**《计算机视觉》实验报告**

**姓名：刘远航 学号：22121883**

实验6

1. **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**

* 1. **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。

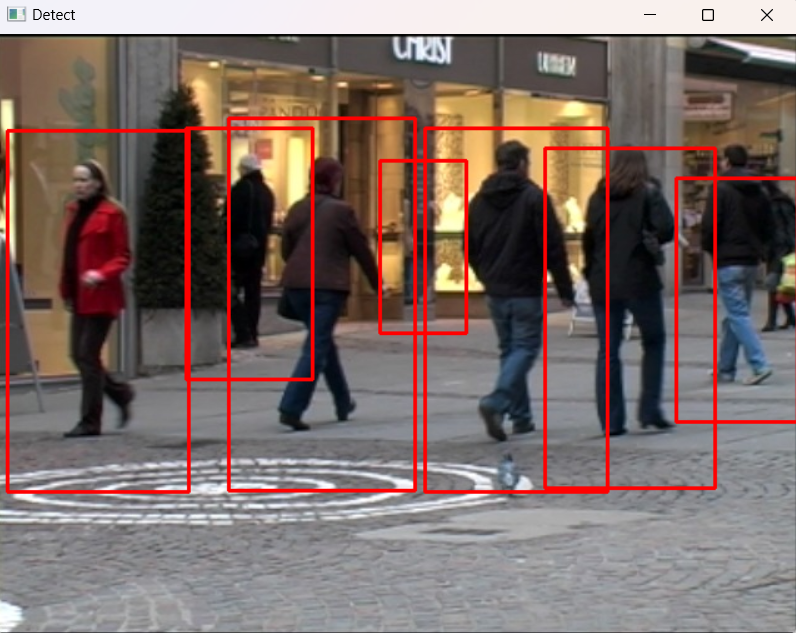
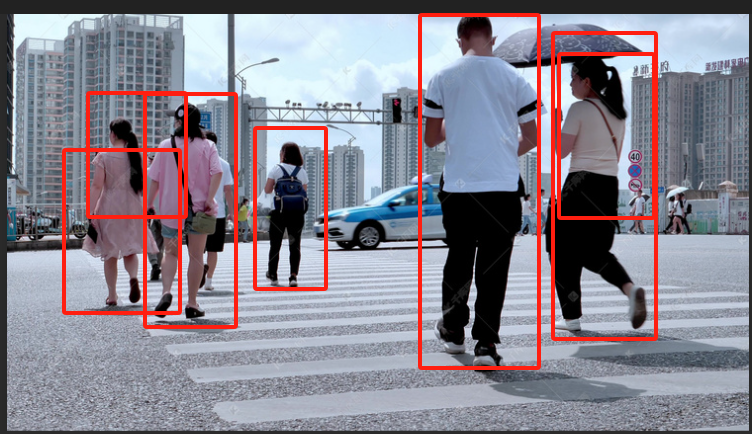
test\_SVM(SVM, test\_data, show\_stats=False): 这个函数用于测试训练好的SVM模型在测试数据上的性能，并返回AUC（ROC曲线下的面积）。

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

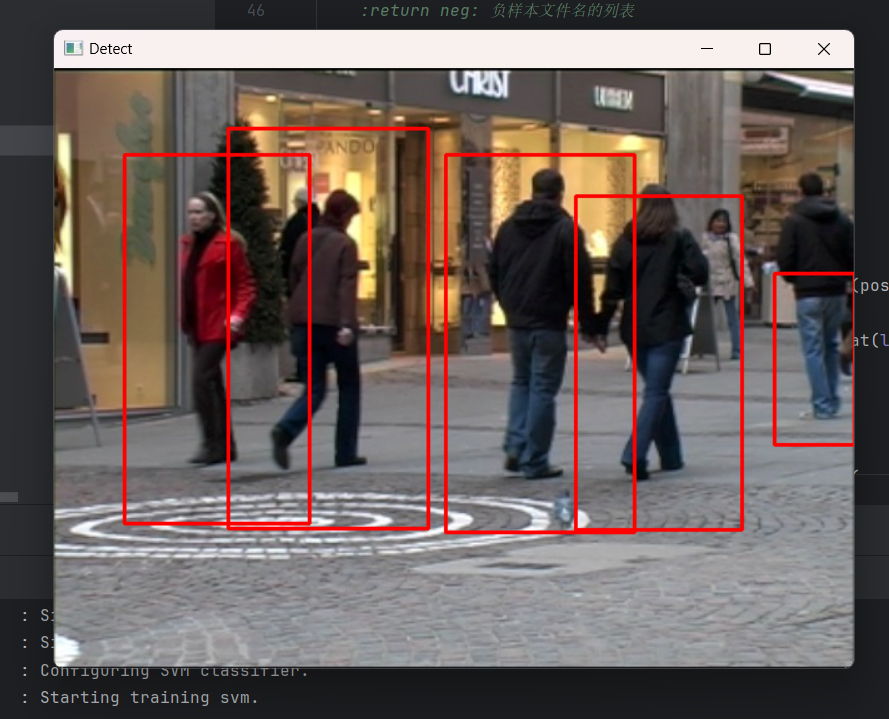
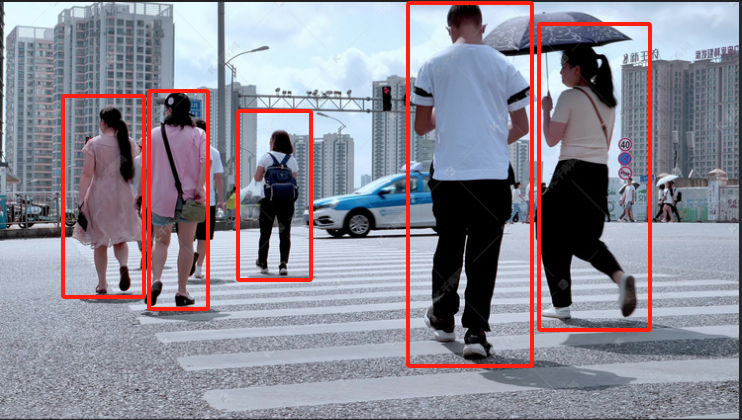
训练好的模型会生成一个文件，行人检测可以直接加载这个文件。

* 1. **实验结果截图**

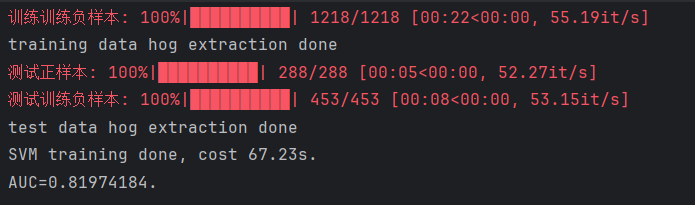
下面是行人检测图片：

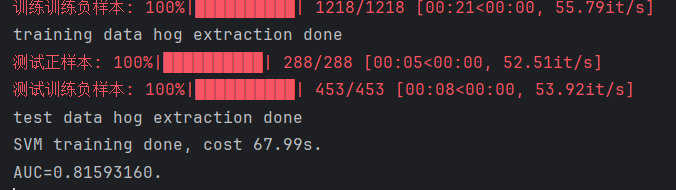


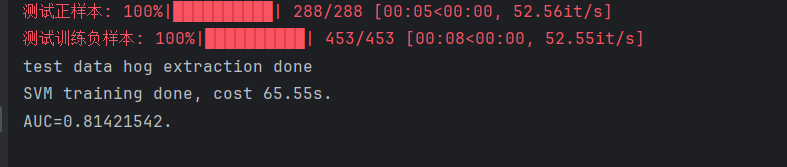
使用非极大值抑制后：



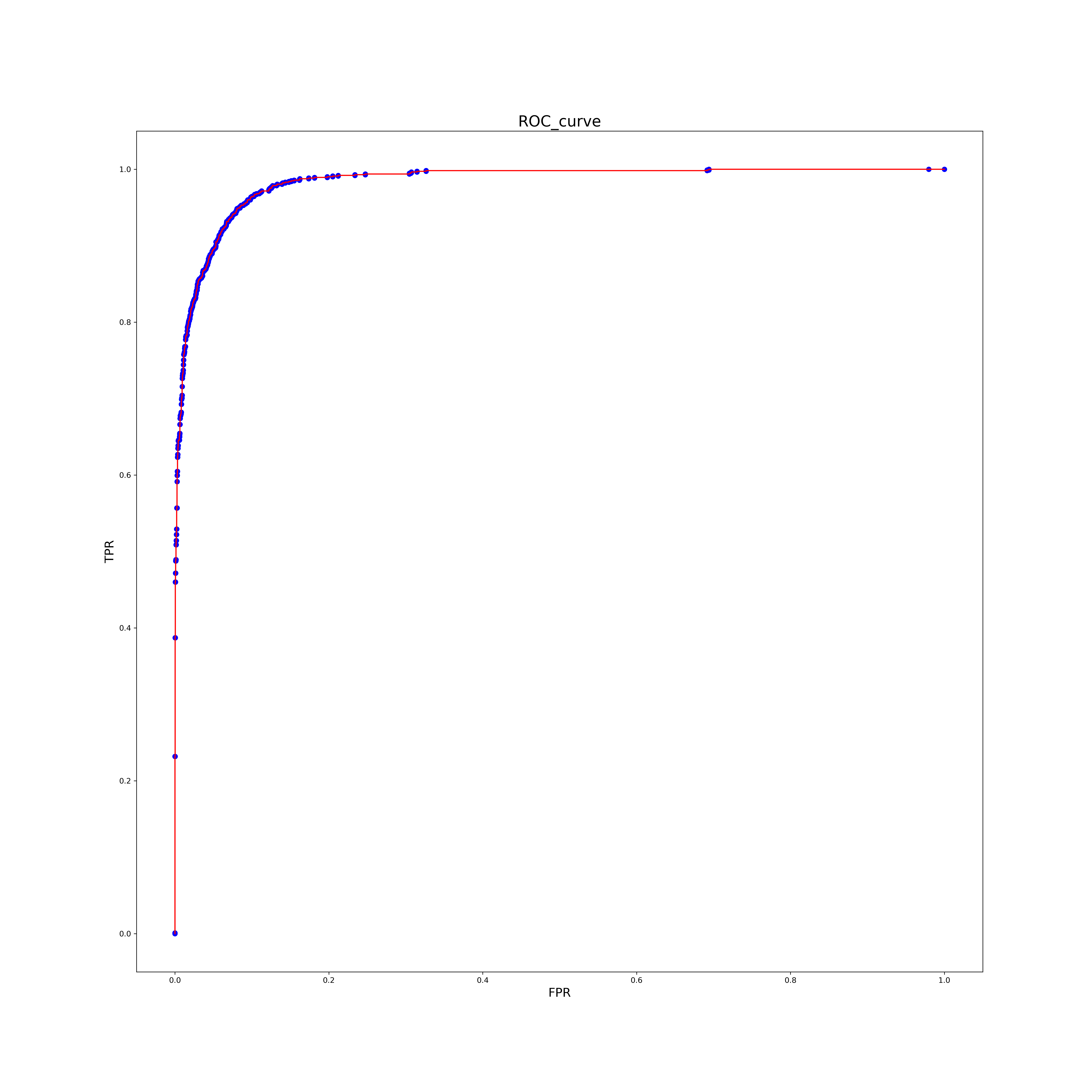
通过调节阈值大小，得到的准确率也不同，经过一系列测试后，发现阈值为0.99时，准确率越大。







ROC曲线：



* 1. **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。