Supervised and Unsupervised Face Image Translation

Chang Liu[1]

MSc Machine Learning

Prof. Niloy Mitra

Submission date: 6th September 2019

Abstract

The image-to-image translation is an important problem in machine vision, where the main goal is to learn the relation between images from different domains. Our objective is to find a mapping between photos and art paintings of human faces. In this thesis, we explore both supervised and unsupervised learning methods to this problem. We will introduce an approach named Latent Features Transfer which could transfer the content and style information separately and recombine them to generate new images. Besides, we propose some improvements in CycleGAN model to achieve more stable training and to generate higher quality images.

Contents

Chapter 1

Introduction

Image-to-image translation is a class of machine vision problem aiming at finding a translation mapping between an image in one domain to a corresponding image in another domain. Super-resolution, attribute transfer and style transfer have shown success in image translation. Our problem could be studied in both supervised and unsupervised learning settings. If paired datasets of different domains are available, then this problem can be considered as a conditional generative problem or a simple regression problem. However, paired data are difficult and expensive to collect sometimes, so we need to learn in an unsupervised manner. Unsupervised learning is considered harder because of lacking corresponding images, but the introducing of generative adversarial networks has significantly improved this task.

In this project, we would explore these learning methods including style transfer, supervised generative model and unsupervised generative model. We apply our methods on face images of photos and artworks to find the relation between them. When seeing a portrait of artwork, people could easily imagine the corresponding human face. Could we perform this task using machine learning methods as well? This is a difficult problem since some portrait would not look the same as the person for some artistic purpose. So what we need to change is not only the colour and shape but also the style of an image. Style transfer is a common method of combine the content information and style information of different images [13]. The shortage of this method is the style of the generated image has to be from one or some particular photos, but we want a more generalized model. On the other hand, CycleGAN is a widely used model which could achieve unpaired image-to-image translation between two domains. However, it performs better only on colour and fine features changing, instead of shape and boundary. So it works well on style transfer of landscape

images [54], but not on face images which require more adaptation for image content. This project will firstly discuss the previous work on image translation in detail in chapter 2, including the background knowledge of convolutional neural network, variational auto-encoder, and generative adversarial networks. Then we will propose two methods modified from neural style transfer and CycleGAN respectively. The first one is called latent features transfer, which aims at finding a mapping of latent features to translate them between two domains. The other one combines the idea of feature matching in neural style transfer and CycleGAN with a packing discriminator. Chapter 4 will cover the experiment details and evaluation of the results.

Chapter 2

Related Work

In this chapter, an overview of the approaches to image-to-image translation is introduced, by firstly discussing these classical approaches that have been applied widely and successfully, and then concentrating on more recently proposed deep generative models. These methods are divided into style transfer and unsupervised deep generative models in this report, and the latter contains the Latent Variable Model and Implicit Probabilistic Model.

2.1 Neural Style Transfer

Changing the style of an image while preserving the semantic content of it is a difficult task in image processing. There is no artificial system with the same capability as human producing fine art because of the complexity of this process. Neural Style Transfer is an artificial system based on a deep convolutional neural network that could combine content and style of different images using neural representation [13]. This algorithm allows us to produce new images containing the content of an arbitrary image but with the style of a well-known masterpiece artwork.

2.1.1 Neural Networks (NN)

Neural networks are multi-layer networks of neurons which are applied successfully in classification, recognition ,and prediction, etc. In Multilayer Perceptron (MLP), the basic form of neural networks is the compositions of linear transforms and non-linear functions, which consist of an input layer, one or more hidden layers ,and an output layer [41, 11]. Each layer consists of neurons (units) and each neuron is fully connected to all neurons in

the previous layer where each connection has a weight associated with it. Neurons in the same layer are completely independent and do not share any connection. Every output unit (y_j) is a weighted (w_i) sum of inputs (x_i) and bias term (b_j).

$$y_j = \sum_i w_i x_i + b_j$$

Hence, a single linear layer network could be described with matrix-vector operations.

$$\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$

Linear layers typically are followed by a non-linear activation function for additional transformation, such as Sigmoid, Tanh, and ReLU. Then the next layer would take the outputs of the previous layer as its inputs. Neural network models could learn by optimizing a loss function over data with respect to the model parameters (weights and bias). The optimization methods are typically gradient descent which using chain rule derivatives and backpropagation on compute graph to update parameters. The total gradient of a neuron in each layer is the matrix multiplication of its local gradient and upstream gradient.

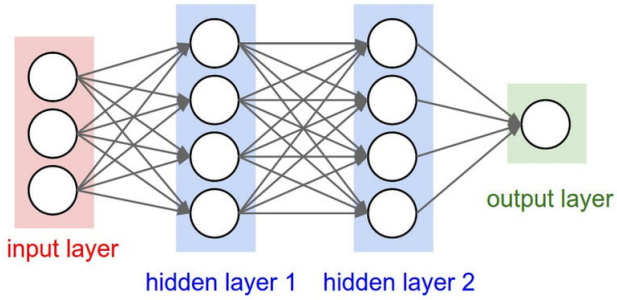


Figure 2.1: Neural Network [7]

2.1.2 Convolutional Neural Network (CNN)

A convolutional Neural Network is a Deep Learning algorithm which is made up of learnable weights and biases as ordinary Neural Networks. CNN was originally invented to solve the problem in machine vision, so the inputs of this model were assumed to be images. Two key properties of objects in images are Locality and Translation Invariance. The architecture of CNN is inspired by the organization of Visual Cortex where a single neuron only responds to stimuli in a restricted region of the visual field [10]. These networks could

preserve the spatial relationship between pixels by learning feature representations using small receptive field of input images. CNN models are designed to be invariant and widely used in image recognition because objects tend to have a local spatial structure and their appearances are independent of locations.

In addition to pixel dependencies, computation is another reason that CNN would outperform Multilayer Perceptron in image processing. Suppose the input images are of size $32 \times 32 \times 3$, then an ordinary neural network model would require the input images to be flattened to 3072×1 vectors. As a result, a single fully-connected neuron in the first hidden layer of a regular neural network would need 3072 weights. Although these amount of weights seem to be manageable, fully-connected structures are not suitable for high-resolution images, since a single neuron is almost certainly not enough for any task.

CNN structures are completely differentiable and parameters could learn through back-propagation as ordinary NN. A typical CNN architecture is a stacking model consists of three main types of layers: Convolutional Layer, Pooling Layer, and Fully-Connected Layer, where the last one is exactly as mentioned in section 2.1.1. In a CNN model, the input images are transformed layer by layer from the original pixel values to classification scores. Notice that some of these layers are parameterized (convolutional layers) and some are not (ReLU layers and pooling layers), and more details would be given in the following two sections.

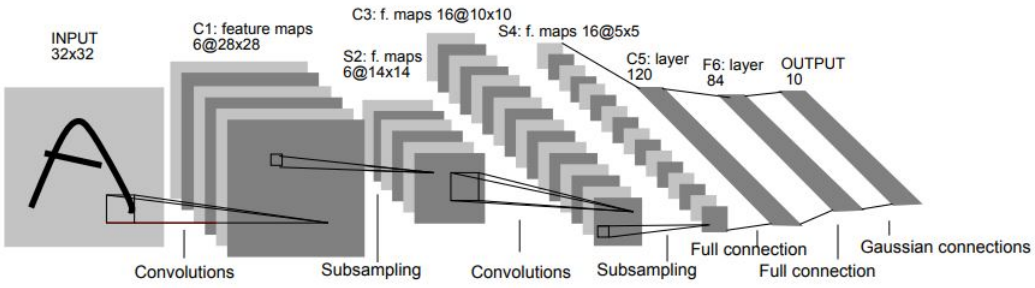


Figure 2.2: Convolutional Neural Network [24]

Convolutional Layers

The objective of convolutional layers is to extract the high-level features from in image. CNN need not to be restricted to only one convolutional layer, the first convolutional layers capture the low-level features and deeper convolutional layers are responsible for higher-level features. The convolutional layer is the core layer in the whole CNN structure and

it is also the most computationally expensive layer. The number of parameters reduced by first making fully-connected layer locally-connected, then sharing the weights of kernels connected to different locations. The parameters of convolutional layers consist of a set of learnable kernels (filters), which corresponds to neurons in ordinary neural networks. In 2D-convolution, we usually suppose that the inputs are three-dimensional matrices with heights, widths, and channels where channels are also the numbers of feature maps.

Each kernel is connected to a local rectangular area, called receptive field, and different kernels connected to different locations. If the convolutional layer follows an input layer, then the input of the convolutional layer is a 3D matrix with pixel values and each kernel takes element-wise dot product with a small region on the image instead of the whole image. And if in deeper network architecture, the input would be the output feature maps from the previous layer and kernels are connected to these feature maps within a window. The appearance of an object is independent of the object location, so we could use the same parameters to compute different spatial position which dramatically reduced the number of parameters. Sharing weights not only reduce the computation but also ensure the translation invariance. Each kernel is applied to all locations and the output is equivariant to translation.

To compute the output feature map, sliding the receptive field of the kernel over the input and every time performing an element-wise dot product between the kernel and corresponding region of input feature maps. Receptive field size is often called filter size which is fixed and the size of every kernel is spatially (width and height) smaller than the input, but including the full depth of the input. The number of output channel is a hyperparameter corresponding to the number of filters. Every filter would generate a one-channel feature map capturing different features and they are stacked together over the channel dimension.

The resolution of output is the dimensionality (height and width) of the output feature map which could be increased, reduced or remain the same compared to the input features. For a given input of size $N \times N$, the output size of feature map ($M \times M$) depends on kernel size (k), padding size (p), stride size (s), dilation rate(r) or fractional-stride size (s). Padded convolution usually is used to preserve resolution by padding the input with p columns and rows on both sides where the padded numbers are usually zeros. Strided convolution is where convolution filter is applied with a stride between receptive fields. The purposes of strided convolution are to reduce spatial resolution for faster processing and achieve

invariance to local translation.

$$M = \lfloor \frac{N + 2p - k}{s} \rfloor + 1$$

Dilated convolution is where the filter is applied with a dilation rate between kernel elements. The purpose of dilated convolution is to reduce computations by receiving the signals from a large receptive field with a small kernel.

$$M = \lfloor \frac{N + 2p - k - (k-1)(r-1)}{s} \rfloor + 1$$

Transposed convolution does the opposite of strided convolution and it increases the resolution of feature maps.

$$M = s(N-1) + k - 2p$$

Pooling Layers

The Pooling layer is usually used to decrease the computational power required to process the data and to achieve invariance to local translation. It is common to insert a Pooling layer between several convolutional layers in CNN. The function of pooling layer is to dramatically reduce the spatial size of the representation (while this operation is independent on each channel) to reduce the number of parameters and computation in the network. As a result, it compresses feature representations of an input image, and hence also reduce the over-fitting of the training data by the model. Besides, it is useful for extracting dominant features which are rotational and positional invariant.

Pooling layer also has a receptive field which is often much smaller than that of the convolutional layer. Besides, the stride size usually equals to the size of the receptive field to cover all area while avoiding overlap. We usually use a pooling layer with kernels of size 2 and stride size 2 to downsampling the input by a scale of 2. The process of downsampling can be performed in many ways including two main types: Max pooling and Average pooling. Max pooling return the max value of input over the receptive field which means it selects the most active neuron covered by the kernel, while average pooling computes the average input over the receptive field.

2.1.3 Deep Learning

In the neural network or convolutional neural network, each parameterized layer performs a linear operation, followed by a non-linear operation such as ReLU or Tanh. The rectified linear activation function (ReLU) will output the input directly if is positive, otherwise, it will output zero. Each layer can be seen as a weak linear classifier itself. More layers means more non-linearities which leads to a more discriminative model.

Generally, a deep learning model is better when it is deeper, but the number of layers in CNN has a limit. Firstly, CNN models used pooling layer after each convolutional layer, so the number of pooling layers is limited by the input image resolution, which is log(N) for NN input. Another reason is computational complexity since obviously a deeper model would have more parameters, and hence more computation is needed when computing backpropagation. Finally, the structure of CNN is deeper, the model is harder to optimize. The major problem cause the phenomenon is gradient instability. When we backprop through many layers, we need to compute the product of local gradients to get the total gradient of loss. If the gradients between layers are close to zeros, then the total gradient is very small and the network would be hard to train, which is called vanishing gradient problem. Some activation functions, like Sigmoid and Tanh, squeeze a large input space into a smaller space, and hence a large change in the input would be reduced to a small change in the output. On the other hand, if the local gradients are too large, the total gradient could overflow and this is gradient exploding.

There are several solutions could be used to solve the problem of gradient instability. The simplest one is to change activation functions, such as ReLU, which does not decrease the gradients. Weights could be initialized from zero-mean normal distribution and variance is adaptively chosen for each layer. Batch normalization could be another solution to this issue. A batch normalization layer simply reduces this problem by normalizing the input to zero mean and unit variance, so that most of it falls in a suitable region where the derivative is not too small or too large. Residual networks could also solve this problem, as they provide residual connections straight to earlier layers. The residual layer is where the output of two convolutional layers is added up by the original input, so the gradients are not decreased to zeros.

2.1.4 Deep Image Representation

The essential part of Neural Style Transfer is image representation, which means we need to extract the content features and style features of images. We use VGG19 network which is a deep convolutional network to achieve that. VGG19 model was originally trained to perform object recognition and localization problem [45]. VGG19 architecture consists of 16 convolutional layer, 5 pooling layers and 3 fully-connected layers as illustrated in Figure 2.3. We use the feature space of the output from some chosen convolutional layers and we do not use any fully-connected layer. The model and the pre-trained weights are publicly available. For image synthesis average pooling would yield slightly more appealing results than maximum pooling operation [13].

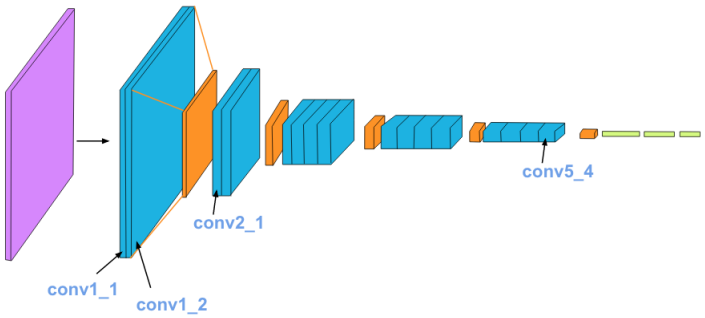


Figure 2.3: VGG19 Architecture [36]

Content Representation

Each layer in the network is a filter bank and the complexity increases as the layer becoming deeper in the architecture. An input image x in the form of a 3D matrix is encoded to different feature spaces by different convolutional layers in the CNN model. This content representation was first introduced by Gatys et al. in 2015 [13]. A layer with C kernels has C feature maps each of size $M \times M$, where M is the height and the width of the output feature map. So the features extracted by a layer can be stored in a matrix $F \in \mathbb{R}^{C \times M^2}$ where F_{ij} is the activation of the i-th kernel at position j in this layer. Visualization of the content information encoded at different layers could be done by performing gradient descent on a white noise image and reconstruct an image to match the features extracted by the content image. Let p and x be the content image and the generated image, and P and F be their feature representation respectively. Then we define the content loss as the

square loss between two feature representations:

$$\mathcal{L}_{content}(p, x) = \frac{1}{2}\sum_{i,j}(F_{ij} - P_{ij})^2$$

Then the gradient of this loss with respect to the activations equals

$$\frac{\partial \mathcal{L}_{content}}{\partial F_{ij}} = \begin{cases} (F - P)_{ij} & \text{if } F_{ij} > 0 \\ 0 & \text{if } F_{ij} < 0 \end{cases}$$

and hence the gradient with respect to the generated image could be computed by back-propagation. We optimize over the image x until its extracted features match the features of content image p. When CNN models are trained on object recognition, the weights are optimized to develop a representation of the image that the object information becoming more explicit as the layer becoming deeper. Therefore, as the input image passing to higher-level layer, the feature maps is increasingly sensitive to the coarse structure, but relatively invariant to its fine details. Hence, deeper layers could capture the higher-level content in terms of shape and border, but do not restrict the exact pixel values of the reconstruction. On the contrary, reconstructing by matching the lower-level features would result to reproduce the exact pixel values of the content image. Therefore, we usually select a relatively high-level feature space to represent the content information, conv3_2 or conv4_2 in Figure 2.3.

Style Representation

We use a texture model to extract the style representation of an input image [12]. This feature space could be built by a linear combination of output feature maps in the VGG19 network. It contains the averaged correlations between different output feature maps over the spatial extent of the feature maps, which could be expressed by Gram matrix $G \in \mathbb{R}^{C\times C}$. G_{ij} is the inner product of feature maps i and j in matrix F:

$$G_{ij} = \sum_k F_{ik}F_{jk}$$

We obtain a stationary representation of the input image by including the feature correlations of different layers. This feature space captures the texture information but not the global structure. Similarly, we could visualize the style information by matching the

Gram matrices of the style representation layers of a white noise image and the input style image. This could be done by using gradient descent on the mean-squared error between Gram matrices of the style image and the generated image. Let s and x be the original style image and generated image, and S and G be their Gram matrices respectively. For a single layer l, the style loss is

$$E_l = \frac{1}{4C_l^2 M_l^4} \sum_{i,j} \left(G_{ij}^l - S_{ij}^l\right)^2$$

and hence the total style loss is

$$\mathcal{L}_{\text{style}}(x, s) = \sum_{l \in L} w_l E_l$$

where w_l are the individual layer style weights and L are the layers that chosen to match the Gram matrix. We select conv1_1, conv2_1, conv3_1, conv4_1 and conv5_1 to be L for the most appealing results. The derivative of E_l with respect to the activations in layer l can be computed as:

$$\frac{\partial E_l}{\partial F_{ij}^l} = \begin{cases} \frac{1}{C_l^2 M_l^4} \left(\left(F^l\right)^{\mathrm{T}} \left(G^l - S^l\right)\right)_{ji} & \text{if } F_{ij}^l > 0 \\ 0 & \text{if } F_{ij}^l < 0 \end{cases}$$

and hence the gradient with respect to the generated image could be computed by back-propagation.

Style Transfer

To transfer the style of a style image s to a content image c, we generate a new image that matches the style information of s and the content information of c at the same time [13]. This could be done by minimizing the weighted sum of style loss of the selected layers in VGG19 between the generated image and the style image, and content loss between the generated image and the content image. The total loss function is:

$$\mathcal{L}_{\text{total}}(c, s, x) = \alpha \mathcal{L}_{\text{content}}(c, x) + \beta \mathcal{L}_{\text{style}}(s, x)$$

where hyperparameters α and β are the weight factor between content loss and style loss. This ratio will affect how stylized the final image is.

2.2 Deep Generative Models

Generative Models learn data distribution using unsupervised learning and have achieved outstanding results in recent years. They aim at learning the training set data distribution of and generate new data points in the input domain with some variations. However, learning the exact distribution is not always feasible either implicitly or explicitly, so we try to model the target distribution using a composition of simpler distributions. Generative models are essentially probabilistic models of high-dimensional data. It could be trained to generating new observation though learning the probabilistic process. And the key point of Generative models is on capturing the dependence between the dimensions. It could be used to model many types of dependence including spatial, temporal and sequential dependence. Generative models usually are used on understanding the data, translating between domains and outlier detection. Variational Autoencoders (VAE) [20] and Generative Adversarial Networks (GAN) [14] are the most commonly used approaches. VAEs intend to maximize the lower bound of log-likelihood and GAN intend to achieve the Nash equilibrium between Generator and Discriminator. Details of structures of VAE and GANs and the intuition behind them will be analyzed in the following section.

2.2.1 Variational Auto-Encoder (VAE)

A generative model is difficult to train when the dependencies between the dimensions are complicated. When generating images of handwritten characters, it would be more efficient if the model decides which character to generate before it computes the exact pixel values. This kind of decision is formally called a latent variable. A latent variable model (LVM) defines a distribution over observations x by using a latent variable z, which explains correlations in $\boldsymbol{x}$ by assuming dependence on latent variables $\boldsymbol{z}$. Correlations in high-dimensional $\boldsymbol{z}$ may be captured by fewer parameters in the latent variable model. There is a prior distribution $p(z)$ for the latent variable and a likelihood $p(x|z)$ that connects the latent variable to the observation. The prior and the likelihood define the joint distribution $p(x, z) = p(x|z)p(z)$, and we will also be interested in the posterior distribution $p(z|x)$. To generate an observation $\boldsymbol{x}$ from the model we simply sample as follows:

$$\boldsymbol{z} \sim p(\boldsymbol{z})$$

$$\boldsymbol{x} \sim p(\boldsymbol{x}|\boldsymbol{z})$$

The posterior distribution for the given observation could be computed as:

$$p(\boldsymbol{z}|\boldsymbol{x}) = \frac{p(\boldsymbol{x}, \boldsymbol{z})}{p(\boldsymbol{x})} = \frac{p(\boldsymbol{x}, \boldsymbol{z})}{\int p(\boldsymbol{x}, \boldsymbol{z}) d\boldsymbol{z}}$$

This is also called inference which is used to explain the observation and train the LVM.

Maximum Likelihood (ML) Learning

Maximum Likelihood is the dominant estimation principle for probabilistic models. The goal of Maximum Likelihood learning is to find the set of parameters (θ) that maximize the likelihood

$$\theta^* = \arg\max_{\theta} \sum_{i=1}^{N} \log p_\theta \left(\mathbf{x}_i\right)$$

This optimization problem does not have a closed-form solution for latent variable models, and hence we use iterative algorithms to compute. Maximizing likelihood is equal to maximizing log-likelihood, so we compute the gradient of local log-likelihood:

$$\nabla_\theta \log p_\theta(\boldsymbol{x}) = \int p_\theta(\boldsymbol{z}|\boldsymbol{x}) \nabla_\theta \log p_\theta(\boldsymbol{x}, \boldsymbol{z}) d\boldsymbol{z}$$

Expectation Maximization (EM)

For many models, maximization might be straightforward if $\boldsymbol{z}$ were not latent, and if θ is given, the values of $\boldsymbol{z}$ could be easily computed. The Expectation Maximization algorithm updates latent variable distribution and parameters iteratively instead of maximizing log-likelihood directly [32]. In E step, simply computes the posterior according to the given data, which is also called inference.

$$q_i^{t+1} = p_{\theta^t}(\boldsymbol{z}|\boldsymbol{x}_i), \text{ for } i = 1, \ldots, N$$

In M step, maximizing likelihood as if latent variables were not hidden, which is also called learning.

$$\theta^{t+1} = \arg\max_{\theta} \sum_{i=1}^{N} \mathbb{E}_{q_i^{t+1}}[\log p_\theta(\boldsymbol{x}_i, \boldsymbol{z})]$$

Essentially, EM algorithm refines a lower bound on the log-likelihood and maximizes it. Since log function is concave, we could use Jensen's inequality to get a lower bound:

$$\log p_\theta(\boldsymbol{x}) = \log \int p_\theta(\boldsymbol{z}, \boldsymbol{x}) d\boldsymbol{z} = \log \int q(\boldsymbol{z}) \frac{p_\theta(\boldsymbol{z}, \boldsymbol{x})}{q(\boldsymbol{z})} d\boldsymbol{z} \geq \int q(\boldsymbol{z}) \log \frac{p_\theta(\boldsymbol{z}, \boldsymbol{x})}{q(\boldsymbol{z})} d\boldsymbol{z} = \mathcal{F}(\boldsymbol{x}, \theta, q)$$

where $\mathcal{F}(\boldsymbol{x}, \theta, q)$ is known as free energy or evidence lower bound (ELBO), and $q(\boldsymbol{z})$ is approximate posterior distribution. Then the free energy can be re-written as

$$\mathcal{F}(\boldsymbol{x}, \theta, q) = \log p_\theta(\boldsymbol{x}) - \mathrm{KL}[q(\boldsymbol{z}) \| p_\theta(\boldsymbol{z}|\boldsymbol{x})]$$

The first term is reconstruction cost which measures the ability of $q(\boldsymbol{z})$ to explain the data y. The second term is the Kullback-Leibler divergence and it is non-negative. It is the penalty which ensure the approximate posterior distribution $q(\boldsymbol{z})$ to be closed to the beliefs $p(\boldsymbol{z})$. Therefore, for a fixed θ, $\mathcal{F}$ is bounded above by the log-likelihood, and the bound is tight when KL divergence equals zero. KL divergence equals zero if and only if $q(\boldsymbol{z}) = p_\theta(\boldsymbol{z}|\boldsymbol{x})$, hence E step actually sets

$$q^{(k)}(\boldsymbol{z}) = p_{\theta^{(k-1)}}(\boldsymbol{z}|\boldsymbol{x})$$

On the other hand, $\mathcal{F}(\boldsymbol{x}, \theta, q)$ could also be re-written as

$$\mathcal{F}(\boldsymbol{x}, \theta, q) = \mathbb{E}_{q(\boldsymbol{z})}[\log p_\theta(\boldsymbol{z}, \boldsymbol{x})] + \mathbf{H}[q]$$

where $\mathbf{H}[q]$ is the entropy of $q(\boldsymbol{z})$. Then M step becomes

$$\theta^{(k)} = \arg\max_\theta \mathbb{E}_{q^{(k)}(\boldsymbol{z})}[\log p_\theta(\boldsymbol{z}, \boldsymbol{x})]$$

E and M step together never decrease the value of the log-likelihood, because E-step brings the free energy to the likelihood for fixed θ, M-step maximizes the free energy with respect to θ and Jensen's inequality would guarantee that free energy is bounded above by log-likelihood.

However, inference in EM algorithm needs to be tractable which requires going through the entire training set for every parameter update.

Variational Inference

Exact inference is tractable for very few models. To solve this problem, we could use approximate inference instead. There are two general approaches of approximate inference, sampling-based approximation, and variational inference. Markov Chain Monte Carlo is a widely used sampling method, but it is computationally expensive and convergence is hard to diagnose in this model. Variational inference is to approximate the exact posterior with a distribution from a tractable family $q_\phi(\boldsymbol{z})$ parameterized by ϕ [15]. Then the inference problem is reduced to optimization of $\mathrm{KL}[q_\phi(\boldsymbol{z})\|p_\theta(\boldsymbol{z}|\boldsymbol{x})]$ with respect to the distribution parameters ϕ. Since $\log p_\theta(\boldsymbol{x})$ is independent of ϕ, then we can simply maximize the free energy $\mathcal{F}(\boldsymbol{x}, \theta, \phi)$ with respect to ϕ instead. We could replace the E step in the EM algorithm by VI to get variational EM. Then the variational E step becomes

$$\phi = \arg\max_{\phi} \mathcal{F}(\boldsymbol{x}, \theta, \phi)$$

Variational EM restricts the expression of a model and thus leads to suboptimal performance. Besides, the variational EM algorithm does not guarantee to increase the likelihood, because the variational E step not necessarily brings free energy to the likelihood. It can under-estimates the variance of the posterior if using limited approximation. However, VI applies to almost all probabilistic models and it converges faster than competing methods.

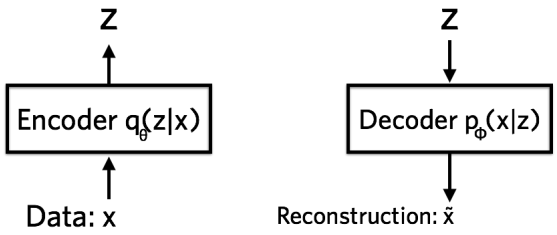


Figure 2.4: Encoder-Decoder in autoencoder [2]

Reparameterization Trick

The gradient of the model parameters is

$$\nabla_\theta \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}[\log p_\theta(\boldsymbol{z}, \boldsymbol{x})] = \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}[\nabla_\theta \log p_\theta(\boldsymbol{z}, \boldsymbol{x})]$$

So we only need to generate one or more samples from the variational posterior and average the resulting gradients. The gradient of variational parameters can not be computed directly compare to model parameters. Reparameterization trick provides an easy way of computing gradients for $\nabla_\phi \langle f(\boldsymbol{z}) \rangle_{q_\phi(\boldsymbol{z})}$. A random variable $\boldsymbol{z}$ sampled from the approximated posterior distribution $q_\phi(\boldsymbol{z})$ for some fixed distribution $p(\boldsymbol{\epsilon})$ could be expressed as a deterministic variable $z = g(\boldsymbol{\epsilon}, \phi)$ for some function g where $\boldsymbol{\epsilon}$ is some auxiliary noise variable [20]. Then as long as $g(\boldsymbol{\epsilon}, \phi)$ is differentiable w.r.t. ϕ, we have

$$\nabla_\phi \mathbb{E}_{q_\phi(\boldsymbol{z})}[f(\boldsymbol{z})] = \mathbb{E}_{p(\boldsymbol{\epsilon})}\left[f'(g(\boldsymbol{\epsilon}, \phi))\nabla_\phi g(\boldsymbol{\epsilon}, \phi)\right]$$

Backpropagation cannot flow through a random node, and reparameterization trick makes $\boldsymbol{z}$ become a deterministic node to do backpropagation, which essentially moves the dependence on the distribution parameters inside the expectation. Take Gaussian variable as an example, and let $z \sim p(z|x) = \mathcal{N}(\mu, \sigma^2)$. In this case, we could use $z = \mu + \sigma\epsilon$ with $\epsilon \sim \mathcal{N}(0, 1^2)$.

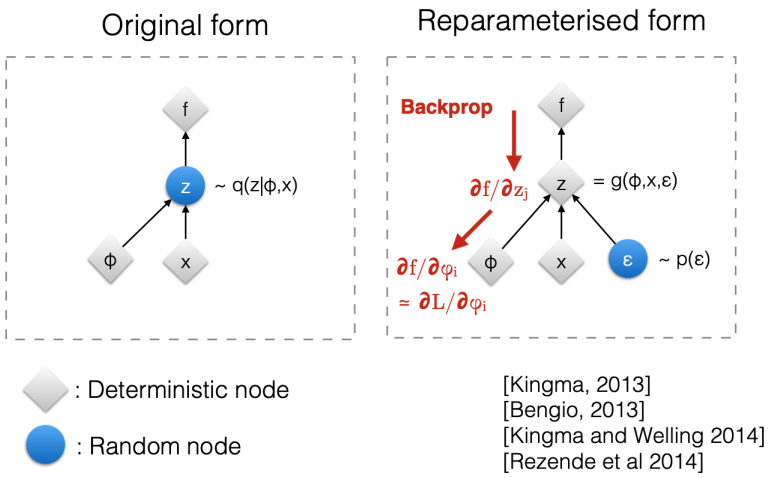


Figure 2.5: Reparameterization Trick [22]

Variational Autoencoder

Before introducing the variational autoencoder, we would give a brief explanation of the standard autoencoder. An autoencoder is a pair of two connected networks, an encoder, and a decoder [20]. The encoder is a neural network which takes a datapoint x as its input and compresses it to a dense representation z into a lower-dimensional space. The

decoder is another neural network that can convert the representation z back to the original input. The output representation z from the encoder has to contain enough information for the decoder to process it into the desired format. Generally, the entire autoencoder network is trained as a whole, optimized through backpropagation [39]. The loss function is the reconstruction loss which is usually the mean-squared error or the cross-entropy loss between the input and the output. This makes sure that the output of the network is closed to the input by penalizing the pixel-wise differences. Since the dimension of z is much less than the input, the encoder must choose to discard the information. The encoder needs to learn to preserve as much of the relevant information as possible in the limited units while discarding the irrelevant information. On the other hand, the decoder learns to reconstruct the image properly using the low-dimension representation z. Although the standard autoencoders compress and reconstruct their input well, the encoded vector can only be converted to the original input. It cannot be used to generate images by randomly sampled z from the latent space, because the latent space may not be continuous, or allow easy interpolation. When building a generative network, we usually want to generate new outputs with some variability. If space is discontinuous and the input to the decoder is a sample in the gap, the decoder would generate an unrealistic output, because the decoder never trained on that latent space.

To solve this problem, the model needs to learn the distribution of the training data. VAE is one of the most popular approaches to learn the complicated data distribution, which is a generative model using neural networks with continuous latent space. This is achieved by output two latent vectors, mean and variance, and sample from them. The encoder is trained to learn the underlying probability distribution of the training data, and the decoder learns to reconstruct the image from a randomly sampled data from the distribution. The variables on the low-dimensional space are called latent variable which is not directly observed but assumed to generate the data. The true probability distribution of latent variable z is denoted by $p(z)$, and a Gaussian distribution $\mathcal{N}(0, 1^2)$ is selected as the prior of $p(z)$ [42]. Related to the previous section, the encoder is the learned variational posterior $q_\phi(z|x)$ and the decoder is the likelihood $p_\theta(x|z)$ where they are both parameterized with neural networks. We use variational posterior $q_\phi(z|x)$ to model the true distribution $p(z)$ and minimize the difference between them using KL-divergence metric approach so that our hypothesis is close to the true distribution. The final objective function of VAE is

$$\mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})\right] - \mathrm{KL}\left(q_\phi(\boldsymbol{z}|\boldsymbol{x})\|p_\theta(\boldsymbol{z})\right)$$

and VAEs are trained by maximizing it. The first term is the negative log-likelihood which measures how well the model reconstructs the observation from a sample from the variational posterior. The second term acts as a regularizer [4], by encouraging the variational posterior not to deviate too much from the prior. Since KL-divergence is always larger or equal to zero, then the optimizing problem is actually maximizing $\mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})\right]$ and minimizing $\mathrm{KL}\left(q_\phi(\boldsymbol{z}|\boldsymbol{x})\|p_\theta(\boldsymbol{z})\right)$. $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ should be close to Gaussian distribution $\mathcal{N}\left(0, 1^2\right)$, so assume that it also follows a Gaussian distribution with parameters $\mu(\boldsymbol{x})$ and $\Sigma(\boldsymbol{x})$. Then the KL-divergence becomes

$$\mathrm{KL}\left(\mathcal{N}\left(\mu(\boldsymbol{x}), \Sigma(\boldsymbol{x})\right) \| \mathcal{N}\left(0, 1\right)\right) = \frac{1}{2}\sum_k \left(\exp(\Sigma(\boldsymbol{x})) + \mu^2(\boldsymbol{x}) - 1 - \Sigma(\boldsymbol{x})\right)$$

We can simply sample from $\mathcal{N}\left(0, 1\right)$ as the input of decoder to generate new images.

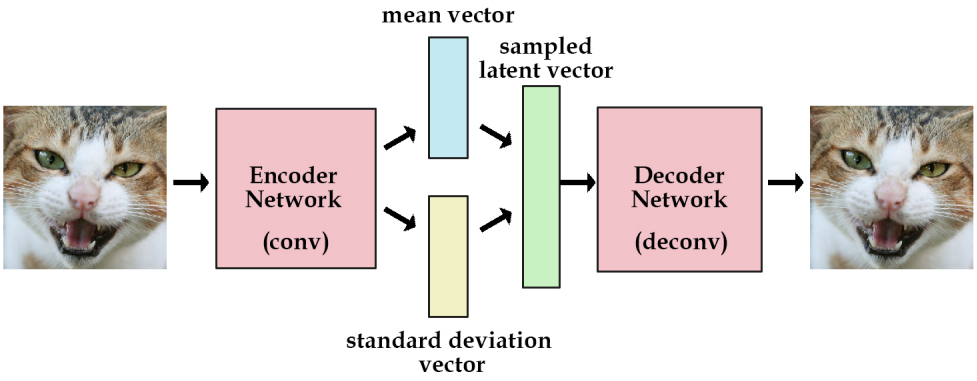


Figure 2.6: Structure of VAE [9]

2.2.2 Generative Adversarial Networks (GANs)

Generative Adversarial Networks (GANs) were introduced by Ian Goodfellow et al. in 2014 [14]. GANs are also neural networks that generate synthetic data like images using given input data. In fact, unlike VAE, GANs do not process any explicit probability density estimation. GANs are based on a game theory approach with an objective to find Nash equilibrium between Generator and Discriminator. The basic idea is to sample from a simple distribution like Gaussian and then learn to transform this noise to data distribution using universal function approximators such as neural networks.

A GAN model consists of two deep neural networks, a generator G that capture the underlying distribution of training data and try to generate samples from scratch following the distribution, and a discriminator D that estimate the probability that a given input data belongs to the real training set. The goal of the generator is to fool the discriminator by closely approximating the underlying distribution in order to generate samples that are indistinguishable from the actual data. To synthesize these new samples, the generator is given random noise as its input and attempts to generate realistic images from the learned distribution of the training data. On the other hand, the task of the discriminator is to distinguish between generated samples from the generator and real samples from the training set, which is essentially a binary classifier. This is achieved by adversarial training of these two networks.

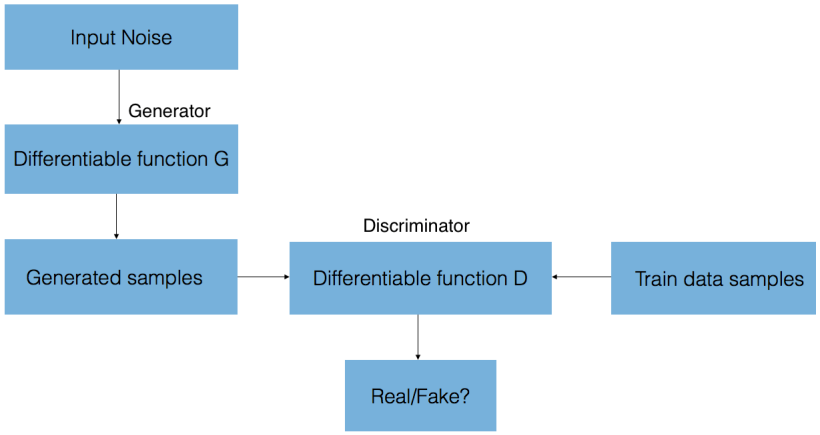


Figure 2.7: Structure of GAN [47]

The generator is a differentiable function G, which parameterized by neural networks and can be learned by gradient descent. The generator takes a fixed-length random vector as its input and outputs a sample in the domain of real dataset. We define a prior on the input variables as $p(z)$, usually a Gaussian distribution, and the latent vector z is drawn randomly from $p(z)$. Essentially, z is the input of the generative process which is a vector of unstructured noise. Conceptually, the input z should represent the latent features of the images generated, for example, the colour and the shape. We do not specifically control the semantic meaning of z, and let the training process to learn it. Like the generator, the discriminator is also a differentiable function D, which parameterized by neural networks and can be learned by gradient descent. The discriminator takes images as its input and

the output is a decision between zero and one, which is the probability of the input images to be in the training data distribution. When training the discriminator, if the input x of D is the sample generated by generator G, say $G(z)$, the ideal output should be a value closed to zero, indicating that the sample is fake. While the input is sampled from the real training data, it should output a value close to one, which means the sample is real.

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

The first part is to recognize real images better and the second part is to recognize generated images better. However, when training the generator, we expect the output of discriminator on generated images to be closed to one.

$$\min_G V(G) = \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}}(\boldsymbol{z})[\log(1 - D(G(\boldsymbol{z})))]$$

In other words, D and G play the following two-player minimax game [14] with objective function $V(G, D)$:

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

We choose to use the logarithm because it is more numerically stable as a loss function. After the objective function is defined, the generator and the discriminator are trained alternatively by gradient descent. When training the discriminator, we fix the parameters of the generator model and perform a single iteration of gradient descent on the discriminator using both real and the generated images. Then we train the generator by fixing the parameters of discriminator and updating the parameters of the generator for a single iteration. Ideally, the generator and the discriminator would reach Nash equilibrium, where each network is at its best configuration with respect to the other [44]. After the training process, we could simply sample from the prior distribution and feed it to the forward path.

DCGANs

Deep Convolutional Generative Adversarial Networks (DCGANs) is one of the earliest, most popular and successful network design on GAN employing Convolutional Neural Network, which is proposed by Radford et al. in 2015 [38]. This network also consists of a generator and a discriminator. Just like the vanilla GANs network, the generator takes

as input a 100 random variable vector z drawn from a prior distribution and outputs an RGB image of shape 64×64. The input of discriminator is 64×64 RGB image and output is a probability between zero and one. The whole network uses convolutional networks in place of multiplayer perceptrons.

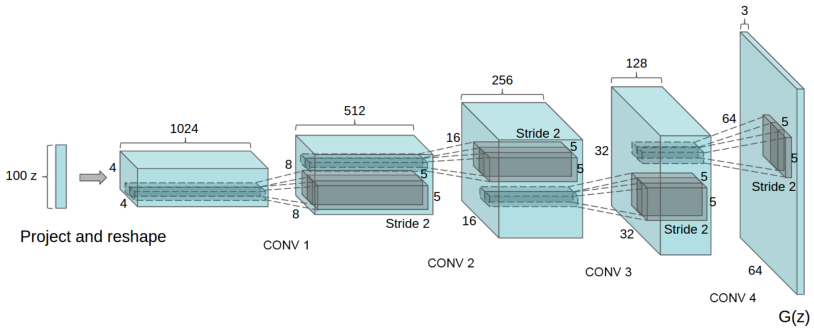


Figure 2.8: Generator of DCGANs [38]

In the generator, it uses deconvolutional layers for upsampling to increase the image resolution to the desired output size. Batch Normalization is used after every convolutional layer except the output layer for the generator and the input layer of the discriminator to stabilize the training network. ReLU activation is used in the generator for all layers except the output layer which uses Tanh layer [34]. In the discriminator, all the max-pooling layers are replaced by convolutional layers with stride size of two for the downsampling and eliminate all the fully connected layers. Besides, Leaky ReLU is used for all layers except for the output which uses sigmoid [29, 51]. DCGANs is trained using mini-batch stochastic gradient descent and Adam optimizer [21] was used to accelerate training with tuned hyperparameters.

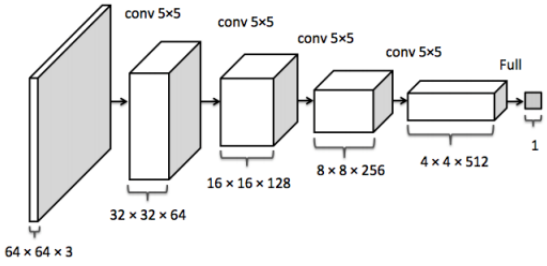


Figure 2.9: Discriminator of DCGANs [6]

conditional GANs

Training a GAN model is non-trivial and some additional information, such as a class label, can make the model perform better. There are mainly two motivations for using a class label information in a GAN model, improving the GAN and targeted image generation. The improvement of GANs might in the form of more stable training, faster training or generating better images. A major limitation of GANs is that it generates a random image and the type of generated image cannot be controlled. The relationship between the variable z on the latent space and the generated image is complex, but labels can act as an extension to the latent space to generate and discriminate images better. When the trained generator network is used as a standalone model to generate images, the type or the class of generated images can be controlled [8]. The labels are also added to the input of discriminator to distinguish real images better. This kind is network is called Conditional Generative Adversarial Networks (cGANs), which was first introduced by Mirza in 2014 [31]. GANs are conditioned on the class label, are called class-conditional GANs. And they can also be conditioned on their input images which is used for image-to-image translation tasks. The objective function of cGANs is

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

where $D(x|y)$ and $G(z|y)$ mean that the outputs of discriminator and generator depend on an image x or the latent variable z as well as a given label y.

WGANs

The Wasserstein GAN (WGAN) is an extension to GAN which could improve the stability when training the model as well as correlate the loss function and the quality of generated images by suing the Wasserstein distance instead of the JS-Divergence to measure the difference between the posterior distribution and target distributions [3]. It only requires a few minor modifications to the standard DCGAN model despite the dense mathematical motivation. This model was originally described by Martin Arjovsky, et al. in 2017 [3]. It uses an alternative way to train the universal function approximator to model the training dataset distribution. The WGAN model changed the discriminator to give a score of realness or fakeness instead of predicting the probability of generated images being real or fake [3]. This method trains the generator to minimize the distance between the distributions of the training set and generated images. Wasserstein distance could provide a meaningful

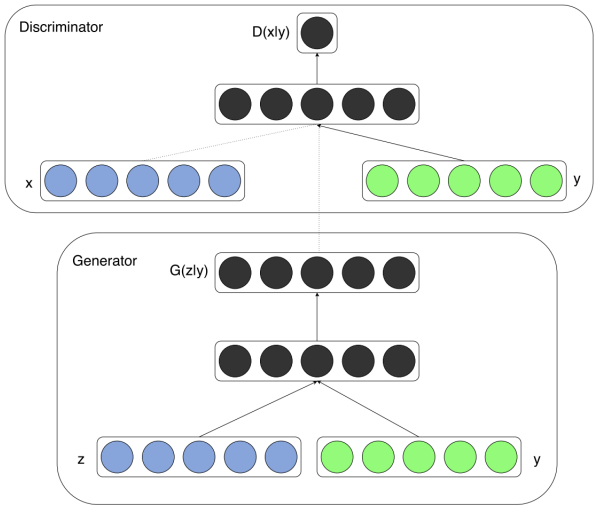


Figure 2.10: Conditional GANs [31]

and smooth representation of the distance in lower-dimensional manifolds which is more stable when training using gradient descents. In the WGAN model, the activation function in the output layer changed from sigmoid to linear. Besides, the class label of real images is changed to -1 and the class label of generated images is changed to 1. Since SGD is a minimization algorithm, we could multiply the realness and minimize the network. In the DCGAN, the generator and the discriminator are updated once in each iteration while in WGAN the discriminator model is updated four times more than the generator [3].

2.3 Image-to-Image Translation

In the following two sections, let us consider some GANs' application in image-to-image translation. Many problems in machine vision and image processing could be described as translating an input image from a domain into a corresponding output image in another domain. In some problems, a single input image may correspond to multiple possible images. The problems could be studied in both supervised and unsupervised fashion. In the supervised methods, paired data from different domains are required. The training process could be beneficial to ground truth in the paired data.

2.3.1 Pix2Pix

GANs are perfectly suited for image-to-image translation task and the Pix2Pix GAN is a general approach designed for this kind of tasks which are nearly impossible to hard-code. This approach was firstly presented by Phillip Isola, et al. in 2016 [17]. It is based on the cGAN we have mentioned in the previous section, where the generation of an image is conditional on a given image. The cGAN model is an extension of the GAN architecture that provides control over the generated image which allows an image of a given class to be generated. In this case, the Pix2Pix GAN changes the loss function so that the generated image is both plausible in the content of the target domain, and is a plausible translation of the input image. The generator model is trained to both fool the discriminator model and to minimize the loss between the generated image and the expected target image. Therefore, the training process of the Pix2Pix GAN model requires paired image datasets consisting of images from the original domain and images from the target domain. Both the generator and discriminator models use standard Convolution-BatchNormalization-ReLU blocks [16] of layers as is common for deep convolutional neural networks [17]. This general architecture allows the Pix2Pix model to be trained for a range of image-to-image translation tasks.

U-Net Generator

The architecture of U-Net generator [40] used in this Pix2Pix model was a very novel component of this paper, which does not take a vector z of size 100×1 on latent space as its input. In a traditional GAN model, the generator would take a latent vector z as its input and project it onto a higher-dimensional space to output an image with desired spatial resolution. Instead, U-Net generator takes in the Image from the source domain, compresses it into a low-dimensional bottleneck vector representation, and then learns how to upsample it into the output image in the target domain. The U-Net architecture is very similar to the standard encoder-decoder generator, but links or skip-connections are made between layers of the same size when downsampling and upsampling. Then just like a ResNet block, the information from earlier layers are integrated into later layers, which allows the bottleneck to be circumvented. This U-shaped model is convenient because it does not require any resizing or projections since the spatial resolution is symmetric and matched. Besides, the source of randomness comes from the use of dropout layers that are used both during training to encourage variation and when a prediction is made instead of random sampling from the latent space.

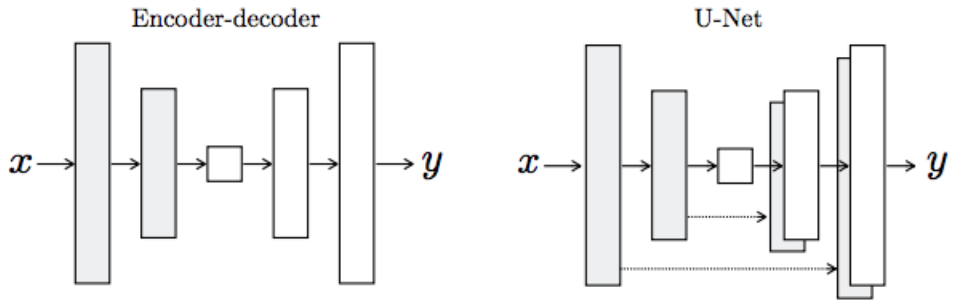


Figure 2.11: U-Net Generator [17]

PatchGAN Discriminator

L2 loss and L1 loss would result in blurry generated images [23] which means these losses cannot capture the high-frequency crispness. The discriminator model in Pix2Pix is called PatchGAN, which is motivated by restricting the discriminator to focus on high-frequency structure. Since we need to consider only on the local image patches, PatchGAN penalizes structure at the scale of patches. This discriminator works by classifying each $N \times N$ individual patch in the image is real or fake and averaging all probabilities instead of predicting on the entire image [17]. Another advantage of PatchGAN is it has much fewer parameters on small patches and runs faster than the traditional discriminator while it could still produce high-quality results. A patch size of 70×70 was found to be effective across a range of image-to-image translation tasks where 70×70 is the size of the receptive fields of the discriminator [54]. The PatchGAN discriminator could understand both content and style loss [25].

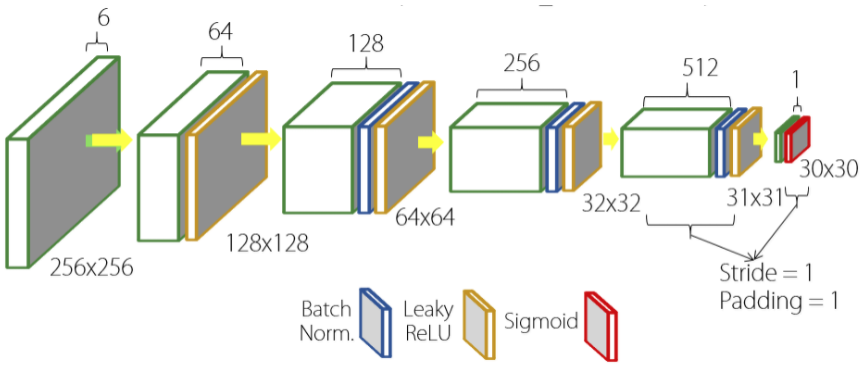


Figure 2.12: Example of 70x70 PatchGAN Discriminator [19]

Evaluation

The training strategy of the discriminator in this model is in the same way as the discriminator in a traditional GAN model, by minimizing the negative log-likelihood of classifying real and fake images with conditions on a source image. The loss of discriminator is halved to slow down the training process because the discriminator is trained much faster than the generator.

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

It has been found that training the generator model using the adversarial loss with a more traditional loss is beneficial and feasible. Then the generator not only aims to generate images in the target domain but also regularized to be near the ground-truth output by, say L1 loss [37]. In this problem set, we choose to use L1 distance or mean absolute pixel difference between the generated translation of the source image and the expected target image rather than L2 as L2 would result in blurry images [23].

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

Hence, the final objective is

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

where λ is a hyperparameter which controls the combination ratio between the adversarial loss and the L1 loss.

2.3.2 BicycleGANs

Pix2Pix network has been shown to generate high-quality results for image-to-image translation task. BicycleGAN model was introduced by Zhu, et al. in 2017 [53], which is an extension of the Pix2Pix framework we just mentioned in the last section. In this model, we enforce a bijection between the output space and the latent space. We not only map the latent code z to the target output domain but also learn an encoder mapping the output back to the latent space [53]. Since the mapping is a bijection, the reconstructed latent vector would be encouraged to be the same. This model consists of two components, Conditional Variational AutoEncoder GAN(cVAE-GAN) and Conditional Latent Regres-

sor GAN (cLR-GAN), each of which control a cyclic GAN model, hence the model is called bicycleGAN.

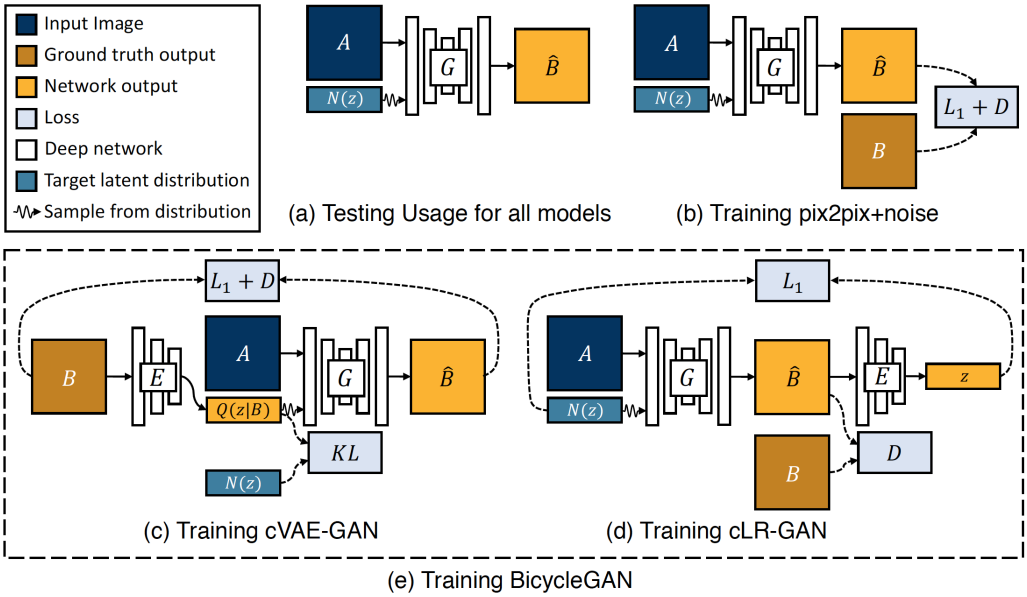


Figure 2.13: Structure of BicycleGAN [53]

cVAE-GAN

The ideal is to learn a low-dimensional representation of the target images. Firstly, as shown in Figure 2.13, it encodes the ground truth image (B) into the latent space through an encoder. This is to ensure that the latent code z is useful and meaningful. Then an input image (A) along with the encoded ground truth is fed into the generator to generate the output image ($\hat{B}$). Hence, cVAE-GAN controls the $B \rightarrow z \rightarrow \hat{B}$ flow, which aims to reconstruct the ground truth image. Extending the unconditional VAE to conditional scenario, the distribution $Q(z|B)$ of latent code z is replaced by the encoder E with Gaussian assumption. Here, reparameterization trick is used to allow backpropagation.

$$\mathcal{L}_{\text{GAN}}^{\text{VAE}} = \mathbb{E}_{\text{A,B}\sim p(\text{A,B})}[\log(D(\mathbf{A},\mathbf{B}))] + \mathbb{E}_{\mathbf{A},\mathbf{B}\sim p(\mathbf{A},\mathbf{B}),\mathbf{z}\sim E(\mathbf{B})}[\log(1 - D(\mathbf{A}, G(\mathbf{A},\mathbf{z})))]$$

Then L1 loss is added to encourage the output of the generator to match the ground truth.

$$\mathcal{L}_1^{\text{VAE}}(G) = \mathbb{E}_{\mathbf{A},\mathbf{B}\sim p(\mathbf{A},\mathbf{B}),\mathbf{z}\sim E(\mathbf{B})}\|\mathbf{B} - G(\mathbf{A},\mathbf{z})\|_1$$

Finally, KL-divergence is used to encourage the latent distribution encoded by $E(B)$ to be closed to a random Gaussian, which would make inference easy by sampling from Gaussian.

$$\mathcal{L}_{\mathrm{KL}}(E) = \mathbb{E}_{\mathbf{B}\sim p(\mathbf{B})}\left[\mathcal{D}_{\mathrm{KL}}(E(\mathbf{B})\|\mathcal{N}(0,I))\right]$$

Thus, the full objective of cVAE-GAN is

$$G^*, E^* = \arg\min_{G,E}\max_{D} \quad \mathcal{L}_{\mathrm{GAN}}^{\mathrm{VAE}}(G,D,E) + \lambda\mathcal{L}_1^{\mathrm{VAE}}(G,E) + \lambda_{\mathrm{KL}}\mathcal{L}_{\mathrm{KL}}(E)$$

cLR-GAN

In this part, we are exploring another method, cLR-GAN, to apply the latent code z. The input image (A) along with a randomly sampled latent vector z from Gaussian distribution is provided to the generator (G) to get the output image ($\hat{B}$). The generated image ($\hat{B}$) does not necessarily look like the ground truth image, but it should look realistic. This is controlled by the discriminator loss on $\hat{B}$. Then the generated image is fed into the encoder (E) to get $\hat{z}$, which attempt to recover the latent vector of the output image [5]. The recovered latent code should be closed to the Gaussian distribution.

$$\mathcal{L}_1^{\mathrm{latent}}(G,E) = \mathbb{E}_{\mathbf{A}\sim p(\mathbf{A}),\mathbf{z}\sim p(\mathbf{z})}\|\mathbf{z} - E(G(\mathbf{A},\mathbf{z}))\|_1$$

So the total objective can be written as:

$$G^*, E^* = \arg\min_{G,E}\max_{D} \quad \mathcal{L}_{\mathrm{GAN}}(G,D) + \lambda_{\mathrm{latent}}\,\mathcal{L}_1^{\mathrm{latent}}(G,E)$$

BicycleGAN

BicycleGAN is the combination of cVAE-GAN and cLR-GAN objectives in a hybrid model. In cVAE-GAN, although the encoder learns from ground truth data, the generated latent code is relatively random and may not yield a realistic image. Additionally, the discriminator does not learn from the results generated under the prior condition. In cLR-GAN, sampling from the prior is easy, but the generator is not trained with ground truth paired data. Hence, the combined model could take advantage of both the model and the final objective for BicycleGAN is

$$G^*, E^* = \arg\min_{G,E}\max_{D} \mathcal{L}_{\mathrm{GAN}}^{\mathrm{VAE}}(G,D,E) + \lambda\mathcal{L}_1^{\mathrm{VAE}}(G,E) + \mathcal{L}_{\mathrm{GAN}}(G,D) + \lambda_{\mathrm{latent}}\,\mathcal{L}_1^{\mathrm{latent}}(G,E) + \lambda_{\mathrm{KL}}\mathcal{L}_{\mathrm{KL}}(E)$$

where the hyperparameters control the importance of each term.

2.4 Unpaired Image-to-Image Translation

In the unsupervised learning way of translation, the goal is to construct a relation between two domains. The data could just be some independent sets of images, each of which corresponding to a different domain, so we do not need paired data to show how an image is translated into another domain. Unsupervised translations are more difficult than supervised problems because of lacking the corresponding information, but they are more applicable in real life since paired data is difficult and expensive to collect.

2.4.1 Domain Transfer Network

Domain Transfer Network (DTN) is an unsupervised method proposed by Taigman et al. in 2016 [48]. In DTN, we studied the problem of transferring a sample from one domain to an analog sample in another domain. We would like to learn a function that maps input from the source domain to the target domain, as well as maps input from the target domain to itself. This model consists of a generator and a discriminator, where the generator is comprised of an encoding function and a generating function.

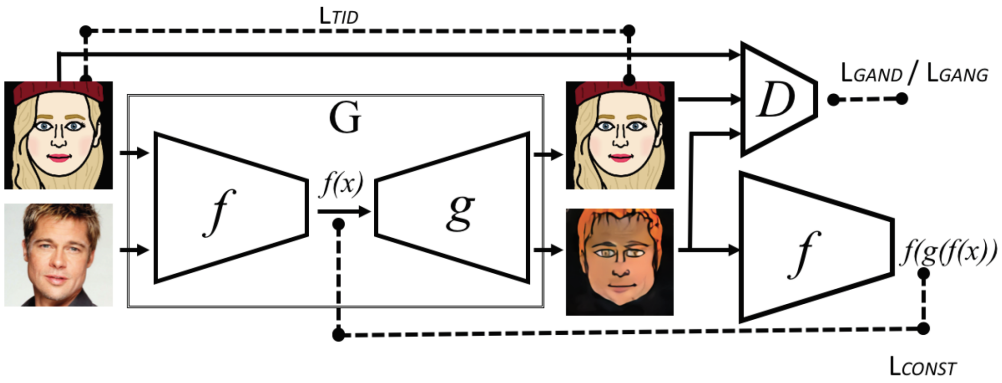


Figure 2.14: Domain Transfer Network [48]

As shown in Figure 2.14, the generator (G) has a similar structure with the encoder-decoder system. But in this model, the function f is pre-trained and only the function g is trainable. Firstly, an input image from the source domain is fed into the network. The function f encodes it into a low-dimensional latent space $f(x)$ and then the generating

function g output a image $G(x)$ on the target domain. The output image $G(x)$ is then encoded by f again, and we try to minimize the difference between the original code $f(x)$ and the reconstructed code $f(g(f(x)))$.

$$L_{\text{CONST}} = \sum_{x \in \mathbf{s}} d(f(x), f(g(f(x))))$$

The output of the generator G is also provided into the discriminator model to compute the GAN loss.

$$L_{\text{GANG}} = -\mathbb{E}_{x \in \text{s}} \log D_3(g(f(x))) - \mathbb{E}_{x \in \text{t}} \log D_3(g(f(x)))$$

$$L_{\text{D}} = -\mathbb{E}_{x \in \text{s}} \log D_1(g(f(x))) - \mathbb{E}_{x \in \text{t}} \log D_2(g(f(x))) - \mathbb{E}_{x \in \text{t}} \log D_3(x)$$

If the input image is from the target domain, we minimize the difference between the output from the generator $G(x)$ and the original input, since the generator should remain the image from target domain unchanged.

$$L_{\text{TID}} = \sum_{x \in \mathbf{t}} d_2(x, G(x))$$

where d_2 is a distance function in T and we choose to use MSE in this model. The output is also fed into the discriminator model. Generally, the function f is more accurate on the images from the source domain than the target domain, and the generator is trained on both domain. Finally, a total variation loss is added to slightly smooth the output image, which is introduced by Rudin et al. in 1992 [43, 30]. The total variation loss is defined as

$$L_{\text{TV}}(y) = \sum_{i,j} ((z_{i,j+1} - z_{i,j})^2 + (z_{i+1,j} - z_{i,j})^2)^{\frac{1}{2}}$$

where y is the output from the generator. Hence, the full objective of the generator is

$$L_{\text{G}} = L_{\text{GANG}} + \alpha L_{\text{CONST}} + \beta L_{\text{TID}} + \gamma L_{\text{TV}}$$

2.4.2 UNIT

Unsupervised Image-to-Image Translation Networks (UNIT) aims at learning the universal approximator function from the source domain to the target domain by using the images from the marginal distribution. This problem is constrained by assuming that there is a

shared latent space between these two domains and Liu, et al. proposed this model based on GANs and VAE in 2017 [28]. The whole network translates an input image by encoding it onto the latent space and then generate a new image in the target domain.

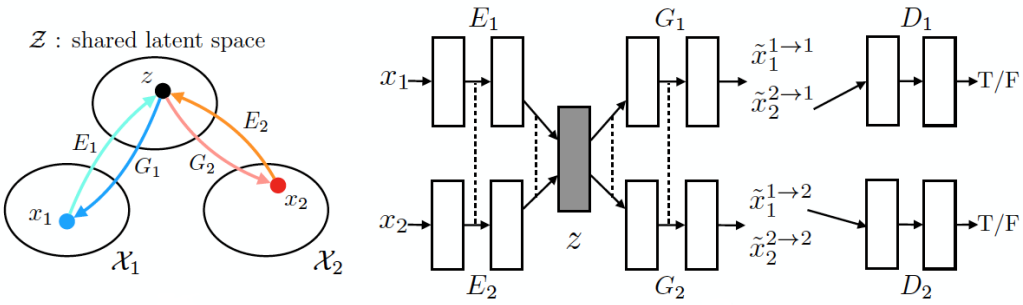


Figure 2.15: Shared latent space assumption and UNIT architecture [28]

Assumption

Let $\mathcal{X}_1$ and $\mathcal{X}_2$ be two image domains. As shown in Figure 2.15, we assume that for any given pair of images x_1 and x_2, there exist a shared latent code z in a shared latent space. So we can recover both images x_1 and x_2 from this latent code z, and we can also compute the latent code z from both images. Suppose there exist functions E_1, E_2, G_1 and G_2 such that for any given pair of images (x_1, x_2) from $\mathcal{X}_1 \times \mathcal{X}_2$, we have $z = E_1(x_1) = E_2(x_2)$ and $x_1 = G_1(z)$ and $x_2 = G_2(z)$. Then the composition of E_1 and G_2 could map an image from $\mathcal{X}_1$ to $\mathcal{X}_2$ as $x_2 = G_2(E_1(x_1))$. Similarly, E_2 and G_1 could map an image from $\mathcal{X}_2$ to $\mathcal{X}_1$ as $x_1 = G_1(E_2(x_2))$. Therefore, there exist a cycle consistency constraint: $x_1 = G_1(E_2(G_2(E_1(x_1))))$ and $x_2 = G_2(E_1(G_1(E_2(x_2))))$, which means the input image could be reconstructed by its translated output image [18]. The assumption of shared latent space implies this cycle consistency constraint, but not vice versa.

Framework

This UNIT network is based on VAEs and GANs that we have explicated in the previous section. It consistent six submodels: two encoders E_1 and E_2, two generators G_1 and G_2, and two discriminators D_1 and D_2. In this network, E_1 and G_1 act like a VAE, E_1 and G_2 are image translator from $\mathcal{X}_1$ to $\mathcal{X}_2$, and G_1 and D_1 work as a GAN.

VAE. The VAE model is comprised of an encoder-decoder pair for the same domain, such as $\{E_1, G_1\}$ is the VAE for domain $\mathcal{X}_1$, namely VAE_1. For any input image $x_1 \in \mathcal{X}_1$, the

VAE_1 maps it to the latent space by the encoder E_1 and then decode a random perturbed version of the latent vector z back to domain $\mathcal{X}_1$ by the generator G_1 exactly as in a standard VAE. We suppose the latent space follow a Gaussian distribution. Then the reconstructed image of x_1 is $x^{1\to1} = G_1(z_1 \sim \mathcal{N}(z_1|E_1(x_1), I))$. Similarly, $\{E_2, G_2\}$ constitutes a VAE for $\mathcal{X}_2$ and the reconstructed image of x_2 is $x^{2\to2} = G_1(z_2 \sim \mathcal{N}(z_2|E_2(x_2), I))$. Here we use the reparameterization trick so that we could use backpropagation when training the VAE [22].

Weight-sharing. In this model, the relation between the two VAEs could be construct by enforcing a weight sharing constraint. To be specific, we choose to share the weights in the last few layers in E_1 and E_2, which control the high-level representation of the input images when encoding. Similarly, we share the weights of the first few layers in G_1 and G_2, which decodes the high-level representation from the latent code to reconstruct the input images.

GANs. This network consist of two GANs, the pair $\{G_1, D_1\}$ constitutes GAN_1 and $\{G_2, D_2\}$ constitutes GAN_2. These GANs work exactly the same as the standard GAN we mentioned previously. In GAN_1, if the input is sampled from $\mathcal{X}_1$, then the output of D_1 should be 1, while the input is generated by G_1, the output should be 0. G_1 can generate reconstructed images from $\mathcal{X}_1$, say $x^{1\to1}$, and also can generate translated images from $\mathcal{X}_2$, say $x^{2\to1}$.

Learning. The UNIT network is trained as solving a combination of VAE and GAN for the problem of image reconstruction, image translation, and cycle consistency. When training the VAE, we want to minimize the variational upper bound. The objective for VAEs are

$$\mathcal{L}_{\text{VAE}_1}(E_1, G_1) = \lambda_1 \text{KL}\left(q_1(z_1|x_1)||\mathcal{N}(z|0, I)\right) - \lambda_2 \mathbb{E}_{z_1\sim q_1(z_1|x_1)}\left[\log p_{G_1}(x_1|z_1)\right]$$

$$\mathcal{L}_{\text{VAE}_2}(E_2, G_2) = \lambda_1 \text{KL}\left(q_2(z_2|x_2)||\mathcal{N}(z|0, I)\right) - \lambda_2 \mathbb{E}_{z_2\sim q_2(z_2|x_2)}\left[\log p_{G_2}(x_2|z_2)\right]$$

We model p_{G_1} and p_{G_2} using Laplacian distributions respectively. So minimizing the negative log-likelihood term is equivalent to minimizing the absolute distance between the original image and the reconstructed image.

The objective for GANs are

$$\mathcal{L}_{\text{GAN}_1}(E_2, G_1, D_1) = \lambda_0 \mathbb{E}_{x_1\sim P_{\mathcal{X}_1}}\left[\log D_1(x_1)\right] + \lambda_0 \mathbb{E}_{z_2\sim q_2(z_2|x_2)}\left[\log\left(1 - D_1(G_1(z_2))\right)\right]$$

$$\mathcal{L}_{\text{GAN}_2}(E_1, G_2, D_2) = \lambda_0 \mathbb{E}_{x_2\sim P_{\mathcal{X}_2}}\left[\log D_2(x_2)\right] + \lambda_0 \mathbb{E}_{z_1\sim q_1(z_1|x_1)}\left[\log\left(1 - D_2(G_2(z_1))\right)\right]$$

Finally, we use a VAE-like objective function to model the cycle consistency constraint

$$\mathcal{L}_{\mathrm{CC}_1}\left(E_1, G_1, E_2, G_2\right) = \lambda_3 \mathrm{KL}\left(q_2\left(z_2|x^{1\to 2}\right)\right)\|p_\eta(z)\big) - \lambda_4 \mathbb{E}_{z_2\sim q_2}\left(z_2|x^{1\to 2}\right)\left[\log p_{G_1}\left(x_1|z_2\right)\right]$$

$$\mathcal{L}_{\mathrm{CC}_2}\left(E_2, G_2, E_1, G_1\right) = \lambda_3 \mathrm{KL}\left(q_1\left(z_1|x^{2\to 1}\right)\right)\|p_\eta(z)\big) - \lambda_4 \mathbb{E}_{z_1\sim q_1}\left(z_1|x^{2\to 1}\right)\left[\log p_{G_2}\left(x_2|z_1\right)\right]$$

where the second term enforce the reconstruction and the KL-divergence try to match the data distribution. Hence, the total objective is

$$\min_{E_1,E_2,G_1,G_2}\max_{D_1,D_2} \mathcal{L}_{\mathrm{VAE}_1} + \mathcal{L}_{\mathrm{GAN}_1} + \mathcal{L}_{\mathrm{CC}_1} + \mathcal{L}_{\mathrm{VAE}_2} + \mathcal{L}_{\mathrm{GAN}_2} + \mathcal{L}_{\mathrm{CC}_2}$$

Translation. After the learning process, we could use the learned weight of E_1 and G_2 to translate images from $\mathcal{X}_1$ to $\mathcal{X}_2$.

Chapter 3

Model Design

In this chapter, we present mainly two different methods for image synthesis. The first one is called latent features transfer which is adapted from the neural style transfer model. In this method, the output images are not generated by a generative network but optimized through gradient descent. The second method uses deep generative models, extending the CycleGAN architecture. We will first describe the standard CycleGAN model, then introduce our novel ideas of improving this model.

3.1 Latent Features Transfer

Since our aim is to transfer images between two domains of photos and paintings, this could be treated as a style transfer problem. In addition, we assume that two different domain share the same latent space in the UNIT network [28], but in this model, we suppose the latent features have distinct distribution for each domain and we want to find the relation between them. Latent Features Transfer is a supervised model adapted from the neural style transfer network, which requires paired data for the training process. Similar to neural style transfer, image representation is also an important part of the latent feature transfer network, where we also need to extract the content features and the style features from images.

This model aims at extracting the common content and style information of each domain, then find a mapping between these features of each domain. We use the feature space of the output from the same layers in VGG19 as used in neural style transfer for the content representation and the style representation. The different convolutional layer has a distinct purpose in the network, which could be found by deconvolutions and filtering the maximal

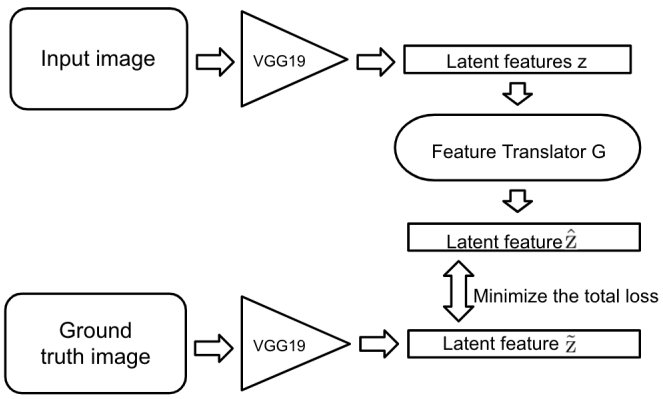


Figure 3.1: Training the Feature Translator G

activations [52]. To be specific, we use conv3_2 layer for the content feature, and conv1_1, conv2_1, conv3_1, conv4_1 and conv5_1 layers for the style feature in Figure 2.3, and we call these latent features z. As shown in Figure 3.1, we want to map these features z so that the mapped features $\hat{z}$ are similar to the features extracted from the other domain. We train a convolutional neural network translator G to translate these features. When training the translator, we match the content information and the style information respectively by minimizing the style loss and content loss between the $\hat{z}$ and the features $\tilde{z}$ extracted directly from the ground truth of the target domain, where the style loss and the content loss are mentioned in the previous section.

$$\mathcal{L}_{\text{total}} = \alpha\mathcal{L}_{\text{style}} + \beta\mathcal{L}_{\text{content}}$$

and

$$\mathcal{L}_{\text{style}} = \alpha_1\mathcal{L}_{\text{conv1\_1}} + \alpha_2\mathcal{L}_{\text{conv2\_1}} + \alpha_3\mathcal{L}_{\text{conv3\_1}} + \alpha_4\mathcal{L}_{\text{conv4\_1}} + \alpha_5\mathcal{L}_{\text{conv5\_1}}$$

where α, β, α_1, α_2, α_3, α_4, α_5 are hyperparameters controlling the relative weight between the output from each layer.

The process of generating new images is shown in Figure 3.2, we firstly extract the latent features z of an input image using the VGG19 network. And then these features are fed into the pre-trained translator G to get the translated features $\hat{z}$ for the target domain. Finally, we initialize the generated image using the input image, process gradient descent on the total loss between the features of generated image $\tilde{z}$ and the translated features $\hat{z}$ with respect to the generated image, and backpropagation through the VGG19 network.

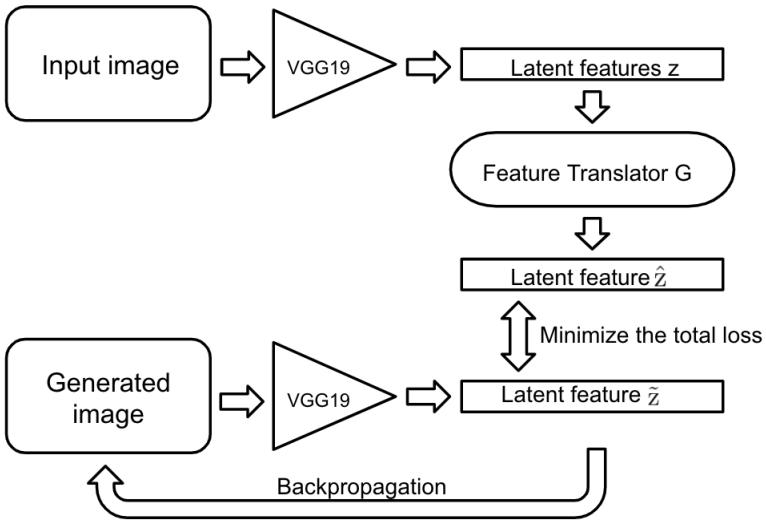


Figure 3.2: Generating new images

3.2 Deep Generative Models

In this section, we would introduce three methods using deep generative model. All of the methods are adapted from the standard CycleGAN model [54]. First, we would give a brief explanation of the standard CycleGAN model, which is our baseline model. Then we propose three architectures: the first one combining the CycleGAN with VAE, the second one combining the CycleGAN with neural style transfer, and the last one change the architecture of discriminator.

3.2.1 CycleGAN

The CycleGAN is another technique that involves the automatic training of image-to-image translation models without paired examples. This architecture is inspired by the Pix2Pix model [17]. The models are trained in an unsupervised manner using a collection of images from the source and target domain that do not need to be related in any way. This simple technique is powerful, achieving visually impressive results on a range of application domains, most notably translating photographs of horses to zebra, and the reverse.

The CycleGAN is an extension of the GAN architecture that involves the simultaneous training of two generator models and two discriminator models [54]. One generator (G) takes images from the first domain (X) as input and outputs images for the second domain (Y), and the other generator (F) takes images from the second domain (Y) as input and

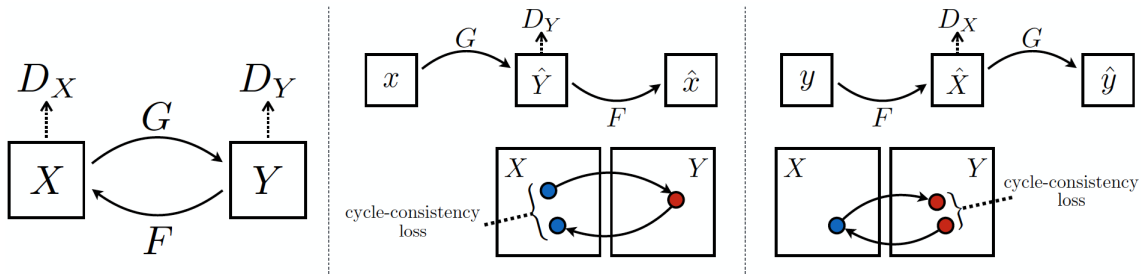


Figure 3.3: CycleGAN [54]

generates images for the first domain (X). Discriminator models are then used to determine how plausible the generated images are and update the generator models accordingly. We need more restriction to not only generate plausible images in each domain but also generate translations of the input images. The CycleGAN uses an additional extension to the architecture called cycle consistency. This is the idea that an image output from the first generator is fed into the second generator to reconstruct the original image. The reverse is also true: that an output from the second generator can be fed as input to the first generator and the result should match the input to the second generator. Cycle consistency is a concept from machine translation where a phrase translated from English to French should translate from French back to English and be identical to the original phrase and the reverse process should also be true. The CycleGAN encourages cycle consistency by adding an additional loss to measure the difference between the reconstructed input and the original image. This acts as a regularization of the generator models, guiding the image generation process in the new domain toward image translation.

We will develop an architecture of composition of two GANs, and each GAN has a discriminator and a generator model. Hence, there are four models in total in the architecture. Each GAN has a conditional generator model that will synthesize an image given an input image. And each GAN has a discriminator model to predict how likely the generated image is to have come from the target image collection. The discriminator and generator models for a GAN are trained under normal adversarial loss like a standard GAN model.

$$\mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) = \mathbb{E}_{y \sim p_{data}(y)}[\log D_Y(y)] + \mathbb{E}_{x \sim p_{data}(x)}[\log(1 - D_Y(G(x)))]$$

Each of the GANs are also updated using cycle consistency loss. The learned mapping function should be cycle-consistent.

$$x \to G(x) \to F(G(x)) \approx x$$

is called the forward cycle consistency. And

$$y \to F(y) \to G(F(y)) \approx y$$

is called the backward cycle consistency. This is designed to encourage the synthesized images in the target domain to be the translations of the input image. Cycle consistency loss compares an input photo to the CycleGAN to the generated photo and calculates the pixel difference between them.

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

Besides, it is found that introducing a identity loss between the input image and the generated image could help the generators to preserve the colour. There might be a mapping which is equally valid under the adversarial loss [54] and cycle consistency loss, but the colour of the generated images is not appropriate for the target domain.

$$\mathcal{L}_{\text{identity}}(G, F) = \mathbb{E}_{y \sim p_{data}(y)}[\|F(y) - y\|_1] + \mathbb{E}_{x \sim p_{data}(x)}[\|G(x) - x\|_1]$$

Then the full objective is

$$\mathcal{L}(G, F, D_X, D_Y) = \mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) + \mathcal{L}_{\text{GAN}}(F, D_X, Y, X) + \lambda \mathcal{L}_{cyc}(G, F) + \gamma \mathcal{L}_{\text{identity}}(G, F)$$

We aim to solve the problem that

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

Then when generating new image for an input, G^* and F^* are the translators from domain X to Y and from domain Y to X respectively.

3.2.2 CycleGAN+VAE

This model is inspired by the VAE part in BicycleGAN. In this model, we combine the CycleGAN model and the VAE model we mentioned before. Firstly, we train a VAE model which could compress the images from both domain to a lower dimensional feature space and then reconstruct them. Then we construct a CycleGAN between the feature spaces of two domains. There is an encoder, a decoder, two generators and two discriminators in the model.

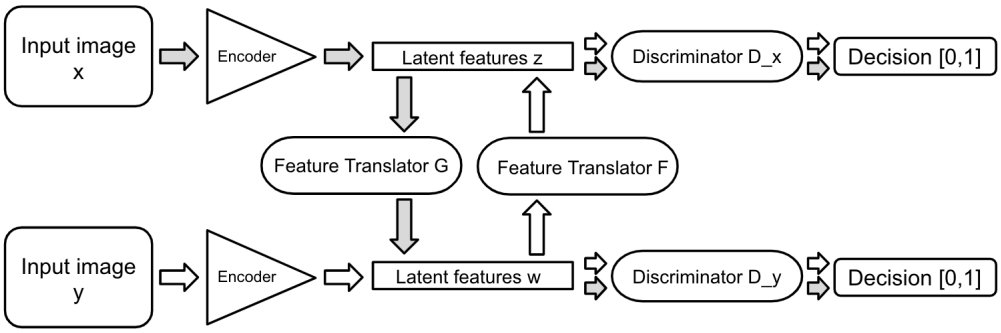


Figure 3.4: Training CycleGAN+VAE model

For an input image x in domain X, it is firstly encoded to the latent feature space z by the pre-trained encoder, then the latent feature z is fed into the generator G and output a translated latent feature $\tilde{w}$. The discriminator D_Y takes w and $\tilde{w}$ as its input and outputs a decision indicating the probability of them to be true. At the same time, an image y from the domain Y is encoded to w and translated to $\tilde{z}$ by translator F. $\tilde{z}$ is also provided to discriminator D_X to update both generator and discriminator. Same as the standard CycleGAN model, the generator G and F aim at fooling the discriminator D_X and D_X respectively, and the discriminators try to distinguish whether the inputs are from the corresponding domain. There is also a cycle consistency in this model. For a set of latent features z, we map it through G into another feature space w, and we want to reconstruct it using F. Similar constraint is required for the other cycle. We minimize the L1 loss between the original latent features and the reconstructed latent features. After the training process, we could generate new images by feeding the translated features into the decoder.

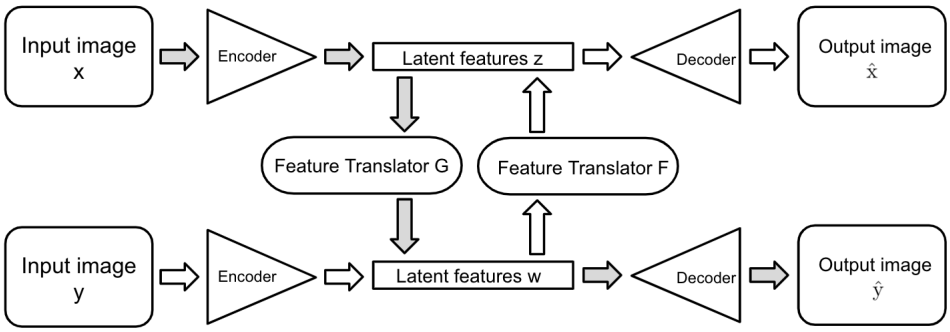


Figure 3.5: Testing CycleGAN+VAE model

3.2.3 CycleGAN+NST

According to the problem setting, in the Cycle GAN model, the generated image not only need to be closed to the other domain, i.e. indistinguishable by the discriminator, but also have the similar content as the input image. We could add more constraint to encourage this. Suppose the generator G takes an input image x from the domain X and it is expected to output a image y on the domain Y. The generated image y should have the same content information as x and the similar style features as images from domain Y. As stated in previous section, we choose L1 loss for an identity loss, which means that we minimize the exact pixel difference between the original input image and the generated image. This could help to preserve colour for the mapping from the input images to the output images, but sometime we do not need the colour to be exactly the same. The L1 loss might be too restrict to limit the possibility that the generator could generate more variant outputs. In the model, we do not match the exact pixel values of the input images and the generated images when computing the identity loss. Instead, we match the content information of the input images and the generated images and the style information of the generated images and a random image from the target domain. This method is inspired by the comparison of latent features in Domain Transfer network. To compute the loss, we use the VGG19 network to extract the content information from both input image and generated image and calculate a content loss, and extract the style features of the generated image and a random image from the target domain and calculate the style loss between them. We choose the conv3_2 layer for the content feature, and conv1_1, conv2_1, conv3_1, conv4_1 and conv5_1 layers for the style feature in Figure 2.3. When training the generator, we

minimize the GAN loss, cycle consistency loss as well as the content loss and style loss. The training process of discriminator remains the same.

3.2.4 PackingGANs

GANs models are sometimes suffering from model collapse. PackingGAN is an approach to handle model collapse which was proposed by Lin et al. in 2018 [26]. The main idea of this method is to change the discriminator to improve the accuracy if multiple samples are given. The packed inputs to the discriminator are from the same class. Instead of using the discriminator $D(x)$ which maps a single input to a probability between 0 and 1, we use an augmented discriminator $D(x_1, x_2, \dots)$ that maps m inputs jointly to a probability. The inputs of discriminator are independently sampled from the same data distribution, either real images or generated images. We do not change the architecture of hidden layers or output layer, but only increase the number of channels in the input layers by a factor of m. In this CycleGAN model, we choose m to be 2. Then we concatenate two inputs along the last dimension and feed it into the discriminator. The training process of the packed discriminator is the same as the standard discriminator, but each updating requires two samples. This means the discriminator would receive either two images from the real data set or two generated images.

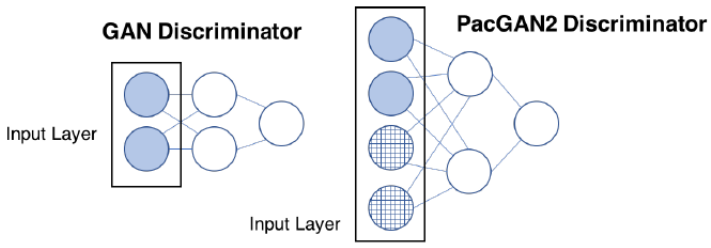


Figure 3.6: PackingGAN discriminator with m=2 [26]

Chapter 4

Experiments and Results

In this chapter, we evaluate the results of the methods in the previous chapter. It starts by looking at the data set we used in these experiments. Then we would explain the architecture and training details. Finally, we would present the evaluation of the results.

4.1 Data sets

For face image translation between photo and painting, we use CelebA-HQ dataset as the photo dataset. We generate a corresponding portrait painting for each photo using an online service (aiportraits.com), so we have paired data for every domain.

CelebA-HQ. This is a publicly available dataset which contains 30,000 high-quality face images of celebrities from CelebA [50]. We resize the initial images to 64×64 for a computational reason. This dataset is the photo domain for both models.

AIPortrait. We collect 300 images from the online service. It takes the images from CelebA-HQ as its inputs and outputs a corresponding painting. These output images are generated by a fully automatic process including face alignment. The style of the generated images is different, but most of them are in the Renaissance style. We also re-scale these images to 64×64. This dataset is prepared fro the method using latent features transfer.

Painting. The art images are downloaded from Wikiart.org. Most of the artworks are impressionistic. The images are resized to 64×64. The dataset contains 1000 paintings in total, and it is randomly split into 900 training set and 100 test set. This dataset is for the CycleGAN related methods which do not require paired images for the training process.

AIPortrait and Painting datasets are available at GitHub.

4.2 Implementation

4.2.1 Latent Features Transfer

The outputs of chosen layers in VGG19 are the features we extracted from an image. When generating features from VGG19 network, we use average pooling instead of max pooling, since it yields slightly more appealing results [33]. For every feature, we transform them using the same network. It contains three convolutional layers with no stride, each convolutional layer is followed by an instance normalization layer [49]. A leaky ReLU layer is after the first two normalization layers. We set the weights between each style layers are $0.5, 1.0, 1.5, 3.0, 4.0$ for layers conv1_1, conv2_1, conv3_1, conv4_1 and conv5_1 respectively. And the importance ratio between content information and style information is 10^5. The network is trained with a learning rate of 2 for the first 500 epochs and linearly decay to 0 over the last 100 epochs.

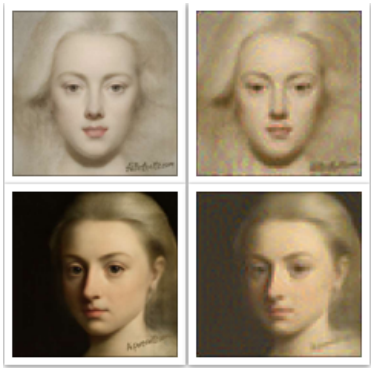
4.2.2 CycleGAN

Some of the generated images would have checkerboard pattern of artifacts. This is usually caused by the uneven overlap of deconvolution. The architecture of the generative network is adopted from the standard generator of Zhu et al. [54]. The only thing we change in this network is the way of upsampling. We do not use deconvolution for increasing the resolution, instead we use resize convolution, which is introduced by Odena et al. [35] in 2016. Firstly, we resize the input image using nearest-neighbor interpolation or bilinear interpolation to the size of twice of itself. Then it is fed into a convolutional layer with no stride. In GANs models, nearest-neighbor resize convolution could dramatically decrease the frequency of checkerboard artifacts [1]. Besides, zero paddings are replaced by reflection padding [27], since reflection padding could reflect the image around borders instead of just having black pixels. For the discriminator network, we still use 70×70 PatchGANs, where the discriminator decide whether a patch of 70×70 size of the input image is real or fake. This means our discriminator has 4 convolutional layers with stride size 2 and window size 4, and convolution to produce a 1-dimensional output at the end. When training the discriminator, we could reduce the model oscillation using the Shrivastava et al.s strategy [46]. We update the discriminators using a history of generated images rather than the latest generated ones. We keep an image buffer that stores the 50 previously created images and randomly sample from it to update the discriminators [54]. Moreover, we

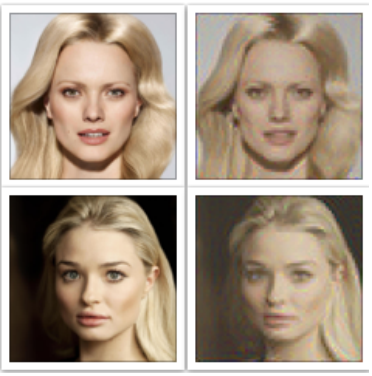
add some white noise to the data to avoid overfitting when training the discriminator. similar to Domain Transfer Network, we add a total variation loss to make the generated images slightly smoother [43, 30]. Finally, the objective function of GANs is changed to a least-squares loss, which is more stable and yields higher quality results.

4.3 Results and Discussion

In this section, we will present the evaluation of the above two methods. Our baseline model is the standard CycleGAN model. For latent feature translation, we will discuss from three aspects, including the trade-off between content and style matching, using different layers of the VGG19 Network, initialization of gradient descent, and architecture of translator. For adapted CycleGAN model, we will evaluate from adversarial accuracy and reconstruction error. Translation from photo to portrait has better results in every setting, which could be seen by training loss (Figure A.6, A.7, A.8).
Firstly, we need to make sure that matching latent features could reconstruct visually appealing results. We minimize the style and content loss between the input image and the generated image with initialization from the mean image of the corresponding domain.



(a) Portrait reconstruction.



(b) Photo reconstruction.

Figure 4.1: The left column show the original image and the right column show the reconstructed image.

4.3.1 Trade-off between content and style matching

Content information and style information of an image cannot be separated completely. When training the translator and synthesizing new images, these images do not match both content and style information both perfectly and simultaneously. We need to find the importance relation between them. The loss function we used is a linear combination of the content loss and style loss and we train the model by minimizing the total loss function. The hyperparameters α and β control the importance ratio between content and style. We could smoothly regulate the emphasis on content and style by increasing or decreasing the hyperparameters [13]. We decide to use $\alpha/\beta = 10^{-5}$ between style and content to create visually more artistic or realistic images. Additional results are shown in appendix A.1.

(a) Portrait to photo.

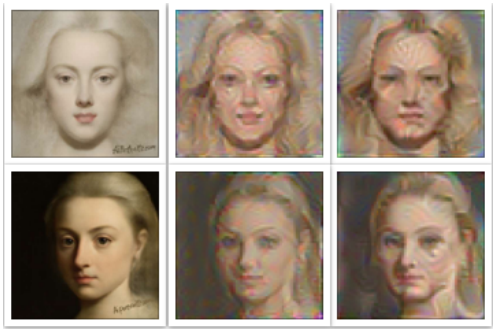
(b) Photo to portrait.

Figure 4.2: The left column show the original image, the middle column $\alpha/\beta = 10^{-5}$ and the right column $\alpha/\beta = 10^{-3}$.

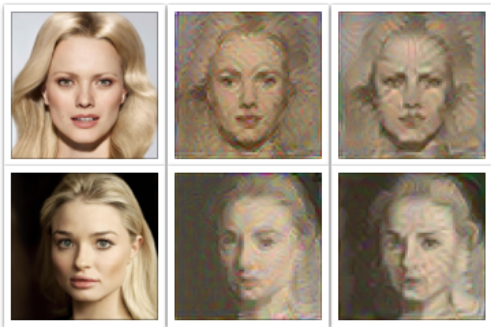
4.3.2 Using different layers of the VGG19 Network

The choice of layers we used to represent the content and style information is another important factor. We used the multi-scale representation of style features described exactly by Gatys et al. [13], which are layers conv1_1, conv2_1, conv3_1, conv4_1 and conv5_1 in the VGG19 network. The number and position of these layers determine the local scale on style features which control different visual experiences. Matching higher layers preserve local image structures and generate smoother images. To analyze the effect of the content layer, we keep the style layers and match conv3_2 and conv4_2 layers for content information respectively. We found that when matching higher layer in the network the generated images would have more distortion. To synthesis visually photorealistic images, we need to control the deformation of generated faces. So we use conv3_2 instead of conv4_2 in

VGG19 network to represent content feature (Figure 4.3). Additional results are shown in appendix A.2.



(a) Portrait to photo.

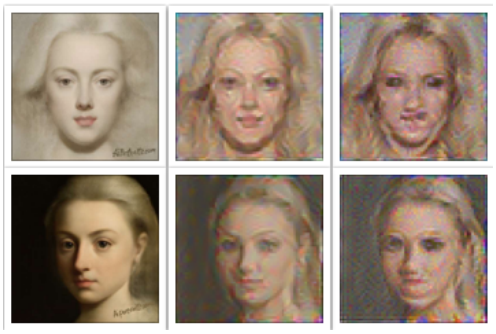


(b) Photo to portrait.

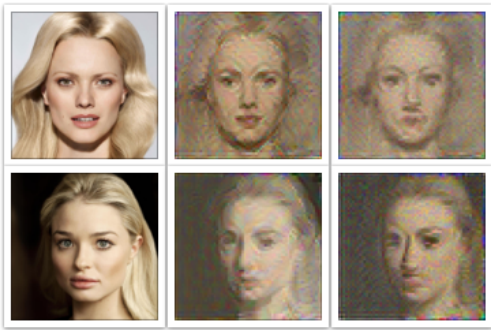
Figure 4.3: The left column show the original image, the middle column matching conv3_2 and the right column matching conv4_2.

4.3.3 Initialisation of gradient descent

If we transfer an artwork image to a photo image, we initialize the generated image with the artwork image with noise and vice versa. We could also initialize the image with white noise only. The results initialized from the corresponding artwork or photo images are much better than from white noise (Figure 4.4). The transfer from portrait to photo initialized from the mean image of the domain perform even slightly better in our case (Figure 4.5), but it strongly depends on the dataset.

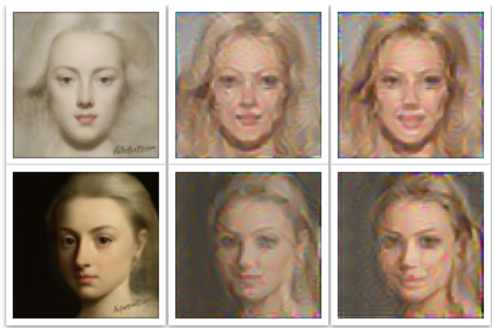


(a) Portrait to photo.

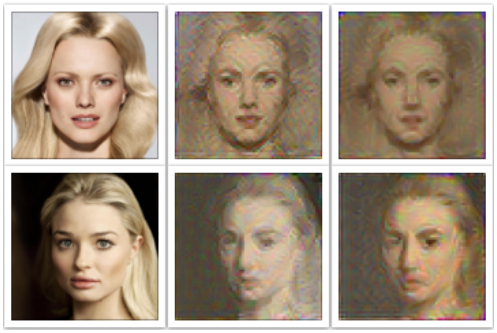


(b) Photo to portrait.

Figure 4.4: The left column show the original image, the middle column initializing from corresponding artwork or photo image and the right column initializing from white noise.



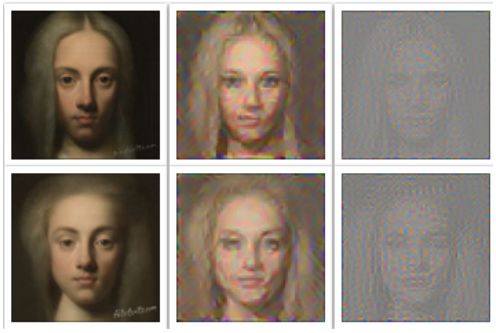
(a) Portrait to photo.



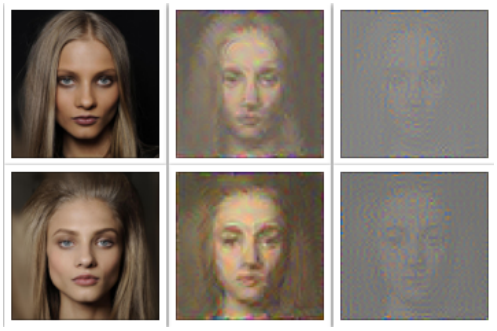
(b) Photo to portrait.

Figure 4.5: The left column show the original image, the middle column initializing from corresponding artwork or photo image and the right column initializing from mean image.

Additionally, we found an interesting phenomenon that the synthesized images are most appealing when we apply gradient descent with a learning rate of 2 for only 100 steps. If we update the synthesized images more, say 1000, the resulting image will be grayish. (Figure 4.6)



(a) Portrait to photo.

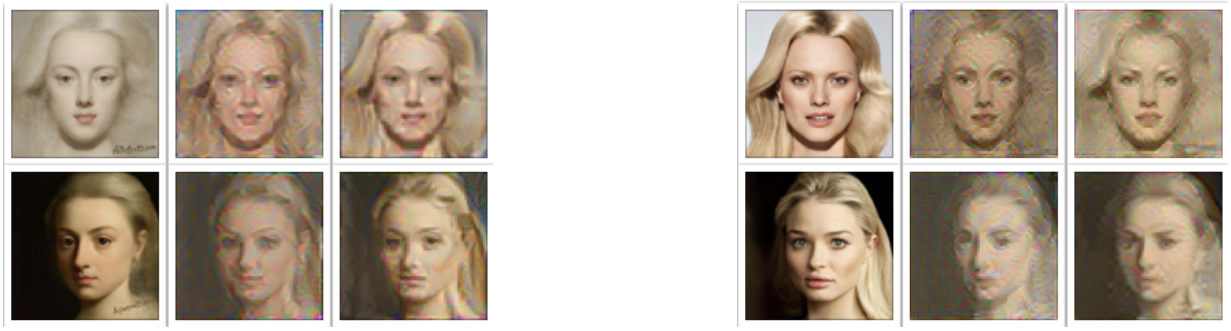


(b) Photo to portrait.

Figure 4.6: The left column show the original image, the middle column applying gradient descent for 100 steps and the right column applying for 1000 steps.

4.3.4 Dense layer and convolutional layer

The architecture of the translator is also an essential component of this model. The structure is described in the previous section, and we could use fully-connected layers and convolutional layers in this model. The results from convolutional layers architecture are slightly better than fully-connected layers (Figure 4.7), but it also needs much more computational resources.



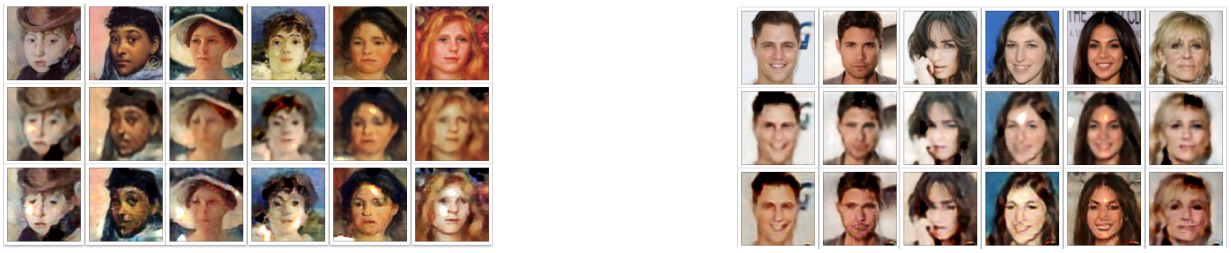
(a) Portrait to photo.

(b) Photo to portrait.

Figure 4.7: The left column show the original image, the middle column using dense layers and the right column using convolutional layers.

4.3.5 CycleGAN

CycleGAN is our baseline model, which is good at colour changing but does not perform well at shape altering. Although it has been seen success on translation between artwork and photo on landscape dataset, it obviously requires more on face dataset when translating. The results of reconstruction of input images are usually visually appealing, but the translation process suffers from model collapse.



(a) Portrait to photo.

(b) Photo to portrait.

Figure 4.8: The top row show the original images, the middle row show the reconstructed images and the bottom row show the translated images.

Chapter 5

Conclusion

In this thesis, we mainly focused on supervised learning method of image-to-image translation. Our objective was to translate an artwork portrait painting to a photorealistic face image and vice versa. We firstly explored the previous study on this topic in a comprehensive literature review, including background knowledge and successfully used models. Then we presented our work, which was predominantly adapted from neural style transfer. Ideally, we could separate the style and content features, and map these features from an image of a certain domain to another. After that, we analyzed the baseline model and our results from four different aspects. The weighting factors chosen in neural style transfer did not suit our problem setting, so we changed the ratio between style and content from 10^3 to 10^{-3} to get more appealing synthesized images. In addition, we altered the position of a layer representing the content information to a lower level layer which led to a more stable shape of faces. Furthermore, we examined different initialization methods and found initializing from the mean image would result in better quality images. Convolutional layer outperforms fully-connected layer in generator architecture but with much more computational demands. Further study could emphasize on unsupervised learning methods since training without paired dataset are more applicable in real life.

Bibliography

[1] Aitken, A., Ledig, C., Theis, L., Caballero, J., Wang, Z. and Shi, W., 2017. Checkerboard artifact free sub-pixel convolution: A note on sub-pixel convolution, resize convolution and convolution resize. *arXiv preprint arXiv:1707.02937*.

[2] Altosaar, J., Tutorial - What is a variational autoencoder? [online] Available from: https://jaan.io/what-is-variational-autoencoder-vae-tutorial/ [Accessed 3 September 2017]

[3] Arjovsky, M., Chintala, S. and Bottou, L., 2017. Wasserstein gan. *arXiv preprint arXiv:1701.07875*.

[4] Bengio, Y., Courville, A. and Vincent, P., 2013. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence, 35*(8), pp.1798-1828.

[5] Chen, X., Duan, Y., Houthooft, R., Schulman, J., Sutskever, I. and Abbeel, P., 2016. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in neural information processing systems* (pp. 2172-2180).

[6] ChengBinJin. 2018. DCGAN TensorFlow Implementation [online] Available from: https://github.com/ChengBinJin/DCGAN-TensorFlow [Accessed 3 September 2017]

[7] Dormehl, L., 2019. What is an artificial neural network? [online] Available from: https://www.digitaltrends.com/cool-tech/what-is-an-artificial-neural-network/ [Accessed 3 September 2017]

[8] Dosovitskiy, A., Tobias Springenberg, J. and Brox, T., 2015. Learning to generate chairs with convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 1538-1546).

[9] Frans, K., 2016. Variational Autoencoders Explained [online] Available from: http://kvfrans.com/variational-autoencoders-explained/ [Accessed 3 September 2017]

[10] Fukushima, K., 1988. Neocognitron: A hierarchical neural network capable of visual pattern recognition. *Neural networks, 1*(2), pp.119-130.

[11] Gardner, M.W. and Dorling, S.R., 1998. Artificial neural networks (the multilayer perceptron)a review of applications in the atmospheric sciences. *Atmospheric environment, 32*(14-15), pp.2627-2636.

[12] Gatys, L., Ecker, A.S. and Bethge, M., 2015. Texture synthesis using convolutional neural networks. In *Advances in neural information processing systems* (pp. 262-270).

[13] Gatys, L.A., Ecker, A.S. and Bethge, M., 2016. Image style transfer using convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 2414-2423).

[14] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. and Bengio, Y., 2014. Generative adversarial nets. In *Advances in neural information processing systems* (pp. 2672-2680).

[15] Hoffman, M.D., Blei, D.M., Wang, C. and Paisley, J., 2013. Stochastic variational inference. *The Journal of Machine Learning Research, 14*(1), pp.1303-1347.

[16] Ioffe, S. and Szegedy, C., 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167.*

[17] Isola, P., Zhu, J.Y., Zhou, T. and Efros, A.A., 2017. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1125-1134).

[18] Kim, T., Cha, M., Kim, H., Lee, J.K. and Kim, J., 2017, August. Learning to discover cross-domain relations with generative adversarial networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70* (pp. 1857-1865). JMLR. org.

[19] Kim, T., 2017. Image to Image Translation: Pix2Pix, CycleGAN, DiscoGAN [online] Available from: https://taeoh-kim.github.io/blog/gan%EC%9D%84-%EC%9D%B4%EC%9A%A9%ED%95%9C-image-to-image-translation-pix2pix-cyclegan-discogan/ [Accessed 3 September 2017]

[20] Kingma, D.P. and Welling, M., 2013. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114.*

[21] Kingma, D.P. and Ba, J., 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980.*

[22] Kingma, D., Max Welling. 2015. Variational Auto-Encoders and Extensions [online]. Available from: http://dpkingma.com/wordpress/wp-content/uploads/2015/12/talk_nips_workshop_2015.pdf [Accessed 3 September 2017]

[23] Larsen, A.B.L., Snderby, S.K., Larochelle, H. and Winther, O., 2015. Autoencoding beyond pixels using a learned similarity metric. *arXiv preprint arXiv:1512.09300.*

[24] LeCun, Y., Haffner, P., Bottou, L. and Bengio, Y., 1999. Object recognition with gradient-based learning. *Shape, contour and grouping in computer vision* (pp. 319-345). Springer, Berlin, Heidelberg.

[25] Li, C. and Wand, M., 2016, October. Precomputed real-time texture synthesis with markovian generative adversarial networks. In *European Conference on Computer Vision* (pp. 702-716). Springer, Cham.

[26] Lin, Z., Khetan, A., Fanti, G. and Oh, S., 2018. Pacgan: The power of two samples in generative adversarial networks. In *Advances in Neural Information Processing Systems* (pp. 1498-1507).

[27] Liu, G., Shih, K.J., Wang, T.C., Reda, F.A., Sapra, K., Yu, Z., Tao, A. and Catanzaro, B., 2018. Partial Convolution based Padding. *arXiv preprint arXiv:1811.11718*.

[28] Liu, M.Y., Breuel, T. and Kautz, J., 2017. Unsupervised image-to-image translation networks. In *Advances in neural information processing systems* (pp. 700-708).

[29] Maas, A.L., Hannun, A.Y. and Ng, A.Y., 2013, June. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml* (Vol. 30, No. 1, p. 3).

[30] Mahendran, A. and Vedaldi, A., 2015. Understanding deep image representations by inverting them. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 5188-5196).

[31] Mirza, M. and Osindero, S., 2014. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*.

[32] Moon, T.K., 1996. The expectation-maximization algorithm. *IEEE Signal processing magazine, 13*(6), pp.47-60.

[33] Mordvintsev, A., Olah, C., and Tyka, M., 2015. Inceptionism: Going Deeper into Neural Networks [online] Available from: https://ai.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html [Accessed 3 September 2017]

[34] Nair, V. and Hinton, G.E., 2010. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)* (pp. 807-814).

[35] Odena, A., Dumoulin, V., and Olah, C., 2016. Deconvolution and Checkerboard Artifacts [online] Available from: https://distill.pub/2016/deconv-checkerboard/ [Accessed 3 September 2017]

[36] Pappu, S., 2018. Style Transfer using Pytorch [online]. Available from: https://medium.com/@sashankpappu/style-transfer-using-pytorch-cb6225cf183e [Accessed 3 September 2017]

[37] Pathak, D., Krahenbuhl, P., Donahue, J., Darrell, T. and Efros, A.A., 2016. Context encoders: Feature learning by inpainting. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 2536-2544).

[38] Radford, A., Metz, L. and Chintala, S., 2015. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434.*

[39] Rezende, D.J., Mohamed, S. and Wierstra, D., 2014. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082.*

[40] Ronneberger, O., Fischer, P. and Brox, T., 2015, October. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention* (pp. 234-241). Springer, Cham.

[41] Rosenblatt, F., 1958. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review, 65*(6), p.386.

[42] Roweis, S.T., 1998. EM algorithms for PCA and SPCA. In *Advances in neural information processing systems* (pp. 626-632).

[43] Rudin, L.I., Osher, S. and Fatemi, E., 1992. Nonlinear total variation based noise removal algorithms. *Physica D: nonlinear phenomena, 60*(1-4), pp.259-268.

[44] Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A. and Chen, X., 2016. Improved techniques for training gans. In *Advances in neural information processing systems* (pp. 2234-2242).

[45] Simonyan, K. and Zisserman, A., 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556.*

[46] Shrivastava, A., Pfister, T., Tuzel, O., Susskind, J., Wang, W. and Webb, R., 2017. Learning from simulated and unsupervised images through adversarial training. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 2107-2116).

[47] Sudhir, K., 2017. Generative Adversarial Networks- History and Overview [online] Available from: https://towardsdatascience.com/generative-adversarial-networks-history-and-overview-7effbb713545 [Accessed 3 September 2017]

[48] Taigman, Y., Polyak, A. and Wolf, L., 2016. Unsupervised cross-domain image generation. *arXiv preprint arXiv:1611.02200*.

[49] Ulyanov, D., Vedaldi, A. and Lempitsky, V., 2016. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022.*

[50] willylulu, 2018. celeba-hq-modified [online] Available from: https://github.com/willylulu/celeba-hq-modified [Accessed 3 September 2017]

[51] Xu, B., Wang, N., Chen, T. and Li, M., 2015. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853.*

[52] Zeiler, M.D. and Fergus, R., 2014, September. Visualizing and understanding convolutional networks. In *European conference on computer vision* (pp. 818-833). Springer, Cham.

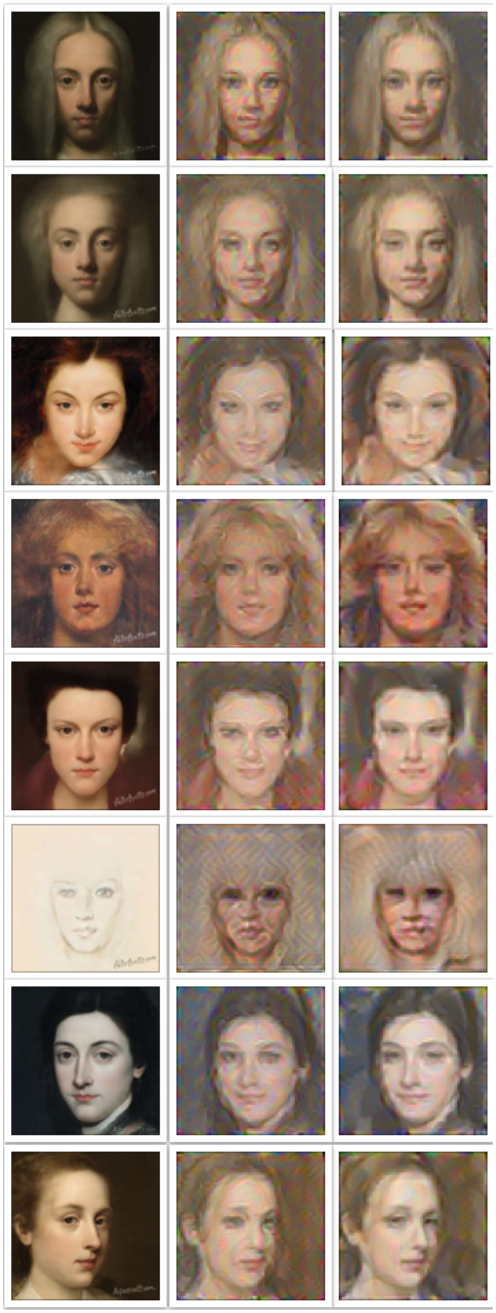
[53] Zhu, J.Y., Zhang, R., Pathak, D., Darrell, T., Efros, A.A., Wang, O. and Shechtman, E., 2017. Toward multimodal image-to-image translation. In *Advances in Neural Information Processing Systems* (pp. 465-476).

[54] Zhu, J.Y., Park, T., Isola, P. and Efros, A.A., 2017. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision* (pp. 2223-2232).

Appendix A

Appendix

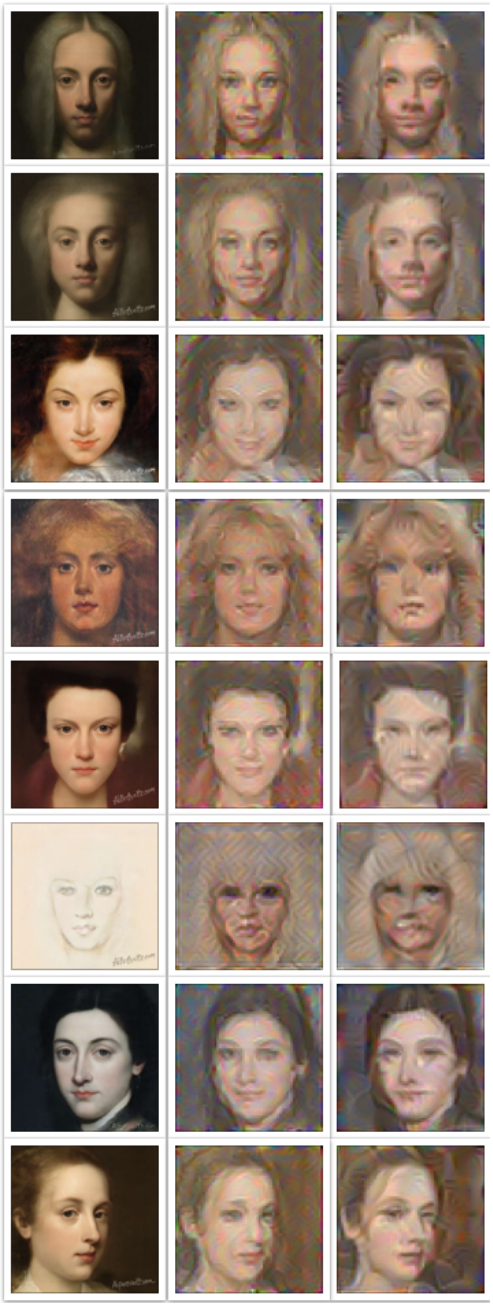
A.1 Additional Qualitative Results



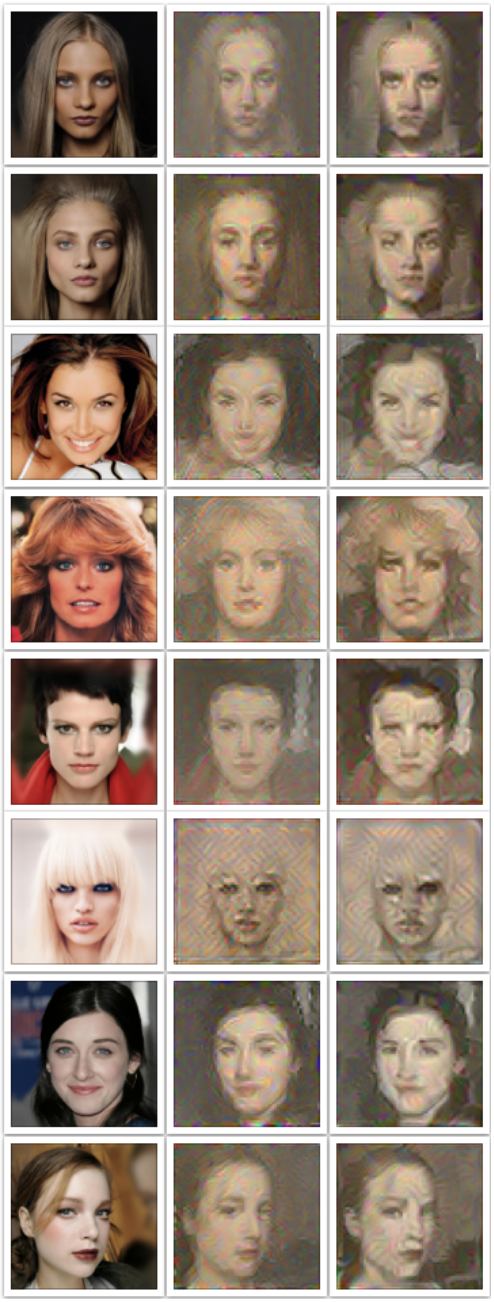
(a) Portrait to photo.

(b) Photo to portrait.

Figure A.1: More results for comparing content and style importance ratio. The left column show the original image, the middle column $\alpha/\beta = 10^{-5}$ and the right column $\alpha/\beta = 10^{-3}$.

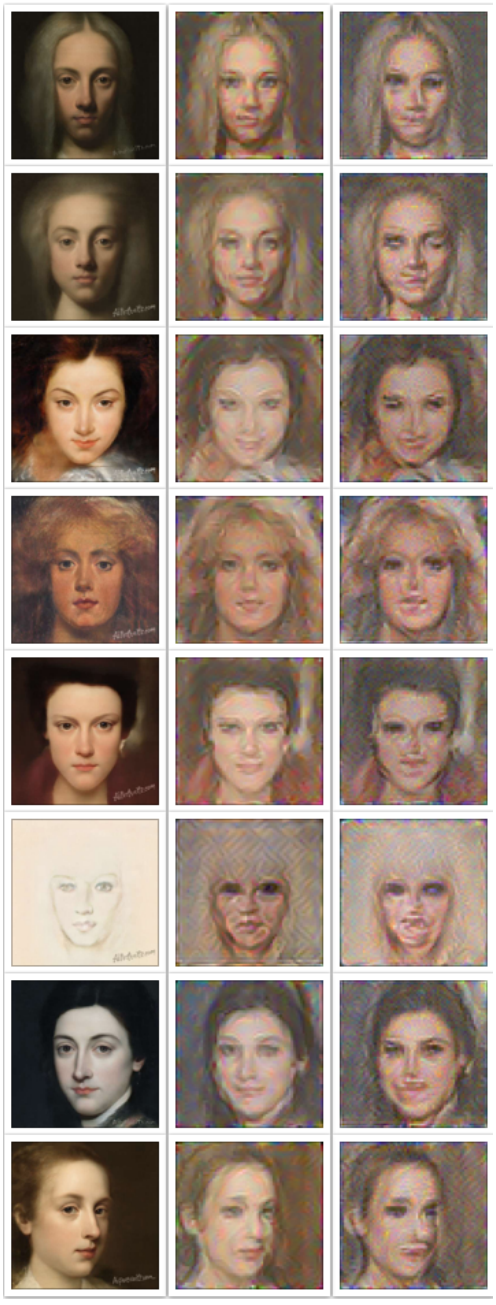


(a) Portrait to photo.

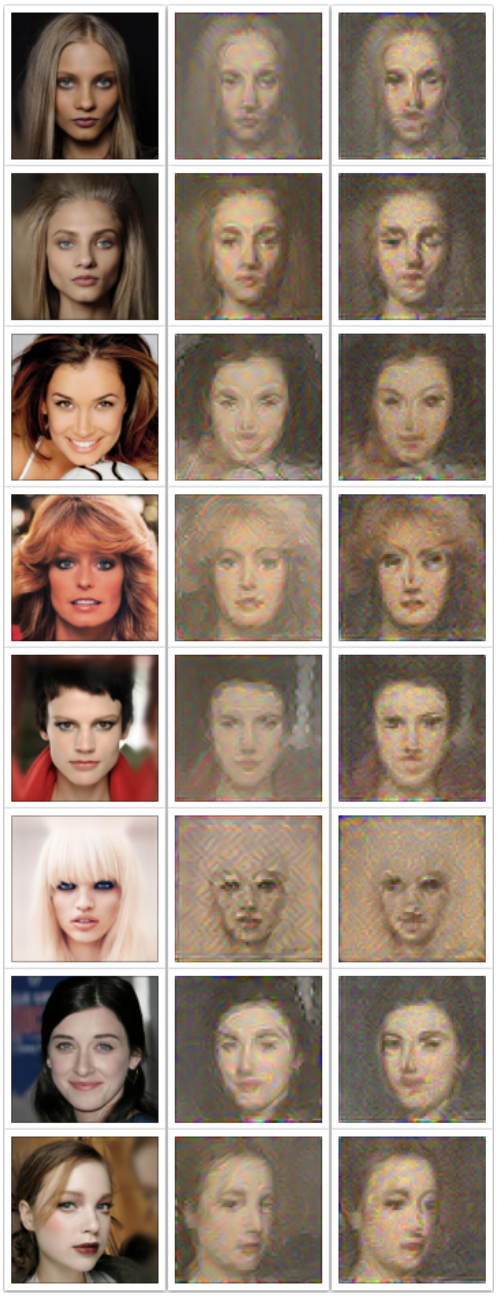


(b) Photo to portrait.

Figure A.2: More results for comparing layer conv3_2 and conv4_2. The left column show the original image, the middle column matching conv3_2 and the right column matching conv4_2.

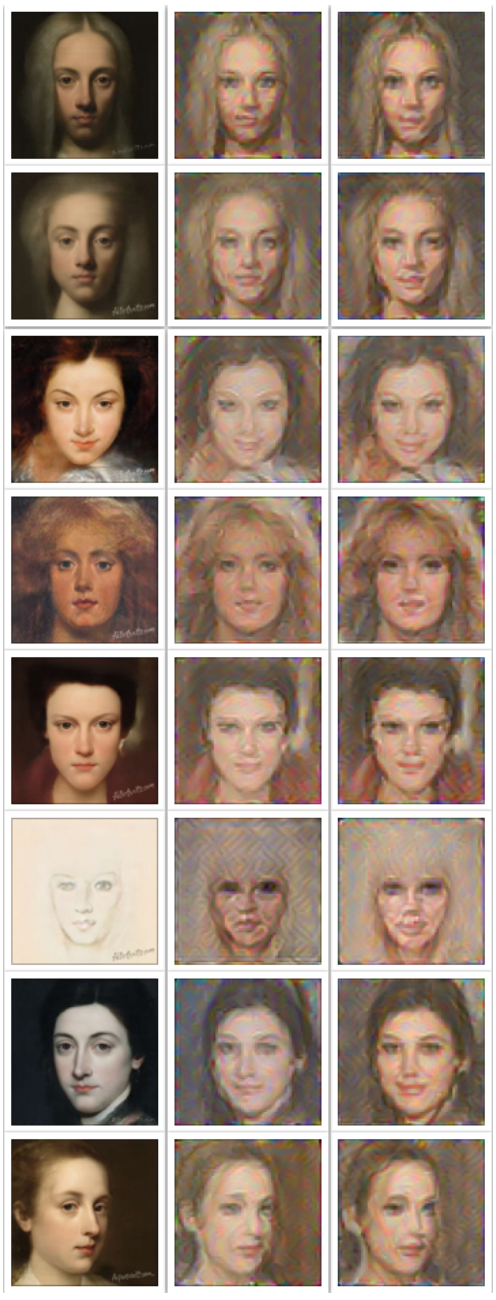


(a) Portrait to photo.

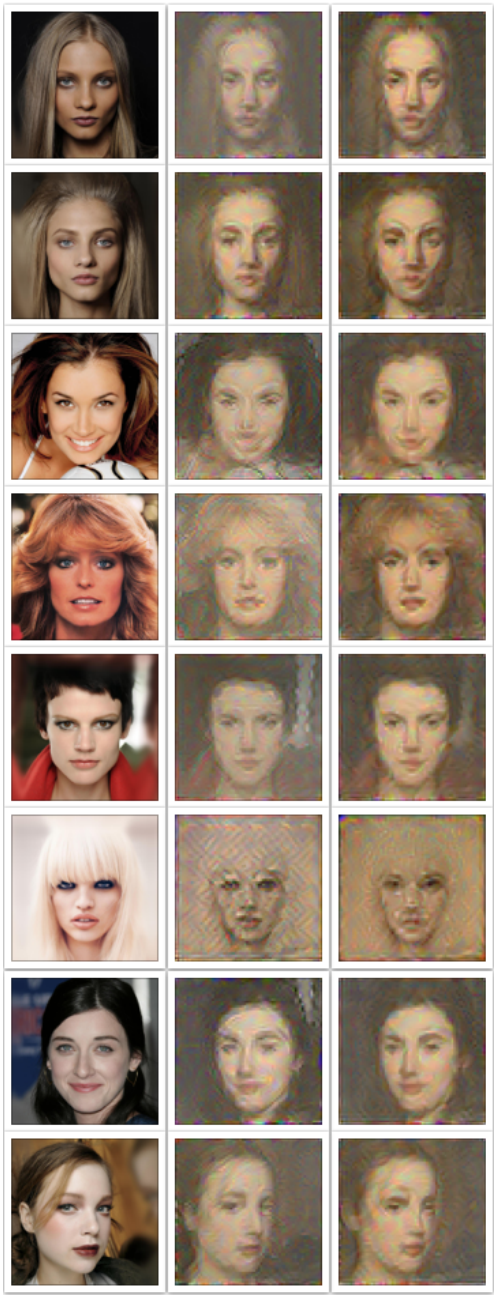


(b) Photo to portrait.

Figure A.3: More results for initialization. The left column show the original image, the middle column initializing from corresponding artwork or photo image and the right column initializing from white noise.



(a) Portrait to photo.



(b) Photo to portrait.

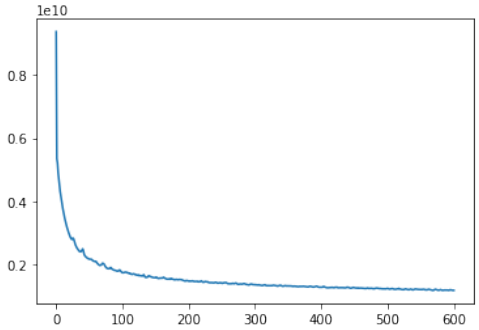
Figure A.4: More results for initialization. The left column show the original image, the middle column initializing from corresponding artwork or photo image and the right column initializing from mean image.

(a) Portrait to photo.

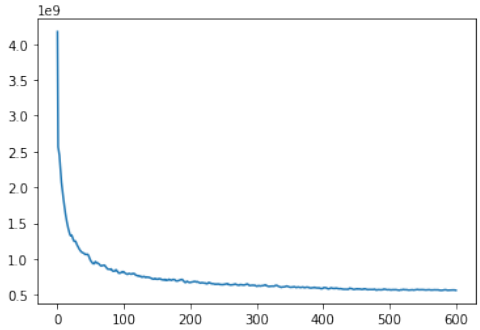
(b) Photo to portrait.

Figure A.5: More results for different layers in generator. The left column show the original image, the middle column using dense layers and the right column using convolutional layers.

A.2 Additional Quantitative Results

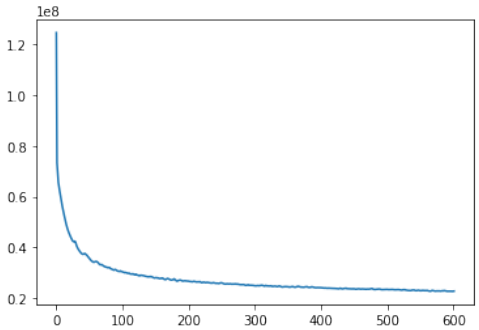


(a) Portrait to photo.

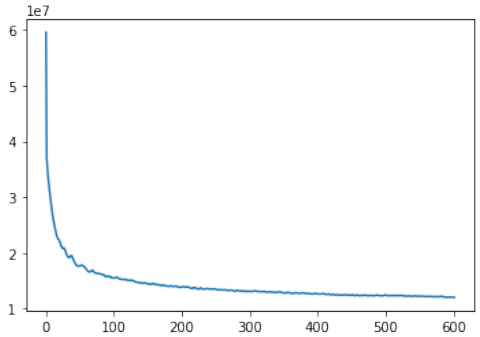


(b) Photo to portrait.

Figure A.6: Training loss when $\alpha/\beta = 10^{-3}$.

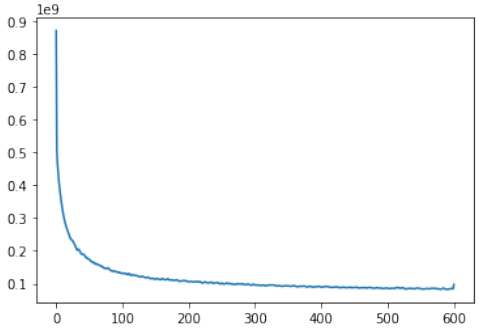


(a) Portrait to photo.

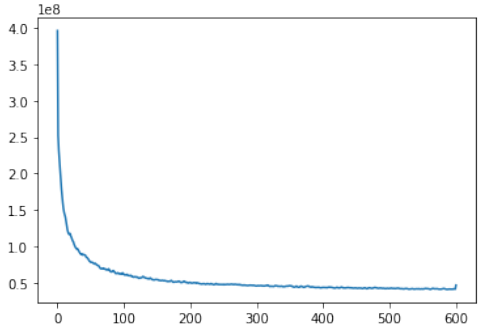


(b) Photo to portrait.

Figure A.7: Training loss when matching conv4_2 as content representation.



(a) Portrait to photo.



(b) Photo to portrait.

Figure A.8: Training loss when using convolutional layers.

A.3 Code

For accompanying code, please see
`https://github.com/ChangLiuuczlcl8/0091-MSc-ML-Project`