

**数字媒体技术大作业报告**

题 目 图片的爬取、压缩及恢复

学 院 计算学部

专 业 软件工程

学 号 120L022109 120L021731 120L030713 学 生 李世轩 李景光 王一桐

任 课 教 师 刘绍辉、刘贤明

哈尔滨工业大学计算机科学与技术学院

2023.6

## 项目介绍

这个大作业的灵感来源于，今年的一次热点“微信头像褪色”。有网友反映自己的微信头像出现褪色的情况。用了一段时间的头像照片与原图相比清晰度和色彩都变暗淡了。随后有不少网友纷纷表示自己也遇到类似情况。人们将其戏称为“电子包浆”。

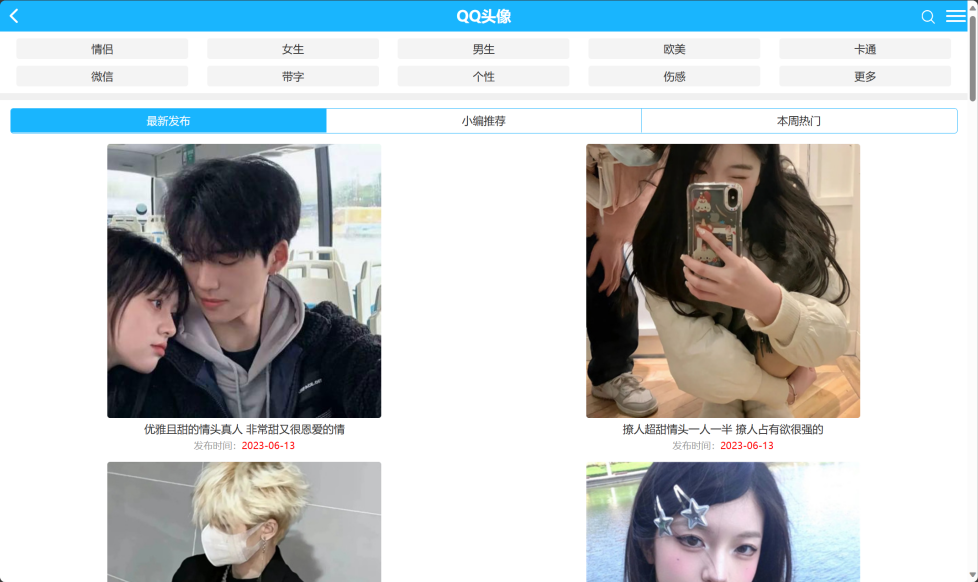


受到该热点的启发，我们决定据此完成这个大作业。首先，为了实现这次大作业，我们需要完成数据的收集：编写爬虫来获取足够的数据。然后，需要完成有损压缩的算法，且尽可能的复现这种“包浆”。最后，通过这些数据，即原本的图片和压缩后的图片，训练一个用于恢复“电子包浆”的超分辨率模型。

## 数据收集

因为深度学习的模型一般需要比较大的数据量才会有较好的效果，所以我们通过编写爬虫，来获取数据。

首先经过搜索，我们找到了一个拥有大量可以当作头像的图片的网站，如下图。



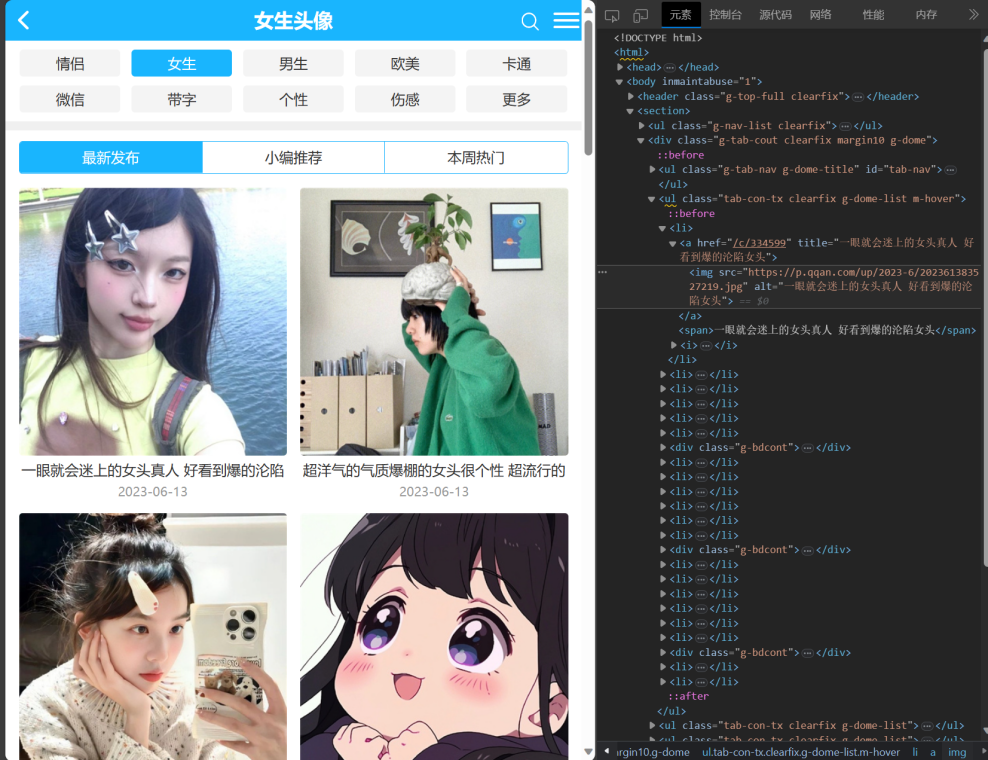
### 源码分析

通过对这个网站源码的分析，我们认为其中的内容具备很强的规律性，可以很方便的爬取。

首先，在这个页面中https://m.qqtn.com/tx\_more.html，有着各种各样头像的标签，如图。查看该网页的源码，发现其中的每个标签都对应一个<a>标签，其中的href即为各个头像标签对应的url，比如女生头像标签对应的“/tx/nvshengtx”。



然后对某一个具体标签页面（如https://m.qqtn.com/tx/nvshengtx）的源码进行分析，我们发现其中各个图像合集也都对应一个<a>标签，其中的href即为各个头像合集对应的url。比如第一个图像合集对应url为：“/c/334599”。

、

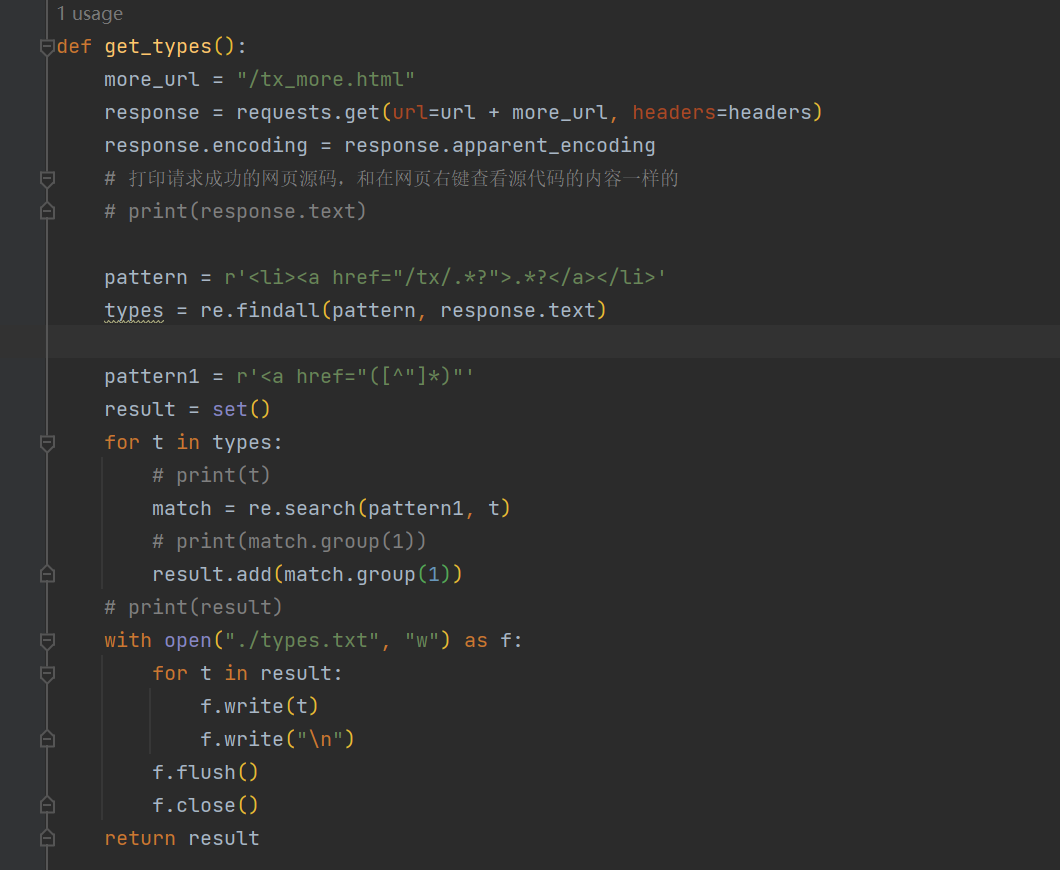
最后查看，某一个图像合集（如https://m.qqtn.com/c/334599）的源码，每一个照片都i对应一个<p>标签，其中嵌套一个<img>标签，比如第一张照片对应“https://p.qqan.com/up/2023-6/16866165153078855.jpg”。到此，我们完成对这个网站的分析。



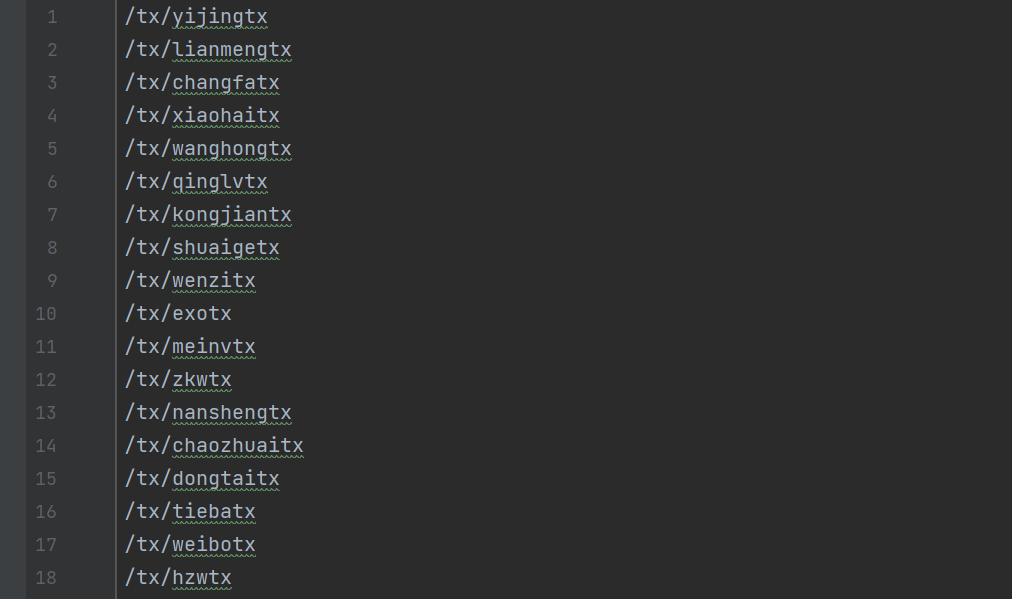
接下来根据这个网站的特点，编写爬虫。

### 爬虫编写

### 首先，获取所有的标签对应的网址。这里介绍大致思路，首先，发送请求。然后，从请求到的文本中使用正则表达式匹配所有标签对应的<li>标签，以此屏蔽掉其余不需要的链接，再从匹配到的<li>中获取对应的href中的内容。然后将其存到文件内。



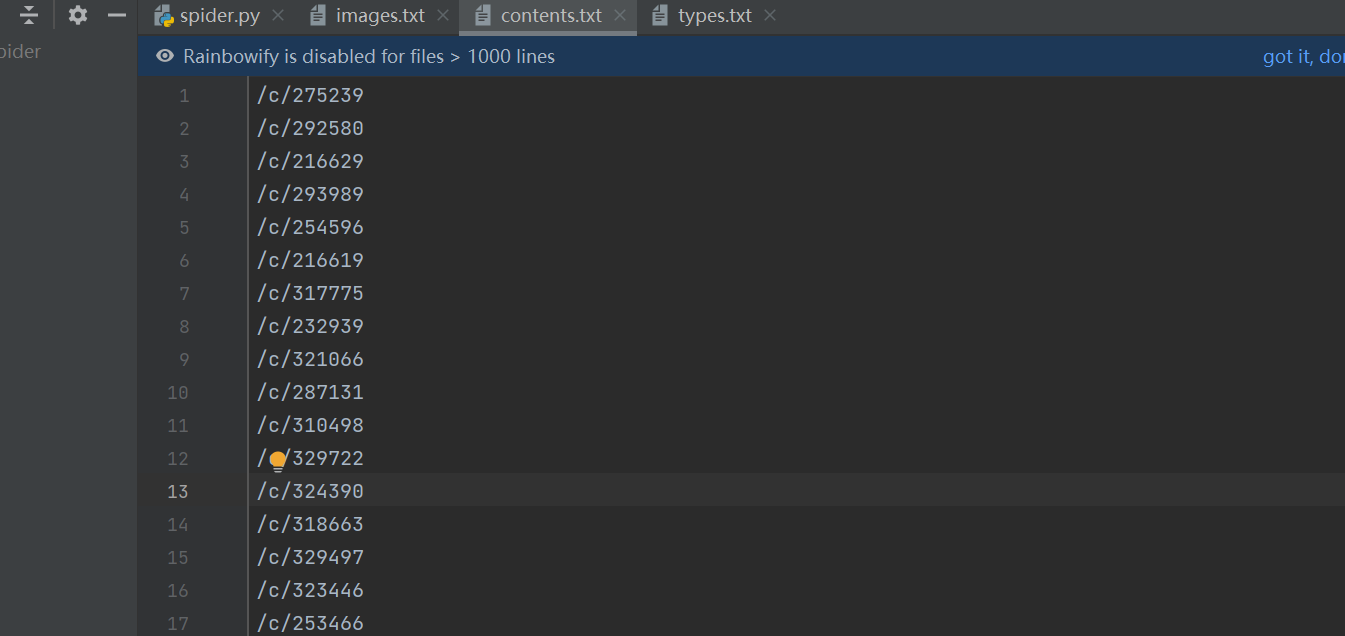
这里展示一下获取到的标签的网址（共66个，只展示一部分）：



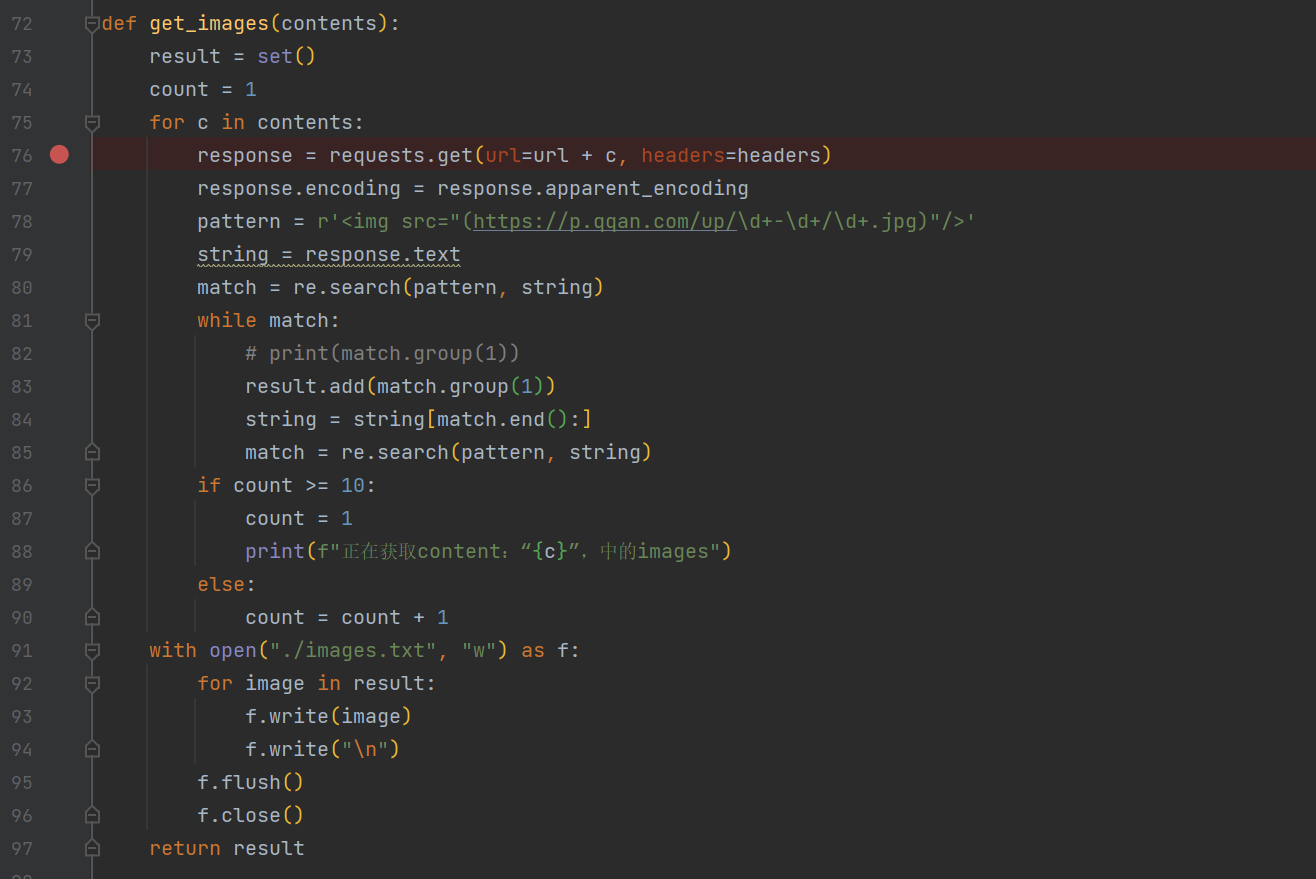
然后，获取每个标签的各个图片合集的网址，遍历传入的每个标签的网址。对每个标签，获取其请求的文本，在其中查找r'<a href="(/c/\d+)"'这个正则表达式对应的所有图片合集的网址。再将其存入集合中，这一步是为了去重。然后再将结果存到文件中

### 

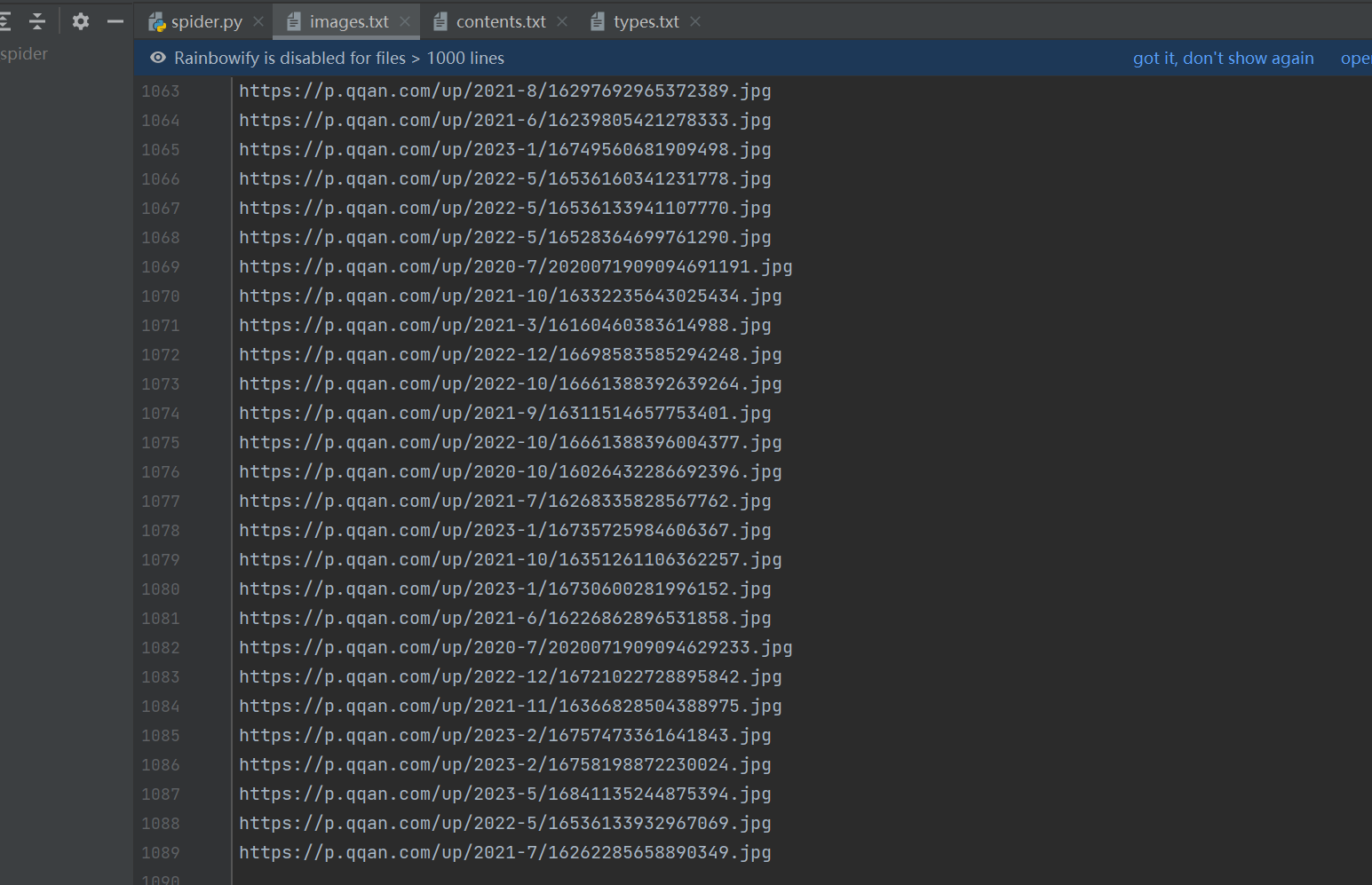
下面展示获取到的图片合集的连接，共2397条：



最后获取图片的链接集合，发现该网站上的图片的格式都符合这个格式：https://p.qqan.com/up/\d+-\d+/\d+.jpg，所以使用这个正则表达式来获取图片链接，其他部分与获取合集链接是一致的，不再赘述。



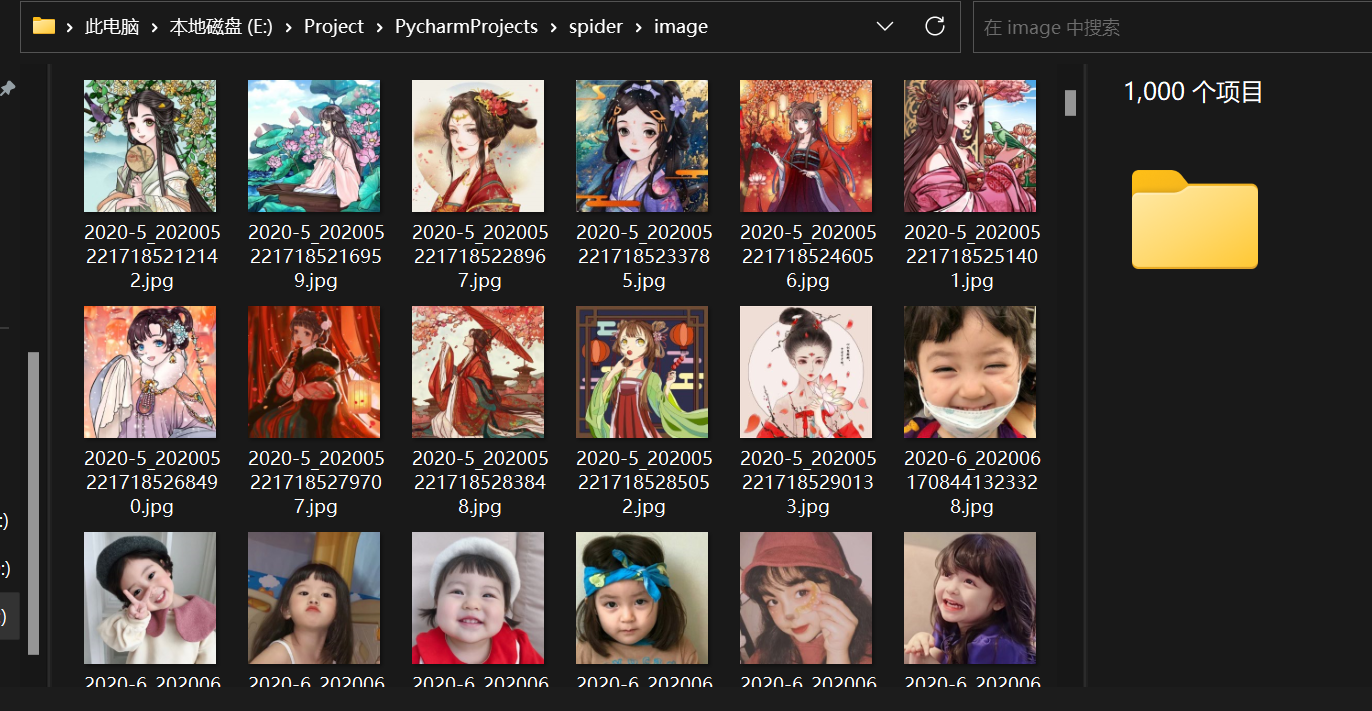
### 下面展示获取到的图片的链接，为了减少运行时间，这里只使用了前200个合集来获取图片，共获得1089个图片的链接。



最后根据图片链接下载图片并将其存到本地即可，这里再次使用正则表达式，是为了将链接中的两部分提取出来，当作本地文件的命名。



### 这里只使用了前1000个照片链接。



## 图片压缩

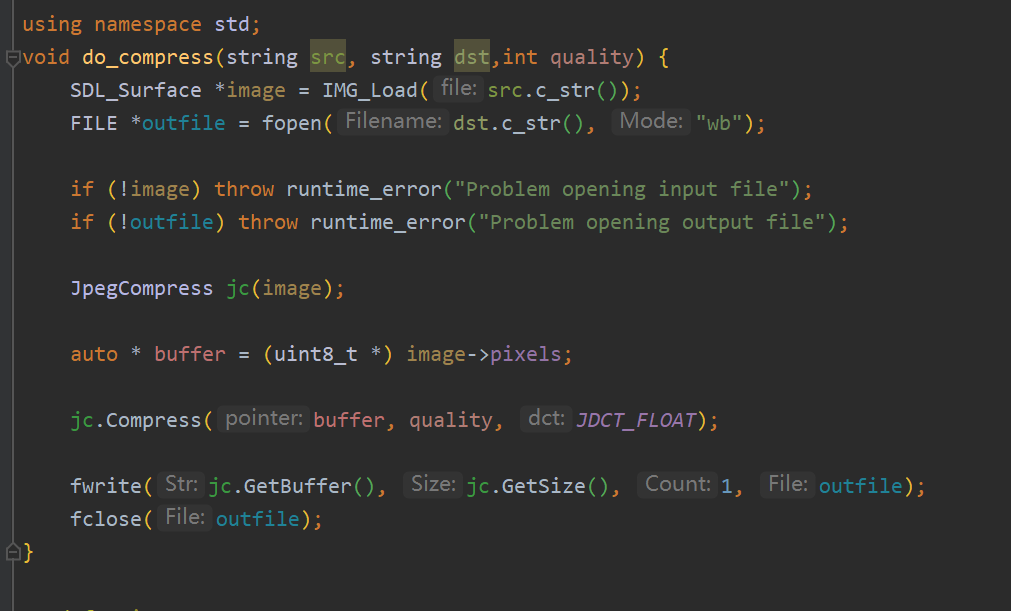
这里采用两种办法来压缩图片。

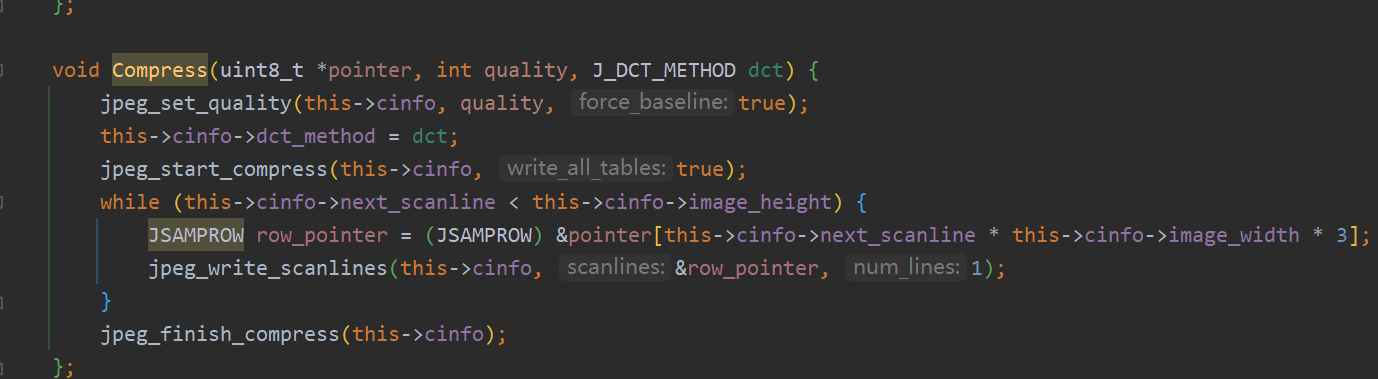
### 随机质量多次压缩

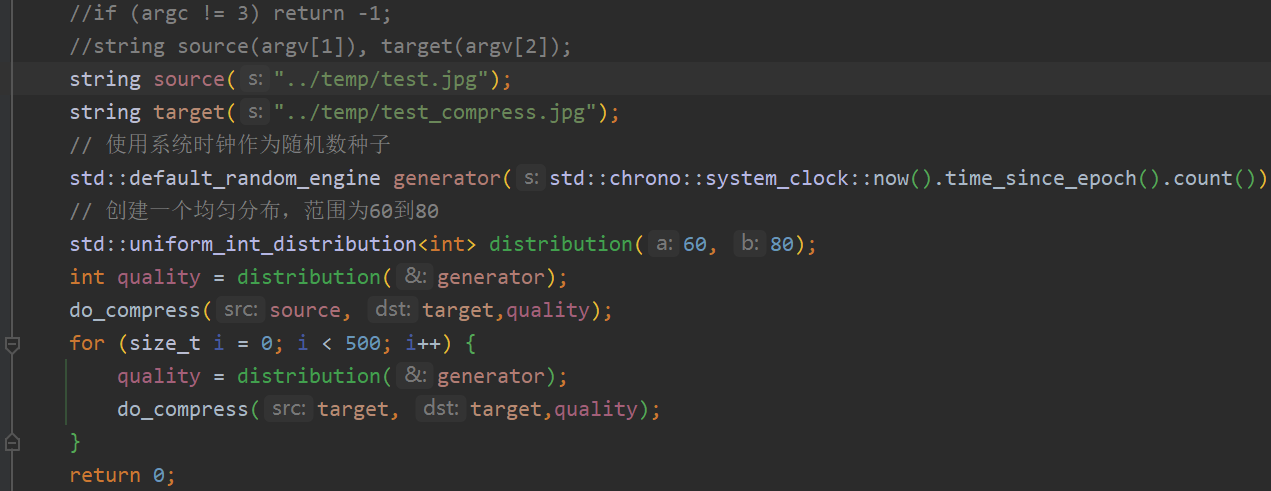
为了模拟真实的网络环境，我们用随机数来作为压缩质量因子，来多次压缩图像。一般来讲，一张照片会随着人们的交流，通过各个渠道来不断传播，而这些渠道对图像压缩时，很有可能会采取不同的压缩质量（这个值我们是无法得知的，所以这里采取随机数）。

该方法受到哔哩哔哩博主epcdiy的视频《"电子包浆"是因为被反复压缩吗？其实没那么简单》[2]的启发。

该方法使用libjpeg这个经典的库来实现，首先使用SDL\_image读入图片，再使用libjpeg进行压缩处理，并在压缩时随机产生质量因子。







接下展示，该算法产生的结果。



### 精度损失绿化压缩

我们发现图片“包浆”的问题其实早在这之前，就已经产生了。其实早在2016年就对这个问题有讨论。

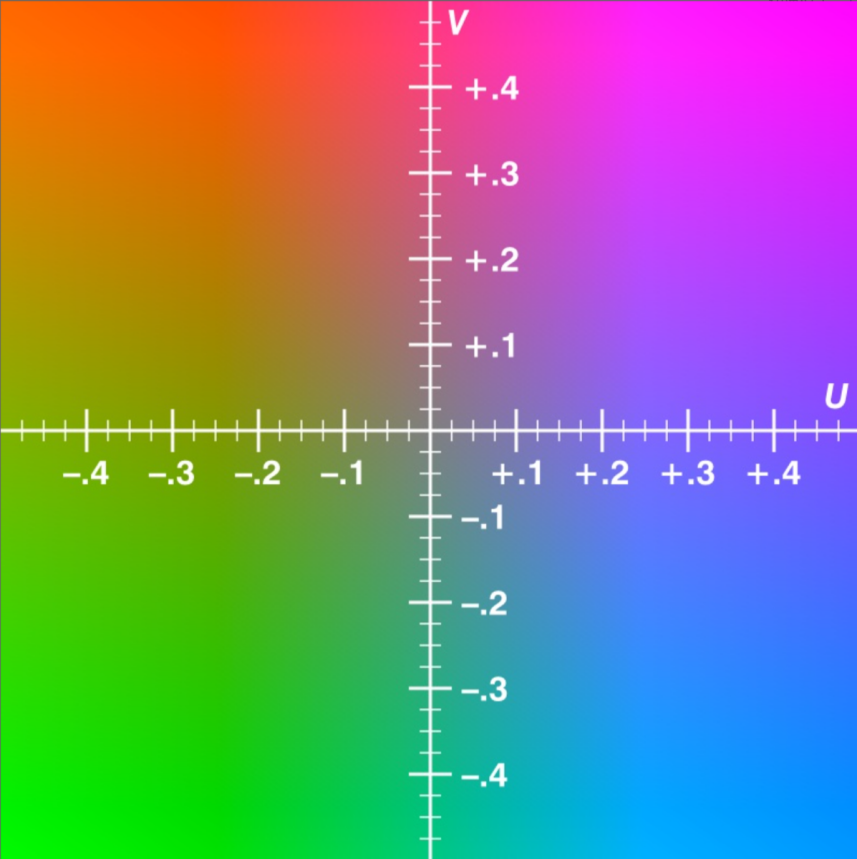


截图截自《为什么图片反复压缩后普遍会变绿，而不是其他颜色？》[3]，其中对这个问题进行了详细的讨论。

图片变绿其实是安卓操作系统核心代码的一个bug。安卓系统给开发者提供了一个图像压缩接口，可以让开发者方便地压缩JPEG图片（也就是jpg图片）。但这个接口的底层实现算法为了加速压缩计算过程，在色彩模式转换的过程中出了一个bug。

我们在手机屏幕上看到的图片都存储着RGB信息（Red红、Green绿、Blue蓝），它能告诉屏幕上每个红绿蓝子像素应该以何等亮度发光，从而在屏幕上显示出图片的样貌。但在图像处理过程中，一般需要RGB信息转换成YUV信息（亮度、蓝色浓度偏移量、红色浓度偏移量）。因为人眼对Y代表的亮度信息更为敏感，算法可以着重压缩UV信息。这样就能在人眼感知差别不大的情况下，尽可能减小图片所占的存储空间。

一般而言，从RGB色彩模式转换到YUV色彩模式是轻微有损的，但损失较小，并不至于让图片朝着变绿的方向一路狂奔。但是开发者为了加速这个转换计算过程，不当地使用了位运算，导致数据在从RGB向YUV转换时会向下取。所以在重复压缩过程时， Y、U、V三个值就会不断减小，亮度Y值减小会让图片不断变暗，而UV不断减小，会让色彩不断向绿色的方向偏移（见下图）。所以，经过多次压缩的图片会变绿、变暗。



Android 系统自起诞生以来就引入了名为 Skia 的图像库（Google 自家产品），用于处理图像，其中包括把图片压缩成 JPEG。而 Skia 又是调用 libjpeg-turbo 来实现真正的压缩过程的。为了达到更好的压缩效果，JPEG 算法本身，将通常屏幕上表示颜色的 RGB数值，转换为 YUV 数值。正常情况下这个算法是轻微有损的。

但是 Skia 将这个变换算法的各个常数复制到自己的代码里，然后降低了精度，以达到更高的速度（从 16 位定点数，降低到了 8 位定点数），这导致了更大的损伤。

最可怕的是，在进行这个变换运算的最后一步，需要除以 256，而代码中，采用了右移操作代替除法以提高执行速度：

int y = ( CYR\*r + CYG\*g + CYB\*b ) >> CSHIFT;

int u = ( CUR\*r + CUG\*g + CUB\*b ) >> CSHIFT;

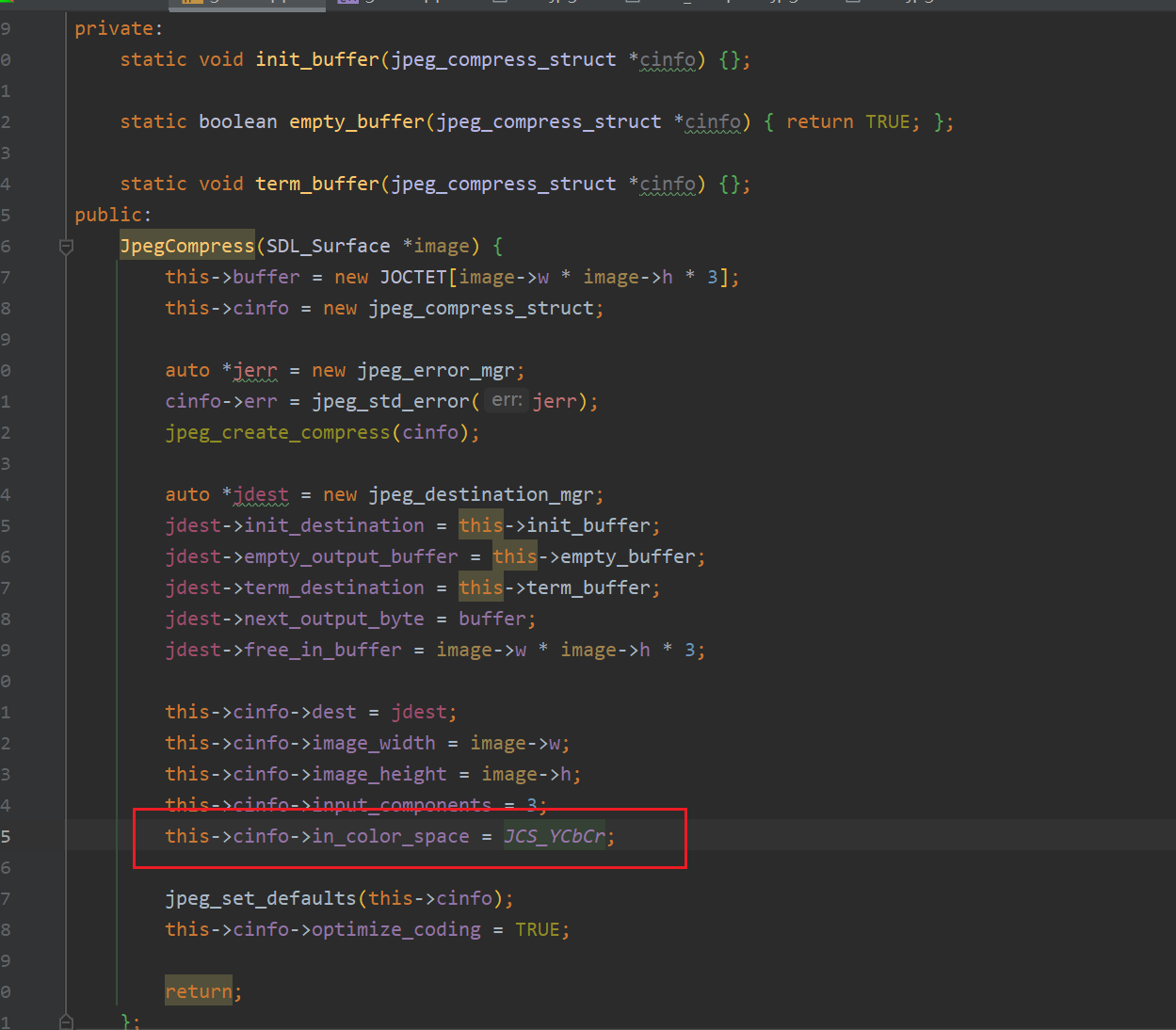
int v = ( CVR\*r + CVG\*g + CVB\*b ) >> CSHIFT;

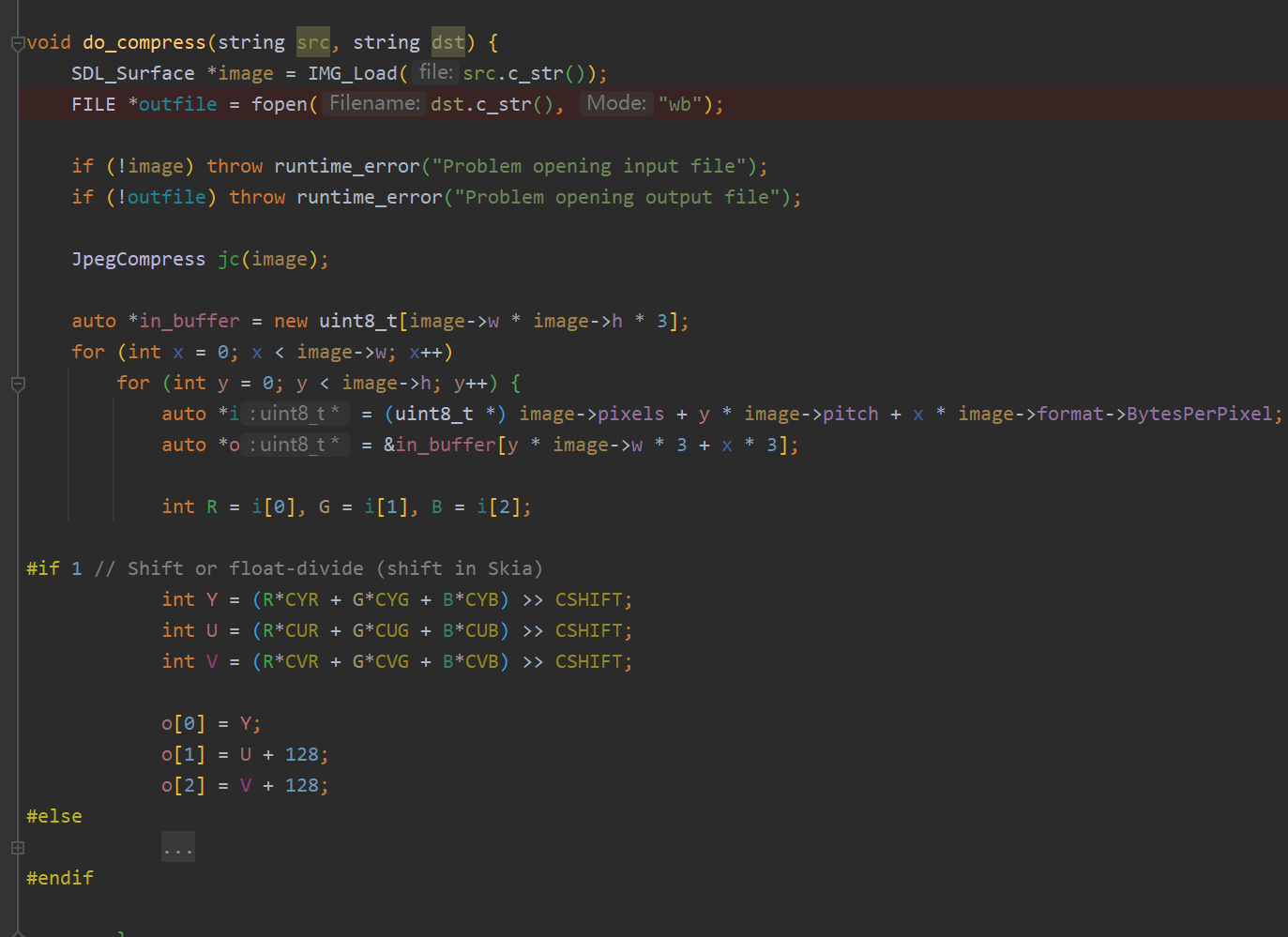
当然这个问题现在已经被修复。[Use libjpeg-turbo for YUV->RGB conversion in jpeg encoder · google/skia@c7d01d3 · GitHub](https://link.zhihu.com/?target=https://github.com/google/skia/commit/c7d01d3e1d3621907c27b283fb7f8b6e177c629d" \t "https://www.zhihu.com/question/_blank)

这个压缩算法的实现参考：GitHub - LionNatsu/greenError: Discover the reason how `terribleGreen`(my another repo.) works on Android.[4]

基本流程与前面的多次压缩算法基本相似，但是需要将jpeg\_compress\_struct的in\_color\_space从JCS\_RGB修改为JCS\_YCbCr。并手动完成从RGB到YUV的转化。

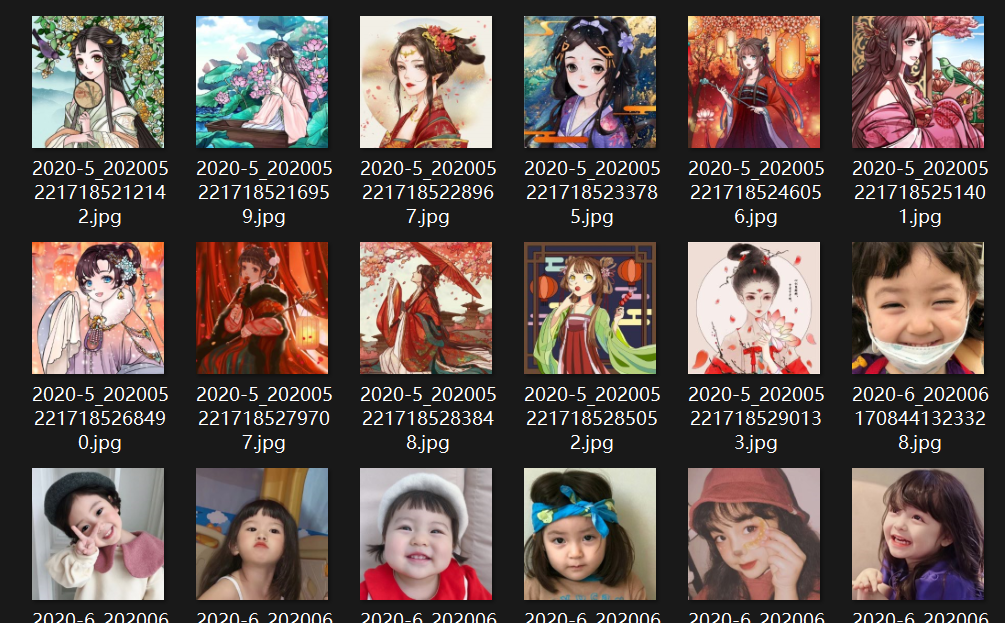


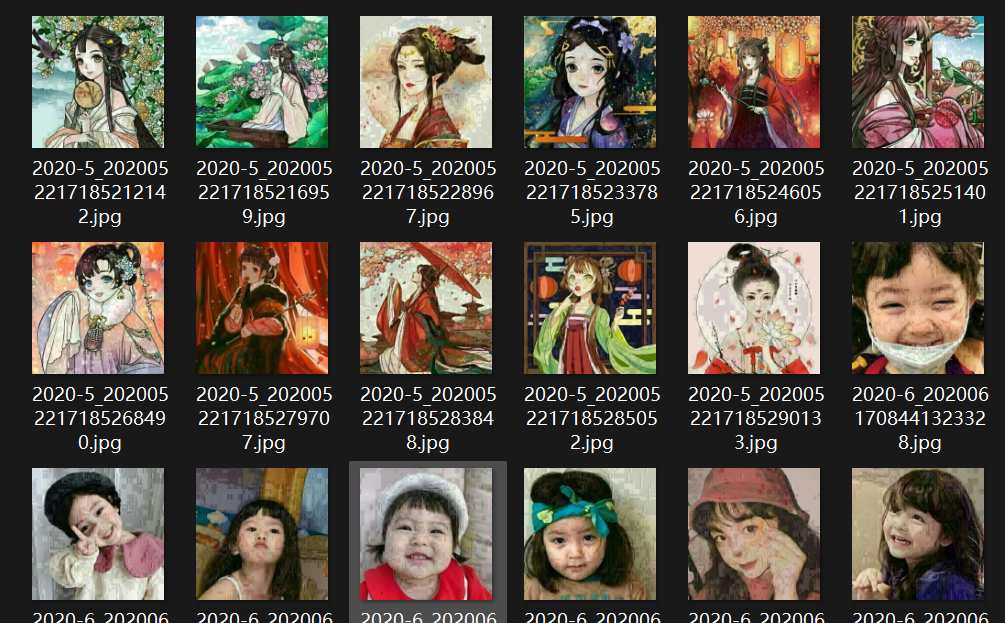




接下来展示实现效果：







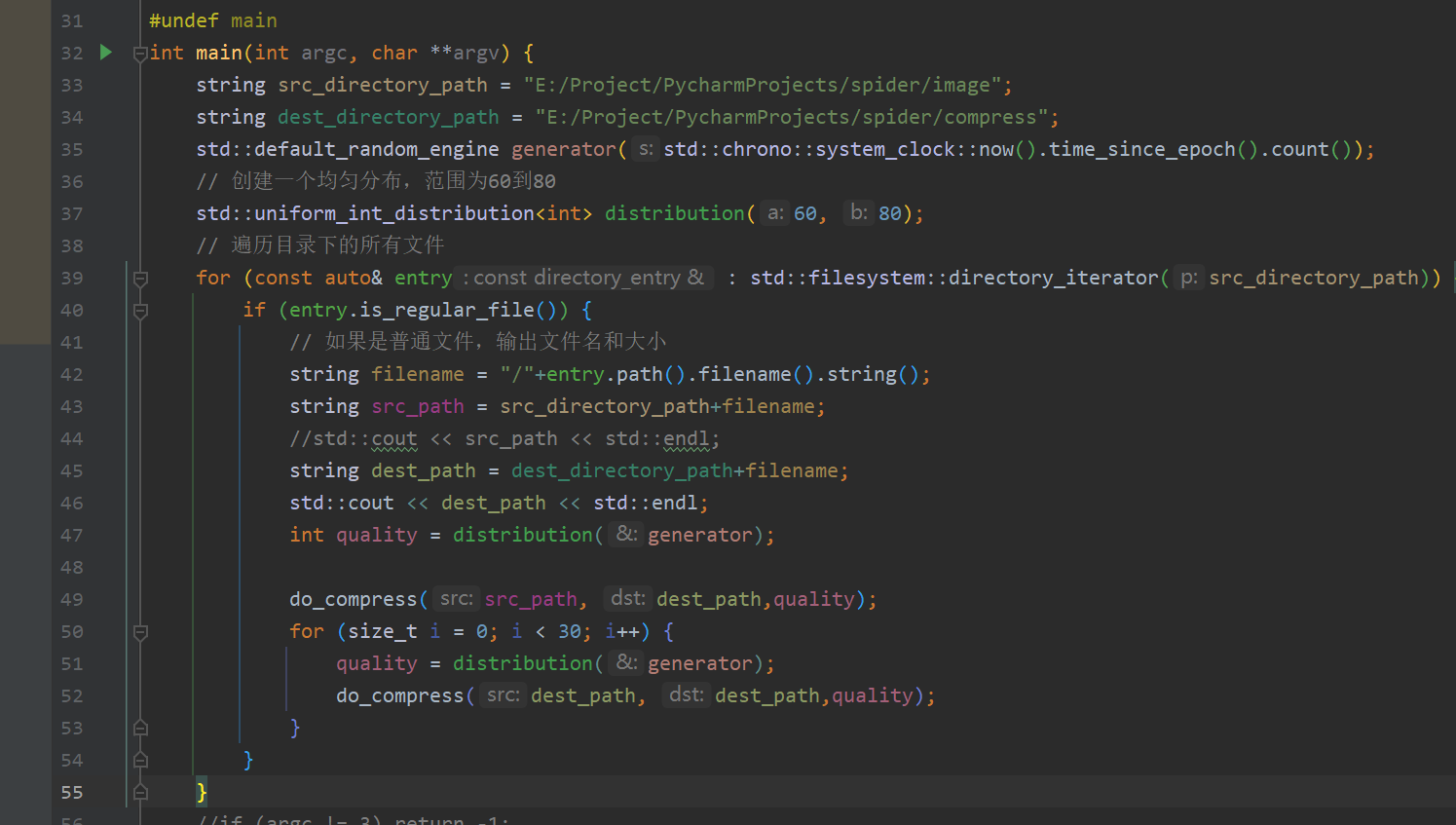
可以看到处理后的照片有比较明显的变绿变暗。

## 训练模型

### 数据准备

在训练模型前需要将爬取到的数据分别使用两种算法进行处理。





### 代码实现

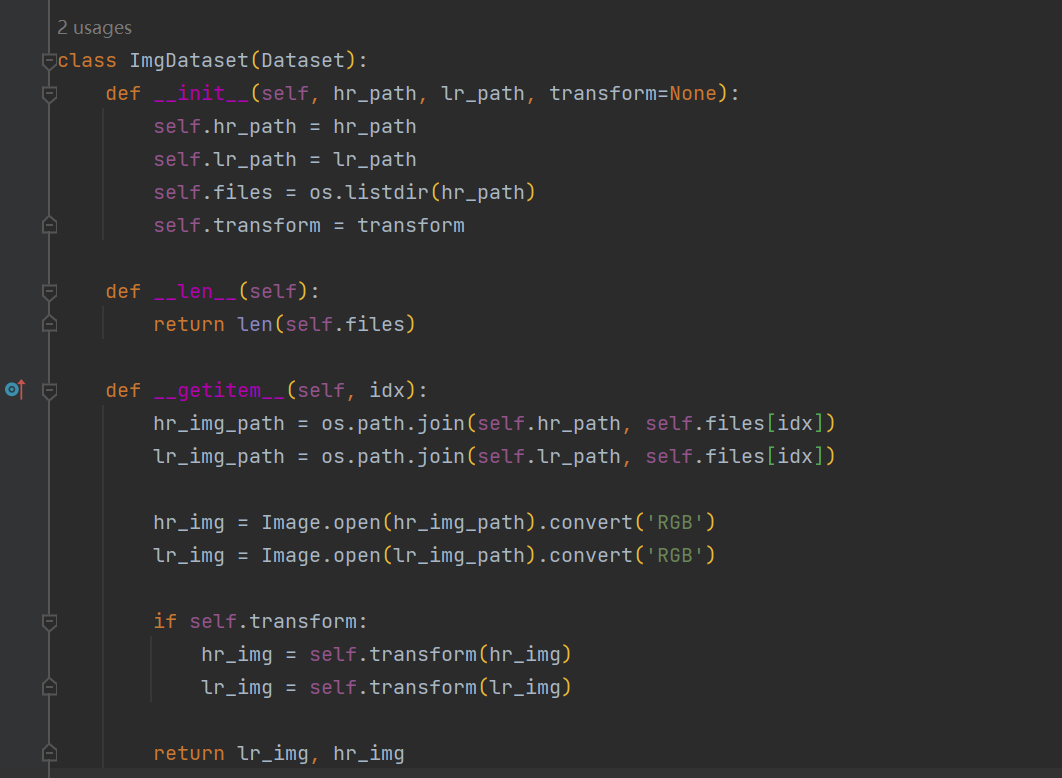
这里使用超分辨率领域的经典模型SRCNN。

SRCNN（Super-Resolution Convolutional Neural Network）是一种用于图像超分辨率重建的深度学习模型。SRCNN模型通过学习低分辨率图像和对应高分辨率图像之间的映射关系，可以将低分辨率图像重建成高分辨率图像。

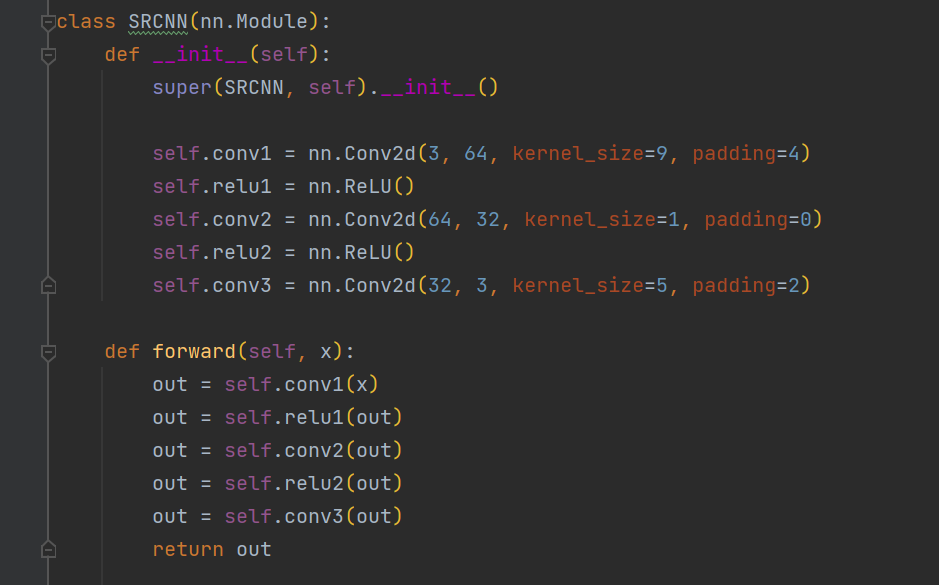
SRCNN模型在超分辨率重建任务上表现出色，在许多标准数据集上都取得了优秀的结果。与传统的插值算法相比，SRCNN模型能够更好地保留细节信息，并且具有更好的视觉效果。此外，SRCNN模型还可以适应不同的场景和噪声条件，具有很强的泛化能力。

代码实现如下，这里使用pytorch实现：

先定义自己的数据集，输入两个路径，一个为存放高分辨率图像的路径（未处理的原图），一个为存放低分辨率图像的路径（多次随机压缩或绿化压缩）。



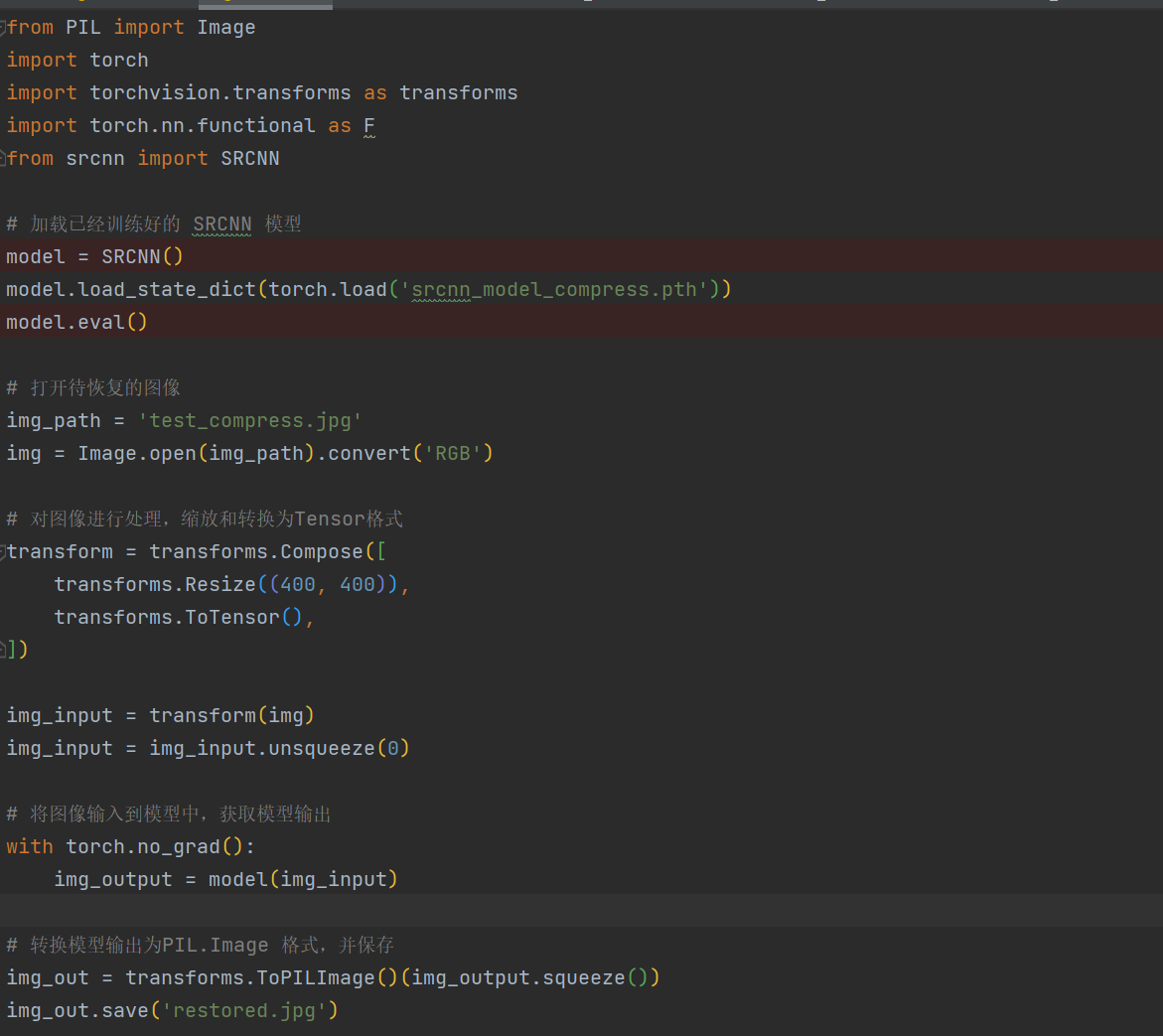
然后定义模型：



对模型进行训练

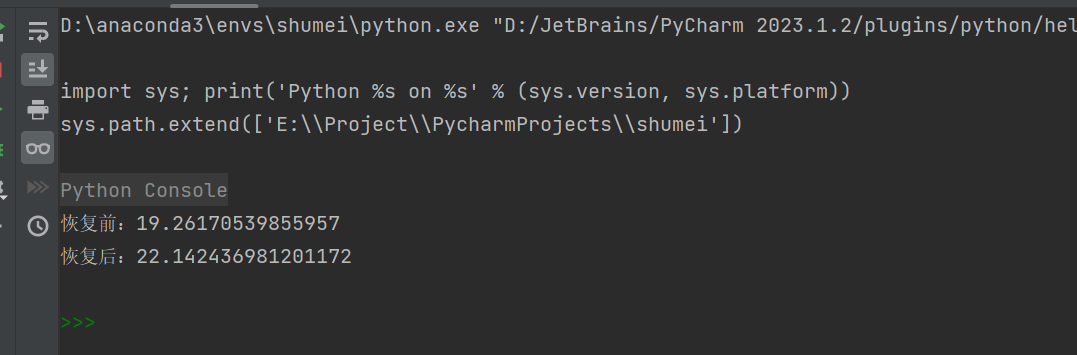
### 

### 使用模型



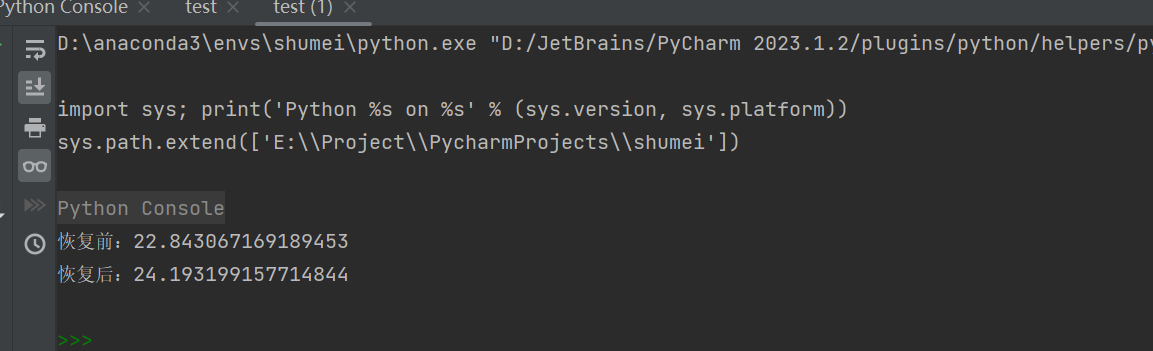
测试随机压缩后照片与原始照片的psnr以及恢复后照片与原始照片的psnr，可以看到有所提升。





测试绿化压缩后照片与原始照片的psnr以及恢复后照片与原始照片的psnr，可以看到有所提升。





## 参考文献

[1] Dong C, Loy CC, He K, Tang X. Image Super-Resolution Using Deep Convolutional Networks. IEEE Trans Pattern Anal Mach Intell. 2016 Feb;38(2):295-307. doi: 10.1109/TPAMI.2015.2439281. PMID: 26761735.

[2] ["电子包浆"是因为被反复压缩吗？其实没那么简单\_哔哩哔哩\_bilibili](https://www.bilibili.com/video/BV1AY411S75C/?spm_id_from=333.1007.top_right_bar_window_custom_collection.content.click&vd_source=b7c2329a8bfdbf1ea6e499aae7bbeebd)

[3] [为什么图片反复压缩后普遍会变绿，而不是其他颜色？ - 知乎 (zhihu.com)](https://www.zhihu.com/question/29355920?utm_id=0)

[4] https://github.com/LionNatsu/greenError