1、Experimental Design:

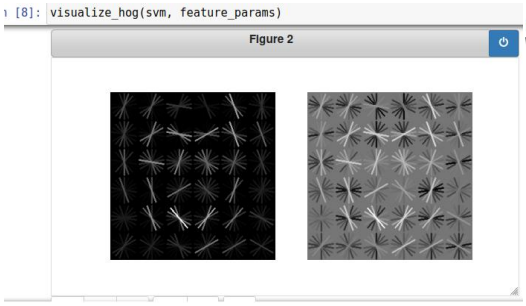
1、1 change number of top detections to feed to NMS topk

1、2 Change the svm constant c :5e-2

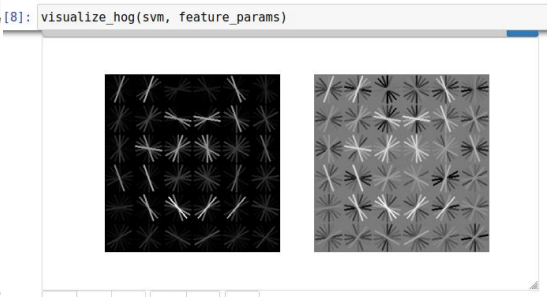
2、Experimental Results Analysis:

2.1、train svm

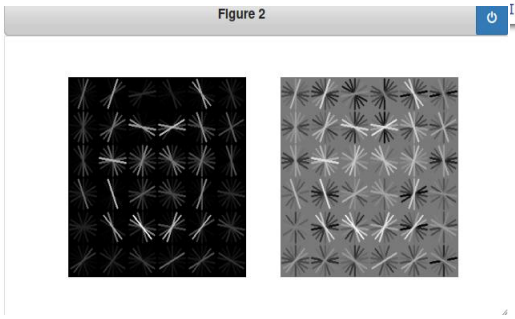
Topk=20



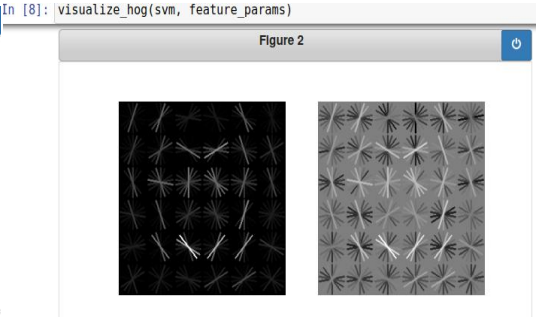
Topk=100



Topk=200

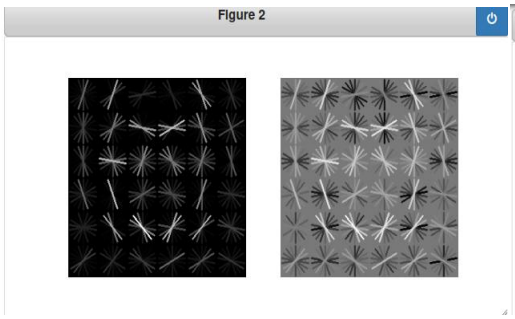


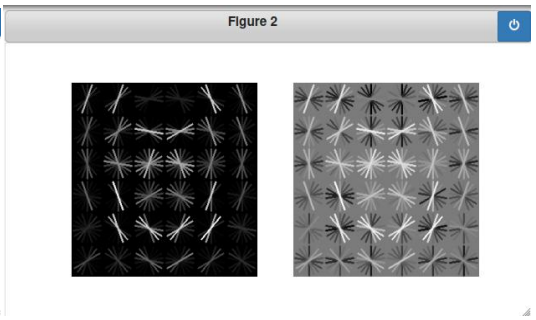
Topk=300



C=5e-2

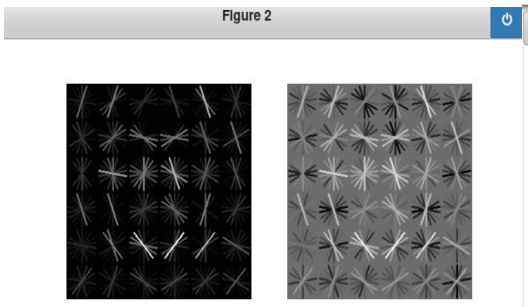
c=4e-3

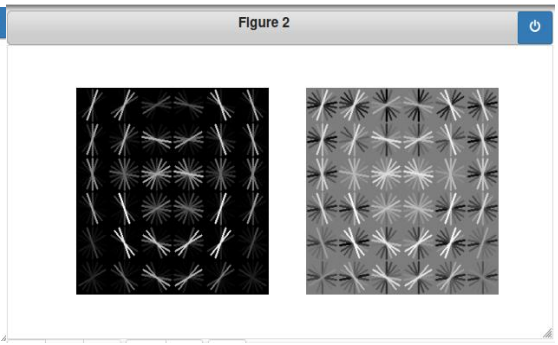




C=0.1

c=1e-4



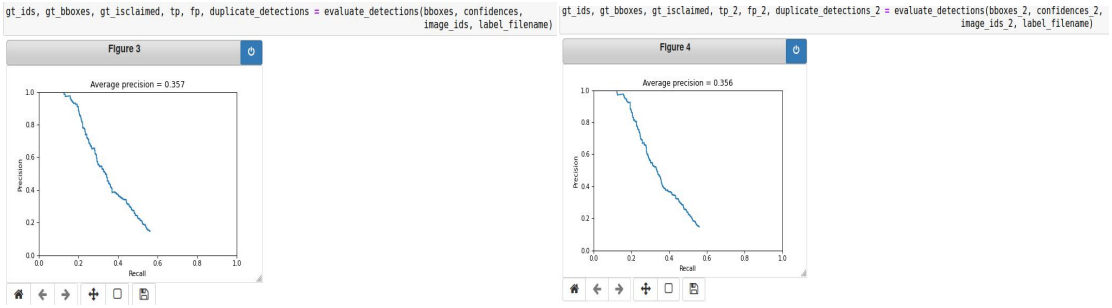


可以看出，随着 c 值的减小，效果有变得更好

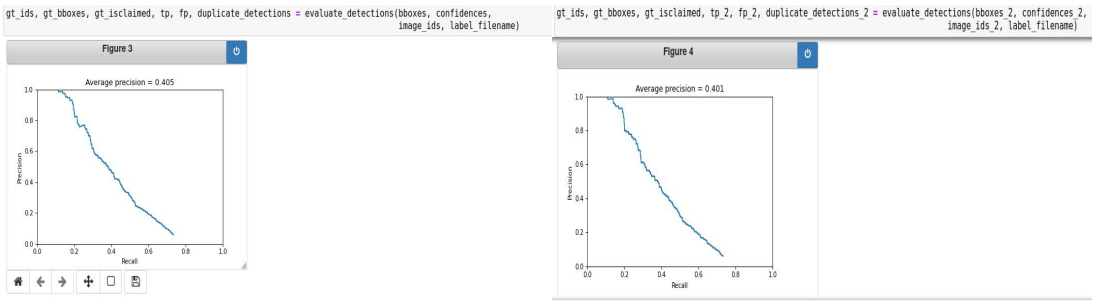
2.2、run detector

Single scale:

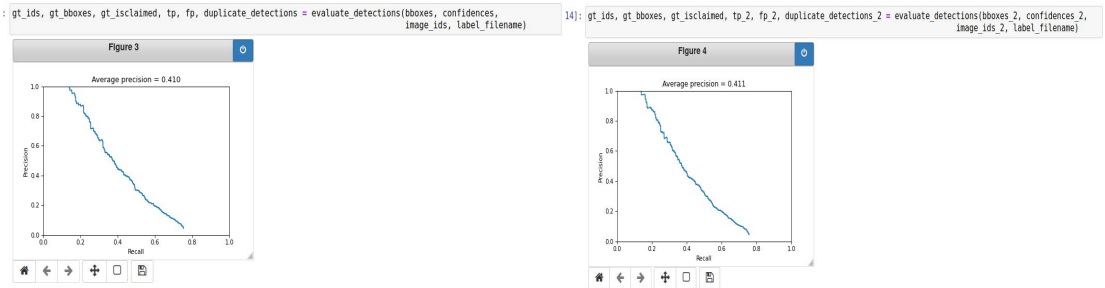
Topk=20



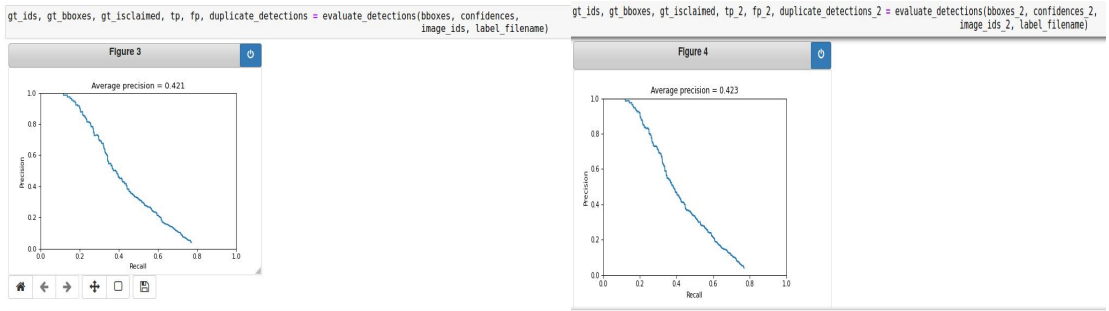
Topk=100



Topk=200

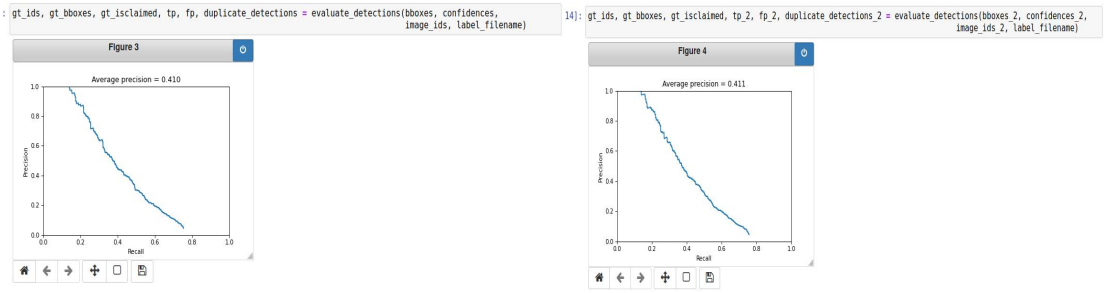


Topk=300

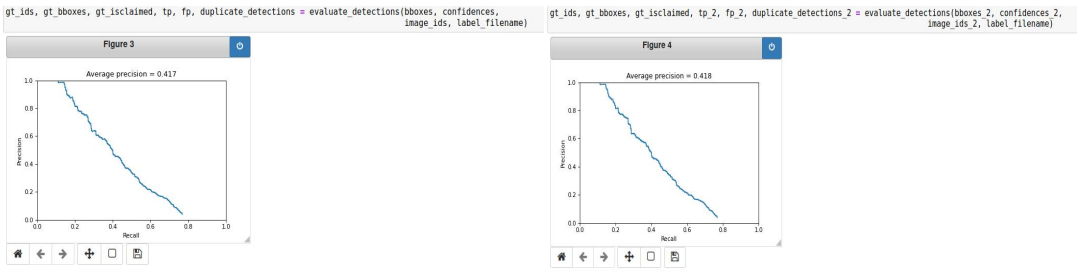


可以看出两个结果的 ap 相差不大，只有零点零几的差值，并且随着 topk 值的增大，ap 总体呈现上升趋势，并且在 20-100 里上升明显，之后的增速放缓

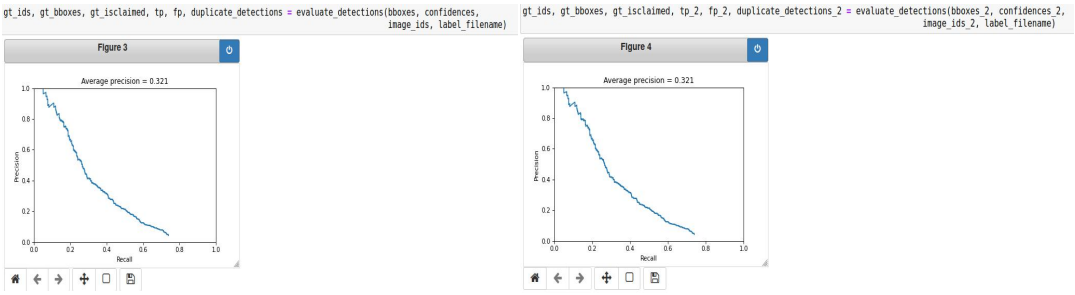
C=5e-2



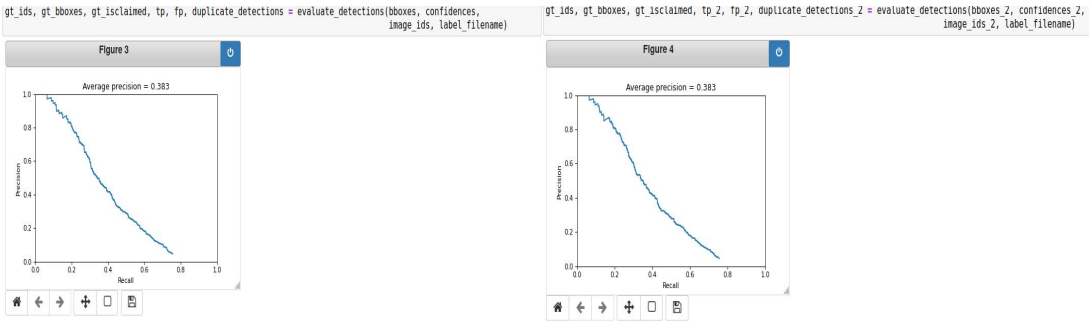
C=4e-3



C=1e-4



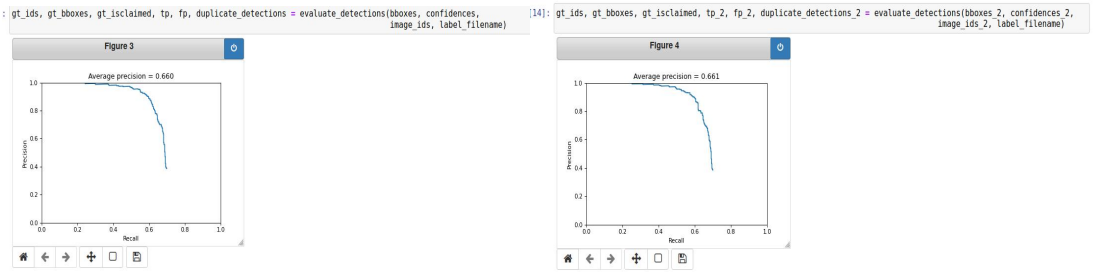
C=0.1



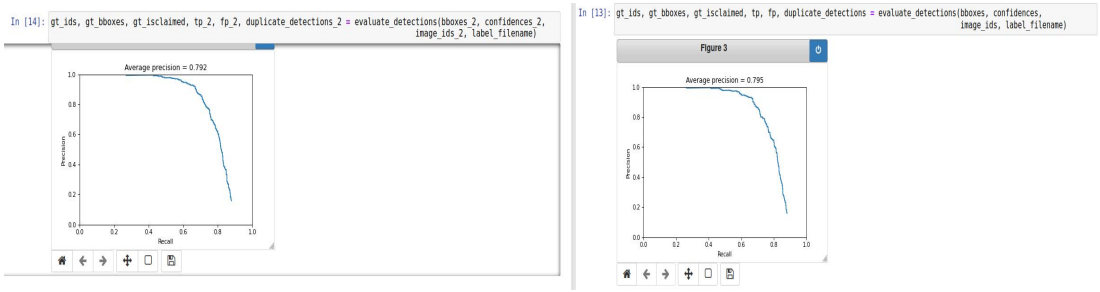
可以看出两个结果的 ap 相差不大，只有零点零几的差值，并且随着 c 值的增大，ap 整体呈现先上升后下降，猜测峰值会出现在 4e-3 和 5e-2 中间

Multiscale:

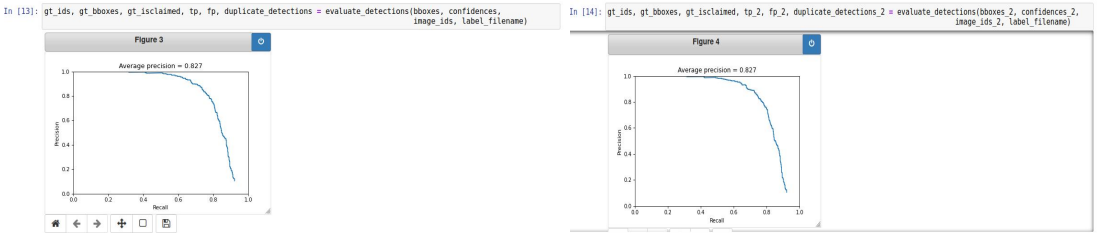
Topk=20



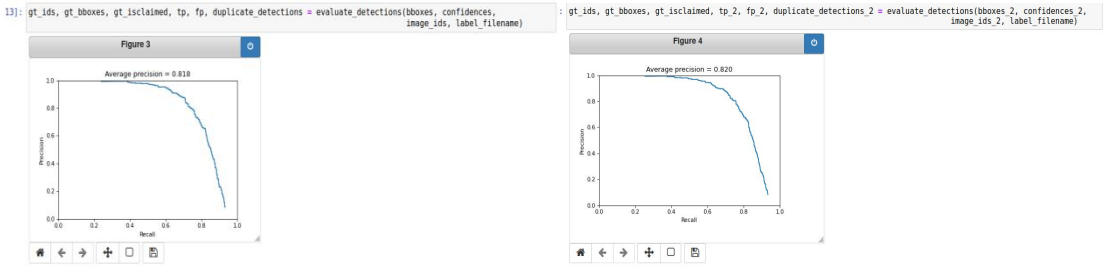
Topk=100



Topk=200

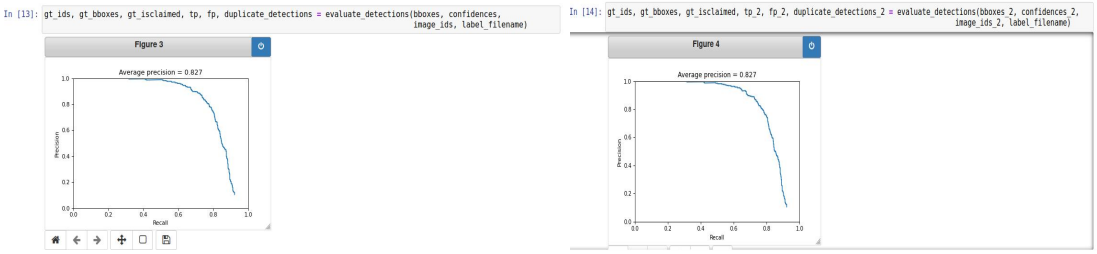


Topk=300

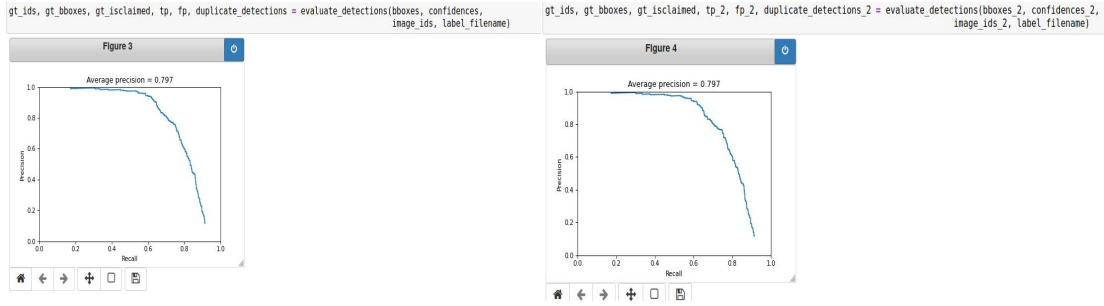


可以看出两个结果的 ap 相差不大，并且随着 topk 值的增大，ap 总体呈现先上升后下降的趋势，并且在 20-100 里上升明显，预测峰值在 200-300 附附近，300 之后可能缓步下降

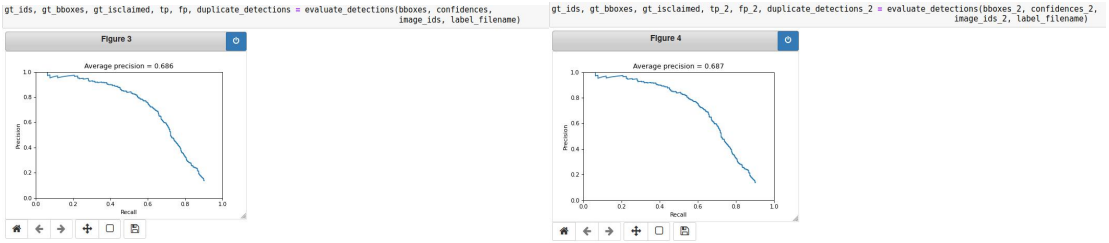
C=5e-2



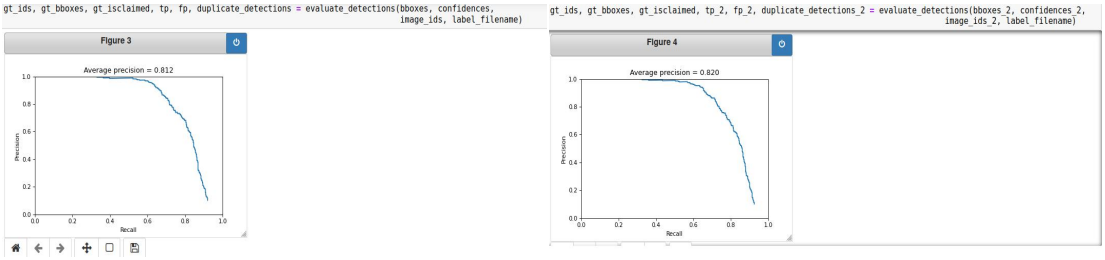
C=4e-3



C=1e-4



C=0.1



可以看出两个结果的 ap 相差不大，只有零点零几的差值，并且随着 c 值的增大，ap 整体呈现先上升后下降，猜测峰值会出现在 5e-2 和 0.1 中间

从整体上看 topk 值对于 multi scale 的影响较大，最大差值为 0.167，而 single scale 的差值为 0.067；同样的 c 值对于 multi scale 的影响较大，最大差值为 0.414，而 single scale 0.097。

Core code：

Get positive feature：

```
    template_size = int(win_size / cell_size)
    # Compute the number of cells per template
    n_cells = template_size * template_size * 31

    # Initialize the feature matrix
    feats = np.zeros((len(positive_files), n_cells))

    for i, file_path in enumerate(positive_files):
        # Load the image in grayscale
        img = load_image_gray(file_path)

        # Compute HOG features
        hog_features = vlfeat.hog.hog(img, cell_size)

        # Flatten the HOG features and store in the feature matrix
        feats[i, :] = hog_features.ravel()

    return feats
```

Get random negative feature：

```
feats = []
for file_path in negative_files:
    # Load the image in grayscale
    img = load_image_gray(file_path)
    img_height, img_width = img.shape

    # Randomly sample patches from the image
    for _ in range(num_samples // len(negative_files)):
        if img_height < win_size or img_width < win_size:
            continue

        y = np.random.randint(0, img_height - win_size)
        x = np.random.randint(0, img_width - win_size)
        patch = img[y:y + win_size, x:x + win_size]

        # Compute HOG features
        hog_features = vlfeat.hog.hog(patch, cell_size)

        # Flatten the HOG features and store in the feature list
        feats.append(hog_features.ravel())

feats = np.array(feats)
```

Train classfier:

```
X = np.vstack((features_pos, features_neg))
y = np.hstack((np.ones(features_pos.shape[0]), -1 * np.ones(features_neg.shape[0])))

# Train linear SVM
svm = LinearSVC(C=C)
svm.fit(X, y)
```

Mine hard negative:

```
template_size = int(win_size / cell_size)
n_cells = template_size * template_size * 31

hard_neg_feats = []

for file_path in negative_files:
    # Load the image in grayscale
    img = load_image_gray(file_path)
    img_height, img_width = img.shape

    # Randomly sample patches from the image
    for _ in range(10):  # we can choose 10 samples per image, for instance
        if img_height < win_size or img_width < win_size:
            continue

        y = np.random.randint(0, img_height - win_size)
        x = np.random.randint(0, img_width - win_size)
        patch = img[y:y + win_size, x:x + win_size]

        # Compute HOG features
        hog_features = vlfeat.hog.hog(patch, cell_size)
        features = hog_features.ravel().reshape(1, -1)

        # Predict using the SVM
        prediction = svm.predict(features)

        if prediction == 1:  # if predicted as face (false positive)
            hard_neg_feats.append(features.ravel())

hard_neg_feats = np.array(hard_neg_feats)
return hard_neg_feats
```

Run detector:

```
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 = 200

# 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)

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

    ###########################################################################
    #                       TODO: YOUR CODE HERE                              #
    ###########################################################################

    W = svm.coef_
    B = svm.intercept_
    conf_thres=-1.0
    #下面这行解注释是single scale
   # scales = [0.9]
 #下面这行解注释是Multi scale
    scales = [1.0, 0.95, 0.9, 0.85, 0.8, 0.75, 0.7, 0.65, 0.6, 0.55, 0.5, 0.45, 0.4, 0.35, 0.3, 0.25, 0.2]
    cur_bboxes = []
    cur_confidences = []

    for scale in scales:
        x = int(scale * im_shape[0])
        y = int(scale * im_shape[1])

        IM_resized = cv2.resize(im, (y, x), interpolation=cv2.INTER_AREA)

        for i in range((IM_resized.shape[0] - win_size)//10):
            for j in range((IM_resized.shape[1] - win_size)//10):
                patch = IM_resized[10*i:10*i+win_size, 10*j:10*j+win_size]
                HOG = np.expand_dims(vlfeat.hog.hog(patch, cell_size).flatten(), axis = -1)
                v = W.dot(HOG) +B
                if v>= conf_thres:
                    cur_confidences.append(v)
                    cur_bboxes.append(np.expand_dims((1.0/scale)*np.array([10*j, 10*i, 10*j+win_size,
i+win_size]).astype(np.float32), axis=0))
                    cur_bboxes[-1] = cur_bboxes[-1].astype(int)



    num_detections = len(cur_confidences)
    if num_detections > 1:
        cur_confidences = np.squeeze(np.concatenate(cur_confidences, 0))
        cur_bboxes = np.concatenate(cur_bboxes, axis=0)
    elif num_detections == 1:
        cur_confidences = np.squeeze(cur_confidences[-1], axis=-1)
        cur_bboxes = cur_bboxes[-1]

    ###########################################################################
    #                         END OF YOUR CODE                                #
    ###########################################################################

    ### 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.
    if num_detections > 0:
        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))

return bboxes, confidences, image_ids
```