Image super-resolution reconstruction SRCNN

Name: Tong Xu

Supervisor: Zhicheng Zhang

Time: 2020/10/19

Table of Contents

1. Overview of image super-resolution reconstruction……………………………………………..3

1.1 Introduction..3

1.2 Application area 4

1.3 Related works..5

1.3.1 Traditional super-resolution reconstruction algorithm...5

1.3.2 Super-resolution reconstruction algorithm based on machine learning.................6

1.3.3 PSNR and SSIM: two commonly used indicators..6

2. SRCNN algorithm and Tensorflow implementation 7

2.1 SRCNN algorithm...7

2.2 Tensorflow implementation...9

2.2.1 Operating environment 9

2.2.2 Code evaluation 9

2.2.3 Experimental results..14

3. Conclusion 17

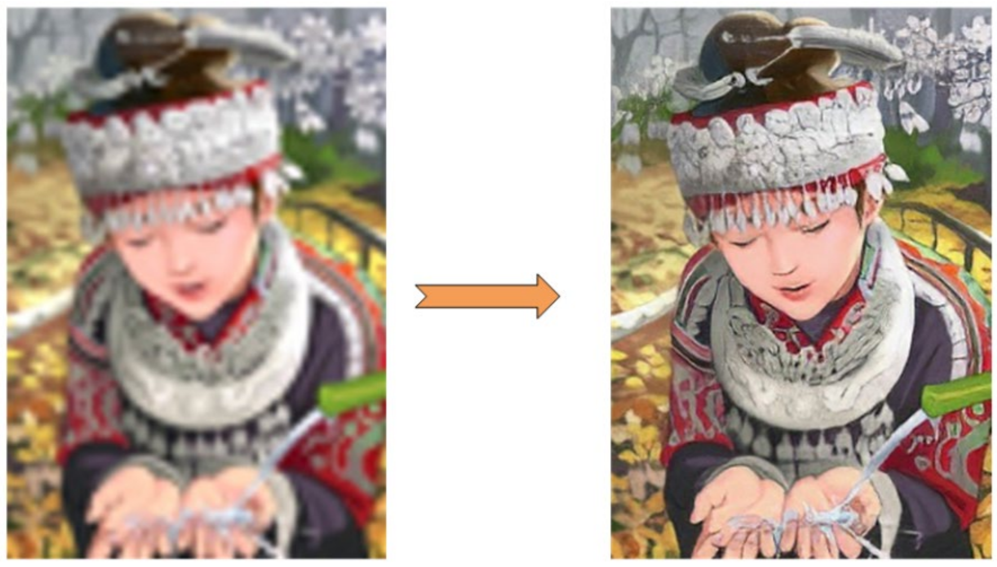
1. Overview of image super-resolution reconstruction

1.1 Introduction

Image resolution is a set of performance parameters used to evaluate the abundance of detailed information contained in an image, including time resolution, spatial resolution and color scale resolution, etc., which reflect the ability of the imaging system to actually reflect the detailed information of the object. Compared with low-resolution images, high-resolution images usually contain greater pixel density, richer texture details, and higher reliability. But in reality, due to the constraints of many factors such as acquisition equipment and environment, network transmission media and bandwidth, and image degradation model itself, we usually cannot directly obtain ideal high-resolution images with edge sharpening and no block blur. The most direct way to improve the image resolution is to improve the optical hardware in the acquisition system. However, because the manufacturing process is difficult to greatly improve and the manufacturing cost is very high, it is often too expensive to physically solve the problem of low image resolution. Therefore, from the perspective of software and algorithms, the technology to achieve image super-resolution reconstruction has become a hot research topic in many fields such as image processing and computer vision.

Image super-resolution reconstruction technology refers to the restoration of a given low-resolution image into a corresponding high-resolution image through a specific algorithm. Specifically, image super-resolution reconstruction technology refers to the use of relevant knowledge in the fields of digital image processing, computer vision, etc., through specific algorithms and processing procedures, to reconstruct a high-resolution image from a given low-resolution image the process of. It aims at overcoming or compensating the problems of blurred image, low quality, and insignificant region of interest due to the limitations of the image acquisition system or the acquisition environment itself.

Super-resolution reconstruction is to transform a small-size image into a large-size image, making the image clearer. The specific effect is shown in the figure below:



(a) Original low resolution image (b) Picture after super-resolution reconstruction

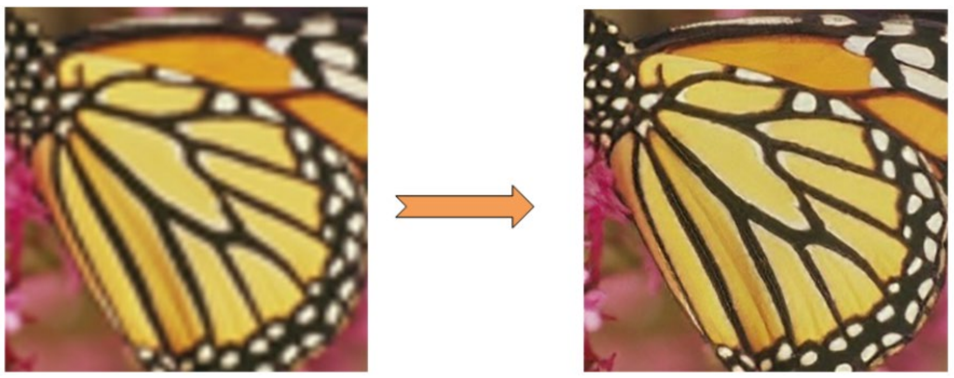

(a) Original low resolution image (b) Picture after super-resolution reconstruction

Figure 1. Samples of Image super-resolution reconstruction

It can be seen that through a specific super-resolution reconstruction algorithm, the originally blurred image becomes clear. Readers may wonder, isn't it enough to "stretch" low-resolution images directly? The answer is yes, but the effect is not good. The traditional "stretching" algorithm mainly uses the nearest neighbor search method, that is, each pixel in the low-resolution image is reconstructed by the nearest neighbor search or nearest neighbor interpolation. This manual setting method only considers the part and cannot meet the special circumstances of each pixel. It is difficult to recover the original detailed information of the low-resolution image. Therefore, a series of effective super-resolution reconstruction algorithms have begun to be proposed by research scholars, and the reconstruction ability has been continuously strengthened. Currently, relying on deep learning technology, image super-resolution reconstruction has achieved extraordinary results, and the effect is becoming more and more realistic and clarity.

1.2 Application area

In 1955, Toraldo Di Francia defined the concept of super-resolution for the first time in the field of optical imaging. It mainly refers to the process of recovering data information beyond diffraction limit by using optical related knowledge. Around 1964, Harris and Goodman first proposed the concept of image super-resolution, which mainly refers to the process of synthesizing a single frame image with more detailed information by extrapolating the spectrum. In 1984, on the basis of predecessors, Tsai and Huang first proposed the method of using multi frame low resolution images to reconstruct high-resolution images. After that, super-resolution reconstruction technology began to attract extensive attention and research in academia and industry.

Image super-resolution reconstruction technology has a wide range of applications and research significance in many fields. It mainly includes:

(1) Image compression

In video conference and other occasions with high real-time requirements, we can compress the image before transmission, wait for the transmission, and then decode it by the receiver, and recover the original image sequence by super-resolution reconstruction technology, which greatly reduces the storage space and transmission bandwidth.

(2) Medical imaging

Super resolution reconstruction of medical images can reduce the requirements of imaging environment without increasing the cost of high-resolution imaging technology. Through the restoration of clear medical images, accurate detection of pathological cells can be realized, which

is helpful for doctors to make better diagnosis of patients' condition.
(3) Remote sensing imaging

The development of high-resolution remote sensing satellite has the characteristics of long time-consuming, high price and complex process. Therefore, researchers have introduced image super-resolution reconstruction technology into this field, trying to solve the challenge of difficult acquisition of high-resolution remote sensing imaging, so as to improve the resolution of observation image without changing the detection system itself.
(4) Public Security

The video collected by the monitoring equipment in public places is often affected by the weather, distance and other factors, and there are problems such as blurred image and low resolution. Through the super-resolution reconstruction of the collected video, we can recover the license plate number, clear face and other important information for the investigators, and provide the necessary clues for the case detection.
(5) Video perception

Image super-resolution reconstruction technology can enhance the quality of video, improve the quality of video, and enhance the user's visual experience.

1.3 Related works

According to the classification of time and effect, the super-resolution reconstruction algorithm can be divided into traditional algorithm and machine learning algorithm.

1.3.1 Traditional super-resolution reconstruction algorithm

Traditional super-resolution reconstruction algorithms mainly rely on the basic digital image processing technology for reconstruction. The common types are as follows:
(1) Super-resolution reconstruction based on interpolation

Based on the interpolation method, each pixel in the image is regarded as a point on the image plane, so the estimation of super-resolution image can be regarded as the process of fitting the unknown pixel information on the plane by using the known pixel information, which is usually completed by a predefined variable function or interpolation kernel. The method based on interpolation is simple and easy to understand, but it also has some obvious defects.

First of all, it assumes that the change of pixel gray value is a continuous and smooth process, but this assumption is not completely true. Secondly, in the reconstruction process, only a pre-defined transformation function is used to calculate the super-resolution image without considering the degradation model of the image, which often leads to blurred and jagged images. The common interpolation methods include nearest neighbor interpolation, bilinear interpolation and bicubic interpolation.
(2) Super resolution reconstruction based on degradation model

Based on the degradation model of image, this method assumes that the high-resolution image is obtained by proper motion transformation, blur and noise. In this method, the key information of low-resolution image is extracted, and the prior knowledge of unknown super-resolution image is combined to constrain the generation of super-resolution image. The common methods include iterative back projection method, convex set projection method and maximum a posteriori probability method.
(3) Learning-based super resolution reconstruction

The learning-based method uses a large number of training data to learn the corresponding relationship between low-resolution image and high-resolution image, and then predicts the high-resolution image corresponding to the low-resolution image according to the mapping relationship learned, so as to realize the image super-resolution reconstruction process. The common learning-based methods include manifold learning and sparse coding.

1.3.2 Super-resolution reconstruction algorithm based on machine learning

Machine learning is an important branch of artificial intelligence, and deep learning is the most important algorithm in machine learning. It aims to extract the high-level abstract features of data through multi-level nonlinear transformation, and learn the potential distribution law of data, so as to obtain the ability to make reasonable judgment or prediction for new data. With the continuous development of artificial intelligence and computer hardware, Hinton et al. Proposed the concept of deep learning in 2006, which aims to extract high-level abstract features of data by using multi-level nonlinear transformation. With its powerful fitting ability, deep learning has begun to emerge in various fields, especially in the field of image and vision, convolution neural network is very different, which makes more and more researchers try to introduce deep learning into super-resolution reconstruction.

In 2014, Dong et al. first applied deep learning to the field of image super-resolution reconstruction. They used a three-layer convolutional neural network to learn the mapping relationship between low-resolution images and high-resolution images. Since then, a wave of deep learning has been set off in the field of super-resolution reconstruction rate. Their designed network model is named SRCNN (Super Resolution Convolutional Neural Network).

1.3.3 PSNR and SSIM: two commonly used indicators

PSNR (peak signal to noise ratio) and SSIM (structure similarity index) are commonly used to evaluate the quality of Super-resolution reconstruction. The higher the two values, the closer the pixel value of reconstruction results is to the standard.

(1) PSNR:

MSE is the mean square error of the current image X and the reference image Y. H and W are the height and width of the image respectively.

$$MSE = \frac{1}{H \times W} \sum_{i=1}^{H} \sum_{j=1}^{W} (X(i,j) - Y(i,j))^2$$

The unit of PSNR is DB. The larger the value, the smaller the distortion. n is the number of bits per pixel, and the general gray image is 8, that is, the pixel gray scale number is 256.

$$PSNR = 10log_{10}(\frac{(2^n - 1)^2}{MSE})$$

(2) SSIM:

SSIM reflects the similarity between the two images, and the value is [0,1]. The higher the SSIM value, the more similar. SSIM calculation method of two images X and Y:

$$SSIM(x,y) = [l(x,y)]^{\alpha} \cdot [c(x,y)]^{\beta} \cdot [s(x,y)]^{\gamma}$$

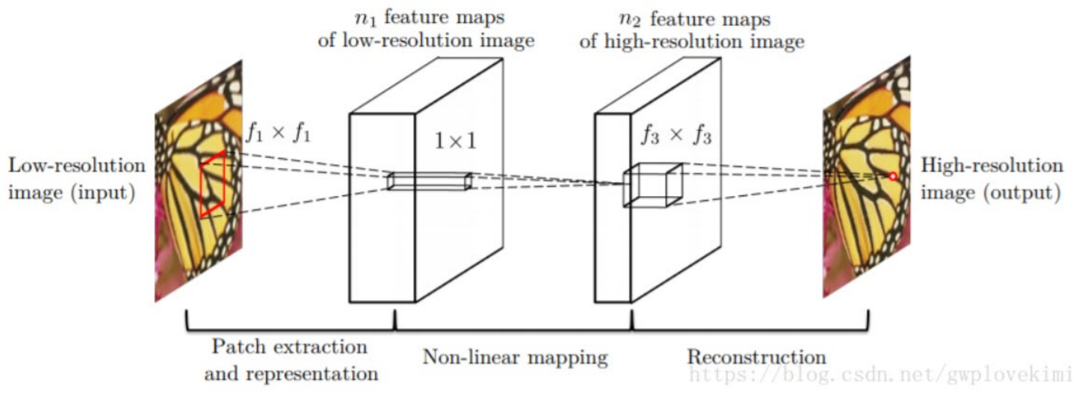
where $l(x,y) = \frac{2\mu_x \mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1}$, $c(x,y) = \frac{2\sigma_x \sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2}$ and $s(x,y) = \frac{\sigma_{xy} + C_3}{\sigma_x \sigma_y + C_3}$. And also,

$C_1 = (K_1 L)^2$, $C_2 = (K_2 L)^2$. Usually, we will denote $K_1 = 0.01,\ K_2 = 0.03$.

2. SRCNN algorithm and Tensorflow implementation

2.1 SRCNN algorithm

SRCNN is the first work of deep learning in image super-resolution reconstruction. Therefore, this paper will take SRCNN as the main line, explain the basic principle and algorithm implementation of SRCNN, and use Tensorflow deep learning framework to complete the reproduction of the above algorithm. SRCNN uses convolutional neural network to realize the end-to-end mapping between low-resolution and high-resolution images. The network structure is shown in the following figure:



n_1 feature maps of low-resolution image
n_2 feature maps of high-resolution image
$f_1 \times f_1$
1×1
$f_3 \times f_3$
Low-resolution image (input)
High-resolution image (output)
Patch extraction and representation
Non-linear mapping
Reconstruction
https://blog.csdn.net/gwplovekimi

Figure 2. SRCNN network structure

For a low-resolution image, bicubic interpolation is used to enlarge the low-resolution image to the target size, and then the nonlinear mapping is fitted by three-layer convolution network. Finally, the high-resolution image is output. In this paper, the author interprets the structure of three-layer convolution into three steps: extraction and representation of image blocks, nonlinear mapping of features and final reconstruction. The convolution kernels used in the three convolution layers are divided into 9 × 9, 1 × 1 and 5 × 5. The numbers of output features of the first two convolution cores are 64 and 32 respectively. Using mean square error (MSE) as the loss function is beneficial to obtain higher PSNR. The process of SRCNN is as follows:

Step1: Patch extraction and representation. Firstly, the low-resolution image is interpolated and magnified to the target size by bicubic interpolation. At this time, the image enlarged to the target size is still called low resolution image, which is the input in the image. Image blocks are extracted from low resolution input images to form high-dimensional feature maps.

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

where W_1 and B_1 are hyperparameter. We will utilize the Rectified Linear Unit (ReLu) as activation function.

Step 2: Non-linear mapping. The first layer convolution: the size of convolution kernel is 9 × 9 (F1 × F1), the number of convolution kernels is 64 (N1), and 64 characteristic graphs are output. The second layer convolution is that the size of convolution kernel is 1 × 1 (F2 × F2), the number of convolution kernels is 32 (N2), and 32 characteristic graphs are output.

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

where W_2 is N1 × 1 × 1 × N2. In this layer, 1 × 1 convolution (single convolutional layer) is adopted to compress the depth of feature map, and also play a role of nonlinear mapping.

Step 3: Reconstruction. The third layer convolution: the size of convolution kernel is 5 × 5 (F3 × F3), the number of convolution kernels is 1 (N3), and the output of a feature image is the final reconstruction of high-resolution image.

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

The structure of the model parameters is as follows:

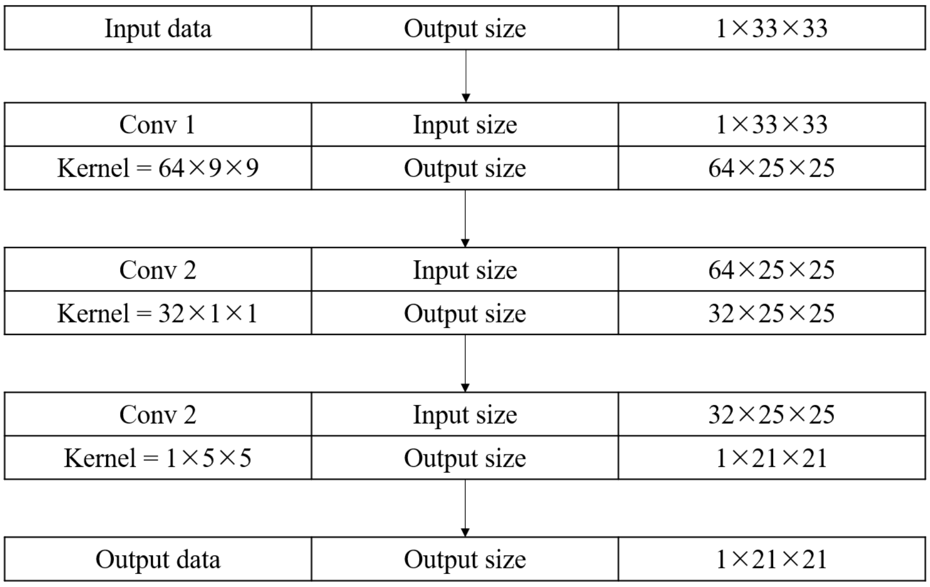

Input data | Output size | 1×33×33
Conv 1 | Input size | 1×33×33
Kernel = 64×9×9 | Output size | 64×25×25
Conv 2 | Input size | 64×25×25
Kernel = 32×1×1 | Output size | 32×25×25
Conv 2 | Input size | 32×25×25
Kernel = 1×5×5 | Output size | 1×21×21
Output data | Output size | 1×21×21

Figure 3. Model parameter

From this figure, we will conclude that in the first layer convolution, the size of convolution kernel is 9 × 9 (F1 × F1), the number of convolution kernel is 64 (N1), and 64 characteristic graphs are output. In the second layer, the size of convolution kernel is 1 × 1 (F2 × F2), the number of convolution kernels is 32 (N2), and 32 characteristic graphs are output. In the third layer of convolution, the size of convolution kernel is 5 × 5 (F3 × F3), the number of convolution kernels is 1 (N3), and the output of a feature image is the final reconstruction of high-resolution image.

As a pioneering research paper in the early stage, SRCNN also laid down the basic process for dealing with super-resolution problems:

(1) Look for a large number of real scene image samples.

(2) Each image is down sampled to reduce the resolution of the image. Generally, there are 2 times down sampling, 3 times down sampling and 4 times down sampling. If it is 2 times down sampling, the length and width of the image become 1 / 2 of the original. The image before down sampling is regarded as H of high-resolution image, and the image after down sampling is taken as L of low-resolution image. H and L are to form an effective image pair for later model training.

(3) When training the model, the low-resolution image L is enlarged and restored to the super-resolution image. And then compared with the original high-resolution image H, the difference is used to adjust the parameters of the model, and the difference is minimized through iterative training. In fact, many loss functions have been proposed to define this difference, and different definition methods will directly affect the final reconstruction effect.

(4) The trained model can be used to reconstruct new low-resolution images and obtain high-resolution images.

2.2 Tensorflow implementation

This section will start from the source code to complete the modeling, training and reasoning of SRCNN algorithm. This paper is based on Tensorflow, a deep learning framework, to complete all the coding work.

2.2.1 Operating environment

(1) Basic configuration

This paper uses Python language for code writing and python version is 3.5.4. Tensorflow deep learning framework is used to model the algorithm, and the corresponding version is Tensorflow 1.13.1. The operating system is windows 10, and the IDE is PyCharm.

(2) Install scipy

This package is mainly used to provide PSNR and SSIM calculations. PSNR and SSIM are two evaluation indexes often used in super score reconstruction.

```
1.  pip install scipy==1.1.0
```

(3) Visualization results

In order to view the running results conveniently in the training process of the model, it is recommended to use Tensorboard. Since Tensorboard is pushed by Tensorflow, we need to install Tensorflow first. However, there is no need to install the GPU version of Tensorflow. We only need to install the CPU version directly. The installation command is as follows:

```
2.  pip install tensorflow
```

The Tensorflow version installed in this paper is 1.13.1. Tensorboard is automatically installed during the installation of Tensorflow.

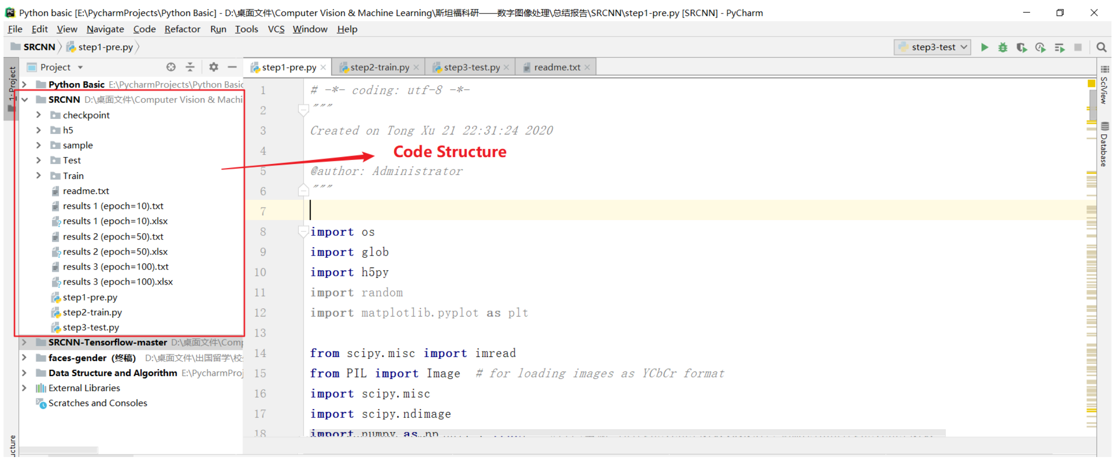
(4) Data set

In this paper, 91 image data set is used for training. Five pictures in set5 and 14 pictures in set14 are used in the test.

2.2.2 Code evaluation

(1) Code structure organization

In order to facilitate to read, run and modify the code, this paper adopts a relatively simple code organization. The complete structure is shown in the following figure:


Python basic [E:\PycharmProjects\Python Basic] - D:\桌面文件\Computer Vision & Machine Learning\斯坦福科研——数字图像处理\总结报告\SRCNN\step1-pre.py [SRCNN] - PyCharm
File Edit View Navigate Code Refactor Run Tools VCS Window Help
SRCNN step1-pre.py
Project
step1-pre.py
step2-train.py
step3-test.py
readme.txt
step3-test
Code Structure

```
Python Basic  E:\PycharmProjects\Python Basic
SRCNN  D:\桌面文件\Computer Vision & Machi
    checkpoint
    h5
    sample
    Test
    Train
    readme.txt
    results 1 (epoch=10).txt
    results 1 (epoch=10).xlsx
    results 2 (epoch=50).txt
    results 2 (epoch=50).xlsx
    results 3 (epoch=100).txt
    results 3 (epoch=100).xlsx
    step1-pre.py
    step2-train.py
    step3-test.py
SRCNN-Tensorflow-master  D:\桌面文件\Comp
faces-gender (终稿)  D:\桌面文件\出国留学\校
Data Structure and Algorithm  E:\PycharmProj
External Libraries
Scratches and Consoles
```

```python
# -*- coding: utf-8 -*-
"""
Created on Tong Xu 21 22:31:24 2020

@author: Administrator
"""

import os
import glob
import h5py
import random
import matplotlib.pyplot as plt

from scipy.misc import imread
from PIL import Image  # for loading images as YCbCr format
import scipy.misc
import scipy.ndimage
import numpy as np
```

Figure 4. Code structure

There are three .py files and five folders under the project root directory. Each file and folder are briefly described below.

Step1-pre.py is the preprocessing program: output the H5 format file for train and test. Step2-train.py is the program of training model: the model parameters output for training are saved in checkpoint folder. Step3-test.py is the file for the prediction of the program: input a picture H5 file, output PSNR value. And save the input image, label image and prediction image in the sample folder to view. Checkpoint folder: save model parameters. H5 folder: store train and test training set. Sample folder: save input, label and forecast image.

(2) System overview

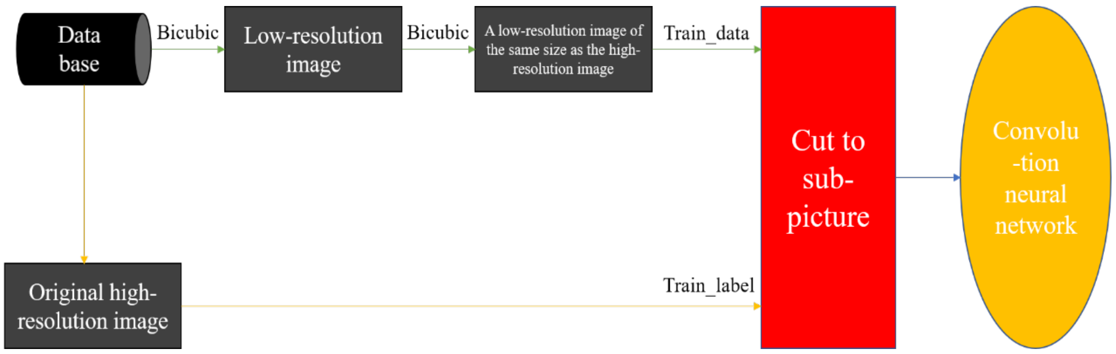

Data base
Bicubic
Low-resolution image
Bicubic
A low-resolution image of the same size as the high-resolution image
Train_data
Cut to sub-picture
Convolu-tion neural network
Original high-resolution image
Train_label

Figure 5. Flow chart of main parts of pre-process

After reading the paper carefully, I divide the reproduction work into the following parts:

Step 1: data set processing: train data set includes 91 pictures, only brightness channel. After that, reduce the resolution of the image by bicubic, and then enlarge it to the same size as the original image. Cut the pictures according to the stride, so that 91 pictures can get more than 20000 sub images of the same size (I think it should be to increase the training data, so as to get better results). Input the sub images into the network for training, and take the data without bicubic as the label.

Step 2: build convolution neural network: There is a three-layer network model. The first two layers have activation functions, and the last layer does not use activation functions. In addition, the filter weight and bias of each layer of network are given, and only the weights of these six parameters need to be learned in training. The loss function gets the value of MSE. It should be noted that the convolution layer is not filled to avoid boundary effect during train, so the input image will be smaller. Therefore, when we are preprocessing, the label should take the corresponding size to calculate the loss function.

Step 3: save the filter weight and bias weight values after training. The whole image needs to be read during the test. To ensure that the output size is consistent with the input size, padding which is equal with "SAME" in the convolution layer is filled to ensure that the output image size remains unchanged. Then PSNR can be calculated.

The py codes corresponding to these three parts are as follows:

Code of Step1:

(1) Get Brightness channel:

```
image=scipy.misc.imread(data[i], flatten=True, mode='YCbCr').astype(np.float)
```

(2) Crop the image to a size that can divide the scale evenly:

```
if len(image.shape) == 3:
  h, w, _ = image.shape
```

```
    h = h - np.mod(h, scale)
    w = w - np.mod(w, scale)
    label_ = image[0:h, 0:w, :]
else:
    h, w = image.shape
    h = h - np.mod(h, scale)
    w = w - np.mod(w, scale)
    label_ = image[0:h, 0:w]
```

(3) Two bicubic operations were performed to obtain a low resolution image of the same size as the original image:

```
label_ = label_ / 255.
input_ = scipy.ndimage.interpolation.zoom(label_, (1./scale), prefilter=False)
input_ = scipy.ndimage.interpolation.zoom(input_, (scale/1.), prefilter=False)
```

(4) Cut the sub image, in which the input image is cropped to 33×33, while the label needs to be cropped to 21×21.

```
for x in range(0, h-image_size+1, stride):  #以 stride 为步长进行取子图片操作
    for y in range(0, w-image_size+1, stride):
        sub_input = input_[x:x+image_size, y:y+image_size] # [33 x 33]
        sub_label = label_[x+int(padding):x+int(padding)+label_size, y+int(padding):y+int(padding)+label_size] # [21 x 21]
```

(5) Finally, the data is converted into. H5 file for storage.

```
with h5py.File(savepath, 'w') as hf:
    hf.create_dataset('data', data=arrdata)
    hf.create_dataset('label', data=arrlabel)
```

Code of Step2:
(1) Construction of convolution neural network:

```
images = tf.placeholder(tf.float32, [None, None, None, c_dim], name='images')
labels = tf.placeholder(tf.float32, [None, None, None, c_dim], name='labels')
weights = {
    'w1': tf.Variable(tf.random_normal([9, 9, 1, 64], stddev=1e-3),trainable=trainable, name='w1'),
    'w2': tf.Variable(tf.random_normal([1, 1, 64, 32], stddev=1e-3),trainable=trainable, name='w2'),
    'w3': tf.Variable(tf.random_normal([5, 5, 32, 1], stddev=1e-3), trainable=trainable,name='w3')
  }
biases = {
    'b1': tf.Variable(tf.zeros([64]),trainable=trainable ,name='b1'),
    'b2': tf.Variable(tf.zeros([32]),trainable=trainable, name='b2'),
    'b3': tf.Variable(tf.zeros([1]),trainable=trainable, name='b3')
  }
conv1 = tf.nn.relu(tf.nn.conv2d(images, weights['w1'], strides=[1,1,1,1], padding=padding) + biases['b1'])
conv2 = tf.nn.relu(tf.nn.conv2d(conv1, weights['w2'], strides=[1,1,1,1], padding=padding) + biases['b2'])
conv3 = tf.nn.conv2d(conv2, weights['w3'], strides=[1,1,1,1], padding=padding) + biases['b3']
pred=conv3
loss = tf.reduce_mean(tf.square(labels - pred)) #loss 函数为 mse 值
```

(2) Train and save the model

```
saver=tf.train.Saver(max_to_keep=5)
with tf.Session() as sess:
  if is_train:
    print("Training...")
    sess.run(tf.initialize_all_variables())

    ckpt = tf.train.get_checkpoint_state("checkpoint")
    if ckpt and ckpt.model_checkpoint_path:  # 加载上次训练保存的模型继续训练
     print("Continuing ")
     saver.restore(sess, ckpt.model_checkpoint_path)

    for ep in range(epoch):
     # Run by batch images
     batch_idxs = len(train_data) // batch_size
     for idx in range(0, batch_idxs):
       batch_images = train_data[idx*batch_size : (idx+1)*batch_size]
```

```
        batch_labels = train_label[idx*batch_size : (idx+1)*batch_size]

        counter +=1
        _, err = sess.run([train_op, loss], feed_dict={images: batch_images, labels: batch_labels})

        if counter % 10 == 0:
          print("%2d, %2d, %4.4f, %.8f" \
            % ((ep+1), counter, time.time()-start_time, err))

        if counter % 500 == 0:
          saver.save(sess,os.path.join('checkpoint', 'SRCNN'),global_step=counter,write_meta_graph=False)
```

Code of Step3:

(1) The filling mode of convolution kernel is changed to “SAME”.

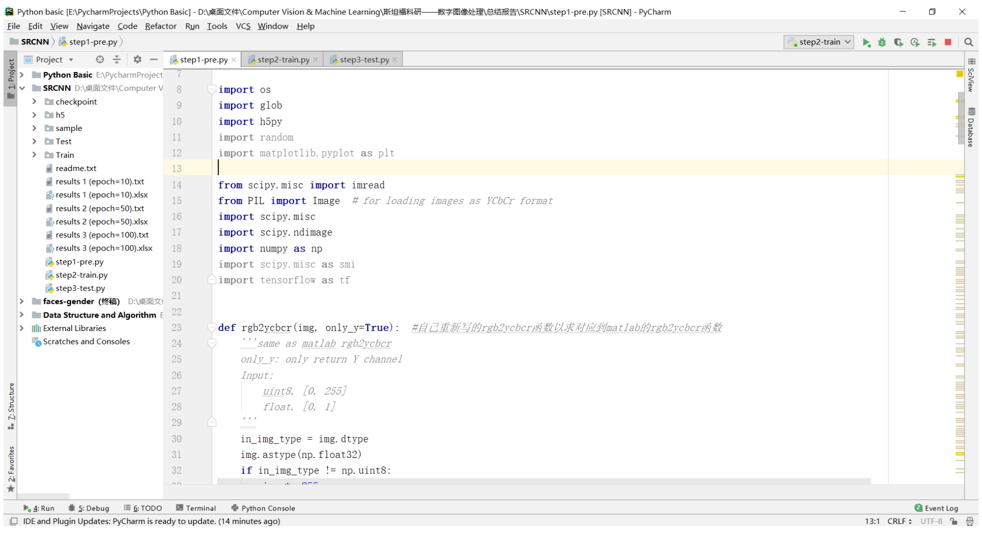
```
data_dir = os.path.join(os.getcwd(), 'h5/test.h5')  #h5 文件的路径
padding="SAME"                                      #因为是预测，所以需要填充
trainable = tf.Variable(False, dtype=tf.bool)
path=data_dir
```

(2) The prediction and PSNR calculation were carried out.

```
with tf.Session() as sess:
    ckpt = tf.train.get_checkpoint_state("checkpoint")
    if ckpt and ckpt.model_checkpoint_path:  # 加载保存的模型
        saver.restore(sess, ckpt.model_checkpoint_path)
        result = pred.eval({images: train_data, labels: train_label}) #得到训练后的结果
        result1 = result.squeeze()                          #降维
        result2 =np.around(result1 ,decimals=4)             #取小数点的后四位
        image_path = os.path.join(os.getcwd(),'sample')     #保存预测的图片到 sample 文件夹中
        image_path = os.path.join(image_path, "test_image.png")
        imsave(result2, image_path)
        label=train_label.squeeze()                         #label 数据降维
        print(psnr(label,result2))
```

2.2.3 Experimental results

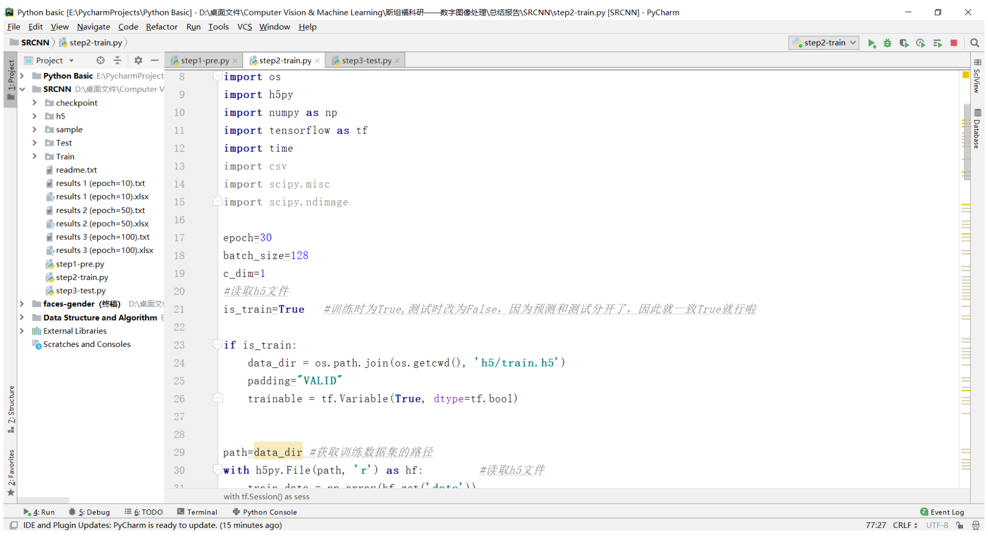
The whole framework of SRCNN has been set up. The next step is to train and see the effect. The PyCharm code framework is as follows.



Python basic [E:\PycharmProjects\Python Basic] - D:\桌面文件\Computer Vision & Machine Learning\斯坦福科研——数字图像处理\总结报告\SRCNN\step1-pre.py [SRCNN] - PyCharm

```python
import os
import glob
import h5py
import random
import matplotlib.pyplot as plt

from scipy.misc import imread
from PIL import Image  # for loading images as YCbCr format
import scipy.misc
import scipy.ndimage
import numpy as np
import scipy.misc as smi
import tensorflow as tf


def rgb2ycbcr(img, only_y=True):  #自己重新写的rgb2ycbcr函数以求对应到matlab的rgb2ycbcr函数
    '''same as matlab rgb2ycbcr
    only_y: only return Y channel
    Input:
        uint8, [0, 255]
        float, [0, 1]
    '''
    in_img_type = img.dtype
    img.astype(np.float32)
    if in_img_type != np.uint8:
```

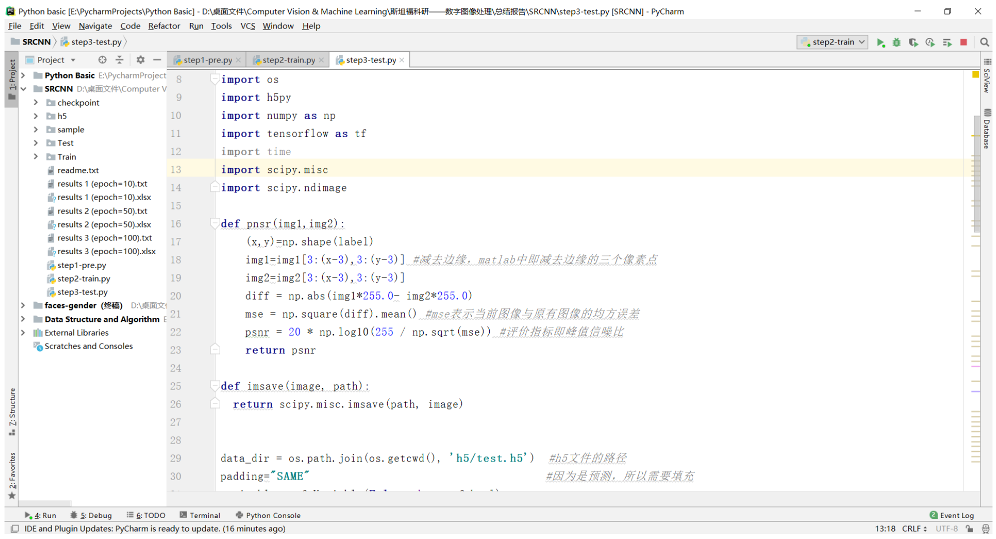
(a) step1-pre.py



Python basic [E:\PycharmProjects\Python Basic] - D:\桌面文件\Computer Vision & Machine Learning\斯坦福科研——数字图像处理\总结报告\SRCNN\step2-train.py [SRCNN] - PyCharm

```python
import os
import h5py
import numpy as np
import tensorflow as tf
import time
import csv
import scipy.misc
import scipy.ndimage

epoch=30
batch_size=128
c_dim=1
#读取h5文件
is_train=True   #训练时为True,测试时改为False, 因为预测和测试分开了, 因此就一致True就行啦

if is_train:
    data_dir = os.path.join(os.getcwd(), 'h5/train.h5')
    padding="VALID"
    trainable = tf.Variable(True, dtype=tf.bool)


path=data_dir #获取训练数据集的路径
with h5py.File(path, 'r') as hf:          #读取h5文件
```

(b) step2-train.py



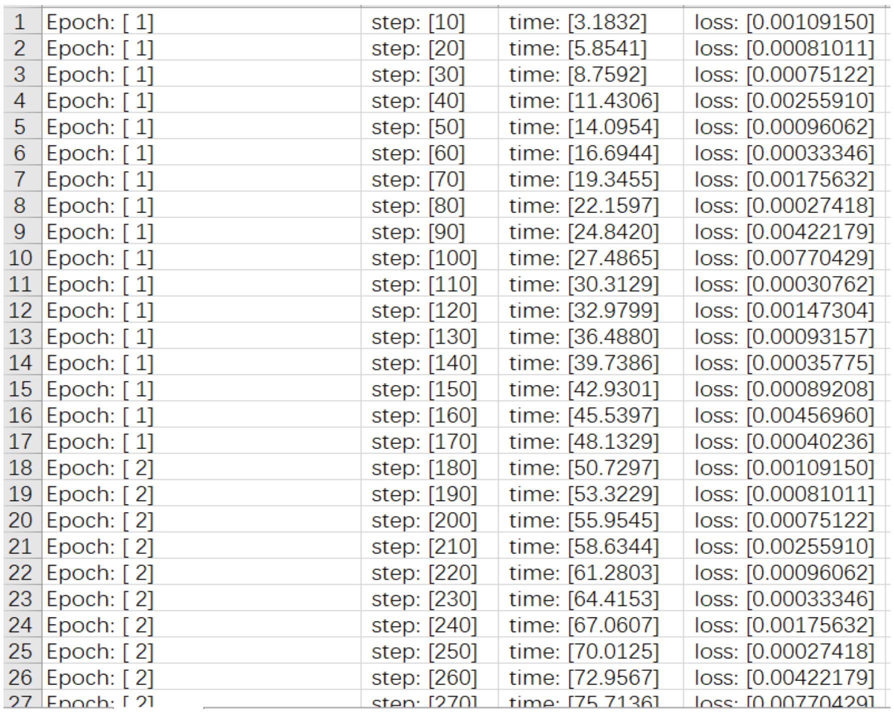
Python basic [E:\PycharmProjects\Python Basic] - D:\桌面文件\Computer Vision & Machine Learning\斯坦福科研——数字图像处理\总结报告\SRCNN\step3-test.py [SRCNN] - PyCharm

```python
import os
import h5py
import numpy as np
import tensorflow as tf
import time
import scipy.misc
import scipy.ndimage

def pnsr(img1,img2):
    (x,y)=np.shape(label)
    img1=img1[3:(x-3),3:(y-3)] #减去边缘, matlab中即减去边缘的三个像素点
    img2=img2[3:(x-3),3:(y-3)]
    diff = np.abs(img1*255.0- img2*255.0)
    mse = np.square(diff).mean() #mse表示当前图像与原有图像的均方误差
    psnr = 20 * np.log10(255 / np.sqrt(mse)) #评价指标即峰值信噪比
    return psnr

def imsave(image, path):
  return scipy.misc.imsave(path, image)


data_dir = os.path.join(os.getcwd(), 'h5/test.h5')  #h5文件的路径
padding="SAME"                                      #因为是预测, 所以需要填充
```

(c) step3-test.py

Figure 6. PyCharm code framework

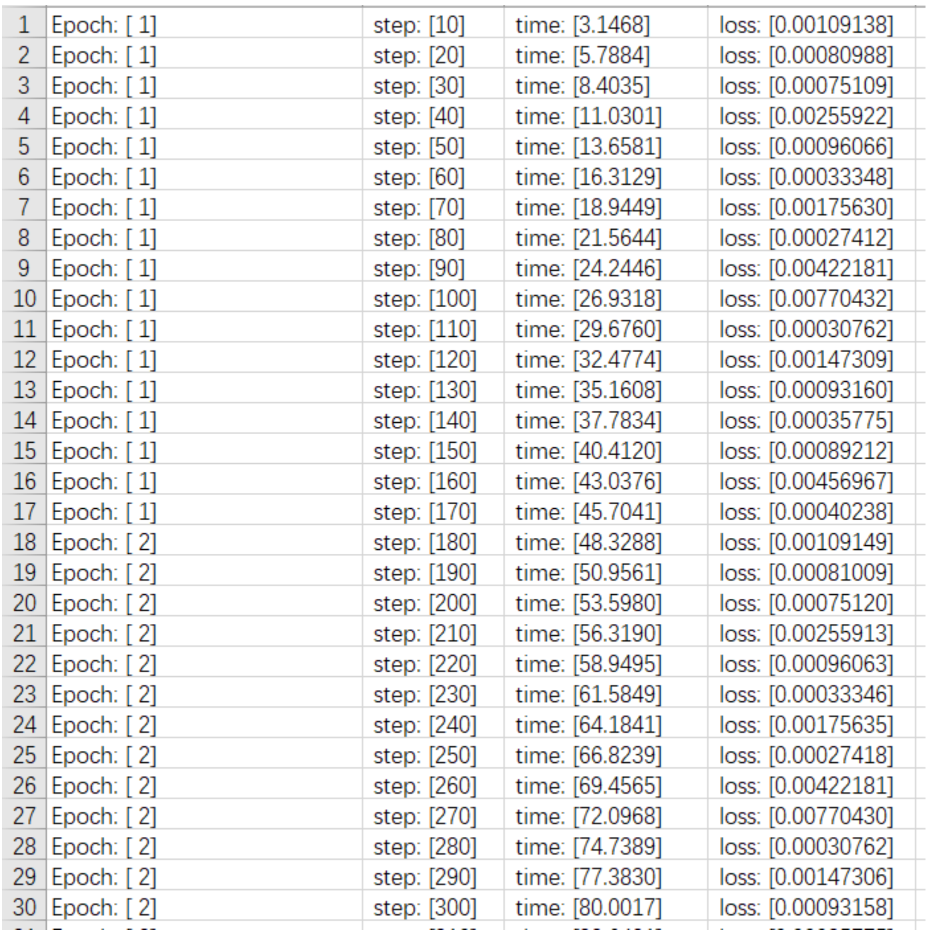
(1) When the epoch is 10, the PSNR is 24.0409. The results are as follows.



```
Epoch: [ 1]    step: [10]     time: [3.1832]     loss: [0.00109150]
Epoch: [ 1]    step: [20]     time: [5.8541]     loss: [0.00081011]
Epoch: [ 1]    step: [30]     time: [8.7592]     loss: [0.00075122]
Epoch: [ 1]    step: [40]     time: [11.4306]    loss: [0.00255910]
Epoch: [ 1]    step: [50]     time: [14.0954]    loss: [0.00096062]
Epoch: [ 1]    step: [60]     time: [16.6944]    loss: [0.00033346]
Epoch: [ 1]    step: [70]     time: [19.3455]    loss: [0.00175632]
Epoch: [ 1]    step: [80]     time: [22.1597]    loss: [0.00027418]
Epoch: [ 1]    step: [90]     time: [24.8420]    loss: [0.00422179]
Epoch: [ 1]    step: [100]    time: [27.4865]    loss: [0.00770429]
Epoch: [ 1]    step: [110]    time: [30.3129]    loss: [0.00030762]
Epoch: [ 1]    step: [120]    time: [32.9799]    loss: [0.00147304]
Epoch: [ 1]    step: [130]    time: [36.4880]    loss: [0.00093157]
Epoch: [ 1]    step: [140]    time: [39.7386]    loss: [0.00035775]
Epoch: [ 1]    step: [150]    time: [42.9301]    loss: [0.00089208]
Epoch: [ 1]    step: [160]    time: [45.5397]    loss: [0.00456960]
Epoch: [ 1]    step: [170]    time: [48.1329]    loss: [0.00040236]
Epoch: [ 2]    step: [180]    time: [50.7297]    loss: [0.00109150]
Epoch: [ 2]    step: [190]    time: [53.3229]    loss: [0.00081011]
Epoch: [ 2]    step: [200]    time: [55.9545]    loss: [0.00075122]
Epoch: [ 2]    step: [210]    time: [58.6344]    loss: [0.00255910]
Epoch: [ 2]    step: [220]    time: [61.2803]    loss: [0.00096062]
Epoch: [ 2]    step: [230]    time: [64.4153]    loss: [0.00033346]
Epoch: [ 2]    step: [240]    time: [67.0607]    loss: [0.00175632]
Epoch: [ 2]    step: [250]    time: [70.0125]    loss: [0.00027418]
Epoch: [ 2]    step: [260]    time: [72.9567]    loss: [0.00422179]
Epoch: [ 2]    step: [270]    time: [75.7136]    loss: [0.00770429]
```

(a) Epoch=10

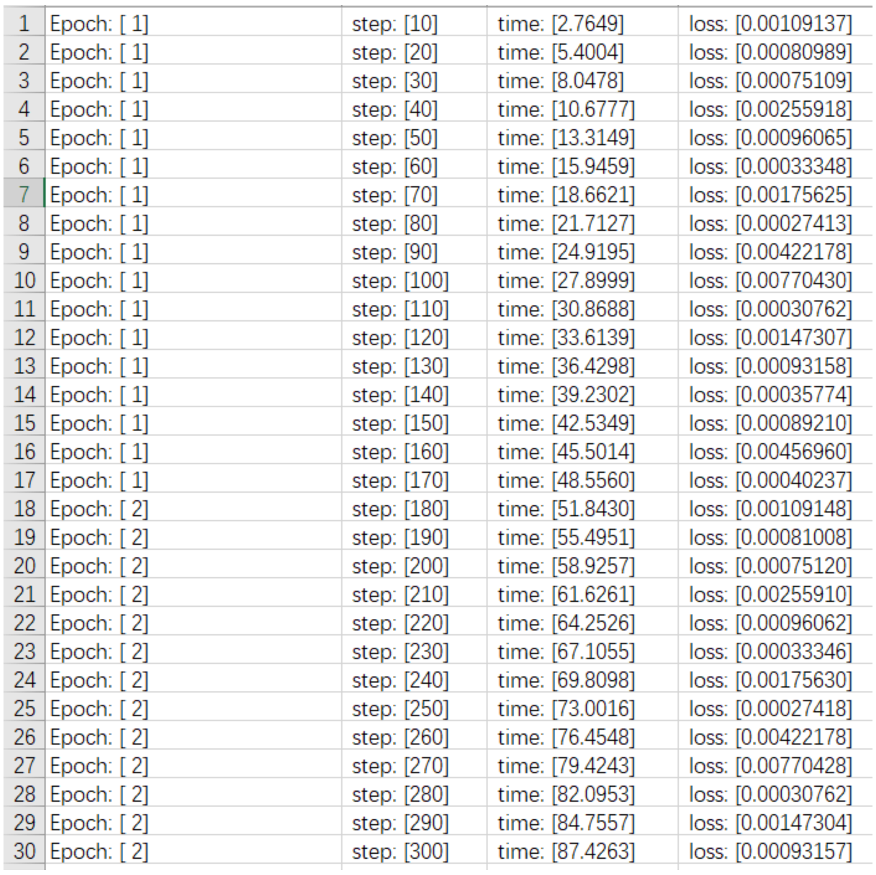
(2) When the epoch is 50, the PSNR is 24.0406. The results are as follows.



```
Epoch: [ 1]    step: [10]     time: [3.1468]     loss: [0.00109138]
Epoch: [ 1]    step: [20]     time: [5.7884]     loss: [0.00080988]
Epoch: [ 1]    step: [30]     time: [8.4035]     loss: [0.00075109]
Epoch: [ 1]    step: [40]     time: [11.0301]    loss: [0.00255922]
Epoch: [ 1]    step: [50]     time: [13.6581]    loss: [0.00096066]
Epoch: [ 1]    step: [60]     time: [16.3129]    loss: [0.00033348]
Epoch: [ 1]    step: [70]     time: [18.9449]    loss: [0.00175630]
Epoch: [ 1]    step: [80]     time: [21.5644]    loss: [0.00027412]
Epoch: [ 1]    step: [90]     time: [24.2446]    loss: [0.00422181]
Epoch: [ 1]    step: [100]    time: [26.9318]    loss: [0.00770432]
Epoch: [ 1]    step: [110]    time: [29.6760]    loss: [0.00030762]
Epoch: [ 1]    step: [120]    time: [32.4774]    loss: [0.00147309]
Epoch: [ 1]    step: [130]    time: [35.1608]    loss: [0.00093160]
Epoch: [ 1]    step: [140]    time: [37.7834]    loss: [0.00035775]
Epoch: [ 1]    step: [150]    time: [40.4120]    loss: [0.00089212]
Epoch: [ 1]    step: [160]    time: [43.0376]    loss: [0.00456967]
Epoch: [ 1]    step: [170]    time: [45.7041]    loss: [0.00040238]
Epoch: [ 2]    step: [180]    time: [48.3288]    loss: [0.00109149]
Epoch: [ 2]    step: [190]    time: [50.9561]    loss: [0.00081009]
Epoch: [ 2]    step: [200]    time: [53.5980]    loss: [0.00075120]
Epoch: [ 2]    step: [210]    time: [56.3190]    loss: [0.00255913]
Epoch: [ 2]    step: [220]    time: [58.9495]    loss: [0.00096063]
Epoch: [ 2]    step: [230]    time: [61.5849]    loss: [0.00033346]
Epoch: [ 2]    step: [240]    time: [64.1841]    loss: [0.00175635]
Epoch: [ 2]    step: [250]    time: [66.8239]    loss: [0.00027418]
Epoch: [ 2]    step: [260]    time: [69.4565]    loss: [0.00422181]
Epoch: [ 2]    step: [270]    time: [72.0968]    loss: [0.00770430]
Epoch: [ 2]    step: [280]    time: [74.7389]    loss: [0.00030762]
Epoch: [ 2]    step: [290]    time: [77.3830]    loss: [0.00147306]
Epoch: [ 2]    step: [300]    time: [80.0017]    loss: [0.00093158]
[illegible]
```

(b) Epoch=50

(3) When the epoch is 100, the PSNR is 24.0407. The results are as follows.

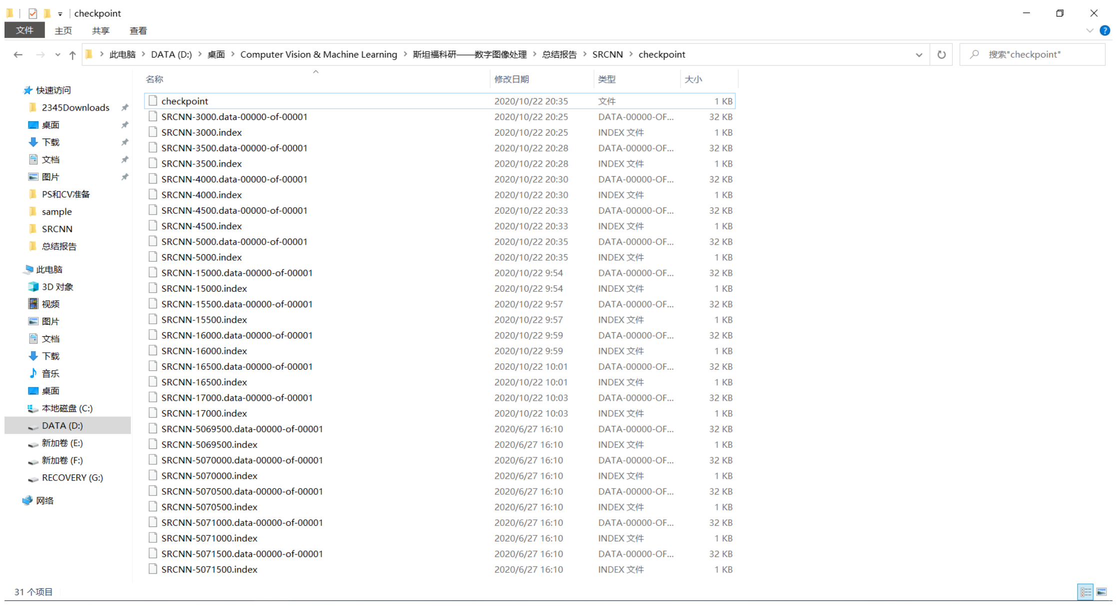


1	Epoch: [1]	step: [10]	time: [2.7649]	loss: [0.00109137]
2	Epoch: [1]	step: [20]	time: [5.4004]	loss: [0.00080989]
3	Epoch: [1]	step: [30]	time: [8.0478]	loss: [0.00075109]
4	Epoch: [1]	step: [40]	time: [10.6777]	loss: [0.00255918]
5	Epoch: [1]	step: [50]	time: [13.3149]	loss: [0.00096065]
6	Epoch: [1]	step: [60]	time: [15.9459]	loss: [0.00033348]
7	Epoch: [1]	step: [70]	time: [18.6621]	loss: [0.00175625]
8	Epoch: [1]	step: [80]	time: [21.7127]	loss: [0.00027413]
9	Epoch: [1]	step: [90]	time: [24.9195]	loss: [0.00422178]
10	Epoch: [1]	step: [100]	time: [27.8999]	loss: [0.00770430]
11	Epoch: [1]	step: [110]	time: [30.8688]	loss: [0.00030762]
12	Epoch: [1]	step: [120]	time: [33.6139]	loss: [0.00147307]
13	Epoch: [1]	step: [130]	time: [36.4298]	loss: [0.00093158]
14	Epoch: [1]	step: [140]	time: [39.2302]	loss: [0.00035774]
15	Epoch: [1]	step: [150]	time: [42.5349]	loss: [0.00089210]
16	Epoch: [1]	step: [160]	time: [45.5014]	loss: [0.00456960]
17	Epoch: [1]	step: [170]	time: [48.5560]	loss: [0.00040237]
18	Epoch: [2]	step: [180]	time: [51.8430]	loss: [0.00109148]
19	Epoch: [2]	step: [190]	time: [55.4951]	loss: [0.00081008]
20	Epoch: [2]	step: [200]	time: [58.9257]	loss: [0.00075120]
21	Epoch: [2]	step: [210]	time: [61.6261]	loss: [0.00255910]
22	Epoch: [2]	step: [220]	time: [64.2526]	loss: [0.00096062]
23	Epoch: [2]	step: [230]	time: [67.1055]	loss: [0.00033346]
24	Epoch: [2]	step: [240]	time: [69.8098]	loss: [0.00175630]
25	Epoch: [2]	step: [250]	time: [73.0016]	loss: [0.00027418]
26	Epoch: [2]	step: [260]	time: [76.4548]	loss: [0.00422178]
27	Epoch: [2]	step: [270]	time: [79.4243]	loss: [0.00770428]
28	Epoch: [2]	step: [280]	time: [82.0953]	loss: [0.00030762]
29	Epoch: [2]	step: [290]	time: [84.7557]	loss: [0.00147304]
30	Epoch: [2]	step: [300]	time: [87.4263]	loss: [0.00093157]

(c) Epoch=100

Figure 7. Different results

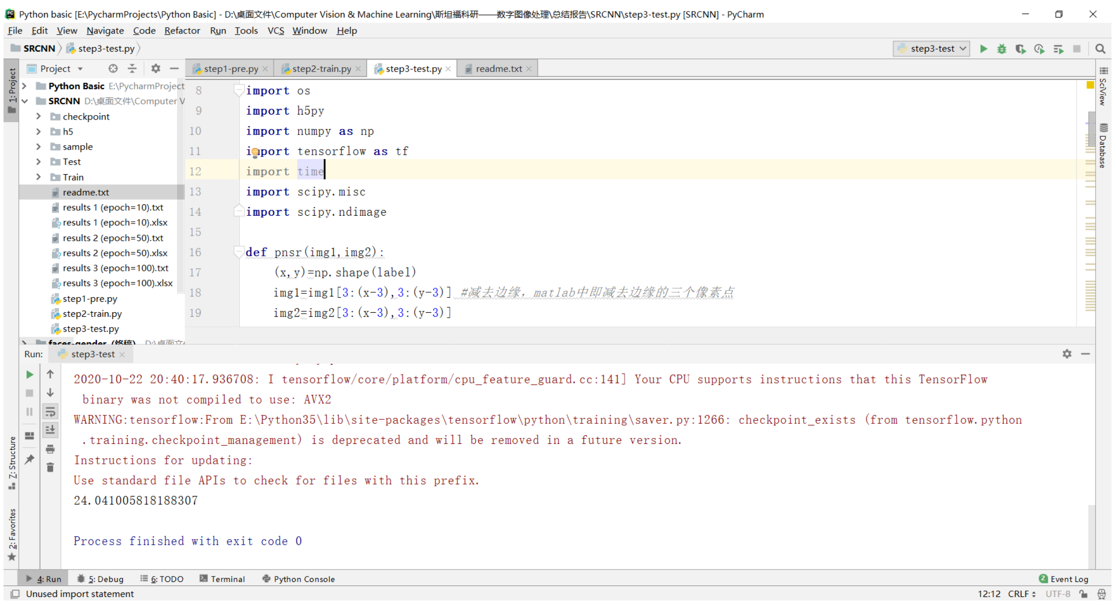
According to different epoch, the training parameters obtained are shown in the figure below.



名称	修改日期	类型	大小
checkpoint	2020/10/22 20:35	文件	1 KB
SRCNN-3000.data-00000-of-00001	2020/10/22 20:25	DATA-00000-OF...	32 KB
SRCNN-3000.index	2020/10/22 20:25	INDEX 文件	1 KB
SRCNN-3500.data-00000-of-00001	2020/10/22 20:28	DATA-00000-OF...	32 KB
SRCNN-3500.index	2020/10/22 20:28	INDEX 文件	1 KB
SRCNN-4000.data-00000-of-00001	2020/10/22 20:30	DATA-00000-OF...	32 KB
SRCNN-4000.index	2020/10/22 20:30	INDEX 文件	1 KB
SRCNN-4500.data-00000-of-00001	2020/10/22 20:33	DATA-00000-OF...	32 KB
SRCNN-4500.index	2020/10/22 20:33	INDEX 文件	1 KB
SRCNN-5000.data-00000-of-00001	2020/10/22 20:35	DATA-00000-OF...	32 KB
SRCNN-5000.index	2020/10/22 20:35	INDEX 文件	1 KB
SRCNN-15000.data-00000-of-00001	2020/10/22 9:54	DATA-00000-OF...	32 KB
SRCNN-15000.index	2020/10/22 9:54	INDEX 文件	1 KB
SRCNN-15500.data-00000-of-00001	2020/10/22 9:57	DATA-00000-OF...	32 KB
SRCNN-15500.index	2020/10/22 9:57	INDEX 文件	1 KB
SRCNN-16000.data-00000-of-00001	2020/10/22 9:59	DATA-00000-OF...	32 KB
SRCNN-16000.index	2020/10/22 9:59	INDEX 文件	1 KB
SRCNN-16500.data-00000-of-00001	2020/10/22 10:01	DATA-00000-OF...	32 KB
SRCNN-16500.index	2020/10/22 10:01	INDEX 文件	1 KB
SRCNN-17000.data-00000-of-00001	2020/10/22 10:03	DATA-00000-OF...	32 KB
SRCNN-17000.index	2020/10/22 10:03	INDEX 文件	1 KB
SRCNN-5069500.data-00000-of-00001	2020/6/27 16:10	DATA-00000-OF...	32 KB
SRCNN-5069500.index	2020/6/27 16:10	INDEX 文件	1 KB
SRCNN-5070000.data-00000-of-00001	2020/6/27 16:10	DATA-00000-OF...	32 KB
SRCNN-5070000.index	2020/6/27 16:10	INDEX 文件	1 KB
SRCNN-5070500.data-00000-of-00001	2020/6/27 16:10	DATA-00000-OF...	32 KB
SRCNN-5070500.index	2020/6/27 16:10	INDEX 文件	1 KB
SRCNN-5071000.data-00000-of-00001	2020/6/27 16:10	DATA-00000-OF...	32 KB
SRCNN-5071000.index	2020/6/27 16:10	INDEX 文件	1 KB
SRCNN-5071500.data-00000-of-00001	2020/6/27 16:10	DATA-00000-OF...	32 KB
SRCNN-5071500.index	2020/6/27 16:10	INDEX 文件	1 KB

Figure 8. Training parameters

Predict and read the parameters in checkpoint / SRCNN-5000, and get the super-resolution image of the ppt3 image in set5.

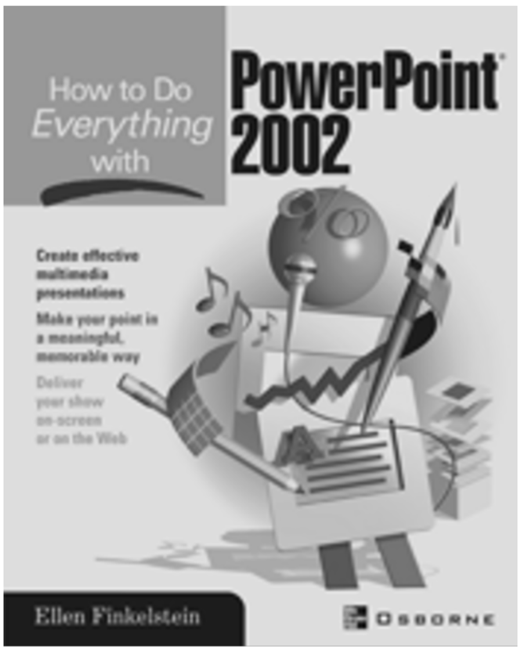


```
import os
import h5py
import numpy as np
import tensorflow as tf
import time
import scipy.misc
import scipy.ndimage

def pnsr(img1, img2):
    (x, y)=np.shape(label)
    img1=img1[3:(x-3),3:(y-3)] #减去边缘，matlab中即减去边缘的三个像素点
    img2=img2[3:(x-3),3:(y-3)]
```

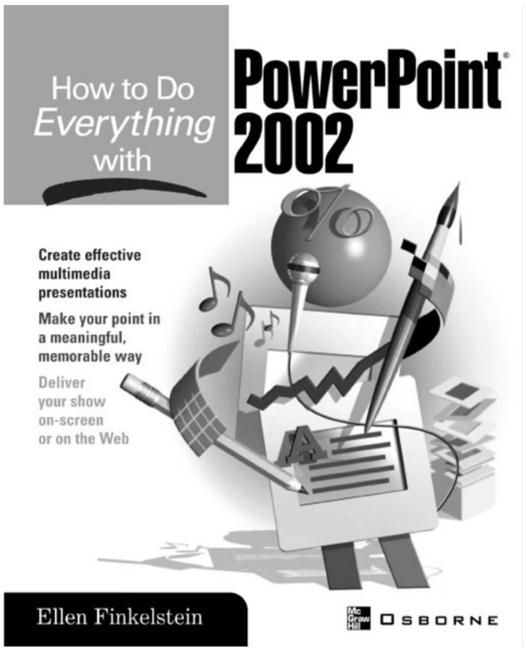
```
2020-10-22 20:40:17.936708: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports instructions that this TensorFlow
 binary was not compiled to use: AVX2
WARNING:tensorflow:From E:\Python35\lib\site-packages\tensorflow\python\training\saver.py:1266: checkpoint_exists (from tensorflow.python
 .training.checkpoint_management) is deprecated and will be removed in a future version.
Instructions for updating:
Use standard file APIs to check for files with this prefix.
24.041005818188307

Process finished with exit code 0
```

Python basic [E:\PycharmProjects\Python Basic] - D:\桌面文件\Computer Vision & Machine Learning\斯坦福科研——数字图像处理\总结报告\SRCNN\step3-test.py [SRCNN] - PyCharm
File Edit View Navigate Code Refactor Run Tools VCS Window Help
SRCNN
step3-test.py
step3-test
Project
step1-pre.py
step2-train.py
step3-test.py
readme.txt
Python Basic E:\PycharmProject
SRCNN D:\桌面文件\Computer V
checkpoint
h5
sample
Test
Train
readme.txt
results 1 (epoch=10).txt
results 1 (epoch=10).xlsx
results 2 (epoch=50).txt
results 2 (epoch=50).xlsx
results 3 (epoch=100).txt
results 3 (epoch=100).xlsx
step1-pre.py
step2-train.py
step3-test.py
Run:
step3-test
4: Run
5: Debug
6: TODO
Terminal
Python Console
Event Log
Unused import statement
12:12 CRLF UTF-8

(a) PSNR=24.0410


How to Do Everything with
PowerPoint 2002
Create effective multimedia presentations
Make your point in a meaningful, memorable way
Deliver your show on-screen or on the Web
Ellen Finkelstein
OSBORNE

(b) input_image


How to Do Everything with
PowerPoint 2002
Create effective multimedia presentations
Make your point in a meaningful, memorable way
Deliver your show on-screen or on the Web
Ellen Finkelstein
OSBORNE

(c) label_image

Figure 9. Results

3. Conclusion

In this paper, I implement the SRCNN algorithm. The PSNR is about 24.0410 dB, which is lower than 30.92 dB in other papers. Therefore, I saved the generated image and found that it seems that the brightness is lower than the input. The problem seems to be here. Moreover, in other codes, they normalize the non-zero region in the receptive field. In this article, the code missed this step, so the final super-resolution generated image is generally dark. In the future research, I will continue to improve the SRCNN code, and continue to study some new models to deepen my understanding of image super-resolution reconstruction.