Imperial College
London

EE462/EE9CS728/EE9SO25

Randomised Decision Forests for Visual Codebook

Tae-Kyun (T-K) Kim

Senior Lecturer

https://labicvl.github.io/

Notations

$\mathbf{d}$: descriptor vector

D : dimension of descriptor vector

N' : number of descriptor vectors

K : number of clusters or groups.

t_n : class label of $\boldsymbol{d}_n$

Visual codebook using randomized forests

Many popular methods for visual object categorisation represent images as collections of independent patches characterized by local descriptors.

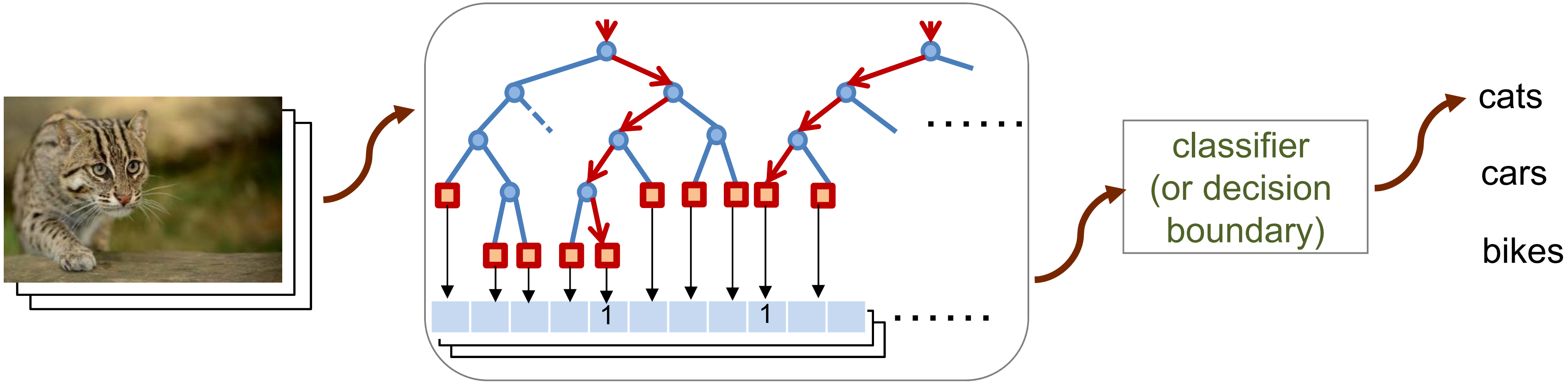
We will do this by

1) selecting or sampling (densely or randomly) patches from the image,

2) characterizing them by vectors of local visual descriptors,

– Various local descriptors exist with different degrees of geometric and photometric invariance, but all encode the local patch appearance as a numerical vector.

3) coding (quantizing) the vectors using a learned visual dictionary, i.e. a process that assigns discrete labels to descriptors, with similar descriptors having a high probability of being assigned the same label.



classifier (or decision boundary)

cats

cars

bikes

Visual codebook using randomized forests

- As in text categorization, the occurrences of each label (“visual codeword”) are then counted to build a global histogram (“bag of words”) summarizing the image (“document”) contents.
- The histogram is fed to a classifier to estimate the image’s category label.
- Unlike text, visual ‘words’ are not intrinsic entities and different quantization methods can lead to very different performances.
- Computational efficiency is important because a typical image yields 10^3 – 10^4 local descriptors and datasets often contain thousands of images.
- Also, many of the descriptors generally lie on the background not the object being classified, so the coding method needs to be able to learn a discriminative labelling despite considerable background ‘noise’.

K-means and tree structured codes

There are various methods for creating visual codebooks.

K-means clustering is currently the most common.

- Visual coding based on K-means vector quantization is effective but slow because it relies on nearest neighbor search, which remains hard to accelerate in high dimensional descriptor spaces.
- It is computationally expensive owing to the cost of assigning visual descriptors to visual codewords during training and testing.
- Nearest neighbour assignments can also be somewhat unstable: in high dimensions, concentration of measure tends to ensure that there are many centres with similar distances to any given point.

K-means: flat structure $O(k)$

tree structure: hierarchy $O(\log k)$

K-means and tree structured codes

Tree structured codes

- Component-wise decision trees offer logarithmic-time coding, but individual trees can rarely compete with a good K-means coding: each path through the tree typically accesses only a few of the feature dimensions, so there is little scope for producing a consensus over many different dimensions.
- Nister et al. 2006 introduced a tree coding based on hierarchical K-means. This uses all components (or features) and gives a good compromise between speed and loss of accuracy.
- Hierarchical k-means are quicker but less discriminative (due to overfitting).

Most methods are generative (i.e. unsupervised learning).

Some recent approaches focus on building more discriminative codebooks (i.e. supervised learning) in a tree structure, so to achieve both speed and good classification accuracy.

Randomized forests for visual codebook

data diversity

Random forests:

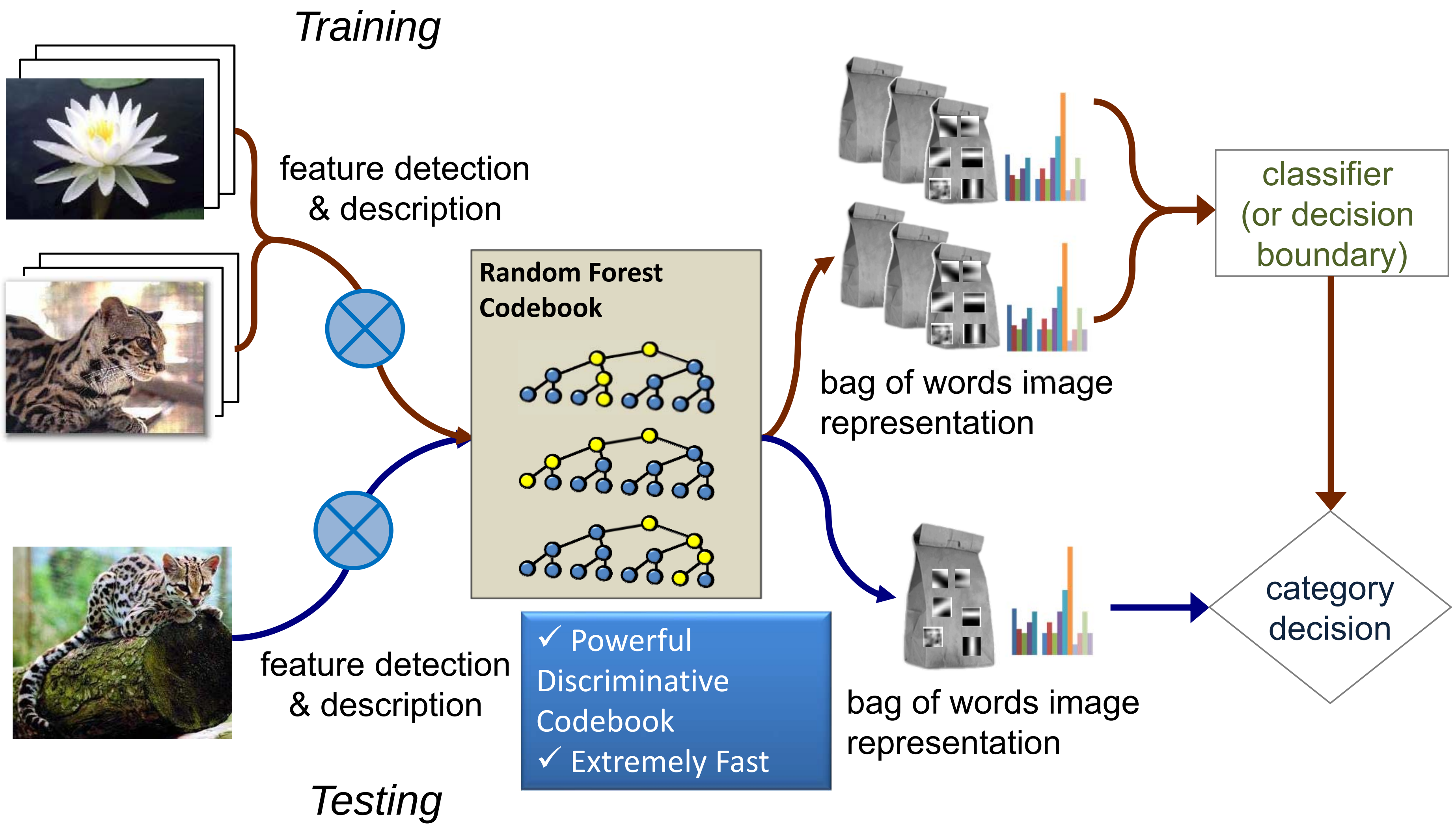
- Despite their popularity, we believe that K-means codes are not the best compromise.
- No single data structure can capture the diversity and richness of high dimensional descriptors. To do this an ensemble approach is needed.
- Ensembles of random trees seem particularly suitable for visual dictionaries owing to their simplicity, speed and performance.
- Sufficiently diverse trees can be constructed using randomized data splits or samples. Compared to standard approaches such as C4.5, randomised tree construction is rapid, depends only weakly on the dimensionality and requires relatively little memory.

Randomized forests for visual codebook

Two main benefits in using randomised forests

- 1. (Small) ensembles of trees eliminate many of the disadvantages of single tree based coders without losing the speed advantages of trees.
- 2. Classification trees contain a lot of valuable discriminative information in descriptor space that helps the subsequent classification problem for object categorisation.
  - One can exploit this by training them for classification then ignoring the class labels and using them as “clustering trees” - simple spatial partitioners that assign a distinct region label (visual codeword) to each leaf.

It has been shown that these have good resistance to background clutter and that they provide much faster training and testing and more accurate results than conventional k-means in state-of-the-art image classification tasks.



Training

feature detection & description

Random Forest Codebook

bag of words image representation

classifier (or decision boundary)

feature detection & description

bag of words image representation

category decision

Testing

✓ Powerful Discriminative Codebook
✓ Extremely Fast

Randomized forests for visual codebook

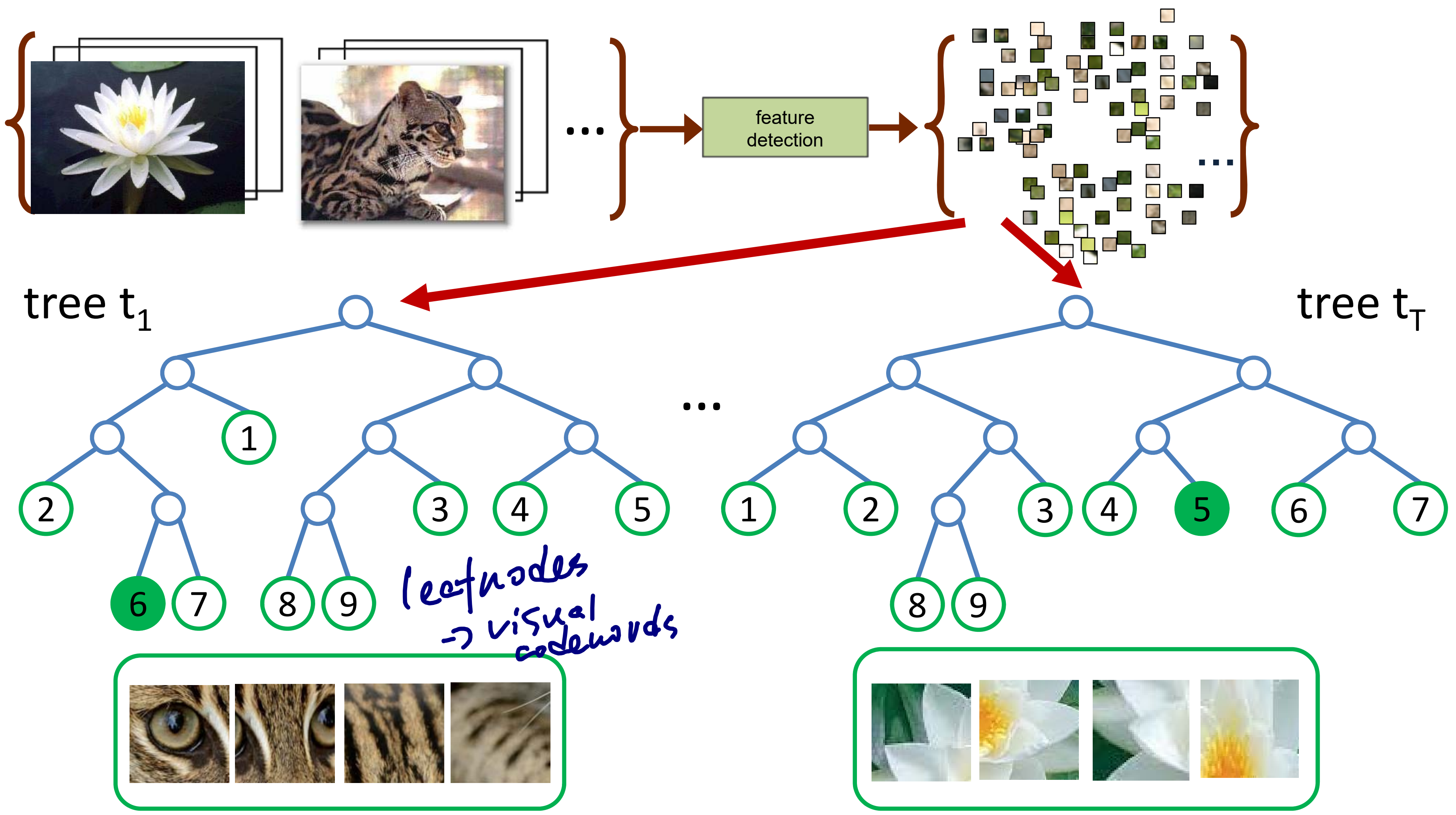
Training:

- Our goal is to build a discriminative coding method.
- The method starts by building randomized decision trees that predict class labels t from visual descriptor vectors $\boldsymbol{d} = [d^1, d^2, \dots, d^D]^T$, where d^i, i = 1,...,D are elementary scalar features.
- For simplicity we assume that all of the descriptors from a given image share the same label t. Or, we provide pixel-wise labels.
- We train the randomised decision trees using a labelled (for now) training set $\{(\boldsymbol{d}_n, t_n)$, n = 1,..., $N'\}$.
- The trees are built recursively top down.
- At each node, corresponding to descriptor space, two children nodes are created by choosing a Boolean test that divides into two disjoint regions.
  - Descriptor: raw pixel intensities or certain descriptors
  - Split function: axis-aligned or two-pixel tests
  - Split objective: Shannon entropy
- Recursion continues until certain stopping criteria are met.
- The parameters control the strength and randomness of the generated trees.

Randomized forests for visual codebook

Training:

$\{(\boldsymbol{d}_n, t_n)$, n = 1,..., $N'\}$



leaf nodes → visual codewords

Example patches

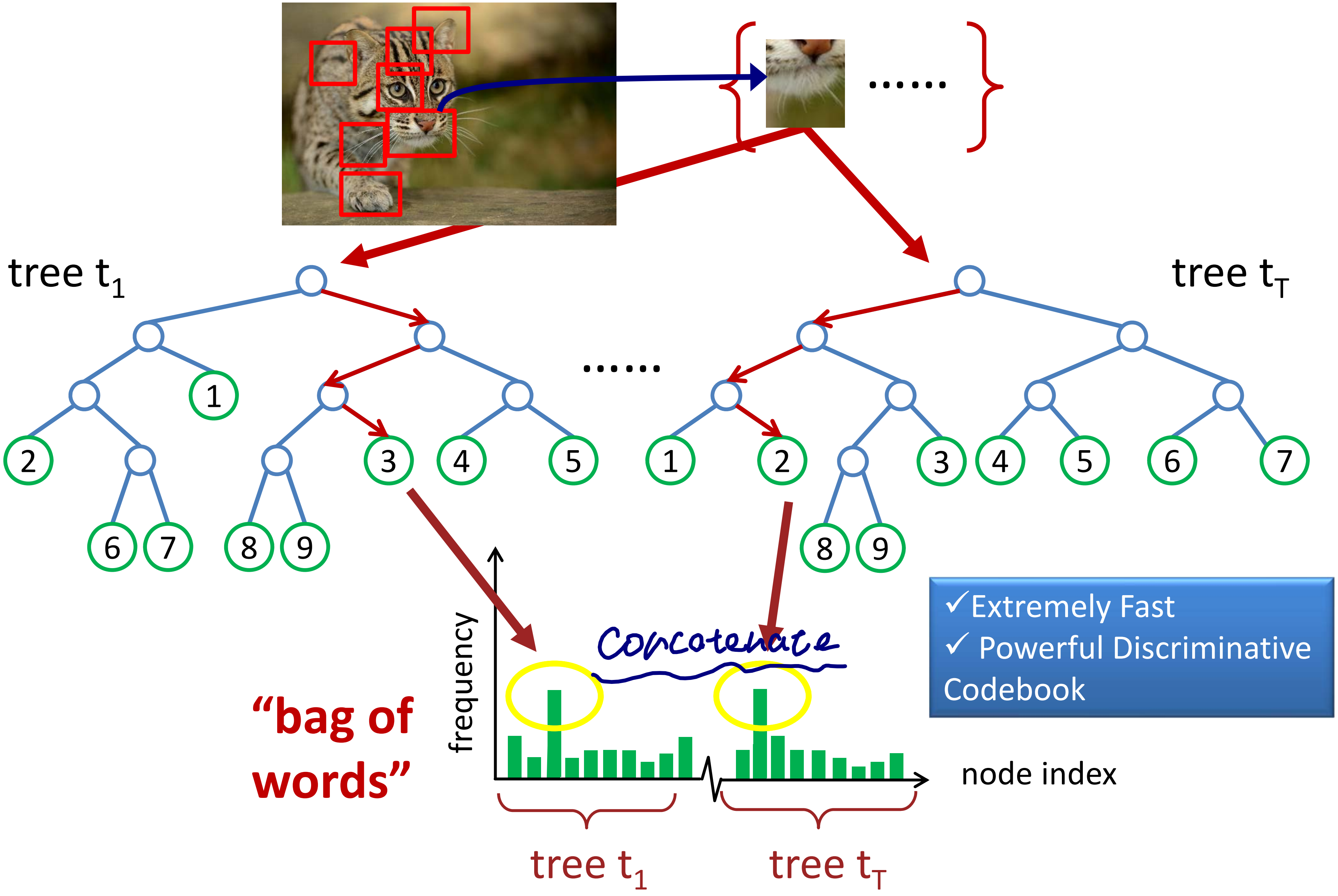
Example patches

Randomized forests for visual codebook

Testing (vector quantisation):

- During a query, for each descriptor tested, each tree is traversed from the root down to a leaf and the returned label is the unique leaf index, not the (set of) descriptor label(s) t associated with the leaf.
- In use, the trees transform each descriptor into a set of leaf node indices with one element from each tree.
- Votes for each index are accumulated into a global histogram and used for classification as in any other bag of features approach.
- Independently of the codebook, the denser the sampling the better the results, so typically we sample images more densely during testing than during codebook training.

Randomized forests for visual codebook (vector quantisation)



"bag of words"

✓Extremely Fast
✓ Powerful Discriminative Codebook

Randomized forests for visual codebook

size too big (large codebook
sparse distribution

Testing (vector quantisation):

- Compared to standard decision tree learning, the trees built using random decisions are larger and have higher variance.
- Class label variance can be reduced by voting over the ensemble of trees, but here, instead of voting we treat each leaf in each tree as a separate visual word and stack the leaf indices from each tree into an extended code vector for each input descriptor, leaving the integration of votes to the final classifier.
- Each tree is responsible for distributing the data across its own set of clusters.
- Experimentally, the trees appear to be rather diverse while still remaining relatively strong, which should lead to good error bounds.
- Codebook size (number of trees, and tree depths) is important.

Computational complexity

Training:

K-means has a complexity of O(DN’K), where N’ is the number of patches and K is the number of clusters (or leaf nodes). D is the descriptor dimensionality.

RF has O(√DN’logK).

K : # codewords
comparisons

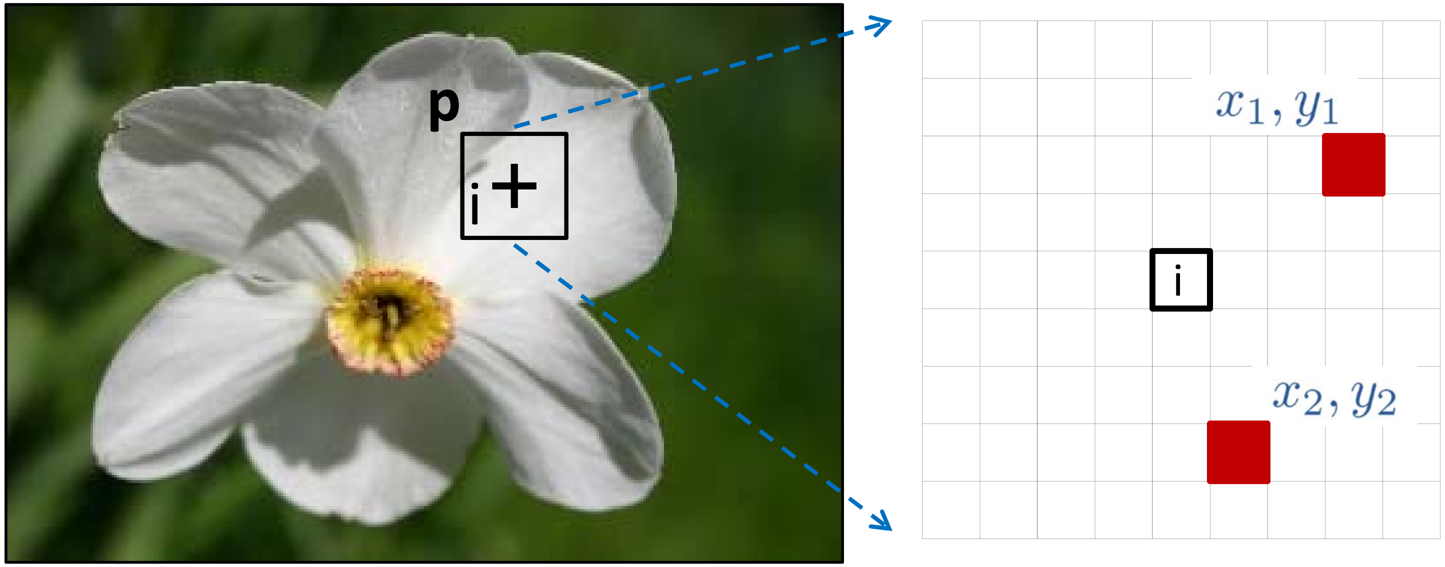
- The worst-case complexity (imbalanced trees) for building a tree is O(T_{max}N’K), where T_{max} is the number of split function trials (i.e. $|\mathcal{T}_j|$ in the random forest lecture).
- In practice, we obtain well balanced trees at the complexity of around O(T_{max}N’logK).
- The dependence on data dimensionality D is hidden in the constant T_{max}, which needs to be set large enough to filter out irrelevant feature dimensions, thus providing better coding and more balanced trees. A value of T_{max} ~ O(√D) has been suggested (Breiman 2001), leading to a total complexity of O(√DN’logK).

In the experiment, the complexity of K-means is more than 104 times larger when we use 768-D wavelet descriptor with N’= 20000 image patches and K = 5000 clusters, not counting the number of iterations that k-means has to perform.

Testing:

RF is also faster in use: a useful property given that reliable image classification requires large numbers of image patches to be labelled. Labelling a descriptor with a balanced tree requires O(logK) operations whereas k-means costs O(KD).

Image Patch Features



Pixel i **gives patch p**

(21x21 pixels in experiments)

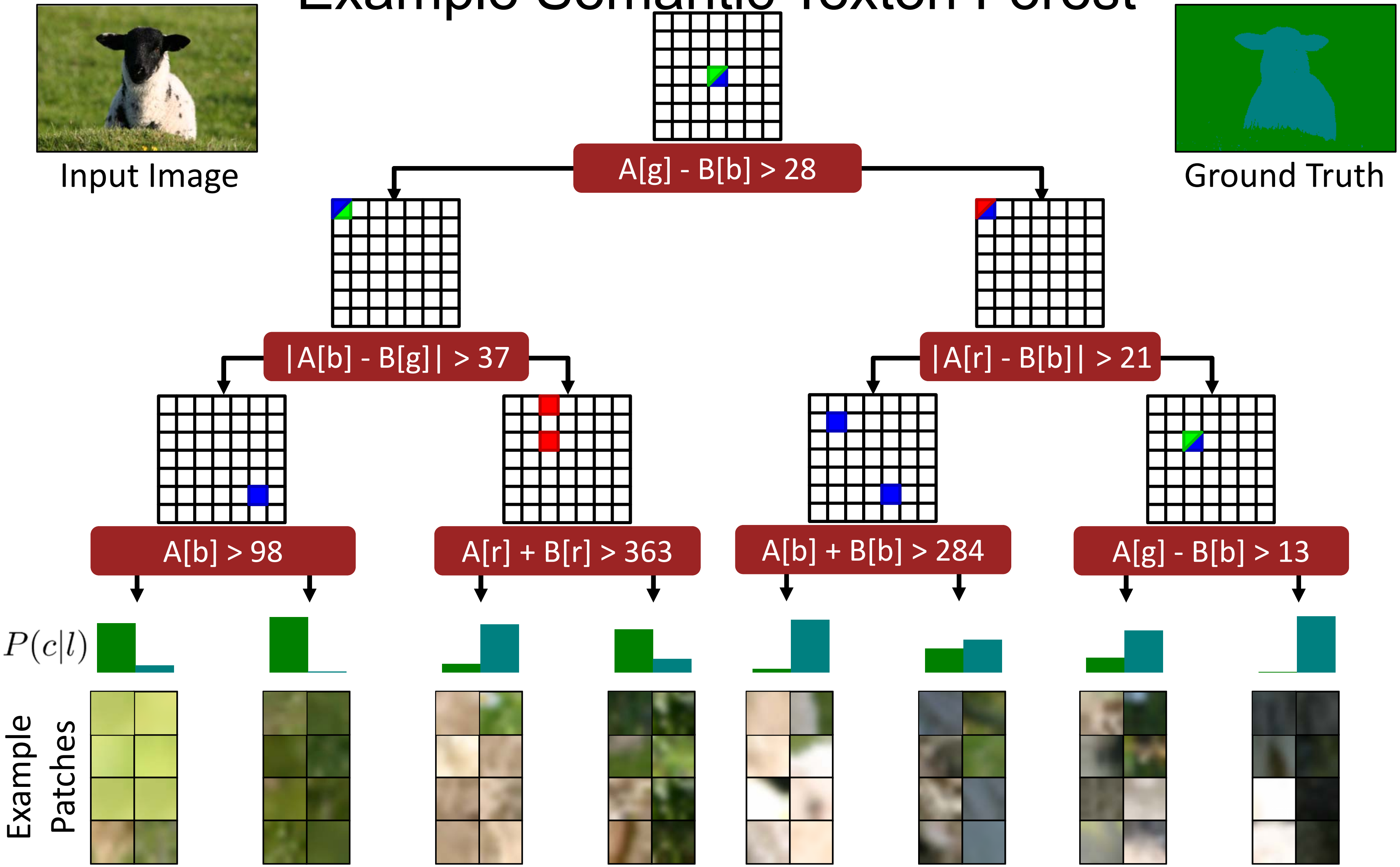
$$
\begin{aligned}
f(\mathbf{p}) &= p_{x_1,y_1,c_1} \\
f(\mathbf{p}) &= p_{x_1,y_1,c_1} + p_{x_2,y_2,c_2} \\
f(\mathbf{p}) &= p_{x_1,y_1,c_1} - p_{x_2,y_2,c_2} \\
f(\mathbf{p}) &= \left| \, p_{x_1,y_1,c_1} - p_{x_2,y_2,c_2} \, \right|
\end{aligned}
$$

$$\underbrace{f(\mathbf{p}) \underset{?}{>} \begin{matrix}\text{learned}\\ \text{threshold}\end{matrix}}_{\text{tree split function}}$$

in pixel space

Examples from Semantic Texton Forest
(Shotton et al. 2008)

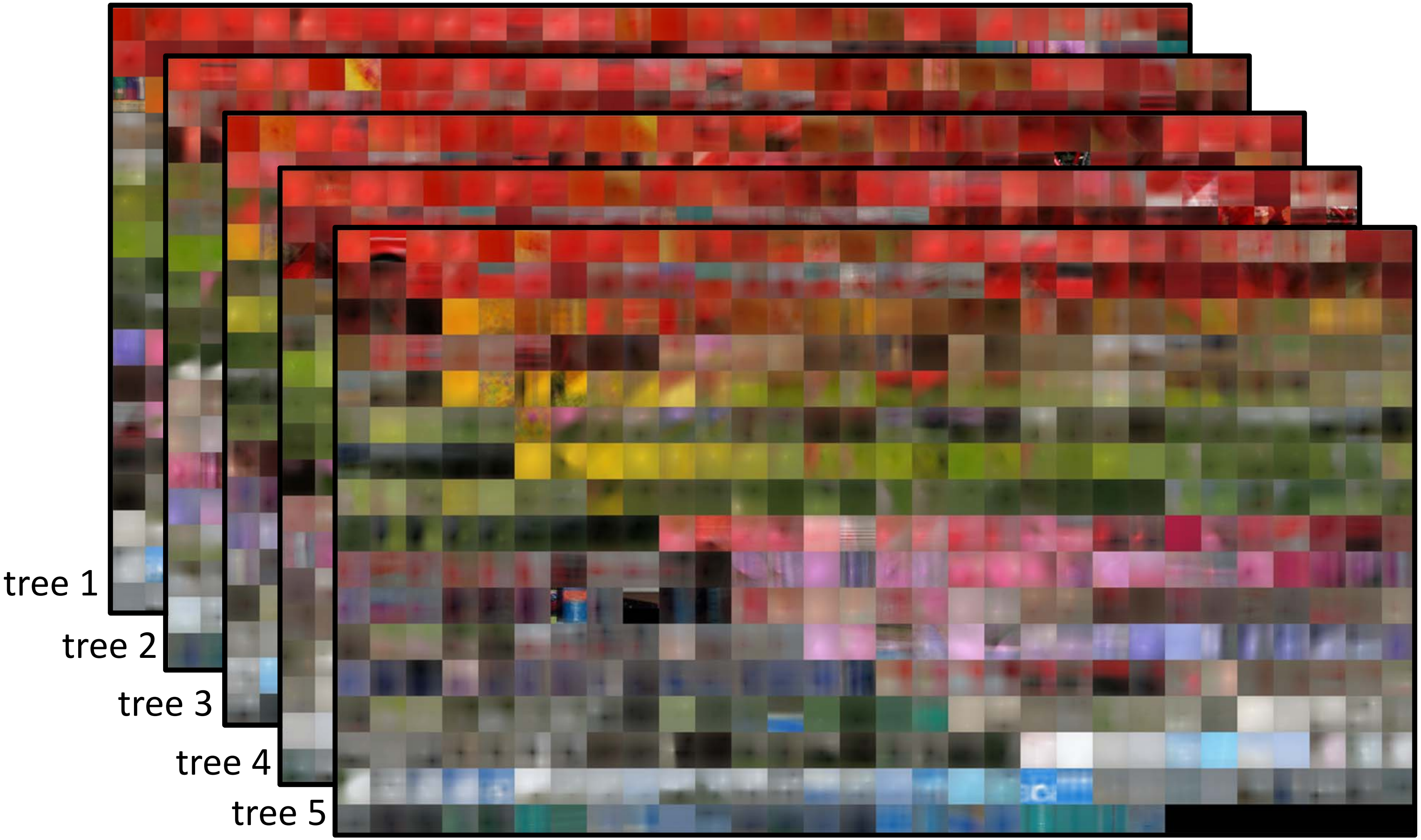
Example Semantic Texton Forest



Examples from Semantic Texton Forest (Shotton et al. 2008)

Leaf Node Visualization

Average of all training patches at each leaf node



Examples from Semantic Texton Forest
(Shotton et al. 2008)