

浙江大学

本科实验报告

课程名称: 人工智能

姓 名: 王若鹏

学 院: 信息与电子工程学院

专 业: 电子科学与技术

学 号: 3170105582

指导教师: 王东辉

2020 年 5 月 28 日

图像恢复

1. 背景

图像是一种非常常见的信息载体，但是在图像的获取、传输、存储的过程中可能由于各种原因使得图像受到噪声的影响。如何去除噪声的影响，恢复图像原本的信息是计算机视觉中的重要研究问题。

常见的图像恢复算法有基于空间域的中值滤波、基于小波域的小波去噪、基于偏微分方程的非线性扩散滤波等，在本次实验中，我们要对图像添加噪声，并对添加噪声的图像进行基于模型的去噪。

2. 实验要求

- (1). 生成受损图像。
 - 受损图像 (X) 是由原始图像 ($I \in R^{H \times W \times C}$) 添加了不同噪声遮罩 (noise masks) ($M \in R^{H \times W \times C}$) 得到的 ($X = I \odot M$)，其中 \odot 是逐元素相乘。
 - 噪声遮罩仅包含 {0,1} 值。对原图的噪声遮罩的可以每行分别用 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。
- (2). 使用你最擅长的算法模型，进行图像恢复。
- (3). 评估误差为所有恢复图像 (R) 与原始图像 (I) 的 2-范数之和，此误差越小越好。 $error = \sum \text{norm}(R_i(\cdot) - I_i(\cdot), 2)$ ，其中 (\cdot) 是向量化操作，其他评估方式包括 Cosine 相似度以及 SSIM 相似度。

3. 开发环境

- 编程语言：Python 3.5
- 相关库：Numpy, OpenCV, sklearn
- 开发工具：Anaconda、Python shell
- 开发平台：Mo 人工智能训练平台
- 操作系统：MacOS Mojave 10.14.6

4. 实验内容与算法设计

4.1 导入相关包

将图像恢复过程中需要的包导入：

```

from matplotlib import pyplot as plt # 展示图片
import numpy as np # 数值处理
import cv2 # opencv库
from sklearn.linear_model import LinearRegression, Ridge, Lasso # 回归分析
from skimage.measure import compare_ssim as ssim
from scipy import spatial
import random

```

4.2 读取图片

读取图片，图片是以 np.array 类型存储，img_path: 图片的路径以及名称，img 为返回值，以 np.array 类型存储：

```

def read_image(img_path):
    """
    读取图片，图片是以 np.array 类型存储
    :param img_path: 图片的路径以及名称
    :return: img np.array 类型存储
    """

    # 读取图片
    img = cv2.imread(img_path)
    # 如果图片是三通道，采用 matplotlib 展示图像时需要先转换通道
    if len(img.shape) == 3:
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    return img

```

4.3 保存图片

OpenCV 保存一个图片使用函数 cv2.imwrite(filename, img[, params]): 其中 filename 为保存文件路径及文件名，文件名要加格式；img 是需要保存的图片。用 cv2.imwrite() 来封装一个保存图片的函数如下：

```

def save_image(filename, image):
    """
    将np.ndarray 图像矩阵保存为一张 png 或 jpg 等格式的图片
    :param filename: 图片保存路径及图片名称和格式
    :param image: 图像矩阵，一般为np.array
    :return:
    """

    # np.copy() 函数创建一个副本。对副本数据进行修改，不会影响到原始数据，它们物理内存不在同一位置
    img = np.copy(image)
    # 从给定数组的形状中删除一维的条目
    img = img.squeeze()
    # 将图片数据存储类型改为 np.uint8
    if img.dtype == np.double:
        # 若img数据存储类型是 np.double，则转化为 np.uint8 形式
        img = img * np.iinfo(np.uint8).max
        # 转换图片数组数据类型
        img = img.astype(np.uint8)
    # 将 RGB 方式转换为 BGR 方式
    img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
    # 生成图片
    cv2.imwrite(filename, img)

```

4.4 归一化

机器学习过程中，数据归一化非常重要，归一化的目标主要有：①把数变为(0,1)或者(-1,1)之间的小数；②把有量纲表达式变为无量纲表达式。

常见的归一化方法有：

- 线性比例变换法： $x_i = x_i / \max(x)$
- min-max 标准化： $x_i = [x_i - \min(x)] / [\max(x) - \min(x)]$
- z-score 标准化： $x_i = [x_i - \text{mean}(x)] / \sigma$

```
def normalization(image):
    """
    将数据线性归一化
    :param image: 图片矩阵, 一般是np.array 类型
    :return: 将归一化后的数据, 在 (0,1) 之间
    """
    # 获取图片数据类型对象的最大值和最小值
    info = np.iinfo(image.dtype)
    # 图像数组数据放缩在 0-1 之间
    return image.astype(np.double) / info.max
```

4.5 生成受损图像

受损图像(X)是由原始图像($I \in R^{H \times W \times C}$)添加了不同噪声遮罩($M \in R^{H \times W \times C}$)得到的($X=I \odot M$)，其中 \odot 是逐元素相乘。

噪声遮罩仅包含{0,1}值。对原图的噪声遮罩的可以每行分别用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。

我采用随机生成噪声的方法，把噪声比率noise_ratio定为出现噪声的概率。对整个图片进行遍历，对每个像素点的每个通道，有noise_ratio成为噪点。实现方式为生成一个0-1之间的随机数，若该随机数小于noise_ratio，则生成噪声，否则无噪声。

```
def noise_mask_image(img, noise_ratio):
    """
    根据题目要求生成受损图片
    :param img: 图像矩阵, 一般为 np.ndarray
    :param noise_ratio: 噪声比率, 可能值是0.4/0.6/0.8
    :return: noise_img 受损图片, 图像矩阵值 0-1 之间, 数据类型为 np.array,
            数据类型对象 (dtype): np.double, 图像形状:(height,width,channel), 通道(channel) 顺序为RGB
    """
    h,w,c = img.shape
    noise_img = np.zeros((h,w,c))
    for i in range(h):
        for j in range(w):
            for k in range(c):
                if random.random() < noise_ratio:
                    noise = 0
                else:
                    noise = 1
                noise_img[i][j][k] = img[i][j][k] * noise
    return noise_img
```

4.6 获得噪声图像

对被噪声污染的图片进行处理，被污染点记为 0，无污染点记为 1，以矩阵形式存储。

```
def get_noise_mask(noise_img):
    """
    获得噪声图像，一般为 np.array
    :param noise_img: 带有噪声的图片
    :return: 噪声图像矩阵
    """
    # 将图片数据矩阵只包含 0 和 1，如果不能等于 0 则就是 1。
    return np.array(noise_img != 0, dtype='double')
```

4.7 评估误差

4.7.1 2-范数

矩阵 A 的 2 范数，就是 A 的转置共轭矩阵与矩阵 A 的积的最大特征根的平方根值，是指空间上两个向量矩阵的直线距离。

```
def compute_error(res_img, img):
    """
    计算恢复图像 res_img 与原始图像 img 的 2-范数
    :param res_img: 恢复图像
    :param img: 原始图像
    :return: 恢复图像 res_img 与原始图像 img 的2-范数
    """
    # 初始化
    error = 0.0
    # 将图像矩阵转换成为np.ndarray
    res_img = np.array(res_img)
    img = np.array(img)
    # 如果2个图像的形状不一致，则打印出错误结果，返回值为 None
    if res_img.shape != img.shape:
        print("shape error res_img.shape and img.shape %s != %s" % (res_img.shape, img.shape))
        return None
    # 计算图像矩阵之间的评估误差
    error = np.sqrt(np.sum(np.power(res_img - img, 2)))
    return round(error, 3)
```

4.7.2 SSIM 相似度

SSIM(structural similarity index)，结构相似性，是一种衡量两幅图像相似度的指标。SSIM 的范围为 0 到 1。当两张图像一模一样时，SSIM 的值等于 1。

```
def calc_ssime(img, img_noise):
    """
    计算图片的结构相似度
    :param img: 原始图片， 数据类型为 ndarray， shape 为 [长, 宽, 3]
    :param img_noise: 噪声图片或恢复后的图片,
                      数据类型为 ndarray, shape 为 [长, 宽, 3]
    :return:
    """
    return ssim(img, img_noise,
               multichannel=True,
               data_range=img_noise.max() - img_noise.min())
```

4.7.3 Cosine 相似度

余弦相似度，是通过计算两个向量的夹角余弦值来评估他们的相似度。余弦相似度的取值范围为-1 到 1。当两张图像一模一样时，余弦相似度的值等于 1。

```

def calc_csim(img, img_noise):
    """
    计算图片的 cos 相似度
    :param img: 原始图片, 数据类型为 ndarray, shape 为 [长, 宽, 3]
    :param img_noise: 噪声图片或恢复后的图片,
                      数据类型为 ndarray, shape 为 [长, 宽, 3]
    :return:
    """
    img = img.reshape(-1)
    img_noise = img_noise.reshape(-1)
    return 1 - spatial.distance.cosine(img, img_noise)

```

4.8 图像恢复模块

图像恢复模块是本实验的核心模块，我的思路主要有四步：计算噪声率→二元线性回归→中值滤波→预测结果修正。

```

def restore_image(noise_img, size=4):
    """
    使用 你最擅长的算法模型 进行图像恢复。
    :param noise_img: 一个受损的图像
    :param size: 输入区域半径, 长宽是以 size*size 方形区域获取区域, 默认是 4
    :return: res_img 恢复后的图片, 图像矩阵值 0-1 之间, 数据类型为 np.array,
            数据类型对象 (dtype): np.double, 图像形状: (height,width,channel), 通道(channel) 顺序为RGB
    """
    # 恢复图片初始化, 首先 copy 受损图片, 然后预测噪声点的坐标后作为返回值。
    res_img = np.copy(noise_img)
    # 获取噪声图像
    noise_mask = get_noise_mask(noise_img)

```

4.8.1 计算噪声率

由于我的算法针对高污染和低污染受损图像有着不同的修复算法，但是噪声率未知，因此需要先计算出噪声率。采用遍历每个像素的方式计算，最后将噪声率大于 70% 的标记为高污染图像。

```

# 第一步: 计算噪声率
noise = 0
for row in range(rows):
    for col in range(cols):
        for chan in range(channel):
            if noise_mask[row,col,chan] == 0.: #未被污染
                noise += 1
nr = 1.0*noise/rows/cols/channel
print("计算得噪声率为: "+str(format(100.0*nr, '.2f'))+"%")
high = 0 #高污染标记
if nr > 0.7:
    high = 1

```

4.8.2 二元线性回归

遍历所有像素及通道，针对每个受损点，采样其周围 4x4 的无损点（若为高污染则采样周围 5x5），建立二元线性回归模型，对每个受损点进行回归预测。线性回归模型的建立调用 LinearRegression(). 若在遍历时遇到无损点，不进行任何操作，直接进入下一轮循环。这也是整个模块最耗时的一部分。

```

# 第二步：二元线性回归
count = 0
for row in range(rows):
    for col in range(cols):
        # 确定处理区域的位置
        row_up, row_down, col_left, col_right = sample_mask(row, rows, col, cols, 2)
        for chan in range(channel):
            if noise_mask[row,col,chan] != 0.: #未被污染
                continue
            # 针对噪声点，采样回归样本
            x_sample=[]
            y_sample=[]
            for i in range(row_up, row_down+high):
                for j in range(col_left, col_right+high):
                    if noise_mask[i,j,chan] == 0. or (i==row and j==col):
                        continue
                    x_sample.append([i,j])
                    y_sample.append([noise_img[i,j,chan]])
            if len(x_sample)==0:
                continue
            # 二元线性回归
            model = LinearRegression()
            model.fit(x_sample, y_sample)
            res_img[row,col,chan] = model.predict([[row,col]])
            count += 1
            if count % 50000 == 0:
                print("正在二元线性回归修复: "+str(format(100.0*count/rows/cols,'.2f'))+"%")

```

4.8.3 中值滤波

针对高污染图片，我还对其进行一轮中值滤波，目的是消除线性回归过程中没有成功预测带来的影响，即周围没有无损点、无法建立回归模型的情况。噪声率越大，中值滤波的效果就越明显。因此我设定噪声率在 0.5-0.7 之间的采用 3x3 滤波模板，0.7 以上的采用 4x4 滤波模板。

```

# 第三步：中值滤波（仅针对高污染图片）
res_img2 = np.copy(res_img)
if nr > 0.5:
    radius = 1
    if nr > 0.7:
        radius = 2
count = 0
for row in range(rows):
    for col in range(cols):
        # 确定处理区域的位置
        row_up, row_down, col_left, col_right = sample_mask(row, rows, col, cols, radius)
        # 处理各个通道
        for chan in range(channel):
            if noise_mask[row,col,chan]!=0.: #未被污染
                continue
            # 采样未被完全修复的像素
            mask=[]
            for i in range(row_up, row_down+high):
                for j in range(col_left, col_right+high):
                    mask.append(res_img[i,j,chan])
            # 中值滤波，消除胡椒噪声
            res_img2[row,col,chan] = np.median(mask)
            count += 1
            if count % 100000 == 0:
                print("正在中值滤波修复: "+str(format(100.0*count/rows/cols,'.2f'))+"%")

```

4.8.4 预测结果修正

在最后返回修复图像之前，需要对预测结果进行修正。原因是我在测试的过程中发现，有时会出现预测值在 0-1 之外的情况，不符合归一化图像的定义。修正过程是：对大于 1 的点设为 1.0，对小于 0 的点设为 0.0，这样就能返回正常的图像了。

5. 实现结果

5.1 原图



5.2 受损图片

使用 `noise_mask_image()` 函数对原图进行处理，生成受损图片。下图从左到右依次为 40%、60%、80% 的噪声污染图片：



5.3 恢复图片

使用 `restore_image()` 函数分别对受损图片进行恢复。下图从上至下、从左到右依次为 40%、60%、80% 的噪声修复图片：



5.4 测试数据

记原始图片为 A，受损图片为 A^* ，恢复图片为 A' ，记录以上噪声率为 40%、60%、80% 的图片恢复数据如下：

噪声率	40%	60%	80%
Notebook 运行时间/s	392.968	640.923	846.904
2-范数 (A 与 A*)	757.341	928.238	1071.842
2-范数 (A 与 A')	27.751	34.655	53.855
SSIM 相似度 (A 与 A*)	0.04882	0.02956	0.01547
SSIM 相似度 (A 与 A')	0.93394	0.89339	0.80965
Cosine 相似度 (A 与 A*)	0.77495	0.63240	0.44711
Cosine 相似度 (A 与 A')	0.99973	0.99958	0.99899

在 Mo 平台上进行系统测试，测试点均正常通过：

测试点	状态	时长	结果
测试噪声图像的生成	✓	3s	生成的噪声图片的噪声比例无误
测试噪声图片的恢复	✓	15s	<p>恢复成功，在 150×150 的测试图片，得到的误差为 15.58, SSIM 相似度为 0.901, Cosine 相似度为 0.994</p> 

6. 评价

- 优点：模型简单，能在较短时间内完成对受损图像的恢复。效果优良，根据评估误差的计算，能够对受损图像进行 95% 的恢复。对于低污染图像，肉眼很难分清修复图与原图的差别。
- 缺点：高污染图像在区域边缘的恢复效果差强人意，容易出现模糊。依赖于噪声遮罩 noise_mask。若只给出受损图像，则去噪的效果不一定好。

7. 心得体会

图像修复是人工智能在图像处理领域的一个重要的应用。回归分析是机器学习中一个基本的监督学习策略，包括线性回归、逻辑斯蒂回归等等。于是我采用了多元线性回归的算法来进行图像修复。尽管不是所有算法中效果最好的，但运行时间非常短。由于我本学期也

上了信电学院的数字图像处理，因此对各种图像处理理论和算法都有了初步的了解。经过了不断的测试，我针对高污染图像的修复添加了中值滤波的过程，虽然一定程度上模糊了图像，但也滤掉了修复过程中产生的脉冲噪点，能降低高污染图像的修复误差。

在代码调试过程中，我遇到了一些错误。比如在刚开始，有时候会计算出来修复图片与原图的 2-范数是一个十亿量级的数字，这让我大吃一惊。经过一番思索，发现是个别像素点的预测值在 0-1 之外，违背了图像的归一化原则。我在算法最后添加了一个预测修正的模块，就解决了这个问题。

人工智能的一大目的在于对信息进行处理和分析，而图像是非常重要的信息载体。图像修复也在图像信息处理的过程中扮演了不可或缺的角色。希望自己以后有机会能继续从事图像处理中 AI 算法的研究。

8. 参考资料

- OpenCV: <https://opencv-python-tutorials.readthedocs.io/>
- Numpy: <https://www.numpy.org/>
- Scikit-Learn: <https://scikit-learn.org/stable/>