浙江大学

浙江大学实验报告

ImgRecovery

学号	专业班级	姓名	性别
3150104669	混合 1505	张子健	男

指导教师：吴飞

电子邮件地址：3150104669@zju.edu.cn

实验日期：2018 年 5 月 12 日

一、 实验目的与要求

1. 给定 3 张受损图像，尝试恢复他们的原始图像。

2. 原始图像包含 1 张黑白图像（A.png）和 2 张彩色图像（B.png, C.png）。
3. 受损图像(X)是由原始图像($I \in \mathcal{R}^{H*W*C}$)添加了不同噪声遮罩(noise masks)（$M \in \mathcal{R}^{H*W*C}$）得到的（$X = I \odot M$），其中⊙是逐元素相乘。
4. 噪声遮罩仅包含{0,1}值。对应原图（A/B/C）的噪声遮罩的每行分别用 0.8/0.4/0.6 的噪声比率产生的，即噪声遮罩每个通道每行 80%/40%/60%的像素值为 0，其他为 1。
5. Code：目前提供的实现代码，利用逐行回归实现图像恢复，效果一般。鼓励大家开发更有效的恢复方法。
6. Data：提供的三幅待恢复图像，提供给大家。要求设计恢复算法，得到恢复图像，和实验报告一起提交。
7. 程序语言必须使用 Python。

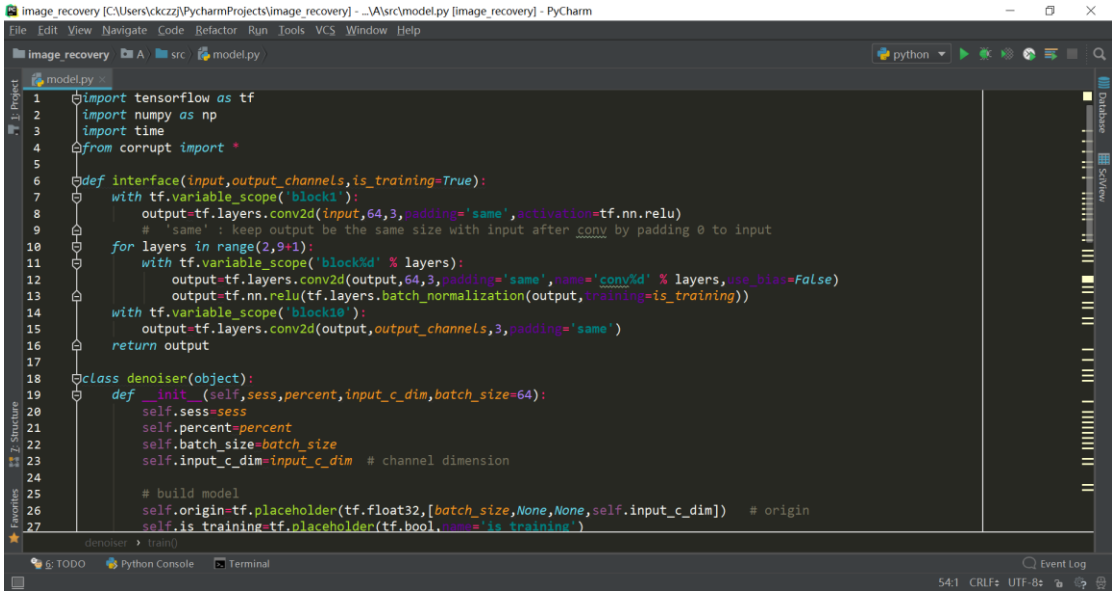
二、 实验环境与分工

1. 开发环境

PC Windows 10 Jet Brains PyCharm 免费教育版

PYTHON_VERSION="2.7"

主要使用的库有 numpy 和 tensorflow，以及图片的读写库 scipy.misc



2. 运行环境

GPU 服务器，由 4 块 GTX 1080 GPU 进行并行计算。

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 390.48                 Driver Version: 390.48                    |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  GeForce GTX 108...  Off  | 00000000:02:00.0 Off |                  N/A |
|  0%   39C    P0    60W / 250W |      0MiB / 11178MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   1  GeForce GTX 108...  Off  | 00000000:03:00.0 Off |                  N/A |
|  0%   42C    P5    16W / 250W |      0MiB / 11178MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   2  GeForce GTX 108...  Off  | 00000000:82:00.0 Off |                  N/A |
|  0%   37C    P0    59W / 250W |      0MiB / 11178MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   3  GeForce GTX 108...  Off  | 00000000:83:00.0 Off |                  N/A |
|  0%   42C    P5    18W / 250W |      0MiB / 11178MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
```

本次实验所有工作均由我本人独立完成。

三、 实验内容

1. 深度学习

计算机视觉和自然语言处理是当下深度学习的两个主要研究和应用领域。

Alex 在 2012 年提出的 AlexNet 网络结构模型引爆了神经网络的应用热潮，并赢得了 2012 届 ImageNet(图像识别大赛)的冠军，使得 CNN 成为在图像分类上的核心算法模型。

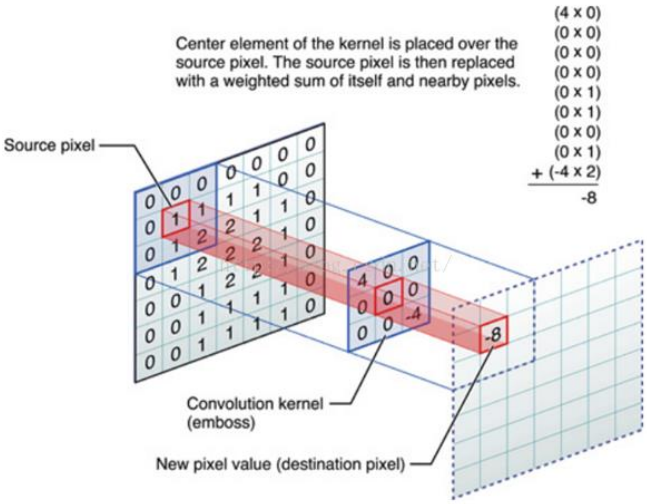
在图像处理方面，CNN 对于传统机器学习的优势在于，它能自动提取图像的特征并进行学习，而不是局限于人为的特征输入。

所以本次实验我选择使用 CNN 完成。

大多数的深度学习任务要考虑的问题有网络架构、目标函数、训练技巧以及训练集等，所以我将从这几个方面介绍我的实验方法。

2. 网络架构

我的网络架构以 CNN 为基础，CNN 的示意图如下:



我的网络架构是一个 n 层(层数可变，n 作为一个超参数)CNN 网络，隐藏层都使用 64 个 filter，卷积核大小都为 3*3，激活函数都为 relu，为了保持图像大小不变，在每层 CNN 运算前，都会对前一层的输出做 padding 0 的操作。输出层使用 input_channel(输入图像的通道个数)个 filter, 卷积核大小仍为 3*3，输出恢复图像。

上述的 CNN 层结构被 tensorflow 封装为函数，可以直接调用。

3. Loss Function

由于实验中给出了测评结果的函数，所以我直接使用各通道所有像素差值的二范数的和作为损失函数进行训练。

4. 训练数据集

我的训练方法也很简单，将训练集中的无噪声的图像输入网络，网络自动按照本实验的加噪方法对图像进行加噪，然后将加噪图像输入网络，得到输出后与原来的无噪声的图像计算损失，由 tensorflow 自动优化网络参数，最小化损失函数。

显然，由于每次网络会自动对输入图像进行加噪(即按比例随机置 0)，所以训练数据几乎是无穷的且不同的。

我选取了一个黑白图像数据集(400 张 180*180*1 的黑白图片)训练恢复黑白噪声图像，该数据集多用于图像处理领域。对于 RGB 图像，由于找不到合适的风景图数据集，所以我使用了 Cifar10 数据集，从中选取了 3000 张 32*32*3 的图像训练恢复 RGB 噪声图像。

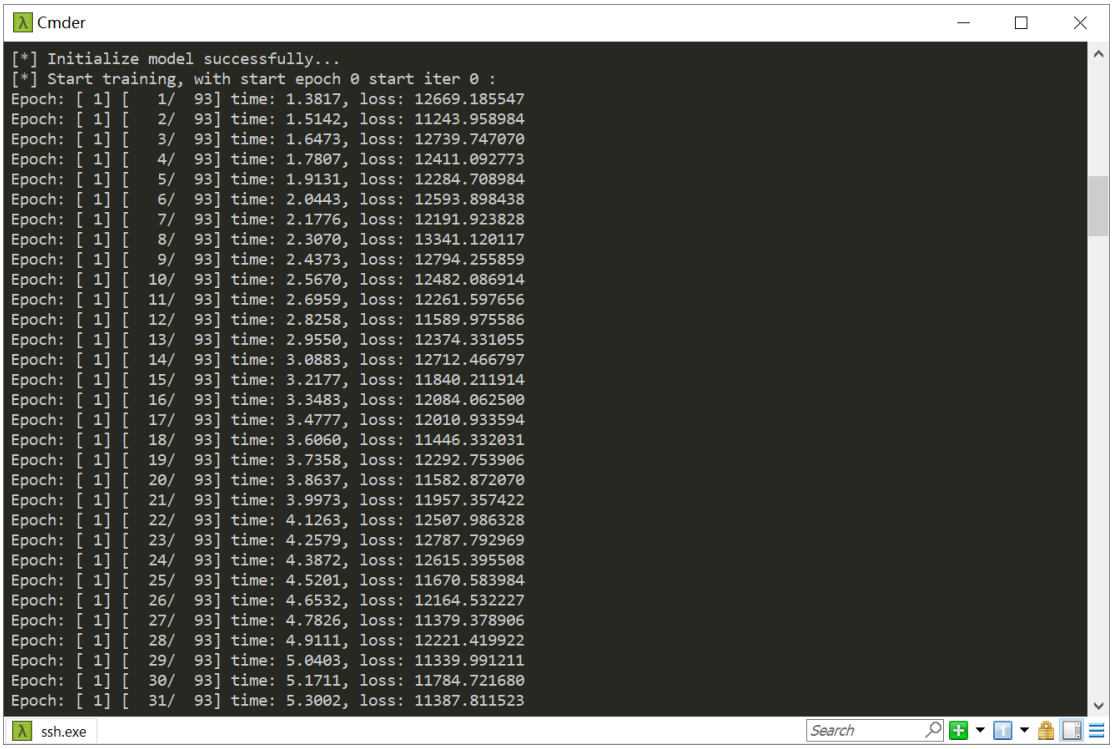
5. 训练技巧

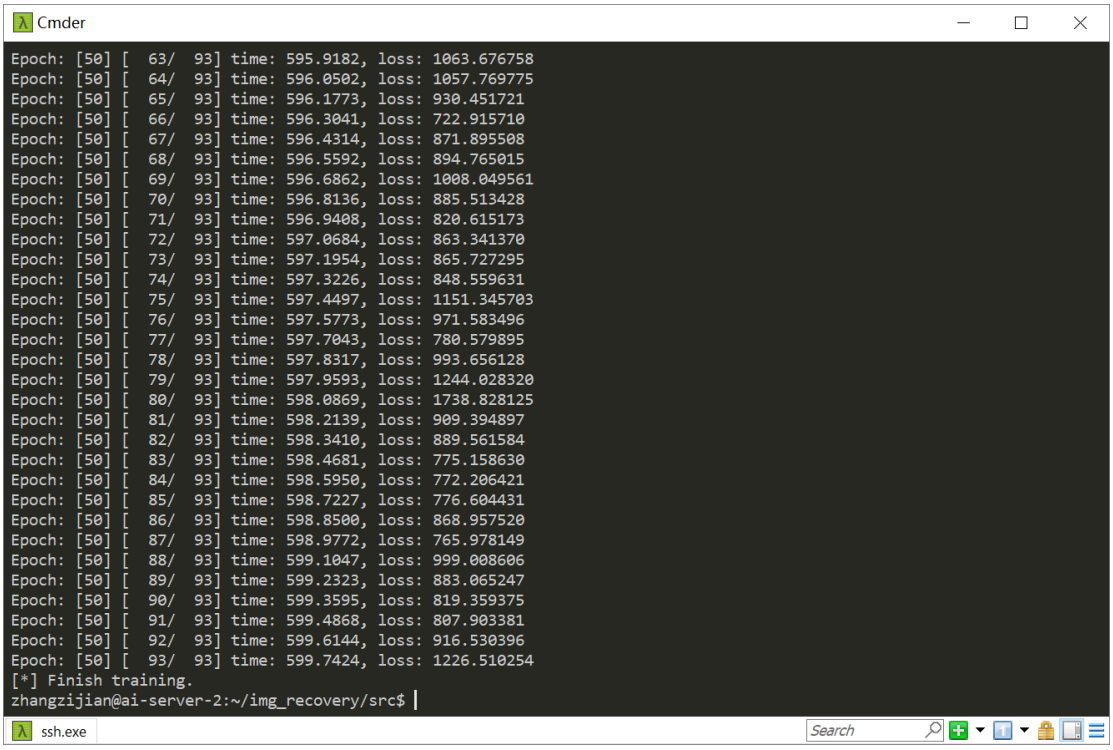
训练中我使用了 batch normalization 的技巧，这是 2015 年 Google 发表的论文《Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift》中提到的算法，它一定程度上能解决梯度弥散、梯度爆炸、过拟合等问题，还能加速神经网络的训练。

在深度网络的训练中，每一层网络的输入都会因为前一层网络参数的变化导致其分布发生改变，这就要求我们必须使用一个很小的学习率和对参数很好的初始化，但是这么做会让训练过程变得慢而且复杂。BN 在每一层的激活函数前对数据进行归一化来解决这个问题。

四、 实验结果

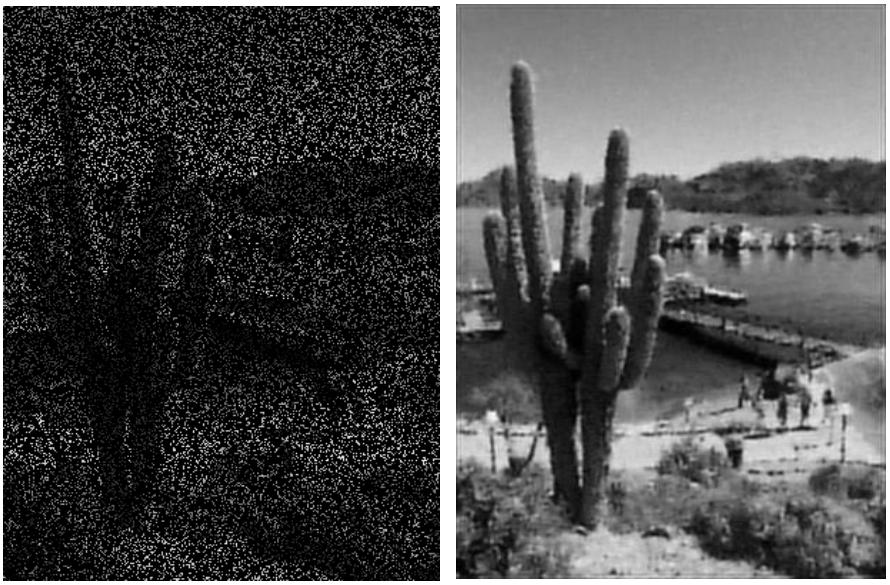
1. 训练过程


Cmder
[*] Initialize model successfully...
[*] Start training, with start epoch 0 start iter 0 :
Epoch: [1] [1/ 93] time: 1.3817, loss: 12669.185547
Epoch: [1] [2/ 93] time: 1.5142, loss: 11243.958984
Epoch: [1] [3/ 93] time: 1.6473, loss: 12739.747070
Epoch: [1] [4/ 93] time: 1.7807, loss: 12411.092773
Epoch: [1] [5/ 93] time: 1.9131, loss: 12284.708984
Epoch: [1] [6/ 93] time: 2.0443, loss: 12593.898438
Epoch: [1] [7/ 93] time: 2.1776, loss: 12191.923828
Epoch: [1] [8/ 93] time: 2.3070, loss: 13341.120117
Epoch: [1] [9/ 93] time: 2.4373, loss: 12794.255859
Epoch: [1] [10/ 93] time: 2.5670, loss: 12482.086914
Epoch: [1] [11/ 93] time: 2.6959, loss: 12261.597656
Epoch: [1] [12/ 93] time: 2.8258, loss: 11589.975586
Epoch: [1] [13/ 93] time: 2.9550, loss: 12374.331055
Epoch: [1] [14/ 93] time: 3.0883, loss: 12712.466797
Epoch: [1] [15/ 93] time: 3.2177, loss: 11840.211914
Epoch: [1] [16/ 93] time: 3.3483, loss: 12084.062500
Epoch: [1] [17/ 93] time: 3.4777, loss: 12010.933594
Epoch: [1] [18/ 93] time: 3.6060, loss: 11446.332031
Epoch: [1] [19/ 93] time: 3.7358, loss: 12292.753906
Epoch: [1] [20/ 93] time: 3.8637, loss: 11582.872070
Epoch: [1] [21/ 93] time: 3.9973, loss: 11957.357422
Epoch: [1] [22/ 93] time: 4.1263, loss: 12507.986328
Epoch: [1] [23/ 93] time: 4.2579, loss: 12787.792969
Epoch: [1] [24/ 93] time: 4.3872, loss: 12615.395508
Epoch: [1] [25/ 93] time: 4.5201, loss: 11670.583984
Epoch: [1] [26/ 93] time: 4.6532, loss: 12164.532227
Epoch: [1] [27/ 93] time: 4.7826, loss: 11379.378906
Epoch: [1] [28/ 93] time: 4.9111, loss: 12221.419922
Epoch: [1] [29/ 93] time: 5.0403, loss: 11339.991211
Epoch: [1] [30/ 93] time: 5.1711, loss: 11784.721680
Epoch: [1] [31/ 93] time: 5.3002, loss: 11387.811523
ssh.exe
Search


Cmder
Epoch: [50] [63/ 93] time: 595.9182, loss: 1063.676758
Epoch: [50] [64/ 93] time: 596.0502, loss: 1057.769775
Epoch: [50] [65/ 93] time: 596.1773, loss: 930.451721
Epoch: [50] [66/ 93] time: 596.3041, loss: 722.915710
Epoch: [50] [67/ 93] time: 596.4314, loss: 871.895508
Epoch: [50] [68/ 93] time: 596.5592, loss: 894.765015
Epoch: [50] [69/ 93] time: 596.6862, loss: 1008.049561
Epoch: [50] [70/ 93] time: 596.8136, loss: 885.513428
Epoch: [50] [71/ 93] time: 596.9408, loss: 820.615173
Epoch: [50] [72/ 93] time: 597.0684, loss: 863.341370
Epoch: [50] [73/ 93] time: 597.1954, loss: 865.727295
Epoch: [50] [74/ 93] time: 597.3226, loss: 848.559631
Epoch: [50] [75/ 93] time: 597.4497, loss: 1151.345703
Epoch: [50] [76/ 93] time: 597.5773, loss: 971.583496
Epoch: [50] [77/ 93] time: 597.7043, loss: 780.579895
Epoch: [50] [78/ 93] time: 597.8317, loss: 993.656128
Epoch: [50] [79/ 93] time: 597.9593, loss: 1244.028320
Epoch: [50] [80/ 93] time: 598.0869, loss: 1738.828125
Epoch: [50] [81/ 93] time: 598.2139, loss: 909.394897
Epoch: [50] [82/ 93] time: 598.3410, loss: 889.561584
Epoch: [50] [83/ 93] time: 598.4681, loss: 775.158630
Epoch: [50] [84/ 93] time: 598.5950, loss: 772.206421
Epoch: [50] [85/ 93] time: 598.7227, loss: 776.604431
Epoch: [50] [86/ 93] time: 598.8500, loss: 868.957520
Epoch: [50] [87/ 93] time: 598.9772, loss: 765.978149
Epoch: [50] [88/ 93] time: 599.1047, loss: 999.008606
Epoch: [50] [89/ 93] time: 599.2323, loss: 883.065247
Epoch: [50] [90/ 93] time: 599.3595, loss: 819.359375
Epoch: [50] [91/ 93] time: 599.4868, loss: 807.903381
Epoch: [50] [92/ 93] time: 599.6144, loss: 916.530396
Epoch: [50] [93/ 93] time: 599.7424, loss: 1226.510254
[*] Finish training.
zhangzijian@ai-server-2:~/img_recovery/src$
ssh.exe
Search

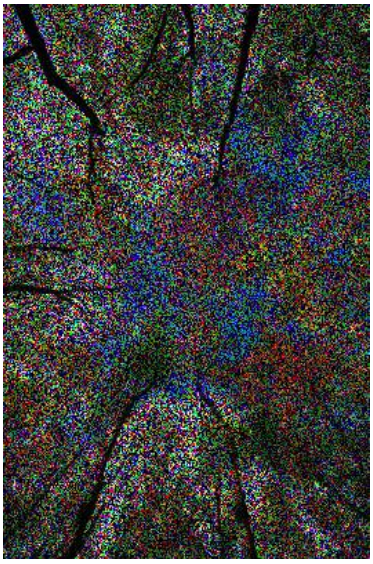
2. 训练结果

对于 A.png，使用 32 的 batch_size，40 的 epoch，8 层隐藏层 CNN，80%的加噪率，结果如下：



对于 B.png，使用 32 的 batch_size，50 的 epoch，10 层隐藏层 CNN，40%的加噪率，结果如下：

对于 C.png，使用 32 的 batch_size，50 的 epoch，10 层隐藏层 CNN，60%的加噪率，结果如下：




五、 实验不足与改进

代码中未实现 Saver 的 checkpoint，导致模型无法保存，每次需要重新训练。

由于时间有限，训练集选的不是很好，如果能选点风景图作为训练集，效果可能会更好。

没有实现验证集和测试集，因为图片大小不一，所以测试 loss 也没什么意义。

未对超参数做一些其它的尝试，对比不同超参数的效果。

网络结构过于简单，只有简单的几层 CNN，未来可以考虑将 mask 作为一个输入进行训练。

六、 参考文献

算法参考:
Alex et al.《ImageNet Classification with Deep Convolutional Neural Networks》
Google《Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift》
编码参考:
Tensorflow 官方文档: https://www.tensorflow.org

七、 声明

本人张子健（3150104669）声明：本次实验从构思、设计到编码实现均由我本人独立自主完成，未参考其它任何去噪算法，未与其他任何人交流。所使用的 IDE 及编码环境均为免费开源的。