人工智能丨面向图像超高分辨率的卷积神经网络示例 - 教程

原创：Alberto V.英特尔开发人员专区9 月 28 日

本教程介绍了一种面向单图像超高分辨率的 CNN（卷积神经网络）的实施方法，使用 Caffe* 深度学习框架和英特尔® Python* 分发包在英特尔® 架构上对它进行了优化，以支持我们利用英特尔处理器和英特尔库加速该 CNN 的训练与测试。

在本教程中使用的 CNN 是快速超高分辨率卷积神经网络 (FSRCNN)，它基于 [1] 和 [2] 的研究成果，他们提出了一种使用 CNN 执行单图像 SR 的新方法。我们在相关文章（《面向图像超高分辨率的卷积神经网络示例》）中更详细地介绍了该网络及其前代产品（超高分辨率卷积神经网络 (SRCNN)）。

FSRCNN 结构

根据相关文章和 [2] 的描述，FSRCNN 包含以下操作：

1. **特征提取：**直接从低分辨率 (LR) 图像中提取一系列特征图。
2.
3. **缩小：**通过使用少量的过滤器（相比特征提取使用的过滤器数量），减小特征向量的维度（从而降低参数数量）。
4.
5.

非线性映射：将代表 LR 图像的特征图映射至高分辨率（HR）图像。使用若干映射层执行该步骤，小于 SCRNN 中使用的过滤器尺寸。

6.
7.

扩大：增大特征向量的维度。为了更准确地生成 HR 图像，该操作执行与缩小层相反的操作。

8.
9.

去卷积：根据 HR 特性生成 HR 图像。

10.

FSRCNN (56, 12, 4) 模型的结构（根据 [2] 的报告和相关文章的描述，它的性能最佳）如图 1 所示。它的 LR 特征维度为 56（第一层卷积层和去卷积层的过滤器数量），包含 12 个缩小过滤器（网络中间层的过滤器数量，执行映射操作），映射深度为 4（在 LR 和 HR 特征空间内执行映射的卷积层数量）。

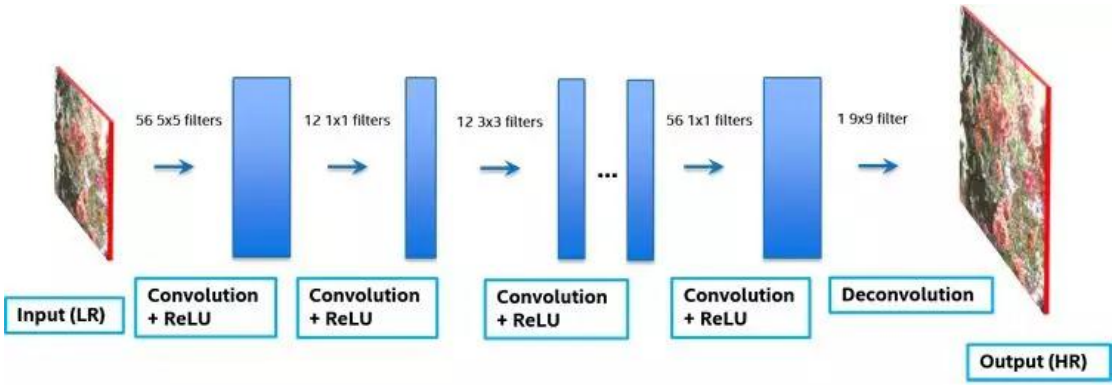

图 1：FSRCNN (56 ,12, 4) 的结构。

训练与测试数据准备

训练与测试该实施的数据集可从作者 [2] 的 网站中获取。训练数据集包含 91 张不同大小的图像。共有两个测试数据集：Set 5（包含 5 张图

像）和 Set 14（包含 14 张图像）。在本教程中，训练与测试数据集被压缩到 HDF5* 文件（https://support.hdfgroup.org/）中，可以在 Caffe 框架中高效地使用它。如欲进一步了解面向英特尔® 架构优化的 Caffe，请访问《使用面向英特尔® 架构优化的 Caffe* 管理深度学习网络》和《方案：基于英特尔® 至强融核™ 处理器 x 200 的面向深度学习优化的 Caffe*》。

训练与测试数据集均需要预处理，如下所示：

- **训练数据集：**首先，图像被转换为 YCrCb 颜色空间 (https://en.wikipedia.org/wiki/YCbCr)，本教程仅使用了亮度通道 Y。因子 k 对 91 张训练数据集图像逐一进行了下采样，k 是超高分辨率所需的比例因子，通过这种方式获得了对应的 LR 与 HR 图像对。接下来，使用步长 s 将每一对图像 (LR/HR) 裁切为小型子图像的子集，最终，对于 91 张原始训练图像，每张均生成 N 对 LR/HR 子图像。裁切图像进行训练的原因是我们想使用小区域内的 LR 和 HR 本地特性来训练模型。子图像的数量 N 取决于子图像的大小和步长 s。[2] 的作者在他们的实验中将 LR 子图像的大小定义为 7x7 像素，将 HR 子图像的大小定义为 21x21 像素，对应比例因子 k=3。
-
- **测试数据集：**测试数据集中每张图像的处理方式都与训练数据集相同，但是步长 s 可以大于训练数据集使用的步长，以加快测试流程。
-

以下 Python 代码段显示了生成训练与测试数据集的一种可能方法。我们使用 OpenCV* (http://opencv.org/) 来处理与预处理图像。第一个代码段展示了如何根据训练数据集（包含 91 张图像）中的一张原始图像生成一组 HR 与 LR 子图像对，在特定情况下，比例因子 k=3，步长 = 19：

```
import os
import sys
import numpy as np
import h5py

sys.path.append('$CAFFE_HOME/opencv-2.4.13/release/lib/')
import cv2

# Parameters
scale = 3
stride = 19
size_ground = 19
size_input = 7
size_pad = 2

#Read image to process
image = cv2.imread('<PATH TO FILES>/Train/t1.bmp')

#Change color-space to YCR_CB
```

```
image_ycrcb = cv2.cvtColor(image, cv2.COLOR_RGB2YCR_CB)
image_ycrcb = image_ycrcb[:,:,0]
image_ycrcb = image_ycrcb.reshape((image_ycrcb.shape[0], image_ycrcb.shape[1], 1))

#Compute size of LR images and resize HR images to a multiple of scale
height_small = int(height/scale)
width_small  = int(width/scale)

image_pair_HR = cv2.resize(image_ycrcb, (width_small*scale, height_small*scale) )
image_pair_LR = cv2.resize(image_ycrcb, (width_small, height_small) )

# Declare tensors to hold 1024 LR-HR subimage pairs
input_HR = np.zeros((size_ground, size_ground, 1, 1024))
input_LR = np.zeros((size_input + 2*size_pad, size_input + 2*size_pad, 1, 1024))

height, width = image_pair_HR.shape[:2]

#Iterate over the train image using the specified stride and create LR-HR subimage pairs
count = 0
```

```
for i in range(0, height-size_ground+1, stride):
    for j in range(0, width-size_ground+1, stride):
                    subimage_HR   =   image_pair_HR[i:i+size_ground,
j:j+size_ground]
        count = count + 1
        height_small = size_input
        width_small   = size_input
            subimage_LR  =  cv2.resize(subimage_HR,  (width_small,
height_small) )

          np.lib.pad(subimage_LR, ((size_pad, 2), (2, 2)), 'constant',
constant_values=(0.0))
        input_HR[:,:,0,count-1] = subimage_HR
        input_LR[:,:,0,count-1] = np.lib.pad(subimage_LR, ((size_pad,
2), (2, 2)), 'constant', constant_values=(0.0))
```

下一个代码段展示了如何使用 python h5py 模块创建一个 hdf5 文件，该文件包含之前代码段创建的 HR 和 LR 子图像对：

```
(...)
#Create an hdf5 file
with h5py.File('train1.h5','w') as H5:
    H5.create_dataset( 'Input', data=input_LR )
    H5.create_dataset( 'Ground', data=input_HR )
(...)
```

前两个代码段可用于创建包含整个训练集（由 91 张图像组成）的 hdf5 文件，以便在 Caffe 训练中使用。

FSRCNN 训练

使用 英特尔® Caffe* 分发包实施前文所述的参考模型，它们经过优化，可以在英特尔 CPU 上运行。该框架的基本知识简介和安装指导详见英特尔® Nervana 人工智能研究院。

在 Caffe 中，使用 protobuf 文件定义模型。FSRCNN 模型可从作者 [2] 的网站中下载。以下代码段展示了作者 [2] 定义的 FSRCNN (56, 12, 4) 模型的输入层和第一个卷积层。输入层从文件中读取训练/测试数据，这些文件的名称被 $HOME_CAFFE/examples 目录（train.txt 和 test.txt）中的源文件所定义。训练的批量大小为 128。

```
name: "SR_test"
layer {
  name: "data"
  type: "HDF5Data"
  top: "data"
  top: "label"
  hdf5_data_param {
    source: "examples/FSRCNN/train.txt"
    batch_size: 128
  }
  include: { phase: TRAIN }
```

```
}
layer {
  name: "data"
  type: "HDF5Data"
  top: "data"
  top: "label"
  hdf5_data_param {
    source: "examples/FSRCNN/test.txt"
    batch_size: 2
  }
  include: { phase: TEST }
}

layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 0.1
```

```
  }
  convolution_param {
    num_output: 56
    kernel_size: 5
    stride: 1
    pad: 0
    weight_filler {
      type: "gaussian"
      std: 0.0378
    }
    bias_filler {
      type: "constant"
      value: 0
    }
  }
}
(...)
```

为了训练上述模型，[2] 的作者在他们的网站中提供了一个包含训练参数和 protobuf 网络定义文件位置的解算器 protobuf 文件：

```
# The train/test net protocol buffer definition
net: "examples/FSRCNN/FSRCNN.prototxt"
test_iter: 100
```

Carry out testing every 500 training iterations.

test_interval: 5000

The base learning rate, momentum and the weight decay of the network.

#base_lr: 0.005

base_lr: 0.001

momentum: 0.9

weight_decay: 0

Learning rate policy

lr_policy: "fixed"

Display results every 100 iterations

display: 1000

Maximum number of iterations

max_iter: 1000000

write intermediate results (snapshots)

snapshot: 5000

snapshot_prefix: "examples/FSRCNN/RESULTS/FSRCNN-56_12_4"

solver mode: CPU or GPU

solver_mode: CPU

如上所示，解算器将使用以下参数训练被模型定义文件 FSRCNN.prototxt 定义的网络：

•

测试间隔为每 5000 次迭代，100 指的是测试应该执行的前向传递数量。

-

- 基本学习速度为 0.005，学习速度策略是固定的，这意味着学习速度不会随着时间而变化。动量为 0.9（常见的选择），weight_decay 为 0（没有通过规整化来惩罚大权重）。

-

- 每隔 5000 次迭代，向磁盘写入中间结果（快照），最大迭代数量（训练停止时）为 1000000。

-

- 快照结果将被写入 examples/FSRCNN/RESULTS 目录（假设我们在安装目录 $CAFFE_ROOT 中运行 Caffe）。模型文件（包含经过训练的权重）的前缀为字符串“FSRCNN-56_12_4”。

- 鼓励读者实验不同的参数。定义较小的最大迭代数量，探索测试误差是如何下降的，以及比较不同参数组之间的速度是一种实用的方法。

网络定义和解算器文件一旦准备就绪，就可以通过运行 build/tools 目录中的 caffe 命令开始训练：

export CAFFE_ROOT=< Path to caffe >

$CAFFE_ROOT/build/tools/caffe train -engine "MKL2017" –solver \ $CAFFE_ROOT/examples/FSRCNN//FSRCNN_solver.prototxt 2>$CAFFE_ROOT/examples/FSRCNN/output.log

使用保存的快照重新开始训练

开始训练 CNN 后，根据快照参数指定的频率将网络参数（权重）写入磁盘。Caffe 将在每个快照创建两个文件：

FSRCNN-56_12_4_iter_1000000.caffemodel

FSRCNN-56_12_4_iter_1000000.solverstate

模型文件包含与指定迭代相对应的学习模型参数，被序列化为二进制协议缓冲区文件。解算器状态文件是一个状态快照，包含了在快照时期恢复解算器状态所需的所有信息。该文件支持我们从快照重新开始训练，而不是从头开始。例如，假设我们运行了 100 万次迭代，然后，我们意识到我们需要另外运行 50 万次迭代，以进一步降低测试误差。我们可以使用 100 万次迭代提取的快照来重新开始训练：

$CAFFE_ROOT/build/tools/caffe train -engine "MKL2017" –solver\

$CAFFE_ROOT/examples/FSRCNN//FSRCNN_solver.prototxt –snapshot\

$CAFFE_ROOT/examples/FSRCNN/RESULTS/FSRCNN-56_12_4_iter_1000000.solverstate\

2>$CAFFE_ROOT/examples/FSRCNN/output_resume.log

那么，在达到解算器文件指定的新迭代次数（在本示例中，为 1500000 次）之前，新的训练将一直运行。

使用预训练参数的 FSRCNN 测试

一旦我们拥有了经过训练的模型，便可以用它在输入 LR 图像上执行超高分辨率。只要生成了模型快照，我们可以在训练过程中随时测试网络。

在实践中，我们可以使用我们训练的超高分辨率模型来增加任何图像或视频的分辨率。但是，在本教程中，我们想要在 LR 图像中测试经过训练的模型，为此，我们以 HR 图像为参照。我们将使用来自 Set5 测试数据集的样本图像，它不仅被 [1] 和 [2] 所采用，在其他文章中，它也常用来测试 SR 模型。

为了执行测试 ,我们将样本图像(蝴蝶)用作参考标准。为了创建输入 LR 图像，我们将模糊与下采样参考标准图像，并通过它为经过训练的网络提供反馈。一旦我们正向运行包含输入图像的网络，并且获得了超高分辨率图像作为输出，我们将比较 3 张图像（参考标准、LR 和超高分辨率），以直观地评估我们训练的 SR 网络的性能。

可以通过几种方式实施上述测试流程。例如，以下 Python 脚本使用面向图像处理的 OpenCV 库来实施测试流程：

```
import os
    import sys
    import numpy as np

    #Set up caffe root directory and add to path
    caffe_root = '$APPS/caffe/'
    sys.path.insert(0, caffe_root + 'python')
```

```
 sys.path.append('opencv-2.4.13/release/lib/')

import cv2
import caffe

# Parameters
scale = 3

#Create Caffe model using pretrained model
net = caffe.Net(caffe_root + 'FSRCNN_predict.prototxt',
                                                caffe_root +
'examples/FSRCNN/RESULTS/FSRCNN-56_12_4_iter_300000.caffe
model', caffe.TRAIN)

#Input directories
input_dir = caffe_root + 'examples/SRCNN/DATA/Set5/'

#Input ground truth image
            im_raw = cv2.imread(caffe_root +
'/examples/SRCNN/DATA/Set5/butterfly.bmp')

#Change format to YCR_CB
ycrcb = cv2.cvtColor(im_raw, cv2.COLOR_RGB2YCR_CB)
im_raw = ycrcb[:,:,0]
im_raw = im_raw.reshape((im_raw.shape[0], im_raw.shape[1], 1))
```

```
  #Blur image and resize to create input for network
  im_blur = cv2.blur(im_raw, (4,4))
     im_small = cv2.resize(im_blur, (int(im_raw.shape[0]/scale),
int(im_raw.shape[1]/scale)))

         im_raw = im_raw.reshape((1, 1, im_raw.shape[0],
im_raw.shape[1]))
         im_blur = im_blur.reshape((1, 1, im_blur.shape[0],
im_blur.shape[1]))
        im_small = im_small.reshape((1, 1, im_small.shape[0],
im_small.shape[1]))

  im_comp = im_blur
  im_input = im_small

  #Set mode to run on CPU
  caffe.set_mode_cpu()

  #Copy input image data to net structure
  c1,c2,h,w = im_input.shape
  net.blobs['data'].data[...]= im_input

  #Run forward pass
  out = net.forward()
```

```
#Extract output image from net, change format to int8 and reshape
mat = out['conv3'][0]
mat = (mat[0,:,:]).astype('uint8')

im_raw = im_raw.reshape((im_raw.shape[2], im_raw.shape[3]))
im_blur = im_blur.reshape((im_blur.shape[2], im_blur.shape[3]))
im_comp = im_blur.reshape((im_comp.shape[2], im_comp.shape[3]))

#Display original (ground truth), blurred and restored images
cv2.imshow("image",im_raw)
cv2.imshow("image2",im_comp)
cv2.imshow("image3",mat)
cv2.waitKey()

cv2.destroyAllWindows()
```

在测试图像上运行上述脚本显示如图 2 所示的输出。建议读者尝试该网络，对参数进行优化，以获得更好的超高分辨率结果。

图 2：测试经过训练的 FSRCNN。左图为参考标准图像。中间的图像为经过模糊处理与下采样的参考标准图像。右图为使用模型快照进行 300000 次迭代的超高分辨率图像。

总结

在这个简短的教程中，我们展示了如何训练与测试面向超高分辨率的 CNN。我们所说的 CNN 是快速超高分辨率卷积神经网络 (FSRCNN) [2]，相关文章（《面向图像超高分辨率的卷积神经网络示例》）对它进行了更详尽的介绍。本教程选择该 CNN 的原因是它相对简单，性能卓越，以及作者在超高分辨率 CNN 领域所从事的研究非常重要。最新的文章介绍了几款全新的超高分辨率 CNN 架构，将它们的性能与 FSRCNN 或 FSRCNN 的前代产品 SRCNN [1] 进行了对比，后两者由相同的作者创建。

本教程中的训练与测试由英特尔® 至强® 处理器和英特尔® 至强融核™ 处理器执行，使用了英特尔 Caffe 分发包深度学习框架和英特尔

Python 分发包，它们经过优化，可以在英特尔至强处理器和英特尔至强融核处理器上运行。

基于深度学习的图像/视频超高分辨率是计算机视觉领域令人振奋的成果。建议读者尝试该网络和新推出的架构，并使用自己的图像和视频测试它们。如欲利用面向机器学习和深度学习的英特尔优化工具，请访问 英特尔® 开发人员专区（英特尔® DZ）。

参考文献

[1] C. Dong，C. C. Loy，K. He 和 X. Tang，“学习面向图像超高分辨率的深度卷积网络”，2014 年。

[2] C. Dong，C. C. Loy 和 X. Tang，“加速超高分辨率卷积神经网络”，2016 年。