图片恢复算法

编写于2019年4月12日

曲岳 *QuYue*

目录

- 1 问题介绍
- 2 算法介绍
  - 2.1 SVD分解排序
    - 2.1.1 原理介绍
    - 2.1.2 结果展示与评价
  - 2.2 SVD分解贪心法
    - 2.2.1 原理介绍
    - 2.2.2 结果展示与评价
  - 2.3 直接贪心法
    - 2.3.1 原理介绍
    - 2.3.2 结果展示与评价
  - 2.4 其它
  - 3 结果展示
    - 3.1 book
    - 3.2 win10
    - 3.3 Baidu
    - 3.3 sketch简笔画
    - 3.4 QR_code二维码
    - 3.5 未来的扩展
- 4 代码与使用

1 问题介绍

如果我们将图片进行**随机的打乱**（当然是只对行或者对列进行打乱），这样图片对于人来说已经是不可以识别出来的了，如图1所示。那么我们是否可以对其进行恢复呢？而这就是本文将准备挑战的问题。

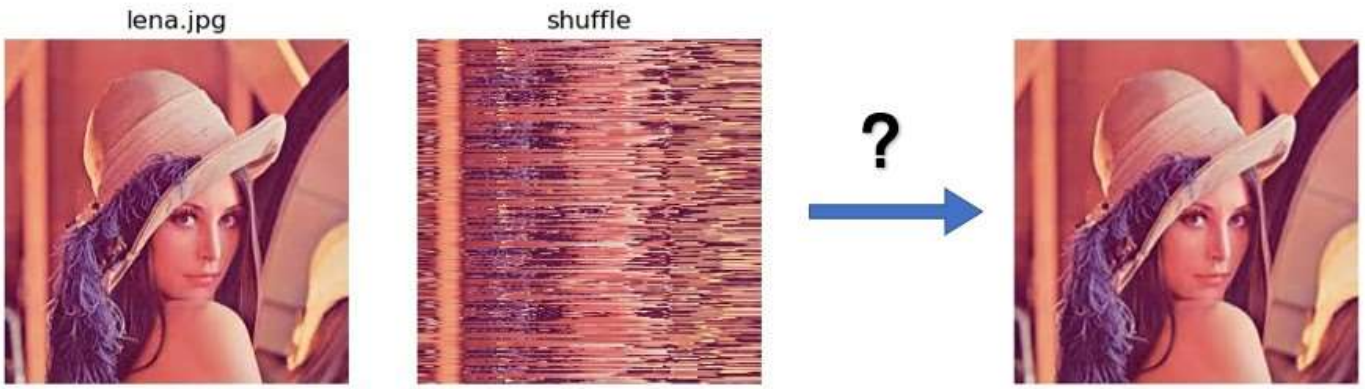

图1. 图片恢复挑战

2 算法介绍

这里使用了**3种算法**来分别对图片进行恢复，为了方便我们这里先只考虑对**灰度图**的行(row)进行打乱这个问题（因为只有一个通道，而RGB有三个通道），即图2的任务。

图2. 灰度图恢复

在这里的图的题目中有一个值，这个是我用来衡量图片的流畅度的值$score$，其是每一行元素值的差分的绝对值，如果一个$m*n$的图片M,那么其流畅度$score(M)$如公式1所示：

$$score(M) = \sum_{i=1}^{m-1}\sum_{j=1}^{n}|(M_{i,j} - M_{i+1,j})|$$

其中$M_{i,*}$是图片M的第一行向量。这个值可以一定程度上衡量图片行与行之间的流畅度，而从图2可以发现打乱后$score$的值会变得很大。

2.1 SVD分解排序

2.1.1 原理介绍

奇异值分解（SVD）是一个比较好的矩阵分解方法，被广泛的用于降维等机器学习任务中。其基本公式可见公式2：

$$M = U\Sigma V^* \tag{2}$$

其中

- $M \in \mathbb{R}^{m\times n}$
- $U \in \mathbb{R}^{m\times m}$，$U$是一个酉矩阵，其每一列组成一套对$M$的正交"输入"或"分析"的基向量。这些向量是$MM^*$的特征向量。
- $V \in \mathbb{R}^{n\times n}$，$V$也是一个酉矩阵，其每一列代表了对$M$的正交"输入"或"分析"的基向量。这些向量是$M^*M$特征向量，$V^*$是$V$的共轭转置。
- $\Sigma \in \mathbb{R}^{m\times n}$，$\Sigma$则是一个半正定的对角阵。其对角线上的元素是奇异值，可视为是在输入与输出间进行的标量的"膨胀控制"，这些是M^*M及MM^*的奇异值，并与U和V的列向量相对应。

U的每一列会对应不同大小的奇异值，而Σ的奇异值通常都是从大到小进行排列的，奇异值越大，对应的信息也越重要，相应的奇异值越小，对应的信息也越不重要，甚至有可能是噪声（这也是去噪以及降维的原理）。

而刘传仁老师提出了以下的三个假设：

1. 因为在计算的时候U第一列的每个元素会与Σ的第一个对角元素（即第一个奇异值）相乘，而该元素恰好又是最重要的元素（是最大的奇异值），所以相应的我们可以认为U的第一列也是最重要的。
2. 对于一张正常的图片来说，通常每一行之间相差的都不大。
3. 基于以上两个假设，我们可以对一个被打乱的图片计算U值，然后使用其第一列的数，进行从大到小排序，这样图片的相邻行的内容相差会很小，图片会在一定程度上进行恢复。

其**过程如算法1**所示：

算法 1 SVD 分解排序算法

输入： 打乱的图片 M，其中 $M \in R^{m \times n}$

输出： 恢复的图片 M_{new}

```
function SVD IMSORT(M)
    对打乱的图片进行 SVD 分解，M = UΣV*
    提取 U 的第一列的元素，获得将其从小到大进行排序的顺序 index
    对打乱的图片的列使用顺序 index 进行重排列，得到恢复的图片 M_new，M_new = M[index, :]
    return M_new
end function
```

2.1.2 结果展示与评价

其**结果如图3**所示：

图3. lena SVD分解排序

我们可以看出，图片相关的部分在一定程度上都成片的聚集在一起，其$score$确实下降了，但总体效果并不是很理想，与原图还有一定的差距。所以可以看出**SVD分解排序算法的缺点是只考虑了最大的奇异值对应的信息**，这样是不够的。

2.2 SVD分解贪心法

2.2.1 原理介绍

即然一个奇异值提供的恢复信息不够，那么为何不多取几个奇异值呢？所以算法SVD分解贪心法就基于这个思想。

首先让我们看一下图片打乱前后的U的前几列的值的分布，如图4所示：

原始图片的U的前4列

	0	1	2	3
0	-0.07132153604231861	0.042678031796367796	-0.02100270507441082	0.048449840166865515
1	-0.07129343230872975	0.04168962203378013	-0.02203298576204976	0.04927360721902868
2	-0.07094080272679466	0.043988842821858776	-0.01825681380486771	0.05402570332398873
3	-0.07038810008915934	0.04784377353208202	-0.012495948651503497	0.06092258150805335
4	-0.07002388948738146	0.05039401668819185	-0.007400339903704367	0.06708345024831372
5	-0.06979211081494541	0.052192161958920784	-0.0028060958388880023	0.07205812643273996
6	-0.06956203111185791	0.054780863895712316	0.002736666866558435	0.0762164910200882
7	-0.0693360238351371	0.0574881242470487	0.007031812097349405	0.07940939944542302
8	-0.06914572965402396	0.058619971284376717	0.012021400364839922	0.08429706736963952
9	-0.06906506441073047	0.059014590581517604	0.01431482447556278	0.08649355372771207
10	-0.06886762477687894	0.060162543658166465	0.018843658584453204	0.08967963905420973

打乱后图片的U的前4列

	0	1	2	3
0	-0.06079752084944323	-0.05388481258707771	-0.11843515938118056	0.005687636717392031
1	-0.06581060179848297	-0.10209533595018194	0.03172398932107609	-0.09891888345639366
2	-0.06648590172794257	-0.0931549436966359	0.05998272046220354	-0.07820929900897404
3	-0.07380707645849852	-0.027526795148505256	0.005889926102946252	0.141711144731141722
4	-0.06979610709633005	-0.04677651807397312	0.12375791464276875	0.04606426672308356
5	-0.06504605841475937	0.06597501623401625	-0.04528129109896578	0.08071605158705486
6	-0.07733830432611173	-0.054153929011856956	0.11255534114861336	0.06099516690895934
7	-0.07722462060058449	-0.049832362475066685	0.07128649246937813	0.1034385769432736
8	-0.07546821198434546	-0.03674661528564278	0.031174008020340694	0.1263941750150011
9	-0.06432801927839923	-0.0415850693631639	-0.10379120134508882	0.0075654673407474365
10	-0.0679600635606105	0.059315315697429136	0.02909201736790847	-0.020144144953608556

图4. 打乱前后的U的前4列

可见打乱前U的前几列每一行过渡的都很均匀，并没有忽然的突变，而打乱后U的前几列变化的很剧烈，不均匀（通过颜色的变化就可以看出）。所以我们只要想办法将突变比较剧烈的U的前几列经过重排列后，使其变化平缓一点就好了。注意：也不能取太多的列，经过我观察，原始图片的U矩阵的列越到后面排列的越不平缓，如果使用的这些列的话会对图片的恢复有着明显的影响。

所以该算法通过选取多个奇异值对应的信息，即U的前几个列的值，然后通过贪心法将U不同的行通过计算前几个奇异值最接近的方式拼接到一起，得到一个新的图片，这样就可以进行图片恢复了。

其具体**过程如算法2**所示：

算法 2 SVD 分解贪心算法

输入: 打乱的图片 M，选取的列数 u，其中 $M \in R^{m \times n}$

输出: 恢复的图片 M_{new}

1: **function** SVD GREED(M, u)
2: 对打乱的图片进行 SVD 分解，$M = U\Sigma V^*$
3: 提取 U 的前 u 列元素，得到新的矩阵 F，$F := U[:, : u]$.
4: 使用贪心法 GREED(F), 计算出图片恢复的顺序 $index$
5: 对打乱的图片 M 的列使用顺序 $index$ 进行重排列，得到恢复的图片 M_{new}，$M_{new} = M[index, :]$
6: **return** M_{new}
7: **end function**

其中贪心算法$GREED$如**如算法3**所示：

算法 3 贪心算法

输入: 特征矩阵 F，其中 $F \in R^{m \times k}$

输出: 恢复的顺序 $index$

```
function GREED(F)
    设定使用列表 used 为空表, 设定未使用列表 unused 为从 1 ~ m 的表。used := [],unused := [1,2...,m]
    随机选取 F 的第 i 行做为起始行，更新 used 和 unused，UPDATELIST(used, unused, i, 0)
    设定 first 是 used 列表的第一个元素，last 是 used 列表的最后一个序号
    while unused 是空列表 do
        分别找到 F 未使用的行与其第 first 行距离最近的行的序号 t1 和距离 d1, 以及与其第 last 行距离最近的行的序号 t2 和距离 d2
        if d1 < d2 then
            将 t1 插入到列表 used 开头，更新 UPDATELIST(used, unused, t1, 0), 同时 first := t1
        else
            将 t2 插入到列表 used 最后，更新 UPDATELIST(used, unused, t2, 1), 同时 last := t2
        end if
    end while
    恢复顺序就是列表 used, index := used
    return index
end function
function UPDATELIST(used, unused, i, where)
    if where = 0 then
        在列表 used 的开头插入元素 i, 即 used := used.insert(i, 0).
    else
        在列表 used 的最后插入元素 i, 即 used := used.insert(i, −1)
    end if
    将列表 unused 中删除元素 i, 即 unused := unused.remove(i).
    return used, unused
end function
```

2.2.2 结果展示与评价

其**结果如图5**所示:

图5. lena SVD分解贪心法

其中u值是选取的矩阵U的前u列，可见SVD分解贪心法的效果还是很好的，**当u值增大，恢复的效果随之提升，但是也不能让u值增的过大**，因为如我上面所说，后面的值分布并不平缓，对于这些权重较小的列我们就不要考虑了（如果非要考虑的话也许在计算距离的时候需要对不同列使用不同的权重），当$u = 12$时可以让$score$(**325280**)十分接近原图（**325187**）。

2.3 直接贪心法

2.3.1 原理介绍

其实根据SVD分解贪心法，我们也可以思考，也许我们可以不使用SVD的U矩阵来计算距离使用贪心法，图片本身的像素也是一个很好的特征，通常图片相邻行的像素之间的过渡也较为平稳。

所以直接将图片的每一行作为特征，直接输入到**算法3**中，这就是直接贪心法。

其具体**过程如算法4**所示：

算法 4 直接贪心算法

输入: 打乱的图片 M, 其中 $M \in R^{m \times n}$

输出: 恢复的图片 M_{new}

1: **function** DIRECT GREED(M)
2: 将矩阵 M 直接作为特征矩阵 F，$F := M$.
3: 使用贪心法 GREED(F), 计算出图片恢复的顺序 $index$
4: 对打乱的图片 M 的列使用顺序 $index$ 进行重排列，得到恢复的图片 M_{new}，$M_{new} = M[index, :]$
5: **return** M_{new}
6: **end function**

2.3.2 结果展示与评价

其**结果如图6**所示：

图6. lena 直接贪心法

可以看出直接贪心法的效果也不错，虽然没有参数可调，但是性能稳定，能较快的找到一个较好的解，恢复后的图片$score$(**325280**)十分接近原图（**325187**），也和SVD分解贪心法的$u = 12$的$score$一模一样。

2.4 其它

那么接下来我们就要重新的看一下最开始的任务，就是分类彩图（对于列打乱就不细说了，先把图片转置一下就行了），我们可以将图片的RGB三个通道横着铺在一起，变成一张单通道就可以了。我们可以看一下效果，**如图7, 8, 9，10**:

图7. 图片恢复

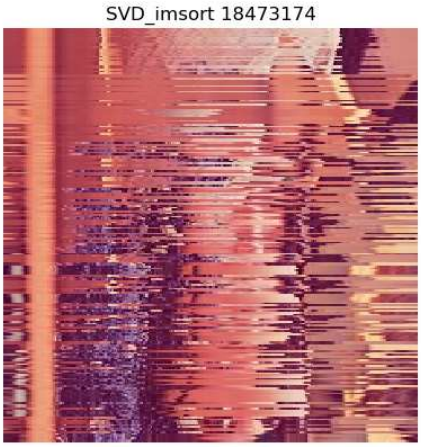

图8. lena SVD分解排序

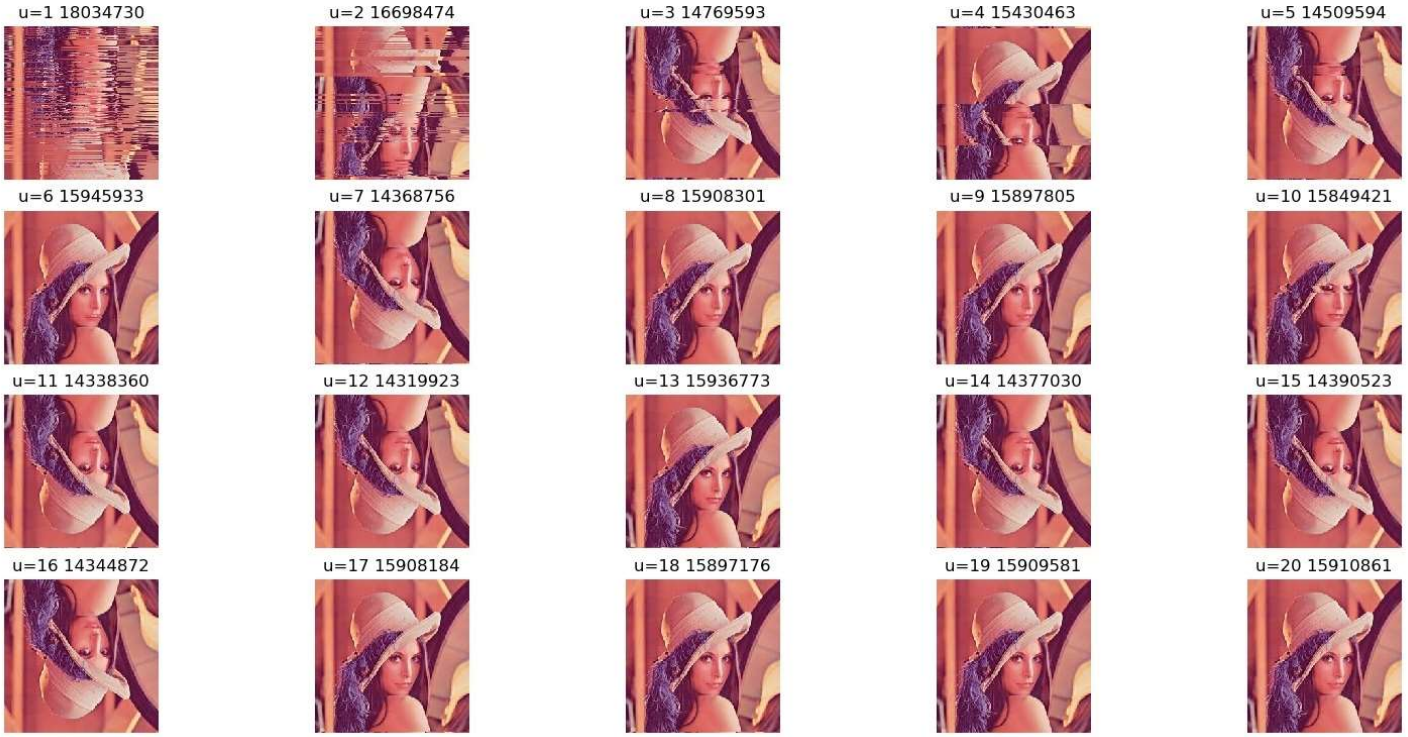

图9. lena SVD分解贪心法

图10. lena 直接贪心法

发现了一个有意思的问题，就是对于SVD分解贪心法，$score$（**14319923**）甚至可以比原图还低（**15906216**），也许是我的评价指标并不好.....

3 结果展示

下面就是一些结果的展示，选取了许多不同的图片，有些比较有趣的结果。

3.1 book

图11. book图片恢复

图12. book SVD分解排序

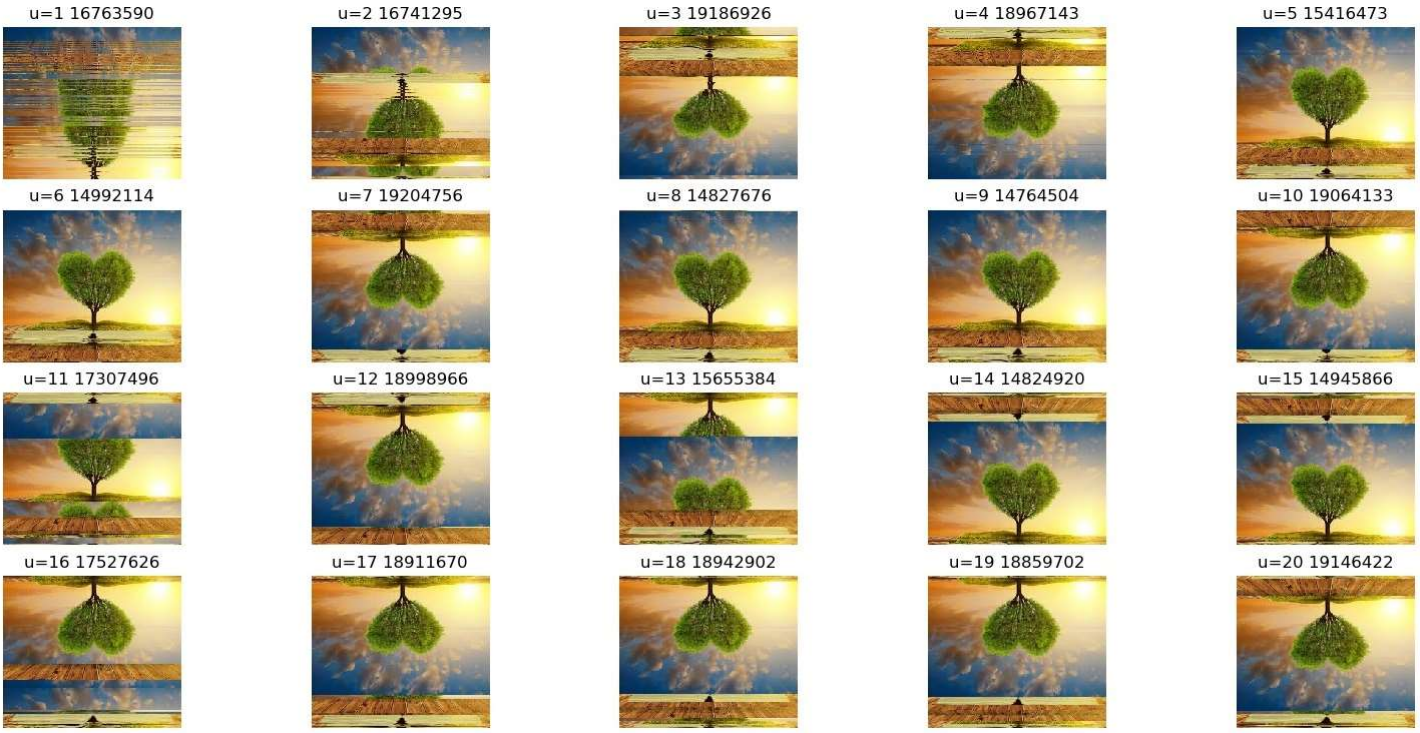

图13. book SVD分解贪心法

图14. book 直接贪心法

我发现对于内容色彩较丰满的图片，基本上都能恢复的很好。那么下面挑战一些难的吧。

3.2 win10

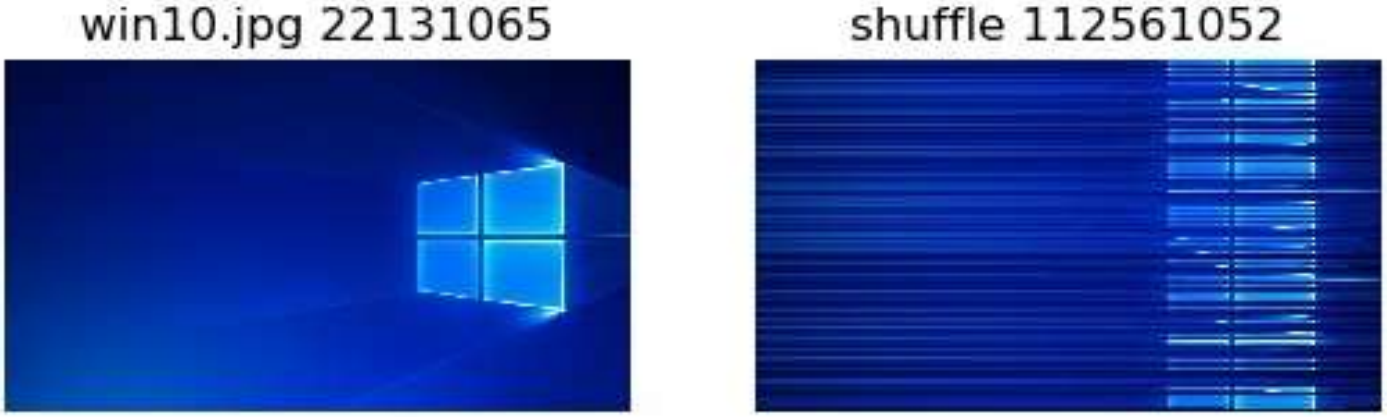

图15. win10图片恢复

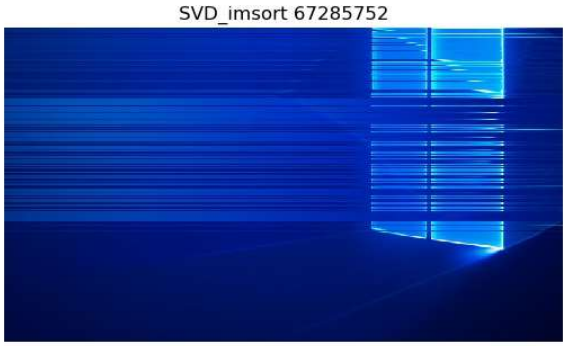

图16. win10 SVD分解排序

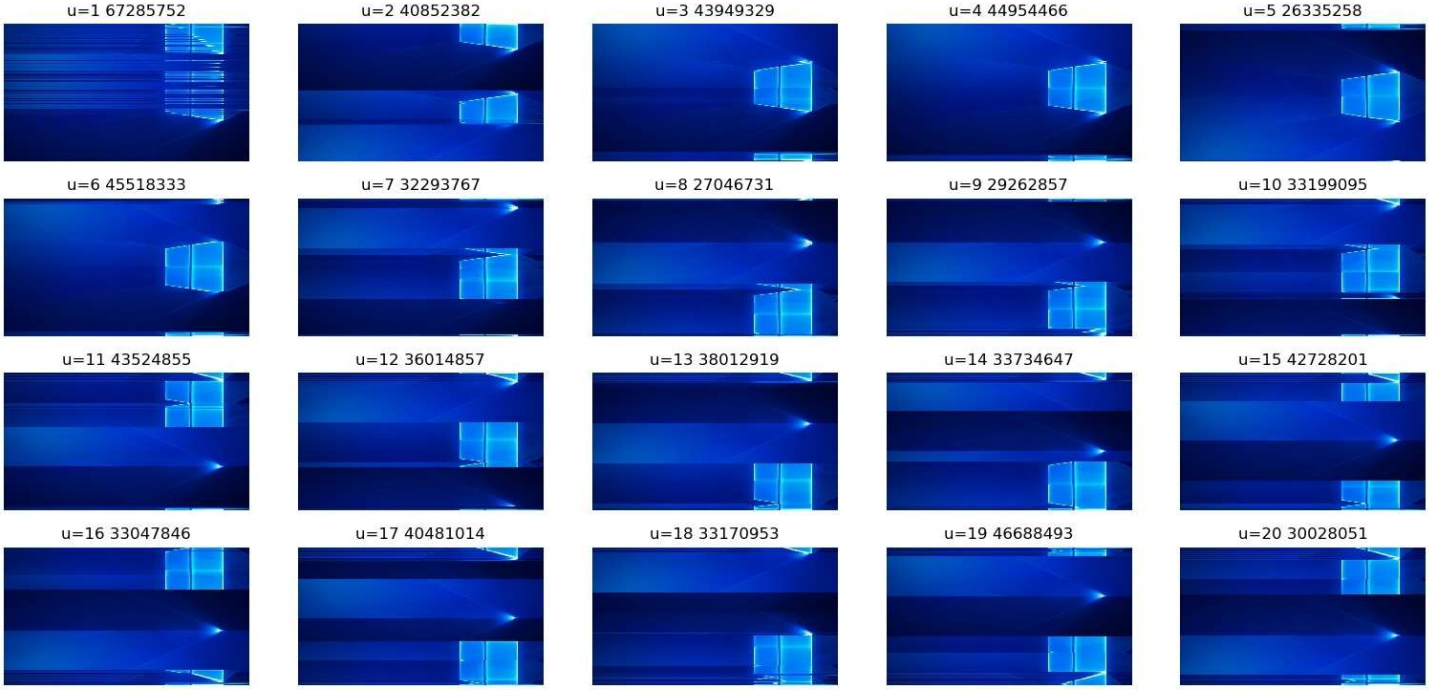

图17. win10 SVD分解贪心法

图18. win10 直接贪心法

win10图片的对称性很好，这是这张图的问题，而且中间的部分总是恢复的有问题，算法认为把上下两个窗放在一起会更好。同时见**图17**可以发现，SVD分解贪心算法的$score$最小的$u = 8$却比较大的$u = 6$效果要差，看来$score$并不适合所有图片。

3.3 Baidu

图19. Baidu图片恢复

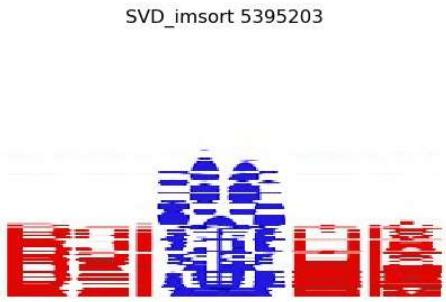

图20. Baidu SVD分解排序

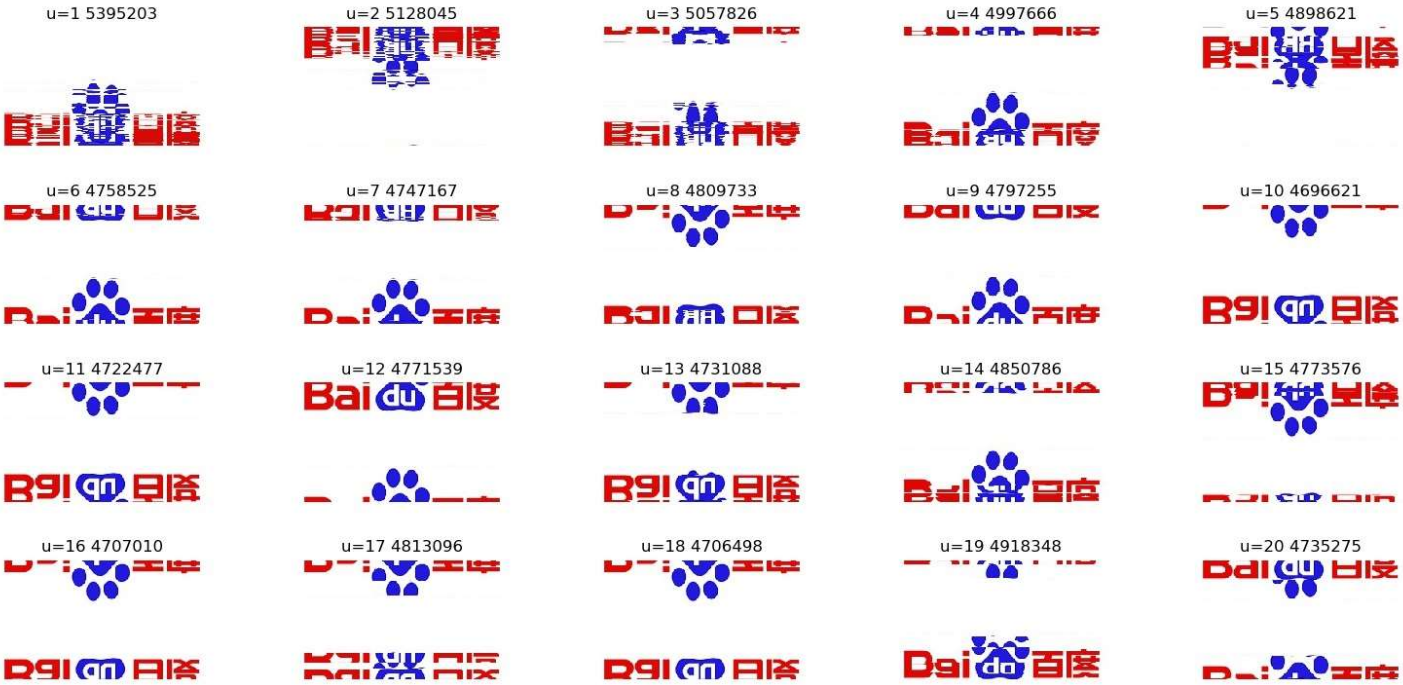

图21. Baidu SVD分解贪心法

图22. Baidu 直接贪心法

Baidu的效果很差，可能是因为白色的地方真的完全是白的，这些白色的行对结果干扰很大，会让中间商标的位置随便上下移动（例如**图21**的$u = 12$）。

3.3 sketch简笔画

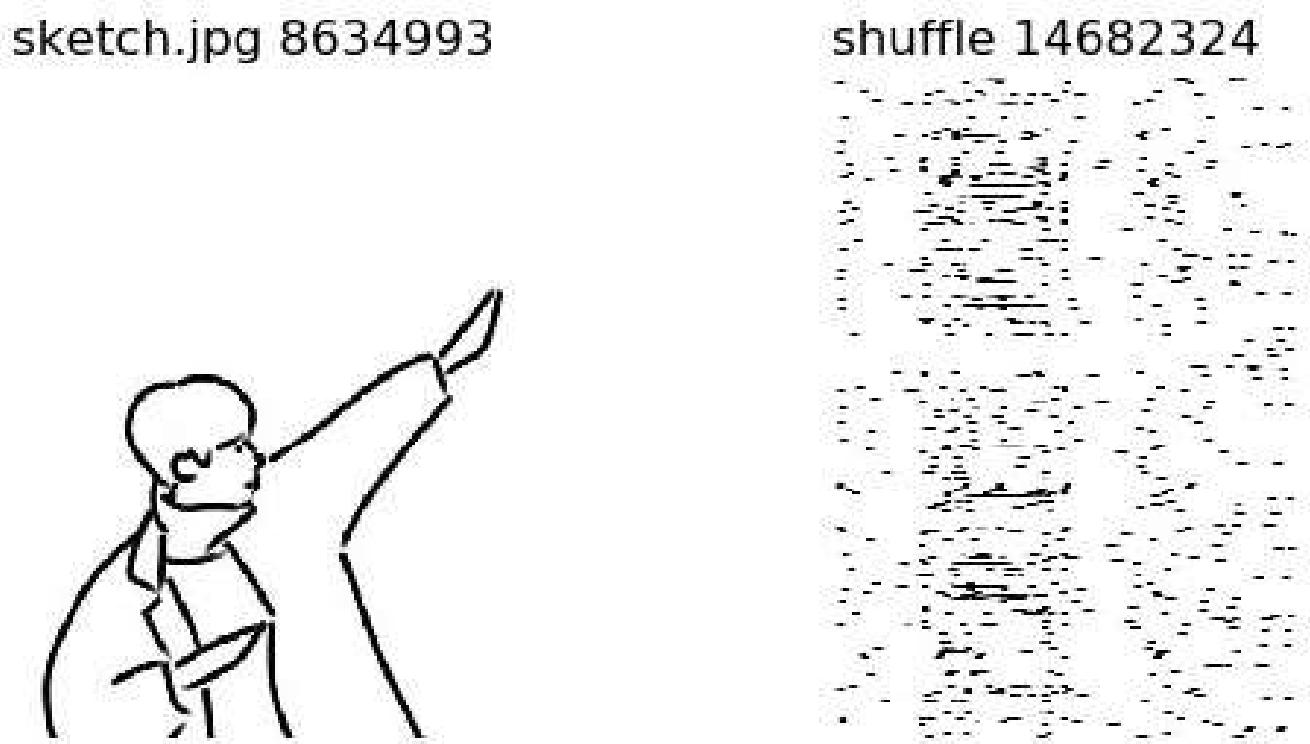

图23. sketch图片恢复

图24. sketch SVD分解排序

图25. sketch SVD分解贪心法

图26. sketch 直接贪心法

简笔画算是特别难的课题了，但没想到SVD分解贪心法效果还不错，远远超过了直接贪心法，这也许可以证明SVD分解贪心法对较难的课题效果比直接贪心法要好。

3.4 QR_code二维码

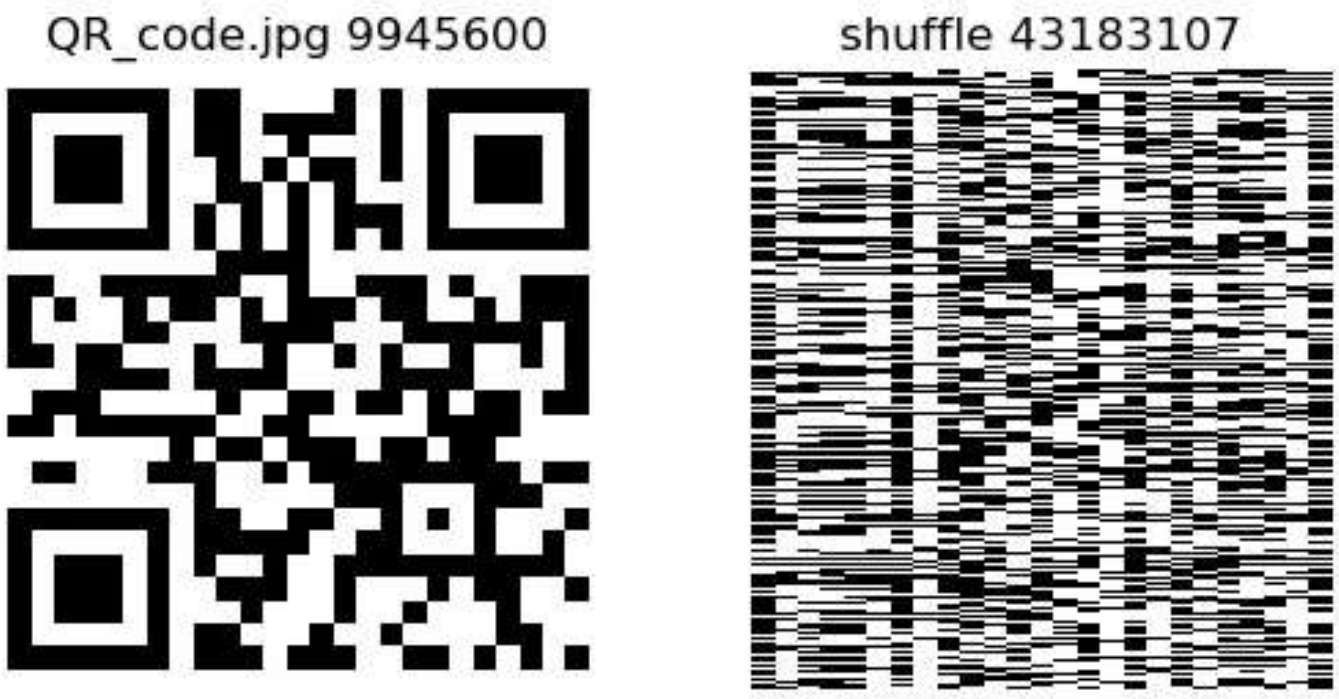

图27. QR_code图片恢复

图28. QR_code SVD分解排序

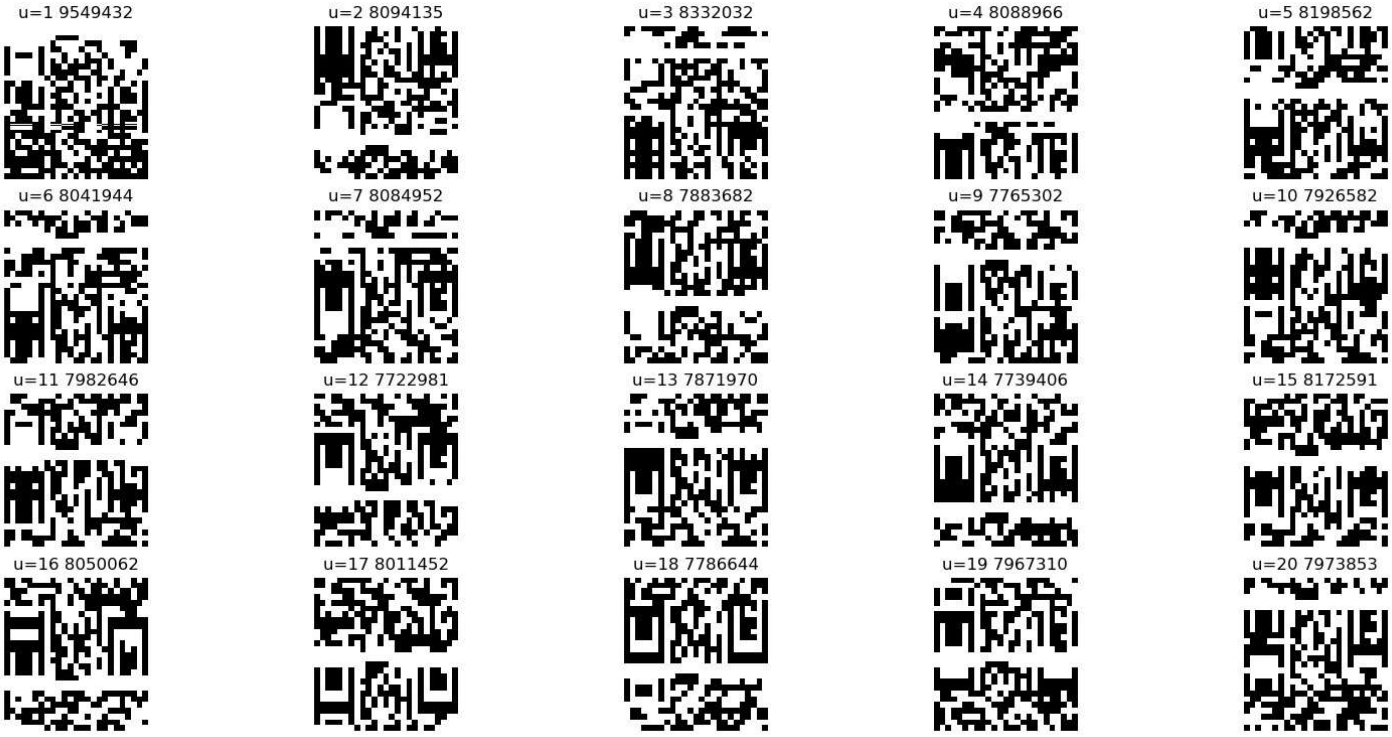

图29. QR_code SVD分解贪心法

图30. QR_code 直接贪心法

二维码也许是最难的挑战了，所有的算法效果都不好，因为二维码跟本就不满足之前图片每一行之间的变化缓慢的假设，**见图29**，可以隐约看出那个识别点能看出了，而**图30**就完全失效了。

3.5 未来的扩展

这里说一下未来本算法的扩展，我觉得可以分成以下3个方面。

1. 多图：我们可以设计对多个图进行打乱，同时进行恢复，这样会让总体效果更好。
2. 图像加密：也许我们可以向图像加密进军，但问题是这些算法有可能无法完全恢复图片，所以也许加密的时候需要加一个标准图片（比较好恢复的图片），让其余需要被加密的图片都和这个标准图片被相同的顺序打乱，这样就可以较好的恢复回去了。
3. 时间序列与CNN：我们可以对矩阵形式的数据（例如时间序列）进行排序，对于时间序列来说，时间维度是不可以动的，但是特征的维度大家一般都没有考虑过，我们可以认为其特征的维度是被打乱过顺序的，我们可以将其恢复成有形状的样子，这样时间序列就可以被CNN进行滤波了（计算机视觉的发展还是属于比较好的领域），这样将不同的领域进行了结合。

4 代码与使用

本项目的代码均上传到github上了，其地址为 https://github.com/QuYue/Images_recover， 有兴趣的话大家可以查看。

其具体**使用方法**如下：

1. **彩图恢复**：

```
在cmd或shell中输入：
python demo.py -i 图片的名字（默认"lena.jpg"） -p 图片的路径（默认"../Data/Images/"）

或者输入查看帮助：
python demo.py -h
结果如下：
usage: demo.py [-h] [-i str] [-p str]

A demo for RGB image recover.

optional arguments:
-h, --help           show this help message and exit
-i str, --image str  the name of the image. (default: 'lena.jpg' )
-p str, --path str   the path of the picture. (default: '../Data/Images/')
```

2. **灰度图恢复**：

```
在cmd或shell中输入：
python demo_gray.py -i 图片的名字（默认"lena.jpg"） -p 图片的路径（默认"../Data/Images/"）

或者输入查看帮助：
python demo_gray.py -h
结果如下：
usage: demo_gray.py [-h] [-i str] [-p str]

A demo for gray image recover.

optional arguments:
-h, --help           show this help message and exit
-i str, --image str  the name of the image. (default: 'lena.jpg' )
-p str, --path str   the path of the picture. (default: '../Data/Images/')
```

To my family,with love

To my friend,with encourage

To my teacher,with gratitude