G-CNN: an Iterative Grid Based Object Detector

Magyar Najibi
Univ. Maryland

Mohammad Rastegari
Univ. Maryland

Larry S. Davis
Univ. Maryland

CVPR, 2016
Motivation: Proposals are Expensive

Example: find the father of the internet

Selective Search
2.24 seconds

EdgeBoxes
0.38 seconds
Motivation: Proposals are Expensive

Example: find the father of the internet

Cheaper Alternative: grids
Motivation: Keep accuracy with iterations!

Downside of using grids: loss of accuracy

In G-CNN, high accuracy is achieved with grid proposals by using an iterative bounding box regression scheme.

Inspired by IEF - move the work into the regression space!
Some members of the *postdeeluvian* object detection family tree

*Serge Bolongieism*
R-CNN authors investigated iterative procedure:

“At test time, we score each proposal and predict its new detection window only once. In principle, we could iterate this procedure (i.e. re-score the newly predicted bounding box and then predict a new bounding box from it, and so on). However, we found that iterating does not improve results”

APPENDIX C - Rich Feature Hierarchies for accurate object detection and semantic segmentation
Discussion
How does it work?
Bounding Box Regression In Object Detection: Recap

Key idea: snakes are not the same shape as donkeys. (i.e. once you have predicted the object category, you should be able to improve your bounding box)

Introduced in the DPM paper (geometric features)

Revisited in the R-CNN paper (CNN features)
Bounding Box Regression In Object Detection: R-CNN style

Training: The goal is to learn a mapping per category from a proposed box $P$ to a ground truth box $G$. Inputs are $N$ training pairs

$$\{(P_i, G_i)\}_{i=1,...,N}, \text{ where } P^i = (P^i_x, P^i_y, P^i_w, P^i_h)$$

Parameterise mapping with linear functions $d_x(P), d_y(P), d_w(P), d_h(P)$ such that:

$$\hat{G}_x = P_w d_x(P) + P_x$$
$$\hat{G}_y = P_h d_y(P) + P_y$$

(scale invariant)

$$\hat{G}_w = P_w \exp(d_w(P)))$$
$$\hat{G}_h = P_h \exp(d_h(P)))$$

(log space)

The functions are learned with ridge regression.
Training Architecture

G-CNN Network Structure

- Convolutional Layers
- ROI Pooling
- FC Layers
- Class-wise Regressor

Bounding box and its target at step $t (B^t, T^t)$

Next Step

Update the target bounding box $(B^{t+1}, T^{t+1})$

Update Position of Bounding Box $(B^{t+1})$

Notes: bounding box colours
Bounding Box Regression: The Nitty Gritty for G-CNN

**Training** - each bounding box with IoU > 0.2 assigned to one of ground truth boxes in the same image, based on its initial grid position.

The function is learned with piece-wise regression, using target boxes at step $1 < s < S_{\text{train}}$

$$\Phi(B^s_i, G^*_i, s) = B^s_i + \frac{G^*_i - B^s_i}{S_{\text{train}} - s + 1}$$
Bounding Box Regression: The Nitty Gritty for G-CNN

Loss function:

\[ L(\{B_i\}) = \sum_{s=1}^{S_{\text{train}}} \sum_{i=1}^{N} \left[ I(B_i^1 \notin B_{BG}) \times L_{\text{reg}}(\delta_{i,l_i}^s - \Delta(B_i^s, \Phi(B_i^s, A(B_i^s), s))) \right] \]

\( L_{\text{reg}} \) is the smooth L1 loss from Fast R-CNN.
Bounding Box Regression: The Nitty Gritty for G-CNN

For efficiency during training, approximate predicted update

\[ B^s_i = B^{s-1}_i + \Delta^{-1}(\delta^{s-1}_{i,l_i}) \]

with the perfect update

\[ B^s_i = \Phi(B^{s-1}_i, G^*_i, s - 1) \]
Optimisation

SGD ftw.

Note: sampling biases early iteration steps
Test-time architecture

Comparison to R-CNN: \( N_{\text{proposal}} \) vs \((S_{\text{test}} \times N_{\text{grid}})\)
Demo
Experiments
Config

2x2

5x5

10x10

Training overlaps: [0.9, 0.8, 0.7]

Test overlaps: [0.7, 0.5, 0]

Regression network is trained for $S = 3$ steps
Experiment 1: VOC 2007

Each network was based on Alexnet, trained on VOC 2007 trainval set and evaluated on the test set.

<table>
<thead>
<tr>
<th>VOC 2007</th>
<th>aero</th>
<th>bike</th>
<th>bird</th>
<th>boat</th>
<th>bottle</th>
<th>bus</th>
<th>car</th>
<th>cat</th>
<th>chair</th>
<th>cow</th>
<th>table</th>
<th>dog</th>
<th>horse</th>
<th>mbike</th>
<th>person</th>
<th>plant</th>
<th>sheep</th>
<th>sofa</th>
<th>train</th>
<th>tv</th>
<th>mAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>FR-CNN [6]</td>
<td>66.4</td>
<td>71.6</td>
<td>53.8</td>
<td>43.3</td>
<td>24.7</td>
<td>69.2</td>
<td>69.7</td>
<td>71.5</td>
<td>31.1</td>
<td>63.4</td>
<td>59.8</td>
<td>62.2</td>
<td>73.1</td>
<td>65.9</td>
<td>57</td>
<td>26</td>
<td>52</td>
<td>56.4</td>
<td>67.8</td>
<td>57.7</td>
<td>57.1</td>
</tr>
<tr>
<td>G-CNN(3) [ours]</td>
<td>63.2</td>
<td>68.9</td>
<td>51.7</td>
<td>41.8</td>
<td>27.2</td>
<td>69.1</td>
<td>67.7</td>
<td>69.2</td>
<td>31.8</td>
<td>60.6</td>
<td>60.8</td>
<td>63.9</td>
<td>75.5</td>
<td>67.3</td>
<td>54.9</td>
<td>26.1</td>
<td>51.2</td>
<td>57.2</td>
<td>69.6</td>
<td>56.8</td>
<td>56.7</td>
</tr>
<tr>
<td>G-CNN(5) [ours]</td>
<td>65</td>
<td>68.5</td>
<td>52</td>
<td>44.9</td>
<td>24.5</td>
<td>69.3</td>
<td>69.6</td>
<td>68.9</td>
<td>34.6</td>
<td>60.3</td>
<td>58.1</td>
<td>64.6</td>
<td>75.1</td>
<td>70.5</td>
<td>55.2</td>
<td>28.5</td>
<td>50.7</td>
<td>56.8</td>
<td>70.2</td>
<td>56.1</td>
<td>57.2</td>
</tr>
</tbody>
</table>

FR-CNN := One step at test time + approx 2000 initial boxes (SS)

G-CNN(3) := Three steps at test time + approx 1500 initial boxes

G-CNN(5) := Five steps at test time + approx 180 initial boxes
Experiment 2: VOC 2007

Each network was based on VGG-16, trained on VOC 2007 trainval set and evaluated on the test set.

**Claim:** G-CNN effectively moves small # of boxes to targets
Experiment 3: VOC 2012

Each network was based on VGG-16 with the following training:

12 : VOC2012 trainval,
07+12 : VOC2007 trainval + VOC2012 trainval
07++12 : VOC2007 trainval/test + VOC2012 trainval

Claim: G-CNN provides best mAP without a proposal stage
Each network was based on Alexnet, trained on VOC 2007 *trainval* set and evaluated on the *test* set, with **five** steps at test time.

IF-FRCNN := Apply FR-CNN iteratively

1Step-Grid := Train G-CNN with all tuples in one step

**Claim**: *Stepwise training matters*
Analysis of Detection Results

Claim: Removing proposal stage did not hurt localisation
Detection Run Time

Benchmarks with two K40 GPUs with VGG16 Net

Fast R-CNN: 0.5 fps

G-CNN: 3 fps
Rough comparison with current state of the art (VOC 2007 test set)

Different training sets give an idea of how well the model scales with additional data.
Table compiled July 2016

<table>
<thead>
<tr>
<th>Model</th>
<th>Training</th>
<th>Speed (juice)</th>
<th>mAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>G-CNN</td>
<td>07</td>
<td>3fps (2xK40)</td>
<td>66.8</td>
</tr>
<tr>
<td>Faster R-CNN</td>
<td>07+12</td>
<td>5fps (K40)</td>
<td>73.2</td>
</tr>
<tr>
<td>SSD-300</td>
<td>07+12</td>
<td>58fps (TITAN X)</td>
<td>72.1</td>
</tr>
<tr>
<td>SSD-500</td>
<td>07+12</td>
<td>23fps (TITAN X)</td>
<td>75.1</td>
</tr>
<tr>
<td>R-FCN</td>
<td>07+12</td>
<td>6fps (K40)</td>
<td>80.5</td>
</tr>
<tr>
<td>Faster R-CNN</td>
<td>07+12+CO</td>
<td>5fps (K40)</td>
<td>85.6</td>
</tr>
<tr>
<td>R-FCN</td>
<td>07+12+CO</td>
<td>6fps (K40)</td>
<td>83.6</td>
</tr>
</tbody>
</table>

NOTE: By the time you are reading this, it is probably out of date…