《计算机视觉》实验报告

姓名：刘远航 学号：22121883

实验 6

一. 任务 1

1、在数据集 INRIADATA 上，使用 hog+svm 实现行人检测

2、模型参数调优，提升检测效果

3、画出 roc 曲线

a) 核心代码：

clip_image(img, left, top, width=64, height=128): 这个函数用于从图像中截取一个指定区域。它接受图像以及左上角坐标(left, top)，以及可选的宽度和高度参数，返回指定区域的图像片段。

extract_hog_feature(img): 这个函数用于提取单个图像的方向梯度直方图（Histogram of Oriented Gradients，HOG）特征。

```
def clip_image(img, left, top,
                       width=64, height=128):
    return img[top:top + height, left:left + width]


def extract_hog_feature(img):
    '''
    提取单个图像 img 的 HOG 特征。
    '''
    return hog(
        img,
        orientations=9,
        pixels_per_cell=(16, 16),
        cells_per_block=(2, 2),
        block_norm='L2-Hys',
        visualize=False
    ).astype('float32')
```

read_images(pos_dir, neg_dir, neg_area_count, description): 这个函数用于从正样本文件夹和负样本文件夹中读取图像，并提取它们的 HOG 特征。它返回一个元组，包含所有图像的 HOG 特征(x)以及它们的分类(y)。

```
def read_images(pos_dir, neg_dir,
                        neg_area_count, description):
    pos_img_files = os.listdir(pos_dir)
    # 正样本文件列表
    neg_img_files = os.listdir(neg_dir)
    # 负样本文件列表
    area_width = 64   # 截取的区域宽度
    area_height = 128   # 截取的区域高度
    x = []   # 图片的 HOG 特征
    y = []   # 图片的分类
    for pos_file in tqdm(pos_img_files,
                                  desc=f'{description}正样本'):
        # 读取所有正样本
        pos_path = os.path.join(pos_dir, pos_file)
        # 正样本路径
        pos_img = imread(pos_path, as_gray=True)
        # 正样本图片
        img_height, img_width = pos_img.shape
        # 该图片的宽、高
        clip_left = (img_width - area_width) // 2
        # 截取区域的左边
        clip_top = (img_height - area_height) // 2
        # 截取区域的上边
        pos_center = clip_image(pos_img,
                                        clip_left, clip_top, area_width, area_height)
        # 截取中间部分
        hog_feature = extract_hog_feature(
            pos_center)   # 提取 HOG 特征
        x.append(hog_feature)   # 加入 HOG 向量
        y.append(1)   # 1 代表正类

    for neg_file in tqdm(neg_img_files,
                                  desc=f'{description}训练负样本'):
        # 读取所有负样本
        neg_path = os.path.join(neg_dir, neg_file)
        # 负样本路径
        neg_img = imread(neg_path, as_gray=True)
        # 负样本图片
        img_height, img_width = neg_img.shape
        # 该图片的宽、高
        left_max = img_width - area_width
        # 区域左边坐标的最大值
        top_max = img_height - area_height
        # 区域
```

```
        for _ in range(neg_area_count):
            # 随机截取 neg_area_count 个区域
            left = random.randint(0, left_max)  # 区域左边
            top = random.randint(0, top_max)  # 区域上边
            clipped_area = clip_image(neg_img,
                                      left, top, area_width, area_height)
            # 截取的区域
            hog_feature = extract_hog_feature(
                clipped_area)  # 提取 HOG 特征
            x.append(hog_feature)
            y.append(0)
    return x, y
```

train_SVM(x, y): 这个函数用于训练一个支持向量机（SVM）分类器，接受训练数据(x, y)，并返回训练好的 SVM 模型。

test_SVM(SVM, test_data, show_stats=False): 这个函数用于测试训练好的 SVM 模型在测试数据上的性能，并返回 AUC（ROC 曲线下的面积）。

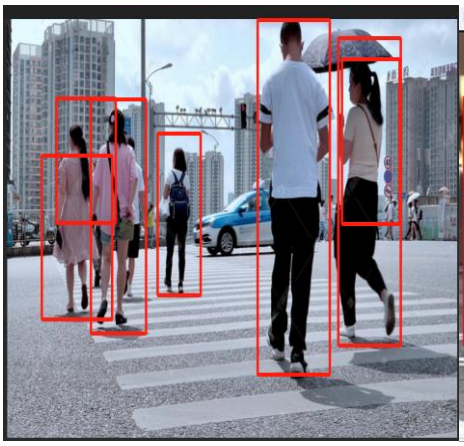
```
def test_SVM(SVM, test_data, show_stats=False):
    hog_features = test_data[0]  # 测试数据的 HOG 特征
    labels = test_data[1]  # 数据标签（0=不是人，1=是人）
    labels = test_data[1].ravel()  # 将标签展平为一维数组
    prob = SVM.predict_proba(hog_features)[:, 1]
    if show_stats:
        # 下面将 prob 和 labels 按 prob 的降序排序
        sorted_indices = np.argsort(
            prob, kind="mergesort")[::-1]
        labels = labels[sorted_indices]
        prob = prob[sorted_indices]
        distinct_value_indices = np.where(np.diff(prob))[0]
        # prob 中不同值第一次出现的下标
        threshold_idxs = np.r_[
            distinct_value_indices, labels.size - 1]
        # 阈值的下标，在末尾增加了最后一个样本的下标
        tps = np.cumsum(labels)[threshold_idxs]
        # 不同概率阈值对应的真正例数。
        # 注意现在已经按 prob 的降序排序，
        # 这种写法正确的原因是：在数组某一位置前的概率
        # 一定大于阈值，在此之后的概率一定小于阈值，
        # 所以真正例数就是在这一位置之前的正样本数。
        fps = 1 + threshold_idxs - tps
        # 不同概率阈值对应的假正例数。
        # threshold_idxs 存储的是下标，
        # 加一后变成个数，
        # 再减去真正例数就是假正例数。
        num_positive = tps[-1]
```

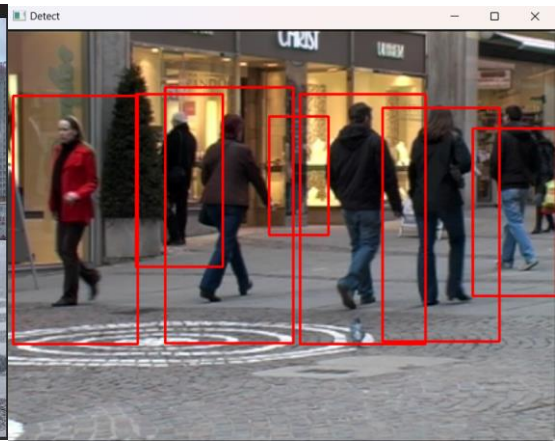
```
    # tps 的最后一项就是 labels 的和，
    # 因此代表正例的个数。
    recall = tps / num_positive
    # 查全率就是在所有正例中查出了多少真正例。
    miss = 1 - recall  # 计算 miss
    num_negative = fps[-1]  # 负例个数
    fpr = fps / num_negative
    # 假阳性率（false positive rate）
    plt.plot(miss, fpr, color='red')
    plt.xlabel('False Positive Rate')
    plt.ylabel('Miss Rate')
    plt.title('Miss Rate - '
              'False Positive Rate Curve')
    plt.show()
AUC = metrics.roc_auc_score(labels, prob)
return AUC
```

训练好的模型会生成一个文件，行人检测可以直接加载这个文件。

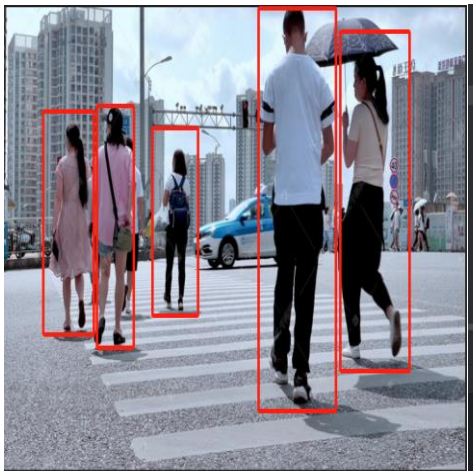
b) 实验结果截图

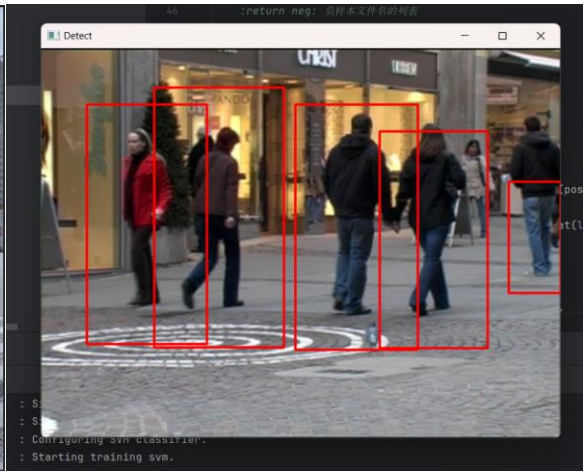
下面是行人检测图片：





使用非极大值抑制后：





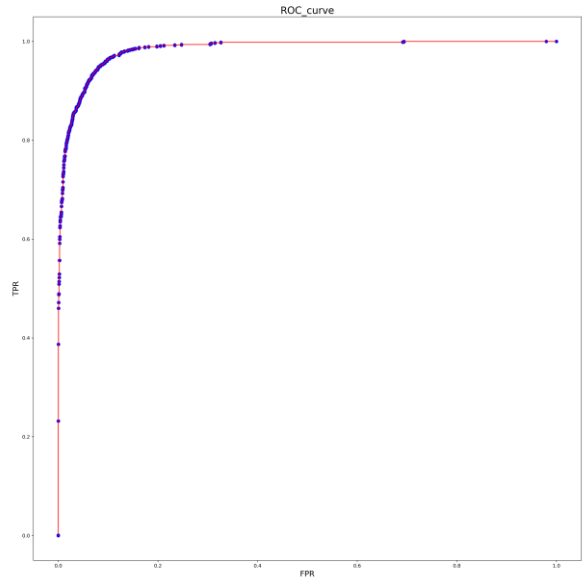
通过调节阈值大小，得到的准确率也不同，经过一系列测试后，发现阈值为 0.99 时，准确率越大。

```
训练训练负样本: 100%|██████████| 1218/1218 [00:22<00:00, 55.19it/s]
training data hog extraction done
测试正样本: 100%|██████████| 288/288 [00:05<00:00, 52.27it/s]
测试训练负样本: 100%|██████████| 453/453 [00:08<00:00, 53.15it/s]
test data hog extraction done
SVM training done, cost 67.23s.
AUC=0.81974184.
```

```
训练训练负样本: 100%|██████████| 1218/1218 [00:21<00:00, 55.79it/s]
training data hog extraction done
测试正样本: 100%|██████████| 288/288 [00:05<00:00, 52.51it/s]
测试训练负样本: 100%|██████████| 453/453 [00:08<00:00, 53.92it/s]
test data hog extraction done
SVM training done, cost 67.99s.
AUC=0.81593160.
```

```
测试正样本: 100%|██████████| 288/288 [00:05<00:00, 52.56it/s]
测试训练负样本: 100%|██████████| 453/453 [00:08<00:00, 52.55it/s]
test data hog extraction done
SVM training done, cost 65.55s.
AUC=0.81421542.
```

ROC 曲线：



c) 实验小结

从这个实验中学到了行人检测的下几个步骤：

数据准备：

从正样本和负样本文件夹中读取图像数据。对每个图像进行预处理，例如截取中心区域，并提取其 HOG 特征作为训练数据。

模型训练：

使用支持向量机（SVM）作为分类器，利用提取的训练数据进行训练。训练过程中，对 HOG 特征进行标准化和归一化，以提高模型性能。

模型评估：

使用测试数据评估训练好的模型性能，通常采用 ROC 曲线和 AUC 值进行评估。在评估过程中，可能会调整模型的超参数以提高性能。

行人检测：

使用训练好的模型对新的图像进行行人检测。对图像中的不同尺度的窗口进行滑动，提取每个窗口的 HOG 特征，并使用模型进行分类。通过非极大值抑制（NMS）算法来移除重叠的检测框，保留最具置信度的行人边界框。

结果可视化：

将检测到的行人边界框绘制到图像上，以便直观地展示行人检测结果。这个实现过程涵盖了从数据准备到模型训练再到行人检测的全流程，通过 HOG 特征和 SVM 分类器的组合，实现了一个基本的行人检测器。