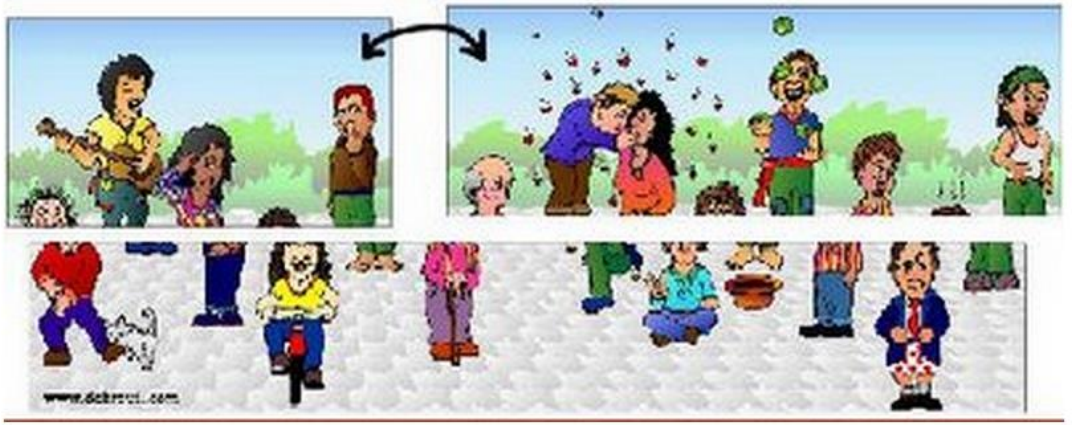
Recognize Objects

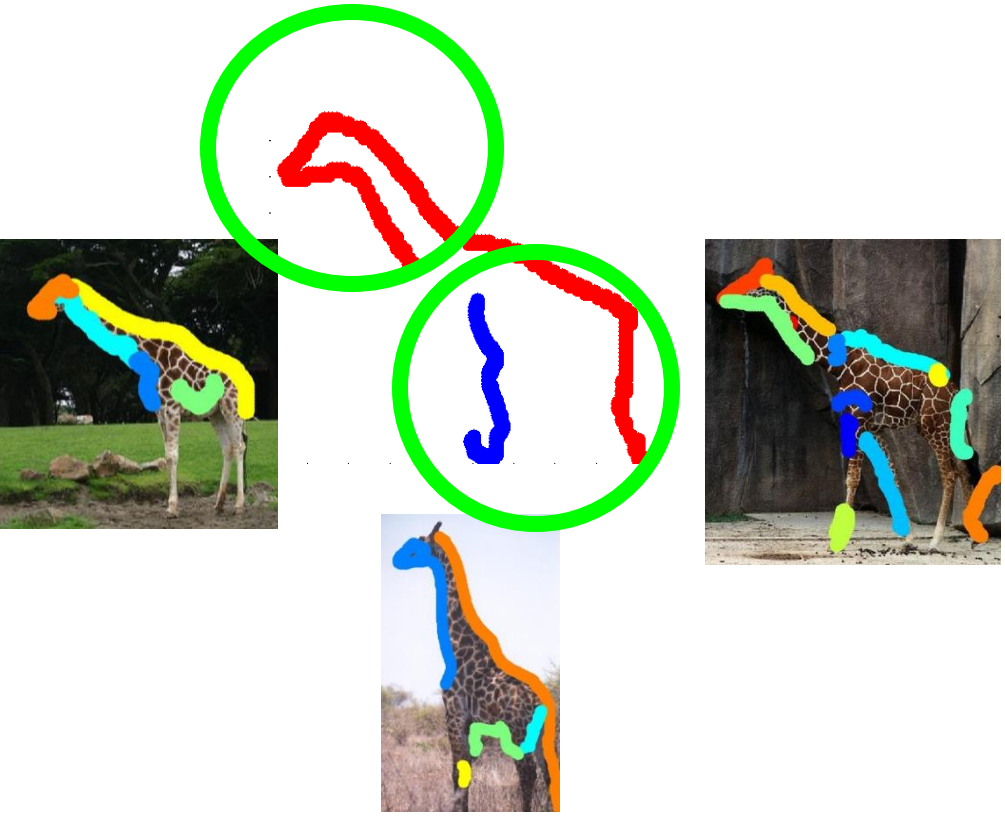
Gluing Pixels



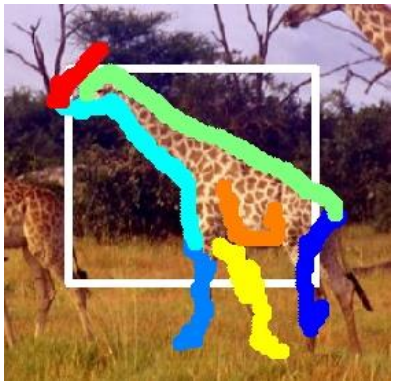
CIS 680, Spring ‘17

recognize objects: Describe and Discriminate

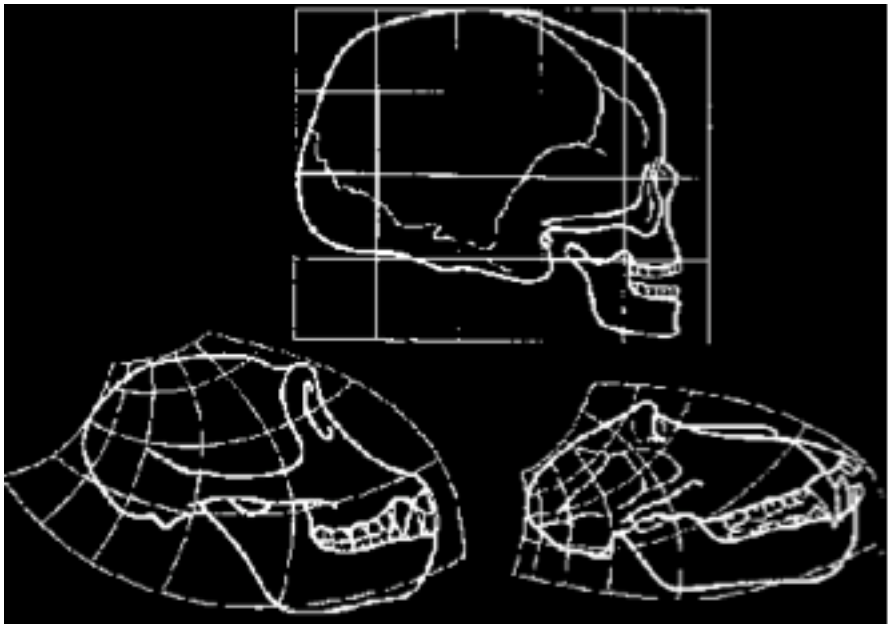
- **Describe** positives by commonalities of shape

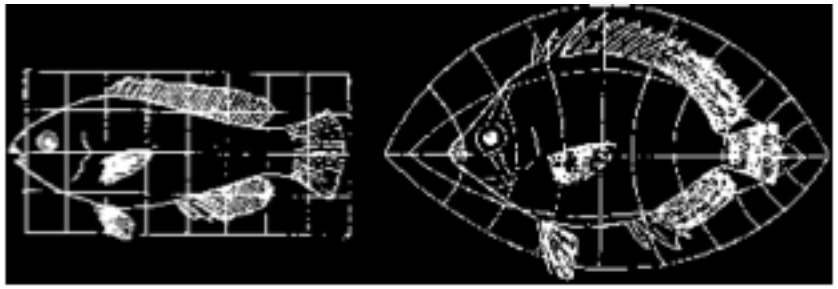


- **Discriminate** negatives by uniqueness of shape

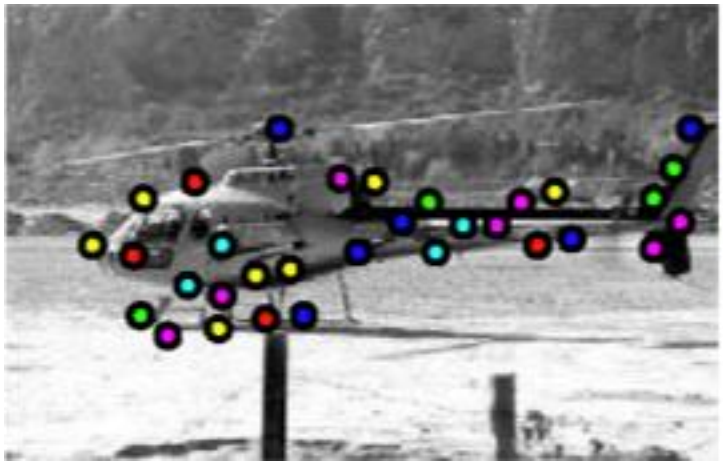


recognize objects: Frame ‘it’ right



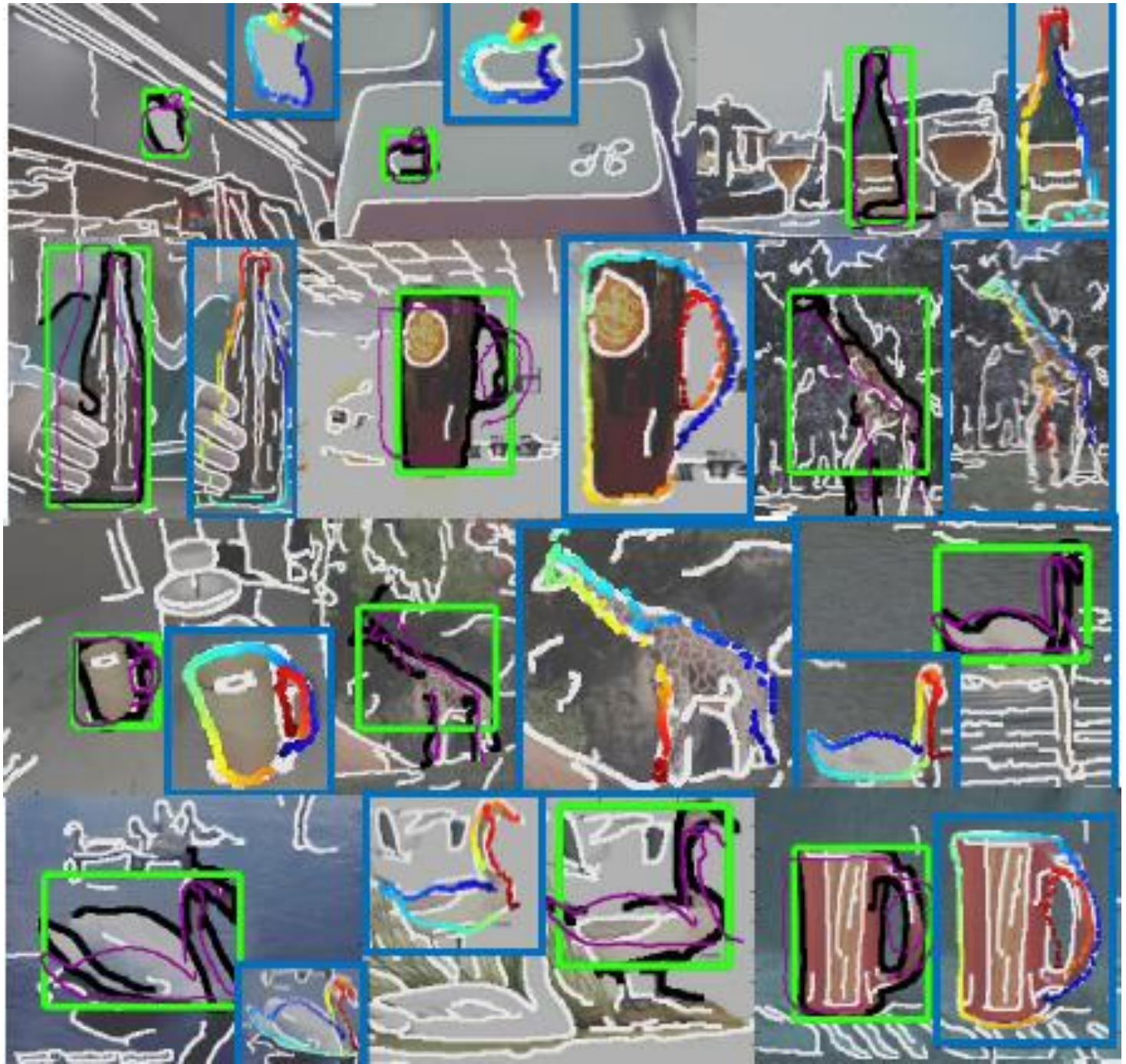


Challenge 1: maintain discriminative under deformation



Similarity of appearance near feature points

Similarity in configuration of the feature points

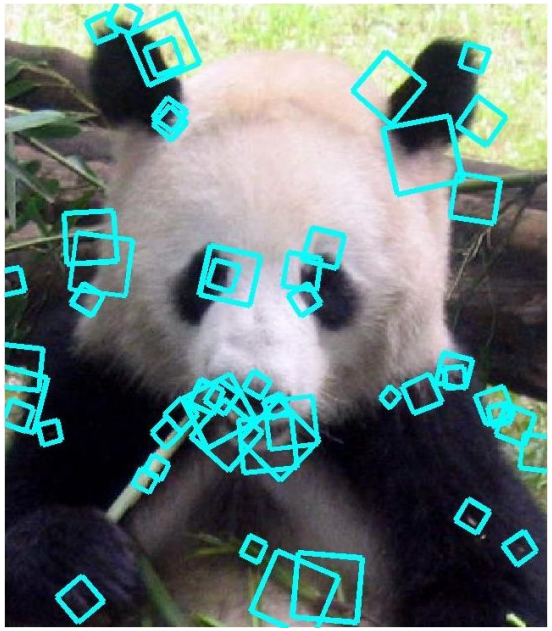


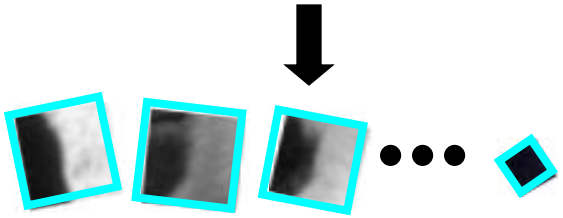
maintain discriminative under deformation

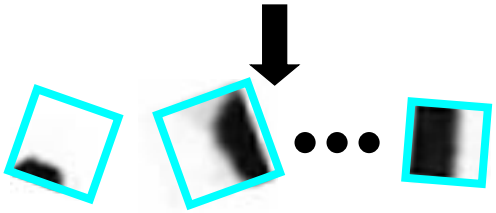
The Pyramid Match Kernel: Discriminative Classification with Sets of Image Features

Kristen Grauman
Trevor Darrell

Sets of features



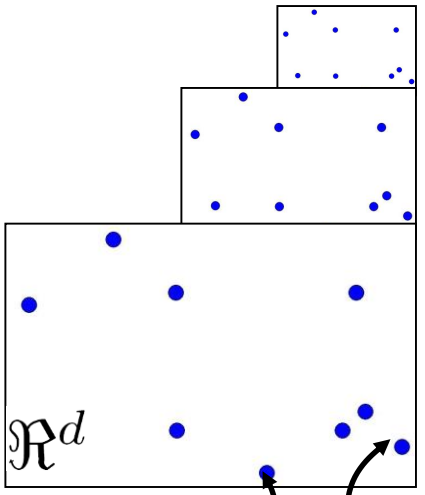




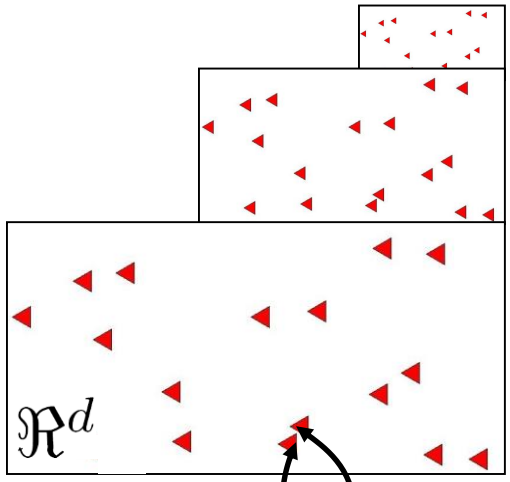
$$\mathbf{X} = \{\vec{\mathbf{x}}_1, \ldots, \vec{\mathbf{x}}_m\}$$

$$\mathbf{Y} = \{\vec{\mathbf{y}}_1, \ldots, \vec{\mathbf{y}}_n\}$$

Pyramid match

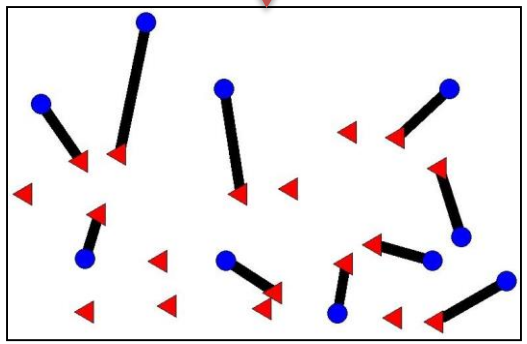


$$\mathbf{X} = \{\vec{\mathbf{x}}_1, \ldots, \vec{\mathbf{x}}_m\}$$
$$\vec{\mathbf{x}}_i \in \Re^d$$



$$\mathbf{Y} = \{\vec{\mathbf{y}}_1, \ldots, \vec{\mathbf{y}}_n\}$$
$$\vec{\mathbf{y}}_i \in \Re^d$$

Related:
Earth Moving Distance (EMD)



optimal partial matching

$$\max_{\pi:\mathbf{X}\to\mathbf{Y}} \sum_{\mathbf{x}_i \in \mathbf{X}} \mathcal{S}(\mathbf{x}_i, \pi(\mathbf{x}_i))$$

Pyramid match overview

Pyramid match kernel measures similarity of a partial matching between two sets:

- Place multi-dimensional, multi-resolution grid over point sets
- Consider points matched at finest resolution where they fall into same grid cell
- Approximate similarity between matched points with worst case similarity at given level

No explicit search for matches!

Pyramid match kernel

Number of newly matched pairs at level *i*

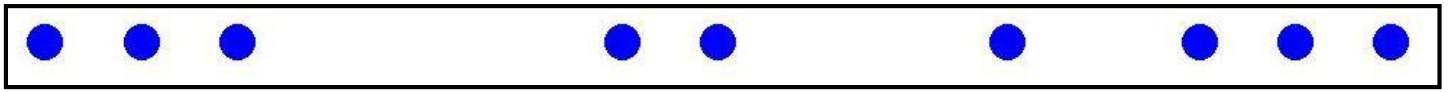
Approximate partial match similarity

$$K_\Delta = \sum_{i=0}^{L} w_i N_i$$

Measure of difficulty of a match at level *i*

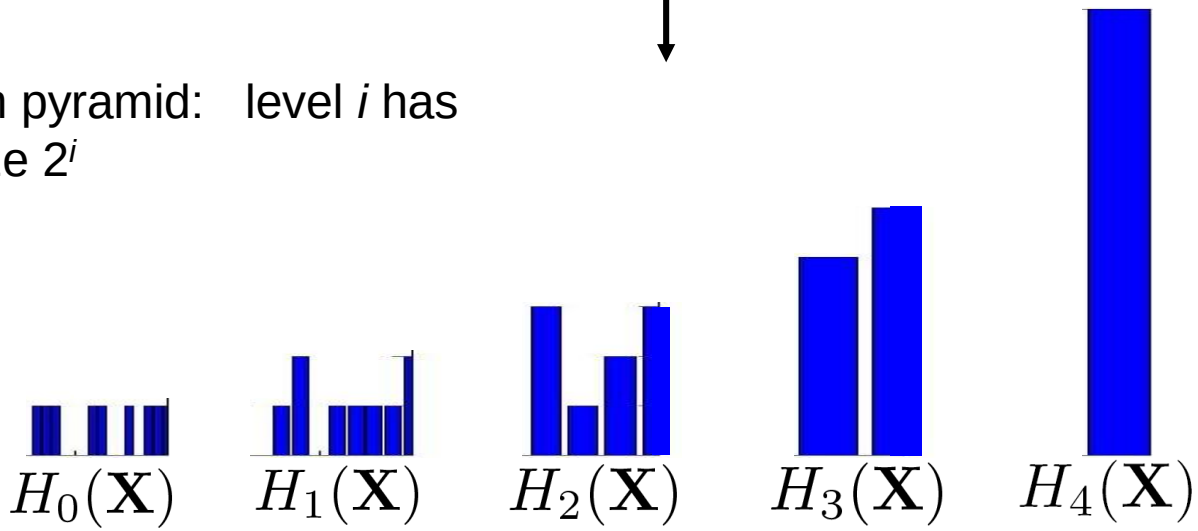
Feature extraction

$$\mathbf{X} = \{\vec{\mathbf{x}}_1, \ldots, \vec{\mathbf{x}}_m\}, \quad \vec{\mathbf{x}}_i \in \Re^d$$



$d = 1$

Histogram pyramid: level *i* has bins of size 2^i

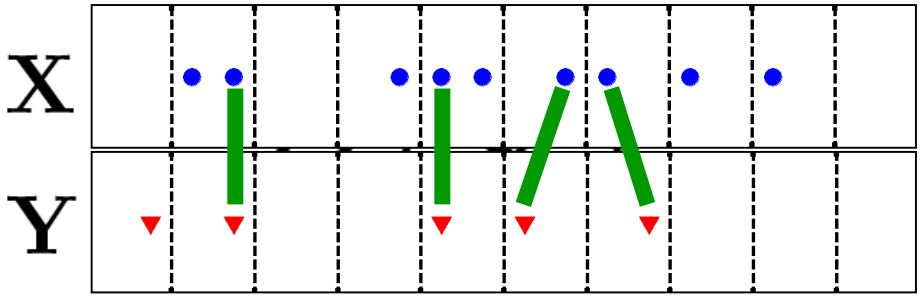


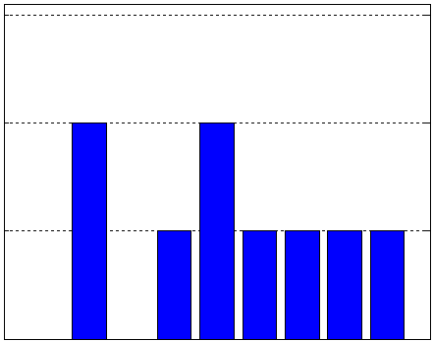
$$\Psi(\mathbf{X}) = [H_0(\mathbf{X}), \ldots, H_L(\mathbf{X})]$$

Counting matches

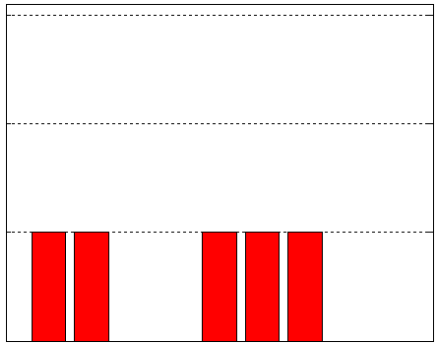
Histogram intersection

$$\mathcal{I}\left(H(\mathbf{X}), H(\mathbf{Y})\right) = \sum_{j=1}^{r} \min\left(H(\mathbf{X})_j, H(\mathbf{Y})_j\right)$$

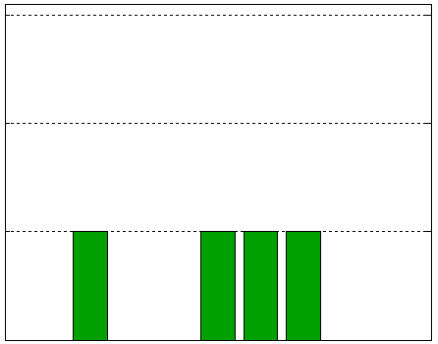




$H(\mathbf{X})$



$H(\mathbf{Y})$



$\mathcal{I}\left(H(\mathbf{X}), H(\mathbf{Y})\right) = 4$

Counting new matches

Histogram intersection

$$\mathcal{I}\left(H(\mathbf{X}), H(\mathbf{Y})\right) = \sum_{j=1}^{r} \min\left(H(\mathbf{X})_j, H(\mathbf{Y})_j\right)$$

matches at this level — matches at previous level

$$N_i = \mathcal{I}\left(H_i(\mathbf{X}), H_i(\mathbf{Y})\right) - \mathcal{I}\left(H_{i-1}(\mathbf{X}), H_{i-1}(\mathbf{Y})\right)$$

Difference in histogram intersections across levels counts *number of new pairs* matched

Pyramid match kernel

histogram pyramids

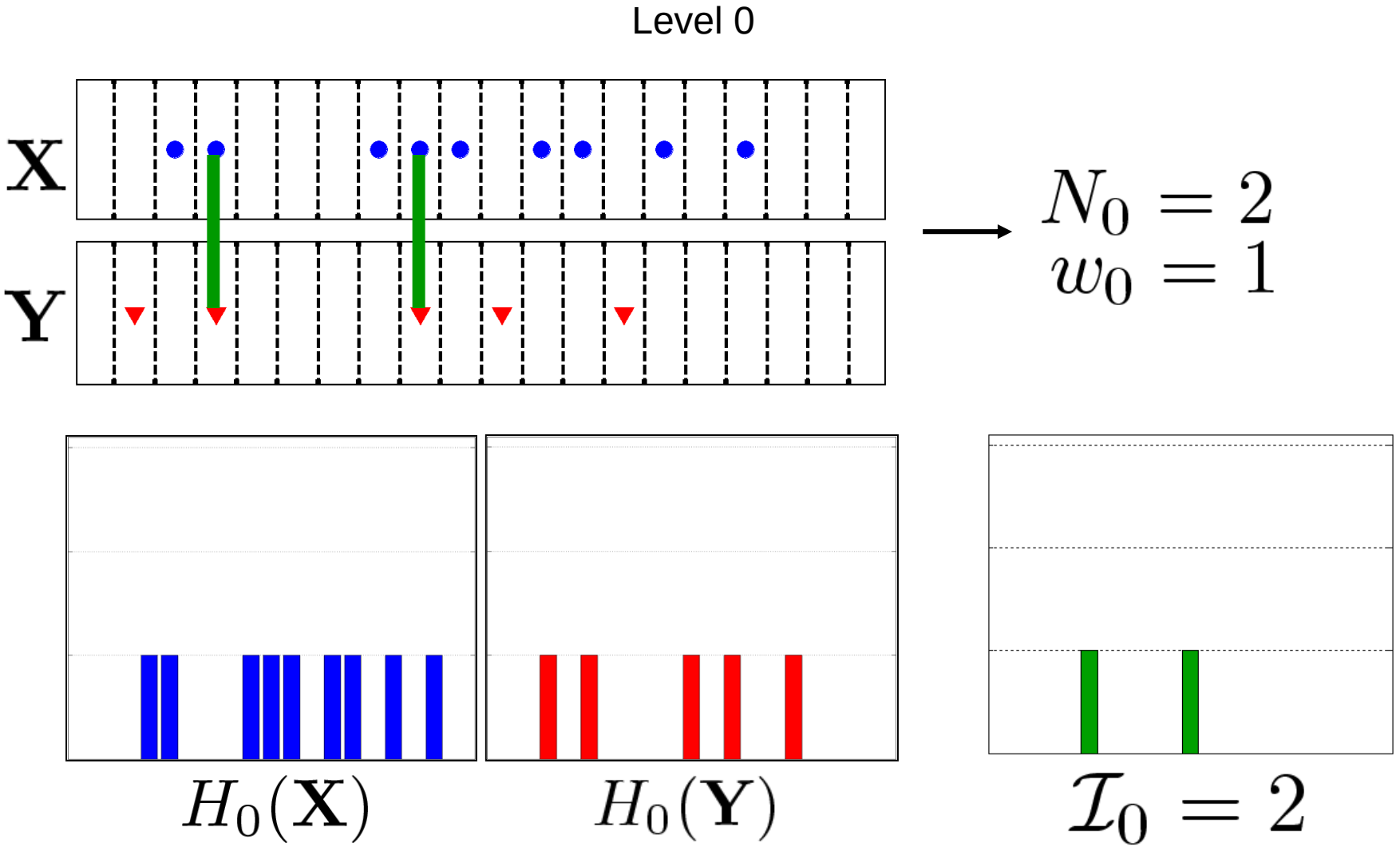
$$K_\Delta\left(\overbrace{\Psi(\mathbf{X}), \Psi(\mathbf{Y})}\right) =$$

$$\sum_{i=0}^{L} \frac{1}{2^i} \Big(\underbrace{\mathcal{I}\left(H_i(\mathbf{X}), H_i(\mathbf{Y})\right) - \mathcal{I}\left(H_{i-1}(\mathbf{X}), H_{i-1}(\mathbf{Y})\right)}_{\text{number of newly matched pairs at level } i} \Big)$$

measure of difficulty of a match at level *i*

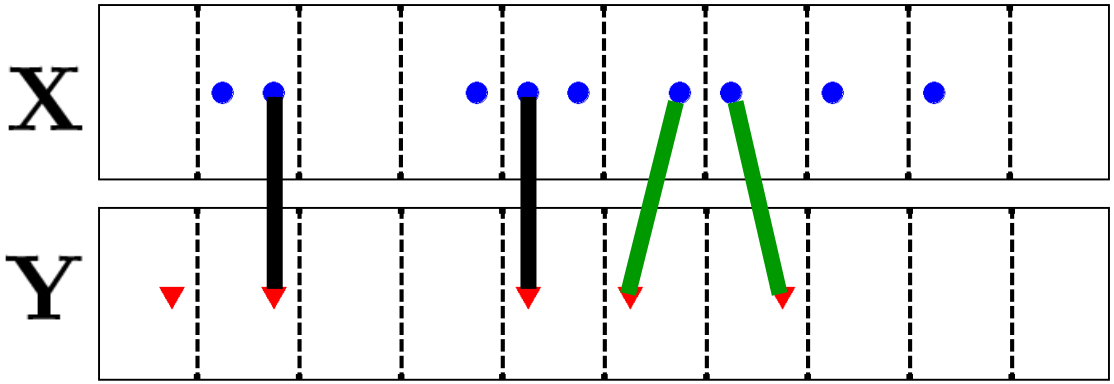
- Weights inversely proportional to bin size
- Normalize kernel values to avoid favoring large sets

Example pyramid match

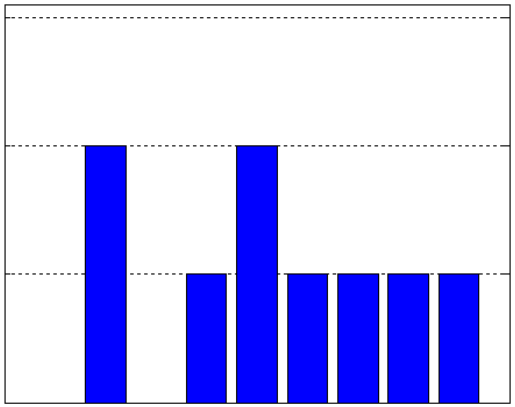


Example pyramid match

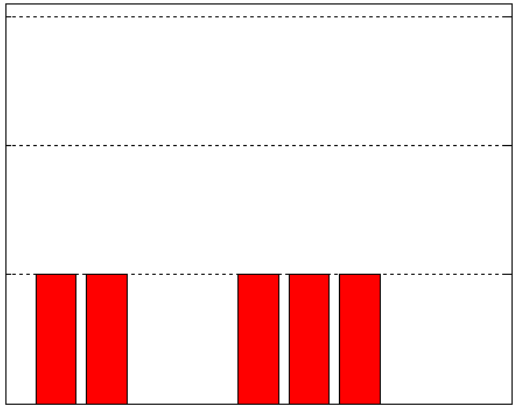
Level 1



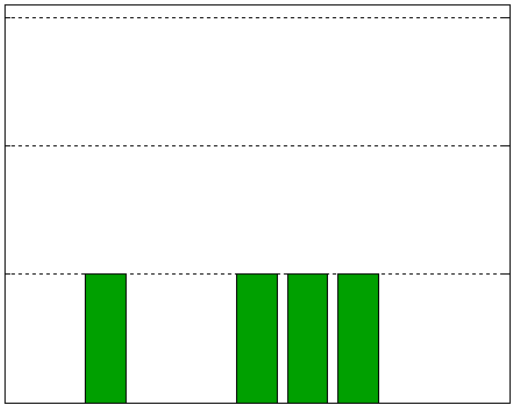
$$N_1 = 4 - 2 = 2$$
$$w_1 = \frac{1}{2}$$



$H_1(\mathbf{X})$



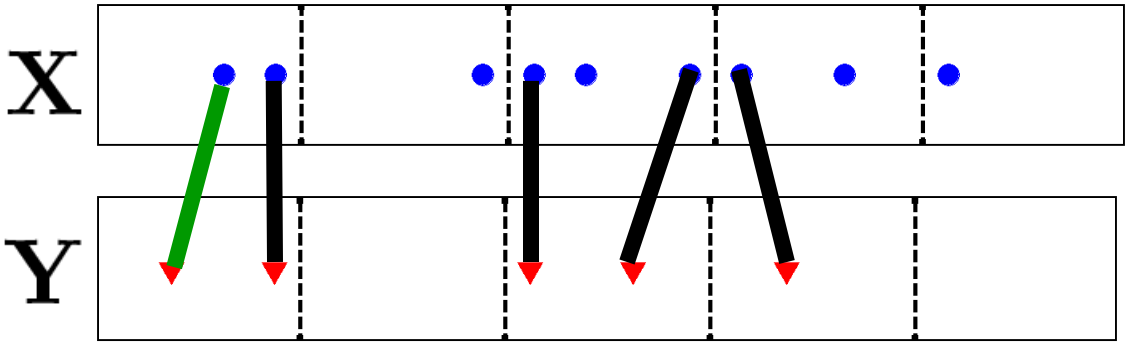
$H_1(\mathbf{Y})$



$\mathcal{I}_1 = 4$

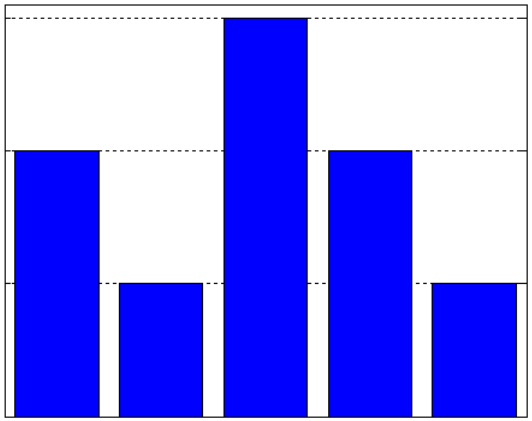
Example pyramid match

Level 2

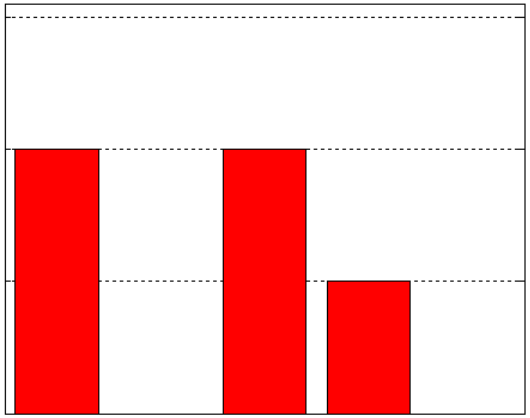


$$N_2 = 5 - 4 = 1$$

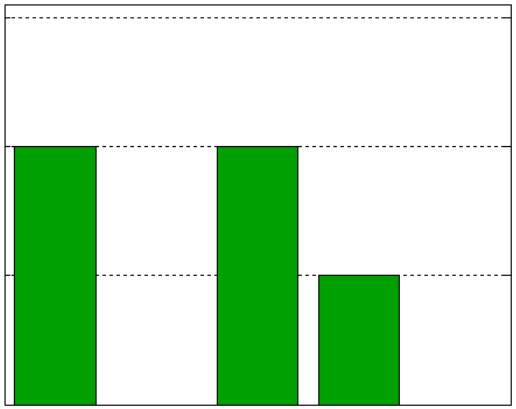
$$w_2 = \frac{1}{4}$$



$H_2(\mathbf{X})$



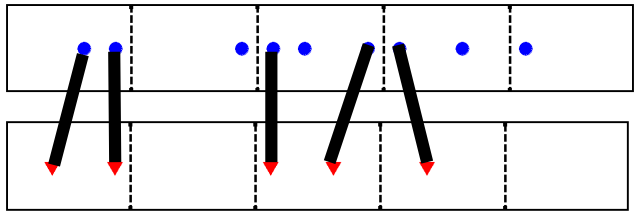
$H_2(\mathbf{Y})$



$\mathcal{I}_2 = 5$

Example pyramid match

pyramid match

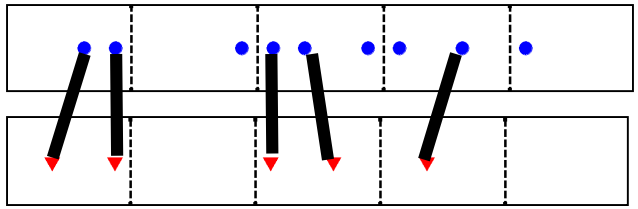


$$K_\Delta = \sum_{i=0}^{L} w_i N_i$$

$$= 1(2) + \tfrac{1}{2}(2) + \tfrac{1}{4}(1) = 3.25$$

---

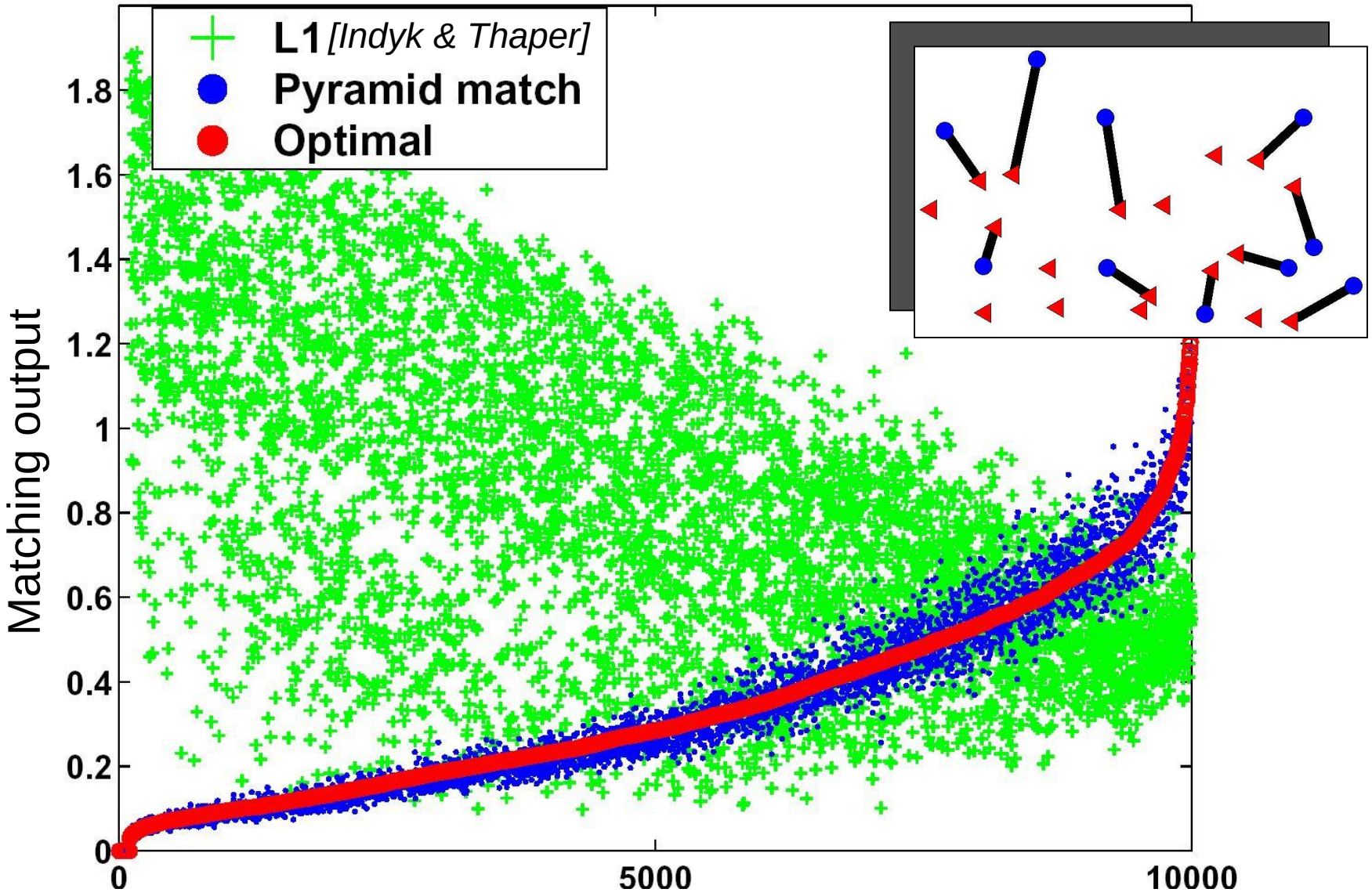
optimal match



$$K = \max_{\pi:\mathbf{X}\to\mathbf{Y}} \sum_{\mathbf{x}_i \in \mathbf{X}} \mathcal{S}(\mathbf{x}_i, \pi(\mathbf{x}_i))$$

$$= 1(2) + \tfrac{1}{2}(3) = 3.5$$

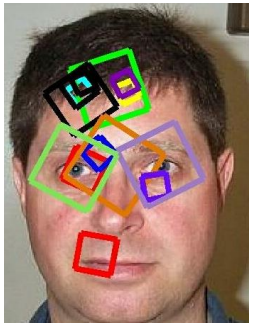
Approximation of the optimal partial matching

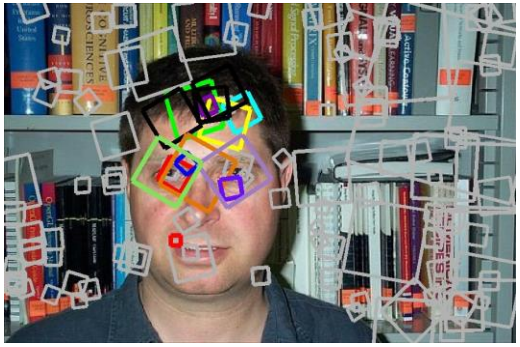


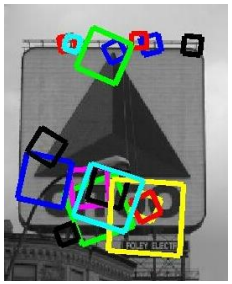
100 sets with 2D points, cardinalities vary between 5 and 100

Localization

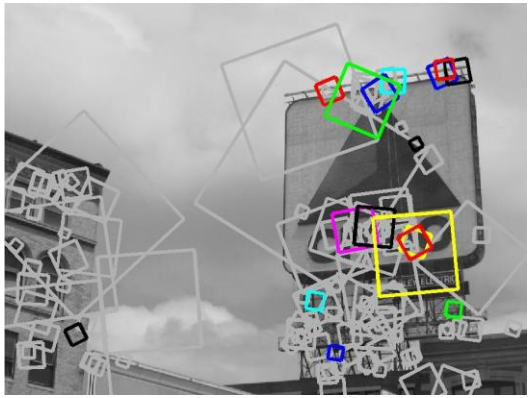
- Inspect intersections to obtain correspondences between features
- Higher confidence correspondences at finer resolution levels





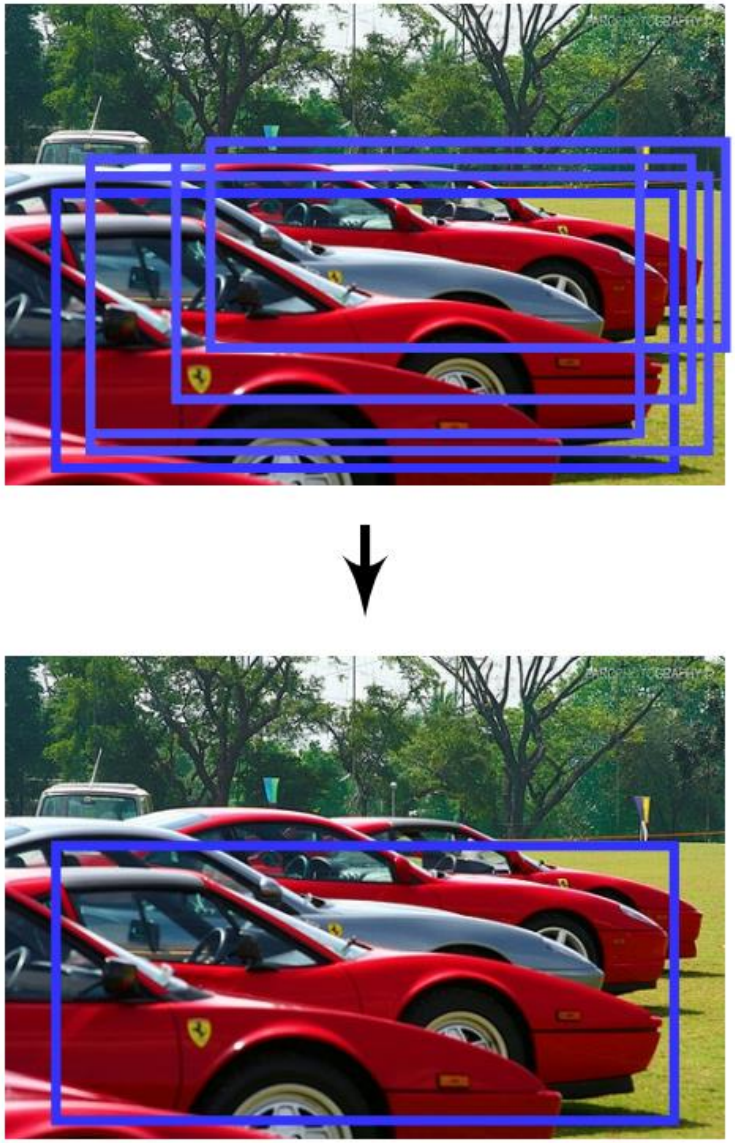


target



observation

Challenge: maintain discriminative under occlusion



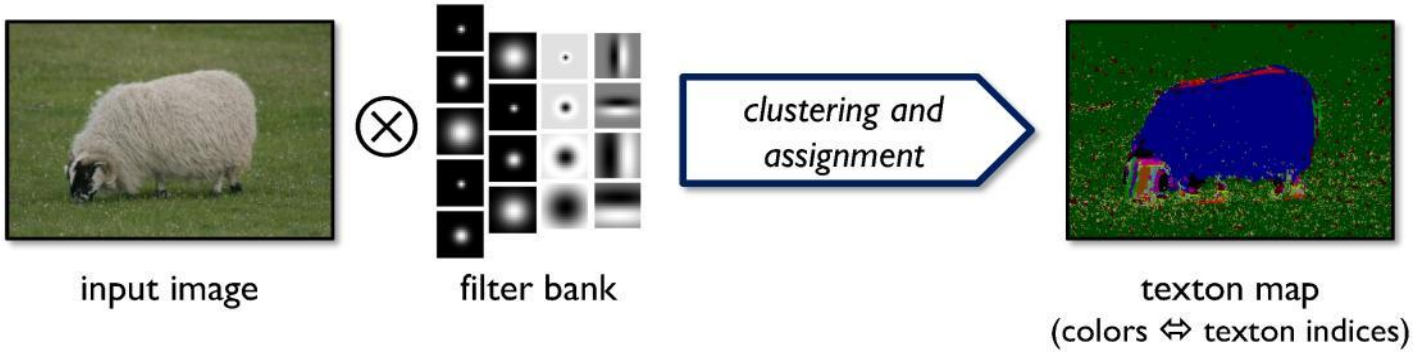
Challenge 2: maintain discriminative under occlusion

Per Pixel Classification

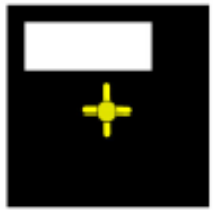
Texton Boost for pixel level semantic classification

Jamie Shotton

- Step 1: Texton Map generation (17 filters, K=400)

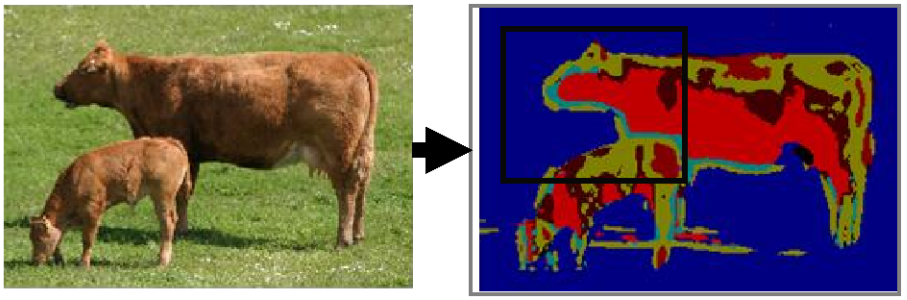


- Step 2: Shape Filter
  - For each texton *t*
    - ✓ Inputs
      - Texton Map
      - (Rectangle mask *r*, texton query *t*)
      - Pixel location *i*
    - ✓ Output
      - Area in rectangle mask that match *t*
  - End result is a texton histogram of area responses

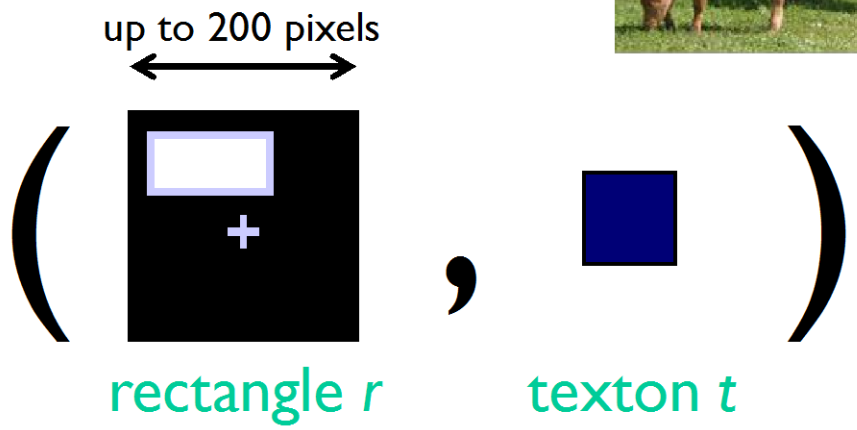


- **How does this capture shape?**

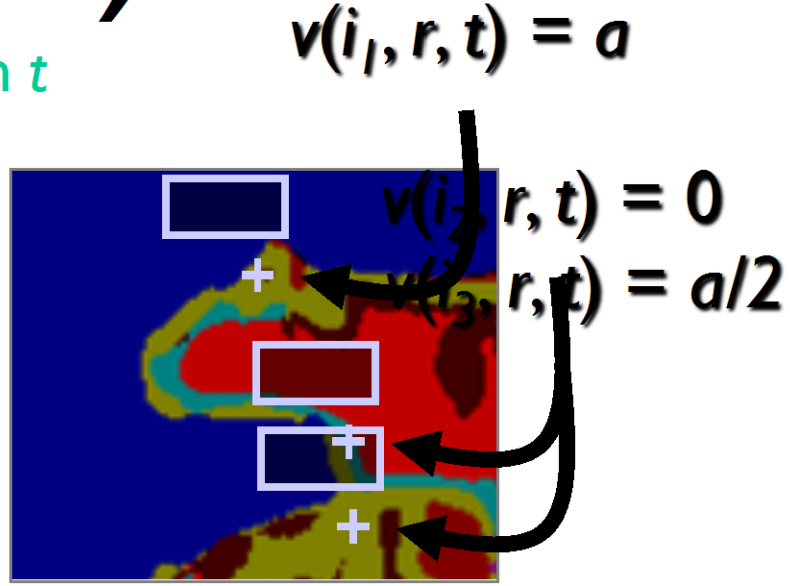
Slides from Shotton's ECCV talk



Shape Filters

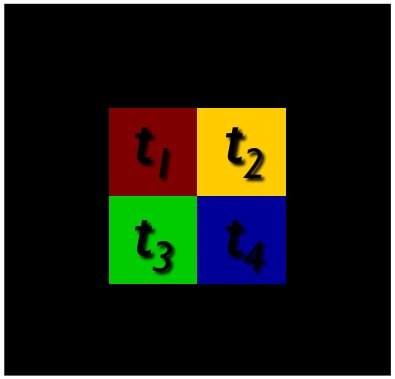


- Pair:
- Feature responses $v(i, r, t)$
- Large bounding boxes enable *long range interactions*
- Integral images

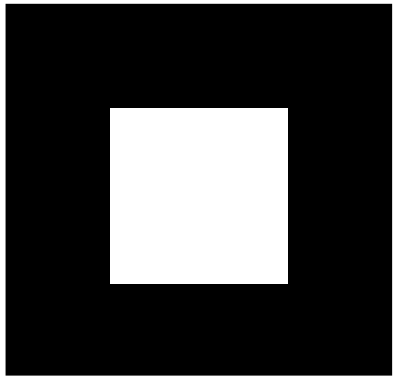


Slides from Shotton's ECCV talk

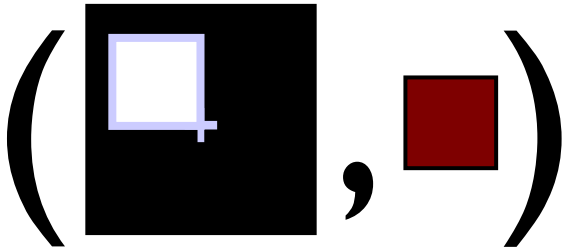
Shape as Texton Layout

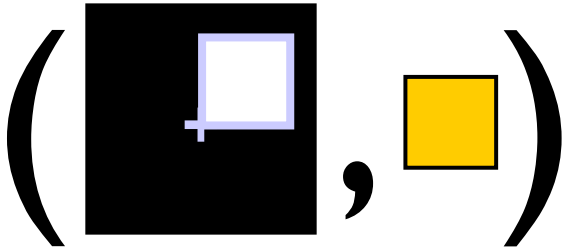


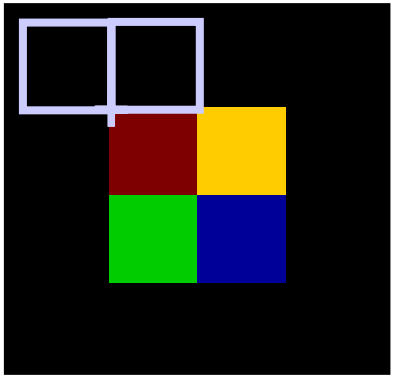
texton map



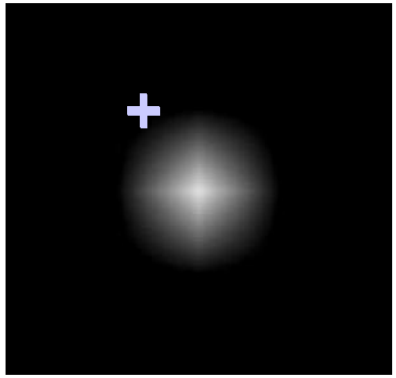
ground truth

$(r_1, t_1) =$ 

$(r_2, t_2) =$ 



texton map

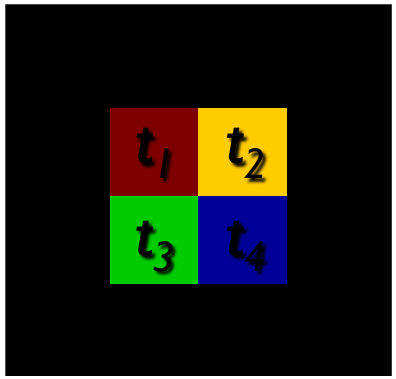


feature response image

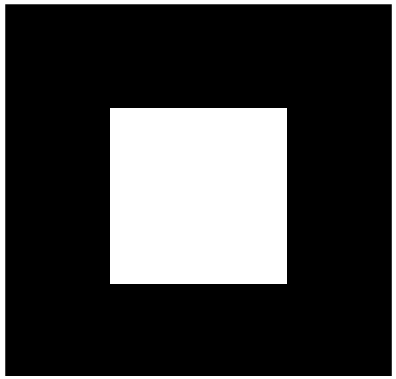
$v(i, r_2, t_2)$

Slides from Shotton's ECCV talk

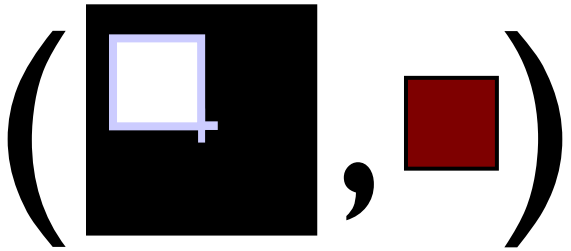
Shape as Texton Layout

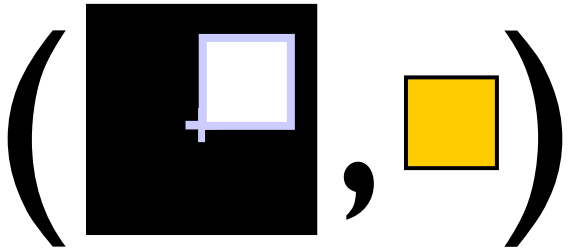


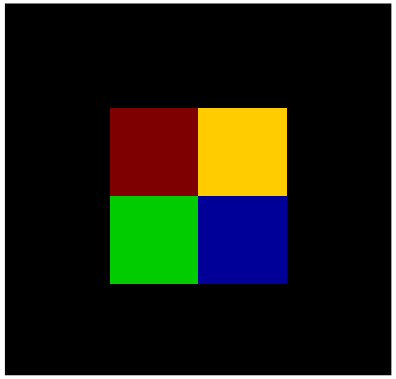
texton map



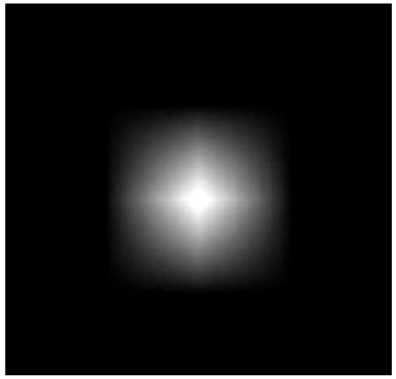
ground truth

$(r_1, t_1) =$ 

$(r_2, t_2) =$ 

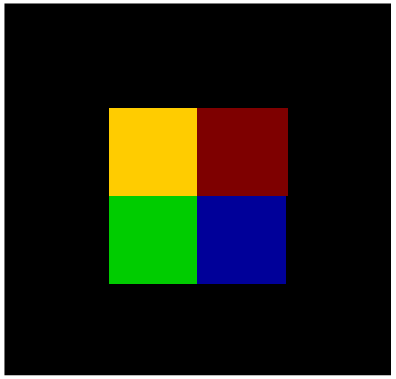


texton map

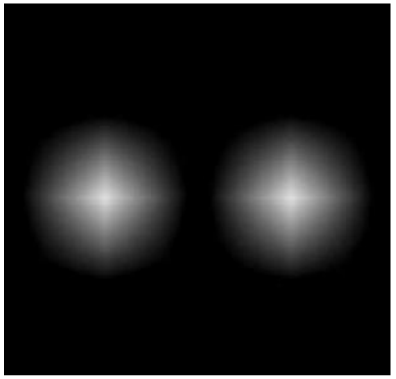


summed response images

$v(i, r_1, t_1) + v(i, r_2, t_2)$

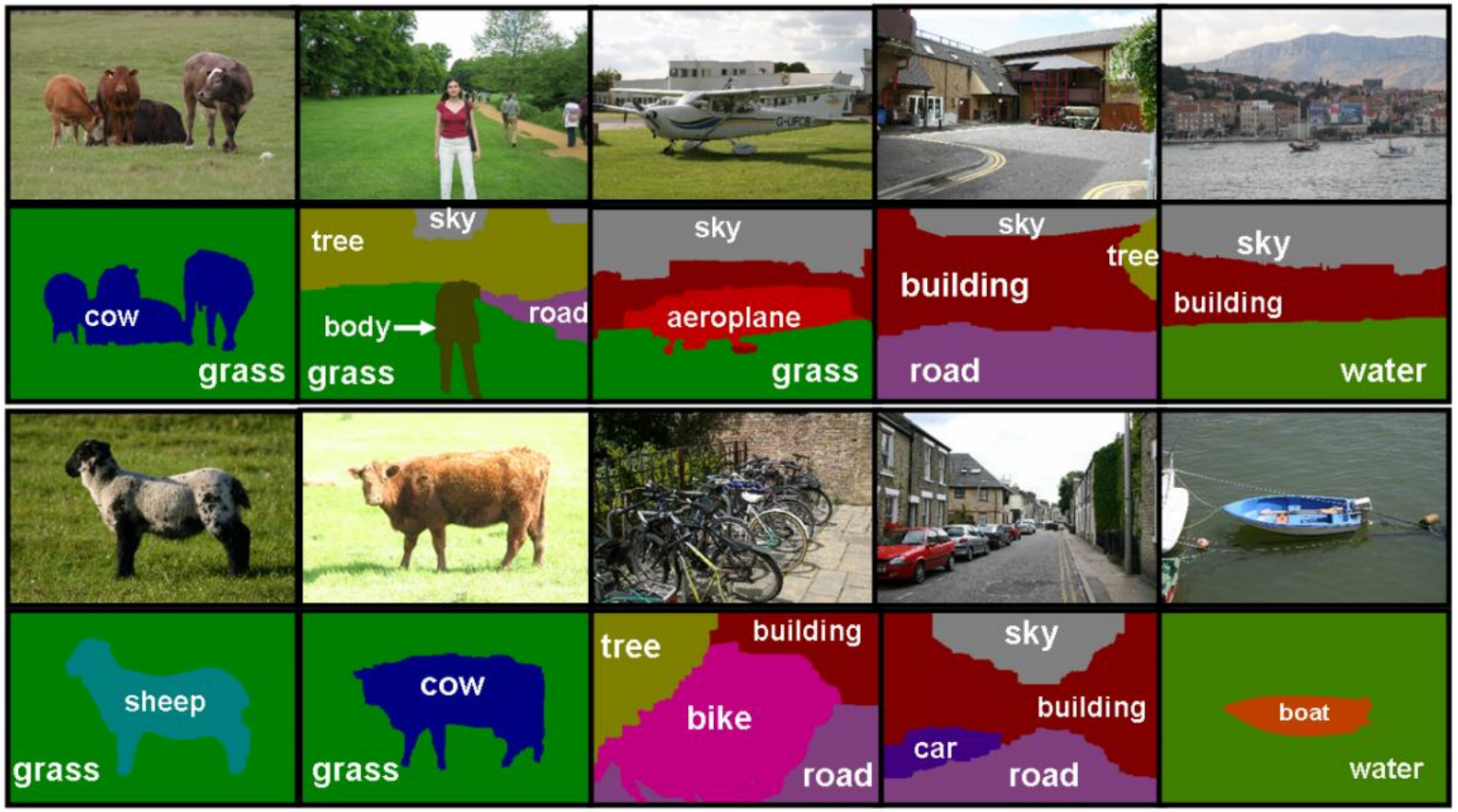


texton map



summed response images

$v(i, r_1, t_1) + v(i, r_2, t_2)$



Object classes	Building	Grass	Tree	Cow	Sheep	Sky	Aeroplane	Water	Face	Car
Bike	Flower	Sign	Bird	Book	Chair	Road	Cat	Dog	Body	Boat

- Failures


sky
cow
grass
sky
tree
water
building
sign
road
dog
road
build
road
bike

Benchmark Challenges

VOC: 20 classes

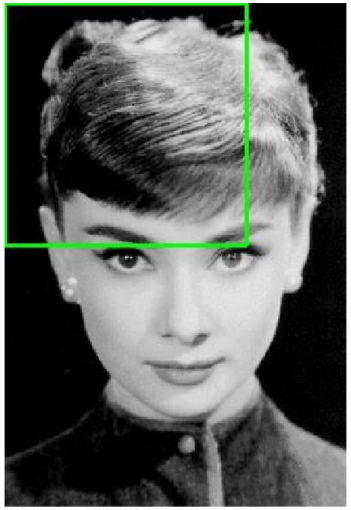
COCO: 200 classes

Challenge 3: Dense vs Sparse

Learning Pictorial Structure (DPM)

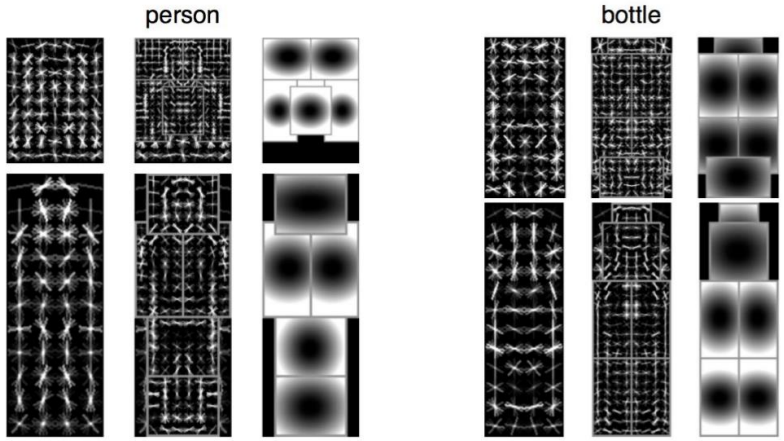
Sliding windows.

- Score every subwindow.

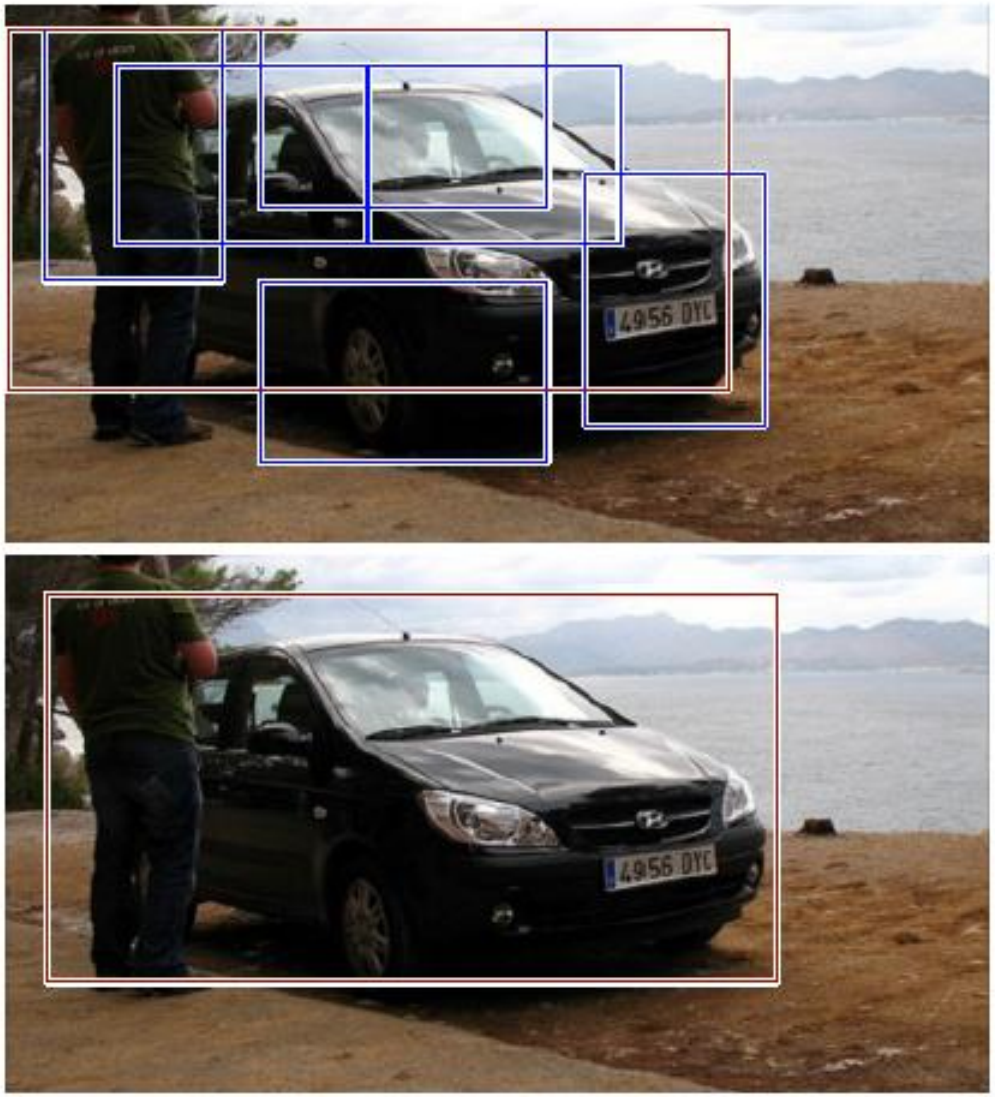


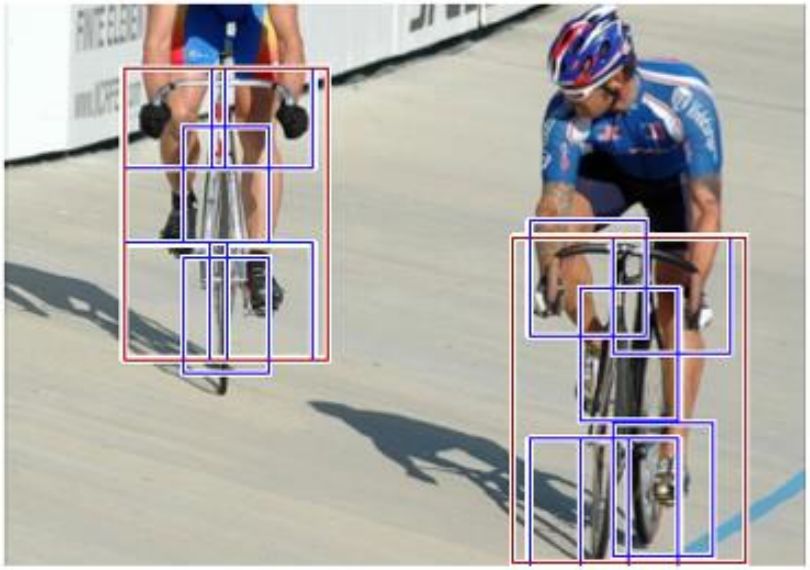
Deformable part models (DPM)

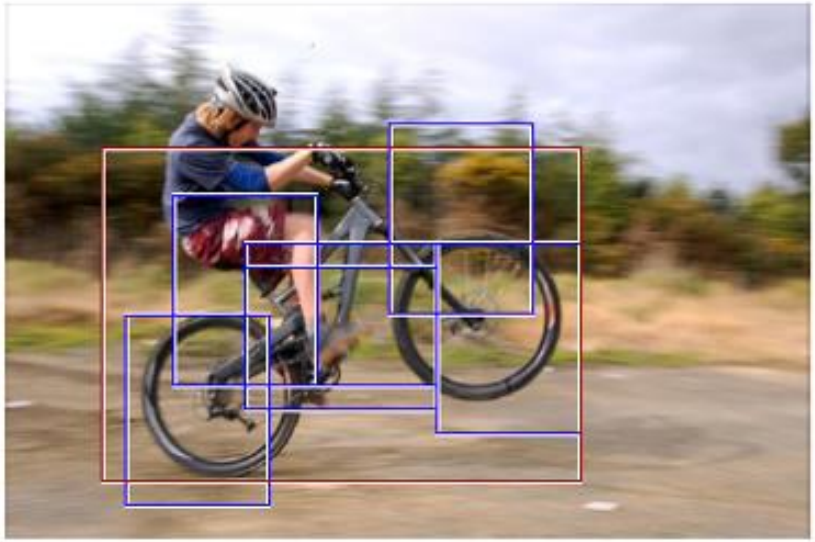
- Uses HOG features
- Very fast

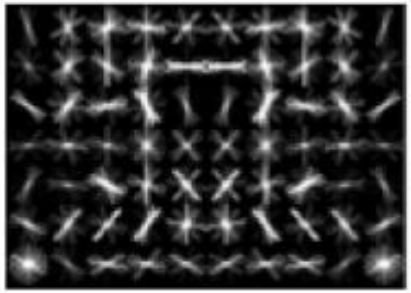


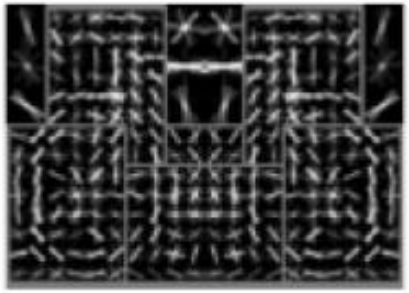
1) fine level with deformable parts
2) coarse level with a fixed template model

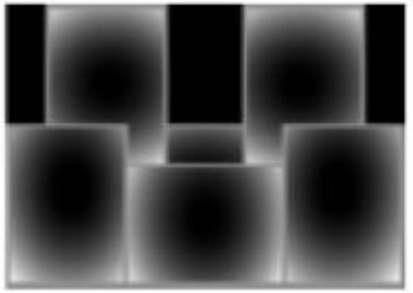


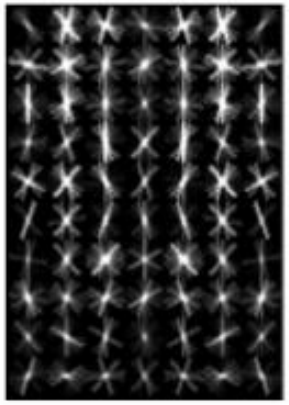


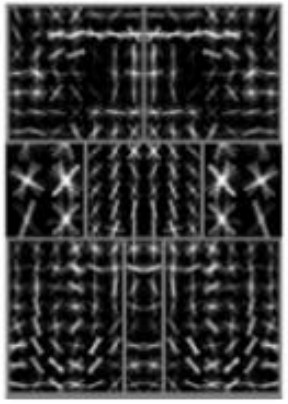


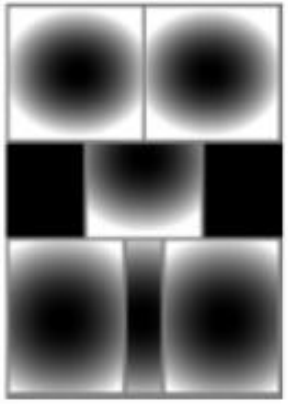


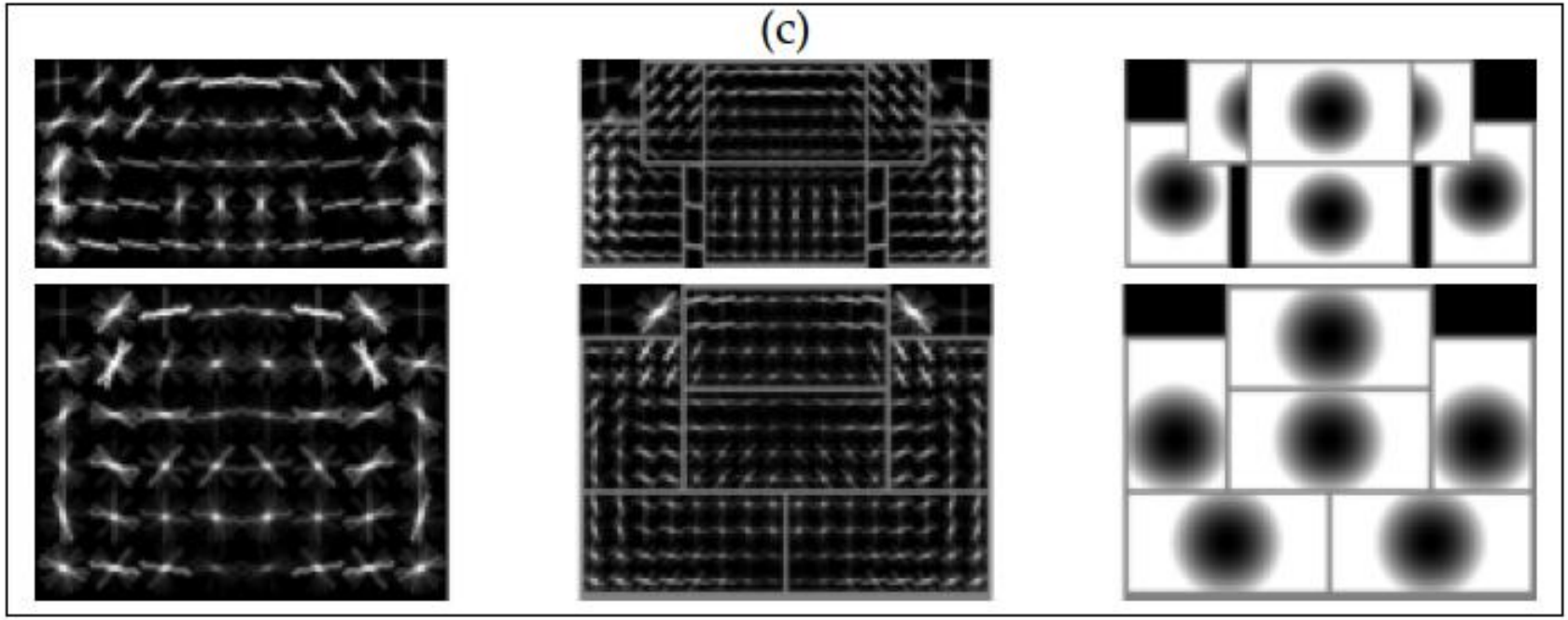




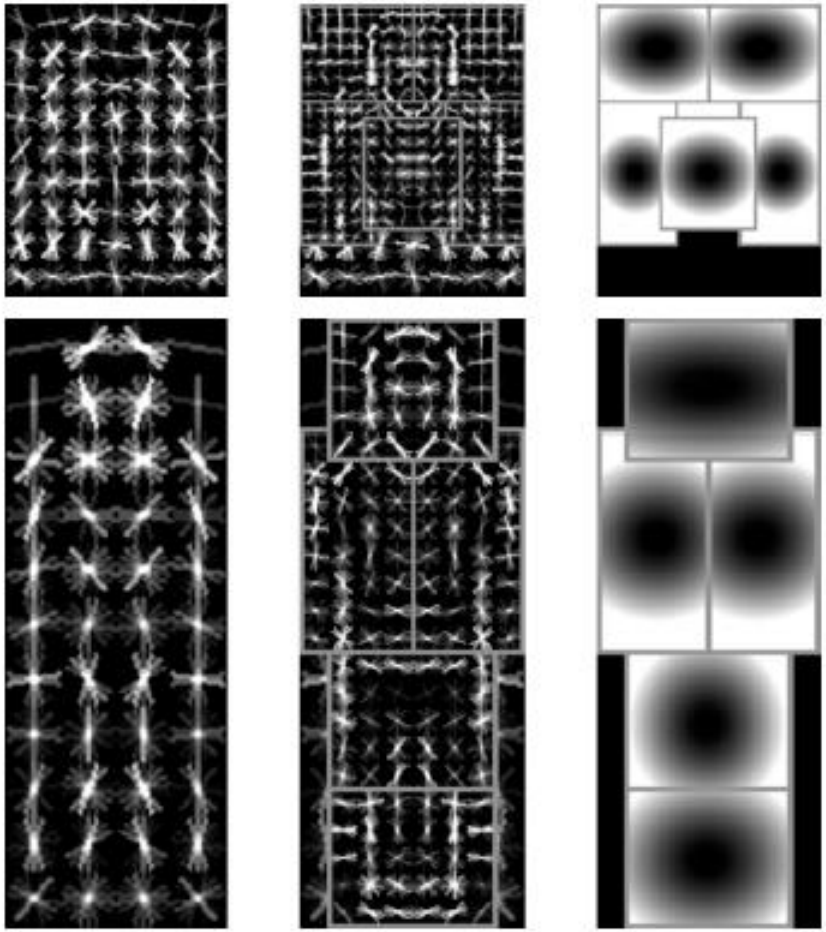




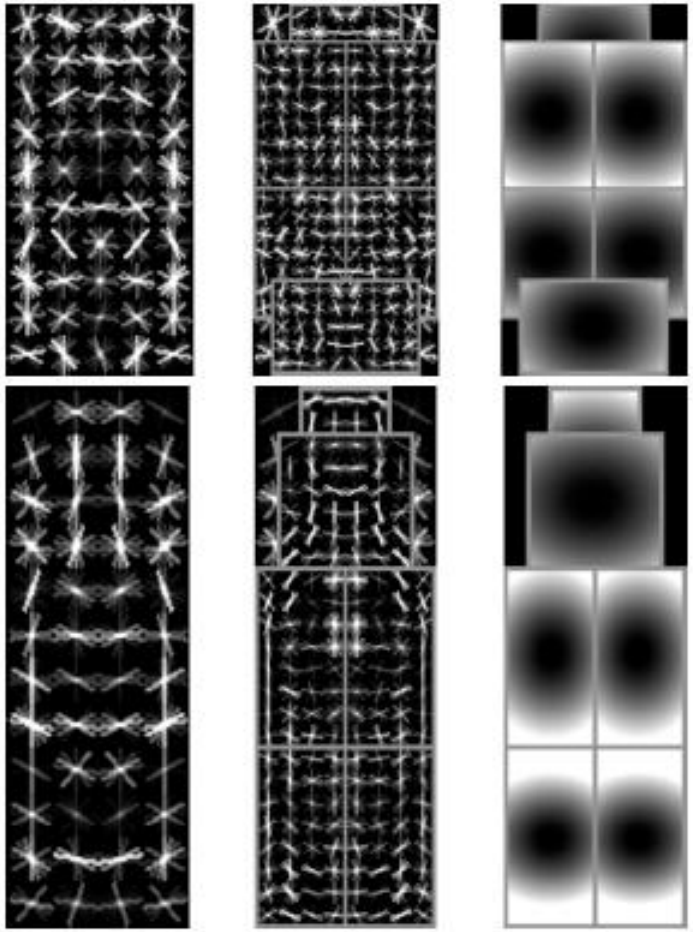


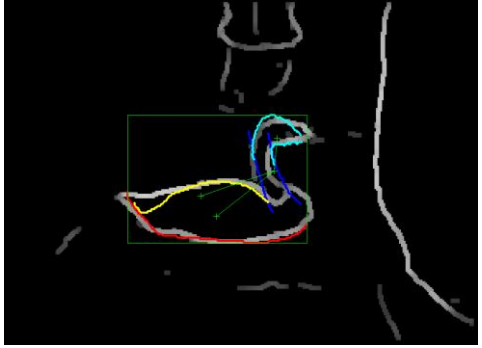

(c)

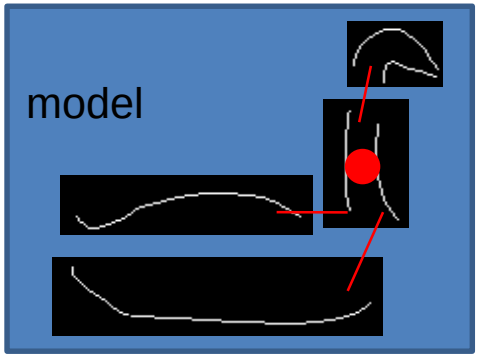
person

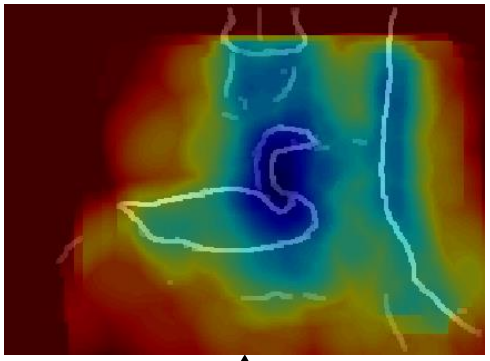


bottle



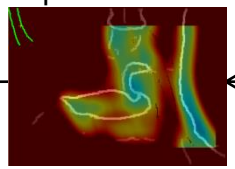


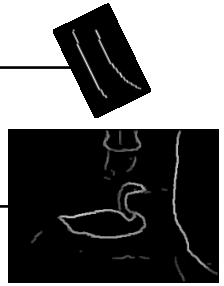


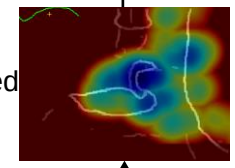


combined cost of root (neck) locations

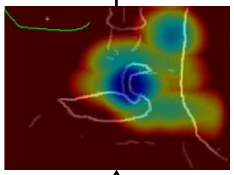
part detection cost

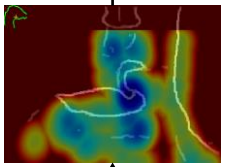


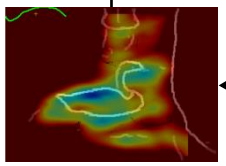




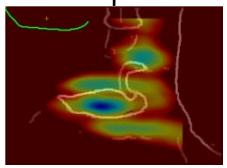
transformed cost

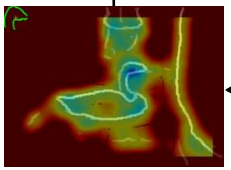


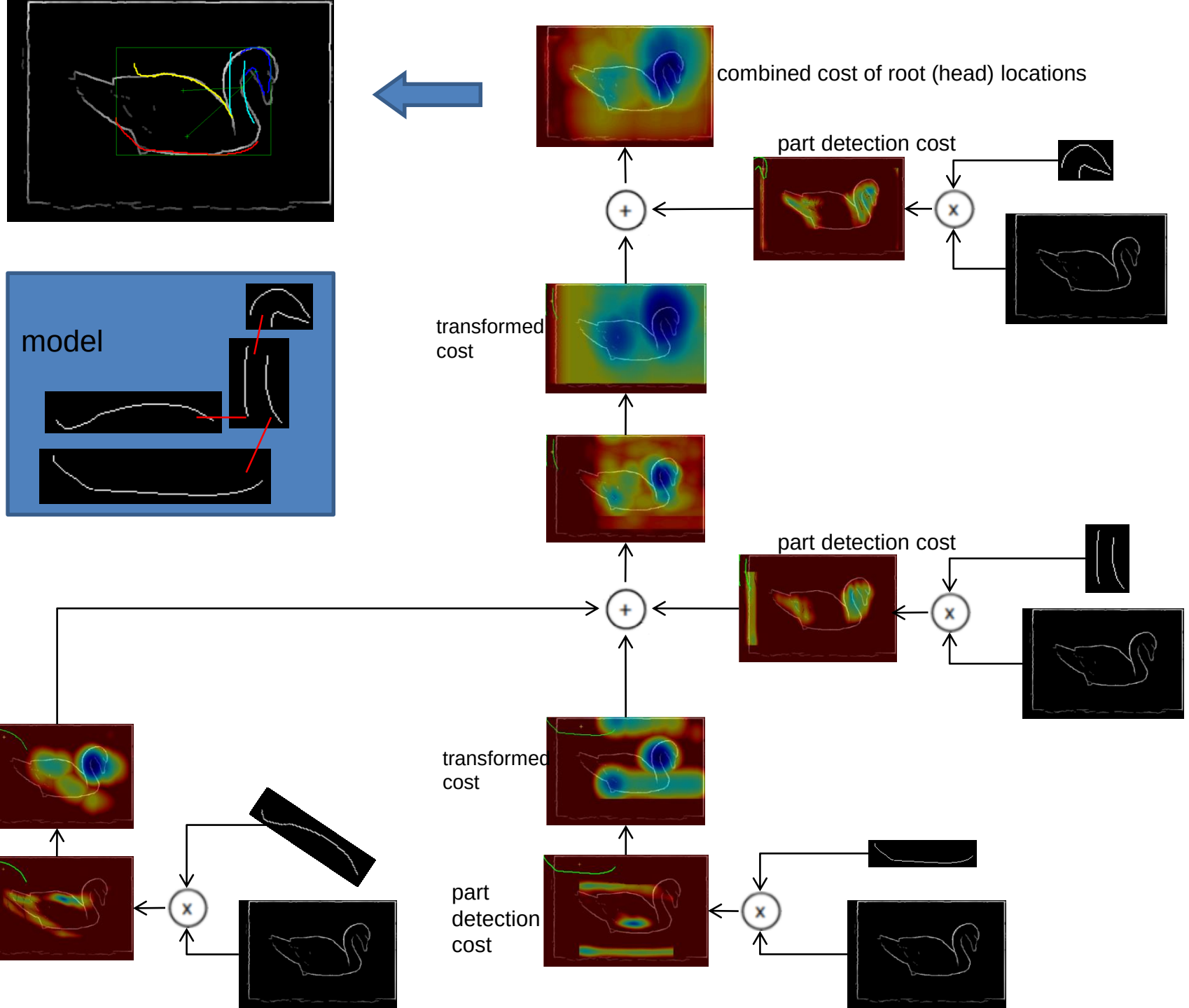


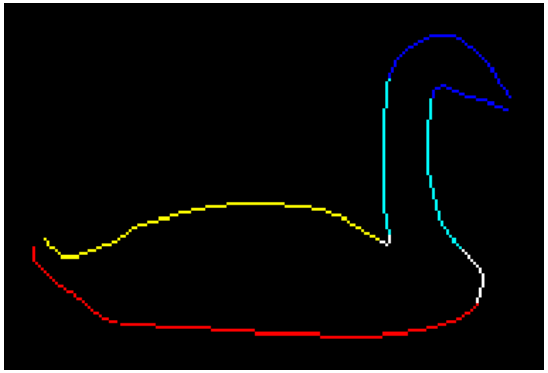


part detection cost

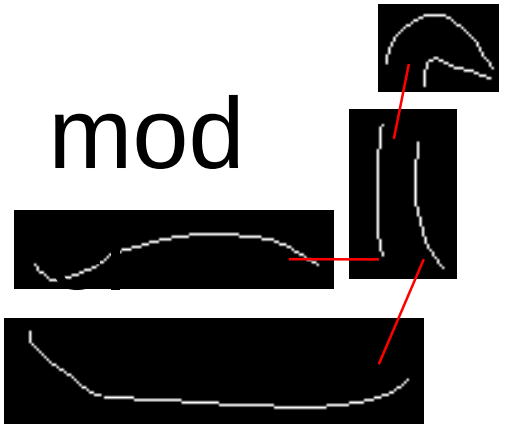


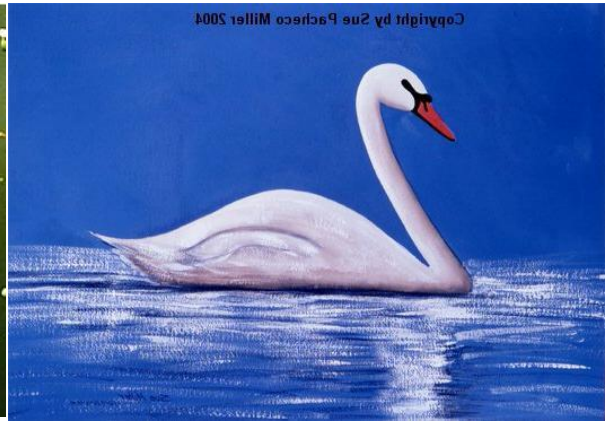


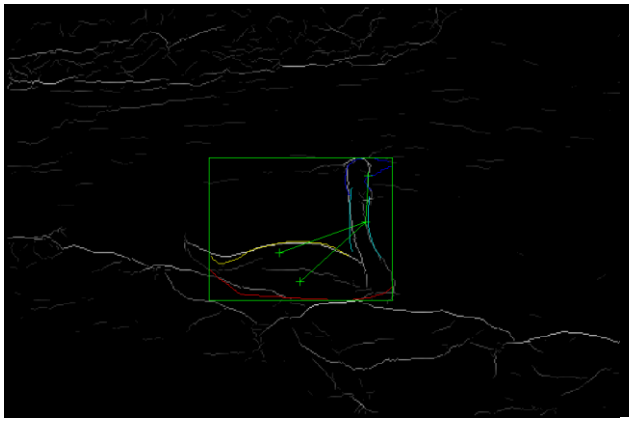

combined cost of root (head) locations
part detection cost
transformed cost
model
part detection cost
transformed cost
transformed cost
part detection cost
part detection cost

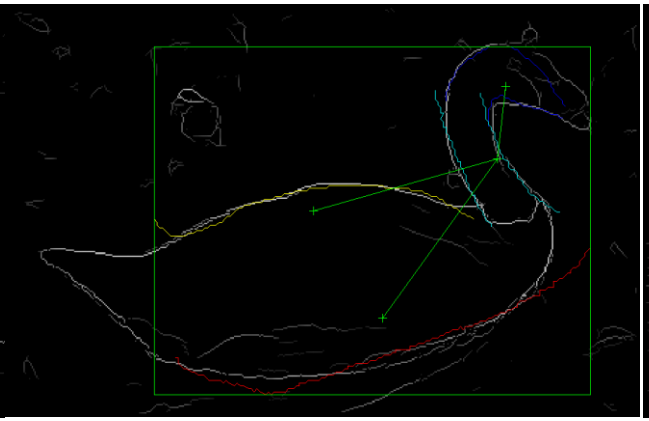


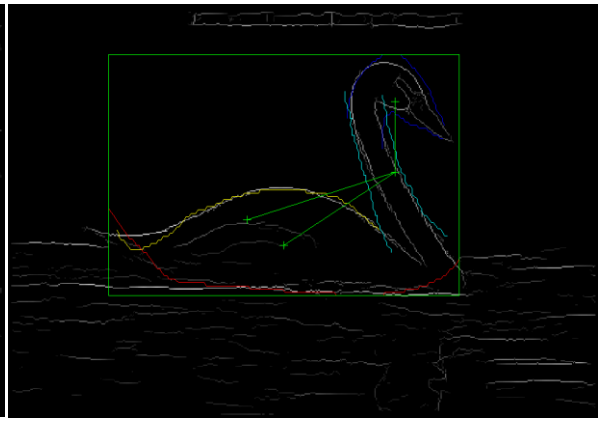
mod



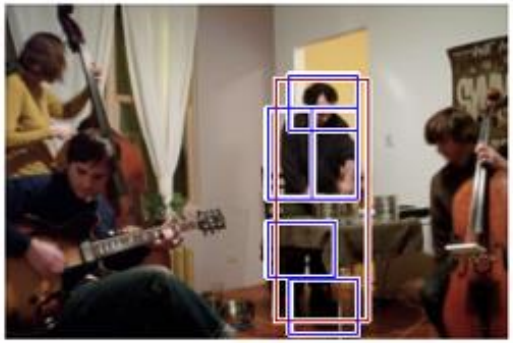
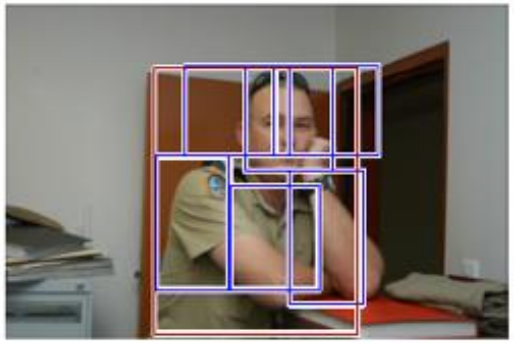
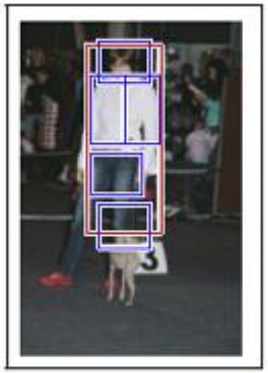




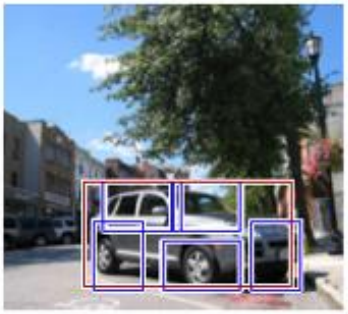


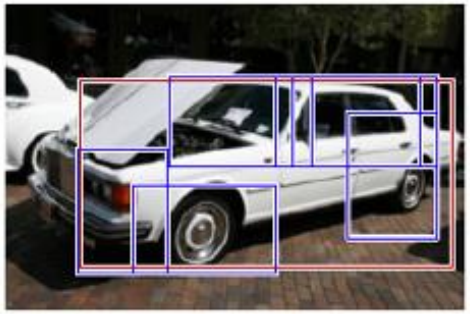
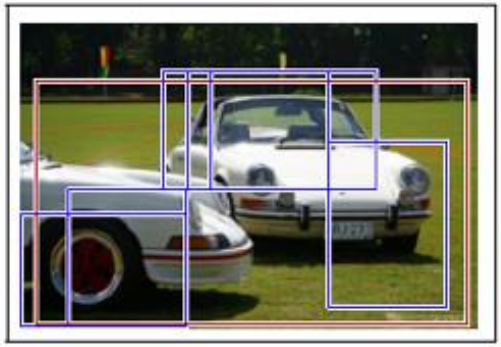
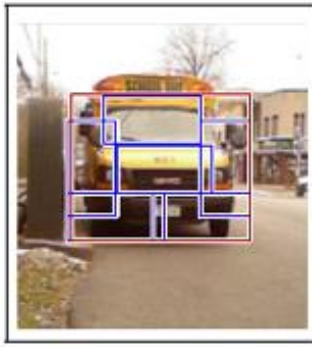


person

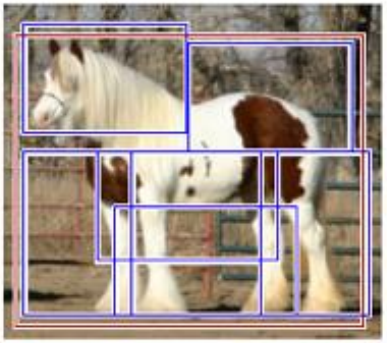





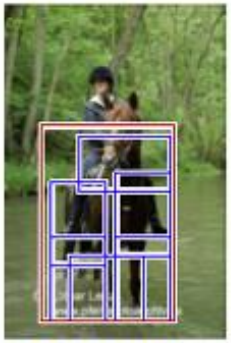
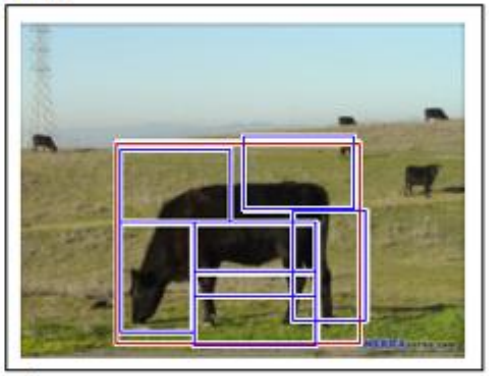
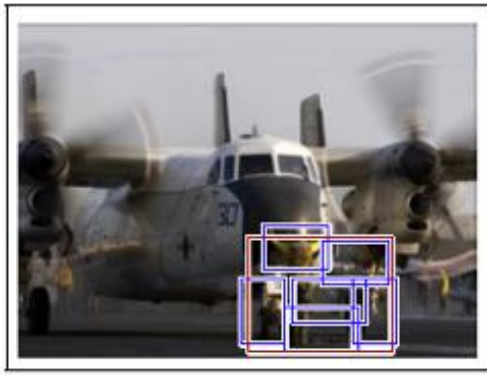

car



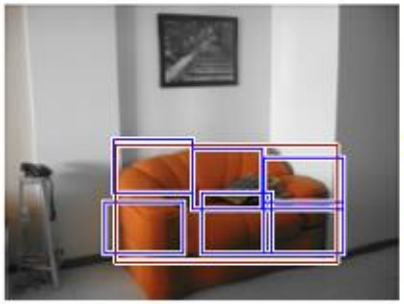




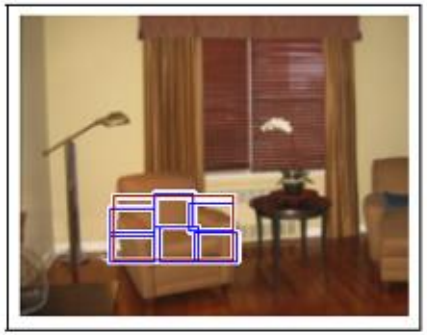
horse



sofa

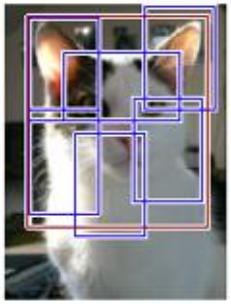
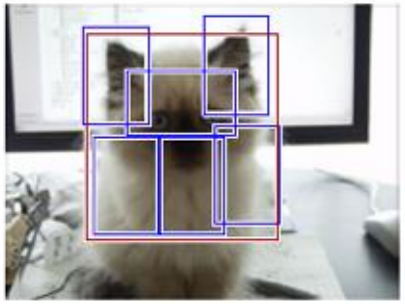
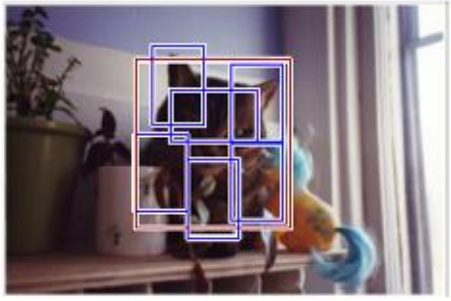
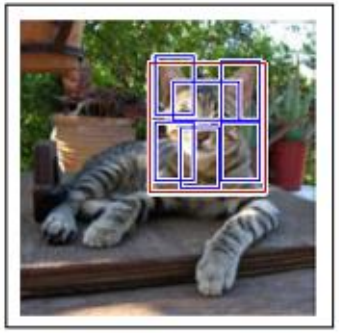
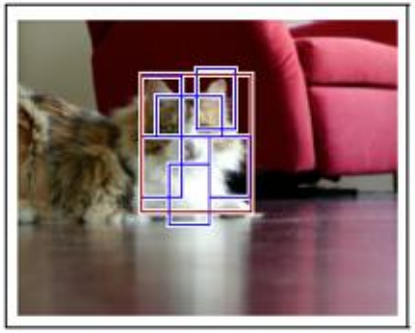


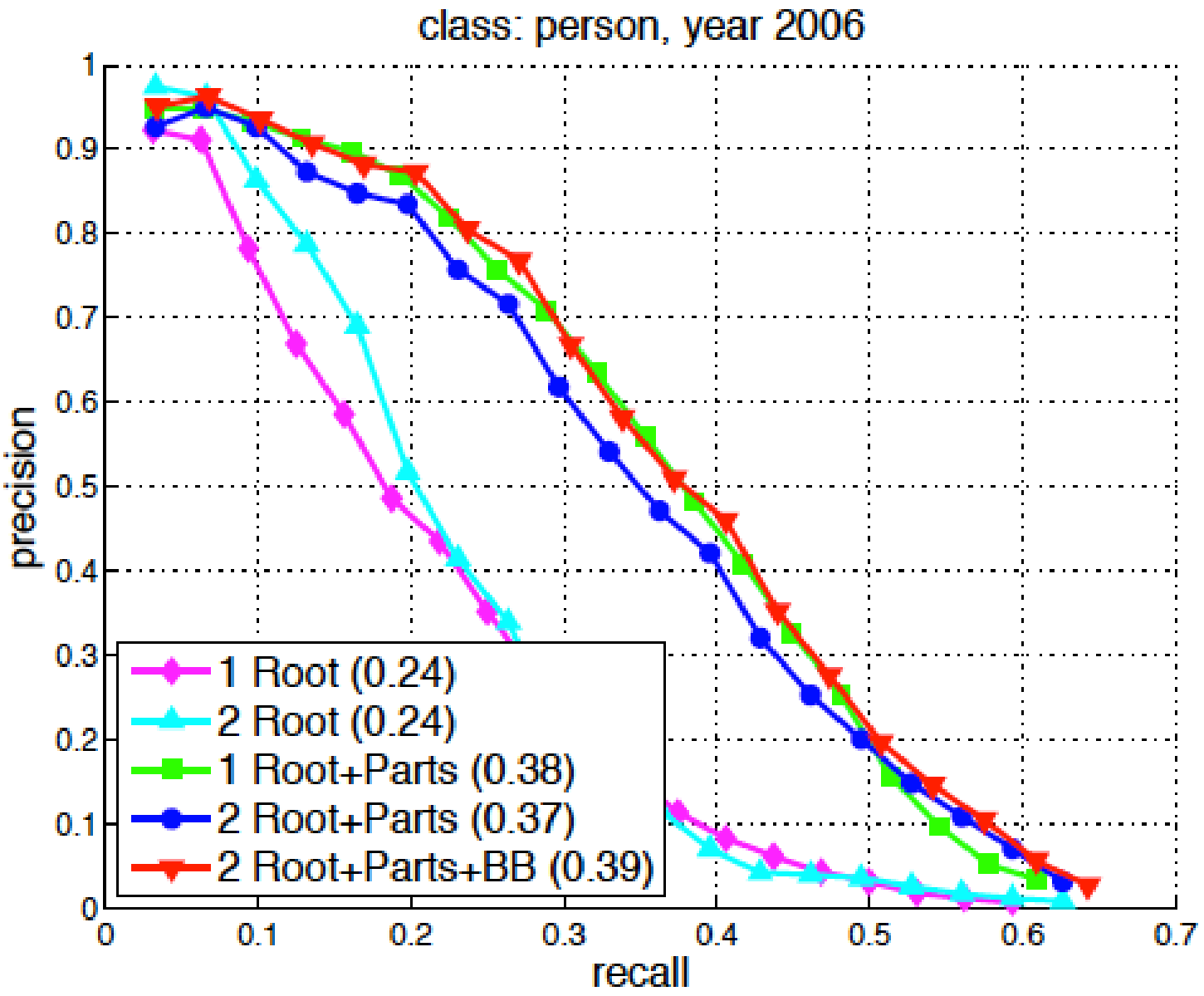



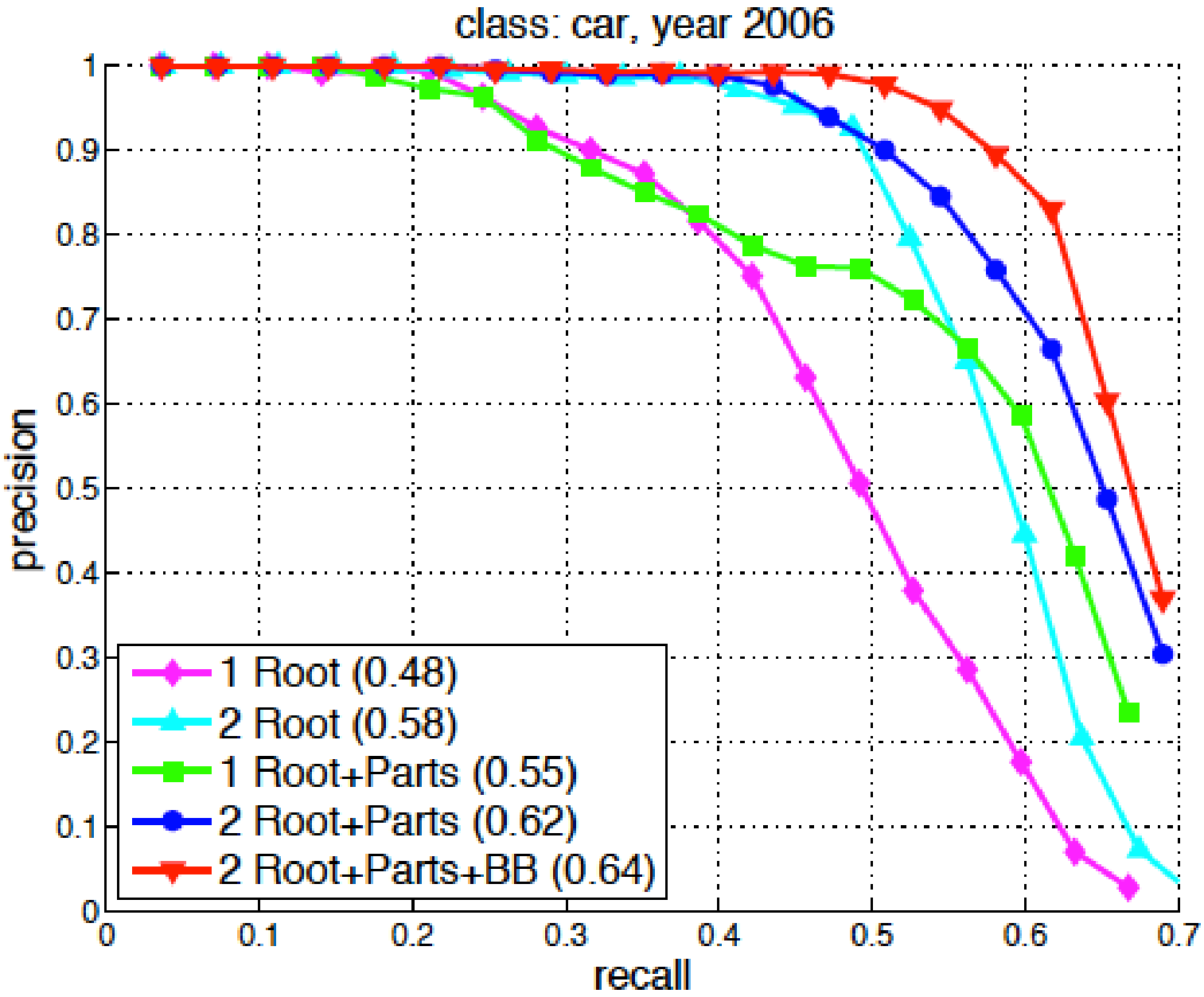
bottle




cat


class: person, year 2006
precision
recall
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
1 Root (0.24)
2 Root (0.24)
1 Root+Parts (0.38)
2 Root+Parts (0.37)
2 Root+Parts+BB (0.39)


class: car, year 2006
precision
recall
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
1 Root (0.48)
2 Root (0.58)
1 Root+Parts (0.55)
2 Root+Parts (0.62)
2 Root+Parts+BB (0.64)

Challenge 3: Dense vs Sparse

Selective Search for Object Recognition

J.R.R. Uijlings*[1,2], K.E.A. van de Sande†[2], T. Gevers[2], and A.W.M. Smeulders[2]

[1]University of Trento, Italy
[2]University of Amsterdam, the Netherlands

Technical Report 2012, submitted to IJCV

Figure 2: Two examples of our selective search showing the necessity of different scales. On the left we find many objects at different scales. On the right we necessarily find the objects at different scales as the girl is contained by the tv.

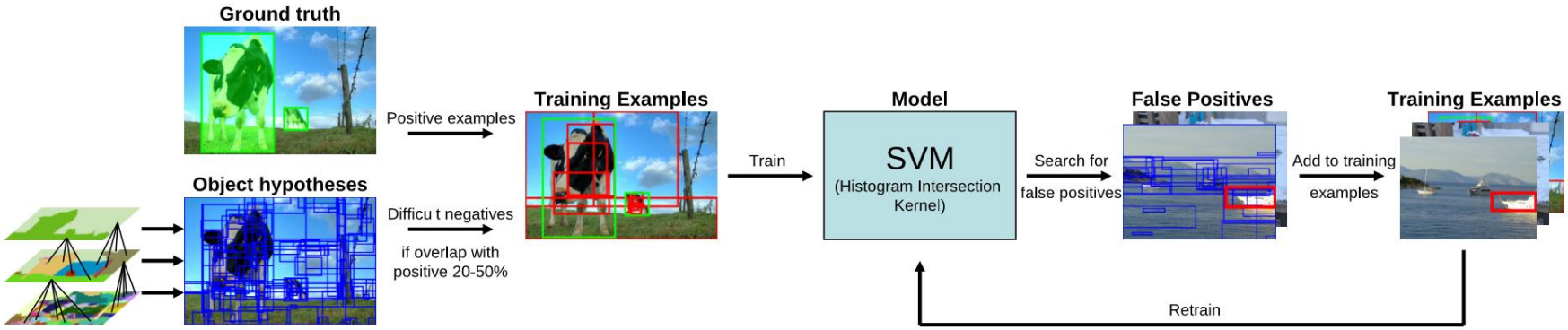


Figure 3: The training procedure of our object recognition pipeline. As positive learning examples we use the ground truth. As negatives we use examples that have a 20-50% overlap with the positive examples. We iteratively add hard negatives using a retraining phase.

(a) Bike: 0.863 (b) Cow: 0.874 (c) Chair: 0.884 (d) Person: 0.882 (e) Plant: 0.873

Figure 5: Examples of locations for objects whose Best Overlap score is around our Mean Average Best Overlap of 0.879. The green boxes are the ground truth. The red boxes are created using the "Quality" selective search.

Lessons and Hard Cases:

Hard Negative Mining

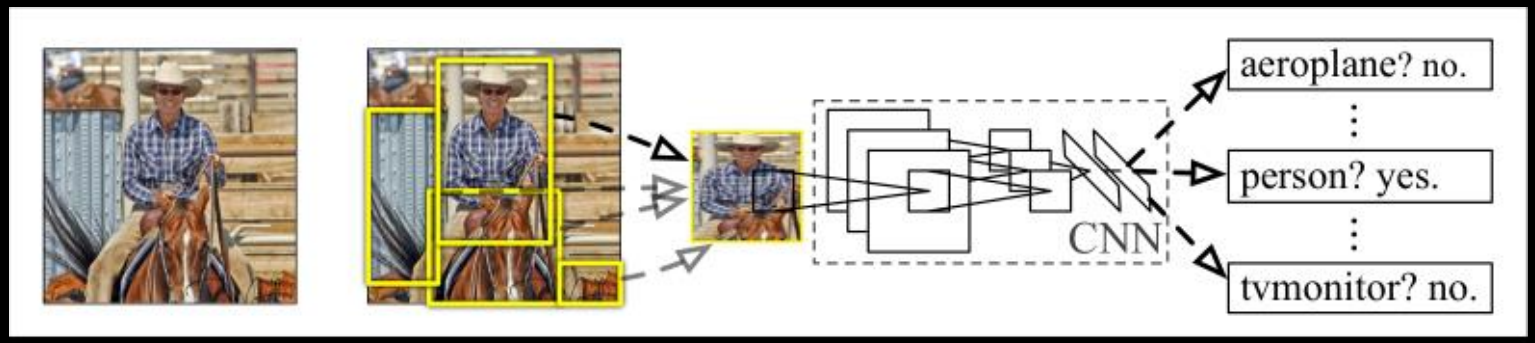
Imbalance between positive and negative examples.

Use negative examples with higher confidence score.

Non Maximum Suppression

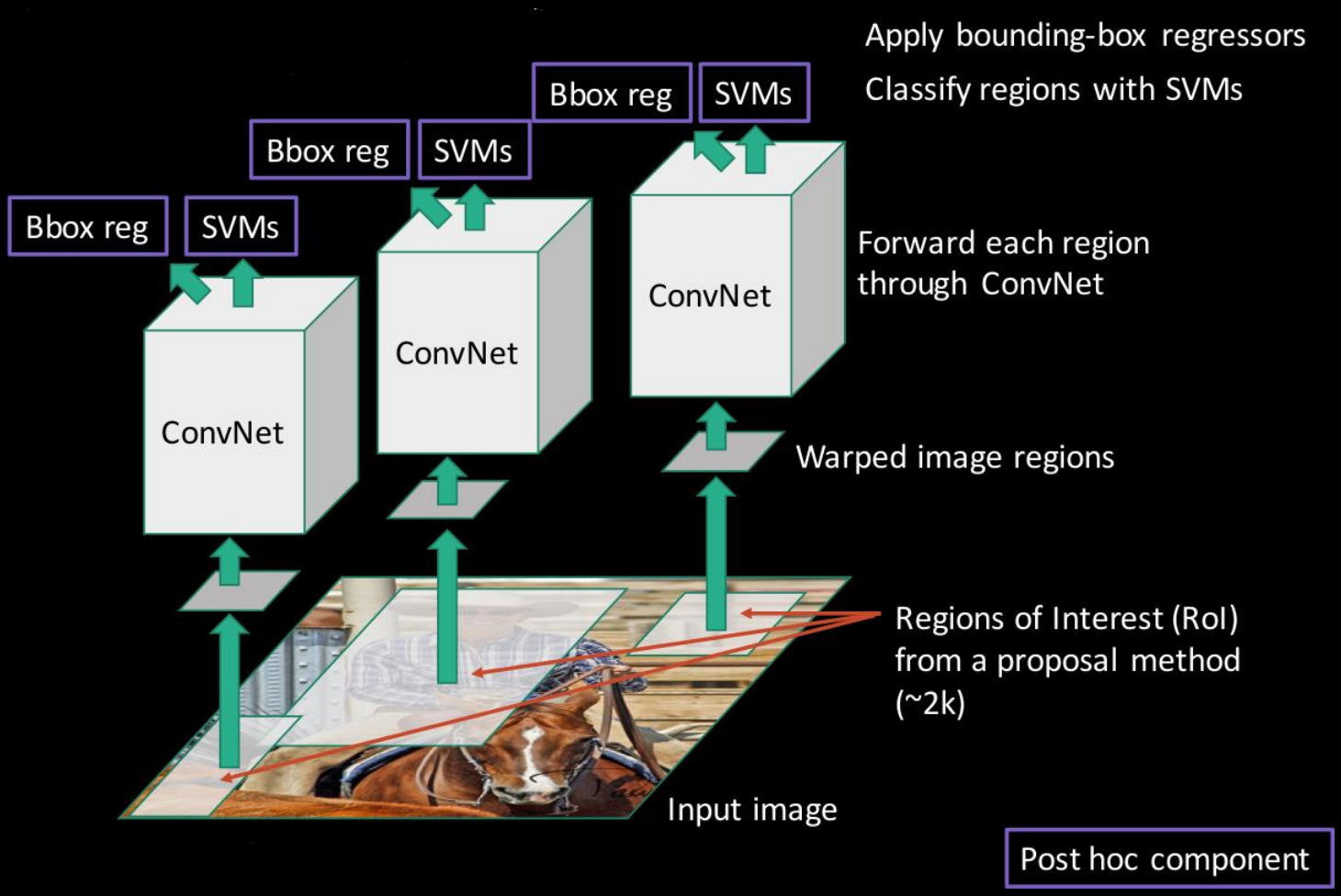
If output boxes overlap, only consider the most confident.

Region based classification: Proposal (where) and Checking (what)



R-CNN: Regions with CNN Features (Girshick et al., 2014)

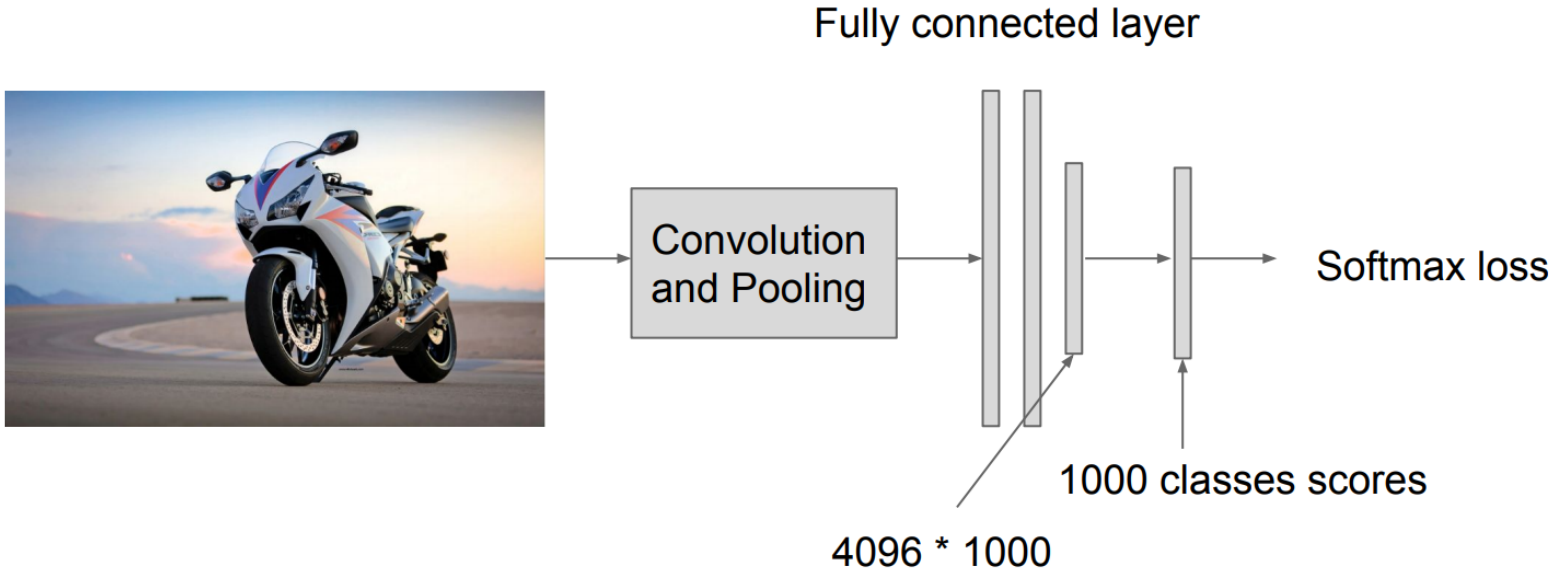
Object Detection



R-CNN: Regions with CNN Features (Girshick et al., 2014)

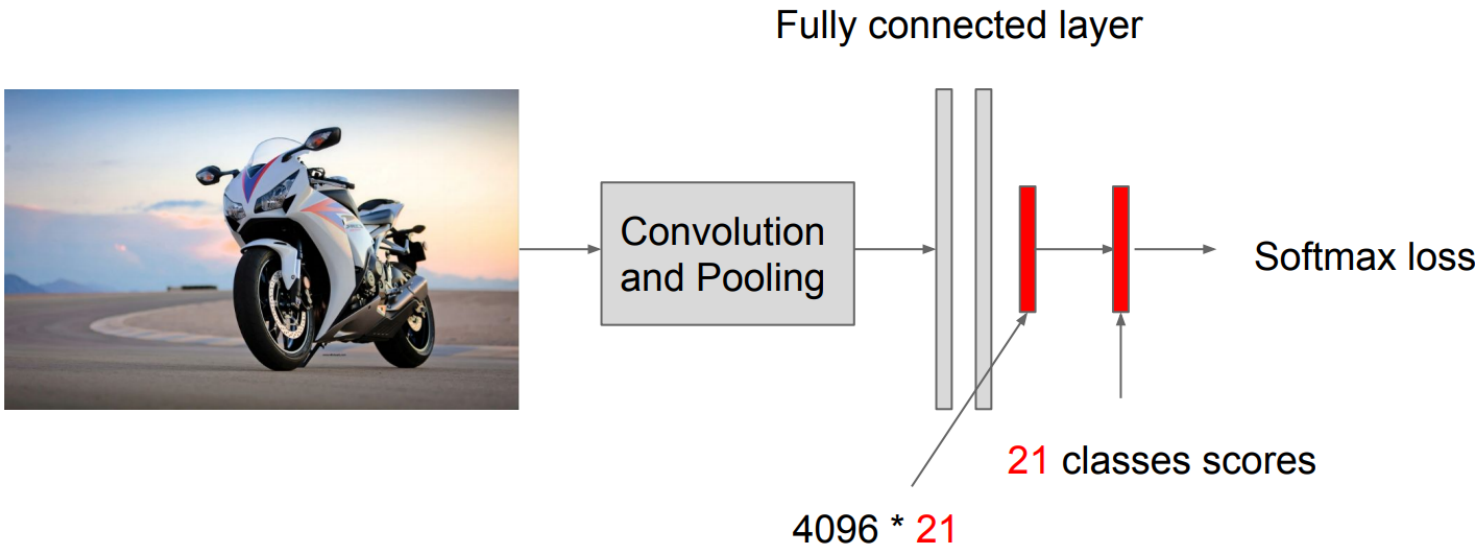
Training RCNN

Step1: train your own CNN model for classification (or use existing model), using ImageNet dataset.



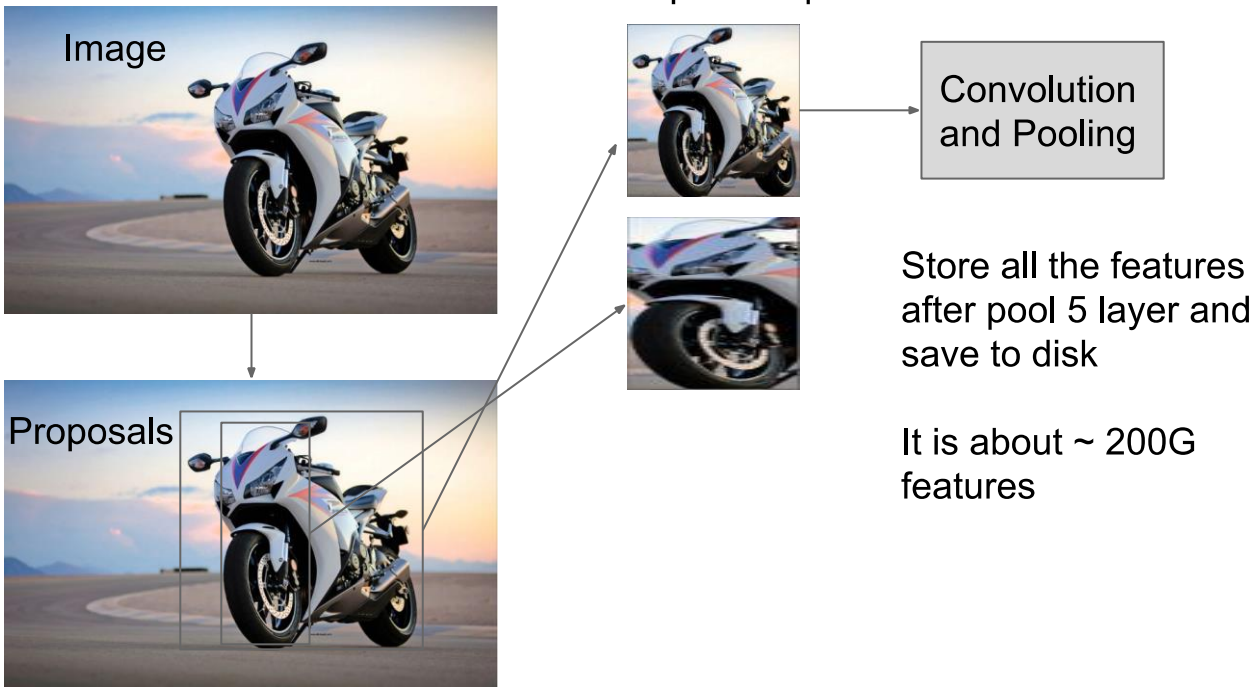
Training RCNN

Step2: focus on 20 classes + 1 background. Remove the last FC layer and replace it with a smaller layer and fine-tune the model using PASCAL VOC dataset



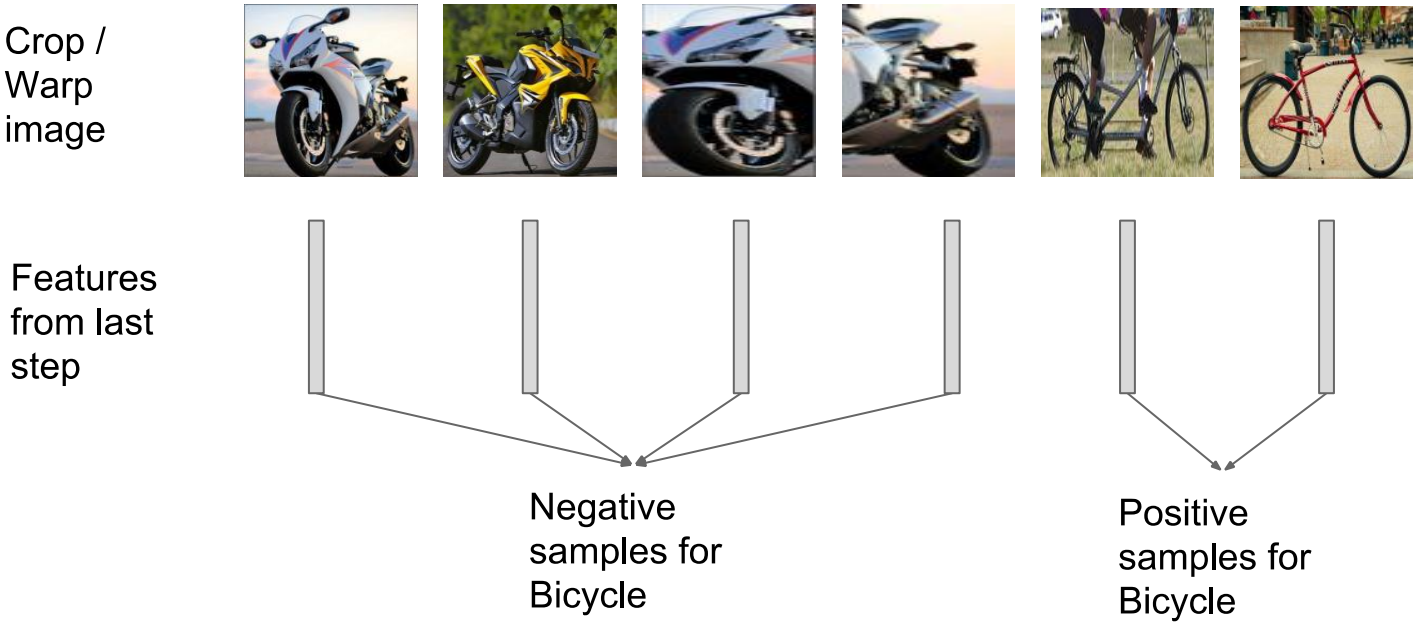
Training RCNN

Step3: extract feature



Training RCNN

Step4: train SVM for each class

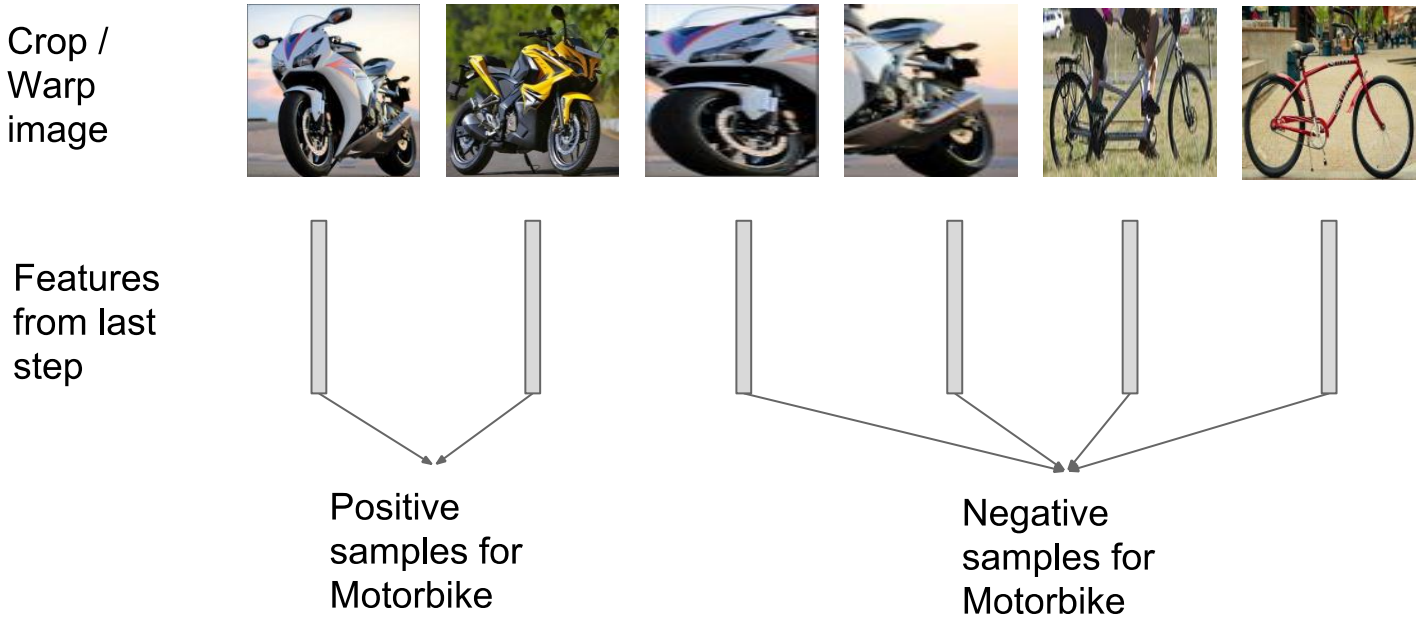


Classify regions by SVM

- linear SVM per class
  (With the sofmax classifier from fine-tuning mAP decreases from 54% to 51%)
- greedy NMS(non-maximum suppression) per class : rejects a region if it has an intersection-overunion (IoU) overlap with a higher scoring selected region larger than a learned threshold

Training RCNN

Step4: train SVM for each class



Object proposal refinement

- Linear bounding-box regression on CNN features (pool_5 feature: mAP ~4% up)

Training R-CNN

•Bounding-box labeled detection data is scarce
•Use supervised pre-training on a data-rich auxiliary task and transfer to detection

Supervised pre-training
Pre-train CNN on ILSVRC2012(1.2 million 1000-way image classification) using image-level annotations only

Domain-specific fine-tuing
Adapt to new task(detection) and new domain(warped proposal)

- random initialize (N+1)-way classification layer (N classes + background)
- Positives: 0.5 IoU overlap with a ground-truth box. Negative: o.w.
- SGD: learning rate: 0.001 (1/10 of original) mini-batch: 32 pos & 96 neg

Train binary SVM

- IoU overlap threshold: grid search over {0, 0.1, ... 0.5}
  IoU = 0.5 : mAP ~5% down
  IoU = 0.0 : mAP ~4% down

SPP-Net

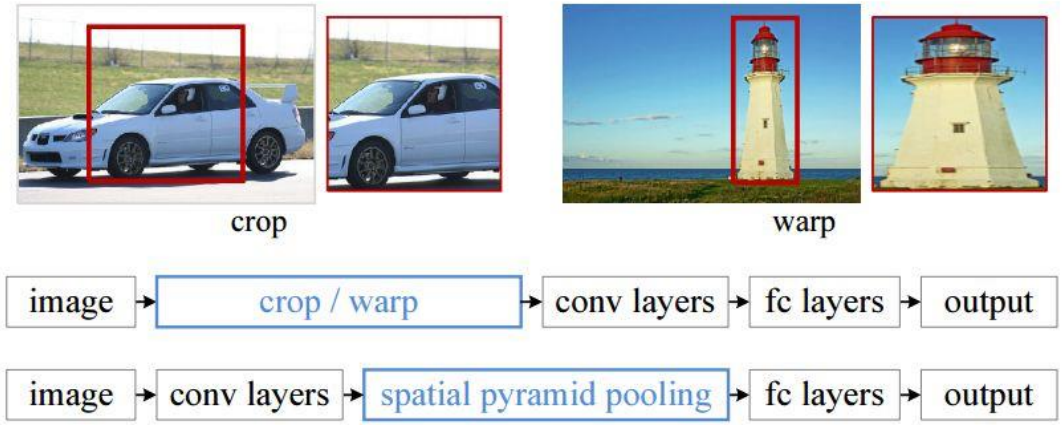
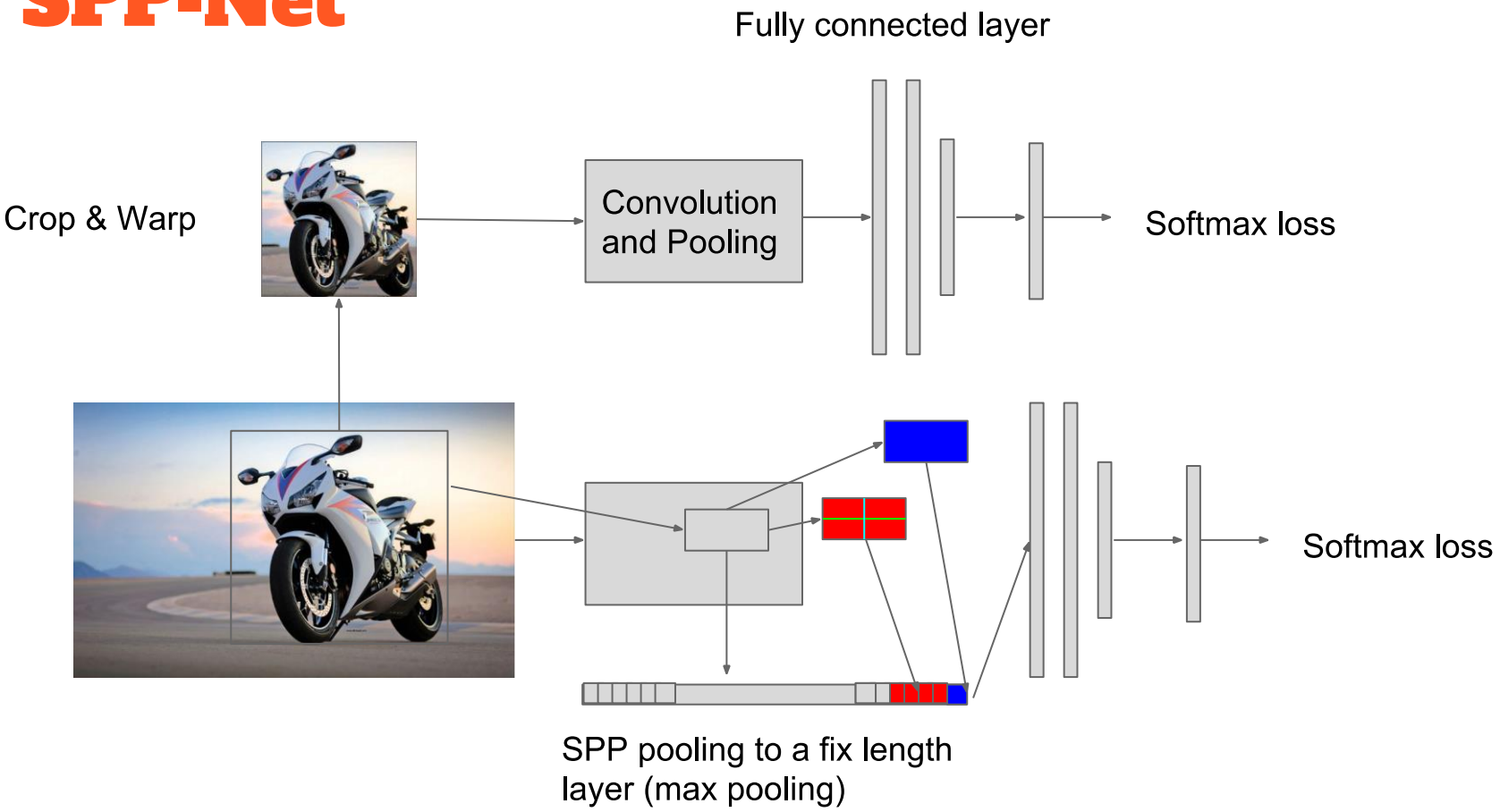


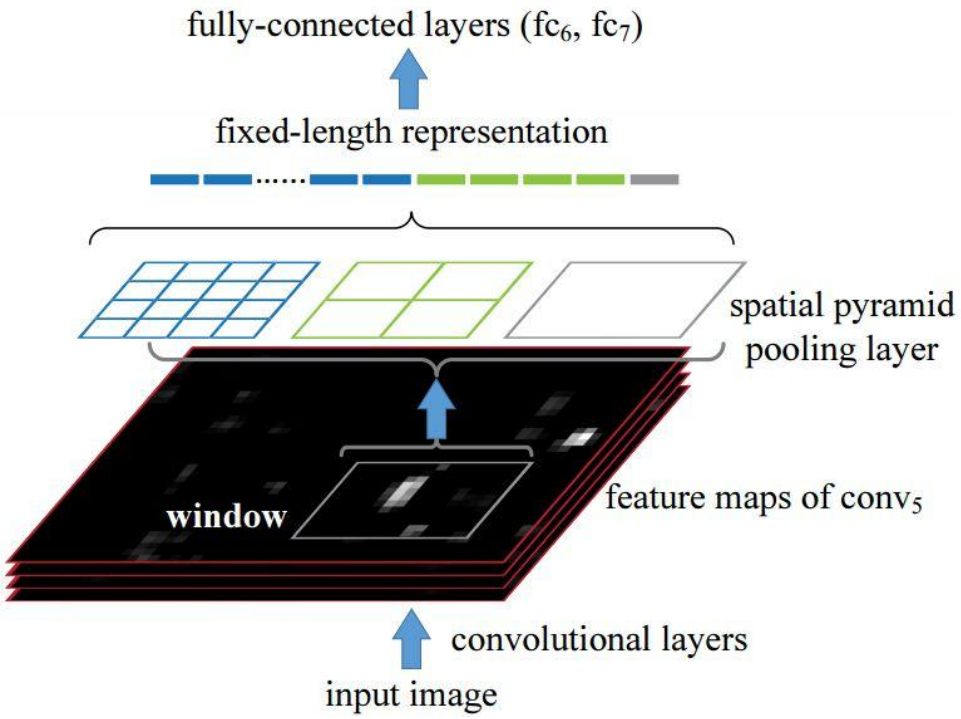
Figure 1: Top: cropping or warping to fit a fixed size. Middle: a conventional CNN. Bottom: our spatial pyramid pooling network structure.

https://arxiv.org/pdf/1406.4729v4.pdf

SPP-Net

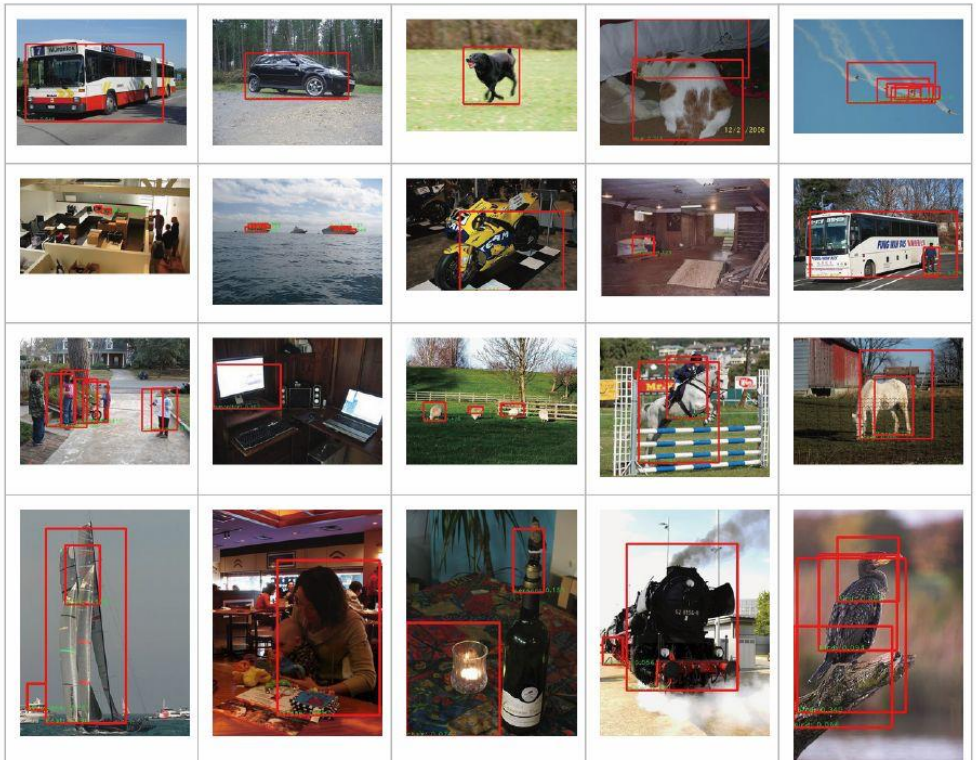

Fully connected layer
Crop & Warp
Convolution and Pooling
Softmax loss
Softmax loss
SPP pooling to a fix length layer (max pooling)

SPP-Net



https://arxiv.org/pdf/1406.4729v4.pdf

Multi-Box



Goal: Achieve a class-agnostic scalable object detection by predicting a set of bounding boxes.

Train a neural network to directly predict:

- The upper-left and lower-right coordinates of each bounding box
- The confidence score for the box containing an object

Fast RCNN

Share convolution layers for proposals from the same image

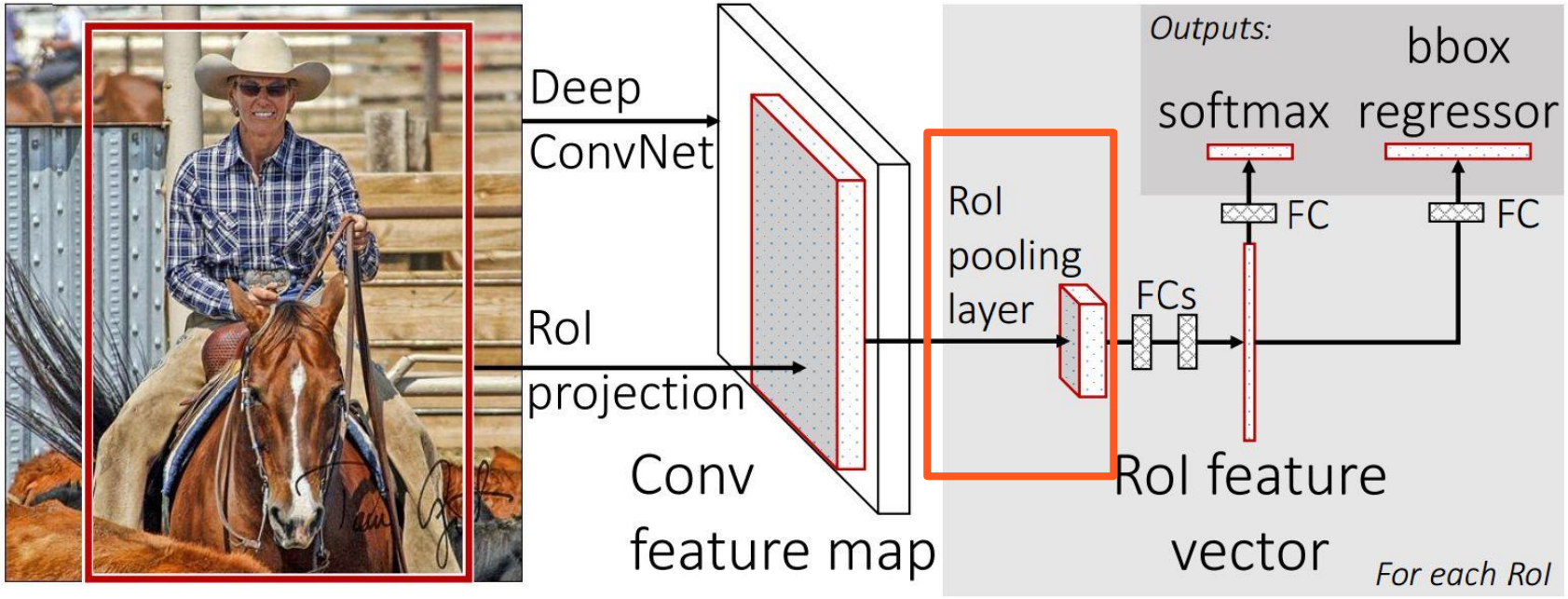
Faster and More accurate than RCNN

ROI Pooling

Fast R-CNN
Ross Girshick
Apr 2015

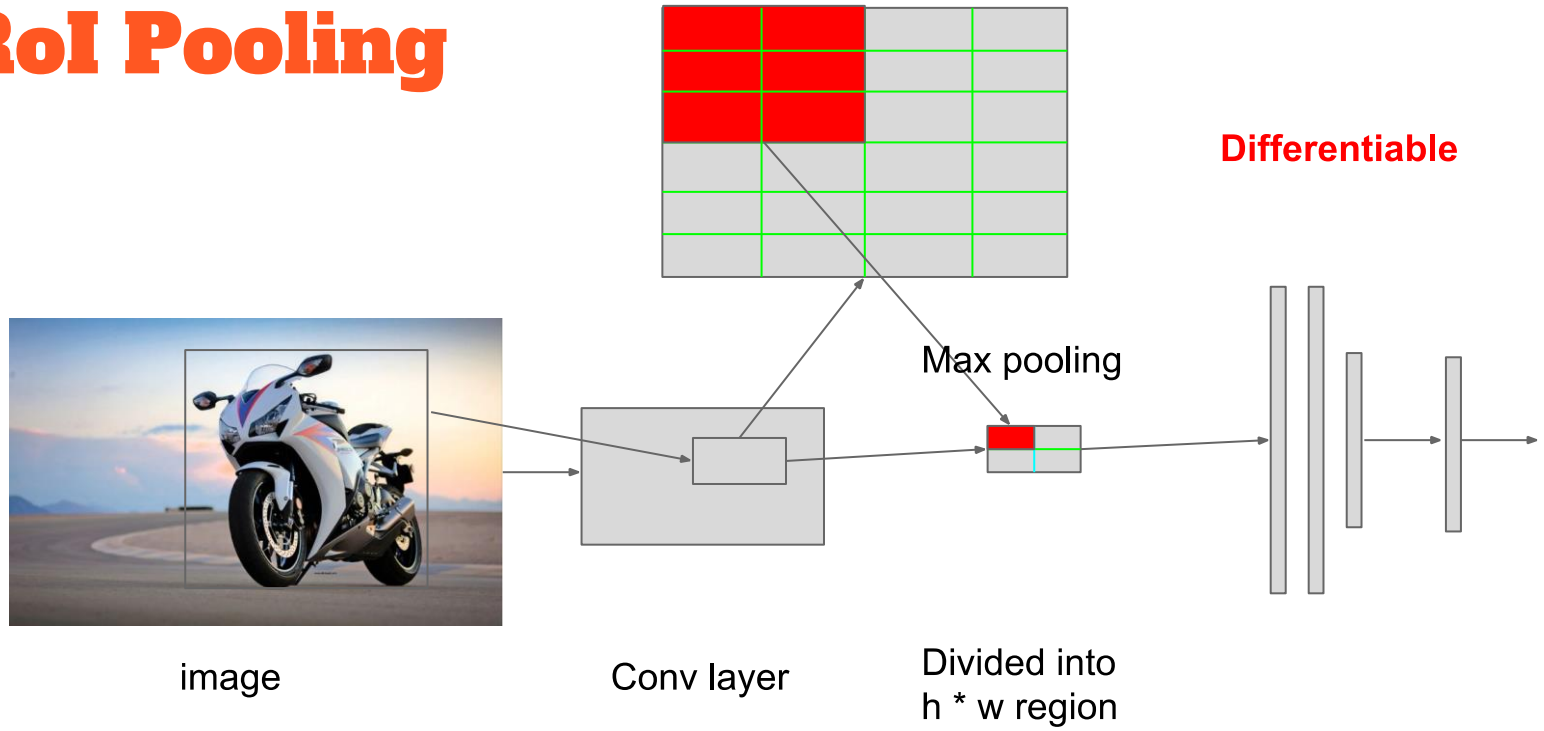
https://arxiv.org/pdf/1504.08083v2.pdf

Fast RCNN



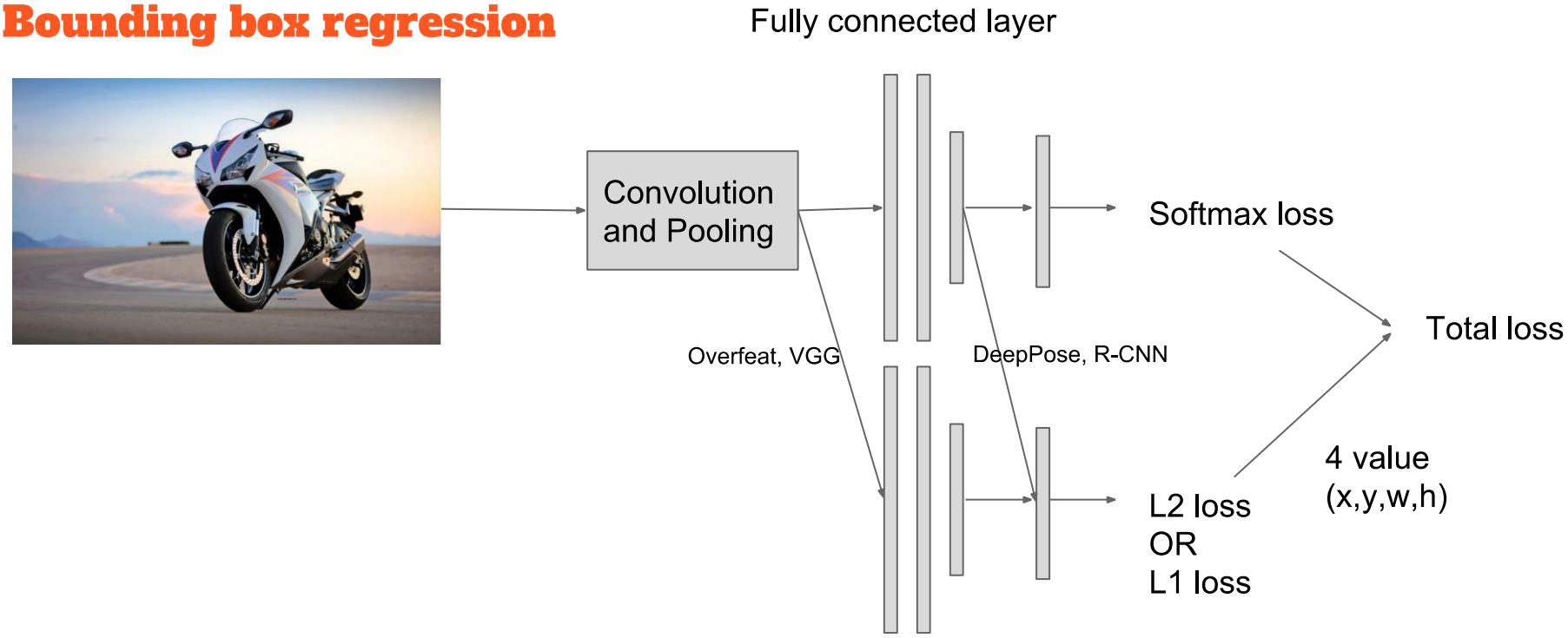
https://arxiv.org/pdf/1504.08083v2.pdf

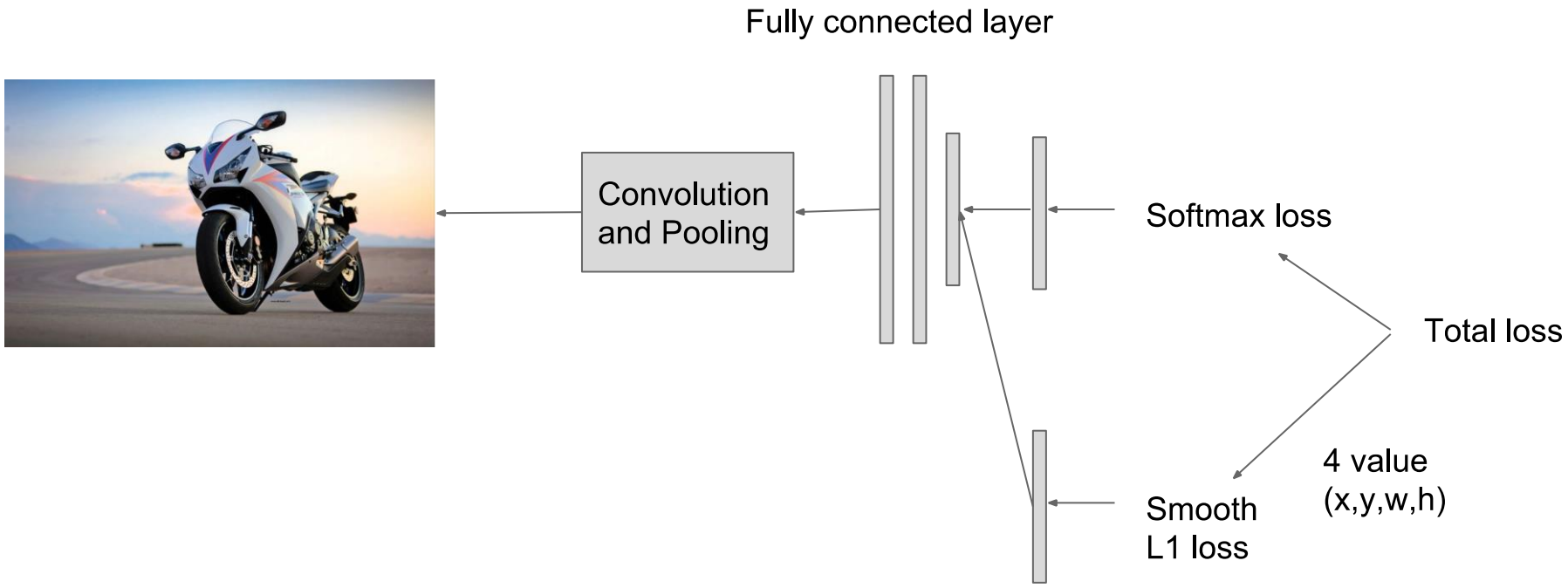
RoI Pooling


Differentiable
Max pooling
image
Conv layer
Divided into
h * w region

What is bbox-regressor?

Bounding box regression

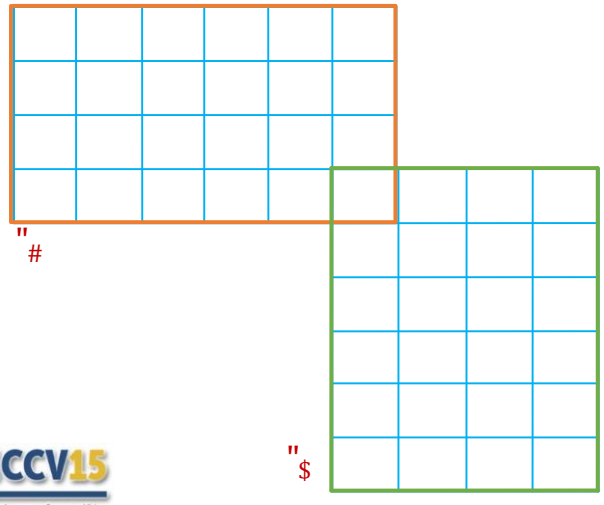



Fully connected layer
Convolution and Pooling
Softmax loss
Total loss
4 value (x,y,w,h)
Smooth L1 loss

facebook

Obstacle #1: Differentiable RoI pooling

RoI pooling / SPP is just like max pooling, except that pooling regions overlap

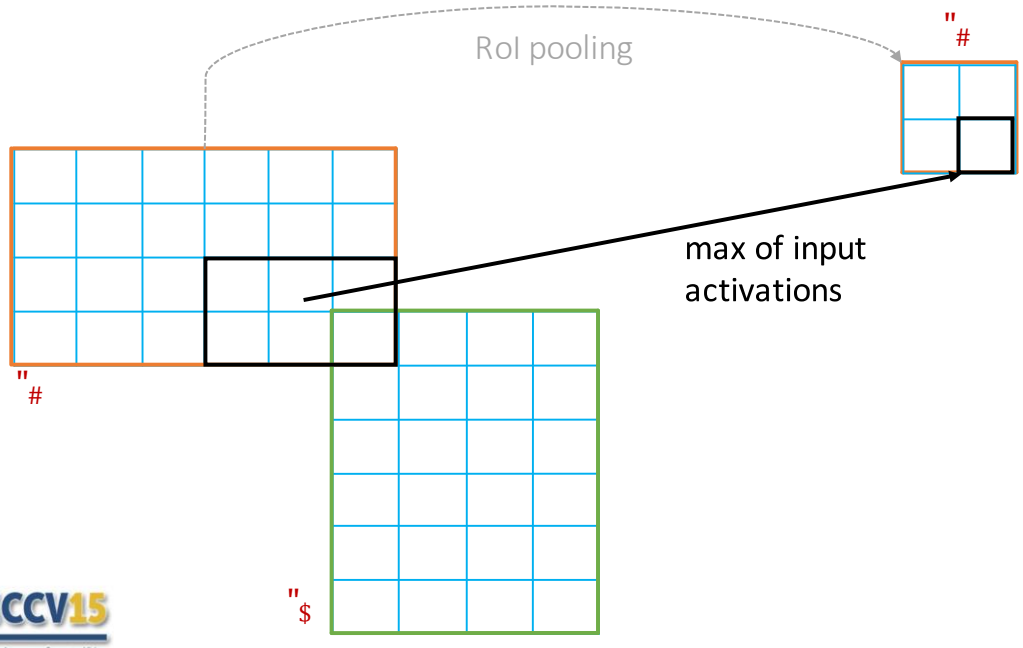


ICCV15 International Conference on Computer Vision

Ross Girshick. "Fast R-CNN". ICCV 2015.

Obstacle'#1:'Differentiable'RoI pooling

RoI pooling'/'SPP'is'just'like'max'pooling,'except'that'pooling'regions'overlap

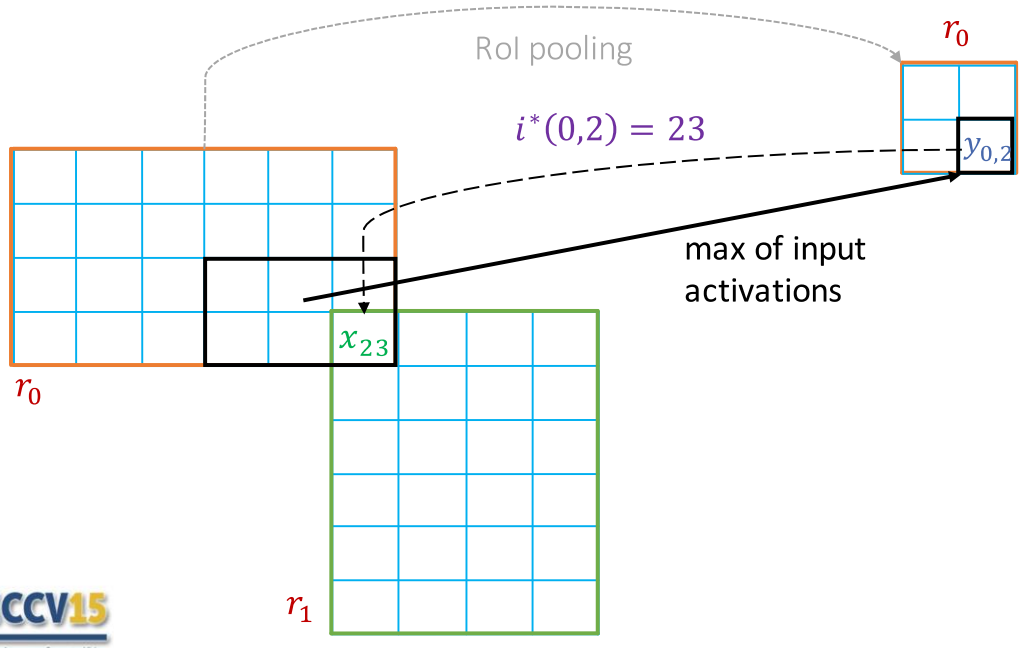


Ross Girshick. "Fast R-CNN". ICCV 2015.

facebook

Obstacle #1: Differentiable RoI pooling

RoI pooling / SPP is just like max pooling, except that pooling regions overlap



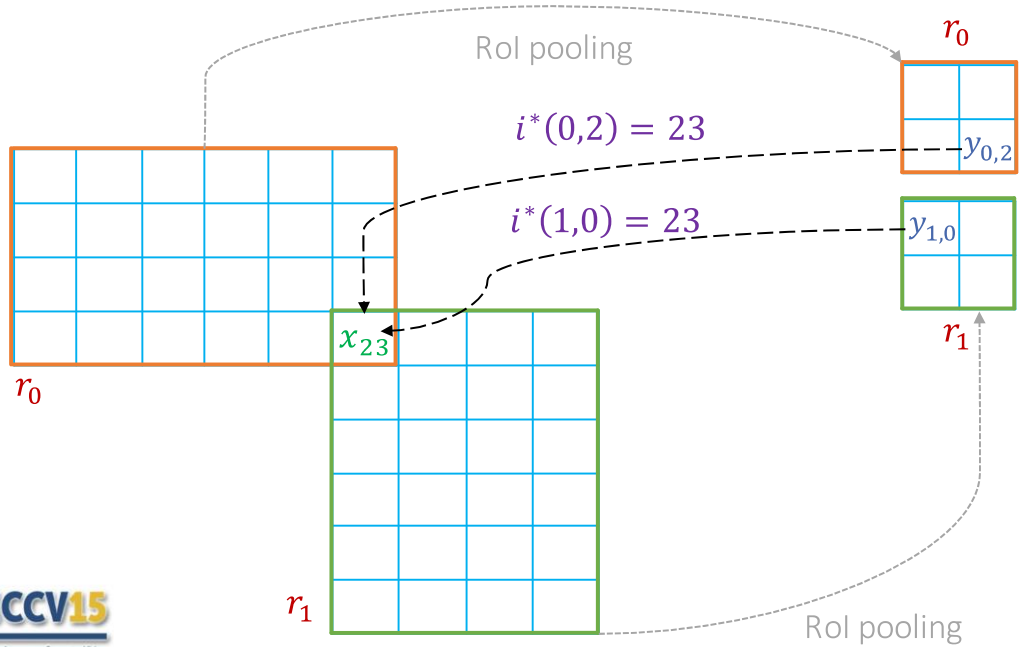
←– – – max pooling "switch" (*i.e.* argmax back-pointer)

Ross Girshick. "Fast R-CNN". ICCV 2015.

ICCV15 International Conference on Computer Vision

Obstacle #1: Differentiable RoI pooling

RoI pooling / SPP is just like max pooling, except that pooling regions overlap



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

1 if r, j "pooled" input i; 0 o/w

Partial for x_i

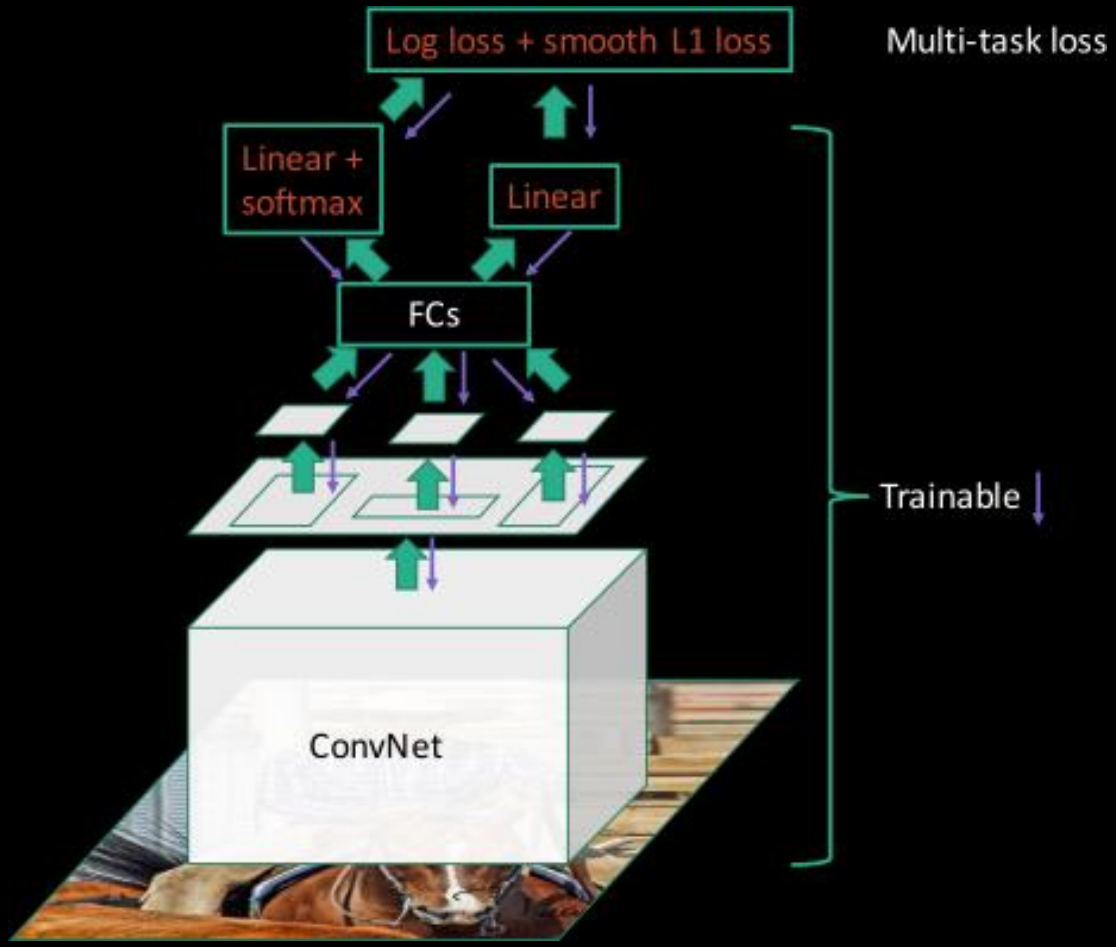
Over regions r, locations j

Partial from next layer

←– – – max pooling "switch" (*i.e.* argmax back-pointer)

Ross Girshick. "Fast R-CNN". ICCV 2015.

Object Detection

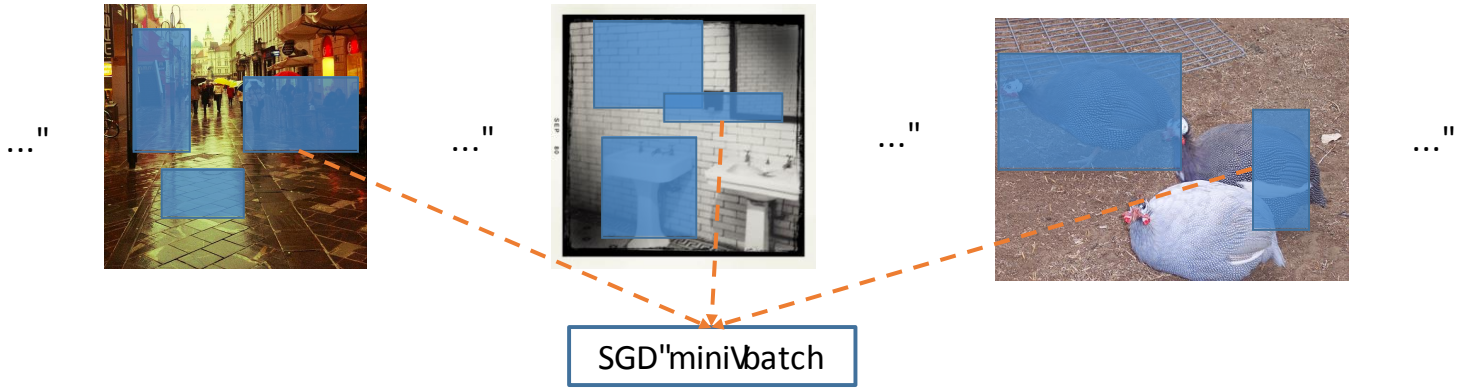


Fast R-CNN (Girshick, 2015)

Obstacle #2: Making SGD steps efficient

Slow R-CNN and SPP-net use region-wise sampling to make mini-batches

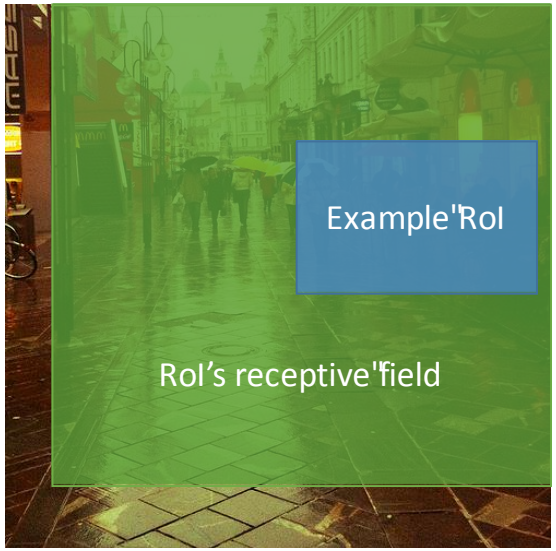
- Sample 128 example RoIs uniformly at random
- Examples will come from different images with high probability



Obstacle'#2:'Making'SGD'steps'efficient

Note'the'receptive'field'for'one'example'RoI is'often'very'large

- Worst'case:'the'receptive'field'is'the'entire'image

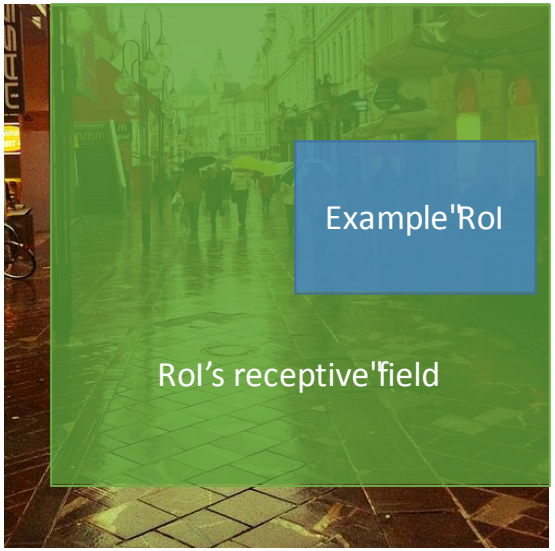


Obstacle #2: Making SGD steps efficient

Worst case cost per mini-batch (crude model of computational complexity)

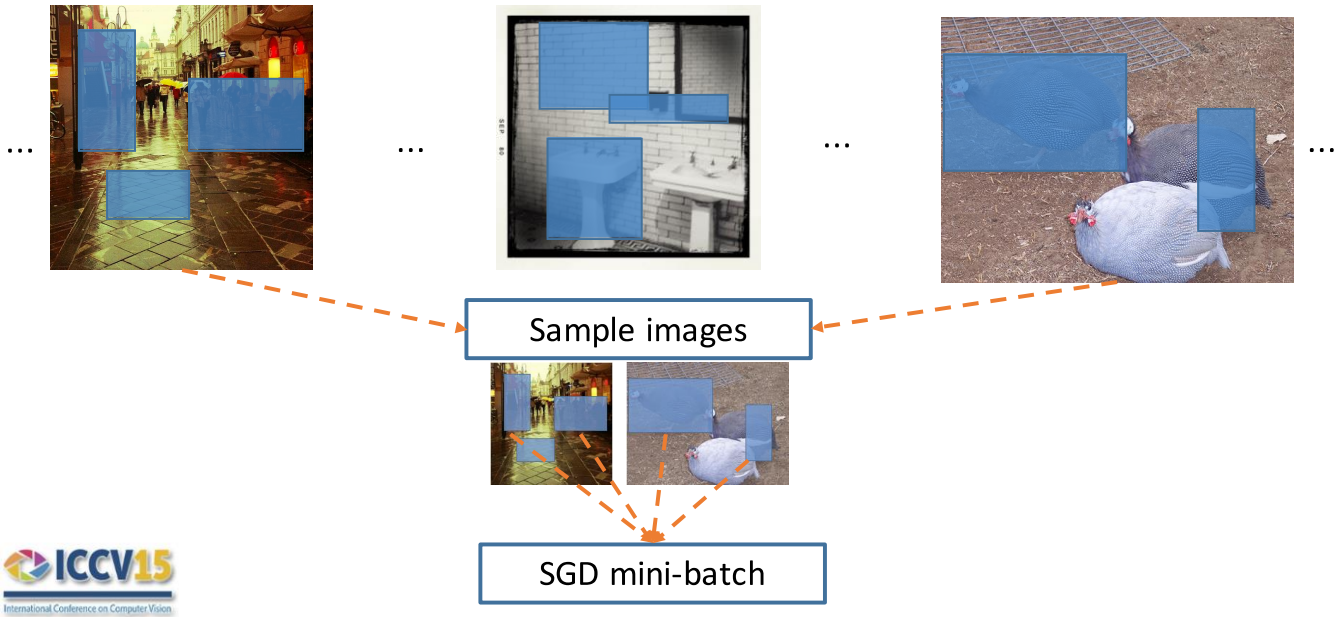
- 128*600*1000 / (128*224 *224) = 12x > computation than slow R-CNN

input size for Fast R-CNN input size for slow R-CNN



Obstacle #2: Making SGD steps efficient

Solution: use hierarchical sampling to build mini-batches



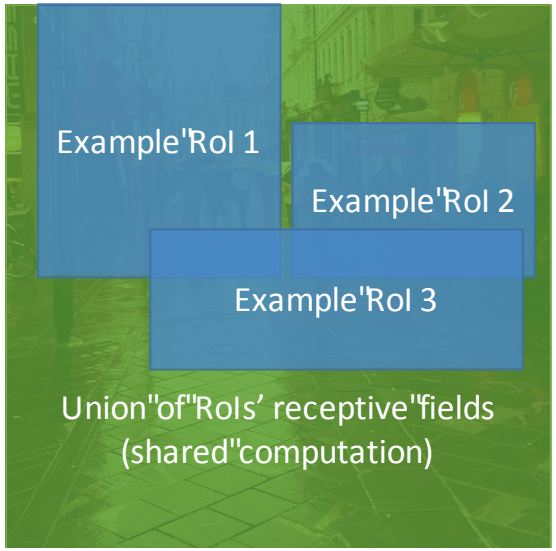
- Sample a small number of images (2)
- Sample many examples from each image (64)

Ross Girshick. "Fast R-CNN". ICCV 2015.

Obstacle'#2:'Making'SGD'steps'efficient

Use'the'test)time'trick'from'SPP)net'during'training

- Share'computation'between'overlapping'examples'from'the'same'image

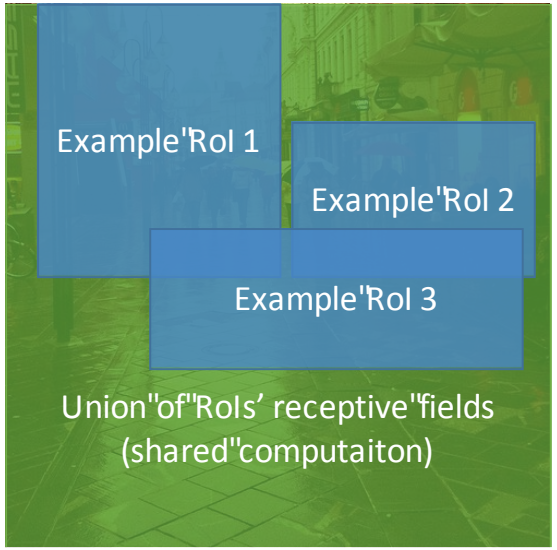


Obstacle #2: Making SGD steps efficient

Cost per mini-batch compared to slow R-CNN (same crude cost model)

input size for Fast R-CNN input size for slow R-CNN

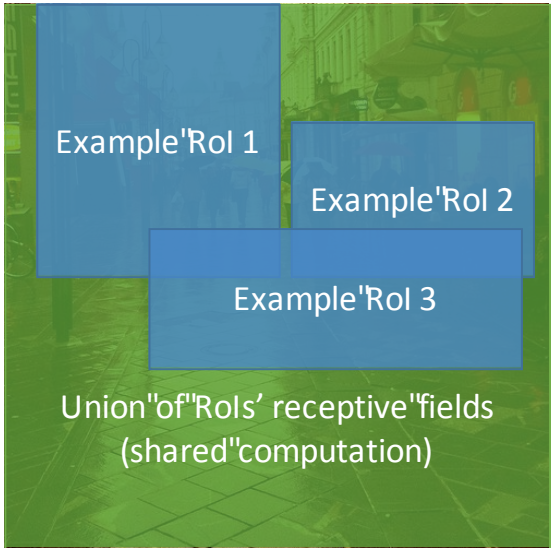
- 2*600*1000 / (128*224*224) = 0.19x < computation than slow R-CNN



Obstacle'#2:'Making'SGD'steps'efficient

Are'the'examples'from'just'2'images'diverse'enough?

- Concern:'examples'from'the'sample'image'may'be'too'correlated



Fast R-CNN outcome

Better training time and testing time with better accuracy than slow R-CNN or SPP-net

- Training time: 84 hours / 25.5 hours / 8.75 hours (Fast R-CNN)
- VOC07 test mAP: 66.0% / 63.1% / 68.1%
- Testing time per image: 47s / 2.3s / 0.32s
  - Plus 0.2 to > 2s per image depending on proposal method
  - With selective search: 49s / 4.3s / 2.32s

Updated numbers from the ICCV paper based on implementation improvements

facebook

Experimental findings

- End-to-end training is important for very deep networks
- Softmax is a fine replacement for SVMs
- Multi-task training is beneficial
- Single-scale testing is a good tradeoff (noted by Kaiming)
- Fast training and testing enables new experiments
  - Comparing proposals

Result compare

	Fast R-CNN	R-CNN
Train time (h)	9.5	84
-speedup	8.8x	1x
Test time/image	0.32s	47.00 s
-test speedup	146x	1x
mAP	66.9	66

Trained using VGG 16 on Pascal VOC 2007 dataset
Not including proposal time

Source: R. Girshick

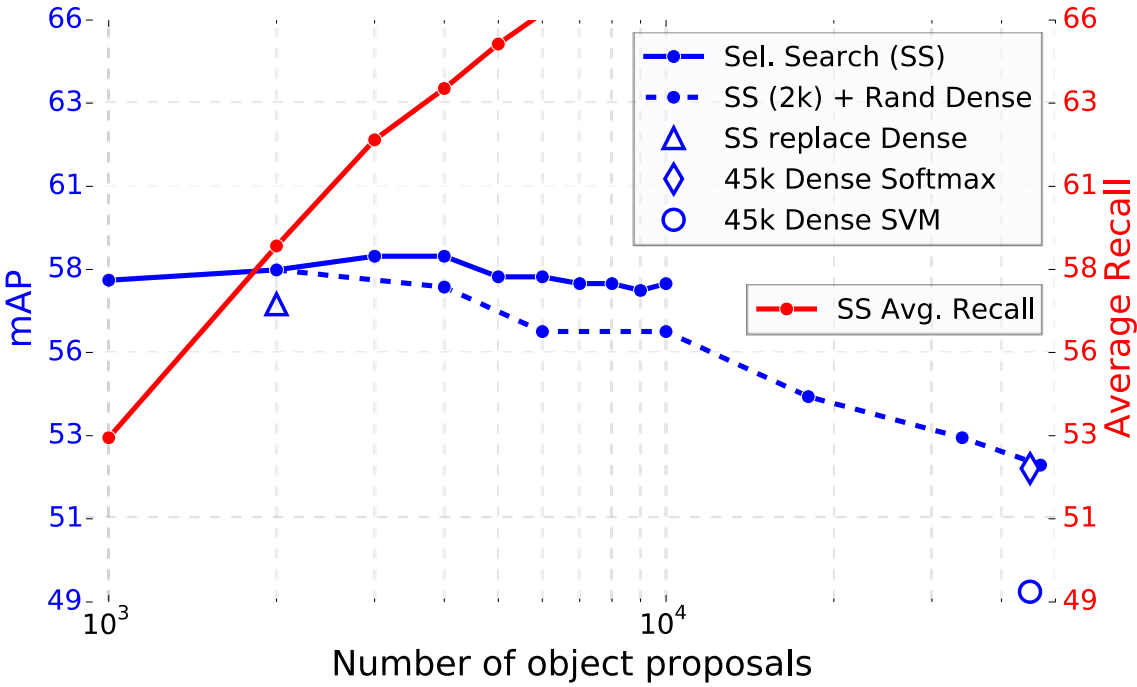
Result compare

	Fast R-CNN	R-CNN
Test time/image	0.32s	47.00 s
-test speedup	146x	1x
Test time/image with proposal	2s	50 s
-test speedup	25x	1x

Source: cs231 standford

Direct region proposal evaluation

- VGG_CNN_M_1024
- Training takes <2 hours
- Fast training makes these experiments possible



Ross Girshick. "Fast R-CNN". ICCV 2015.

Drawback of R-CNN and the modification:

1. Training is a multi-stage pipeline. -> End-to-end joint training.
2. Training is expensive in space and time. -> Convolutional layer sharing. Classification in memory.

For SVM and regressor training, features are extracted from each warped object proposal in each image and written to disk.(VGG16, 5k VOC07 trainval images : 2.5 GPU days). Hundreds of gigabytes of storage.

3. Test-time detection is slow. -> Single scale testing, SVD fc layer.

At test-time, features are extracted from each warped proposal in each img. (VGG16: 47s / image).

Contributions:

1. Higher detection quality (mAP) than R-CNN
2. Training is single-stage, using a multi-task loss
3. All network layers can be updated during training
4. No disk storage is required for feature caching

•**Mini-batch Sampling**

128: 2 randomly sampled images with 64 PoI sampled from each image

25% positive: IoU > 0.5

75% background:IoU in [0.1, 0.5)

horizontally flipped with prob = 0.5

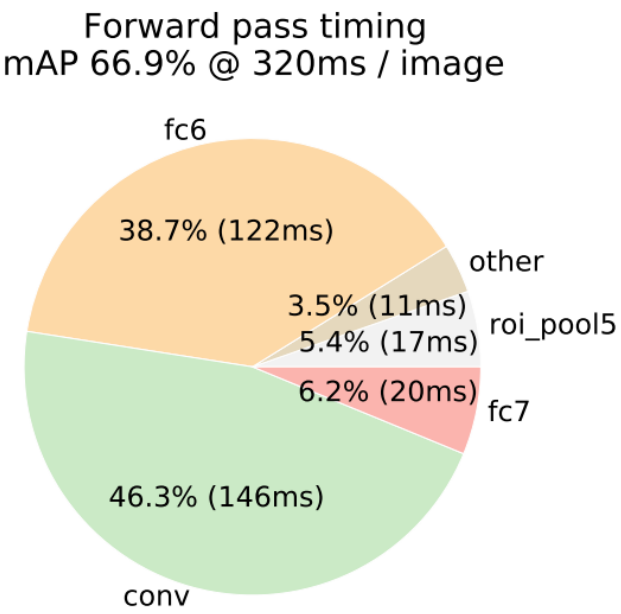
Truncated SVD for faster detection

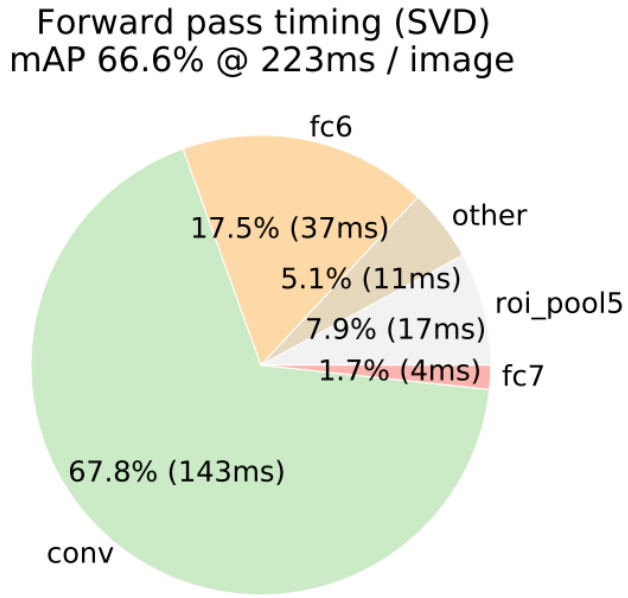
mAP ~ 0.3% down; speed ~ 30% up

number of RoI for detection is large -> time spent on fc

$W \sim U\Sigma_t V^T$ (U : u*t, Sigma_t: t*t, V: v*t)

Compression : $(Wx + b)$ fc -> $(\Sigma_t V^T x)$ fc + $(Ux + b)$ fc





Faster RCNN

Don’t need to have external regional proposals

RPN - Regional Proposal Network

Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks

Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun

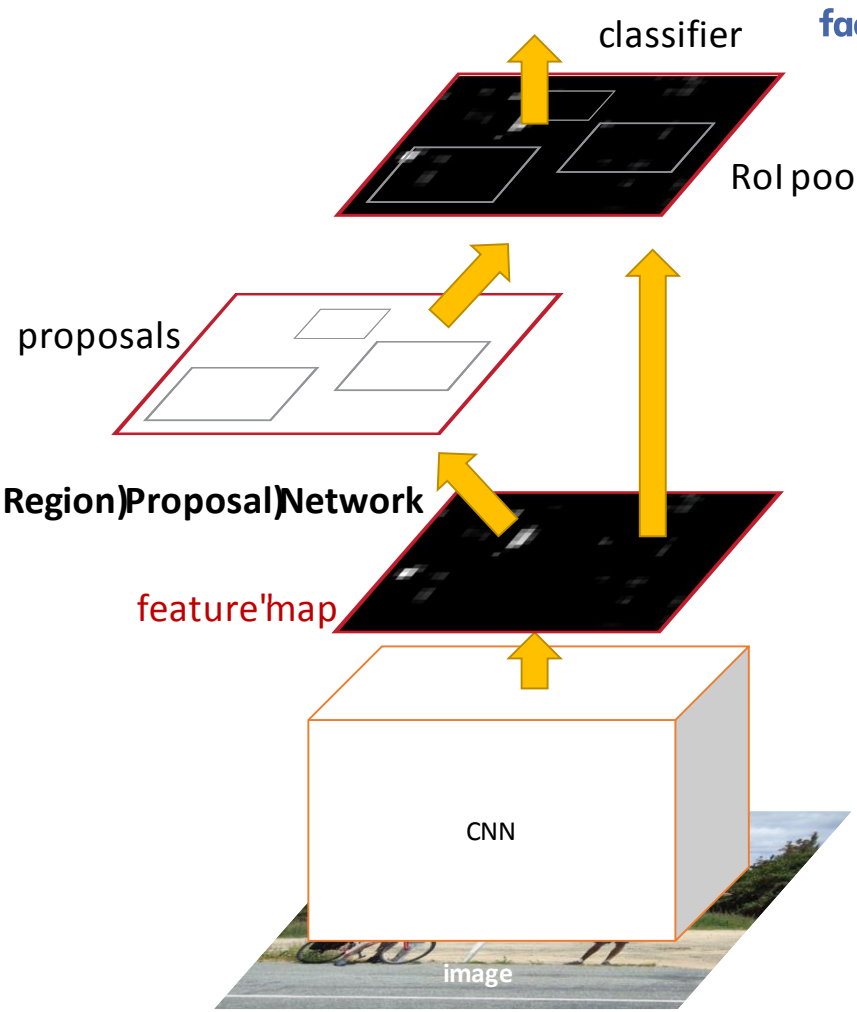
Jun 2015

https://arxiv.org/pdf/1506.01497v3.pdf

What is Faster R-CNN?

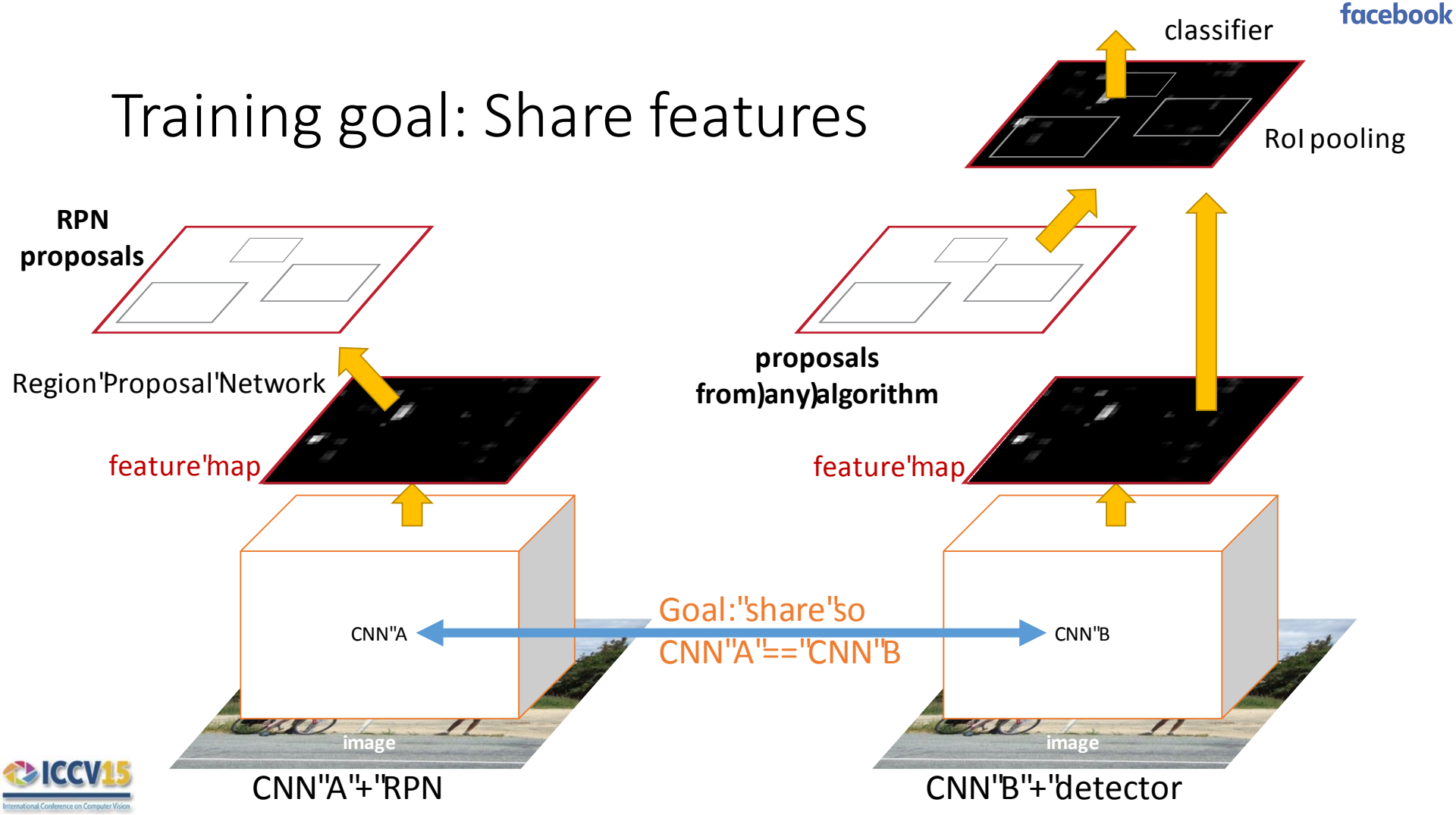
- Presented in Kaiming's section

- Review:
  Faster R-CNN = Fast R-CNN + Region Proposal Networks
  - Does *not* depend on an external region proposal algorithm
  - Does object detection in a single forward pass

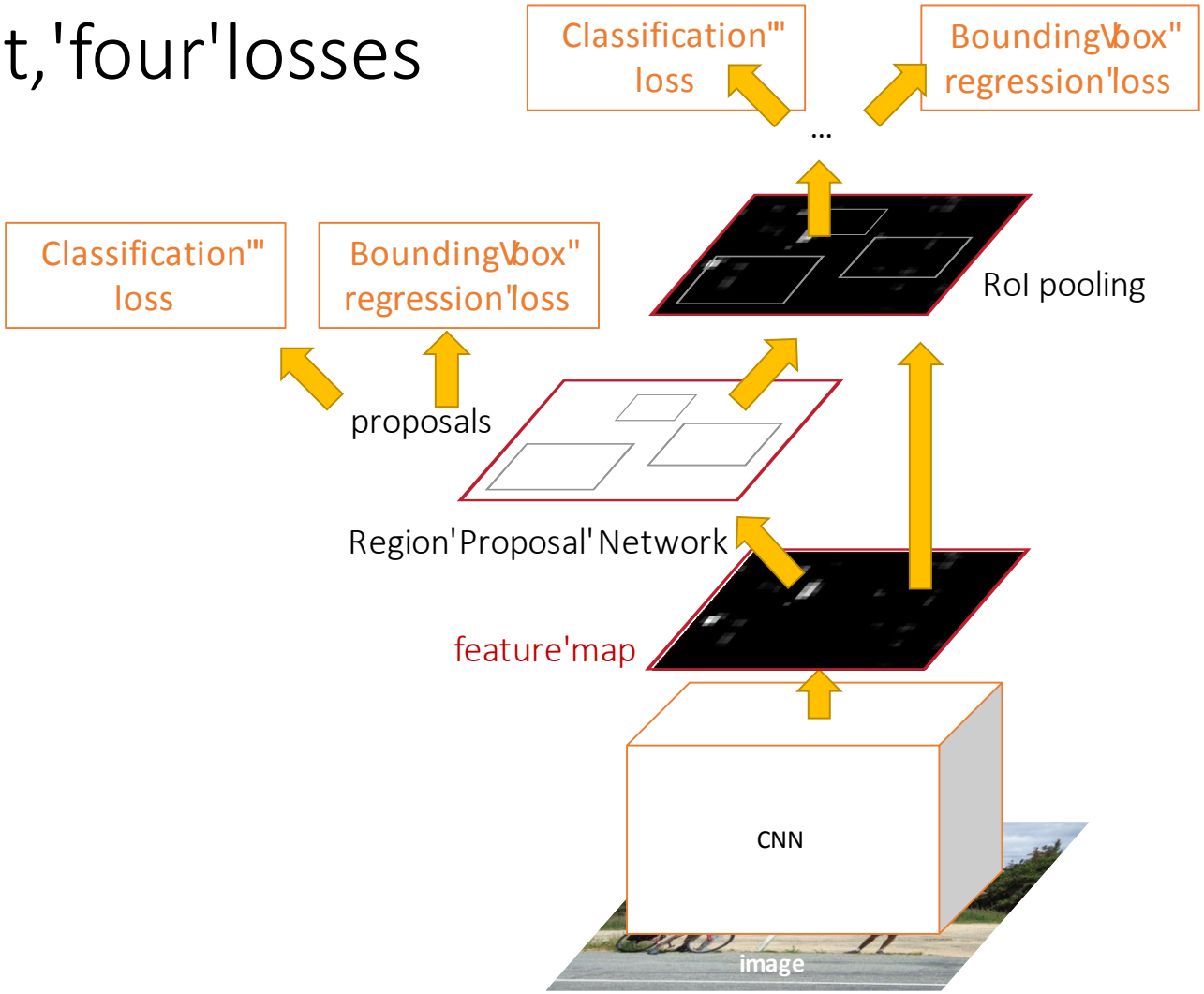


ICCV15
International Conference on Computer Vision

Training goal: Share features



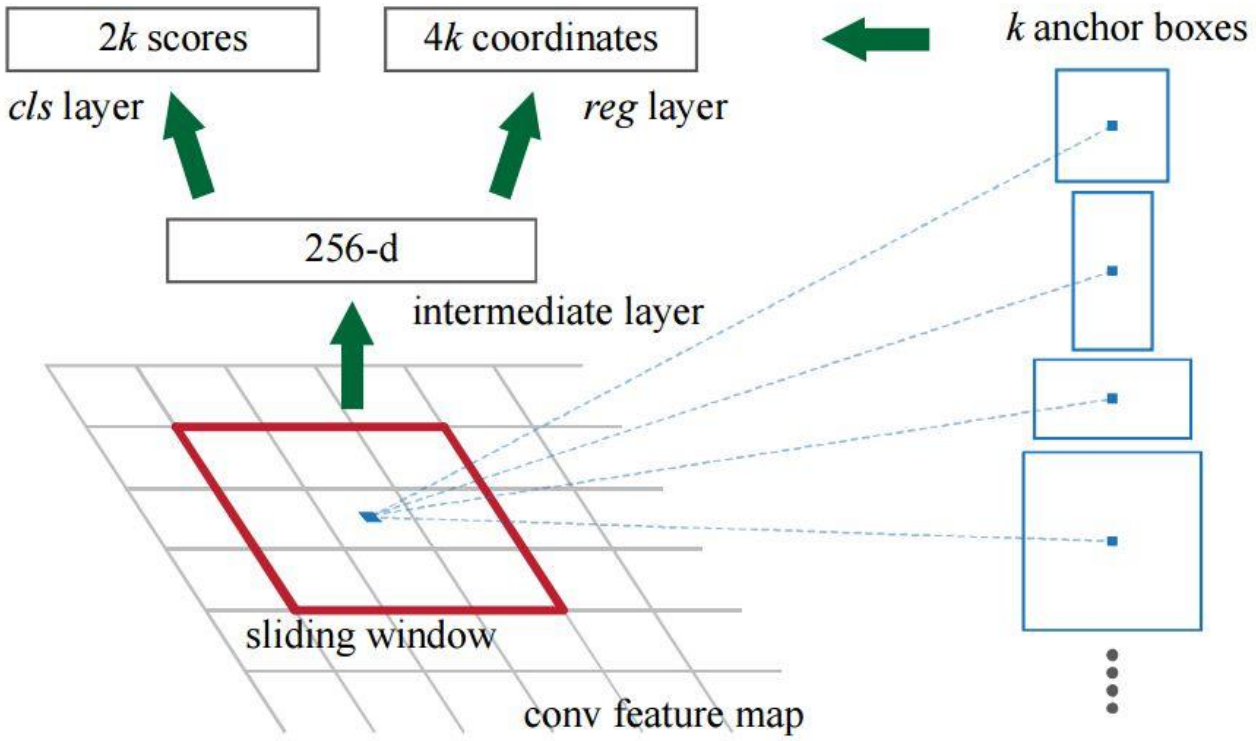
One net, four losses



facebook

ICCV15
International Conference on Computer Vision

Faster RCNN



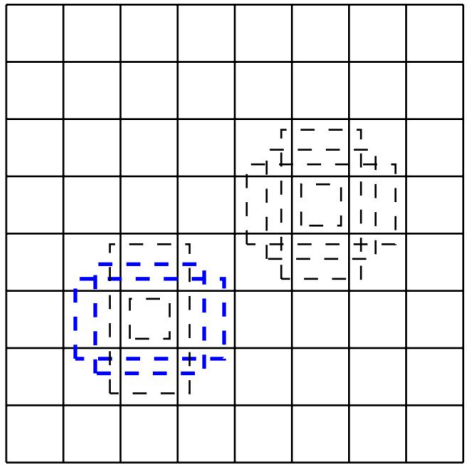
https://arxiv.org/pdf/1506.01497v3.pdf

- **Translation-Invariant Anchors**

  At each sliding window loc, pridict k proposal: 4k outputs for reg layer, 2k outputs for cls layer (binary softmax).

  Anchor: centered at sliding window with scale and aspect ratio: (128^2, 256^2, 512^2; 1:2, 2:1, 1:1)

  For a conv feature map: $W * H * k$ (k=9 anchors) (2+4)*9 output layer



- Optimization

  fcn trained by end-to-end by bp and sgd

  image-centric sampling strategy, sample 256 anchors in an image(Pos:neg = 1:1)

  new layer initialization ~ N(0, 0.01)

  tune ZFnet and conv3_1 and up for VGGnet, lr=0.001 for 60k batches, 0.0001 for 20k on PASCAL

- **Loss function for Learning Region Proposal**
  positive label: the anchor has highest IoU with a gt-box or has an IoU>0.7 with any gt-box
  negative label: IoU<0.3 for all gt-box
  Objective function with multi-task loss: Similar to Fast R-CNN.

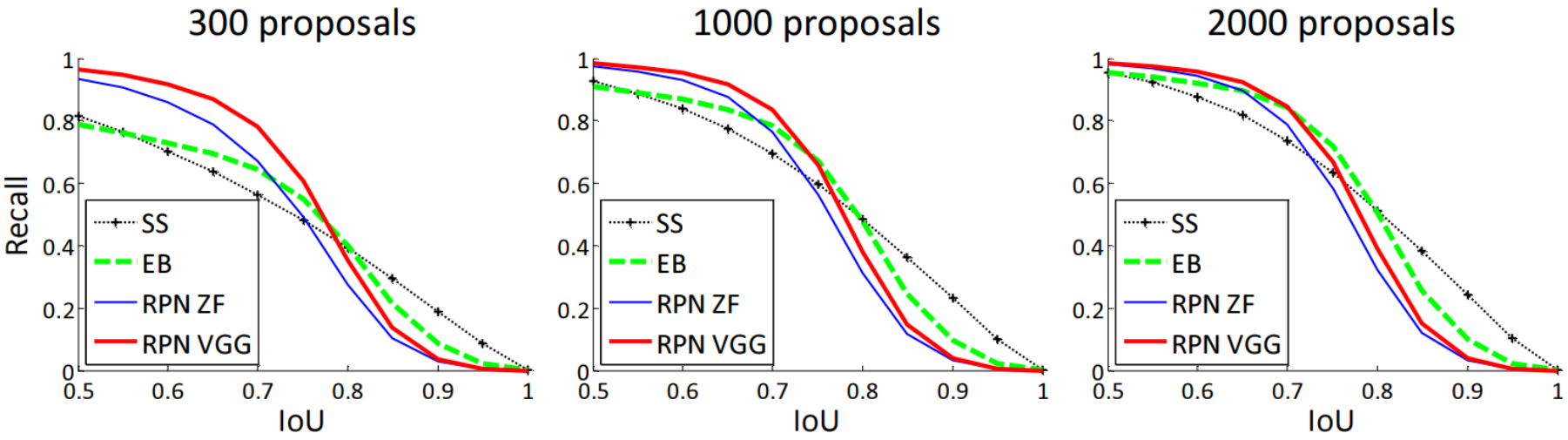
$$L(p_i, t_i) = L_{cls(p_i, p_i^*)} + \lambda p_i^* L_{reg}(t_i, t_i^*)$$

  where p_i^* is 1 if the anchor is labeled positive, and is 0 if the anchor is negative.
  $\lambda = 10$ bias towards better box location

- **Share Convolutional Features for Region Proposal and Objection Detection**

  Four-step training algorithm:

  1. Train RPN, initialized with ImageNet pre-trained model
  2. Train a separate detection network by Fast R-CNN using proposals generated by step-1 RPN, initialized by ImageNet pre-trained model
  3. Fix conv layer, fine-tune unique layers to RPN, initialized by detector network in Step2
  4. Fix conv layer, fine-tune fc-layers of Fast R-CNN



Result compare

	Faster R-CNN	Fast R-CNN	R-CNN
Test time/image With proposal	0.2S	2s	50s
-test speedup	250x	25x	1x
mAP	66.9	66.9	66

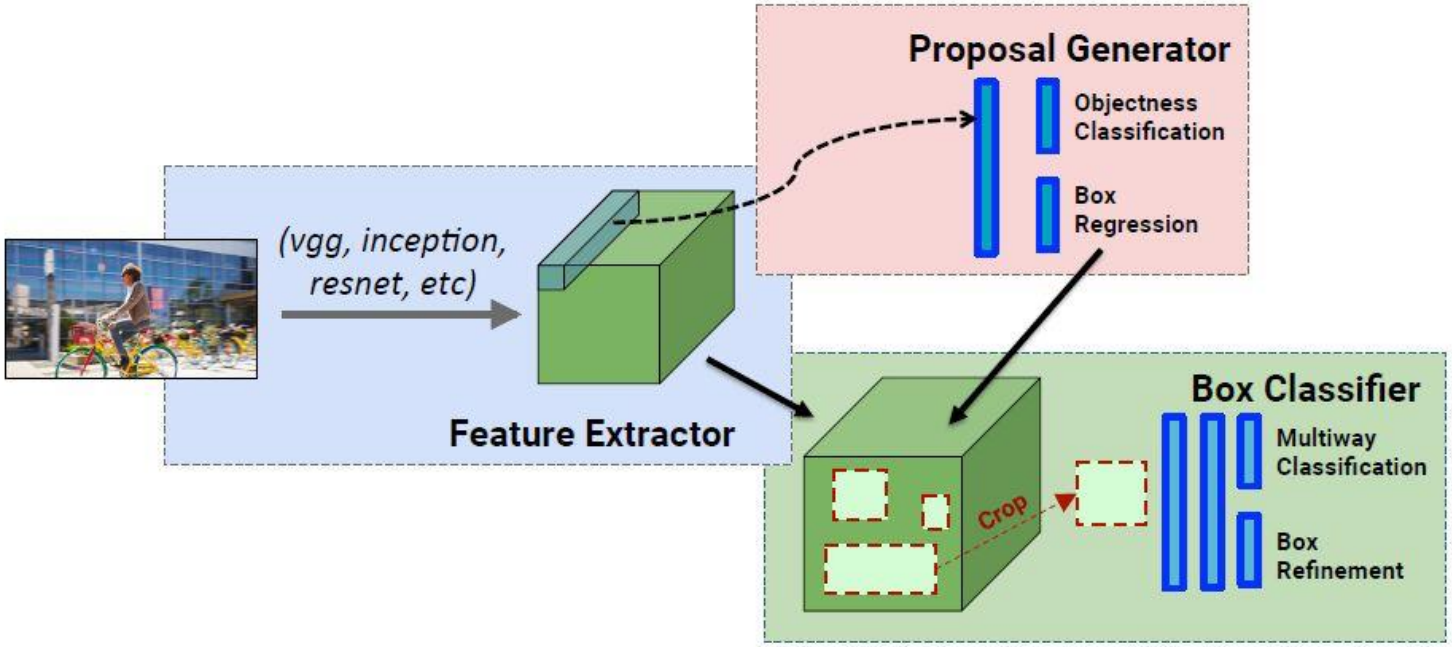
Trained using Pascal VOC 2007 dataset

Source: cs231 standford

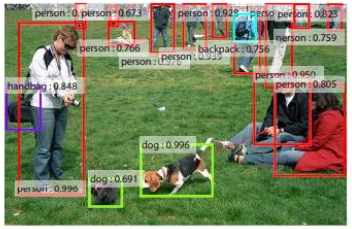
Table 5: **Timing** (ms) on a K40 GPU, except SS proposal is evaluated in a CPU. "Region-wise" includes NMS, pooling, fully-connected, and softmax layers. See our released code for the profiling of running time.

model	system	conv	proposal	region-wise	total	rate
VGG	SS + Fast R-CNN	146	1510	174	1830	0.5 fps
VGG	RPN + Fast R-CNN	141	**10**	47	**198**	**5 fps**
ZF	RPN + Fast R-CNN	31	**3**	25	**59**	**17 fps**

Recap: Faster RCNN



https://pan.baidu.com/s/1pKIKIIB


person : 0.673
person : 0.925
person : 0.823
person : 0.766
backpack : 0.756
person : 0.759
handbag : 0.848
person : 0.805
dog : 0.996
dog : 0.691
person : 0.996


airplane : 0.997
person : 0.867

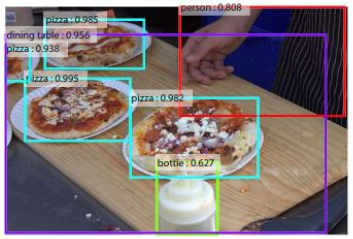

traffic light : 0.802
person : 0.772
umbrella : 0.824
person : 0.897
person : 0.841
car : 0.957
person : 0.931
person : 0.970
motorcycle : 0.713
bicycle : 0.891
bicycle : 0.639
motorcycle : 0.827

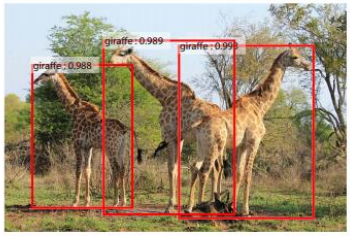

clock : 0.986
clock : 0.981
person : 0.800

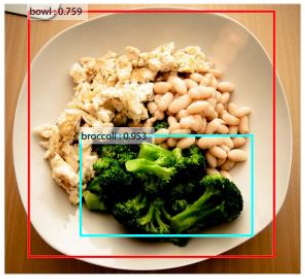

person : 0.998
skis : 0.919

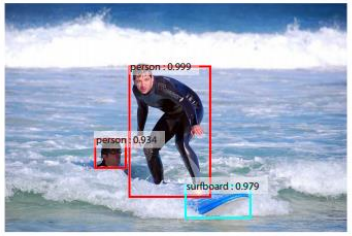

clock : 0.982

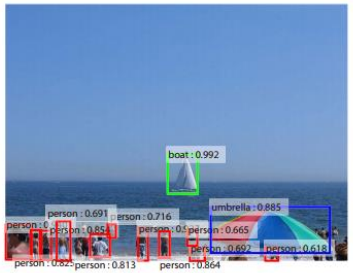

bed : 0.999


person : 0.808
pizza : 0.985
dining table : 0.956
pizza : 0.938
pizza : 0.995
pizza : 0.982
bottle : 0.627


giraffe : 0.989
giraffe : 0.988


bowl : 0.759
broccoli : 0.953


person : 0.999
person : 0.934
surfboard : 0.979

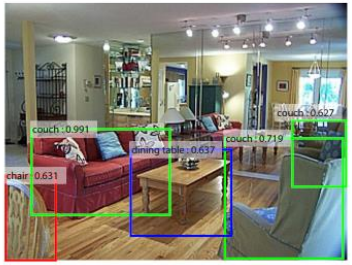

boat : 0.992
person : 0.691
umbrella : 0.885
person : 0.665
person : 0.813
person : 0.864

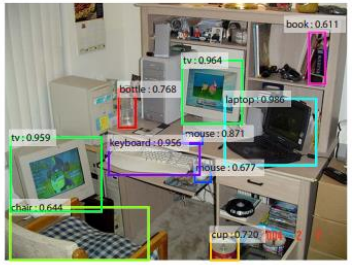

bus : 0.999
person : 0.869

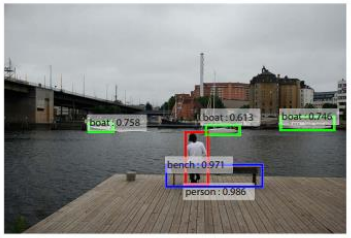

teddy bear : 0.999
teddy bear : 0.738
teddy bear : 0.890
teddy bear : 0.802


potted plant : 0.769
bowl : 0.602
sink : 0.969
toilet : 0.921

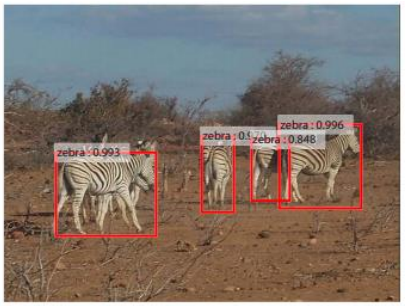

sink : 0.994
sink : 0.938
sink : 0.992

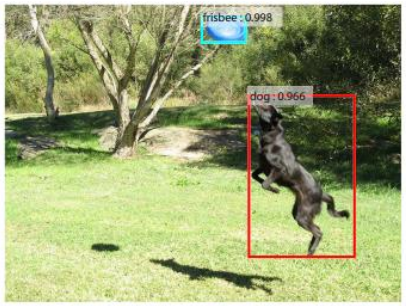

couch : 0.991
couch : 0.627
couch : 0.719
dining table : 0.637
chair : 0.631

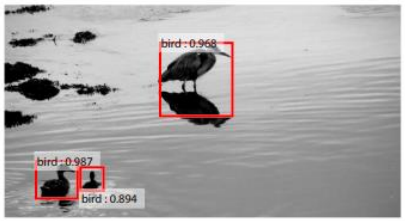

book : 0.611
tv : 0.964
bottle : 0.768
laptop : 0.986
tv : 0.959
keyboard : 0.956
mouse : 0.871
mouse : 0.677
chair : 0.644
cup : 0.720

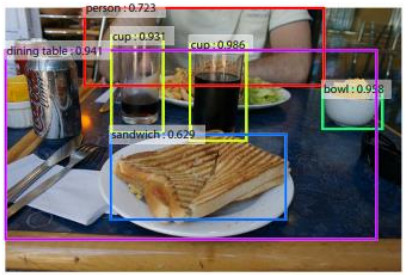

boat : 0.758
boat : 0.613
boat : 0.746
bench : 0.971
person : 0.986

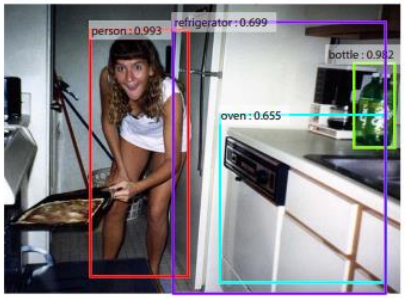

traffic light : 0.869
traffic light : 0.713
train : 0.965

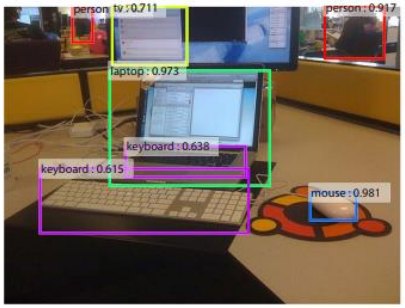

zebra : 0.993
zebra : 0.996
zebra : 0.848

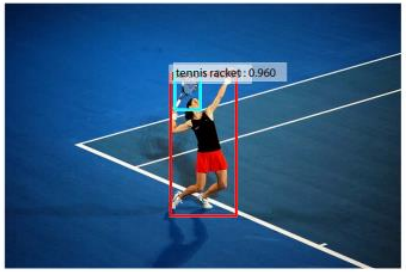

frisbee : 0.998
dog : 0.966

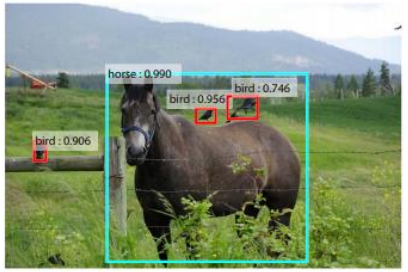

bird : 0.968
bird : 0.987
bird : 0.894

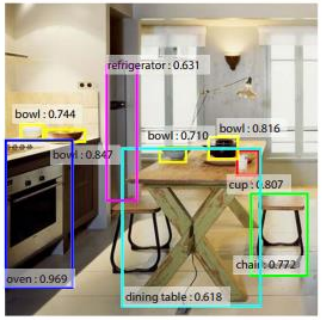

person : 0.723
cup : 0.931
dining table : 0.941
cup : 0.986
bowl : 0.958
sandwich : 0.629


person : 0.993
refrigerator : 0.699
bottle : 0.982
oven : 0.655


person
tv : 0.711
person : 0.917
laptop : 0.973
keyboard : 0.638
keyboard : 0.615
mouse : 0.981


tennis racket : 0.960

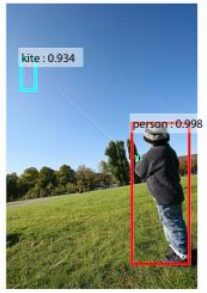

horse : 0.990
bird : 0.956
bird : 0.746
bird : 0.906

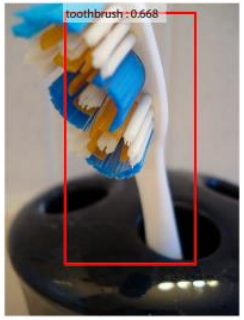

refrigerator : 0.631
bowl : 0.744
bowl : 0.710
bowl : 0.816
bowl : 0.847
cup : 0.807
chair : 0.772
oven : 0.969
dining table : 0.618

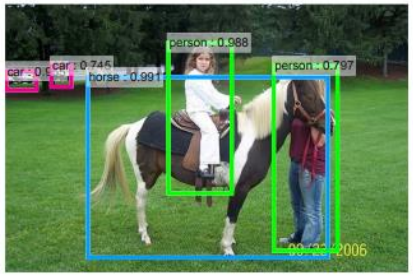

dining table : 0.888
cup : 0.990
pizza : 0.919

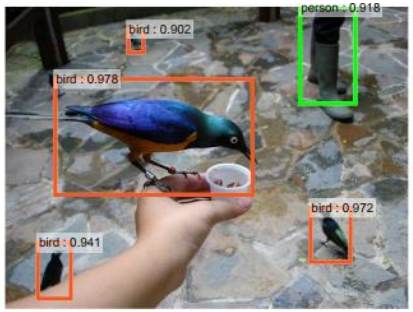

car : 0.816
pizza : 0.965


clock : 0.988

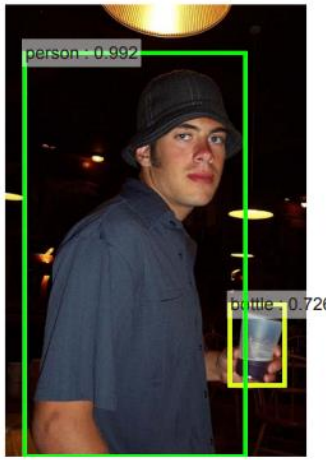

kite : 0.934
person : 0.998

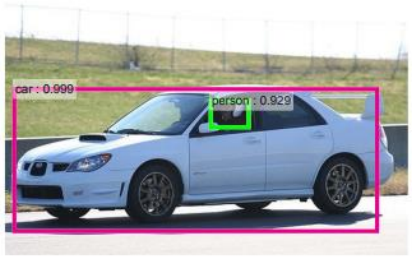

toothbrush : 0.668

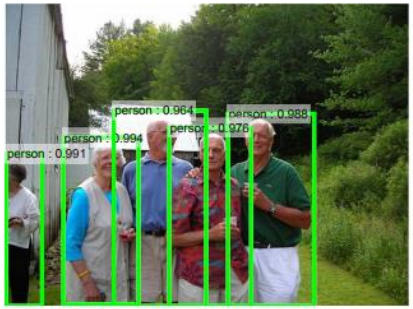

car : 0.745
person : 0.988
horse : 0.991
person : 0.797

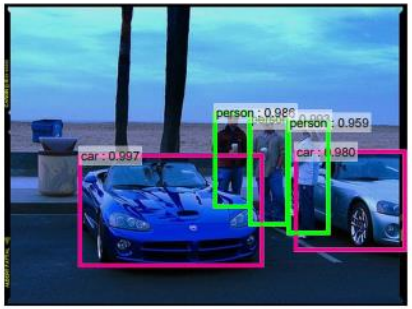

bird : 0.902
person : 0.918
bird : 0.978
bird : 0.972
bird : 0.941


cow : 0.995
cow : 0.998

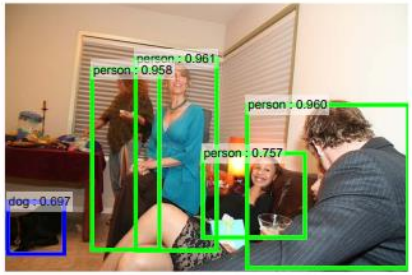

person : 0.992
bottle : 0.726

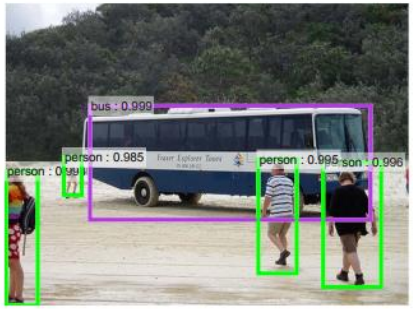

car : 0.999
person : 0.929

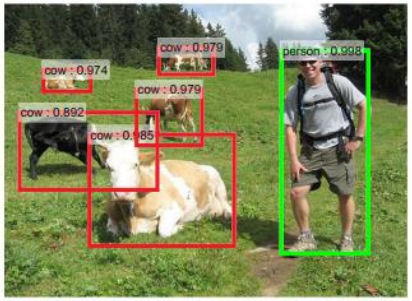

person : 0.964
person : 0.988
person : 0.994
person : 0.976
person : 0.991


person : 0.986
person : 0.959
car : 0.997

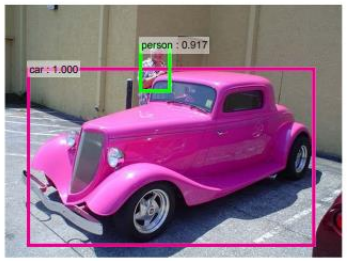

dog : 0.981

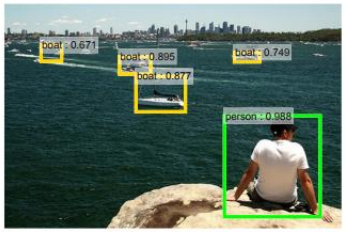

person : 0.961
person : 0.958
person : 0.960
person : 0.757
dog : 0.697

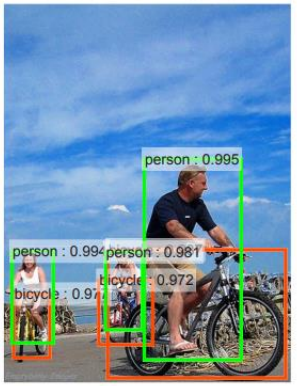

bus : 0.999
person : 0.985
person : 0.995
son : 0.996

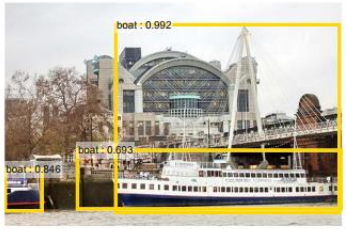

cow : 0.979
cow : 0.974
person : 0.998
cow : 0.979
cow : 0.892
cow : 0.985

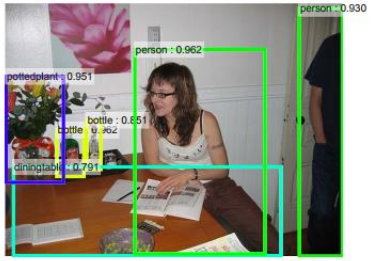

cat : 0.998


person : 0.917
car : 1.000


boat : 0.671
boat : 0.895
boat : 0.749
boat : 0.877
person : 0.988

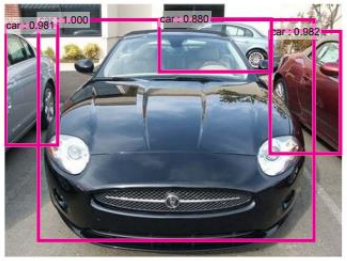

person : 0.995
person : 0.994
person : 0.981
bicycle : 0.972

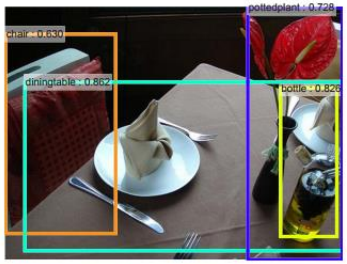

boat : 0.992
boat : 0.693
boat : 0.846

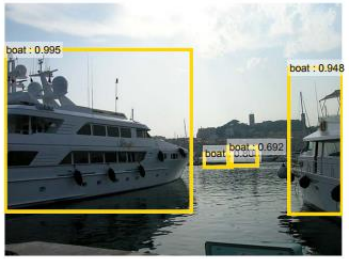

person : 0.962
pottedplant : 0.951
bottle : 0.851
diningtable : 0.791


dog : 0.967

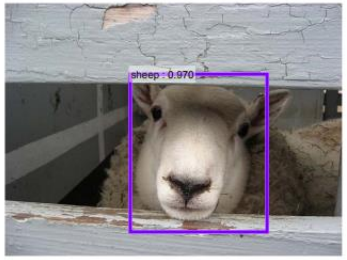

person : 0.948
person : 0.972
person : 0.919
person : 0.975

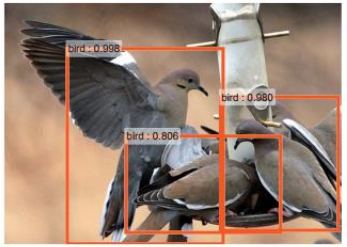

car : 0.880
car : 0.982

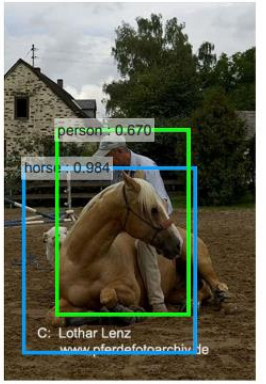

pottedplant : 0.728
diningtable : 0.862

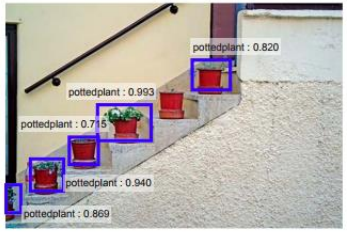

boat : 0.995
boat : 0.948


aeroplane : 0.992
aeroplane : 0.986

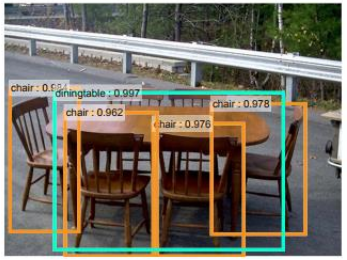

sheep : 0.970


bird : 0.998
bird : 0.980
bird : 0.806


person : 0.670
C: Lothar Lenz


pottedplant : 0.820
pottedplant : 0.993
pottedplant : 0.715
pottedplant : 0.940
pottedplant : 0.869


aeroplane : 0.998
person : 0.993
person : 0.987


diningtable : 0.997
chair : 0.962
chair : 0.978
chair : 0.976

- Recognition = Description + Discrimination. We are doing well on Discrimination

- Challenges of Deformation and Occlusion. This is also known as hard negative mining and non-maximum suppression. The past solutions assume the features are hard coded. CNN solutions extract 'deformable' features. Occlusion is dealt with per pixel labels over many object classes.

- R-CNN, Fast R-CNN, Faster R-CNN: sparse sampling of `where' jointly with 'what'. Using one shared feature extraction front-end, and multiple regression objectives at the FC layers. Enough information from the front-end CNN features allows both detection (of location) and classification (of classes).

- Memory allocation and flow is important for CNN learning and speed up.