电 子 科 技 大 学

实 验 报 告

学生姓名：刘文晨　　学 号：2018080901006　　指导教师：沈复民, 徐行

一、实验项目名称：

基于滑窗的人脸检测（Face Detection with a Sliding Window）

二、实验原理：

1. HOG 特征

方向梯度直方图（Histogram of Oriented Gradient, HOG）特征是一种在计算机视觉和图像处理中用来进行物体检测的特征描述子。HOG 特征通过计算和统计图像局部区域的梯度方向直方图来构成特征。

HOG 特征的提取与计算步骤：

1) 色彩与伽马归一化

   为了减少光照因素的影响，首先需要将整个图像进行规范化（归一化）。在图像的纹理强度中，局部的表层曝光贡献的比重较大，所以，这种压缩处理能够有效地降低图像局部的阴影和光照变化。

2) 计算图像梯度

   对于灰度图像，一般为了去除噪点，所以会先利用离散高斯平滑模板进行平滑，再计算图像横坐标和纵坐标方向的梯度，并据此计算每个像素位置的梯度方向值；求导操作不仅能够捕获轮廓，人影和一些纹理信息，还能进一步弱化光照的影响。

3) 构建方向的直方图

   细胞单元中的每一个像素点都为某个基于方向的直方图

通道投票。投票是采取加权投票的方式，即每一票都是带有权值的，这个权值是根据该像素点的梯度幅度计算出来，可以采用幅值本身或者它的函数来表示这个权值。

4) 将细胞单元组合成大的区间

由于局部光照的变化以及前景-背景对比度的变化，使得梯度强度的变化范围非常大。这就需要对梯度强度做归一化。归一化能够进一步地对光照、阴影和边缘进行压缩。把各个细胞单元组合成大的、空间上连通的区间。这样，HOG 描述符就变成了由各区间所有细胞单元的直方图成分所组成的一个向量。这些区间是互有重叠的，这意味着每一个细胞单元的输出都多次作用于最终的描述器。

三、实验目的：

基于 HOG 和多尺度滑动窗口实现人检测算法。

四、实验内容：

1. 了解并学习基于滑窗的人脸检测的知识和操作。
2. 利用给定的训练集与测试集图片，获得积极（人脸）特征和随机消极（非人脸）特征，训练一个线性分类器，并使用分类器在多个尺度上对数百万个滑动窗口进行分类。

五、实验步骤：

1. 获取积极（人脸）特征

完善“student.py”文件中的 get_positive_features ()函数，加载许多人脸图片作为正训练的例子，从中提取 HOG 特征。经过修改完善后的代码如下：

```
def get_positive_features(train_path_pos, feature_params):
    """
    This function should return all positive training examples (faces) from
```

```python
    36x36 images in 'train_path_pos'. Each face should be converted into a
    HoG template according to 'feature_params'.
    Useful functions:
    -   vlfeat.hog.hog(im, cell_size): computes HoG features
    Args:
    -   train_path_pos: (string) This directory contains 36x36 face images
    -   feature_params: dictionary of HoG feature computation parameters.
        You can include various parameters in it. Two defaults are:
            -   template_size: (default 36) The number of pixels spanned by
            each train/test template.
            -   hog_cell_size: (default 6) The number of pixels in each HoG
            cell. template size should be evenly divisible by hog_cell_size.
            Smaller HoG cell sizes tend to work better, but they make things
            slower because the feature dimensionality increases and more
            importantly the step size of the classifier decreases at test time
            (although you don't have to make the detector step size equal a
            single HoG cell).
    Returns:
    -   feats: N x D matrix where N is the number of faces and D is the template
            dimensionality, which would be (feature_params['template_size'] /
            feature_params['hog_cell_size'])^2 * 31 if you're using the default
            hog parameters.
    """
    # params for HOG computation
    win_size = feature_params.get('template_size', 36)
    cell_size = feature_params.get('hog_cell_size', 6)

    positive_files = glob(osp.join(train_path_pos, '*.jpg'))

    import time
    start_time = time.time()

    n_cell = np.ceil(win_size/cell_size).astype('int')
    feats = np.random.rand(len(positive_files)*2, n_cell*n_cell*31)

    for i, im_pth in enumerate(positive_files):
        im = load_image_gray(im_pth)

        #add noise to image
        noise = np.empty(im.shape, np.uint8)
        cv2.randu(noise,(0),(20))
        im_n = im + noise

        #change image contrast
        #contrast   = 1
        #brightness = 0.5
        #im_l = cv2.addWeighted(im, contrast, im, 0, brightness)

        #extract feature
        feats[i] = (vlfeat.hog.hog(im, cell_size)).flatten()
        feats[i+len(positive_files)] = (vlfeat.hog.hog(im_n, cell_size)).flatten()
        #feats[i+len(positive_files)*2] = (vlfeat.hog.hog(im_l, cell_size)).flatten()

    #debug print
    print(feats.shape)
    elapsed_time = time.time() - start_time
    print(" Elaspsed time: " + str(elapsed_time) + " seconds")

    return feats
```

2. 获取随机消极（非人脸）特征

完善“student.py”文件中的 get_random_negative_features ()函数，从不包含人脸的场景中随机抽样负面示例并将其转换为 HOG 特征，并尝试在多个尺度上抽样随机负面示例(随机裁剪图像)以获得最佳性能。经过修改完善后的代码如下：

```python
def get_random_negative_features(non_face_scn_path, feature_params, num_samples):
    """
    This function should return negative training examples (non-faces) from any
    images in 'non_face_scn_path'. Images should be loaded in grayscale because
    the positive training data is only available in grayscale (use
    load_image_gray()).
    Useful functions:
    -   vlfeat.hog.hog(im, cell_size): computes HoG features
    Args:
```

```python
    -   non_face_scn_path: string. This directory contains many images which
            have no faces in them.
    -   feature_params: dictionary of HoG feature computation parameters. See
            the documentation for get_positive_features() for more information.
    -   num_samples: number of negatives to be mined. It is not important for
            the function to find exactly 'num_samples' non-face features. For
            example, you might try to sample some number from each image, but
            some images might be too small to find enough.
    Returns:
    -   N x D matrix where N is the number of non-faces and D is the feature
            dimensionality, which would be (feature_params['template_size'] /
            feature_params['hog_cell_size'])^2 * 31 if you're using the default
            hog parameters.
    """
    # params for HOG computation
    win_size = feature_params.get('template_size', 36)
    cell_size = feature_params.get('hog_cell_size', 6)

    negative_files = glob(osp.join(non_face_scn_path, '*.jpg'))

    import random
    import time
    start_time = time.time()

    n_cell = np.ceil(win_size/cell_size).astype('int')
    feats = np.random.rand(len(negative_files), n_cell*n_cell*31)

    feats  = np.empty((0, n_cell*n_cell*31))

    num_im    = int(num_samples/len(negative_files))*2

    #go through all images
    for im_pth in negative_files:

        im = load_image_gray(im_pth)

        f_im = np.empty((0, n_cell*n_cell*31))

        #randomly scale image
        rs   = random.randint(70,100)/100
        im_s = cv2.resize(im, None, fx = rs , fy = rs, interpolation = cv2.INTER_AREA)

        #spread out data collection
        step_size = win_size

        #setup for sliding crop window
        h, w  = im_s.shape
        x_max = w - win_size
        y_max = h - win_size

        cur_x = 0
        #sliding window
        while (cur_x < x_max):
            cur_y  = 0
            while (cur_y < y_max):
                #crop "winsize X winsize" scenes from images
                im_c = im_s[cur_y:cur_y+win_size, cur_x:cur_x+win_size]

                #extract hog feature
                f  = vlfeat.hog.hog(im_c, cell_size)
                f_im = np.vstack((f_im, f.flatten()))

                cur_y = cur_y + step_size

            #end traversing image height
            cur_x = cur_x + step_size
        #end traversing image width

        feats = np.vstack((feats,f_im))

    #random num_samples
    if(feats.shape[0] > num_samples):
        shuffle(feats)
        feats = feats[:num_samples]


    #debug print
    print(feats.shape)
    elapsed_time = time.time() - start_time
    print(" Elaspsed time: " + str(elapsed_time) + " seconds")

    return feats
```

完善“student.py”文件中的 mine_hard_negs ()函数，这个函数非常类似于 get_random_negative_features ()函数。唯一的区别是，不是返回所有提取的特征，而是只返回具有假阳性预测的特征。经过修改完善后的代码如下：

```python
def mine_hard_negs(non_face_scn_path, svm, feature_params):
    """
    This function is pretty similar to get_random_negative_features(). The only
    difference is that instead of returning all the extracted features, you only
    return the features with false-positive prediction.
    Useful functions:
    -   vlfeat.hog.hog(im, cell_size): computes HoG features
    -   svm.predict(feat): predict features
    Args:
    -   non_face_scn_path: string. This directory contains many images which
            have no faces in them.
    -   feature_params: dictionary of HoG feature computation parameters. See
            the documentation for get_positive_features() for more information.
    -   svm: LinearSVC object
    Returns:
    -   N x D matrix where N is the number of non-faces which are
            false-positive and D is the feature dimensionality.
    """

    # params for HOG computation
    win_size = feature_params.get('template_size', 36)
    cell_size = feature_params.get('hog_cell_size', 6)

    negative_files = glob(osp.join(non_face_scn_path, '*.jpg'))

    import random
    import time
    start_time = time.time()

    n_cell = np.ceil(win_size/cell_size).astype('int')
    feats = np.random.rand(len(negative_files), n_cell*n_cell*31)

    feats  = np.empty((0, n_cell*n_cell*31))

    decision_thres = 0
    step_size  = win_size

    for im_pth in negative_files:

        im = load_image_gray(im_pth)

        #randomly scale image
        rs   = random.randint(50,100)/100 #random.uniform(0.5,1)
        im_s = cv2.resize(im, None, fx = rs , fy = rs, interpolation = cv2.INTER_AREA)

        #spread out data collection
        step_size = win_size

        #setup for sliding crop window
        h, w  = im_s.shape
        x_max = w - win_size
        y_max = h - win_size

        cur_x = 0
        #sliding window
        while (cur_x < x_max):
            cur_y  = 0
            while (cur_y < y_max):
                #crop "winsize X winsize" scenes from images
                im_c = im_s[cur_y:cur_y+win_size, cur_x:cur_x+win_size]

                #extract hog feature
                f  = vlfeat.hog.hog(im_c, cell_size)
                f  = f.flatten()
                fp = f.reshape(1,-1)

                if(svm.predict(fp) > decision_thres):
                    feats = np.vstack((feats, f))

                cur_y = cur_y + step_size
            #end traversing image height
            cur_x = cur_x + step_size
        #end traversing image width


    #debug print
    print(feats.shape)
    elapsed_time = time.time() - start_time
    print(" Elaspsed time: " + str(elapsed_time) + " seconds")

    return feats
```

3. 训练分类器

完善“student.py”文件中的 train_classifier ()函数，标记正特征 1，负特征-1，并使用它们来训练一个线性 SVM 分类器。经过修改完善后的代码如下：

```python
def train_classifier(features_pos, features_neg, C):
    """
    This function trains a linear SVM classifier on the positive and negative
    features obtained from the previous steps. We fit a model to the features
    and return the svm object.
    Args:
    -   features_pos: N X D array. This contains an array of positive features
            extracted from get_positive_feats().
    -   features_neg: M X D array. This contains an array of negative features
            extracted from get_negative_feats().
    Returns:
    -   svm: LinearSVC object. This returns a SVM classifier object trained
            on the positive and negative features.
    """

    import time
    start_time = time.time()

    #combine pos/neg features to train on
    train_im_feats = np.vstack((features_pos,features_neg))
    train_labels   = np.hstack((np.ones(features_pos.shape[0]), -np.ones(features_neg.shape[0])))

    #train linear svm
    svm = LinearSVC(random_state=0, tol=1e-3, loss='hinge', C=C, max_iter=10000)
    svm.fit(train_im_feats, train_labels)

    #debug print
    elapsed_time = time.time() - start_time
    print(" Elaspsed time: " + str(elapsed_time) + " seconds")

    return svm
```

4. 运行分类器

完善“student.py”文件中的 run_detector ()函数，在测试集中运行 SVM 分类器。经过修改完善后的代码如下：

```python
def run_detector(test_scn_path, svm, feature_params, verbose=False):
    """
    This function returns detections on all of the images in a given path. You
    will want to use non-maximum suppression on your detections or your
    performance will be poor (the evaluation counts a duplicate detection as
    wrong). The non-maximum suppression is done on a per-image basis. The
    starter code includes a call to a provided non-max suppression function.
    The placeholder version of this code will return random bounding boxes in
    each test image. It will even do non-maximum suppression on the random
    bounding boxes to give you an example of how to call the function.
    Your actual code should convert each test image to HoG feature space with
    a _single_ call to vlfeat.hog.hog() for each scale. Then step over the HoG
    cells, taking groups of cells that are the same size as your learned
    template, and classifying them. If the classification is above some
    confidence, keep the detection and then pass all the detections for an
    image to non-maximum suppression. For your initial debugging, you can
    operate only at a single scale and you can skip calling non-maximum
    suppression. Err on the side of having a low confidence threshold (even
    less than zero) to achieve high enough recall.
    Args:
    -   test_scn_path: (string) This directory contains images which may or
            may not have faces in them. This function should work for the
            MIT+CMU test set but also for any other images (e.g. class photos).
    -   svm: A trained sklearn.svm.LinearSVC object
    -   feature_params: dictionary of HoG feature computation parameters.
        You can include various parameters in it. Two defaults are:
            -   template_size: (default 36) The number of pixels spanned by
            each train/test template.
            -   hog_cell_size: (default 6) The number of pixels in each HoG
            cell. template size should be evenly divisible by hog_cell_size.
            Smaller HoG cell sizes tend to work better, but they make things
            slowerbecause the feature dimensionality increases and more
            importantly the step size of the classifier decreases at test time.
    -   verbose: prints out debug information if True
    Returns:
    -   bboxes: N x 4 numpy array. N is the number of detections.
            bboxes(i,:) is [x_min, y_min, x_max, y_max] for detection i.
    -   confidences: (N, ) size numpy array. confidences(i) is the real-valued
            confidence of detection i.
    -   image_ids: List with N elements. image_ids[i] is the image file name
            for detection i. (not the full path, just 'albert.jpg')
    """
    im_filenames = sorted(glob(osp.join(test_scn_path, '*.jpg')))
    bboxes = np.empty((0, 4))
    confidences = np.empty(0)
    image_ids = []

    # number of top detections to feed to NMS
    topk = 40

    # params for HOG computation
    win_size = feature_params.get('template_size', 36)
    cell_size = feature_params.get('hog_cell_size', 6)
    scale_factor = feature_params.get('scale_factor', 0.65)
    template_size = int(win_size / cell_size)

    import time
    start_time = time.time()

    for idx, im_filename in enumerate(im_filenames):
        print('Detecting faces in {:s}'.format(im_filename))
        im = load_image_gray(im_filename)
        im_id = osp.split(im_filename)[-1]
        im_shape = im.shape
        # create scale space HOG pyramid and return scores for prediction

        # cur_x_min = (np.random.rand(15,1) * im_shape[1]).astype('int')
        # cur_y_min = (np.random.rand(15,1) * im_shape[0]).astype('int')
        # cur_bboxes = np.hstack([cur_x_min, cur_y_min, \
        #     (cur_x_min + np.random.rand(15,1)*50).astype('int'), \
        #     (cur_y_min + np.random.rand(15,1)*50).astype('int')])
        # cur_confidences = np.random.rand(15)*4 - 2

        #free parms
        decision_thres = -1
        step_size      = 1

        cur_bboxes = np.empty((0, 4))
        cur_confidences = np.empty(0)

        #scale image
        multi_scale_factor = np.array([0.9, 0.5, 0.3, 0.25])

        for sf in multi_scale_factor:
            im_s = cv2.resize(im, None, fx = sf, fy = sf, interpolation = cv2.INTER_LINEAR) #AREA
            r    = 1/sf

            #image to hog feature
            f  = vlfeat.hog.hog(im_s, cell_size)

            y_max = f.shape[0] - template_size
            x_max = f.shape[1] - template_size


            cur_y_min  = 0
            #sliding window at multiple scales
            while (cur_y_min < y_max):
                cur_x_min  = 0
                while (cur_x_min < x_max):

                     #extract feature for current bounding box
                     bb_f = f[cur_y_min:cur_y_min+template_size, cur_x_min:cur_x_min+template_size]
```

```python
                bb_f = bb_f.flatten()
                bb_f  = bb_f.reshape(1, -1)

                #classify & threshold classification confidence
                conf = svm.decision_function(bb_f)
                if(conf > decision_thres):
                    bb = np.array([round(cur_x_min * cell_size * r), \
                                   round(cur_y_min * cell_size * r), \
                                   round((cur_x_min+template_size) * cell_size * r), \
                                   round((cur_y_min+template_size) * cell_size * r)])
                    cur_bboxes      = np.vstack((cur_bboxes, bb))
                    cur_confidences = np.hstack((cur_confidences, conf))

                #update sliding window
                cur_x_min = cur_x_min + step_size
            #end sliding horizontally cur_x_min

            cur_y_min = cur_y_min + step_size
        #end sliding vertically cur_y_min

    #call non_max_supression_bbox (remove duplicates)

    ### non-maximum suppression ###
    # non_max_supr_bbox() can actually get somewhat slow with thousands of
    # initial detections. You could pre-filter the detections by confidence,
    # e.g. a detection with confidence -1.1 will probably never be
    # meaningful. You probably _don't_ want to threshold at 0.0, though. You
    # can get higher recall with a lower threshold. You should not modify
    # anything in non_max_supr_bbox(). If you want to try your own NMS methods,
    # please create another function.

    idsort = np.argsort(-cur_confidences)[:topk]
    cur_bboxes = cur_bboxes[idsort]
    cur_confidences = cur_confidences[idsort]
```

```

        is_valid_bbox = non_max_suppression_bbox(cur_bboxes, cur_confidences,
            im_shape, verbose=verbose)

        print('NMS done, {:d} detections passed'.format(sum(is_valid_bbox)))
        cur_bboxes = cur_bboxes[is_valid_bbox]
        cur_confidences = cur_confidences[is_valid_bbox]

        bboxes = np.vstack((bboxes, cur_bboxes))
        confidences = np.hstack((confidences, cur_confidences))
        image_ids.extend([im_id] * len(cur_confidences))


    #debug print
    elapsed_time = time.time() - start_time
    print(" Elaspsed time: " + str(elapsed_time) + " seconds")

    return bboxes, confidences, image_ids
```

六、实验数据及结果分析：

以 data 中的人脸图片作为正训练的例子，积极（人脸）HOG 特征的可视化结果如图 1 所示，平均精度如图 2 所示。

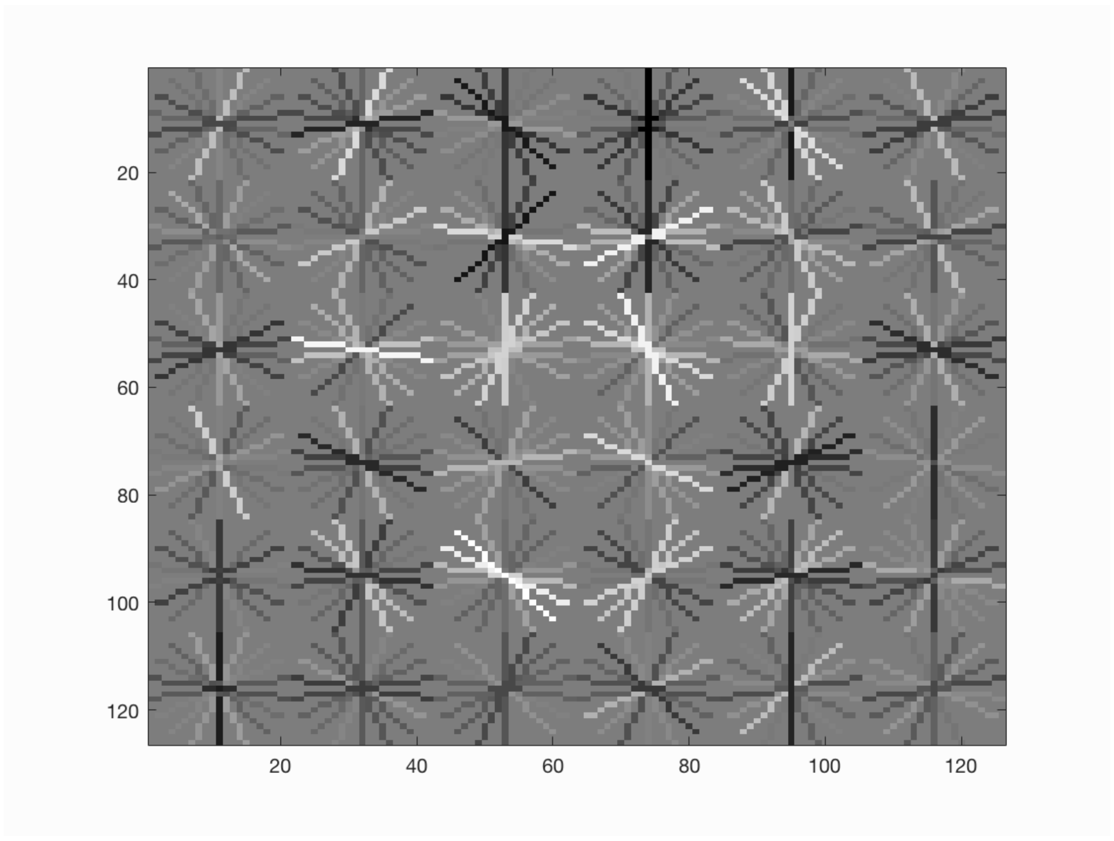


图 1 积极（人脸）HOG 特征的可视化结果图

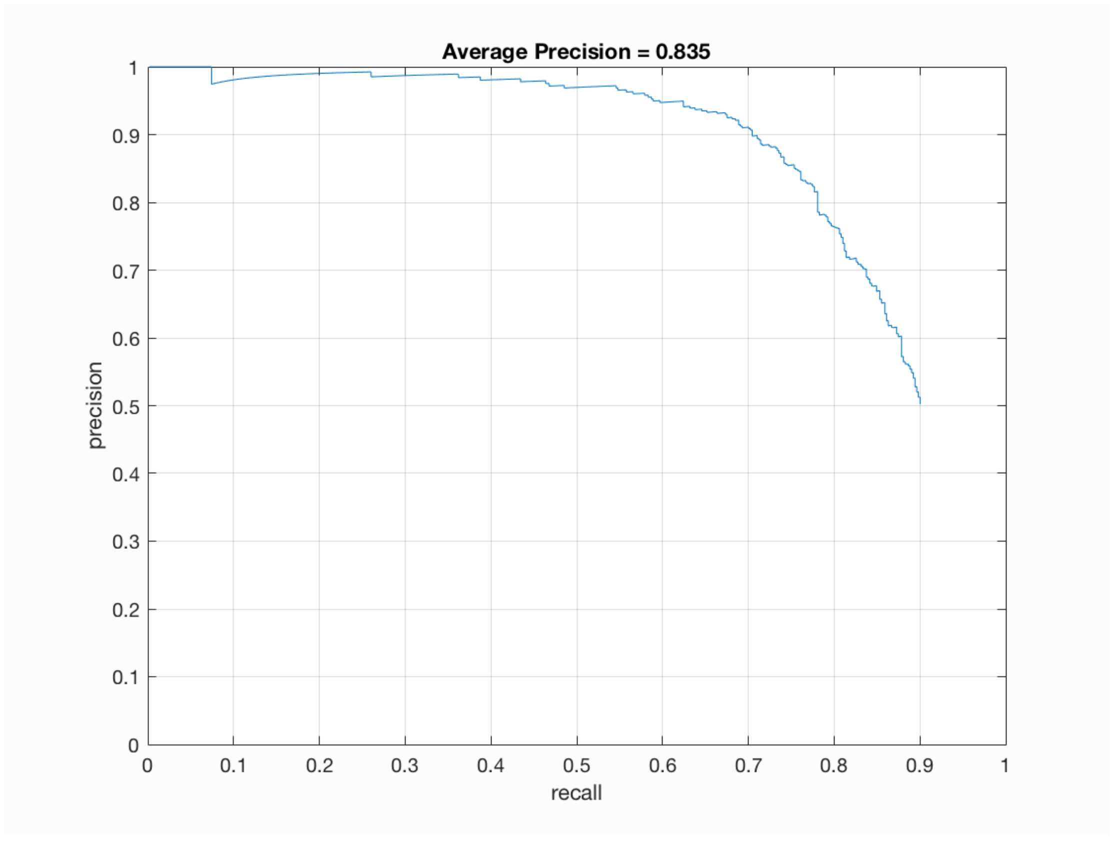
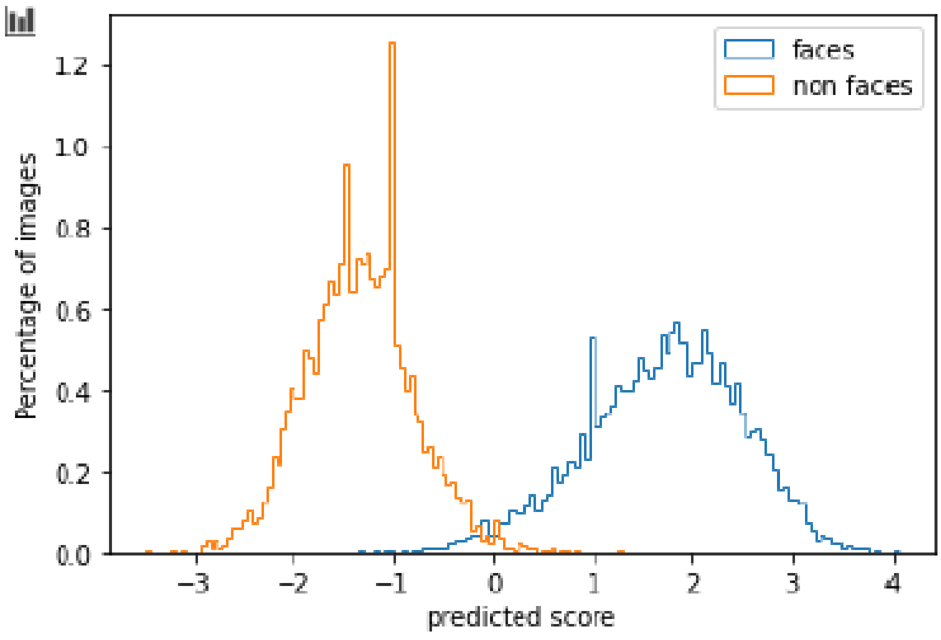
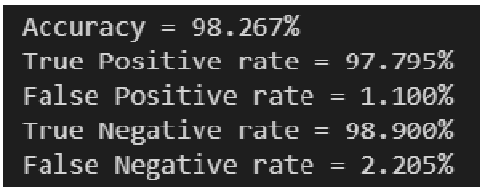


图 2 积极（人脸）HOG 特征的平均精度结果图

分类器训练结果如图 3 所示。





```
Accuracy = 98.267%
True Positive rate = 97.795%
False Positive rate = 1.100%
True Negative rate = 98.900%
False Negative rate = 2.205%
```

图 3 分类器训练结果

测试集的检测示例如图 4 所示。

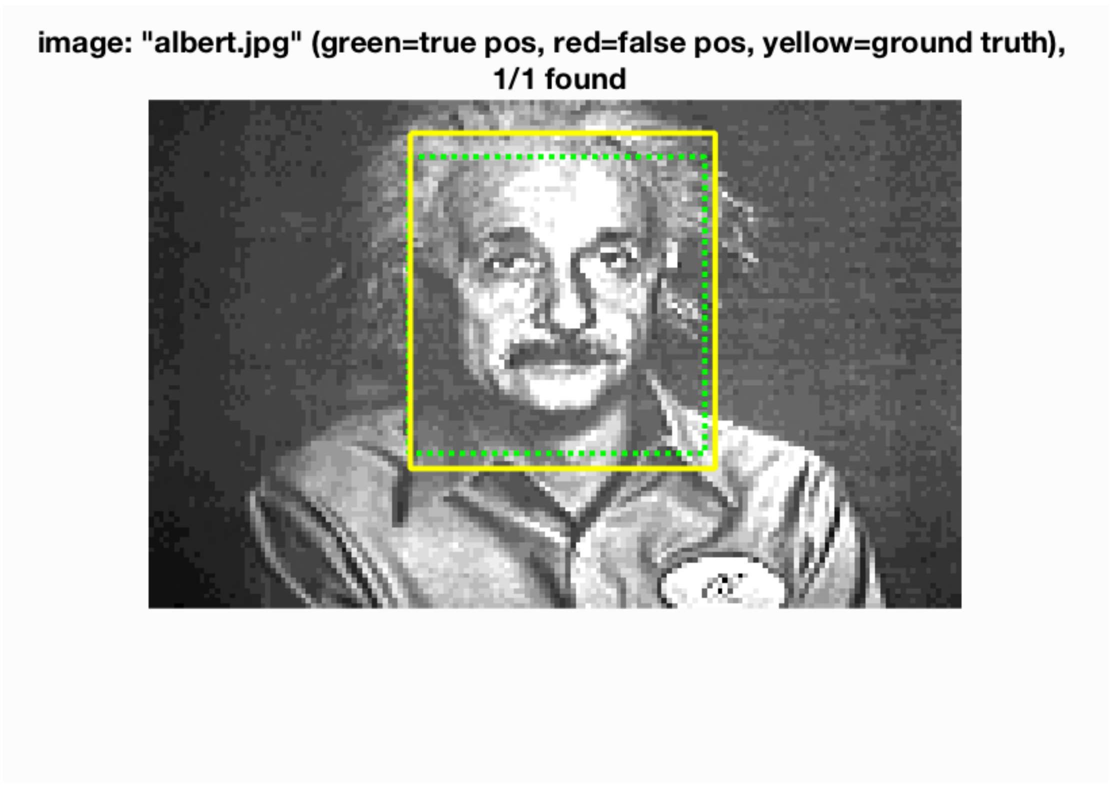
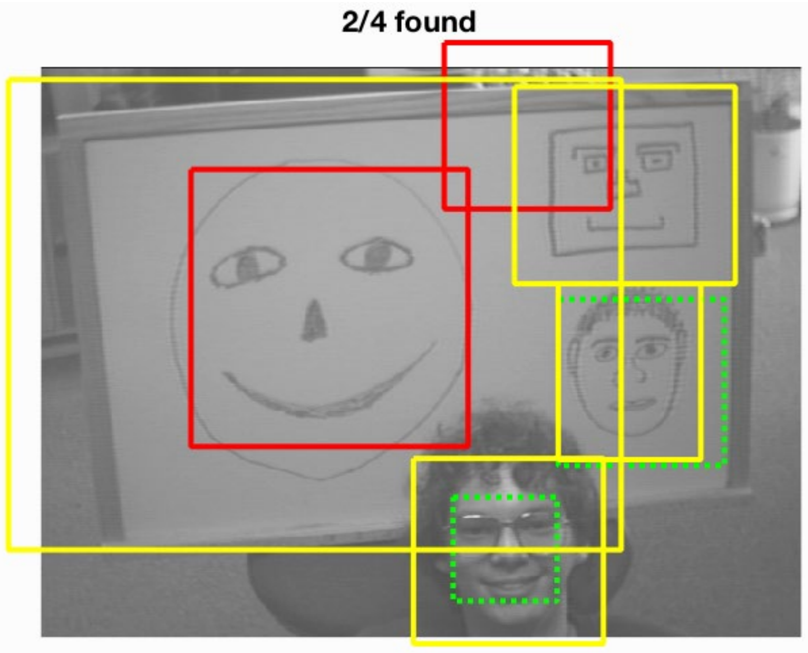


image: "albert.jpg" (green=true pos, red=false pos, yellow=ground truth),
1/1 found


2/4 found

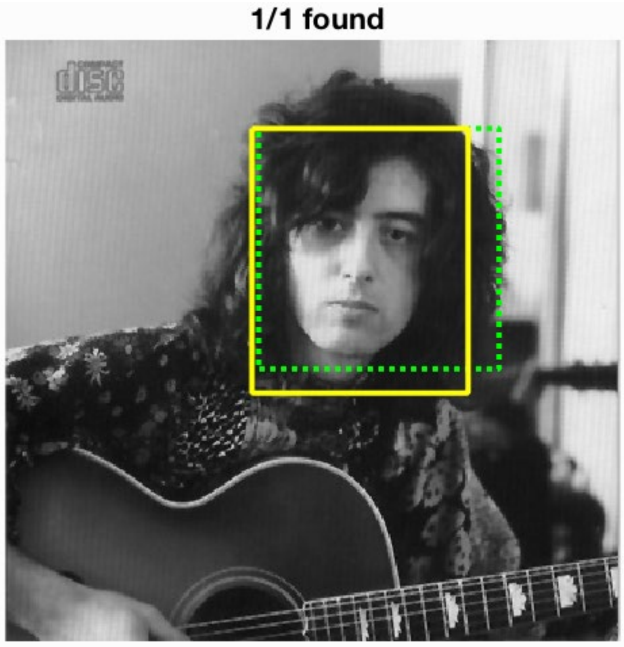
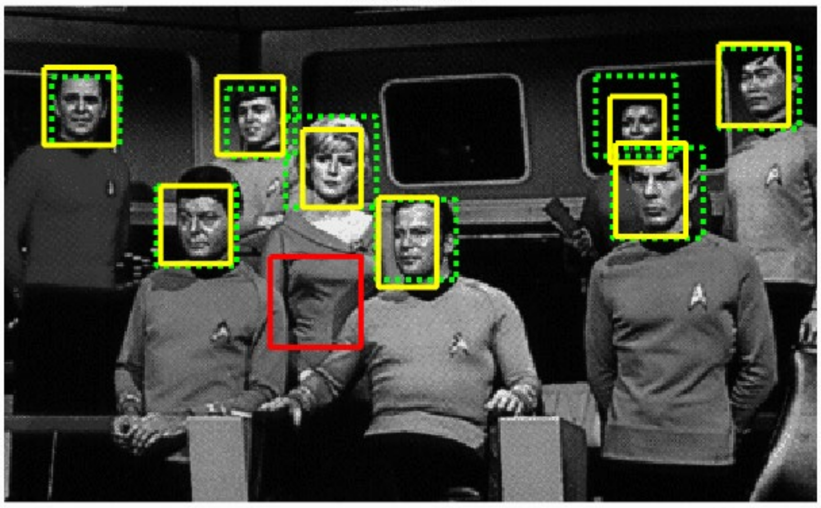
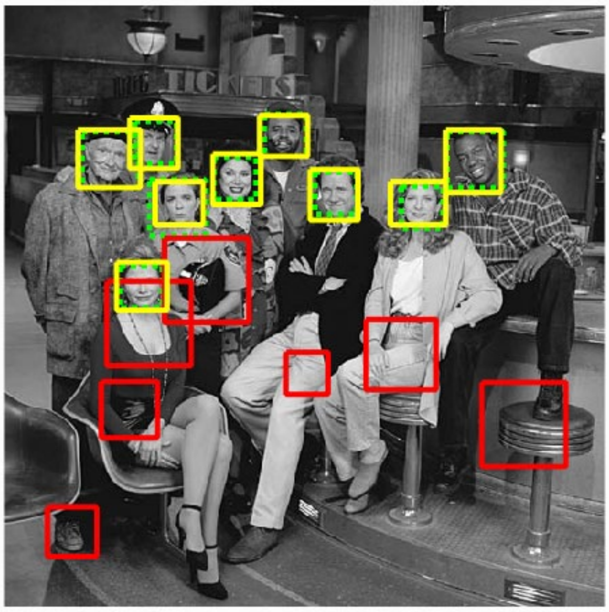

1/1 found

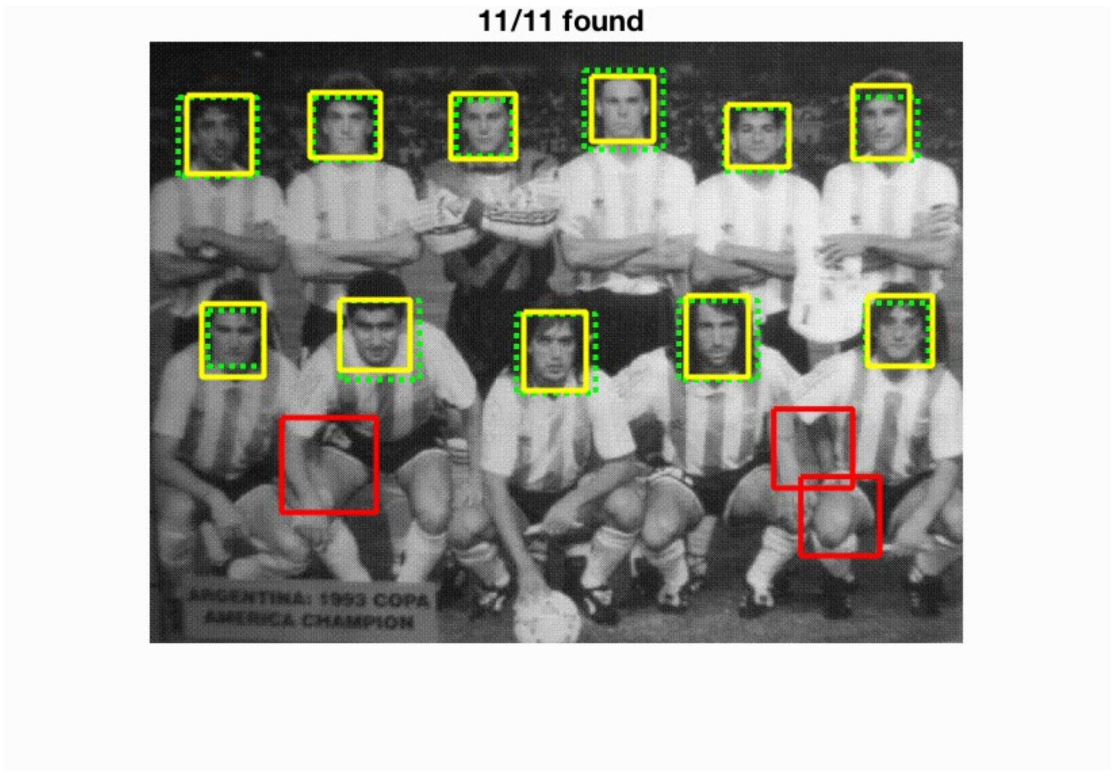
image: "original1.jpg" (green=true pos, red=false pos, yellow=ground truth), 8/8 found



9/9 found



额外测试集的检测示例如图 5 所示。



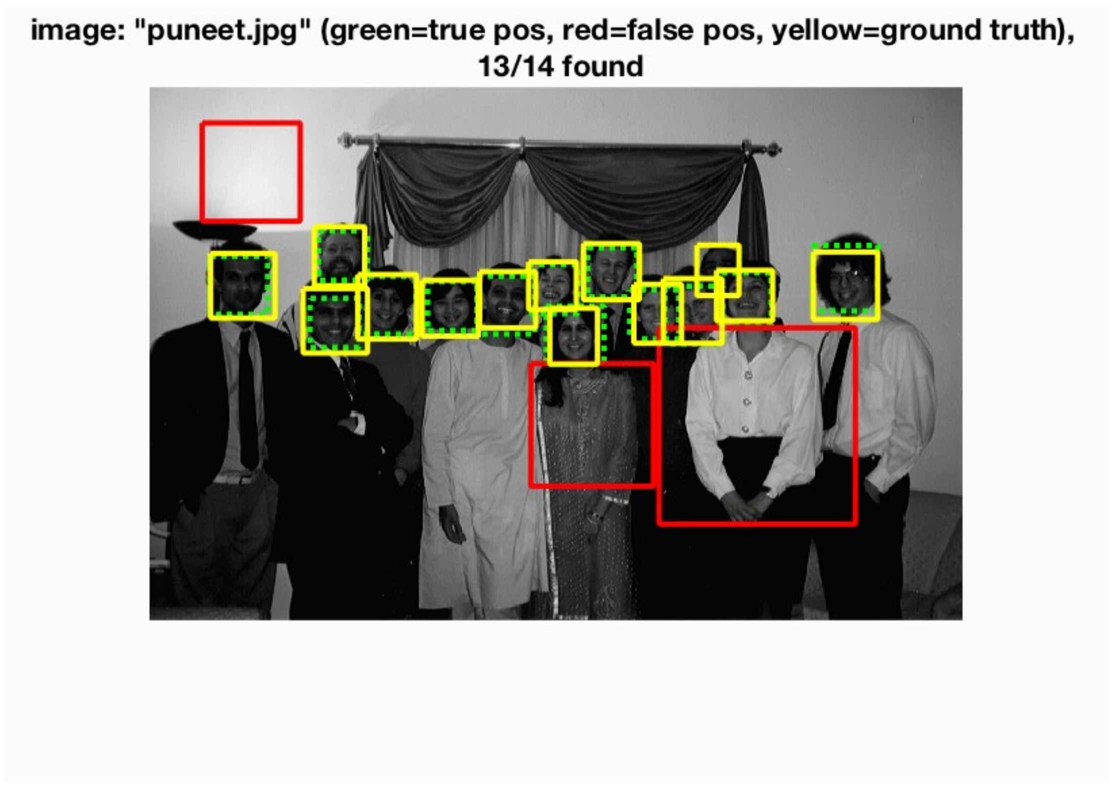
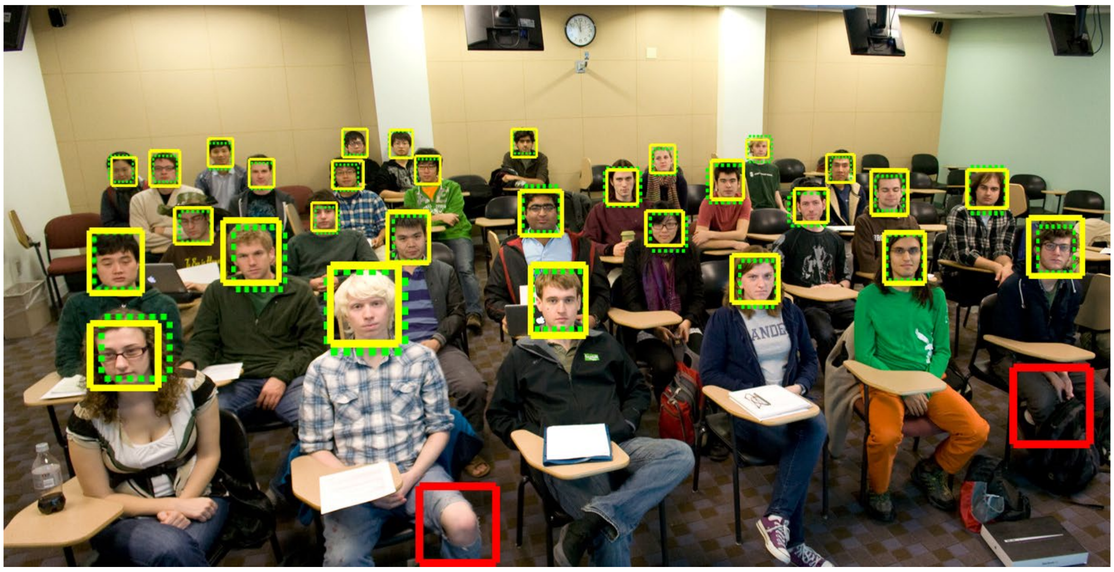


image: "puneet.jpg" (green=true pos, red=false pos, yellow=ground truth),
13/14 found

图 4 测试集的检测示例

额外测试集的检测示例如图 5 所示。



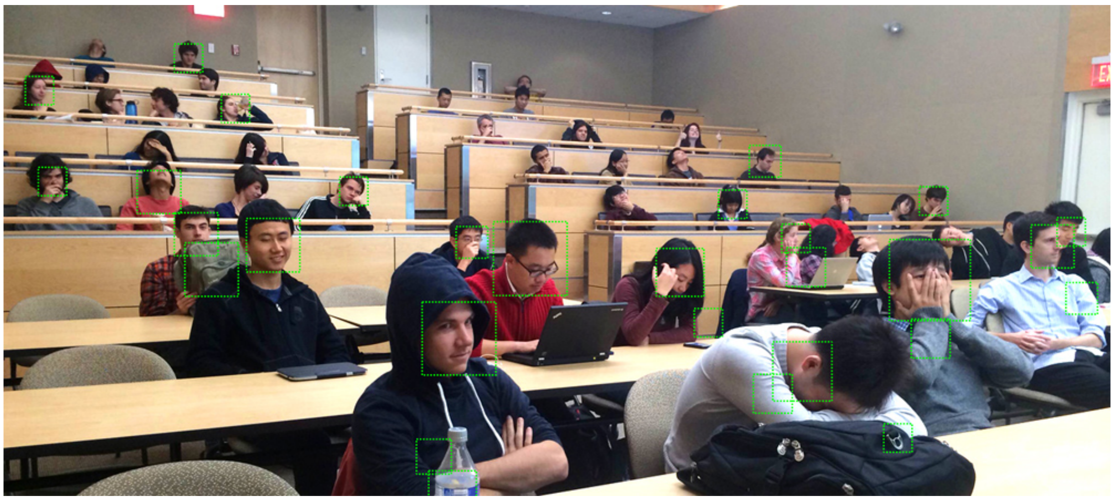


图 5 额外测试集的检测示例

七、实验结论：

从积极（人脸）HOG 特征的可视化结果可以看出，平均精度为 0.835，可以认为取得了较好的效果。

从测试集和额外测试集的检测示例可以看出，人脸检测的分类效果比较好。

八、总结及心得体会：

本实验利用给定的训练集与测试集图片，获得积极（人脸）特征和随机消极（非人脸）特征，训练了一个线性 SVM 分类器，并使用分类器在

多个尺度上对数百万个滑动窗口进行分类，取得了较好的效果。

九、对本实验过程及方法的改进建议：

提供实验指导书。

报告评分：

指导教师签字：