Fast R-CNN

Abstract

- Fast R-CNN trains VGG16 (9x faster). 213x faster

1. Introduction

- A single-stage training algorithm that jointly learns to classify object proposals and refine their spatial location
- At runtime, the detection process takes 0.3s, achieving mAP 66% (R-CNN - 62%)

1.1 R-CNN and SPPnet

R-CNN's drawbacks

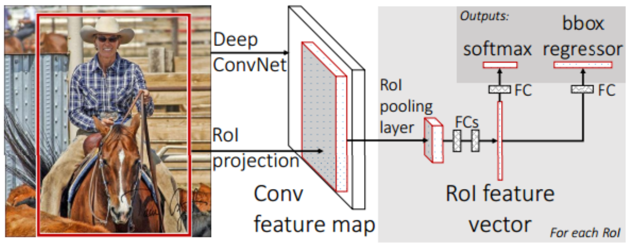
1. Training is a multi-stage pipeline / fine-tuning on object proposals, object detectors, bounding-box regressor
2. Training is expensive in space and time
3. Object detection is slow - 47s/img

R-CNN is slow because it performs a ConvNet forwardpass for each object proposal without sharing computation. SPPnets shares feature map in entire image and they are concatenated as in spatial pyramid pooling. SPPnet accelerates R-CNN by 10-100x at test time. The training time is also reduced by 3x due to faster proposal feature extraction. However, it cannot update the convolutional layers that precede the spatial pyramid pooling and limits the accuracy of deeper network.

1.2 Contributions

1. Higher mAP
2. Training is single-stage, using a multi-task loss
3. Training can update all network layers
4. No disk storage required for feature caching

2. Fast R-CNN architecture and training



1. The network first processes the whole image with several conv and max pooling layers to produce a conv feature map. 2. For each proposal, a region of interest *(ROI)* pooling layer extracts a fixed-length feature vector from teh feature map. 3. Each feature vector is fed into a sequence of FC layers that finally branch into two sibling output layers: one that softmax probabilty estimates over K object classes plus a catch-all “background” class and another layer that outputs bounding-box positions

2.1 The RoI pooling layer

- It uses max-pooling to convert the features inside with a fixed spatial extent of $H \times W$ (empirically set 7×7).
- It divides 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 performs max-pooling the values

2.2 Initializing from pre-trained networks

- Experimented with three pre-trained ImageNet networks.
- The last max pooling layer is replaced by RoI pooling layer.
- FC layer and softmax are replaced with two sibiling layers.
- Input takes both images and RoI

2.3 Fine-tuning for detection

- R-CNN and SPPnet suffers from inefficient back-propagation
- Fast R-CNN takes advantage of feature sharing during training. SGD minibates are sampled hierarchically. During training, it first samples N images and then samples R/N RoIs from each image
- It might cause slow convergence because RoIs from the same image are correlated. This is not a practical issue, selectin $N = 2, R = 128$.

Multi-task loss

$$L(p, u, t^u, v) = L_{cls}(p, u) + \lambda[u \geq 1] L_{loc}(t^u, v)$$

where $L_{cls}(p, u) = -\log p_u$ and $L_{loc}(t^u, v) = \sum_i \text{smooth}_{L_1}(t_i^u - v_i)$ $\lambda = 0$ denotes conventionally background.

Mini-batch sampling

- It takes 25% of RoIs from proposals that have more than 0.5 IoU

Back-propagation through RoI pooling layers

For clarity, in case of $N = 1$, * Forward pass is computed by $y_{rj} = x_{i*(r,j)}$ where $i*(r, j) = \text{argmax}_{i' \in \mathcal{R}(\nabla, |)} x'_i$, $\mathcal{R}(\nabla, |)$ is the index of max pooled sub-window * Backward pass would be,

$$\frac{\partial L}{\partial x_i} = \sum_r \sum_j [i = i*(r, j)] \frac{\partial L}{\partial y_{rj}}$$

2.4 Scale invariance

1. Brute-force
2. multi-scale approach - provides approximately through an image pyramid. randomly sampled a pyramid scale

3. Fast R-CNN detection

- Assumes $\Pr(\text{class} = k \mid r) \triangleq p_k$

3.1. Truncated SVD for faster Detection

- For classification, less time spent on FC layers compared to the conv layers
- For detection, # of RoIs is large and nearly half of the forward pass time is spent on FC It can be accelerated using truncated SVD.

  $$W \approx U \Sigma_t V^t$$

  It reduces # of parameters from uv to $t(u+v)$, which is extremely efficient as $t \ll min(u, v)$

A single layer is splited. The first layer computes $\Sigma_t V^t$ and the second layer computes U.