20377199 赵芮箐 第6周作业

卷积神经网络（CNN）在图像分类、目标检测等领域取得了远超传统模型的效果，使得大量图片数据的应用成为可能。该模型中的卷积块，灵感一定程度上来自于早期图像处理领域的“滤波器（filter）”，其用于提取图像中的某类特定特征或对图像做某些特定变换。本次作业要求基于类继承，体会多种滤波器的使用效果与实际应用，并初步掌握python对图片的简单处理。

1. 具体要求见附件。

2. 要求用类及其继承实现所有功能要求，并体现会多态。

3. 安装并学习pillow

任务一：实现基类Filter

```
class Filter:
    def __init__(self, path, *plist):
        """
        两个数据属性
        image:待处理的图片实例，即PIL库的Image实例
        plist:参数列表，用以存储可能使用参数的滤波器的参数
        """
        self._image = Image.open(path)
        self._plist = plist

    def filter(self):
        """
        能够对Image实例的特定处理，在子类中具体实现
        """
        pass
```

任务二：实现Filter 类的多个子类，对filter()方法进行实现

```
class FIND_EDGES_Filter(Filter):
    '''
    提取边缘
    '''
    def __init__(self, path, *plist):
        super().__init__(path, plist)

    def filter(self):
        image_filter = self._image.filter(ImageFilter.FIND_EDGES)
        return image_filter

class SHARPEN_Filter(Filter):
    '''
    锐化
    '''
    def __init__(self, path, *plist):
        super().__init__(path, plist)

    def filter(self):
        image_filter = self._image.filter(ImageFilter.SHARPEN)
        return image_filter
```

```
class BLUR_Filter(Filter):
    '''
    模糊
    '''
    def __init__(self, path, *plist):
        super().__init__(path, plist)

    def filter(self):
        image_filter = self._image.filter(ImageFilter.BLUR)
        return image_filter

class RESIZE_Filter(Filter):
    '''
    大小调整
    '''
    def __init__(self, path, *plist):
        super().__init__(path, plist)

    def filter(self):
        w,h = self._image.size
        wrate = self._plist[0]
        hrate = self._plist[1]
        image_filter = self._image.resize((int(w*wrate),int(h*hrate)))
        return image_filter
```

任务三：实现类ImageShop，实现对图片的加载、处理、展示及输出

```
class ImageShop:
    def __init__(self, formation, filepath):
        """
        三个数据属性
        formation:图片格式
        filepath:图片文件（应该支持目录）
        imglist:存储待处理图片列表
        imgpro:储存处理后的图片列表
        """
        self._formation = formation
        self._filepath = filepath
        self._imglist = []
        self._imgpro = []

    def load_images(self):
        """
        从路径加载特定格式的图片
        """
        dirs = os.listdir(self._filepath)                    # 获取指定路径下的文件
        for path in dirs:
            if os.path.splitext(path)[1] == self._formation:# 加载目录中的所有特定格
式图片
                img = Filter(self._filepath + '/' + path)
                self._imglist.append(img)                    # 以Fliter类储存
```

```
    def __batch_ps(self, Filter, *plist):
        """
        处理图片的内部方法，利用某个过滤器(Filter)对所有图片进行处理
        """
        for i in range(len(self._imgpro)):
            img = self._imgpro[i]
            img._plist = plis                                   # 更新参数属性
            img_pro = Filter.filter(img)                        # 处理图片
            img._image = img_pro                                # 更新处理的图片

    def batch_ps(self, *args):
        """
        批量处理图片的对外公开方法，可以同时进行若干操作
        args:不定长操作参数，参数按一种特定格式的tuple输入，比如（操作名，参数）
        """
        self._imgpro = self._imglist                            # 初始化处理后的图片列表
        for op in args:
            if op == 'find_edges':
                self.__batch_ps(FIND_EDGES_Filter)
            elif op == 'sharpen':
                self.__batch_ps(SHARPEN_Filter)
            elif op == 'blur':
                self.__batch_ps(BLUR_Filter)
            elif op[0] == 'resize':
                self.__batch_ps(RESIZE_Filter, *op[1:])
            else:
                print("Error, do not find operation named {}.".format(op[0]))
                break

    def display(self, row=5, col=5, max_num=100):
        """
        处理效果显示
        row&col:图片呈现可能需要行和列
        max_num:最多显示多少张
        """
        if len(self._imgpro) > max_num:                         # 控制最大显示图片数
            self._imgpro = self._imgpro[:max_num]
        for num in range(0,len(self._imgpro), row*col):
            for i in range(row*col):                            # 控制每张子图展示图片数
量
                if num + i < len(self._imgpro):
                    img = self._imgpro[num+i]
                    plt.subplot(row, col, i+1)
                    plt.imshow(img._image)
            plt.show()

    def save(self, path, formation = '.png'):
        """
        处理结果输出
        path:输出路径
        formation:输出格式，默认为png
        """
        for i in range(len(self._imgpro)):
            img = self._imgpro[i]
            img._image.save(path+str(i+1)+formation)
```

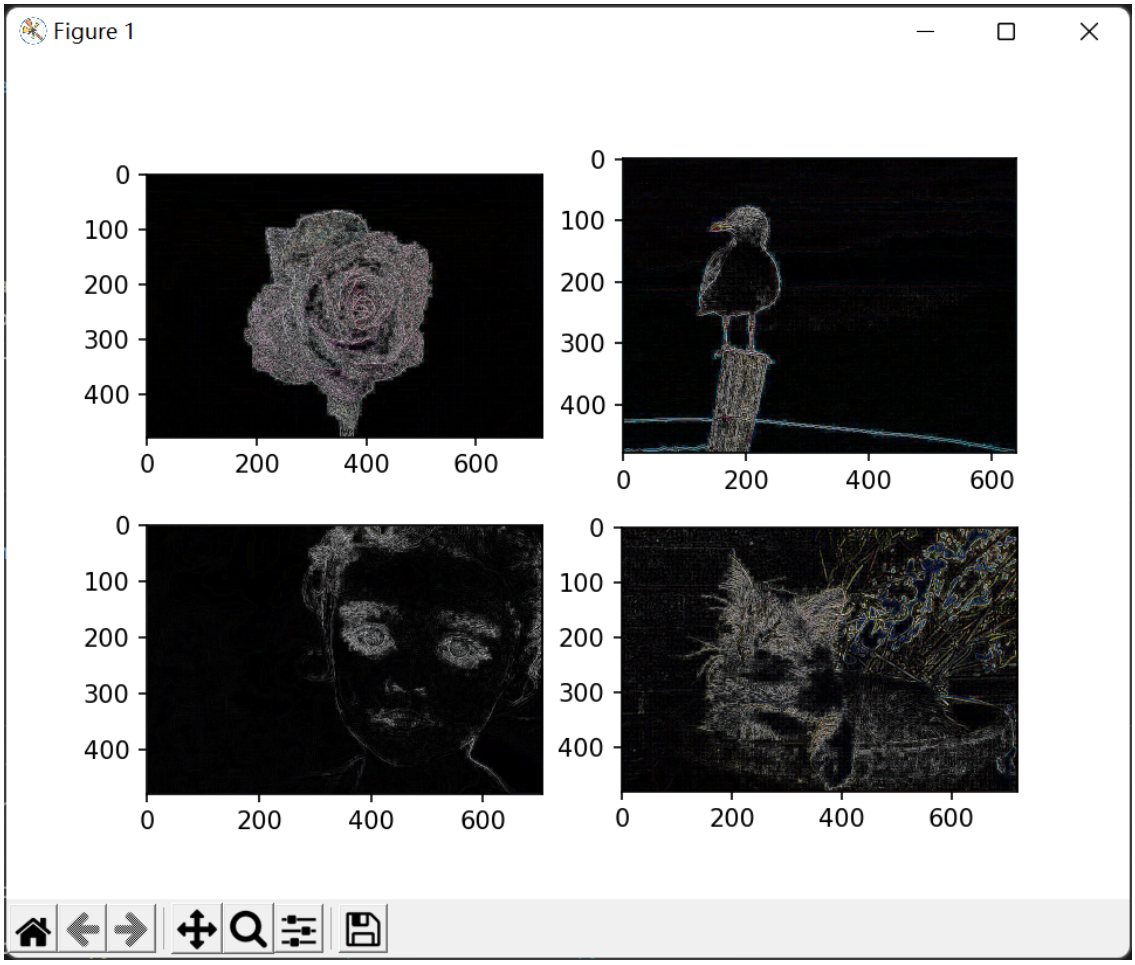
任务四：实现测试类TestImageShop，实现对ImageShop类的测试

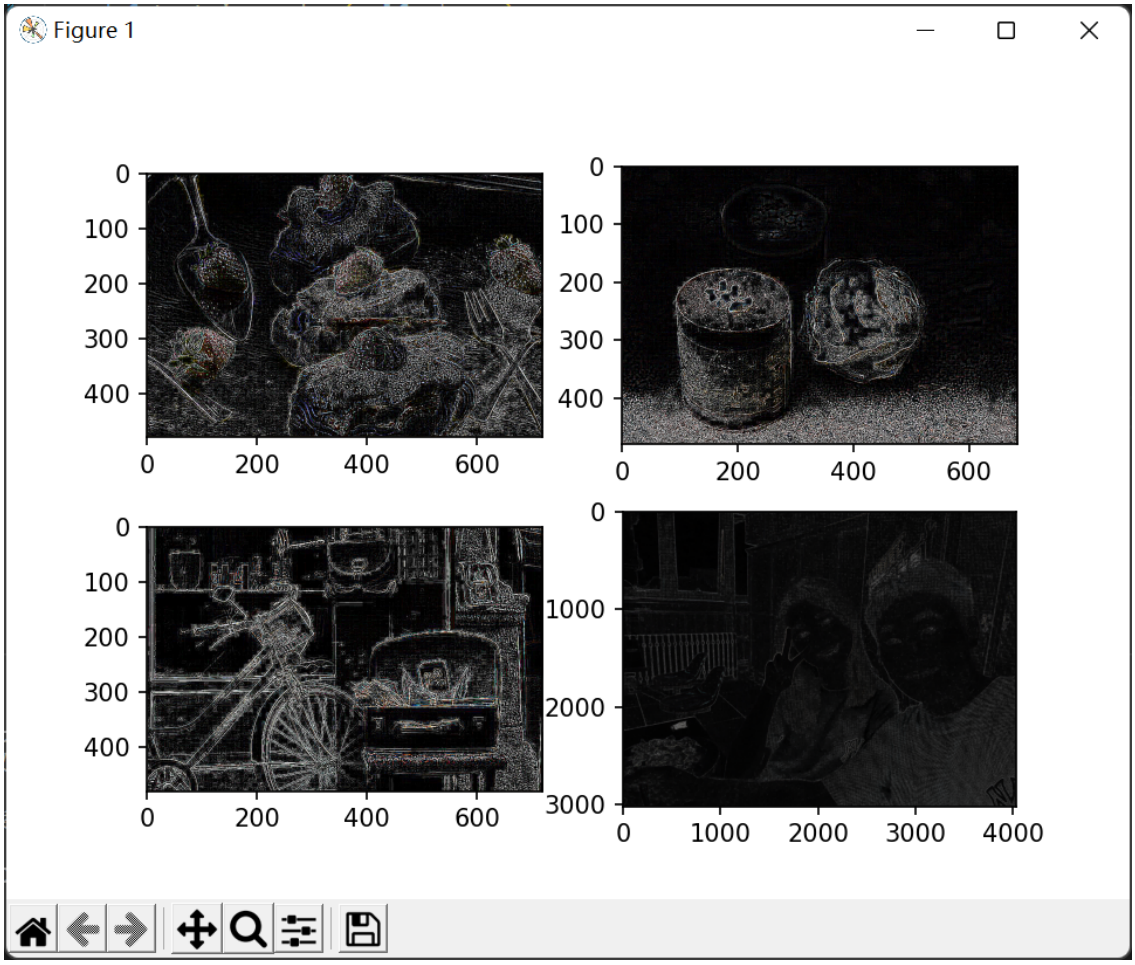
```
class TestImageShop:
    def __init__(self, formation, filepath):
        self._test = ImageShop(formation, filepath)

    def text_imageshop(self, *args):
        self._test.load_images()
        self._test.batch_ps(args)
        self._test.display()
        self._test.save('week6/photo_fliter/')
```

结果展示：

- display( ) 效果展示

  以8张照片为例，参数设置为2x2



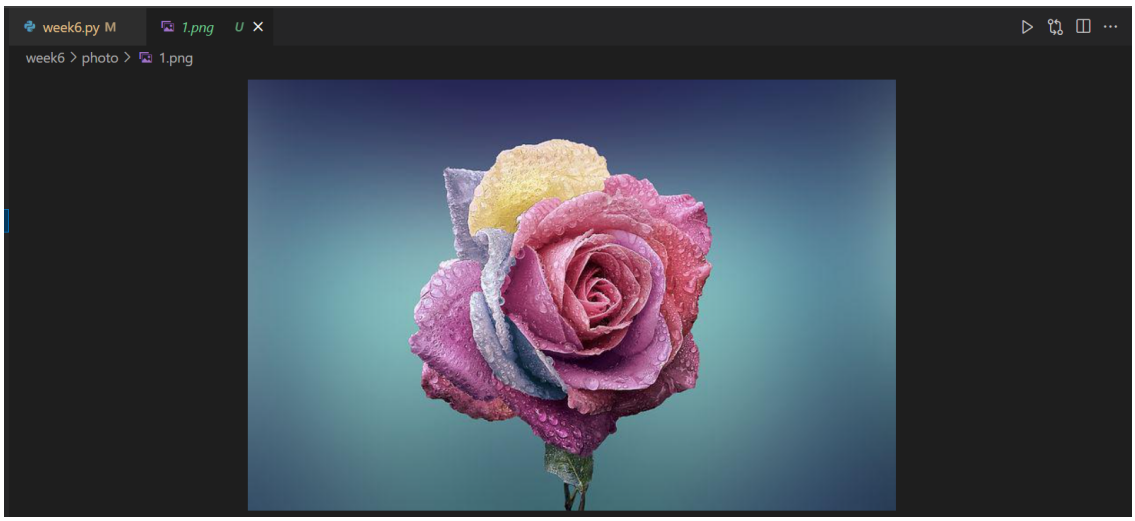


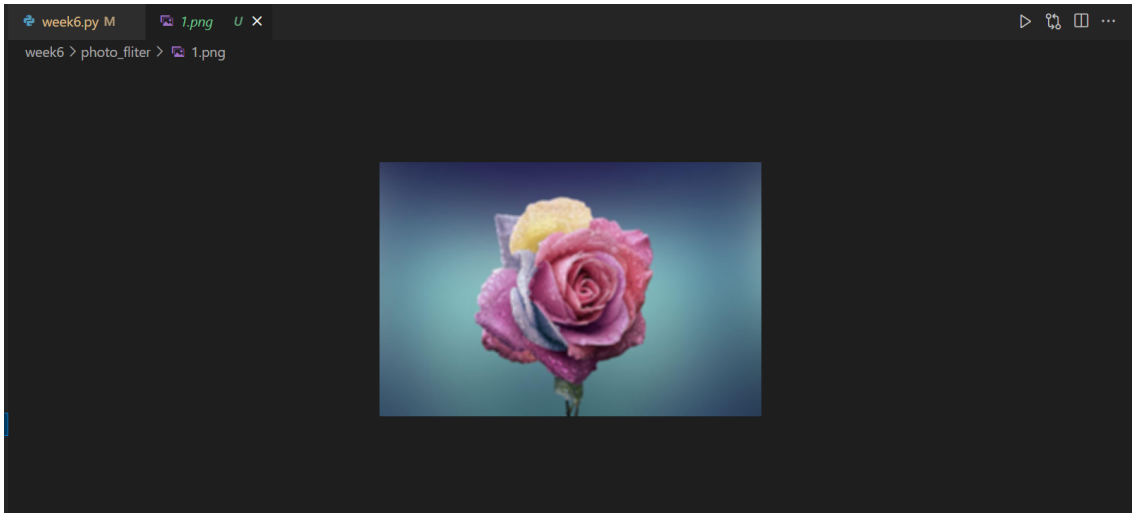
> 最后一张照片是我和lpj合照，真的看到结果心碎了

- filter( ) 效果展示

  锐化+提取边缘操作如上面所展示（实测先锐化再提取边缘效果会更好）

  下展示模糊+调整大小操作：





思考：

（附加）观察一些经过过滤后图片的变化，思考这些处理对图片本身的一些相关的计算，如图片的相似性等有无影响，并进行简单的实验验证。另外，这些预处理本身对下游的机器学习模型会带来哪些好处?

- filter操作对图片本身的计算有无影响

  由于成像系统、传输介质和记录设备等的不完善，数字图像在其形成、传输记录过程中往往会受到多种噪声的污染。另外，在图像处理的某些环节当输入的像对象并不如预想时也会在结果图像中引入噪声。

  图像往往由多块组成，各块内像素相似且过渡缓慢，块与块相邻部分称作边缘(Edge)。而噪声在图像上常表现为一引起较强视觉效果的孤立像素点或像素块。图像滤波希望实现的是图像块内平滑降噪、去细节，并最大程度保留图像边缘。

  所以滤波的实际上对原图像的每个像素周围一定范围内的像素进行运算，如对各像素灰度值如乘一个权值然后求和，或者求一个像素周围3x3范围内最大值、最小值、中值、均值等。由于相邻区域内的像素是相似的，所以这样的计算对图片本身会有改变，但影响很小。

  例如图片间的相似度值肯定有变化，但改变甚微，以下为测试：

```
from PIL import Image
from numpy import average, dot, linalg

# 对图片进行统一化处理
def get_thum(image, size=(64, 64), greyscale=False):
    # 利用image对图像大小重新设置, Image.ANTIALIAS为高质量的
    image = image.resize(size, Image.ANTIALIAS)
    if greyscale:
        # 将图片转换为L模式，其为灰度图，其每个像素用8个bit表示
        image = image.convert('L')
    return image

# 计算图片的余弦距离
def image_similarity_vectors_via_numpy(image1, image2):
    image1 = get_thum(image1)
    image2 = get_thum(image2)
    images = [image1, image2]
    vectors = []
    norms = []
    for image in images:
```

```
            vector = []
            for pixel_tuple in image.getdata():
                vector.append(average(pixel_tuple))
            vectors.append(vector)
            # linalg=linear（线性）+algebra（代数），norm则表示范数
            # 求图片的范数
            norms.append(linalg.norm(vector, 2))
        a, b = vectors
        a_norm, b_norm = norms
        # dot返回的是点积，对二维数组（矩阵）进行计算
        res = dot(a / a_norm, b / b_norm)
        return res

image1 = Image.open('week6/photo/1.png')
#image2 = Image.open('week6/photo/1.png')
image2 = Image.open('week6/photo_filter/1.png')
cosin = image_similarity_vectors_via_numpy(image1, image2)
print('图片余弦相似度', cosin)
```

同样图片的相似度：

进行了filter操作后两张图片的相似度：

看得出相似度有变化，但变化很小，几乎无影响，两张图片仍旧相似度很高

- 图像预处理对下游机器学习模型带来的好处

  图像预处理的主要目的是消除图像中无关的信息，恢复有用的真实信息，增强有关信息的可检测性和最大限度地简化数据（就像图像滤波能实现尽量保留原图像细节特征的条件下，对该图像的噪声进行抑制）从而提高特征抽取、图像分割、匹配和识别的可靠性。

代码：

https://github.com/rachhhhing/mp2022_python/blob/master/week6/week6.py

Ref:

- PIL：https://zhuanlan.zhihu.com/p/360184046
- ImageFilter：https://pillow.readthedocs.io/en/stable/reference/ImageFilter.html
  https://blog.csdn.net/alice_tl/article/details/89290804
- 图片相似度计算：https://zhuanlan.zhihu.com/p/93893211
- torchvision.transforms：https://blog.csdn.net/zandaoguang/article/details/126736752
- 深度卷积网络：https://blog.csdn.net/chenfang0529/article/details/108102384
  https://blog.csdn.net/wutongyutt/article/details/80496860