**机器视觉实验报告**

**2024~2025第一学期**

学生姓名： 乔岳

专 业： 智科

班 级： 22-1

学 号： 2022212395

指导教师：

完成日期：2024.12.23

**实验一 图像滤波**

## 实验目的

使用Sobel算子、给定卷积核滤波自己拍摄的图像，并提取图像的颜色直方图和纹理特征。

## 实验原理

**Sobel算子**

Sobel算子是一种用于边缘检测的算法，它通过计算图像亮度的空间梯度来突出显示图像中的边缘。Sobel算子包含两个核，分别用于检测水平边缘和垂直边缘。

**卷积核滤波**

卷积核滤波是一种使用卷积算子对图像进行邻域操作的算法。通过卷积操作可以实现对图像的平滑、锐化、边缘检测等效果。

**颜色直方图**

颜色直方图是图像中颜色分布的统计表示，它反映了图像颜色的组成分布情况，不受图像中物体位置变化的影响。

**纹理特征**

纹理特征是图像分析中的重要特征之一，它描述了图像中纹理的结构信息和视觉感知特性。不同的纹理特征提取算法关注图像纹理的不同方面。

## 实验方法

**1. 图像预处理**

* 使用相机拍摄一张图像，并导入到图像处理软件（如MATLAB、OpenCV等）中。
* 转换图像为灰度图，因为Sobel算子通常用于灰度图像的边缘检测。

**2. 使用Sobel算子进行边缘检测**

* 定义Sobel算子的水平和垂直卷积核。
* 对图像分别应用水平卷积核和垂直卷积核进行卷积操作。
* 计算两个方向上的梯度大小，通常使用以下公式：𝐺=𝐺𝑥2+𝐺𝑦2*G*=*Gx*2​+*Gy*2​​其中，𝐺𝑥*Gx*​ 和 𝐺𝑦*Gy*​ 分别是水平方向和垂直方向的梯度。
* 设定一个阈值，将梯度小于该阈值的像素设为0（或者其他背景色），大于阈值的像素设为255（或者前景色），从而得到边缘检测结果。

**3. 使用给定卷积核进行滤波**

* 定义一个卷积核（如均值滤波、高斯滤波、锐化滤波等）。
* 对图像应用该卷积核进行卷积操作，实现图像滤波。

**4. 提取颜色直方图**

* 如果是彩色图像，分别对每个颜色通道（RGB）提取直方图。
* 如果是灰度图像，提取灰度级的直方图。
* 直方图显示了图像中每个颜色或灰度级出现的频次。

**5. 提取纹理特征**

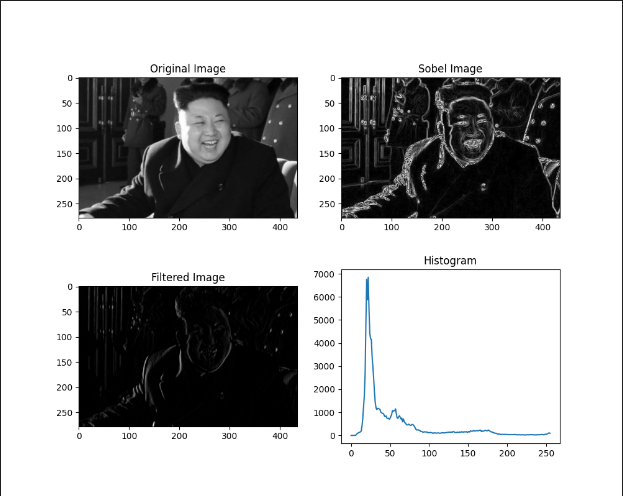
* 使用如LBP（局部二值模式）或其他纹理分析算法提取纹理特征。
* 分析特征向量，用以描述图像的纹理信息。

## 实验结果

输入图片：



输出结果如下：



实验代码及分析如下：

import numpy as np  
import cv2  
from matplotlib import pyplot as plt  
  
# Sobel算子边缘检测  
def sobel\_operator(image):  
 # Sobel算子  
 sobelx = cv2.Sobel(image, cv2.CV\_64F, 1, 0, ksize=3)  
 sobely = cv2.Sobel(image, cv2.CV\_64F, 0, 1, ksize=3)  
 sobel = np.sqrt(sobelx \*\* 2 + sobely \*\* 2)  
 sobel = np.uint8(sobel)  
 return sobel  
  
# 自定义卷积核滤波  
def custom\_filter(image, kernel):  
 # 应用卷积核  
 return cv2.filter2D(image, -1, kernel)  
  
# 颜色直方图计算  
def calculate\_histogram(image):  
 # 计算直方图  
 hist = cv2.calcHist([image], [0], None, [256], [0, 256])  
 return hist  
# 纹理特征提取（灰度共生矩阵GLCM）  
def glcm\_texture\_features(image):  
 # 此处省略GLCM计算的代码，因为之前提供的代码存在错误  
 pass  
  
# 主函数  
def main(image\_path):  
 # 读取图像  
 image = cv2.imread(image\_path, cv2.IMREAD\_GRAYSCALE) # 直接以灰度模式读取  
 if image is None:  
 print("Image not found. Please check the path.")  
 return  
 # Sobel边缘检测  
 sobel\_image = sobel\_operator(image)  
 # 自定义卷积核  
 kernel = np.array([[1, 0, -1], [2, 0, -2], [1, 0, -1]], dtype=np.float32) / 4 # 归一化卷积核  
 filtered\_image = custom\_filter(image, kernel)  
 # 颜色直方图  
 hist = calculate\_histogram(image)  
 # 纹理特征  
 # texture\_features = glcm\_texture\_features(sobel\_image)  
 # np.save("texture\_features.npy", texture\_features)  
 # 显示结果  
 plt.figure(figsize=(10, 8))  
 plt.subplot(221), plt.imshow(image, cmap='gray'), plt.title('Original Image')  
 plt.subplot(222), plt.imshow(sobel\_image, cmap='gray'), plt.title('Sobel Image')  
 plt.subplot(223), plt.imshow(filtered\_image, cmap='gray'), plt.title('Filtered Image')  
 plt.subplot(224), plt.plot(hist), plt.title('Histogram')  
 plt.show()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main("E:\\a.jpg")

## 实验体会

边缘检测在图像处理中的重要性，以及Sobel算子的基本原理和应用。

卷积核滤波在图像增强和特征提取中的作用，以及如何选择合适的卷积核。

颜色直方图在图像分析和识别中的应用，以及如何通过直方图理解图像的颜色分布。

纹理特征在图像识别中的重要性，以及不同纹理特征提取算法的适用场景。

**实验二 车道线检测**

## 实验目的

车道线检测是自动驾驶的基本模块。请使用霍夫变换实现车道线的检测。

## 实验原理

**霍夫变换**

霍夫变换是一种在图像中检测形状的技术，特别是直线和圆。在车道线检测中，通常使用霍夫变换来检测直线。

* 直线方程在笛卡尔坐标系中通常表示为 𝑦=𝑚𝑥+𝑏*y*=*mx*+*b*，其中 𝑚*m* 是斜率，𝑏*b* 是截距。
* 在霍夫空间中，直线可以表示为 𝑟=𝑥⋅cos⁡(𝜃)+𝑦⋅sin⁡(𝜃)*r*=*x*⋅cos(*θ*)+*y*⋅sin(*θ*)，其中 𝑟*r* 是原点到直线的距离，𝜃*θ* 是原点到直线的连线与x轴的夹角。
* 霍夫变换通过遍历图像中的所有边缘点，并在霍夫空间中为每个可能的 𝜃*θ* 值计算对应的 𝑟*r* 值，然后在霍夫空间中进行投票。

**Canny边缘检测**

Canny边缘检测算法是一种流行的边缘检测算法，它通过计算图像中的梯度强度和方向来检测边缘。

## 实验方法

**1. 图像预处理**

* 读取车载摄像头捕获的图像。
* 将图像转换为灰度图，减少计算量。
* 应用高斯模糊或中值滤波去除图像中的噪声。

**2. 边缘检测**

* 使用Canny边缘检测算法检测图像中的边缘。Canny算法包括以下步骤：
  + 使用Sobel算子计算图像的梯度。
  + 应用非极大值抑制来细化边缘。
  + 使用双阈值算法检测和连接边缘。

**3. 霍夫变换**

* 对边缘检测后的图像应用霍夫变换，以检测图像中的直线。
* 在霍夫空间中累计投票，找到代表车道线的直线。

**4. 线条筛选**

* 根据车道线的几何特性（如斜率、位置等）筛选出合适的线条。
* 可以排除那些斜率不符合车道线特性的线条。

**5. 车道线绘制**

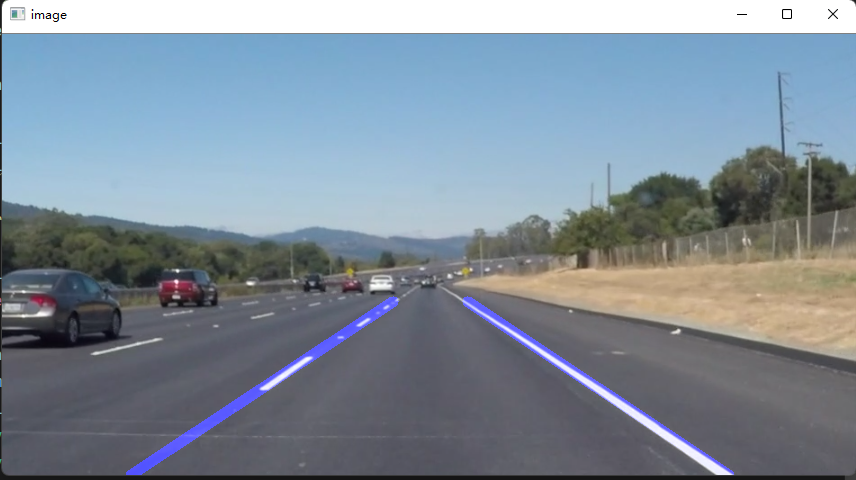
* 将检测到的车道线绘制在原始图像上。
* 可以使用多项式拟合来平滑车道线。

## 实验结果

实验输入如下：



输出结果如下：



实验代码及分析如下：

import numpy as np  
import cv2 as cv  
import matplotlib.pyplot as plt  
  
# 灰度图转换  
def grayscale(image):  
 return cv.cvtColor(image, cv.COLOR\_RGB2GRAY)  
# Canny边缘检测  
def canny(image, low\_threshold, high\_threshold):  
 return cv.Canny(image, low\_threshold, high\_threshold)  
# 高斯滤波  
def gaussian\_blur(image, kernel\_size):  
 return cv.GaussianBlur(image, (kernel\_size, kernel\_size), 0)  
  
# 生成感兴趣区域即Mask掩模  
def region\_of\_interest(image, vertices):  
  
 mask = np.zeros\_like(image) # 生成图像大小一致的zeros矩  
 # 填充顶点vertices中间区域  
 if len(image.shape) > 2:  
 channel\_count = image.shape[2]  
 ignore\_mask\_color = (255,) \* channel\_count  
 else:  
 ignore\_mask\_color = 255  
 # 填充函数  
 cv.fillPoly(mask, vertices, ignore\_mask\_color)  
 masked\_image = cv.bitwise\_and(image, mask)  
 return masked\_image  
  
# 原图像与车道线图像按照a:b比例融合  
def weighted\_img(img, initial\_img, a=0.8, b=1., c=0.):  
 return cv.addWeighted(initial\_img, a, img, b, c)  
def draw\_lines(image, lines, color=[255,0,0], thickness=2):  
  
 right\_y\_set = []  
 right\_x\_set = []  
 right\_slope\_set = []  
  
 left\_y\_set = []  
 left\_x\_set = []  
 left\_slope\_set = []  
  
 slope\_min = .35 # 斜率低阈值  
 slope\_max = .85 # 斜率高阈值  
 middle\_x = image.shape[1] / 2 # 图像中线x坐标  
 max\_y = image.shape[0] # 最大y坐标  
  
 for line in lines:  
 for x1, y1, x2, y2 in line:  
 fit = np.polyfit((x1, x2), (y1, y2), 1) # 拟合成直线  
 slope = fit[0] # 斜率  
  
 if slope\_min < np.absolute(slope) <= slope\_max:  
  
 # 将斜率大于0且线段X坐标在图像中线右边的点存为右边车道线  
 if slope > 0 and x1 > middle\_x and x2 > middle\_x:  
 right\_y\_set.append(y1)  
 right\_y\_set.append(y2)  
 right\_x\_set.append(x1)  
 right\_x\_set.append(x2)  
 right\_slope\_set.append(slope)  
  
 # 将斜率小于0且线段X坐标在图像中线左边的点存为左边车道线  
 elif slope < 0 and x1 < middle\_x and x2 < middle\_x:  
 left\_y\_set.append(y1)  
 left\_y\_set.append(y2)  
 left\_x\_set.append(x1)  
 left\_x\_set.append(x2)  
 left\_slope\_set.append(slope)  
  
 # 绘制左车道线  
 if left\_y\_set:  
 lindex = left\_y\_set.index(min(left\_y\_set)) # 最高点  
 left\_x\_top = left\_x\_set[lindex]  
 left\_y\_top = left\_y\_set[lindex]  
 lslope = np.median(left\_slope\_set) # 计算平均值  
  
 # 根据斜率计算车道线与图片下方交点作为起点  
 left\_x\_bottom = int(left\_x\_top + (max\_y - left\_y\_top) / lslope)  
  
 # 绘制线段  
 cv.line(image, (left\_x\_bottom, max\_y), (left\_x\_top, left\_y\_top), color, thickness)  
  
 # 绘制右车道线  
 if right\_y\_set:  
 rindex = right\_y\_set.index(min(right\_y\_set)) # 最高点  
 right\_x\_top = right\_x\_set[rindex]  
 right\_y\_top = right\_y\_set[rindex]  
 rslope = np.median(right\_slope\_set)  
  
 # 根据斜率计算车道线与图片下方交点作为起点  
 right\_x\_bottom = int(right\_x\_top + (max\_y - right\_y\_top) / rslope)  
  
 # 绘制线段  
 cv.line(image, (right\_x\_top, right\_y\_top), (right\_x\_bottom, max\_y), color, thickness)  
  
def hough\_lines(img, rho, theta, threshold, min\_line\_len, max\_line\_gap):  
  
 # rho：线段以像素为单位的距离精度  
 # theta : 像素以弧度为单位的角度精度(np.pi/180较为合适)  
 # threshold : 霍夫平面累加的阈值  
 # minLineLength : 线段最小长度(像素级)  
 # maxLineGap : 最大允许断裂长度  
 lines = cv.HoughLinesP(img, rho, theta, threshold, np.array([]), minLineLength=min\_line\_len, maxLineGap=max\_line\_gap)  
 return lines  
  
def process\_image(image):  
  
 rho = 1 # 霍夫像素单位  
 theta = np.pi / 180 # 霍夫角度移动步长  
 hof\_threshold = 20 # 霍夫平面累加阈值threshold  
 min\_line\_len = 30 # 线段最小长度  
 max\_line\_gap = 60 # 最大允许断裂长度  
 kernel\_size = 5 # 高斯滤波器大小size  
 canny\_low\_threshold = 75 # canny边缘检测低阈值  
 canny\_high\_threshold = canny\_low\_threshold \* 3 # canny边缘检测高阈值  
 alpha = 0.8 # 原图像权重  
 beta = 1. # 车道线图像权重  
 lambda\_ = 0.  
 imshape = image.shape # 获取图像大小  
  
 # 灰度图转换  
 gray = grayscale(image)  
 # 高斯滤波  
 blur\_gray = gaussian\_blur(gray, kernel\_size)  
 # Canny边缘检测  
 edge\_image = canny(blur\_gray, canny\_low\_threshold, canny\_high\_threshold)  
 # 生成Mask掩模  
 vertices = np.array([[(0, imshape[0]), (9 \* imshape[1] / 20, 11 \* imshape[0] / 18),  
 (11 \* imshape[1] / 20, 11 \* imshape[0] / 18), (imshape[1], imshape[0])]], dtype=np.int32)  
 masked\_edges = region\_of\_interest(edge\_image, vertices)  
 # 基于霍夫变换的直线检测  
 lines = hough\_lines(masked\_edges, rho, theta, hof\_threshold, min\_line\_len, max\_line\_gap)  
 line\_image = np.zeros\_like(image)  
 # 绘制车道线线段  
 draw\_lines(line\_image, lines, thickness=10)  
 # 图像融合  
 lines\_edges = weighted\_img(image, line\_image, alpha, beta, lambda\_)  
 return lines\_edges  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 image = cv.imread("E:\\b.png")  
 line\_image = process\_image(image)  
 cv.imshow('image',line\_image)  
 cv.waitKey(0)

## 实验体会

**算法理解**：深入理解霍夫变换和Canny边缘检测算法的原理，以及它们在图像处理中的应用。

**参数调整**：认识到参数选择对检测结果的影响，如Canny算法中的阈值、霍夫变换中的累加器阈值等。

**图像预处理的重要性**：理解图像预处理步骤对于提高检测准确性的重要性，如噪声去除和边缘增强。

**实际应用**：体会到车道线检测在自动驾驶系统中的关键作用，以及如何将理论算法应用于实际问题。

**挑战**：在实际应用中可能会遇到各种挑战，如光照变化、路面状况、遮挡等，这些都会影响车道线检测的准确性。

**实验三 学号识别**

## 实验目的

手写数字的识别是机器视觉的入门级项目，是机器视觉的“Hello word”，其在实际场景中有广泛的应用场景。请设计手写数字识别方法识别自己的学号照片。

## 实验原理

**卷积神经网络（CNN）**

* CNN是一种用于图像识别的深度学习模型，它能够有效地提取图像特征。
* CNN由多个卷积层、池化层和全连接层组成。
* 卷积层通过卷积操作提取图像中的局部特征。
* 池化层降低特征的空间维度，减少计算量。
* 全连接层将提取的特征映射到最终的分类结果。

**激活函数和损失函数**

* 激活函数（如ReLU）用于引入非线性因素，使得神经网络能够学习和模拟更复杂的函数。
* 损失函数（如交叉熵）用于量化模型预测与真实标签之间的差异，指导模型学习。

## 实验方法

**1. 数据收集**

* 拍摄自己的学号照片，确保数字清晰可辨，背景单一，光线均匀。
* 将照片裁剪成单个数字的图像，每个数字单独保存为一个文件。

**2. 数据预处理**

* 调整图像大小，使其符合神经网络的输入要求（例如，28x28像素）。
* 将图像转换为灰度图，减少计算复杂度。
* 应用图像增强技术（如旋转、缩放、扭曲等）来扩充数据集，提高模型的泛化能力。

**3. 模型选择**

* 选择一个适合手写数字识别的模型，如卷积神经网络（CNN）。
* 可以使用现成的模型架构，如LeNet、AlexNet等，或者设计自己的网络结构。

**4. 模型训练**

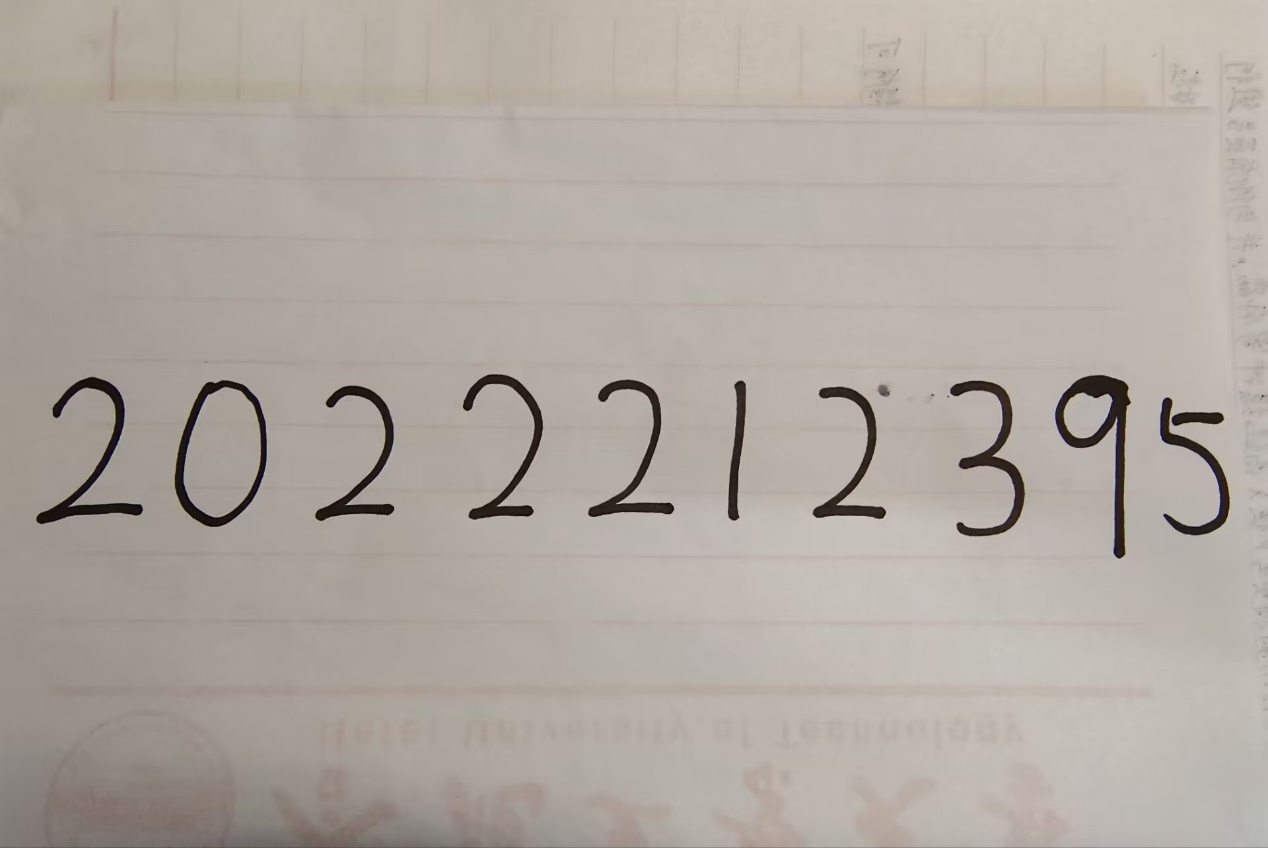
* 将预处理后的图像数据分为训练集、验证集和测试集。
* 使用训练集数据来训练神经网络模型。
* 使用验证集来调整模型参数（如学习率、批次大小等）。
* 评估模型性能，使用交叉熵损失函数和梯度下降优化算法。

**5. 模型评估**

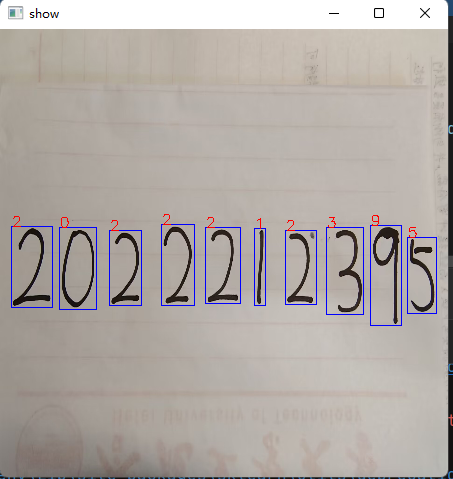
* 使用测试集来评估模型的准确性。
* 计算模型的精确率、召回率和F1分数等指标。

## 实验结果

输入图片：



输出结果：



实验代码及分析如下：

import os.path  
import cv2  
import numpy as np  
from matplotlib import pyplot as plt  
from torchvision import datasets  
from torchvision import transforms  
from sklearn import svm  
from sklearn import preprocessing  
from sklearnex import patch\_sklearn  
import pickle  
from torch.utils.tensorboard import SummaryWriter  
from sklearn.model\_selection import learning\_curve  
  
def LoadMnistDataset(train=True, data\_enhance=False):  
 mnist\_set = datasets.MNIST(root="./MNIST", train=train, download=True)  
 x\_, y\_ = list(zip(\*([(np.array(img), target) for img, target in mnist\_set])))  
 sets\_raw = []  
 sets\_r20 = []  
 sets\_invr20 = []  
 y = []  
 y\_r20 = []  
 y\_invr20 = []  
 sets = []  
 matrix\_r20 = cv2.getRotationMatrix2D((14, 14), 25, 1.0)  
 matrix\_invr20 = cv2.getRotationMatrix2D((14, 14), -25, 1.0)  
 select = 0  
 for idx in range(len(x\_)):  
 # 对图像进行二值化以及数据增强  
 \_, img = cv2.threshold(x\_[idx], 255, 255, cv2.THRESH\_OTSU)  
 sets\_raw.append(np.array(img.data).reshape(784))  
 y.append(y\_[idx])  
 if data\_enhance:  
 if select % 2 == 0:  
 img\_r20 = ~cv2.warpAffine(~img, matrix\_r20, (28, 28), borderValue=(255, 255, 255))  
 sets\_r20.append(np.array(img\_r20.data).reshape(784))  
 y\_r20.append(y\_[idx])  
 else:  
 img\_invr20 = ~cv2.warpAffine(~img, matrix\_invr20, (28, 28), borderValue=(255, 255, 255))  
 sets\_invr20.append(np.array(img\_invr20.data).reshape(784))  
 y\_invr20.append(y\_[idx])  
 select += 1  
  
 # 数据增强  
 sets = sets\_raw + sets\_r20 + sets\_invr20  
 sets = np.array(sets)  
 print(sets.shape)  
 if data\_enhance:  
 y = y + y\_r20 + y\_invr20  
 return sets, y  
  
def SaveSvcModel(svc\_model, file\_path="./SVC"):  
 with open(file\_path, 'wb') as fs:  
 pickle.dump(svc\_model, fs)  
  
def LoadSvcModel(file\_path="./SVC"):  
 if not os.path.exists(file\_path):  
 assert "Model Do Not Exist"  
 with open(file\_path, 'rb') as fs:  
 svc\_model = pickle.load(fs)  
 return svc\_model  
  
def TrainSvc(c, enhance):  
 # 读取数据集，训练集及测试集  
 images\_train, targets\_train = LoadMnistDataset(train=True, data\_enhance=enhance)  
 images\_test, targets\_test = LoadMnistDataset(train=False, data\_enhance=enhance)  
  
 # 训练  
 svc\_model = svm.SVC(C=c, kernel='rbf', decision\_function\_shape='ovr')  
 svc\_model.fit(images\_train, targets\_train)  
  
 # 在测试集上测试准确度  
  
 res = svc\_model.predict(images\_test)  
 correct = (res == targets\_test).sum()  
 accuracy = correct / len(images\_test)  
 print(f"测试集上的准确率为{accuracy \* 100}%")  
 return svc\_model  
  
def PreProcessFatFont(image, show=False):  
 # 白底黑字转黑底白字  
 pre\_ = ~image  
  
 # 转单通道灰度  
 pre\_ = cv2.cvtColor(pre\_, cv2.COLOR\_BGR2GRAY)  
 # 二值化  
 \_, pre\_ = cv2.threshold(pre\_, thresh=0, maxval=255, type=cv2.THRESH\_OTSU)  
  
 # resize后添加黑色边框，亲测可提高识别率  
 pre\_ = cv2.resize(pre\_, (112, 112))  
 \_, pre\_ = cv2.threshold(pre\_, thresh=0, maxval=255, type=cv2.THRESH\_OTSU)  
  
 back = np.zeros((300, 300), np.uint8)  
 back[29:141, 29:141] = pre\_  
 pre\_ = back  
  
 if show:  
 cv2.imshow("show", pre\_)  
 cv2.waitKey(0)  
  
 # 做一次开运算(腐蚀 + 膨胀)  
 kernel = np.ones((2, 2), np.uint8)  
 pre\_ = cv2.erode(pre\_, kernel, iterations=1)  
 kernel = np.ones((3, 3), np.uint8)  
 pre\_ = cv2.dilate(pre\_, kernel, iterations=1)  
  
 # 第二次resize  
 pre\_ = cv2.resize(pre\_, (56, 56))  
 \_, pre\_ = cv2.threshold(pre\_, thresh=0, maxval=255, type=cv2.THRESH\_OTSU)  
  
 # 做一次开运算(腐蚀 + 膨胀)  
 kernel = np.ones((2, 2), np.uint8)  
 pre\_ = cv2.erode(pre\_, kernel, iterations=1)  
 kernel = np.ones((3, 3), np.uint8)  
 pre\_ = cv2.dilate(pre\_, kernel, iterations=1)  
  
 # resize成输入规格  
 pre\_ = cv2.resize(pre\_, (28, 28))  
 \_, pre\_ = cv2.threshold(pre\_, thresh=0, maxval=255, type=cv2.THRESH\_OTSU)  
  
 # 转换为SVM的输入格式  
 pre\_ = np.array(pre\_).flatten().reshape(1, -1)  
 return pre\_  
  
def PreProcessThinFont(image, show=False):  
 # 白底黑字转黑底白字  
 pre\_ = ~image  
  
 # 转灰度图  
 pre\_ = cv2.cvtColor(pre\_, cv2.COLOR\_BGR2GRAY)  
  
 # 增加黑色边框  
 pre\_ = cv2.resize(pre\_, (112, 112))  
 \_, pre\_ = cv2.threshold(pre\_, thresh=0, maxval=255, type=cv2.THRESH\_OTSU)  
 back = np.zeros((170, 170), dtype=np.uint8) # 这里不指明类型会导致后续矩阵强转为float64，无法使用大津法阈值  
 back[29:141, 29:141] = pre\_  
 pre\_ = back  
  
 if show:  
 cv2.imshow("show", pre\_)  
 cv2.waitKey(0)  
  
 # 对细字体先膨胀一下  
 kernel = np.ones((3, 3), np.uint8)  
 pre\_ = cv2.dilate(pre\_, kernel, iterations=2)  
  
 # 第二次resize  
 pre\_ = cv2.resize(pre\_, (56, 56))  
  
 \_, pre\_ = cv2.threshold(pre\_, thresh=0, maxval=255, type=cv2.THRESH\_OTSU)  
  
 # 做一次开运算(腐蚀 + 膨胀)  
 kernel = np.ones((2, 2), np.uint8)  
 pre\_ = cv2.erode(pre\_, kernel, iterations=1)  
 kernel = np.ones((3, 3), np.uint8)  
 pre\_ = cv2.dilate(pre\_, kernel, iterations=1)  
  
 # resize成输入规格  
 pre\_ = cv2.resize(pre\_, (28, 28))  
 \_, pre\_ = cv2.threshold(pre\_, thresh=0, maxval=255, type=cv2.THRESH\_OTSU)  
  
 # 转换为SVM输入格式  
 pre\_ = np.array(pre\_).flatten().reshape(1, -1)  
  
 return pre\_  
  
def ShowRoi(res\_list):  
 back = 255 \* np.ones((1000, 1500, 3), dtype=np.uint8)  
 # 图片x轴偏移量  
 tlx = 0  
  
 for roi in res\_list:  
 if tlx + roi.shape[1] > back.shape[1]:  
 break  
 # 每次在原图上加上一个roi  
 back[0:roi.shape[0], tlx:tlx + roi.shape[1], :] = roi  
 tlx += roi.shape[1]  
  
 cv2.imshow("show", back)  
 cv2.waitKey(0)  
  
def FindNumbers(src, thin=True):  
 # 拷贝  
 dst = src.copy()  
 paint = src.copy()  
 roi = src.copy()  
 dst = ~dst  
  
 # 预处理  
 paint = cv2.resize(paint, (448, 448))  
 dst = cv2.resize(dst, (448, 448))  
  
 # 记录缩放比例,后来看这一步好像没啥意义  
 fx = src.shape[1] / 448  
 fy = src.shape[0] / 448  
  
 # 转单通道  
 dst = cv2.cvtColor(dst, cv2.COLOR\_BGR2GRAY)  
  
 # 边缘检测后二值化，直接二值化的话由于采光不同的原因灰度直方图峰与峰之间可能会差距过大，导致二值图的分割不准确  
 # 而边缘检测对像素突变更加敏感，因此采用Canny边缘检测后二值化  
 cv2.Canny(dst, 200, 200, dst)  
  
 # 对于平常笔写的字太细，膨胀一下  
 if thin:  
 kernel = np.ones((5, 5), np.uint8)  
 dst = cv2.dilate(dst, kernel, iterations=1)  
  
 # 寻找外围轮廓  
 contours, \_ = cv2.findContours(dst, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)  
  
 # 提取roi  
 roi\_list = []  
 rect\_list = []  
 for contour in contours:  
 rect = cv2.boundingRect(contour)  
 if not ((rect[2] \* rect[3] < 400 or rect[2] \* rect[3] > 448 \* 448 / 2.5) or (rect[3] < rect[2])):  
 cv2.rectangle(paint, rect, (255, 0, 0), 1)  
 x\_min = rect[0] \* fx  
 x\_max = (rect[0] + rect[2]) \* fx  
 y\_min = rect[1] \* fy  
 y\_max = (rect[1] + rect[3]) \* fy  
 roi\_list.append(roi[int(y\_min):int(y\_max), int(x\_min):int(x\_max)].copy())  
 rect\_list.append(rect)  
 return paint, roi\_list, rect\_list  
  
def ShowDataTxt(data):  
 print("------------------------------")  
 for i in range(28):  
 for j in range(28):  
 print(0 if data[0][i \* 28 + j] == 255 else 1, end='')  
 print('\n')  
 print("------------------------------")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 # 加载  
 patch\_sklearn()  
 model\_path = "./SVC\_C1\_enhance.pkl"  
  
 if os.path.exists(model\_path):  
 print("Model Exist, Load Form Serialization")  
 model = LoadSvcModel(model\_path)  
 else:  
 print("Model Do Not Exist, Train")  
  
 # 训练  
 model = TrainSvc(1, False)  
  
 # 保存  
 SaveSvcModel(model, model\_path)  
  
 # 测试  
 paint, nums, rects = FindNumbers(cv2.imread("E:\\c.jpg"))  
 predict\_nums = []  
 for img in nums:  
 data = PreProcessThinFont(img, show=False)  
 # ShowDataTxt(data)  
 predict\_nums.append(model.predict(data)[0])  
 for i in range(len(predict\_nums)):  
 cv2.putText(paint, str(predict\_nums[i]), (rects[i][0], rects[i][1]), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 0, 255),  
 1)  
 recognized\_student\_id = ''.join(map(str, predict\_nums))  
 print(recognized\_student\_id, 'recognized student ID')  
 cv2.imshow("show", paint)  
 cv2.waitKey(0)

## 实验体会

**数据预处理的重要性**：通过实验可以体会到，良好的数据预处理是模型成功的关键。

**模型选择与调整**：理解不同神经网络结构对模型性能的影响，以及如何根据问题调整模型参数。

**训练过程**：通过训练过程，可以观察到模型是如何逐渐学习并提高识别准确性的。

**过拟合与泛化**：实验中可能会遇到过拟合问题，理解如何通过数据增强、正则化等技术提高模型的泛化能力。

**实际应用**：将模型应用于实际场景，如识别自己的学号，可以加深对机器视觉应用的理解。

**图像质量**：实际拍摄的照片可能存在各种问题，如倾斜、模糊等，这些都会增加识别难度。

**模型复杂性**：设计一个既不过于简单也不过于复杂的模型是一个挑战，需要平衡模型性能和计算资源。