Technische Universität München
Fakultät für Informatik
PD Dr. Slobodan Ilic

WS 18/19
Exercise Sheet 3
November 2, 2017

Exercises in Tracking & Detection

Object Classification and Object Detection are important topics in Computer Vision. The task of Classification is to predict the label of a given image, and Detection aims at finding out where and what of a specific object in the image. Random forest is a powerful machine learning algorithm and has been widely applied in many computer vision related problems. A random forest is an ensemble of multiple randomly trained decision trees, and has more robust performance compared to a single tree. In this exercise, we will use Random Forest to solve the problem of Object Classification and Detection.

Task 1 Image processing and HOG descriptor

OpenCV is a famous and popular open source Computer Vision library. It is released under a BSD license and hence it's free for both academic and commercial use. In this exercise, OpenCV for C++ is required for all the implementation. In this task, you're required to master the basic usage of OpenCV on image processing and then use the built-in class HOGDescriptor to detect HOG descriptors on given images.

Basic Operations in OpenCV As a popular Computer Vision library, OpenCV has a very active community, detailed documentation and plentiful tutorials online, which makes it much easier for beginners to start with. It is also a cross-platform library, which means it can be installed on both Linux/Unix-based operation systems [1] and Windows[2]. You can choose to compile your code with OpenCV either using gcc and CMake [3] or using Microsoft Visual Studio [4]. Besides installation and configuration of OpenCV on your computer, you're also required to master some basic image operations in OpenCV, which are necessary for the accomplishment of following tasks:

- Load and save image – cv::imread, cv::imwrite
- Convert imknage to gray scale – cv::cvtColor
- Image visualization – cv::imshow
- Image rotation and flip – cv::rotate, cv::flip
- Image padding – cv::copyMakeBorder
- More useful operations can be found the OpenCV document about **Operations on arrays**[5]

[1] `https://docs.opencv.org/4.0.0/d7/d9f/tutorial_linux_install.html`
[2] `https://docs.opencv.org/4.0.0/d3/d52/tutorial_windows_install.html`
[3] `https://docs.opencv.org/4.0.0/db/df5/tutorial_linux_gcc_cmake.html`
[4] `https://docs.opencv.org/4.0.0/d6/d8a/tutorial_windows_visual_studio_Opencv.html`
[5] `https://docs.opencv.org/4.0.0/d2/de8/group__core__array.html`

HOG Descriptor The histogram of oriented gradients (HOG) is a feature descriptor used in computer vision and image processing. The technique counts occurrences of gradient orientation in localized portions of an image. This method is similar to that of edge orientation histograms, scale-invariant feature transform descriptors, and shape contexts, but differs in that it is computed on a dense grid of uniformly spaced cells and uses overlapping local contrast normalization for improved accuracy.[6] An overall pipeline of extracting HOG descriptors are shown in Fig. 1.

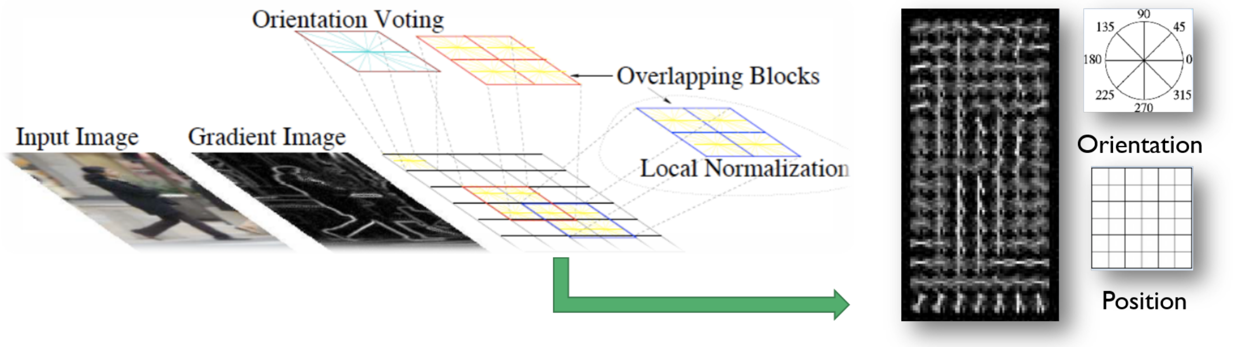


Figure 1: Process of extracting HOG descriptors

OpenCV provides a class **cv::HOGDescriptor** for computing HOG descriptors for a given image[7]. Although it is specially designed for pedestrian detection, and has built-in functions for detection using SVM, we will mainly use it's function **void cv::HOGDescriptor::compute** to get the HOG descriptors we need. For initializing a HOGDescriptor instance, you need to specify several parameters:

- **cv::Size _winSize**: window size. A window contains multiple blocks, and the descriptor for a window is a concatenation of all the descriptors of those blocks within that window.
- **cv::Size _blockSize**: block size. A block contains multiple cells, and the descriptor for a block is a concatenation of all the histograms from cells within that block.
- **cv::Size _blockStride**: stride of sliding block
- **cv::Size _cellSize**: cell size, the minimal unit for computing histogram.
- **int _nbins**: number of bins for histograms

Your task is to using OpenCV to load a image, apply different operations on the image (gray-scale conversion, resize, rotation, flip, .etc), compute HOG descriptors for the images, and then visualize as shown in Fig. 2. For visualization, you can use our helper function provided.

Task 2 Object classification

After extraction of HOG descriptors from a image, we can use it to train a classifier. In this task, we will use Binary Decision Tree and Random Forest to classify images using their HOG descriptors. A Random Forest is an ensemble of Random Trees. By aggregating all the predictions from different trees, a forest can in general yield a more robust prediction than a single tree. The relationship between a Random Tree and a Random Forest is shown in Fig. 3

[6] `https://en.wikipedia.org/wiki/Histogram_of_oriented_gradients`
[7] `https://docs.opencv.org/4.0.0/d5/d33/structcv_1_1HOGDescriptor.html`

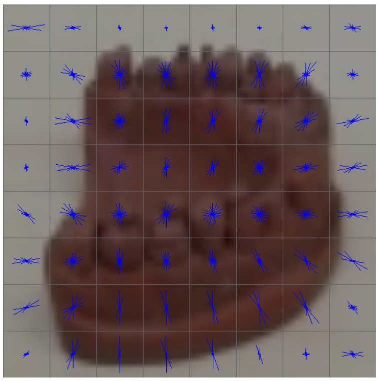
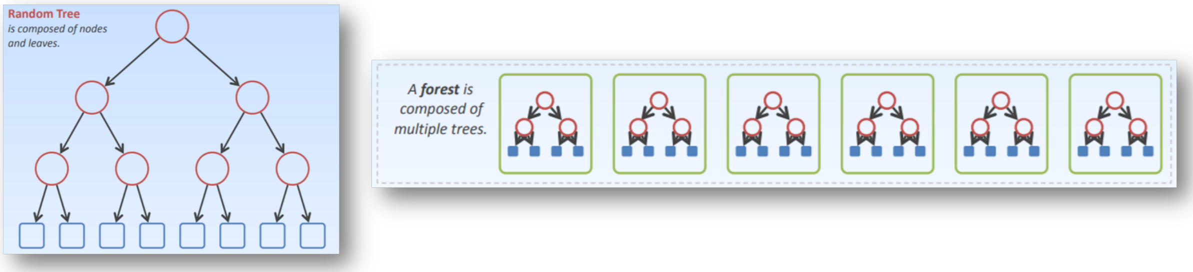


Figure 2: Visualization of computed HOG descriptor on a image



Random Tree
is composed of nodes and leaves.
A forest is composed of multiple trees.

Figure 3: Relatioship between a Random Tree and a Random Forest

For the implementation of Random Tree, we use the Binary Decision Tree provided by OpenCV. It is defined as **cv::ml::DTrees**[8]:

- Create a decision tree – cv::ml::DTrees.create()
- Some parameters to set:
  - void setCVFolds(int val); // set num cross validation folds
  - void setMaxCategories(int val); // set max number of categories
  - void setMaxDepth(int val); // set max tree depth
  - void setMinSampleCount(int val); // set min sample coun
- Train a decision tree – cv::ml::DTrees.train()
- Predict class using decision tree – cv::ml::DTrees.predict()

After being able to do classification with one Binary Decision Tree, you're required to implement a Random Forest class composed of a group of Binary Decision Trees. You have to implement at least those three methods:

- create – construct a forest with a given number of trees and initialize all the trees with given parameters
- train – train each tree with a random subset of the training data
- predict – aggregate predictions from all the trees and vote for the best classification result as well as the confidence (percentage of votes for that winner class)

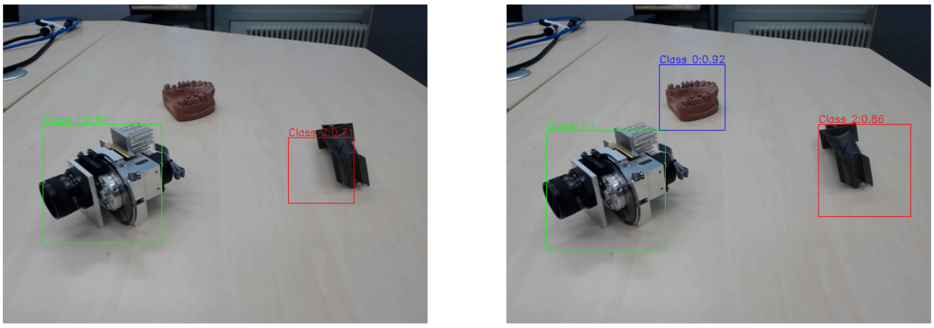
[8] `https://docs.opencv.org/4.0.0/d8/d89/classcv_1_1ml_1_1DTrees.html`

In this task, you should train a Binary Decision Tree and Random Forest separately as your classifiers, record and compare the classification results.

Task 3 Object detection

In this task, you will need to detect objects in images with random forest.

In the training stage, you have images from different objects organized in different directories. Especially, there's one background class, whose images are generated from the possible backgrounds you would see in the test images. You should train your Random Forest with the capability to distinguish between those images from different classes. Note that the data you have for training may not be sufficient enough, you may need to augment it (add rotation, flip etc.) to generate more samples for training. As shown in Fig.4, you should see improvements in the following detection results with data augmentation in training.



Detector trained without data augmentation

Detector trained with data augmentation

Figure 4: Object Detection Result Comparison

In the test stage, the pipeline for applying object detection consists mainly four steps:

a) Generate Bounding Boxes.

b) Classify Contents within each Bounding Box

c) Non-Maximum Suppression

d) Evaluate the detection result.

Generate Bounding Boxes For detecting whether some object exist in one image and what the class of that object is, we need to first generate a bunch of bounding boxes, which could possibly contain an object, and then give them to classifier to predict their classes. For this purpose, there are many algorithms out there, like Sliding Windows, EdgeBoxes, SelectiveSearch and etc. If you're interested in those different algorithms, you can find more details in this paper[9]. Here we only introduce Sliding Windows. The key idea of Sliding Windows is simple, that a window of fixed size is sliding through the whole image with certain strides. It's an exhaustive way to generate proposals. In general, multiple windows with different settings (aspect ratio, scale, and etc.) would be applied. For simplicity, you can fixed he aspect ratio of each window to 1:1 for this task, but should definitely try it with multiple scales. You are also free to choose other methods to generate bounding boxes.

[9] `https://arxiv.org/pdf/1502.05082.pdf`

Classify Contents within each Bouding Box We treat content within each bounding box as an independent image, and classify it to be either one of the objects or background with a confidence score(the percentage of votes from trees for that specific class). Here you will use your pre-trained random forest to do this job.

Non-Maximum Suppression After getting all the classification results from random forest for those bounding boxes, we can filter out some proposals with low confidence. But still, there might be multiple bounding boxes with high confidence are kept, as shown in Fig.5. So we should use NMS to filter out those bounding boxes with large overlaps with each other, and only the one with highest confidence is kept.

Evaluate the detection result Now you have the detection result, and you need to quantitatively evaluate how good it is. We follow the metrics used in **pascal visual object classes challenge**[10] to evaluate the precision and recall with regarding to the ground truth. You need to compute the IOU (insection over union) ratio between the predicted bounding boxes and ground truth bounding boxes, and use a threshold (like 0.5) to verify the prediction is correct or wrong. Then you need to compute the precision and recall.Precision is the number of corrected predicted bounding boxes divided by the number of predicted bounding boxes. Recall is the number of corrected predicted bounding boxes divided by the number of ground truth bounding boxes. Note that by adjusting the threshold value for confidence, we can get different number of predictions, hence a series of precision/recall can be retrieved. You should recorded all the precisions and recalls under difference confidence threshold, and then plot a PR curve as shown in Fig. 6.

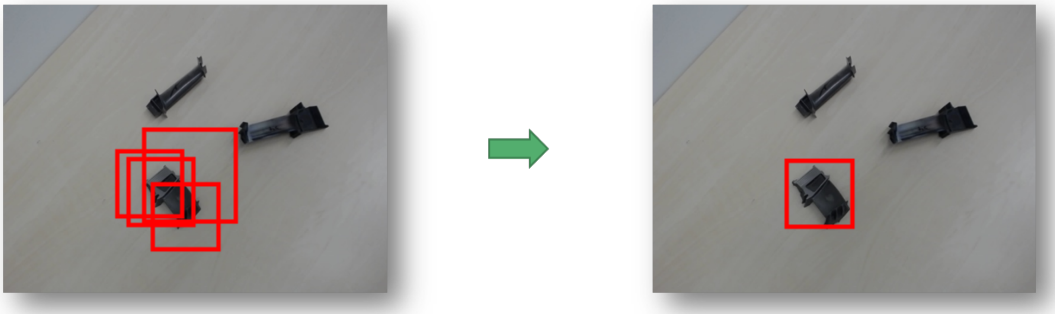


Figure 5: An illustration of nms result

In this task, you need to implement a complete pipeline of object detection using HOG descriptors and Random Forest, report the final precision/recall results on the test dataset, and also show some qualitative results as in Fig.4.

[10] http://homepages.inf.ed.ac.uk/ckiw/postscript/ijcv_voc09.pdf

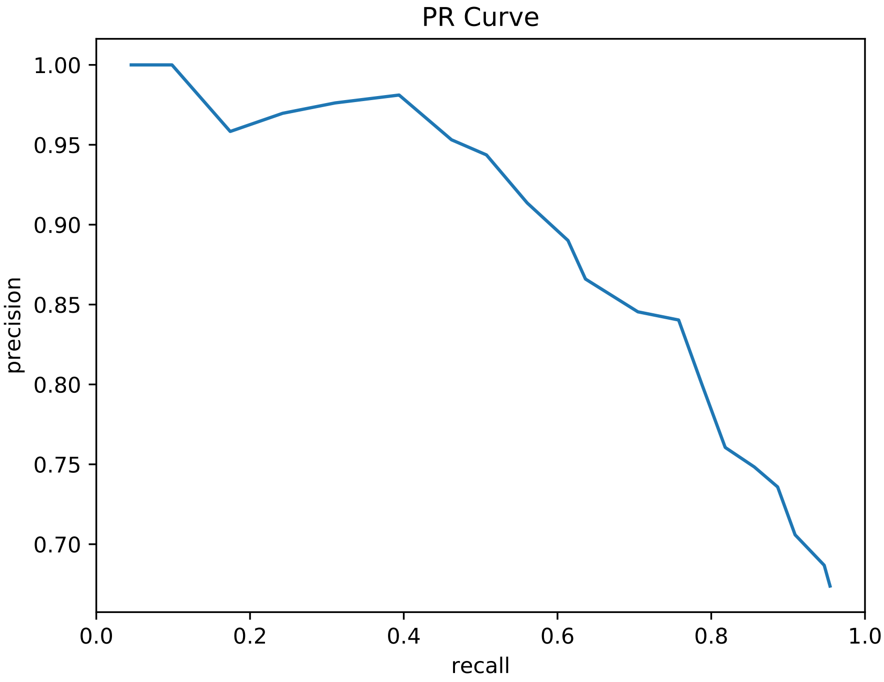


Figure 6: An example of PR Curve for detection result