课程实验一：图像滤波

实验内容：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： $\begin{matrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{matrix}$

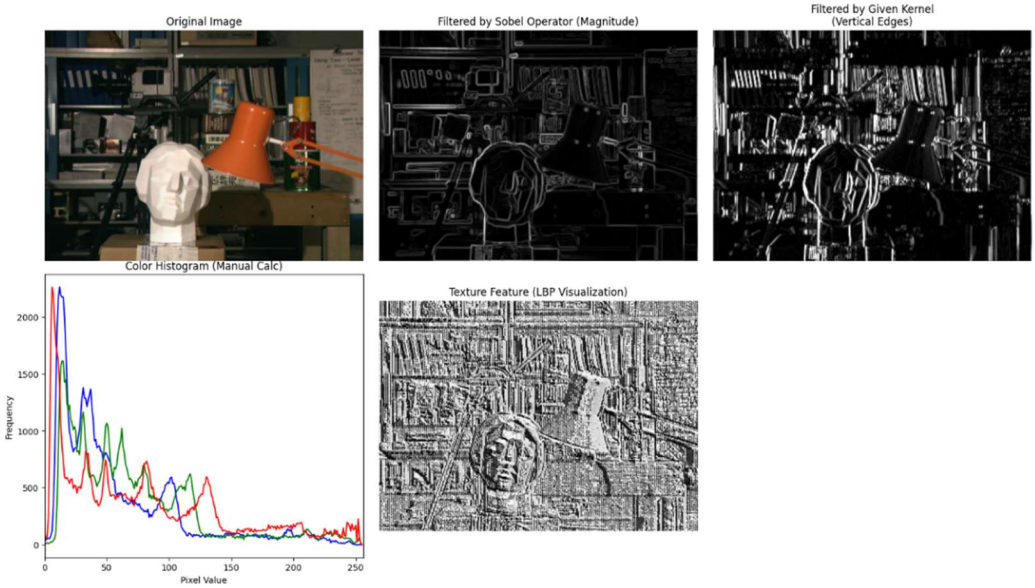
实验原理：

- 边界处理：对卷积使用零填充（zero padding），保证输出与输入同尺寸。
- Sobel：X 与 Y 卷积结果合成幅值（sqrt(gx^2 + gy^2)）常用于边缘强度度量。
- 直方图：手动统计展示原理即可；实现上可用布尔比较 + 求和替代逐像素循环以提高效率。
- LBP：对 3x3 区域与中心比较，按固定权值编码 8 位模式，即纹理描述子。
- 可视化：对结果做线性归一化或取绝对值并裁剪到 [0,255] 以便显示。

实验方法（核心代码）：

```
# 手动二维卷积（zero padding + 滑窗）
def manual_convolution(image, kernel):
    pad_h, pad_w = kernel.shape // 2
    padded = zero_pad(image, pad_h, pad_w)
    output = zeros_like(image, float)
    for each pixel (i,j) in image:
        region = padded[i:i+kh, j:j+kw]
        output[i,j] = sum(region * kernel)
    return output
# 手动彩色直方图（按通道计数）
def manual_color_histogram(image):
    hist = zeros((3,256), int)
    for ch in [B,G,R]:
        for value in 0..255:
            hist[ch, value] = count_pixels(image[:, :, ch] == value)
    return hist
# 手动 LBP（3x3 邻域）
def manual_lbp_texture(gray):
    output = zeros((h-2, w-2), uint8)
    weights = [[1,2,4],[128,0,8],[64,32,16]]
    for i in 1..h-2:
        for j in 1..w-2:
            neighborhood = gray[i-1:i+2, j-1:j+2]
            binary = (neighborhood >= gray[i,j]).astype(int)
            output[i-1,j-1] = sum(binary * weights)
    return output
```

实验效果：

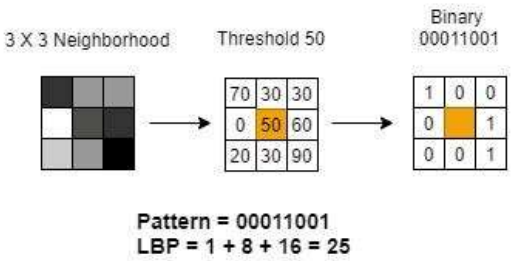


分析：

- 原始图像：用于对比的输入彩色图（已转为 RGB 显示），用于后续灰度、直方图与纹理分析的基准。
- Sobel 算子（幅值）：对灰度图分别计算 X、Y 方向卷积后取幅值，强调图像中整体的边缘与细节轮廓（高梯度处亮）。结果显示头部轮廓、台灯边缘等明显被增强，验证手工卷积实现正确。
- 给定卷积核（垂直边缘）：给定核等价于 SobelX（符号相反），因此主要响应竖直方向边缘。与 Sobel 幅值相比更强调垂直结构，背景书架竖直条纹被明显增强。
- 彩色直方图（手工统计）：三通道（B/G/R）灰度级分布。可观察到图像偏暗（低灰度处峰值显著），红绿通道在中灰值处有后续峰，反映场景的色彩与亮度分布。
- LBP 纹理可视化：对灰度图手工计算 LBP，得到的纹理图呈二值/局部模式结构，能突出细节与微小纹理（例如材质纹理、书脊细节）。输出尺寸为原图减 2（边界未计算）。

体会：

在手动实现卷积函数的过程中，深入理解了矩阵运算与边界填充的底层逻辑，以及严谨处理坐标变换对防止数组越界的重要性。利用 Sobel 算子进行的边缘检测实验，直观验证了“边缘即像素梯度剧烈突变”的数学本质。在纹理特征提取方面，通过 LBP（局部二值模式）实验发现，仅需对比 3×3 邻域像素并生成二进制编码，即可高效提取复杂的纹理信息。最后，彩色直方图分析揭示了光照环境对算法表现的显著影响：原图亮度偏低导致直方图峰值集中于低灰度区间，这说明在数据采集阶段优化光照条件，其重要性往往不亚于算法本身的改进。



课程实验二：车道线检测

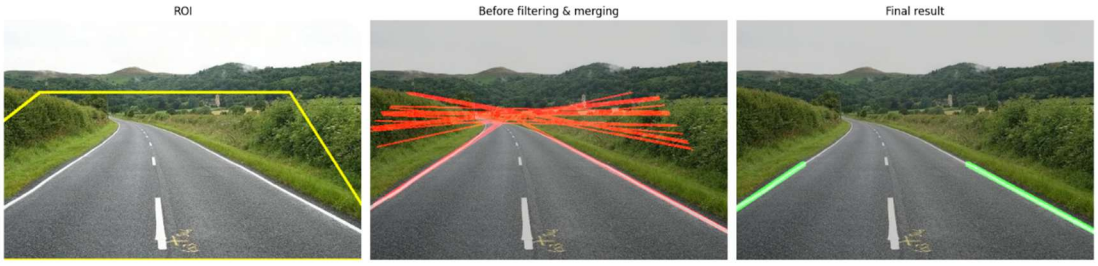
车道线检测是自动驾驶的基本模块。请使用霍夫变换实现车道线的检测。

实验原理：

- 预处理：BGR -> 灰度 -> GaussianBlur，降低噪声后再做边缘检测。
- 边缘检测：Canny（示例阈值 50/150），得到稀疏边缘图。
- ROI：用多边形掩码截取感兴趣区域，仅保留车道可能出现的部分。
- Hough：使用 HoughLinesP 提取线段（ρ=1, θ=1°，阈值/最小长度/最大间隙需调参）。
- 过滤：按线段长度与角度（示例：20°–80°）过滤噪声，分左右车道（斜率正负）。
- 合并拟合：对每侧线段用斜率与截距加权平均或拟合，生成左右两条代表线，并延伸到图像底部与某一高度（如 60%）。
- 可视化：在原图上绘制合成线，用 alpha 混合（cv2.addWeighted）；再次用 ROI 截断避免超出区域。
- 视频处理：读取视频帧，读不到时可重置帧索引实现循环播放。

实验方法（核心代码）：

```
def lane_detection_pipeline(frame):
    gray = to_gray(frame)
    blurred = gaussian_blur(gray)
    # 边缘 -> ROI
    edges = canny(blurred, 50, 150)
    masked = region_of_interest(edges, roi_polygon)
    # Hough 提取线段
    lines = hough_lines_p(masked, rho=1, theta=pi/180, thresh=70,
min_len=100, max_gap=100)
    # 过滤并合并成左右两条代表线
    merged = filter_and_merge_lines(lines, frame.shape)
    # 绘制并返回融合后的图像
    return draw_lines(frame, merged, roi_polygon)

# 制作 ROI
def region_of_interest(img, poly):
    mask = zeros_like(img)
    fillPoly(mask, poly, 255)
    return bitwise_and(img, mask)
# 筛除短线和接近水平/垂直的线（角度/长度阈值）
def filter_and_merge_lines(lines, img_shape):
    left, right = collect_lines_by_slope(lines)
    # 对每侧取斜率/截距平均或线性拟合，然后根据图像高度生成两点坐标
    return [left_line, right_line]  # 可能为 None 或 [[x1,y1,x2,y2], ...]
```



分析：

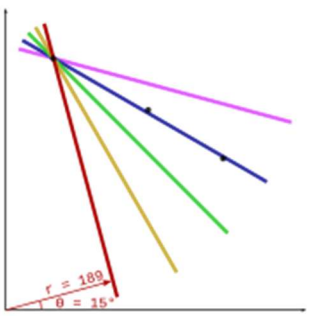
- ROI：显示选定的搜索区域（黄色多边形），把注意力集中在车道可能出现的下半部分，若多边形过大或位置不准会包含无关结构导致误检。
- Before filtering & merging：原始 Hough 检测（红色）展示了大量候选线段，包含噪声、短段和角度不合适的线，便于观察哪些线会被后续规则剔除。
- Final result：经过角度/长度过滤并平均合并后（绿色），得到更平滑的左右车道线，效果更稳健；仍可能在光照差、遮挡或弯道处失准，后续可用 RANSAC、时序平滑等方法改进。

体会：

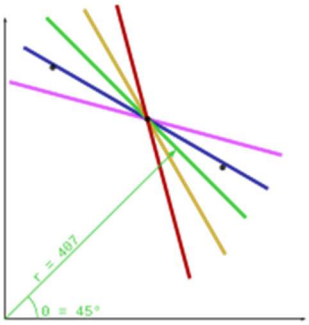
实验表明，尽管霍夫变换在理论上具备直线检测能力，但在复杂工况下极易受阴影、护栏及地面杂迹等噪声干扰。

针对上述问题，本实验对检测算法进行了多项优化。首先，在预处理阶段，将初始的矩形感兴趣区域（ROI）调整为基于透视关系的倒梯形多边形，精确聚焦于有效车道区域，大幅削减了背景干扰。其次，在直线拟合环节，通过斜率极性对左右车道线进行分类处理，并设定角度阈值剔除水平或垂直的异常线段。

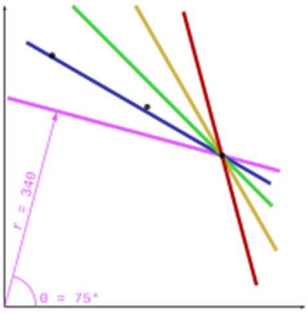
最终，通过对离散线段的聚类与平滑处理，实现了稳定、连续的车道线拟合。此次实验证明，在自动驾驶感知逻辑中，针对特定场景的启发式规则设计与精细化参数调优，对于提升算法的鲁棒性至关重要。



θ	r
15	189.0
30	282.0
45	355.7
60	407.3
75	429.4



θ	r
15	318.5
30	376.8
45	407.3
60	409.8
75	385.3



θ	r
15	419.0
30	443.6
45	438.4
60	402.9
75	340.1

课程实验三：学号识别

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：

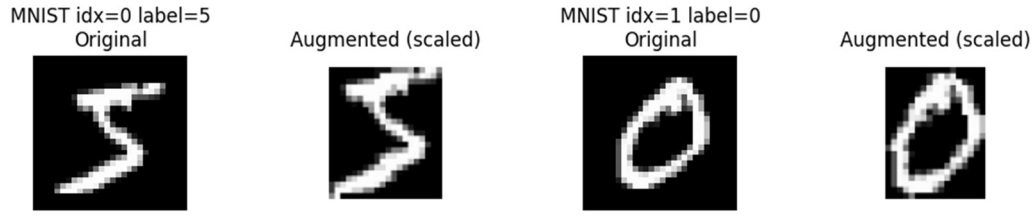
- 模型结构：轻量 CNN（Conv(1→32) → ReLU → MaxPool → Conv(32→64) → ReLU → MaxPool → FC(64*7*7→128) → FC(128→10)）。
- 训练细节：MNIST 训练，数据增强（RandomAffine 小幅旋转/平移）、CrossEntropyLoss、Adam lr=0.001、batch=64、保存 state_dict。
- 预处理（目标：把任意切出的数字变成 MNIST 格式）：灰度 →Otsu 二值化（根据背景色选择 THRESH_BINARY 或 THRESH_BINARY_INV）→ 找轮廓 → 按面积过滤并按 x 排序 → 对每个 ROI 保持长宽比缩放到 20x? 再居中填充到 28x28 → ToTensor + Normalize(0.1307,0.3081)。
- 推理要点：加载与训练相同结构模型并 load_state_dict，model.eval，with torch.no_grad()，输出取 argmax；可视化在原图画框与标注。

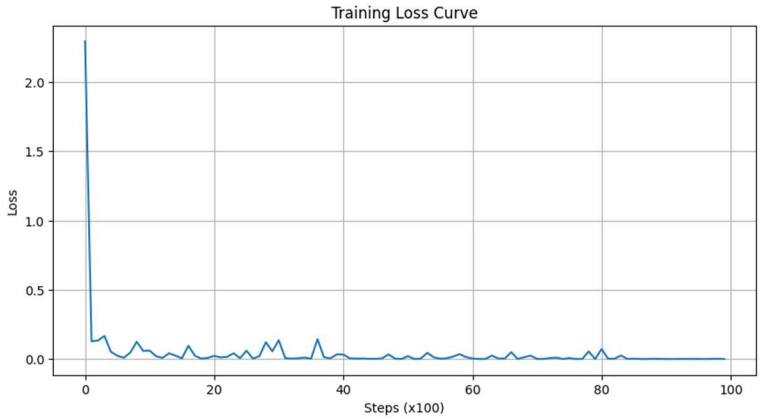
实验方法（核心代码）：

```
# 模型结构
class SimpleCNN(nn.Module):
    def __init__(self):
        # conv1: 1->32, conv2: 32->64, pool, fc1, fc2
    def forward(self, x):
        x = pool(relu(conv1(x)))
        x = pool(relu(conv2(x)))
        x = flatten(x)              # 64 * 7 * 7
        x = relu(fc1(x))
        return fc2(x)

# 训练流程
def train():
    dataset = MNIST(transform = RandomAffine + ToTensor +
Normalize(mean=0.1307,std=0.3081))
    loader = DataLoader(dataset, batch_size=64, shuffle=True)
    model = SimpleCNN().to(device)
    opt = Adam(model.params, lr=0.001)
    crit = CrossEntropyLoss()
    for epoch in epochs:
        for data, label in loader:
            out = model(data)
            loss = crit(out, label)
            loss.backward(); opt.step(); opt.zero_grad()
    torch.save(model.state_dict(), "mnist_cnn.pth")
```

```
# 预处理 + 推理
img = imread(path); gray = to_gray(img)
thresh = threshold_otsu(gray, mode = choose_inv_or_not)
contours = find_contours(thresh)
filtered = [b for b in contours if area(b) > min_area]
filtered.sort(key=lambda b: b.x)  # left-to-right

for bbox in filtered:
    roi = thresh[bbox]
    # 保持长宽比缩放到 20x?，再居中 pad 到 28x28
    tensor = preprocess_to_28x28_tensor(roi) # ToTensor + Normalize + unsqueeze(batch)
    with torch.no_grad():
        pred = argmax(model(tensor))
    draw_bbox_and_label(img, bbox, pred)

show_result(img)
```

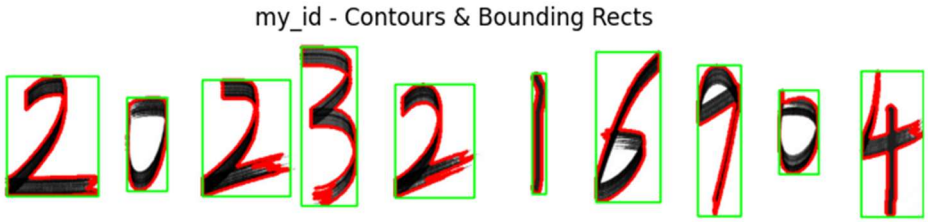
训练过程:

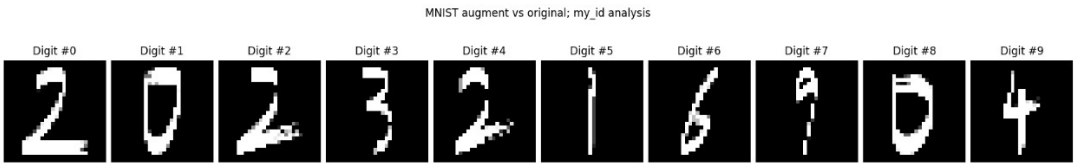




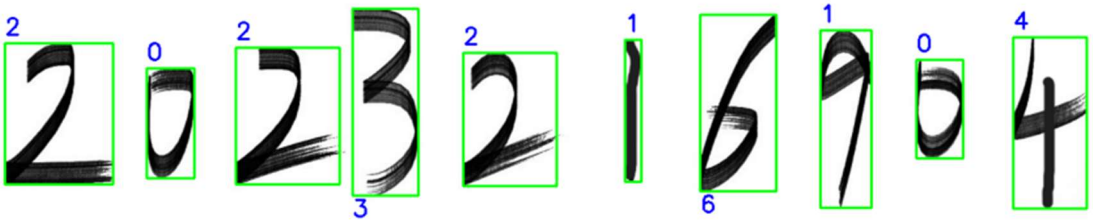
- MNIST 数据增强示例：上方展示了原始手写数字图片及其经过仿射变换（旋转、平移等）增强后的样本，增强后数字形态略有变化，有助于提升模型泛化能力。
- 训练损失曲线: 下方折线图记录了训练过程中损失值的变化, 整体呈快速下降趋势, 表明模型有效收敛，具备良好的学习能力。

推理过程：





Recognition Result: 2023216104

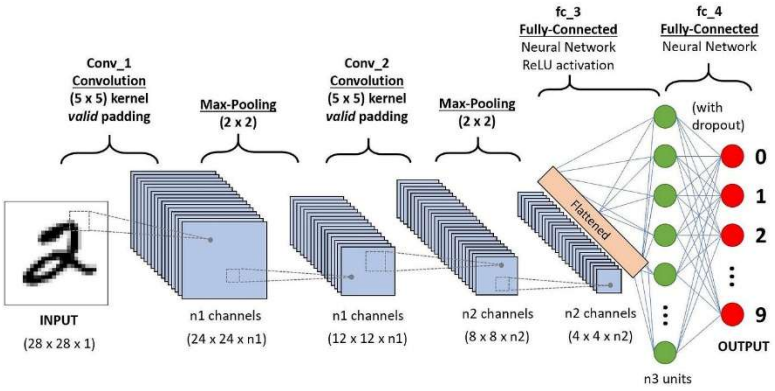


- 轮廓与边界框分析：中间部分显示了每个数字的轮廓（红色）及其外接矩形（绿色），直观反映了数字分割与定位效果。
- 数字预处理展示：下方依次给出每个分割数字经过归一化、居中处理后的 28×28 灰度图像，格式与 MNIST 数据集一致。
- 学号识别结果可视化：上方展示了模型对手写学号图片的逐位数字识别过程，最终输出完整学号字符串（2023216104）。

体会：

本实验采用双层卷积神经网络。模型在 MNIST 数据集上表现良好，但在处理手写学号时，数字“1”因笔触过细导致二值化后特征丢失，识别率较低。

针对此问题，在预处理阶段引入形态学膨胀操作以增强特征显著性。同时，改进缩放策略，采用保持长宽比缩放至 20×20 后居中填充至 28×28 的方式，避免了暴力拉伸导致的形变。实验证明，确保推理数据与训练数据在预处理逻辑上的一致性，是提升模型泛化能力与落地性能的关键。



课程实验四：校园共享单车检测

实验内容：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。

实验原理：

- 输入处理：支持本地/URL 图片；BGR(OpenCV)↔RGB；ImageNet 预处理（resize、除以 255、减均值/除方差、CHW）。
- Detectron2：加载 Faster R-CNN + FPN；在 backbone 的 FPN（p2/p3/p4/p5）层挂 forward hook，获取特征图；用 Visualizer 绘制检测结果（按类别筛选 bicycle）。
- Grad-CAM（timm resnet50）：在指定的 residual layer（layer1..layer4）挂 forward hook，前向得到激活；选定 ImageNet 中与「自行车」相关类别，反向 target logit，记录激活的梯度；按 Grad-CAM 权重（全局平均池化梯度）加权激活并 ReLU，放大到原图尺寸。
- 可视化：FPN 特征强度（通道平均→归一化）、Grad-CAM 热力图叠加、检测结果三图并排保存。
- 注意：hooks 在使用后移除；梯度计算要确保对目标 logit backward；数值稳定性做归一化/eps 防护；用 CPU/GPU 设备选择。

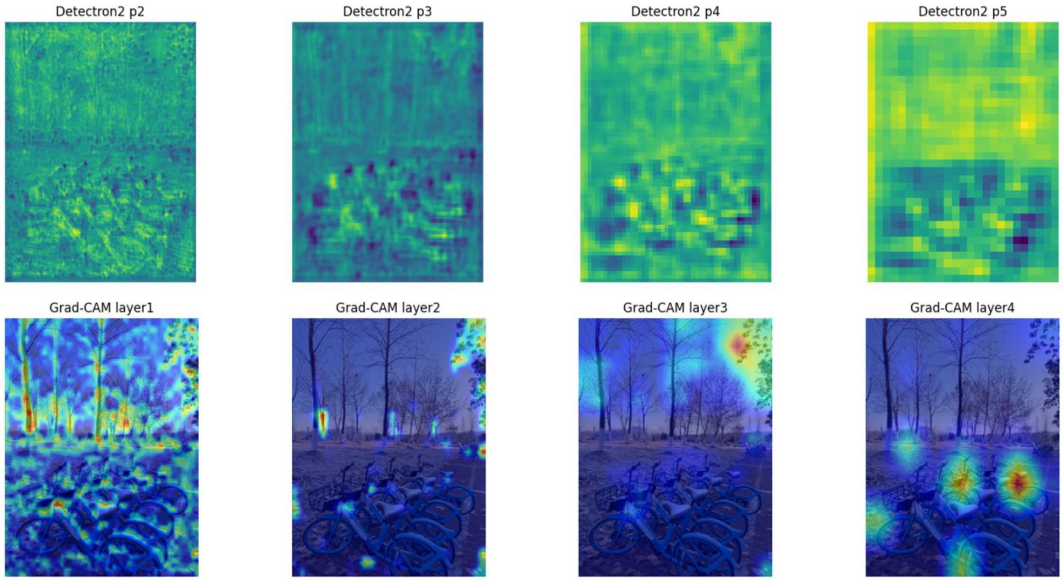
实验方法（核心代码）：

```
# Detectron2：挂钩 FPN 层并推理
def hook(module, inp, out): features['fpn']=out
    handle = predictor.model.backbone.register_forward_hook(hook)
    outputs = predictor(bgr)
    handle.remove()
    fpn_map = features['fpn'][det2_layer]  # shape CxHxW
    intensity = mean_over_channels(fpn_map)  # HxW -> 归一化
    # 只保留 bicycle 的实例并用 Visualizer 绘制
    instances = outputs["instances"].to("cpu")
    bicycle_instances = filter_by_class(instances, 'bicycle')
    det_vis =
    Visualizer(...).draw_instance_predictions(bicycle_instances)
    return intensity, det_vis

# Grad-CAM（timm + ResNet50）
def gradcam_resnet(rgb, gradcam_layer='layer4'):
    model = timm.create_model("resnet50",
    pretrained=True).eval().to(device)
    acts, grads = {}, {}

# 在激活上 register_hook 获取反向梯度
def fwd_hook(m, inp, out): acts['feat']=out
```

```
    handle_fwd = getattr(model,
    gradcam_layer).register_forward_hook(fwd_hook)
    input_t = preprocess_imagenet(rgb).to(device)
    logits = model(input_t)
    target_idx = find_imagenet_bicycle_class_index()
    target_logit = logits[0, target_idx]
    handle_bwd = acts['feat'].register_hook(lambda g:
    grads.__setitem__('grad', g))
    target_logit.backward()
    weights = grads['grad'].mean(dim=(2,3), keepdim=True)
    cam = relu((weights *
    acts['feat']).sum(dim=1)).squeeze(0).cpu().numpy()
    return cam, category_name(target_idx)
```

实现效果：

中间过程：



分析：

- **目标检测结果（上图）**：使用 Detectron2 Faster R-CNN R50-FPN 对图像进行检测，并按类别筛选 *bicycle*，仅保留自行车实例的框与标签，用于确认后续可视化关注的目标区域。

- **Detectron2 FPN 多尺度特征（p2–p5，中间一行）**：对 backbone 输出的各层特征图做通道均值并归一化展示。p2 分辨率最高，包含更多细粒度纹理与边缘；p3/p4 逐渐聚合区域结构；p5 语义更强但更粗糙，突出大范围目标与背景布局。
- **ResNet50 多层 Grad-CAM（layer1–layer4，底部一行）**：在 ImageNet 预训练 ResNet50 上对“bicycle”相关类别的 logit 反传得到 Grad-CAM 热力图。浅层（layer1/2）更偏向边缘与局部纹理响应；深层（layer3/4）更集中在与“自行车”语义相关的区域，能更清晰体现模型用于判别的关键位置。
- **对比结论**：FPN 特征图体现“多尺度表征”（从细节到语义的逐层聚合），而 Grad-CAM 体现“判别性关注”（模型为目标类别做决策时最依赖的空间区域）；两者结合可同时解释“特征如何形成”与“模型为何这样预测”。

体会：

通过 Detectron2 与 Grad-CAM 的对比实验，可以更深入地理解 CNN 中“纹理特征”与“语义特征”的演变逻辑：

低层特征图（如 P2）具有高分辨率，侧重于记录物体的几何纹理与局部细节（如自行车辐条）；而高层特征图（如 P5）经过深度抽象，呈现为低分辨率的语义响应，代表了模型对物体类别的宏观认知。

在 ResNet 架构中，浅层网络倾向于提取边缘与轮廓特征；随着深度增加，Layer 4 的热力图精确聚焦于物体最具辨识性的关键组件，如自行车的车把和车座。这表明深度学习模型是通过识别局部核心特征来执行分类决策，而非简单的像素匹配。

这种可解释性分析证明了模型决策的逻辑性。在目标检测任务中，多尺度特征融合至关重要，它实现了低层定位精度（纹理）与高层分类性能（语义）的有效平衡。