

17

Non-Vector Images

非矢量图片

本质上为红绿蓝像素数据矩阵



博爱是好事，因为真正的力量就蕴藏在其中。深爱之人会付出更多，做成更多事情，并且能够取得更大的成就。而在爱的驱使下所做的一切都会变得出色。

It is good to love many things, for therein lies the true strength, and whosoever loves much performs much, and can accomplish much, and what is done in love is well done.

—— 文森特·梵高 (Vincent van Gogh) | 荷兰后印象派画家 | 1853 ~ 1890



- ▶ `matplotlib.image.imread()` 读取图像文件并返回对应的图像数据
- ▶ `matplotlib.pyplot.hist()` 绘制直方图
- ▶ `matplotlib.pyplot.imshow()` 显示图像数据
- ▶ `numpy.zeros()` 返回给定形状和类型的新数组，用零填充
- ▶ `numpy.zeros_like()` 用来生成和输入矩阵形状相同的零矩阵
- ▶ `skimage.color.rgb2gray()` 将彩色图像转换为灰度图像
- ▶ `skimage.io.imread()` 读取图像文件并返回对应的图像数据

17.1 像素

matplotlib.image 模块提供了读取和处理图像的函数，其中最常用的函数是 `imread`。`imread` 函数可以读取图像文件，并将其解码为一个三维的 `numpy` 数组。

`imshow` 是 matplotlib 中用于显示图像的函数。将如图 1 所示鸢尾花照片导入后，容易发现这幅图像实际上是一个 $2990 \times 2714 \times 3$ 的数组。

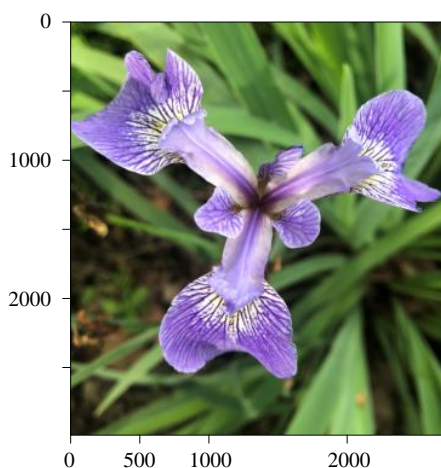


图 1. 鸢尾花照片

图片像素 (pixel) 是图片的基本单位，是构成图片的最小元素。它是一个有限的、离散的、二维的点，有着特定的位置、颜色和亮度值。在数字图像中，每个像素都有一个确定的坐标和值。图片中的像素数量越多，图片的分辨率就越高，图片的清晰度和细节也就越好。

像素的颜色通常使用 RGB 值 (红、绿、蓝三种颜色的强度组合) 表示。每个像素都有一个红、绿、蓝三个通道的值。红、绿、蓝可以分别被编码为一个数字，例如 8 位的数字可以表示 256 种颜色。

也就是说，图 1 这幅图中每个像素首先分解成红绿蓝三个数值。这些数值的取值范围都在 $[0, 255]$ 之间。换个角度，图 1 可以理解成是由三幅图片叠加而成，如图 2 所示。

此外，我们可以获得如图 3 所示的红绿蓝颜色的分布。越靠近 0，颜色越靠近黑，越靠近 255 颜色越靠近纯色。本书前文已经和大家聊过 $[0, 0, 0]$ 代表纯黑， $[255, 255, 255]$ 代表纯白。注意，在 matplotlib 中 $[1, 1, 1]$ 代表纯白。

在彩色图像中，每个像素的颜色可以由三个 8 位数字 (红、绿、蓝) 组成，因此彩色图像中的每个像素可以表示 $2^{3 \times 8}$ 种不同的颜色，约为 1600 万种。

在数字图像处理中，对图像进行各种操作，例如缩放、旋转、裁剪、调整亮度和对比度等，都会涉及到像素的处理和修改。

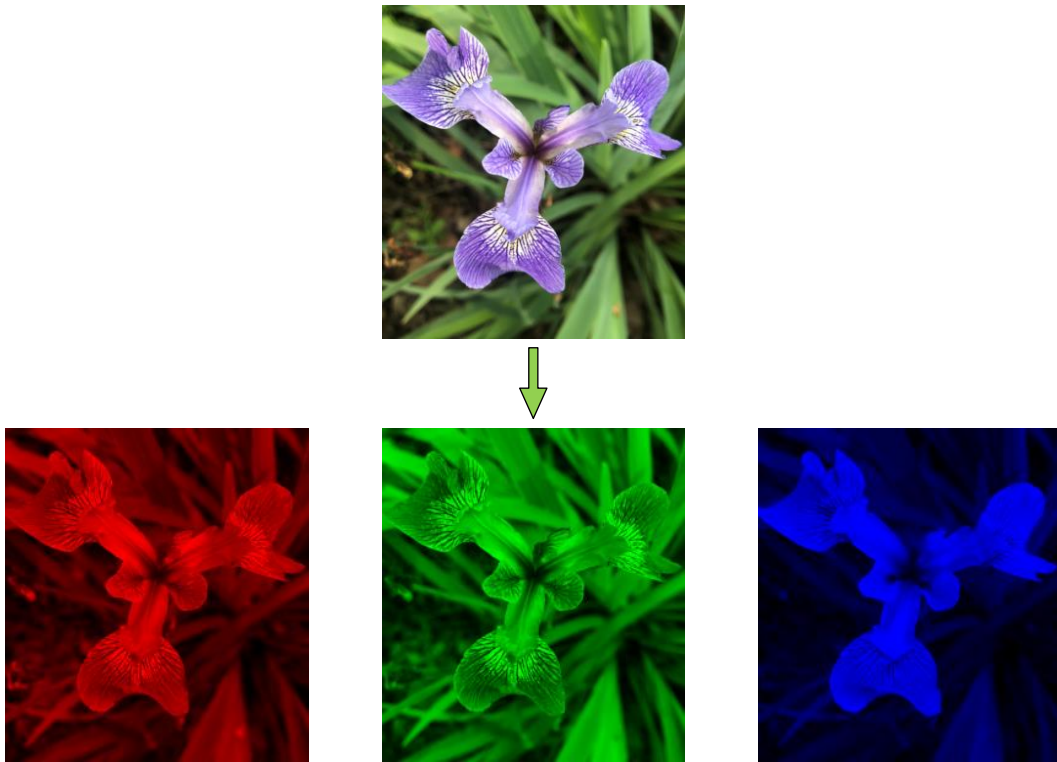


图 2. 鸢尾花照片分解成红绿蓝三个通道

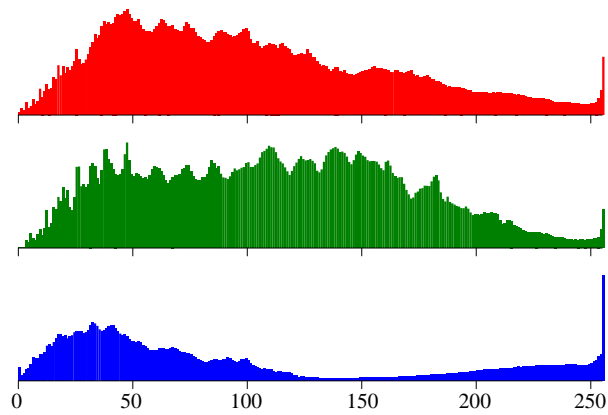


图 3. 鸢尾花照片红绿蓝颜色分布

红绿蓝三个通道

图 4 给出的三幅子图，每幅图仅保留两色通道，另外一个通道数值全部置零。



图 4. 鸢尾花照片，只保留两色通道

色谱

它可以用来显示二维数组或图像文件中的图像。imshow 函数有很多参数可以控制图像的外观。例如，可以使用 cmap 参数指定要使用色谱。图 5 所示为使用色谱展示红色通道。Jupyter notebook 中还给出更多范例。

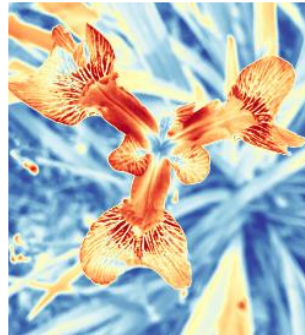


图 5. 使用色谱展示红色通道

灰度

Scikit-image (skimage) 是一个用于图像处理和计算机视觉的 Python 包。它提供了一系列算法，函数和工具，可用于图像处理，包括图像滤波，几何变换，色彩空间转换，图像分割，特征提取等等。具体来说，skimage 可以用于：a) 加载和保存图像；b) 调整图像大小，旋转，裁剪等几何变换；c) 进行图像滤波和增强；d) 在不同颜色空间之间进行转换；e) 检测边缘和角点；f) 进行图像分割和分析；g) 进行特征提取和图像匹配。

图 6 所示为使用 skimage 将彩色图片转化为灰度图片。注意图片的每个像素的取值在 $[0, 1]$ 之间。此外，图像识别一般都使用灰度图像。



图 6. 将彩色图片转化成灰度

修改部分像素

由于图片本身就是一个数组，我们可以通过修改数组的具体值来修改图片。如所示，我们将灰度照片的左上角 500×500 的像素变为白色。



图 7. 修改图片像素

降低像素

图 8 所示为通过采样降低图像像素。图 1 这幅图片的像素大小为 2990×2714 。每 200 个像素采样一个像素，我们便得到图 8。这幅图的像素为 15×14 ，很明显图片的颗粒度很粗糙。

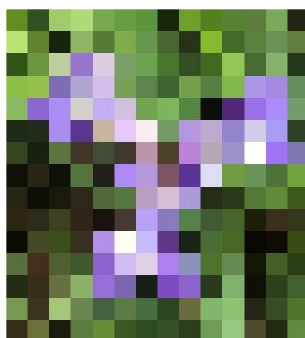


图 8. 采样降低像素

当图像像素较低时，为了让图片看上去更细腻，我们可以采用插值。

17.2 插值

`imshow()` 函数中，我们可以通过设置 `interpolation` 参数来控制如何在图像像素之间进行插值，以生成更平滑的图像。

`imshow()` 函数 `interpolation` 参数的默认值是 `'antialiased'`，它使用反走样技术来平滑图像，使其在缩放时更加清晰。这意味着在缩放图像时，`imshow()` 函数会自动对图像进行插值，以获得更平滑的外观。

除了默认的 `'antialiased'` 插值，`imshow()` 函数还支持其他插值方法，包括 `'nearest'`，`'bilinear'`，`'bicubic'` 等。这些插值方法可以通过 `interpolation` 参数来设置。例如，`'nearest'` 插值只是在最近的

像素值之间进行插值，而 'bicubic' 插值使用更复杂的算法来生成更平滑的图像。图 9 所示为图 8 的两种插值结果。本节的 Jupyter notebook 中给出更多插值方法。

《数据有道》一册将详细讲解常见插值算法。

选择不同的插值方法会影响图像的视觉效果，因此选择合适的插值方法可以使图像更清晰或更平滑，更符合数据的视觉表达。

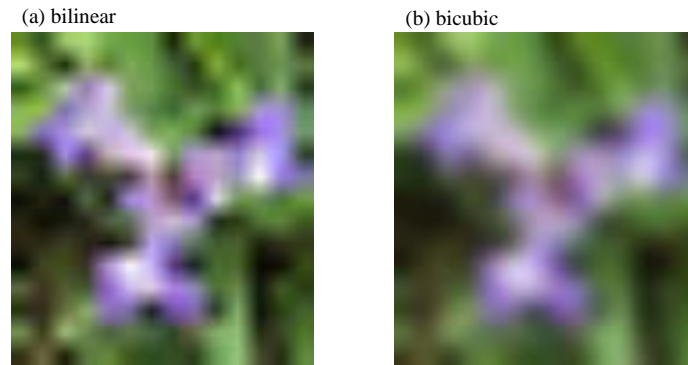


图 9. 插值平滑