**实验3 数字图像分割与边缘检测实验**

**作业一：使用一阶和二阶检测算子（导数）进行图像的边缘检测。**

问题1（编写程序实现一阶Sobel算子，进行图像的边缘提取；）&问题2（编写程序实现一阶Prewitt算子，进行图像的边缘提取；）&问题3（编写程序实现一阶Roberts算子，进行图像的边缘提取。)

class Dataset():

    def \_\_init\_\_(self, path):

        self.path = path

        self.image\_list = [x for x in listdir(path) if is\_image\_file(x)]

        self.image\_list = sorted(self.image\_list)

        self.sobel\_kernal\_x = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 2, 1]])

        self.sobel\_kernal\_y = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])

        self.prewitt\_kernal\_x = np.array([[-1, 0, 1], [-1, 0, 1], [-1, 0, 1]])

        self.prewitt\_kernal\_y = np.array([[-1, -1, -1], [0, 0, 0], [1, 1, 1]])

        self.roberts\_kernal\_x = np.array([[0, -1], [1, 0]])

        self.roberts\_kernal\_y = np.array([[-1, 0], [0, 1]])

        self.laplacian\_kernal = np.array([[0, 1, 0], [1, -4, 1], [0, 1, 0]])

    def edge\_detection(self, index, modal="sobel", dir="x"):

        try:

            image\_path = os.path.join(self.path, self.image\_list[index - 1])

        except:

            print("ERROR！ 并不包含你想要进行RGB处理的这张图片", index)

        else:

            image\_name = re.findall(r'(.+?)\.', self.image\_list[index - 1])

            c\_image = default\_loader(image\_path)

            image = gray\_loader(image\_path)

            # [b, g, r] = cv2.split(image)

            if (modal == "sobel"):

                if (dir == "x"):

                    kernal = self.sobel\_kernal\_x

                elif (dir == "y"):

                    kernal = self.sobel\_kernal\_y

            elif (modal == "prewitt"):

                if (dir == "x"):

                    kernal = self.prewitt\_kernal\_x

                elif (dir == "y"):

                    kernal = self.prewitt\_kernal\_y

            elif (modal == "roberts"):

                if (dir == "x"):

                    kernal = self.roberts\_kernal\_x

                elif (dir == "y"):

                    kernal = self.roberts\_kernal\_y

            elif (modal == "laplacian"):

                kernal = self.laplacian\_kernal

            # b\_lap = signal.convolve2d(b, kernal, boundary="symm", mode='same')

            # g\_lap = signal.convolve2d(g, kernal, boundary="symm", mode='same')

            # r\_lap = signal.convolve2d(r, kernal, boundary="symm", mode='same')

            # b\_lap = np.absolute(b\_lap)

            # g\_lap = np.absolute(g\_lap)

            # r\_lap = np.absolute(r\_lap)

            lap = signal.convolve2d(image,

                                    kernal,

                                    boundary="symm",

                                    mode='same')

            lap = np.absolute(lap)

            # lap = cv2.merge((b\_lap, g\_lap, r\_lap))

            lap = np.array(lap, dtype=np.uint8)

            if (modal == "laplacian"):

                cv2.imwrite(

                    str(\*image\_name) + " " + str(modal.title()) + ".png", lap)

            else:

                cv2.imwrite(

                    str(\*image\_name) + " " + str(dir) + " " +

                    str(modal.title()) + ".png", lap)

**作业二： Hough线检测**

问题1：对作业一中边缘检测的结果，进行Hough线检测。

def hough\_line(self, index):

    try:

        image\_path = os.path.join(self.path, self.image\_list[index - 1])

    except:

        print("ERROR！ 并不包含你想要进行RGB处理的这张图片", index)

    else:

        image\_name = re.findall(r'(.+?)\.', self.image\_list[index - 1])

        c\_image = default\_loader(image\_path)

        image = gray\_loader(image\_path)

        edges = cv2.Canny(image, 50, 100, apertureSize=3)

        minLineLength = 100

        maxLineGap = 10

        lines = cv2.HoughLinesP(edges, 1, np.pi / 180, 100, minLineLength,

                                maxLineGap)

        for line in lines:

            for x1, y1, x2, y2 in line:

                # print(x1, y1, x2, y2)

                cv2.line(c\_image, (x1, y1), (x2, y2), (0, 255, 255), 2)

        cv2.imwrite(str(\*image\_name) + " Hough.png", c\_image)

问题2：调节参数，提取较长的边界。def edge\_detection(self, index, modal="sobel", dir="x"):

    try:

        image\_path = os.path.join(self.path, self.image\_list[index - 1])

    except:

        print("ERROR！ 并不包含你想要进行RGB处理的这张图片", index)

    else:

        image\_name = re.findall(r'(.+?)\.', self.image\_list[index - 1])

        c\_image = default\_loader(image\_path)

        image = gray\_loader(image\_path)

        # [b, g, r] = cv2.split(image)

        if (modal == "sobel"):

            if (dir == "x"):

                kernal = self.sobel\_kernal\_x

            elif (dir == "y"):

                kernal = self.sobel\_kernal\_y

        elif (modal == "prewitt"):

            if (dir == "x"):

                kernal = self.prewitt\_kernal\_x

            elif (dir == "y"):

                kernal = self.prewitt\_kernal\_y

        elif (modal == "roberts"):

            if (dir == "x"):

                kernal = self.roberts\_kernal\_x

            elif (dir == "y"):

                kernal = self.roberts\_kernal\_y

        elif (modal == "laplacian"):

            kernal = self.laplacian\_kernal

        # b\_lap = signal.convolve2d(b, kernal, boundary="symm", mode='same')

        # g\_lap = signal.convolve2d(g, kernal, boundary="symm", mode='same')

        # r\_lap = signal.convolve2d(r, kernal, boundary="symm", mode='same')

        # b\_lap = np.absolute(b\_lap)

        # g\_lap = np.absolute(g\_lap)

        # r\_lap = np.absolute(r\_lap)

        lap = signal.convolve2d(image,

                                kernal,

                                boundary="symm",

                                mode='same')

        lap = np.absolute(lap)

        # lap = cv2.merge((b\_lap, g\_lap, r\_lap))

        lap = np.array(lap, dtype=np.uint8)

        if (modal == "laplacian"):

            cv2.imwrite(

                str(\*image\_name) + " " + str(modal.title()) + ".png", lap)

        else:

            cv2.imwrite(

                str(\*image\_name) + " " + str(dir) + " " +

                str(modal.title()) + ".png", lap)

**作业三：采用阈值处理方法进行图像分割**

问题1：参考相关文献，编写程序实现Otsu自动阈值法；

def otsu\_adaptive(self, index):

    try:

        image\_path = os.path.join(self.path, self.image\_list[index - 1])

    except:

        print("ERROR！ 并不包含你想要进行RGB处理的这张图片", index)

    else:

        image\_name = re.findall(r'(.+?)\.', self.image\_list[index - 1])

        c\_image = default\_loader(image\_path)

        image = gray\_loader(image\_path)

        otsu = cv2.adaptiveThreshold(image, 255,

                                        cv2.ADAPTIVE\_THRESH\_GAUSSIAN\_C,

                                        cv2.THRESH\_BINARY, 11, 2)

        cv2.imwrite(str(\*image\_name) + " Otsu.png", otsu)

问题2：实现直方图阈值法，具体方法为采用灰度直方图求双峰或多峰，选择两峰之间的谷底作为阈值，将图像转换为2值图像。

def otsu\_histogram(self, index):

    try:

        image\_path = os.path.join(self.path, self.image\_list[index - 1])

    except:

        print("ERROR！ 并不包含你想要进行RGB处理的这张图片", index)

    else:

        image\_name = re.findall(r'(.+?)\.', self.image\_list[index - 1])

        c\_image = default\_loader(image\_path)

        image = gray\_loader(image\_path)

        plt.hist(image.ravel(), 256, color="black")

        plt.title("Otsu Gray Scale Histogram", fontsize=24)

        plt.xlim([0, 256])

        plt.tick\_params(labelsize=14)

        plt.savefig(str(\*image\_name) + " Otsu Gray Scale Histogram.png")

        # 使用 Otsu 算法自动求解双峰中的谷底

        hist = cv2.calcHist([image], [0], None, [256], [0, 256])

        hist\_norm = hist.ravel() / hist.max()

        Q = hist\_norm.cumsum()

        bins = np.arange(256)

        fn\_min = np.inf

        thresh = -1

        for i in range(1, 256):

            p1, p2 = np.hsplit(hist\_norm, [i])  # probabilities

            q1, q2 = Q[i], Q[255] - Q[i]  # cum sum of classes

            b1, b2 = np.hsplit(bins, [i])  # weights

            # finding means and variances

            m1, m2 = np.sum(p1 \* b1) / (q1 + 1), np.sum(p2 \* b2) / (q2 + 1)

            v1, v2 = np.sum(((b1 - m1)\*\*2) \* p1) / (q1 + 1), np.sum(

                ((b2 - m2)\*\*2) \* p2) / (q2 + 1)

            # calculates the minimization function

            fn = v1 \* q1 + v2 \* q2

            if fn < fn\_min:

                fn\_min = fn

                thresh = i

        # find otsu's threshold value with OpenCV function

        ret, otsu = cv2.threshold(image, 0, 255,

                                  cv2.THRESH\_BINARY + cv2.THRESH\_OTSU)

        cv2.imwrite(

            str(\*image\_name) + " ret " + str(int(ret)) + " Otsu.png", otsu)