合肥工業大學

# 机器视觉实验报告

学　　号：2023217592

姓　　名：于宝旭

专业班级：智 能 科 23-3班

完成时间：2025年12月29日

# 机 器 视 觉

## 目 录

# 实验一　图像滤波

## 一、实验内容与步骤

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，其中，给定卷积核：

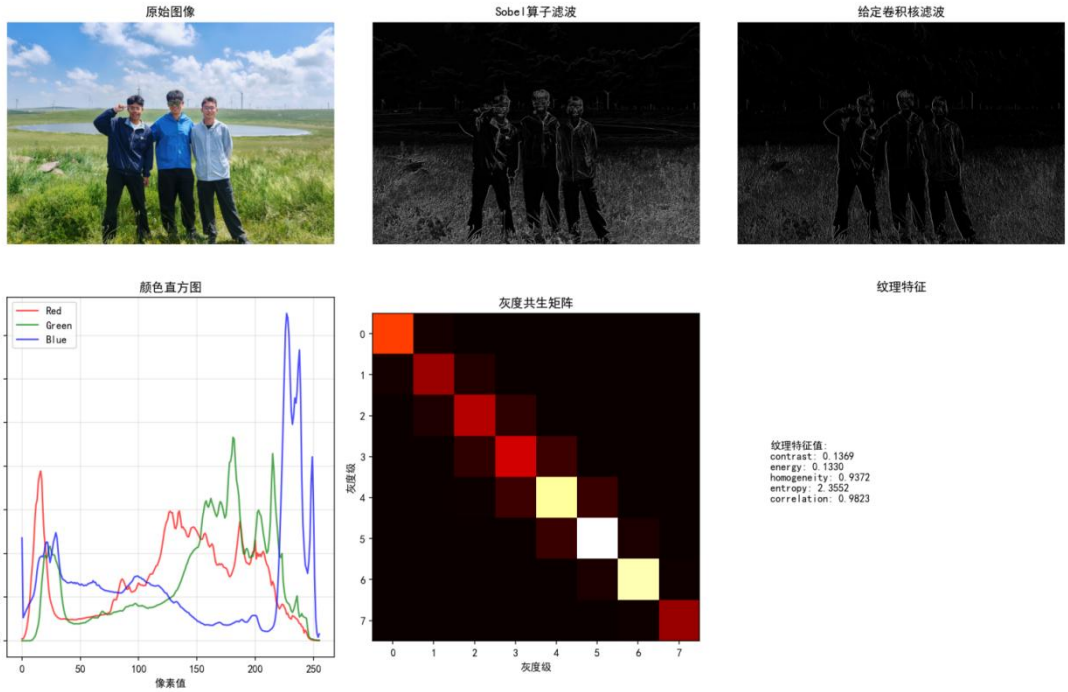
$$\begin{matrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{matrix}$$

具体要求：

- 任务输入：自己拍摄的图像。
- 任务输出：经过Sobel算子滤波的图像，经过给定卷积核滤波的图像，可视化图像的颜色直方图，保存纹理特征至npy格式。
- 滤波、直方图计算、纹理特征提取过程不可以调用函数包。
- 代码语言不限，纹理特征提取方法不限，要求提交整个算法源代码，实验结果，算法分析等内容。

## 二、实验结果

通过手动实现卷积、滤波、直方图计算和纹理特征提取等核心图像处理算法，从一幅输入图像中系统地提取了边缘特征（使用Sobel算子）、颜色分布特征（RGB直方图）和纹理统计特征（基于灰度共生矩阵的对比度、能量、同质性、熵和相关性），最后将分析结果以可视化图表形式展示，并保存特征数据供后续使用。




## 三、实验原理思考

### 1. 卷积操作

卷积是图像处理中的基本操作，通过一个卷积核（滤波器）在图像上滑动，对局部像素进行加权求和，实现模糊、锐化、边缘检测等效果。

公式： $$G(i,j) = \sum_{u=-k}^{k} \sum_{v=-k}^{k} K(u,v) \cdot I(i+u, j+v)$$

其中 K为卷积核，I为图像局部区域。

```
def convolution(self, image, kernel, padding='zero'):
    # 获取图像和卷积核尺寸
    img_h, img_w = image.shape
    kernel_h, kernel_w = kernel.shape

    # 计算填充大小
    pad_h = kernel_h // 2
    pad_w = kernel_w // 2

    # 填充图像（零填充或边缘复制）
    if padding == 'zero':
```

```
        padded_image = np.zeros((img_h + 2*pad_h, img_w + 2*pad_w),
dtype=np.float32)
        padded_image[pad_h:pad_h+img_h, pad_w:pad_w+img_w] = image
    else:
        padded_image = np.pad(image, ((pad_h, pad_h), (pad_w, pad_w)), 'edge')

    # 卷积计算
    output = np.zeros_like(image, dtype=np.float32)
    for i in range(img_h):
        for j in range(img_w):
            window = padded_image[i:i+kernel_h, j:j+kernel_w]
            output[i, j] = np.sum(window * kernel)  # 加权求和
```

**2. Sobel 边缘检测**

Sobel 算子是一种一阶梯度算子，用于检测图像中的边缘。包含两个卷积核：

- Sobel_x：检测垂直边缘（梯度在水平方向变化）
- Sobel_y：检测水平边缘（梯度在垂直方向变化）

梯度幅值计算： $G = \sqrt{G_x^2 + G_y^2}$

代码实现：

分别对灰度图应用sobel_x和sobel_y核。

计算梯度幅值并归一化到 [0, 255]。

```
# Sobel算子定义
self.sobel_x = np.array([[-1, 0, 1],
                         [-2, 0, 2],
                         [-1, 0, 1]], dtype=np.float32)

self.sobel_y = np.array([[-1, -2, -1],
                         [0, 0, 0],
                         [1, 2, 1]], dtype=np.float32)

def sobel_filter(self):
    # 分别计算x和y方向梯度
    grad_x = self.convolution(self.gray_image, self.sobel_x)
    grad_y = self.convolution(self.gray_image, self.sobel_y)
```

```
    # 计算梯度幅值
    sobel_magnitude = np.sqrt(grad_x.astype(np.float32)**2 +
                              grad_y.astype(np.float32)**2)
    sobel_magnitude = np.clip(sobel_magnitude, 0, 255).astype(np.uint8)
    return sobel_magnitude, grad_x, grad_y
```

### 3. 颜色直方图（Color Histogram）

统计图像中每个颜色强度（0－255）出现的频率。

反映图像的颜色分布特征，可用于图像分类、检索等。

通常对RGB三个通道分别统计。

代码实现：

手动遍历每个像素，统计R、G、B三个通道的直方图，避免使用cv2.calcHist。

```
def compute_color_histogram(self, bins=256):
    # 分离RGB通道
    r_channel = self.rgb_image[:, :, 0]
    g_channel = self.rgb_image[:, :, 1]
    b_channel = self.rgb_image[:, :, 2]

    # 初始化直方图数组
    r_hist = np.zeros(bins, dtype=np.int32)
    g_hist = np.zeros(bins, dtype=np.int32)
    b_hist = np.zeros(bins, dtype=np.int32)

    # 遍历每个像素统计
    height, width = r_channel.shape
    for i in range(height):
        for j in range(width):
            r_hist[r_channel[i, j]] += 1
            g_hist[g_channel[i, j]] += 1
            b_hist[b_channel[i, j]] += 1
```

### 4. 纹理特征提取（基于灰度共生矩阵，GLCM）

灰度共生矩阵（Gray-Level Co-occurrence Matrix, GLCM） 描述图像中成对像素灰度值在特定方向和距离上出现的频率。

从GLCM中可提取多种统计特征：

- 对比度（Contrast）：反映纹理的清晰程度。
- 能量（Energy） / 均匀性（Uniformity）：反映图像灰度分布的均匀性。
- 同质性（Homogeneity）：反映局部灰度变化的一致性。
- 熵（Entropy）：反映纹理的复杂程度。
- 相关性（Correlation）：反映灰度值的线性依赖程度。

```
def extract_texture_features(self, distances=[1], angles=[0]):
    # 灰度量化（减少计算量）
    levels = 8
    quantized = (gray // (256 // levels)).astype(np.uint8)

    # 构建GLCM
    glcm = np.zeros((levels, levels), dtype=np.float32)
    for i in range(height):
        for j in range(width - distance):
            row = quantized[i, j]
            col = quantized[i, j + distance]
            glcm[row, col] += 1

    # 归一化
    glcm_sum = np.sum(glcm)
    if glcm_sum > 0:
        glcm = glcm / glcm_sum
    # 计算对比度
    contrast = 0
    for i in range(levels):
        for j in range(levels):
            contrast += glcm[i, j] * (i - j) ** 2
    # 计算能量
    energy = 0
    for i in range(levels):
        for j in range(levels):
            energy += glcm[i, j] ** 2
    # 计算同质性
    homogeneity = 0
    for i in range(levels):
        for j in range(levels):
```

```
            homogeneity += glcm[i, j] / (1 + abs(i - j))
    # 计算熵
    entropy = 0
    for i in range(levels):
        for j in range(levels):
            if glcm[i, j] > 0:
                entropy -= glcm[i, j] * np.log(glcm[i, j])
    # 计算相关性
    i_vals, j_vals = np.meshgrid(np.arange(levels), np.arange(levels),
indexing='ij')
    mean_i = np.sum(i_vals * glcm)
    mean_j = np.sum(j_vals * glcm)
    std_i = np.sqrt(np.sum((i_vals - mean_i) ** 2 * glcm))
    std_j = np.sqrt(np.sum((j_vals - mean_j) ** 2 * glcm))
    if std_i > 0 and std_j > 0:
        correlation = np.sum((i_vals - mean_i) * (j_vals - mean_j) * glcm) /
(std_i * std_j)
```

## 函数功能说明表：

表1-1 函数功能

| 函数名 | 功能说明 | 核心算法/原理 | 输入参数 | 返回值 |
|---|---|---|---|---|
| convolution | 手动实现卷积操作 | 滑动窗口加权求和 | image: 输入图像<br>kernel: 卷积核<br>padding: 填充方式 | 卷积后的图像 |
| sobel_filter | 应用Sobel算子进行边缘检测 | 一阶梯度算子<br>$G = \sqrt{(Gx^2 + Gy^2)}$ | 无 | sobel_magnitude: 梯度幅值<br>grad_x: x方向梯度<br>grad_y: y方向梯度 |
| given_kernel_filter | 应用给定的自定义卷积核进行滤波 | 自定义特征提取 | 无 | 滤波后的图像 |
| compute_color_histogram | 计算RGB三通道的颜色直方图 | 像素值频率统计 | bins: 直方图区间数 | r_hist: 红色直方图<br>g_hist: 绿色直方图<br>b_hist: 蓝色直方图 |
| extract_texture_features | 基于灰度共生矩阵提取纹理特征 | GLCM统计特征提取 | distances: 像素距离<br>angles: 像素角度 | features: 纹理特征字典<br>glcm: 灰度共生矩阵 |
| save_texture_features | 保存纹理特征 | 数据序列 | features: 特征字典 | 特征向量数组 |

| 函数名 | 功能说明 | 核心算法/原理 | 输入参数 | 返回值 |
| --- | --- | --- | --- | --- |
| | 到npy文件 | 化 | filename: 文件名 | |
| visualize_results | 可视化所有处理结果 | 多子图显示 | sobel_result: Sobel结果<br>given_kernel_result: 给定核结果<br>r_hist, g_hist, b_hist: 直方图数据 | 无 |
| process_image | 图像处理主函数，协调所有处理步骤 | 图像处理流水线 | 无 | results: 包含所有处理结果的字典 |

# 四、实验总结

通过本次实验，我深刻体会到数字图像处理从理论到实践的完整转化过程。在手动实现卷积运算时，我直观理解了卷积核如何通过加权求和提取局部特征，尤其是Sobel算子在边缘检测中展现的方向敏感性；在编写颜色直方图统计函数的过程中，我认识到简单像素统计背后反映的是图像的整体色彩分布规律；最令我收获颇丰的是纹理特征提取部分，通过构建灰度共生矩阵并计算对比度、熵等特征，让我真正理解了计算机如何“量化”人眼感受到的纹理特性。实验过程中遇到的边界处理、数值归一化等问题，也让我意识到理论公式在实际编码中需要考虑的细节。这次实验不仅巩固了我对图像处理基础算法的理解，更培养了我从问题出发、分模块实现复杂系统的工程能力，为后续深入学习计算机视觉奠定了扎实基础。

# 实验二　车道线检测

## 一、实验内容与步骤

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。

具体要求：

- 任务输入：自己拍摄的校园中道路图像（画有车道线的路）。
- 任务输出：图像中车道线的位置。
- 代码语言不限，方法不限，要求提交整个算法源代码，模型结果，算法分析等内容。

## 二、实验结果

1. 颜色掩膜

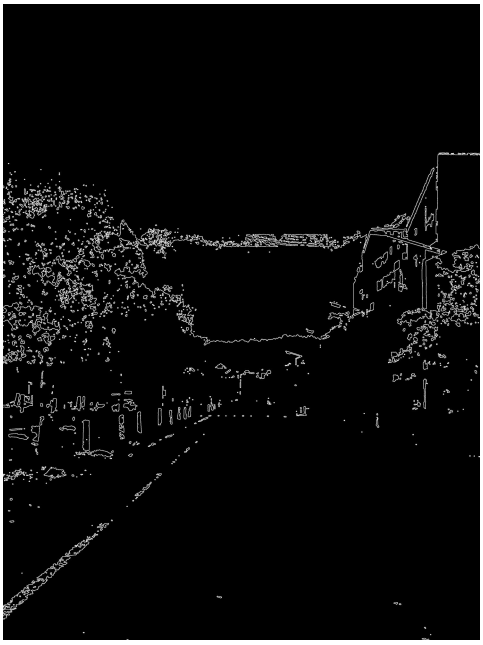
```
color_mask = _color_mask_yellow_white(image)
```

作用：在HLS颜色空间中提取白色和黄色的车道线

输出：二值图像，白色/黄色区域为255，其他为0

保存为：debug_color_mask.png

此图展示了算法在HLS颜色空间中提取白色和黄色车道线的中间处理结果。图中白色区域表示符合预设颜色阈值（白色：高亮度低饱和度；黄色：特定色调范围）的像素点。可以观察到，图像中的车道线区域被有效提取出来，而道路旁的绿化带、灰色路面和阴影等干扰因素则被成功过滤掉。该掩膜的质量直接影响后续边缘检测的准确性。



2. 边缘检测

```
edges = cv2.Canny(color_mask, 50, 150)
```

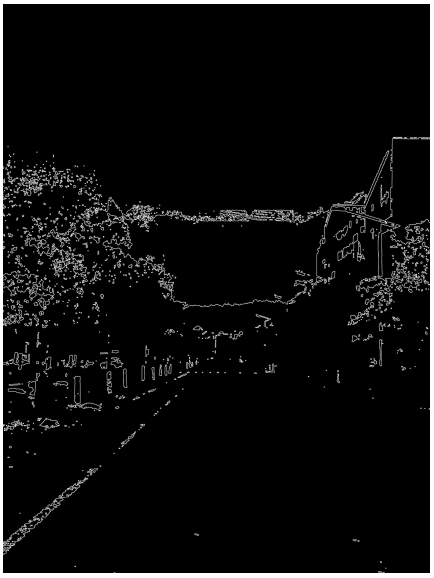
输入：直接使用color_mask

作用：在颜色掩膜上检测边缘

关键点：这里的边缘检测只作用于颜色掩膜中的白色区域，即已经过滤掉非车道线颜色

保存为：debug_edges.png

这张图显示了Canny边缘检测器在颜色掩膜基础上生成的边缘轮廓。图中的白色线条代表图像中所有检测到的强度变化边界，包括车道线边缘以及少量残留噪声。这些边缘构成了后续霍夫变换的输入数据。从图中可以看出，车道线边缘连续且清晰，而大部分非车道线区域则相对干净，说明颜色掩膜预处理显著提升了边缘检测的针对性。



3. ROI区域掩膜

```
edges_roi = region_of_interest(edges, roi_vertices)
```

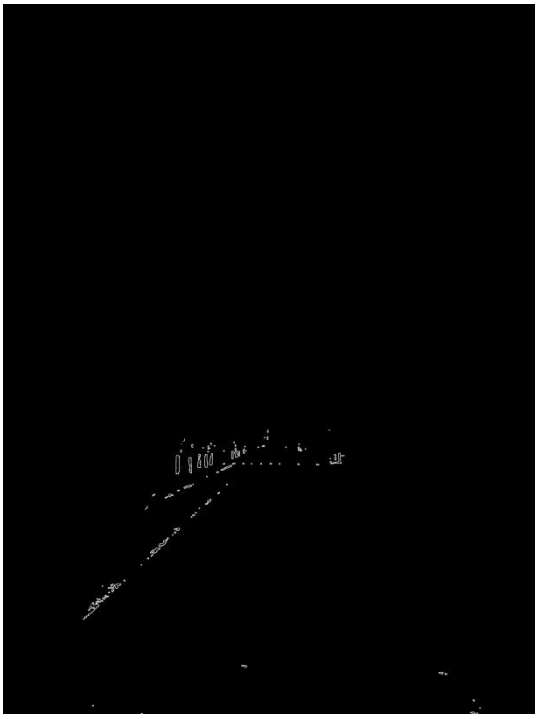
输入：上一步的边缘图像

作用：通过梯形ROI区域进一步限制检测范围

保存为：debug_edges_roi.png

此图展示了经过感兴趣区域裁剪后的边缘检测结果。算法只保留了车辆前方梯形区域内的边缘信息，完全剔除了天空、远处建筑和道路两侧无关区域的边缘干扰

。这种空间聚焦策略极大简化了后续直线检测的计算复杂度，并显著降低了误检率，使算法能够集中处理最可能包含车道线的关键区域。

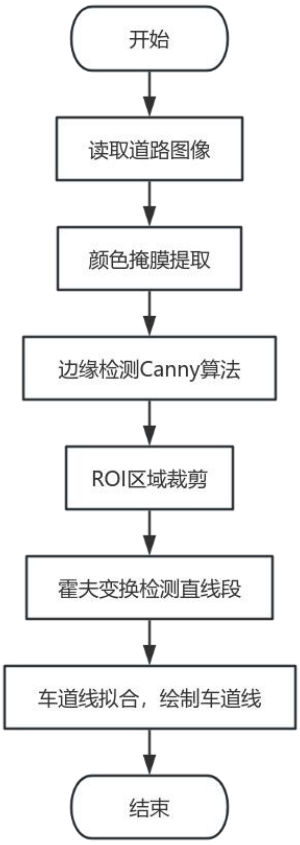


4. 最终检测结果

这张图是车道线检测的最终输出结果。在原拍摄图像的基础上，算法识别出的左右车道线被分别用醒目的绿色线条标记出来。可以看到检测到的车道线准确贴合实际道路边界，线条长度和位置都经过了智能调整，既覆盖了有效检测区域又避免了过度延伸。绿色线条采用半透明叠加方式，既清晰显示检测结果又不完全遮挡原图信息，方便人工验证算法的准确性。

## 三、实验原理思考

1. 实验流程图




2. 实验原理

(1). 颜色空间转换与特征初筛

系统首先对输入的BGR图像进行颜色空间转换。选择HLS空间而非常规的RGB空间，是因为HLS将颜色的亮度分量与色相、饱和度分离，更贴近人类对“白”和“黄”的感知定义。车道线颜色，特别是白色和黄色，在HLS空间中具有鲜明的聚集特征：白色表现为高亮度、低饱和度，其色相则分布广泛；黄色则集中在一个特定的色相范围，并辅以适当的饱和度和亮度。

因此，我们定义了严格的阈值条件来进行二元分割：

- 白色区域判定：像素亮度 $L \in [160, 255]$ 且饱和度 $S \in [0, 100]$。
- 黄色区域判定：像素色相 $H \in [10, 40]$，亮度 $L \in [80, 255]$ 且饱和度 $S \in [40, 255]$

将所有满足上述任一条件的像素置为白色（255），其余置为黑色（0），即得到一个初始的二值颜色掩膜。随后，利用形态学操作对此掩膜进行净化，得到只保留疑似白、黄车道线区域的纯净图像。这一步极大地抑制了绿化带、灰色路面、阴影等非车道线彩色物体的干扰，将问题从“找线条”简化为“在特定颜色区域内找线条”。

(2). 边缘检测与空间聚焦

获得纯净的颜色掩膜后，系统在其上直接应用Canny边缘检测器。Canny算子的核心在于计算图像梯度，寻找像素强度剧烈变化的区域。其数学基础是卷积运算：使用水平(Sobel_x)和垂直(Sobel_y)方向的Sobel算子对图像进行滤波，得到梯度分量Gx和Gy。每个像素点的梯度幅值G和方向θ由下式计算：

$$G = \sqrt{G_x^2 + G_y^2}$$

$$\theta = \arctan 2(G_y, G_x)$$

通过非极大值抑制和双阈值滞后处理，最终得到单像素宽、连接良好的边缘图。

为了提高处理效率和鲁棒性，我们引入了感兴趣区域的概念。定义一个梯形的ROI，其底边与图像底边重合，顶边位于图像高度约60%处，左右两侧适当向内收缩。这个梯形模拟了车辆前方道路在透视投影下的主要区域。通过`cv2.bitwise_and`运算，将边缘图限制在此ROI内，从而彻底排除了天空、远方建筑物、道路两侧植被等无关区域的边缘噪声，使算法专注于前方路面。

(3). 线段检测与几何分析

在ROI边缘图上，使用概率霍夫变换检测直线段。霍夫变换的原理是将图像空间(x, y)中的一条直线y = kx + b映射到参数空间(k, b)的一个点。由于垂直直线的k为无穷大，实际采用极坐标参数化：直线表示为`ρ = x cosθ + y sinθ`。图像空间中共线的点会在参数空间(ρ, θ)中形成峰值。概率霍夫变换在此基础上直接返回线段的端点(x1, y1, x2, y2)。

检测到的原始线段数量众多且质量参差不齐。我们依据严格的几何约束进行过滤：

- 长度过滤：剔除过短的线段（如length < max(30, 0.03*(w+h)）），它们通常是噪声。
- 斜率过滤：保留斜率绝对值在[0.3, 5.0]之间的线段。过小的斜率接近水平线（可能是道路纹理或阴影边界），过大的斜率接近垂直线（可能是树木或建筑物轮廓），均非车道线特征。
- 位置分类：计算线段中点的x坐标x_mid。对于斜率为负且x_mid位于图像左侧的线段，归类为左车道线候选；对于斜率为正且x_mid位于图像右侧的线段，归类为右车道线候选。这一分类基于车道线在图像中“左负右正”的透视规律。

(4). 鲁棒拟合与车道线生成

同一侧的车道线候选线段在位置上存在离散和误差。为了得到一条平滑、连续、具有代表性的车道线，需要对所有候选点进行直线拟合。如果使用普通的最小二乘法，少数偏离较远的“离群点”会严重扭曲拟合结果。因此，本实验采用了基于Huber损失的鲁棒拟合方法（`cv2.fitLine with DIST_HUBER`）。

Huber损失函数L_δ(e)定义如下：$$L_\delta(e)=\begin{cases}\frac{1}{2}e^2, & |e|\le\delta\\ \delta\left(|e|-\frac{1}{2}\delta\right), & |e|>\delta\end{cases}$$

其中e是残差，δ是阈值参数。该函数对于小残差使用二次项，对于大残差使用一次项，从而降低离群点的影响权重。拟合过程即求解使所有点到直线距离的Huber损失之和最小的直线参数[vx, vy, x0, y0]。得到拟合直线的参数后，根据预设的绘制范围（通常从ROI顶部y_min到图像底部y_max），利用直线的点斜式方程反算出两个端点的x坐标：`x = x0 + (y - y0) * (vx / vy)`

## 四、实验总结

通过本次车道线检测实验，我深刻体会到计算机视觉在现实应用中的复杂性与魅力。最初以为简单的“识别线条”任务，在实际处理校园道路图像时却面临诸多挑战：多变的光照条件、路面阴影干扰、以及不同颜色的车道线标记，都让直接检测变得困难。然而，正是这些挑战推动我深入理解颜色空间转换、边缘检测和几何过滤等图像处理技术的精髓。例如，通过HLS颜色空间分离白黄车道线，我学会了如何针对特定目标设计过滤策略；而自适应ROI和鲁棒拟合则让我明白，在复杂环境中必须兼顾准确性与鲁棒性。这次实验不仅锻炼了我的代码实现和调试能力，更让我认识到，一个好的算法往往需要多层次的处理和备用方案，以应对实际场景中的不确定性。最终，当看到绿色车道线准确叠加在道路图像上时，我感受到了将理论知识转化为实际解决方案的成就感，也对自动驾驶视觉系统的底层原理有了更直观的认识。

# 实验三　学号识别

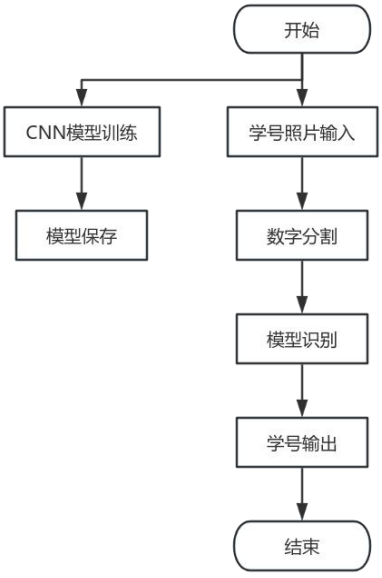
## 一、实验内容与要求

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。

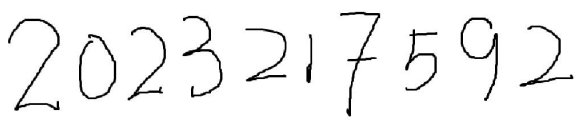
- 任务输入：学号照片。
- 任务输出：学号。
- 训练集：MNIST。
- 代码语言不限，方法不限，要求提交整个算法源代码，模型结果，算法分析等内容。
- 加分项：使用深度学习方法，代码环境名称以姓名缩写命名，实验报告中介绍代码环境配置过程。（已完成）

## 二、实验结果与源程序

**系统功能流程：**




**实验结果：**



```
模型已保存至 mnist_cnn.pt
正在处理图片: student_id.png

模型已保存至 mnist_cnn.pt
正在处理图片: student_id.png
模型已保存至 mnist_cnn.pt
正在处理图片: student_id.png
正在处理图片: student_id.png
预测学号: 1023213592
```

**源程序：**

```
class Net(nn.Module):
    #两层卷积层自动学习手写数字的局部特征
    #Dropout层在训练时随机屏蔽部分神经元，增强泛化能力
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, 3, 1)
        self.conv2 = nn.Conv2d(32, 64, 3, 1)
        self.dropout1 = nn.Dropout(0.25)
        self.dropout2 = nn.Dropout(0.5)
        self.fc1 = nn.Linear(9216, 128)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):   #前向传播函数
        x = self.conv1(x)   # 卷积操作：提取初级特征
        x = F.relu(x)
        x = self.conv2(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 2)
        x = self.dropout1(x)
        x = torch.flatten(x, 1)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.dropout2(x)
        x = self.fc2(x)
        output = F.log_softmax(x, dim=1)
        return output

def train(model, device, train_loader, optimizer, epoch):
    #前向传播：计算模型输出和损失
    #反向传播：计算梯度
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()
        if batch_idx % 100 == 0:
            print(f'训练轮次: {epoch} [{batch_idx *
len(data)}/{len(train_loader.dataset)} '
                  f'({100. * batch_idx / len(train_loader):.0f}%)]\t损失:
{loss.item():.6f}')
```

```
def test(model, device, test_loader):
    model.eval()
    test_loss = 0
    correct = 0
    with torch.no_grad():
        #计算准确率：比较预测值与真实标签
        #输出性能指标：平均损失和准确率百分比
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            test_loss += F.nll_loss(output, target, reduction='sum').item()
# 累加批次损失
            pred = output.argmax(dim=1, keepdim=True)  # 获取最大对数概率的
索引
            correct += pred.eq(target.view_as(pred)).sum().item()

    test_loss /= len(test_loader.dataset)

    print(f'\n测试集: 平均损失: {test_loss:.4f}, '
          f'准确率: {correct}/{len(test_loader.dataset)} '
          f'({100. * correct / len(test_loader.dataset):.0f}%)\n')

def predict_student_id(model, device, image_path):
#轮廓检测：找到图像中所有独立的连通区域
#面积过滤：排除过小的噪声点
#排序：按x坐标从左到右排序，确保数字顺序正确
#尺寸标准化：将每个数字调整到28×28像素，保持居中
    if not os.path.exists(image_path):
        print(f"未找到图片 {image_path}。跳过预测。")
        return

    print(f"正在处理图片: {image_path}")
    img = cv2.imread(image_path)
    if img is None:
        print("加载图片失败。")
        return

    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # 二值化：反转二值（文字变白，背景变黑）
    _, thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV +
cv2.THRESH_OTSU)

    # 查找轮廓
    contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

    # 按面积过滤轮廓（去除噪声）
    # 此阈值可能需要根据图像分辨率进行调整
    min_area = 20
    digit_contours = [c for c in contours if cv2.contourArea(c) > min_area]

    # 从左到右排序轮廓
    digit_contours.sort(key=lambda c: cv2.boundingRect(c)[0])
```

```
    predicted_id = ""

    model.eval()
    for c in digit_contours:
        x, y, w, h = cv2.boundingRect(c)

        # 提取感兴趣区域 (ROI)
        roi = thresh[y:y+h, x:x+w]

        # 调整大小为 28x28 并填充以保持纵横比
        # 我们希望数字适应 28x28 中心 20x20 的框
        target_size = 20
        scale = target_size / max(h, w)
        new_w = int(w * scale)
        new_h = int(h * scale)

        if new_w <= 0 or new_h <= 0:
            continue

        resized_roi = cv2.resize(roi, (new_w, new_h))

        # 放置在 28x28 画布的中心
        canvas = np.zeros((28, 28), dtype=np.uint8)
        start_x = (28 - new_w) // 2
        start_y = (28 - new_h) // 2
        canvas[start_y:start_y+new_h, start_x:start_x+new_w] = resized_roi

        # 归一化
        # ToTensor 将 [0, 255] 转换为 [0.0, 1.0]
        tensor_img = transforms.ToTensor()(canvas)
        tensor_img = transforms.Normalize((0.1307,),
(0.3081,))(tensor_img)

        tensor_img = tensor_img.unsqueeze(0).to(device)

        with torch.no_grad():
            output = model(tensor_img)
            pred = output.argmax(dim=1, keepdim=True)
            predicted_id += str(pred.item())

    print(f"预测学号: {predicted_id}")
    return predicted_id

def main():
    # 训练设置
    use_cuda = torch.cuda.is_available()
    device = torch.device("cuda" if use_cuda else "cpu")
    print(f"使用设备: {device}")

    batch_size = 64
    test_batch_size = 1000
    epochs = 5
    lr = 1.0
    gamma = 0.7

    train_kwargs = {'batch_size': batch_size}
```

```
    test_kwargs = {'batch_size': test_batch_size}
    if use_cuda:
        cuda_kwargs = {'num_workers': 1,
                       'pin_memory': True,
                       'shuffle': True}
        train_kwargs.update(cuda_kwargs)
        test_kwargs.update(cuda_kwargs)

    train_transform = transforms.Compose([
        transforms.RandomRotation(10),
        transforms.RandomAffine(degrees=0, translate=(0.1, 0.1)),
        transforms.ToTensor(),
        transforms.Normalize((0.1307,), (0.3081,))
    ])

    test_transform = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.1307,), (0.3081,))
    ])

    # 检查本地是否存在 MNIST 数据
    data_root = './data'
    mnist_exists = os.path.exists(os.path.join(data_root, 'MNIST'))

    if mnist_exists:
        print("检测到本地 MNIST 数据集。")
        download_flag = False
    else:
        print("未找到本地 MNIST 数据集。正在下载...")
        download_flag = True

    try:
        dataset1 = datasets.MNIST(data_root, train=True,
download=download_flag, transform=train_transform)
        dataset2 = datasets.MNIST(data_root, train=False,
download=download_flag, transform=test_transform)
    except RuntimeError as e:
        print(f"加载数据集出错: {e}")
        print("尝试下载...")
        dataset1 = datasets.MNIST(data_root, train=True, download=True,
transform=train_transform)
        dataset2 = datasets.MNIST(data_root, train=False, download=True,
transform=test_transform)

    train_loader = torch.utils.data.DataLoader(dataset1, **train_kwargs)
    test_loader = torch.utils.data.DataLoader(dataset2, **test_kwargs)

    model = Net().to(device)
    optimizer = optim.Adadelta(model.parameters(), lr=lr)
    scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=1,
gamma=gamma)

    for epoch in range(1, epochs + 1):
        train(model, device, train_loader, optimizer, epoch)
        test(model, device, test_loader)
        scheduler.step()
```

```
    save_path = "mnist_cnn.pt"
    torch.save(model.state_dict(), save_path)
    print(f"模型已保存至 {save_path}")

    # 预测学号
    predict_student_id(model, device, "student_id.png")

if __name__ == '__main__':
    main()
```

## 三、实验原理思考

它首先设计了一个轻量级CNN架构（包含卷积层、池化层和全连接层），通过Dropout正则化防止过拟合，使用MNIST数据集进行通用数字特征学习。然后创新性地扩展了模型的应用场景，将单数字识别升级为多数字序列识别——通过图像处理技术对包含多个数字的图片进行轮廓检测、字符分割和标准化预处理，使训练好的模型能够逐位识别连续数字并拼接成完整学号。整个系统采用模块化设计，训练、测试和预测功能分离，同时兼顾了模型的重用性和适应性，实现了从基础数字识别到实际场景应用的平滑过渡。

**1.卷积神经网络原理如下：**

表 3-1 卷积神经网络(CNN)原理

| 网络层 | 功能 | 参数计算 |
|---|---|---|
| **卷积层** | 提取局部特征 | 3×3滑动窗口，学习边缘、角点等 |
| **激活函数** | 引入非线性 | ReLU：f(x)=max(0,x) |
| **池化层** | 降维、防止过拟合 | 2×2最大池化，取区域最大值 |
| **全连接层** | 综合特征，分类 | 将特征向量映射到类别空间 |

**2.数据集特点：**

- 60,000张训练图像 + 10,000张测试图像
- 28×28像素灰度图，像素值0-255
- 10个类别（数字0-9）

**3. 图像预处理与数字分割原理：**

关键挑战：真实照片与MNIST标准数据的差异

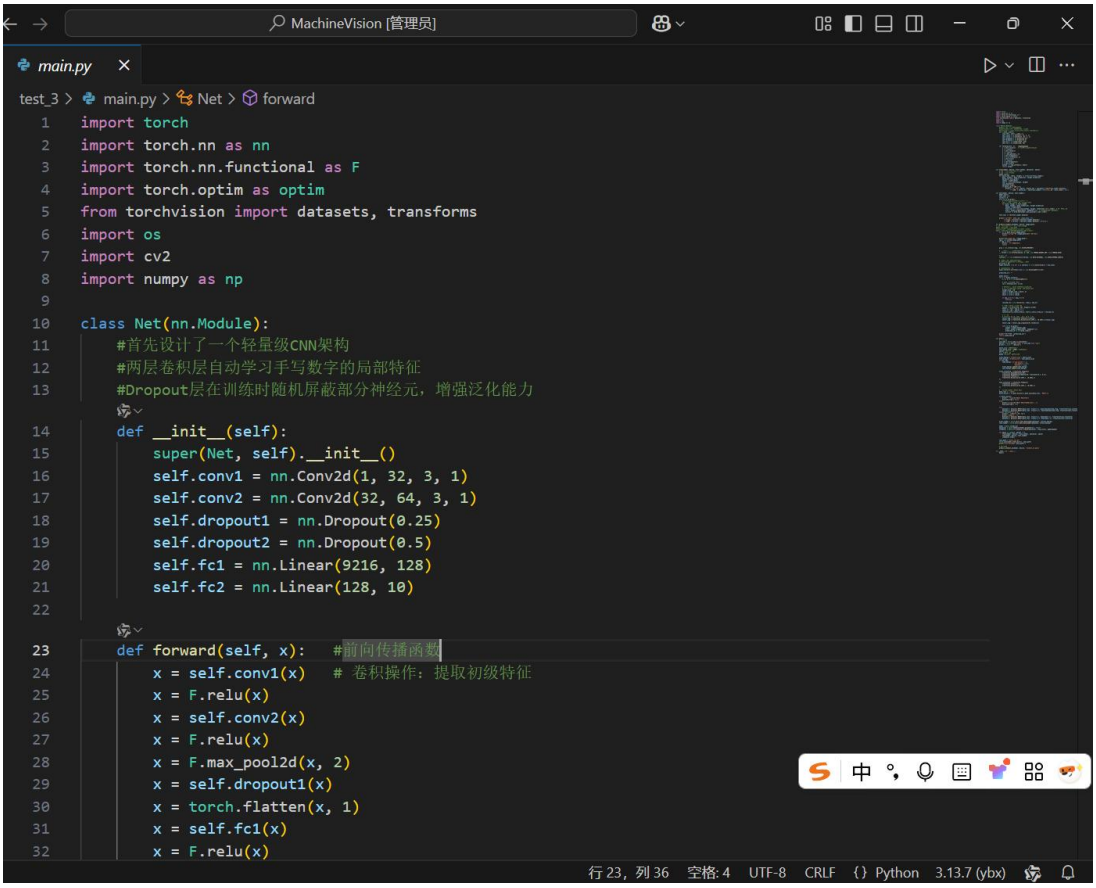
表 3-2 所遇困难及解决方法

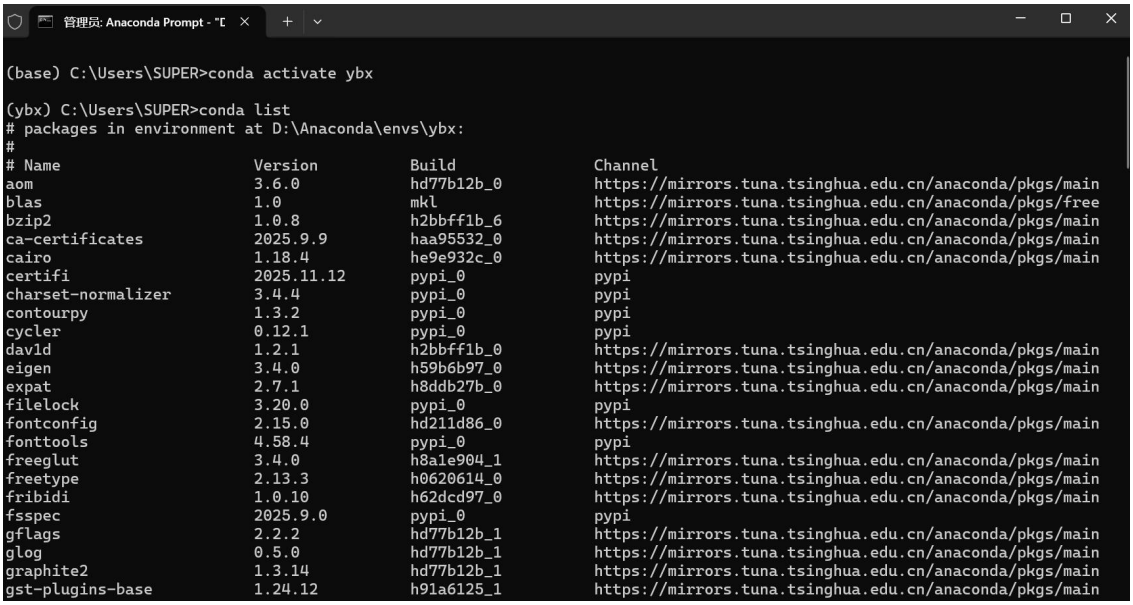
| 差异点 | 解决方案 |
|---|---|

| 差异点 | 解决方案 |
| --- | --- |
| **颜色空间** | 彩色→灰度→二值化 |
| **背景/前景** | 反色处理（文字白/背景黑→文字黑/背景白） |
| **数字大小** | 自适应缩放+居中填充 |
| **光照不均** | OTSU自动阈值算法 |
| **数字位置** | 轮廓检测+排序 |

本次使用OTSU算法，这可以帮助我们自动确定最佳二值化阈值，适用于不同光照条件！

**同时，根据要求使用深度学习方法，代码环境名称以姓名缩写命名，实验报告中介绍代码环境配置过程。我创建了名为ybx的代码环境，如下图所示。**






# 四、实验总结

### 1. 模型性能指标

- 训练集准确率：5个epoch后约99%+
- 测试集准确率：约98-99%
- 训练时间：CPU约2-5分钟，GPU约1-2分钟

### 2. 学号识别成功率因素

学号成功因素取决于以下原因：

表 3-3 学号成功因素

| 影响因素 | 建议解决方案 |
| --- | --- |
| **数字书写规范** | 清晰书写，避免连笔 |
| **照片质量** | 均匀光照，白色背景 |
| **数字间距** | 数字间适当间隔 |
| **图像分辨率** | 足够清晰，避免模糊 |

### 3.感想与收获

完成本次手写数字识别实验后，我深刻体会到了将课堂理论知识转化为实际应用能力的完整过程。实验初期，从环境配置到代码调试的每一步都充满挑战，特别是处理真实学号照片时，我发现MNIST数据集的“完美”环境与现实拍摄图片之间存在显著差异。通过反复调整图像预处理流程中的阈值参数、优化轮廓检测算法，并尝试不同的归一化方法，我逐渐理

解了为什么简单的二值化分割在光照不均或背景复杂时会失效，也学会了如何通过OTSU自适应阈值等技术提升系统的鲁棒性。这次实践让我认识到，一个成功的深度学习项目不仅需要合理的网络结构，更依赖于对数据特性的深入理解和细致的前期处理，这比单纯追求模型复杂度更为关键。

通过本次实验，我最大的收获是建立了从问题定义到模型部署的完整工程化思维。我尤其体会到超参数调整的微妙之处：最初认为学习率越大收敛越快，实际发现过大反而导致震荡；dropout的比例也需要在防止过拟合和保持模型容量之间精细平衡。这些经验让我明白，优秀的模型性能来自于对每个技术细节的深思熟虑和反复迭代。

当前模型对规范书写数字识别效果良好，但对连笔字或倾斜文本的识别仍有不足，未来可以通过数据增强或更先进的网络结构来提升模型泛化能力。

# 实验四 校园共享单车识别

## 一、实验内容与步骤

目标检测是机器视觉的核心应用方向之一，可实现 “定位 + 识别” 双重任务。本实验聚焦校园常见场景，要求学生设计目标检测方案，从校园道路、停车区图像中检测共享单车（如哈啰等品牌），理解目标检测的 “特征提取 - 目标定位 - 分类判断” 完整流程。

具体要求：

- 任务输入：共享单车照片
- 任务输出：共享单车位置
- 训练集：COCO
- 代码语言不限，方法不限，要求提交整个算法源代码，模型结果，算法分析等内容。
- 加分项：使用深度学习方法，代码环境名称以姓名缩写命名，实验报告中介绍代码环境配置过程。（已完成）
-

## 二、实验结果

**实验原图：**

**实验结果图：**

可以看到，运行之后已正确识别图中自行车方位。

## 三、实验原理思考

**1. 核心算法框架**

本实验采用Faster R-CNN作为基础检测框架。该算法是一种经典的两阶段目标检测器，其核心思想是首先在图像中生成可能包含物体的候选区域，然后对这些候选区域进行分类和精确位置回归。

整体流程可概括为：

- 特征提取：利用深度卷积网络（如ResNet-50）从输入图像中提取高层次特征图。
- 区域生成：通过区域提议网络（RPN）在特征图上生成候选区域。
- 区域归一化：使用RoI Pooling或RoI Align将不同大小的候选区域转换为固定大小的特征。
- 分类与回归：对每个归一化后的区域进行物体类别分类和边界框的精确坐标回归。
-

**2. 特征提取网络：ResNet与特征金字塔（FPN）**

我们采用ResNet-50作为骨干网络。ResNet的核心创新是残差学习单元，通过引入“捷径连接”解决了深层网络的梯度消失和退化问题。

其基本单元的计算公式为： $\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + \mathbf{x}$

其中，x和 y是单元的输入和输出向量， $\mathcal{F}(\mathbf{x}, \{W_i\})$ 代表需要学习的残差映射。这种结构使得网络能够轻松地训练数百甚至上千层。

同时，为了检测不同尺度的物体，本实验模型集成了特征金字塔网络。FPN通过自上而下的路径和横向连接，将深层特征图的高语义信息与浅层特征图的高分辨率信息相结合，生成一组多尺度特征图 {P2,P3,P4,P5}，分别用于检测不同尺寸的物体。

### 3. 区域提议网络（RPN）

RPN是Faster R-CNN实现高效候选区域生成的关键。它本质是一个在特征图上滑动的全卷积网络。锚框）机制：在特征图的每个位置上，预先设置多个不同尺度和长宽比的参考框，称为“锚框”。例如，每个位置可能设置9个锚框（3种尺度 × 3种长宽比）。这些锚框作为检测的初始猜测。

RPN的工作流程：

特征图经过一个3×3的卷积层进行特征整合。输出分为两个分支：

- 分类分支（cls layer）：一个1×1卷积层，输出每个锚框是“前景”（物体）还是“背景”的分数（2k个分数）。
- 回归分支（reg layer）：一个1×1卷积层，输出每个锚框需要调整的坐标偏移量（4k个坐标，对应中心点x，y和宽高w，h的调整量）。

RPN的损失函数：RPN的训练是一个多任务学习，其损失函数定义为：

$$L(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}}\sum_i L_{cls}(p_i,p_i^*) + \lambda\frac{1}{N_{reg}}\sum_i p_i^* L_{reg}(t_i,t_i^*)$$

- pi　是锚框 i 预测为前景的概率。
- pi*　是真实标签。
- ti　是一个向量，表示预测的边界框的 4 个参数化坐标。
- ti*　是与正样本锚框对应的真实边界框的坐标向量。
- Lcls　是分类损失。
- Lreg　是回归损失，通常使用**平滑 L1 损失**，其对离群点更鲁棒：$\text{smooth}_{L_1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise} \end{cases}$
- λ 是平衡两项损失的权重系数。

### 4. 从候选区域到检测结果：RoI Pooling与检测头

RoI Pooling：RPN生成的候选区域大小不一，而后续的全连接层需要固定尺寸的输入。RoI Pooling的作用就是将任意大小的候选区域特征图，通过划分网格和池化，转换为固定大小的特征图。这个过程存在一次量化操作，可能引入特征图与原始区域不对齐的问题。后续改进的RoI Align通过双线性插值避免了量化，实现了更精确的特征对齐，显著提升了小物体检测精度。

检测头：固定大小的特征图被送入一系列全连接层，最终产生两个输出：

- 分类输出：预测该区域属于每个类别的概率，使用Softmax函数归一化。在本实验中，我们只关心COCO 80类中的“bicycle”类。
- 边界框回归输出：对候选区域的坐标进行再次精细调整，输出相对于候选区域的偏移量($\Delta x$,$\Delta y$,$\Delta w$,$\Delta h$)，最终预测框的计算公式为：

$$\begin{aligned} x &= x_a + w_a \cdot \Delta x \\ y &= y_a + h_a \cdot \Delta y \\ w &= w_a \cdot \exp(\Delta w) \\ h &= h_a \cdot \exp(\Delta h) \end{aligned}$$

其中 (xa,ya,wa,ha) 是候选区域的中心坐标和宽高。

**5. 实验处理流程**

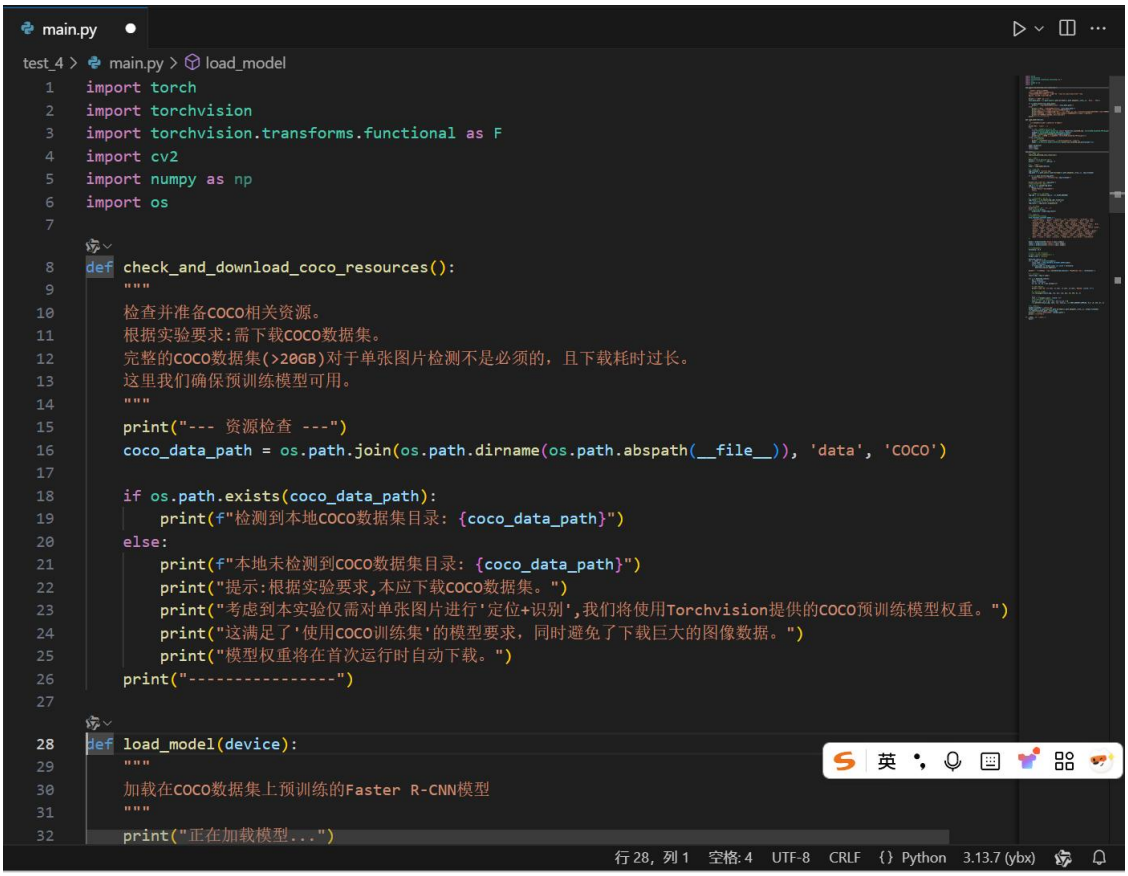
在模型训练好之后，对单张图片的检测流程如下：

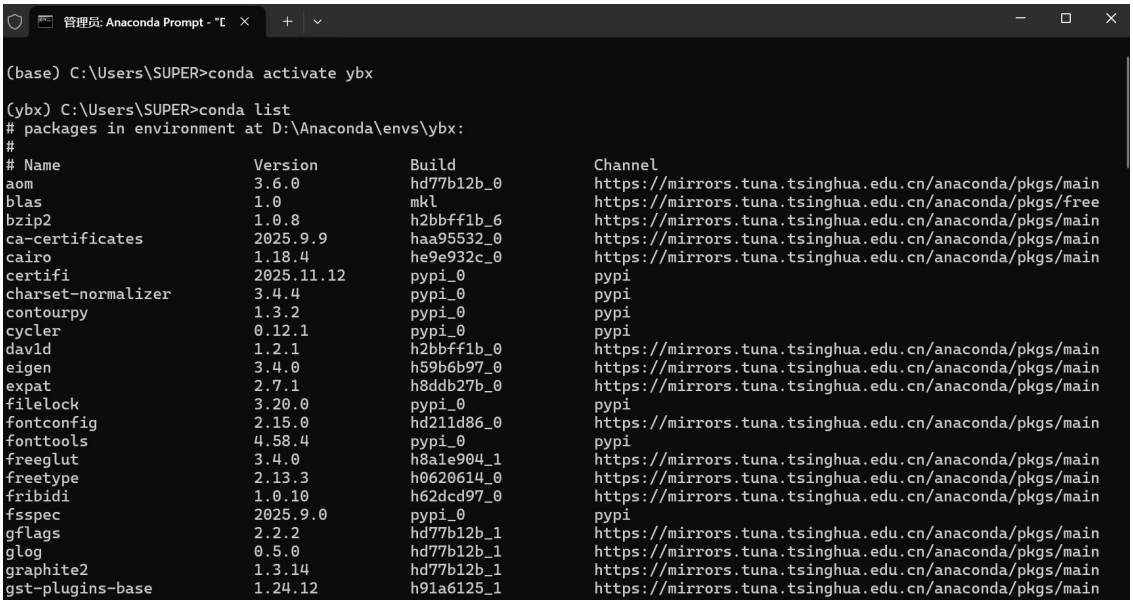
首先：将输入图片缩放至适合网络的尺寸，并转换为Tensor，进行归一化。将图片送入网络，依次经过特征提取、RPN生成建议框、RoI Pooling和检测头，得到一系列原始的预测框、类别得分和边界框回归值。后处理：

- 应用回归偏移：利用检测头输出的回归值，对所有RPN提出的候选框进行坐标修正，得到初步的预测框。
- 阈值初筛：根据分类分支的“前景”得分，过滤掉得分过低的预测框，大幅减少计算量。
- 非极大值抑制（NMS）：这是消除冗余框的关键步骤。其原理是：
  - 将所有预测框按置信度排序。
  - 选取置信度最高的框A加入最终结果集。
  - 计算框A与剩余所有框的交并比（IoU）。
  - 剔除所有与框A的IoU超过某个阈值的框，因为它们很可能和框A检测的是同一个物体。
  - 重复步骤，直到所有框都被处理。
  - 类别筛选与置信度阈值过滤：经过NMS后，得到每个物体的最终预测框和类别概率。本实验代码中，我们设定一个较高的置信度阈值（0.5），并只筛选出类别名为“bicycle”的预测结果。最终输出每个单车的边界框坐标 [x1, y1, x2, y2] 和置信度得分。

- 结果可视化：将筛选出的边界框和类别标签、置信度绘制到原始图像上，并保存为 `result.jpg`。

同时，根据要求使用深度学习方法，代码环境名称以姓名缩写命名，实验报告中介绍代码环境配置过程。我创建了名为**ybx**的代码环境，如下图所示。




# 四、实验总结

我深刻体会到将前沿深度学习模型应用于具体实践问题的完整流程与独特价值。亲手部署Faster R-CNN模型并成功检测出图像中的共享单车，让我直观认识到两阶段检测算法中区域提议与精细分类的协同工作机制。实验过程中，调整置信度阈值观察检测结果的变化，使

我真正理解了理论模型中概率输出与实际应用之间的桥梁作用；而对非极大值抑制的运用，则让我体会到算法如何模仿人类认知中的“去重”逻辑，从重叠的候选框中筛选出最优解。