UNIVERSITY OF CALIFORNIA

Los Angeles

Incomplete Image Filling by Popular Deep Learning Methods

A thesis submitted in partial satisfaction

of the requirements for the degree

Master of Science in Statistics

by

Nan Ji

2019

ABSTRACT OF THE THESIS

Incomplete Image Filling by
Popular Deep Learning Methods

by

Nan Ji
Master of Science in Statistics
University of California, Los Angeles, 2019
Professor Mark Stephen Handcock, Chair

Incomplete image filling task, often known as the image inpainting task, is a popular topic in the applied deep learning field. This thesis paper considers several popular designs for deep neural networks including the supervised autoencoder and unsupervised generative adversarial networks. The goal of this thesis is to give researcher guidelines for solving the image inpainting task by deep learning methods with limited resources.

The thesis of Nan Ji is approved.

Yingnian Wu

Hongquan Xu

Mark Stephen Handcock, Committee Chair

University of California, Los Angeles

2019

To my Mum and Dad

who supporting me for all these years of life

and tolerate me for all my mistakes.

To all professors in the UCLA Statistics Department

who gave me the best experience

during the years of my study life.

To my friends

who will always be my good companions.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

Introduction

Incomplete image filling, usually called image inpainting, is one common task in the basic computer vision field. This task usually has an input partially masked image and returns an output image with the missing part filled in with convincing contents. To make the filling patches visually plausible becomes the most difficult obstacle to conquer. A very huge difference between the human and the machine is that the human beings have the ability to imagine similar things based on a very small set of examples. However, the machine needs to learn the patterns from very huge training datasets. When it comes to the image inpainting task, it won't be a huge problem for people to know what is missing when they see an incomplete image. Human beings could fill the missing image quickly by reading in the background context. So when the machine tries to solve this problem, we would want it to give something not only resonable but also natural.

Typical applications of the incomplete image filling are related to image editing. These applications include the image watermark removal and damaged photos restoration. Though image watermark removal might involve the copy-right problem, damaged photos restoration is commonly needed in people's lifes. Photo taking and editing are very important applications in smart phones. People love to take a photo and edit the photo in a way they like. If people find that something appears in the photo that make the photo not perfect enough, people would just want the smart phone to remove that part and restore some alternative contents to fill in. That's why solving image inpainting issue becomes more and more popular.

Many fancy methods in the deep learning field have already been published and commer-

cialized by different technology companies such as Google or Adobe. Restricted to resources I can access, my thesis here is not going to focus on very deep neural network models for large-scale real-life messy data. My intension is to introduce methods that normal students could try by on their own laptops to have a better intuition about how to deal with the image inpainting task. From basic Convolutional Networks to the generative models, students could follow this thesis paper as a guideline to have a better understanding of one deep-learning application.

1.1 Problem Statement and Dataset

The problem of incomplete image filling is large. Actually, any kind of real-life image with a random masked area or random missing patches can be chosen as the training data for this task. However, I mentioned before that restricted to my resources of computing power, it would not be possible to realize a more flexible way of filling missing patches in high resolution images. So to be more realistic for a tutorial or a guideline purpose, this task would be completed with a more economic and fundamental design.

The images consist of the missing patches we try to reconstruct and the context. I chose to mask an n x n square patch of the original image and manually create the flawed images used for training. Though currently there are numbers of open-source images freely available online, I used a relatively basic image dataset, the CIFAR10 dataset, for the task. The dimensionality for one training image will be 32 x 32 x 3, which is 32 pixels by 32 pixels for 3 channels (RGB).

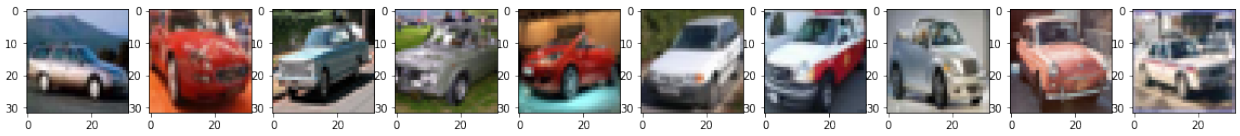

Figure 1.1: A sample of CIFAR10 car images.

To generate a better dataset to prevent the overfitting problem, I coded to create a much larger training dataset by randomly picking a place in an image to put this squared mask on. As a result, one simple image could be expanded to different types of flawed images. A basic deep learning training technique is to send small batches of training data every time. So, each batch of data here will be a randomly chosen set of images after the procedure of removing a random square patches. The networks would not just simply remember the training images but have an understanding of the overall patterns.

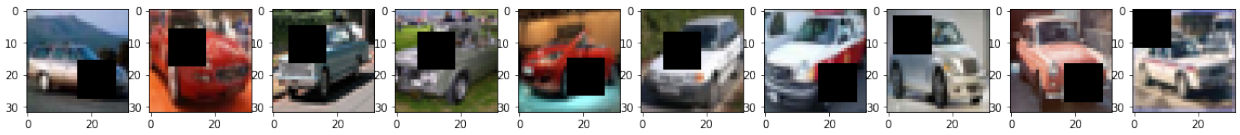

Figure 1.2: A sample of CIFAR10 car images with random masks.

According to the design of the task, we could separate image inpainting into two different types: blind and non-blind inpainting. The blindness here means that we are given the location of the missing patches to the model. The networks for the non-blind inpainting knows the exact location and the shape of what is missing in the image but the networks for the blind inpainting does not know the information (Deep image inpainting). The blind inpainting task is always very hard to solve, especially for normal students who want to taste the power of deep neural networks. As a result, I will focus on the non-blind inpainting task, which is relatively easier to accomplish, for this tutorial thesis paper.

Actually it is always easy to fill in something but hard to fill something plausible enough. The deep neural networks, especially the Convolutional neural networks could efficiently encode the important patterns, such as the shape or the colors of the targets, and reconstruct what is missing by a possible content matching what we want. However, one more problem would still exist that the smoothness between the fill-in content and the original image cannot be guaranteed. It needs to be taken more carefully both in the training process and the evaluations of the results. Detailed instructions will be introduced later in the thesis.

1.2 Related Work

In this section, I will introduce several techniques with fabulous results to address the same issue. Most of these techniques are published by the resourceful research groups in companies like Google or Adobe. I would not suggest students to try these methods alone because without enough computing power, it would be very hard to do the replications. However, it is necessary to have the insight into the current trends in the side of deep learning applications field. Additionally, most of these very advanced techniques can not only solve the image inpainting task perfectly but also have high flexibility in image generation tasks.

One recent paper[8] published in 2018 introduced a partial convolutional layer design with customized loss function targeting on solving irregular masks on the images. It addressed the color discrepancy and blurriness problem when the common methods deal with the flawed image having unpredicted large amount of masked areas. The authors of the paper[8] used the partial convolution operation and mask update function together to form a Partial Convolutional Layer for the networks to detect and reconstruct the flaws location by location. The final model in the paper[8] is proved to be capable of handling various masks with different size and shape with the potential to greatly improve the translation of images from low resolution to super resolution.

Another paper[9] published in early 2019 proposed a creative method for the image inpainting. Not filling in the masked areas directly, the model in the paper[9] would predict the edges of the missing parts by an edge detector and draw the entire item back based on the completed edges. The edge generator will input the incomplete grayscale image with an edge map and predict the full edge map. The image completion network will input the combination of the predicted edge map and the incomplete color image and solve the inpainting problem. This EdgeConnect deep learning model from the paper[9] has a huge flexibility on real life images and it is believed to be used as an interactive image editing tool because of its excellent results.

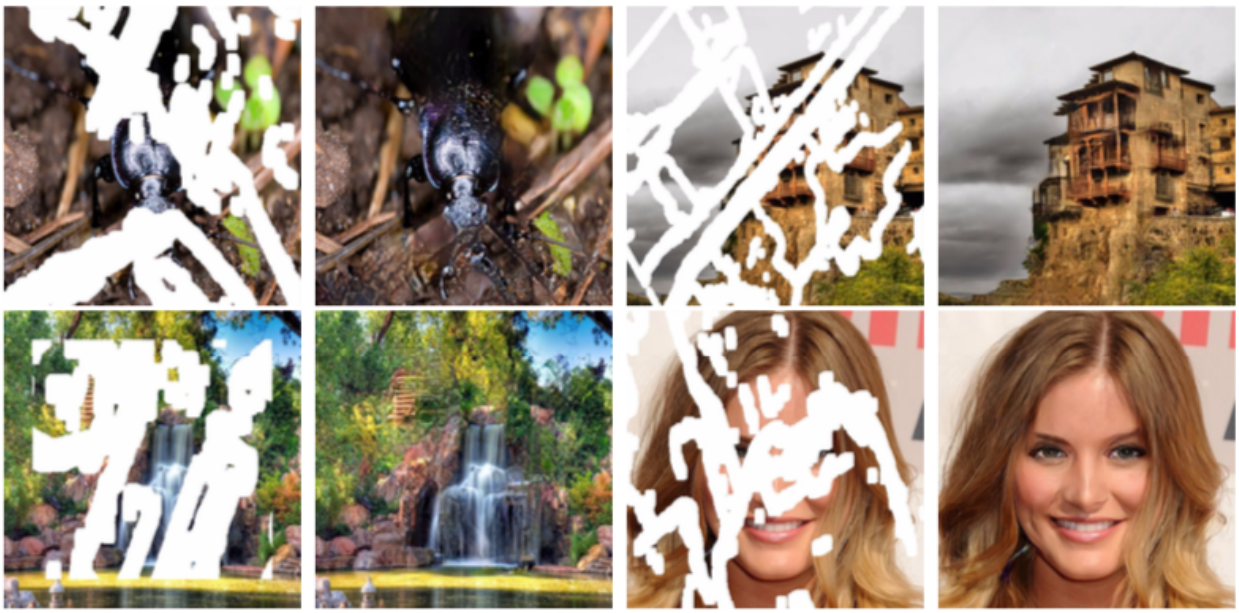

Figure 1.3: The performance of the networks using the partial convolution layer.

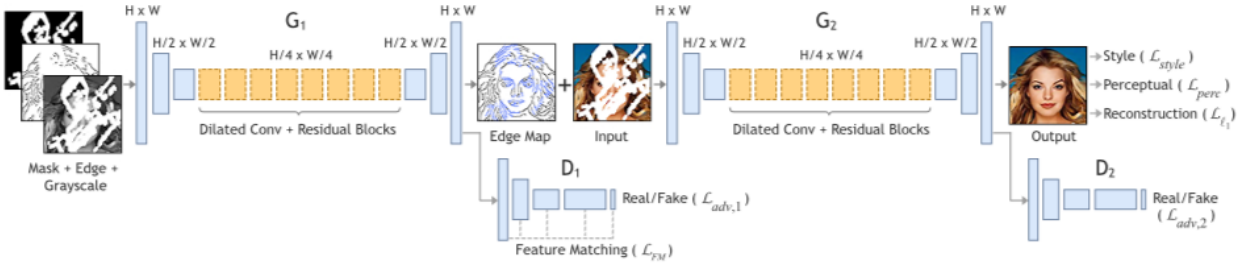

Figure 1.4: The structure of the EdgeConnect model.

CHAPTER 2

Methodology Background and Intuition

For this thesis, various popular deep learning methods and networks structures are applied. In this methodology section, the background information of the development of these methods and the general mathematical proofs are presented step by step. It will be hard to explain everything in detail. However, this would help to build the intuitions for a better understanding of what is happening behind the packages and functions students could directly use in python.

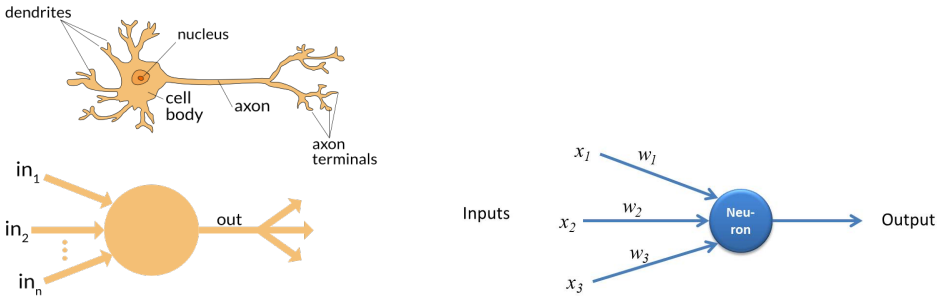

Figure 2.1: Left: A neuron. Right: A perceptron.

Why neural networks based deep learning or machine learning methods are so powerful of dealing with different kinds of difficult tasks these days? One simple explanation is that the structure of neural networks can be considered as a very high-dimensional nonlinear function that gives a great performance in approximating the real relationship behind the data. According to the book[6] chapter 6, we call the basis of deep neural networks as deep feedforward networks, feedforward neural networks or multilayer perceptrons. The math formula gives a better intuition of function approximation provided by parameters θ that the feedforward network can be represented as a mapping $y = f(x;\theta)$. Here, f is our neural

network functions. Figure 2.1 shows the intuition of the neuron and the structure of the perceptron. Usually we just have the formula $f(X;\theta) = WX + b$ for a perceptron.

A full forward structure of neural networks is represented below:

$$X \to^{W_1} h_1 \to^{W_2} h_2 \to ... \to^{W_{l_1}} h_{l-1} \to^{W_l} h_l \to_{\cdot} .. \to^{W_L} h_L \to S(orY)$$

h_l means values of the hidden layer and the formula is $h_l = W_{l-1}H_{l-1}$. The first hidden layer is directly from the input data: $h_1 = W_1X$. And the final output is usually just the values of the final hidden layer or a transformation of the final hidden layer values. The dimension of each hidden layer is usually designed by ourself. However, when using neural networks, we need to match the dimensions of the input and the output data. Because of the multilayer structure, neural networks have super-high dimensionality, which makes it hard to calculate the parameters W_l for each layer. The breakthrough is the back-propagating methods.

The paper[5] published in 1986 first stated the back-propagation method which make the training of neural networks possible. Actually from what I introduced above, we could find that $Y = W_L(W_{L-1}(...(W_1X)))$. According to the matrix multiplication, when we have the loss function for Y, we could do derivation for each layer until the input X. And based on these back-forward derivatives, the gradient descent based algorithms can be applied to update the loss function step by step. Let's see the back-propagation presentation.

$$\frac{\partial L}{\partial X} \leftarrow^{\frac{\partial L}{\partial W_1}} \frac{\partial L}{\partial h_1} \leftarrow ... \leftarrow^{\frac{\partial L}{\partial W_l}} \frac{\partial L}{\partial h_l} ... \leftarrow^{\frac{\partial L}{\partial W_{L-1}}} \frac{\partial L}{\partial h_{L-1}} \leftarrow^{\frac{\partial L}{\partial W_L}} \frac{\partial L}{\partial h_L}$$

Each stage of this process consists of two parts: derivation on the hidden layer h_l and the derivation on the parameters W_l for each layer. The derivations can be shown by two formulas.

1. $\frac{\partial L}{\partial h_{l-1}^T} = \frac{\partial L}{\partial h_l^T}\delta_l W_l$
2. $\frac{\partial L}{\partial W_l} = \delta_l \frac{\partial L}{\partial h_l} h_{l-1}^T$

The δ here means the activation layer. The activation layer contains the activation function that does a further transformation of the hidden layer outputs. The activation gives a

chance for the neural networks to decided which neurons should be fired, especially when we use the Relu (Rectified Linear Unit) activation function. The Relu activation function is $A(x) = max(0, x)$, which gives a clear bound to the outputs from the hidden layer. We can consider the δ here as the vector form of Relu activation function: $\delta : [\delta_i = 0, \delta_j = 1]$ if $h_i < 0$ and $h_j \geq 0$. If we add the Relu activation layer into one stage of the forward networks, we will have a better understanding.

$$W_l h_{l-1} \rightarrow \delta_l \rightarrow^{Relu} h_l$$

Then based on our choice of loss function L, we can do gradient descent based algorithms such as stochastic gradient descent with the momentum or the Adam algorithm to train the neural networks.

2.1 Supervised Learning Models

To do the masked image reconstruction task, the direct way is to input the masked images and compare the outputs with the complete images to train the model. This is a supervised learning mission. Popular deep learning models for image constructions include the direct convolution neural networks and the autoencoder model. These models will be introduced below.

2.1.1 Convolution Neural Networks

The deep learning book[6] chapter 9 gives a thorough introduction of the convolutional networks. This chapter starts with the convolution operation that contains the multidimensional array of input data and the multidimensional array of parameters as kernel implementing the infinite summation as a summation over a finite number of array elements. We can use a two-dimensional kernel K for a two-dimensional input image I to show a convolution operation:

$$S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(m, n)K(i - m, j - n)$$

In addition, the convolution operation is commutative and the commutative formula is implemented more straightforward because of less variation in the range of valid values of m and n. The commutative formula is shown below.

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n)$$

We could see that we could choose the values for m and n based on the dimensionality of the input image. Based on the design of the convolution, it is not necessary to do linear operations on the whole image directly but select small regions of the image to condense and detect the patterns. As the kernel moves around the image, different kinds of patterns would be detected based on the specific kernel we use. For example, if we have a 3 x 3 kernel with only one row of 1 and other indices equal to 0, this kernel would detect a short horizontal bar in the image. And these patterns will be stored in various filters. The total number of filters can be chosen by our design, just like the dimension increasing for the hidden layers in the forward neural networks.

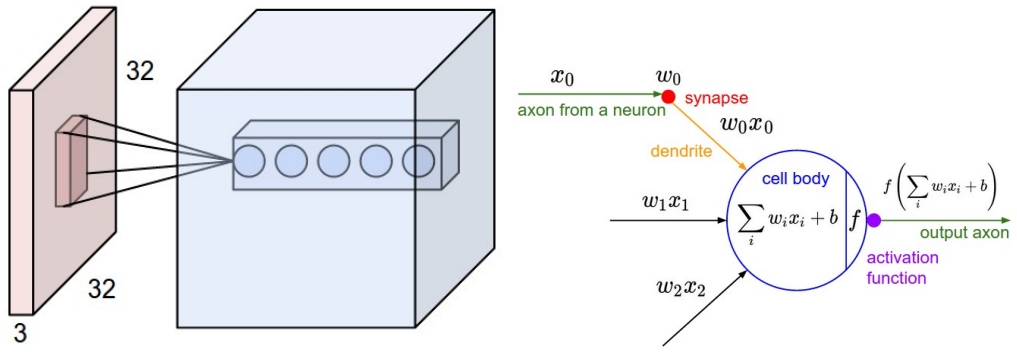

Figure 2.2: Left: One Convolutional Layer. Right: Math formula of one neuron.

Figure 2.2 are two pictures from the Stanford CS231 note[2]. The left one shows an example of neurons in one Convolutional layer for a CIFAR-10 image. We could see that according to the kernel, we could store the pattern into a filter with different dimension. Here, the next layer filter contains five neurons. The right figure shows that the neuron is still as same as what we have in a normal forward neural network with a dot product of the weights and the input. However, the difference is that this time the input restricted to be local spatially

because of the connectivity caused by the kernel.

When coding for the convolution networks, there are two more important parameters. We need to specify the "stride" and the size of the zero-padding. The stride determines the number of pixels that the filter slides. If stride is 1, then the filter will slide one pixel at a time. The zero-padding is used to pad the input volume with zeros around the border so that it would be easier to control the size of the output volumes. According to the CS231 note[2], we could use a function to calculate the size of the output volume:

$$Size_{output} = \frac{W - F + 2P}{S} + 1$$

W is the input volume size, F is the size of the filter in the receptive field size, S is the

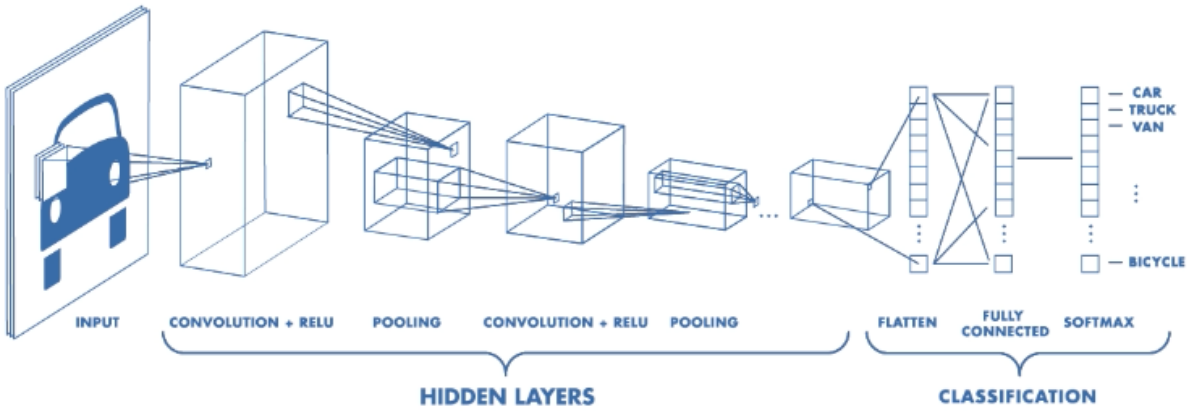

Figure 2.3: A structure of Convolution Neural Networks for the classification task.

stride and P is the amount of zero-padding used on the border. Only If the result is an integer, we could consider the output size would be a fit or valid. For instance, a 7 x 7 input and a 3 x 3 filter with stride 1 and pad 0 will give a 5 x 5 output. After one layer of the convolution network, we could add a pooling layer to stabilize the extracted pattern and an activation layer to reduce the activated neurons. It all depends on how we are going to

design the structure.

For the image inpainting, not like the final part for a classification purpose in the Figure 2.3, we would use multi-layer convolution neural networks with an output having the size as same as the input image. The common patterns will still be learnt and used to reconstruct the missing parts.

2.1.2 AutoEncoder

The autoencoder is an efficient neural network that attempts to copy its input to its output with the hidden layer h that encodes the patterns representing the input data[6]. The Autoencoder model as a structure consists of one top-down model and a bottom up model. These two models are called encoder $h = f(x)$ and the decoder $x = g(h)$. It will compress the information first and then release the information again. The structure is shown in the Figure 2.4.

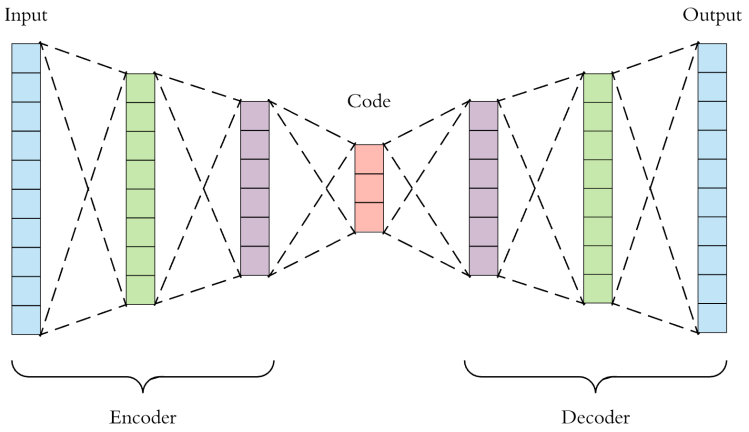

Figure 2.4: Each layer is a layer of a selected layer of neural network, such as the fully connected network, convolutional network or LSTM

The basic idea is to use layers of neural networks to encode the original data into a low-dimensional vectors of values first. It's like a summarization of the high-dim data by the low-dim data. However, unlike PCA, with the neural networks sturcture, the original data would be sent to a high dimensional space first and then to a very low dimensional space.

Specifically, by using the ConvNet, the decoder layers would be called Deconv layers.

To use the autoencoder, we would want to choose proper values for the number of layers, the number of nodes per layer and the size of the middle latent layer. By adjusting these values, we would have results that close to our expectations. However, it does not mean that the autoencoder model is perfect when it outputs perfect copy of the input data. The deep learning book chapter 14[6] tells us the autoencoder aims to not learn everything perfectly and it is designed to give an approximation that prioritize to learn the useful properties of the data. Same to all the other model training in the statistics, the overfitting model will never be our solution.

2.2 Unsupervised Generative Model

The generative model is a powerful way of learning different kinds of data distributions. The past several years of the development in the deep learning field helped the unsupervised learning methods achieve tremendous success. The purpose of generative models is to learn the data distribution to generate new data similar to the training data with some variations. However, it would still be hard to learn the exact distribution. Since deep neural networks give good approximations to the hidden relations within the data, efficient approaches such as Variational Autoencoders and the Generative Adversarial Networks are commonly used now.

2.2.1 Conditional Variational AutoEncoder

The variational autoencoder model is based on the structure of the autoencoder. However, the idea of variational autoencoder combines efficient approximate maximum likelihood or maximum a posterior estimation of the global parameters and approximate posterior inference of the latent variables[4]. The encoder and the decoder give strong power of approximation for the distributions. The conditional variational autoencoder is directly based on the variational autoencoder with the extension of some other data the model conditional on.

We would start with how the variational autoencoder works first.

2.2.1.1 Variational AutoEncoder (VAE)

The basic statistical design is to approximate the true posterior distribution $p_\theta(z|x)$ by a recognition model $q_\phi(z|x)$[4]. Here, θ represents the decoder model and ϕ represents the encoder model. Assume that we have the data x and some missing information z that directly relates with the model. Then the model we really want would be a joint distribution $p_\theta(z,x)$. We propose $q_\phi(z,x)$ to approximate $p_\theta(z,x)$ by minimizing the Kullback Leibler divergence:

$$min_{\theta,\phi}KL(q_\phi(z,x)|p_\theta(z,x)) = min_{\theta,\phi}E_q(log\frac{p_\theta(z,x)}{q_\phi(z,x)}) = min_{\theta,\phi}E_q(log\frac{p_\theta(x)p_\theta(z|x)}{q_{data}(x)q_\phi(z|x)}) \quad (2.1)$$

$$= min_{\theta,\phi}E_q(log\frac{p_\theta(x)}{q_{data}(x)} + log\frac{p_\theta(z|x)}{q_\phi(z|x)}) \quad (2.2)$$

$$= min_{\theta,\phi}E_q(log\frac{p_\theta(x)}{q_{data}(x)}) + E_q(log\frac{p_\theta(z|x)}{q_\phi(z|x)}) \quad (2.3)$$

$$= min_{\theta,\phi}KL(q_{data}(x)|p_\theta(x)) + KL(q_\theta(z|x)|p_\theta(z|x)) \quad (2.4)$$

We consider that the data itself follows a prior distribution $x \sim q_{data}(x)$. To minimize the first term $KL(q_{data}(x)|p_\theta(x))$ is actually to maximize the log-likelihood.

$$\hat{\theta}_{MLE} = argmin_\theta KL(q_{data}(x)|p_\theta(x))$$

So accuracy beyond the MLE divergence of the VAE (variational autoencoder) is controlled by the second term, the KL divergence between the recognition model and the true posterior distribution I mentioned above.

$$\hat{\phi}_{VAE} = argmin_\phi KL(q_\phi(z|x)|p_\theta(z|x))$$

However, there is a natural difficulty in directly minimizing this objective function. The traditional calculation of the normalizing constant $p_\theta(x)$ is too hard from $q_\phi(z|x)$. A reparameterization trick, similar to the idea of EM (expectation maximization) algorithm , is used and creates the computational efficiency.

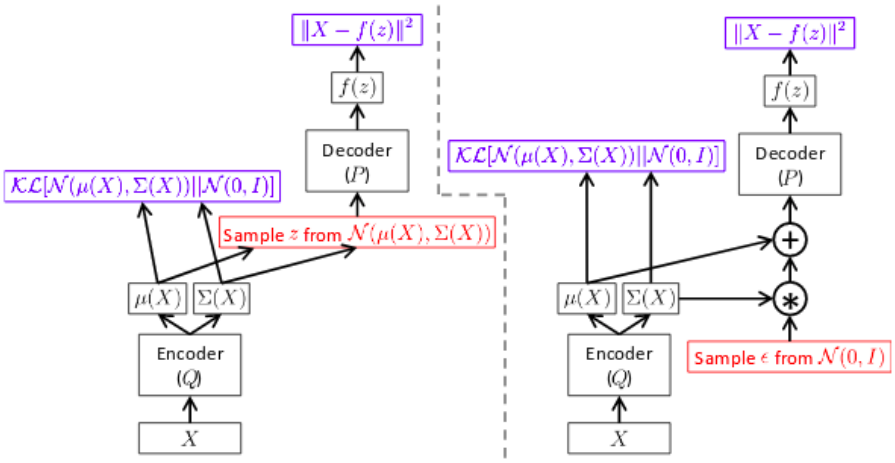

Figure 2.5: Reparameterization trick guide

Generating the missing values or structure data and then calculating the expectation maximized estimation iteratively is a common application of EM algorithm. Similar to this process, an alternative method that we assume missing data z follows the distribution $q_\theta(z|x)$ and we generate samples of z from $q_\theta(z|x)$[4]. According to the paper[4], though the form of $q\phi(z|x)$ has a lot of choices, we could assume that we use a Gaussian form with an approximately diagonal covariance to approximate the true posterior, which is intractable. As a result, we have the form:

$$q_\phi(z|x^{(i)}) = N(z; \mu^{(i)}, \sigma^{2(i)} I)$$

The paper[4] fully explained that the approximate posterior has mean $\mu^{(i)}$ and the standard deviation $\sigma^{(i)}$ as the outputs of the encoder model ϕ, the nonlinear functions of datapoint $x^{(i)}$. The paper[4] also gives a detailed process of how to calculate the differentiable KL divergence when we assume the posterior $z^{(i,l)} \sim q_\phi(z|x^{(i)})$ by using $z^{(i,l)} = g_\phi(x^{(i)}, \epsilon^l) = \mu^{(i)} + \sigma^{(i)} \cdot \epsilon^{(l)}$ where $\epsilon^{(l)} \sim N(0, I)$:

$$L(\theta, \phi; x^{(i)}) \simeq \frac{1}{2} \sum_{j=1}^{J} (1 + log((\sigma_j^{(i)})^2)) - (\mu_j^{(i)})^2 - (\sigma_j^{(i)})^2)) + \frac{1}{L} \sum_{l=1}^{L} logp_\theta(x^{(i)}|z^{(i,l)})$$

where $z^{(i,l)} = \mu^{(i)} + \sigma^{(i)} \cdot \epsilon^{(l)}$ and $\epsilon^{(l)} \sim N(0, I)$

Above will be a general form of the loss function we use for the VAE and the $\frac{1}{L}\sum_{l=1}^{L} logp_\theta(x^{(i)}|z^{(i,l)})$ part is actually for the decoder. Depending on the type of our data, we could choose different kinds of loss function for the decoder part such as the mean sqaured error or the cross-entropy.

2.2.1.2 Conditional Variational Autoencoder (CVAE)

Conditional Variational Autoencoder is based on the assumption of VAE with an improvement of introducing new conditioning parts for the probability density. The difference is very straightforward. We could simply look at the derived form from the KL divergence equation.

VAE:

$$logP(x) - KL(q_\phi(z|x)|p_\theta(z|x)) = E(logp_\theta(x|z)) - KL(q_\phi(z|x)|p_\theta(z))$$

CVAE:

$$logP(x|c) - KL(q_\phi(z|x,c)|p_\theta(z|x,c)) = E(logp_\theta(x|z,c)) - KL(q_\phi(z|x,c)|p_\theta(z|c))$$

where c is something new that we want the model conditioning on for the specific reason.

Now, the real latent variable is distributed under $p_\theta(z|c)$. It's a conditional probability distribution that for each possible value of c, we would have a $p_\theta(z)$. We could also use this form of thinking for the decoder. It gives us a chance to learn the distribution of specific groups of the data, such as the hand writing digits seperated by labels, the human face images seperated by genders and especially the image inpainting conditioning on the masked areas. The detailed operations in coding will be introduced in later chapters.

The variational autoencoder based model can be considered as an unsupervised learning model because we are learning latent distributions generating the actual training data we want. After the training process is done, we could just use the learnt distributions and the decoder to generate images. This design is an improvised version of the supervised autoencoder with the power of learning a generator.

2.2.2 Generative Adversarial Networks

The generative adversarial networks (GAN) introduce innocations in the design. Generally, the DCGAN (deep convolutional generative adversarial networks) consists of two networks, a generator and a discriminator. The network is doing a game theory approach trying to find Nash equilibrium between the generator and the discriminator instead of working with an explicit density estimation [10]. The Figure 2.6 gives the intuition about how we combine the generator and the discriminator.

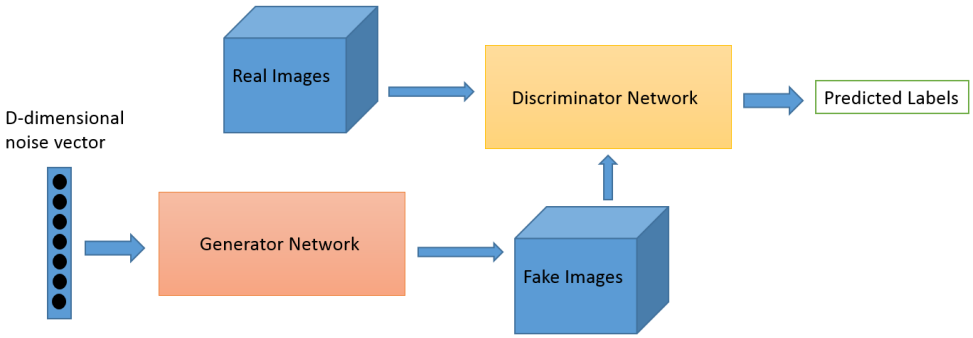

Figure 2.6: The structure of the GAN model

The generator is designed to create plausible images and the discriminator is going to judge if the generated images are plausible enough to be considered as the real images. It is a great idea that lets two models compete with each other and finally we would want the generator can be good enough to cheat the discriminator so that it can give us natural looking images. Both of the networks will improve their performance during this competition. Though it would be easy to construct a GAN model by coding according to large numbers of sources online, we need to look into the statistical ideas behind it because such a structure has the freedom to cooperate with various methods by proper derivations.

According to the original GAN paper[7], the input of the generator is a noise variable $z \sim p_z(z)$ and we want to learn the generator's distribution p_g over data x. The mapping, $G(z;\theta_g)$, from the input noise z to the data x is a differentiable function by the multilayer neural networks with parameters θ_g[7]. The discriminator $D(x;\theta_d)$ is another multilayer

neural networks model that calculates the probability of x coming from the data space rather than the generator's distribution, p_g[7]. In practice, we need to label the fake images from the generator and the real images separately. So the discriminator $D(x)$ will maximize the probability of assigning correct labels to the fake images and the real images during the training process[7]. As a result, the global loss function needs to combine both the generator perspective and the discriminator perspective. In the GAN paper[7], a value function $V(G, D)$ is proposed to train G to minimize $log(1 - D(G(z)))$:

$$min_G max_D V(D, G) = E_{x \sim p_{data}(x)}[logD(x)] + E_{z \sim p_z(z)}[log(1 - D(G(z)))]$$

We are training G and D as a pair by an adversarial zero sum game. To compute these two expectation exactly is very hard. Two approximations are needed. $E_{x \sim p_{data}(x)}$ can be approximated by averaging the training samples we have. We could use the empirical loss function over the training sample such as the averaged classification error or the cross entropy. For $E_{z \sim p_z(z)}$, we could use the Monte Carlo average over the faked images created by the generator to do the approximation. In addition, to avoid vanishing gradient issues, we could modify $min_G E_{z \sim p_z(z)}[log(1 - D(G(z)))]$ to be $max_G E_{z \sim p_z(z)}[log(D(G(z)))]$ to train G.

If a perfect discriminator is $D(x) = \frac{p_{data}(x)}{p_{data}(x) + p_g(x)}$ when the real and fake images have the same numbers and $p_g(x)$ is the distribution of the generator $G(z; \theta_g)$ with $z \sim p_z(z)$, our value function will become

$$\begin{aligned} min_G max_D V(D, G) &= E_{x \sim p_{data}(x)}[logD(x)] + E_{z \sim p_z(z)}[log(1 - D(G(z)))] && (2.5) \\ &= min_{p_g} E_{p_{data}(x)}[log \frac{p_{data}(x)}{p_{data}(x) + p_g(x)}] + E_{p_g(x)}[log(\frac{p_g(x)}{p_{data}(x) + p_g(x)})] && (2.6) \\ &= min_{p_g} KL(p_{data} | p_{data} + p_g) + KL(p_g | p_{data} + p_g) && (2.7) \\ &= min_{p_g} KL(p_{data} | p_{mix}) + KL(p_g | p_{mix}) && (2.8) \end{aligned}$$

where $p_{mix} = \frac{1}{2}(p_{data} + p_g)$.

According to the formula, we could see that there would be a trade-off between the quality

of the generated fake images and how much we captured the data space.

For the image inpainting task, we can use a DCGAN model directly to generate fake images to replace the flawed images. However, more complicated designs can be combined into the generative adversarial network structure for efficiency and precision. The detailed methods and their performance will be introduced in the later chapter.

CHAPTER 3

Models Structures, Experiments and Results

Based on the methods introduced above, 5 models for the image inpainting task are constructed. The detailed design and results will be evaluated. Good examples and failed cases will be presented as well. The training process is coded by Keras in the Jupyter Notebook file and uploaded to my Github account (nji3).

3.1 Direct Convolution Neural Networks

The first method is a direct convolution network that only contains convolution layers. The loss function for this method is the mean squared error between the reconstructed flawed car images and the original images in the training dataset. The power of convolution networks to efficiently encode the information of images is confirmed by many studies. For the image inpainting task, I choose to use mainly convolution layers to construct all the other networks. As a result, I would find out how well a purely direct convolution network can perform.

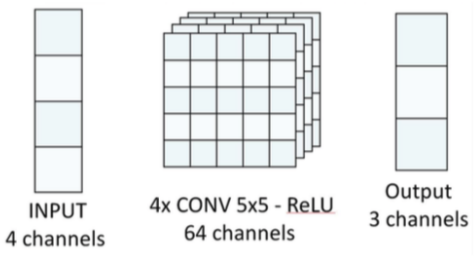

Figure 3.1: An example of the Direct Convolution Networks structure

The Figure 3.1 from the paper[3] shows the structure of a well-performed direct convolution network. The input data will be a combination of the 3-channel masked training images with

a location map indicating where the masks are. The location map is a matrix with 1s at the unmasked region and 0s at the masked part. Adding the location map as the fourth channel of the input images could give the model a more specific target to work on. Additionally, the detail of the networks I used is presented in the table below.

Table 3.1: Direct Convolution Networks Summary

Layers	Filter Size	Stride Number	Padding	Output Shape
CONV1 + RELU1	5x5	1	Same	32x32x64
CONV2 + RELU2	5x5	1	Same	32x32x64
CONV3 + RELU3	5x5	1	Same	32x32x64
CONV4 + RELU4	5x5	1	Same	32x32x64
CONV5 + Sigmoid	5x5	1	Same	32x32x3

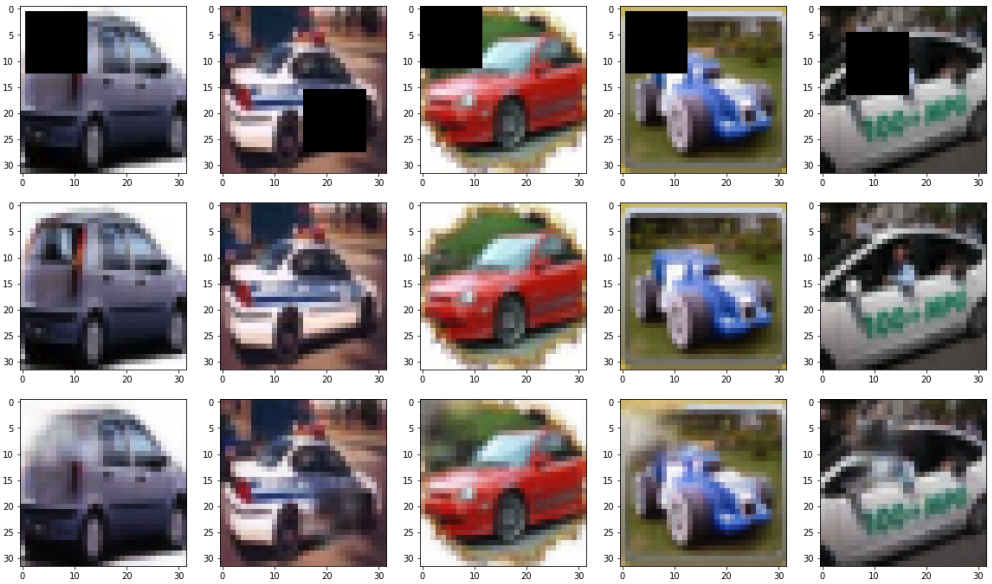

Figure 3.2: Row1: Masked images, Row2: Original images, Row3: Filled images

The parameters of the model used were determined after various tests. With the ADAM optimization algorithm, the loss drops and goes to converge very quickly. In my experiment, I did not adjust the filter size and the stride number a lot. However, according to the paper[3], it seems that 5x5 could be a good filter size for each convolution layer. A sample

of the refilled images from the validation set is presented in Figure 3.2.

The direct convolution network shows the advantage to do very good reconstruction on the unmasked area, the given information. On the other hand, the masked area is filled by something close to an averaged pooling based on the nearby contents and colors. Though the blur problem is still huge, it gives a good direction to use such a convolution-layer structure in other methods.

3.2 AutoEncoder

Image inpainting is part of numbers of image generating problems including deblurring and denoising. Autoencoder-like models are usually very efficient to solve these problems. The autoencoder networks are introduced in the methodology chapter. The encoder networks will encode the input image into a lower-dimensional representation and the decoder networks will release the information to reconstruct the images.

Table 3.2: AutoEncoder to Reconstruct the Whole Image, Encoder Summary

Layers	Filter Size	Stride Number	Padding	Output Shape
CONV1 + RELU1 + BatchNorm1	3x3	1	Same	32x32x32
CONV2 + RELU2 + BatchNorm2	3x3	2	Same	16x16x64
CONV3 + RELU3 + BatchNorm3	3x3	2	Same	8x8x128
CONV4 + RELU4	3x3	2	Same	4x4x256

According to the performance of the direct convolution networks, I built the encoder networks with only convolution layers. The size of the bottleneck inner layer between the encoder and decoder is 4x4x256. The decoder networks contain the transposed convolution layers and one convolution layer to generate the final output. The transposed convolution is a mathematical operation using matrices to go back from the convolved values to the original. So the transposed convolution layers go an opposite direction of the convolution layers and

Table 3.3: AutoEncoder to Reconstruct the Whole Image, Decoder Summary

Layers	Filter Size	Stride Number	Padding	Output Shape
Transposed CONV1 + RELU5	3x3	2	Same	8x8x256
Transposed CONV2 + RELU6	3x3	2	Same	16x16x128
Transposed CONV3 + RELU7	3x3	2	Same	32x32x64
CONV5 + Sigmoid	3x3	1	Same	32x32x3

increase the layer output shapes step by step, which is very useful to release the condensed patterns. The detail of the encoder is in Table 3.2 and the detail of the decoder is in Table 3.3.

In the encoder, batch normalization layers are used after the RELU activation layers. The batch normalization layers help reduce the covariance shift between layers and allow each layer to learn by itself more independently. Here, adding the batch normalization layers helped a quicker convergence in the training process.

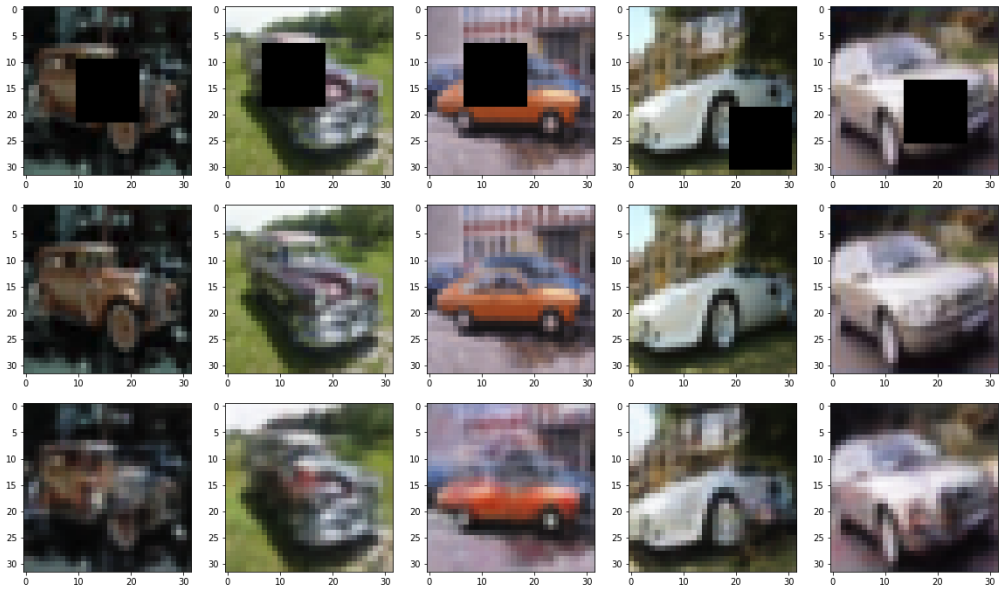

Figure 3.3: Row1: Masked images, Row2: Original images, Row3: Filled images

Practically, there is a reason to use a convolution layer as the final layer to generate the output images. In my experiments, a full transposed-convolution-layer decoder would fail to

reconstruct the boundary of the output images. Adding one convolution layer would help stabilize the result we want.

A sample of the results from the test dataset is chosen to present in Figure 3.3. The number of parameters in the autoencoder model I built actually is just a bit larger than the direct convolution network I used before. However, the reconstructed masked areas have more specific contents, which is a clear improvement. Though the blurriness is not efficiently solved, another typical problem of the autoencoder appeared as well. Because the reconstruction is based on the embedded inner layer, it is possible that some patterns of the cars will be predicted wrongly. In the middle case, the car's color is not very accurate comparing to the original image. In the second image from the right, the model mistakenly filled a smaller tire for the masked area. These reconstructions might be resonable, but the results do not seem plausible enough.

3.2.1 Reconstruct the masked area only

One solution to maintain the accuracy of the unmasked areas is to only output the masked regions through the networks. This idea is proposed and implemented in the paper[1]. Figure 3.5 is from this paper[1] showing the design of an autoencoder structure to predict the masked area only.

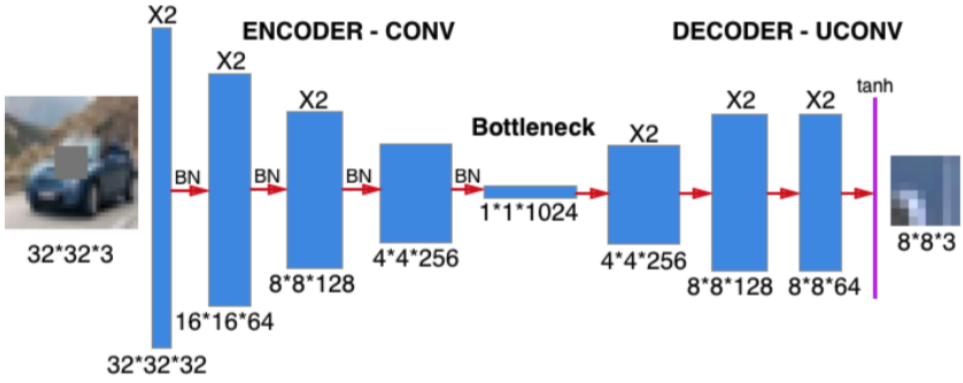

Figure 3.4: The Autoencoder structure to predict the masked area only

According to the paper[1], the loss function is the average element-wise L_2 between the

predicted masked region and the original region in the image. The loss function is $L = \frac{1}{n^2}\sum_{p,q,r}(R_{p,q,r} - \hat{R}_{p,q,r})^2$. The author in the paper[1] chose $n = 8$, but here I chose to use $n = 12$, which is a bit larger masked area.

Figure 3.5: Row1: Original patches, Row2: Predicted patches, Row5: Filled with predicted patches

Figure 3.5 presents a sample of the results from the test dataset. To be honest, the result is not very promising. The biggest problem is overfitting. Though the training results do not show in the paper, my training process could give perfect predictions. Considering the perfect results for the training dataset, the overfitting problem has a long way to work through.

3.3 Conditional Variational AutoEncoder

The supervised model showed limitation when dealing with the image inpainting task. It is resonable to test the capability of the unsupervised model mentioned in the methodology chapter. The structure of my autoencoder model gave a good insight of the layers for the conditional variational autoencoder model. After experiments using different combinations of the parameters, I constructed the networks based on the training time and efficiency. The detail is shown in the Table 3.4, Table 3.5 and Table 3.6.

The encoder network in CVAE is almost a duplication of the encoder in the autoencoder

Table 3.4: CVAE Encoder Summary

Layers	Filter Size	Stride Number	Padding	Output Shape
CONV1 + RELU1 + BatchNorm1	3x3	2	Same	16x16x32
CONV2 + RELU2 + BatchNorm2	3x3	2	Same	8x8x64
CONV3 + RELU3 + BatchNorm3	3x3	2	Same	4x4x128
CONV4 + RELU4	3x3	2	Same	2x2x256

model. One difference is that the stride number for each layer is not exactly identical. The output of the encoder network in CVAE has a smaller shape, which is 2x2x256. It embedded patterns more densely. The smaller shape would also efficiently reduce the total parameter numbers of the fully connected layers to predict the means and log variances (Table 3.5). Though the larger dimensions of the latent variables would help learn more complicated multi-normal distributions, the effects were not directly proportional to the size of the latent layers. The size 1024x1 had a high cost performance.

Table 3.5: CVAE Inner Layers for Representation Tricks

Layers	Operations	Output Shape
Flatten layer1	flatten the encoder output	1024x1
Fully connected layer1	means	1024x1
Fully connected layer2	log variances	1024x1
Lambda layer1	representation trick	1024x1
Flatten layer2 + Concatenate layer	combine location map	2048x1
Reshape layer	input for the decoder	2x2x512

When the representation tricks were done, it is important to combine the sampled outputs and the flattened location map as an input for the decoder networks. The combined input will be reshaped from 2048x1 to 2x2x512 for the next convolution layer.

The decoder networks have one more transposed convolution layer than the decoder in the

Table 3.6: CVAE Decoder Summary

Layers	Filter Size	Stride Number	Padding	Output Shape
Transposed CONV1 + RELU5	3x3	2	Same	4x4x256
Transposed CONV2 + RELU6	3x3	2	Same	8x8x128
Transposed CONV3 + RELU7	3x3	2	Same	16x16x64
Transposed CONV4 + RELU8	3x3	2	Same	32x32x32
CONV5 + Sigmoid	3x3	1	Same	32x32x3

autoencoder model I used before. Several examples of the reconstructions from the test dataset are presented in Figure 3.6.

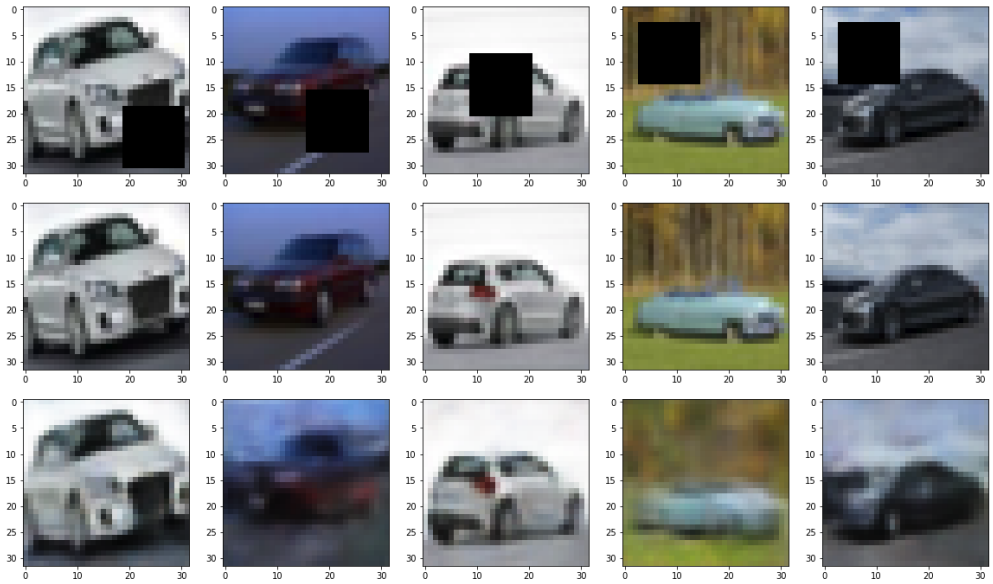

Figure 3.6: Row1: Original patches, Row2: Predicted patches, Row3: Filled images

In general, the CVAE model is not very popular for the image inpainting. As a generative model restricted to the latent multi-normal distributions, the VAE based networks are hard to adjust and improve. It happened a lot that the whole reconstructed image is a blur and adding or changing the layers could not effectively solve this problem. The more powerful generative adversarial network should be proposed for this challenge.

3.4 Generative Adversarial Networks

Though GAN is very popular and easy to implement in programming languages, there are several modifications specifically needed for the image inpainting task. A DCGAN model usually takes in the random noise and generates the real-life-like images. However, this could not guarantee the output images are similar to the incomplete images we want to deal with. To generate a plausible enough fill into the incomplete image, the loss function of the generator need not simply be the binary cross-entropy any more. It will be a weighted average of the binary cross-entropy between the labels and the mean squared errors between the unmasked areas. The new loss function of the generator will be:

$$L_G = \alpha \times BinaryCrossEntropy(labels, D(G(h))) + (1 - \alpha) \times MSE(IM \times I - G(h) \times I)$$

IM is the original complete image. I is the location map that indicate the masked area with 0s and the unmasked area with 1s. Additionally, one hyper-parameter α is introduced to balance how real the generated images are and the similarity between the generated images and the training images. α is a hyper-parameter that could not be learnt through the back-propagation. We need to adjust α for different networks to achieve the best performance.

3.4.1 Inspiration from the Globally and Locally Consistent Image Completion Design

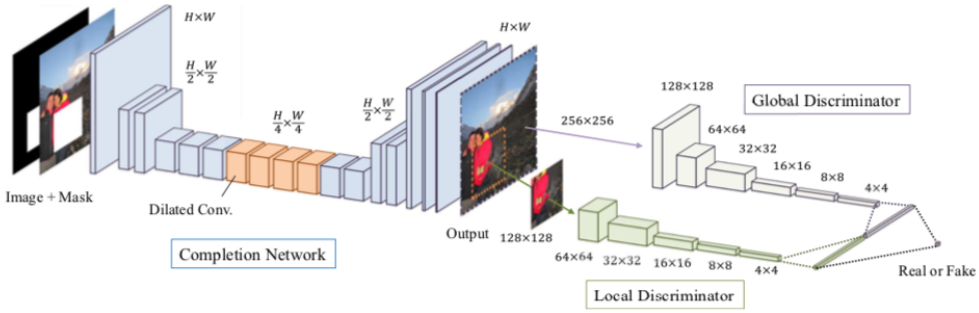

Figure 3.7: The structure of Globally and Locally Consistent Image Completion Networks

Figure 3.7 is from the paper[11] introducing the architecture of the globally and locally consistent image completion networks specifically for the image inpaiting. This model takes the

masked images as the input and generates the complete images first. Then two discriminators taking in the generated masked area and the generated unmasked area are trained to help the whole generated image real enough.The global-and-local model is much more reasonable than just using the random noise to generate the filled image. Since the CIFAR10 car images have very low resolutions, the networks I designed did not have the global discriminator.

According to the performance of the autoencoder model I used, I tried a small autoencoder structure for the completion network. It contains 4 convolution layers and 4 transposed convolution layers all with RELU activations. Each layer has strides number equal to 2 with the 'same' padding. The filter size of each layer is 5x5. The last layer of the completion network after the transposed convolution layer is a convolution layer with the TANH activation. GAN-like networks' generators usually output the digits in the range $[-1, 1]$ by the TANH activation. As a result, when inputing the real images to train the discriminator, we need to normalize the image digits in $[-1, 1]$ first.

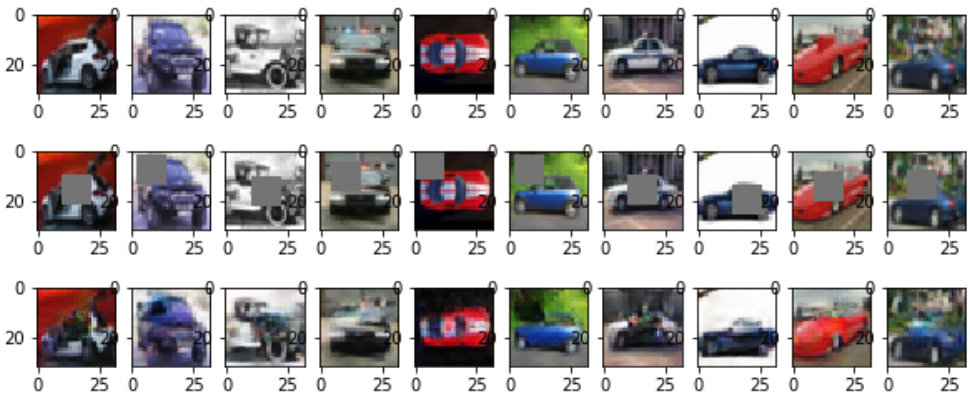

Figure 3.8: A sample of results by the autoencoder completion network

Figure 3.8 shows a sample of results generated from an autoencoder completion network. These reconstructions are not very good. Many of the masked areas are filled with lots of noise. Training generative adversarial networks is hard because we need to balance the strength of the generator and the discriminator. If the discriminator is too strong or trained too fast, the generator would hardly have any update. On the other hand, more parameters

or more complex model of the generator does not necessarily guarantee good results either. In my case, a direct convolution generator worked much better. The summary of the generator is in Table 3.7 and the summary of the discriminator is in Table 3.8.

Table 3.7: Generator Summary

Layers	Filter Size	Stride Number	Padding	Output Shape
CONV1 + RELU1	5x5	1	Same	32x32x64
CONV2 + RELU2	5x5	1	Same	32x32x64
CONV3 + RELU3	5x5	1	Same	32x32x64
CONV4 + RELU4	5x5	1	Same	32x32x64
CONV5 + RELU5	5x5	1	Same	32x32x64
CONV6 + RELU6	5x5	1	Same	32x32x64
CONV7 + Tanh	5x5	1	Same	32x32x3

Table 3.8: Discriminator Summary

Layers	Filter Size	Stride Number	Padding	Output Shape
CONV8 + RELU7	3x3	1	Zero	30x30x64
CONV9 + RELU8	3x3	2	Same	15x15x64
CONV10 + RELU9	3x3	2	Same	8x8x64
CONV11 + RELU10	3x3	2	Same	4x4x64
CONV12 + RELU11	3x3	2	Same	2x2x64
Flatten + Dropout				256
Dense + Sigmoid				1

Some good reconstructions and failed reconstructions are presented in the Figure 3.9 and Figure 3.10. The image completion GAN model having a relatively small number of parameters could give good outputs in short training time. And the idea of GAN extends the flexibility of the networks' structure. We can not only choose different kinds of neural network layers but also combine special design of the generator and discriminator. However, it is true than GAN-like model is still the most difficult method to adjust.

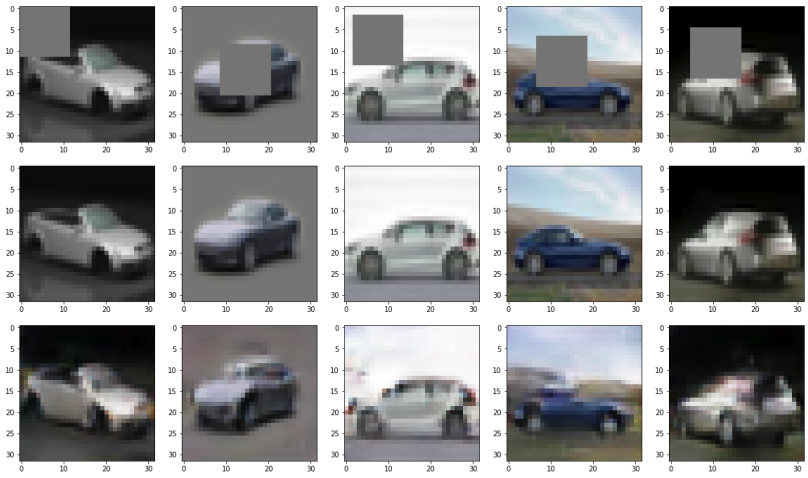

Figure 3.9: A sample of good reconstructions by a direct convolution generator

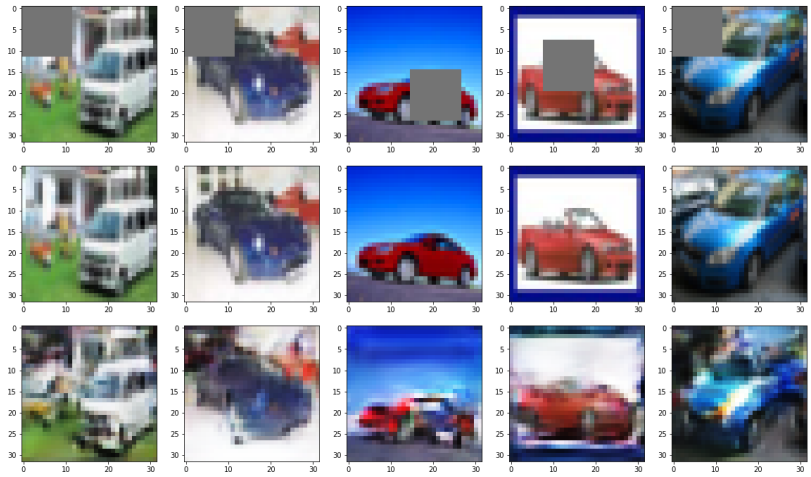

Figure 3.10: A sample of failed reconstructions by a direct convolution generator

CHAPTER 4

Future Discussion and Conclusion

Incomplete image completion is a very hard task according to all the experiments and research I did. Though some of the methods I used did give some good results, the CIFAR10 car images do not have resolutions high enough to be representative. To reconstruct a 128x128 mask in a 256x256 image is definitely harder than to reconstruct a 12x12 mask in a 32x32 image. Another very huge challenge is the irregularly shaped mask. One prime usage of the image completion is to remove the contents that people don't want in the image. These images taken by the smart phone or the camera usually have very high quality with very complicated context. As a result, very large training datasets and very deep neural networks will be needed if we want to solve this problem by deep learning methods.

One direct improvement for future study is to focus on the design of the loss function. The loss functions of supervised models mainly compares the outputs and the target images. One suggestion to improve the loss functions of the supervised models is to use an average with different weights on the masked inner contents and the boundaries of the masked area. We could even overlap the masked and unmasked areas a bit and give the overlapping boundaries larger weights in the loss to solve the blurriness problem. The loss functions of unsupervised models will be more complicated because these models need to consider different networks or the latent distributions. For instance, the loss function of the GAN model I used needs the hyper-parameter α to be adjusted for better results.

Economically, one efficient way to do image inpainting is to combine the traditional method and the deep learning method together. The traditional methods could fill in the contents

based on the nearby context. We could reconstruct some simple background of the images or the boundaries of the missing areas first and then predict the most important or most complicated part by the deep neural networks. This is also a suggestion for students with limited resources to undertake image inpainting projects.

REFERENCES

[1] Charles Burlin, Yoann Le Calonnec and Louis Duperier. Deep Image Inpainting. `http://cs231n.stanford.edu/reports/2017/pdfs/328.pdf`, Accessed: 2019-06-03.

[2] CS231n Convolutional Neural Networks for Visual Recognition. `http://cs231n.github.io/convolutional-networks/#conv`, Accessed: 2019-05-09.

[3] Christopher Sauer, Russell Kaplan and Alexander Lin. Neural Fill: Content Aware Image Fill with Generative Adversarial Neural Networks. `http://cs231n.stanford.edu/reports/2016/pdfs/209_Report.pdf`, Accessed: 2019-06-03.

[4] Diederik P Kingma and Max Welling. 2014. Auto-Encoding Variational Bayes. *arXiv preprint arXiv:1312.6114v10* (2014).

[5] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. 1988. Learning representations by back-propagating errors. In James A. Anderson and Edward Rosenfeld, editors, *Neurocomputing: foundations of research*, pages 696-699. MITP.

[6] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MITP, 2017.

[7] Goodfellow, Ian and Pouget-Abadie, Jean and Mirza, Mehdi and Xu, Bing and Warde-Farley, David and Ozair, Sherjil and Courville, Aaron and Bengio, Yoshua. 2014. Generative Adversarial Nets. In Z. Ghahramani and M. Welling and C. Cortes and N. D. Lawrence and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc.

[8] Guilin Liu, Fitsum A Reda, Kevin J Shih, Ting-Chun Wang, Andrew Tao, and Bryan Catanzaro. 2018. Image Inpainting for Irregular Holes Using Partial Convolutions. In Z. Ghahramani and M. Welling and C. Cortes and N. D. Lawrence and K. Q. Weinberger, editors, *arXiv preprint arXiv:1804.07723* (2018).

[9] Kamyar Nazeri, Eric Ng, Tony Joseph, Faisal Z. Qureshi, and Mehran Ebrahimi. 2019. EdgeConnect: Generative Image Inpainting with Adversarial Edge Learning. *arXiv preprint arXiv:1901.00212* (2019).

[10] Prakash Pandey. Deep Generative Models. `https://towardsdatascience.com/deep-generative-models-25ab2821afd3`, Accessed: 2019-05-21.

[11] Satoshi Iizuka, Edgar Simo-Serra, and Hiroshi Ishikawa. 2017. Globally and locally consistent image completion. In *ACM Trans. Graph. 36, 4, Article 107* (2017).