Structured Random Forests

Ben Haghi Cody Han Yury Tokpanov

Department of Computing and Mathematical Sciences
California Institute of Technology

CS159, Spring 2017

Outline

Prototypical Types of Tasks

Problems related to automatic analysis of complex data such as text, photographs, videos, and n-dimensional medical images can be categorized into these prototypical tasks:

- **Classification.** Recognizing type or category of a scene captured in a photograph, output is a discrete, categorical label.
- **Regression.** Predicting the price of a house as a function of its distance from a good school; output is a continuous variable.
- **Density estimation.** Learning a probability density function for healthy individual scans to detect abnormalities.
- **Manifold learning.** Capturing intrinsic variability of size and shape of different structures in human brain from MRIs.

Prototypical Types of Tasks

- **Semi-supervised.** Interactive image segmentation, for example, where user's brush strokes define labeled data and other pixels provide already available unlabeled data.
- **Active learning.** Learning a general rule for detecting tumors using only a few manual annotations, input mostly unlabeled.

What follows is a unified model of decision forests that can be used in all of these prototypical learning tasks. Can implement and optimize inference algorithms once and use them in many applications.

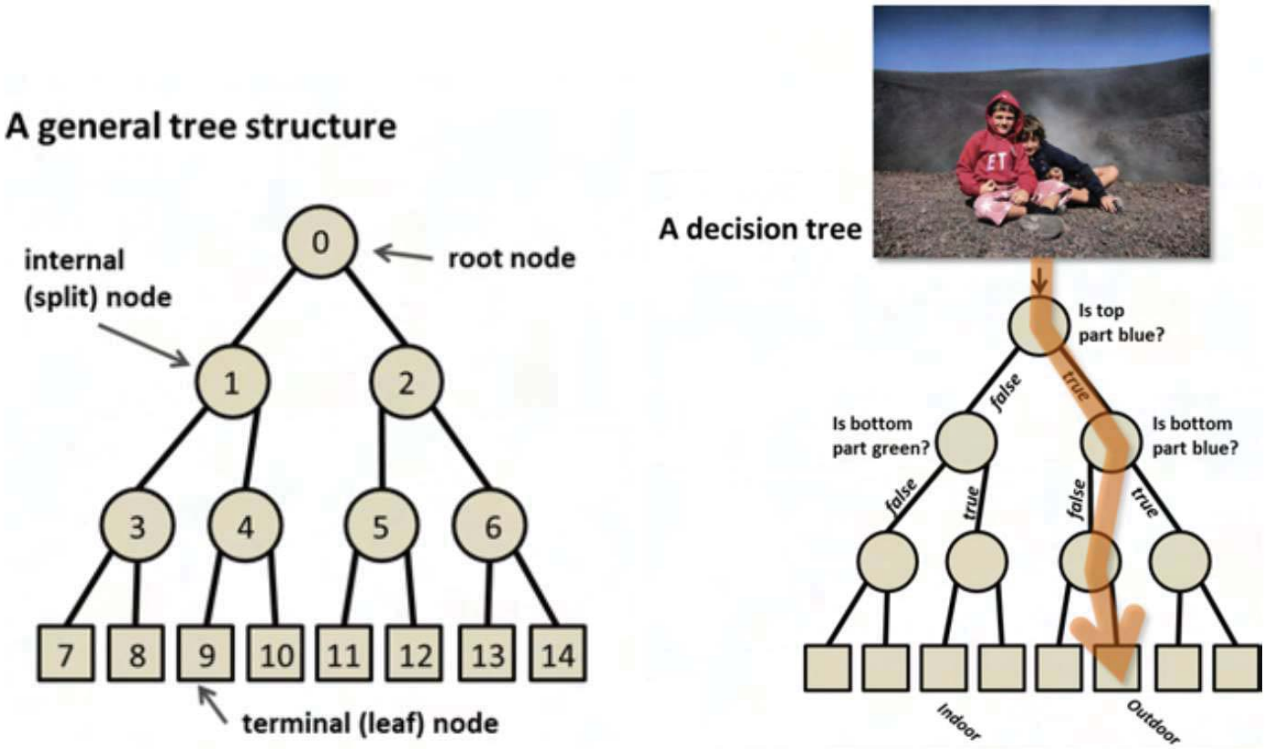
Review: Decision Tree Basics

A decision tree is a set of questions organized in a hierarchical manner and represented graphically as a tree.

- To estimate an unknown property of an input object, the tree asks successive questions about its known properties. Each question depends on the answers to the previous questions — represented as a path through the tree. The terminal node on the path determines the decision.
- Requires (1) tests associated to internal nodes and (2) decision-making predictors associated with each leaf.

In general, decision trees can be compared to splitting complex problems into sets of simpler ones with a hierarchical model. Model parameters can be selected by hand (simple) or learned from data.

Review: Decision Tree Basics



Left: general decision tree structure. Right: a decision tree for determining whether a photo was taken indoors or outdoors.

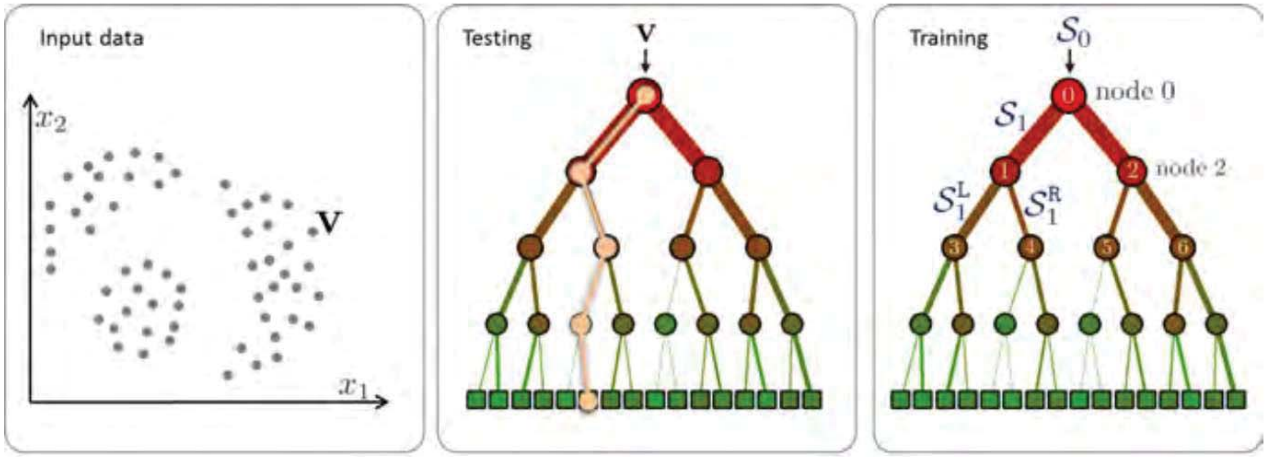
Notation

Definition (Data points and features)

A generic object, called a *data point*, is denoted by a vector $\mathbf{v} = (x_1, x_2, \ldots, x_d) \in \mathcal{F}$ where the components x_i represent some attributes of the data point called *features*.

Features vary between applications; for computer vision $\mathbf{v}$ may correspond to a pixel in an image and x_i may represent responses of a chosen filter bank at that location.

Notation



Notation example. Input is collected in d-dimensional space. Testing is done by each node on $\mathbf{v}$, which is sent to child node. Training involves sending all training data $\mathcal{S}_0$ into the tree and optimizing the split node parameters over an energy function. (Criminsi 2012)

Notation

- Number of features depends on the type of data point and application.
- Dimensionality of feature space $\mathcal{F} = d$ can be large, even infinite, so we extract a small portion of d features as we need them.

Definition (Features of interest)

Define $\phi(\mathbf{v}) = \left(x_{\phi_1}, x_{\phi_2}, \ldots, x_{\phi_{d'}}\right) \in \mathcal{F}^{d'} \subset \mathcal{F}$ as the selected subset of features where d' denotes the dimensionality of the subspace and $\phi_i \in [1, d]$ denote the selected dimensions.

We usually choose a subspace $\mathcal{F}^{d'}$ that is much smaller than the original ($d' \ll d$).

Notation

Test functions, split functions, weak learners

We use the terms "test function," "split function," and "weak learner" interchangeably.

Definition (Test function)

A test function at split node j is a function

$$h(\mathbf{v}, \theta_j) : \mathcal{F} \times \mathcal{T} \rightarrow \{0, 1\},$$

where 0 and 1 can be interpreted as false or true respectively, and $\theta_j \in \mathcal{T}$ denote the parameters of the test function at the jth split node.

A data point $\mathbf{v}$ arriving at j is sent to its left or right child according to the result of j's test function.

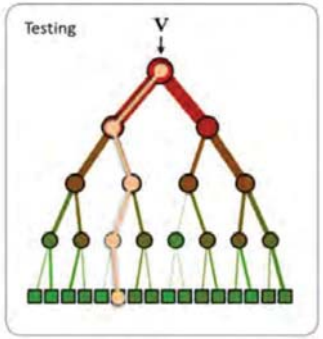
Notation

Training points and training sets

- A *training point* is a data point for which the attributes that we are seeking for are actually known. For example, a set of photos with "indoor" or "outdoor" labels.
- A *training set* $\mathcal{S}_0$ is a collection of different training points.
- In supervised tasks, a training point is a pair $(\mathbf{v}, \mathbf{y})$ where $\mathbf{v}$ is the input feature vector and $\mathbf{y}$ represents a generic, known label.
- In unsupervised tasks, a training point is represnted by its feature response and it has no associated label.

Decision tree testing

Decision trees are separated into an offline training phase and an online testing phase.



(Criminsi 2012)

Testing

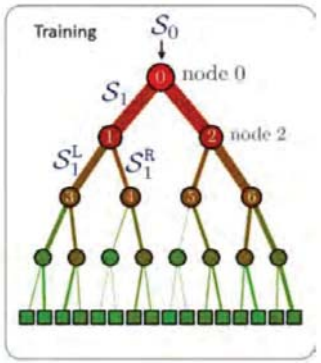
A new data point **v** starts at the root and makes it way down to a leaf node, which contains uses a predictor to construct an output (label or continuous value) for **v**.

Decision tree training

Training

Selects the type and parameters of the test function $h(\mathbf{v}, \theta_j)$ associated with each split node j by optimizing a chosen objective function defined on the training set.

- At each node j with input $\mathcal{S}_j$, learn the function that "best" splits S_j into S_j^R and S_j^L.
- Do this by maximizing an objective function.



(Criminsi 2012)

Training objective function

The objective function at node j, in general, is defined as

$$\theta_j^* = \arg\max_{\theta_j \in \mathcal{T}} I_j \tag{1}$$

where

$$I_j = I(\mathcal{S}_j, \mathcal{S}_j^L, \mathcal{S}_j^R, \theta_j)$$
$$S_j^L = \{(\mathbf{v}, \mathbf{y}) \in \mathcal{S}_j \mid h(\mathbf{v}, \theta) = 0\}$$
$$S_j^R = \{(\mathbf{v}, \mathbf{y}) \in S_j \mid h(\mathbf{v}, \theta_j) = 1\}$$

I_j captures some notion of information gain from the split while the second two equations state that data points on which h evaluates to 0 go left and to 1 go right.

Stopping criteria for training

We need some way to know when to stop the training algorithm.

- Can stop when the tree has some maximum number of levels D.
- Can also stop when we reach some minimum value of $\max_{\theta_j} I_j$, which is when the attributes that we care about in the leaf nodes are similar to one another.
- Can stop growing tree when a node contains too few training points.

Stopping criteria prevent overfitting and loss of generalization power.

Weak Learner Models

Consider a geometric parameterization for split functions, with a weak learner model formulated as $\theta = (\phi, \psi, \tau)$ where:

- ψ defines the geometric primitive used to separate the data (e.g. axis-aligned hyperplane, oblique hyperplane, general surface).
- τ is a parameter vector that captures thresholds for inequalities used in the binary test.
- ϕ is a filter function that selects features from input vector $\mathbf{v}$.

We perform the optimization from equation (1) over these subparameters.

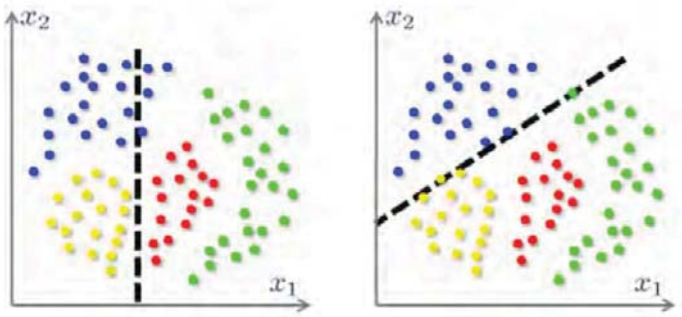
Linear and Nonlinear Data Separation

A simple parameterization is the linear model

$$h(\mathbf{v}, \theta_j) = [\tau_1 > \phi(\mathbf{v}) \cdot \psi > \tau_2] \tag{2}$$

where $[\cdot]$ is the indicator function.

- In 2D, $\phi(\mathbf{v}) = (x_1, x_2, 1)^T$ and $\psi \in \mathbf{R}^3$ denote a generic line in homogenous coordinates.
- Setting $\tau_1 = \infty$ or $\tau_2 = -\infty$ corresponds to using a single inequality test function.
- Lines ψ aligned with an axes of the feature space (e.g. $\psi = (1, 0, \psi_3)$ or $\psi = (0, 1, \psi_3)$) are often used in boosting and are referred to as *stumps*.



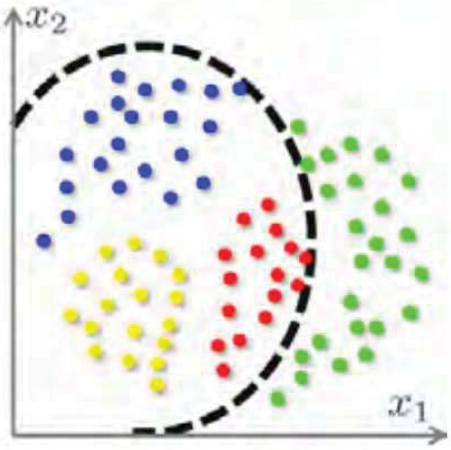
Axis aligned vs general line (Criminsi 2012)

Linear and Nonlinear Data Separation

We can replace hyperplanes with surfaces that have more degrees of freedom. In 2D we can use conic sections with

$$h(\mathbf{v}, \theta_j) = \left[\tau_1 > \phi^T(\mathbf{v})\ \psi\ \phi(\mathbf{v}) > \tau_2\right] \quad (3)$$

with $\psi \in \mathbf{R}^{3\times 3}$ representing the conic section in homogenous coordinates.



Quadratic separation

Remark

Low dimensional weak learners can be used for high dimensional data because the selector ϕ_j can select a small set of features for different nodes.

Energy Models (discrete)

Entropy

By training objective functions, we reduce uncertainty using the concepts of *entropy* and *information gain*. We can quantify the intuition of uncertainty reduction with measures for entropy and information gain.

Shannon entropy

For discrete probability distributions we typically use Shannon entropy.

$$H(\mathcal{S}) = -\sum_{c \in \mathcal{C}} p(c) \log(p(c)) \tag{4}$$

where $\mathcal{S}$ is the set of training points and c indicates the class label. $\mathcal{C}$ denotes the set of all classes and $p(c)$ is the empirical distribution extracted from the training points in $\mathcal{S}$.

Energy Models (discrete)

Information gain

Improvements from decision tree splits can be quantified by measuring information gain

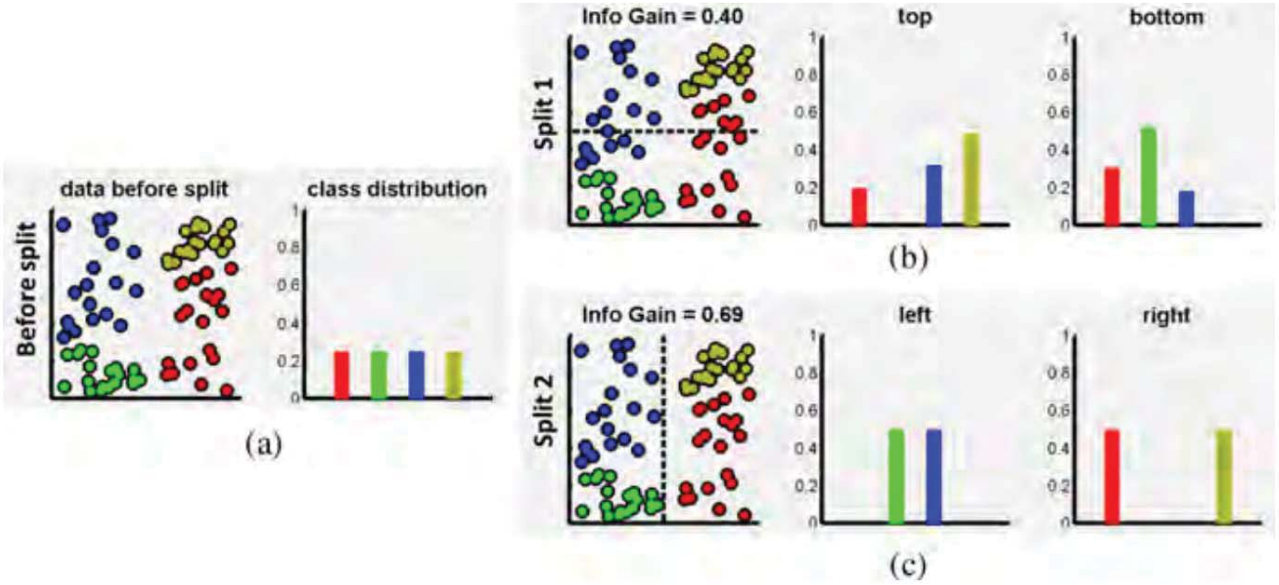
$$I = H(\mathcal{S}) - \sum_{i \in \{L,R\}} \frac{|\mathcal{S}^i|}{|\mathcal{S}|} H(\mathcal{S}^i) \tag{5}$$

Remark

Maximizing information gain gives us split parameters which produce the highest confidence in the final distributions.

Energy Models (discrete)

Example



Information gain from axis aligned discrete splits (Criminsi 2012)

Energy Models (continuous)

Entropy

We can also define entropy and information gain for continuous labels and distributions.

Differential entropy

In place of the Shannon entropy, we use differential entropy

$$H(\mathcal{S}) = -\int_{y \in \mathcal{Y}} p(y) \log(p(y))\, dy \tag{6}$$

where y is a continuous label and p is the probability density functino estimated from the training points in $\mathcal{S}$.

Energy Models (continuous)

Gaussian-based models are often used to approximate $p(y)$ because of their simplicity. The differential entropy of a d-variate Gaussian is

$$H(\mathcal{S}) = \frac{1}{2} \log \left[(2\pi e)^d \left| \Lambda(\mathcal{S}) \right| \right] \tag{7}$$

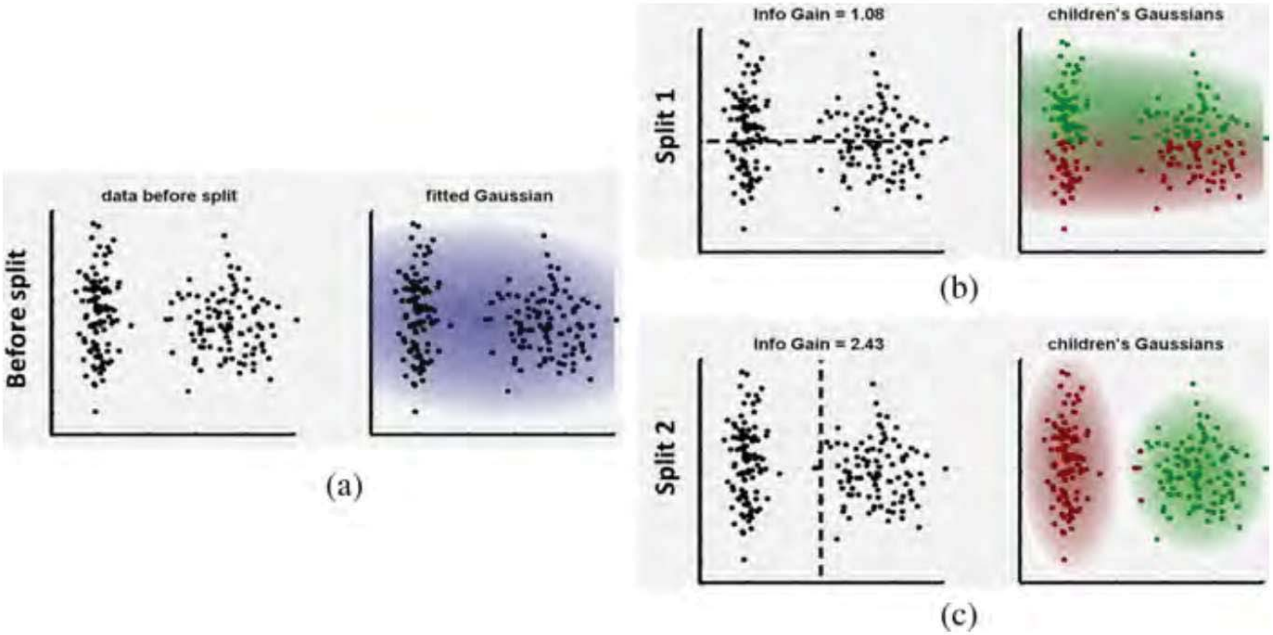
where $\Lambda(\mathcal{S})$ is the covariance matrix of the training set.

Remark

Information gain (5) *can be defined the same way in continuous models as discrete models because of the flexibility of its definition.*

Energy Models (continuous)

Example



Information gain from Gaussian model (Criminsi 2012)

Leaf Prediction Models

After training, each leaf is associated with a subset of the labeled training data.

- A test point traverses the tree until it reaches a leaf, and we generate a label for that point using the test statistics gathered in that leaf.
- In general, leaf statistics can be captured with posterior distributions

$$p(c \mid \mathbf{v}) \quad \text{and} \quad p(y \mid \mathbf{v}) \tag{8}$$

where c and y represent categorical (discrete) or continuous labels and $\mathbf{v}$ is the data point being tested. Conditioning denotes the fact that the distributions depend on the specific leaf note reached by $\mathbf{v}$.

Leaf Prediction Models

MAP predictor

We can do a maximum a posteriori (MAP) estimate with

$$c^* = \arg\max_c p(c \mid \mathbf{v}) \tag{9}$$

in the discrete case. In general, however, keeping the entire distribution around lets us reason about uncertainties better.

Randomness Model

During training, we inject randomness into trees in order to give the trained trees better generalization power. Two popular ways to do this are:

1. Random training set sampling (bagging). This typically yields good training efficiency.
2. Randomized node optimization. This enables us to train trees on the entire training data and yields margin-maximization properties for ensemble models.

These two methods can be used together.

Randomness Model

Randomized node optimization

In the optimization equation (1), we optimize with respect to the entire parameter space $\mathcal{T}$. For large dimensional problems, it can be infeasible to optimize over $\mathcal{T}$.

Idea

At node j, use a small random subset $\mathcal{T}_j \subset \mathcal{T}$ of parameter values, optimizing

$$\theta_j^* = \arg\max_{\theta_j \in \mathcal{T}_j} I_j \tag{10}$$

The amount of randomness is controlled by $|\mathcal{T}_j|/|\mathcal{T}|$. We can define $\rho = |\mathcal{T}_j|$ so that when $\rho = |\mathcal{T}|$ there is no randomness and when $\rho = 1$ we have maximum randomness and no optimization.

Random Decision Forests

A random decision forest is an emsemble of randomly trained decision trees.

Key aspect

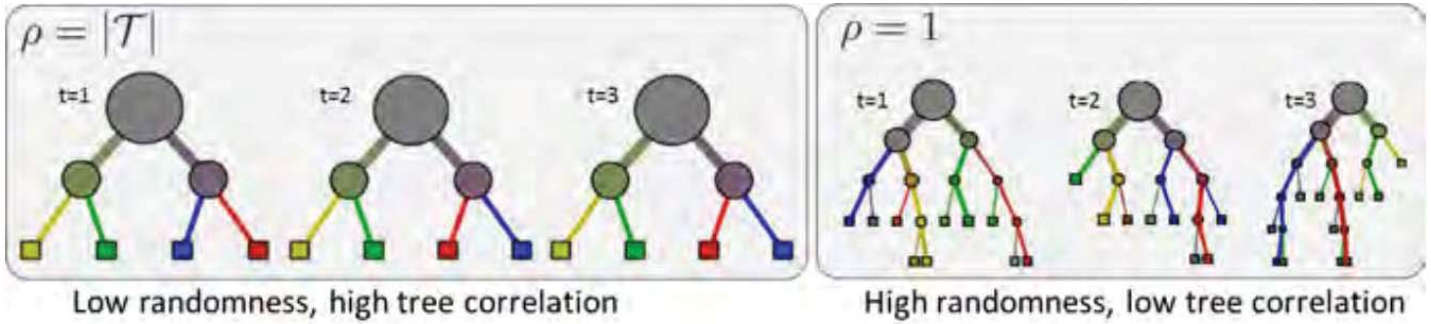
The component trees in a random forest model are all randomly different from one another, which decorrelates individual tree predictions and results in improved generalization and robustness.

Forest properties

The weak learners, energy model, leaf predictors, and types of randomness in its component trees influence the prediction and estimation properties in a forest.

Random Decision Forests

The randomness parameter $\rho = |\mathcal{T}_j|$ controls the correlation between trees in a forest.



Randomness control in forest (Criminsi 2012)

Random Decision Forests
Training and prediction

Notation

In a forest with T trees, we use $t \in 1, \ldots, T$ to index component trees.

- Component trees are trained individually — this can be done in parallel.
- A test point $\mathbf{v}$ is simultaneously passed through all trees until it reaches the leaves — can also be done in parallel.

Random Decision Forests
Training and prediction

We can combine all tree predictions into a forest prediction in multiple ways. For classification, we can use a simple average

$$p(c \mid \mathbf{v}) = \frac{1}{T} \sum_{t=1}^{T} p_t(c \mid \mathbf{v}) \tag{11}$$

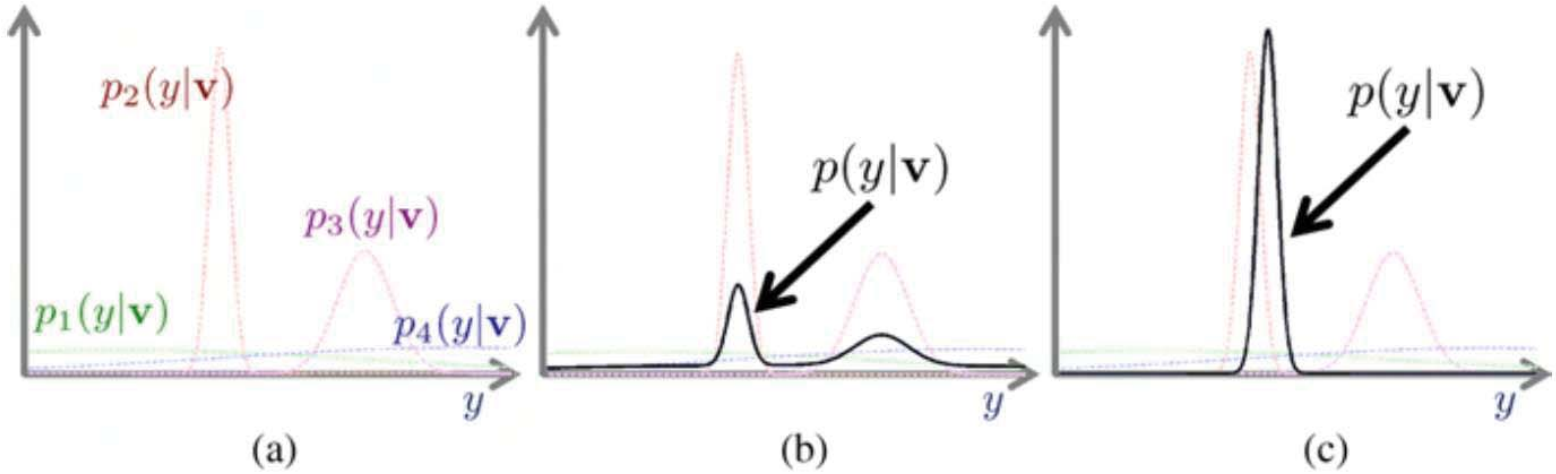
where $p_t(c \mid \mathbf{v})$ represents the posterior distribution found in the leaf of the tth tree. We can also multiply tree outputs together

$$p(c \mid \mathbf{v}) = \frac{1}{Z} \prod_{t=1}^{T} p_t(c \mid \mathbf{v}) \tag{12}$$

where the partition function Z ensures probabilistic normalization.

Random Decision Forests

Example



Forest ensemble model (Criminsi 2012)

Key Parameters

The parameter models that most influence decision forest behavior:

- Maximum allowed tree depth D. Very deep trees can lead to overfitting.
- Amount of randomness (controlled by ρ) and its type. Randomness affects tree correlation and generalization properties.
- Forest size (number of trees) T. Testing accuracy generally increases as T increases, but training time also increases.
- Choice of weak learner model
- Training objective function
- Choice of features in practical applications

Random Forest Specializations

The generic decision forest model we just defined can be applied to various specific tasks:

- Classification forests
- Regression forests
- Density estimation forests
- Manifold forests
- Semi-supervised forests
- Random ferns and other variants

Each of the above models can be defined with slight modifications and specifications on top of the random forest framework. If interested, see the original paper in the references.

For Further Reading I

A. Criminsi, J. Shotton, E. Konukoglu.
Decision forests: A unified framework for classification, regression, density estimation, manifold learning and semi-supervised learning.
Foundations and Trends in Computer Graphics and Vision, vol. 7, no. 2-3, pp. 81-227, 2012.

Structured Class-Labels in Random Forests for Semantic Image Labelling

Peter Kontschieder, Samuel Rota Balu, Horst Bischof and Marcello Pelillo

Caltech
CS159

Benyamin A. Haghi

May 9th, 2017

Abstract

- A simple and effective way to integrate structural information in random forests
  - Available topological distribution of object classes
  - Coherently labelled regions
- Contents:
  - Augmentation of random forests and structured label information
  - A novel data-splitting function
    - ✓ Joint distributions
    - ✓ Learning typical label transitions between object classes
  - Two possibilities for integrating the structured output predictions into semantic labeling

Introduction and Motivation

- Great attention on the field of visual object classification
  - Object detection
  - Classification
  - Tracking
  - Action recognition

- Supervised learning algorithms for semantic image labelling
  - Large amount of densely labelled training data

- Structured information
  - A typical street:
    - ✓Road, car, bicyclist, etc.

Introduction and Motivation

- Goal of structured learning:
  - Providing ideas to take this form of additional, structural information into account in learning process
    - ✓ Fitting to the needs of semantic image labelling/segmentation
  - Example: Street scene
    - ✓ A car should be driving on a road, but not on top of a building
- State-of-the-art approaches:
  - Using Complementary features at different levels within random field models
- Structural information is mostly incorporated on the highest, semantic level

Features

Low level

- Mostly calculated on a per-pixel basis
- Incorporate local color or texture statistics

Mid level

- Operating on regions or super-pixels
- Providing shape, continuity or symmetry information

High level

- Introducing global image statistics on the image level
- Providing information about inter-object or contextual relations
- Seeking for proper scene configurations

Introduction and Motivation

- At this paper:
  - Simple, effective way for incorporating structural information in the popular random forest
  - Competitive to the other state-of-the-art learning techniques
    - ✓ Boosting
    - ✓ SVM
  - A novel way for incorporating joint statistics about the local label neighborhood
    - ✓ learning typical labelling transitions among object class categories
    - ✓ Disadvantage of standard classification:
      - – Can only deal with a single (atomic) label per training sample
      - – Meaningless label configurations
  - CamVid and MSRCv2 databases

Introduction and Motivation

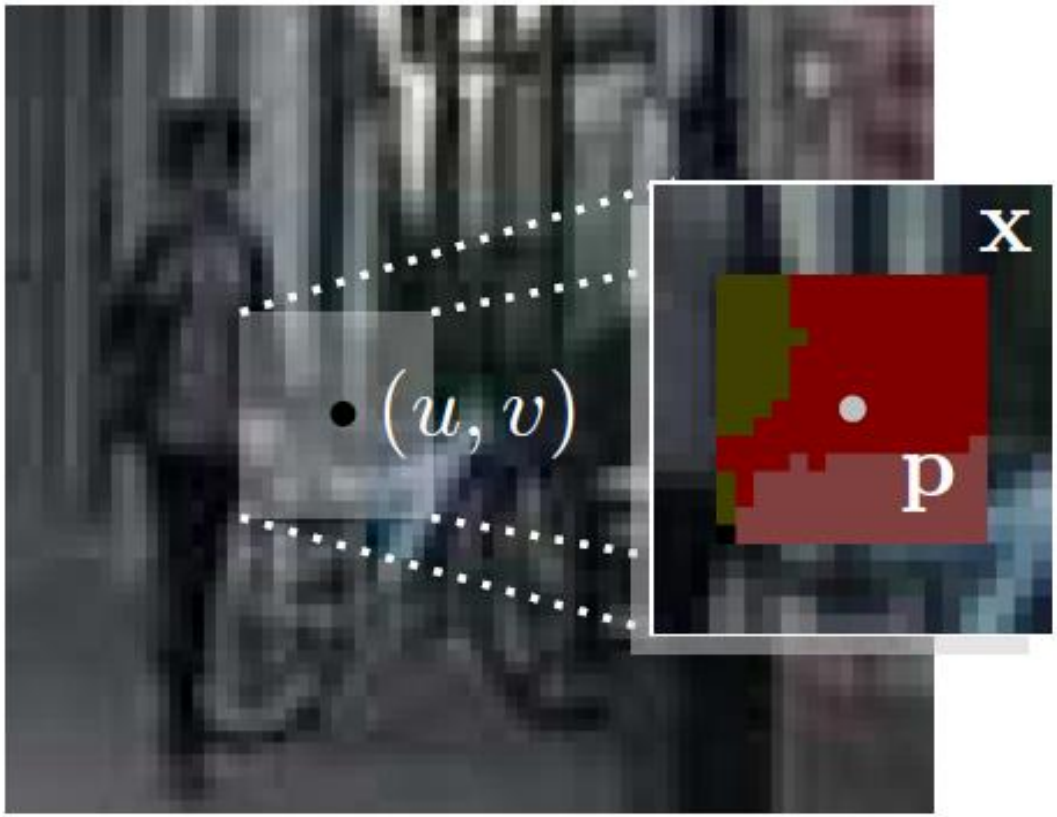


Fig. 1. Training data example

Advantages and Disadvantages

- Advantages
  1. Including the label topology in the training stage
     - respecting the label configurations observed during training
  2. Avoiding assigning implausible label transitions

- Disadvantages:
  - Need for densely labelled training data
    - Shared problem with state-of-the-art image labelling algorithms
    - The results of experiments on MSRCv2:
      - Well handled Non-completely labelled training data

Randomized Decision Forests

- Extremely fast for training and classification
- Can be easily parallelized
- Inherently multi-class capable
- Tending not to over-fit
- Robust to label noise

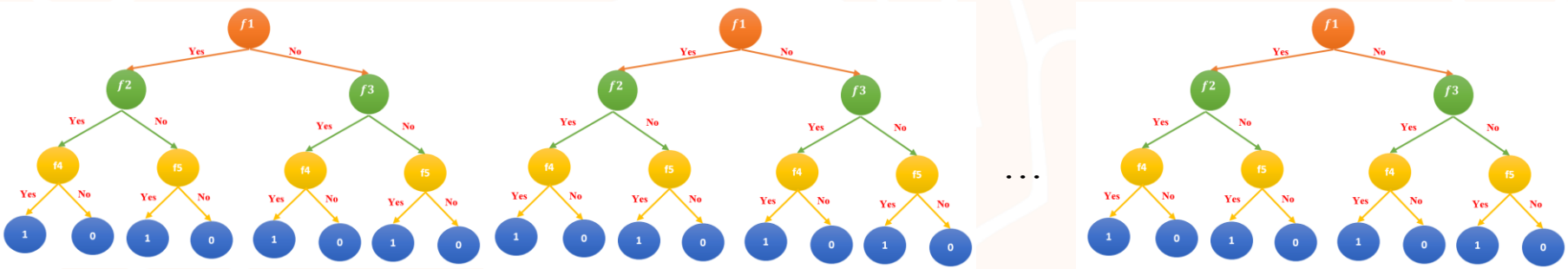


Fig. 2. Binary decision forest: Ensemble of binary decision trees, mitigate the risk of over-fitting

Randomized Decision Forests

- A (binary) decision tree:
  - Tree-structured classifier
  - Making a prediction by routing a feature sample $x \in \chi$ through the tree to the leaf
  - A *node* $ND(\psi, t_l, t_r) \in \mathbb{T}$

Binary split function
$\psi(x)$: $\chi \rightarrow \{0,1\}$
$\psi(x) = 0 \rightarrow t_l$
$\psi(x) = 1 \rightarrow t_r$

Left decision sub-tree

Right decision sub-tree

- A leaf:

$$LF(\pi) \in \mathbb{T}$$

  - The simplest form of a decision tree
  - Able to cast a class prediction $\pi \in \Upsilon$ for any sample it is reached by

Class Prediction

- For a sample $x \in \chi$
  - Recursively branching the sample down the tree until a leaf is reached
  - Tree prediction function:

$$h(x|t): \chi \to \Upsilon \quad for\ t \in \mathbb{T}$$

$$h(x|ND(\psi, t_l, t_r)) = \begin{cases} h(x|t_l) & \text{if } \psi(x) = 0 \\ h(x|t_r) & \text{if } \psi(x) = 1 \end{cases}$$

$$h(x|LF(\pi)) = \pi$$

- Forest F prediction function: Majority of the votes

$$y^* = \underset{y \in \Upsilon}{\operatorname{argmax}} \sum_{t \in F} [h(x|t) = y] \quad (1)$$

→ Iverson bracket

Randomized Training

- Extremely randomized trees algorithm
  - Independent training
  - On a random subset of the training set $D \subseteq \chi \times \Upsilon$
  - Recursive learning procedure
  - If $|D| < minimum\ size\ or\ E(D) < threshold$
    - $LF(\pi), \pi$: $the\ most\ represented\ class\ in\ D$

$$\pi \in \underset{z \in \Upsilon}{\operatorname{argmax}} \sum_{(x,y) \in D} [y = z] \qquad (2)$$

  - Else: $ND(\psi, t_l, t_r)$ is grown
    - Ψ: a test function selected from a randomly generated set Ψ
      - Maximizing the expected information gain about the label distribution due to the split $\left\{D_l^{\psi}, D_r^{\psi}\right\}$ of the training data

Randomized Training

$$\psi = \underset{\psi' \in \Psi}{\operatorname{argmax}} \{E(D) - E(D;\psi')\} = \underset{\psi' \in \Psi}{\operatorname{argmin}} E(D;\psi')$$

$$= \underset{\psi' \in \Psi}{\operatorname{argmin}} \{ \frac{\left|D_l^{\psi'}\right|}{|D|} E\left(D_l^{\psi'}\right) + \frac{\left|D_r^{\psi'}\right|}{|D|} E\left(D_r^{\psi'}\right) \}$$

- t_l and t_r:
  - Recursively grown with their respective training data D_l^{ψ} and D_r^{ψ}
- Unbalanced training data case:
  - Weighting each label $z \in \Upsilon$ according to the inverse class frequencies observed in the training data D

$$\left(\sum_{(x,y)\in D} [y = z]\right)^{-1}$$

  - Considering in the computation of the expected information gain
    - ✓ Reducing the class average prediction error

Random Forests in Computer Vision

- Classification tasks in the image domain
  - Anchoring the feature space to a pixel grid topology
  - Trained on a specific feature space χ
    - ✓ Extracted set of $d \times d$ patches from multi-channel images I
    - ✓ Channels: Including color features such as gradients, filter banks, etc.
  - A multi-channel training image:
    - ✓ 3D matrix I
    - ✓ $I_{(u,v,c)}$: the value at pixel (u, v) and channel c
  - A patch:
    - ✓ A triple $(u, v, I) \in \chi$
      - Coordinate (u, v) of the patch center in image I
  - The label space:
    - ✓ $\Upsilon = \{1,2, \ldots, k\}$: set of k object classes

Most Common Used Test Functions

$$\psi^{(1)}(x|\theta_1,\tau) = \left[I_{(u,v,0)+\theta_1} > \tau\right],$$
$$\psi^{(2)}(x|\theta_1,\theta_2,\tau) = \left[I_{(u,v,0)+\theta_1} - I_{(u,v,0)+\theta_2} > \tau\right],$$
$$\psi^{(3)}(x|\theta_1,\theta_2,\tau) = \left[I_{(u,v,0)+\theta_1} + I_{(u,v,0)+\theta_2} > \tau\right],$$
$$\psi^{(4)}(x|\theta_1,\theta_2,\tau) = \left[|I_{(u,v,0)+\theta_1} - I_{(u,v,0)+\theta_2}| > \tau\right],$$

- $\theta_i = (\delta u_i, \delta v_i, c_i)\,, \quad i = 1,2$
  - Displacement parameters relative to the patch center
  - Indexing a point in the patch
- $\tau \in \mathbb{R}$: Threshold

Test

- Random forest F has been trained
- Classification of a test image:
  - Labelling each pixel with the most probable class predicted by the forest, centered on the $d \times d$ patch

Structured Learning in Random Forest

- Input data in Traditional approaches
  - Assigned to single, *atomic* class labels
  - Acting as arbitrary identifiers
  - Without any dependencies among them
  - Quite noisy results

- Many computer vision problems
  - Limited model
    - ✓Inherently topological structure
    - ✓Rendering the class labels explicitly interdependent
    - ✓Awareness of the local topological structure

Structured Label Space

- $P = \Upsilon^{d' \times d'}$
  - Consists of $d' \times d'$ pixels
  - Patches of object class labels
  - $p_{ij} \in \Upsilon$: ij^{th} entry of the label patch $\boldsymbol{p}$
  - $index\ (0,0)$: Central position
  - *Feature patch* $x \in \chi$ and *label patch* $\boldsymbol{p} \in \boldsymbol{P}$
  - Association of $x = (u, v, I)$ and $\boldsymbol{p}$
    - ✓$\boldsymbol{p}$: Holding the labels of all pixels of image I within a $d' \times d'$ neighborhood of (u, v)
    - ✓p_{ij}: Label of pixel $(u + i, v + j)$
      - Note: d (size of feature patch) may differ from d' (size of label patch)
  - Training patches: $D \subseteq \chi \times P$

Structured Label Space

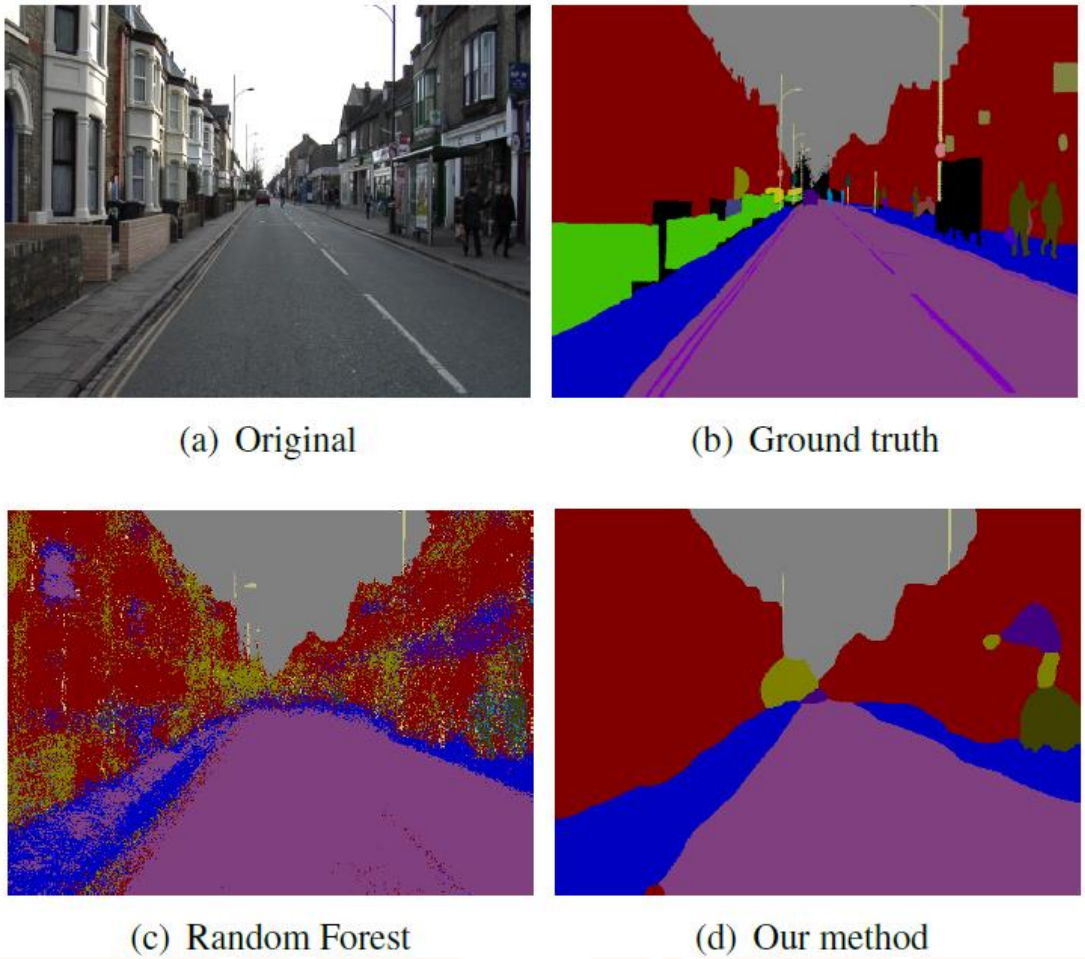
- $P_t \subseteq P$: Set of label patches used to grow the leaf t
- The class label π
  - Parametrizing the leaf
  - A structured label of size $d' \times d'$ from P
  - Not just an atomic label from Υ
  - Joint distribution of the label patches
- Low complexity for this step:
  - Pixel independent assumption

$$\Pr(\boldsymbol{p}|P_t) = \prod_{i,j} Pr^{(i,j)}(p_{ij}|P_t) \quad (3)$$

$$\pi = \underset{\boldsymbol{p} \in P_t}{\operatorname{argmax}} \Pr(\boldsymbol{p}|P_t) \quad (4)$$

Marginal class distribution over all the label patches of pixel (i, j)

Structured Label Space



(a) Original (b) Ground truth

(c) Random Forest (d) Our method

Fig. 2. Examples of object class segmentation using unary classifiers. Best viewed in color

Structured Label Space

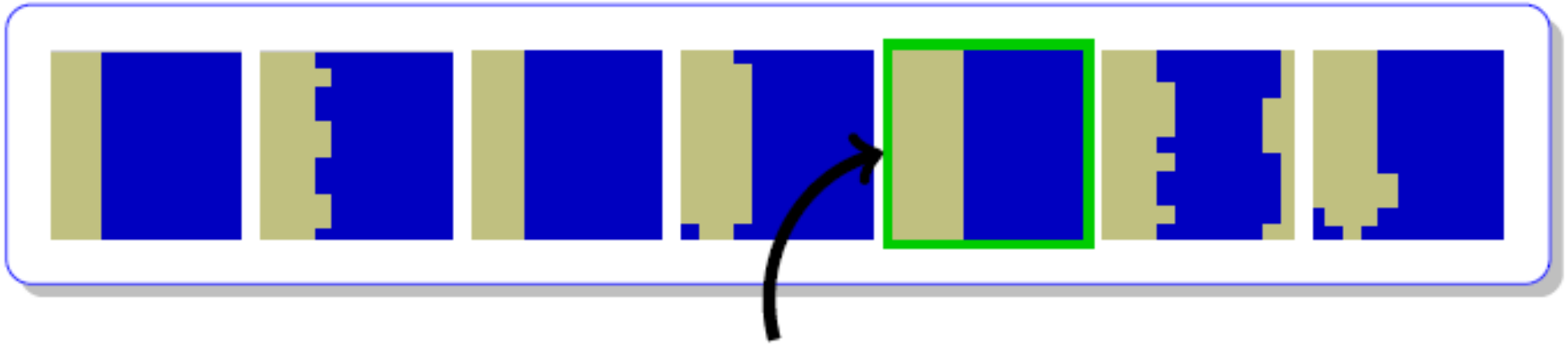


Fig. 3. Label patches reaching a leaf during training

Test Function Selection for Structured Labels

- Adaptation of the test function selection

- Naïve approach:
  - Porting the test selection criterion used in the standard random forest to our context
  - Simply associating each patch with the label we find in the center of the associated label patch **p**
  - Disadvantage: Like traditional random forest

- Clever approach
  - Selection based on the information gain with respect to a two-label joint distribution
  - Associating $(\boldsymbol{x}, \boldsymbol{p})$ with two labels:
    - ✓ First: Provided by p_{00}
    - ✓ Second: Provided by p_{ij}, position has been uniformly drawn
  - Advantage:
    - ✓ All entries of a label patch have the chance to influence the way a feature patch is branched
  - ✓ Disadvantage:
    - ✓ Increased complexity ($|\Upsilon|^2$ elements)

Test Function Selection for Structured Labels

- Solving the drawback:
  - Considering a different test function selection method
    - ✓Associating each training pair with just one label p_{ij}
      - All entries of the label patch still influence the learning procedure
      - At lower computational cost

Structured Label Predictions

- Gathering from the trees of a forest
  - Combining into single label patch prediction

- F: A trained forest
- $\boldsymbol{x} = (u, v, I)$: a test patch
- P_F: Set of predictions for $\boldsymbol{x}$ gathered from each tree $t \in F$

$$P_F = \{h(\boldsymbol{x}|t) \in P : t \in F\}. \quad (5)$$

- The label patch prediction for feature patch $\boldsymbol{x}$

$$\boldsymbol{p}^* = \underset{\boldsymbol{p} \in P_F}{\operatorname{argmax}} \Pr(\boldsymbol{p}|P_F), \quad (6)$$

Structured Label Predictions

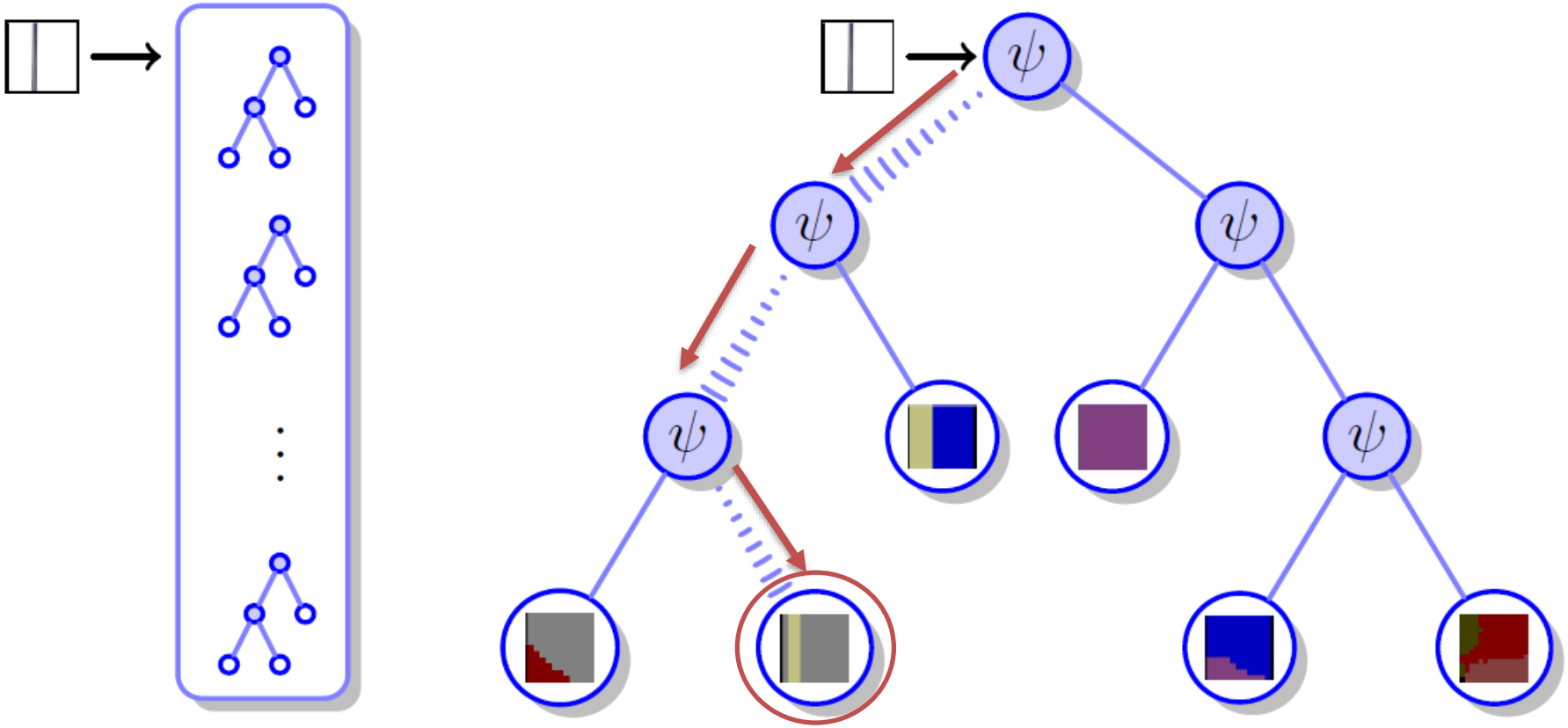


Fig. 4. Prediction of the structured label of a feature patch in a random forest.

Simple Fusion of Structured Predictions

- $p \in P$: Patch label predicted for pixel (u, v)
  - $(u + i, v + j)$ could be classified as $p_{ij} \in \Upsilon$
  - Collecting $d' \times d'$ class predictions
    - Have to be integrated into single class prediction
    - Simple way: Selecting the most voted class per pixel
- Outcome of fusion step: labeling $l \subseteq L$
  - $l_{uv} \in \Upsilon$: The class label associated with pixel (u, v)

Simple Fusion of Structured Predictions

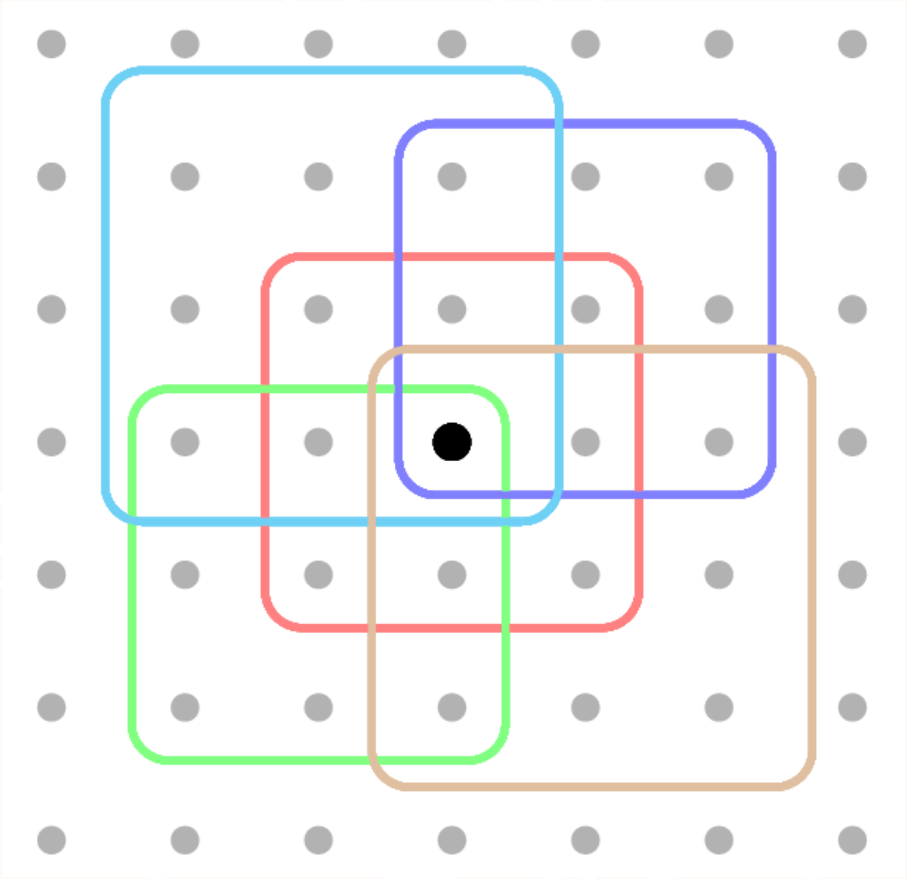


Fig. 5. Fusion of structured prediction

Optimizing the Label Patch Selection

- Different and more principled approach to the computation of the final labeling
  - Optimizing the label patch selection with respect to a given labelling

- *Agreement* of an individual label patch $\boldsymbol{p}$:
  - Located at $(i, j) \in I$
  - With a given labeling $\boldsymbol{l} \subseteq L$
  - $z \in Z_I$: An assignment of label patches to pixels in I
    - ✓ $z_{uv} \in P_F$: Label patch for pixel (u, v)

$$\varphi^{(i,j)}(\boldsymbol{p}, \boldsymbol{l}) = \sum_{(u,v)\in I} \left[p_{(u-i)(v-j)} = l_{uv}\right]. \quad (7)$$

Optimizing the Label Patch Selection

- *Total agreement:*

$$\Phi(\boldsymbol{z}, \boldsymbol{l}) = \sum_{(u,v)\in I} \varphi^{(u,v)}(z_{uv}, \boldsymbol{l}). \quad (8)$$

- Finding the label patch configuration that leads to the maximum total agreement with the labelling of a test image

$$(z^*, l^*) \in \underset{(z,l)}{\operatorname{argmax}}\{\Phi(z, l) | (z, l) \in Z_I \times L\}. \quad (9)$$

- Solve: iterative optimization method
  - Selecting the best agreement label patch per pixel
  - Producing a new labeling

Experiments

- CamVid and MSRCv2 databases
  - CamVid: 11 classes
    - ✓ ROAD, BUILDING, SKY, TREE, SIDEWALK, CAR, COLUMN POLE, SIGN-SYMBOL, FENCE, PEDESTRIAN, BICYCLIST
- 24×24 feature patch
- 10 trees
- 500 iterations
- Stop rule: Less than 5 samples per leaf available

1. CamVid Experiments

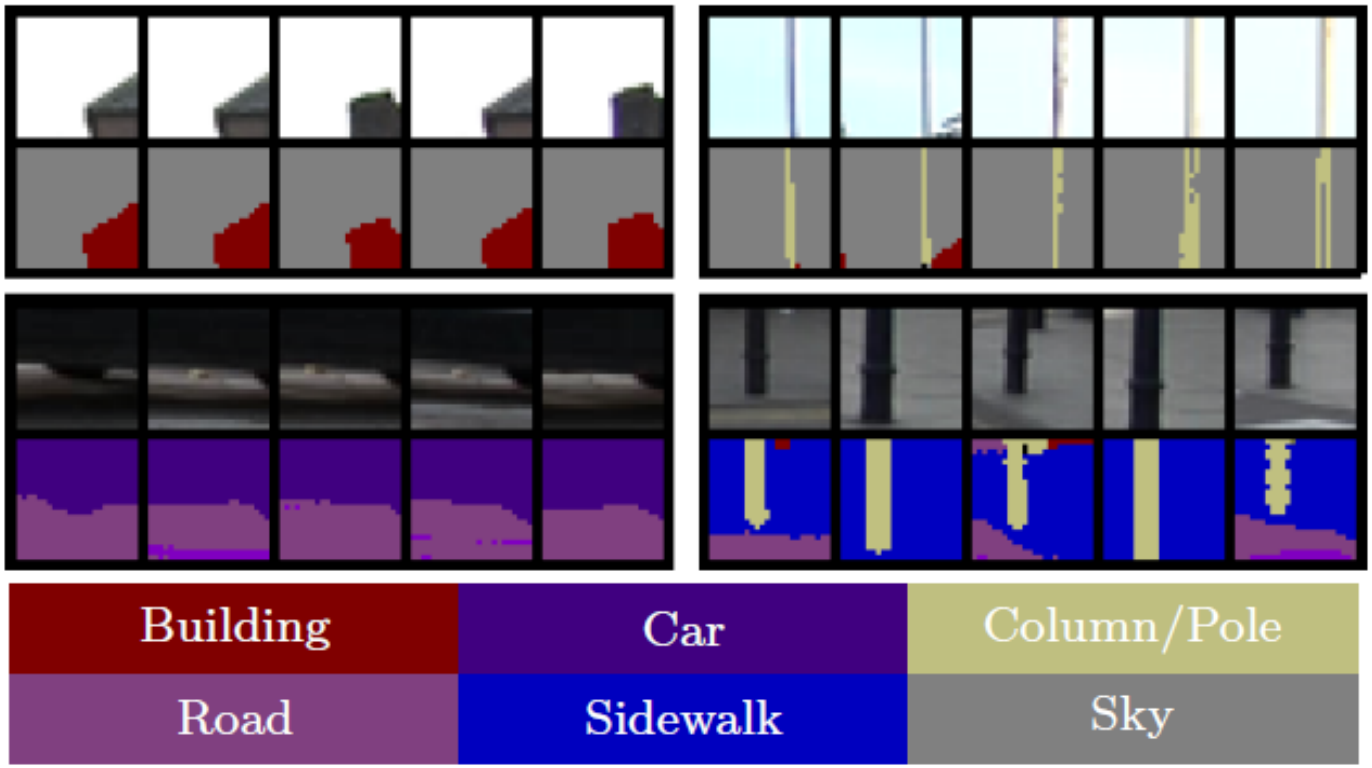


Fig. 6. Illustration of feature patches with corresponding label patches, collected from different leaf nodes when trained on CamVid database. Bottom rows: Label sets and associated colors.

1. CamVid Experiments

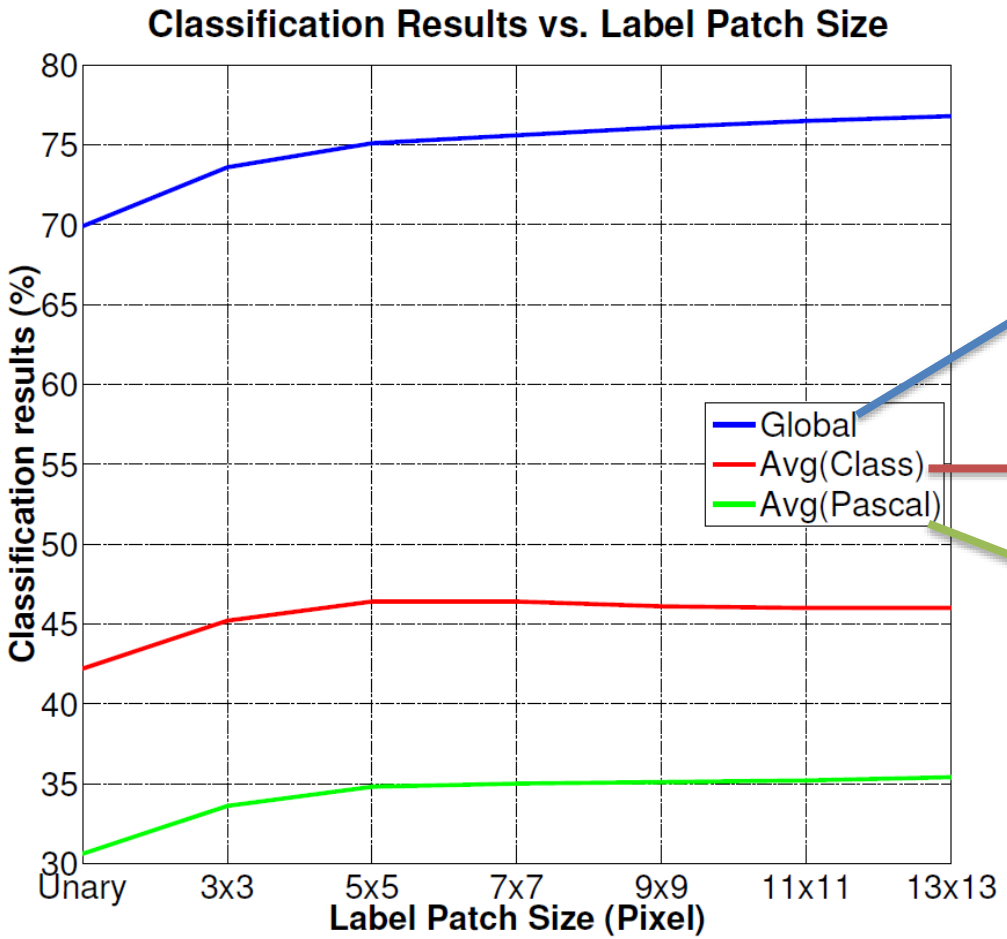
Method	*Global*	*Avg(Class)*	*Avg(Pascal)*
RF using Motion and Structure cues [9]	61.8	43.6	-
Our Baseline RF	69.9	42.2	30.6
Our Baseline RF + CRF	74.5	45.4	33.8
Our method (Structure + Simple Fusion)	74.8	45.0	34.1
Our method (Full + Simple Fusion)	76.8	46.1	35.4
Our method (Full + Optimized Selection)	79.2	46.0	36.2

Table 1. Classification results of CamVid database for label patch size 13×13

Method	*Global*	*Avg(Class)*	*Avg(Pascal)*
Our Baseline RF	63.8	44.2	29.8
Our Baseline RF + CRF	68.2	48.2	33.3
Our method (Structure + Simple Fusion)	69.9	50.4	35.0
Our method (Full + Simple Fusion)	71.6	50.1	35.8
Our method (Full + Optimized Selection)	72.5	51.4	36.4

Table 2. Classification results of CamVid database for label patch size 11×11

Influence of the Patch Size



Percentage of all pixels that were correctly classified

Average recall over all classes

Average intersection vs. union score

Fig. 7. Classification results of CamVid database as a function of the label patch size using Simple Fusion

2. MSRCv2 Experiments

Method	*Global*	*Avg(Class)*	*Avg(Pascal)*
Texton forests naïve (supervised) [26]	49.7	34.5	-
RF using covariance features [14]	55.8	42.2	-
Our Baseline RF	54.8	43.4	28.3
Our Baseline RF + CRF	61.0	52.8	35.1
Our method (Structure + Simple Fusion)	60.8	51.0	33.8
Our method (Full + Simple Fusion)	60.8	51.1	33.9
Our method (Full + Optimized Selection)	63.9	55.6	37.6

Table 3. Classification results of MSRCv2 database for label patch size 11×11

2. MSRCv2 Experiments

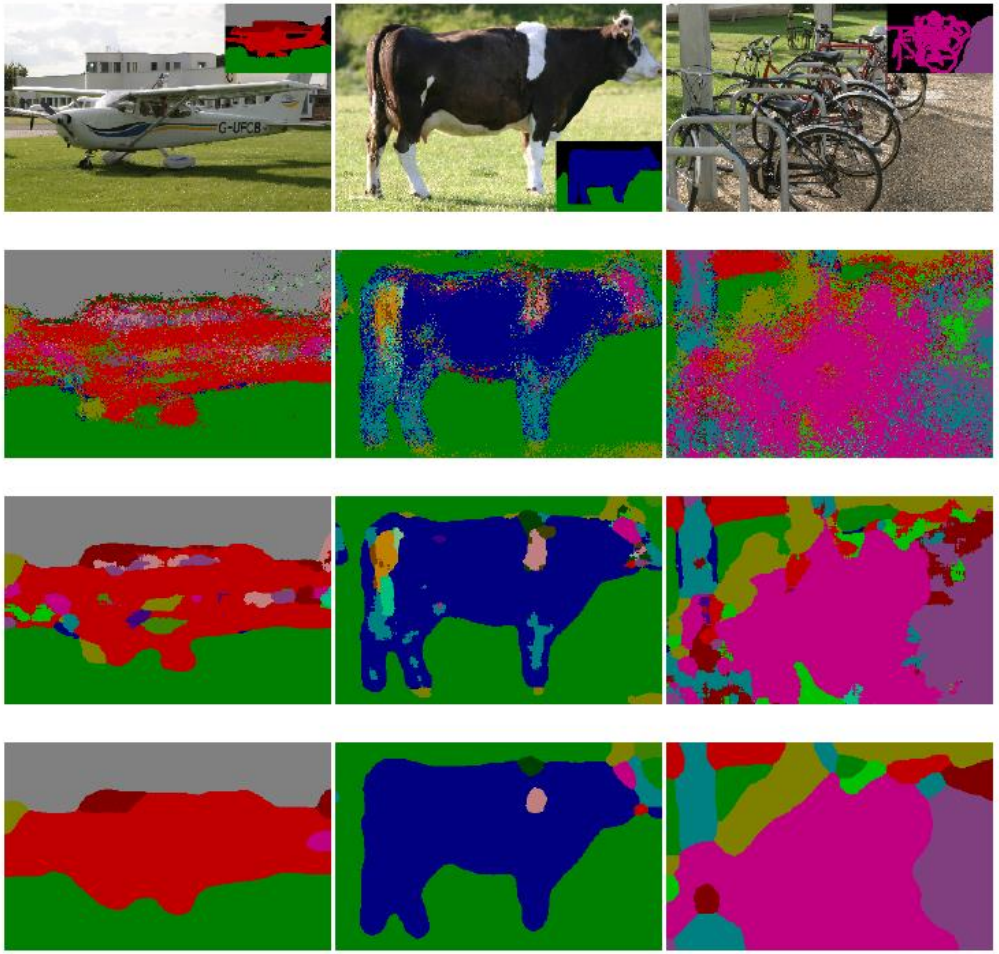


Fig. 8. Qualitative labelling results on images of the MSRCv2 database. Top row: Original images with ground truth annotations. Second row: Labelling using our baseline random forest classifier. Third row: Full + Simple Fusion. Last row: Full + Optimized Selection.

Conclusion

- Simple and effective way for semantic image labelling
  - Integrating ideas from structured learning into random forest framework
  - Incorporating the topology of the local label neighborhood in training process
  - Using topological joint label statistics of the training data in the node split functions
- Two possibilities for fusing the structured label predictions
  - Using overlapping predictions
  - Selecting most compatible label patches in the neighborhood

Structured Forests for Fast Edge Detection

Piotr Dollar, C. Lawrence Zitnick
Microsoft Research

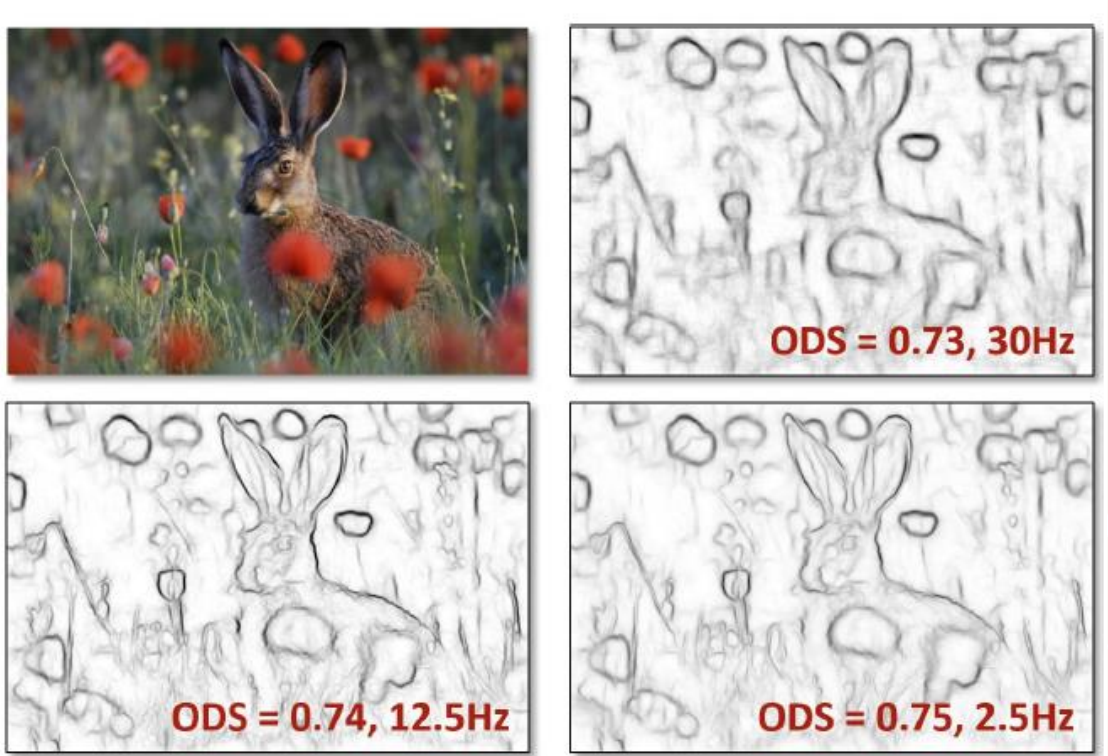
Caltech
CS159

Yury Tokpanov

May 9th, 2017

Introduction

- Create *segmentation map* – segment membership for each pixel.
- Or label each pixel, whether it contains edge or not – *edge map*.



Approach

- Random forest: train a bunch of trees on random samples + features, and then somehow combine their predictions.

- Training data consists of pairs of patches with corresponding segmentation masks (or edge maps).

✓ Can store any structured output in leaf nodes.
✓ Fast training and prediction, easy parallelization.

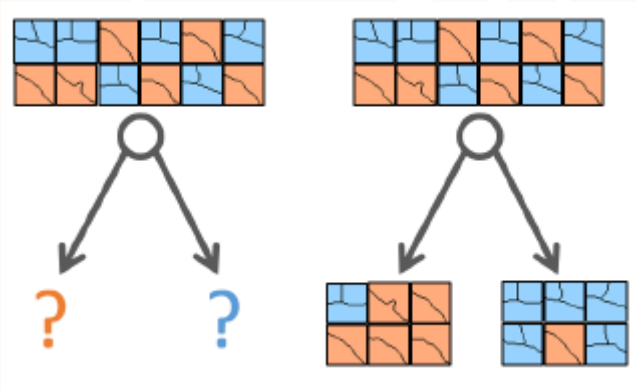
➢ Need a way of comparing structured labels.

❖ Biggest limitation: unable to synthesize novel labels, only labels observed in training can be predicted.

Problem formalization

- Image is divided into patches of size 32x32.
- Predict 16x16 segmentation mask (or edge map) for every patch.

- Training data consists of pairs of patches with corresponding segmentation masks (or edge maps).



PAMI 2015 paper of the same authors.

Image preprocessing

- 32 x 32 image patch.
  - Augmented by additional channels:
    - 3 color (CIE-LUV)
    - 2 gradient magnitude
    - 8 gradient orientation

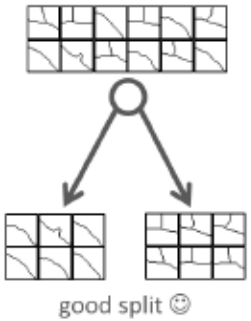
➢ $x \in R^{32 \times 32 \times 13}$

Input features

- Two types of features are used:
  - Pixel lookups $x(i, j, k)$
  - Pairwise differences $x(i_1, i_1, k) - x(i_2, i_2, k)$
- Total number of features per patch after downsampling and blurring:
  - $16 \cdot 16 \cdot 13 + \binom{5 \cdot 5}{2} \cdot 13 = 7228$

Output

- Output is structured: y = 16x16 segmentation map (or edge map).
- We want to use standard information gain criteria based on Shannon entropy or Gini impurity.



- Need to efficiently estimate similarities between segmentation masks (structured labels).
  1. Transformation $\Pi: Y \rightarrow Z$. Similarity defined in Z.
  2. Map structured labels $y \in Y$ into discrete set of labels $c \in C = \{1, \ldots, k\}$, such that labels with similar z have the same c.

Mapping to Z

- Transformation $\Pi: Y \rightarrow Z$.
- Compare pairs of pixels from two segmentation masks.
- Too many => use sampling of $m = 256$ pixels (empirically tested).

✓This also injects additional randomness into the learning process, helping with trees diversity.

Mapping to discrete labels

- Map z to c (discrete label), two options:
  a) Cluster z into k clusters using K-means.
  b) Top $\log_2 k$ PCA dimensions, assign according to orthant.
  - Both perform similarly.
  - After experimentation, they use PCA and $k = 2$.

Training

- ✓ Now we have the ability to cluster our training data.
- ✓ Standard splitting criteria can be used:

$$I_j = H(S_j) - \sum_{k \in \{L,R\}} \frac{|S_j^k|}{|S_j|} H(S_j^k)$$

$H(S) = -\sum_y p_y \log p_y$ (Shannon entropy)

OR

$H(S) = \sum_y p_y (1 - p_y)$ (Gini impurity)

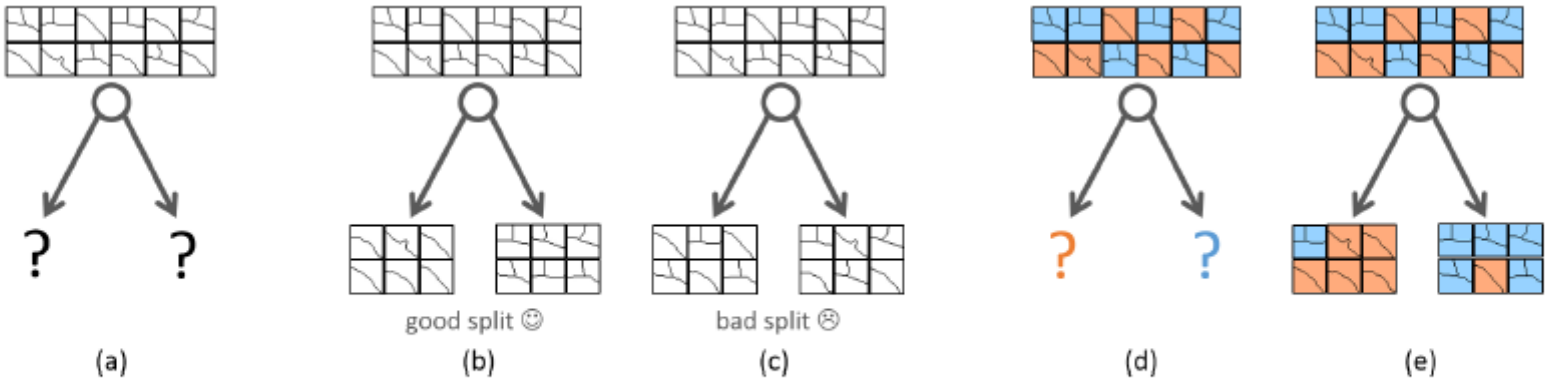


Fig. 2. Illustration of the decision tree node splits: (a) Given a set of structured labels such as segments, a splitting function must be determined. Intuitively a good split (b) groups similar segments, whereas a bad split (c) does not. In practice we cluster the structured labels into two classes (d). Given the class labels, a standard splitting criterion, such as Gini impurity, may be used (e).

PAMI 2015 paper of the same authors.

Ensembling model

- For edge maps: just average of tree predictions.
  - Use stride of 2 => each pixel receives $\frac{256T}{4} = 64T$ predictions.
  - To decorrelate predictions, train 2T total trees and evaluate an alternating set of T trees at each adjacent location.
- For segmentation mask: select y whose z is medoid (minimizes the sum of distances to all other z's).

Multiscale detection

- Run this structured edge detector on original, half, and double resolution of input image.
- Average results after resizing to original resolution.
- Improves edge quality.

Parameters

- Image and label patch size.
- Channels and feature parameters (e.g. blurring).
- Decision forests parameters (stopping criteria, number of trees T, mapping parameters *m* and *k*).

- Each tree is trained on one million randomly selected patches.

Results

- Accuracy measures:
  - fixed contour threshold (ODS), per-image best threshold (OIS), and average precision (AP).
- Standard non-maximal suppression technique applied to obtain thinned edges.
- Two datasets:
  - Berkeley Segmentation Dataset and Benchmark (BSDS 500).
  - NYU Depth dataset (v2), contains depth => 11 additional channels are used.

Example

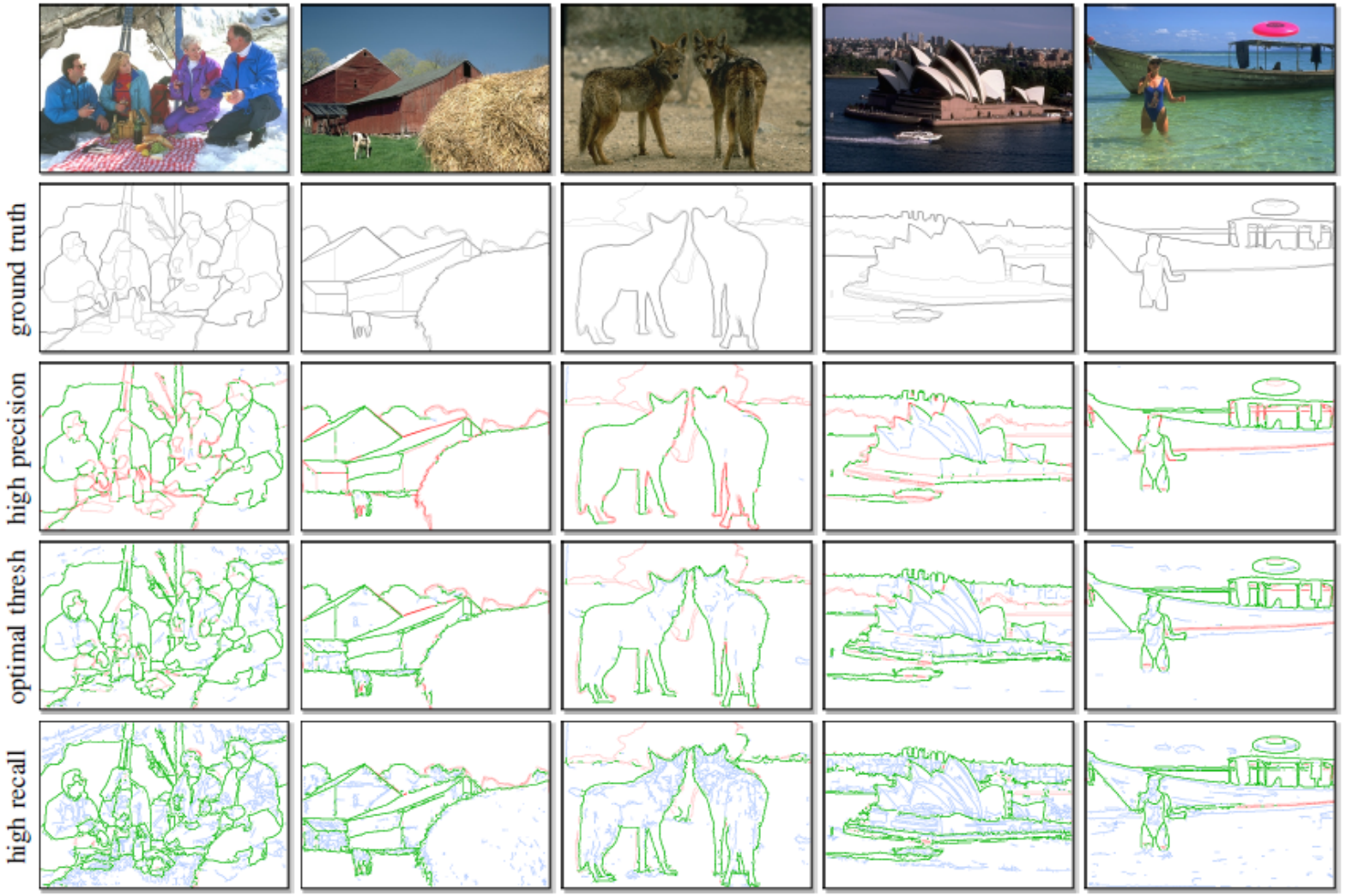


Fig. 4. Visualizations of matches and errors of SE+MS+SH compared to BSDS ground truth edges. Edges are thickened to two pixels for better visibility; the color coding is green=true positive, blue=false positive, red=false negative. Results are shown at three thresholds: high precision (T≈.26, P≈0.88, R=.50), ODS threshold (T≈.14, P=R≈.75), and high recall (T≈.05, P=.50, R≈0.93).

PAMI 2015 paper of the same authors.

BSDS 500

- The dataset contains 200 training, 100 validation and 200 testing images

PAMI 2015 paper of the same authors

BSDS 500

	ODS	OIS	AP	FPS
Human	.80	.80	-	-
Canny	.60	.64	.58	15
Felz-Hutt [11]	.61	.64	.56	10
Hidayat-Green [16]	.62†	-	-	20
BEL [9]	.66†	-	-	1/10
gPb + GPU [6]	.70†	-	-	1/2‡
gPb [1]	.71	.74	.65	1/240
gPb-owt-ucm [1]	.73	**.76**	.73	1/240
Sketch tokens [21]	.73	.75	**.78**	1
SCG [31]	**.74**	**.76**	.77	1/280
SE-SS, T=1	.72	.74	.77	**60**
SE-SS, T=4	.73	.75	.77	30
SE-MS, T=4	**.74**	**.76**	**.78**	6

Table 1. Edge detection results on BSDS500 [1]. Our Structured Edge (SE) detector achieves top performance on BSDS while being 1-4 orders of magnitude faster than methods of comparable accuracy. Three variants of SE are shown utilizing either single (SS) or multiscale (MS) detection with variable number of evaluated trees T. SE-SS, $T = 4$ achieves nearly identical accuracy as gPb-owt-ucm [1] but is dramatically faster. [†Indicates results were measured on BSDS300; ‡indicates a GPU implementation.]

NYU Depth dataset (v2)

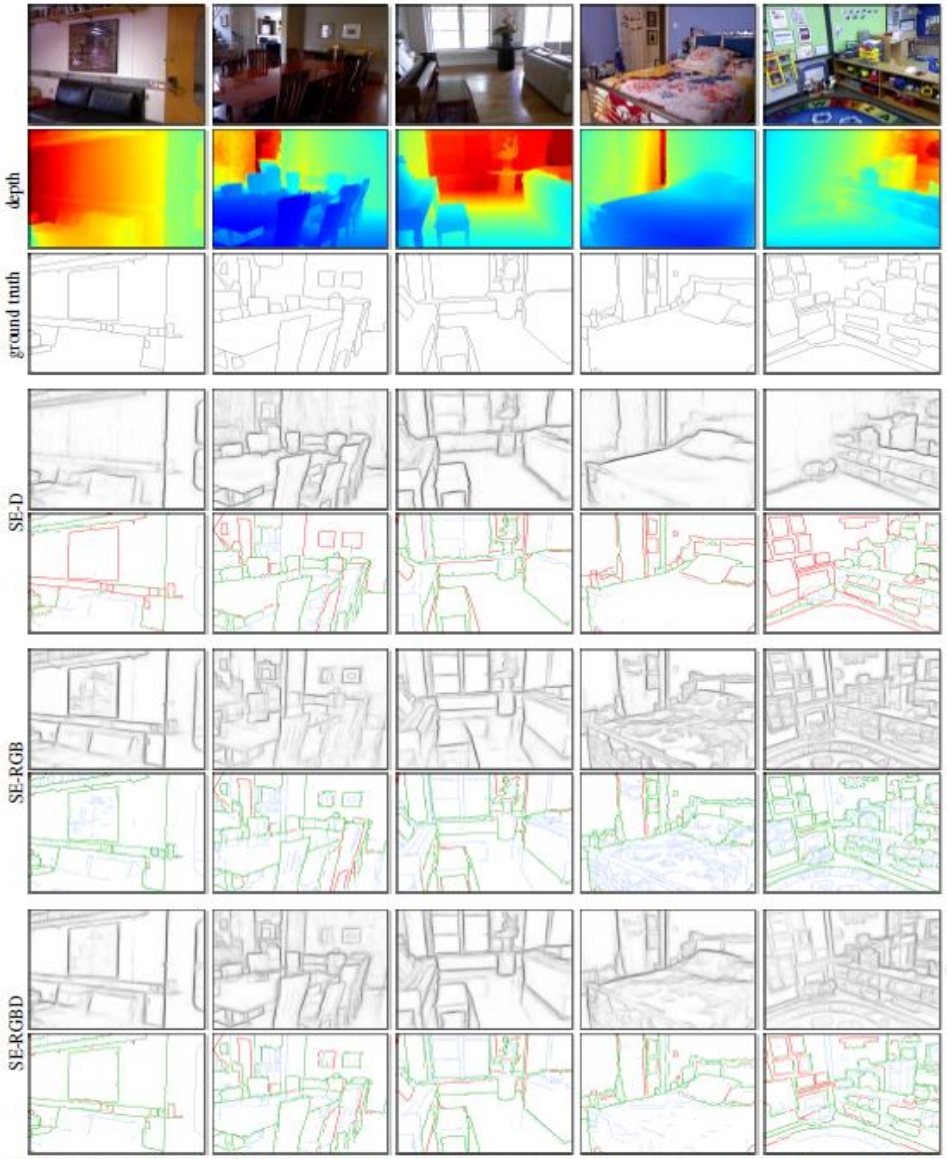


Fig. 12. Edge detection results on the NYUD dataset. Edges from depth features (SE-D) have good precision, edges from intensity features (SE-RGB) give better recall, and simultaneous use of intensity and depth (SE-RGBD) gives best results. For details about visualizations of matches and errors see Figure 4; all visualizations were generated using each detector's ODS threshold.

- The NYU Depth dataset (v2) contains 1,449 pairs of RGB and depth images with corresponding semantic segmentations.
- Image size 320x240.
- 11 additional channels due to depth.

Depth only

RGB only

RGB+depth

PAMI 2015 paper of the same authors

NYU Depth dataset (v2)

	ODS	OIS	AP	FPS
gPb [1] (rgb)	.51	.52	.37	1/240
SCG [31] (rgb)	.55	.57	.46	1/280
SE-SS (rgb)	.58	.59	.53	**30**
SE-MS (rgb)	**.60**	**.61**	**.56**	6
gPb [1] (depth)	.44	.46	.28	1/240
SCG [31] (depth)	.53	.54	.45	1/280
SE-SS (depth)	.57	.58	.54	**30**
SE-MS (depth)	**.58**	**.59**	**.57**	6
gPb [1] (rgbd)	.53	.54	.40	1/240
SCG [31] (rgbd)	.62	.63	.54	1/280
SE-SS (rgbd)	.62	.63	.59	**25**
SE-MS (rgbd)	**.64**	**.65**	**.63**	5

Table 2. Edge detection results on the NYU Depth dataset [33] for RGB-only (top), depth-only (middle), and RGBD (bottom). Across all modalities on all measures SE outperforms both gPb and SCG while running 3 orders of magnitude faster.

Cross-dataset generalization

	ODS	OIS	AP	FPS
BSDS /BSDS	.74	.76	.78	6
NYU / BSDS	.72	.73	.76	6
BSDS / NYU	.55	.57	.46	6
NYU / NYU	.60	.61	.56	6

Table 3. Cross-dataset generalization for Structured Edges. TRAIN/TEST indicates the training/testing dataset used. Our approach exhibits strong cross-dataset generalization, a critical component for widespread applicability.

Conclusions

- State-of-the-art accuracies.
- Real-time frame rates.
- Can be useful for time sensitive object recognition.

Deep Neural Decision Forests

Peter Kontschieder, Madalina Fiterau, Antonio Criminisi, Samuel Rota Bulo

DNN + decision forests

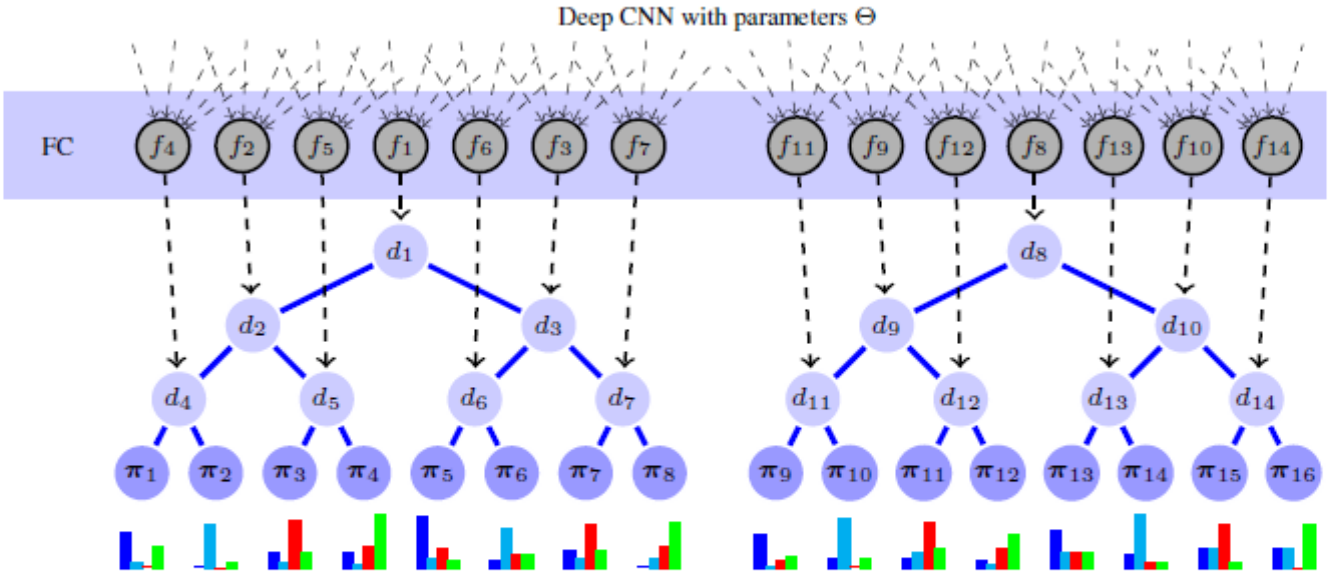


Figure 2. Illustration how to implement a deep neural decision forest (dNDF). Top: Deep CNN with variable number of layers, subsumed via parameters Θ. FC block: Fully Connected layer used to provide functions $f_n(\cdot;\Theta)$ (here: inner products), described in Equ. (3). Each output of f_n is brought in correspondence with a split node in a tree, eventually producing the routing (split) decisions $d_n(x) = \sigma(f_n(x))$. The order of the assignments of output units to decision nodes can be arbitrary (the one we show allows a simple visualization). The circles at bottom correspond to leaf nodes, holding probability distributions π_ℓ as a result from solving the convex optimization problem defined in Equ. (10).

Training

- ✓ Defined a new stochastic, differentiable framework for decision trees.
- ✓ Can be solved through SGD.
- ✓ Train iteratively simultaneously with NN.

Algorithm 1 Learning trees by back-propagation

Require: $\mathcal{T}$: training set, nEpochs

```
random initialization of Θ
for all i ∈ {1, ..., nEpochs} do
    Compute π by iterating (11)
    break T into a set of random mini-batches
    for all B: mini-batch from T do
        Update Θ by SGD step in (7)
    end for
end for
```

Most impressive result

Authors modified GoogleNet and used in their dNDF.

	GoogLeNet [36]						GoogLeNet⋆	dNDF.NET		
# Models	1			7			1	1		7
# Crops	1	10	144	1	10	144	1	1	10	1
Top5-Errors	10.07%	9.15%	7.89%	8.09%	7.62%	6.67%	10.02%	7.84%	7.08%	**6.38%**

Table 2. Top5-Errors obtained on ImageNet validation data, comparing our dNDF.NET to GoogLeNet(⋆).