Faster R-CNN

Yan-Frederic Thorsting
Matthias Greshake
Christian Dingkuhn

March 7, 2018

Abstract

Object detection is a computer technology prominent in the fields of computer vision and image processing. It thus does not come as a surprise that research in object detection is critical in the development of technologies that involves face detection and pedestrian detection, among other domains. The *Regional Convolutional Neural Network* (R-CNN) [1] is an early attempt to achieve object detection through the use of neural networks, it has since then been supplanted by the *Spatial Pyramid Pooling network* (SPPnet) [2] or improved into the *Fast R-CNN* [3], both in terms of speed and accuracy. In this work, we introduce a "toy"-size implementation of more recent *Faster R-CNN* of Ren et al. (2016) [4], which combines the so called *Region Proposal Network* (RPN) [4] with the Fast R-CNN for even better running time and accuracy score.

1 Introduction

Object detection being used in a wide range of domains, from face recognition to tracking a ball in a football match, developing fast and accurate methods to achieve it is therefore critical to the progress in the aforementioned domains.

The problem of object detection can be split into two smaller ones. The localization problem consists of localizing objects in the image, whereas assigning labels to the found objects amounts to solve the classification problem. Localization is especially important when the input image possibly contain more than one object. Without first localizing potential candidates, a traditional CNN would get confused with the multiple labels during the training, and as a result it may assign the wrong label to an object that just happened to frequently co-occur in the training data with the object to which that label actually corresponds. For example, if a CNN is trained to recognize the object banana in an image, but that in the training data bananas often co-occur with grocery bags, the CNN once trained may start to classify images featuring grocery bags as featuring bananas even though they may actually contain no banana at all.

Furthermore, localization is achieved by selecting the *Region of Interests* (RoIs) on the input image that have a high "objectness" score. Earlier version of the R-CNN use either *Selective Search* (SS), which greedily merges superpixels based on engineered low-level features [7], or *Edge Boxes* (EB), which use the number of contour wholly contained in a bounding box as object indicator [8], to generate the RoIs. In a CPU implementation, SS takes 2 s/img, whereas

EB takes 10× times less this value. The *Region Proposal Network* (RPN) of the Faster R-CNN on the other hand achieves localization at nearly no cost (10 ms/img) by taking advantage of the GPU and sharing convolutions at test-time [4]. The difference between the R-CNN and the Fast R-CNN in terms of object localization is that the former runs a CNN on top of each RoI, whereas the latter perform the feature extraction before generating the RoIs, thus running only one CNN over the entire image instead of many (∼2,000) [3]. By doing so, the Fast R-CNN actually share computation and greatly reduce the running time of the algorithm. For further processing of the RoIs in the Fast R-CNN, these are all resized and reshaped to the same size and form in the next (pooling) layer [3].

As for classification, the output of the multiple CNNs (for the R-CNN) or that of the resized and reshaped RoIs (for the Fast R-CNN) are fed into *fully connected* (fc) layers where they are classified by a SVM (for the R-CNN) or a softmax function (for the Fast R-CNN). From that point on, depending on the classifier's output, linear regression is applied to the RoIs' bounding boxes with high objectness score to refine their position [3].

All in all, the main difference between the Fast R-CNN and the Faster R-CNN is the latter's use of a RPN for localization instead of a SS or a EB. In the following section we will thus introduce and explain our own implementation of both the RPN and the Fast R-CNN, followed by the training of this Faster R-CNN on a toy-size problem. A third section will be on the different experiments we conducted with this Faster R-CNN. And last and not least, the fourth section of this paper will be to discuss the results of the experiments, the performance of the Faster R-CNN and how it could possibly be improved.

2 Data and Model Implementation

This implementation of the Faster R-CNN consists of a fixed VGG-16 model pre-trained on ILSVRC-2012, and custom-made RPN and Fast R-CNN. Code is available at ???.

2.1 Data Generation

For the training, the validation and the test phases, we used ???, ??? and ??? $256 \times 256 \times 1$ images respectively. Those images were automatically generated by randomly putting 2 to ??? images from the MNIST database together on an unified black background. The MNIST numbers were randomly distributed on each individual images and could possibly overlap. Because the used images are colorless, the third dimension of the data happens to be redundant, sharing the same value across all three channels. This was not changed however, as this format was conveniently fitting the design of the VGG. Furthermore, each MNIST collage had assigned to it a list of the form $[img_1, img_2, img_3, img_4]$, with each element corresponding to a seven-tuple of the form ($mnist\_number$, x_{center}, y_{center}, w, h, $angle$, $scale$). Each of these seven-tuples represent a MNIST image on the collage, with $mnist\_number$ standing for the represented number, x_{center} and y_center for the x and y coordinates of the image on the collage respectively, w for the width of the image, h for the height of the image, $angle$ for the angle of the image, and $scale$ for the scale of the image.

2.2 Region Proposal Network

As a localizer for the Faster R-CNN, the RPN takes an image (of any size) as input and outputs a set of rectangular RoIs, each with an objectness score. Unlike with the Faster R-CNN of Ren et al. [4] however, this network will not be sharing the convolution with the Fast R-CNN in an end-to-end manner.

To generate the RoIs, a small network is slid over the $16 \times 16 \times 512$ feature tensor output by the first convolutional (conv) layer. This network is fully connected to a 3×3 spatial window of the input conv feature map. Each sliding window is mapped to a lower-dimensional vector (512-d in our case). This vector is fed into two sibling fully-connected layers—a $1 \times 1 \times 512 \times 36$ box-regression layer and a $1 \times 1 \times 512 \times 18$ box-classification layer. The last dimension of the box-regression layer corresponds to the x (1-9) and y (10-18) coordinates, the width (19-27), and to the height (28-36) of the RoIs; whereas the last dimension of the box-classification layer corresponds to the objectness score (1-9) and non-objectness score (10-18) of the anchors. Because the mini-network operates in a sliding-window fashion, the fully-connected layers are shared across all spatial locations. ReLUs are applied to the output of the 3×3 conv layer [4].

Translation-Invariant Anchors As it has been previously been hinted, there are 9 anchors for each sliding position. In Ren et al. (2016) [4] they represent 3 scales and 3 aspect ratios, we however opted for 9 identical anchors instead. For a conv feature map of a size 16×16, there are 2304 anchors in total. An important property of this approach is that it is *translation invariant* [4], both in terms of the anchors and the functions that compute proposals relative to the anchors.

A Loss Function for Learning Region Proposals For the RPN training, a binary class label (of being an object or not) is assigned to each anchor. A positive label is assigned to two kinds of anchors: (i) the anchor/anchors with the highest *Intersection-over-Union* (IoU) [4] overlap with a ground-truth box, or (ii) an anchor that has an IoU overlap higher than 0.7 with any ground-truth box. Note that a single ground-truth box may assign positive labels to multiple anchors. A negative label is assigned to a non-positive anchor if its IoU ratio is lower than 0.3 for all ground-truth boxes. Anchors that are neither positive nor negative do not contribute to the training objective. With these definitions, an objective function is minimized following the multi-task loss in Fast R-CNN [3].

For this implementation, we adopt the single image loss function of Ren et al. (2016) [4]:

$$L(p_i, t_i) = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, p_i^*) + \lambda \frac{1}{N_{reg}} \sum_i p_i^* L_{reg}(t_i, t_i^*).$$

Here, i is the index of an anchor in a mini-batch and p_i is the predicted probability of anchor i being an object. The ground-truth label p_i^* is 1 if the anchor is positive, and is 0 if the anchor is negative. t_i is a vector representing the 4 parameterized coordinates of the predicted bounding box, and t_i^* is that of the ground-truth box associated with a positive anchor. The classification loss L_{cls} is log loss over two classes (object vs. not object). For the regression loss,

$L_{reg}(t_i, t_i^*) = R(t_i - t_i^*)$ is used, where R is the robust loss function (smooth L_1) defined in Girshick (2015) [3]. The term $p_i^* L_{reg}$ means the regression loss is activated only for positive anchors ($p_i^* = 1$) and is disabled otherwise ($p_i^* = 0$). The outputs of the box-classification and box-regression layers consist of p_i and t_i respectively. The two terms are normalized with N_{cls} and N_{reg}, and a balancing weight λ [4].

Like in Ren et al. (2016) [4], the parameterizations for regression of the 4 coordinates is adopted following Girshick et al. (2014) [1]:

$$t_x = (x - x_a)/w_a, t_y = (y - y_a)/h_a, t_w = log(w/w_a), t_h = log(h/h_a),$$

$$t_x^* = (x^* - x_a)/w_a, t_y^* = (y^* - y_a)/h_a, t_w^* = log(w^*/w_a), t_h^* = log(h^*/h_a),$$

where x, y, w, and h denote the two coordinates of the box center, width and height. Variables x, x_a, and x^* are for the predicted box, anchor box, and ground-truth box respectively (likewise for y, w, h). This can be thought of as bounding-box regression from an anchor box to a nearby ground-truth box [4].

In this method, the features used for regression are of the same spatial size ($n \times n$) on the feature maps. To account for varying sizes, a set of k bounding-box regressors are learned. Each regressor is responsible for one scale and one aspect ratio, and the k regressors do not share weights. As such, it is still possible to predict boxes of various sizes even though the features are of a fixed size/scale [4].

Optimization Finally, the RPN can be trained end-to-end by back-propagation and *stochastic gradient descent* (SGD).

2.3 Fast R-CNN

A Fast R-CNN network takes as input an entire image and a set of object proposals. The network first processes the whole image with several conv and max pooling layers to produce a conv feature map. Then, for each RoI a pooling layer extracts a fixed-length feature vector from the feature map. Each feature vector is fed into a sequence of fc layers that finally branch into two sibling output layers: one that produces softmax probability estimates over (in our case) 10 object classes plus a catch-all "background" class and another layer that outputs four real-valued numbers for each of the 10 object classes. Each set of 4 values encodes refined bounding-box positions for one of the 10 classes [3].

The RoI pooling layer uses max pooling to convert the features inside any valid region of interest into a small feature map with a fixed spatial extent of $H \times W$, where H and W are layer hyper-parameters that are independent of any particular RoI. Like in Girshick (2015) [3], an RoI here is a rectangular window into a conv feature map. Each RoI is defined by a four-tuple (x, y, h, w) that specifies its center (x, y) and its height and width (h, w) [3].

RoI max pooling works by dividing the $h \times w$ RoI window into an $H \times W$ grid of sub-windows of approximate size $h/H \times w/W$ and then max-pooling the

values in each sub-window into the corresponding output grid cell. Pooling is applied independently to each feature map channel [3].

As for the training, the Fast R-CNN does it end-to-end by back-propagation and SGD.

In addition to hierarchical sampling, Fast R-CNN uses a streamlined training process with one fine-tuning stage that jointly optimizes a softmax classifier and bounding-box regressors. The components of this procedure (the loss, mini-batch sampling strategy, back-propagation through RoI pooling layers, and SGD hyper-parameters) are described below.

Multi-task loss A Fast R-CNN network has two sibling output layers. The first outputs a discrete probability distribution (per RoI), $p = (p_0, ..., p_K)$, over $K + 1$ categories. p is computed by a softmax over the $K + 1$ outputs of a fully connected layer. The second sibling layer outputs bounding-box regression offsets, $t^k = (t^k_x, t^k_y, t^k_w, t)$, for each of the K object classes, indexed by k. We use the parameterization for t_k given in Girshick et al. (2014) [1], in which t_k specifies a scale-invariant translation and log-space height/width shift relative to an object proposal.

2.4 Training Steps

Explain the IQHE in your own words. What does the density of states look like in a 2-DEG when $B = 0$? What are Landau levels and how do they arise? What are edge states? What does the electron transport look like when you change the magnetic field? What do you expect to measure?

3 Experiments

3.1 Experimental Set-up

Explain a step-by-step recipe for fabrication here. How long did you etch and why? What is an Ohmic contact?

3.2 Results

Explain the experimental set-up here. Use a schematic picture (make it yourself in photoshop, paint, ...) to show how the components are connected. Briefly explain how a lock-in amplifier works.

4 Discussion

On the *PASCAL VOC 2012* benchmark [5] with the very deep convolutional neural network *VGG-16* [6], the Faster R-CNN of Ren et al. (2016) achieved 198 ms/img and 70.4% *mean Average Precision* (mAP) [4]. It represents a net improvement over the earlier methods, with the R-CNN of Girshick et al. (2014) scoring 62.4% mAP [3], and the Fast R-CNN of Girshick (2015) reaching 1830 ms/img and 68.4% mAP [4].

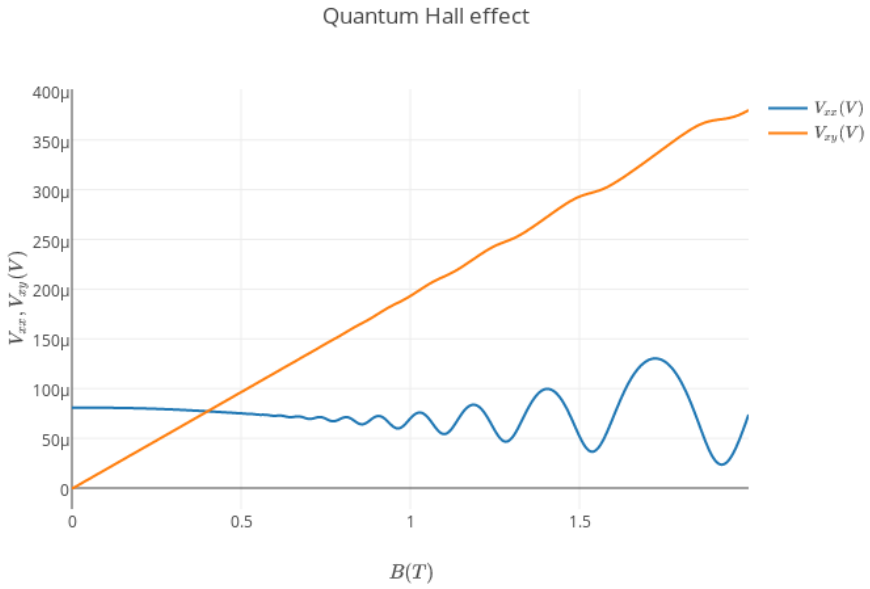

Figure 1: Raw (unprocessed) data. Replace this figure with the one you've made, that shows the resistivity.

4.1 Classical regime

Calculate the sheet electron density n_s and electron mobility μ from the data in the low-field regime, and refer to the theory in section. Explain how you retrieved the values from the data (did you use a linear fit?). Round values off to 1 or 2 significant digits: 8.1643 = 8.2. Also, 5e-6 is easier to read than 0.000005.

!OBS: This part is optional (only if you have time left). Calculate the uncertainty as follows:
$u(f(x,y,z)) = \sqrt{(\frac{\delta f}{\delta x}u(x))^2 + (\frac{\delta f}{\delta y}u(y))^2 + (\frac{\delta f}{\delta z}u(z))^2}$, where f is the calculated value (n_s or μ), x, y, z are the variables taken from the measurement and $u(x)$ is the uncertainty in x (and so on).

4.2 Quantum regime

Calculate n_s for the high-field regime. Show a graph of the longitudinal conductivity (ρ_{xx}) and Hall conductivity(ρ_{xy}) **in units of the resistance quantum** ($\frac{h}{e^2}$), depicting the integer filling factors for each plateau. Show a graph of the plateau number versus its corresponding value of $1/B$. From this you can determine the slope, which you use to calculate the electron density. Again, calculate the uncertainty for your obtained values.

5 Conclusion

Discuss your results. Compare the two values of n_s that you've found in the previous section. Compare your results with literature and comment on the difference. If you didn't know the value of the resistance quantum, would you be able to deduce it from your measurements? If yes/no, why?

Figure 2: This frog was uploaded to writeLaTeX via the project menu.

Item	Quantity
Widgets	42
Gadgets	13

Table 1: An example table.

6 Some LaTeX tips

6.1 How to Include Figures

First you have to upload the image file (JPEG, PNG or PDF) from your computer to writeLaTeX using the upload link the project menu. Then use the includegraphics command to include it in your document. Use the figure environment and the caption command to add a number and a caption to your figure. See the code for Figure 2 in this section for an example.

6.2 How to Make Tables

Use the table and tabular commands for basic tables — see Table 1, for example.

6.3 How to Write Mathematics

LaTeX is great at typesetting mathematics. Let $X_1, X_2, \ldots, X_n$ be a sequence of independent and identically distributed random variables with $\mathrm{E}[X_i] = \mu$ and $\mathrm{Var}[X_i] = \sigma^2 < \infty$, and let

$$S_n = \frac{X_1 + X_2 + \cdots + X_n}{n} = \frac{1}{n}\sum_{i}^{n} X_i \tag{1}$$

denote their mean. Then as n approaches infinity, the random variables $\sqrt{n}(S_n - \mu)$ converge in distribution to a normal $\mathcal{N}(0, \sigma^2)$.

The equation 1 is very nice.

6.4 How to Make Sections and Subsections

Use section and subsection commands to organize your document. LaTeX handles all the formatting and numbering automatically. Use ref and label commands for cross-references.

6.5 How to Make Lists

You can make lists with automatic numbering ...

1. Like this,
2. and like this.

... or bullet points ...

- Like this,
- and like this.

... or with words and descriptions ...

Word Definition

Concept Explanation

Idea Text

We hope you find writeLaTeX useful, and please let us know if you have any feedback using the help menu above.

References

[1] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014.

[2] K. He, X. Zhang, S. Ren, and J. Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. In *ECCV*, 2014.

[3] R. Girshick. Fast R-CNN. *arXiv:1504.08083*, 2015.

[4] S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In *arXiv:1506.01497v3*, 2016.

[5] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results, 2007.

[6] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

[7] J. R. Uijlings, K. E. van de Sande, T. Gevers, and A.W. Smeulders. Selective search for object recognition. emphIJCV, 2013.

[8] C. L. Zitnick and P. Dollár. Edge boxes: Locating object proposals from edges. In *ECCV*, 2014.