A Guideline for Object Detection Using Convolutional Neural Networks

Xingguo Zhang, Guoyue Chen(✉), Kazuki Saruta, and Yuki Terata

Akita Prefectural University, 84-4 Aza Ebinokuchi Tsuchiya, Yurihonjo City 015-0055, Japan
chen@akita-pu.ac.jp

Abstract. The main purpose of object detection is to detect and locate specific targets from images. The traditional detection methods are usually complex and require prior knowledge of the detection target. In this paper, we will introduce how to use convolutional neural networks to perform object detection from image. This is one of the important areas of computer vision. In order to build up to object detection, we first learn about how we can get the object localization or landmark by a neural network. And then I will give the detail of sliding windows detection algorithm and introduce how to use the convolutional implementation of sliding windows to speed up the process. Then we will introduce the transfer learning and how to prepare your own learning data for training networks.

Keywords: Object detection · Bounding box · Transfer learning

1 Introduction

The main purpose of object detection is to detect and locate specific targets from images. The traditional detection model usually represents the target object by handcraft features and then predicts the category and location by classifier [1]. These methods are usually intuitive and easy to understand. However, the design of these methods is often complex and requires prior knowledge of the detection target. In addition, it is highly dependent on specific tasks and has poor portability. Once the detection target changes significantly, it is necessary to redesign the algorithm.

In recent years, with the improvement of hardware and algorithm, convolutional neural networks (ConvNet) have achieved great success in image classification, which led researchers to study its effects in other areas of computer vision. The early algorithm based on deep learning is generally divided into three steps, selecting the object candidate region (proposal), extracting the features of the candidate region, and finally putting the extracted features into the classifier to predict the object category. In 2014, Girshick designed the R-CNN model [2] based on ConvNet. The mean average precision (mAP) of this model in the object detection task of PASCAL VOC [3] was 62.4%, which was nearly 20% higher than the traditional algorithm. He et al. proposed Spatial Pyramid Pooling Net (SPP net) [4], which only performed convolution operation on the whole picture once and added pyramid pooling layer after convolution layer, so as to fix the feature map to the required size. This greatly saves time, reducing

Q. Liang et al. (Eds.): *Artificial Intelligence in China*, LNEE 572, pp. 157–164, 2020.
https://doi.org/10.1007/978-981-15-0187-6_18

the processing time of single-frame image to 2.5 s. However, SPP net failed to optimize the hard disk storage space of R-CNN. To address the issue, Girshic proposed the Fast R-CNN [5].

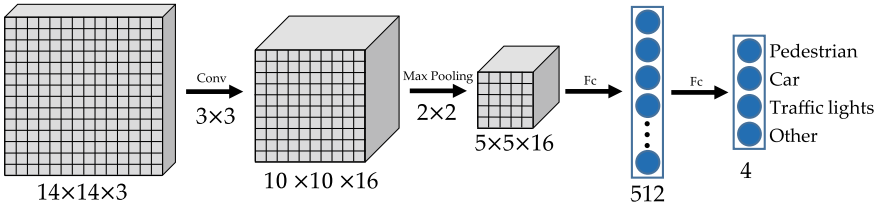
Fast R-CNN proposed a multitask loss function, which adds the loss of target positioning to the traditional loss function to correct the position information. After that, Ren et al. designed the structure of Faster R-CNN [6] and used the ConvNet to generate candidate regions directly. Faster R-CNN realizes end-to-end training and realizes real-time detection. On this basis, some improved methods such as region-based fully convolutional networks (R-FCN) [7], Mask R-CNN [8] have been proposed successively. However, most of these methods are based on the three-step strategy of candidate region selection, feature extraction, and classification.

After that, some researchers proposed the regression-based object detection method, which was represented by YOLO [9, 10], SSD [11], etc. These methods only use one ConvNet for detection and use the regression method to correct the object location, making the detection speed much faster than the candidate region-based detection methods such as Faster R-CNN. However, their disadvantages are large localization error, which is mainly because it is very difficult to get an accurate location by regression directly in the absence of proposal regions. In the training, there will be a wide range jitter of the bounding box and the losses function is hard to converge. In addition, the detection performance often becomes poor when multiple adjacent small objects appear. The rest of this paper, I will focus on the key techniques used in object detection using ConvNet.

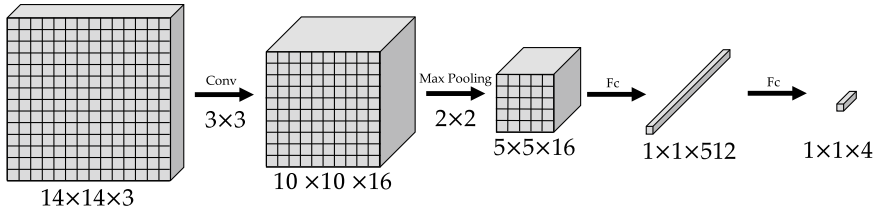
2 Convolutional Implementation of Sliding Windows

In this section, we will introduce to build a convolutional neural network to implementation of sliding windows. The traditional sliding window method is very slow due to a lot of repeated convolution computation. In this part, let us see how we can improve the processing speed by using the convolutional implementation of sliding windows.

Firstly, let us see how we can transform fully connected layers into convolutional layers. For illustrative purposes, let us see a simple network as shown in Fig. 1a. Our object detection algorithm inputs $14 \times 14 \times 3$ images, then 16 of 3×3 filters are used to map the inputs images from $14 \times 14 \times 3$ to $10 \times 10 \times 16$, then does a 2×2 max pooling to reduce it to $4 \times 4 \times 16$, then has a fully connected layer to connect 512 units, and then finally, outputs y using a softmax layer. Here, we assume that there are four categories, which are pedestrian, car, traffic light, and background. Then y have four units, corresponding to the crossed probabilities of the four classes that the softmax unit is classifying among. And the four classes could be pedestrian, car, traffic lights, or background.



(a) Object classification with fully connected layers



(b) Object classification without fully connected layers

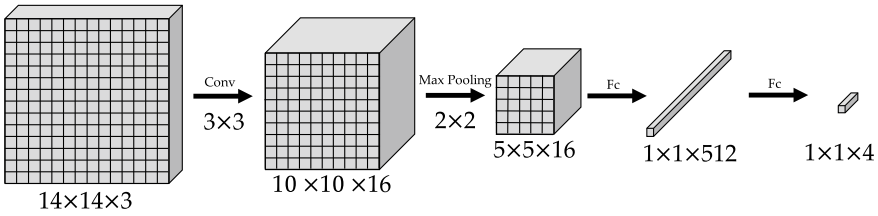
Fig. 1. Convolutional implementation for fully connected layers

Now, let us show you how these fully connected layers can be turned into convolutional layers. As shown in Fig. 1b, the first few layers of the ConvNet have the same structure. After that, to implement the convolutional layer, we use 512 of $5 \times 5 \times 16$ filters to do the convolution, and the output dimension is going to be $1 \times 1 \times 512$. In mathematically, this is the same as a fully connected layer, because each of these 512 nodes has a filter of dimension $5 \times 5 \times 16$, and so each of those 512 values is some arbitrary linear function of these $5 \times 5 \times 16$ activations from the previous layer.

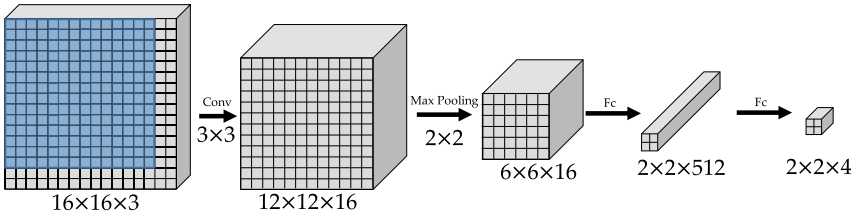
Finally, we are going to use four of $1 \times 1 \times 512$ filters by a softmax activation to get a $1 \times 1 \times 4$ volume as the output of this network. So, this shows how you can take these fully connected layers and implement those using convolutional layers, and these fully connected layers are now implemented as $1 \times 1 \times 512$ and $1 \times 1 \times 4$ volumes.

When the size of the test image we inputted changed to $16 \times 16 \times 3$, assume our trained detection window is still 14×14 (shown in Fig. 2a). So in the original sliding windows algorithm, you might want to input the first 14×14 regions into a ConvNet and run that once to generate a classification 0 or 1. Then slide the window to the right by a stride = 2 pixels to get the second rectangular area, and run the whole ConvNet for this window to get another label 0 or 1. Then repeat this process by slide the window until get the output of lower right window. You will find for this small input image, we run this ConvNet from above four times in order to get four labels. But we can see that many of the operations by these four ConvNet are highly duplicated. So what the convolutional implementation of sliding windows does is it allows these four forward passes of the ConvNet to share a lot of computation.

As shown in Fig. 2b, you can take the ConvNet and just run it by the same $5 \times 5 \times 16$ filters with same parameters, and you can get a $12 \times 12 \times 16$ output volume, and then do the max pool same as before, get a $6 \times 6 \times 16$ output, run



(a) Trained object detector



(b) Object detection

Fig. 2. Object detection by convolutional implementation

through your same 400 of 5 × 5 filters to get a 2 × 2 × 400 volume output. So, now instead of a 1 × 1 × 400 volume, you can get a 2 × 2 × 400 volume. Do that one more time, now you are left with a 2 × 2 × 4 output volume instead of 1 × 1 × 4. So, in this final output layer, which turns out the upper left 1 × 1 × 4 subset gives you the result of running in the upper left corner 14 × 14 regions of input image, the upper right 1 × 1 × 4 volume gives you the upper right result and same argument for the lower left and right 1 × 1 × 4 volume. If you step through all the steps of the calculation, you will find that the value of each cell is the same as if you cut out each region and input it to ConvNet.

So, what this convolutional implementation does is you need not run forward propagation on four subsets of the input image independently. We only need to do forward propagation once for the entire image and share a lot of the computation in the regions of the image that is common. You can refer to [12] to get more details about this part.

3 Transfer Learning

When your network is ready, prepare training data for learning. The process of optimizing the neural network is to find the value of each layer of parameters that minimize the output of the loss function. But if you do a little bit of calculation, you will find that there are huge parameters in a neural network that needs to be trained, i. e, a medium-sized vgg-16 network [13] has at least 1.5×10^7 parameters.

If you have a new network that needs to be trained from scratch, then you have to prepare a lot of training samples, such as a database like ImageNet [14], MS COCO

[15], etc. And, you probably need a high-performance GPU computing cluster, and sometimes, this training takes several weeks or even months to make the output of the loss function converge.

So, how can we train neural networks to develop our own image processing applications without such a GPU computing cluster or lots of training samples? One solution is transfer learning. Transfer learning is to transfer the model parameters that have been learned from other places to the new model to help train the new model.

Due to the explosive growth of research related to neural network in recent years, people often associate transfer learning with training of neural network. These two concepts are unrelated at the beginning. Transfer learning is a branch of machine learning, and many transfer methods did not need to use neural network. But now, people find that deep neural network model has strong transferability. Because DNN is a hierarchical representation of data obtained through pretrain and then classified with high-level semantic classification. The low-level semantic features (such as texture, edge, color information, etc.) are at the bottom of the model. Such features are actually invariable in different classification tasks, and the real difference is the high-level features.

If you apply the same network architecture to implement a vision task, you can usually download weights that someone else has trained on the same network as initialization, rather than training it from random values. And use transfer learning to sort of transfer knowledge from some of these very large public data sets to our own problem. Usually, this transfer process can greatly reduce the convergence time of the model.

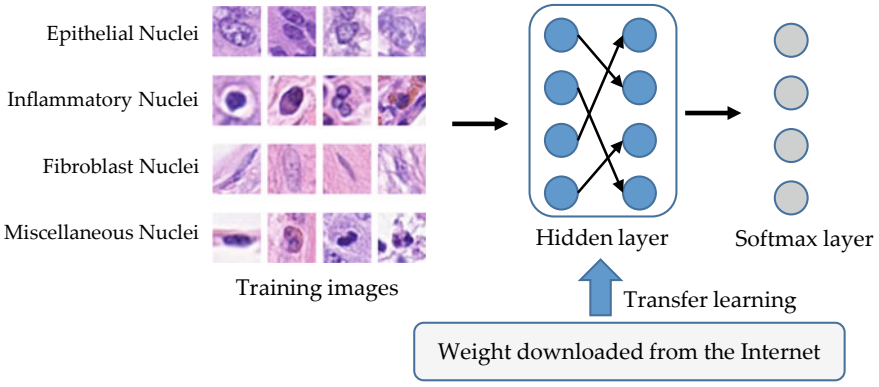


Fig. 3. A sample of transfer learning

Let us see an example of medical image recognition, as shown in Fig. 3. If we need building a cell detector to recognize colon cancer histology images, we were prepared to recognize four clinically meaningful cells: epithelial nuclei, inflammatory nuclei, fibroblasts, and miscellaneous nuclei. And we assume that each cropped image patch contains only one cell. So, we have a classification problem with four classes. But our

training set is small. Then we would better download some open-source implementation of a neural network and download not just the code, but also the weights of this network. There are a lot of networks; you can download that have been trained on, i.e., the ImageNet dataset, which has 1000 different classes. The network typically contains a softmax layer used as the output layer, which contains 1000 units. You usually need to modify the softmax layer and create your own softmax unit that outputs epithelial, inflammatory, fibroblasts, or miscellaneous.

Since our training dataset is small, we would better freeze the first few layers of the neural network and then just train the parameters associated with finial several layers. For most machine vision tasks, the first few layers of the network are usually used to extract common fundamental features of input images, such as edges, gradients, etc. And by using the pretrained weights, we can get very good performance even with a small data sets. Many current deep learning frameworks support using parameters to specify training or freezing the weights associated with a particular layer. This method is usually used to solve the problem of overfitting caused by insufficient training data. But if you have enough data, you could use the downloaded weights just as initialization to replace random initialization. And then you can do gradient descent training, updating all the weights in all the layers to train the whole network.

This is the transfer learning for the training of ConvNet. In practice, because many network weights are publicly available on the internet, you can easily download these open source weights that someone else has spent weeks working with a large computing device to initialize your network model. This will greatly improve your work efficiency. You can refer to [16] for more details.

4 Generate Training Dataset

When your network is built, you need to provide the image it will use to train a new detection classifier. Most deep neural network models require at least hundreds of images to train a detection classifier. Many computer vision researchers are willing to share their data online, i.e., ImageNet. And other researchers can also train their algorithms on these databases

To train a robust classifier, the training images should have random objects in the image along with the desired objects and should have a variety of backgrounds and lighting conditions. There should be some images where the desired object is partially obscured, overlapped with something else, or only halfway in the picture. Make sure the images are not too large. The larger the images are, the longer it will take to train the classifier. As my advice, the images size should be less than 200 KB each, and their resolution should not be more than 1280×720.

In practice, after you have all the pictures you need, we usually use 80% of them as the training images and 20% of them as testing images. Make sure there are a variety of pictures in both directories of the training and testing data.

With all the pictures gathered, it is time to label the desired objects in every picture. LabelImg [17] is a tool for labeling images, and its GitHub page has very clear instructions on how to install and use it. Once you have labeled each image, there will

be generated one *.xml file for each image in the images directory. These will be used to train the new object detection classifier.

5 Conclusion

Object detection has been developing rapidly in the field of computer vision and has repeatedly created amazing achievements. The ConvNet has a strong versatility and portability, and it is widely used in robot, autonomous driving, medical assistance, etc. This chapter focuses on several core problems in object detection: object localization, sliding windows object detection, and transfer learning. In addition, we also introduced how to classify and locate cells in medical image by ConvNet.

In summary, the success of ConvNet in recent years mainly depends on three pillars: data, model, and calculation power. A large amount of manually annotated data makes it possible to conduct supervised training. A deeper and larger model improves the recognition ability of the neural network. The combination with GPU and the rapid development of computer hardware makes large-scale training become time-saving and effective. However, the research on ConvNet is just beginning, and many aspects need further study.

At present, ConvNet needs training samples of tens of thousands or even millions of levels, and the training process for such a large number of samples is also extremely long. And in many areas, it is very expensive to obtain large numbers of precisely labeled samples. How to generate a good neural network from a small amount of data will be the future research direction. How to get satisfactory performance by train a small amount of dataset will be the future research direction.

In addition, the trend is that the deeper the network, the better the performance of ConvNet, and some networks even reach thousands of layers. However, as the network deepens, overfitting and gradient disappeared become more serious. Although the research such as residual network [18], etc., is devoted to solving such problems, the large model also limits the application of ConvNet on common devices, especially mobile devices. The ConvNet needs to optimize the structural design to find more efficient neurons and structural units.

References

1. Wang X, Han TX, Yan S (2009) An HOG-LBP human detector with partial occlusion handling. In: 2009 IEEE 12th international conference on computer vision, Sept 2009, pp 32–39
2. Girshick R, Donahue J, Darrell T, Malik J (2014) Rich feature hierarchies for accurate object detection and semantic segmentation. In: Proceedings of the IEEE computer society conference on computer vision and pattern recognition, pp 580–587
3. Everingham M, Gool L, Williams CKI, Winn J, Zisserman A (2009) The pascal visual object classes (VOC) challenge. Int J Comput Vis 88(2):303–338
4. He K, Zhang X, Ren S, Sun J (2015) Spatial pyramid pooling in deep convolutional networks for visual recognition. IEEE Trans Pattern Anal Mach Intell 346–361

5. Girshick R (2015) Fast R-CNN. In: Proceedings of the IEEE international conference on computer vision, pp 1440–1448
6. Ren S, He K, Girshick R, Sun J (2017) Faster R-CNN: towards real-time object detection with region proposal networks. IEEE Trans Pattern Anal Mach Intell 6:1137–1149
7. Dai J, Li Y, He K, Sun J (2016) R-FCN: object detection via region-based fully convolutional networks. Adv Neural Inf Process Syst 379–387
8. He K, Gkioxari G, Dollar P, Girshick R (2017) Mask R-CNN. In: Proceedings of the IEEE international conference on computer vision, pp 2980–2988
9. Redmon J, Divvala S, Girshick R, Farhadi A (2016) You only look once: unified, real-time object detection. In: Proceedings of IEEE conference on computer vision and pattern recognition, pp 779–788
10. Redmon J, Farhadi A (2017) YOLO9000: better, faster, stronger. In: IEEE conference on computer vision and pattern recognition, CVPR 2017, pp 7263–7271
11. Liu W, Anguelov D, Erhan D, Szegedy C, Reed S, Fu CY, Berg AC (2016) SSD: single shot multibox detector. In: European conference on computer vision, pp 21–37
12. Sermanet P, Eigen D, Zhang X, Mathieu M, Fergus R, LeCun Y (2013) OverFeat: integrated recognition, localization and detection using convolutional networks. arXiv Prepr. arXiv
13. Long J, Shelhamer E, Darrell T (2015) Fully convolutional networks for semantic segmentation. In: Proceedings of the IEEE conference on computer vision and pattern recognition
14. Deng J, Dong W, Socher R, Li L-J, Li K, Fei-Fei L (2009) ImageNet: a large-scale hierarchical image database. In: 2009 IEEE conference on computer vision and pattern recognition
15. Lin TY, Maire M, Belongie S, Hays J, Perona P, Ramanan D, Dollár P, Zitnick CL (2014) Microsoft COCO: common objects in context. In: Lecture Notes in Computer science (including subseries Lecture Notes in Artificial intelligence and lecture notes in bioinformatics)
16. Yosinski J, Clune J, Bengio Y, Lipson H (2014) How transferable are features in deep neural networks? Adv Neural Inf Process Syst 3320–3328
17. Tzutalin (2015) LabelImg. https://github.com/tzutalin/labelImg
18. He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition. In: IEEE conference on computer vision and pattern recognition, pp 770–778