GR5243 Project 3 Main script

for Improved Model (SRCNN)

Group 7

Our PSNR based on our 300 test images is 27.93

Step 0: import libraries and specify directories.

Import libraries and set the working directory to the SRSNN folder. In order to obain reproducible results, random.seed() randomization is used.

In [1]:

```
import cv2
import numpy as np
import tensorflow as tf
import os
import glob
import h5py
import time
import pprint
import random
import math
random.seed(83)
```

```
/anaconda3/lib/python3.6/site-packages/h5py/__init__.py:36: FutureWarning: Conversion of the
second argument of issubdtype from `float` to `np.floating` is deprecated. In future, it will be
treated as `np.float64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
```

Step 1: utilities.

Define funtions used later in the model and main part.

In [2]:

```
####################################
#                                  #
#          FOR TRAINING ONLY       #
#                                  #
####################################

def load_data1(is_train):
    if is_train:
        data_dir1 = os.path.join(os.getcwd(), 'train_set', "LR") # Join the Train dir to current di
rectory
        data1 = glob.glob(os.path.join(data_dir1, "*.jpg"))[:300] + glob.glob(os.path.join(data_dir
1, "*.jpg"))[500:800] + glob.glob(os.path.join(data_dir1, "*.jpg"))[1000:1200]# make set of all
dataset file path

        data_dir2 = os.path.join(os.getcwd(), 'train_set', "HR") # Join the Train dir to current di
rectory
        data2 = glob.glob(os.path.join(data_dir2, "*.jpg"))[:300] + glob.glob(os.path.join(data_dir
2, "*.jpg"))[500:800] + glob.glob(os.path.join(data_dir2, "*.jpg"))[1000:1200] # make set of all
dataset file path
    return data1, data2


def make_sub_data1(data1,data2, padding, config):
    """
        Make the sub_data set
        Args:
            data : the set of all file path
```

```
            padding : the image padding of input to label
            config : the all flags
    """
    sub_input_sequence = []
    sub_label_sequence = []
    for i in range(len(data1)):
        if config.is_train:
            input_=data1[i]
            label_=data2[i]

            input_=cv2.imread(input_)
            input_=cv2.resize(input_,None,fx = 2 ,fy = 2, interpolation = cv2.INTER_CUBIC)
            label_=cv2.imread(label_)

        if len(input_.shape) == 3: # is color
            h, w, c = input_.shape
        else:
            h, w = input_.shape # is grayscale

        nx, ny = 0, 0
        for x in range(0, h - config.image_size + 1, config.stride):
            nx += 1; ny = 0
            for y in range(0, w - config.image_size + 1, config.stride):
                ny += 1

                sub_input = input_[x: x + config.image_size, y: y + config.image_size] # 33 * 33
                sub_label = label_[x + padding: x + padding + config.label_size, y + padding: y + pa
dding + config.label_size] # 21 * 21

                # Reshape the subinput and sublabel
                sub_input = sub_input.reshape([config.image_size, config.image_size, config.c_dim])
                sub_label = sub_label.reshape([config.label_size, config.label_size, config.c_dim])
                # Normialize
                sub_input =  sub_input / 255.0
                sub_label =  sub_label / 255.0

                # Add to sequence
                sub_input_sequence.append(sub_input)
                sub_label_sequence.append(sub_label)

    # NOTE: The nx, ny can be ignore in train
    return sub_input_sequence, sub_label_sequence, nx, ny


def make_data_hf1(input_, label_, config):
    """
        Make input data as h5 file format
        Depending on "is_train" (flag value), savepath would be change.
    """
    # Check the check dir, if not, create one
    if not os.path.isdir(os.path.join(os.getcwd(),config.checkpoint_dir)):
        os.makedirs(os.path.join(os.getcwd(),config.checkpoint_dir))

    if config.is_train:
        savepath = os.path.join(os.getcwd(), config.checkpoint_dir + '/train.h5')

    with h5py.File(savepath, 'w') as hf:
        hf.create_dataset('input', data=input_)
        hf.create_dataset('label', data=label_)


def input_setup1(config):
    """
        Read image files and make their sub-images and saved them as a h5 file format
    """

    # Load data path, if is_train False, get test data
    data1,data2 = load_data1(config.is_train)

    padding = abs(config.image_size - config.label_size) // 2

    # Make sub_input and sub_label, if is_train false more return nx, ny
    sub_input_sequence, sub_label_sequence, nx, ny = make_sub_data1(data1,data2, padding, config)


    # Make list to numpy array. With this transform
    arrinput = np.asarray(sub_input_sequence) # [?, 33, 33, 3]
```

```
    arrlabel = np.asarray(sub_label_sequence) # [?, 21, 21, 3]

    make_data_hf1(arrinput, arrlabel, config)

    return nx, ny


def checkpoint_dir1(config):
    if config.is_train:
        return os.path.join('./{}'.format(config.checkpoint_dir), "train.h5")


def read_data1(path):
    """
        Read h5 format data file

        Args:
            path: file path of desired file
            data: '.h5' file format that contains  input values
            label: '.h5' file format that contains label values
    """
    with h5py.File(path, 'r') as hf:
        input_ = np.array(hf.get('input'))
        label_ = np.array(hf.get('label'))
        return input_, label_


#####################################
#                                   #
#          FOR TESTING ONLY         #
#                                   #
#####################################


# Get the Image
def imread(path):
    img = cv2.imread(path)
    return img


def imsave(image, path, config):
    #checkimage(image)
    # Check the check dir, if not, create one
    if not os.path.isdir(os.path.join(os.getcwd(),config.result_dir)):
        os.makedirs(os.path.join(os.getcwd(),config.result_dir))
    # NOTE: because normial, we need mutlify 255 back
    cv2.imwrite(os.path.join(os.getcwd(),path),image * 255.)


def checkimage(image):
    cv2.imshow('test', image)
    cv2.waitKey(0)


def modcrop(img, scale = 3):
    """
        To scale down and up the original image, first thing to do is to have no remainder while
scaling operation.
    """
    # Check the image is grayscale
    if len(img.shape) ==3:
        h, w, _ = img.shape
        h = (h // scale) * scale
        w = (w // scale) * scale
        img = img[0:h, 0:w, :]
    else:
        h, w = img.shape
        h = (h // scale) * scale
        w = (w // scale) * scale
        img = img[0:h, 0:w]
    return img


def checkpoint_dir(config):
    if config.is_train:
        return os.path.join('./{}'.format(config.checkpoint_dir), 'train.h5')
    else:
```

```
        return os.path.join('./{}'.format(config.checkpoint_dir), 'test.h5')


def prepare_data(dataset='Train',Input_img=''):
    """
        Args:
            dataset: choose train dataset or test dataset
            For train dataset, output data would be ['.../t1.bmp', '.../t2.bmp',..., 't99.bmp']
    """
    if dataset == 'Train':
        data_dir = os.path.join(os.getcwd(), dataset) # Join the Train dir to current directory
        data = glob.glob(os.path.join(data_dir, '*.*')) # make set of all dataset file path
    else:
        if Input_img !='':
            data = [os.path.join(os.getcwd(),Input_img)]
        else:
            data_dir = os.path.join(os.path.join(os.getcwd(), dataset), 'Set5')
            data = glob.glob(os.path.join(data_dir, '*.*')) # make set of all dataset file path
    print(data)
    return data


def load_data(is_train, test_img):
    """
        Args:
            is_train: decides if we choose train dataset or test dataset
            For train dataset, output data would be ['.../t1.bmp', '.../t2.bmp',..., 't99.bmp']
    """
    if is_train:
        data_dir = os.path.join(os.getcwd(), 'Train') # Join the Train dir to current directory
        data = glob.glob(os.path.join(data_dir, '*.*')) # make set of all dataset file path
    else:
        if test_img != '':
            return [os.path.join(os.getcwd(), test_img)]
        data_dir = os.path.join(os.path.join(os.getcwd(), 'Test'), 'Set5')
        data = glob.glob(os.path.join(data_dir, '*.*')) # make set of all dataset file path
    return data


def make_sub_data2(data, padding, config):
    """
        Make the sub_data set
        Args:
            data : the set of all file path
            padding : the image padding of input to label
            config : the all flags
    """
    sub_input_sequence = []
#     sub_label_sequence = []
    for i in range(len(data)):
        input_ = cv2.imread(data[i])
        input_=cv2.resize(input_,None,fx = 2 ,fy = 2, interpolation = cv2.INTER_CUBIC)

        if len(input_.shape) == 3: # is color
            h, w, c = input_.shape
        else:
            h, w = input_.shape # is grayscale

        nx, ny = 0, 0
        for x in range(0, h - config.image_size + 1, config.stride):
            nx += 1; ny = 0
            for y in range(0, w - config.image_size + 1, config.stride):
                ny += 1

                sub_input = input_[x: x + config.image_size, y: y + config.image_size] # 33 * 33
#                 sub_label = label_[x + padding: x + padding + config.label_size, y + padding: y
+ padding + config.label_size] # 21 * 21

                # Reshape the subinput and sublabel
                sub_input = sub_input.reshape([config.image_size, config.image_size, config.c_dim])
#                 sub_label = sub_label.reshape([config.label_size, config.label_size, config.c_di
])
                # Normialize
                sub_input =  sub_input / 255.0
#                 sub_label =  sub_label / 255.0

                # Add to sequence
```

```
                sub_input_sequence.append(sub_input)
#                 sub_label_sequence.append(sub_label)

#     return sub_input_sequence, sub_label_sequence, nx, ny
    return sub_input_sequence, nx, ny


def read_data(path):
    """
        Read h5 format data file

        Args:
            path: file path of desired file
            data: '.h5' file format that contains  input values
            label: '.h5' file format that contains label values
    """
    with h5py.File(path, 'r') as hf:
        input_ = np.array(hf.get('input'))
        label_ = np.array(hf.get('label'))
        return input_, label_


def make_data_hf2(input_, config):
    """
        Make input data as h5 file format
        Depending on 'is_train' (flag value), savepath would be change.
    """
    # Check the check dir, if not, create one
    if not os.path.isdir(os.path.join(os.getcwd(),config.checkpoint_dir)):
        os.makedirs(os.path.join(os.getcwd(),config.checkpoint_dir))

    if config.is_train:
        savepath = os.path.join(os.getcwd(), config.checkpoint_dir + '/train.h5')
    else:
        savepath = os.path.join(os.getcwd(), config.checkpoint_dir + '/test.h5')

    with h5py.File(savepath, 'w') as hf:
        hf.create_dataset('input', data=input_)


def merge(images, size, c_dim):
    """
        images is the sub image set, merge it
    """
    h, w = images.shape[1], images.shape[2]
    print(h, w)
    print(size[0], size[1])
    img = np.zeros((h*size[0], w*size[1], c_dim))
    for idx, image in enumerate(images):
        i = idx % size[1]
        j = idx // size[1]
        img[j * h : j * h + h, i * w : i * w + w, :] = image
    return img


def input_setup2(config):
    """
        Read image files and make their sub-images and saved them as a h5 file format
    """

    # Load data path, if is_train False, get test data
    data = load_data(config.is_train, config.test_img)

    padding = abs(config.image_size - config.label_size) // 2

    # Make sub_input and sub_label, if is_train false more return nx, ny
#     sub_input_sequence, sub_label_sequence, nx, ny = make_sub_data2(data, padding, config)
    sub_input_sequence, nx, ny = make_sub_data2(data, padding, config)

    # Make list to numpy array. With this transform
    arrinput = np.asarray(sub_input_sequence) # [?, 33, 33, 3]
#     arrlabel = np.asarray(sub_label_sequence) # [?, 21, 21, 3]

    make_data_hf2(arrinput, config)
    return nx, ny
```

```
def psnr(img1, img2):
    mse = np.mean( (img1 - img2) ** 2 )
    if mse == 0:
        return 100
    PIXEL_MAX = 255.0
    return 20 * math.log10(PIXEL_MAX / math.sqrt(mse))
```

Step 2: the SRCNN model.

Define a SRCNN `class` which can train and test data.

In the next step, an instance of will be created and used.

In [3]:

```
class SRCNN(object):

    def __init__(self,sess, image_size, label_size,c_dim):
        self.sess = sess
        self.image_size = image_size
        self.label_size = label_size
        self.c_dim = c_dim
        self.build_model()


    def train(self, config):
            # NOTE : if train, the nx, ny are ingnored
            nx, ny = input_setup1(config)
            print(0)
            data_dir = checkpoint_dir1(config)
            print(data_dir)

            input_, label_ = read_data1(data_dir)
            print('input_ =', input_[0:3])
            #print(input_)
            # Stochastic gradient descent with the standard backpropagation
            #self.train_op =
tf.train.GradientDescentOptimizer(config.learning_rate).minimize(self.loss)
            self.train_op = tf.train.AdamOptimizer(learning_rate=config.learning_rate).minimize(sel
f.loss)
            tf.global_variables_initializer().run()
            counter = 0
            time_ = time.time()
            print('time_: ', time_)
            # Train
            if config.is_train:
                print("Now Start Training...")
                for ep in range(config.epoch):
                    # Run by batch images
                    batch_idxs = len(input_) // config.batch_size
                    print('len(input_) =', batch_idxs)
                    for idx in range(0, batch_idxs):
                        batch_images = input_[idx * config.batch_size : (idx + 1) *
config.batch_size]
                        batch_labels = label_[idx * config.batch_size : (idx + 1) *
config.batch_size]
                        counter += 1
                        _, err = self.sess.run([self.train_op, self.loss], feed_dict={self.images:
atch_images, self.labels: batch_labels})

                        if counter % 10 == 0:
                            print("Epoch: [%2d], step: [%2d], time: [%4.4f], loss: [%.8f]" % ((ep+1
), counter, time.time()-time_, err))
                            #print(label_[1] - self.pred.eval({self.images: input_})
[1],'loss:]',err)
                        if counter % 10 == 0:
                            self.save(config.checkpoint_dir, counter)


    def build_model(self):
        self.images = tf.placeholder(tf.float32, [None, self.image_size, self.image_size, self.c_di
m], name='images')
        self.labels = tf.placeholder(tf.float32, [None, self.label_size, self.label_size, self.c_di
m], name='labels')
```

```
        self.weights = {
            'w1': tf.Variable(tf.random_normal([9, 9, self.c_dim, 128], stddev=1e-3), name='w1'),
            'w2': tf.Variable(tf.random_normal([1, 1, 128, 64], stddev=1e-3), name='w2'),
            'w3': tf.Variable(tf.random_normal([5, 5, 64, self.c_dim], stddev=1e-3), name='w3')
        }

        self.biases = {
            'b1': tf.Variable(tf.zeros([128], name='b1')),
            'b2': tf.Variable(tf.zeros([64], name='b2')),
            'b3': tf.Variable(tf.zeros([self.c_dim], name='b3'))
        }

        self.pred = self.model()
        self.loss = tf.reduce_mean(tf.square(self.labels - self.pred))
        self.saver = tf.train.Saver() # To save checkpoint


    def model(self):
        conv1 = tf.nn.relu(tf.nn.conv2d(self.images, self.weights['w1'], strides=[1,1,1,1], padding
='VALID') + self.biases['b1'])
        conv2 = tf.nn.relu(tf.nn.conv2d(conv1, self.weights['w2'], strides=[1,1,1,1], padding='SAME
') + self.biases['b2'])
        conv3 = tf.nn.conv2d(conv2, self.weights['w3'], strides=[1,1,1,1], padding='VALID') + self.
biases['b3'] # This layer don't need ReLU
        return conv3


    def save(self, checkpoint_dir, step):
        """
            To save the checkpoint use to test or pretrain
        """
        model_name = "SRCNN.model"
        model_dir = "%s_%s" % ("srcnn", self.label_size)
        checkpoint_dir = os.path.join(checkpoint_dir, model_dir)

        if not os.path.exists(checkpoint_dir):
            os.makedirs(checkpoint_dir)

        self.saver.save(self.sess,os.path.join(checkpoint_dir, model_name),global_step=step)


    def test(self, config):
        print('Testing...')
        nx, ny = input_setup2(config)
        data_dir = checkpoint_dir(config)
        input_, label_ = read_data(data_dir)
        self.load(config.checkpoint_dir)
        # Test
        result = self.pred.eval({self.images: input_})
        image = merge(result, [nx, ny], self.c_dim)

        base, ext = os.path.basename(config.test_img).split('.')

        test_files =random.sample(files, len(files)//5) # a list of strings (the paths of each imag
e)
        LR_image = imread(os.path.join(os.path.join(os.getcwd(),'train_set',"LR"), base + '.jpg'))
        LR_h = LR_image.shape[1]*2
        LR_w = LR_image.shape[0]*2
        image = cv2.resize(image,(LR_h,LR_w))

        imsave(image, os.path.join(config.result_dir, base + '.png'), config)


    def load(self, checkpoint_dir):
        """
            To load the checkpoint use to test or pretrain
        """
        model_dir = '%s_%s' % ('srcnn', self.label_size)# give the model name by label_size
        checkpoint_dir = os.path.join(checkpoint_dir, model_dir)
        ckpt = tf.train.get_checkpoint_state(checkpoint_dir)

        # Check the checkpoint is exist
        if ckpt and ckpt.model_checkpoint_path:
            ckpt_path = str(ckpt.model_checkpoint_path) # convert the unicode to string
            self.saver.restore(self.sess, os.path.join(os.getcwd(), ckpt_path))
#             print('Success! %s'% ckpt_path)
```

```
        else:
            print('Loading failed.')


    def save(self, checkpoint_dir, step):
        """
            To save the checkpoint use to test or pretrain
        """
        model_name = 'SRCNN.model'
        model_dir = '%s_%s' % ('srcnn', self.label_size)
        checkpoint_dir = os.path.join(checkpoint_dir, model_dir)

        if not os.path.exists(checkpoint_dir):
            os.makedirs(checkpoint_dir)

        self.saver.save(self.sess,
                        os.path.join(checkpoint_dir, model_name),
                        global_step=step)
```

Step 3: the main part.

In this cell, we first establish a `class this_config()` which decides all the parameters of the model.

We then create an instance `class SRCNN()` and use it to train and/or test data.

Instruction

- For training and testing (with PSNR computed), run the cell below.
- For testing (with PSNR computed only), comment line 27 `srcnn.train(FLAGS)` and run the cell below.
- For testing (with PSNR computed only) with your own (.jpg) images, go to line 34 and replace `os.getcwd(), 'train_set', 'LR'` with the directory of your own test images folder.

In [4]:

```
class this_config():
    def __init__(self, is_train=True):
        self.epoch = 12
        self.image_size = 33
        self.label_size = 21
        self.c_dim = 3
        self.is_train = is_train
        self.scale = 3
        self.stride = 21
        self.checkpoint_dir = "checkpoint1"
        self.learning_rate = 1e-4
        self.batch_size = 128
        self.result_dir = 'result'
        self.test_img = '' # Do not change this.

arg = this_config()

with tf.Session() as sess:
    print("Start running...")
    FLAGS = arg
    srcnn = SRCNN(sess,
                  image_size = FLAGS.image_size,
                  label_size = FLAGS.label_size,
                  c_dim = FLAGS.c_dim)

    # Training
#     srcnn.train(FLAGS)

    # Testing
    print(os.getcwd())
    files = glob.glob(os.path.join(os.getcwd(), 'train_set', 'LR', '*.jpg'))
#     test_files = files[400:500] + files[900:1000] + files[1400:1500]
    # a list of strings (the paths of each image)
    test_files = glob.glob(os.path.join(os.getcwd(), 'train_set', 'LR', '*.jpg'))
    FLAGS.is_train = False
    count = 1
    for f in test_files:
        FLAGS.test_img = f
        print('Saving ', count, '/', len(test_files), ': ', FLAGS.test_img, '\n')
```

```
        count += 1
        srcnn.test(FLAGS)

    # PSNR
    #result_files = glob.glob(os.path.join(os.getcwd(), 'result', '*.png'))
    res_imgs = glob.glob(os.path.join(os.getcwd(), 'result', '*.png'))
    psnr_total = []
#     print(test_files[0:3])
#     print(res_imgs[0:3])
    for test_img, res_img in zip(test_files, res_imgs):
        print(test_img)
        print(res_img)
        img1 = cv2.imread(test_img, cv2.IMREAD_COLOR)
        img1 = cv2.cvtColor(img1, cv2.COLOR_BGR2YCrCb)[6: -6, 6: -6, 0]
        img2 = cv2.imread(res_img, cv2.IMREAD_COLOR)
        img2 = cv2.cvtColor(img2, cv2.COLOR_BGR2YCrCb)[6: -6, 6: -6, 0]
        img1 = cv2.resize(img1,(img2.shape[1],img2.shape[0]))
        psnr_total.append(psnr(img1,img2))

    print('The PSNR between the ground truth and the test is: ', np.mean(psnr_total))

    # Transcode .png results into .jpg format
    for res_img in res_imgs:
        img = cv2.imread(res_img)
        cv2.imwrite(res_img[:-3] + 'jpg', img)
```