Numpy 和图像基础

Jupyter Lab

安装：

```
conda install -c conda-forge jupyterlab

conda install jupterlab （换源后）
```

或：

```
pip install jupyterlab
```

运行：

```
jupyter-lab
```

用浏览器打开地址： http://localhost:8888/

小提示：在 jupyter notebook 里面，使用 shift+tab 键可以唤出函数的帮助文档。

Numpy 基础用法

一个列表，可以使用 np.array(list_a) 转化成 np 的列表类型 numpy.ndarry。

np.arrange(0,10) 可以快速创建 0~9 的一维数组。

np.arrange(0,10,2) 可以快速创建 0~9 的偶数一维数组。

np.ones() 可以创建一个全是 1 的数组。

np.zeros() 可以创建一个全是 0 的数组。

np.ones(shape=(10,3)) 可以创建创建一个 10x3 的数组。

np.random.randint(0,100,10) 可以创建 10 个 0~99 的随机整数。

arr.reshape((5,2)) 可以将数组 arr 转成 5x2 的形式。

为了了解图像本质，我们需要先了解一下数组和矩阵的概念

首先回忆一下，如何创建一个列表

```
list_a = [1,2,3,4,5]
```

打印一下

```
list_a
```

```
   >[1,2,3,4,5]
```

使用 type 命令查看数据类型，

```
type(list_a)
```

```
   >list
```

可以看到是 Python 的列表

再创建一个列表，使这个列表的元素仍然是列表

```
list_b = [ [1,2], [3,4], [5,6] ]
```

打印列表

```
list_b
```

```
   >[ [1,2], [3,4], [5,6] ]
```

通过找数字索引，打印第 list_b 第 2 个元素的第 1 个元素

```
list_b[1][0]
```

```
   >3
```

这种结构我们也叫数组，比如 list_a 是一维数组，list_b 是二维数组

为了更高效的处理数组，我们常用 numpy 在这个包

首先导入 numpy 包，重命名一下

```
import numpy as np
```

那 numpy 如何创建数组呢？

可以用 Python 列表直接转换

首先创建一个 Python 列表

```
list_c = [1,2,3,4]
```

检查类型

```
type(list_c)
```

```
   >list
```

在使用 np.array()将 Python 列表转换为 numpy 的数组

```
my_array = np.array(list_c)
```

我们检查一下 my_array 的类型，

```
type(my_array)

   >numpy.ndarry
```

可以看到这个变量已经是 numpy 的数组了

打印一下

```
my_array

   >[1,2,3,4]
```

那 numpy 还有一些内置函数可以快速地创建数组

比如我们使用 np.arange()可以快速创建连续数字的数组

比如我创建一个 0~9 的一维数组

```
np.arange(0,10)

   >array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

jupyterlab 中使用 shift+tab 可以查看函数的帮助文档（查看一下，此刻输入 np.arange()，弹出对应函数帮助文档）

可以看到这个 arange()函数有 start、stop 和 step 参数，分别代表了起始值，终止值，以及步长

如果我希望创建 0~10 中连续偶数的数组，只需将步长设为 2（此刻输入 np.arange(0,10,2)）

```
np.arange(0,10,2)

   >array([0, 2, 4, 6, 8])
```

还可以用 np.ones 创建全是 1 的数组，（此时输入 np.ones()，弹出帮助说明）

比如我要创建一个大小为 3*3 的全 1 数组

```
np.ones(shape=(3,3))

   >array([[1., 1., 1.],

       [1., 1., 1.],

       [1., 1., 1.]])
```

再创建一个大小为 10*3 的全 1 数组

可以看到 10 是行数，3 是列数

```
np.ones(shape=(10,3))
```

```
   array([[1., 1., 1.],
          [1., 1., 1.],
          [1., 1., 1.],
          [1., 1., 1.],
          [1., 1., 1.],
          [1., 1., 1.],
          [1., 1., 1.],
          [1., 1., 1.],
          [1., 1., 1.],
          [1., 1., 1.]])
```

或者使用 np.zeros()全 0 数组

比如创建大小为 5*5 的全 0 数组

```
np.zeros(shape=(5,5))
```

```
   array([[0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0.],
          [0., 0., 0., 0., 0.]])
```

我们再演示一下 numpy 的一些其他方法

首先使用 np.randint 函数一些随机整数

```
arr = np.random.randint(0,100,10)
```

打印一下

```
arr
```

```
   >array([53, 86, 55, 17, 87, 12, 28, 95, 2, 44])
```

使用 max 获取最大值

```
arr.max()
    >95
```

再使用 argmax() 获取最大值的索引

```
arr.argmax()
    >7
```

使用 min 函数获取最小值

```
arr.min()
    >2
```

使用 argmin 获取最小值索引

```
arr.argmin()
    >8
```

使用 mean()方法获取取平均值

```
arr.mean()
    >47.9
```

如果要获取 humpy 数组的大小，使用 numpy.shape，

```
arr.shape
    >(10,)
```

也可以使用 reshape 函数转换数组的形状，比如我将 arr 转换成 5*2 的数组

```
arr.reshape((5,2))
    >array([[53, 86],
        [55, 17],
        [87, 12],
        [28, 95],
        [ 2, 44]])
```

可以看到变成了 5 行 2 列

再变形成 2 行 5 列

```
arr.reshape((2,5))
    >array([[53, 86, 55, 17, 87],
```

```
       [12, 28, 95,  2, 44]])
```

如果尝试变形为大小为 2*10 呢？看看效果

```
arr.reshape((2,10))

   >---------------------------------------------------------------------------

ValueError                                Traceback (most recent call last)

<ipython-input-34-abc2abe0ff88> in <module>

      1 # 尝试错误变形 2*10

----> 2 arr.reshape((2,10))

ValueError: cannot reshape array of size 10 into shape (2,10)
```

可以看到报错了，因为变形后的元素要求是 20 个，而我们原数组只有 10 个元素

那二维数组，我们在数学上也称为矩阵，我们再看一下 numpy 对矩阵的操作

首先创建一个 10*10 的矩阵

```
matrix = np.arange(0,100).reshape((10,10))

matrix

   >array([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9],

       [10, 11, 12, 13, 14, 15, 16, 17, 18, 19],

       [20, 21, 22, 23, 24, 25, 26, 27, 28, 29],

       [30, 31, 32, 33, 34, 35, 36, 37, 38, 39],

       [40, 41, 42, 43, 44, 45, 46, 47, 48, 49],

       [50, 51, 52, 53, 54, 55, 56, 57, 58, 59],

       [60, 61, 62, 63, 64, 65, 66, 67, 68, 69],

       [70, 71, 72, 73, 74, 75, 76, 77, 78, 79],

       [80, 81, 82, 83, 84, 85, 86, 87, 88, 89],

       [90, 91, 92, 93, 94, 95, 96, 97, 98, 99]])
```

查看一下大小

```
matrix.shape

   >(10, 10)
```

使用中括号中加索引方式，获取矩阵对应元素，比如我获取第 3 行第 5 列元素

```
matrix[2,4]
```

```
>24
```

再获取矩阵第 9 行第 7 列元素

```
matrix[8,6]
```

```
>86
```

如果要获取某一行所有元素，我们需要使用 numpy 的切片：

比如我要获取第 3 行所有元素，只需将第二个位置变成冒号:

```
matrix[2,:]
```

```
>array([20, 21, 22, 23, 24, 25, 26, 27, 28, 29])
```

类似的，比如我要获取第 6 列所有元素，只需将第一个位置变成冒号:

```
matrix[:,5]
```

```
>array([ 5, 15, 25, 35, 45, 55, 65, 75, 85, 95])
```

查看 shape

```
matrix[:,5].shape
```

```
>(10,)
```

用 reshape 恢复成原来的样子

```
matrix[:,5].reshape((10,1))
```

```
>array([[ 5],
       [15],
       [25],
       [35],
       [45],
       [55],
       [65],
       [75],
       [85],
       [95]])
```

我们再说 numpy 获取矩阵一个区域的用法

再看一下输出一下原来的 matrix

```
matrix
```

```
   >array([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14, 15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24, 25, 26, 27, 28, 29],
       [30, 31, 32, 33, 34, 35, 36, 37, 38, 39],
       [40, 41, 42, 43, 44, 45, 46, 47, 48, 49],
       [50, 51, 52, 53, 54, 55, 56, 57, 58, 59],
       [60, 61, 62, 63, 64, 65, 66, 67, 68, 69],
       [70, 71, 72, 73, 74, 75, 76, 77, 78, 79],
       [80, 81, 82, 83, 84, 85, 86, 87, 88, 89],
       [90, 91, 92, 93, 94, 95, 96, 97, 98, 99]])
```

比如我要获取第 1~3 行，第 2~4 列矩阵，我们可以用数字配合冒号的方式来获取

```
matrix[0:3,1:4]
```

```
   >array([[ 1,  2,  3],
       [11, 12, 13],
       [21, 22, 23]])
```

当然我们可以使用等号赋值语句，比如我将这些位置赋值 0

```
matrix[0:3,1:4] = 0
```

查看新的矩阵长啥样

```
matrix
```

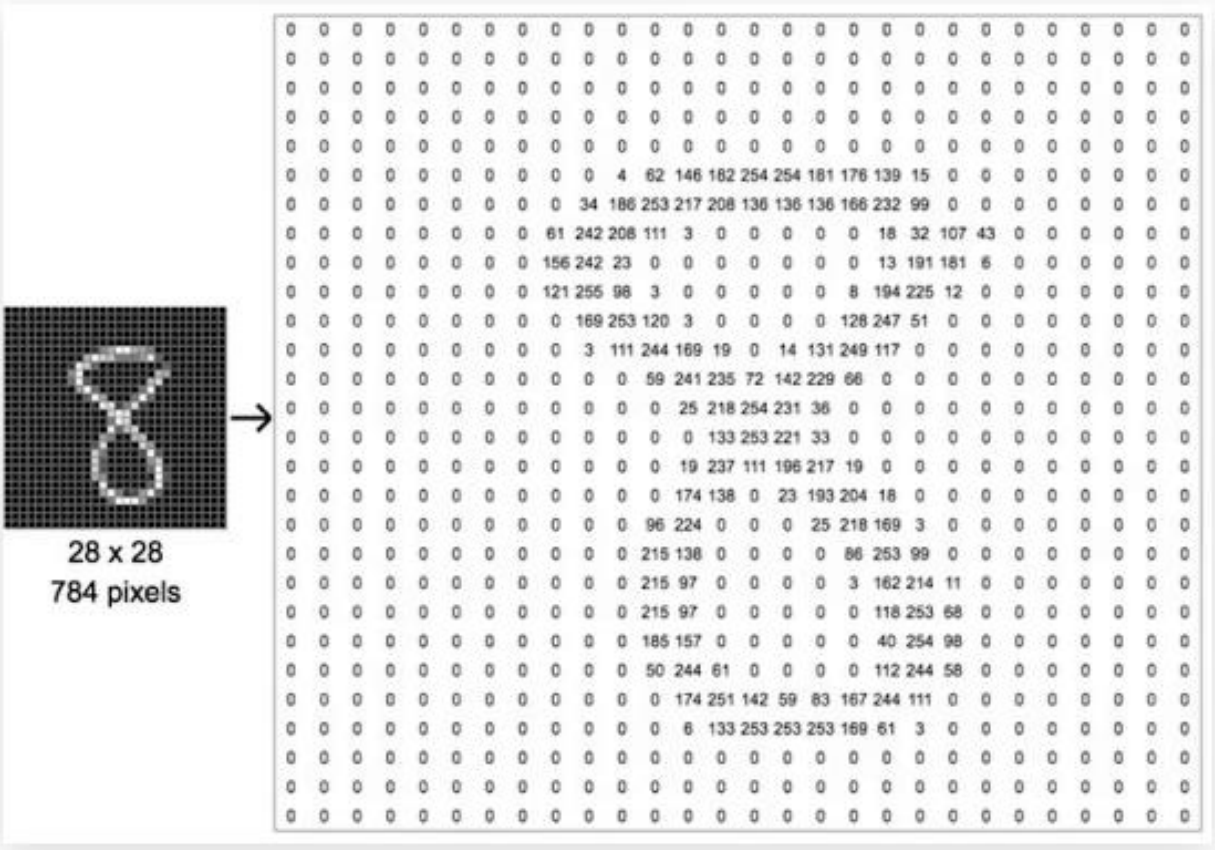
```
   >array([[ 0,  0,  0,  0,  4,  5,  6,  7,  8,  9],
       [10,  0,  0,  0, 14, 15, 16, 17, 18, 19],
       [20,  0,  0,  0, 24, 25, 26, 27, 28, 29],
       [30, 31, 32, 33, 34, 35, 36, 37, 38, 39],
       [40, 41, 42, 43, 44, 45, 46, 47, 48, 49],
       [50, 51, 52, 53, 54, 55, 56, 57, 58, 59],
       [60, 61, 62, 63, 64, 65, 66, 67, 68, 69],
```

```
    [70, 71, 72, 73, 74, 75, 76, 77, 78, 79],

    [80, 81, 82, 83, 84, 85, 86, 87, 88, 89],

    [90, 91, 92, 93, 94, 95, 96, 97, 98, 99]])
```

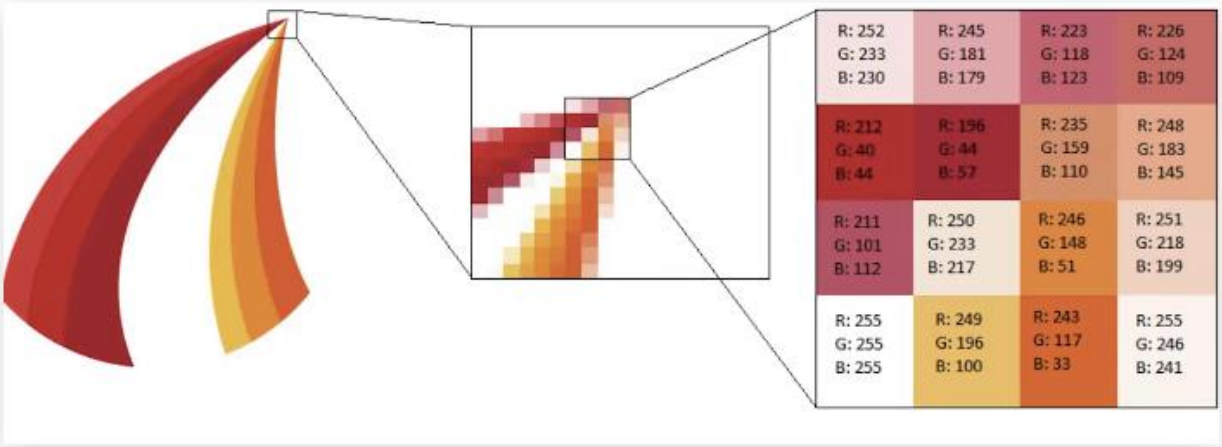
好，以上就是 numpy 对数组和矩阵的操作用法

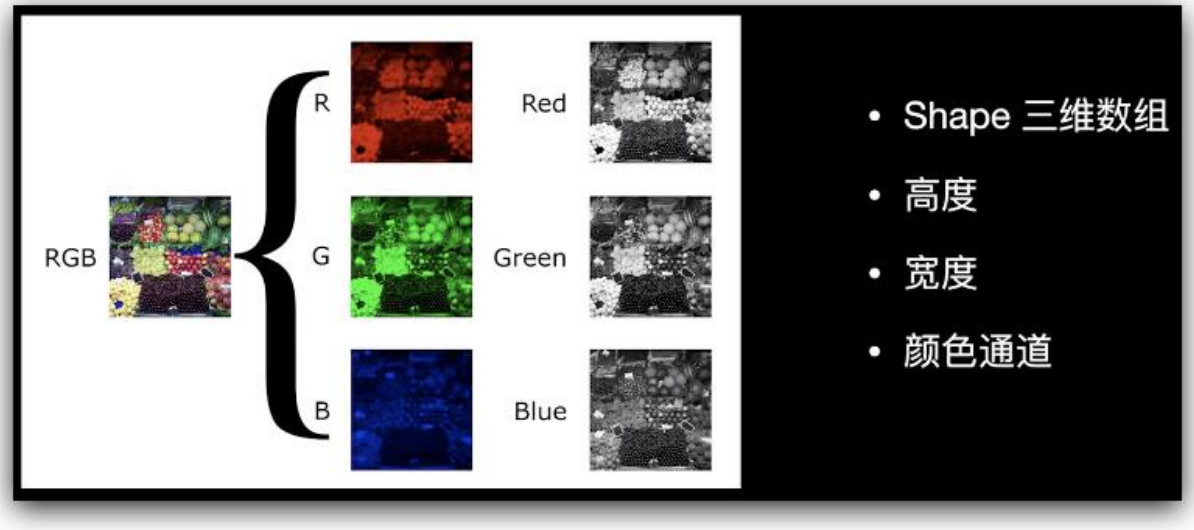
图像的本质

每个图片可以看成数组



灰度图片可以理解成是由暗部（0）和亮部（大于零的数字）组成，越接近 255 表示这个区域越亮。





彩色图片就是一个三维数组，使用 Numpy 读取出来的图片就是上述的三维数组。但是计算机并不知道哪一个通道是红色，它只知道有三个表达颜色的通道。所以我们需要标注通

道对应的颜色。每一个通道本质是等同于一张灰度图。

首先导入 numpy

```
import numpy as np
```

为了在 notebook 中显示图片，导入 matplotlib 库

```
import matplotlib.pyplot as plt
```

加这行在 Notebook 显示图像

```
%matplotlib inline
```

再使用一个 PIL 库，用于读取图像

```
from PIL import Image
```

我在 img 文件夹下放了一张图片（演示一下）

我们用 PIL 库读取图片，注意路径要正确

```
img = Image.open('./img/cat.jpg')
```

显示图像

```
img
```

可以看到这是一张彩色的猫咪图片

查看一下变量的类型

```
type(img)

   >PIL.JpegImagePlugin.JpegImageFile
```

可以看到这个不是 numpy 的数组格式，那 numpy 还不能处理它

首先我们需要将它转化为 numpy 数组，使用 numpy.asarray()函数

```
img_arr = np.asarray(img)
```

查看类型，

```
type(img_arr)

    >numpy.ndarray
```

发现已经变成了 numpy 数组，现在我们就可以用 numpy 来处理它了

我们查看大小

```
img_arr.shape
```

可以看到这张照片是 1880 像素宽，1253 像素高，3 个颜色通道

再使用 matplot 的 imshow()方法显示 Numpy 数组形式的图片

```
plt.imshow(img_arr)

    ><matplotlib.image.AxesImage at 0x7f9b86ed0c10>
```

可以看到横坐标和纵坐标显示了图片的长度是 1800 多，高度是 1200 多

我们继续对这个图片操作，先使用 numpy 的 copy 方法复制一份原图

```
img_arr_copy = img_arr.copy()
```

显示一下

```
plt.imshow(img_arr_copy)
```

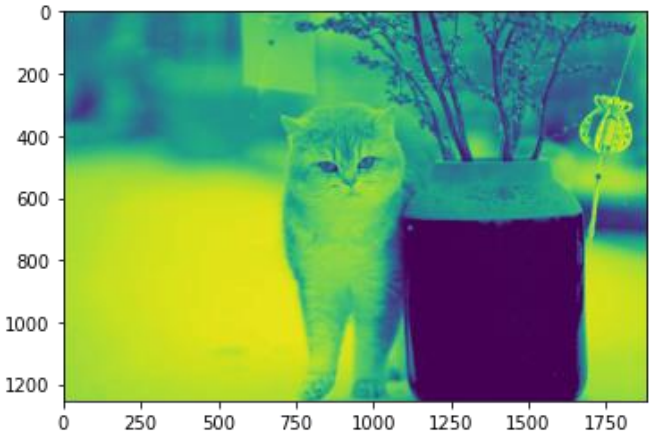
检查一下大小

```
img_arr_copy.shape
```

```
>(1253, 1880, 3)
```

首先使用 numpy 切片，将 R,G,B 三个颜色通道中的 R 红色通道显示出来

```
plt.imshow(img_arr_copy[:,:,0])
```



大家会发现这个颜色很奇怪，都是翠绿色，为什么会显示成这样呢？

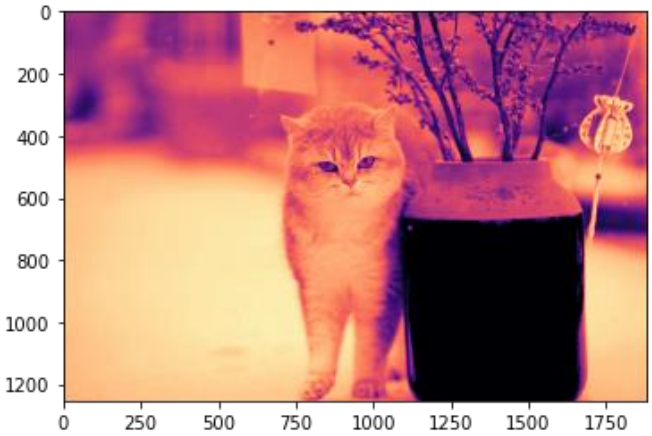
我们打开 matplot 的官网关于颜色表 colormap 的说明：

https://matplotlib.org/stable/gallery/color/colormap_reference.html

可以看到默认的颜色：是翠绿色（viridis ）。那这个颜色方便色盲观看的

我们也可以将 cmap 颜色设置成火山岩浆样式：magma

```
plt.imshow(img_arr_copy[:,:,0],cmap='magma')
```



我们打印一下红色 R 通道的数组

```
img_arr_copy[:,:,0]
```

```
>array([[111, 111, 111, ..., 193, 195, 197],
```

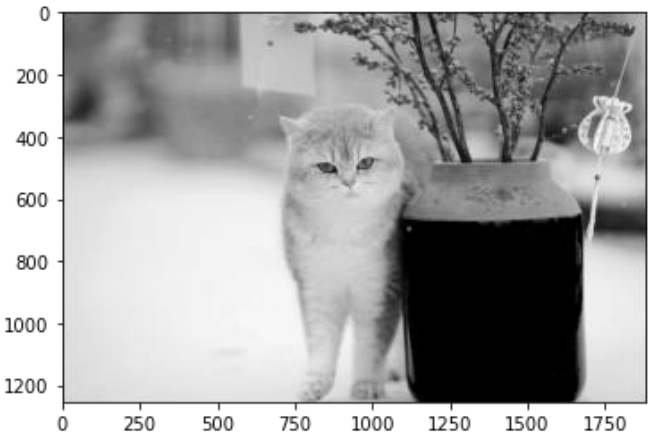
```
       [111, 111, 111, ..., 193, 195, 195],
       [111, 111, 111, ..., 193, 193, 195],
       ...,
       [213, 213, 213, ..., 215, 214, 214],
       [213, 213, 213, ..., 215, 214, 214],
       [213, 213, 213, ..., 215, 214, 214]], dtype=uint8)
```

好，我们知道，计算机是分不清到底哪一个通道是红色的，每一个颜色通道其实都是一个灰度图，我们首先将 cmap 颜色设置为 gray 灰度

看一下

```
plt.imshow(img_arr_copy[:,:,0],cmap='gray')
```

```
   ><matplotlib.image.AxesImage at 0x7f9b8f8038d0>
```



我们知道，红色通道中的 0 呢就是没有红色，代表纯黑色，而越接近 255 呢，就代表越红，255 就纯红色

那看这个灰度图，颜色越浅，表示这里越红

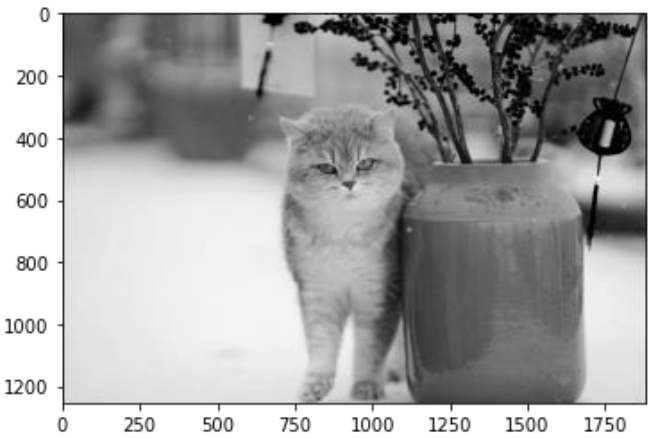
我们可以看一下红色通道的灰度图上颜色最浅的就是这个吊坠（鼠标指示）

那回到原来彩色图片，可以看到这个吊坠确实是最红的

类似的，我们将绿色通道也显示为灰度模式

```
plt.imshow(img_arr_copy[:,:,1],cmap='gray')
```

```
   ><matplotlib.image.AxesImage at 0x7f9b8f0d9450>
```



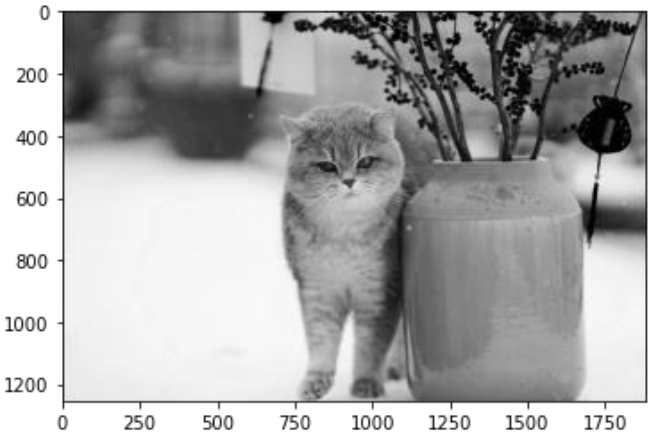
那 0 呢代表没有绿色或纯黑色，255 呢就代表纯绿色

可以看到，灰度图上颜色越浅，表示这里越绿

再看一下蓝色通道

```
plt.imshow(img_arr_copy[:,:,2],cmap='gray')
```

```
><matplotlib.image.AxesImage at 0x7f9b8f75df50>
```



0：没有蓝色或纯黑色，255 代表纯蓝色

灰度图颜色越浅，表示这里越蓝 ，可以看到这里相比前面红色、绿色的灰度图，这个花瓶是颜色比较浅的，代表颜色接近蓝色

当然，我们可以将某个颜色通道颜色全部设为 0，我们看一下效果，

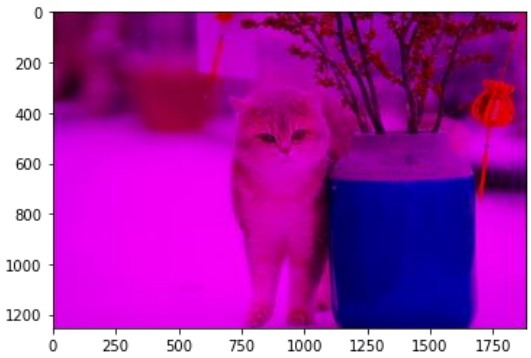
我这里把绿色通道全部变为 0

```
img_arr_copy[:,:,1] = 0
```

显示一下

```
plt.imshow(img_arr_copy)
```

```
><matplotlib.image.AxesImage at 0x7f9b8fc4a550>
```



那会发现画面颜色特别紫，这是因为只剩红色、蓝色通道，而（红+蓝就是紫色）

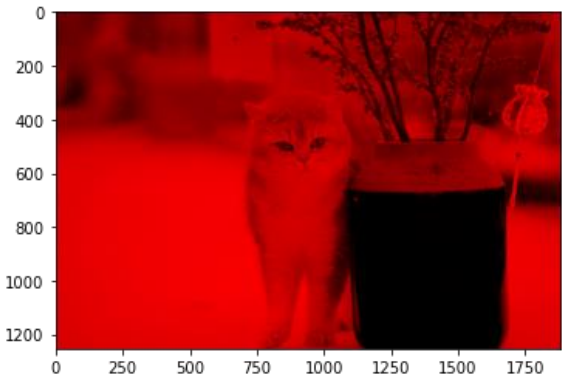
我们再将蓝色通道全部变为 0

```
img_arr_copy[:,:,2] = 0
```

显示一下

```
plt.imshow(img_arr_copy)
```

```
><matplotlib.image.AxesImage at 0x7f9b8bc841d0>
```



那只剩红色通道了，所以画面特别红，

那这个红色图片和前面的红色灰度图之所不同，是因为现在是我们三个通道一起合成在看

可以查看大小 shape，会发现大小仍然不变

```
img_arr_copy.shape
```

```
>(1253, 1880, 3)
```

而单独看一个通道的时候，大小会变化

```
img_arr_copy[:,:,0].shape
```

```
>(1253, 1880)
```