LARGE MULTI-CLASS IMAGE CATEGORIZATION WITH ENSEMBLES OF LABEL TREES

Yang Wang

Department of Computer Science
University of Manitoba

David Forsyth

Department of Computer Science
University of Illinois

ABSTRACT

We consider sublinear test-time algorithms for image categorization when the number of classes is very large. Our method builds upon the label tree approach proposed in [1], which decomposes the label set into a tree structure and classify a test example by traversing the tree. Even though this method achieves logarithmic run-time, its performance is limited by the fact that any errors made in an internal node of the tree cannot be recovered. In this paper, we propose *label forests* – ensembles of label trees. Each tree in a label forest will decompose the label set in a slightly different way. The final classification decision is made by aggregating information across all trees in the label forest. The test running time of label forest is still logarithmic in the number of categories. But using an ensemble of label trees achieves much better performance in terms of accuracies. We demonstrate our approach on an image classification task that involves 1000 categories.

Index Terms— image classification, large-scale learning

1. INTRODUCTION

Image categorization is one of the most fundamental problems in computer vision. One of the challenges of this problem is the sheer number of object/scene categories available in our visual world. Evidence from psychology suggests that people can recognize at least tens of thousands of objects and scenes. In the computer vision literature, a lot of progress has been made in recognizing objects and scenes with hundreds of categories. But we are still far from what people can recognize.

Recently, large-scale image collections with tens of thousands of categories are becoming available. A representative example is ImageNet [6]. The availability of such large-scale image collections has enabled research on image categorization with very large number of categories (e.g. [5, 12]). As datasets keep growing, a key challenge is how to cope with the large number of categories. For most current approaches (e.g. multi-class SVM with one-vs-rest strategy), test time grows at least linearly with the number of categories. For many multimedia applications, e.g. image search or photo annotation, linear run-time is simply too slow when we have a very large number of concepts. To make image categorization with large numbers of classes practical, we need algorithms whose test time is sublinear in the number of categories.

What makes sublinear run-time algorithms possible? We believe the answer lies in the fact that the categories are correlated. Imagine we have tens of thousands of object models. A naive approach will run all the models and classify an image according to their scores. The test time is linear to the number of categories (i.e. object models). However, if we run a "car" model on an image and get a very low score, we most likely do not need to run the "truck" model. The reason is that cars and trucks are similar objects, so an image of a truck should probably give a reasonably high score for a car model. Similarly, if a "car" model gives a very high score on an image, we probably do not need to run the "dog" model since cars and dogs are very dissimilar.

One specific form of correlation of categories is their taxonomy, which is also related to the concepts of *basic level*, *sub-ordinate level* and *super-ordinate level* categories in psychology [15]. Most work uses tree-structured taxonomies, e.g. [14, 1, 8]. Our work is most related to the label tree approach in [1]. In that work, a binary tree is constructed to represent the set of category labels. Each node in the tree corresponds to a subset of labels. The label set of a parent node is partitioned into several disjoint subsets, one for each child node. Each leaf node in the tree corresponds to a class label. Classifying a new example involves traversing the label tree from the root to a leaf node. At each node, a binary classifier is used to decide which child node (left or right) to traverse. If the label tree is balanced, it can be shown that the run-time of classifying an example is $O(\log Y)$ where Y is the number of categories.

Tree-structured taxonomies, such as label tree, have a distinctive advantage in terms of computational efficiency. If we decide to traverse the left child at a node in the tree, we never have to visit the right child or any of its descendants. This is the key property of tree structures that allow $O(\log Y)$ test time. But the problem of tree structures is that if we make a mistake at a node, the mistake will be propagated to a leaf node and cannot be recovered. This means the tree structure (i.e. how to split label set at each node) in a label tree is critical. However, we do not have a "ground-truth" tree structure for organizing all the visual concepts in the world. In fact, it is hard to believe that there exists a tree structure that can

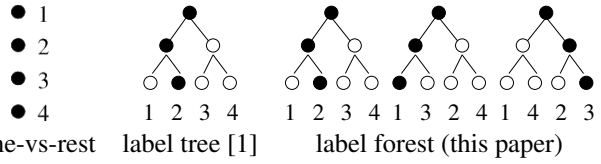


Fig. 1. Illustration of different strategies for a synthetic multi-class classification problem with 4 categories. In the one-vs-rest strategy, classification involves evaluating a binary classifier for each class (indicated by black circles). In the label tree approach [1], category labels are organized as leaf nodes in a binary tree. Classification involves traversing a path in the tree. In our label forest approach, we use several trees to organize category labels in different ways. Classification involves traversing a path in each tree, and combining the information together.

perfectly represent our visual world.

In this paper, we address these issues by using an ensemble of tree structures. We call our approach the *label forest*. Each tree in a label forest represents a different way of constructing the taxonomy of categories. The classification of a new example involves combining information across all the trees. Since we combine several trees in the end, we can recover mistakes made by a single tree and potentially achieve higher accuracies. However, if there are K trees in the label forest, we now have to evaluate K trees instead of one tree. Although the run time of a label forest is still $O(\log Y)$ asymptotically, the actual test time will increase by a constant factor of K compared with a single tree. We demonstrate that, with this small cost in test time, we achieve significantly better results than using a single tree. In addition, the label forest can outperform the naive one-vs-rest strategy in terms of both accuracy and efficiency. Figure 1 illustrates the difference between one-vs-rest, label tree, and label forest for multi-class classification problems.

The rest of this paper is organized as follow. Section 2 introduces label tree, which is a way of building a tree structure for representing category labels. Section 3 gives details on how to build a sequence of label trees to form the label forest. We present experimental results on a large multi-class image categorization task in Sec. 5 and conclude in Sec. 6.

2. LABEL TREE: TREE-STRUCTURED LABEL TAXONOMY

We briefly review the label tree approach in [1]. Denote the set of all possible class labels by $\mathcal{L}$. Here we assume $|\mathcal{L}|$ is very large, e.g. in the order of thousands. A label tree $\mathcal{T} = (\mathcal{V}, \mathcal{E}, \{\ell_v\}_{v\in\mathcal{V}})$ defines a hierarchical partition of $\mathcal{L}$. Each vertex $v \in \mathcal{V}$ in the tree is associated with a label set $\ell_v \subseteq \mathcal{L}$. For each non-leaf node v, the label sets of its child nodes defines a partition of ℓ_v into several disjoint label sets. In this paper, we assume binary label trees, i.e. each node has at most two children. Let $p, q \in \mathcal{V}$ be the two children of a node v, we have $\ell_p \bigcap \ell_q = \emptyset$ and $\ell_v = \ell_p \bigcup \ell_q$. The root label set ℓ_0 contains all the class labels, i.e. $\ell_0 = \mathcal{L}$. A leaf label set contains a single label. Each non-leaf node v is also associated with a predictor f_v. Given an example x, the predictor f_v will decide which child label set (left or right) it belongs to. Since we assume a binary tree, each predictor f_v is a binary classifier.

To classify an example, we need to traverse the label tree from the root to a leaf. At each non-leaf node, we use its predictor to decide whether to choose left or right child node, and repeat the process until we reach a leaf node. If the tree is balanced, classifying an example only involves evaluating $\log|\mathcal{L}|$ binary classifiers, as opposed to $|\mathcal{L}|$ binary classifiers in a standard one-vs-rest classification scheme.

In [1], each leaf node is associated with a single label. An example reaching a leaf node is classified using the class label associated with this leaf node. In our work, we adopt a simple modification of the label tree which will allow us to develop the label forest algorithm in Sec. 3. Instead of having a single label associated with a leaf node, we encode a distribution of class labels $p_v(y) : \forall y \in \mathcal{Y}$ at each leaf node v. To classify the example reaching a leaf node in a label tree, we can choose the class y^* that has the highest probability according to the distribution associated with this leaf node, i.e. $y^* = \arg\max_y p_v(y)$. See Algorithm 1.

Given the structure (i.e. the split of label set at each node) of a label tree, we learn a binary linear SVM as the predictor for each non-leaf node. The training examples whose class labels belong to the left-child (right-child) label set of a node are considered as positive (negative) examples for learning the predictor for this node. See Algorithm 2. Note that here we train the SVM classifiers at each node independently. One can also use more sophisticated learning algorithm (e.g. the tree loss optimization [1]) that learns all the SVM classifiers jointly, but this is slow to train.

We then pass training examples from the root to leaf nodes. At each node, we send training examples to the left/right child node depending on the outputs of the corresponding predictor. For each leaf node, we learn the class probability associated with it from training examples reaching this leaf node. The probability of a particular class is learned as the proportion of training images (at this leaf node) belonging to that class.

Note that the label tree in [1] is in fact a special case of ours, where the distribution is a Dirac delta function, i.e. $p_v(y) = 1$ if y is the class label corresponding to the leaf node v, and $p_v(y) = 0$ otherwise.

3. LABEL FOREST: ENSEMBLE OF LABEL TREES

The limitation of the label tree approach in [1] is that any wrong decision made at a non-leaf node will propagate to the leaf, and there is no way to recover it. To overcome this lim-

Algorithm 1 Prediction using a label tree

input : test example x, probabilistic label tree $\mathcal{T}$
  $v = 0$
  repeat
    if $f_v(x) > 0$ **then**
      $v = left\_child(v; \mathcal{T})$
    else
      $v = right\_child(v; \mathcal{T})$
    end if
  until v is a leaf node of $\mathcal{T}$
  Return $p_v(y)$

Algorithm 2 Learning the predictors (for non-leaf nodes) and the class distributions (for leaf nodes) given the structure of a label tree

input : A label tree structure $\mathcal{T}$ and a set of training data (x_i, y_i)
  repeat
    $v_l = left\_child(v; , \mathcal{T})$
    $v_r = right\_child(v; , \mathcal{T})$
    $S^+ = \{x_i : y_i \in \ell_{v_l}\}$
    $S^- = \{x_i : y_i \in \ell_{v_r}\}$
    $f_v = learn\_svm(S^+, S^-)$
  until v is a leaf node of $\mathcal{T}$
  Learn class distribution p_v for each leaf v by passing training examples through the tree and count the ratio of examples of each class reaching this leaf

itation, we introduce *label forests*. A label forest is an ensemble of K label trees. For a new test image x, the t-th label tree will output a distribution p^t of class labels (cf. Algorithm 1). The final prediction is chosen as the class label that has the highest probability averaged over all label trees, i.e. $y^* = \arg\max_y \sum_{t=1}^{K} p^t(y)$. It is also possible to learn a weight w^t associated with the t-th tree to reweight its distribution $p^t(y)$. We will leave this as future work.

The label forest aggregates information over several label trees to make the final prediction. If one label tree makes an error, it might not be detrimental as long as the majority of the label trees make the correct predictions.

Now the key challenge is to learn the ensemble of label trees. In Sec. 2, we introduced an algorithm for learning the predictors (for non-leaf nodes) and class distributions (for leaf nodes) of a label tree, assuming the structure (i.e. the split of label sets at each node) is known. Now we describe how to obtain the structure for each tree in the label forest.

As pointed out in [1], the learnability of a label tree is implicitly influenced by how we split the label sets at each node. Intuitively if some classes are often confused, we should not split them into two branches too early. For example, consider four class labels {*cat, dog, car, bus*} and two possible splits at the root node: (1) {*cat, dog*} versus {*car, bus*}; (2) {*cat, car*} versus {*dog, bus*}. The first split seems more learnable.

To capture this intuition, Bengio et al. [1] propose to learn the tree structure by first learning a standard multi-class SVM classifier, then apply the classifier on a validation set to get a confusion matrix C. An entry C_{ij} of this matrix indicates how often class i is confused with class j. They then recursively partition the label set using spectral clustering according to a symmetric affinity matrix $A = \frac{1}{2}(C + C^\top)$. Note that since A encodes how much confusion there is between every pair of labels, the spectral clustering will try to partition the label sets between classes that there is little confusion. In addition, since spectral clustering penalizes unbalanced partitions, the resulting tree structure tends to be balanced, which is crucial to achieve the logarithmic run-time.

In our case, we need a tree structure for every label tree in the label forest. Note that we cannot use the same tree structure for all label trees, since this will cause each label tree to produce the same answer. A possible solution is to use the idea of randomization, i.e. randomly splitting the label set into equal sizes at each node. Due to the randomness, each tree will have a different structure. However, as we will demonstrate in Sec. 5, tree structures obtained from random splitting usually have poor learnability, so it is hard to achieve good accuracies out of them. In the following, we propose an alternative approach for learning a sequence of tree structures.

We use an iterative algorithm for learning an ensemble of label trees. During each iteration, the algorithm uses the label tree learned from previous iteration to generate a new affinity matrix, which is then used to create the tree structure for the current iteration. The algorithm is summarized in Algorithm 3. We highlight some important steps of the algorithm in the following.

Our algorithm bears some resemblance to the boosting algorithm for classification, which iteratively learn a ensemble of weak learners. In our problem, we can consider a label tree as a "weak learner". In boosting, after learning the t-th weak learner, the $(t+1)$-th weak learner is learned from a set of weighted training examples. The weight of each sample is adjusted at each iteration depending on whether this example is correctly classified by the t-th weak learner.

We use a similar strategy for learning an ensemble of label trees. However, instead of using the t-th weak learner to reweight training example, we use the t-th label tree to obtain a new affinity matrix used for generating the tree structure for the $(t+1)$-th label tree. The algorithm is summarized in Algorithm 3. We highlight some important steps of the algorithm:

Initialization: We need an affinity matrix A between pairs of labels in order to create the tree structure. In our method, we use the distances between labels in the WordNet hierarchy to construct this matrix for the first label tree. Let d_{ij} be the distance between labels i and j in the WordNet hierarchy. We construct a matrix $A^{(0)}$ in order to learn the structure of the first label tree. The i, j-th entry $A^{(0)}$ of this matrix is defined as $A_{ij} = \exp(-\lambda \cdot d_{ij}^2)$.

Even though WordNet is a pure language-based taxonomy, there is evidence [5] suggesting that there is a correlation

between the structure of the semantic hierarchy of WordNet and visual confusion between categories. So the matrix $A^{(0)}$ constructed from the WordNet hierarchy will capture some information about visual confusion.

Note that the method in [1] uses the confusion matrix on validation data to build the matrix C. In order to get this confusion matrix, they have to learn a multi-class SVM classifier in the first place. When the number of categories is large, learning a multi-class SVM itself is time-consuming.

Learning $(t+1)$-th label tree: When learning the $(t+1)$-th label tree, we would like it to capture the visual confusion between categories that are not captured by previous trees. More specifically, we construct a confusion matrix $C^{(t)}$ from the t-th label tree and define an affinity matrix $A^{(t)} = \frac{1}{2}(C^{(t)} + {C^{(t)}}^\top)$. We then recursively run spectral clustering (see [1] for details) on $A^{(t)}$ to get the tree structure for the $(t+1)$-th label tree. By doing so, two categories that are often confused from previous label trees will likely not to be split apart too early, which is the behavior we want.

Algorithm 3 Learning the label forest

input A set of training data $\{x_i, y_i\}$, an initial affinity matrix $A^{(0)}$ created from WordNet hierarchy, the maximum number K of label trees in the label forest.

1) Generate the tree structure for the 1st label tree $\mathcal{T}^{(1)}$ by recursively split label sets according to the affinity matrix $A^{(0)}$.

2) Use Algorithm 1 to learn the parameters (i.e. predictors $f_v^{(1)}$: $\forall v$ is non-leaf, and class distributions $p_v^{(1)}$: $\forall v$ is leaf) of the 1st label tree

3) Compute the confusion matrix $C^{(1)}$ using the 1st label tree

for $t = 2 \rightarrow K$ **do**

4) Generate a tree structure by recursively splitting label sets according to the affinity matrix $A = C^{(t-1)} + {C^{(t-1)}}^\top$

5) Use Algorithm 2 to learn the parameters (i.e. $f_v^{(t)}$ and $p_v^{(t)}$) of the t-th label tree $\mathcal{T}^t$

6) Compute the confusion matrix $C^{(t)}$ using the t-th label tree

end for

Return the ensemble of K label trees

4. RELATED WORK

Multi-class classification is a very well-studied area in machine learning. Most approaches in this area use one-vs-rest or one-vs-one to convert the multi-class into a set of binary classification problems. However, the test time of these approaches grows at least linearly with the number of categories, which is prohibitive when dealing with very large numbers of categories.

There are several popular methods of exploiting the correlation of categories to speedup the run time. The first method is based on label embedding, i.e. projecting the category labels into another (either smaller, or possibly continuous) space. For example, Fergus et al. [7] propose *label sharing* that project categories into a continuous space by taking into account the WordNet distance between two categories. Hsu et al. [10] use compressed sensing based approaches to embed class labels as continuous vectors. Exploiting label taxonomy or hierarchy is another popular approach for dealing with a large number of categories. In the domain of text analysis, there has been work (e.g. [4]) on hierarchical classification of documents. In computer vision, we usually do not have a pre-defined visual taxonomy. But work has been done on data-driven approaches for building visual taxonomy in order to organize object/scene categories [14, 1, 11].

Our work is also related to random forests [3, 9], which consist of a set of decision trees. Each leaf node encodes a distribution of class labels. Each non-leaf node of a decision tree consists of a binary test that splits the data and sends them to it child nodes. The binary test at each non-leaf node is constructed via randomization. The class label of a new test image is obtained by combining the posterior class distribution from each decision tree.

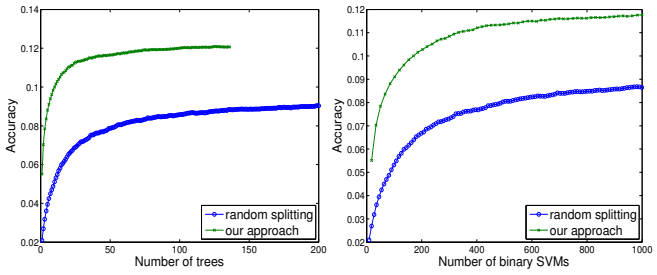
5. EXPERIMENTS

We test our approach on a subset of ImageNet [6] collected for the large scale visual recognition challenge [2]. The task is to classify images from 1000 object categories. We should emphasize that this is a very challenging task, and chance performance is only 0.1% in terms of accuracy.

We use the image features provided in [2]. The features are computed by extracting SIFT descriptors [13] on dense overlapping grids, then vector-quantized to 1000 visual words. Each image is represented as a 1000 dimensional vector using the standard bag-of-words representation. See [2] for more details.

In Fig. 2(a), we plot the overall accuracy as a function of the number of label trees in the ensemble. We show the results of using random splitting and using our method (cf. Algorithm 3). There are several conclusions we can draw from the results. First, using multiple label trees always outperforms a single label tree. Second, our algorithm for learning the tree structure outperforms random splitting. A standard multi-class SVM achieves 9% on this task [2]. For random splitting, it does reach this accuracy in the end when the number of label trees is large (close to 200). With our approach, we can reach and outperform SVMs with just a very small number (e.g. 10) of label trees.

For each label tree, the number of binary classifiers we need to evaluate for a new test image depends on the maximum depth of the tree in the worst case. For random splitting, we have split the label sets into equal sizes at each node, so the tree is guaranteed to be balanced. The maximum depths of all label trees are identical. But for our algorithm, even though the spectral clustering will penalize unbalanced splitting, there is no guarantee that the tree is completely balanced. A possibly fairer comparison is to examine the accuracy ver-



(a) (b)

Fig. 2. The results of using random splitting and our method (Algorithm 3). We show two plots: (a) test accuracy versus the number of trees in the label forest; (2) test accuracy versus the number of binary SVMs need to be evaluated (worst case) during run time. The number of categories is 1000, so the standard one-vs-rest strategy will need to evaluate 1000 binary SVMs.

method	accuracy(%)	speedup
Our method (10 trees)	10	$\times 5.88$
Multi-class SVM	9* [2]	$\times 1$

Table 1. Comparison of our method (using 10 trees) and the multi-class SVM trained using one-vs-rest strategy on the ImageNet challenge dataset. With 10 trees, we already achieve better accuracy. In addition, the number of binary SVM classifiers we need to evaluate during testing is 5.88 times smaller than the multi-class SVM. *This number is provided in [2].

sus the total number of binary SVM evaluations needed (in the worse case) for a new test image. The total number of binary SVM evaluations needed is simply the sum of maximum depths of all trees. We plot this information in Fig. 2(b).

We can compare our results with the standard multi-class SVM classifier trained using one-vs-rest strategy. The results are shown in Table 1. Note that for the multi-class SVM, we need to evaluate 1000 binary SVMs for a test image. If we use the first 10 label trees returned by our algorithm, we already achieve better accuracy than one-vs-rest multi-class SVM. With 10 trees, the total number of binary SVMs we need to evaluate is 170 (i.e. the sum of maximum depths of these 10 trees). So we have achieved $1000/170 = 5.88$ speedup.

In order to get some insights about how labels are split in the label forest, we use the following visualization method. For a given label tree, we can define the distance between a pair of category labels i and j as the length of the path between two leaf nodes i and j. We then define the distance $d(i, j)$ between i and j by averaging their distance across the 10 trees used in the label forest. We exclude the first tree which is created by recursive spectral clustering based on the WordNet distance matrix. Figure 3 (a) visualizes the distance between every pair of category labels. It has a very interesting block diagonal structure that separates the first 500 categories from the last 500 ones. This is intuitive because the first 500 categories are mostly natural living things (plants, animals, etc.), and the last 500 categories are mostly man-made artifacts. The diagonal block structure in Fig. 3(a) indicate that categories of living things are being put close (in term of tree distance) to each other in the label trees (so do categories of man-made artifacts), but are put far apart from categories of man-made artifacts. It is informative to compare Fig. 3(a) to the tree distance directly obtained from the WordNet hierarchy shown in Fig. 3(b). It is interesting to see that our learning method automatically recover some of the structure in WordNet. As a comparison, we visualize the tree distance (averaged over 10 trees) obtained from randomly splitting label trees (Fig. 3(c)). Not surprisingly, the tree distances do not reveal any interesting structures.

It is informative to examine elements in Fig. 3(a) that have dramatic changes from the corresponding elements in Fig. 3(b). These elements will reveal the correlations of categories that are not well captured by WordNet. In Fig. 4, we show example pairs of categories whose distances have dramatic changes between WordNet distance (Fig. 3(b)) and the tree distance of the learned label forest (Fig. 3(a)). The first two examples (hazelnut↔kidney bean, lentil ↔soy) are categories that are far apart in the WordNet hierarchy, but are close in the learned label forest. The last two examples (jigsaw puzzle↔crossword puzzle, bearskin↔bonnet) are categories that are close in the WordNet hierarchy, but are far apart in the label forest. We can see that the results make intuitive sense. For example, "hazelnut" and "kidney bean" are far apart in the WordNet hierarchy, but they are visually very similar. In contrast, "jigsaw puzzle" and "crossword puzzle" are similar semantically (i.e. close in the WordNet hierarchy), but visually they are very different.

6. CONCLUSION

We have presented label forests, ensembles of label trees, for dealing with a large number of class labels in image categorization. Our approach is an extension of the label tree method [1]. Our experimental results demonstrate that using multiple label trees performs better than using a single tree. By using a small number of trees, we have shown that we can outperform the multi-class SVM in terms of both accuracy and speed.

Acknowledgement: YW would like to thank the funding support of NSERC and the University of Manitoba. We thank Weilong Yang for helpful discussions.

7. REFERENCES

[1] S. Bengio, J. Weston, and D. Grangier. Label embedding trees for large multi-class tasks. In *NIPS*, 2010.

[2] A. Berg, J. Deng, and L. Fei-Fei. ImageNet large

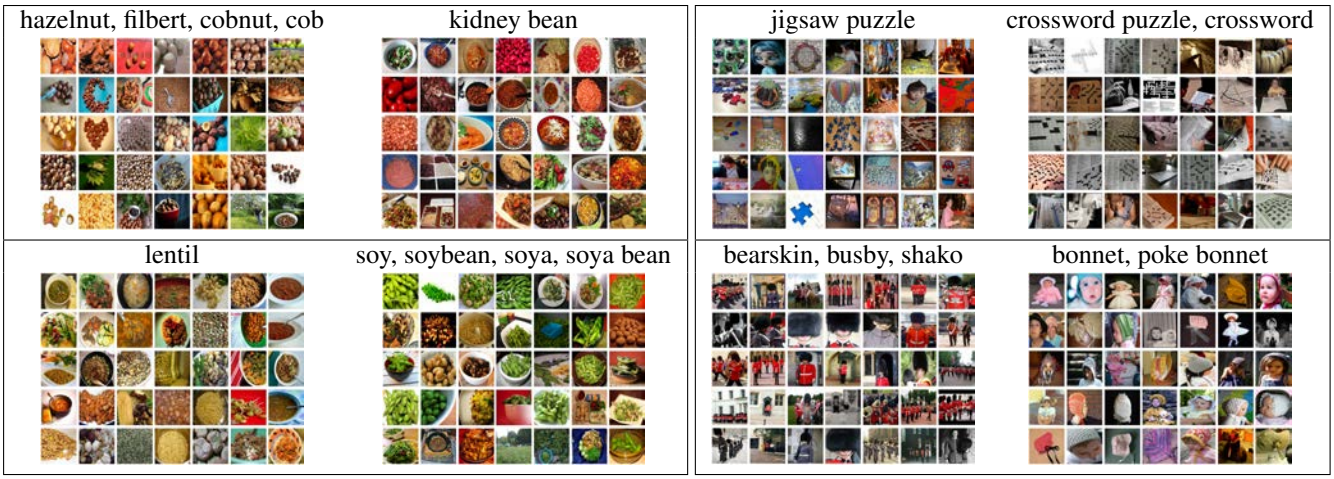
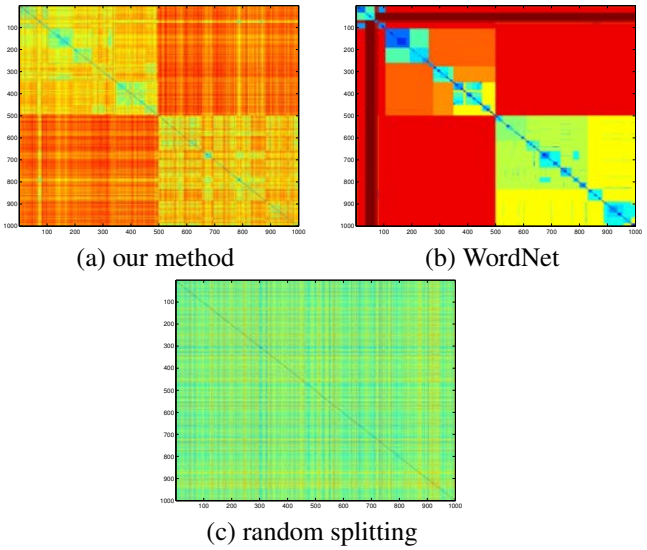


Fig. 4. Each cell shows a pair of categories whose distances have dramatic changes between WordNet hierarchy (Fig. 3(b)) and the learned label forest (Fig. 3(a)). The first two examples on the left (hazelnut↔kidney bean, lentil ↔soy) are categories that are far apart in WordNet hierarchy, but are close in the learned label forest. The two examples on the right (jigsaw puzzle↔crossword puzzle, bearskin↔bonnet) are categories that are close in the WordNet hierarchy, but are far in the label forest.



(a) our method (b) WordNet

(c) random splitting

Fig. 3. Visualization of tree distance (average of ten label trees) for every pair of category labels. We show the visualizations of our method (a) and random splitting (c) using ten trees. For our method (a), we exclude the first tree which is created by recursive spectral clustering on the WordNet distance matrix. As comparison, we also visualize the pairwise tree distance using the WordNet hierarchy (b). Not surprisingly, the tree distance of random splitting does not reveal any structures. But for our method as well as the WordNet hierarchy, the tree distance has a clear block structure.

scale visual recognition challenge. http://www.image-net.org/challenges/LSVRC/2010/index.

[3] L. Breiman. Random forests. *Machine Learning*, 2001.

[4] L. Cai and T. Hofmann. Exploiting known taxonomies in learning overlapping concepts. In *IJCAI*, 2007.

[5] J. Deng, A. C. Berg, K. Li, and L. Fei-Fei. What does classifying more than 10,000 image categories tell us? In *ECCV*, 2010.

[6] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A large-scale hierarchical image database. In *CVPR*, 2009.

[7] R. Fergus, H. Bernal, Y. Weiss, and A. Torralba. Semantic label sharing for learning with many categories. In *ECCV*, 2010.

[8] T. Gao and D. Koller. Discriminative learning of relaxed hierarchy for large-scale visual recognition. In *ICCV*, 2011.

[9] P. Geurts, D. Ernst, and L. Wehenkel. Extremely randomized trees. *Machine Learning*, 2006.

[10] D. Hsu, S. M. Kakade, J. Langford, and T. Zhang. Multi-label prediction via compressed sensing. In *NIPS*, 2009.

[11] L.-J. Li, C. Wang, Y. Lim, D. M. Blei, and L. Fei-Fei. Building and using a semantivisual image hierarchy. In *CVPR*, 2010.

[12] Y. Lin, F. Lv, S. Zhu, M. Y, T. Cour, K. Yu, L. Cao, and T. Huang. Large-scale image classification: Fast feature extraction and SVM training. In *CVPR*, 2011.

[13] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 2004.

[14] M. Marszalek and C. Schmid. Constructing category hierarchies for visual recognition. In *ECCV*, 2010.

[15] E. Rosch and B. Lloyd. Principles of categorization. *Cognition and Categorization*, pages 27–48, 1978.