Object Detection

Yali Amit[1], Pedro Felzenszwalb[2], and Ross Girshick[3]
[1]Department of Computer Science, University of Chicago, Chicago, IL, USA
[2]School of Engineering, Brown University, Providence, RI, USA
[3]Facebook AI Research, Menlo Park, CA, USA

Related Concepts

▶ Object Recognition
▶ Image Classification

Definition

Object detection involves detecting instances of objects from one or several classes in an image.

Background

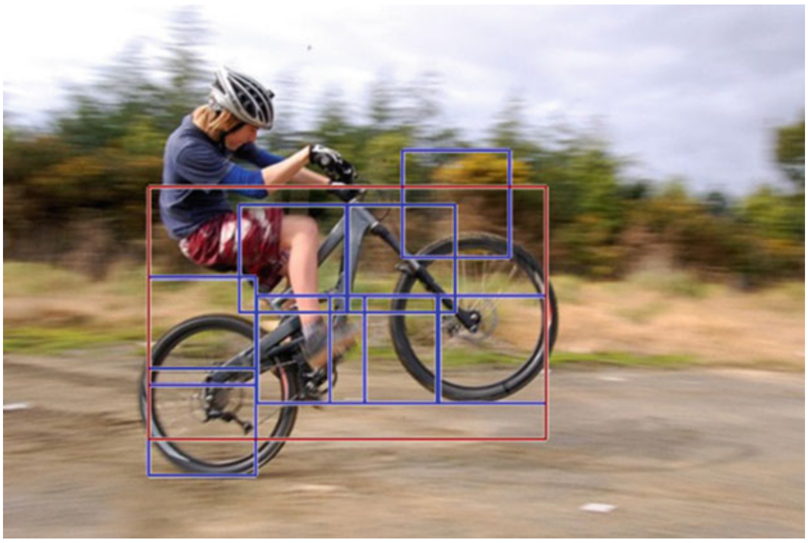
The goal of object detection is to detect all instances of objects from one or several known classes, such as people, cars, or faces in an image. Typically only a small number of objects are present in the image, but there is a very large number of possible locations and scales at which they can occur and that need to somehow be explored.

Each detection is reported with some form of *pose* information. This could be as simple as the location of the object, a location and scale, a bounding box, or a segmentation mask. In other situations the pose information is more detailed and contains the parameters of a linear or nonlinear transformation. For example a face detector may compute the locations of the eyes, nose, and mouth, in addition to the bounding box of the face. An example of a bicycle detection that specifies the locations of certain parts is shown in Fig. 1. The pose could also be defined by a three-dimensional transformation specifying the location of the object relative to the camera.

Object detection systems construct a model for an object class from a set of training examples. In the case of a fixed rigid object, only one example may be needed, but more generally multiple training examples (often hundreds or thousands) are necessary to capture certain aspects of class variability. Broadly speaking, less training data is needed when more information about class variability can be explicitly built into the model. However, it may be difficult to specify models that capture the vast variability found in images. An alternative approach is to use methods such as convolutional neural networks [1] that learn about class variability from large datasets.

Object detection approaches typically fall into one of two major categories, *generative* methods (see, e.g., [2–6]) and *discriminative* methods (see, e.g., [7–11]). A generative method consists of a probability model for the pose variability of the objects together with an appearance model:

© Springer Nature Switzerland AG 2020
K. Ikeuchi, *Computer Vision*,
https://doi.org/10.1007/978-3-030-03243-2_660-1



Object Detection, Fig. 1 A bicycle detection specified in terms of the locations of certain parts

a probability model for the image appearance conditional on a given pose, together with a model for background, i.e., non-object images. The model parameters can be estimated from training data, and the decisions are based on ratios of posterior probabilities. A discriminative method typically builds a classifier that can discriminate between images (or sub-images) containing instances of the target object classes and those not containing them. The parameters of the classifier are selected to minimize mistakes on the training data, often with a regularization bias to avoid overfitting.

Other distinctions among detection algorithms have to do with the *computational* tools used to scan the entire image or search over possible poses, the type of image *representation* with which the models are constructed, and what type and how much training data is required to build a model.

Theory

Images of objects from a particular class are highly variable. One source of variation is the actual imaging process. Changes in illumination and changes in camera position as well as digitization artifacts all produce significant variations in image appearance, even in a static scene. The second source of variation is due to the intrinsic appearance variability of objects within a class, even assuming no variation in the imaging process. For example, people have different shapes and wear a variety of clothes, while the handwritten digit 7 can be written with or without a line through the middle, with different slants, stroke widths, etc. The challenge is to develop detection algorithms that are *invariant* with respect to these variations and are computationally efficient.

Invariance

The brute force approach to invariance assumes training data is plentiful and represents the entire range of object variability. Invariance is implicitly learned from the data while training the models. In recent years, with the increase in annotated dataset size and computational acceleration using GPUs, this has been the approach of choice in the context of the multi-layer convolutional neural network paradigm, as discussed later in this article.

When training data is limited, it is necessary to build invariance into the models. There are two complementary methods to achieve this. One involves computing invariant functions and features; the other involves searching over latent variables. Most algorithms contain a combination of these approaches. For example many algorithms choose to apply local transformations to pixel intensities in such a way that the transformed values are invariant to a range of illumination conditions and small geometric variations. These local transformations lead to *features*, and the array of feature values is the *feature map*. More significant transformations are often handled through explicit search of latent variables or by learning the remaining variability from training data.

Invariant functions and features This method constructs functions of the data that are invariant with respect to the types of variability described above and can still distinguish between object and background images. This may prove difficult if object variability is extensive. Invariant functions that produce the same output no matter the pose and appearance of the object necessarily have less discriminative power.

There are two common types of operations leading to invariant functions. The first involves computing local features that are invariant to certain image transformations. The second operation involves computing geometric quantities that are invariant to some or all three-dimensional pose variation. For example the cross-ratio among distinguished points is a projective invariant that has been used to recognize rigid objects (see, e.g., [12]).

An example of a local feature, invariant to certain photometric variations and changes in illumination, is the *direction* of the image gradient, from which a variety of *edge* features can be computed. More complex features capture the appearance of small image patches and are often computed from edge features. An example would be the histogram of gradient (HOG) features [10]. Local features are usually computed at a *dense* grid of locations in the image, leading to a dense feature map. Features such as HOG were designed by practitioners based on a variety of considerations involving the desired invariance and at times were motivated by certain analogies to biological processing in the visual system. An alternative approach, implemented in multi-layer convolutional neural networks, learns the local features as well as intermediate and higher-level features as part of the training process. Interestingly, the low-level features learned by such networks often resemble oriented edge detectors, like the designed features.

Local pooling of features is commonly used to introduce some degree of invariance to small geometric variations. A typical example is the *max* or *sum* operation [2, 13]. In this case a quantity that is to be computed at a pixel is replaced by the maximum or sum of the quantity in a neighborhood of the pixel. When the region is extended over the entire window, the result is a *bag of features* model [14], which counts the number of binary features of different types that occur within a window. In this case all spatial information is lost, leading to models that are invariant to fairly large geometric transformations.

For computational reasons it is often useful to sparsify the feature map by applying local decisions to find a small set of *interest points*. The assumption is that only certain features are useful (or necessary) for object detection. The approach yields *sparse* feature maps that can be processed much more efficiently. Examples of commonly used sparse features are SIFT descriptors [15], corner detectors, and edge conjunctions [2]. One drawback of sparse features is that hard decisions are being made on their presence, and if some are missed, an algorithm may fail to detect an instance of the object.

Note that it is possible to predefine a very large family of features that is never fully computed, rather, in training an informative subset is selected that can produce the required classification for a particular object class. One example are the Haar features that compute differences of intensity averages in adjacent rectangles of varying sizes and locations [8]. Another example are geometric edge arrangements of increasing complexity.

Latent variables An explicit parameterization of the object variability can be defined via *latent* variables that are not directly observable from the image data. These are not necessarily needed for the final report on the object detections, but their values simplify the solution of the detection problem. For example to detect faces at a range of orientations, at each candidate region, one could decide, for *each* possible orientation, whether or not the region contains a face at that orientation. In general a set Θ defines latent parameters that could capture global illumination parameters, a linear or nonlinear map from a model domain into the image domain, or specify the locations of a finite set of object parts. The last case is common in part-based models where latent part placements are used to decide if the object might be present at a particular location in the image [11]. The set of possible latent values, Θ, can be quite large or infinite. This leads to computational challenges that have been addressed by a variety of methods including coarse-to-fine computation, dynamic programming, and geometric alignment.

Detection via Classification

The most common approach to object detection reduces the problem to one of classification. Consider the problem of detecting instances from one object class of fixed size but varying positions in the image. Let W denote a reference window size that an instance of the object would occupy. Let L denote a grid of locations in the image. Let X_{s+W} denote the image features in a window (sub-image) with top-left corner at $s \in L$. One can reduce the object detection problem to binary classification as follows. For each location $s \in L$ classify X_{s+W} into two possible classes corresponding to windows that contain an object and windows that do not contain an object. The sliding-window approach to object detection involves explicitly considering and classifying every possible window. Note that the same approach can be used to detect objects of different sizes by considering different window sizes or alternatively windows of fixed size at different levels of resolutions in an image pyramid. Clearly the number of windows where the classifier needs to be computed can be prohibitive. Many computational approaches find ways to narrow down the number of windows where the classifier is implemented.

Generative Models

A general framework for object detection using generative models involves modeling two distributions. A distribution $p(\theta; \eta_p)$ is defined on the possible latent pose parameters $\theta \in \Theta$. This distribution captures assumptions on which poses are more or less likely. An appearance model defines, the distribution of the image features in a window *conditional* on the pose, $p(X_{s+W}|object, \theta; \eta_a)$. The appearance model might be defined by a *template* specifying the probability of observing certain features at each location in the detection window under a canonical choice for the object pose, while θ specifies a transformation of the template. Warping the template according to θ leads to probabilities for observing certain features at each location in X_{s+W} [2, 3, 5].

Training data with images of the object are used to estimate the parameters η_p and η_a. Note that the images do not normally come with information about the latent pose variables θ, unless annotation is provided. Estimation thus requires inference methods that handle unobserved variables, for example, the different variants of the expectation maximization algorithm [4, 5]. In some cases a probability model for background images is estimated as well using large numbers of training examples of images not containing the object.

The basic detection algorithm then scans each candidate window in the image, computes the most likely pose under the object model, and obtains the "posterior odds," i.e., the ratio between the conditional probability of the window under the object hypothesis at the optimal pose and the conditional probability of the window under the background hypothesis. This ratio is then compared to a threshold τ to decide if the window contains an instance of the object

$$\frac{p(X_{s+W}|\text{object}, \theta; \eta_a)p(\theta; \eta_p)}{p(X_{s+W}|\text{background})} > \tau.$$

When no background model has been trained offline, a simple *adaptive* background model can be estimated online for each window being tested.

In this case no background training data is needed [5]. Alternative background models involve sub-collections of parts of the object model [16].

Discriminative Models

If no explicit latent pose variables are used, the underlying assumption is that the training data is sufficiently rich to provide a sample of the entire variation of object appearance. The discriminative approach trains a standard classifier to discriminate between image windows containing the target object classes and a broad background class. This is done using large amounts of data from the object classes and from background. Many classifier types have been used, including neural networks, SVMs, boosted decision trees, and radial basis functions.

Cascades Because of the large size of the background population and its complexity, discriminative methods are often organized in *cascades* [8]. An initial classifier is trained to distinguish between the object and a manageable amount of background data. The classifier is designed to have very few false negatives at the price of a larger number of false positives. Then a large number of background examples are evaluated, and the misclassified ones are collected to form a new background data set. Once a sufficient number of such false positives is accumulated, a new classifier is trained to discriminate between the original object data and the new "harder" background data. Again this classifier is designed to have no false negatives. This process can be continued several times.

At detection time the classifiers in the cascade are applied sequentially. Once a window is classified as background the testing terminates with the background label. If the object label is chosen, the next classifier in the cascade is applied. Only windows that are classified as object by all classifiers in the cascade are labelled as object by the cascade.

Pose variables Certain discriminative models can also be implemented with latent pose parameters [11]. Assume a generic classifier defined in terms of a space of classifier functions $f(x;u)$ parameterized by u. Usual training of a discriminative model consists of solving an equation of the form

$$\min_u \sum_{i=1}^{n} D(y_i, f(x_i;u)) + C(u),$$

for some regularization term $C(u)$ which prevents overfitting and a loss function D measuring the distance between the classifier output $f(x_i;u)$ and the ground truth label $y_i = 1$ for object and $y_i = 0$ for background.

The minimization above can be replaced by

$$\min_u \sum_{y_i=1} \min_{\theta\in\Theta} D(1, f(\theta(x_i);u)) + \sum_{y_i=0} \max_{\theta\in\Theta} D(0, f(\theta(x_i);u)) + C(u).$$

Here $\theta(x)$ defines a transformation of the example x. Intuitively for a positive example, one would like there to be some transformation under which x_i is classified as object, while for a negative example one would like it to be the case that there is no transformation under which x_i is classified as object.

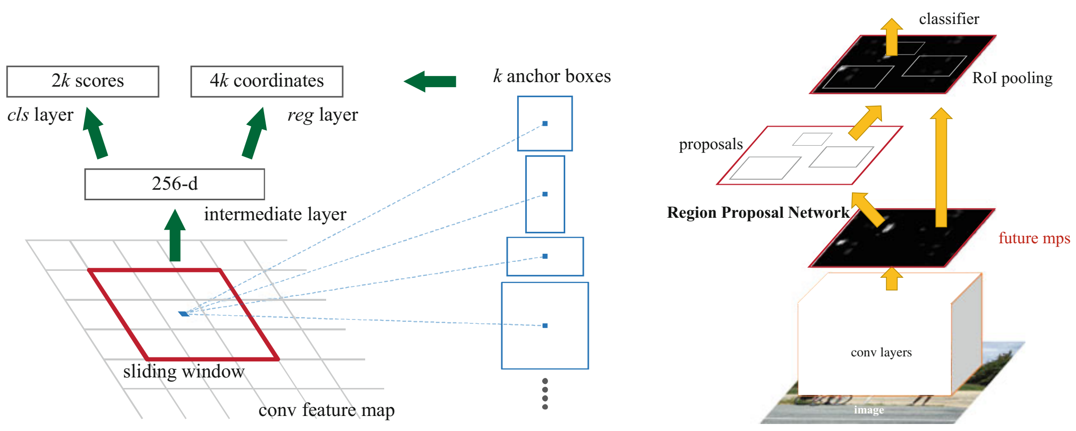
Convolutional neural networks Due to the limitations of low-level local features and the difficulty of manually specifying higher-level features, neural networks have become increasingly popular as a method to learn effective feature representations, from low-level to high-level, using large annotated datasets. These networks are composed of a hierarchy of layers indexed by grids of decreasing resolution. The input layer is the raw pixels of the input image. Each subsequent layer computes a vector output at each grid point using a list of local filters applied to the data in the preceding layer. This linear operation is typically followed by a nonlinear operation applied coordinate-wise and at certain layers the grid resolution is reduced by subsampling following a local max or averaging operation. The network terminates in one or more output layers that make predictions according to the design of the model (e.g., an object category classifier and a pose estimator, if the model outputs pose information).

All of the filter coefficients and the parameters of the output layers are trainable. Training is done through stochastic gradient descent on a loss function defined in terms of the output layers and the labels in the dataset. Thus the network jointly learns linear classifiers and a complex hierarchy of nonlinear features that yield the final feature representation. Such networks were demonstrated to work for large-scale image classification tasks [17] and then subsequently for the more complex task of object detection [18].

Networks for image classification have a relatively straightforward design since they terminate with a single classification output for the entire image. Networks for object detection involve additional components that are designed to address the more complex nature of object detection. There are two dominant designs: *one stage* (e.g., [19]) and *two stage* (e.g., [20]), both of which are based on a sliding-window approach, described next.

In both of these designs, two convolutional sub-networks are applied in parallel. At each location on a relatively coarse feature grid, one of these sub-networks acts as a classifier and the other as a pose predictor (see Fig. 2 left "*cls* layer" and "*reg* layer," for classification and box regression, respectively). The pose estimator predicts a relative shift and scaling of a detection window, while the classifier predicts if it is an object or background. By creating multiple pairs of such layers, with each pair specializing to a window of a specific size and aspect ratio, the set of predefined sliding windows can better approximate the set of all possible image windows. The pose estimator is tasked with predicting the residual error between the quantized windows and the ground-truth object bounding boxes. In the first design paradigm, often termed "one-stage" methods, the sliding window classifier makes a multi-class prediction over the set of all object categories and background. These predictions, together with the refined windows, comprise the output of the model.

In the second design paradigm, the sliding-window classifier performs two-class classification between object (of any category) *vs.* background. The refined windows are then used as candidate object locations, often called regions of interest (RoIs), for a subsequent classification and window refinement stage. This second stage, as is typical in cascaded processing, only receives high-scoring RoIs from the object *vs.* background classifier. The input features to the second stage are typically computed by extracting features within each RoI from a feature map.



Object Detection, Fig. 2 Left: A sliding-window Region Proposal Network (RPN) that performs classification ("*cls* layer") and window shift and scaling regression ("*reg* layer") for a set of reference windows ("anchor boxes") at each grid position. Right: The Faster R-CNN system that uses RPN to generate candidate object proposals for processing in a second stage as a part of an overall convolutional neural network for object detection. (The figure is reproduced with permission from [20])

The RoI feature extraction process may involve quantizing the RoI coordinates and using max pooling [20] or bilinear interpolation of the feature map without performing quantization [21]. The output of the second stage is a multi-class classification prediction and window refinement (shift and scaling) for each RoI. See Fig. 2 right. Two-stage methods can also be extended in a straightforward manner to predict a binary segmentation mask for each RoI [21]. In either case, all parameters of these networks are trained jointly using stochastic gradient descent on a multi-task loss function that includes terms for classification and pose estimation.

Computational Methods

The basic detection process consists of searching over pose parameters to classify each hypothesis. At a minimum this usually involves searching over locations and sizes and is clearly a very intensive computation. There are a number of methods to make it more efficient.

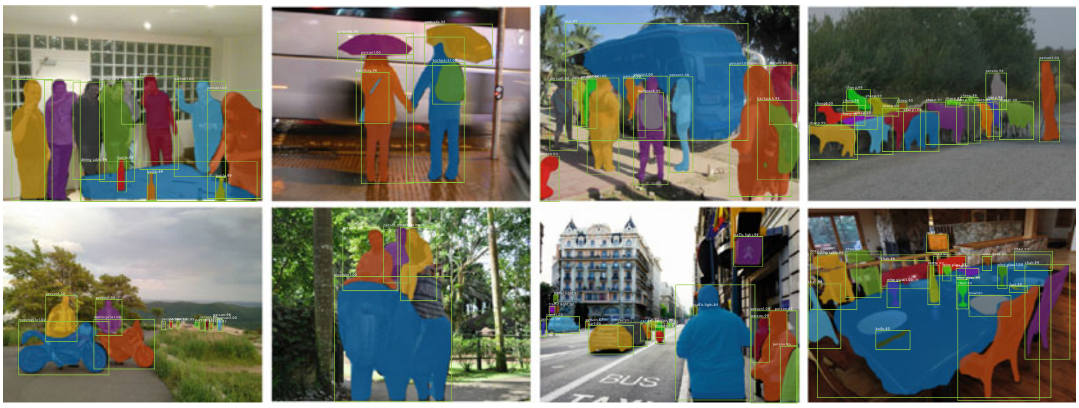
Sparse features When sparse features are used, it is possible to focus the computation only in regions around features. The two main approaches that take advantage of this sparsity are *alignment* [22] and the *generalized Hough transform*. Alignment uses information regarding the relative locations of the features on the object. In this case the locations of some features determine the possible locations of the other features. Various search methods enable a quick decision on whether a sufficient number of features were found to declare object, or not. The Hough transform typically uses information on the location of each feature type relative to some reference point in the object. Each detected feature votes with some weight for a set of candidate locations of the reference point. Locations with a sufficiently large sum of weighted votes determine detections. This idea can also be generalized to include identification of scale as well. The voting weights can be obtained either through discriminative training or through generative training [2].

Cascades As mentioned above the cascade method trains a sequence of classifiers with successively more difficult background data. Each such classifier is usually designed to be simple and computationally efficient. When the data in the window X_{s+W} is declared background by any classifier of the cascade, the decision is final, and the computation proceeds to the next window. Since most background windows are rejected early in the cascade, most of the windows in the image are processed very quickly.

Coarse to fine The cascade method can be viewed as a coarse to fine decomposition of background that gradually makes finer and finer discriminations between object and background images that have significant resemblance to the object. An alternative is to create a coarse to fine decomposition of object poses [9]. In this case it is possible to train classifiers that can rule out a large subset of the pose space in a single step. A general setting involves a rooted tree where the leaves correspond to individual detections and internal nodes store classifiers that quickly rule out all detections below a particular node. The idea is closely related to branch-and-bound methods [14] that use admissible lower-bounds to search a space of transformations or hypotheses.

Dynamic programming There are a number of object detection algorithms that represent objects by a collection of parts arranged in deformable configurations or as hierarchies of such arrangements of parts of increasing complexity. When the hierarchies and the arrangements have the appropriate structure, dynamic programming methods can be used to efficiently search over the spaces of arrangements [2, 3].

Convolutional neural networks Object detection systems based on convolutional networks make use of many classical techniques for reducing computation, including cascades as previously described. Coarse-to-fine approaches are also common. By progressively reducing spatial resolution, these networks operate on a coarse grid of object locations. The loss in resolution is



Object Detection, Fig. 3 Bounding boxes and segmentation masks produced by Mask R-CNN. The system is trained to detect and segment 80 object categories from the COCO dataset. (Reproduced with permission from [21])

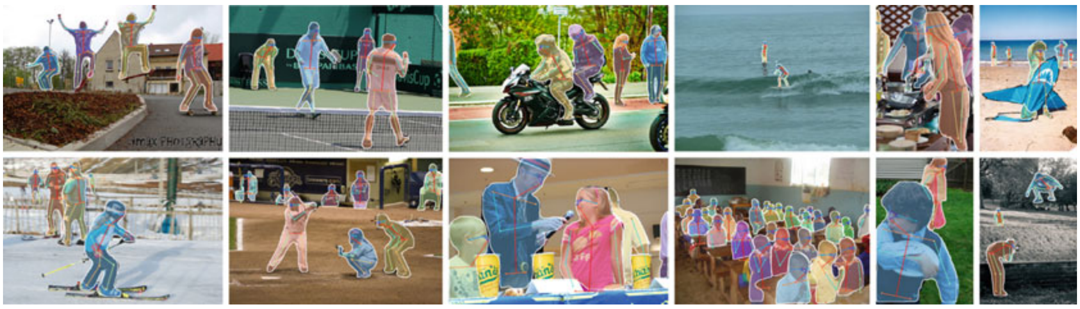
compensated by predicting pose parameters that recover some of the quantization error, resulting in fine predictions. The progressive reduction in spatial resolution can also be used to efficiently construct a multi-scale pyramid representation from a single input image scale [23], which is more efficient than processing multiple input image scales independently. These networks also share nearly all the computation of the hierarchy of features across all object categories, rather than retraining a separate hierarchy for each one *vs.* rest binary classification models, one for each object category. The shared computation enables such networks to scale to thousands of object categories; the marginal per category cost grows linearly but is small relative to the computation shared between categories. Finally, typical convolutional networks can be accelerated both algorithmically via fast convolution methods and with specialized hardware that makes extensive use of parallel computation (e.g., GPUs).

Application

Object detection methods have a wide range of applications in a variety of areas including robotics, medical image analysis, surveillance, and human computer interaction. Current methods work reasonably well in constrained domains but still rely on thousands of training examples per category in order to achieve reasonable results.

A popular benchmark for object detection is the COCO object detection challenge [24]. The goal of the challenge is to detect objects from 80 common categories such as people, cars, horses, and tables in photographs. The challenge has attracted significant attention in the computer vision community over the last few years, and the performance of the best systems has been steadily increasing each year by a significant amount.

Detection methods that segment objects are also steadily improving and are now deployed for various applications, particularly in augmented and virtual reality scenarios. Figure 3 shows example output of Mask R-CNN [21]. Models that additionally predict human joint locations, in addition to bounding boxes and segmentation masks, are also useful in many applications including video conferencing systems that can automatically keep participants framed within the video steam. Mask R-CNN can be extended to predict human pose, as illustrated in Fig. 4.



Object Detection, Fig. 4 Bounding boxes, segmentation masks, and joint positions produced by Mask R-CNN. The system is trained to detect, segment, and predict pose on the *person* categories from the COCO dataset. (Reproduced with permission from [21])

References

1. LeCun Y, Boser B, Denker JS, Henderson D, Howard RE, Hubbard W, Jackel LD (1989) Backpropagation applied to handwritten zip code recognition. Neural Comput 1(4):541–551
2. Amit Y (2002) 2D object detection and recognition: models, algorithms and networks. MIT Press, Cambridge
3. Felzenszwalb P, Huttenlocher D (2005) Pictorial structures for object recognition. Int J Comput Vis 61(1):55–79
4. Fergus R, Perona P, Zisserman A (2003) Object class recognition by unsupervised scale-invariant learning. In: IEEE CVPR 2003
5. Amit Y, Trouvé A (2007) POP: patchwork of parts models for object recognition. Int J Comput Vis 75(2):267–282
6. Jin Y, Geman S (2006) Context and hierarchy in a probabilistic image model. In: IEEE CVPR 2006
7. Rowley HA, Baluja S, Kanade T (1998) Neural network-based face detection. IEEE Trans Pattern Anal Mach Intell 20(1):23–38
8. Viola P, Jones MJ (2004) Robust real time face detection. Int J Comput Vis 57(2):137–154
9. Fleuret F, Geman D (2001) Coarse-to-fine face detection. Int J Comput Vis 41(1–2):85–107
10. Dalal N, Triggs B (2005) Histograms of oriented gradients for human detection. In: IEEE CVPR 2005
11. Felzenszwalb P, Girshick R, McAllester D, Ramanan D (2010) Object detection with discriminatively trained part based models. IEEE Trans Pattern Anal Mach Intell 32(9):1627–1645
12. Reiss T (1993) Recognizing planar objects using invariant image features. Springer, Berlin
13. Riesenhuber M, Poggio T (2000) Models of object recognition. Nat Neurosci 3(11s):1199–1204
14. Lampert C, Blaschko M, Hofmann T (2009) Efficient subwindow search: a branch and bound framework for object localization. IEEE Trans Pattern Anal Mach Intell 31(12):2129–2142
15. Lowe D (2004) Distinctive image features from scale-invariant keypoints. Int J Comput Vis 60(2):91–110
16. Chang LB, Jin Y, Zhang W, Borenstein E, Geman S (2011) Context computation, and optimal ROC performance in hierarchical models. Int J Comput Vis 93(2):117–140
17. Krizhevsky A, Sutskever I, Hinton G (2012) Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems
18. Girshick R, Donahue J, Darrell T, Malik J (2014) Rich feature hierarchies for accurate object detection and semantic segmentation. In: IEEE CVPR 2014
19. Liu W, Anguelov D, Erhan D, Szegedy C, Reed S, Fu CY, Berg AC (2016) SSD: single shot multibox detector. In: ECCV 2016
20. Ren S, He K, Girshick R, Sun J (2015) Faster R-CNN: towards real-time object detection with region proposal networks. In: Advances in neural information processing systems
21. He K, Gkioxari G, Dollár P, Girshick R (2017) Mask R-CNN. In: ICCV 2017
22. Ullman S (1996) High-level vision. MIT Press, Cambridge, MA
23. Lin TY, Dollár P, Girshick R, He K, Hariharan B, Belongie S (2017) Feature pyramid networks for object detection. In: IEEE CVPR 2017
24. Lin TY, Maire M, Belongie S, Hays J, Perona P, Ramanan D, Dollár P, Zitnick CL (2014) Microsoft COCO: common objects in context. In: ECCV 2014