CSCI 3290: Computational Imaging and Vision (Due: 23:59, 29 Apr. 2020)

Assignment 4: Single Image Super Resolution

1 Introduction

Single image super resolution (SISR) is a classical image restoration problem which aims to recover a high-resolution (HR) image from the corresponding low-resolution (LR) image.

In this assignment, you'll need to implement a super resolution convolutional neural network (SRCNN) with PyTorch. We use "Learning a Deep Convolutional Network for Image Super-Resolution" [1] as the basic reference. The basic network architecture and implementation details will be provided in the following sections. In the end, you should submit the source code and the well-trained model after your finish this assignment.

2 Implementation details

2.1 SRCNN

SRCNN uses pairs of LR and HR images to learn the mapping between them. For this purpose, image databases containing LR and HR pairs are created and used as a training set. The learned mapping can be used to predict HR details in a new image.

The SRCNN consists of the following operations:

1. **Preprocessing**: Upscales LR image to desired HR size (using bicubic interpolation).
2. **Feature extraction**: Extracts a set of feature maps from the upscaled LR image.
3. **Non-linear mapping**: Maps the feature maps representing LR to HR patches.
4. **Reconstruction**: Produces the HR image from HR patches.

Operations 2–4 above can be cast as a convolutional layer in a CNN that accepts the upscaled images as input, and outputs the HR image. This CNN consists of three convolutional layers:

- **Conv. Layer 1: Patch extraction**
    - 64 filters of size 3 x 9 x 9 (padding=4, stride=1)
    - Activation function: ReLU
    - Output: 64 feature maps
- **Conv. Layer 2: Non-linear mapping**
    - 32 filters of size 64 x 1x 1 (padding=0, stride=1)
    - Activation function: ReLU
    - Output: 32 feature maps
- **Conv. Layer 3: Reconstruction**
    - 3 filter of size 32 x 5 x 5 (padding=2, stride=1)
    - Activation function: Identity
    - Output: HR image

The overall structure of SRCNN is shown in Figure 1.

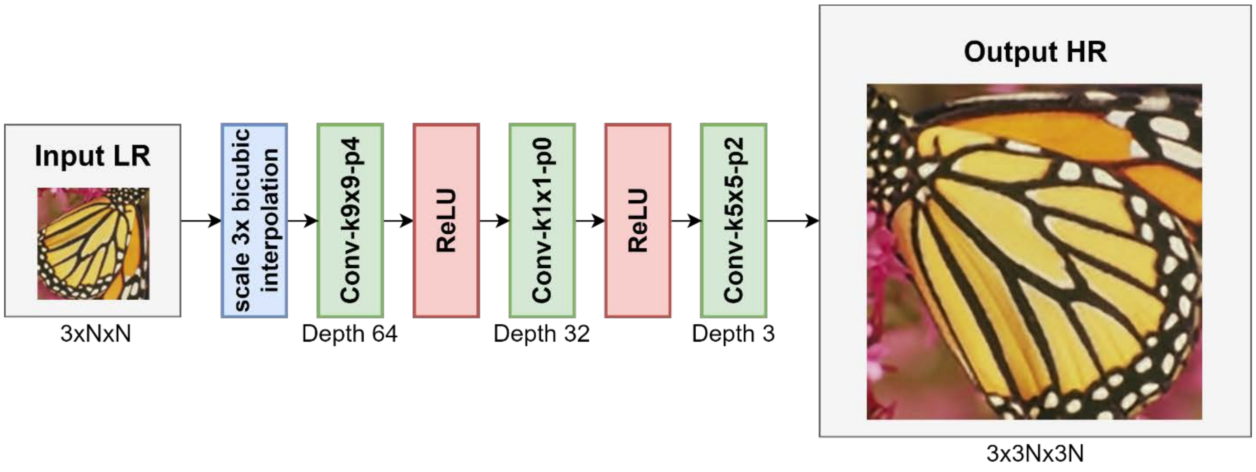

Figure 1. Network Architecture of SRCNN with upscaling factor=3

In this assignment, you will need to implement a SRCNN with upscaling factor 3 in PyTorch. Let $Y_\theta(x)$ denote this SRCNN model in the following sections.

2.2 Model Training

A typical training framework for a neural network is as follows:

- Define the neural network that has some learnable parameters (or weights)
- Iterate over a dataset of inputs
- Process input through the network
- Compute the loss between output and the ground truth (how far is the output from being correct)
- Propagate gradients back into the network's parameters
- Update the weights of the network, typically using a simple update rule: `weight = weight - learning_rate * gradient`

The SRCNN is a simple feed-forward neural network. It upscaled the input LR, feeds the upscaled image through several layers one after the other, and then finally gives the output. The overall training procedure of this network is the same as the above framework. To be specific, with PyTorch, the pseudocode of training procedure for SRCNN can be described as follows:

```
procedure TrainOneEpoch($model\ Y_\theta, optimizer, trainSet$)
    for each ($LR_i, HR_i$) pair in $trainSet$ do
        zero the gradient buffers of $optimizer$
        compute $output_i = Y_\theta(LR_i)$
        compute the loss $\ell = \boldsymbol{loss\_function}(output_i, HR_i)$
        back-propagate the gradients from $\ell$ to the parameters $\theta$ of model $Y_\theta$
        use $optimizer$ to update the parameters $\theta$
        record the loss for training statistics [optional]
```

Note that the actual code might differ from the pseudocode. Please check tutorial notes and PyTorch document for related APIs. Besides, we use mean squared error (MSE) as the ***loss_function***:

$$L(\theta) = \frac{1}{n}\sum_{i=1}^{n} \|Y_\theta(LR_i) - HR_i\|^2$$

where n is the number of training samples. This loss functions can be found in PyTorch APIs. Using MSE as the loss function favors a high peak signal-to-noise ratio (PSNR). The PSNR is a widely used metric for quantitatively evaluating image restoration quality and is at least partially related to the perceptual quality. We will also use PSNR (the higher the better) to measure the performance of the trained model. The PSNR related snippets are provided in the skeleton code.

In this assignment, we use 91-Image dataset as our training dataset and Set-5 dataset as the testing dataset. The data related part is provided in the skeleton code.

Other hyperparameters related to training are listed below:

- Training epoch=100; one epoch means completing one loop over whole dataset
- Optimizer: Adam
- Learning rate=0.0001
- Training batch size=128; the number of inputs being feed into the network at once

Note that the above hyperparameters might not lead to reasonable performance. You are encouraged to find other possible hyperparameters to achieve better performance.

2.3 Skeleton code usage

2.3.1 Project structure

The skeleton code consists of 6 files:

- `train.py`: a CLI program, which contains the procedure of model training [to be completed]
- `model.py`: SRCNN model [need to be completed]
- `data.py`: dataset related codes
- `utils.py`: helper functions
- `super_resolve.py`: a CLI program, which can super resolve images given a well-trained model
- `info.py`: submission info [need to be completed]

In this assignment, you are required to implement a SRCNN in **PyTorch 1.2+**. In order to make the skeleton code functional, you need to complete these three files in the skeleton code: `train.py, model.py, info.py.`

2.3.2 train.py

The usage of `train.py` can be describe as follows:

```
# train the SRCNN model using GPU, set learning rate=0.0005, batch size=256,
# make the program train 100 epoches and save a checkpoint every 10 epoches
python train.py train --cuda --lr=0.0005 --batch-size=256 --num-epoch=100 --save-
freq=10
# train the SRCNN model using CPU, set learning rate=0.001, batch size=128,
# make the program train 20 epoches and save a checkpoint every 2 epoches
python train.py train --lr=0.001 --batch-size=128 --num-epoch=20 --save-freq=2

# resume training with GPU from "checkpoint.x" with saved hyperparameters
python train.py resume checkpoint.x --cuda
# resume training from "checkpoint.x" and override some of saved hyperparameters
python train.py resume checkpoint.x --batch-size=16 --num-epoch=200

# inspect "checkpoint.x"
python train.py inspect checkpoint.x
```

Note that the checkpoint consists of the parameters of a trained model, the state of an optimizer, and the arguments (or hyperparameters) used in current training procedure. Thus, you can use checkpoint to resume training.

2.3.3 super_resolve.py

The usage of `super_resolve.py` can be describe as follows:

```
# use the model stored in "checkpoint.x" to super resolve "lr.bmp"
python super_resolve.py  --checkpoint checkpoint.x  lr.bmp
```

You may use this program to perform qualitative comparison using the images inside the `image_examples.zip` file. This file contains LR images, upscaled images with bicubic interpolation, and ground truth (GT) HR images.

3 Grading Scheme

The assignment will be graded by the following marking scheme:

- Code [40 points]
  - The network implementation: 20 points
  - The codes for training: 20 points
- Model Training [60 points]
  - The testing result of your well-trained model. The higher evaluation metrics are, the higher your score will be.

4 Submission guidelines

You will need to submit `train.py`, `info.py`, `model.py` and `checkpoint.pth` to the Blackboard. The saved checkpoint can have various filenames, you should select one and rename it to `checkpoint.pth.`

You need to archive all the mentioned files in `.zip` or `.7z` format, name this archive file with your name and student ID (e.g. **`1155xxxxxx_lastname_firstname.zip`**), and then submit this file to the Blackboard.

In case of multiple submissions, only the latest one will be considered.

4.1 Submission requirements

In your source code files, type your full name and student ID, just like:

```
#
# CSCI3290 Computational Imaging and Vision *
# --- Declaration --- *
# I declare that the assignment here submitted is original except for source
# material explicitly acknowledged. I also acknowledge that I am aware of
# University policy and regulations on honesty in academic work, and of the
# disciplinary guidelines and procedures applicable to breaches of such policy
# and regulations, as contained in the website
# http://www.cuhk.edu.hk/policy/academichonesty/ *
# Assignment 4
# Name:
# Student ID:
# Email Addr:
#
```

Besides, complete the following parts in the `info.py`. The info object will be saved to the checkpoint, i.e. well-trained model.

```
info = Info(
    name="your name",
    id="1155xxxxxx",
    email="example@example.com"
)
```

4.2 Late submission penalty

If you submit your solution after the due date, 10 marks will be deducted per day, and the maximal deduction is 30 marks even you delay more than 3 days. But there are hard deadlines as we need time to grade and submit grade. The hard deadline for this assignment is 8 May 2020. After this day, we will not grade your assignment.

References

[1] Dong, C., Loy, C. C., He, K., & Tang, X. (2015). Image super-resolution using deep convolutional networks. *IEEE transactions on pattern analysis and machine intelligence*, *38*(2), 295-307.

Useful links

1. PyTorch tutorial: https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html
2. Tensorboard usage: https://pytorch.org/tutorials/intermediate/tensorboard_tutorial.html
3. PyTorch document: https://pytorch.org/docs/stable/index.html