Single Image Super-Resolution

Yujing Song
Division of Science and Mathematics
University of Minnesota, Morris
Morris, Minnesota, USA 56267
songx827@morris.umn.edu

ABSTRACT

Super-Resolution (SR) of a single image is a classic problem in computer vision. The goal of image super-resolution is to produce a high-resolution image from a low-resolution image. This paper presents a popular model, super-resolution convolutional neural network (SRCNN), to solve this problem. This paper also examines an improvement to SRCNN using a methodology known as generative adversarial network (GAN) which is better at adding texture details to the high resolution output.

1. INTRODUCTION

Image super-resolution is a concept based on the human visual system. David Hubel, Torsten Wisel, the winners of 1981 Nobel Prize in medicine, found that information processing in the human visual system is hierarchical [8]. Using convolutional neural networks to deal with SR problems mimics the processing of the human visual system. Therefore, computer vision is a good applications of neural networks. Super-resolution is a classic application of computer vision, and tends to produce images that people find either realistic or aesthetically pleasing.

This paper looks into the working processes of super-resolution using convolutional neural networks (SRCNN) and the improvements that result from a modification to the process using a generative adversarial network known as SRGAN. Section 2 gives the background knowledges. Section 3 introduces each step for establishing a SRCNN model. Section 4 discusses, SRGAN, a recent popular method, that has some improvement over SRCNN. Section 5 explains the relationship between SRCNN and SRGAN.

2. BACKGROUND

In this section, we introduce some significant concepts related to understanding the use of convolutional neural networks for image problems, including classical neural networks, convolutional neural networks, and loss functions. We also discuss recognizing images' construction.

2.1 Images

Two concepts are important for understanding an image, one is channels, the other is pixels.

UMM CSci Senior Seminar Conference, November 2018 Morris, MN.

A color image usually is an RGB image, which means it is composed of red, green and blue components. If an image is split into its components, then we have three channels, one for each color. There are other ways, than RGB, to store color information for an image. Another common approach is YCbCr– which also uses 3 channels, but in this encoding Y represents luminance (brightness) while Cb and Cr are a more complicated way to represent colors (see [1] for details).

Pixels are the smallest unit of a digital image. For the images used by SRCNN and SRGAN these pixel values are numbers from 0 to 255. If we have an image with size $w \times l$, this means that the image is divided evenly into w rows of pixels, and each row has l pixels. Each channel has its own pixel value in every pixel position. Every pixel position in a channel corresponds to the same positions in the other channels. When these pixel values are put together, a specific color shows up in specific position in the image. In short, a pixel is a location in an image in an RGB image it takes 3 values (one for each channel) to indicate the color associated to that pixel.

Patches of an image means a part of an image, which contains the specific size of pixel values of an image. For example, the original image will be cut into as many 33×33 patches to set as inputs in the convolutional neural network.

Image upscaling is trying to express an image using more pixels. For example, if an image is expressed by 10×10 pixels in the beginning, after 2 times upscaling, it is expressed by 20×20 pixels. Since pixels are the smallest unit of an image, the size of each pixel stays the same, so the upscaled image is enlarged by the process of image upscaling. The popular two algorithms to upscale are bicubic interpolation and sparse-coding base method [1].

Bicubic interpolation is the most commonly used interpolation method in two-dimensional space. In this method, the value of the function at a specific point can be obtained by the weighted average of the nearest 16 sampling points in the rectangular grid, where two polynomial interpolation cubic functions are needed, one in each direction [4]. Sparse-coding method is trying to represent an image as a linear combination of patches, and to require only a few patches to represent the image [9].

These two methods are also the two classical models for images super-resolution. Similarly, image down-scaling is trying to express an image using fewer pixels.

Reconstructing an original image from a down-scaled version of that image is the goal of the convolutional neural network. The process of reconstructing image is the process that introduces high-frequency details, preferably ones

that are consistent with the image being upscaled. High-frequency means the frequency changes fast. The frequency in an image represent the difference between adjacent color blocks. Details in images can be described using terms like low-frequency or high-frequency. The first refers to properties that change slowly– for example in a picture of a grassy hill and a clear blue sky, the blue pixels that are part of the sky may have similar values that don't change quickly from pixel to pixel. However the pixels that make up the grass on the hill will have edges and shadows where the values of the pixels will change rapidly over short distances. The first type of detail I called low-frequency, and the second is called high-frequency. When an image is down-scaled, low-frequency details tend to be preserved, but high-frequency details are often lost, or blurred.

2.2 Classical Neural Network

Neural networks are a mathematical model that estimates the working process of human brains to recognize potential relationship within a set of data. Here in our case, the neural network is used as the most important part of a process that produces a high-resolution image from a low-resolution image.

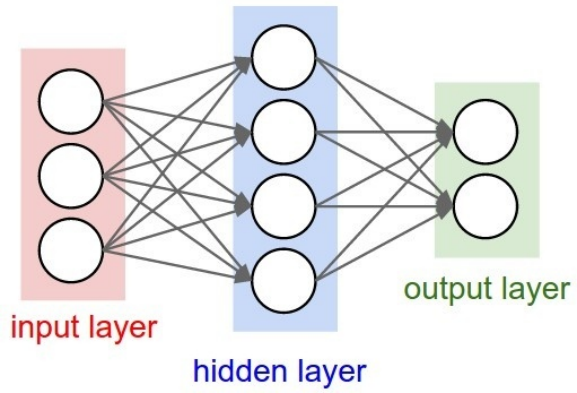

Figure 1: Basic structure of a neural network with one hidden layer.

Figure 1 illustrates the main components of any neural network. The circles are called nodes and represent numeric values. The edges between nodes also have a numeric value called a weight. The red area on the left is the input layer, the green area on the right is the output layer, and the blue one in the middle is the hidden layer. Input layer above has 3 nodes, output layer has 2 nodes, hidden layer has 4. Note that when designing a neural network, the number of nodes are usually fixed in the input layer and output layer, but hidden layer are free. That is to say, we can set one or more hidden layers, and two or more nodes for each hidden layer. The topology and arrows in the structure of neural network diagram represent the flow of values during the calculations that transform input to output. The point of the diagram is not the nodes, but the weights. These weights need training to determine their values. Training values of the weights will be mentioned in Section 2.2.4, here we focus on understanding the structure of a neural network. We call the network fully-connected neural network because for each node in any layer, it is connected to every node in the next layer.

For every pair of adjacent layers, the nodes in a previous layer, $\{x_1, x_2, ..., x_n\}$, pass their values to the next layer through the edges, each edge contain a weight $\{w\}$. Associated to each node is both a value and a function. The value in each node, except for the values in the input layer, are produced by a calculation explained below:

$$y = f(b + \sum_{i=1}^{n} x_i \cdot w_i)$$

The function associated to a node, known as the activation function, is part of the calculation that determines a nodes value. It is the function f in the equation above. In all our neural networks, all nodes in the same layer use the same activation function. The bias b in each hidden layer is an extra weight that does not connected to any previous layer. It is added after the previous calculating process to scale a linear output up and down.

There are three types of input and output in this paper. The input will be an image and the output will be an image in the convolutional neural network (see Section 3 for details on how images are represented as the kind of numeric values that can be represented by the nodes in Figure 1). When the neural network is used in the generative network of GAN (see Section 4), the input will be a set of random numbers and the output will be an image. Whereas when the neural network is used in the discriminator network of GAN, the input will be an image and the output will be a probability (see Section 4 for details).

2.2.1 *Activation function*

The activation function is used to introduce the non-linearity in the neural network. Though there are a lot of activation functions, rectified linear unit (ReLU) is the only activation function we will use in this paper. The definition of ReLU is:

$$\mathrm{R}(z) = \max(0, z)$$

2.2.2 *Training for neural network*

Before introducing the training processes of neural networks, we need to distinguish supervised methods and unsupervised methods. Supervised methods are "learning algorithms that learn to associate some input with some output, given a training set of examples of inputs and outputs" [2]. Unsupervised methods are those "that experience only 'features' but not a supervision signal" and try to group inputs based on their features [2]. The neural network used in CNN is a supervised method because we provide a training set for input, that are the low-resolution images, and outputs that are our original images.

In our case these will be a set of images that are down-scaled before being used as input to the SR-process. Down-scaling removes details from the image. The neural network is trained to reintroduce these details as its output. The differences between the output of the neural network and the original image are measured using a loss function (Section 2.2.3). After an image is generated the weights in the neural network are adjusted in such a way that the loss function is made smaller. This process is called stochastic gradient descent with back-propagation (details in Section 2.2.4). Multiple images are used in the training process. The goal is not for the neural network to become good at recreating one single image, but to do a good job adding details to any

image.

2.2.3 Loss function

The loss function is a method for evaluating how closely the actual output of a neural network matches the desired output. Two loss functions are used in this paper: Peak signal-to-noise ratio (PSNR) and mean square error (MSE).

MSE calculates the deviation from the original values by averaging the squared difference between the estimated values, $\widehat{X_i}$ and the original values X_i:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(X_i - \widehat{X_i})^2$$

The lower the MSE value, the better the weights that have been chosen.

If we say MSE is used to judge the quality of a specific technique, PSNR is used to evaluate the quality of reconstruction of these techniques. It is an engineering term for the "ratio between the maximum possible power of a signal and the power of corrupting noise that affects the fidelity of its representation" [6]. The formula expression is the following:

$$\text{PSNR} = 10\log_{10}\left(\frac{{\max_I}^2}{\text{MSE}}\right)$$

where $\max_I$ is the maximum possible pixel value of the image. When the pixels are represented by 8-bits per sample, its value is 255 [6]. Different with MSE, the higher value of PSNR get, the better super-resolution results a method will get.

2.2.4 Stochastic gradient descent

Since MSE evaluates the difference between the true image values and our neural networks predicted values, we want this difference as little as possible, which means we need to minimize the error. Stochastic gradient descent with back-propagation is a technique for modifying the weights in the neural network to realize this goal. These gradients will lead us to a minimum in our loss function.

In the classic neural network, the values are calculated from left to right, from input layer to output layer. But we want to minimize the mean square error by adjusting the values of weights, we need to update weights layer by layer from the output layer back to the input layer. This backward process is called back-propagation. The method that we used in back-propagation process is called stochastic gradient descent. It is called stochastic is because the sample weight value that will be updated is chosen randomly. The formula is expressed as following:

$$W_{j+1} = W_j + \eta \cdot \frac{\partial L}{\partial W_j}$$

where L is a loss function, and it is MSE in our case. j represents the iteration times. W_{j+1} and W_j are the sets of all weights at iteration j. W_{j+1} is the new weight vector, it is set to the original weight vector W_j plus the learning rate η times a partial derivative $\frac{\partial L}{\partial W_j}$.

Learning rate, is a parameter that influences how the training occurs. The value of η controls how much $\frac{\partial L}{\partial W_j}$ changes the weight. The partial derivative is the fraction in the equation that displays how much the loss function will change for every unit of original weight has changed. Note that if a weight is near its optimal value then changing the weight does not have much influence on the loss function so $\frac{\partial L}{\partial W_j}$ is near 0.

2.3 Convolutional Neural Network

The architecture of a neural network refers to the way in which the different layers are connected to each other. Two common ways to connect layers produce convolutional layers and fully connected layers.

2.3.1 Convolutional layer

The core of a convolutional neural network is its convolutional layer. To understand the details, three important concepts are defined: kernel, feature maps, and convolutional calculation.

Features represent some specific characteristics in an image. Extracting features, for example, can extract edges of an image. Kernels in convolutional network represent a feature extracting method, which constrains a set of weight. A set of kernels form a filter. The number of filters in a convolutional layer is called the depth of this layer. Like each node has its own weight in neural network, each channel has its own kernel in each filter. The kernel is a small array of values (in our example in Figure 2 it is 3x3). These values are treated like a sliding window and moved over the surface of the image. A sliding window is used in each channels. A sliding window usually starts from the left top of the original channel, then doing convolutional calculation with its kernel. The convolutional calculation will be explained in next paragraph. After the convolutions, we slide our window right-side with one pixel length. Repeat the convolutional process and slide the window again, until the end of this line. Then we move to the second row and continue to do so until the window scans all the pixels in the original channels. The three sliding windows associated to a filter move across each channel at the same time and in the same fashion. At each different location a new convolutional calculation is performed.

The convolutional results are stored in the feature map. That is to say, feature map is the collection of outputs for a given filter. In our description above the sliding window was moved one pixel at a time, but often the window is moved more than one pixel– the number of pixels by which the window is moved is called the stride. In our example the stride is one, but it could be any reasonable number, it depends on the question.

When it comes to the convolutional process, we start with one channel situation. Fig.2 shows an example of convoluting with a single channel. From the left to the right, we see the original channel, the kernel, and the feature map. Each location of the kernel K over the image I results in a single value in the feature map. We denote the feature map as $I * K$ where $*$ for the convolution. In Figure 2 the value of 4 (highlighted in green) is the result of multiplying the values in red from image I with the corresponding values of the kernel K and adding the results. Still start with top left of the sliding window, pixel value in that position is 1. The weight in the corresponding position in the kernel is also 1, we calculate their product which is 1. Then we look at 0 in the second column first row in the sliding window, multiplying this with the corresponding value in kernel, got 0. Continuously to do so until we get all the production results

for each position. Summing up all these results, we got 4. Notice the similarity to the process of calculating the value of a node in the neural network process outlined in Section 2.2 This occurs for each position of K as it is slid across the image I.

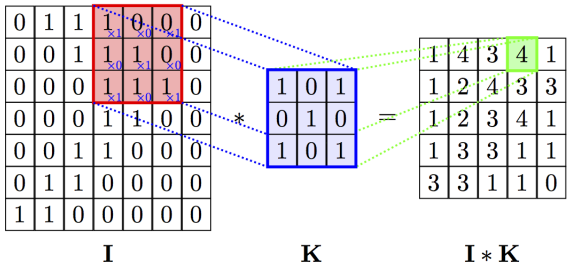

Figure 2: The convolutional process example. Left is the original input channel I, the middle part is a convolutional kernel K, and right side $I * K$ is the feature map.

When calculating convolution in multi-channel input, say we have three channels C_1, C_2, C_3, for an RGB image, and we know the convolutional kernels corresponding to each channel. Suppose two filters (F_1 and F_2) and their biases (b_1 and b_2) are set. Let's call the kernels K_{mn}, where m stands for the index of the filter and n stands for the index of the channel. For example, K_{21} means it is the kernel for C_1 in F_2. So the outcome will have two feature maps. The first feature map is obtained by $C_1 * K_{11} + C_2 * K_{12} + C_3 * K_{13} + b_1$. The second feature map is then $C_1 * K_{21} + C_2 * K_{22} + C_3 * K_{23} + b_2$, where $*$ represents the convolutional operation.

2.3.2 *Fully connected layer*

Fully connected layer acts like a generator in our case, it is trying to collect all the features in the previous layers. The fully connected layer in this case will be used in solving SR problem by CNN (details in Section 3.2), and used in the generator network of GAN (details in Section 4).

3. SRCNN

This section discusses how convolutional neural networks are applied to the SR problem, the proposed model is called SRCNN. Given a single low-resolution image, the final high-resolution image is the output resulting from three stages of operation: patch extraction and representation (section 3.1), non-linear mapping (section 3.2), and reconstruction (section 3.3). An overview of the procedure can be seen in Figure 3.

Before we get started there is some pre-processing that must occur. To the single low-resolution image, we upscale it to the desired output resolution by first using a conventional technique such as bicubic interpolation. This upscaled image will be used as an input for our SRCNN. Denote the upscaled low-resolution image as input Y. We are trying to find $F(Y)$ that is as similar as possible to the true high-resolution image X. So $F(Y)$ is what we obtain through SRCNN from Y, X is the original image that set as a label for adjusting weights in CNN.

3.1 Patch extraction and representation

Suppose our input image Y has size $A \times B$, where A, B represent the length and width respectively. Y is convolved with kernels with size $f_1 \times f_1$. The depth of this layer is n_1, that is to say, n_1 filters are set in this layer. The structure of this layer looks like what we showed in Figure 1, input layer contain the input image, the n_1 hidden layers are n_1 filters, and the output layer contains the feature maps. ReLU is applied on the filter response, so we have the following equation:

$$F_1(Y) = \max(0, W_1 * Y + B_1),$$

where W_1 and B_1 stand for the kernels in filters and the biases respectively, and $*$ is the convolutional operation. The value in W_i, as a kernel, represents a large number of edges. All of which share the same weights, and training occurs on this small, shared, set of weights, so it's much more efficient to train the weights in the kernel then to train a large collection of independent weights. After this step, we have n_1 feature maps with size $(A - f_1 + 1) \times (B - f_1 + 1)$, and have removed all the negative values. [1]

The feature maps that are produced by this layer are what we mean by patch extraction and representation. The input image is the patch because it is not the original whole image but a firstly down-scaled then up-scaled sub-image, which we will introduce in Section 3.4. It is extracted by n_1 kernels, and the representations of these extractions form the feature maps.

3.2 Non-linear mapping

The second step converts the n_1-dimensional features into n_2 dimensions by setting n_2 filters and kernels with size $f_2 \times f_2$ in this layer. The operation of the second layer is:

$$F_2(Y) = \max(0, W_2 * F_1(Y) + B_2)$$

W_2 and B_2 are the kernels and biases in the second layer, $F_1(Y)$ is the feature maps from the last layer. Again, the size of feature maps in $F_1(Y)$ is $(A - f_1 + 1) \times (B - f_1 + 1)$. In order to express easily, we set $A_1 = (A - f_1 + 1)$ and $B_1 = (B - f_1 + 1)$. W_2 has n_2 filters and B_2 is n_2-dimensional. The output we get are n_2 feature maps with size $(A_1 - f_2 + 1) \times (B_1 - f_2 + 1)$. Similarly, we set $(A_1 - f_2 + 1)$ as A_2, $(B_1 - f_2 + 1)$ as B_2.

Researchers in *Image Super-Resolution Using Deep Convolutional Networks* [1] set $f_2 = 1$ in this layer. So we have $A_2 = A_1, B_2 = B_1$. It is equal to fully connected layer but utilized 1×1 filters, which makes every position of the last layer's feature maps share their parameters.

3.3 Reconstruction

The operation applied by the third layer produces the final high-resolution image:

$$F(Y) = W_3 * F_2(Y) + B_3$$

W_3 responds to a filter that has n_3 kernels of size $f_3 \times f_3$. B_3 is the bias for this reconstruction layer. This layer is from the $n_2 \times 1$ vector produced by the previous layer. To rebuild $f_3 \times f_3$ image, the final high-resolution image is realized by taking the averages of these overlapped image patches.

3.4 Training

The images used in training are not the original whole pictures. Instead, 33×33 sub-images are extracted from the original images with stride 14. The original $395,909$ images are from the "ILSVRC 2013 ImageNet detection training partition" [1]. So over 5 million of sub-images are down-scaled as input in the training set. These original images

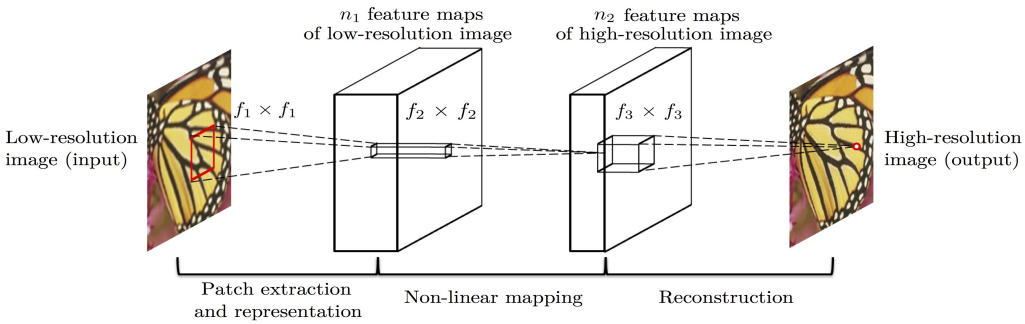

Figure 3: "Given a low resolution image Y, the first convolutional layer of the SRCNN extracts a set of feature maps. The second layer maps these feature maps non-linearly to high-resolution patch representations. The last layer combines the predictions within a spatial neighbourhood to produce the final high-resolution image $F(Y)$"[1]

X are down-scaled to low-resolution samples first, and then upscaled to the target resolution using bicubic interpolation.

Settings used by the researchers [1] were $f_1 = 9, f_2 = 1, f_3 = 5, n_1 = 64, n_2 = 32$. These values were determined by experiments and chosen to be a good balance between high-resolution quality and efficiency. MSE is used as loss function $L(\Theta)$, $\Theta = \{W_1, W_2, W_3, B_1, B_2, B_3\}$.:

$$L(\Theta) = \frac{1}{n}\sum_{i=1}^{n}(F(Y_i;\Theta) - X_i)^2$$

where n is the number of training samples, and X is the original image. The high-resolution images $F(Y_i;\Theta)$ are created by taking low-resolution images Y as input and go through mapping function $F(\Theta)$.

Stochastic gradient descent with back-propagation is used to minimize the loss function:

$$\Delta_{i+1} = 0.9 \cdot \Delta_i + \eta \cdot \frac{\partial L}{\partial {W_i}^l},$$
$${W_{i+1}}^l = {W_i}^l + \Delta_{i+1}$$

where $l \in \{1, 2, 3\}$ is the index of the channels, i is the iterations, η is the learning rate. Δ is a parameter that its initial number is $\Delta_0 = 0$.

The weights in filters are initialized by random selection. The weights will be updated by hundred of millions times via back-propagation. The learning rate is 10^{-4} for the first two layers but is 10^{-5} for the third layer.

3.5 Advantages

When the neural network is sufficiently trained, the quality of the SR image is measured using the Peak Signal-To-Noise Ratio (PSNR) discussed in Section 2.2.3. The higher of PSNR, the closer the SR image is to the original. The following table shows SRCNN is achieving better results than traditional methods in different magnifications [1]:

Eval. Mat	Scale	Bicubic	SC	SRCNN
PSNR	2	33.66	-	**36.66**
	3	30.39	31.42	**32.75**
	4	28.42	-	**30.49**

Here "Bicubic" stands for bicubic interpolation, "SC" stands for sparse-coding base method. "Scale" means the factor of upscaling.

The table implies when upscaled two to four times of low-resolution images, SRCNN had the best PSNR values in comparison to bicubic interpolation and sparse coding base methods. That is to say, in mathematical perspective, the high-resolution images through SRCNN are closer to the original images.

4. GAN

Study [5], uses generative adversarial network (GAN) to address the SR problem, the method is abbreviated as SRGAN. Based on the article, although the results of the SRCNN process can have high PSNR values when MSE is used as the loss function, the resulting super-resolution images usually lose high-frequency details.

SRGAN is a neural network architecture composed of two competing networks, generator network and discriminator network.

The generator network creates an image by taking a set of random numbers Z. Therefore, we have two datasets in GAN, one is the true data set and the other is the fake dataset. The true data set contains original images while the fake dataset is created by the generator network.

The discriminator network is a classifier that determines if a given image is a real image from the dataset or an image made by generator network. Discriminator network is a binary classifier in the form of a CNN where the input is an image and the output is a probability. If the probability is greater than 0.5, the input is classified as belonging to the true dataset while a probability less than 0.5 classifies the input as belonging to the fake dataset.

The aim for the generator is to create a high-resolution image that will trick the discriminator network into classifying it as true. When the output of the discriminator network is close to 0.5, it means that the discriminator cannot distinguish the difference. At this point the high-resolution image is considered to be good enough.

4.1 Training for SRGAN

True images from the original set are labeled as 1, and generated images created by the generator of SRGAN are labeled as 0. The job for the discriminator network is to classify the input image as having a label 0 or 1. The training

processes is similar to what we have introduced in CNN (Section 2.3).

Training a generator network also requires a discriminator network. This discriminator network is put after the generator network so that we can have errors to calculate the loss function. Here the labels are not important but the errors are important. The significant operation is when training the generator network, the parameters of the discriminator network should remain unchanged so that in the training process only the weights of generator network are updated. [5]

In contrast to the training process used by the generator network, which must have a discriminator network in order to calculate the error, the training discriminator network only need the discriminator network itself.

Z is a set of random numbers, and $D(X)$ represents the probability that X came from the data. We train the weights of D, W_D, to maximize the probability of assigning the correct label to both training examples and samples from G. We simultaneously train the weight of G, W_G, to minimize $\log(1 - D(G(Z)))$. [3]

The value function is:

$$\min_{W_G} \max_{W_D}([\log D(X)] + [\log(1 - D(G(Z)))])$$

The aim function above can be divided into two parts, one is for discriminator network and the other is for the generator network.

Optimizing the discriminator does not directly involve the generator network, so we only care about maximum $\log D(X)$. X is the images in the true data set, for all the images from the true dataset, the result of discriminator network $D(X)$ should as close to 1 as possible. So we want to find the weights of discriminator network when the result of $D(X)$ is maximum.

When training the generator network, minimum $\log(1 - D(G(Z)))$ is the goal. $G(Z)$ is equivalent to the existed fake samples. Since the samples created by $G(Z)$ are not the real ones, we want $D(G(Z))$ closer to 0.

4.2 Relationship with conventional neural network

The PSNR for SRGAN is less than SRCNN, but the high-resolution image from SRGAN is visually closer to the original image, as showed in Fig.4. The car and tree in SRGAN looks more clear than that in SRCNN, though the left has PSNR 24.83db and right side SRGAN has 23.36db [7].

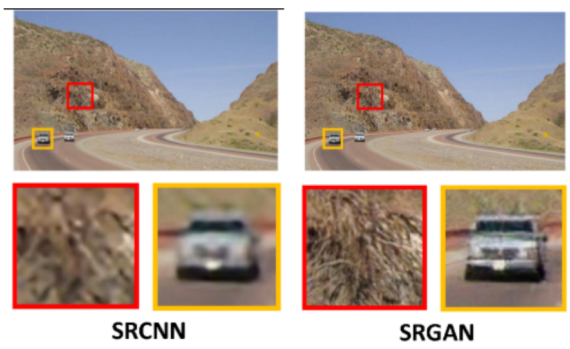

Figure 4: The results of high-resolution image from SRCNN and SRGAN

SRCNN used MSE as loss function, which ensured a high value for the PSNR, but a lack of high-frequency information in the image, so it over-smooths the image in a way that is obvious to people. However, SRGAN insists that the created high-resolution images should be closer to the true original images not only in the pixel values but also in the abstract characteristics. Therefore, it uses a discriminator network to judge the created images. If the discriminator network cannot distinguish if a high-resolution image is one of the true original ones, people visually cannot distinguish as well.

5. CONCLUSION

This paper has introduced two methods to solve SR problems. SRCNN is a forerunner of deep learning used in SR reconstruction. SRCNN's network structure is very simple, using only three convolutional layers, but it improves the efficiency of the SR process and produces better PSNR than traditional methods. However, SRCNN is not perfect, it inspired many of other methods to improve that. Every work is a great step forward but we chose SRGAN in Section 4 in this paper because it not only focus on increase the value of PSNR, but try to enhance more details in the images to improve human visual perception.

6. REFERENCES

[1] C. Dong, K. He, C. Change Loy, and X. Tang. Image super-resolution using deep convolutional networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38:30–52, February 2016.

[2] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press., Cambridge, MA, USA, 2016.

[3] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. *NIPS'14 Proceedings of the 27th International Conference on Neural Information Processing Systems*, 2:2672–2680.

[4] R. Keys. Cubic convolution interpolation for digital image processing. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 29:1153 – 1160, December 1981.

[5] C. Ledig, L. Theis, F. Huszar, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, and W. Shi. Photo-realistic single image super-resolution using a generative adversarial network. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, November 2017.

[6] A. T. Nasrabadi, M. A. Shirsavar, A. Ebrahimi, and M. Ghanbari. Investigating the PSNR calculation methods for video sequences with source and channel distortions. *2014 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting*, August 2014.

[7] X. Wang, K. Yu, C. Dong, and C. C. Loy. Recovering realistic texture in image super-resolution by deep spatial feature transform.

[8] T. Wiesel. The neural basis of visual perception.

[9] J. Yang, J. Wright, T. S. Huang, and Y. Ma. Image super-resolution via sparse representation. *IEEE Transactions on Image Processing*, 19:2861 – 2873, November 2010.