Report

Jiankai Sun

February 19, 2017

Contents

1 Object Detection

1.1 Faster R-CNN

Faster R-CNN computes with a deep convolutional neural network. It is composed of two modules. The first module is a deep fully convolutional network that proposes regions,and the second module is the Fast R-CNN detector that uses the proposed regions.the Region Proposal Networks (RPNs) module tells the Fast R-CNN module where to look.

There are two techniques: **Translation-Invariant Anchors** and **Multi-Scale Anchors as Regression References**.The translation-invariant property reduces the model size.
The loss function for an image is defined as:

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

For bounding box regression, the parameterizations of the 4 coordinates are adopted as following:

$$\begin{aligned} t_x = (x - x_a)/w_a, \quad t_y = (y - y_a)/h_a, \\ t_w = log(w/w_a), \quad t_h = log(h/h_a), \\ t_x^* = (x^* - x_a)/w_a, \quad t_y^* = (y^* - y_a)/h_a, \\ t_w^* = log(w^*/w_a), \quad t_h^* = log(h^*/h_a), \end{aligned} \tag{2}$$

The RPN is trained end-to-end by back-propagation and stochastic gradient descent (SGD)

Implementation Details For anchors, Thye use 3 scales with box areas of 128^2, 256^2, and 512^2 pixels,and 3 aspect ratios of 1:1, 1:2, and 2:1. They also use the ZF net. To reduce redundancy, they adopt non-maximum suppression (NMS) on the proposal regions based on their *cls* scores. They fix the IoU threshold for NMS at 0.7, which leaves about 2000 proposal regions per image. NMS does not harm the ultimate detection accuracy, but substantially reduces the number of proposals. After NMS, They use the top-N ranked proposal regions for detection. In the following, Thye train Fast R-CNN using 2000 RPN proposals, but evaluate different numbers of proposals at test-time.

1.2 Results of Faster R-CNN

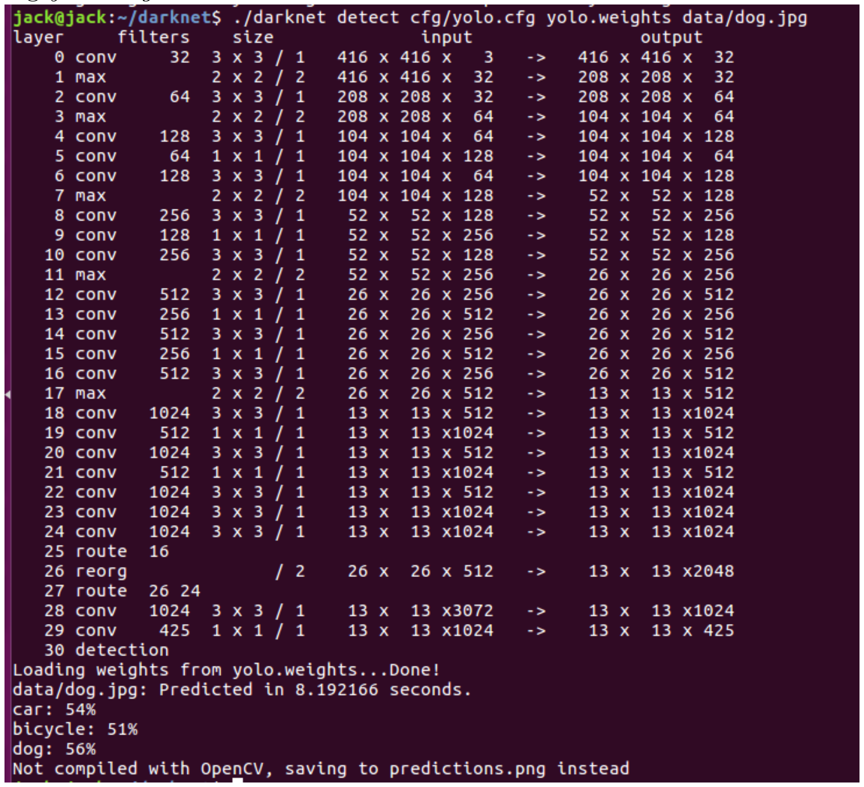
Related Blogs:
Faster-RCNN Analysis `http://blog.csdn.net/luopingfeng/article/details/51245694`
Faster-RCNN Notebook `http://blog.csdn.net/XZZPPP/article/details/51582810`

1.3 Yolo9000

To improve recall and localization while maintaining classification accuracy, Yolo9000 uses **Batch Normalization, High Resolution Classifier, Convolutional with Anchor Boxes, Dimension Clusters, Direct location prediction, Fine-Grained Features, Multi-Scale Training** to improve accuracy. It uses **Darknet-19,** to make it faster. They remove the fully connected layers from YOLO and use anchor boxes to predict bounding boxes. Darknet-19 has 19 convolutional layers and 5 maxpooling layers. To make it stronger, they use WordTree to combine the labels from ImageNet and COCO.

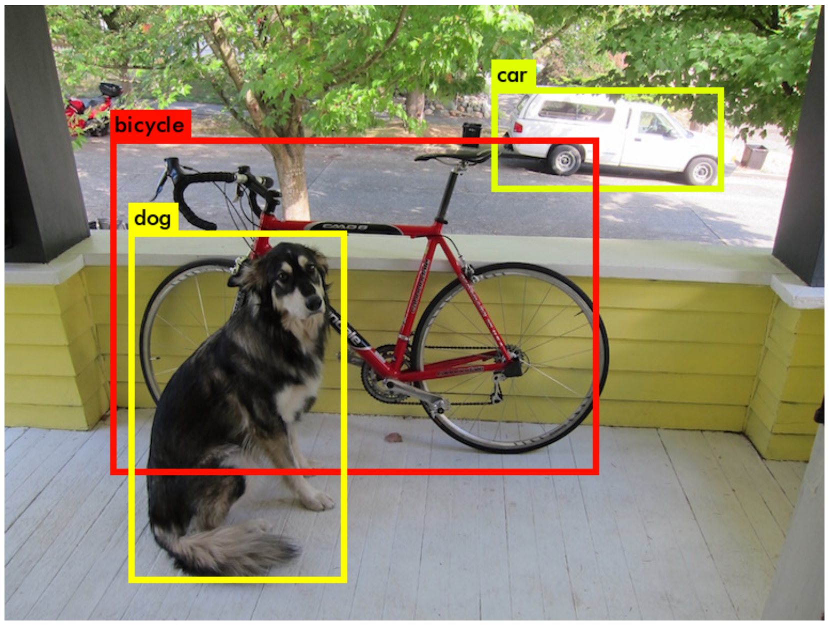
1.4 Results of Yolo9000

Using *yolo.weights*:



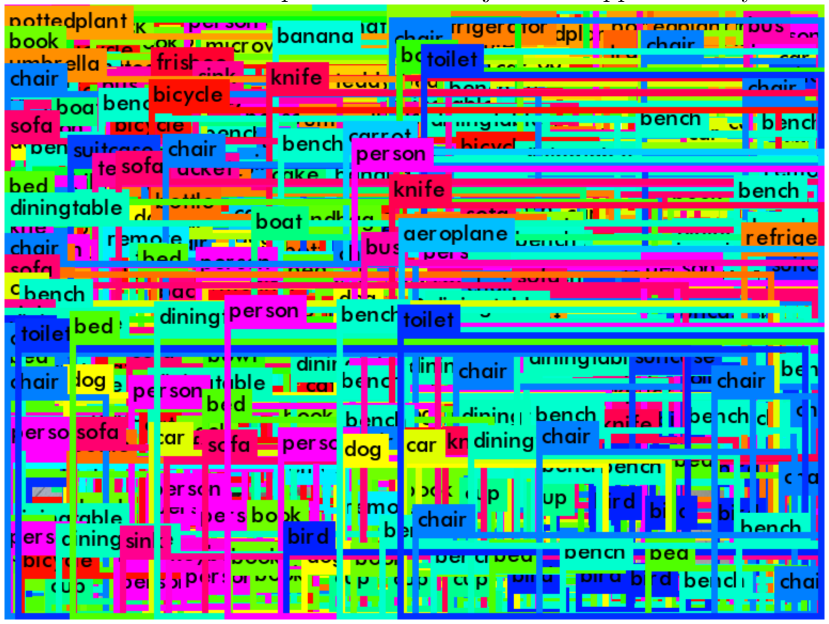
```
jack@jack:~/darknet$ ./darknet detect cfg/yolo.cfg yolo.weights data/dog.jpg
layer     filters    size              input                output
    0 conv     32  3 x 3 / 1   416 x 416 x   3   ->   416 x 416 x  32
    1 max          2 x 2 / 2   416 x 416 x  32   ->   208 x 208 x  32
    2 conv     64  3 x 3 / 1   208 x 208 x  32   ->   208 x 208 x  64
    3 max          2 x 2 / 2   208 x 208 x  64   ->   104 x 104 x  64
    4 conv    128  3 x 3 / 1   104 x 104 x  64   ->   104 x 104 x 128
    5 conv     64  1 x 1 / 1   104 x 104 x 128   ->   104 x 104 x  64
    6 conv    128  3 x 3 / 1   104 x 104 x  64   ->   104 x 104 x 128
    7 max          2 x 2 / 2   104 x 104 x 128   ->    52 x  52 x 128
    8 conv    256  3 x 3 / 1    52 x  52 x 128   ->    52 x  52 x 256
    9 conv    128  1 x 1 / 1    52 x  52 x 256   ->    52 x  52 x 128
   10 conv    256  3 x 3 / 1    52 x  52 x 128   ->    52 x  52 x 256
   11 max          2 x 2 / 2    52 x  52 x 256   ->    26 x  26 x 256
   12 conv    512  3 x 3 / 1    26 x  26 x 256   ->    26 x  26 x 512
   13 conv    256  1 x 1 / 1    26 x  26 x 512   ->    26 x  26 x 256
   14 conv    512  3 x 3 / 1    26 x  26 x 256   ->    26 x  26 x 512
   15 conv    256  1 x 1 / 1    26 x  26 x 512   ->    26 x  26 x 256
   16 conv    512  3 x 3 / 1    26 x  26 x 256   ->    26 x  26 x 512
   17 max          2 x 2 / 2    26 x  26 x 512   ->    13 x  13 x 512
   18 conv   1024  3 x 3 / 1    13 x  13 x 512   ->    13 x  13 x1024
   19 conv    512  1 x 1 / 1    13 x  13 x1024   ->    13 x  13 x 512
   20 conv   1024  3 x 3 / 1    13 x  13 x 512   ->    13 x  13 x1024
   21 conv    512  1 x 1 / 1    13 x  13 x1024   ->    13 x  13 x 512
   22 conv   1024  3 x 3 / 1    13 x  13 x 512   ->    13 x  13 x1024
   23 conv   1024  3 x 3 / 1    13 x  13 x1024   ->    13 x  13 x1024
   24 conv   1024  3 x 3 / 1    13 x  13 x1024   ->    13 x  13 x1024
   25 route  16
   26 reorg              / 2    26 x  26 x 512   ->    13 x  13 x2048
   27 route  26 24
   28 conv   1024  3 x 3 / 1    13 x  13 x3072   ->    13 x  13 x1024
   29 conv    425  1 x 1 / 1    13 x  13 x1024   ->    13 x  13 x 425
   30 detection
Loading weights from yolo.weights...Done!
data/dog.jpg: Predicted in 8.192166 seconds.
car: 54%
bicycle: 51%
dog: 56%
Not compiled with OpenCV, saving to predictions.png instead
```

result1



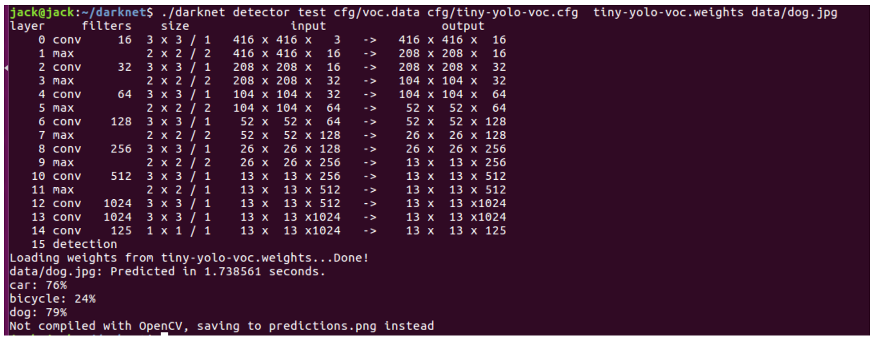
prediction1

Changing The Detection Threshold:
The result tells that all the predicitions of objects are approximately 0%



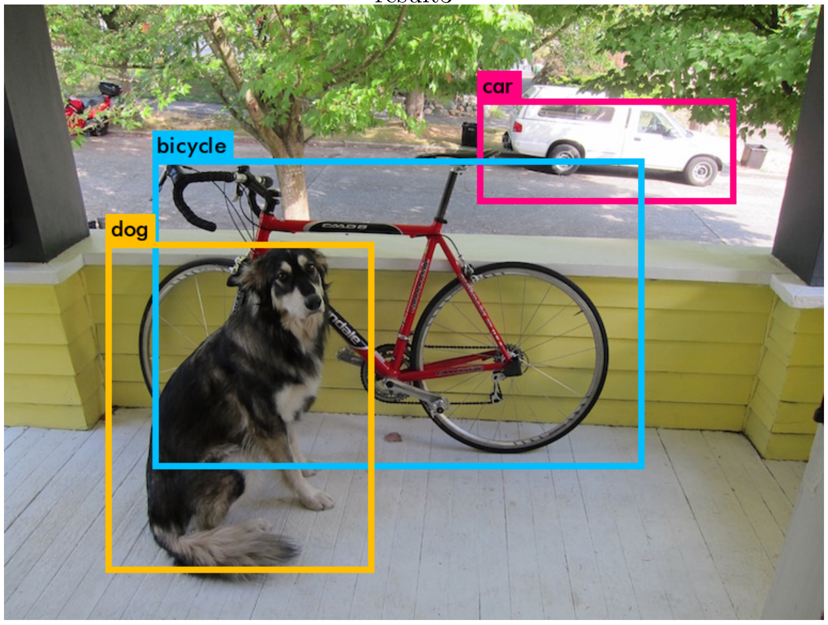
prediction2

Tiny Yolo:



```
jack@jack:~/darknet$ ./darknet detector test cfg/voc.data cfg/tiny-yolo-voc.cfg  tiny-yolo-voc.weights data/dog.jpg
layer     filters    size              input                output
    0 conv     16  3 x 3 / 1   416 x 416 x   3   ->   416 x 416 x  16
    1 max          2 x 2 / 2   416 x 416 x  16   ->   208 x 208 x  16
    2 conv     32  3 x 3 / 1   208 x 208 x  16   ->   208 x 208 x  32
    3 max          2 x 2 / 2   208 x 208 x  32   ->   104 x 104 x  32
    4 conv     64  3 x 3 / 1   104 x 104 x  32   ->   104 x 104 x  64
    5 max          2 x 2 / 2   104 x 104 x  64   ->    52 x  52 x  64
    6 conv    128  3 x 3 / 1    52 x  52 x  64   ->    52 x  52 x 128
    7 max          2 x 2 / 2    52 x  52 x 128   ->    26 x  26 x 128
    8 conv    256  3 x 3 / 1    26 x  26 x 128   ->    26 x  26 x 256
    9 max          2 x 2 / 2    26 x  26 x 256   ->    13 x  13 x 256
   10 conv    512  3 x 3 / 1    13 x  13 x 256   ->    13 x  13 x 512
   11 max          2 x 2 / 1    13 x  13 x 512   ->    13 x  13 x 512
   12 conv   1024  3 x 3 / 1    13 x  13 x 512   ->    13 x  13 x1024
   13 conv   1024  3 x 3 / 1    13 x  13 x1024   ->    13 x  13 x1024
   14 conv    125  1 x 1 / 1    13 x  13 x1024   ->    13 x  13 x 125
   15 detection
Loading weights from tiny-yolo-voc.weights...Done!
data/dog.jpg: Predicted in 1.738561 seconds.
car: 76%
bicycle: 24%
dog: 79%
Not compiled with OpenCV, saving to predictions.png instead
```

result3



prediction3

2 Pose Estimation

2.1 RMPE

They present a straight-forward two-step framework, where basic human detector and single person pose estimator are introduced. Subsequently, then proposed regional multi-person pose estimation (RMPE) is presented. The training and testing on single person images are carried out by SPPE.
In the straight-forward framework, human proposal is firstly detected by human detector(SSD is employed) into **Symmetric STN + SPPE** and single person pose estimator is subsequently applied to those human proposals individually. They draw multiple SPPEs for better understanding.

There are three main technical contributions:

Symmetric STN Human proposal is extended with 20% along both height and width direction. A spatial transformer network(STN) is used help SPPE to extract high quality single person images. More specifically, the STN performs a 2D affine transformation and the point-wise transformation can be mathematically expressed as

$$\begin{pmatrix} x_i^s \\ y_i^s \end{pmatrix} = [\theta_1 \quad \theta_2 \quad \theta_3] \begin{pmatrix} x_i^t \\ y_i^t \\ 1 \end{pmatrix} \tag{3}$$

A spatial detransformer network (SDTN) is required to be employed, which simply computes the γ for detransform and generates grids based on γ.

$$\begin{pmatrix} x_i^t \\ y_i^t \end{pmatrix} = [\gamma_1 \quad \gamma_2 \quad \gamma_3] \begin{pmatrix} x_i^s \\ y_i^s \\ 1 \end{pmatrix} \tag{4}$$

Since SDTN is the inverse procedure of STN, we can obtain

$$[\gamma_1 \quad \gamma_2][\theta_1 \quad \theta_2] = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{5}$$

$$[\gamma_1 \quad \gamma_2]\theta_3 + \gamma_3 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{6}$$

They jointly optimize STN and SDTN by back propagation technique. A spatial remapping highway is constructed to connect STN and SDTN.

$$[\gamma_1 \quad \gamma_2] = [\theta_1 \quad \theta_2]^{-1} \tag{7}$$

$$\gamma_3 = -1 \times [\gamma_1 \quad \gamma_2]\theta_3 \tag{8}$$

For back propagation of SDTN $\frac{\partial J(W,b)}{\partial \theta}$ can be derived.
In experiment, SqueezeNet is adopted as localization network, since it is small yet powerful enough to extract desired features. They add a parallel SPPE in training phrase. For the parallel SPPE, They specify its labels to be pose on single person image. It shares the same STN with Symmetric STN. They minimize the sum error of parallel SPPE and symmetric STN together.

Deep Proposals Generator They randomly sample a huge number of region proposals. The network(dizygotic-Siamese network) will select part of them as augmented samples according to their output scores (possibilities).

Parametric Pose NMS Pose non-maximum suppression(NMS) is required to eliminate redundant detections. Firstly, most confident pose is selected as reference, and some poses close to it are eliminated by an appropriate elimination criterion. This process is re-carried out in the remaining poses set until all poses are eliminated.
Elimination criterion can be written as

$$f(P_i, P_j|\Lambda, \eta) = 1[d(P_i, P_j|\Lambda, \lambda) \leq \eta] \tag{9}$$

The soft matching function is as

$$K_{Sim}(P_i, P_j|\sigma_1) = \begin{cases} \sum_n \tanh \frac{c_i^n}{\sigma_1} \cdot \tanh \frac{c_j^n}{\sigma_1}, & if\ k_j^n\ is\ within\ ß(k_i^n) \\ 0, & otherwise \end{cases} \tag{10}$$

The spatial distance among parts which can be written as

$$H_{Sim}(P_i, P_j|\sigma_2) = \sum_n exp[-\frac{(k_i^n - k_j^n)^2}{\sigma_2}] \tag{11}$$

The final distance function is:

$$d(P_i, P_j|\Lambda) = K_{Sim}(P_i, P_j|\sigma_1) + \lambda H_{Sim}(P_i, P_j|\sigma_2) \tag{12}$$

2.2 Results of RMPE

2.3 Cao et al

Confidence Maps for Part Detection They take the maximum of the confidence maps instead of average

$$S_(j,k)^*(p) = exp(-\frac{||p - x_{j,k}||_2^2}{\sigma}) \tag{13}$$

$$S_(j)^*(p) = \max_k S_(j,k)^*(p) \tag{14}$$

Part Affinity Fields for Part Association One way to obtain the measurement is to predict confidence maps for n interpolated midpoints along the line segment connecting two body parts.

$$E = \int_{u=0}^{u=1} S_c(p(u))du \tag{15}$$

$$p(u) = (1-u)d_{j1} + ud_{j2} \tag{16}$$

The ideal part affinity vector field, $L_{c,k}^*$ at an image point p is

$$L_{c,k}^*(p) = \begin{cases} v, & if\ p\ on\ limb\ c,k \\ 0, & otherwise \end{cases} \tag{17}$$

$$L_c^*(p) = \frac{1}{n_p} \sum_k L_{c,k}^*(p) \tag{18}$$

$$E = \int_{u=0}^{u=1} L_c(p(u)) \cdot \frac{d_{j2} - d_{j1}}{||d_{j2} - dj1||_2} du \tag{19}$$

MultiPerson Parsing using PAFs To find a matching with maximum weight for the chosen edges.

$$\max_{Z_c} E_c = \max_{Z_c} \sum_{m \in D_{j1}} \sum_{n \in D_{j2}} E_{mn} \cdot z_{mnj_1j_2} \tag{20}$$

$$\forall m \in D_{j1}, \sum_{n \in D_{j2}} z_{mnj_1j_2} \leq 1, \tag{21}$$

$$\forall n \in D_{j2}, \sum_{m \in D_{j1}} z_{mnj_1j_2} \leq 1, \tag{22}$$

The optimization is decomposed simply as

$$\max_{Z} E = \sum_{c=1}^{C} \max_{Z_c} E_c \tag{23}$$

we design an architecture that learns to predict both jointly, because they share common visual clues to infer and are tightly coupled spatially. PAFs for parts association are suitable to be jointly learned with the parts confidence maps for parts detection in one end-to-end trainable CNN.
Their **loss functions** at both branches at stage t are:

$$f_1^t = \sum_{j=1}^{J} \sum_{p \in I} W(p) \cdot ||S_j^t(p) - S_j^*(p)||_2^2, \tag{24}$$

$$f_2^t = \sum_{c=1}^{C} \sum_{p \in I} W(p) \cdot ||L_c^t(p) - L_c^*(p)||_2^2, \tag{25}$$

The intermediate supervision at each stage addresses the vanishing gradient problem by replenishing the gradient periodically. The overall objective is

$$f = \sum_{t=1}^{T} (f_1^t + \lambda f_2^t), \tag{26}$$

2.4 Results of Cao et al

3 Multi-object tracking

3.1 Learn to track

Multi-Object Tracking(MOT) has focused on the tracking-by-detection principal, where the main challenge is the data association problem in linking object detections. Online methods solve the data association problem either probabilistically or determinatively. In their framework, they consider a single object tracker to be an agent in Markov decision processes(MDP), whose task is to track the target. Then they learn a good policy for the MDP with reinforcement learning, and employ multiple MDPs to track multiple targets.

Online Multi-Object Tracking Framework They give a introduction of Markov decision processes by defining the *States, Actions and Transition Function, Reward Function.* After that, they describe policies designed for the Active subspace and the Tracked subspace, then they present a novel reinforcement learning algorithm to learn a good policy for data association in the Lost subspace.
About Policy in an Active State, They train a binary Support Vector Machine (SVM) offline to classify a detection into tracked or inactive using a normalized 5D feature vector $\phi_{Active}(s)$. The reward function in Active:

$$R_{Active}(s,a) = y(a)(w_{Active}^T \phi_{Active}(s) + b_{Active}) \tag{27}$$

About Policy in a Tracked State, They describe our implementation based on the TLD tracker. This part includes *Template Representation, Template Tracking and Template Updating.*
As to Policy in a Lost State, this part includes *Data Association, Reinforcement Learning and Feature Representation*

3.2 Results

4 Object Segmentation

4.1 Instance-aware

Multi-task Network Cascades(MNC) model has three stages: proposing box-level instances, regressing mask-level instances, and categorizing each instance. The network structure and loss function of Regressing Box-level Instances follow RPNs. RPN loss function

$$L_1 = L_1(B(\Theta)) \tag{28}$$

As to Regressing Mask-level Instances, They extract a feature of the box predicted by stage 1 by Region-of-Interest (RoI) pooling. The loss term L_2 of stage

2 is

$$L_2 = L_2(M(\Theta)|B(\Theta) \tag{29}$$

The third stage(Categorizing Instances) takes the shared convolutional features, stage-1 boxes, and stage-2 masks as input. It outputs category scores for each instance. The loss term L_3 is

$$L_3 = L_3(C(\Theta)|B(\Theta), M(\Theta)) \tag{30}$$

The loss function of the entire cascade is

$$L(\Theta) = L_1(B(\Theta)) + L_2(M(\Theta)|B(\Theta) + L_3(C(\Theta)|B(\Theta), M(\Theta)) \tag{31}$$

Then they develop a *Differentiable RoI Warping Layers* to account for the gradient w.r.t. predicted box positions and address the dependency on $B(\Theta)$. The dependency on $M(\Theta)$ is also tackled accordingly.

Implementation Details For stage 2, They use *Non-maximum suppression* to reduce redundant candidates.
*Positive/negative samples.*On stage 2, If the overlapping ratio (IoU) is greater than 0.5, this proposed box is considered as positive and contributes to the mask regression loss; otherwise is ignored in the regression loss. On stage 3, They consider two sets of positive/negative samples. In the first set, the positive samples are the instances that overlap with ground truth boxes by box-level IoU ≥ 0.5. In the second set, the positive samples are the instances that overlap with ground truth instances by box-level IoU ≥ 0.5 and mask-level IoU ≥ 0.5.
Hyper-parameters for training. They adopt an image-centric training framework. They use 5-stage inference for both 3-stage and 5-stage trained structures.

4.2 Results

5 Scene Parsing

5.1 Pyramid-Parsing

They summarize several common issues for complex-scene parsing. They are *Mismatched Relationship, Confusion Categories, Inconspicuous Classes.*

Pyramid Pooling Module fuses features under four different pyramid scales. The output of different levels in the pyramid pooling module contains the feature map with varied sizes. To maintain the weight of global feature, They use 1×1 convolution layer after each pyramid level to reduce the dimension of context representation to $1 = N$ of the original one if the level size of pyramid is N. Then they directly upsample the low-dimension feature maps to get the same size feature as the original feature map via bilinear interpolation. Finally, different levels of features are concatenated as the final pyramid pooling global feature.

Network Architecture Pyramid scene parsing network (PSPNet) uses a pre-trained ResNet model with the dilated network strategy to extract the feature map.

Implementation Details The implementation is based on the public platform Caffe. They use the poly learning rate policy where current learning rate equals to the base one multiplying $(1 - \frac{iter}{max_{iter}})^power$. Base learning rate is 0.01 and power is 0.9. The performance can be improved by increasing the iteration number, which is set to 150K for ImageNet experiment, 30K for PASCAL VOC and 90K for Cityscapes. Momentum and weight decay are set to 0.9 and 0.0001 respectively. For data augmentation, they adopt random mirror and random resize between 0.5 and 2 for all datasets, and additionally add random rotation between -10 and 10 degrees, and random Gaussian blur for ImageNet and PASCAL VOC. This comprehensive data augmentation scheme makes the network resist overfitting.

5.2 Results