Exercise 10

Training and evaluating a pedestrian classifier

IAML 2020

Purpose In this exercise you will implement a pedestrian classifier using support vector machines. In this exercise you will:

- Use Histogram of Oriented Gradients (HOG) feature and a Support Vector Machine (SVM) for pedestrian detection.
- Test and evaluate the classifier on images and videos.

Notes

- All exercises are non-mandatory and we have purposefully added (hopefully) enough exercises to keep everyone busy. We mark less important exercises with ^{extra}. If you don't have time to do all exercises, wait until we release the solutions and use them as a guide.
- In the exercises based on script (i.e. `.py`) files, you are expected to implement code at certain points in the file. Although it is typically clear from context which piece of code you have to change, we have included specifier comments of the type `# <Exercise x.x (n)>`, where `x.x` is the exercise number and `(n)` is the letter associated with the specific task.

Exercise 10.1

Pedestrian detector

In this exercise, you will train a pedestrian detector using a support vector machine (SVM) and HOG features. Use the script `training.py` to answer this exercise. We use OpenCV's built-in implementation of SVMs for this exercise since it allows easy detection at multiple scales using image pyramids.

Dataset

Figure 1: True positive images with pedestrians from the INRIA Person Dataset.

Figure 2: Images without pedestrians from the INRIA Person Dataset.

This exercise is based on Dalal and Triggs (2005)[1]. You will use the INRIA Person Dataset[2] to train your pedestrian detector. This dataset is divided in two subsets, namely: (i) *positive*, a subset of 2416 images (96 × 160) with pedestrians; and (ii) *negative*, a subset with 1218 images (of various resolutions) without pedestrians. Figure 1 shows five positive images with pedestrians and Figure 2 shows three negative images.

In the following, you will randomly generate the negative training windows from the initial negative training dataset. Make sure you understand the function `sample_negative_images()`. The idea of the function is to generate a fixed set of 12180 patches sampled randomly from 1218 person-free training photos provided the initial negative set. Figure 3 shows ten 64 × 128 regions selected from a negative image.

Notice:

- The variable `negativeList` contains the 1218 negative training photos constituting the initial negative training dataset;
- The 12180 patches sampled are contained in `negative_sample`

[1] Link to article.

[2] Link to overview page.

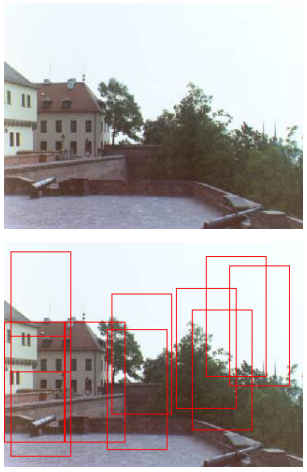

Figure 3: Example of ten windows selected from a negative image.

Task 1: *Preprocessing and initial training*

In the following you will generate HOG-features for the training images and use them to fit an SVM classifier.

1. **Compute HOG:** Implement the function `compute_hog()`. The function should compute the Histogram of Oriented Gradients (HOG) of all images in the dataset, where the HOG features of the 2416 positive images should be stored in `positive_list` and the 12180 negative images in the variable `negative_sample`.

   Make sure that the function `compute_hog()` performs the following items:

   - Use the centered 64×128 pixels window in the positive images
   - Convert the image to grayscale before calculating the histogram
   - Use the function `cv2.HOGDescriptor().compute()` [3] to compute the HOG feature
   - Add the computed HOG feature in the variable `hog_list`.

   [3] Link to docs

2. **Train the pedestrian detector:** The code for labeling the training data has been provided. Use 1 for positive examples and -1 for negative examples. The values are stored in `labels`. The parameters used for the SVM by Dalal and Triggs (2005) are provided. Note the function `cv2.ml.SVM_create()` is used to create the SVM model. Use the function `svm.train()` to train a SVM model using the calculated HOG features. Example:

   If you want to learn more about using SVMs in OpenCV, see the following tutorial.

   ```
   svm.train(np.array(hogList), cv2.ml.ROW_SAMPLE, np.array(labels
       ↪ ))
   ```

Task 2: *Identify false positives*^{extra}

The pedestrian detector may produce quite a bit of false positives. However, from the negative training examples we can easily identify false positives called "*hard examples*" and use it to retrain the classifier. In the following steps you will make a list (i.e. the variable `hard_negative_list`) of examples where the classifier made false positives (e.g. on the negative training images) .

1. Use the classifier on all images in `negative_list`. Although these training images are person-free, your detector will likely detect false positives. These false positive regions can be reused to improve the detector by simply adding the false positive examples to the negatives. Here you will add the false positives to `hard_negative_list`.

   Use the following guidelines:

   - Use the function `hog.detectMultiScale()` [4,5] to detect pedestrians (at various scales) in the input image e.g.

     ```
     rectangles, weights = hog.detectMultiScale(image, winStride
         ↪ =(4, 4), padding=(8, 8), scale=1.05)
     ```

   - Use `rectangles`, to define the Regions of Interest in the input image, i.e. the detected pedestrian

   [4] Link to docs
   [5] Link to a detailed description of the purpose of each parameter.

- Resize the ROI image to 64×128 and add the resized image to `hard_negative_list`.

2. **Re-train the pedestrian detector:** Now it is time to re-train the classifier using positive, negative and "hard" examples.

   - Compute the HOG feature of each hard example using the function `compute_hog()`;
   - Update the variable `labels`, adding the label "-1" to those images images available on `hard_negative_list`
   - Re-train the SVM model using the function `svm.train()`.

The exercise file exports the trained SVM model to `outputs/feature.npy` The code to export the 3781×1 HOG feature matrix into a Numpy file is given in the exercise file. Use the exported file in the next exercise when detecting pedestrians in images and videos.

Exercise 10.2

Testing the classifier

In this exercise you will use the trained classifier to detect pedestrians in test images. Use the script `test.py` to answer this exercise.

The script already contains code to import the trained SVM model (saved in a Numpy file) and to add the model in a HOG Descriptor (`cv2.HOGDescriptor()`).

The scripts also read all test images from the INRIA Person Dataset and create a vector called `positive_list` . You have to use the image available on `positive_list` to test the performance of your detector.

In this exercise, you will detect pedestrians of different sizes/ scales. You should use the function `cv2.HOGDescriptor.detectMultiScale()` to detect the objects. The function returns a list of `rectangles` where each rectangle contains the detected object, and a vector of `weights` the classifiers classifier with a larger confidence. *Hint:* You can use a thresholds to reduce the number of false positives. Figure 4 shows some results.

Figure 4: Pedestrian detection using HOG+SVM.

1. **Detect pedestrians:** Use the function `cv2.HOGDescriptor.detectMultiScale()` to detect objects in the training data.
2. **Draw the detected pedestrians:** Use the function `cv2.rectangle()` to draw a red rectangle around each detected pedestrian in the input images.
3. How well does the method perform on the training and test data?
4. **Detect in video:** Change your implementation so it now detect pedestrians in image sequences. Use the video file `pedestrians.mov` in the folder `inputs` to test your detector. *Hint:* Use pyramids to downsample or upsample the video frames and to speed up your pedestrian detector.
5. **Evaluation:**^{extra} Evaluate the classifier by performing the following tasks:
   - Make a few videos with pedestrians to test the detector and make the videos so you can argue for reasons when the detector works and when i it fails.
   - Load the negative samples from the test dataset and load labels for all samples. *hint:* You can reuse code from `training.py` .

- Use the function `calculate_metrics(y_true, y_predict)` from exercise 9 to calculate the confusion matrix and other metrics. Use the results to evaluate your detector.