Selected Topics in Visual Recognition using Deep Learning

Homework 4

Kuan-Chung Lin 0856735

Project repository: https://github.com/purpleFar/super-resolution

---

Introduction

The proposed challenge is super resolution task, which is transfer the low-resolution image to the high-resolution image. The training dataset contains 291 images of different resolutions, and the test set contains 14 images. Our goal is converted all of images in the test set to 3x resolution images. And our output image needs to be as clear as possible.

low-resolution (177*96)

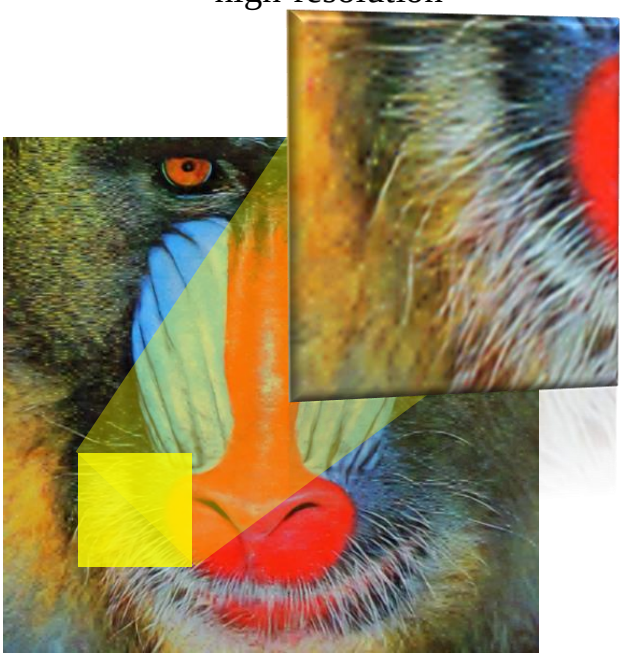
high-resolution (351*288)

In my opinion, the difficulty with this challenge is that we do different processing of the training set and the test set, which will affect the effect. I observed that the original images of training set and the test set are of similar resolution. During the training time, we shrink all images in the training set by three times. And input the resize image (low-resolution images in training

time) in our model to revert the original images (high-resolution images in training time). But at the inference time, we directly enter the original image (low-resolution images at inference time) of test set into our model and output the 3x resolution image (high-resolution images at inference time). In other words, the quality of high-resolution images in training time are only the same as low-resolution images at inference time. Moreover, there are only 291 images in training set, which do not contain various environmental images. That's why I don't think we can make models that can show many details of output images. For instance, there is an image like below:

low-resolution

high-resolution



If we want to get the great score in result, our model needs to show the details of hair. But there are no such clear images of hair in the training set. Therefore, our model is limited unless we use higher-quality additional data for training.

Environment

- System: Ubuntu 18.04.3 LTS
- CPU: Intel® Xeon® Platinum 8260M CPU @ 2.40GHz
- GPU: NVIDIA Tesla T4
- Python: 3.6.12
- Extra modules: Pytorch-1.7.1

Data Processing

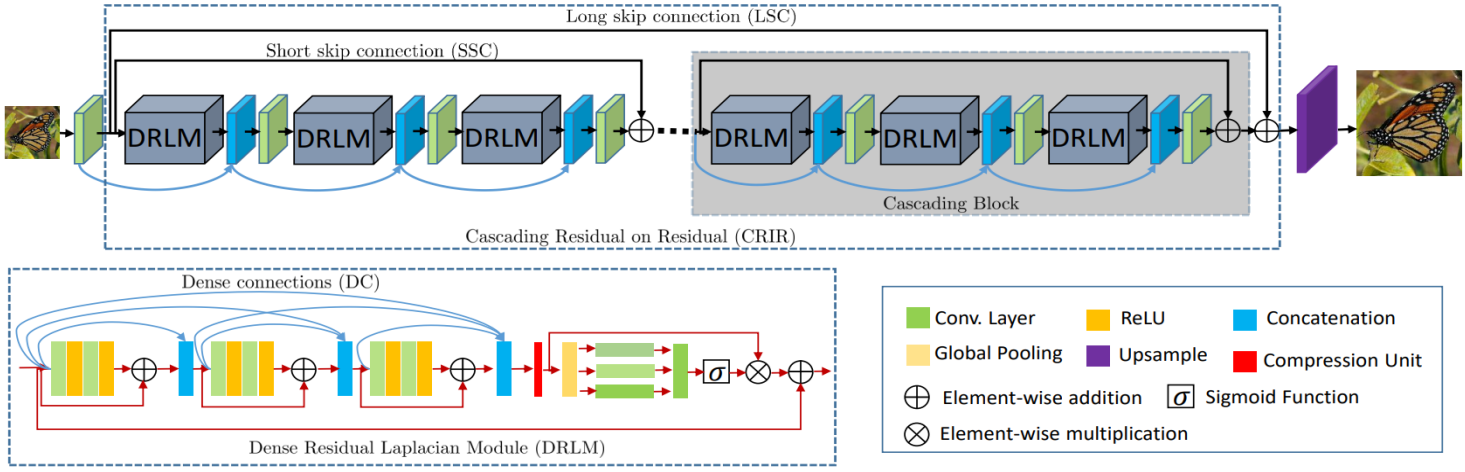
I used all the training data to train my model and the test set to validate my model. Note that, I am not using the original images as input when validation. I perform the same processing as training set, which is input the 1/3 of the original image resolution and get the original images.

Data augmentation

- Randomly crop patches of random size. In training process, 12*12 is the smallest of the patch size and 144*144 is the biggest. (ie. 4*4 ~ 48*48 for input size). Because there are some images size smaller than my patches size, I do the padding process when the image size is not large enough.
- Randomly adjust below processing order and perform them:
  1. Randomly adjust brightness of an Image
  2. Randomly adjust contrast of an Image
  3. Randomly perform gamma correction on an image
  4. Randomly adjust color saturation of an image
- Randomly horizontal flip
- Randomly vertical flip
- Randomly rotate 90 degrees

Model

I used DRLN+ be my basic structure, which is the state-of-the-art model in "Image Super-Resolution on Set5 - 3x upscaling" challenge.



For the architecture, I made the following changes:

- In order to make the model more powerful, I add 4 DRLM blocks after the last DRLM block.
- Because I consider the padding will affects my model, I add a mask layer at the end of the model. The shape of mask layer has the same shape as output, it does the element-wise product with the output of the previous layer. When the pixel corresponding to the mask comes from padding, the masks are zero and the others are one.
- I clip the value to 1 if the output bigger than 1 (the converted original value is 255), and 0 if the output small than 0.

Training setting

- Adam(lr=0.0001, betas=(0.9, 0.999), eps=1e-08)
- lr_scheduler.StepLR(optimizer, epoch=200, gamma=0.5)
- 500 epochs
- Batch size = 8

Other technical details

I used the different from general process to train my model. Because my model has 290 convolutional layers, this is a deep network. It is difficult to optimize from scratch.
I trained my model from 7 steps:

1. Trained the first three DRLM blocks and all of component after last DRLM block without other DRLM block. Go to the next step when the training loss is stuck.
   (Randomly crop patches of random size: 12*12 ~ 18*18)
2. Add 3 DRLM blocks and train it. Go to the next step when the training loss is stuck.
   (Randomly crop patches of random size: 33*33 ~ 39*39)
3. Add 3 DRLM blocks and train it. Go to the next step when the training loss is stuck.
   (Randomly crop patches of random size: 54*54 ~ 60*60)
4. Add 3 DRLM blocks and train it. Go to the next step when the training loss is stuck.
   (Randomly crop patches of random size: 75*75 ~ 81*81)
5. Add 4 DRLM blocks and train it. Go to the next step when the training loss is stuck.
   (Randomly crop patches of random size: 96*96 ~ 102*102)
6. Add 4 DRLM blocks and train it. Go to the next step when the training loss is stuck.

(Randomly crop patches of random size: 117*117 ~ 123*123)

7. Used whole model and train it. And validate the model each epoch to save the best parameters. Training until 500 epochs.
   (Randomly crop patches of fixed size: 144*144)

Summary of Results

Finally, I got psnr 25.867 by using my own model.