Superresolution using an efficient sub-pixel convolutional neural network

This example illustrates how to use the efficient sub-pixel convolution layer described in "Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network" - Shi et al. for increasing spatial resolution within your network for tasks such as superresolution.

```
usage: main.py [-h] --upscale_factor UPSCALE_FACTOR [--batchSize BATCHSI
               [--testBatchSize TESTBATCHSIZE] [--nEpochs NEPOCHS] [--lr
               [--cuda] [--threads THREADS] [--seed SEED]

PyTorch Super Res Example

optional arguments:
  -h, --help            show this help message and exit
  --upscale_factor      super resolution upscale factor
  --batchSize           training batch size
  --testBatchSize       testing batch size
  --nEpochs             number of epochs to train for
  --lr                  Learning Rate. Default=0.01
  --cuda                use cuda
  --threads             number of threads for data loader to use Default
  --seed                random seed to use. Default=123
```

This example trains a super-resolution network on the BSD300 dataset, using crops from the 200 training images, and evaluating on crops of the 100 test images. A snapshot of the model after every epoch with filename model_epoch_.pth

Example Usage:

Train

```
python main.py --upscale_factor 3 --batchSize 4 --testBatchSize
100 --nEpochs 30 --lr 0.001
```

Super Resolve

```
python super_resolve.py --input_image dataset/BSDS300/images/
test/16077.jpg --model model_epoch_500.pth --output_filename
out.png
```

```
from os.path import exists, join, basename
from os import makedirs, remove
from six.moves import urllib
import tarfile
from torchvision.transforms import Compose,
CenterCrop, ToTensor, Resize

from dataset import DatasetFromFolder


def download_bsd300(dest="dataset"):
    output_image_dir = join(dest, "BSDS300/images")

    if not exists(output_image_dir):
        makedirs(dest)
        url = "http://www2.eecs.berkeley.edu/
Research/Projects/CS/vision/bsds/BSDS300-images.tgz"
        print("downloading url ", url)

        data = urllib.request.urlopen(url)

        file_path = join(dest, basename(url))
        with open(file_path, 'wb') as f:
            f.write(data.read())

        print("Extracting data")
        with tarfile.open(file_path) as tar:
            for item in tar:
                tar.extract(item, dest)

        remove(file_path)

    return output_image_dir


def calculate_valid_crop_size(crop_size,
upscale_factor):
```

```
    return crop_size - (crop_size % upscale_factor)


def input_transform(crop_size, upscale_factor):
    return Compose([
        CenterCrop(crop_size),
        Resize(crop_size // upscale_factor),
        ToTensor(),
    ])


def target_transform(crop_size):
    return Compose([
        CenterCrop(crop_size),
        ToTensor(),
    ])


def get_training_set(upscale_factor):
    root_dir = download_bsd300()
    train_dir = join(root_dir, "train")
    crop_size = calculate_valid_crop_size(256,
upscale_factor)

    return DatasetFromFolder(train_dir,

input_transform=input_transform(crop_size,
upscale_factor),

target_transform=target_transform(crop_size))


def get_test_set(upscale_factor):
    root_dir = download_bsd300()
    test_dir = join(root_dir, "test")
    crop_size = calculate_valid_crop_size(256,
upscale_factor)
```

```
    return DatasetFromFolder(test_dir,

input_transform=input_transform(crop_size,
upscale_factor),

target_transform=target_transform(crop_size))
import torch.utils.data as data

from os import listdir
from os.path import join
from PIL import Image


def is_image_file(filename):
    return any(filename.endswith(extension) for
extension in [".png", ".jpg", ".jpeg"])


def load_img(filepath):
    img = Image.open(filepath).convert('YCbCr')
    y, _, _ = img.split()
    return y


class DatasetFromFolder(data.Dataset):
    def __init__(self, image_dir,
input_transform=None, target_transform=None):
        super(DatasetFromFolder, self).__init__()
        self.image_filenames = [join(image_dir, x)
for x in listdir(image_dir) if is_image_file(x)]

        self.input_transform = input_transform
        self.target_transform = target_transform

    def __getitem__(self, index):
        input =
```

```
load_img(self.image_filenames[index])
        target = input.copy()
        if self.input_transform:
            input = self.input_transform(input)
        if self.target_transform:
            target = self.target_transform(target)

        return input, target

    def __len__(self):
        return len(self.image_filenames)
from __future__ import print_function
import argparse
from math import log10

import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
from model import Net
from data import get_training_set, get_test_set

# Training settings
parser =
argparse.ArgumentParser(description='PyTorch Super
Res Example')
parser.add_argument('--upscale_factor', type=int,
required=True, help="super resolution upscale
factor")
parser.add_argument('--batchSize', type=int,
default=64, help='training batch size')
parser.add_argument('--testBatchSize', type=int,
default=10, help='testing batch size')
parser.add_argument('--nEpochs', type=int,
default=2, help='number of epochs to train for')
parser.add_argument('--lr', type=float,
default=0.01, help='Learning Rate. Default=0.01')
```

```
parser.add_argument('--cuda', action='store_true',
help='use cuda?')
parser.add_argument('--threads', type=int,
default=4, help='number of threads for data loader
to use')
parser.add_argument('--seed', type=int,
default=123, help='random seed to use. Default=123')
opt = parser.parse_args()

print(opt)

if opt.cuda and not torch.cuda.is_available():
    raise Exception("No GPU found, please run
without --cuda")

torch.manual_seed(opt.seed)

device = torch.device("cuda" if opt.cuda else "cpu")

print('===> Loading datasets')
train_set = get_training_set(opt.upscale_factor)
test_set = get_test_set(opt.upscale_factor)
training_data_loader =
DataLoader(dataset=train_set,
num_workers=opt.threads, batch_size=opt.batchSize,
shuffle=True)
testing_data_loader = DataLoader(dataset=test_set,
num_workers=opt.threads,
batch_size=opt.testBatchSize, shuffle=False)

print('===> Building model')
model =
Net(upscale_factor=opt.upscale_factor).to(device)
criterion = nn.MSELoss()

optimizer = optim.Adam(model.parameters(),
lr=opt.lr)
```

```
def train(epoch):
    epoch_loss = 0
    for iteration, batch in
enumerate(training_data_loader, 1):
        input, target = batch[0].to(device),
batch[1].to(device)

        optimizer.zero_grad()
        loss = criterion(model(input), target)
        epoch_loss += loss.item()
        loss.backward()
        optimizer.step()

        print("===> Epoch[{}]({}/{}): Loss: {:.
4f}".format(epoch, iteration,
len(training_data_loader), loss.item()))

    print("===> Epoch {} Complete: Avg. Loss: {:.
4f}".format(epoch, epoch_loss /
len(training_data_loader)))


def test():
    avg_psnr = 0
    with torch.no_grad():
        for batch in testing_data_loader:
            input, target = batch[0].to(device),
batch[1].to(device)

            prediction = model(input)
            mse = criterion(prediction, target)
            psnr = 10 * log10(1 / mse.item())
            avg_psnr += psnr
    print("===> Avg. PSNR: {:.4f}
dB".format(avg_psnr / len(testing_data_loader)))
```

```
def checkpoint(epoch):
    model_out_path =
"model_epoch_{}.pth".format(epoch)
    torch.save(model, model_out_path)
    print("Checkpoint saved to
{}".format(model_out_path))

for epoch in range(1, opt.nEpochs + 1):
    train(epoch)
    test()
    checkpoint(epoch)
import torch
import torch.nn as nn
import torch.nn.init as init


class Net(nn.Module):
    def __init__(self, upscale_factor):
        super(Net, self).__init__()

        self.relu = nn.ReLU()
        self.conv1 = nn.Conv2d(1, 64, (5, 5), (1,
1), (2, 2))
        self.conv2 = nn.Conv2d(64, 64, (3, 3), (1,
1), (1, 1))
        self.conv3 = nn.Conv2d(64, 32, (3, 3), (1,
1), (1, 1))
        self.conv4 = nn.Conv2d(32, upscale_factor
** 2, (3, 3), (1, 1), (1, 1))
        self.pixel_shuffle =
nn.PixelShuffle(upscale_factor)

        self._initialize_weights()

    def forward(self, x):
```

```
        x = self.relu(self.conv1(x))
        x = self.relu(self.conv2(x))
        x = self.relu(self.conv3(x))
        x = self.pixel_shuffle(self.conv4(x))
        return x

    def _initialize_weights(self):
        init.orthogonal_(self.conv1.weight,
init.calculate_gain('relu'))
        init.orthogonal_(self.conv2.weight,
init.calculate_gain('relu'))
        init.orthogonal_(self.conv3.weight,
init.calculate_gain('relu'))
        init.orthogonal_(self.conv4.weight)
from __future__ import print_function
import argparse
import torch
from PIL import Image
from torchvision.transforms import ToTensor

import numpy as np

# Training settings
parser =
argparse.ArgumentParser(description='PyTorch Super
Res Example')
parser.add_argument('--input_image', type=str,
required=True, help='input image to use')
parser.add_argument('--model', type=str,
required=True, help='model file to use')
parser.add_argument('--output_filename', type=str,
help='where to save the output image')
parser.add_argument('--cuda', action='store_true',
help='use cuda')
opt = parser.parse_args()

print(opt)
```

```
img = Image.open(opt.input_image).convert('YCbCr')
y, cb, cr = img.split()

model = torch.load(opt.model)
img_to_tensor = ToTensor()
input = img_to_tensor(y).view(1, -1, y.size[1],
y.size[0])

if opt.cuda:
    model = model.cuda()
    input = input.cuda()

out = model(input)
out = out.cpu()
out_img_y = out[0].detach().numpy()
out_img_y *= 255.0
out_img_y = out_img_y.clip(0, 255)
out_img_y = Image.fromarray(np.uint8(out_img_y[0]),
mode='L')

out_img_cb = cb.resize(out_img_y.size,
Image.BICUBIC)
out_img_cr = cr.resize(out_img_y.size,
Image.BICUBIC)
out_img = Image.merge('YCbCr', [out_img_y,
out_img_cb, out_img_cr]).convert('RGB')

out_img.save(opt.output_filename)
print('output image saved to ', opt.output_filename)
```