Stat 212b: Topics in Deep Learning
Lecture 9

Joan Bruna
UC Berkeley

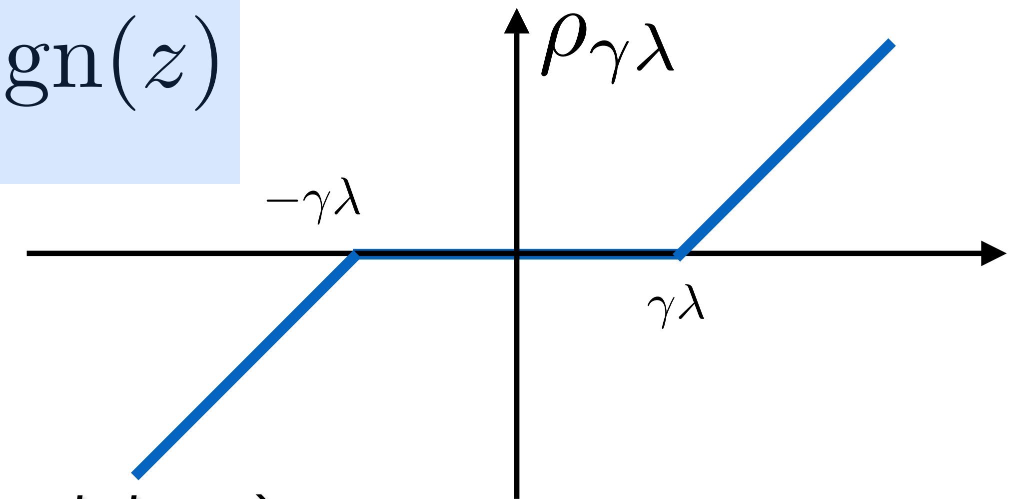
Berkeley
UNIVERSITY OF CALIFORNIA

Review: Proximal Splitting and ISTA

- When $h_2(z) = \lambda \|z\|_1$, the proximal operator becomes

$$\text{prox}_{\gamma h_2}(z) = \max(0, |z| - \gamma\lambda) \cdot \text{sign}(z)$$

$\rho_{\gamma\lambda}$: *soft thresholding*



- ISTA algorithm (*iterative soft thresholding*):

$$z_{n+1} = \text{prox}_{\gamma_n h_2}(z_n - \gamma_n \nabla h_1(z_n))$$
$$\nabla h_1(z_n) = -D^T(x - Dz_n)$$

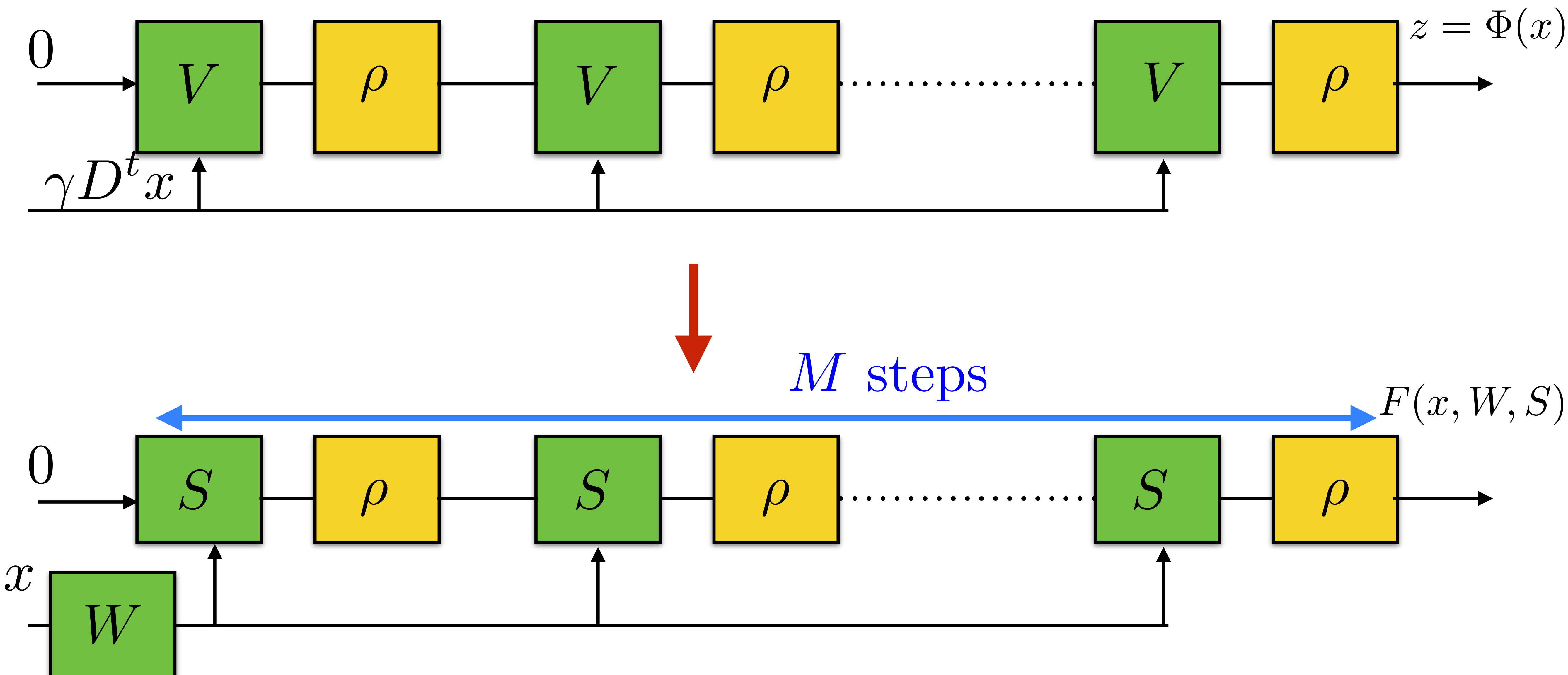
$$z_{n+1} = \rho_{\gamma\lambda}((\mathbf{1} - \gamma D^T D) z_n + \gamma D^T x)$$

  - converges in sublinear time $O(1/n)$ if $\gamma \in (0, 1/\|D^T D\|)$

- FISTA [*Beck and Teboulle,'09*]:
  - adds Nesterov momentum.
  - proven accelerated convergence $O(1/n^2)$

Review:From supervised Lasso to DNNs

- The Lasso (sparse coding operator) can be implemented as a specific deep network
- Can we accelerate the sparse inference with a shallower network, with trained parameters?



Objectives

- Random Forests and DNNs.
- Deformable Parts Model and CNNs.
- Structured Output Prediction
  - Graph Transformer Networks
  - CRFs and MRFs.
  - Examples: Detection, Segmentation, Pose Estimation.
- Embeddings
- Extensions to non-Euclidean domains
  - Spectral Networks
  - Spatial Transformer Networks

Review: Decision Trees

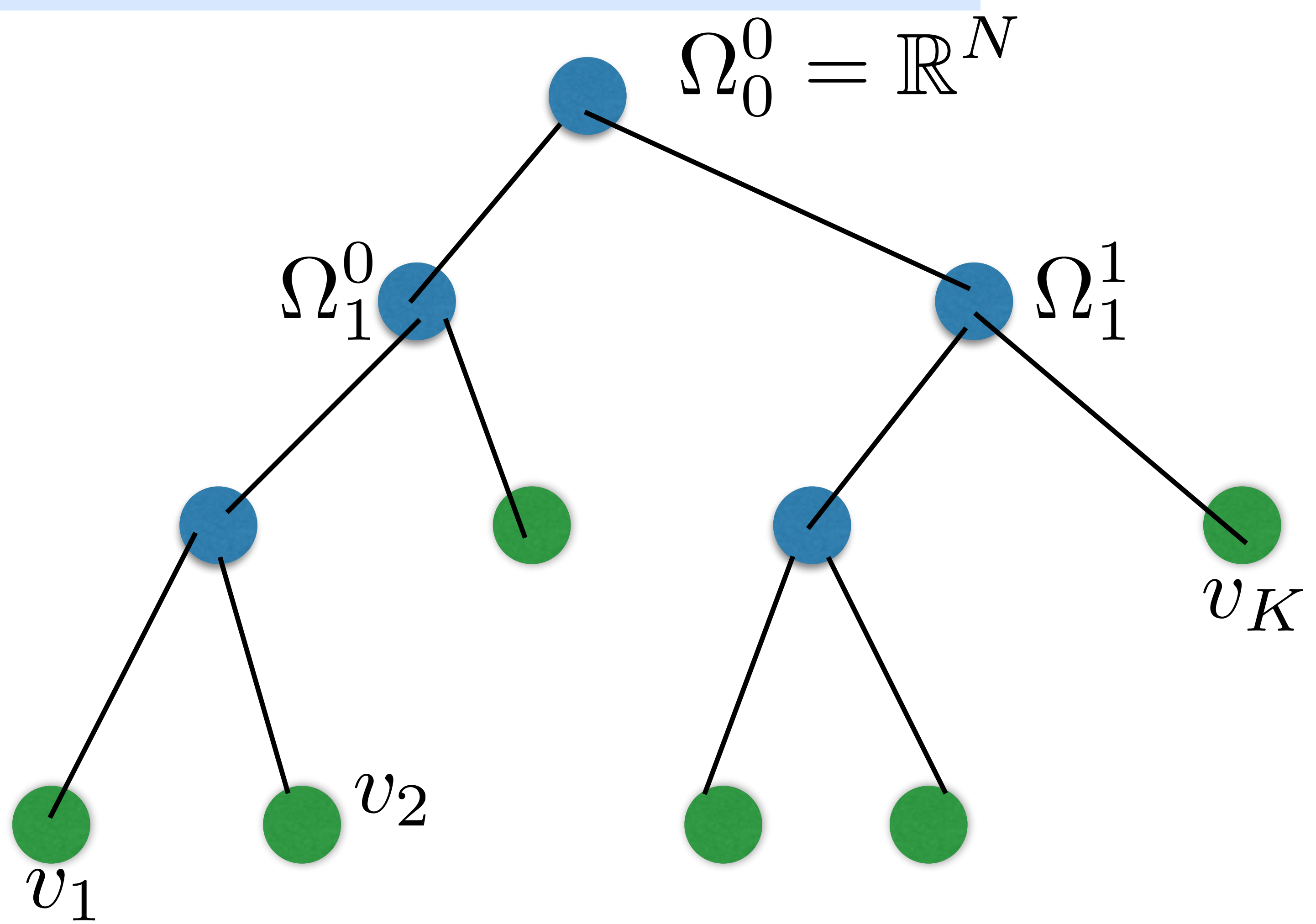
- Let $x = \{(x_1, y_1), \ldots, (x_T, y_T)\}$ be the input data, with $x_i \in \mathbb{R}^N$.
- Each node of the three selects a variable and splits using a threshold.

$$\Omega_i^j \subset \mathbb{R}^N \rightarrow (\Omega_{i+1}^l, \Omega_{i+1}^{l+1})$$
$$\Omega_i^j = \Omega_{i+1}^l \cup \Omega_{i+1}^{l+1} \;,\; \emptyset = \Omega_{i+1}^l \cap \Omega_{i+1}^{l+1}$$

Review: Decision Trees

- Let $x = \{(x_1, y_1), \ldots, (x_T, y_T)\}$ be the input data, with $x_i \in \mathbb{R}^N$.
- Each node of the three selects a variable and splits using a threshold.

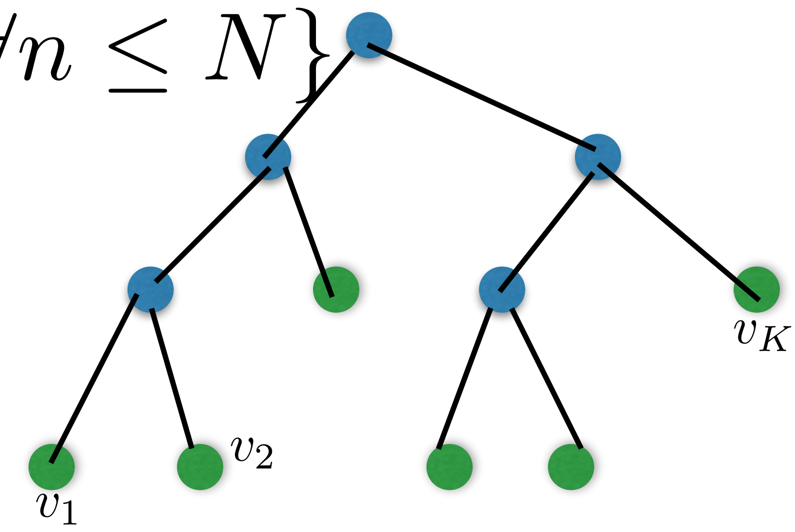
$$\Omega_i^j \subset \mathbb{R}^N \rightarrow (\Omega_{i+1}^l, \Omega_{i+1}^{l+1})$$
$$\Omega_i^j = \Omega_{i+1}^l \cup \Omega_{i+1}^{l+1} \ , \ \emptyset = \Omega_{i+1}^l \cap \Omega_{i+1}^{l+1}$$



Decision Trees

The leaves $\{v_k\}$ of the tree define a partition of the input space into cubic sections:

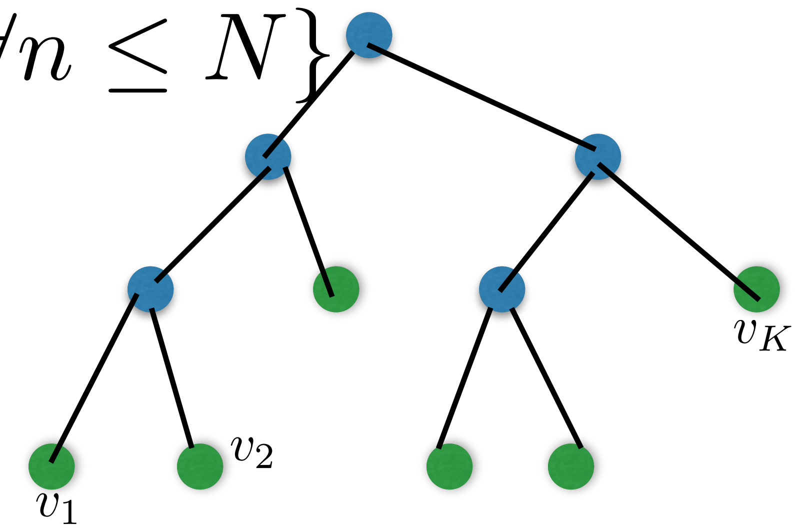
$$\Omega_\infty^k = \{x \in \mathbb{R}^N \, ; \, \alpha_{k,n} \leq x_n \leq \beta_{k,n} \; \forall n \leq N\}$$



Decision Trees

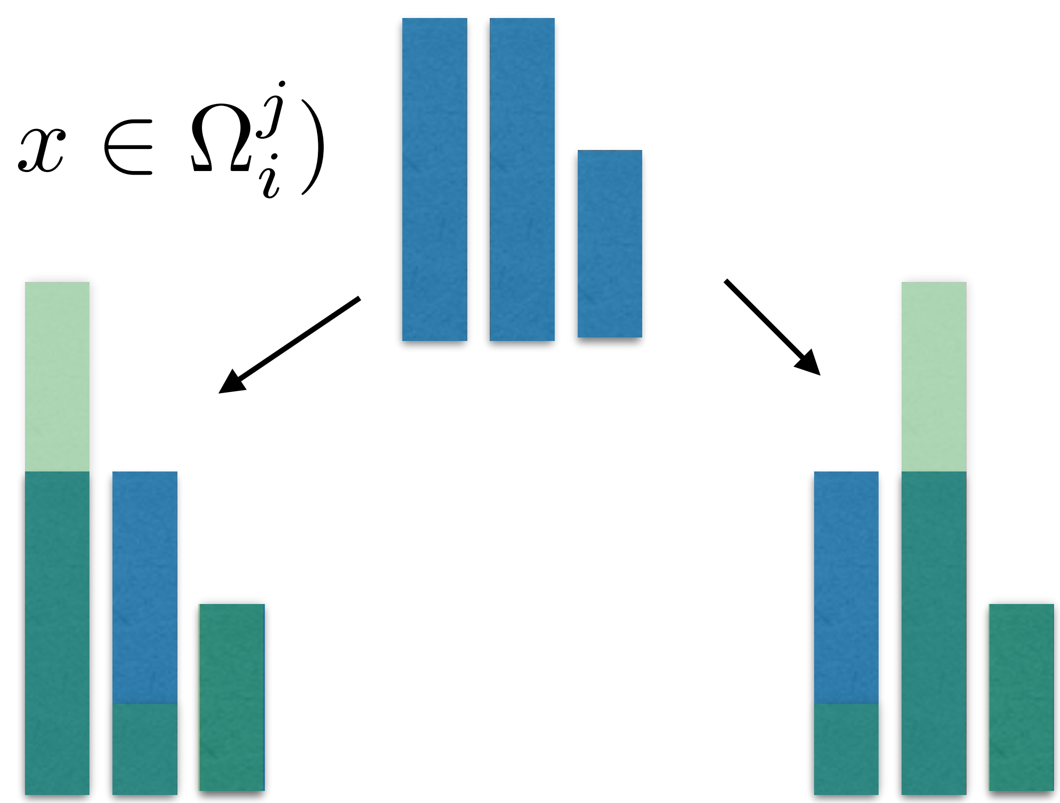
The leaves $\{v_k\}$ of the tree define a partition of the input space into cubic sections:

$$\Omega^k_\infty = \{x \in \mathbb{R}^N \,;\, \alpha_{k,n} \leq x_n \leq \beta_{k,n} \; \forall n \leq N\}$$



- Each split optimizes the entropy in the label distribution:

$p(y \mid x \in \Omega_i^j)$



$p(y \mid x \in \Omega_{i+1}^l)$

$p(y \mid x \in \Omega_{i+1}^{l+1})$

Random Forests

- A decision tree can capture interactions between different variables, but it is very noisy (ie unstable).
- Evaluation and training are extremely efficient.

Random Forests

- A decision tree can capture interactions between different variables, but it is very noisy (ie unstable).
- Evaluation and training are extremely efficient.
- By appropriately introducing *randomization,* we can construct an ensemble of random trees: the so-called *random forests.*

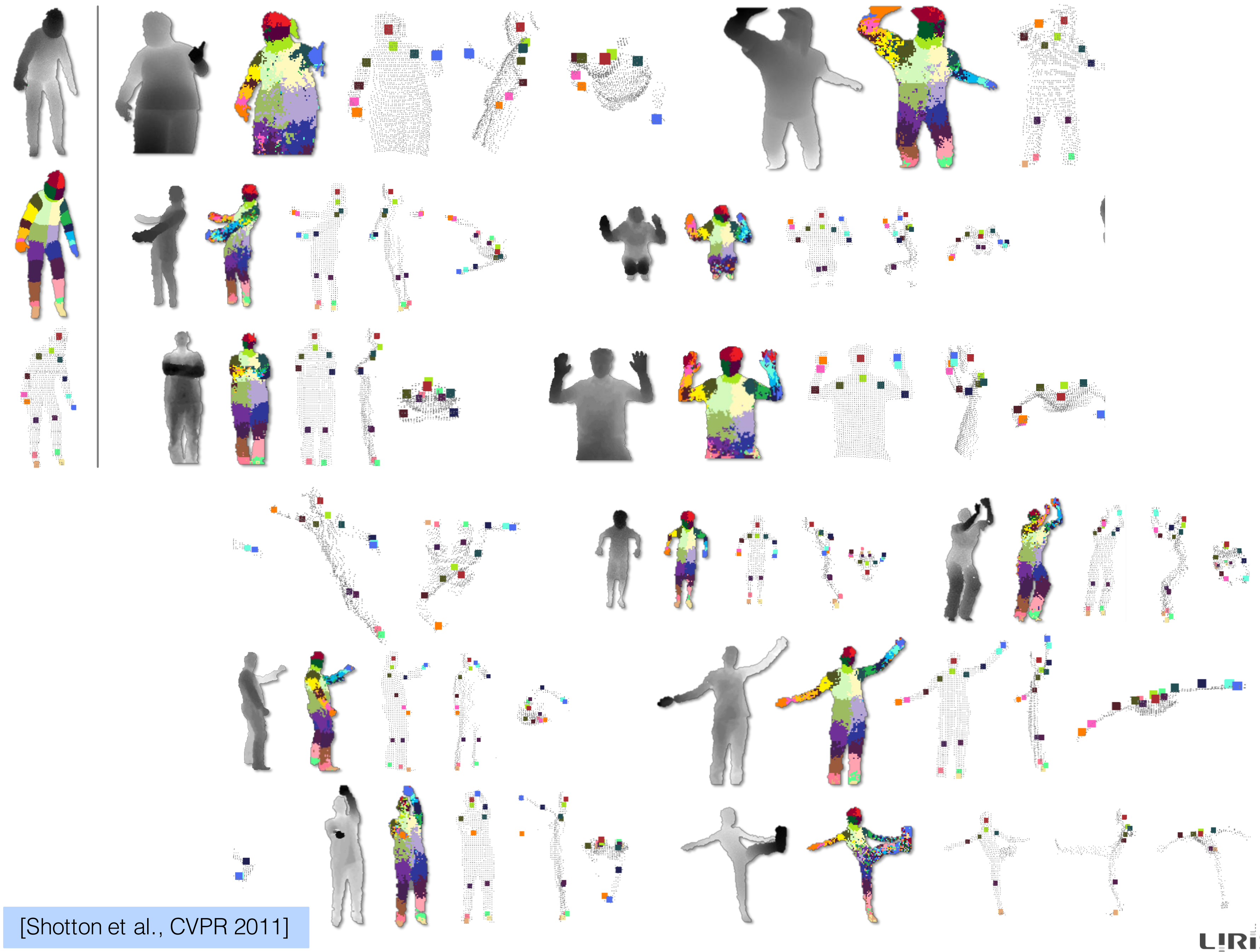
Random Forests

- A decision tree can capture interactions between different variables, but it is very noisy (ie unstable).
- Evaluation and training are extremely efficient.
- By appropriately introducing *randomization*, we can construct an ensemble of random trees: the so-called *random forests*.

- We draw bootstrapped samples of the training set, and each split in the tree is calculated *only on a small random subset of variables* (typically of size $O(\sqrt{N})$).
- The prediction is the aggregate prediction (ie voting) of each tree.

Random Forests

- Successful across a wide range of classification and regression problems.



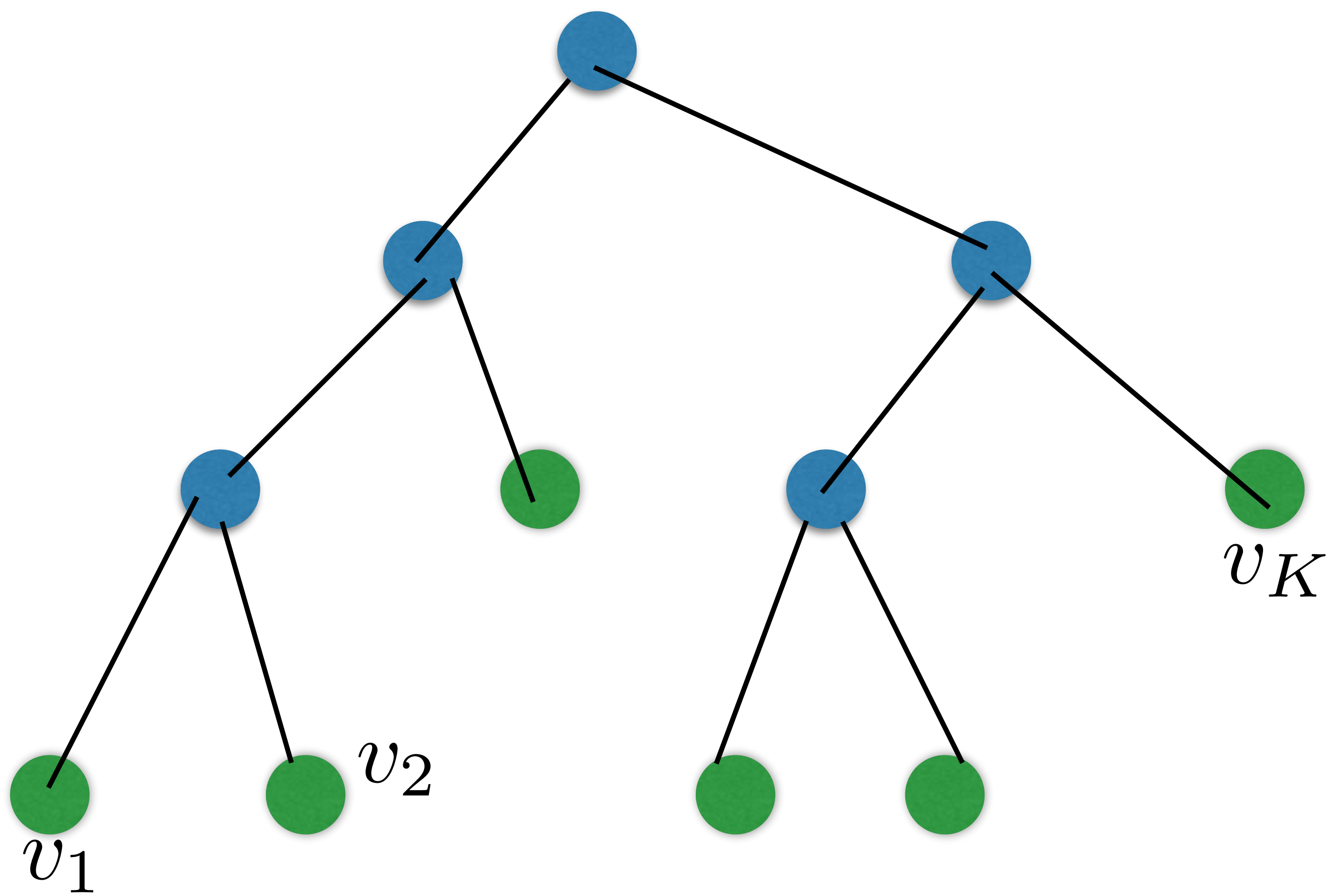
Real-Time Pose Estimation from Kinect measurements (CVPR'11)

LIRIS

(figure from Ch. Wolf slides)

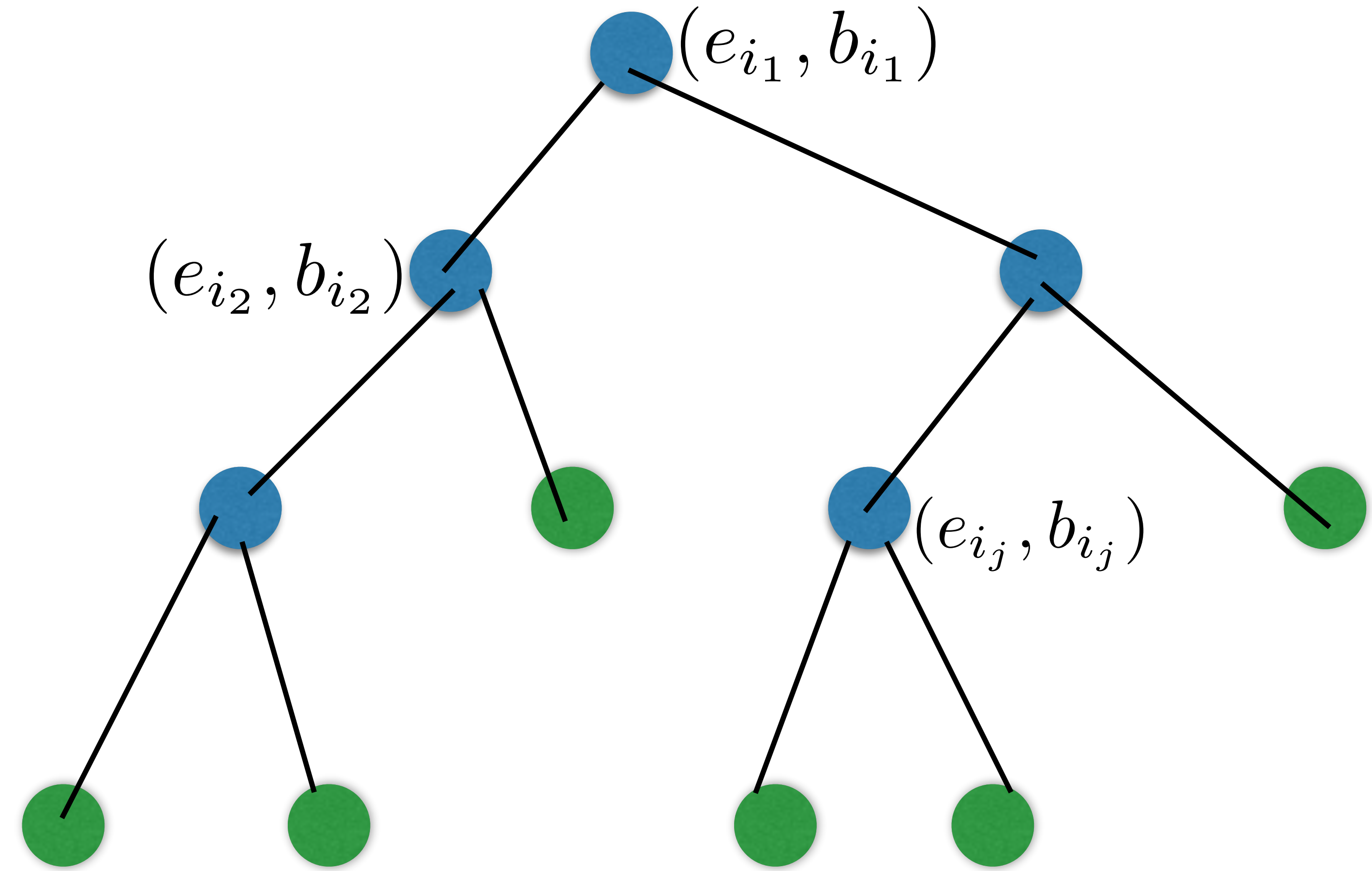
Embed RF into Deep Neural Networks?

- Q: How to write a Decision Tree in terms of a network?



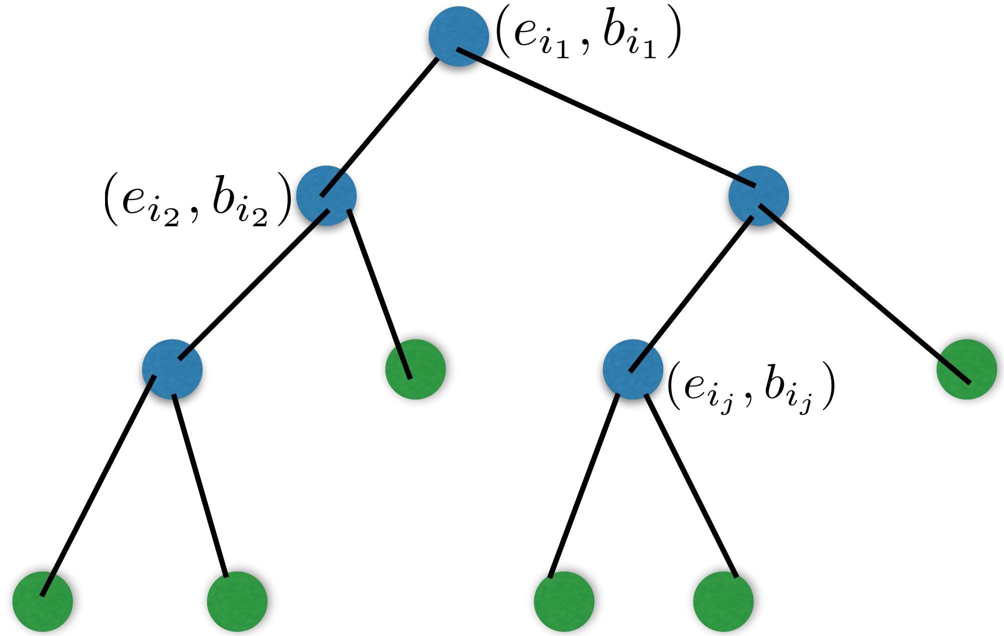
Embed RF into Deep Neural Networks?

- Q: How to write a Decision Tree in terms of a network?



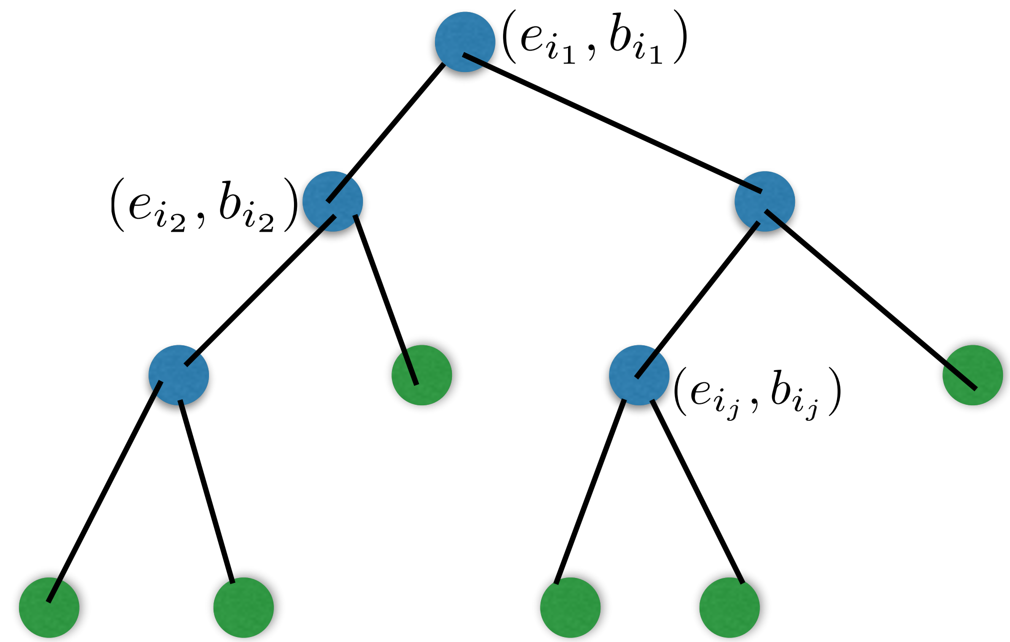
- Each node in the tree amounts to a comparison of the form $\langle x, e_{i_j} \rangle \leq b_{i_j}$

Embed RF into Deep Neural Networks?



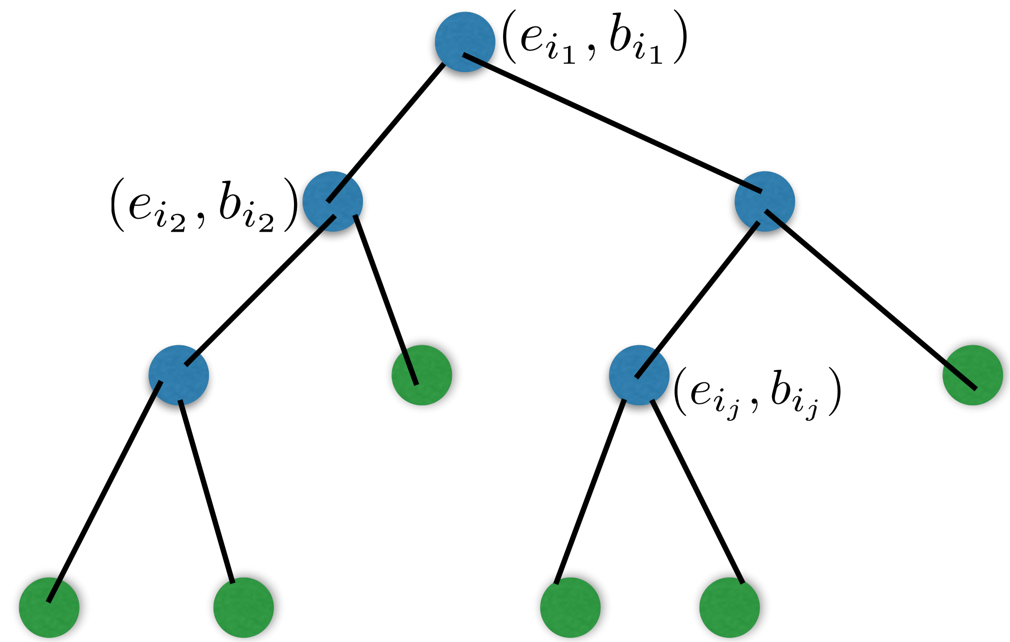
- Let $W = (e_{i_1}, e_{i_2}, \ldots, e_{i_S}) \in \mathbb{R}^{S \times N}$ (S: number of nodes)
- Let $b = (b_{i_1}, b_{i_2}, \ldots, b_{i_S}) \in \mathbb{R}^{S}$ (L: number of leaves)

Embed RF into Deep Neural Networks?



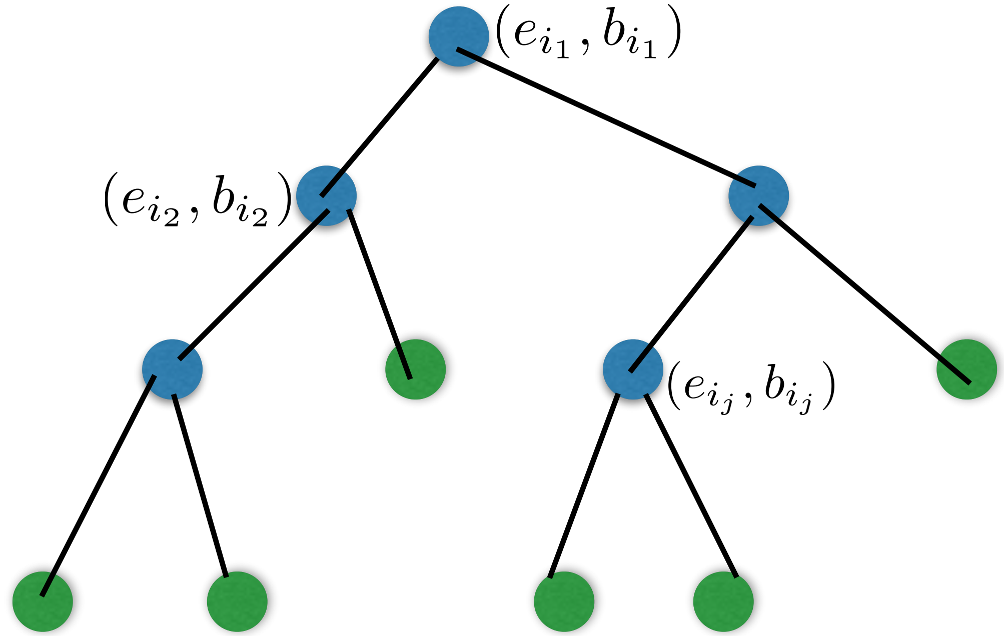
- Let $W = (e_{i_1}, e_{i_2}, \ldots, e_{i_S}) \in \mathbb{R}^{S \times N}$ (S: number of nodes)
- Let $b = (b_{i_1}, b_{i_2}, \ldots, b_{i_S}) \in \mathbb{R}^S$ (L: number of leaves)
- Let $y = \text{sign}(Wx + b)$
- For each leaf l, let $v_l = (\pm 1, \ldots, \pm 1)$ encoding the tree path, and $V = (v_1, \ldots, v_l) \in \mathbb{R}^{L \times S}$.

Embed RF into Deep Neural Networks?



- Let $W = (e_{i_1}, e_{i_2}, \ldots, e_{i_S}) \in \mathbb{R}^{S \times N}$ (S: number of nodes)
- Let $b = (b_{i_1}, b_{i_2}, \ldots, b_{i_S}) \in \mathbb{R}^S$ (L: number of leaves)
- Let $y = \text{sign}(Wx + b)$
- For each leaf l, let $v_l = (\pm 1, \ldots, \pm 1)$ encoding the tree path, and $V = (v_1, \ldots, v_l) \in \mathbb{R}^{L \times S}$.
- Let $\tilde{b} = (k_1, \ldots, k_L)$ the depth of each leaf.

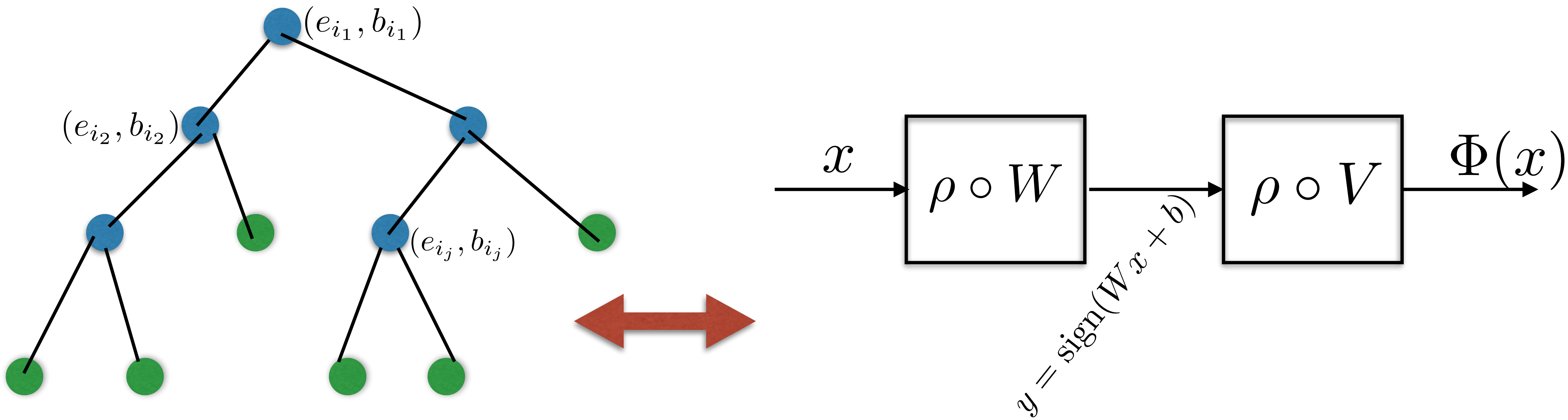
Embed RF into Deep Neural Networks?



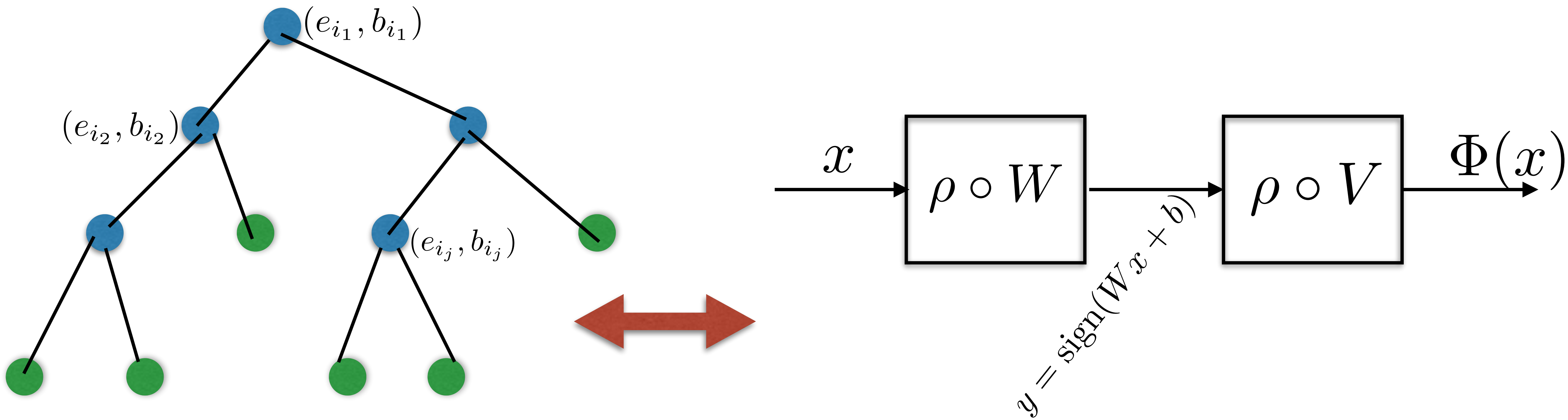
- Let $W = (e_{i_1}, e_{i_2}, \ldots, e_{i_S}) \in \mathbb{R}^{S \times N}$ (S: number of nodes)
- Let $b = (b_{i_1}, b_{i_2}, \ldots, b_{i_S}) \in \mathbb{R}^{S}$ (L: number of leaves)
- Let $y = \text{sign}(Wx + b)$
- For each leaf l, let $v_l = (\pm 1, \ldots, \pm 1)$ encoding the tree path, and $V = (v_1, \ldots, v_l) \in \mathbb{R}^{L \times S}$.
- Let $\tilde{b} = (k_1, \ldots, k_L)$ the depth of each leaf.

Let $\Phi(x) = \text{sign}(Vy - \tilde{b})$

Embed RF into Deep Neural Networks?



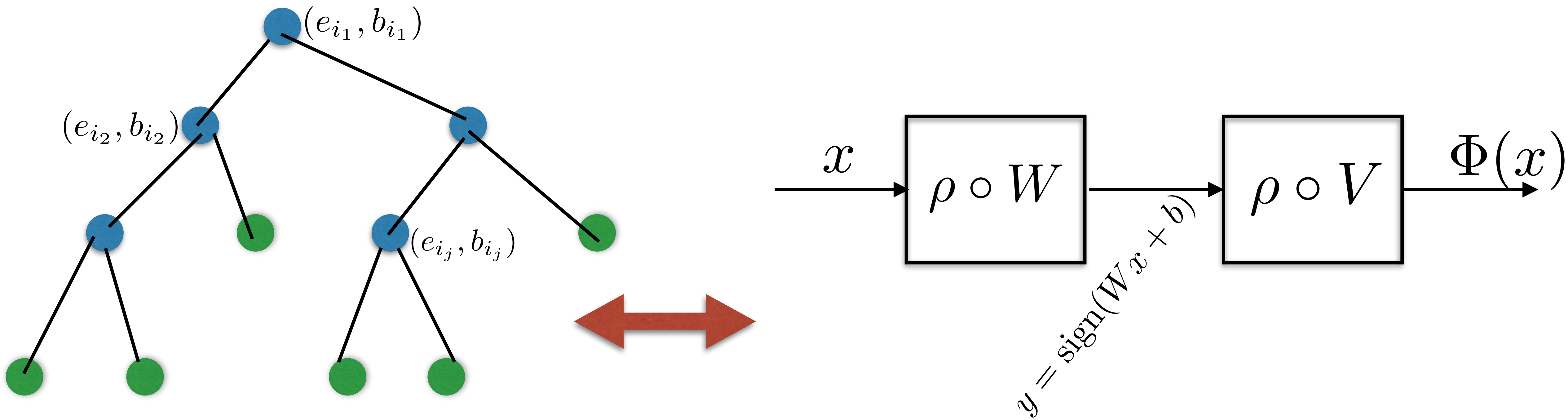
Embed RF into Deep Neural Networks?



- $\Phi(x) \in \mathbb{R}^L$ is a *one-hot* vector encoding $x \in \Omega_\infty^l$, $l \leq L$.
  $(x \in \Omega_\infty^l \Leftrightarrow \Phi(x)_l = 1, \Phi(x)_k = 0\,, k \neq l)$

- A decision tree can thus be thought as a special two-layer network.

Embed RF into Deep Neural Networks?



- $\Phi(x) \in \mathbb{R}^L$ is a *one-hot* vector encoding $x \in \Omega_\infty^l$, $l \leq L$. $\left(x \in \Omega_\infty^l \Leftrightarrow \Phi(x)_l = 1, \Phi(x)_k = 0\,, k \neq l\right)$
- The Random Forest is obtained with an ensemble of two-layer networks.
- Training is radically different: greedy in RF versus gradient descent in Deep Learning.

Random Forests and CNNs

- Random Forests thus also consider piecewise linear regions of the input space.

- However the encoding of these regions is different from that of a deep ReLU network.

- Computationally more efficient
- No gradient descent training
- Less expressive

Random Forests and CNNs

- Random Forests thus also consider piecewise linear regions of the input space.

- However the encoding of these regions is different from that of a deep ReLU network.

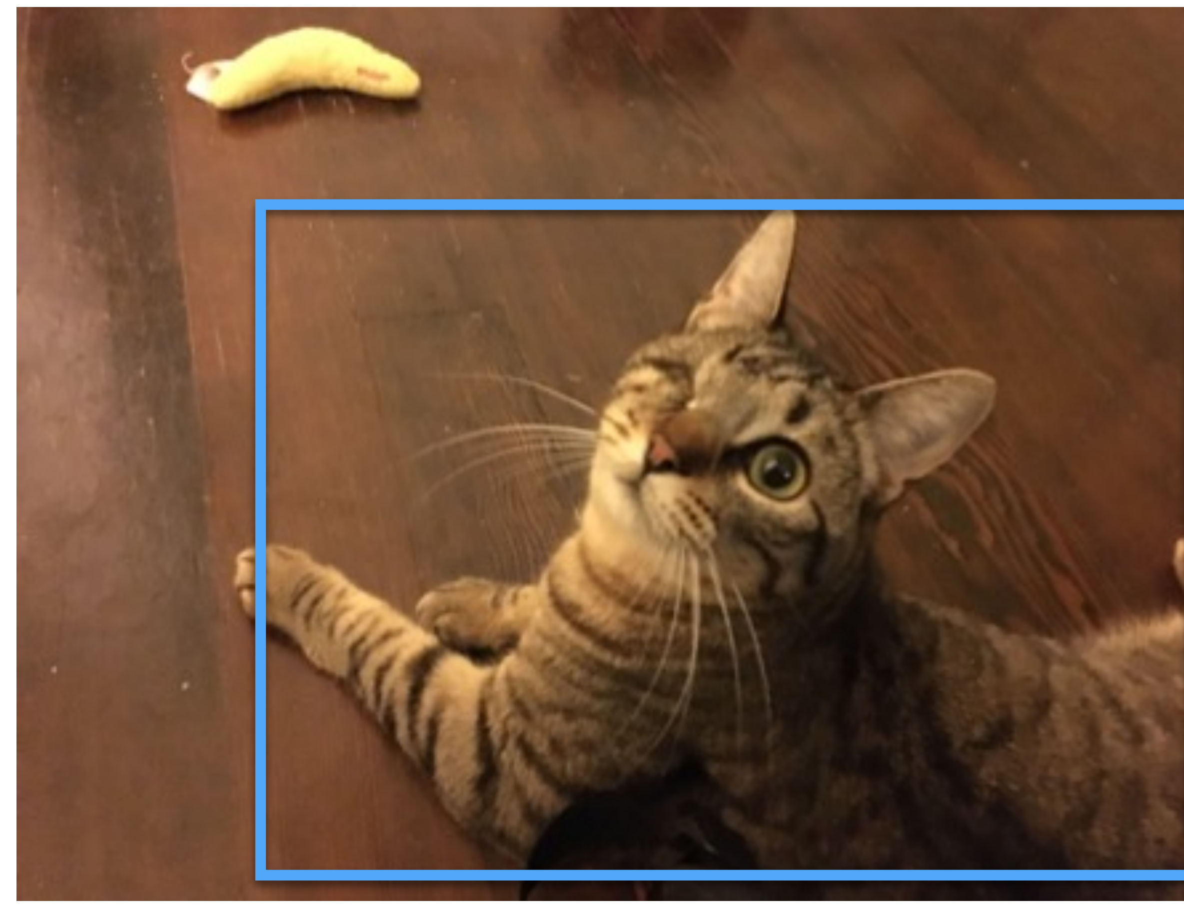
- Computationally more efficient
- No gradient descent training
- Less expressive
- One can also combine both models (eg "Deep Neural Decision Forests", [Kontscheider et al, MSR,'15]).

Beyond Object Classification

classification

"cat"

classification and localization



"cat"

single object problems

Beyond Object Classification

object detection

{"bicycle", "man", "woman", "house", "house", "house"}

semantic segmentation

multiple object problems

Deformable Parts Model

[Fischler & Elschlager '73, Felzenszwalb & Huttenlocher '00]

- It is a graphical model with two key components:
  - *Parts:* local structures or templates.
  - *Springs:* Connections between parts encoding our geometric prior.

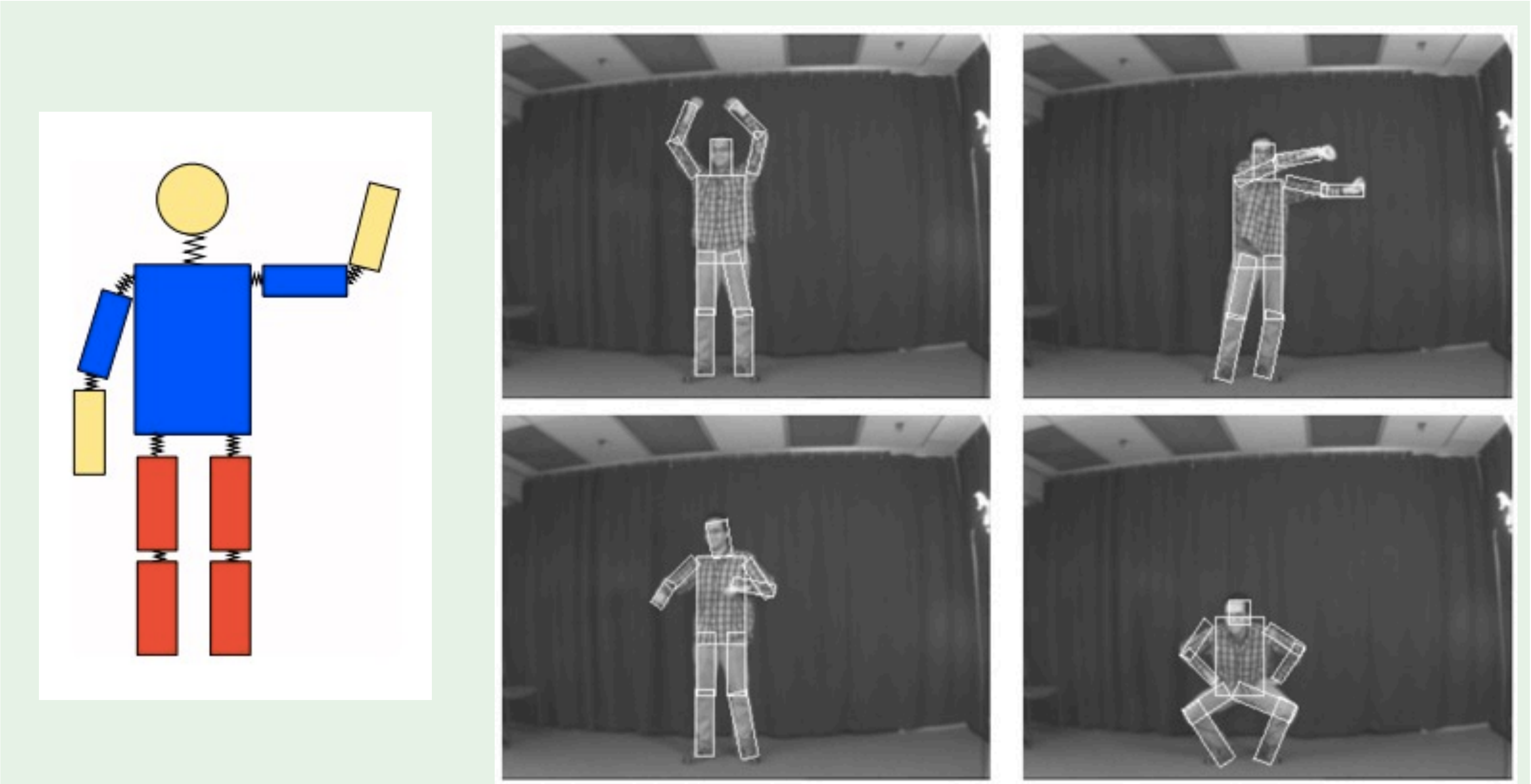
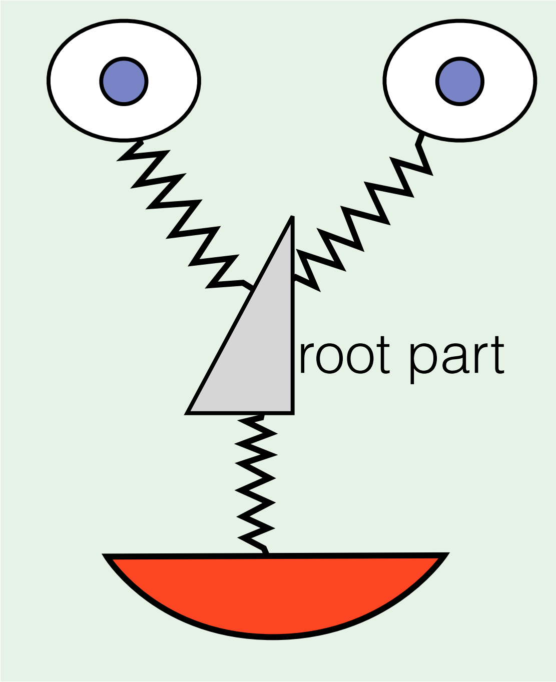
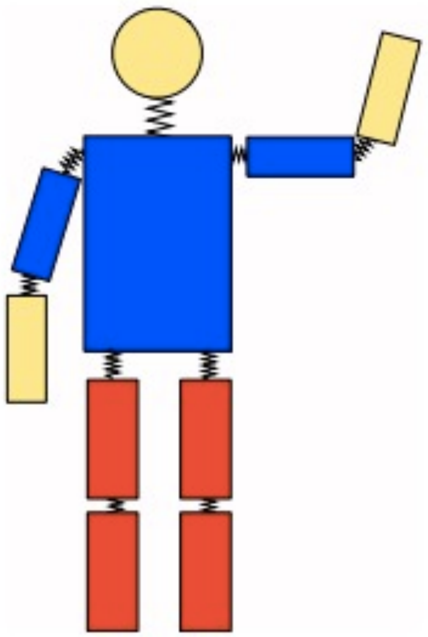


Image: [Felzenszwalb and Huttenlocher 05]

(figure credit: Ross Girshick)

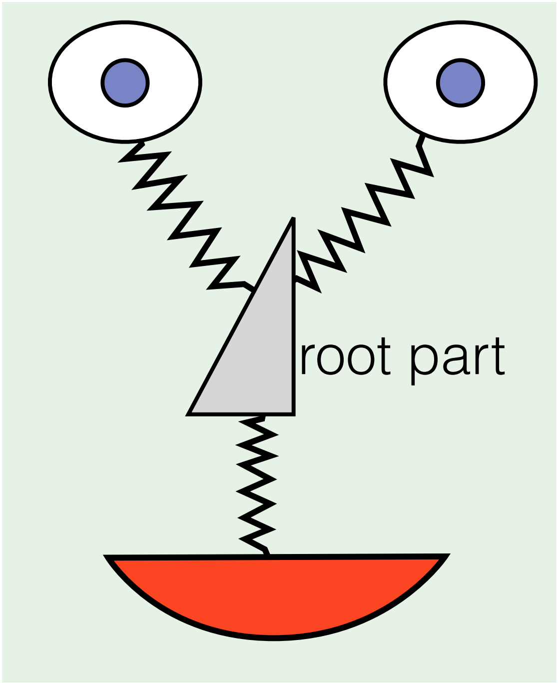
Deformable Parts Model

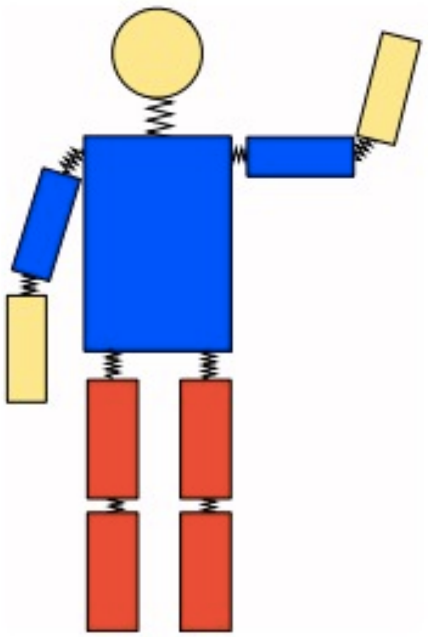




- Intuition:
  - Modeling each subpart is easier than modeling whole objects because they are shared across different instances.

Deformable Parts Model

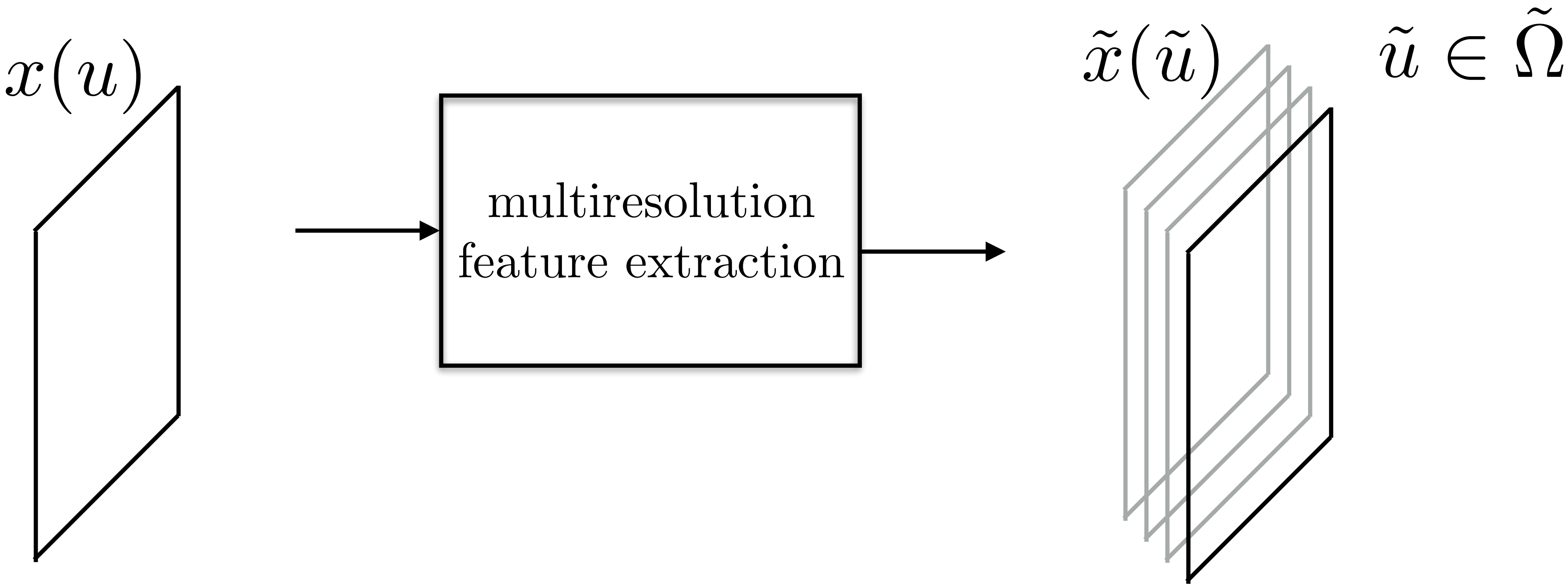




- Intuition:
  - Modeling each subpart is easier than modeling whole objects because they are shared across different instances.
  - The model also needs to capture the typical deformation between parts.
  - Parts can be either localized in space or global if extracted from low-frequency measurements (MultiResolution Analysis such as Laplacian Pyramid).

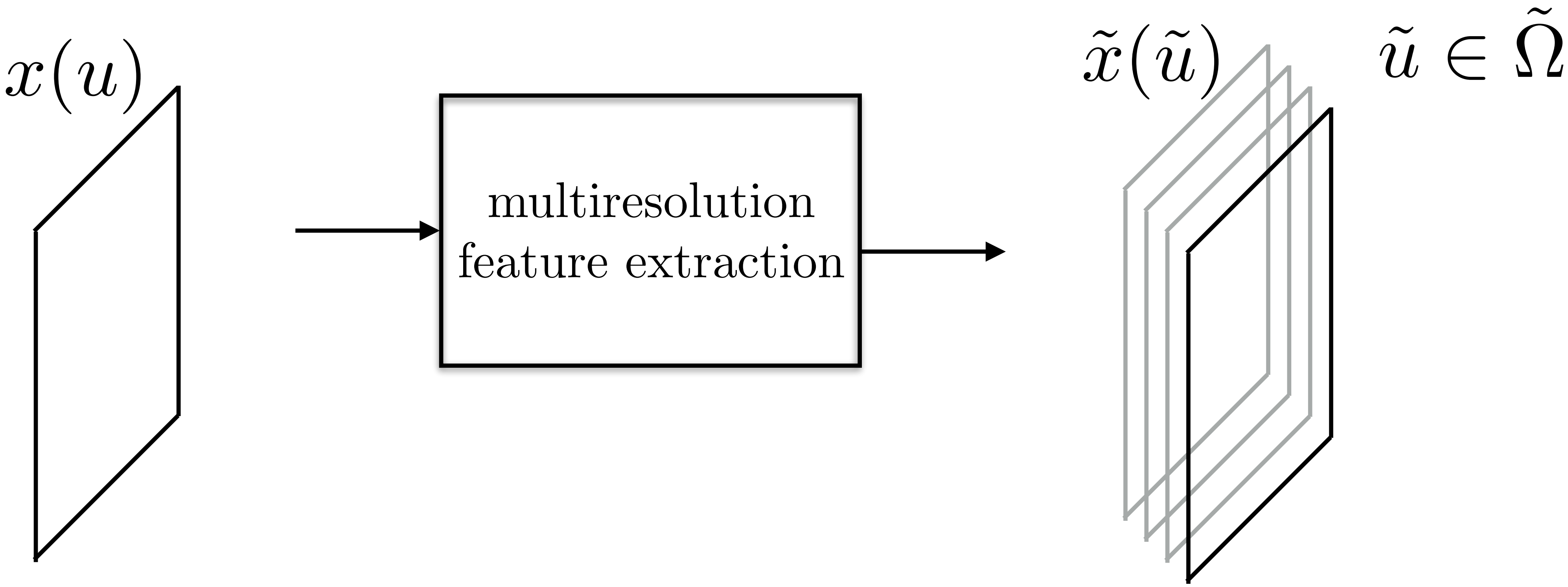
Deformable Parts Model

- Typically we start by extracting multi-resolution features, either by a predefined transform (such as SIFT, HoG, Scattering) or using CNN features:



Deformable Parts Model

- Typically we start by extracting multi-resolution features, either by a predefined transform (such as SIFT, HoG, Scattering) or using CNN features:



- Deformations on coordinates $\tilde{u}$ model more general transformations (rotations, dilations, appearance, etc.)

Deformable Parts Model

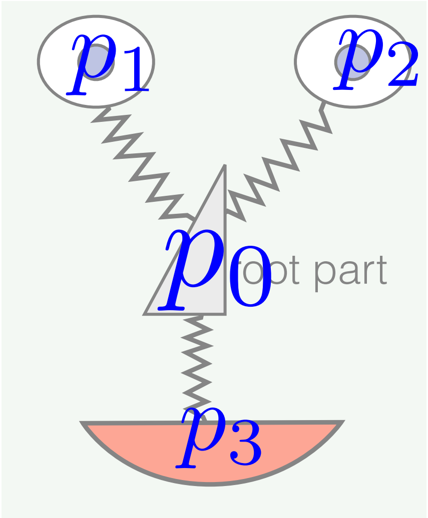
- A DP model for an object with n parts is given by a $(n+2)$-tuple $(F_0, P_1, \ldots, P_n, b)$ where:
  - F_0: *root* filter.
  - b: bias term
  - $P_i = (F_i, v_i, d_i)$ part model, where F_i is the filter for part i, v_i the *anchor* and d_i specifies deformation cost wrt anchor.

Deformable Parts Model

- A DP model for an object with n parts is given by a $(n+2)$-tuple $(F_0, P_1, \ldots, P_n, b)$ where:
  - F_0: *root* filter.
  - b: bias term
  - $P_i = (F_i, v_i, d_i)$ part model, where F_i is the filter for part i, v_i the *anchor* and d_i specifies deformation cost wrt anchor.

- An object hypothesis specifies the locations of root and parts:

$$z = (p_0, \ldots, p_n) \ , \ p_i \in \tilde{\Omega}$$



Deformable Parts Model

- An object hypothesis specifies the locations of root and parts: $z = (p_0, \ldots, p_n)\ ,\ p_i \in \tilde{\Omega}$

- The *score* of an object hypothesis is

$$E(z) = \sum_{i=0}^{n} (F_i \star \tilde{x})(p_i) - \sum_{i=1}^{n} d_i^T \phi(p_i - (p_0 + v_i)) + b\ .$$

$\phi(\tau)$: deformation features (typically first two moments $|\tau|_j$, $|\tau|_j^2$)

Deformable Parts Model

- An object hypothesis specifies the locations of root and parts: $z = (p_0, \ldots, p_n)\ ,\ p_i \in \tilde{\Omega}$

- The *score* of an object hypothesis is

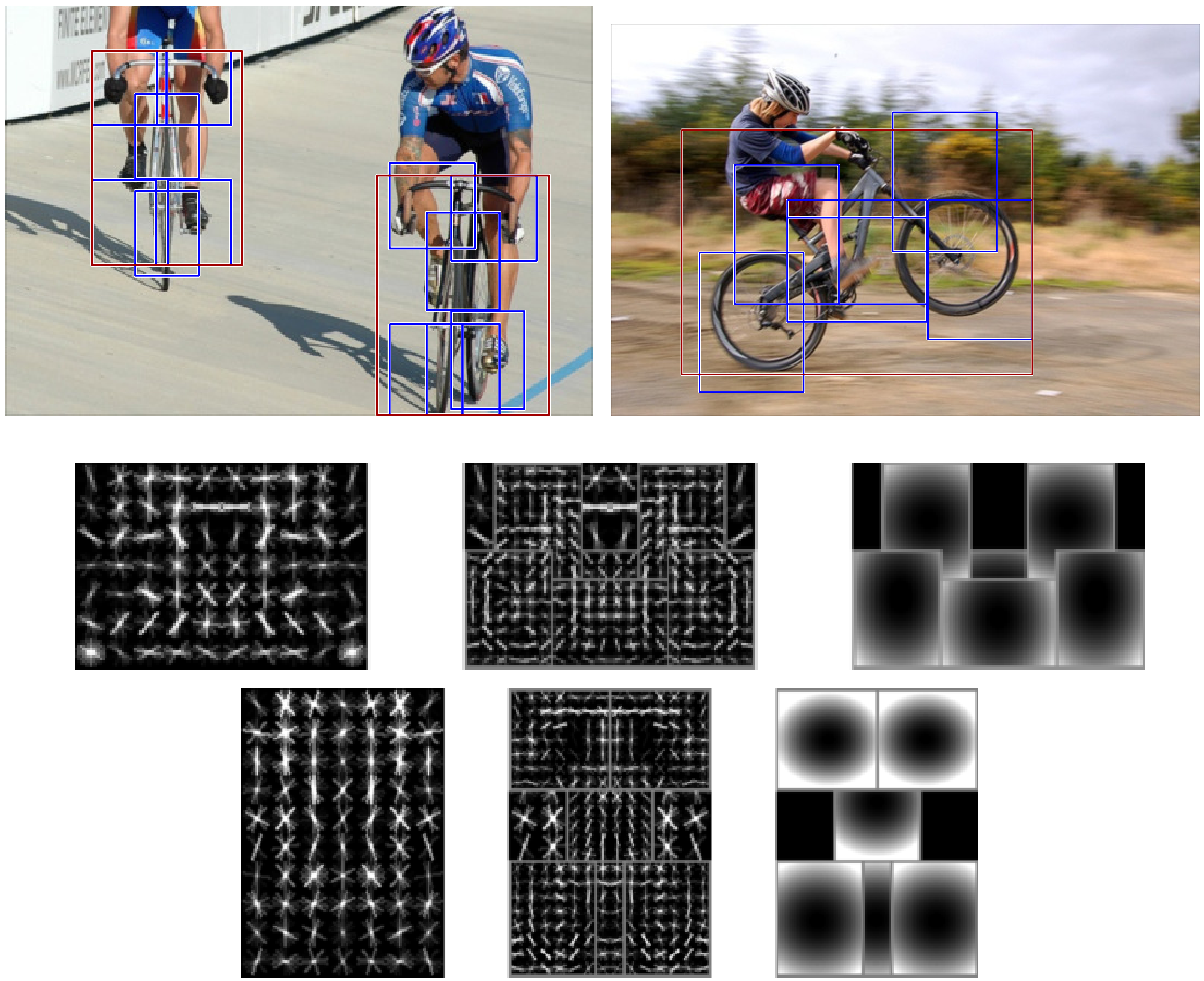
$$E(z) = \sum_{i=0}^{n} (F_i \star \tilde{x})(p_i) - \sum_{i=1}^{n} d_i^T \phi(p_i - (p_0 + v_i)) + b\ .$$

$\phi(\tau)$: deformation features (typically first two moments $|\tau|_j$, $|\tau|_j^2$)

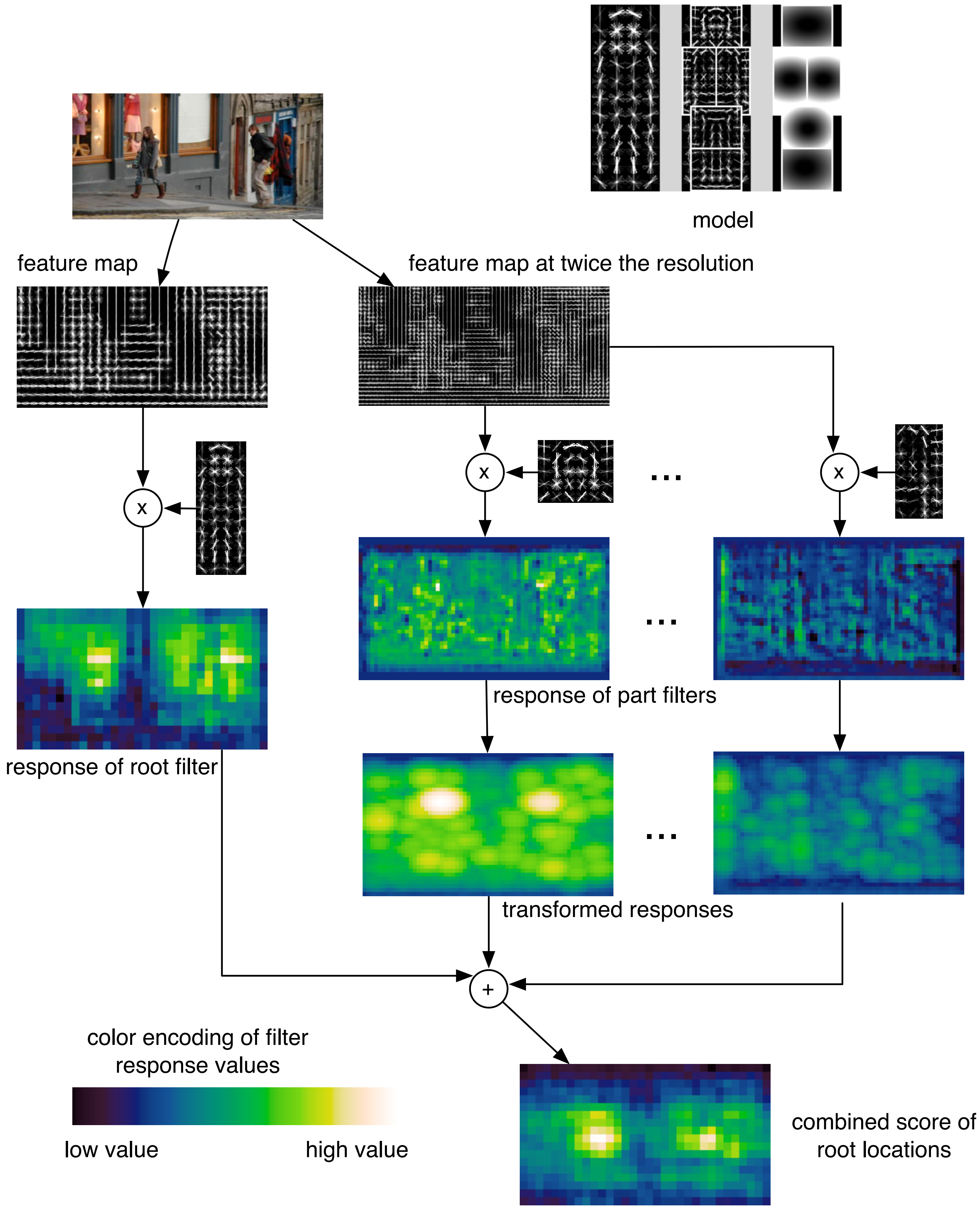
- Efficient implementation using dynamic programming.
- Extension to include mixture models for objects.
- Trained with *Latent* SVM.

Deformable Parts Model

"Object Detection with discriminatively trained Deformable Parts Model", Felzenszwalb, Girshick et al. PAMI'10

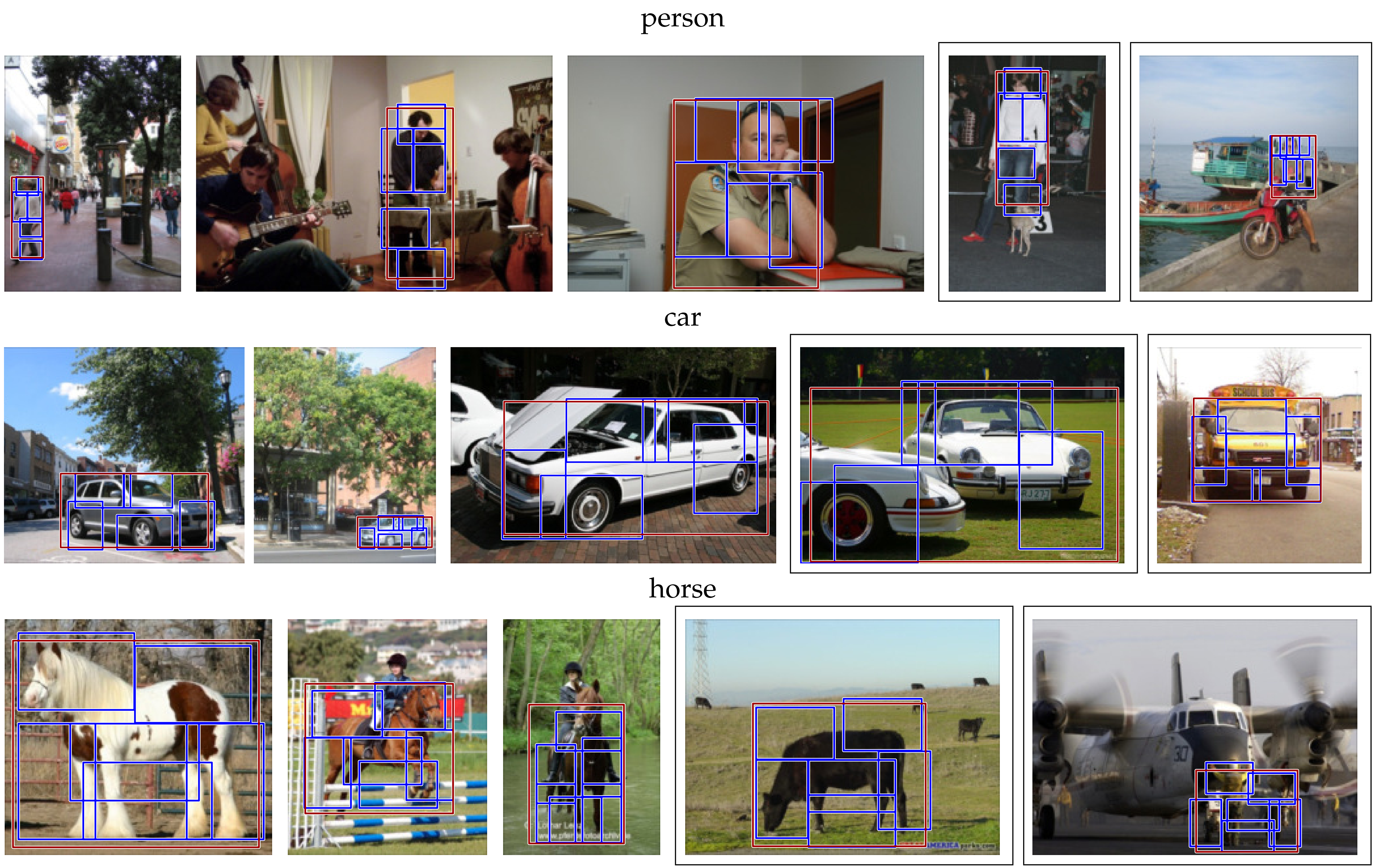


Deformable Parts Model



"Object Detection with discriminatively trained Deformable Parts Model", Felzenszwalb, Girshick et al.'10

Deformable Parts Model



- State-of-the-art in PASCAL object detection pre-2012.
- Model contains "convolutions with penalized offsets".
- Can we relate it to generic CNNs?

DPM and CNNs

- The optimization of part offsets with respect to the anchor is a *distance transform*:

The distance transform of $x : \Omega \to \mathbb{R}$ is a function $D_x : \Omega \to \mathbb{R}$ defined by $D_x(u) = \max_q x(q) - d(u - q)$

DPM: $d(r) = r^T A r + b r$ quadratic form

Max-Pooling: $d(r) = \begin{cases} 0 & \text{if } |r| \leq K \ , \\ \infty & \text{otherwise } . \end{cases}$

DPM and CNNs

- The optimization of part offsets with respect to the anchor is a *distance transform*:

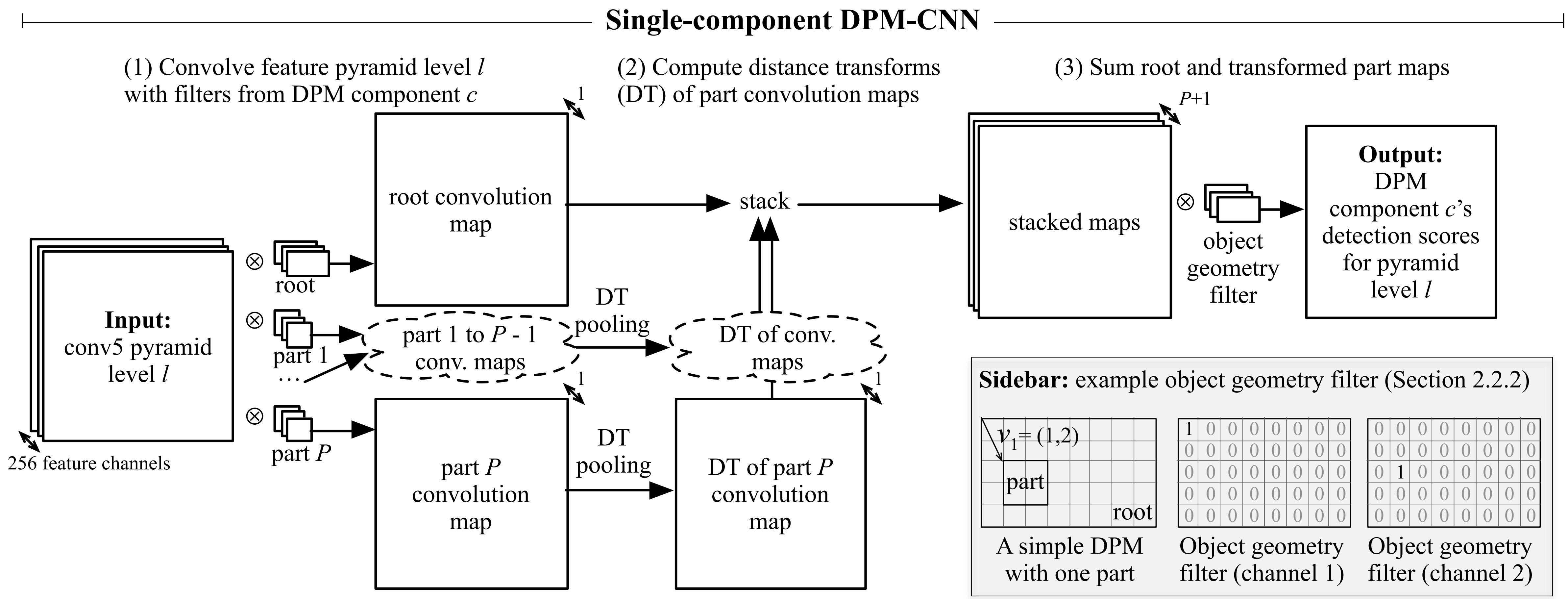
The distance transform of $x : \Omega \to \mathbb{R}$ is a function $D_x : \Omega \to \mathbb{R}$ defined by $D_x(u) = \max_q x(q) - d(u - q)$

$$\text{DPM: } d(r) = r^T A r + b r \text{ quadratic form}$$

$$\text{Max-Pooling: } d(r) = \begin{cases} 0 & \text{if } |r| \leq K \ , \\ \infty & \text{otherwise} \ . \end{cases}$$

- Therefore, one can train a DPM end-to-end as a particular instance of a CNN.

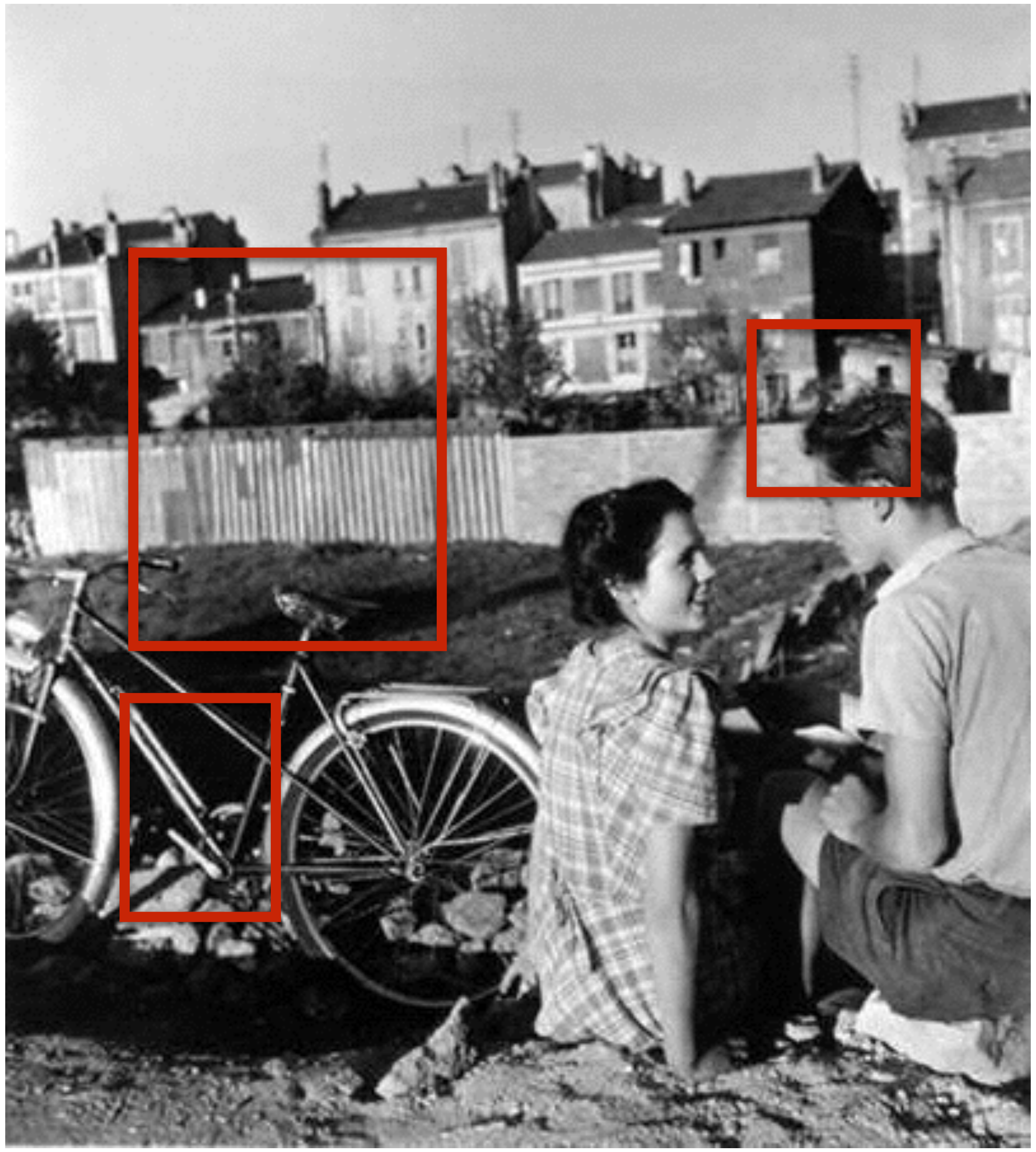
DPMs and CNNs



[Girshick et al, CVPR'15]

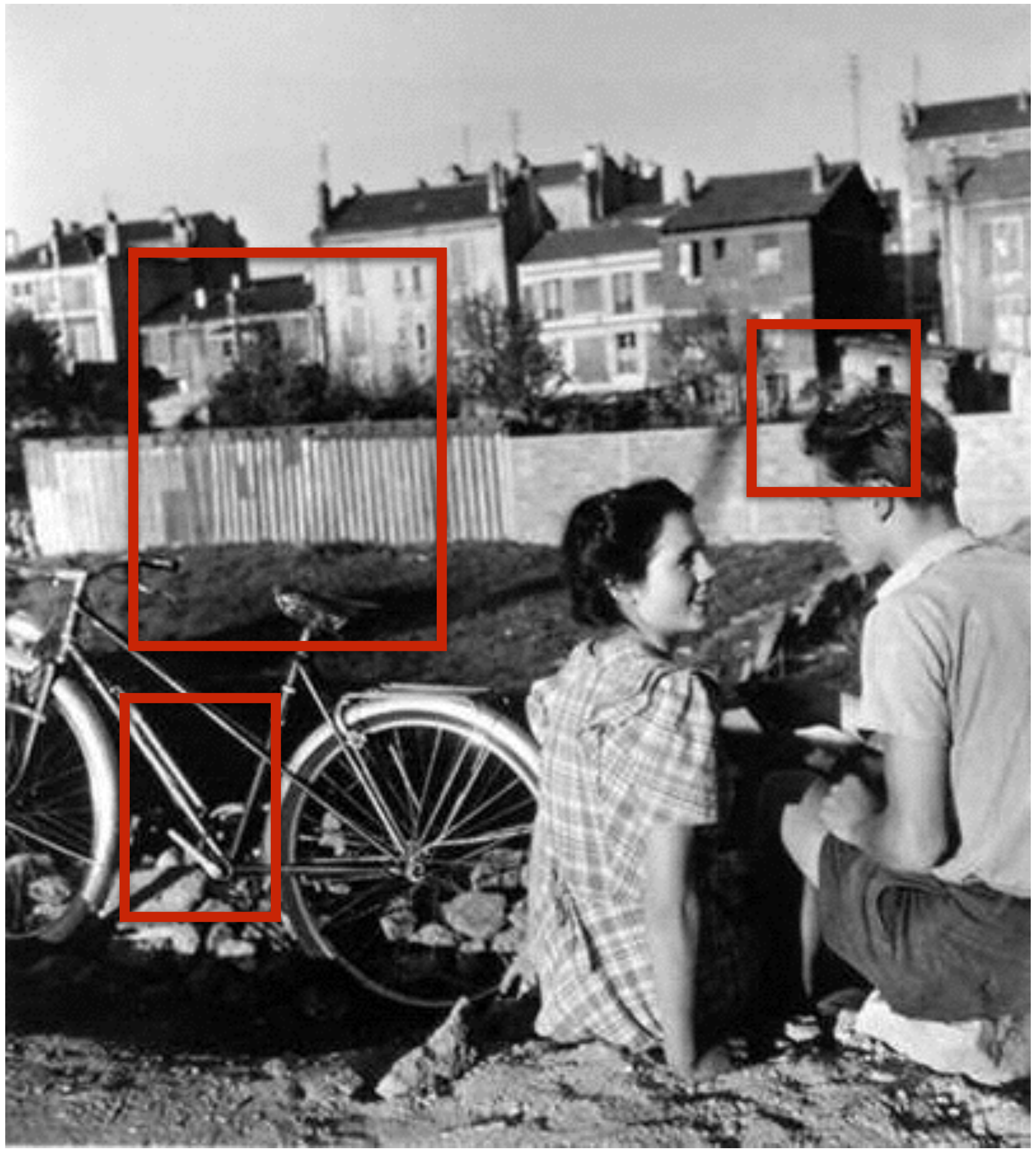
- Large improvement from using handcrafted features.
- State-of-the-art amongst "*sliding windows*" methods, i.e. translation invariant.

Region-based CNN (R-CNN)



- Suppose that for each bounding box we ask: is there a {house, bicycle, dog, man, …, *none*} ?

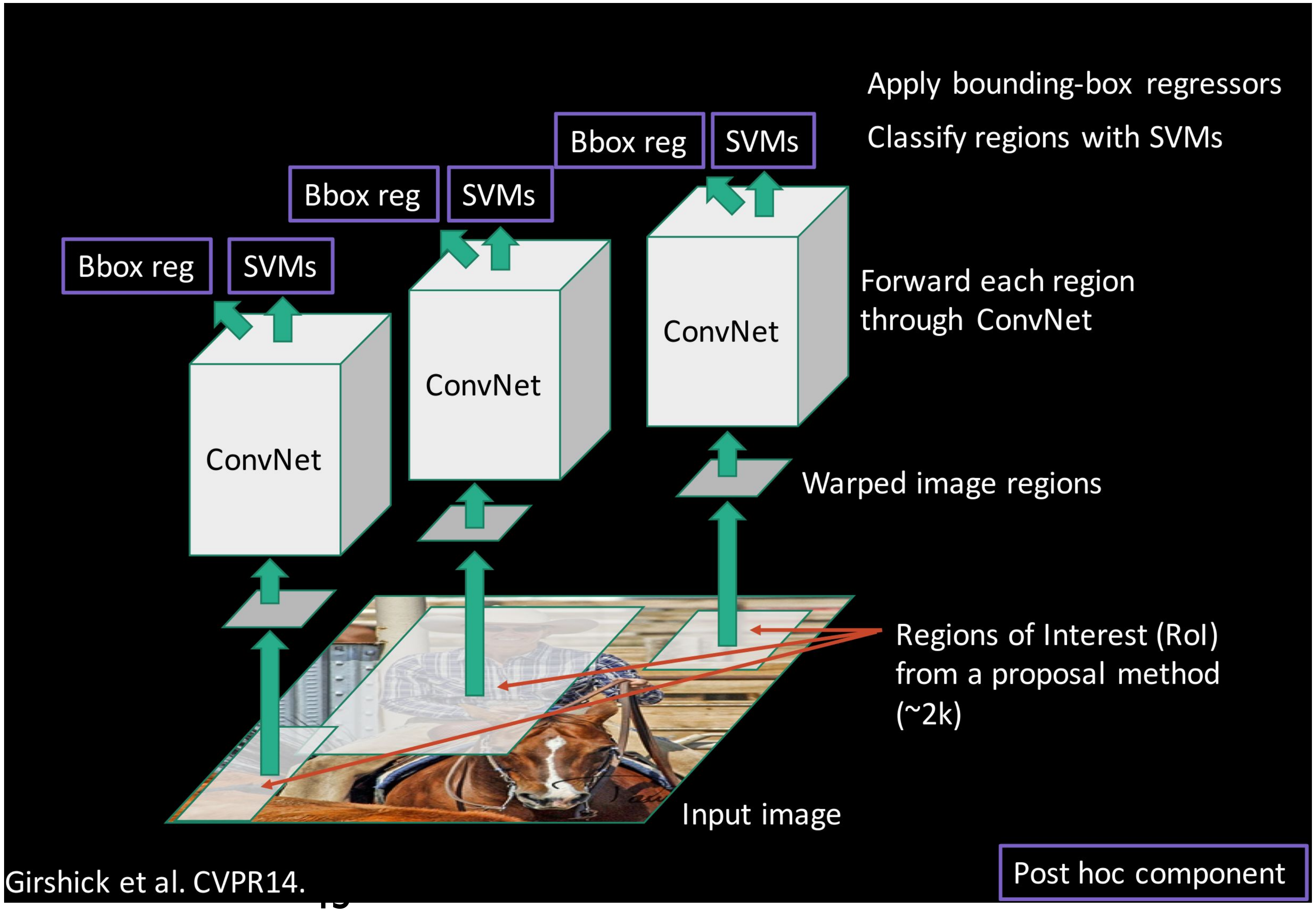
Region-based CNN (R-CNN)



- Suppose that for each bounding box we ask: is there a {house, bicycle, dog, man, …, *none*} ?
- This is standard object classification.

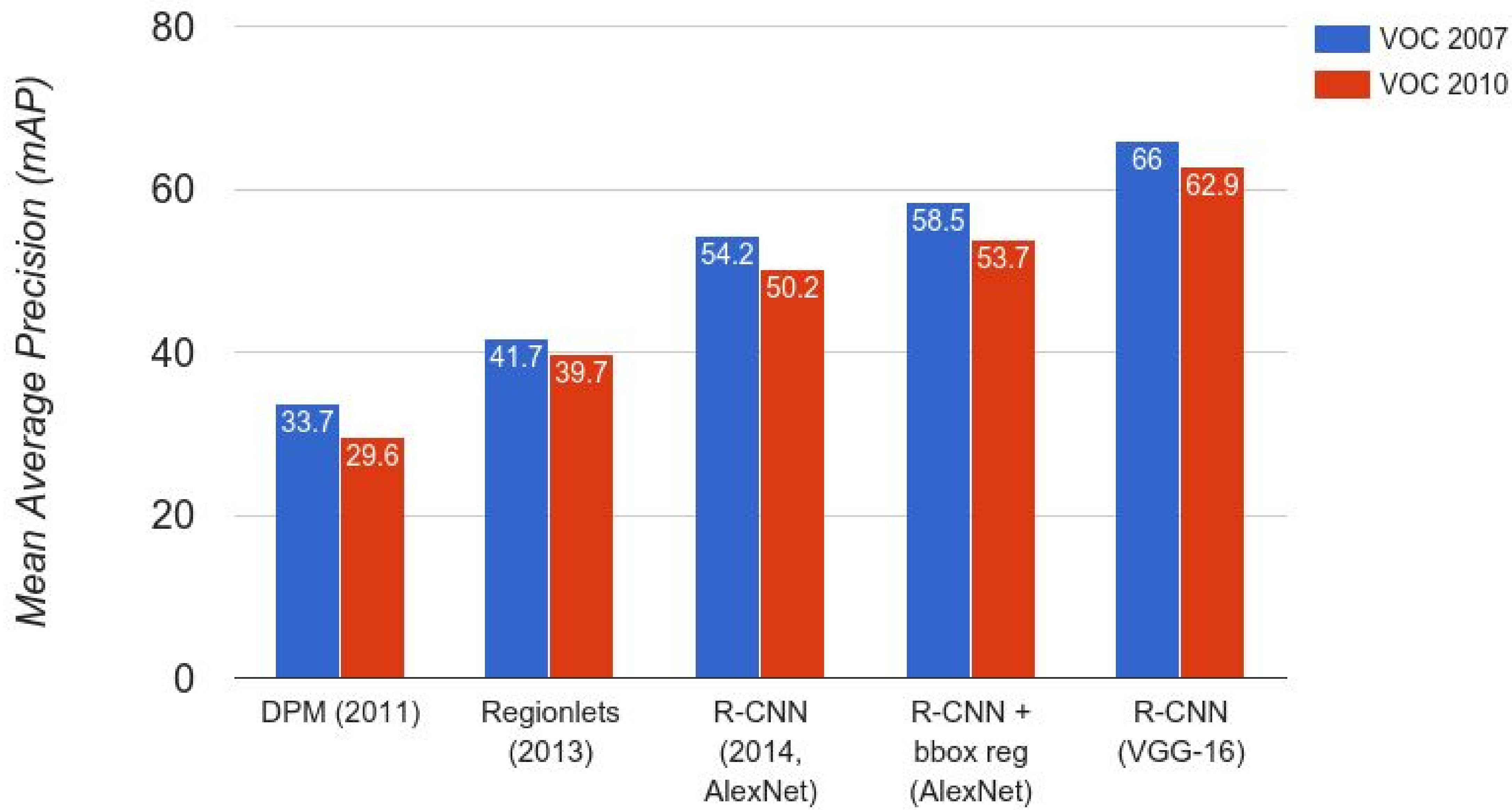
R-CNN [R. Girshick et al, 14-15]

- Rather than testing every possible rectangular region, we rely on a Region Proposal algorithm (which can also be done by a CNN).
- Each proposal region is warped and analyzed with another CNN.



R-CNN [R. Girshick et al, 14-15]

- Several improvements relating speed and performance (Fast R-CNN, Faster R-CNN) and replacing pre-trained CNN architectures (ResNet).



(figure credit: Stanford CS-231n lecture 8)

Structured Output Prediction

- Standard classification is only concerned with estimating conditional probabilities of the form $p(y \mid x)$:

$$x \in \mathcal{X} \longrightarrow y \in \mathcal{Y} \qquad \mathcal{Y} = \{s_1, \ldots, s_L\} \text{ (classification)}$$

Structured Output Prediction

- Standard classification is only concerned with estimating conditional probabilities of the form $p(y \mid x)$:

$$x \in \mathcal{X} \longrightarrow y \in \mathcal{Y} \qquad \mathcal{Y} = \{s_1, \ldots, s_L\} \text{ (classification)}$$

- The previous task was an example of *structured output prediction*:

$$x \in \mathcal{X} \longrightarrow y \in (\mathcal{Y}, \mu)$$

$\mu(y)$ models the output unconditional probability.

Structured Output Prediction

- Standard classification is only concerned with estimating conditional probabilities of the form $p(y \mid x)$:

$$x \in \mathcal{X} \longrightarrow y \in \mathcal{Y} \qquad \mathcal{Y} = \{s_1, \ldots, s_L\} \text{ (classification)}$$

- The previous task was an example of *structured output prediction*:

$$x \in \mathcal{X} \longrightarrow y \in (\mathcal{Y}, \mu)$$

$\mu(y)$ models the output unconditional probability.

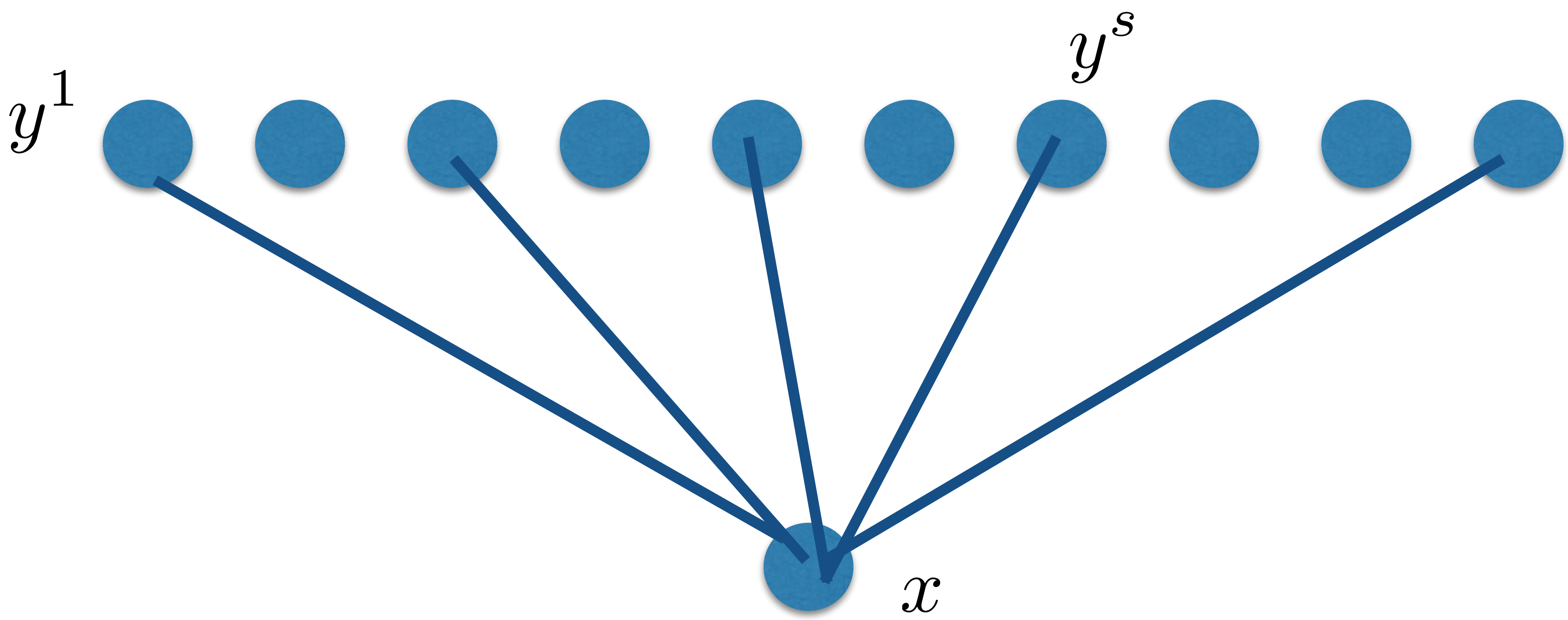
- Q: How to regularize the estimation of $p(y \mid x)$ with $\mu(y)$

Structured Output Prediction

- Examples:
  - Natural Language Processing: Translation, Summarization, Question Answering.
  - Image Segmentation.
  - Speech Recognition.
- Probabilistic Graphical Models are generic structured prediction models.
  - Bayesian Networks
  - Markov Random Fields
  - Sequence-to-Sequence Models (in a future lecture).
- Other models also considered (e.g. Structured SVM)

Structured Output Prediction

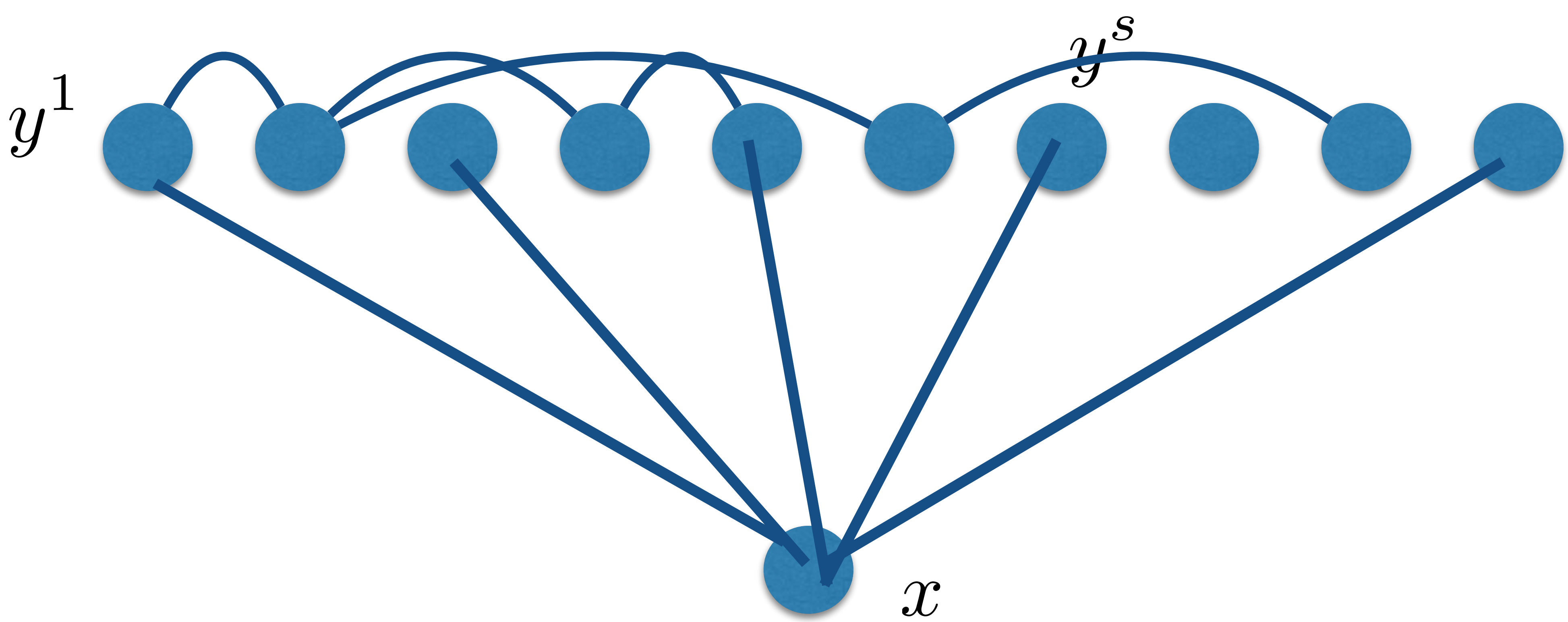
- Suppose $y = (y^1, \ldots, y^s, \ldots)$.



- If $p(y \mid x) = \prod_i p(y^i \mid x)$, the outputs are conditionally independent: we can estimate them separately.

Structured Output Prediction

- Suppose $y = (y^1, \ldots, y^s, \ldots)$.

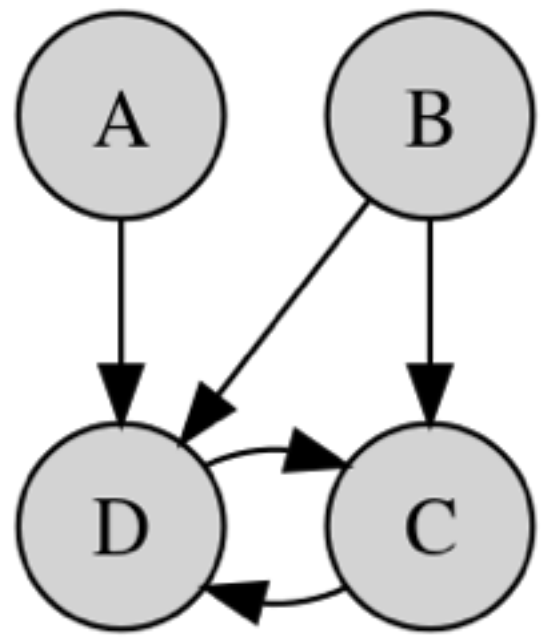


- But when we introduce statistical dependencies across outputs, the general model becomes

$$p(y \mid x) = \frac{\exp\left(-F(y, x, \Theta)\right)}{Z}$$

Graphical Models

- Broad class of probabilistic models that express a joint distribution as a product of factors.
- The dependency is expressed in terms of a graph:



(source: wikipedia)

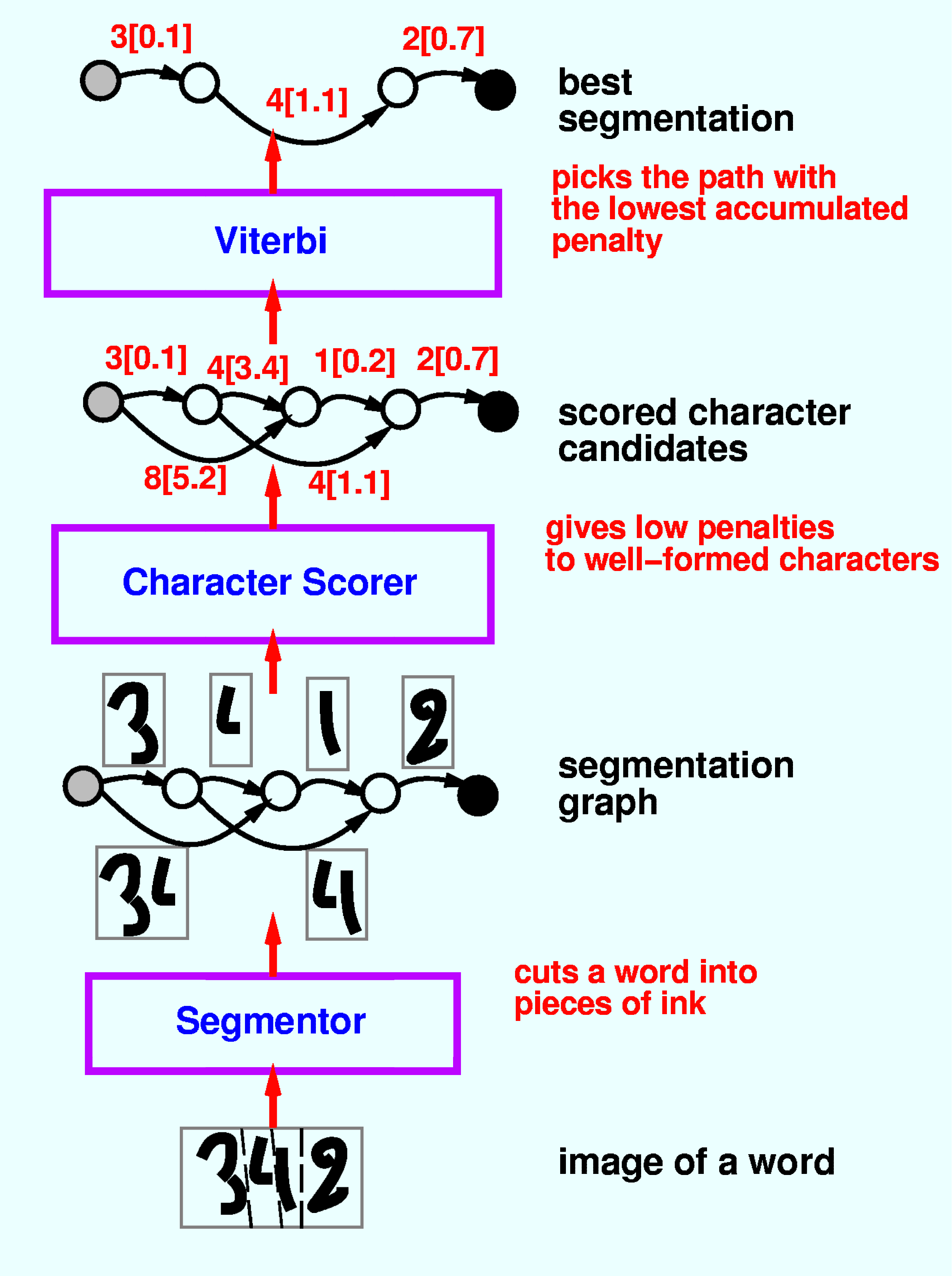
$$p(X_1, \ldots, X_n) = \prod_{i=1}^{n} P(X_i \mid \mathcal{P}_i)$$

$\mathcal{P}_i$: context associated with X_i

- Many instances: trees, factor graphs, Restricted Boltzmann machines (more on that later), Markov random fields,…

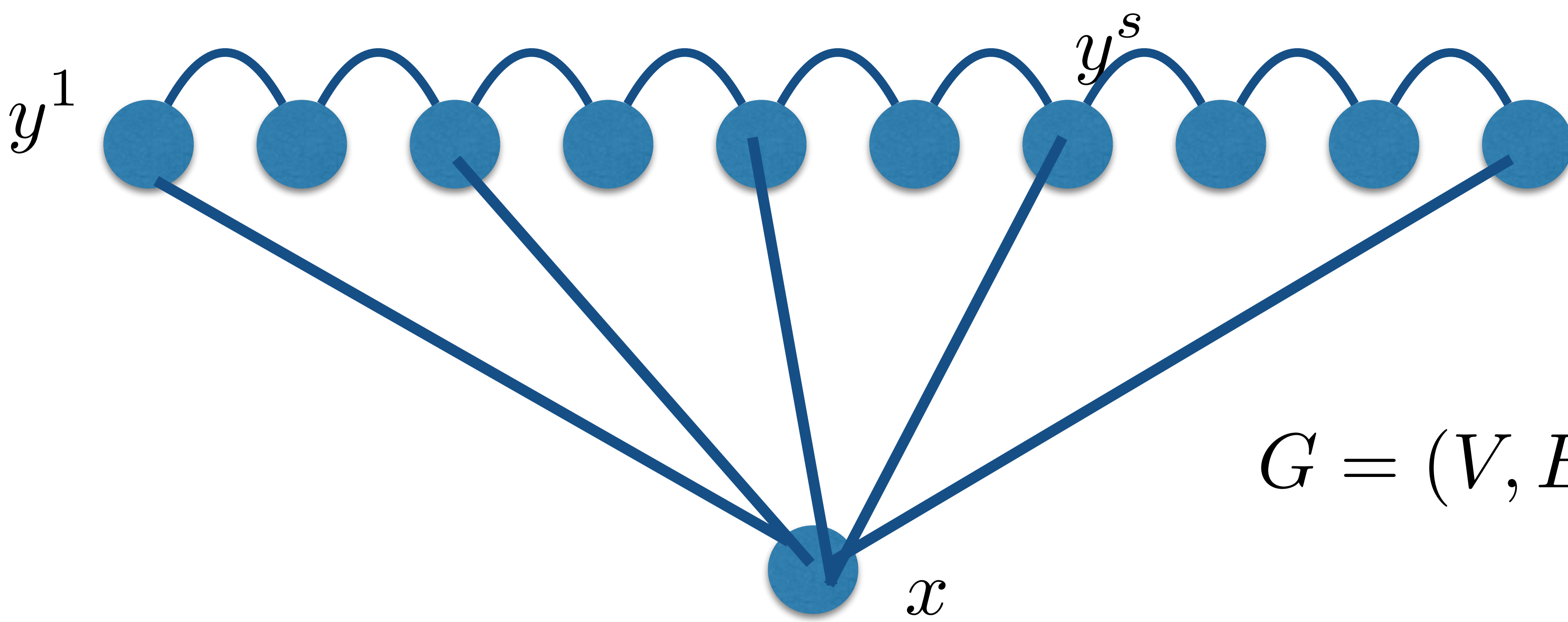
Graph Transformer Network

- [Bottou, Bengio & LeCun, '97]
- Graphical model over possible "segmentations" of handwritten characters
- Used commercially to read ~10% checks in the US (1996).



Conditional Random Fields

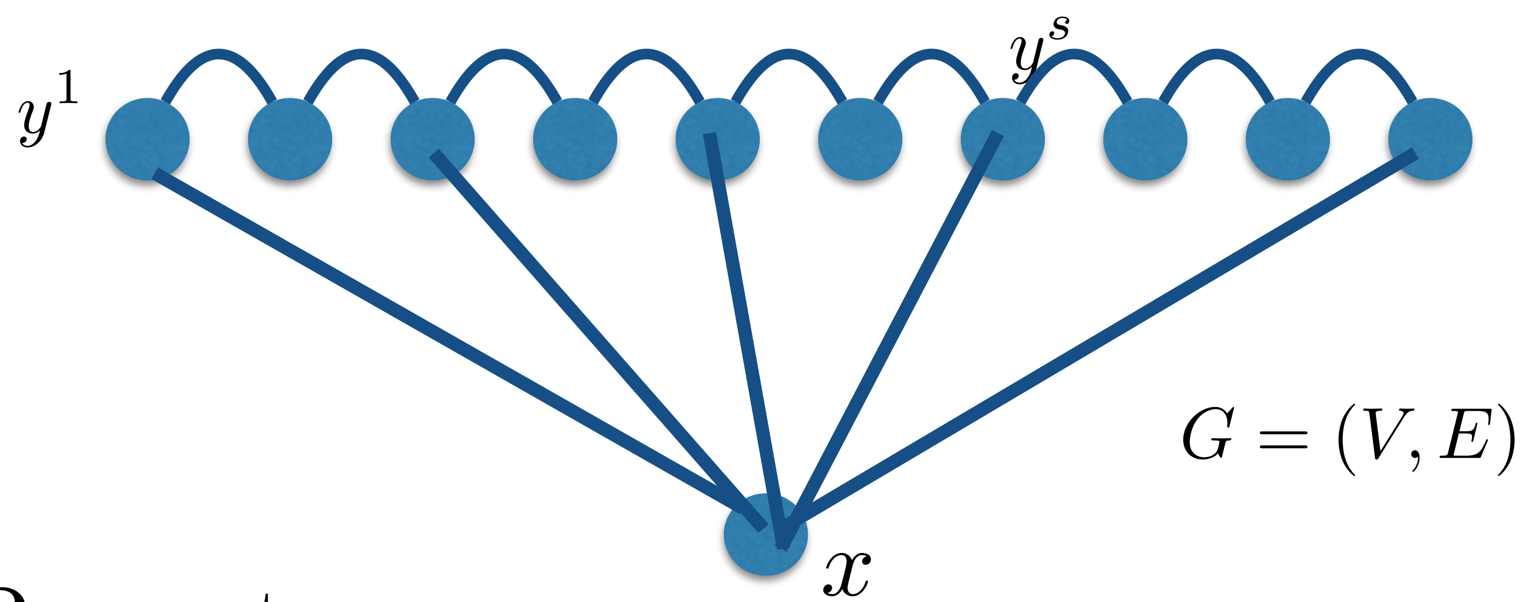
- Many problems ask to predict an output with temporal or spatial structure (eg speech, image (segmentation), natural language text).
- A Markov Random Field is a graphical model on an undirected graph:



$$G = (V, E)$$

Conditional Random Fields

- A Markov Random Field is a graphical model on an undirected graph:



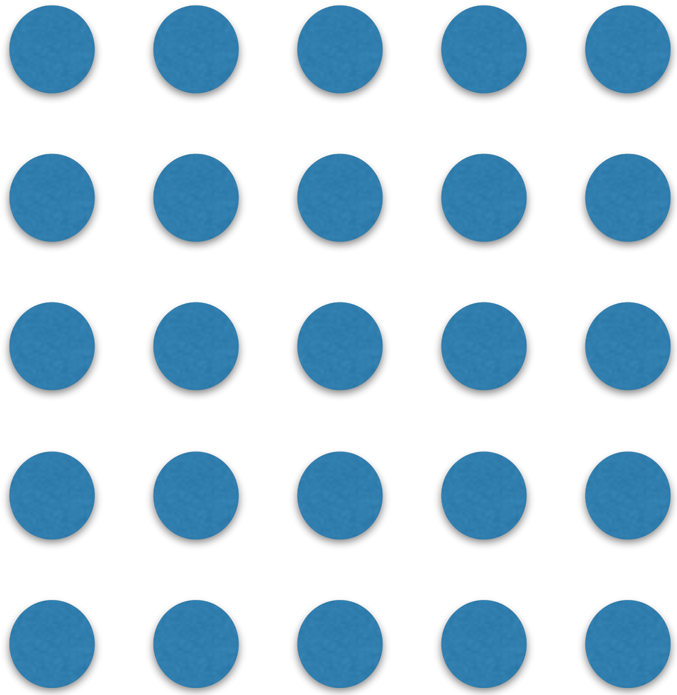
Markov Property:

$$p(y_i \mid X, y_j, j \neq i) = p(y_i \mid X, y_j, j \sim i)$$

- Inference is intractable for general graphs
  - trees and chains are exceptions
  - Algorithms for approximate inference: message passing, Viterbi, mean field inference.

Conditional Random Fields

- In images, pixels form a 2D lattice graph:



- In pixel labeling tasks (ie segmentation), the output configuration probability is expressed as

$$p(y \mid x) = \frac{e^{-E(y,x)}}{Z} \;, \;\; (Z: \;\; \text{partition function})$$

Conditional Random Fields

$$p(y \mid x) = \frac{e^{-E(y,x)}}{Z} \;, \;(Z:\; \text{partition function})$$

$$E(y,x) = \sum_{u} \psi_u(y,x) + \sum_{u \neq v} \psi_{u,v}(y,x)$$

ψ_u: "unary" potentials measure cost of pixel u being labeled y_u.

$\psi_{u,v}$: pairwise potentials measure cost of jointly assigning labels y_u, y_v at pixels u and v.

- unary potentials predict labels at each location as if they were independent from the rest
- pairwise potentials provide data-dependent smoothing.

CRFs as Convolutional Neural Networks

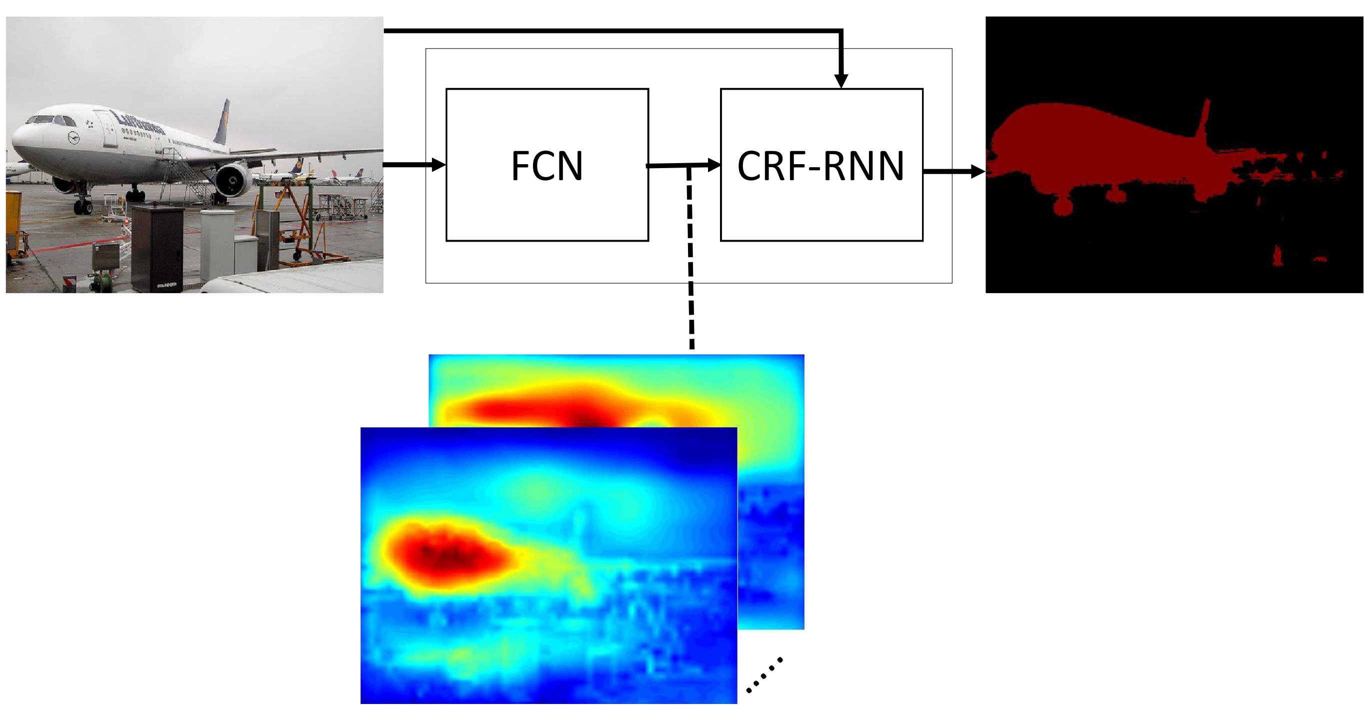
- An approximate posterior inference for the CRF model is done with mean-field approximation:

$$\text{Approximate } p(y \mid x) \text{ with } q(y \mid x) = \prod_i q_i(y_i \mid x) \text{ iteratively.}$$

- One can also consider belief propagation as an alternative to mean-field approximation (see http://www.eecs.berkeley.edu/~wainwrig/Talks/A_GraphModel_Tutorial for a great tutorial!)

CRFs as Convolutional Neural Networks

- [Zheng et al,'15] approximate the mean-field message passing iterations with CNN layers with shared parameters.
- The system can be efficiently trained end-to-end.



Algorithm 1 Mean-field in dense CRFs [27], broken down to common CNN operations.

$Q_i(l) \leftarrow \frac{1}{Z_i} \exp\left(U_i(l)\right)$ for all i ▷ Initialization

while not converged **do**

$\tilde{Q}_i^{(m)}(l) \leftarrow \sum_{j \neq i} k^{(m)}(\mathbf{f}_i, \mathbf{f}_j) Q_j(l)$ for all m ▷ Message Passing

$\check{Q}_i(l) \leftarrow \sum_m w^{(m)} \tilde{Q}_i^{(m)}(l)$ ▷ Weighting Filter Outputs

$\hat{Q}_i(l) \leftarrow \sum_{l' \in \mathcal{L}} \mu(l, l') \check{Q}_i(l)$ ▷ Compatibility Transform

$\breve{Q}_i(l) \leftarrow U_i(l) - \hat{Q}_i(l)$ ▷ Adding Unary Potentials

$Q_i \leftarrow \frac{1}{Z_i} \exp\left(\breve{Q}_i(l)\right)$ ▷ Normalizing

end while

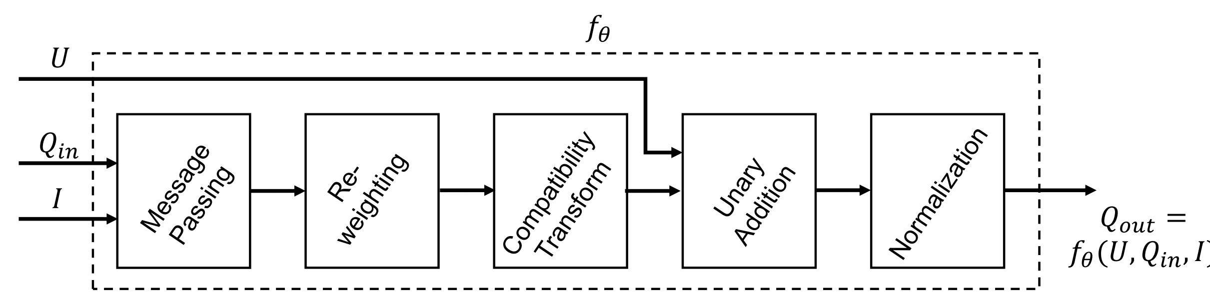
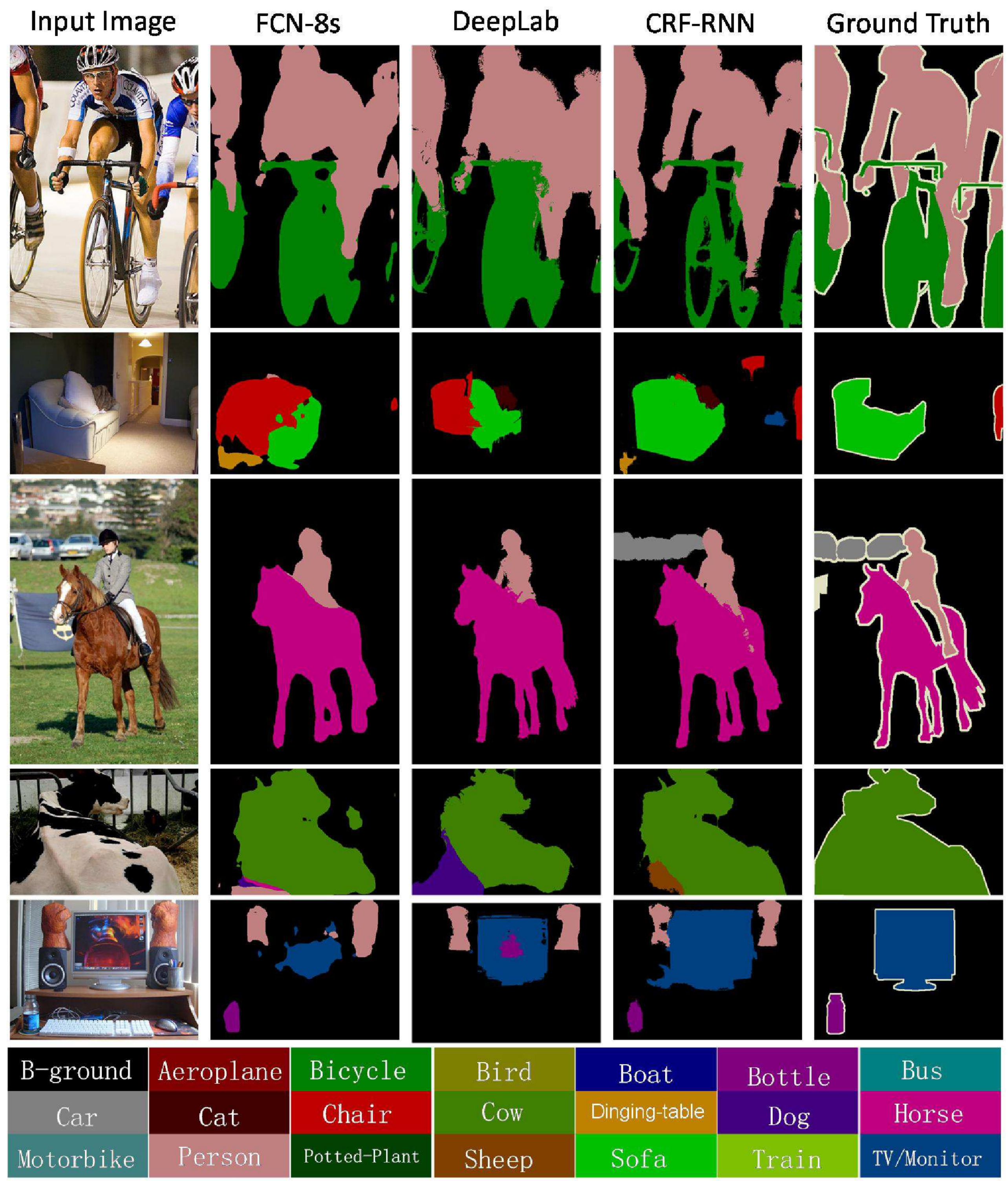


Figure 1. **A mean-field iteration as a CNN.** A single iteration of the mean-field algorithm can be modelled as a stack of common CNN layers.

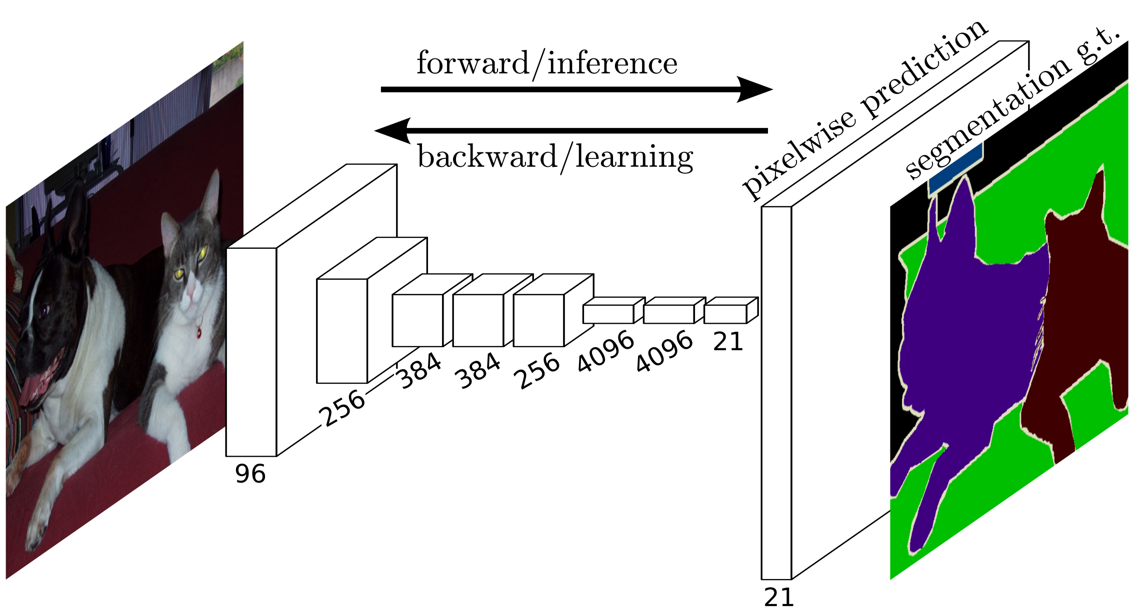
Example: Segmentation

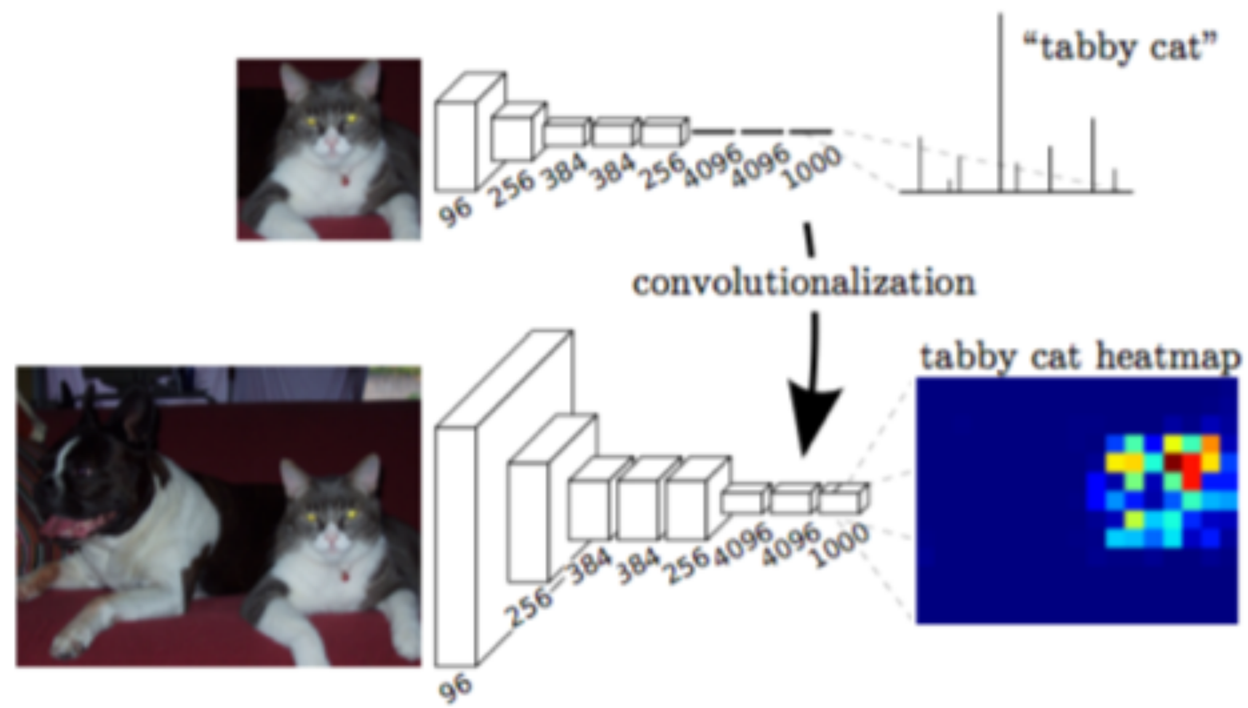
- Results from [Zheng et al, ICCV'15]



Example: Segmentation

- The CRF approximation is a specific CNN model.
- [Long, Shelhamer et al,CVPR'15] proposed a simpler CNN architecture that also produces excellent results.
- Idea: Combine outputs from different layers and refine the spatial resolution of the output.





- See also "Learning to Segment Object Candidates" [Pinheiro et al'15].

Example: Human Pose Estimation

- Human muscle joints are *very structured.*
- [Tompson et al, NIPS'14] considered a joint training of CNN and Markov Random Fields.

Example: Pose Estimation

- The unary potentials are modeled as detection CNNs.
- The pairwise potentials between different parts are modeled as convolutional priors.
- The marginal likelihoods for each part are of the form

$$\bar{p}(A \mid x) = \frac{1}{Z} \prod_{B} (p(A \mid B, x) \star p(B \mid x) + b_{B \to A})$$

($b_{B \to A}$: bias term for the mssage from B to A)