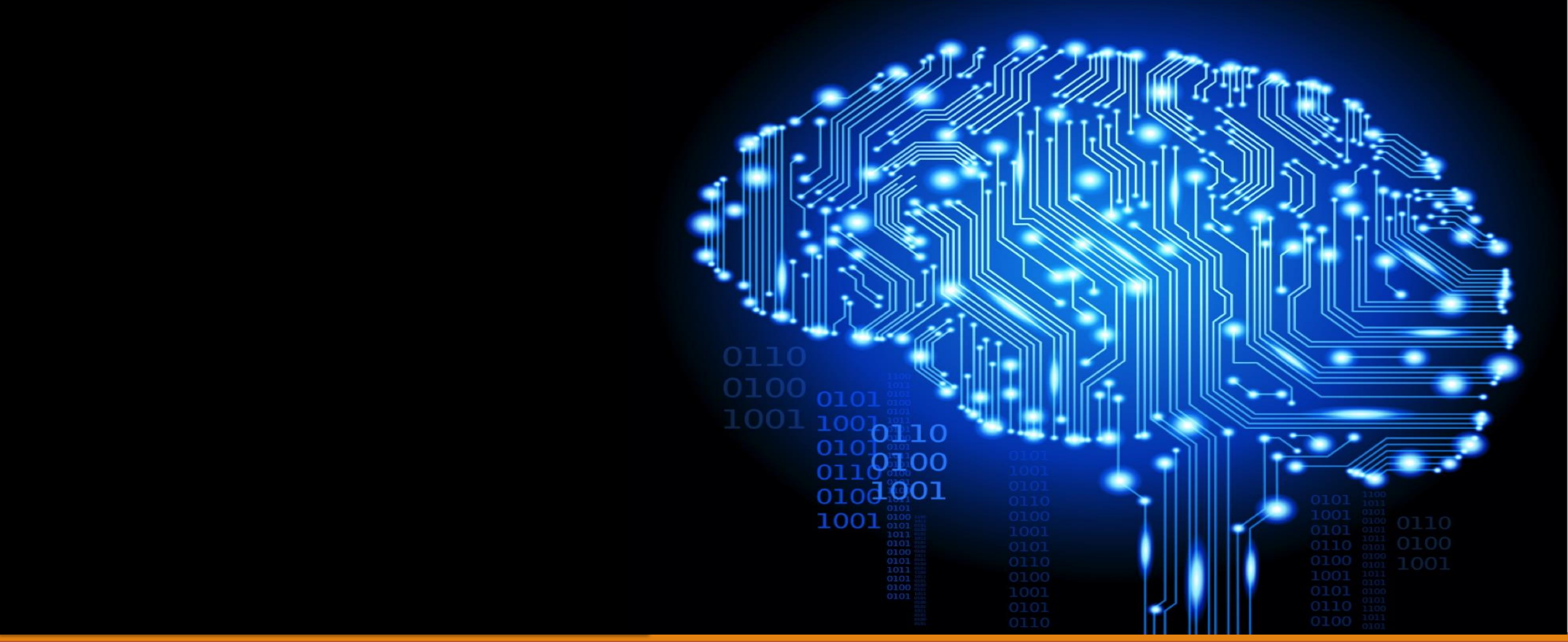


Lecture 6: Convnets for object detection and segmentation

Deep Learning @ UvA

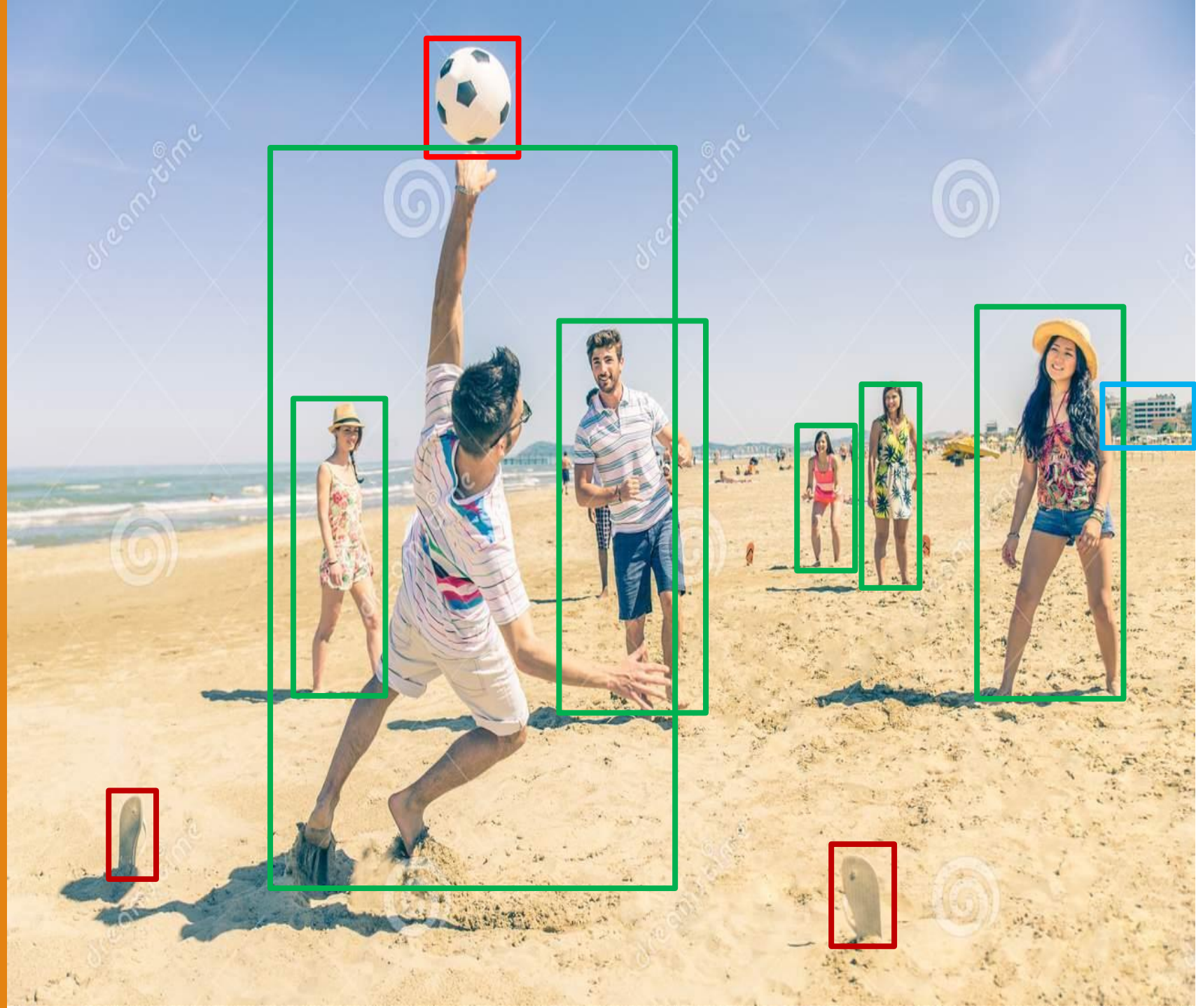
Previous Lecture

- What do convolutions look like?
- Build on the visual intuition behind Convnets
- Deep Learning Feature maps
- Transfer Learning

Lecture Overview

- What is object detection, segmentation and structured output
- How to use a convnet to localize objects?
- What is the relation convnets and Conditional Random Fields (CRF)

Object localization



Object localization

- The task of discovering the location of one or more object in an image
- Images
  - Detect spatial object location
- Videos
  - Detect spatial object location
  - Optionally, detect object temporal location
- Generic
  - Define the same model for all types of categories
- Specific
  - Define specialized models for particular categories (“face”, “pedestrian”)

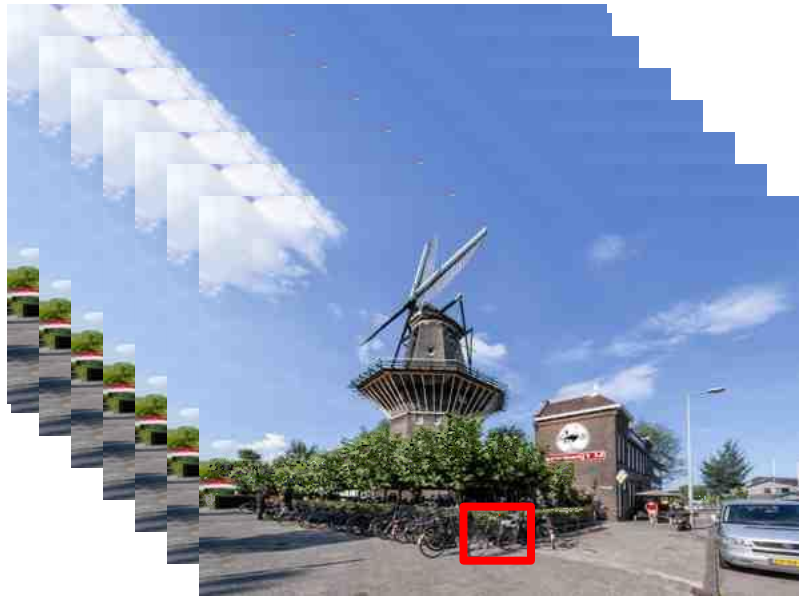
Object localization

- The task of discovering the location of one or more object in an image
- Images
  - Detect spatial object location
- Videos
  - Detect spatial object location
  - Optionally, detect object temporal location

- Generic
  - Define the same model for all types of categories
- Specific
  - Define specialized models for particular categories (“face”, “pedestrian”)

Object localization

- The task of discovering the location of one or more object in an image
- Images
  - Detect spatial object location
- Videos
  - Detect spatial object location
  - Optionally, detect object temporal location
- Generic
  - Define the same model for all types of categories
- Specific
  - Define specialized models for particular categories (“face”, “pedestrian”)

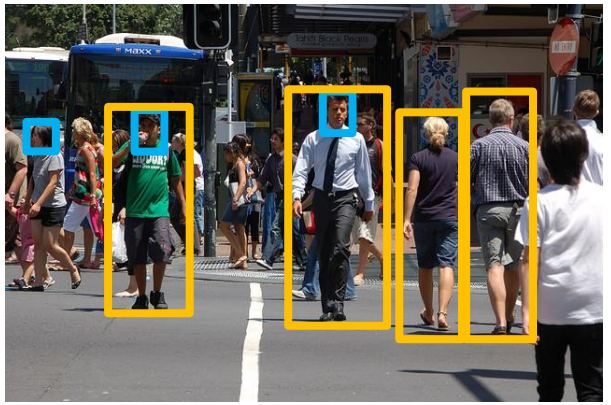


Object localization

- The task of discovering the location of one or more object in an image
- Images
  - Detect spatial object location
- Videos
  - Detect spatial object location
  - Optionally, detect object temporal location
- Generic
  - Define the same model for all types of categories
- Specific
  - Define specialized models for particular categories (“face”, “pedestrian”)

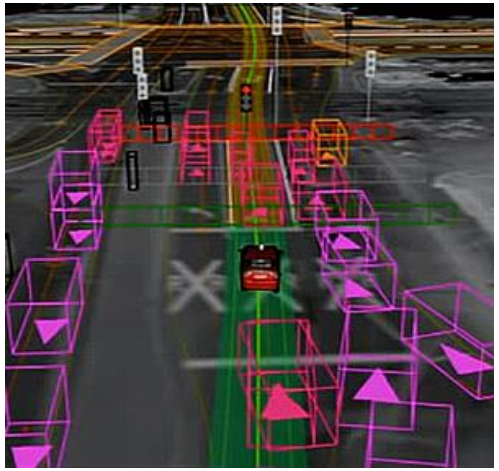
Object localization

- The task of discovering the location of one or more object in an image
- Images
  - Detect spatial object location
- Videos
  - Detect spatial object location
  - Optionally, detect object temporal location
- Generic
  - Define the same model for all types of categories
- Specific
  - Define specialized models for particular categories (“face”, “pedestrian”)



Why is localization important?

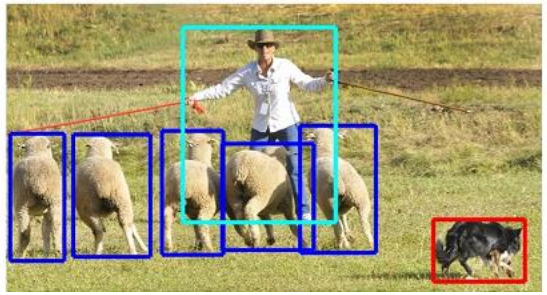
- Robotics
- Self-driving cars
- Better classification
  - Isolates the object signal from the background signal
- Huge pictures
  - E.g. in astronomy pictures have ultra-high resolution
  - Detecting a new star or galaxy goes beyond image classification



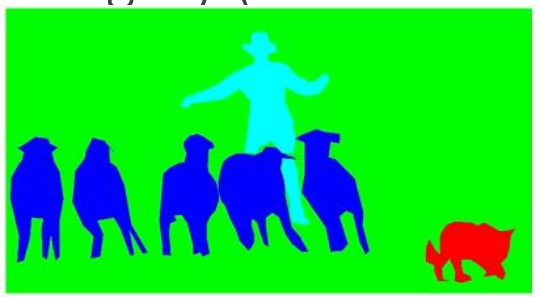
Localization granularity

- Box level, aka object detection
  - Predict a bounding box surrounding the objects
- Pixel level, aka semantic segmentation
  - Predict the category label for each pixel
- Pixel and instance level
  - Predict a free-form delineation around object instances and predict their category
  - Separate between instances of the same category (much more difficult!)

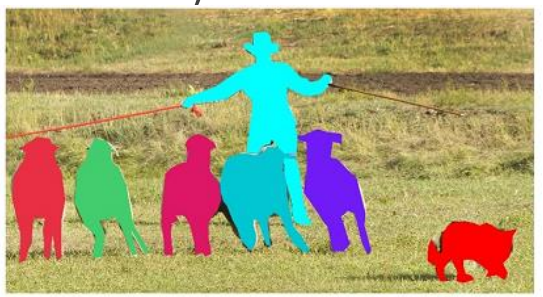
Image classification



Object detection



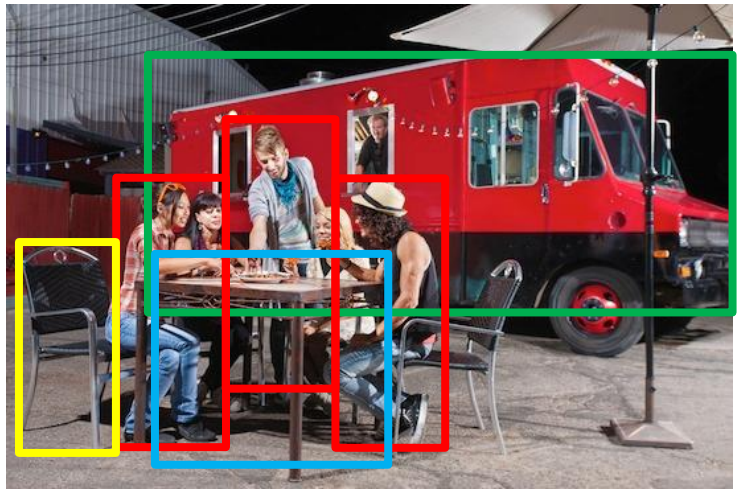
Pixel classification



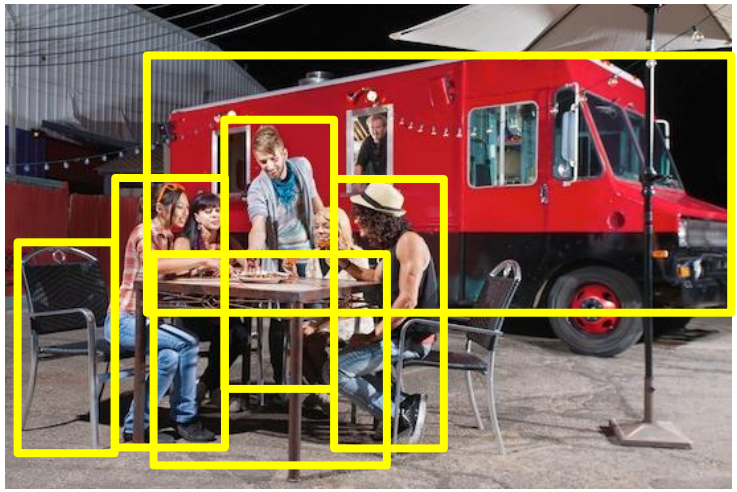
Pixel and instance classification

Localization supervision

- Supervised (category specific)
  - Examples for “bicycle”, “dog”, “person”, “face”
- Unsupervised (category agnostic)
  - aka bounding box/segment proposal algorithm
  - Method typically returns hundreds possible object locations



Supervised (category specific)



Agnostic to category specific detection

- Training
  - Crop objects from the images
  - Train a classifier (SVM) for each of the categories
- Inference
  - Detect possible, category agnostic object locations
  - Crop all of them from the image
  - Assign a per category score for each of them
  - Keep the ones with score more than a threshold
  - Success: $> 50\%$ overlap with ground truth

Agnostic to category specific detection

- Training
  - Crop objects from the images
  - Train a classifier (SVM) for each of the categories
- Inference
  - Detect possible, category agnostic object locations
  - Crop all of them from the image
  - Assign a per category score for each of them
  - Keep the ones with score more than a threshold
  - Success: $> 50\%$ overlap with ground truth

Agnostic to category specific detection

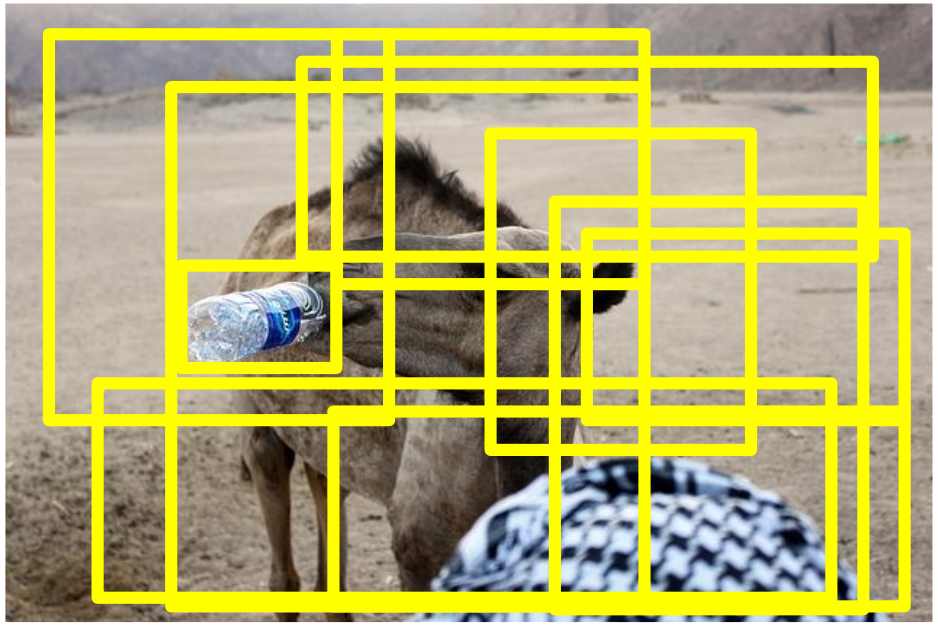
- Training
  - Crop objects from the images
  - Train a classifier (SVM) for each of the categories
- Inference
  - Detect possible, category agnostic object locations
  - Crop all of them from the image
  - Assign a per category score for each of them
  - Keep the ones with score more than a threshold
  - Success: $> 50\%$ overlap with ground truth

Agnostic to category specific detection

- Training
  - Crop objects from the images
  - Train a classifier (SVM) for each of the categories
- Inference
  - Detect possible, category agnostic object locations
  - Crop all of them from the image
  - Assign a per category score for each of them
  - Keep the ones with score more than a threshold
  - Success: $> 50\%$ overlap with ground truth

Agnostic to category specific detection

- Training
  - Crop objects from the images
  - Train a classifier (SVM) for each of the categories
- Inference
  - Detect possible, category agnostic object locations
  - Crop all of them from the image
  - Assign a per category score for each of them
  - Keep the ones with score more than a threshold
  - Success: $> 50\%$ overlap with ground truth



Agnostic to category specific detection

- Training
  - Crop objects from the images
  - Train a classifier (SVM) for each of the categories
- Inference
  - Detect possible, category agnostic object locations
  - Crop all of them from the image
  - Assign a per category score for each of them
  - Keep the ones with score more than a threshold
  - Success: $> 50\%$ overlap with ground truth

Sliding window and direct classification

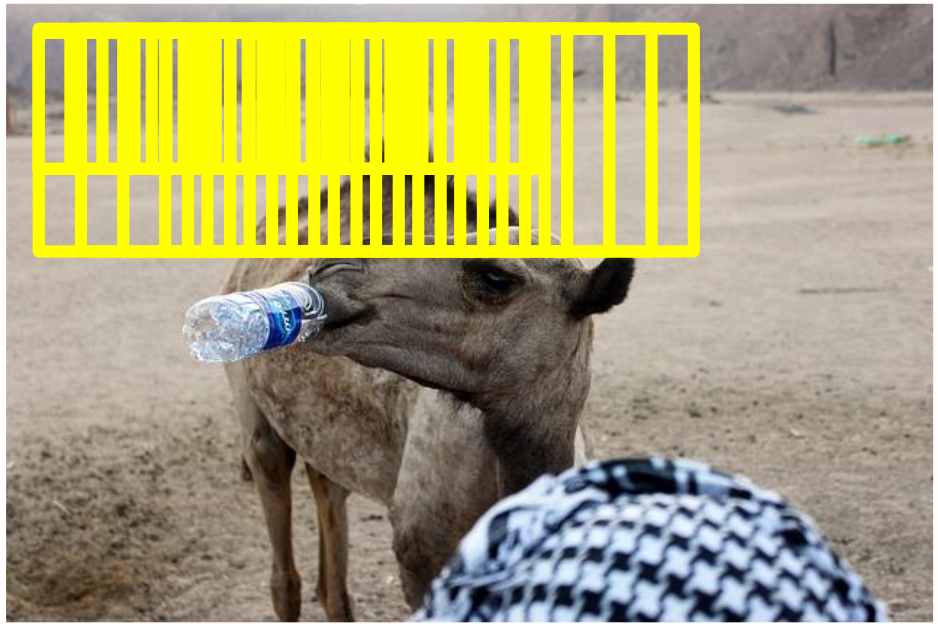
- Training
  - Crop objects from the images
  - Train a classifier for each of the categories
- Inference
  - Parse image at several locations and scales
  - For each location compute category score
  - Keep boxes with high enough score
  - More thorough search
  - Potentially more expensive (or maybe not)

Sliding window and direct classification

- Training
  - Crop objects from the images
  - Train a classifier for each of the categories
- Inference
  - Parse image at several locations and scales
  - For each location compute category score
  - Keep boxes with high enough score
  - More thorough search
  - Potentially more expensive (or maybe not)

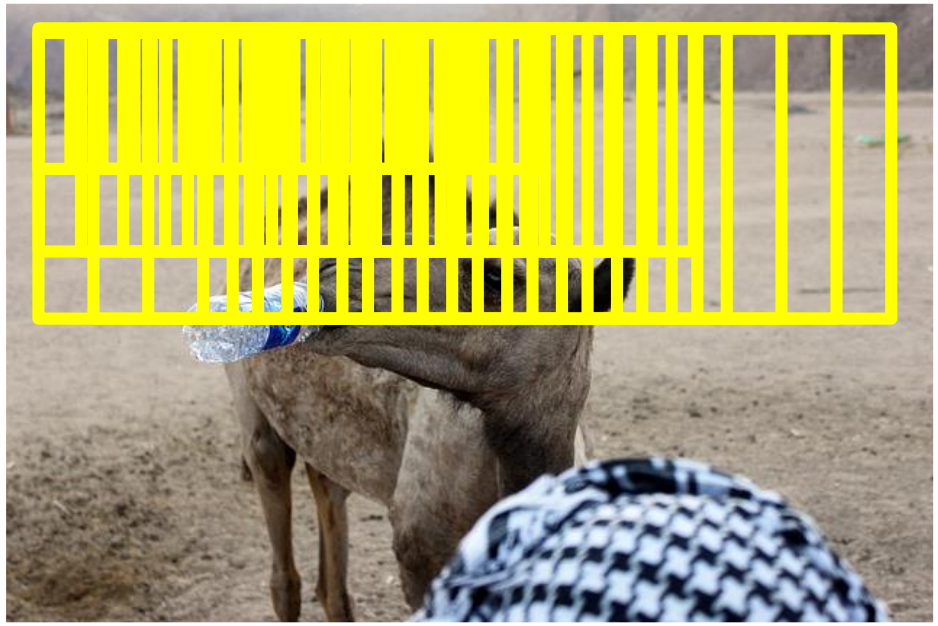
Sliding window and direct classification

- Training
  - Crop objects from the images
  - Train a classifier for each of the categories
- Inference
  - Parse image at several locations and scales
  - For each location compute category score
  - Keep boxes with high enough score
  - More thorough search
  - Potentially more expensive (or maybe not)



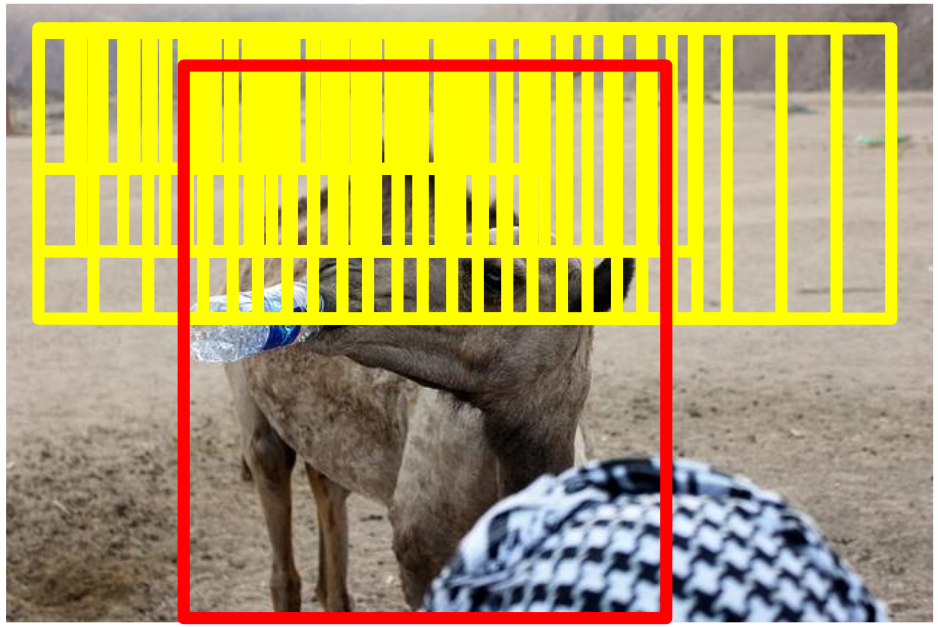
Sliding window and direct classification

- Training
  - Crop objects from the images
  - Train a classifier for each of the categories
- Inference
  - Parse image at several locations and scales
  - For each location compute category score
  - Keep boxes with high enough score
  - More thorough search
  - Potentially more expensive (or maybe not)



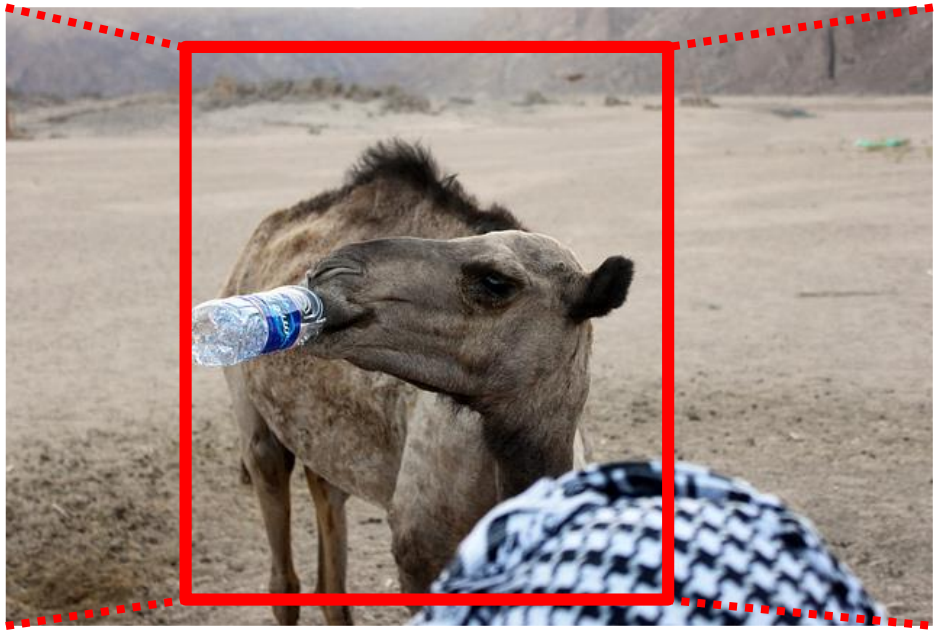
Sliding window and direct classification

- Training
  - Crop objects from the images
  - Train a classifier for each of the categories
- Inference
  - Parse image at several locations and scales
  - For each location compute category score
  - Keep boxes with high enough score
  - More thorough search
  - Potentially more expensive (or maybe not)

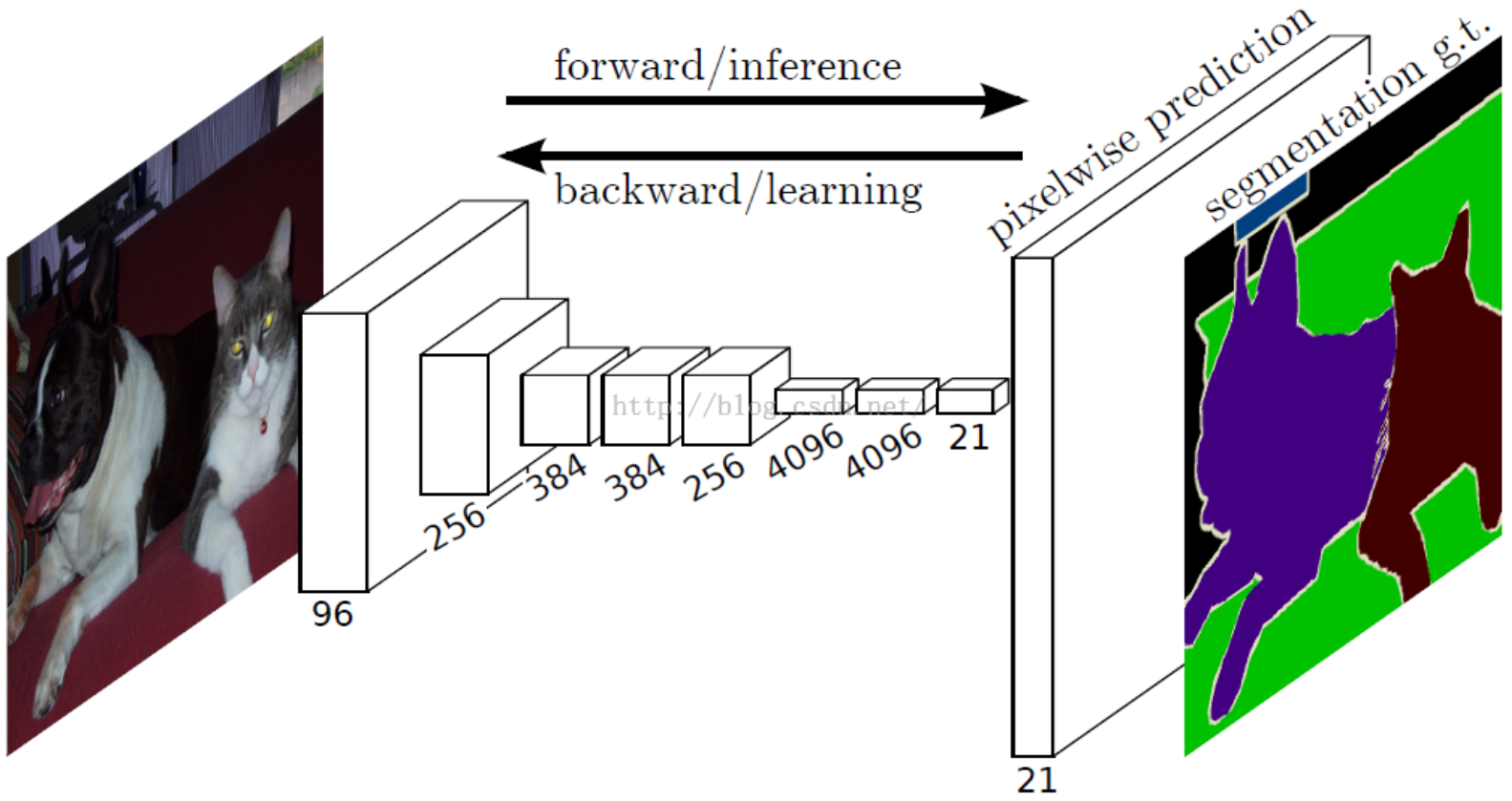


Sliding window and direct classification

- Training
  - Crop objects from the images
  - Train a classifier for each of the categories
- Inference
  - Regress bounding box coordinates
  - Cheap
  - Not accurate enough

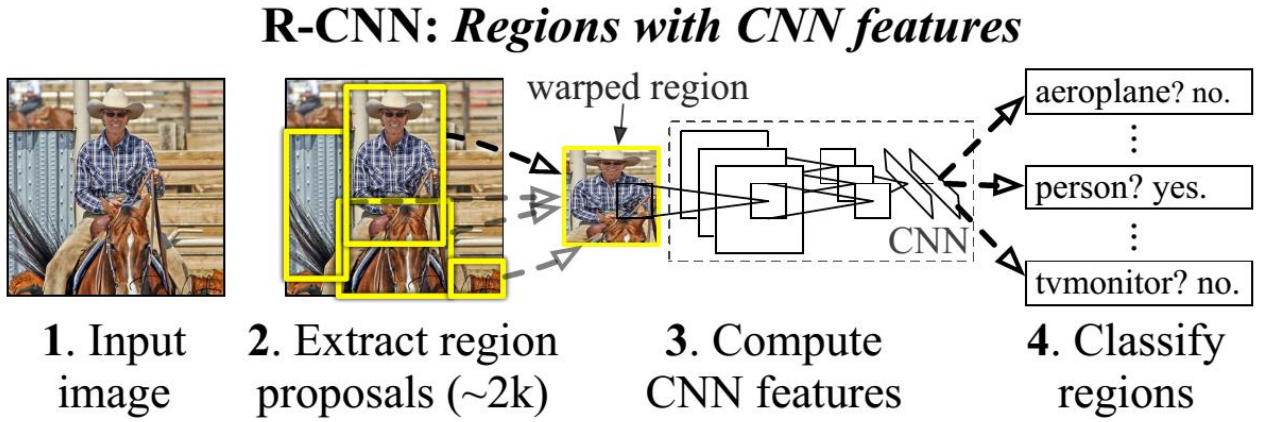


Convnets and object localization



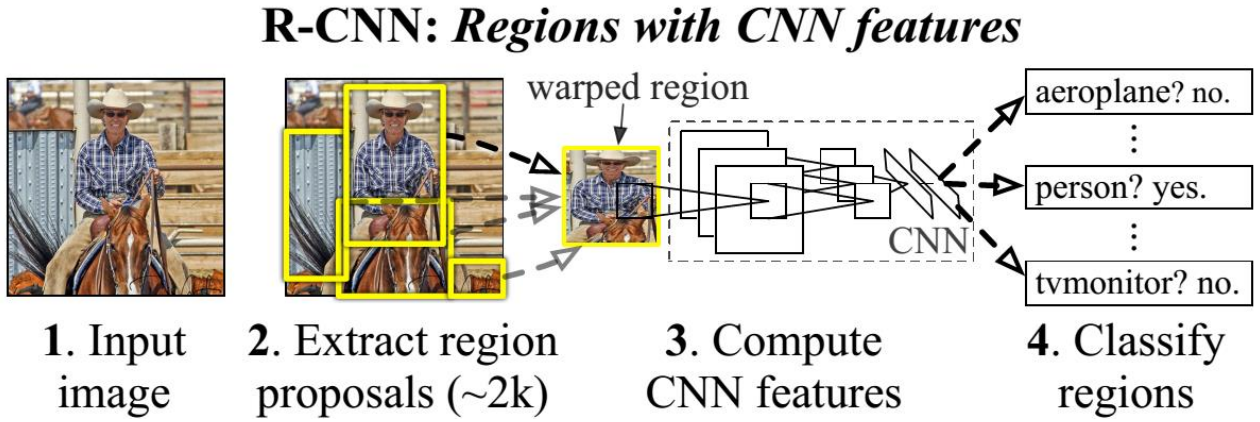
R-CNN [Girshick2013]

- Simple pipeline!
- Use the convnet as a feature extractor
- Given a bounding box, compute a “deep” feature
- Run an SVM (one per category) on the deep features



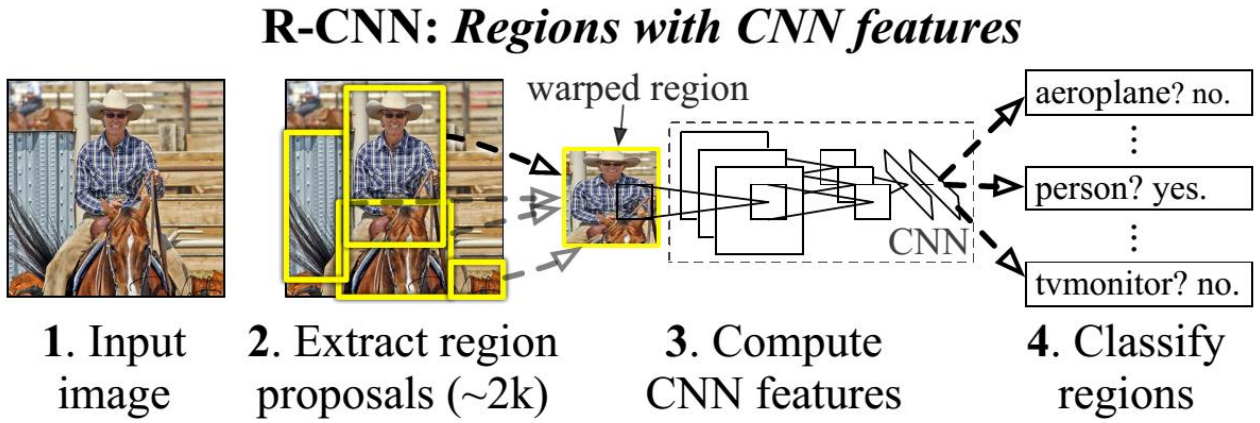
R-CNN [Girshick2013]

- Simple pipeline!
- Use the convnet as a feature extractor
- Given a bounding box, compute a “deep” feature
- Run an SVM (one per category) on the deep features
- Warp cropped images to make them “square”



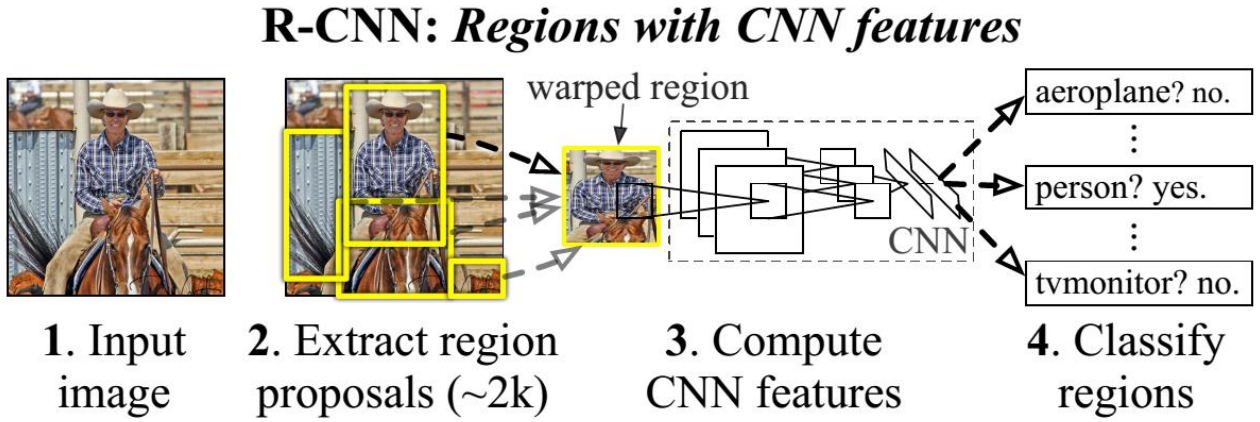
R-CNN [Girshick2013]

- Simple pipeline!
- Use the convnet as a feature extractor
- Given a bounding box, compute a “deep” feature
- Run an SVM (one per category) on the deep features
- Warp cropped images to make them “square”
- Add some background context



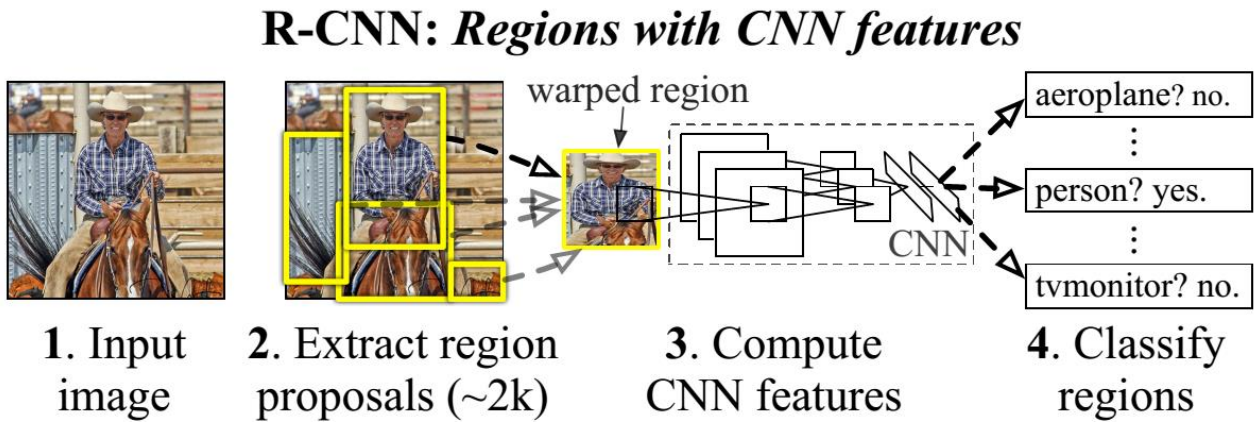
R-CNN [Girshick2013]

- Simple pipeline!
- Use the convnet as a feature extractor
- Given a bounding box, compute a “deep” feature
- Run an SVM (one per category) on the deep features
- Warp cropped images to make them “square”
- Add some background context
- Very accurate!!!



R-CNN [Girshick2013]

- Simple pipeline!
- Use the convnet as a feature extractor
- Given a bounding box, compute a “deep” feature
- Run an SVM (one per category) on the deep features
- Warp cropped images to make them “square”
- Add some background context
- Very accurate!!!
- Unfortunately slow
  - More than 60 seconds per image



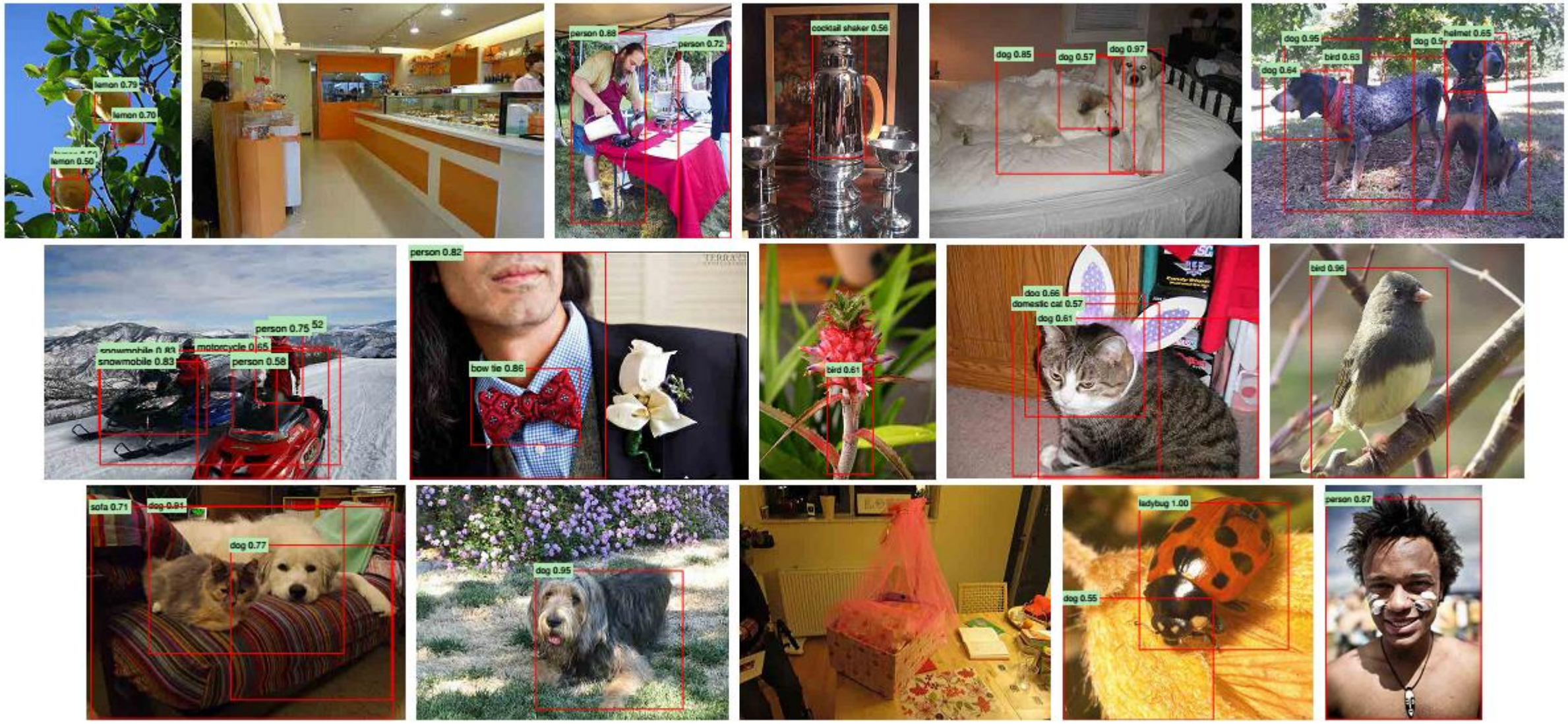
R-CNN results

+25%!!!!!

VOC 2007 test	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv	mAP
R-CNN $pool_5$	51.8	60.2	36.4	27.8	23.2	52.8	60.6	49.2	18.3	47.8	44.3	40.8	56.6	58.7	42.4	23.4	46.1	36.7	51.3	55.7	44.2
R-CNN fc_6	59.3	61.8	43.1	34.0	25.1	53.1	60.6	52.8	21.7	47.8	42.7	47.8	52.5	58.5	44.6	25.6	48.3	34.0	53.1	58.0	46.2
R-CNN fc_7	57.6	57.9	38.5	31.8	23.7	51.2	58.9	51.4	20.0	50.5	40.9	46.0	51.6	55.9	43.3	23.3	48.1	35.3	51.0	57.4	44.7
R-CNN FT $pool_5$	58.2	63.3	37.9	27.6	26.1	54.1	66.9	51.4	26.7	55.5	43.4	43.1	57.7	59.0	45.8	28.1	50.8	40.6	53.1	56.4	47.3
R-CNN FT fc_6	63.5	66.0	47.9	37.7	29.9	62.5	70.2	60.2	32.0	57.9	47.0	53.5	60.1	64.2	52.2	31.3	55.0	50.0	57.7	63.0	53.1
R-CNN FT fc_7	64.2	69.7	50.0	41.9	32.0	62.6	71.0	60.7	32.7	58.5	46.5	56.1	60.6	66.8	54.2	31.5	52.8	48.9	57.9	64.7	54.2
R-CNN FT fc_7 BB	**68.1**	**72.8**	**56.8**	**43.0**	**36.8**	**66.3**	**74.2**	**67.6**	**34.4**	**63.5**	**54.5**	**61.2**	**69.1**	**68.6**	**58.7**	**33.4**	**62.9**	**51.1**	**62.5**	**64.8**	**58.5**
DPM v5 [20]	33.2	60.3	10.2	16.1	27.3	54.3	58.2	23.0	20.0	24.1	26.7	12.7	58.1	48.2	43.2	12.0	21.1	36.1	46.0	43.5	33.7
DPM ST [28]	23.8	58.2	10.5	8.5	27.1	50.4	52.0	7.3	19.2	22.8	18.1	8.0	55.9	44.8	32.4	13.3	15.9	22.8	46.2	44.9	29.1
DPM HSC [31]	32.2	58.3	11.5	16.3	30.6	49.9	54.8	23.5	21.5	27.7	34.0	13.7	58.1	51.6	39.9	12.4	23.5	34.4	47.4	45.2	34.3

R. Girshick, J. Donahue, T. Darrell, J. Malik, Rich feature hierarchies for accurate object detection and semantic segmentation, arXiv 2013

R-CNN examples



R-CNN examples

Fast R-CNN

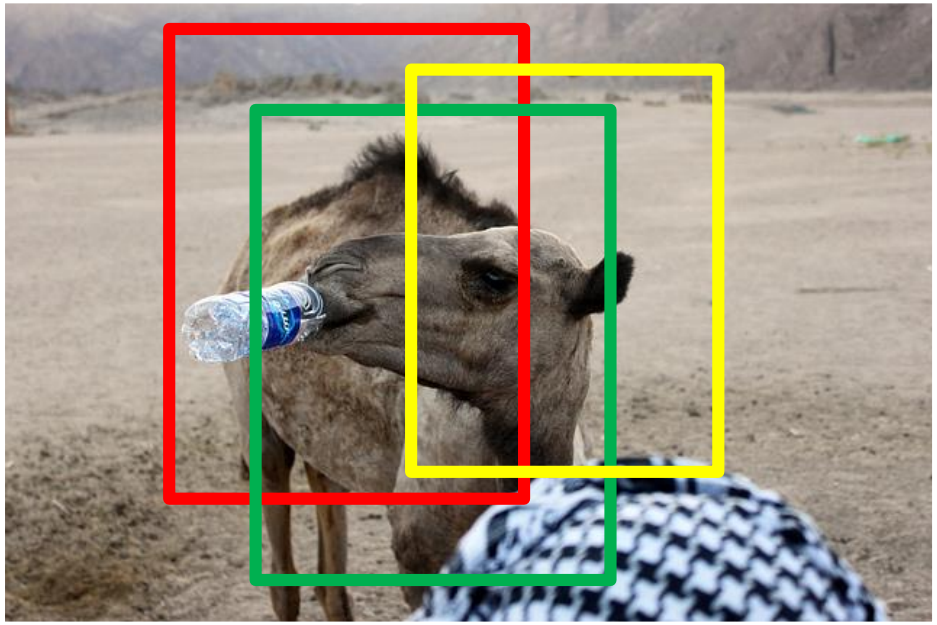
- Observation: A Convnet is convolutional!!!
- Convolutions are location specific
  - The same filter is applied on different locations
- R-CNN assumes that the image locations for different bounding boxes are different
- However, because of the convolutions the convolved image locations are shared between box proposals
- Can we reused the convolutional feature maps

Fast R-CNN

- Observation: A Convnet is convolutional!!!
- Convolutions are location specific
  - The same filter is applied on different locations
- R-CNN assumes that the image locations for different bounding boxes are different
- However, because of the convolutions the convolved image locations are shared between box proposals
- Can we reused the convolutional feature maps

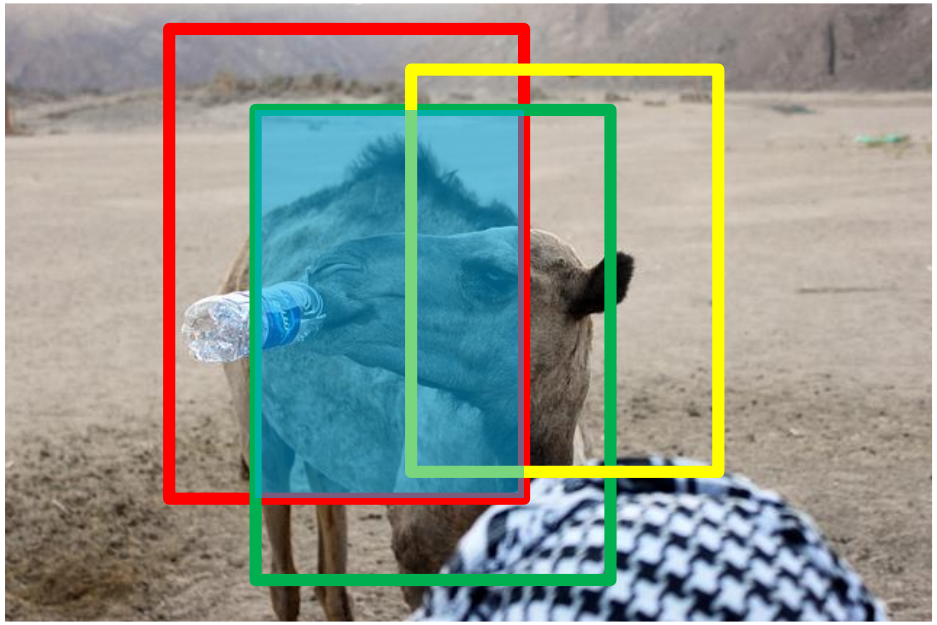
Fast R-CNN

- Observation: A Convnet is convolutional!!!
- Convolutions are location specific
  - The same filter is applied on different locations
- R-CNN assumes that the image locations for different bounding boxes are different
- However, because of the convolutions the convolved image locations are shared between box proposals
- Can we reused the convolutional feature maps



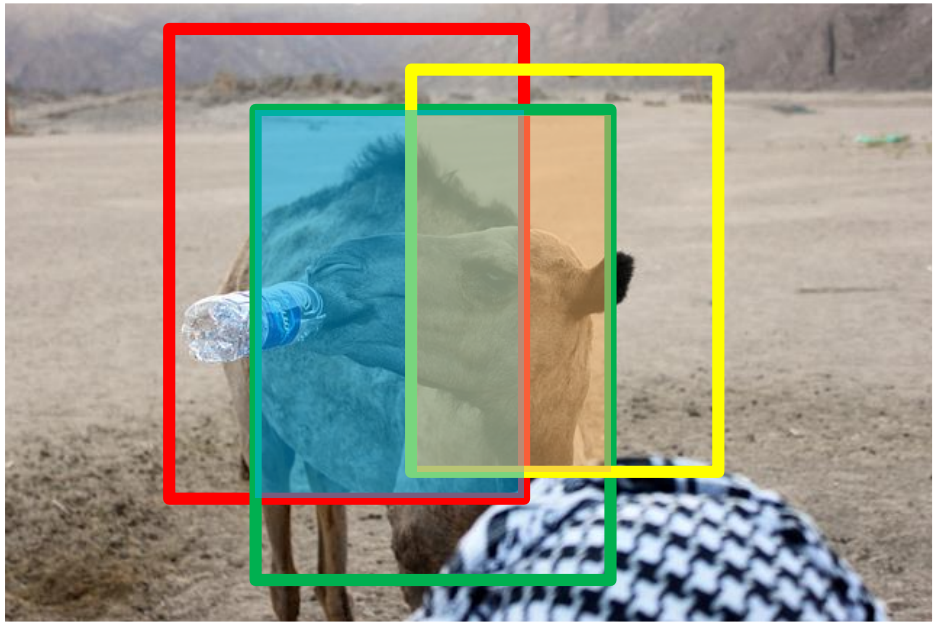
Fast R-CNN

- Observation: A Convnet is convolutional!!!
- Convolutions are location specific
  - The same filter is applied on different locations
- R-CNN assumes that the image locations for different bounding boxes are different
- However, because of the convolutions the convolved image locations are shared between box proposals
- Can we reused the convolutional feature maps



Fast R-CNN

- Observation: A Convnet is convolutional!!!
- Convolutions are location specific
  - The same filter is applied on different locations
- R-CNN assumes that the image locations for different bounding boxes are different
- However, because of the convolutions the convolved image locations are shared between box proposals
- Can we reused the convolutional feature maps

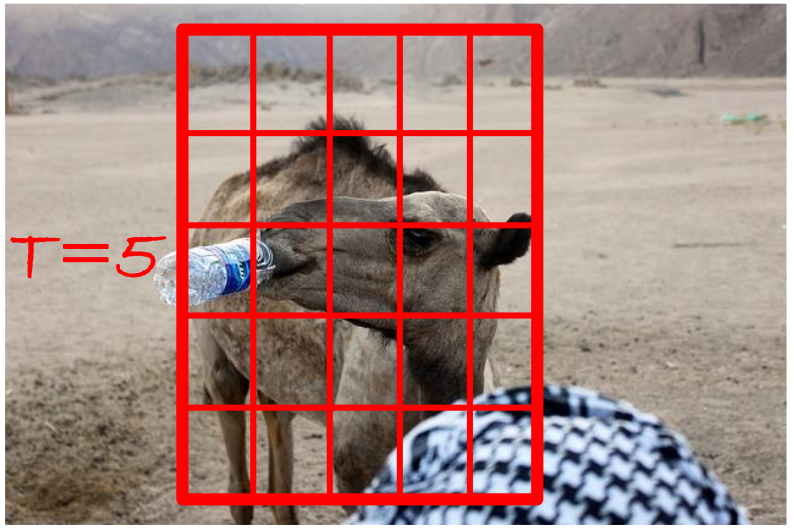


Fast-RCNN

- Do not compute forward propagate from scratch for each box
- Compute feature maps once
- Reuse the convolutions for different boxes

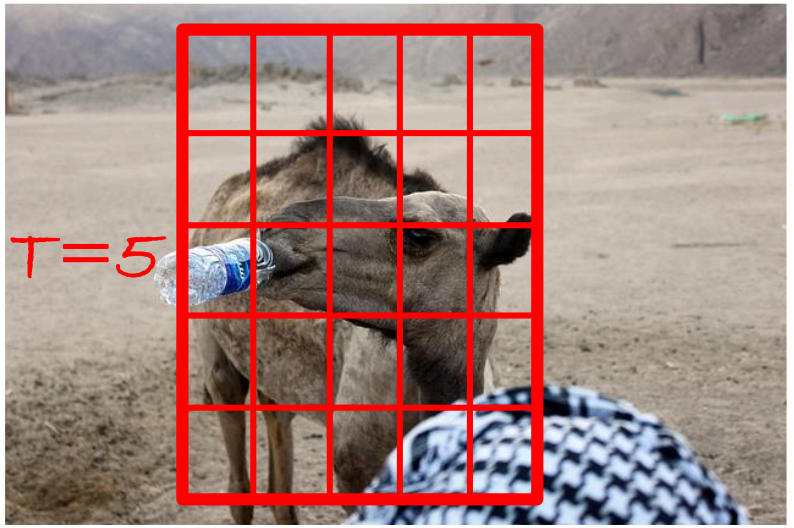
Fast-RCNN

- Do not compute forward propagate from scratch for each box
- Compute feature maps once
- Reuse the convolutions for different boxes
- Region-of-Interest pooling
  - Don’t define the stride absolutely (e.g. sample every 4 pixels)
  - Define stride relatively (box width divided by predefined number of “poolings” T)
  - Fixed length vector



Fast-RCNN

- Do not compute forward propagate from scratch for each box
- Compute feature maps once
- Reuse the convolutions for different boxes
- Region-of-Interest pooling
  - Don’t define the stride absolutely (e.g. sample every 4 pixels)
  - Define stride relatively (box width divided by predefined number of “poolings” T)
  - Fixed length vector
- End-to-end training!
- More accurate
- Much faster
  - Less than a second per image
- Still, external box proposals needed



Fast-RCNN results

method	train set	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	persn	plant	sheep	sofa	train	tv	mAP
SPPnet BB [11]†	07 \ diff	73.9	72.3	62.5	51.5	44.4	74.4	73.0	74.4	42.3	73.6	57.7	70.3	74.6	74.3	54.2	34.0	56.4	56.4	67.9	73.5	63.1
R-CNN BB [10]	07	73.4	77.0	63.4	45.4	**44.6**	75.1	78.1	79.8	40.5	73.7	62.2	79.4	78.1	73.1	64.2	**35.6**	66.8	67.2	70.4	**71.1**	66.0
FRCN [ours]	07	74.5	78.3	69.2	53.2	36.6	77.3	78.2	82.0	40.7	72.7	67.9	79.6	79.2	73.0	69.0	30.1	65.4	70.2	75.8	65.8	66.9
FRCN [ours]	07 \ diff	74.6	**79.0**	68.6	57.0	39.3	79.5	**78.6**	81.9	**48.0**	74.0	67.4	80.5	80.7	74.1	69.6	31.8	67.1	68.4	75.3	65.5	68.1
FRCN [ours]	07+12	**77.0**	78.1	**69.3**	**59.4**	38.3	**81.6**	**78.6**	**86.7**	42.8	**78.8**	**68.9**	**84.7**	**82.0**	**76.6**	**69.9**	31.8	**70.1**	**74.8**	**80.4**	70.4	**70.0**

Table 1. **VOC 2007 test** detection average precision (%). All methods use VGG16. Training set key: **07**: VOC07 trainval, **07 \ diff**: **07** without "difficult" examples, **07+12**: union of **07** and VOC12 trainval. †SPPnet results were prepared by the authors of [11].

	Fast R-CNN			R-CNN			SPPnet
	S	**M**	**L**	**S**	**M**	**L**	†**L**
train time (h)	**1.2**	2.0	9.5	22	28	84	25
train speedup	**18.3**×	14.0×	8.8×	1×	1×	1×	3.4×
test rate (s/im)	0.10	0.15	0.32	9.8	12.1	47.0	2.3
▷ with SVD	**0.06**	0.08	0.22	-	-	-	-
test speedup	98×	80×	146×	1×	1×	1×	20×
▷ with SVD	169×	150×	**213**×	-	-	-	-
VOC07 mAP	57.1	59.2	**66.9**	58.5	60.2	66.0	63.1
▷ with SVD	56.5	58.7	66.6	-	-	-	-

Table 4. Runtime comparison between the same models in Fast R-CNN, R-CNN, and SPPnet. Fast R-CNN uses single-scale mode. SPPnet uses the five scales specified in [11]. †Timing provided by the authors of [11]. Times were measured on an Nvidia K40 GPU.

R. Girshick, Fast R-CNN, CVPR, 20152013

Fast-RCNN results

method	train set	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	persn	plant	sheep	sofa	train	tv	mAP
SPPnet BB [11]†	07 \ diff	73.9	72.3	62.5	51.5	44.4	74.4	73.0	74.4	42.3	73.6	57.7	70.3	74.6	74.3	54.2	34.0	56.4	56.4	67.9	73.5	63.1
R-CNN BB [10]	07	73.4	77.0	63.4	45.4	**44.6**	75.1	78.1	79.8	40.5	73.7	62.2	79.4	78.1	73.1	64.2	**35.6**	66.8	67.2	70.4	**71.1**	66.0
FRCN [ours]	07	74.5	78.3	69.2	53.2	36.6	77.3	78.2	82.0	40.7	72.7	67.9	79.6	79.2	73.0	69.0	30.1	65.4	70.2	75.8	65.8	66.9
FRCN [ours]	07 \ diff	74.6	**79.0**	68.6	57.0	39.3	79.5	**78.6**	81.9	**48.0**	74.0	67.4	80.5	80.7	74.1	69.6	31.8	67.1	68.4	75.3	65.5	68.1
FRCN [ours]	07+12	**77.0**	78.1	**69.3**	**59.4**	38.3	**81.6**	**78.6**	**86.7**	42.8	**78.8**	**68.9**	**84.7**	**82.0**	**76.6**	**69.9**	31.8	**70.1**	**74.8**	**80.4**	70.4	**70.0**

Table 1. **VOC 2007 test** detection average precision (%). All methods use VGG16. Training set key: **07**: VOC07 trainval, **07 \ diff**: **07** without "difficult" examples, **07+12**: union of **07** and VOC12 trainval. †SPPnet results were prepared by the authors of [11].

+4%!!!!!

	Fast R-CNN			R-CNN			SPPnet
	S	**M**	**L**	**S**	**M**	**L**	†**L**
train time (h)	**1.2**	2.0	9.5	22	28	84	25
train speedup	**18.3**×	14.0×	8.8×	1×	1×	1×	3.4×
test rate (s/im)	0.10	0.15	0.32	9.8	12.1	47.0	2.3
▷ with SVD	**0.06**	0.08	0.22	-	-	-	-
test speedup	98×	80×	146×	1×	1×	1×	20×
▷ with SVD	169×	150×	**213**×	-	-	-	-
VOC07 mAP	57.1	59.2	**66.9**	58.5	60.2	66.0	63.1
▷ with SVD	56.5	58.7	66.6	-	-	-	-

Table 4. Runtime comparison between the same models in Fast R-CNN, R-CNN, and SPPnet. Fast R-CNN uses single-scale mode. SPPnet uses the five scales specified in [11]. †Timing provided by the authors of [11]. Times were measured on an Nvidia K40 GPU.

R. Girshick, Fast R-CNN, CVPR, 20152013

Fast-RCNN results

method	train set	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	persn	plant	sheep	sofa	train	tv	mAP
SPPnet BB [11]†	07 \ diff	73.9	72.3	62.5	51.5	44.4	74.4	73.0	74.4	42.3	73.6	57.7	70.3	74.6	74.3	54.2	34.0	56.4	56.4	67.9	73.5	63.1
R-CNN BB [10]	07	73.4	77.0	63.4	45.4	**44.6**	75.1	78.1	79.8	40.5	73.7	62.2	79.4	78.1	73.1	64.2	**35.6**	66.8	67.2	70.4	**71.1**	66.0
FRCN [ours]	07	74.5	78.3	69.2	53.2	36.6	77.3	78.2	82.0	40.7	72.7	67.9	79.6	79.2	73.0	69.0	30.1	65.4	70.2	75.8	65.8	66.9
FRCN [ours]	07 \ diff	74.6	**79.0**	68.6	57.0	39.3	79.5	**78.6**	81.9	**48.0**	74.0	67.4	80.5	80.7	74.1	69.6	31.8	67.1	68.4	75.3	65.5	68.1
FRCN [ours]	07+12	**77.0**	78.1	**69.3**	**59.4**	38.3	**81.6**	**78.6**	**86.7**	42.8	**78.8**	**68.9**	**84.7**	**82.0**	**76.6**	**69.9**	31.8	**70.1**	**74.8**	**80.4**	70.4	**70.0**

Table 1. **VOC 2007 test** detection average precision (%). All methods use VGG16. Training set key: **07**: VOC07 trainval, **07 \ diff**: **07** without "difficult" examples, **07+12**: union of **07** and VOC12 trainval. †SPPnet results were prepared by the authors of [11].

+4%!!!!!

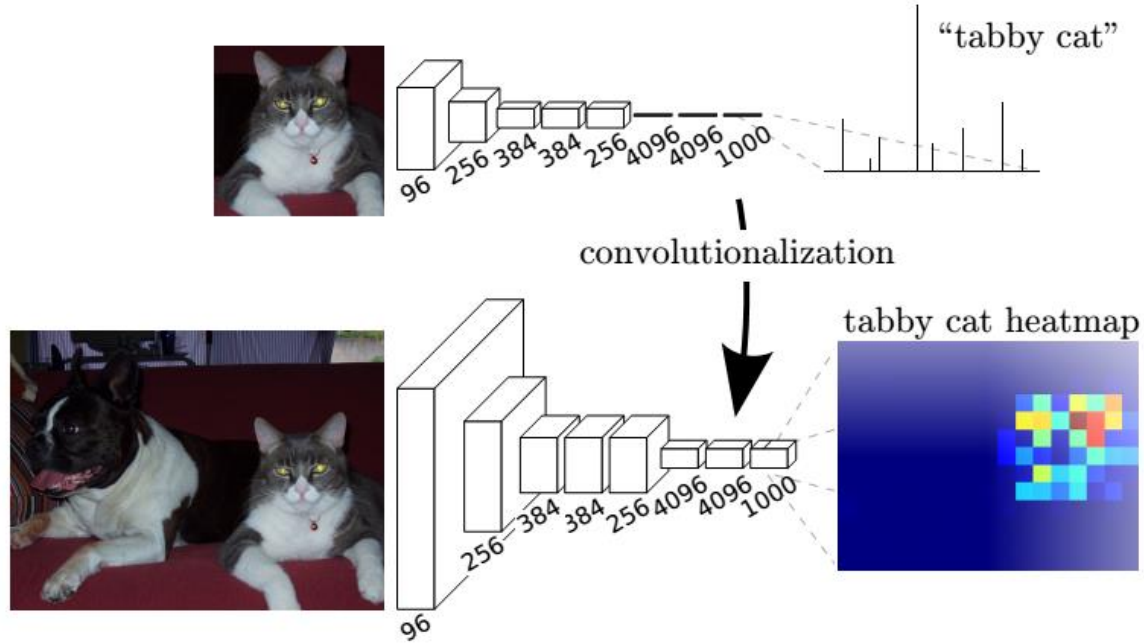
	Fast R-CNN			R-CNN			SPPnet
	S	**M**	**L**	**S**	**M**	**L**	†**L**
train time (h)	**1.2**	2.0	9.5	22	28	84	25
train speedup	**18.3**×	14.0×	8.8×	1×	1×	1×	3.4×
test rate (s/im)	0.10	0.15	0.32	9.8	12.1	47.0	2.3
▷ with SVD	**0.06**	0.08	0.22	-	-	-	-
test speedup	98×	80×	146×	1×	1×	1×	20×
▷ with SVD	169×	150×	**213**×	-	-	-	-
VOC07 mAP	57.1	59.2	**66.9**	58.5	60.2	66.0	63.1
▷ with SVD	56.5	58.7	66.6	-	-	-	-

Table 4. Runtime comparison between the same models in Fast R-CNN, R-CNN, and SPPnet. Fast R-CNN uses single-scale mode. SPPnet uses the five scales specified in [11]. †Timing provided by the authors of [11]. Times were measured on an Nvidia K40 GPU.

R. Girshick, Fast R-CNN, CVPR, 20152013

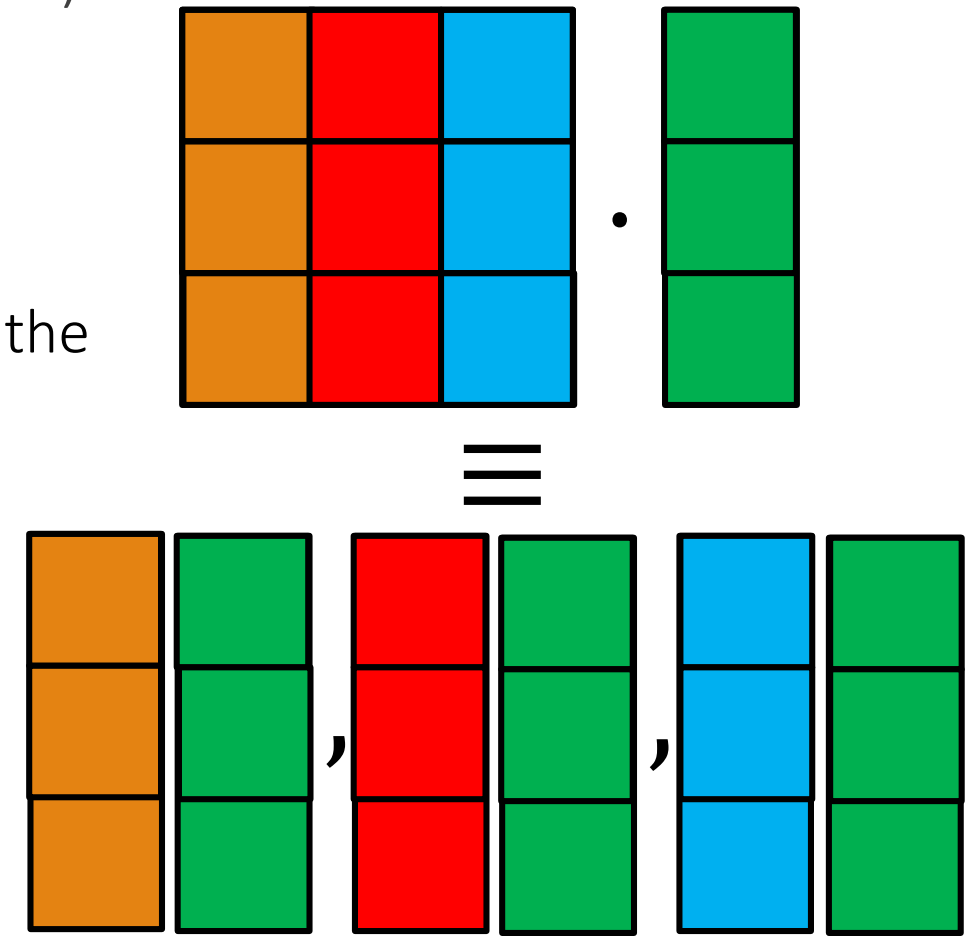
Fully Convolutional Networks [LongCVPR2014]

- Not single number output (“car” vs. “not car”)
  - Pixel-wise output (as many outputs, as inputs)



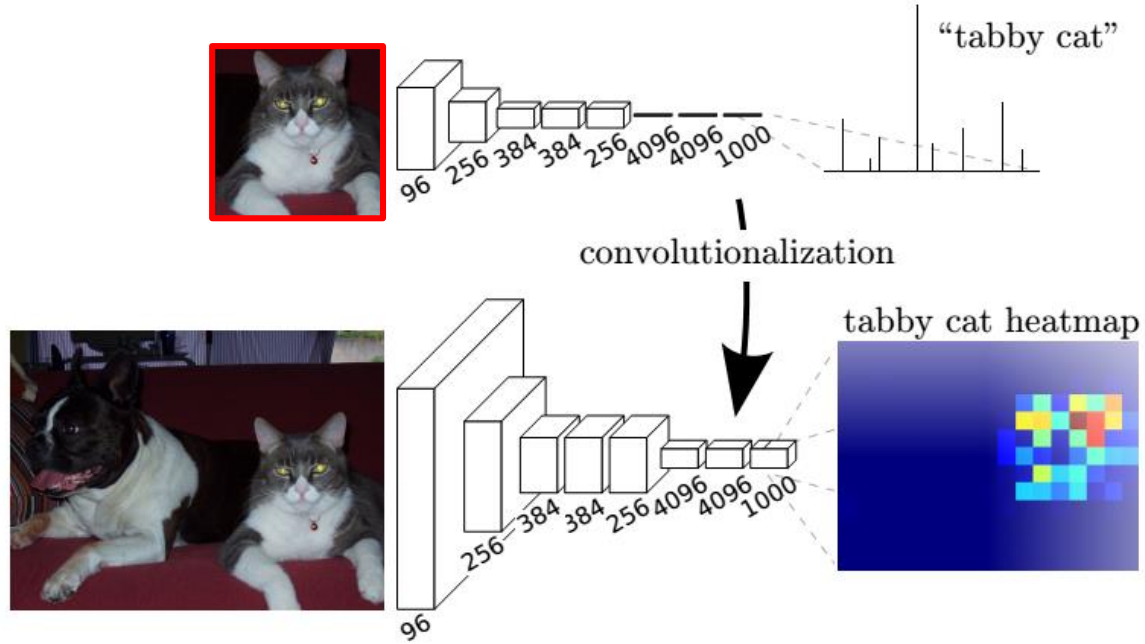
Fully Convolutional Networks [LongCVPR2014]

- Not single number output (“car” vs. “not car”)
  - Pixel-wise output (as many outputs, as inputs)
- A fully connected layer can be viewed as a convolution
  - The size of the convolution filter equals the size of the whole layer



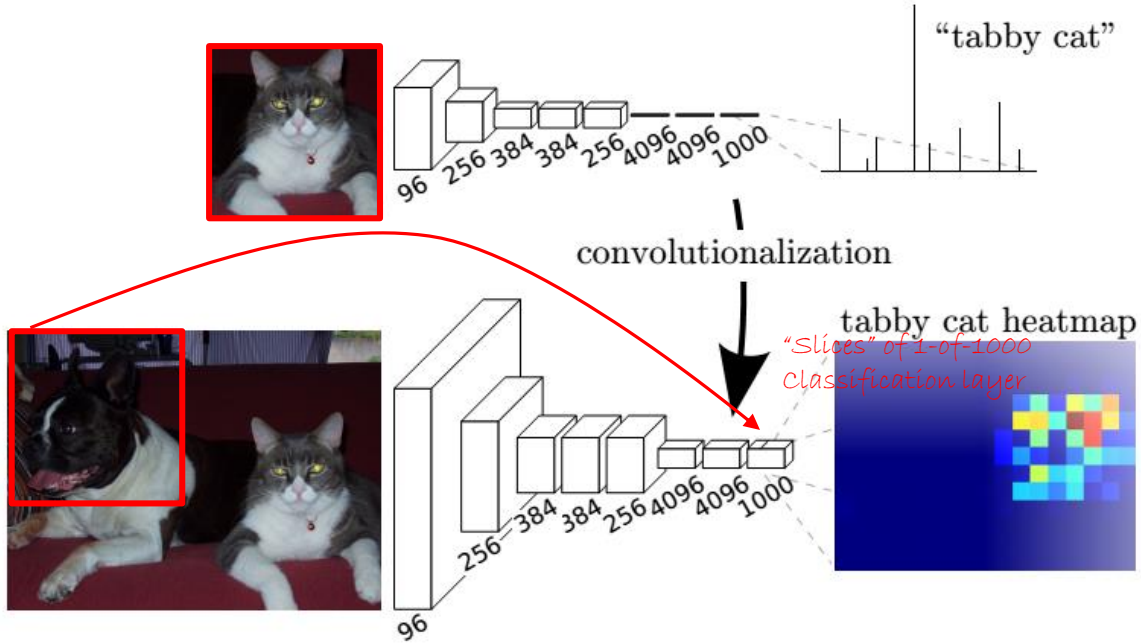
Fully Convolutional Networks [LongCVPR2014]

- Not single number output (“car” vs. “not car”)
  - Pixel-wise output (as many outputs, as inputs)
- A fully connected layer can be viewed as a convolution
  - The size of the convolution filter equals the size of the whole layer



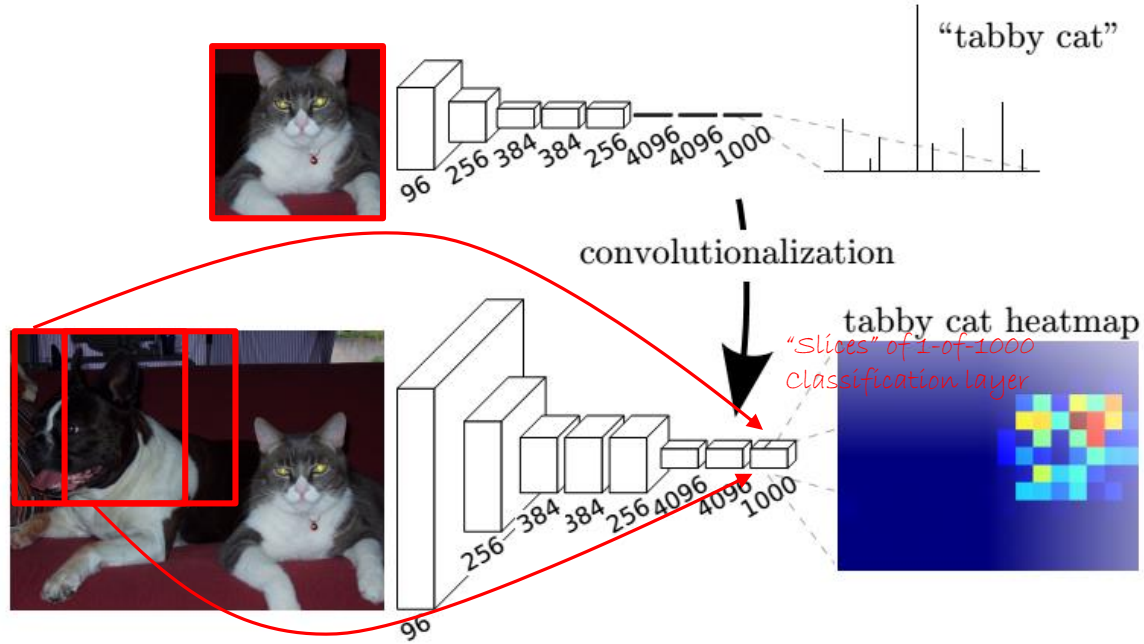
Fully Convolutional Networks [LongCVPR2014]

- Not single number output (“car” vs. “not car”)
  - Pixel-wise output (as many outputs, as inputs)
- A fully connected layer can be viewed as a convolution
  - The size of the convolution filter equals the size of the whole layer



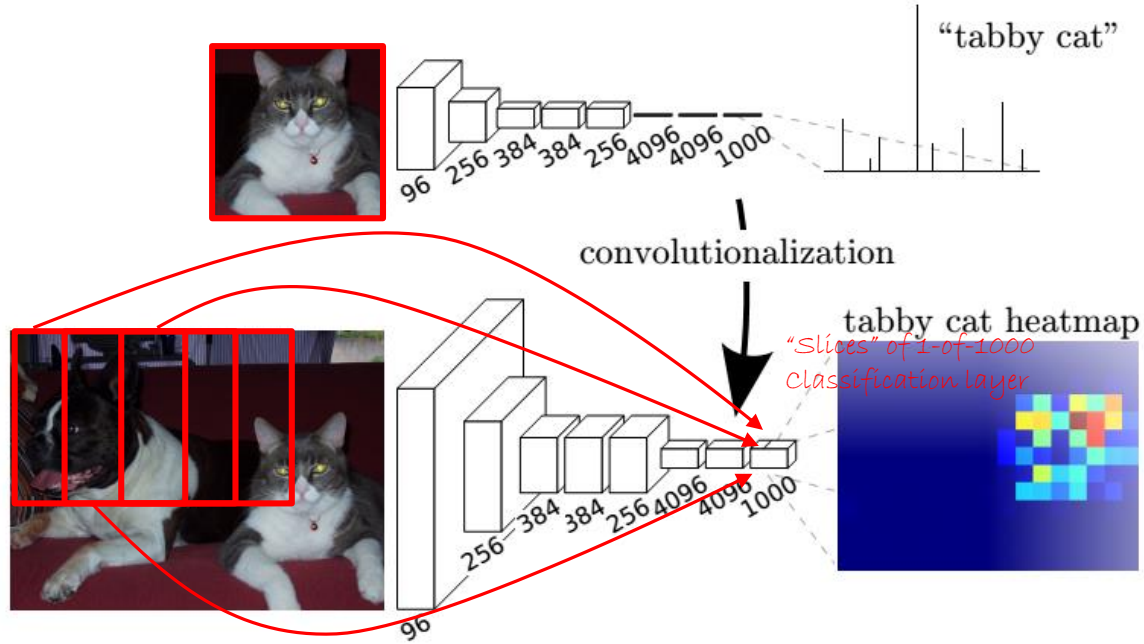
Fully Convolutional Networks [LongCVPR2014]

- Not single number output (“car” vs. “not car”)
  - Pixel-wise output (as many outputs, as inputs)
- A fully connected layer can be viewed as a convolution
  - The size of the convolution filter equals the size of the whole layer



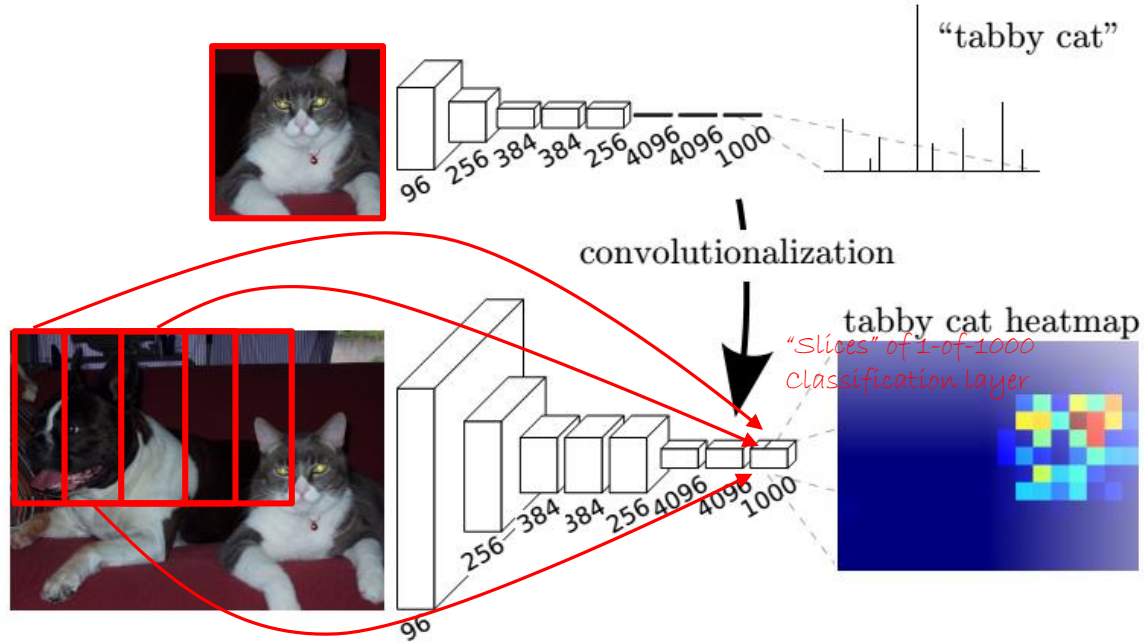
Fully Convolutional Networks [LongCVPR2014]

- Not single number output (“car” vs. “not car”)
  - Pixel-wise output (as many outputs, as inputs)
- A fully connected layer can be viewed as a convolution
  - The size of the convolution filter equals the size of the whole layer



Fully Convolutional Networks [LongCVPR2014]

- Not single number output (“car” vs. “not car”)
  - Pixel-wise output (as many outputs, as inputs)
- A fully connected layer can be viewed as a convolution
  - The size of the convolution filter equals the size of the whole layer
- Upsampling
  - Deconvolution filters learnt with backpropagation
  - Deconvolution filter return feature maps larger than input
  - Outputs progressively bigger till they reach input size



Fully Convolutional Networks architecture

- Connect intermediate layers to output

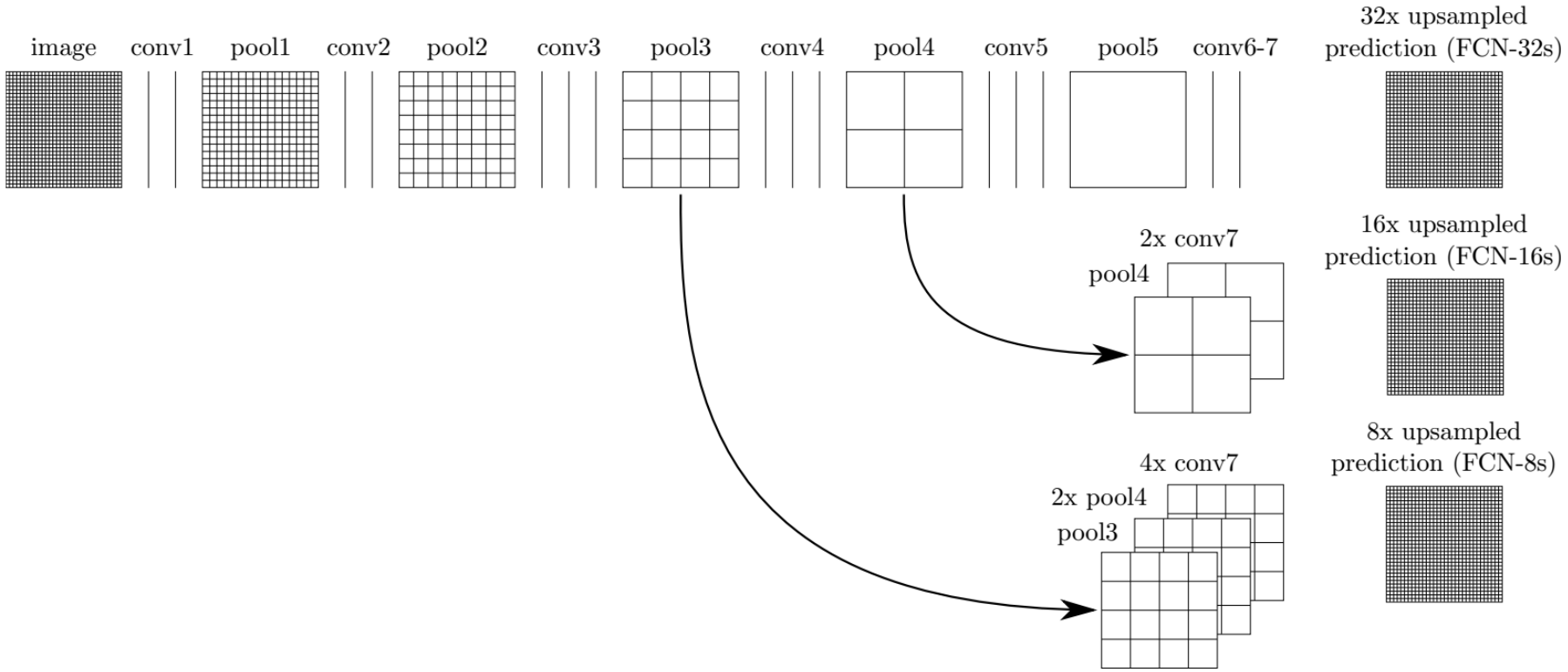


Figure 3. Our DAG nets learn to combine coarse, high layer information with fine, low layer information. Pooling and prediction layers are shown as grids that reveal relative spatial coarseness, while intermediate layers are shown as vertical lines. First row (FCN-32s): Our single-stream net, described in Section 4.1, upsamples stride 32 predictions back to pixels in a single step. Second row (FCN-16s): Combining predictions from both the final layer and the `pool4` layer, at stride 16, lets our net predict finer details, while retaining high-level semantic information. Third row (FCN-8s): Additional predictions from `pool3`, at stride 8, provide further precision.

Fully Convolutional Networks results

Table 3. Our fully convolutional net gives a 20% relative improvement over the state-of-the-art on the PASCAL VOC 2011 and 2012 test sets and reduces inference time.

	mean IU VOC2011 test	mean IU VOC2012 test	inference time
R-CNN [12]	47.9	-	-
SDS [17]	52.6	51.6	$\sim$ 50 s
FCN-8s	**62.7**	**62.2**	$\sim$ **175 ms**

Deep Face Recognition

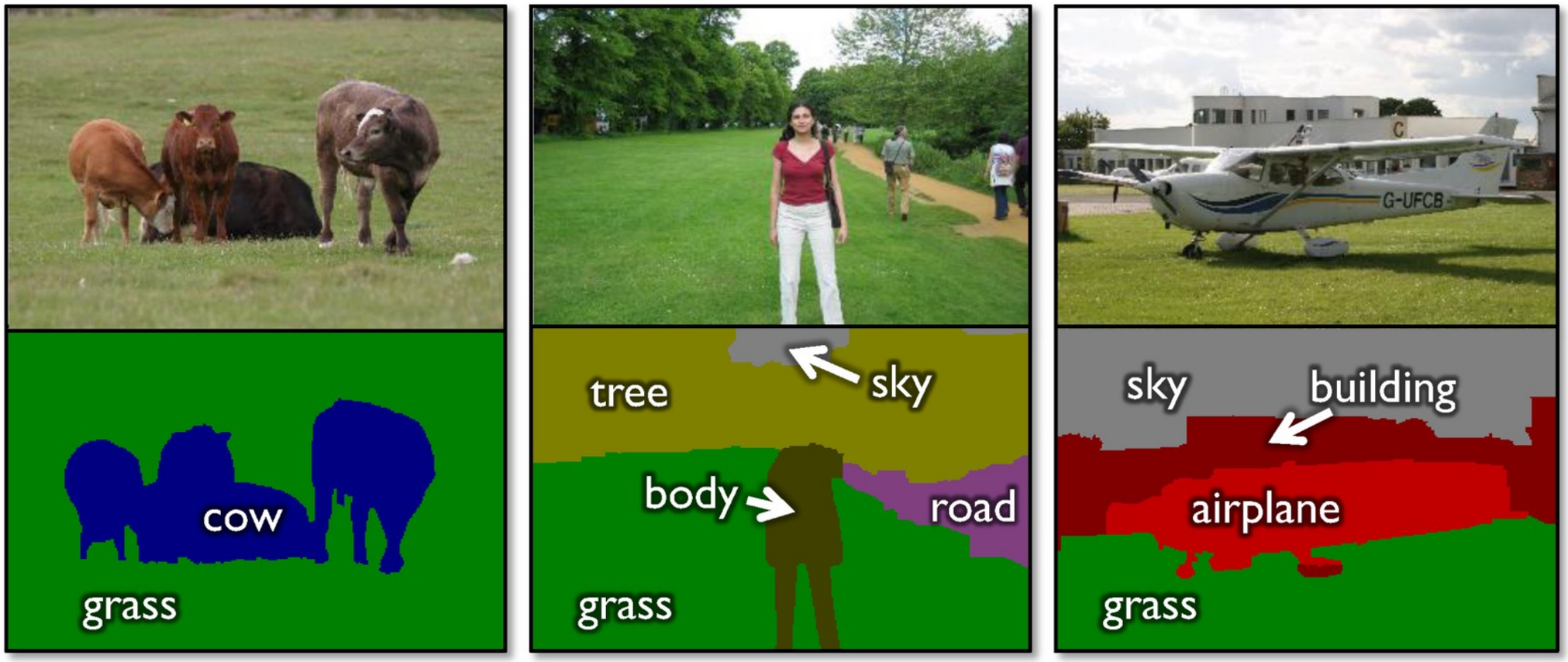
- Identity verification
  - Find whether two pictures belong to the same person
- Essentially similar to metric learning
  - Find a projection matrix (metric), with which distances between different person faces are smaller than distances from different persons
- Triplet "Euclidean" loss

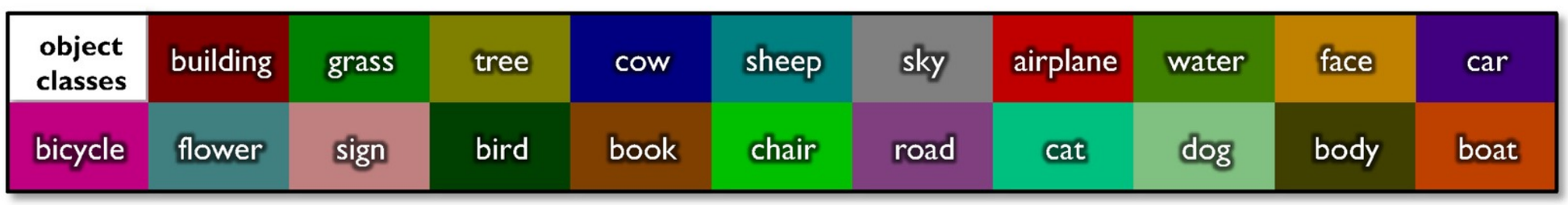
$$E(W') = \sum_{(a,p,n)\in T} \max\left\{0, a - |x_a - x_n|_2^2 + \left|x_a - x_p\right|_2^2\right\}, x_i = W' \frac{\varphi(\ell_i)}{|\phi(\ell_i)|_2}$$

- Start from very deep architecture
- More similar to classification, but face detection already very accurate

Deep Face Recognition results

No.	Method	Images	Networks	Acc.
1	Fisher Vector Faces [21]	-	-	93.10
2	DeepFace [29]	4M	3	97.35
3	Fusion [30]	500M	5	98.37
4	DeepID-2,3		200	99.47
5	FaceNet [17]	200M	1	98.87
6	FaceNet [17] + Alignment	200M	1	99.63
7	Ours	2.6M	1	98.95



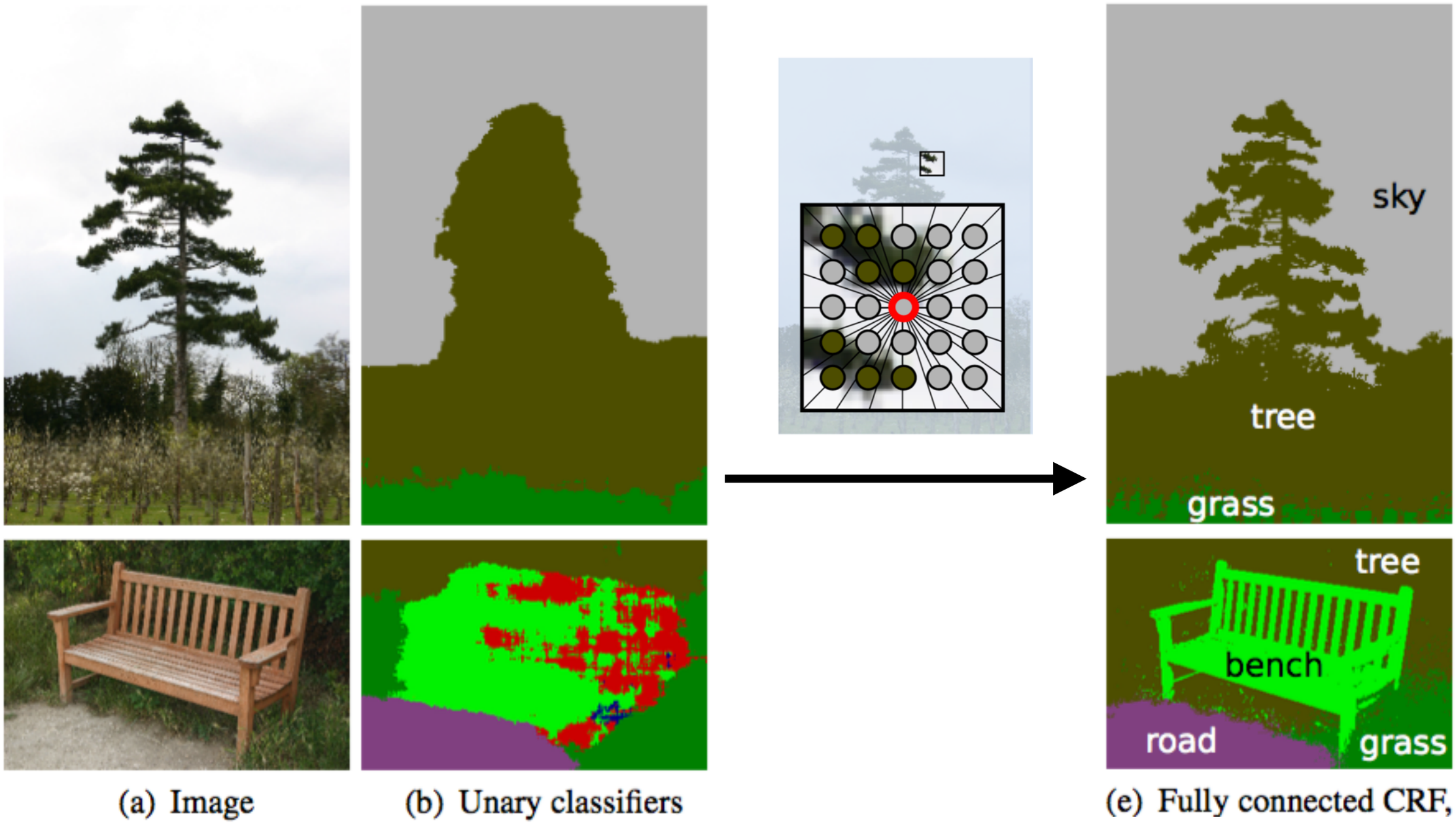


[Shotton et al., 2009]

Image Segmentation using Conditional Random Fields

Deep Learning @ UvA
Patrick Putzky

Outlook



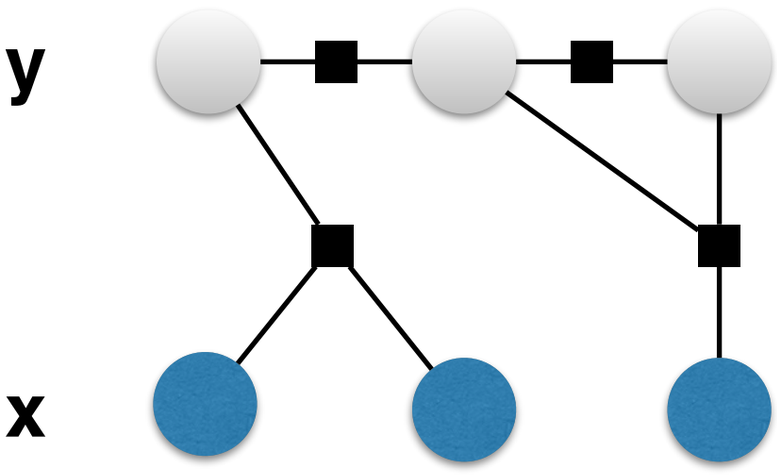
(a) Image (b) Unary classifiers (e) Fully connected CRF,

[Krähenbühl & Koltun, 2011]

Conditional Random Fields

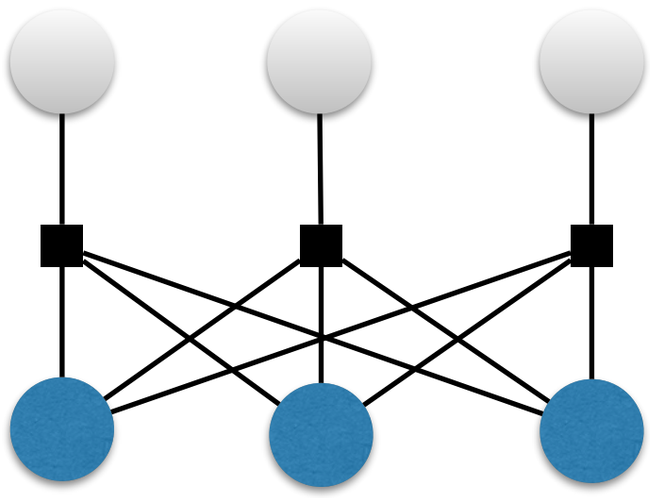
$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \exp(-E(\mathbf{y}|\mathbf{x}))$$

General form



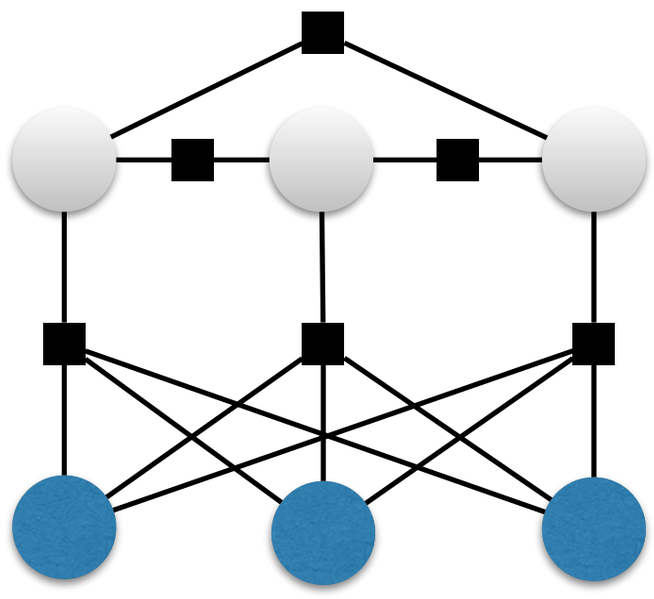
$$E(\mathbf{y}|\mathbf{x}) = \sum_{c \in \mathcal{C}_\mathcal{G}} \psi_c(\mathbf{y}|\mathbf{x})$$

Unary form



$$E(\mathbf{y}|\mathbf{x}) = \sum_i \psi_u(y_i|\mathbf{x})$$

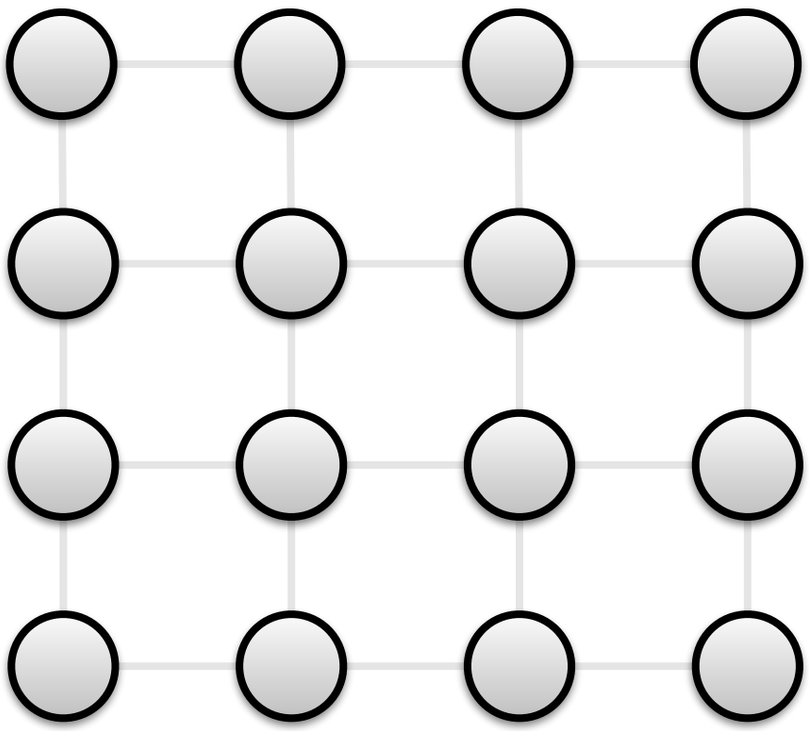
Pairwise CRF



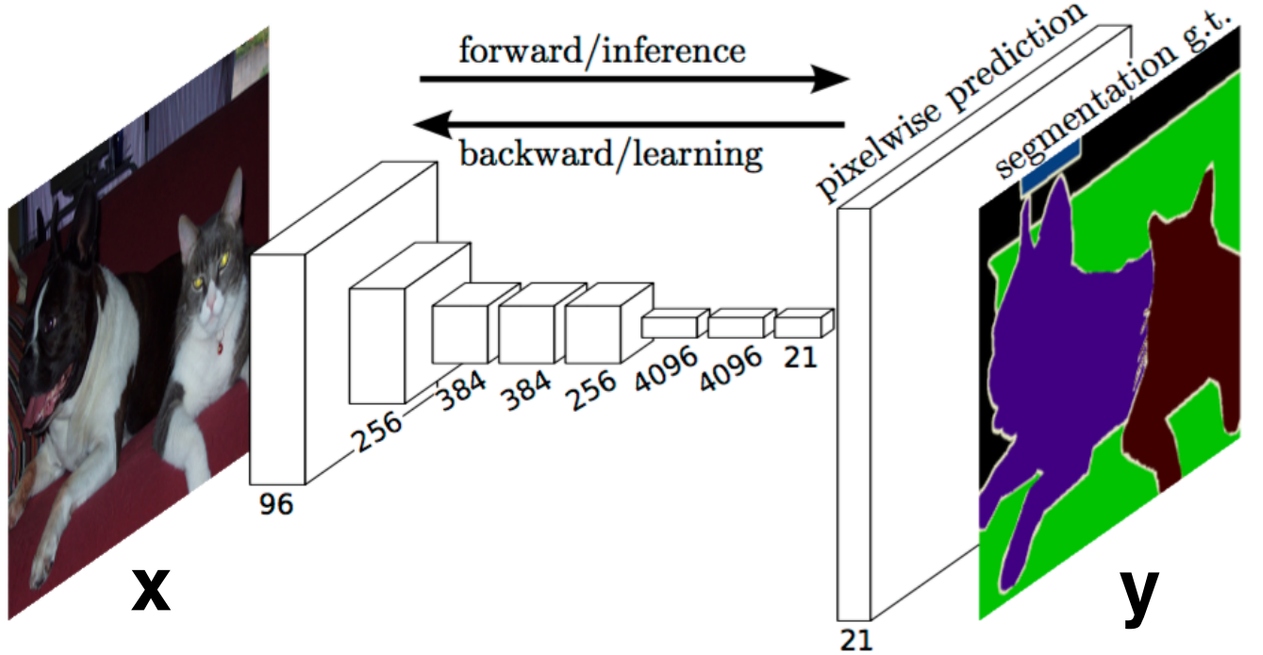
$$E(\mathbf{y}|\mathbf{x}) = \sum_i \psi_u(y_i|\mathbf{x}) + \sum_{i<j} \psi_p(y_i, y_j|\mathbf{x})$$

Unary models

$$E(\mathbf{y}|\mathbf{x}) = \sum_i \psi_u(y_i|\mathbf{x})$$



Example: Fully Convolutional Networks

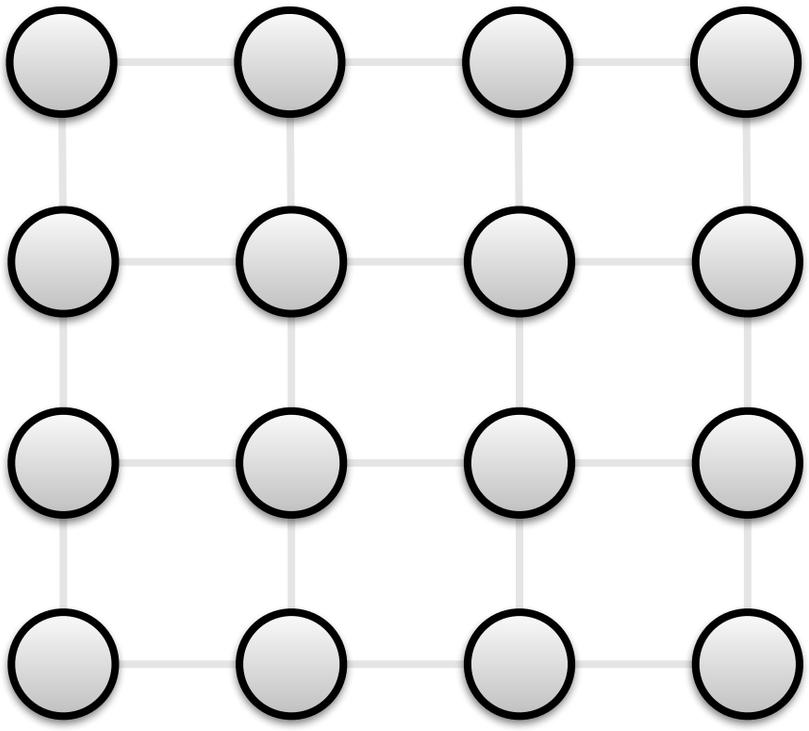


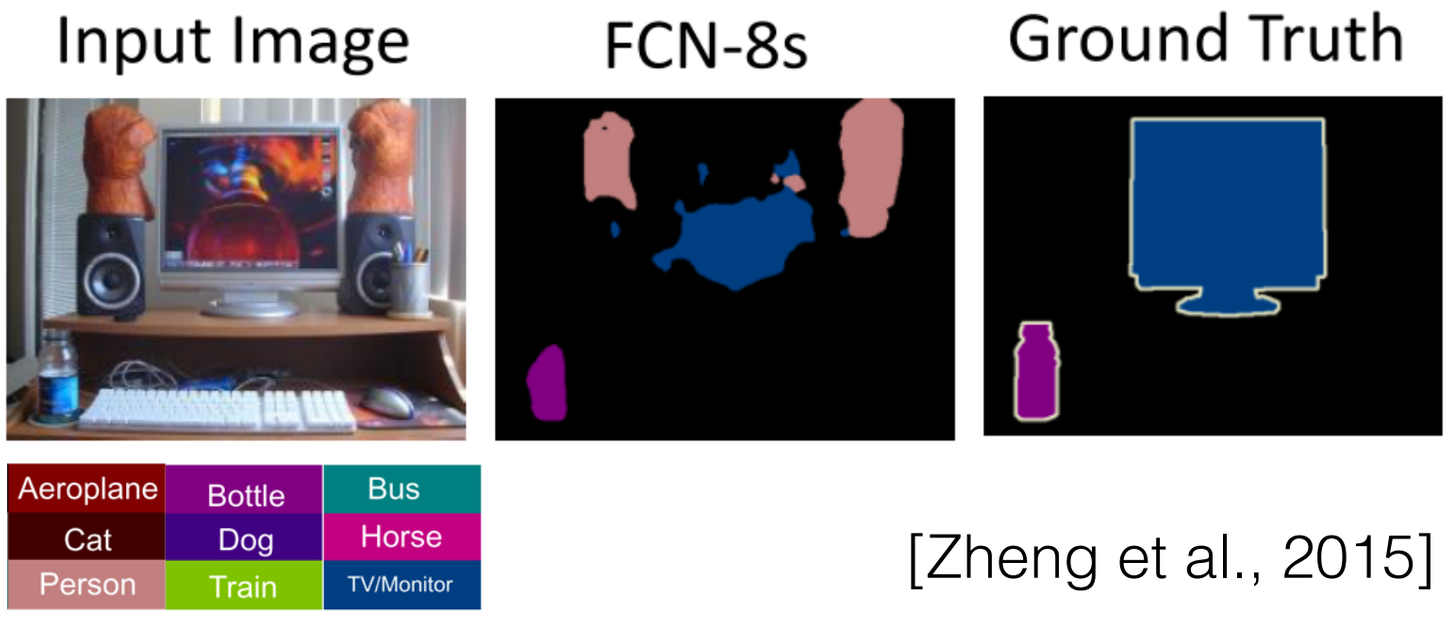
[Long et al., 2015]

- Per pixel predictions
- No interactions between neighbouring class predictions

Unary models

Fully Convolutional Networks: Fail Case



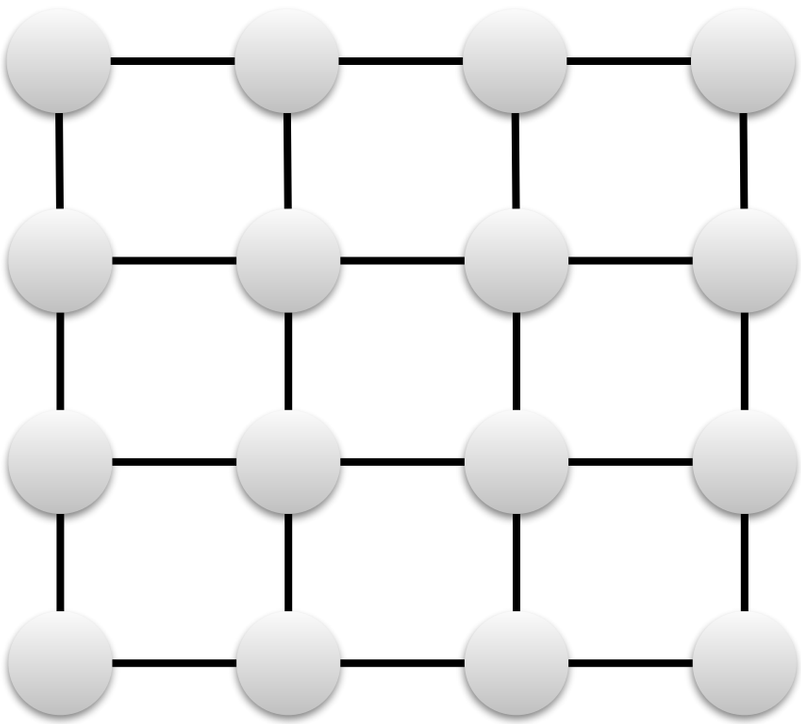


[Zheng et al., 2015]

- Per pixel predictions
- No interactions between neighbouring class predictions
- **No object coherency**

Adjacency CRFs

$$E(\mathbf{y}|\mathbf{x}) = \sum_i \underbrace{\psi_u(y_i|\mathbf{x})}_{\text{unary potential}} + \sum_{j \in \mathcal{N}(i)} \underbrace{\psi_p(y_i, y_j|\mathbf{x})}_{\text{pairwise potential}}$$

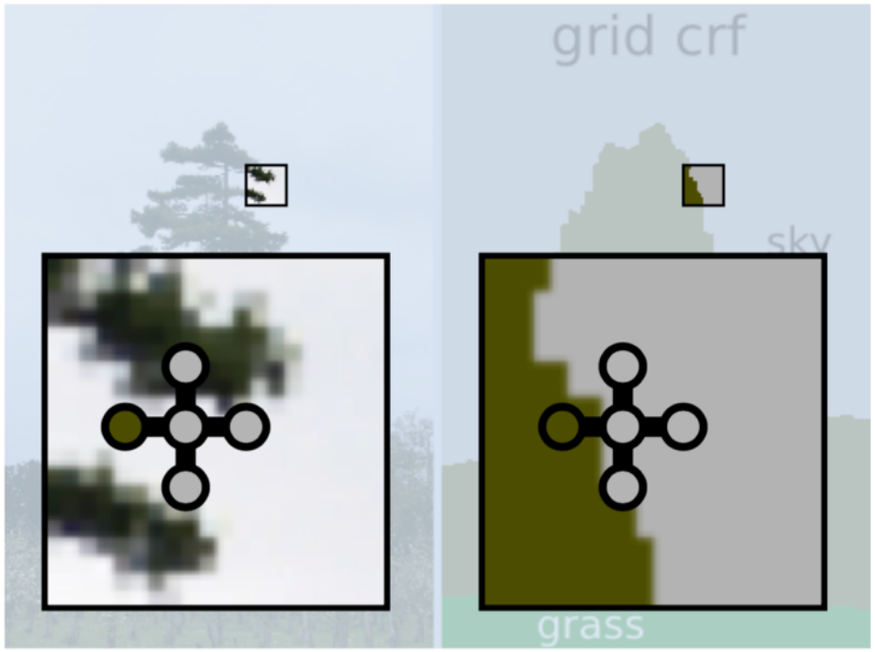


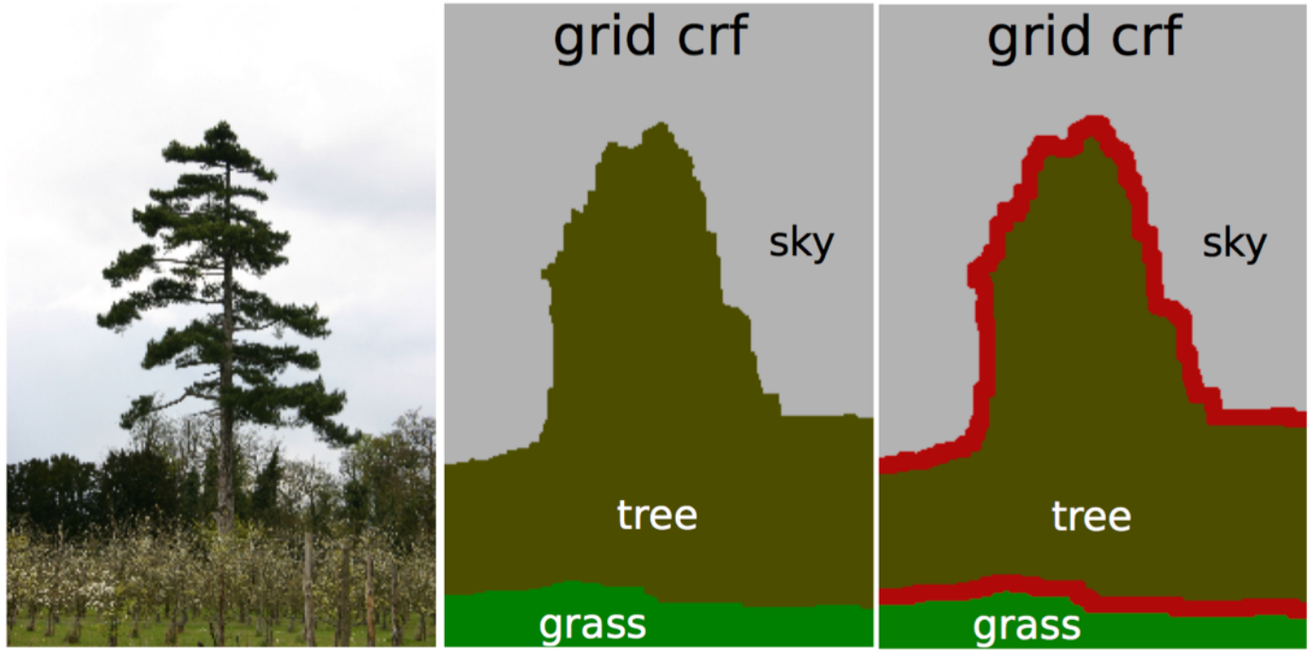
- define a neighbourhood graph
- arises naturally in images
- efficient inference
- **only local interactions**
- **graph is not trainable**

Adjacency CRFs

Popular: Potts model

$$\psi_p(y_i, y_j|\mathbf{x}) = 1_{[y_i \neq y_j]} \left(w_1 \exp\left(-\beta\|\mathbf{x}_i - \mathbf{x}_j\|^2\right) + w_2\right)$$



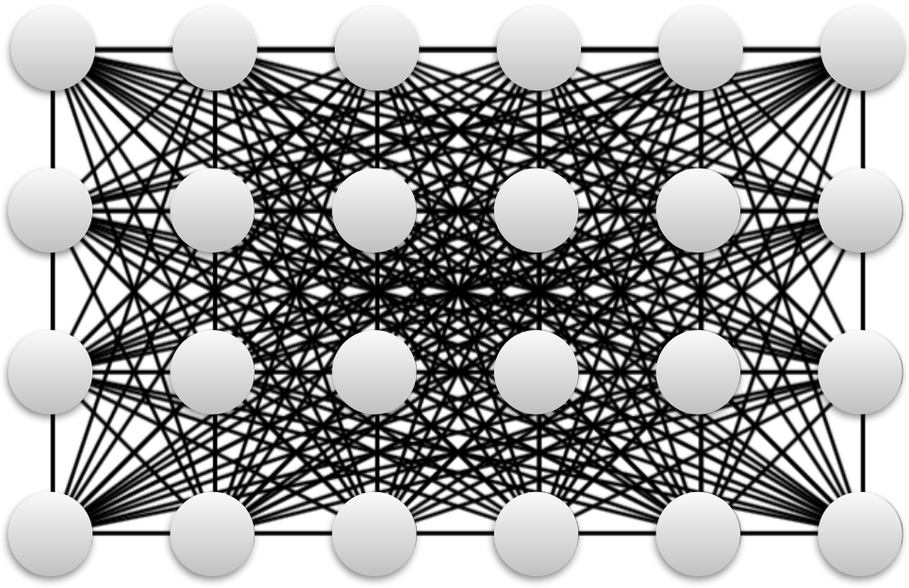


- **No distinction between classes**

- **Shrinking bias**
- Limited edge-awareness

Fully connected CRFs

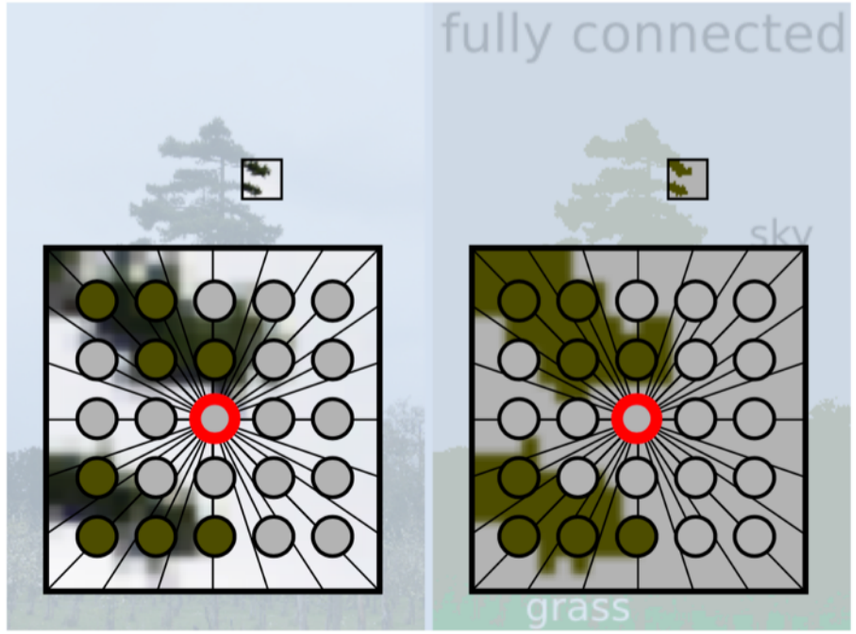
$$E(\mathbf{y}|\mathbf{x}) = \sum_i \underbrace{\psi_u(y_i|\mathbf{x})}_{\text{unary potential}} + \sum_{j \neq i} \underbrace{\psi_p(y_i, y_j|\mathbf{x})}_{\text{pairwise potential}}$$

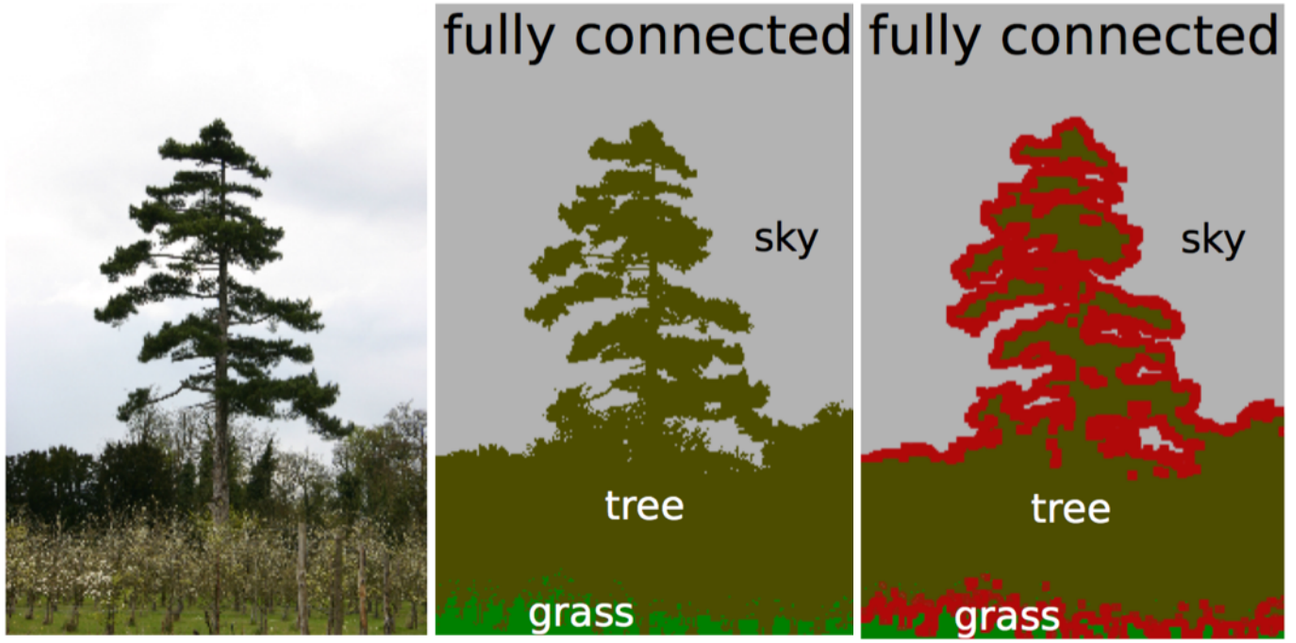


- Edges between all pairs of nodes
- Long-range interactions

Fully connected CRFs

$$E(\mathbf{y}|\mathbf{x}) = \sum_i \underbrace{\psi_u(y_i|\mathbf{x})}_{\text{unary potential}} + \sum_{j \neq i} \underbrace{\psi_p(y_i, y_j|\mathbf{x})}_{\text{pairwise potential}}$$





- Strength of edges define connectivity
- No more shrinking bias
- **N^2 edges -> computationally expensive**

Gaussian Edge Potentials

Potts model

$$\psi_p(y_i, y_j|\mathbf{x}) = 1_{[y_i \neq y_j]} \left(w_1 \exp\left(-\beta \|\mathbf{x}_i - \mathbf{x}_j\|^2\right) + w_2 \right)$$

Generalisation

$$\psi_p(y_i, y_j|\mathbf{x}) = \boxed{\mu(y_i, y_j)} \sum_{m=1}^{M} \boxed{w_m} k_m\left(\mathbf{f}_i, \mathbf{f}_j\right)$$

With Gaussian kernels

trainable parameters

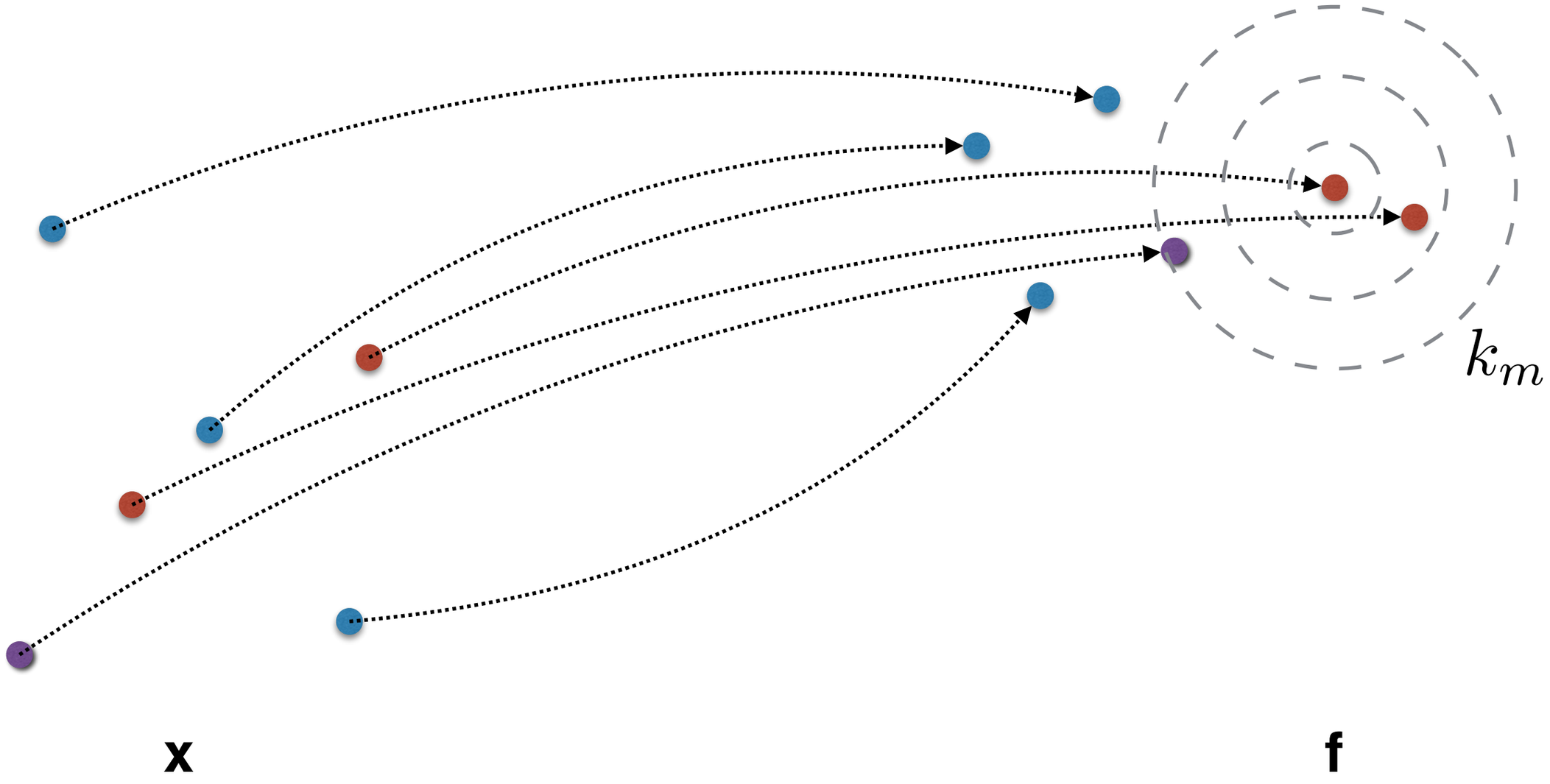
$$k_m\left(\mathbf{f}_i, \mathbf{f}_j\right) = \exp\left(-\frac{1}{2}\left(\mathbf{f}_i - \mathbf{f}_j\right)^{\mathsf{T}} \boxed{\mathbf{Q}_m} \left(\mathbf{f}_i - \mathbf{f}_j\right)\right)$$

Label compatibility $\mu(y_i, y_j)$ models interactions between classes

Potts $\mu(y_i, y_j) = 1_{[y_i \neq y_j]}$

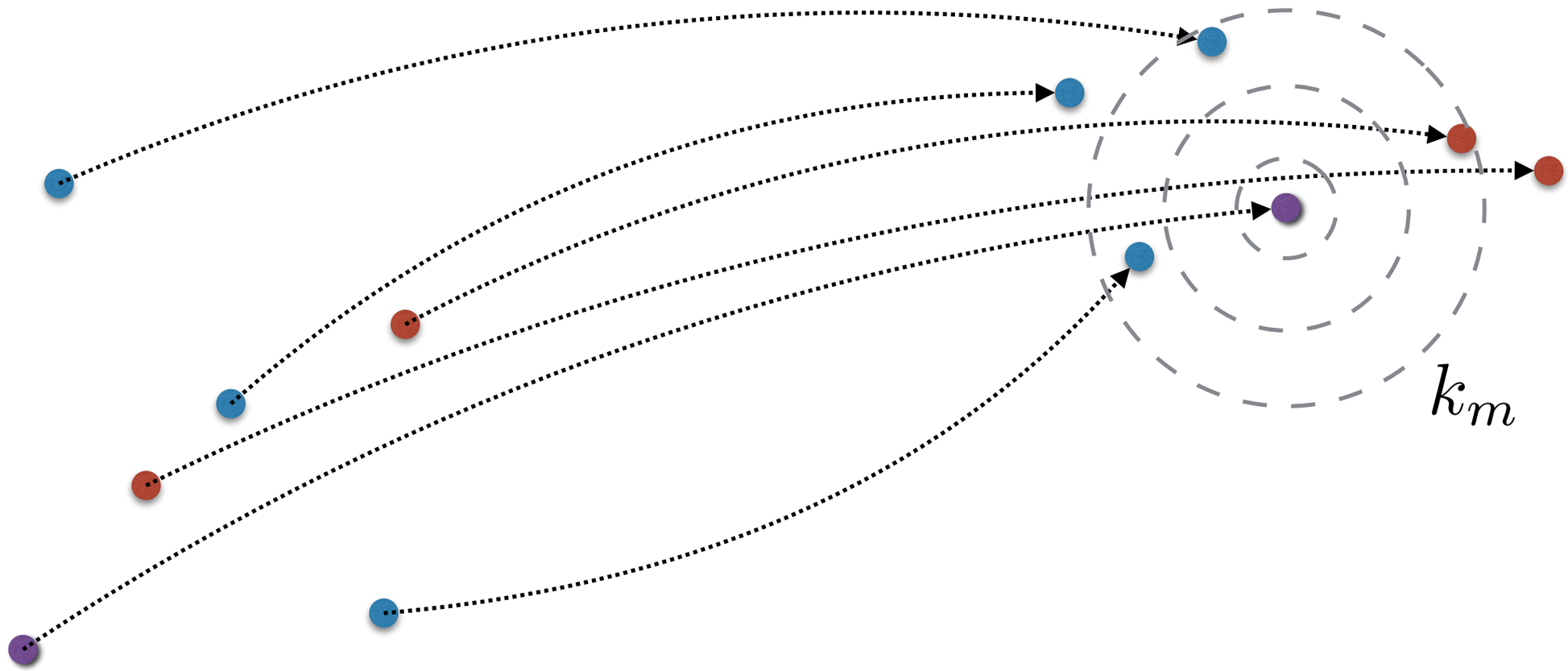
Gaussian Edge Potentials

$$k_m\left(\mathbf{f}_i, \mathbf{f}_j\right) = \exp\left(-\frac{1}{2}\left(\mathbf{f}_i - \mathbf{f}_j\right)^{\mathtt{T}} \mathbf{Q}_m \left(\mathbf{f}_i - \mathbf{f}_j\right)\right)$$



Gaussian Edge Potentials

$$k_m\left(\mathbf{f}_i, \mathbf{f}_j\right) = \exp\left(-\frac{1}{2}\left(\mathbf{f}_i - \mathbf{f}_j\right)^{\mathtt{T}} \mathbf{Q}_m \left(\mathbf{f}_i - \mathbf{f}_j\right)\right)$$



Edge potentials as convolutions

$$E_p(\mathbf{y}|\mathbf{x}) = \sum_i \sum_{j \neq i} \psi_p(y_i, y_j|\mathbf{x})$$

Edge potentials as convolutions

$$E_p(\mathbf{y}|\mathbf{x}) = \sum_i \sum_{j \neq i} \psi_p(y_i, y_j|\mathbf{x})$$

$$= \sum_i \sum_{j \neq i} \mu(y_i, y_j) \sum_{m=1}^{M} w_m k_m \left(\mathbf{f}_i, \mathbf{f}_j\right)$$

Edge potentials as convolutions

$$
\begin{aligned}
E_p(\mathbf{y}|\mathbf{x}) &= \sum_i \sum_{j \neq i} \psi_p(y_i, y_j|\mathbf{x}) \\
&= \sum_i \sum_{j \neq i} \mu(y_i, y_j) \sum_{m=1}^{M} w_m k_m \left(\mathbf{f}_i, \mathbf{f}_j\right) \\
&= \sum_{m=1}^{M} w_m \sum_i \sum_{j \neq i} \mu(y_i, y_j) k_m \left(\mathbf{f}_i, \mathbf{f}_j\right)
\end{aligned}
$$

Edge potentials as convolutions

$$
\begin{aligned}
E_p(\mathbf{y}|\mathbf{x}) &= \sum_i \sum_{j \neq i} \psi_p(y_i, y_j|\mathbf{x}) \\
&= \sum_i \sum_{j \neq i} \mu(y_i, y_j) \sum_{m=1}^{M} w_m k_m \left(\mathbf{f}_i, \mathbf{f}_j\right) \\
&= \sum_{m=1}^{M} w_m \sum_i \sum_{j \neq i} \mu(y_i, y_j) k_m \left(\mathbf{f}_i, \mathbf{f}_j\right) \\
&= \sum_{m=1}^{M} w_m \sum_i \sum_{j=1}^{N} \mu(y_i, y_j) k_m \left(\mathbf{f}_i, \mathbf{f}_j\right) - \mu(y_i, y_i) k_m \left(\mathbf{f}_i, \mathbf{f}_i\right)
\end{aligned}
$$

Edge potentials as convolutions

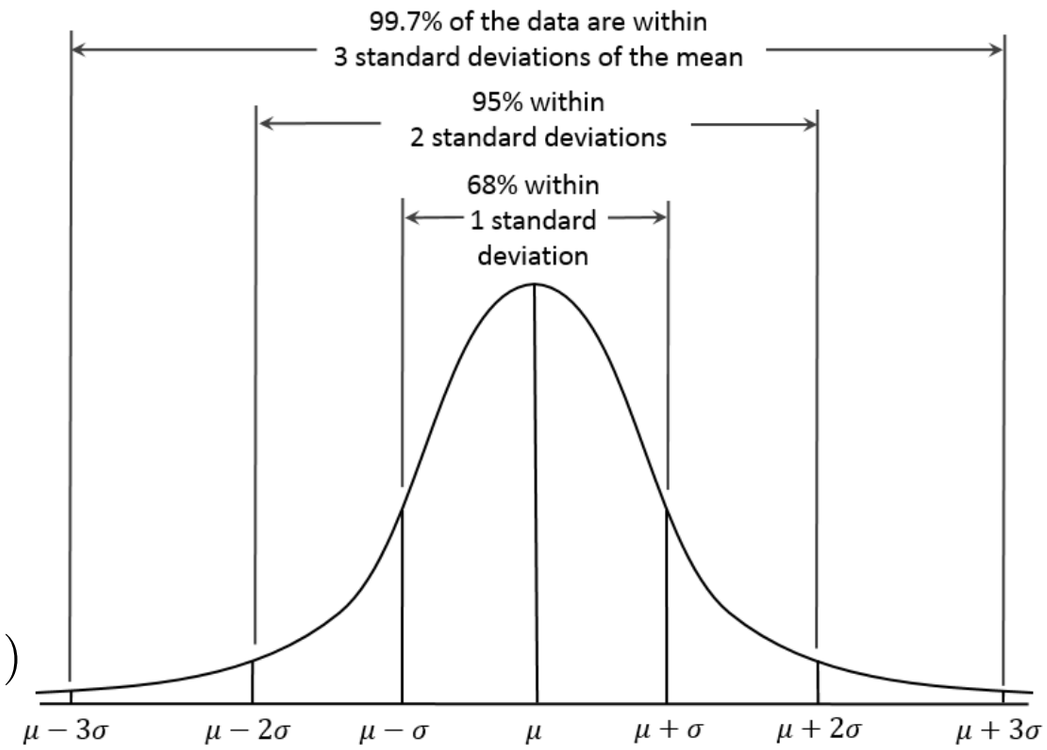
$$E_p(\mathbf{y}|\mathbf{x}) = \sum_i \sum_{j \neq i} \psi_p(y_i, y_j|\mathbf{x})$$

$$= \sum_i \sum_{j \neq i} \mu(y_i, y_j) \sum_{m=1}^{M} w_m k_m \left(\mathbf{f}_i, \mathbf{f}_j\right)$$

$$= \sum_{m=1}^{M} w_m \sum_i \sum_{j \neq i} \mu(y_i, y_j) k_m \left(\mathbf{f}_i, \mathbf{f}_j\right)$$

$$= \sum_{m=1}^{M} w_m \sum_i \sum_{j=1}^{N} \mu(y_i, y_j) k_m \left(\mathbf{f}_i, \mathbf{f}_j\right) - \mu(y_i, y_i) k_m \left(\mathbf{f}_i, \mathbf{f}_i\right)$$

$$= \sum_{m=1}^{M} w_m \sum_i \sum_{l=i-1}^{i-N} \mu(y_i, y_{i-l}) k_m \left(\mathbf{f}_i, \mathbf{f}_{i-l}\right) - \mu(y_i, y_i) k_m \left(\mathbf{f}_i, \mathbf{f}_i\right)$$

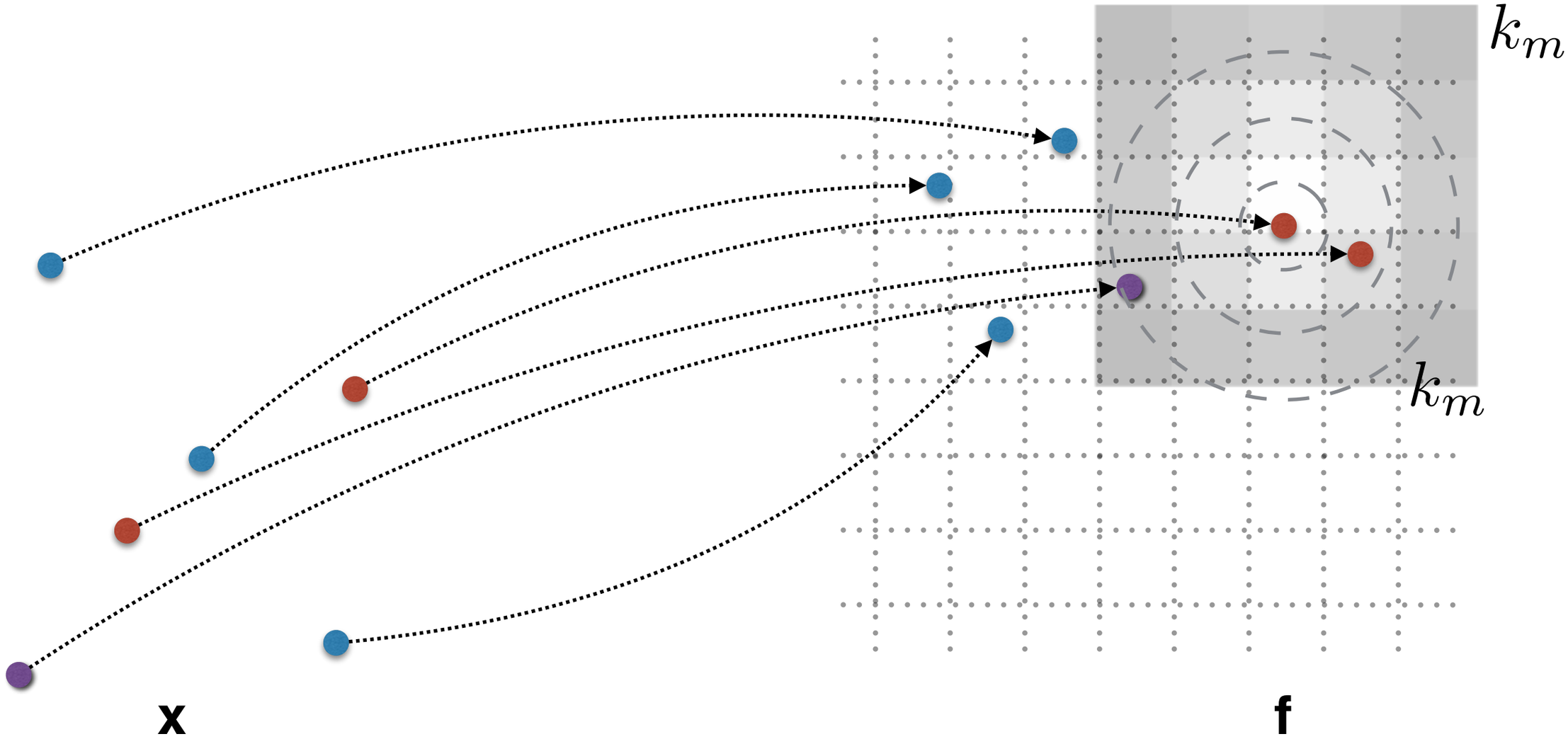


[Wikipedia]

- If k_m is a symmetric kernel the pairwise potential can be implemented as a convolution
- Gaussian has most probability mass around a close region about it's mean
- Approximate this using a truncated Gaussian

Gaussian Edge Potentials

$$k_m\left(\mathbf{f}_i, \mathbf{f}_j\right) = \exp\left(-\frac{1}{2}\left(\mathbf{f}_i - \mathbf{f}_j\right)^{\mathtt{T}} \mathbf{Q}_m \left(\mathbf{f}_i - \mathbf{f}_j\right)\right)$$

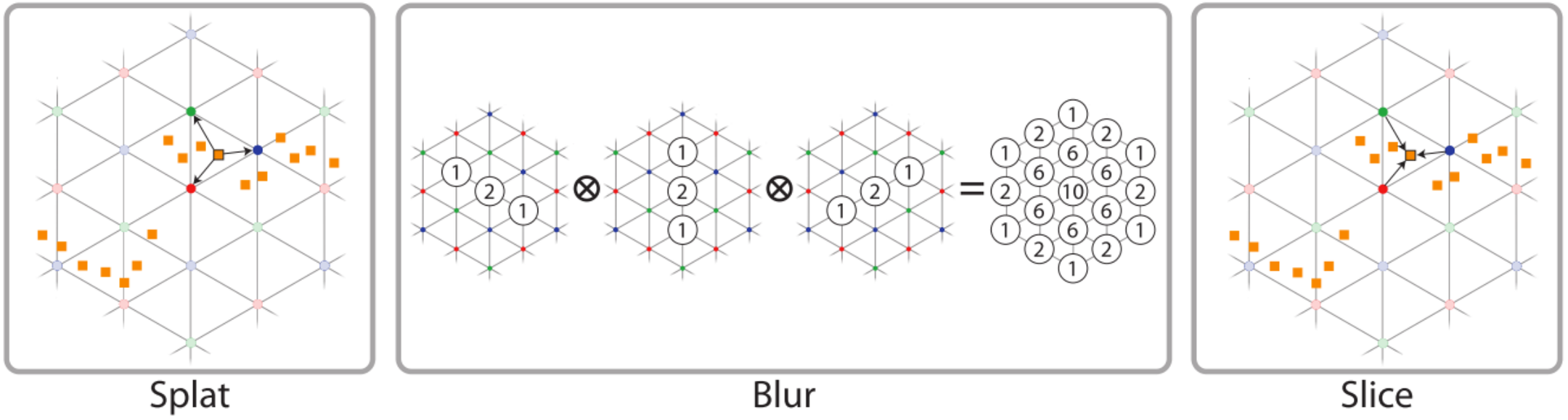


- **Convolution in a high dimensional space**

Permutohedral Lattices for convolution in high dimensional space

Problem:

- $\mathbf{f}_i$ can be high dimensional
- as a result the feature space will be sparsely populated



[Adams et al., 2010]

- Convolutions in $\mathcal{O}(d^2n)$ and $\mathcal{O}(dn)$ for separable filters
- Naive convolution on a grid is $\mathcal{O}(2^dn)$

Efficient inference in dense CRFs

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \exp\left(-\sum_i \psi_u(y_i|\mathbf{x}) - \sum_{j \neq i} \psi_p(y_i, y_j|\mathbf{x})\right)$$

Hard to evaluate

Mean-field approximation: $Q(\mathbf{y}|\mathbf{x}) = \prod_i Q(y_i|\mathbf{x})$

Mean field updates:

$$Q_i(y_i = l) = \frac{1}{Z_i} \exp\left(-\psi_u(y_i|\mathbf{x}) - \sum_{l' \in \mathcal{L}} \mu(l, l') \sum_{m=1}^{M} w_m \sum_{j \neq i} k_m(\mathbf{f}_i, \mathbf{f}_j) Q_j(l')\right)$$

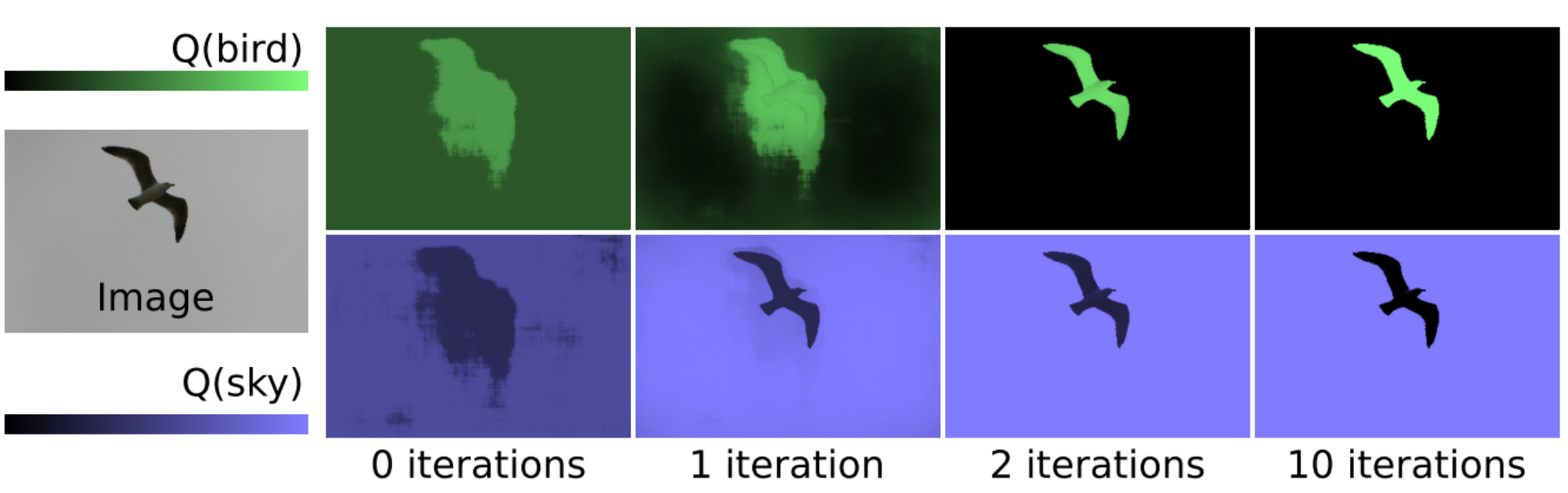
- Naively implemented an update over all Q_i costs $O(N^2)$
- With the approximations from before it is $O(N)$

[Krähenbühl & Koltun, 2011]

Efficient inference in dense CRFs

Mean field updates:

$$Q_i(y_i = l) = \frac{1}{Z_i} \exp\left(-\psi_u(y_i|\mathbf{x}) - \sum_{l' \in \mathcal{L}} \mu\left(l, l'\right) \sum_{m=1}^{M} w_m \sum_{j \neq i} k_m\left(\mathbf{f}_i, \mathbf{f}_j\right) Q_j(l')\right)$$

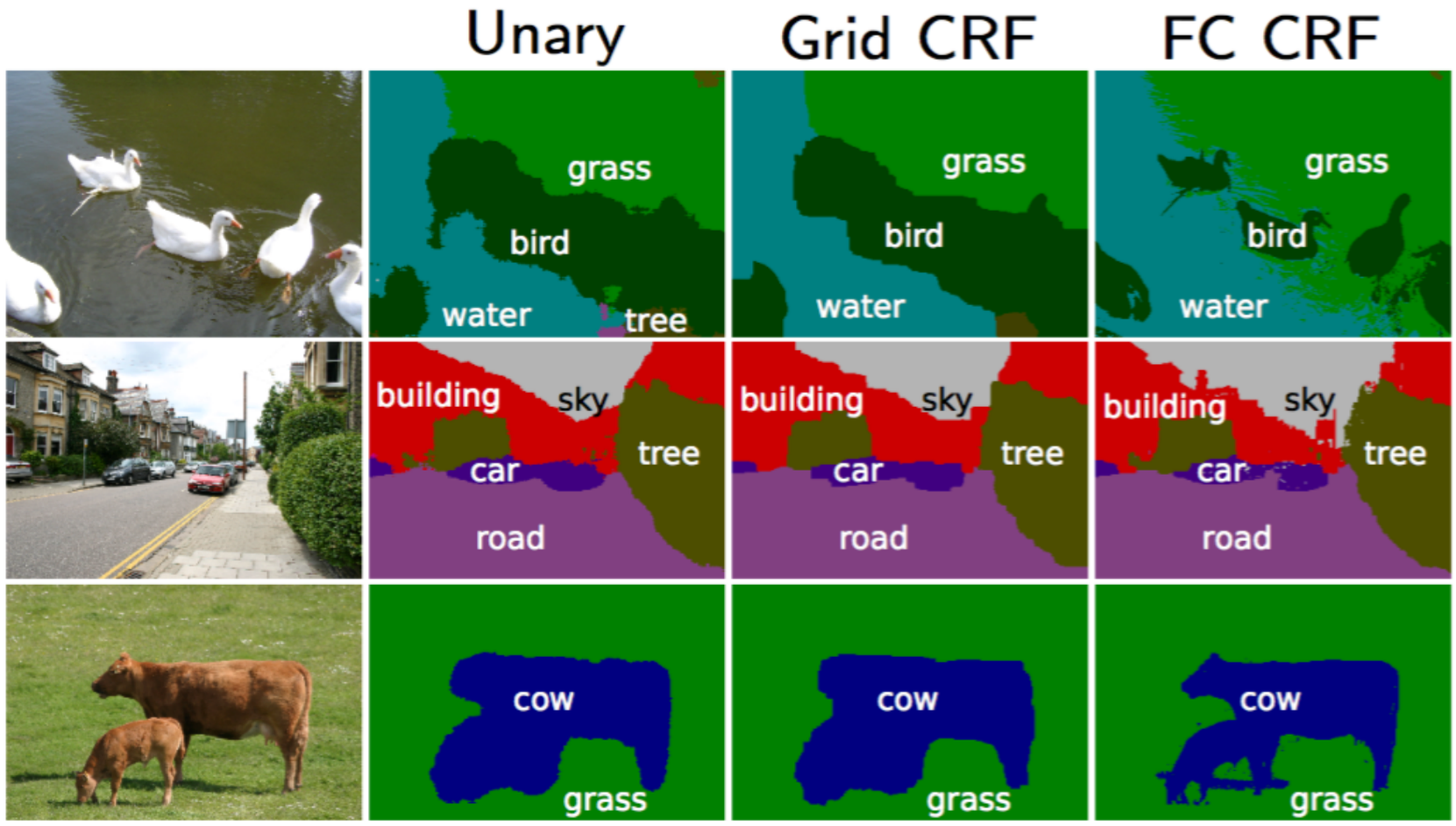


(b) Distributions $Q(X_i = \text{“bird”})$ (top) and $Q(X_i = \text{“sky”})$ (bottom)

10 Iterations own 0.2 seconds

[Krähenbühl & Koltun, 2011]

Efficient inference in dense CRFs



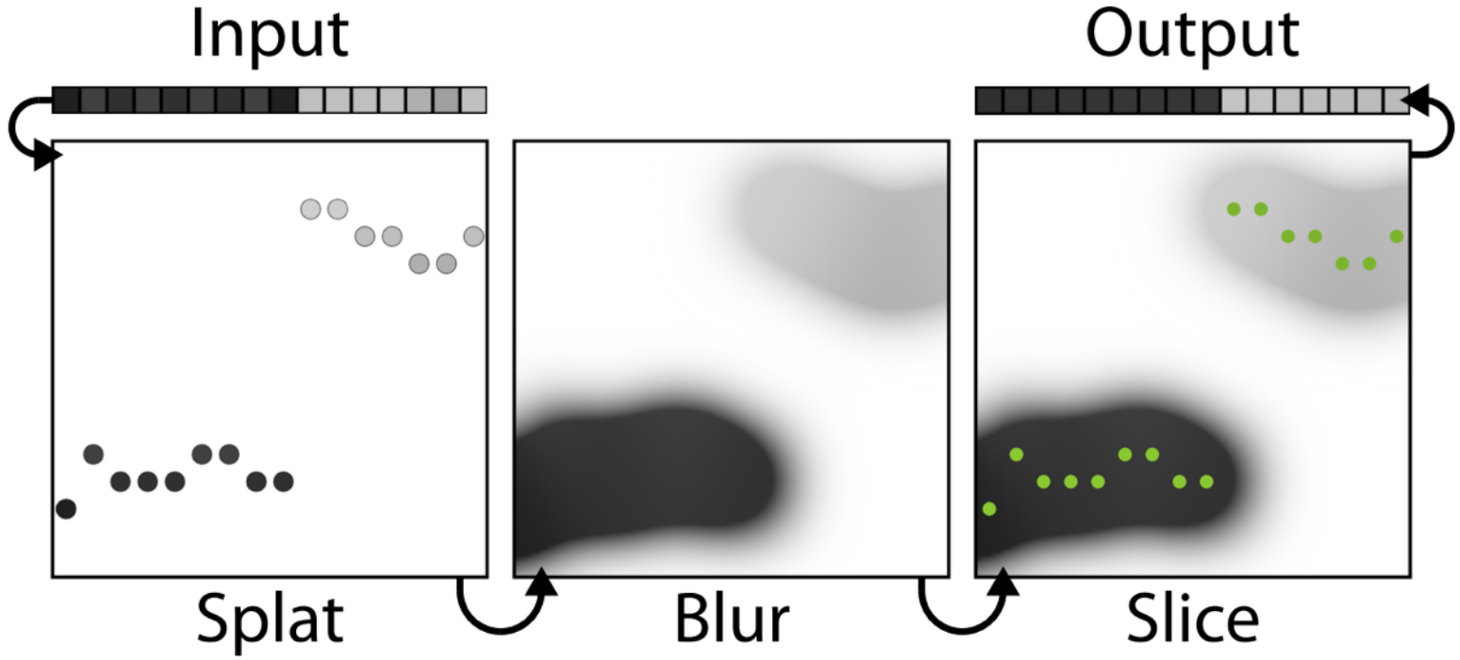
	Time	Global	Avg
Unary	-	84.0	76.6
Grid CRF	1s	84.6	77.2
FC CRF	**0.2s**	**86.0**	**78.3**

[Krähenbühl & Koltun, 2011]

What are these features?

$$k_m\left(\mathbf{f}_i, \mathbf{f}_j\right) = \exp\left(-\frac{1}{2}\left(\mathbf{f}_i - \mathbf{f}_j\right)^{\mathsf{T}} \mathbf{Q}_m \left(\mathbf{f}_i - \mathbf{f}_j\right)\right)$$

$$\mathbf{f}_i = \begin{pmatrix} \mathbf{p}_i \\ \mathbf{x}_i \end{pmatrix}$$



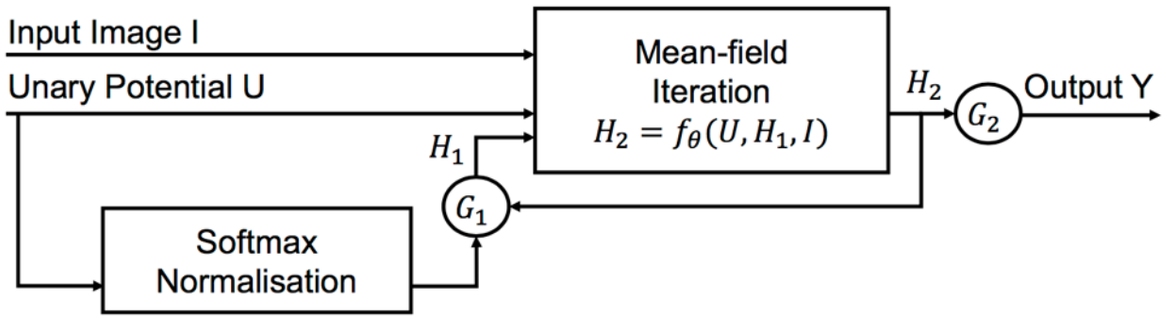
[Adams et al., 2010]

$\mathbf{p}_i$ position vector

$\mathbf{x}_i$ vector of pixel values

- This is known as bilateral filtering
- It is an edge-preserving approach to filtering

CRFs as RNNs

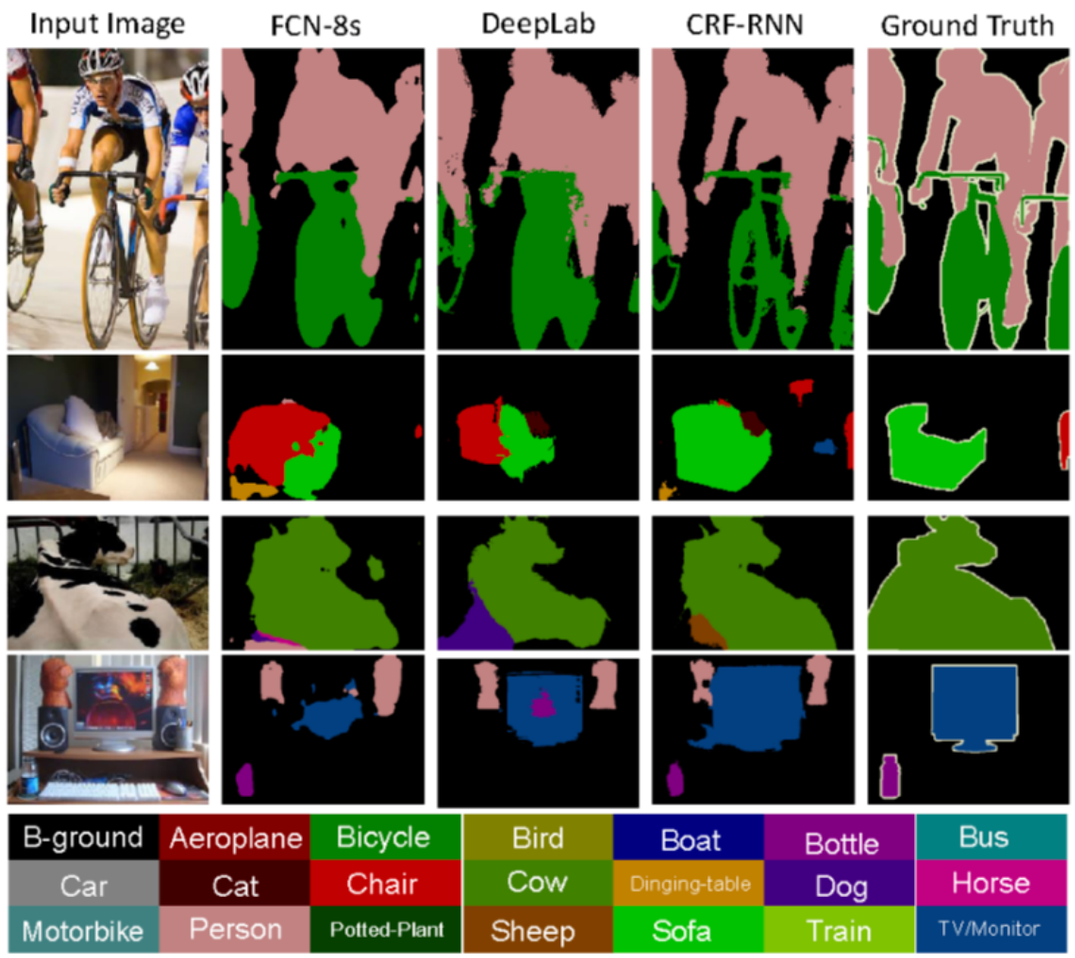


[Zheng et al., 2015]

- Mean field iterations as steps in an RNN
- End-to-end training

PASCAL VOC

Our approach achieved state-of-the-art comparable performance in PASCAL VOC semantic image segmentation dataset, with mean intersection-over-union (IOU) score 74.7% on VOC 2012 test set.

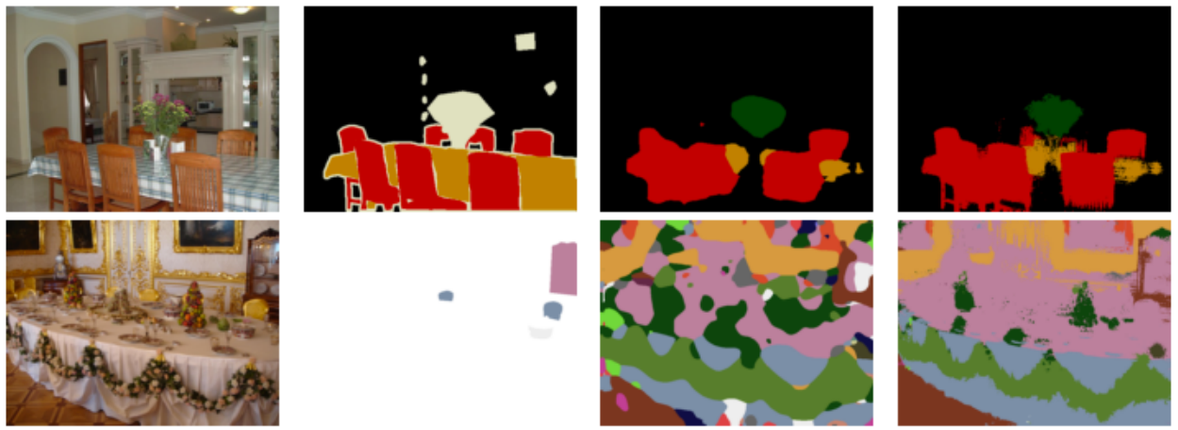


Demo: http://www.robots.ox.ac.uk/~szheng/crfasrnndemo

Learning the filters

	+ MF-1step	+ MF-2 step	+ *loose* MF-2 step
Semantic segmentation (IoU) - CNN [16]: 72.08 / 66.95			
Gauss CRF	+2.48	+3.38	+3.38 / +3.00
Learned CRF	+2.93	+3.71	**+3.85 / +3.37**
Material segmentation (Pixel Accuracy) - CNN [11]: 67.21 / 69.23			
Gauss CRF	+7.91 / +6.28	+9.68 / +7.35	+9.68 / +7.35
Learned CRF	+9.48 / +6.23	+11.89 / +6.93	**+11.91 / +6.93**

Table 2. **Improved mean-field inference with learned potentials.** (top) Average IoU score on Pascal VOC12 validation/test data [20] for semantic segmentation; (bottom) Accuracy for all pixels / averaged over classes on the MINC test data [11] for material segmentation.



(a) Input (b) Ground Truth (c) CNN (d) +*loose*MF

Figure 4. **Segmentation results.** An example result for semantic (top) and material (bottom) segmentation. (c) depicts the unary results before application of MF, (d) after two steps of *loose*-MF with a learned CRF. More examples with comparisons to Gaussian pairwise potentials are in Sec. C.5 and Sec. C.6.

[Kiefel et al., 2015]

Summary

- We can improve performance of segmentation algorithms by adding pairwise interactions between pixel predictions
- Approximate Inference in dense CRFs can be done efficiently
- End-to-end learning can improve performance
- Pairwise kernels can be learned and lead to potential improvements
- These methods could have an impact in other problem domains

References

(1) Adams A, Baek J, Davis MA. Fast high-dimensional filtering using the permutohedral lattice. Comput Graph Forum. 2010;29(2):753-762.

(2) Krähenbühl P, Koltun V. Efficient Inference in Fully Connected CRFs with Gaussian Edge Potentials. In: Advances in Neural Information Processing Systems.; 2011:109-117. http://papers.nips.cc/paper/4296-efficient-inference-in-fully-connected-crfs-with-gaussian-edge-potentials. Accessed January 8, 2016.

(3) Kiefel M, Gehler P V, Kiefel M, Mpg T, Gehler P, Mpg T. Sparse Convolutional Networks using the Permutohedral Lattice. 2015. http://arxiv.org/abs/1503.04949. Accessed January 8, 2016.

(4) Shotton J, Winn J, Rother C, Criminisi A. TextonBoost for Image Understanding: Multi-Class Object Recognition and Segmentation by Jointly Modeling Texture, Layout, and Context. Int J Comput Vis. 2009. http://research.microsoft.com/apps/pubs/default.aspx?id=117885.

(5) Zheng S, Jayasumana S, Romera-Paredes B, et al. Conditional Random Fields as Recurrent Neural Networks. In: International Conference on Computer Vision (ICCV).; 2015:16. http://arxiv.org/abs/1502.03240. Accessed January 8, 2016.