Image is taken from [59].

G and F to produce the full object for the model:

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

The relative weight of the adversarial loss and the cycle-consistency loss is controlled by λ . The training is done with the aim of solving:

$$G^*, F^* = \arg \min_{G, F} \max_{D_X, D_Y} \mathcal{L}(G, F, D_X, D_Y)$$

Training with this objective has enabled CycleGAN to achieve some impressive results, as shown in Figure 2.16.

2.4 Summary

In this chapter, we have briefly presented some notable background knowledge on Artificial Neural Networks and Deep Convolutional Neural Networks. We have also reviewed some notable developments in Generative Adversarial

Figure 2.16: Some applications of CycleGAN in translating images between two domains. (a) Between Monet styled paintings and photorealistic images. (b) Between zebras and horses. (c) Between summer and winter scenery. Images are taken from [59].

Networks, such as Adversarial Nets, cGANs, DCGANs, Pixel Level Domain Transfer, Pix2Pix, and CycleGAN. We discussed how GANs started from the groundbreaking Adversarial Nets architecture, which was applied in a non-conditional setting. Following that, cGANs allowed GANs to be applied in a conditional setting. Then, DCGANs saw the incorporation of convolutional neural networks in GANs. Together, these developments allowed for the use of GANs in image to image translation, as demonstrated in Pixel Level Domain Transfer, Pix2Pix, and Cycle GAN.

Based on these developments, the following chapters will present the application of GANs in the task of *stuff* painting, which can be seen as a special case of image to image translation. We will first demonstrate the results of applying the Pix2Pix architecture for this task, as it has been shown to serve as a general solution for image-to-image translation tasks. Afterwards, we will present some improvements to the architecture to improve the performance of the GAN in *stuff* painting.

Chapter 3

Experiment Data Preparation

This chapter will describe the dataset used for the experiment, and the process through which the training datasets are prepared for the experiment.

3.1 Data Preparation Process

Training data is one of the most important aspects of any machine learning model. Without the right training data, nothing useful can be done with even the most sophisticated models. Since we are seeking an automated approach to *stuff* painting, we need appropriate data, which is images whether only the *stuff* classes take on a painterly aspect. However, there is a lack of such data as far as we know, therefore we have developed a pipeline for generating such data. This chapter will describe this data generation process.

3.1.1 COCO-Stuff Dataset

The COCO-Stuff [8] Dataset is an image segmentation dataset extended from the popular Microsoft COCO [60] dataset, which is also an image segmentation dataset. The COCO-Stuff dataset focuses on *stuff* detection whereas the COCO dataset focuses on *thing* detection.

Figure 3.1: Images and segmentation masks from the COCO-Stuff dataset. Images are taken from [8].

Conventional image recognition and segmentation tasks focus on detection and segmentation of objects or *things* [9]. In contrast, the COCO-Stuff dataset is focused on *stuff* detection and segmentation. The COCO dataset contains 118k pairs of images and their corresponding *thing* instance segmentation masks. The dataset contains segmentation masks for 80 *thing* classes. The COCO-Stuff dataset extends the COCO dataset with segmentation masks for 91 classes of *stuff*, an example of which can be seen in Figure 3.1. The 91 classes are arranged in a hierarchy of three levels, ranging from broad supercategories (indoor vs outdoor) to type supercategories (wall, water, plan, etc.) to the individual classes (wall-marble, river, grass, etc.).

3.1.2 Data Generation Pipeline

To train a GAN to produce partially painted images, we need to provide partially painted training images in order for the GAN to learn the desired painting style and the areas that should be painted and which to leave unchanged. There is no such dataset publicly available as far as we know, therefore we introduce a partially painted image creation pipeline, with which we

Figure 3.2: The hierarchy of stuff classes in COCO-Stuff dataset. (a) The indoor categories and supercategories. (b) The outdoor categories and supercategories. Images are taken from [8].

have created a dataset of images where the *stuff* classes are painted and the main subjects of the pictures are photorealistic. This section will describe the process by which the partially painted dataset is created.

The partially painted dataset is generated using the aforementioned COCO-Stuff dataset and NPR software. The process consists of 2 stages, namely the painting stage, and the stitching stage. A diagram of the generation process is shown in Figure 3.3. The process is flexible, where different components of the pipeline can be changed, but the overall process will still be the same. For example, a different painting mechanism can be used for different styles, or even a different way of re-rendering the image can be done, depending on the desired effect. The final product will still be an image that is partially

re-rendered.

Figure 3.3: Partially painted image dataset generation pipeline.

In the first stage, which is the painting stage, the original images are

rendered in a non-photorealistic style using NPR software. This stage will determine what the painted regions of the final partially painted images will look like.

In the next stage, which is the stitching stage, the painted background is combined with the non-painted foreground from the original image to produce the partially painted image. To differentiate between the *stuff* and the *things* of each image in the COCO-Stuff dataset, we used the manually annotated segmentation masks included in the dataset. Using the masks, we can determine which pixels in the image belong to *thing* classes and which pixels belong to *stuff* classes. Therefore, we use the segmentation mask as a guide to determine which pixels of the partially painted image come from the painted image and which pixels come from the original image. The general algorithm for the stitching stage is described in Algorithm 1.

Algorithm 1: Stitching painted and non-painted images

Data: originalImage, paintedImage, segmentationMask

Result: partiallyPaintedImage

```

thingColour = colour value of thing class in segmentationMask;
/* Initialize output image with zeros */
partiallyPaintedImage = zeros(size(originalImage));
for coordinates  $(x,y)$  in original image do
    if segmentationMask[x,y] == thingColour then
        | partiallyPaintedImage[x,y] = originalImage[x,y];
    else
        | partiallyPaintedImage[x,y] = paintedImage[x,y];
    end
end

```

This is a flexible pipeline process that is easily modifiable and where the components can be easily swapped out to achieve different effects in the final dataset. For example, different datasets with different desired characteristics can be used in the same pipeline, as long as annotations indicating the region of the image to be re-rendered is present.

The painting algorithm in the painting stage can also be swapped out for other forms of re-rendering, such as filtering or blurring. For example, given a set of facial images and related annotations, a portion of the face can be automatically isolated for smoothening effects to be applied, resulting in a dataset that can be used to train a model for facial beautification.

The pipeline can even be stacked with itself to make images with multiple effects in different regions of the image, given sufficient image annotations. The output of one pipeline can be fed into another instance of the pipeline for the re-rendering of a different portion of the image.

3.2 NPR Software

The modular nature of the data generation pipeline means that any NPR software can be used in the painting stage to generate images rendered in any desired styles. In this work, we will look at the performance of *stuff* painting for images rendered in two kinds of painting styles. The first kind of painting style is the texture-based style, where the painting style is based on textures such as dots and brushstrokes. The second kind of painting style is the non-texture-based style, where the rendering is more abstract and cannot be expressed as the application of a single texture.

Any NPR software is suitable as long as the required effects are achieved. We use the Painterly software to render images in texture-based styles due to its ability to apply a number of different textures to images. For non-texture based styles, there are many different style transfer software that can be used to apply non-texture based styles, such as Pix2Pix [5], CycleGAN [59], and CartoonGAN [61]. We use CartoonGAN [61] for rendering images in cartoon styles, as there are pre-trained CartoonGAN models readily available.

3.2.1 Painterly: Painting Generators

The Painterly software is a program made to perform painting-like Non-Photorealistic Rendering of input images. Given an input image, the program can convert it to a painting-like image. Equipped with a variety of customizable "paintbrushes", it is a versatile program that produces different art styles. It features both an automatic mode that automatically converts the entire image and a manual mode that allows the user more fine-grained control over the painting process.

The Painterly software works by sampling the input picture to produce painting-like approximations of the picture. During the conversion process, an instance of the selected "paintbrush" will randomly traverse the image and apply "brush strokes". At each point of application, the corresponding point in the input image is sampled to determine the colour of the brush stroke. The shape of the brush stroke depends on the paintbrush currently used.

Depending on the paintbrush used, a variety of painting styles can be approximated and applied to the image. As the manual application of strokes to specific regions of the image is also possible, multiple styles can be applied to a single image. Figure 3.4 demonstrates some of the styles possible with Painterly. For example, with the oval brush, the image can be converted to a pointillist painting, which is a painting style where distinct dots of colour are used to compose a painting.

3.2.2 CartoonGAN

The CartoonGAN is a GAN framework that specializes in the cartoonization of images. As cartoon styles have unique characteristics like the simplification and abstraction of image details and the tendency to use clear outlines, simple textures, and smooth/flat colour shading[61], two novel losses that aim to preserve semantic content and clear edges were incorporated into the

Figure 3.4: The Painterly program can render images in various different NPR styles. Input image in top row. Three different styles generated in bottom row. (a) Coarse dotted brush. (b) Fine dotted brush. (c) Waterdrop brush.

framework. An initialization phase to improve network convergence was also introduced. The authors have also released pre-trained CartoonGAN mod-

els¹ for transforming images into the iconic styles of some famous animators. Figure 3.5 shows cartoonizing examples in these 2 styles by Chen et al. [61].

Figure 3.5: Cartoonization examples in the styles Makoto Shinkai and Miyazaki Hayao. (a) Original images. (b) Makoto Shinkai style cartoonization. (c) Miyazaki Hayao style cartoonization. Images are taken from [61].

3.3 Summary

In this chapter, we have briefly described the COCO-Stuff Dataset, from which we will generate our training datasets for our experiments. We have also described the data generation pipeline through which we generate partially painted training datasets from the COCO-Stuff Dataset. Finally, we

¹<https://cg.cs.tsinghua.edu.cn/people/Yongjin/Yongjin.htm>

briefly describe the NPR software that we use in the image painting stage of the data generation pipeline. In the next chapter, we will describe our initial experiments at training the Pix2Pix GAN framework to perform image *stuff* painting and report on the results.

Chapter 4

Stuff painting with GANs

The task of image *stuff* painting is a novel one, with very little previous research in this area. Therefore, we perform some initial experiments to determine the performance of an existing framework in this task. In this chapter, we will present the results of training the popular Pix2Pix network to perform image *stuff* painting. We choose the Pix2Pix framework as it has been shown to be a good general solution for image-to-image translation tasks. We will also analyze and discuss the results of these experiments.

4.1 Experiment Setup

4.1.1 Network Architecture

We adopt the framework of the Pix2Pix network [5] to train a model for painting backgrounds, as illustrated in Figure 4.1. For convenience, in this thesis we will refer to this framework as **Pix2Pix-Stuff**.

Generator

The generator is a U-Net consisting of stacked Convolution-BatchNorm-ReLU layers to form both the encoder and decoder parts of the net. For

Figure 4.1: The network architecture of Pix2Pix-Stuff, consisting of a generator and a discriminator.

each layer, the dropout rate is 0.5. In the encoder portion of the U-Net, each convolutional layer results in downsampling by a factor of 2 and in the decoder portion of the U-Net, each convolution layer results in upsampling by a factor of 2. Also, the encoder portion of the U-Net uses Leaky ReLU layers whereas the decoder portion of the U-Net uses normal ReLU layers.

Discriminator

The discriminator uses a 70×70 PatchGAN, which looks at an input image one patch at a time. The output from each patch of the image is then averaged to be the final real/fake prediction of the input image. The PatchGAN is also composed of Convolution-BatchNorm-ReLU layers, with the exception of the first layer where BatchNorm is not applied. After the final such layer in the network, a convolution is applied to map to a single dimensional output. This output is then passed through a Sigmoid function to obtain the final output of the network for that particular image patch.

Loss Function

A hybrid loss function was used, as it was noted in previous research [5, 62] that combines the GAN objective with a more conventional loss, such as the L1 norm or L2 norm, benefited performance. Therefore in this framework, a

hybrid loss combining the standard GAN objective and L1 norm was adopted.

The GAN objective in the conditional setting is defined as

$$\begin{aligned}\mathcal{L}_{cGAN}(G, D) = & \mathbb{E}_{x,y} [\log D(x, y)] \\ & + \mathbb{E}_{x,z} [\log(1 - D(x, G(x, z)))]\end{aligned}$$

which the generator G tries to minimize while the discriminator D tries to maximize. The conditional setting is reflected here as D takes in x as input as well as the ground truth and generator output i.e. $D(x, y)$ and $D(x, G(x, z))$ are used instead of $D(y)$ and $D(G(x, z))$.

The L1 distance is defined as

$$\mathcal{L}_{L1}(G) = \mathbb{E}_{x,y,z} [\| y - D(G(x, z)) \|_1]$$

L1 loss was chosen as it encourages less blurring in the output image. The final hybrid objective is then defined as

$$G^* = \arg \min_G \max_D \mathcal{L}_{cGAN}(G, D) + \lambda \mathcal{L}_{L1}(G)$$

The relative importance of the L1 loss relative to the standard GAN loss can be adjusted by the λ term.

4.1.2 Training Data

Data Selection

Previous results [5, 59] have indicated that training a GAN can require as little as 400 training images. The painting process is time-consuming, with a heuristically derived average of at least 7 seconds required per image for a reasonably well-painted image, which means that trying to paint the entire dataset of 118k images will require an estimated 229 hours. Therefore, for this study, we randomly chose 2000 training images from the COCO-Stuff dataset for processing into partially painted images for training.

The large variety of thing and stuff classes that the dataset contains might also prove to be too noisy for the GAN to learn a good representation of the image domain. In image-to-image translation, GANs have generally been trained to perform translation from between specific domains, such as between horses and zebras [59], and dogs and cats [63]. Therefore, we also impose the constraint to the data selection process that the images must contain at least one stuff class from at least one of these supercategories: *water, sky, ground, plant*. Since we are interested in training the model to only recognise and re-render the background, we do not place any restrictions on the thing classes that the images contain. In fact, a dataset with a large variety of thing classes would be preferable, as the model could then learn to focus only on the stuff classes and ignore the thing classes.

Finally, we randomly selected 2000 images for training and 200 images for testing from the images that fit this constraint.

Partially Painted Datasets

We evaluate our model on four datasets containing partially painted images for two texture-based styles and two non-texture based styles. We generated these four datasets using the COCO-Stuff dataset and a suitable NPR software along with the aforementioned data preparation process. For the texture-based style, we use the Painterly software to render the source images in a pointillist style and brushstroke style. For the non-texture-based style, we use a pretrained CartoonGAN model to render the source images in cartoon styles modeled after the works of the animators Miyazaki Hayao and Makoto Shinkai. We then used these painted images to generate partially painted images, which we then used to form four datasets of partially painted images, termed **Stuff-Point** (Pointillist style), **Stuff-Brush** (Brushstroke style), **Stuff-Shinkai** (Makoto Shinkai style), and **Stuff-Hayao** (Miyazaki Hayao style). All four datasets are generated from the same set of images. A summary of the generated training dataset is presented in Table 4.1.

Table 4.1: Summary of generated partially painted datasets.

Styles	Pointillist, Brush, Hayao, Shinkai
Size	2000
<i>Stuff</i> supercategories	Water, sky, ground, plant
No. of allowed <i>stuff</i> categories	27
No. of allowed <i>thing</i> categories	No restrictions

4.1.3 Evaluation Methods

The performance of image generative models are notoriously difficult to evaluate objectively [64, 49, 65, 2]. Furthermore, a study has found that there is no clear evidence any particular generation model is better than others when considering several different dataset and evaluation metrics, if there is enough computation done for hyperparameter tuning [66]. It has also been stated that generative models should only be evaluated in the context of the application that the models are used for [64]. Further complicating the problem is the difficulty in objectively evaluating NPR results. Art is a subjective topic and finding an objective method to evaluate the success of NPR results is still an open problem [14].

In light of this, we consider three of the most popular metrics - the Peak Signal-to-Noise Ratio (PSNR), Structural Similarity Index (SSIM), and the Inception Score - for the evaluation of image quality in image generation tasks and evaluate their suitability for the evaluation of *stuff* painting task. To evaluate the suitability of these metrics, we establish two criteria that a good partially painted image should have:

1. Only the desired image regions should be painted (i.e. the *stuff* regions), and the other regions should not be painted
2. The painting style must be reasonably close to the desired painting style. Note that it only has to be *stylistically* similar, not to be exactly the same as that of the ground truth.

Therefore, a suitable evaluation metric should be able to differentiate well between images that satisfies these two criteria or not.

PSNR And SSIM

The PSNR [67] is a pixel-level evaluation metric that is derived from the mean squared error(MSE) and the maximum possible intensity of a pixel in the image. It is meant to measure how closely an image resembles another image. However, as this metric is based on a pixel-to-pixel comparison, it has a limited relationship to human perceived quality of an image [67, 68]. In contrast to this pixel-level comparison of absolute differences, the SSIM [68] is based on the difference in image structural information, such as degradation due to noise or blurring. It has been shown that SSIM correlates well with human-perception based quality assessments [69, 70], and has been used in a wide variety of tasks such as visual cognition [71] and image enhancement [72]. SSIM was designed to be an improvement on pixel-based metrics such as PSNR and MSE. However, there are also previous works [73, 74] that showed that there exists an analytical link between PSNR and SSIM, and that it is possible to predict one from the other. With this in mind, we focus our evaluation on the suitability of SSIM for our task of partial image painting.

We first evaluate the SSIM for the test results of a texture-based style and a non-texture based style, namely the pointillist style and the hayao style. The generated results for the test images were evaluated against their respective ground truth to obtain the SSIM scores. For each style, four images with the highest SSIM score and four images with the lowest SSIM score are shown in Figures 4.2 and 4.3. From the figures, we can see that there is not much correlation between the SSIM scores and quality of the partially painted images. For both styles, in both the cases for lowest and highest scores, there are a mix of images that either satisfies or fails both criteria for a good partially painted image. This means that SSIM is not able

Figure 4.2: Test images with 4 highest and 4 lowest SSIM scores for the Stuff-Point dataset. Regions that should be unpainted are outlined in black.

Figure 4.3: Test images with 4 highest and 4 lowest SSIM scores for the Stuff-Hayao dataset. Regions that should be unpainted are outlined in black.

to differentiate well between a good partially painted image and a bad one.

A look into the SSIM scores for image patches give us some clue as to why this is so. We look at a triplet of image consisting of an original non-painted image, its ground truth partially painted image, and its generated partially painted image, as seen in Figure 4.4. From the figure, we see that the difference in SSIM scores between pairs of image patches show limited correlation with the perceived existence of applied painting in the image. In the first and second rows of the bottom half of Figure 4.4, we can see that there is some correlation when comparing between painted and non-painted images. However, in the third row, when comparing between a pair of image patches where both patches are painted and a pair where only one of the patches were painted, the score of the first pair was lower than that of the second pair, even though by human perception the first pair is more similar than the second pair. This is because the textures applied to the second pair are only similar in style, they are comprised of different dots in different locations, so this pair will get a much lower score when comparing structural distortion. The PSNR [67] is a pixel-level evaluation metric that is derived from the mean squared error (MSE) and the maximum possible intensity of a pixel in the image. It is meant to measure how closely an image resembles another image. However, as this metric is based on a pixel-to-pixel comparison, it has a limited relationship to human perceived quality of an image [67, 68]. In contrast to this pixel-level comparison of absolute differences, the SSIM [68] is based on the difference in image structural information, such as degradation due to noise or blurring. It has been shown that SSIM correlates well with human-perception based quality assessments [69, 70], and has been used in a wide variety of tasks such as visual cognition [71] and image enhancement [72].

From the above analysis, we can conclude that PSNR and SSIM are poor metrics for use in comparing whether two images have similar painting styles. However, in partial painting, certain image regions must be left unchanged.

Original		Ground Truth		Generated Image

Original (O)	Ground Truth (T)	Generated Image (I)	SSIM(T, I)	Comments
0.6757	Mistakenly unpainted regions have higher SSIM scores than correctly painted regions
0.7735	Mistakenly painted region has higher SSIM than correctly painted region
0.4926	Correctly painted regions have lowest SSIM	

Figure 4.4: A look at the SSIM scores of image patches. Top: The images from which the image patches were taken from. Bottom: A comparison of the SSIM scores of each triplet of image patches.

Therefore, this metric can still be used to determine whether the *thing* image regions are left unchanged.

Inception Score

The Inception Score [49] is an evaluation metric proposed for automatically evaluating the quality of images produced by image generation models. One of the most widely used evaluation metric for GANs [75, 65], this metric is

based on an Inception v3 model [76] that is pre-trained on ImageNet.

By applying the image results of the generative model in question to the pre-trained Inception model, this metric jointly evaluates two criteria that are generally considered desirable for generative models:

1. The generated images should contain meaningful, clear, and recognizable objects. This means that the Inception model should be able to clearly recognize the objects in the image, and the resultant conditional object class distribution should have a low entropy.
2. The generated images should have highly varied content. This means that a wide variety of different objects recognizable by the Inception model should be found across the generated images, and the marginal object class distribution should have a high entropy.

Satisfying these two objectives will result in a high Inception Score. There have been derivatives of this score, such as the Fréchet Inception Distance [77]. This score has been found to correlate well with human judgment on the quality of images [49]. However, we argue that the Inception Score is not a good fit as an evaluation metric for the task of *stuff* painting.

Firstly, according to the first criterion for the metric (as outlined above), the Inception model should be able to clearly recognize the objects in the image. However, as noted in [65], even though ImageNet is a very large dataset containing a very large variety of object classes, the marginal class distribution of the object classes in the dataset is likely to differ from that of other datasets, and the same objects might be labeled differently in different datasets (e.g. dogs in one dataset can be separated into different breeds of dogs in another). As such, it is a not good fit for evaluating models trained on a different dataset.

Secondly, in the second criteria of the metric, the generated images should have highly varied object classes. However, this is not a desirable attribute in our task of image *stuff* painting. As the Inception Score was proposed

for GANs in a non-conditional setting and our task is set in a conditional setting, we consider the possible variation of output images for each *unique* input image in our task of *stuff* painting. For each input image, a good output image should contain the exact same *thing* as the input image, while the rest of the image should have been applied with a painting style. A single painting style can have possibly infinite variations, as a single style can be rendered in a multitude of different ways while still retaining the characteristics of the style (e.g. a pointillist painting of a specific tree can be rendered with many combinations of dots in different positions and still look like the same tree). Therefore, in our task, a good output image for each *unique* input image should have an entropy of 0 for the marginal object class distribution, as the object(*thing*) should always be the same across the possible variations of good output images. Therefore, when considering the second criterion of the Inception Score in the conditional setting, a good output image for our task would fail these criteria.

Another point is that these Inception-based scores have shown that they are sensitive to noise. In usual image translation cases, this is a good trait as clear and noise free images are usually desired. However, with style transfer this is not a reliable measure of quality, as images that are stylistically similar can seem noisy from the point of view of the Inception-based scores i.e. Images may satisfy the second criteria for a good partially painted image but still obtain a low score. For example, in pointillist styles, two pointillist style images derived from the same content image can be interpreted as versions of each other but with salt and pepper noise added. This would result in a worse score, even though human judgement would consider those two images to be stylistically similar.

In view of the above points, we conclude that the Inception Score is not a good fit as an evaluation metric for the task of *stuff* painting.

Discussion

We present in Table 4.2 a summary of PSNR, SSIM, and Inception Score.

Table 4.2: Summary of PSNR, SSIM, and Inception Score.

	PSNR	SSIM	Inception
Criteria	Pixel-level differences	Image structural information differences	Generated images contains clear, recognizable, and varied objects
Correlation with partial painted image quality	Low	Low	Low
Measure similarities between images	Yes	Yes	No

Due to the peculiarity of our task, the evaluated metrics are deemed not suitable for the task of partial image painting. However, PSNR and SSIM work by measuring the similarities between two images. Therefore, we can use PSNR and SSIM to determine whether the *thing* object classes in images remain unchanged. To do so, we use the segmentation masks to crop out the *thing* image regions and evaluate them with PSNR and SSIM to determine whether they are entirely left unchanged. We also follow [2] and evaluate our results based on whether they satisfy the aforementioned criteria for a good partially painted image with human perception and judgment.

4.1.4 Training details

For this experiment, the Stuff-Point, Stuff-Brush, Stuff-Shinkai, and Stuff-Hayao datasets were used. A Pix2Pix network was trained for each dataset.

Both networks used similar hyperparameters. Following the results presented by Isola et.al [5], we use a batch size of 1 and a learning rate of 0.0002. Each network was trained for 200 epochs, as results indicated that the networks achieved convergence by then. The learning rate was constant for the first 100 epochs, and linearly decayed to 0 over the next 100 epochs. The models were also tested with 300 test images that is separate from the training images.

4.2 Results And Discussion

For the evaluation of image quality, we recall the criteria that we previously defined for a good partially painted image. A good partially painted image should 1) only be painted only at image regions where painting is desired, and 2) the painting style should be similar to the desired painting style.

4.2.1 Comparison Baselines

For comparison, we use images generated via a two-step process as baselines. This process generates partially painted images by using a generated segmentation map to combine an original image and its painted version, as illustrated in Figure 1.2. We use two different models¹ to generate two sets of segmentation maps, which then generates two set of images as baselines. The two architectures used for the segmentation models are the U-Net [12] and the Pyramid Scene Parsing Network [78]. Both architectures use pre-trained VGG-16 [13] weights for the encoder portions of the networks. Therefore, we term images generated using segmentation maps generated by the U-Net as the **VGG-UNet** dataset and the images generated using segmentation maps produced by the PSPNet as the **VGG-PSPNet** dataset.

¹Code for models can be found at <https://github.com/divamgupta/image-segmentation-keras>

4.2.2 Success Cases

We find that by the end of training, the networks are able to perform background painting with the desired styles on most of the training images. We find that the network is able to apply a painted style over the background, while leaving the foreground not painted. The non-painted portions of the image also retain the sharpness and detail of the source image. As we are interested in *stuff* painting, a good model should be able to paint the *stuff* classes regardless of the type of *things* present in the image. To test whether the models were invariant to the type of *things* present, we test with images containing a variety of *thing* classes. Figures 4.5, 4.6, 4.7, and 4.8 show examples of the results of the test images that were passed through the trained models. In the examples, each image pairs is labelled by the *thing* class in the images, which should remain *unpainted*. From the results of the test, we can see that the models demonstrate this invariance.

One of the key motivations behind using a GAN for perform partial image painting in one step is that the more conventional two-step way of doing partial painting would result in abrupt and unnatural boundaries between the painted and non-painted regions. By training to perform these two steps end-to-end in one step, the boundaries between the painted and non-painted image regions in the partially painted images should be become more natural looking.

By comparing the test results of this experiment with the baseline VGG-UNet and VGG-PSPNet datasets, we can conclude that this is indeed the case. As shown in Figures 4.9 and 4.10, we can see that there are improvements in the images generated by the GAN in terms of natural looking boundaries. For both images, the enlarged image regions (marked by black boxes) show where training with a GAN resulted in a more natural boundary between painted and non-painted regions. In the VGG-UNet and VGG-PSPNet datasets, we can see that there are instances where there are abrupt patches of unpainted areas that should have been painted. In Figure 4.10, there is

Figure 4.5: Successful *stuff* painting for the Stuff-Point dataset.

also an instance where image regions that should have been painted in the colours of the trees in the background, but instead were painted in the colours of the hands of the baseball batter. This was due to the cutting and pasting

Figure 4.6: Successful *stuff* painting for the Stuff-Brush dataset.

of that image region from the original fully painted image. This artifact was not present in the partially painted image generated by the GAN.

Figure 4.7: Successful *stuff* painting for the Stuff-Hayao dataset.

Figure 4.8: Successful *stuff* painting for the Stuff-Shinkai dataset.

4.2.3 Evaluation Metrics

For partial *stuff* painting, it is important that only the *stuff* image region is painted and the *thing* image region is left unchanged. Therefore, we crop

Figure 4.9: Results comparison between the GAN approach, the VGG-UNet and VGG-PSPNet two-step processes for the Pointillist and Hayao styles. (a) Pointillist style results. (b) Hayao style results.

out the *thing* regions in the generated and ground truth images and calculate their PSNR and SSIM scores. We calculate the metrics for Pix2Pix-Stuff, VGG-UNet, and VGG-PSPNet for comparison. These metrics are calculated

Figure 4.10: Results comparison between the GAN approach, the VGG-UNet and VGG-PSPNet two-step processes for the Brushstroke and Shinkai styles. (a) Brushstroke style results. (b) Shinkai style results.

for the Stuff-Point and Stuff-Hayao dataset. The histogram plots of the metrics are shown in Figures 4.11 and 4.12. We can see that the PSNR graph for Pix2Pix-Stuff are shifted to the right as compared to the other two segmentation models for both styles. This indicates that Pix2Pix-Stuff perform better in ensuring that the *thing* image regions remain unchanged.

The SSIM graphs are similar for both styles across the three models. This is probably because SSIM looks at structural similarity, and the images look similar on a global structural scale.

4.2.4 Texture Learning

We can think of the application of a texture as a transformation filter applied over an image patch. This transformation filter samples a certain pixel in the image patch for its pixel value and uses that pixel value to apply a transformation over the whole image patch. A simple example of this is illustrated in Figure 4.13. In this example, given a source image patch, the filter samples a pixel P in the image for its value, then uses that value to apply a certain style on the image. In this example, the filter replaces parts of the image patch with the value of P . The resultant texture is then dependent on the filter parameters.

Therefore, we can say that when a model learns a certain image re-rendering style, it is learning a transformation filter that fits the desired style. That is, we want to learn the function t where

$$o(x, y) = r(i(x, y)) = \sum_{s=-a}^a \sum_{t=-b}^b r(s, t, p, i(x-s, y-t))$$

i denotes the input image patch, r denotes the transformation filter function, o denotes the output image patch, and p denotes the value of the sampled pixel from the input image. s and t denotes the coordinates of each element in the convolution kernel and $-a \leq s \leq a$, $-b \leq t \leq b$.

We can see that this is a very similar operation to a typical convolution operation, defined as w in

$$g(x, y) = w * f(x, y) = \sum_{s=-a}^a \sum_{t=-b}^b w(s, t) f(x-s, y-t)$$

Figure 4.11: Comparing the PSNR and SSIM histograms for Pix2Pix-Stuff (GAN) with VGG-UNet (UNet) and VGG-PSPNet (PSPNet), for the Stuff-Point dataset.

Figure 4.12: Comparing the PSNR and SSIM histograms for Pix2Pix-Stuff (GAN) with VGG-UNet (UNet) and VGG-PSPNet (PSPNet), for the Stuff-Hayao dataset.

Figure 4.13: Simple example of a style transformation filter.

The main difference is that the filter kernel values in t is not a fixed value as in a typical convolution kernel, but is a function that outputs a value based on the current image pixel value and the sampled pixel value p . This allows the resultant style pattern to vary based on the value of the pixel. This means that each element of the filter kernel is now a function with input p ,

$$r(s, t, p, i(x - s, y - t)) = e_{s,t}(p, i(x - s, y - t))$$

where $e_{s,t}$ is an element of kernel r at position (s, t) . For the example in Figure 4.13, the functions at the elements marked 1 are functions that simply ignore the values of the current pixel value and returns p ,

$$e_{s,t}(p, i(x - s, y - t)) = p,$$

and the other elements in that transformation filter ignore p instead and returns the current pixel value,

$$e_{s,t}(p, i(x - s, y - t)) = i(x - s, y - t)$$

Given the similarity between a style transformation filter and a convolution operation, it is an intuitive way for convolution layers to learn to approximate a certain style transformation. The complexity of having functions instead of fixed weights at each kernel element can be modeled by a

hierarchy of convolution layers.

Based on our test results, as can be seen in Figure 4.5, we can conclude that this is indeed what our model is learning. This is evident from the fact that the colour of each pointillist dot in the painted output image is similar to the original colour of at least some pixels in the region covered by the dot in the input image. We can also conclude from this that the model is not simply learning to paste the texture from the ground truth into the output image as the test images are not seen by the model during training.

Using GANs for this task also has the advantage of using a discriminator network as a form of loss function, as it aids in the learning of this type of transformation filters. The patterns in textures contain structures on the image patch level, which cannot be adequately modeled by pixel level losses such as L1 and L2 norms. A typical discriminator sees the entire image, which gives it a view of the structure of the texture patterns. Previous work [5] has noted that a PatchGAN discriminator can be seen as a form of texture loss. Therefore, the use of PatchGAN as the discriminator in this task is fitting as it helps the discriminator focus on discerning the correct texture pattern.

4.2.5 Advantage Of U-Nets

We believe that the skip connections between layers that are characteristic of the U-Net architecture in the generator play a big part in the sharpness and detail of the non-painted image portions. With these skip connections, if a portion of the image is determined to not require re-rendering, the image details from the source image can be directly used in the output image through the skip connections. This means that low-level local detail can be preserved in the final output image if needed.

Figure 4.14: Training results for generators with varying number of skip connections for the Stuff-Point dataset.

Testing Skip Connections

We test the effect of skip connections on the ability of the generator to learn by training models with varying number of skip connections in the generator. To test the effects for both texture based and non-texture based styles, we train models for the pointillist style and hayao style. The results are shown in

Figure 4.15: Training results for generators with varying number of skip connections for the Stuff-Hayao dataset.

Figures 4.14 and 4.15. We find that skip connections play a large role in helping the generator learn texture-based styles, with the training results showing marked improvement with the increase in skip connections. Apparently, skip connections play a less important role for non-texture based styles. As seen in Figures 4.14 and 4.15, the generated images are still relatively sharp and

detailed even with only one skip connection in the generator, which is unlike the results for pointillist style, where having just one skip connection resulted in a very blurry generated image. However, we see that having skip connections still helps the generator learn to preserve the original image details of non-painted regions.

Role Of Skip Connections

As the image is passed through the convolution layers in the encoder part of the generator, it is distilled into a high level representation that includes information about the objects in the image and their relative position. This information can then be used in the decoder part of the generator to determine which parts of the image to re-render and which parts of the image to take preserve unchanged. With this, as the image is progressively upsampled and generated, the inputs from the source image can be accessed through the skip connections to be directly added to the output image. An example of this process is shown in Figure 4.16. In the example, each decoding layer decides which part of the image to paint and which part to preserve, as indicated by the example activation map.

On the other hand, the skip connections are also beneficial in helping the model learn the style transformation filter function. As each element function in the filter requires an input p , which is the value of a sampled pixel from the original image. This value can be obtained through the skip connections. Without the skip connections, the only way that a p can be obtained would be through the reconstructed image representations, which might not be an accurate value. Transforming using an inaccurate p value can result in an output that has a bigger difference from the ground truth than if an accurate p value was used. This can affect learning negatively.

Figure 4.16: Example of how skip connections help to preserve source image information.

4.2.6 Analysis Of Loss Functions

As an improvement to the traditional sigmoid cross entropy discriminator loss function [4], Least Squared GAN (LSGAN) [79] was proposed to solve the shortcoming where data samples that were correctly classified but were far from real data caused vanishing gradients in the generator. The LSGAN introduces a discriminator that uses the least-squared loss function. On the other hand, the Wasserstein GAN (WGAN) [80] was proposed to overcome the shortcomings brought when the discriminator becomes too well trained and saturates, which also leads to the vanishing gradient problem. The WGAN introduces a loss function based on the Earth Mover distance. However, WGAN uses weight clipping, which can lead to undesirable behaviour. An improvement that uses gradient penalty instead of weight clipping was introduced, dubbed WGAN-GP [81]. Isola et al. [5] also recommended that an L1 absolute distance loss be included to model low-frequency image information as the PatchGAN is optimized for modeling high-frequency image

structure. Therefore, we also look at the impact of removing L1 loss for our task. We look at the effects of each loss function for our task.

For this test, we train models for a texture-based style (Stuff-Point dataset) and a non-texture-based style (Stuff-Hayao dataset). For each style, four models were trained. Three of the models were trained with different hybrid loss functions (BCE+L1, LSGAN+L1, WGANGP+L1), whereas the fourth model for each style was trained with only BCE loss without L1 loss, to demonstrate the effect of removing L1 loss from training. The results are shown in Figures 4.17 and 4.18.

Figure 4.17: Results comparison with the Stuff-Point dataset for various loss functions in training. Each column from left to right: Ground truth for *stuff* painting, model trained with BCE loss, model trained with least squared loss, model trained with Wasserstein loss, model trained with BCE loss but without L1 loss.

From the results, we can see that L1 loss plays an important part in helping the generator learn to perform partial painting. In Figures 4.17 and

Figure 4.18: Results comparison with the Stuff-Hayao dataset for various loss functions in training. Each column from left to right: Ground truth for *stuff* painting, model trained with BCE loss, model trained with least squared loss, model trained with Wasserstein loss, model trained with BCE loss but without L1 loss.

4.18, we can see that when the model is trained without L1 loss, the generator does not learn to produce partially painted images at all. Instead, the entire image is transformed into a painted style. Also, the painted regions are also not as sharp as when L1 loss is used in training.

From the results, we can also see that out of the three tested loss functions, BCE loss produced the best results. Training with least-squared loss (LSGAN) resulted in very little texture transfer. This is especially the case for texture-based style, where in Figure 4.17, we see that the resultant images exhibit no pointillist style at all. As for non-texture-based styles, in Figure 4.18, we see that the resultant images show some characteristics of the intended style, but it was mainly expressed as a change in the colour

scheme, whereas more abstract characteristics such as clouds being drawn in a flatter style was absent.

Training with WGANP loss has the worst performance out of the three tested loss functions. Training with WGANP loss resulted in blurry images and checkerboard artifacts in the results. Also, the generator does not seem to be able to only selectively paint over the *stuff* regions, with most of the resultant images having the entire image being painted over instead. In the case of texture-based styles, we can see that the learnt texture is also not satisfactory, as the learnt style is far from that of the desired style.

Based on these results, we find that a hybrid loss of BCE+L1 is most suitable for our task. However, it might also be the case that the tested architectures and training methods are not well suited for training with LS-GAN or WGANP losses, but due to time constraints, we could not explore other forms of training methods and architectures.

4.2.7 Limitations

There are some cases where partial painting is not ideally done. Some examples are shown in Figures 4.19 and 4.20. The failure modes here generally stem from failure to correctly identify the regions to re-render. The re-rendering style is correct in all cases. The failure cases are generally due to low contrast between the *thing* regions and the *stuff* regions in the images. This can be from the *things* in question having a similar texture or colour with the *stuff* around it.

In some cases, it is also ambiguous which part of the pictures belong to *thing* or *stuff* classes, some examples of which can be seen in Figure 4.21. For example, if the background is packed with people, it can resemble a patch of texture instead of a collection of people, and hence it can be ambiguous as to whether or not those people should be considered as a part of the background. Another commonly seen ambiguous element are buildings. As *stuff* generally have no definite parts [8] and is defined by their texture or

Figure 4.19: Examples of failure results for the Stuff-Point dataset generated from test data. For each image pair, the original image is on the left and the generated image is on the right.

Figure 4.20: Examples of failure results for the Stuff-Brush dataset generated from test data. For each image pair, the original image is on the left and the generated image is on the right.

patterns [9], for a model that is trained to detect *stuff* that generally fits these criteria, buildings may or may not be classified as *stuff* depending on factors such as the angle that the building is depicted from, which parts of the building is visible, the type of building etc. For example, a house photographed in its entirety from would likely including specific elements such as doors and windows, and seem like an *thing*. However, if only a part of the house is photographed from the side such that only a plain wall with a portion of the roof is visible, it will seem more like a *stuff*

Figure 4.21: Examples of test images that have ambiguous backgrounds.

Another limitation is that there were cases where blurry distortions were present in the cartoonized regions of the image for the Stuff-Hayao and Stuff-Shinkai dataset, some examples of which are shown in Figure 4.22. This might be due to the network architecture not being able to sufficiently model the unique characteristics of cartoon styles, as noted by [61]. This might possibly be mitigated by adapting some features of CartoonGAN into this model, such as the losses that was proposed in [61] to improve image cartoonization.

Figure 4.22: Blurry distortions were introduced in some results during partial cartoonization. For each pair of images, the original image is on the left, and the output image is on the right.

4.3 Summary

This chapter presented a proof-of-concept for partial non-photorealistic rendering of *stuff* in images using a GAN. We first evaluated some evaluation metrics toward our task of image *stuff* painting, and concluded that human subjective evaluation is the current most appropriate metric for our task as a whole, though we still use PSNR and SSIM to aid in determining whether the *thing* regions are left unchanged. We presented the results of training a GAN to perform the partial re-rendering of images into painted styles. Four datasets representing four painting styles were tested, two of which were texture-based styles (Stuff-Point and Stuff-Brush) and the other two were non-texture-based styles (Stuff-Hayao and Stuff-Shinkai). We also showed that using a GAN resulted in a higher PSNR score, meaning that it was better at leaving the *thing* regions unchanged. We then discussed how texture learning can be performed with stacked convolution layers, and also

how the U-Net architecture is suitable for this task of partial re-rendering of images. We also evaluated how different loss functions affect the training of the model for our task. Finally, we present a discussion of the limitations of performing this task, which mainly comes from training data where it is difficult to distinguish the *stuff* classes from *thing* classes.

Chapter 5

Stuff Painting GAN(SPGAN): Reducing *Stuff* Painting Error

As stated in the previous chapter, the task of partially painting an image can be seen as a combination of a semantic segmentation subtask and a style transfer subtask. We have already seen in the previous chapter that the GAN is able to learn to perform partial painting of an image, and the GAN is able to perform the style transfer part of the task well. However, there is still room for improvement in the semantic segmentation part of the task. In this chapter, we incorporate image segmentation masks into the training process to improve the ability of the GAN in correctly identifying regions for painting. The segmentation mask is incorporated via an additional discriminator. The new discriminator is incorporated into a standard GAN model and the resultant architecture is called Stuff Painting GAN (SPGAN)¹. We also introduce a Gaussian error correction to improve the learning of the segmentations in the GAN.

¹The work in this chapter has been submitted to AAAI 2020

5.1 Semantic Segmentation Discriminator

The adversarial loss function of the current GAN architecture encourages the generator to generate images that can deceive the discriminator, while the discriminator learns to discriminate between generated images and real images. The generator is guided by the discriminator to produce images with our desired characteristics in this way. For the task of partial *image* painting, the discriminator has to learn two things: 1) The desired image will comprise a mix of painted and photorealistic images, each of which have very different characteristics, and 2) which image regions should be painted and which regions should be photorealistic. We have already seen that the current discriminator learns task 1 well but not as well for task 2. We present an improvement to the GAN architecture for improving the ability of the GAN to differentiate between *stuff* and *things*, and hence improve the ability of the generator to only paint *stuff* regions.

To improve the segmentation ability of the generator, we propose an improvement that makes use of segmentation masks to determine whether only the correct image regions have been painted. Previously, pairs of images, (x, y) , were used to train the GAN, where x was the original image, and y was the ground truth partially painted image. In our proposed method, the GAN now uses *triplets* of images, (x, y, m) , for training, where m is a segmentation mask for x . m is a binary mask, and only indicates whether each pixel in the image belongs to a *stuff* class or a *thing* class. Therefore, in addition to the current discriminator, D , which discriminates between real and fake images, we introduce a new discriminator D_s that determines whether the correct image regions were painted. D_s takes in an image pair, (m, i) , as input and outputs a scalar value indicating how well the painted regions match the segmentation mask. The intuition is that D_s will be trained to recognize from the segmentation mask whether a particular image region should appear photorealistic or painted.

We define the loss $\mathcal{L}_{seg}$ for the semantic segmentation discriminator D_s

as follows,

$$\begin{aligned}\mathcal{L}_{seg}(G, D_s) = & \mathbb{E}_{m,y} [\log D(m, y)] \\ & + \mathbb{E}_{m,z} [\log(1 - D(m, G(x, z)))]\end{aligned}$$

The discriminator D_s is trained to output a high value when the presence/absence of painted style application to the image region matches the *stuff/thing* labels in the segmentation map.

In summary, we propose a conditional GAN architecture that uses both a real/fake discriminator and a semantic segmentation discriminator for improving the accuracy of partial image painting. The final hybrid objective is then defined as

$$G^* = \arg \min_G \max_D \mathcal{L}_{cGAN}(G, D) + \mathcal{L}_{seg}(G, D_s) + \lambda \mathcal{L}_{L1}(G)$$

Figure 5.1: State diagram of SPGAN1.

The final network architecture thus comprises three networks, a generator and two discriminators. Figure 5.1 shows our proposed network architecture,

which we refer to as **SPGAN1**. Now there are three loss functions $\mathcal{L}_d$ (style discriminator), $\mathcal{L}_{seg}$ (segmentation discriminator) and $\mathcal{L}_g$ (generator) that are used for training.

5.2 Gaussian Failure Correction

Presence of complex objects and colors may result in failure cases where the foreground is painted instead of the background towards the end of training. For example, when the colors of foreground are similar to the background or the shape of foreground object resembles the background. In order to overcome this we use a Gaussian function to represent the evolution of failures in the generated images. Figure 5.2 illustrates the error correction $\mathcal{L}_e$ to the generator in each iteration. We denote the failure function for the total error between the target pixel value x_i and the generated pixel value $\hat{x}_i$. The image data is normalized to have values between $[-1, 1]$ for three color intensities, namely Red, Green and Blue. Since, we want to preserve the image of the foreground objects and only paint the background stuff, we make use of the mask information to compute the total error β_i for each pixel in the mask region as follows:

$$\beta_i = \begin{cases} 1 - \exp^{\alpha \times \text{abs}(x_i - \hat{x}_i)}, & \text{if } \text{abs}(x_i - \hat{x}_i) \geq 0 \\ 0, & x_i - \hat{x}_i = 0 \end{cases}$$

where α is a constant that represents the rate at which the failure evolves in the GAN. For slowly developing faults the value of α is small, while for abrupt faults α becomes large. In the limit that $\alpha \rightarrow \infty$ the evolution becomes a step function where $\beta_i = 0$ for $x_i - \hat{x}_i = 0$ and $\beta_i = 1$ for $x_i - \hat{x}_i > 0$. Here, we set α to the learning rate of the GAN. We sum over β values for all the pixels in the mask region and add it to the loss function so that it can correct the failure in the next few iterations. Figure 5.3 illustrates the

failure correction using Gaussian distributions. Since, the pixel values are in $\in [-1, 1]$ for three channels hence the maximum possible error is 6. With increasing error between predicted and target pixels the Gaussian value is lower and hence the correction is higher. The value of α controls the rate of evolution of pixel errors. For small values of α the pixel error variance is larger and for smaller values it's lower.

Figure 5.2: Error correction in each iteration of Gaussian Failure Correction.

Figure 5.3: Failure correction using Gaussian distribution.

5.3 Experiments

We evaluate two versions of the proposed SPGAN model. First, we evaluate the **SPGAN1** model, which includes the semantic segmentation discriminator but without Gaussian failure correction, for *stuff* painting performance on the Stuff-Point and Stuff-Hayao datasets. We compare the SPGAN1 model with a Pix2Pix-Stuff model. Next, we evaluate the **SPGAN2** model, which includes both the semantic segmentation discriminator and Gaussian failure correction, and compare it with the SPGAN1 and Pix2Pix-Stuff model.

We also evaluate the SPGAN2 model on a real world dataset for semantic segmentation. In this case, we treat the task of semantic segmentation as a

Table 5.1: Summary of deep drive dataset.

Size	2000
Road types	City, country, unpaved, unmarked, etc.
Weather types	Clear, rainy, cloudy, etc.

partial painting task where the painting style to apply is to turn the painted regions white. Therefore, the *thing* regions will be left unchanged, while the rest of the images will be white, which is essentially semantic segmentation.

5.3.1 Training Data

COCO-Stuff Datasets

We use the Stuff-Point and Stuff-Hayao datasets for evaluating *stuff* painting performance. These two datasets were chosen for evaluating both texture-based and non-texture-based styles. We used the same datasets as the previous experiment for a fair comparison of results between the Pix2Pix-Stuff model and SPGAN. A summary of these datasets can be found in Table 4.1 in Chapter 4.

Deep Drive Dataset

We also evaluate the models on the Deep Drive dataset. The Deep Drive dataset is a semantic segmentation dataset for driving. This dataset consists of driving videos from the perspective of a human driver, taken with an in-car camera. The data includes data on driving in various situations, such as driving on city roads, residential streets, unpaved roads and roads without lane markings. It also includes various kinds of weather such as clear, rainy, and cloudy weather [82]. 2000 road images with a mask for the cars were used for training. As summary of this dataset is listed in Table 5.1.

5.4 Results And Discussion

For evaluation of image quality, once again we recall the criteria that we previously defined for a good partially painted image. A good partially painted image should 1) only be painted only at image regions where painting is desired, and 2) the painting style should be similar to the desired painting style.

5.4.1 Improved *Stuff* Painting

Figures 5.4 and 5.5 show some sample test results from the models trained on the Stuff-Point and Stuff-Hayao datasets, respectively. In the figures, we can see that training with SPGAN produces partially painted images that are better or at least similar to that of the baseline.

From the results, we can see that both the baseline and SPGAN can satisfy criteria 2 well for both datasets. The Stuff-Point models produced pointillist textures that matched the original texture well and the Stuff-Hayao models rendered the *stuff* regions in a style similar to that seen in the ground truth. It is regarding criteria 1 where we see improvement. In the figures, we can see that certain *thing* image regions where the baseline GAN incorrectly painted over have been left untouched by SPGAN (e.g. the head and jacket of the man, the head of the dog). However, there is still room for improvement, as there are still image regions that have not been correctly identified as hence, were mistakenly ignored or painted over.

A point of note is that the SPGAN achieves the most improvement in images where the *thing* and *stuff* classes have similar characteristics, such as colour and pattern. In these images, the baseline model face difficulties when differentiating between the *stuff* and *thing* regions. However, as the SPGAN includes a discriminator that uses the segmentation masks for discrimination when training, the generator is able to explicitly learn which regions are *stuff* and hence, correct for errors better.

Figure 5.4: Test results for Stuff-Point dataset. Notable improvements in segmentation are in black boxes and enlarged for better view.

Figure 5.5: Test results for Stuff-Hayao dataset. Notable improvements in segmentation are in black boxes and enlarged for better view.

5.4.2 Effect Of Second Discriminator

Yoo et al. [58] introduced a second discriminator in their GAN as a way of introducing a second training objective to the GAN. We take inspiration from this architecture when incorporating segmentation masks into the training process. However, adding an additional discriminator is costly, as it increases training time and space requirements. What if we introduce segmentation masks directly into the existing real/fake discriminator by introducing it as an additional input on top of the source image in the normal conditional GAN setting? That is to say, the single discriminator takes in a *triplet* of images consisting of source, target, and mask images, as illustrated in Figure 5.6. To test the benefits of adding a second discriminator as compared to directly adding segmentation masks to the existing single discriminator, we train the model as shown in Figure 5.6 and compare the results against the results in the previous section(Figures 5.4 and 5.5).

Figure 5.6: Directly adding segmentation mask to a GAN with a single discriminator.

Some sample test results for the comparison are shown in Figures 5.7 and 5.8. We find from these results that the naive approach heavily degrades the performance of the stuff painting. In the case of the Stuff-Point dataset, large portions of the images that were supposed to be painted were left unpainted, and black and white artifacts are commonly seen in the painting styles. As

for the case of the Stuff-Hayao dataset, the test results heavily deviated in content from the source image, and the results display no apparent link to their source images. These results are noticeably worse than even the baseline results of training without segmentation masks at all. In comparison, the results of training with a second discriminator (Figures 5.4 and 5.5) showed improvement over the baseline results. Therefore, we conclude that the benefits from the addition of a second discriminator to incorporate segmentation masks outweighs the addition cost in training time and space requirements.

5.4.3 Impact Of Receptive Field Sizes

Luc et al. [26] have noted that the use of a discriminator provided a field of view over the entire image, which allows the model to enforce a higher-order consistency, thereby increasing segmentation accuracy. Meanwhile, Isola et al. [5] have shown that discriminators with limited receptive field size (PatchGAN) work just as well as discriminators with a full receptive field size. However, they have also stated that PatchGANs can be seen as a form of style/texture loss due to the nature of each of its outputs having a limited receptive field. We are interested in a discriminator that is able to look at the structure of the entire image and differentiate whether the image it is looking at is correctly segmented or not, according to the segmentation mask it is given. Therefore, we test the ability of the semantic segmentation discriminator in enforcing segmentation accuracy when the outputs have varying receptive field sizes.

Figures 5.9 and 5.10 show some sample test results of models with varying receptive field sizes in the semantic segmentation discriminator. We find that for both datasets, varying the receptive field sizes do not seem to have significant effect on the ability of the model in determining the correct image regions to paint. This is probably due to the *stuff* and *things* having very different characteristics which is already identifiable on the smaller patch levels. For example, grass and sand have a rather uniform textured appearance,

Figure 5.7: Test results for directly adding segmentation mask to single discriminator as compared to SPGAN for Stuff-Point dataset.

Figure 5.8: Test results for directly adding segmentation mask to single discriminator as compared to SPGAN for Stuff-Hayao dataset.

Figure 5.9: Test results for the Stuff-Point dataset with varying receptive field sizes in the semantic segmentation discriminator. From left to right: 16×16 , 70×70 , 286×286 .

Figure 5.10: Test results for the Stuff-Hayao dataset with varying receptive field sizes in the semantic segmentation discriminator. From left to right: 16×16 , 70×70 , 286×286 .

whereas *things* like planes tend to have well defined lines. A patch-level field of view is sufficient to model these information.

5.4.4 Gaussian Failure Correction

Gaussian Failure Correction is an addition to the loss function that aims to correct segmentation errors in partial stuff painting, which helps the model learn a better segmentation. We refer to SPGAN with Gaussian Failure Correction as SPGAN2. Figure 5.11 shows a comparison between the various models for *stuff* painting. From the results, we can see that even though SPGAN improves upon Pix2Pix-Stuff, there are still limitations in *Stuff* segmentation where part of the *thing* image region was still painted. Introducing Gaussian Failure Correction to the model (SPGAN2) improves on this limitation and helps the model learn to better segment the *stuff* regions of the images. The test results show more natural boundaries between the *thing* and *stuff* image regions.

We also test the segmentation performance of SPGAN2 on the Deep Drive dataset. Figure 5.12 compares the segmentation by SPGAN2 with a baseline deep learning model VGGNet [83], which is commonly used for semantic segmentation. We find that the SPGAN2 model is able to produce much better segmentations with natural looking boundaries between the background and foreground. The SPGAN2 model is also able to recognize small cars in the images, which is not present in the VGGNet segmentation.

5.4.5 Evaluation Metrics

As in Chapter 4, we use PSNR and SSIM to evaluate the performance of the models in ensuring that the *thing* image regions are left unchanged. In Figure 5.13 we compare the histogram of PSNR and SSIM for images generated with baseline GAN, SPGAN1 and SPGAN2 for the Stuff-Point dataset. We can see that the proposed SPGAN2 model is significantly shifted

CHAPTER 5. STUFF PAINTING GAN(SPGAN): REDUCING *STUFF* PAINTING ERROR

Figure 5.11: Comparison of background painting for Stuff-Point and Stuff-Hayao dataset.

Figure 5.12: Car segmentation results by VGGNet and SPGAN2 on the Deep Drive dataset.

Figure 5.13: Comparing the PSNR and SSIM histograms for baseline GAN (Pix2Pix-Stuff) with SPGAN1 and SPGAN2.

to the right for PSNR showing higher quality images. The SSIM index is only marginally better in the new model. This is probably because it looks at global structural differences, therefore it might not capture differences in partial stuff painting performance well.

5.5 Summary

This chapter presented SPGAN, a GAN for performing partial painting of *stuff* in images. To increase the ability of the model in determining the correct image regions for painting, we added a novel semantic segmentation discriminator that uses the segmentation mask of the images to guide the generator in learning to better differentiate between *stuff* and *things* in images, which helps in the accuracy of partial painting. We also introduced an additional loss called Gaussian Failure Correction to the model, which penalizes the model when painting is done on the *thing* region. We then demonstrated that SPGAN performed at least on par with our baseline comparison model, and that it performed better for images where the *thing* and *stuff* regions have similar characteristics. We demonstrated the benefit of having a second discriminator that takes segmentation masks as input, as compared to directly adding the segmentation mask as an additional input to a single discriminator. We then perform a study on the effect of varying receptive field sizes in the discriminator, as it has been noted in literature regarding semantic segmentation in GANs that a receptive field that covers the entire image is preferable. We find that this is indeed the case for our semantic segmentation discriminator, and that a receptive field size that covers the entire image works best for this discriminator. We also demonstrated that incorporating Gaussian Failure Correction improved segmentation performance of the SPGAN, and also demonstrated the segmentation performance of the final SPGAN by segmenting cars in road images from the Deep Drive dataset.

Chapter 6

Conclusion And Future Work

6.1 Conclusion

In this thesis, we presented our process of developing an approach to the automated Non-Photorealistic Rendering of the *stuff* classes of images.

We first conducted a brief review into the key ideas behind GANs and presented some notable GAN frameworks and architectures that have shown impressive results in the area of image-to-image translation.

Given the lack of data for training such a model, we then presented a data generation process that generates images that are partially re-rendered in a non-photorealistic fashion from an image segmentation dataset. This 2-stage process is flexible and extensible to achieve any variety of effects in the resultant dataset, which can then be used to train models for new applications.

We then presented our experimentation results on training a Pix2Pix network for automatic partial *stuff* painting. We experimented with two NPR styles, namely a texture-based style (pointillist style) and a non-texture-based style (Miyazaki hayao cartoon style). We investigate with both texture-based and non-texture-based styles as they have different fundamental characteristics. We showed that the network could perform well in this task gen-

erally, although it is not perfectly accurate. We then performed an analysis on how the task of partially re-rendering images can actually be interpreted as a combination of a segmentation subtask and a re-rendering subtask, and showed how the network performs in learning these 2 subtasks. We then showed how a hierarchy of convolution layers is an ideal fit for the task of learning to apply a certain style or texture to an image, and also provided an argument for how having a discriminator network aids in learning to apply styles. Next, we provided an analysis of how using a U-Net architecture aids in the task of partial image re-rendering. Lastly, we look at the limitations of this model for the task.

Finally, we introduced SPGAN, a GAN architecture which improved on *stuff* painting accuracy by incorporating a semantic segmentation discriminator, which takes as input the segmentation masks of the images and the real/fake images, into the training process. We demonstrate that SPGAN achieves results that are on par or better than our baseline Pix2Pix model. We also introduced a new loss called Gaussian Failure Correction into the model, which we refer to as SPGAN2. The main improvements come in images where the *stuff* and *thing* regions have similar regions. The baseline Pix2Pix-Stuff model faced difficulties for this type of images, but SPGAN could differentiate better between *stuff* and *thing* regions in these images due to the addition of segmentation masks in the training process. We then investigated the effects of receptive field sizes in the semantic segmentation discriminator and found that a receptive field size that covered the entire image resulted in the best partially painted images. We also show that SPGAN2 has better segmentation performance than both Pix2Pix-Stuff and SPGAN1. We also demonstrated that SPGAN2 performs better at car segmentation than a VGG semantic segmentation model.

This research has been a fruitful learning journey, and much insight was gained into the fantastic framework known as GANs. This research sought to build upon previous work done in the use of GANs for image-to-image

translation, and present a novel application. We hope that our work can inspire more novel applications and improvements in this area.

6.2 Future Work

This project was meant to be a proof-of-concept; it was meant to explore whether this goal can be achieved, and how to achieve it. We have shown a way of achieving it, and also a data generation pipeline through which others can also try it out. To test the limits of this task, future work can be on trying to model more complex styles, multiple styles or styles with structure on a larger scale.

Currently, we have only performed partial image painting with a predefined style. It would be an interesting and useful improvement if one could specify the style one preferred for the partial painting as part of the input. On the other hand, it would also be useful if one could dynamically specify what part of the image to paint/re-render, instead of the current way of specifying through the training data what part of the image to paint (Model learns to paint background because it only observes images where the background is painted). It would then be ideal for a model where we can both specify where in the image we want to re-render, and what style do we want it to be re-rendered in.

Bibliography

- [1] Paul Rosin and John Collomosse, editors. *Image and Video-Based Artistic Stylisation*. Springer London, 2013.
- [2] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. A neural algorithm of artistic style. *CoRR*, 2015.
- [3] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *ECCV (2)*, volume 9906, pages 694–711. Springer, 2016.
- [4] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.
- [5] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks. *CoRR*, 2016.
- [6] Chen-Hsuan Lin, Ersin Yumer, Oliver Wang, Eli Shechtman, and Simon Lucey. ST-GAN: Spatial transformer generative adversarial networks for image compositing. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. IEEE, jun 2018.

BIBLIOGRAPHY

- [7] Neal Wadhwa, Rahul Garg, David E. Jacobs, Bryan E. Feldman, Nori Kanazawa, Robert Carroll, Yair Movshovitz-Attias, Jonathan T. Barron, Yael Pritch, and Marc Levoy. Synthetic depth-of-field with a single-camera mobile phone. 2018.
- [8] Holger Caesar, Jasper Uijlings, and Vittorio Ferrari. Coco-stuff: Thing and stuff classes in context. In *CVPR*, pages 1209–1218. IEEE Computer Society, 2018.
- [9] David A. Forsyth, Jitendra Malik, Margaret M. Fleck, Hayit Greenspan, Thomas Leung, Serge Belongie, Chad Carson, and Chris Bregler. Finding pictures of objects in large collections of images. In *Object Representation in Computer Vision II*, pages 335–360. Springer Berlin Heidelberg, 1996.
- [10] Jeremy Heitz and Daphne Koller. Learning spatial context: Using stuff to find things. In *Lecture Notes in Computer Science*, pages 30–43. Springer Berlin Heidelberg, 2008.
- [11] Joseph Tighe and Svetlana Lazebnik. Finding things: Image parsing with regions and per-exemplar detectors. In *2013 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, jun 2013.
- [12] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation.
- [13] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition.
- [14] Yongcheng Jing, Yezhou Yang, Zunlei Feng, Jingwen Ye, Yizhou Yu, and Mingli Song. Neural style transfer: A review. *CoRR*, 2017.
- [15] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, jun 2016.

BIBLIOGRAPHY

- [16] Alex J. Champandard. Semantic style transfer and turning two-bit doodles into fine artworks. *CoRR*, 2016.
- [17] Ming Lu, Hao Zhao, Anbang Yao, Feng Xu, Yurong Chen, and Li Zhang. Decoder network over lightweight reconstructed feature for fast semantic style transfer. In *2017 IEEE International Conference on Computer Vision (ICCV)*. IEEE, oct 2017.
- [18] Yi-Lei Chen and Chiou-Ting Hsu. Towards deep style transfer: A content-aware perspective. In *Proceedings of the British Machine Vision Conference 2016*. British Machine Vision Association, 2016.
- [19] Carlos Castillo, Soham De, Xintong Han, Bharat Singh, Abhay Kumar Yadav, and Tom Goldstein. Son of zorn’s lemma: Targeted style transfer using instance-aware semantic segmentation. In *ICASSP*, pages 1348–1352. IEEE, 2017.
- [20] Xinyuan Chen, Chang Xu, Xiaokang Yang, Li Song, and Dacheng Tao. Gated-gan: Adversarial gated networks for multi-collection style transfer. *IEEE Trans. Image Processing*, 28:546–560, 2019.
- [21] Artsiom Sanakoyeu, Dmytro Kotovenko, Sabine Lang, and Björn Ommer. A style-aware content loss for real-time hd style transfer. *ECCV (8)*, 11212:715–731, 2018.
- [22] Huiwen Chang, Jingwan Lu, Fisher Yu, and Adam Finkelstein. Paired-CycleGAN: Asymmetric style transfer for applying and removing makeup. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. IEEE, jun 2018.
- [23] Sveinn Palsson, Eiríkur Agustsson, Radu Timofte, and Luc Van Gool. Generative adversarial style transfer networks for face aging. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE, jun 2018.

BIBLIOGRAPHY

- [24] Song Yuheng and Yan Hao. Image segmentation algorithms overview. *CoRR*, 2017.
- [25] Joao Carreira and Cristian Sminchisescu. CPMC: Automatic object segmentation using constrained parametric min-cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(7):1312–1328, jul 2012.
- [26] Pauline Luc, Camille Couprie, Soumith Chintala, and Jakob Verbeek. Semantic segmentation using adversarial networks. *CoRR*, abs/1611.08408, 2016.
- [27] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. In *ICLR (Poster)*, 2015.
- [28] Guosheng Lin, Chunhua Shen, Anton van den Hengel, and Ian Reid. Efficient piecewise training of deep structured models for semantic segmentation. In *CVPR*, pages 3194–3203, 2015.
- [29] E. Shelhamer, J. Long, and T. Darrell. Fully convolutional networks for semantic segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(4):640–651, 2014.
- [30] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. Learning deconvolution network for semantic segmentation. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1520–1528, December 2015.
- [31] Arnab Kumar Mondal, Jose Dolz, and Christian Desrosiers. Few-shot 3d multi-modal medical image segmentation using generative adversarial learning. *CoRR*, abs/1810.12241, 2018.

BIBLIOGRAPHY

- [32] Zhongyi Han, Benzheng Wei, Ashley Mercado, Stephanie Leung, and Shuo Li. Spine-GAN: Semantic segmentation of multiple spinal structures. *Medical Image Analysis*, 50:23–35, dec 2018.
- [33] Thomas Joyce, Agisilaos Chartsias, and Sotirios A. Tsaftaris. Deep multi-class segmentation without ground-truth labels. In *Medical Imaging with Deep Learning*, 2018.
- [34] Mina Rezaei, Konstantin Harmuth, Willi Gierke, Thomas Kellermeier, Martin Fischer, Haojin Yang, and Christoph Meinel. Conditional adversarial network for semantic segmentation of brain tumor. In *Brainlesion: Glioma, Multiple Sclerosis, Stroke and Traumatic Brain Injuries*, pages 241–252, Cham, 2018. Springer International Publishing.
- [35] Zeju Li, Yuanyuan Wang, and Jinhua Yu. Brain tumor segmentation using an adversarial network. In *Brainlesion: Glioma, Multiple Sclerosis, Stroke and Traumatic Brain Injuries*, pages 123–132. Springer International Publishing, 2018.
- [36] Edward Collier, Kate Duffy, Sangram Ganguly, Geri Madanguit, Subodh Kalia, Gayaka Shreekant, Ramakrishna Nemani, Andrew Michaelis, Shuang Li, Auroop Ganguly, and Supratik Mukhopadhyay. Progressively growing generative adversarial networks for high resolution semantic segmentation of satellite images. In *ICDM Workshops*, pages 763–769. IEEE, 2019.
- [37] Wei-Chih Hung, Yi-Hsuan Tsai, Yan-Ting Liou, Yen-Yu Lin, and Ming-Hsuan Yang. Adversarial learning for semi-supervised semantic segmentation. In *BMVC*, page 65, 2018.
- [38] Nasim Souly, Concetto Spampinato, and Mubarak Shah. Semi supervised semantic segmentation using generative adversarial network. In *2017 IEEE International Conference on Computer Vision (ICCV)*. IEEE, oct 2017.

BIBLIOGRAPHY

- [39] Xinming Zhang, Xiaobin Zhu, Xiao-Yu Zhang, Naiguang Zhang, Peng Li, and Lei Wang. SegGAN: Semantic segmentation with generative adversarial network. In *2018 IEEE Fourth International Conference on Multimedia Big Data (BigMM)*. IEEE, sep 2018.
- [40] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. The MIT Press, 2017.
- [41] Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th Annual International Conference on Machine Learning*. ACM Press, 2009.
- [42] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning.
- [43] Augustus Odena, Vincent Dumoulin, and Chris Olah. Deconvolution and checkerboard artifacts. *Distill*, 2016.
- [44] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, mar 1994.
- [45] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterton, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.
- [46] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks.

BIBLIOGRAPHY

- [47] Kaiming He and Jian Sun. Convolutional neural networks at constrained time cost. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, jun 2015.
- [48] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, jun 2016.
- [49] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *NIPS*, pages 2226–2234, 2016.
- [50] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks.
- [51] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets.
- [52] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets.
- [53] Scott Reed, Zeynep Akata, Xincheng Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee. Generative adversarial text to image synthesis.
- [54] Deepak Pathak, Philipp Krahenbuhl, Jeff Donahue, Trevor Darrell, and Alexei A. Efros. Context encoders: Feature learning by inpainting.
- [55] Yipin Zhou and Tamara L. Berg. Learning temporal transformations from time-lapse videos. In *Computer Vision – ECCV 2016*, pages 262–277. Springer International Publishing, 2016.

BIBLIOGRAPHY

- [56] Christian Ledig, Lucas Theis, Ferenc Huszar, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, and Wenzhe Shi. Photo-realistic single image super-resolution using a generative adversarial network.
- [57] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, 2015.
- [58] Donggeun Yoo, Namil Kim, Sunggyun Park, Anthony S. Paek, and In So Kweon. Pixel-level domain transfer.
- [59] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *ICCV*, pages 2242–2251. IEEE Computer Society, 2017.
- [60] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context. *CoRR*, 2014.
- [61] Yang Chen, Yu-Kun Lai, and Yong-Jin Liu. CartoonGAN: Generative adversarial networks for photo cartoonization. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9465–9474. IEEE, jun 2018.
- [62] Deepak Pathak, Philipp Krahenbuhl, Jeff Donahue, Trevor Darrell, and Alexei A. Efros. Context encoders: Feature learning by inpainting.
- [63] Hsin-Ying Lee, Hung-Yu Tseng, Jia-Bin Huang, Maneesh Kumar Singh, and Ming-Hsuan Yang. Diverse image-to-image translation via disentangled representations. In *ECCV (1)*, volume 11205 of *Lecture Notes in Computer Science*, pages 36–52. Springer, 2018.

BIBLIOGRAPHY

- [64] Lucas Theis, Aäron van den Oord, and Matthias Bethge. A note on the evaluation of generative models. In Yoshua Bengio and Yann LeCun, editors, *ICLR*, 2016.
- [65] Shane Barratt and Rishi Sharma. A note on the inception score. *CoRR*, 2018.
- [66] Mario Lucic, Karol Kurach, Marcin Michalski, Sylvain Gelly, and Olivier Bousquet. Are gans created equal? a large-scale study. In *NeurIPS*, pages 698–707, 2018.
- [67] Stefan Winkler. *Digital Video Quality*. John Wiley & Sons Ltd., feb 2005.
- [68] Z. Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, apr 2004.
- [69] Zhou Wang and A.C. Bovik. Mean squared error: Love it or leave it? a new look at signal fidelity measures. *IEEE Signal Processing Magazine*, 26(1):98–117, jan 2009.
- [70] Yang Gao, Abdul Rehman, and Zhou Wang. CW-SSIM based image classification. In *2011 18th IEEE International Conference on Image Processing*. IEEE, sep 2011.
- [71] David M. Rouse and Sheila S. Hemami. Analyzing the role of visual structure in the recognition of natural image content with multi-scale SSIM. In Bernice E. Rogowitz and Thrasyvoulos N. Pappas, editors, *Human Vision and Electronic Imaging XIII*, volume 6806, page 680615. SPIE, feb 2008.
- [72] W. P. Cockshott, S. L. Balasuriya, I. P. Gunawan, and J. P. Siebert. Image enhancement using vector quantisation based interpolation. In

BIBLIOGRAPHY

- Proceedings of the British Machine Vision Conference 2007*. British Machine Vision Association, 2007.
- [73] Richard Dosselmann and Xue Dong Yang. A comprehensive assessment of the structural similarity index. *Signal, Image and Video Processing*, 5(1):81–91, nov 2009.
- [74] Alain Hore and Djemel Ziou. Image quality metrics: PSNR vs. SSIM. In *2010 20th International Conference on Pattern Recognition*. IEEE, aug 2010.
- [75] Qiantong Xu, Gao Huang, Yang Yuan, Chuan Guo, Yu Sun, Felix Wu, and Kilian Weinberger. An empirical study on evaluation metrics of generative adversarial networks. *CoRR*, abs/1806.07755, 2018.
- [76] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *CVPR*, pages 2818–2826. IEEE Computer Society, 2015.
- [77] 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*, pages 6626–6637, 2017.
- [78] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Ji-aya Jia. Pyramid scene parsing network.
- [79] Xudong Mao, Qing Li, Haoran Xie, Raymond Y. K. Lau, Zhen Wang, and Stephen Paul Smolley. Least squares generative adversarial networks.
- [80] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan.
- [81] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of wasserstein gans.

BIBLIOGRAPHY

- [82] Fisher Yu, Wenqi Xian, Yingying Chen, Fangchen Liu, Mike Liao, Vashisht Madhavan, and Trevor Darrell. BDD100K: A diverse driving video database with scalable annotation tooling. *CoRR*, abs/1805.04687, 2018.
- [83] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *ECCV (7)*, volume 11211 of *Lecture Notes in Computer Science*, pages 833–851. Springer, 2018.