Hierarchical Multiclass Object Classification

Fereshte Khani
Massachusetts Institute of Technology
fereshte@mit.edu

Anurag Mukkara
Massachusetts Institute of Technology
anurag_m@mit.edu

Abstract

Humans can use similarity between objects in order to recognize rare objects. They also make many abstract concepts when they see some objects very often. Interestingly, a large part of brain is associated with common classes like faces rather than rare objects like Ostrich. In our work we want to propose a model that has four mentioned characteristics. 1. Use more resources for categories that have many examples and less resources for categories that have few examples, 2. Has the ability recognize objects that it has never seen before, but can be synthesized by combining previously seen objects, 3. Objects with few examples can borrow statistical strength from similar objects with many examples, 4. Models that have lot of sharing between model parameters of different classes should be favored.

1. Introduction

Object classification is one of the fundamental problems in computer vision. It becomes a lot harder when number of categories is large. When number of categories is very large, the complexity of the model should not be related to the number of categories. It should capture the similarities between categories in order to classify objects in an easier and more accurate way.

We all have a hierarchy based abstraction for recognizing objects around us. We make more hierarchical concepts for object that we encounter more often. For example, if you ask a person about furniture they can categorize it by chair, desk, sofa, table, etc. and then for chair it goes like office chair, armchair, rock chair, or even wheelchair. These sub categories are related to each other visually. In this work, we try to propose a model that makes more abstract concepts when there are lots of similar classes and less otherwise. This model also share many parameters between similar objects. It helps rare objects that we have less training example from them use parameters of common examples. In other words, common classes have lots of training example by these examples algorithm can set accurate parameters. On other hand, rare objects have very few examples and set the parameters in an accurate way is almost impossible . Sharing parameters between these classes help rare classes to borrow statistical strength from common classes.

We all make a model for every concept and when we encounter a new object we try to check to which concept model it matches more. All learning algorithms try to add additional information to the problem to avoid over fitting. They usually use Occam-razor to have models as simple as possible. Here we want to change this regularization in a way that if a model uses previously learned objects it can have more complex shape. By this structure, models that have lot of sharing between model parameters of different classes will be favored.

There are many objects around us. In the model that we have many categories, when we want to train a classifier to learn objects we should be able to model the similarities between different categories otherwise we should store a lot of redundant information. In one vs all methods there is no difference between bicycle vs motorcycle and dogs vs motorcycle. While we like that if we have a false positive it should be bicycle rather than dogs. Even humans sometimes are confused between different dogs or wrongly assume another animal to be a dog. So, we want a classifier to model this similarities.

The rest of paper is as follows, in section 2 we give an overview of related work. There is an overview about independent multiclass classification in section 3. In section 4 we explain our method which is a hierarchical multi-class model for object recognition. We discuss experimental results in section 4 and conclude in section 5.

2. Related work

There are lots of work being done in the field of transfer learning in object classification. There are three main questions that people should answer when they want to use transfer learning. 1) What to transfer?, 2) How to transfer?, When to transfer?. For a good survey in transfer learning see [5]. For the first question, there are methods that share parameters between classes like you can learn parameter of deep network [?] by Imagenet [1] and then test it in on another benchmark. One other method is sharing training examples. One important thing in how we should transfer is that between which classes we should transfer parameters/features. For example when we add a new class we should decide that with which classes this class should share features. If there are only a few examples from each class so deciding which classes should share information and which classes should not is a hard question.

For finding which classes should share information some methods use a global priori [2] (share information between all classes) which provides small benefits but still is better than independent classifiers. There are some models [4] that use an external hierarchy. For example, in [4], wordnet [3] is a semantic network for English language. Words are grouped together if they are synonyms and record semantic relation between them. With this dictionary one can understand both bicycle and care belong to vehicle.

Ruslan et al. [8] make a tree in which every class is a leaf. In their setting, height of tree is fixed and is equal to two. However, number of super-categories in the tree is not fixed and they use Chinese Restaurant Prior [?] to decide if they should add a new super-category to the tree or not. Classes which have a common parent share features. They add classes incrementally to the tree. At each step they decide where in the tree they should place the new class. Although this setting support feature 3, it does not incorporate the other features mentioned above. Ruslan et al. improve their model in [7] with deep network features.

Sivic et al. [9] group images using multi-layer hierarchy tree. Tree is based on common visual elements. They used generative Hierarchical Latent Dirichlet Allocation(hLDA)[?]. This model is usually used for unsupervised learning of topic hierarchies. In their work, they automatically (without supervision) discover a hierarchical structure for the visual world using unlabeled images. They are also some models that try to find some relation between categories and use them to classify images [6]. For the last question that when we should transfer, algorithms deal with questions like this: should we use transfer information just for testing or we can use them for training as well? If we want to use transfer information during learning should we use them at first or in the middle? Another main issue in this area is that when it is good to transfer information, and when it make it worse when we transfer information. See [?] for more details.

3. Hierarchical Classification Model

3.1. Learning Independent Classification Models

Consider a classification problem where we observe a dataset D of labeled training examples. Each example belongs belongs to one K classes(eg. 40 object categories). The dataset D has n_k labeled examples for for each class $k \in 1, 2, ..., K$. We $x_i^{(k)} \in R^D$ be the input feature vector of length D for the i^{th} training example of class k and $y_i^{(k)}$ be the corresponding class label. $y_i^k = 1$ indicates that the i^{th} training example is a positive example for class K and $y_i^k = -1$ indicates that the i^{th} training example is a negative example for class K.

For a binary classification problem we can use a simple logistic regression model. In particular, for each class k, the probability of a positive instance can be modeled as:

$$p(y_i^{(k)} = 1|\beta^{(k)}) = \frac{\exp(\beta^{(k)T} x_i^{(k)})}{1 + \exp(\beta^{(k)T} x_i^{(k)})} \tag{1}$$

where k ranges over the K classes and $\beta^k \in R^D$ is the unknown model parameter of length D for class k. In addition, we place a zero-mean spherical Gaussian prior over each model parameter β^k:

$$p(\beta) = \prod_{k=1}^{K} p(\beta^{(k)}) = \prod_{k=1}^{K} N(0, \frac{1}{\lambda}) \tag{2}$$

where $N(\mu, \Sigma)$ indicates a Gaussian probability distribution with mean μ and covariance Σ. Here, λ is the prior parameter. With this modeling framework, the log posterior distribution of the unknown parameters can be expresses as

$$\log p(\beta|D) \propto \sum_{k=1}^{K} \Big[\sum_{i=1}^{n_k} \log p(y_i^{(k)}|x_i^{(k)}, \beta^{(k)}) + \log p(\beta^{(k)}) \Big] \tag{3}$$

Maximizing the log posterior(or finding the MAP estimate) over the unknown model parameters β is equivalent to minimizing the probabilistic loss function with quadratic regularization terms:

$$E = \sum_{k=1}^{K} \Big[\sum_{i=1}^{n_k} Loss(y_i^{(k)}, x_i^{(k)}, \beta^{(k)}) + \frac{\lambda}{2} \|\beta^{(k)}\|^2 \Big] \tag{4}$$

where the probabilistic loss function is given by:

$$Loss(y, x, \beta) = -\sum_{j \in -1,1} I\{y = j\} \log p(y = j|x, \beta) \tag{5}$$

One important property of the loss function is that it is convex and hence can be efficiently optimized using standard convex optimization solvers.

The main drawback of this formulation is that all the K classes are treated as unrelated entities and hence do not share any model parameters. In fact, it is observed that the objective function decomposes into K sub-problems, each of which is a binary classification problem that can be trained independently using the training data for that particular class.

3.2. Learning a Hierarchical Classification Model

In our framework, a hierarchical tree structure is used to model sharing of parameters between visually related classes. Each node in the tree has a model parameter, a vector of length D associated with it. Let $\theta = \{\theta_1, \theta_2, ..., \theta_n\}$ denote the model parameters of all the nodes in the tree, where n is the number of nodes starting from the root node to the all the leaf nodes. The leaf nodes correspond to the K object categories or classes. The parameters for a particular class k is the sum of the parameters of all the nodes, say $i_1, i_2, ..., i_l$, along the path from the leaf node corresponding to that class up until the root node of the tree:

$$\beta^{(k)} = \theta_{i_1} + \theta_{i_2} + ... + \theta_{i_l} \tag{6}$$

We place zero-mean spherical Gaussian priors on the parameters of each node. However, the covariance matrix is dependent on the level of a particular node in the tree i.e all the nodes at a particular level have the same prior parameters.

$$p(\theta_i) = N(0, \frac{1}{\lambda_l} I) \tag{7}$$

where node i is at level l in the tree. The root node has level 0 and the level increases as we move towards the leaf nodes.

With this modification in the way the model parameters for classes are related, the hierarchical learning problem can be formulated as minimizing the log posterior or equivalently the loss function with respect to β:

$$\begin{aligned} E = Loss(Y, X, \beta) + \frac{\lambda_0}{2}\|\theta_0\|^2 + \sum_{i=1}^{k_1} \frac{\lambda_1}{2}\|\theta_{i1}\|^2 \\ + \sum_{i=1}^{k_2} \frac{\lambda_2}{2}\|\theta_{i2}\|^2 + ... + \sum_{i=1}^{k_l} \frac{\lambda_l}{2}\|\theta_{il}\|^2 \end{aligned} \tag{8}$$

where $Loss(Y, X, \beta)$ is the loss function as explained in the previous section but with the additional constraint that β satisfies equation 6 according to the tree structure. Here there are k_j nodes in level j and $\theta_{1j}, \theta_{2j}, ..., \theta_{k_j j}$ are the parameters of nodes in level j.

3.3. Learning the tree structure

If we are given the tree structure that indicates the relation between different classes, then the learning algorithm only needs to optimize for the loss in equation 8. However, as discussed in [8], learning the tree structure dynamically based on the input classes and training data could lead to better results in object classification. We adopt the algorithm used in [8] to multilevel trees and fine-grained partitioning. In [8], the author use a fixed number of levels i.e 3 in the tree and use a non-parametric Chinese Restaurant Process Prior(CRP) to model the position of a class in the tree. Although this simple CRP prior allows one to have unknown and potentially unbounded number of groups of object classes, this is can only be used for two-level trees. For trees which have a fixed number of levels more than 3, one could use a nested CRP prior [**?**]. Since this approach has the drawback of limiting the tree to a fixed number of levels, we use a simple extension(modified-CRP) of the CRP prior used in [8] to allow the flexibility of having different depths at different parts of the tree.

A new class could go into one of the existing super-categories or create a new super-category. A new super-category can be attached to any node which doesn't have any children that are leaf nodes. That is, we allow the parent of a new super-category to be any node that is not itself a super-category that has leaf nodes, which correspond to

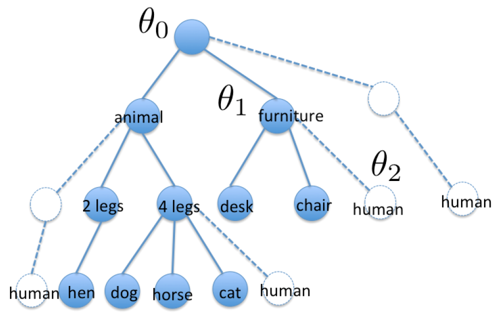

Figure 1. Example of caption. It is set in Roman so that mathematics (always set in Roman: $B \sin A = A \sin B$) may be included without an ugly clash.

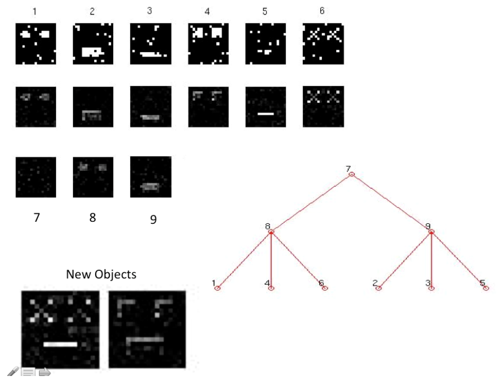

Figure 2. Example of caption. It is set in Roman so that mathematics (always set in Roman: $B \sin A = A \sin B$) may be included without an ugly clash.

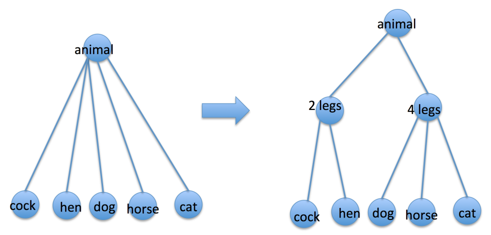

Figure 3. Example of caption. It is set in Roman so that mathematics (always set in Roman: $B \sin A = A \sin B$) may be included without an ugly clash.

classes, as children. By following the procedure in [8], for each possible position of the new class we calculate a likelihood term and modified-CRP prior term. The position of the new class will be the one that maximizes the sum of likelihood term and prior term.

Let i denote the node number of the leaf node corresponding to the new class. To calculate the likelihood at a possible position in the tree, we find the model parameter θ_i of the leaf node, that maximizes the log likelihood of the training data at that position. This likelihood will depend on the position in the tree, when we choose a position it means that we include the parameters of all the nodes that are ancestors of the leaf node i at that position. If the position corresponds to a category that is visually dissimilar to the new class, then the best we can with θ_i with still result in a lower likelihood term. If the the category is visually similar to the new class, we can achieve a far better fit and the likelihood term goes up. We have observed that the maximum of the likelihood over the model parameter θ_i is greatly affected by the position we choose for the leaf node. In addition to the likelihood term, the modified-CRP prior term favors super-categories that have a large number of leaf nodes or class under them. The modified-CRP prior is given by:

$$p(z=j) = \begin{cases} \frac{m_j}{n+\gamma * m} & : j \text{ is old} \\ \frac{\gamma}{n+\gamma * m} & : j \text{ is new} \end{cases}$$

where n is the total number of leaf nodes, m_j is the number of leaf nodes under super-category j, γ is the concentration parameter that controls the probability of creating a new super-category, m is the total number of positions of new super-categories in the tree. m is equal to number of nodes in the tree that have only super-categories as children. In other words, m is the cardinality of the set that includes the root node as well all nodes that are not leaf nodes or parents of leaf nodes. It can be easily verified that this defines a valid probability distribution over the choice z of possible positions in the tree where the new class can be placed.

We use same model-fitting algorithm described in [8] with modifications to CRP-prior term and tree structure as described above. Basically, the algorithm alternates between deciding the position in the tree of a new class and optimizing the model parameters of all the nodes in the tree for a given tree structure. This alternating procedure will reach a local minima of the loss function described in equation 8. Since the loss function is convex, this is also the global minima of the loss function. For optimizing the model parameters for a fixed tree structure , we use iterative coordinate-descend procedure. Since the parameters of all the nodes at any given level of the tree are independent of each other, we parallelize the loss minimization routine for these nodes. This leads to significant reduction in time taken to learn the model.

We also use a simple heuristic to maintain a fine-partitioning of the classes among into among super-categories. We need to avoid many leaf nodes accumulating under the same super-category as this would lead to a very coarse-grain partitioning of the classes. So, whenever the number of leaf nodes under a super-category goes above a certain threshold $maxChild$, we break down the subtree under that node to create a more fine-grained partitioning. To do this we run the entire optimization algorithm described above on the classes that were originally under this super-category. Basically, we consider the node corresponding to the super-category as a root node and keep adding new leaf nodes to form a tree under this root node. In most cases, the $maxChild$ number of classes that previously shared the same parent are dividing several finer super-categories and have different parents. In the rare case when the classes are very similar to each other, this process will add redundant to the tree. We identify this pathology and remove the redundant node while appropriately modifying the modle parameters of the relevant nodes. The parameter $maxChild$ can be configured to control the degree of fine-grained partitioning of classes. A lower value of $maxChild$ leads to very fine-grained partitions whereas a higher value of $maxChild$ results in coarser partitions. We found that a value of four results in best results.

4. Experiments

4.1. Implementation

We implement the whole algorithm in MATLAB the code is available in [?]. We use Hog features for images. for optimizing functions we use *fminunc* in MATLAB. One can use HOGgles [?] in order to see what is the shape of abstract concepts.

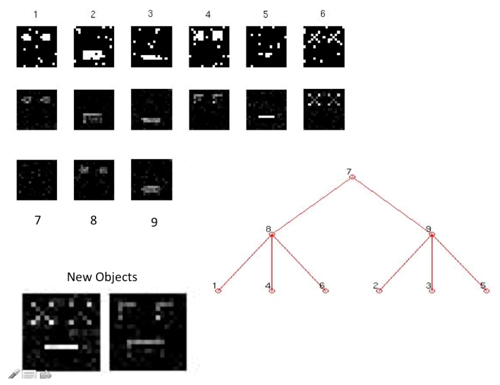

Figure 4. (a) First row shows eight classes of smiley faces. class 1,4,6 just have eys and class 2,3,5 just have eyes. The second row show the visualization for parameter θ for each node. The third line shows the visualization of θ for node 7,8,8 respectively. (b) shows the tree corossponding to these examples. As you can see in the visualization of parameter of node 7,8 they capture that there are eyes or mouthes in their children. (c) shows two visualizatoin of smiley faces by adding parameter by adding parameter θ of different nodes. If a new example which is look like these two come for testing we can identify it by these two

4.2. Data Sets

4.2.1 Smiley Faces

We make a syntactic data set in order to check our algorithm abilities. (merging two subcategories, make meaningful abstract concepts, make more nodes when there are more classes). There are 320 classes in this data set. each class has different nose, mouth, face and eyes. There are 1000 positive and negative examples in each classes there are noises added to each class.

4.2.2 Fine Grained Data Set

We use data set of fine grained challenge [**?**].There are five different domains (Cars, Birds, Aircrafts, Shoes and dogs) in this data set. In each domain there are about 100 classes.

4.3. Results

4.3.1 Smiley Faces

As mentioned before the main reason for making this dataset was to visualize our algorithm's abilities.

merging merging is ...

Meaningful super categories merging is ... hello

favoring shared parameters merging is ... hello

using simillar classes examples hello

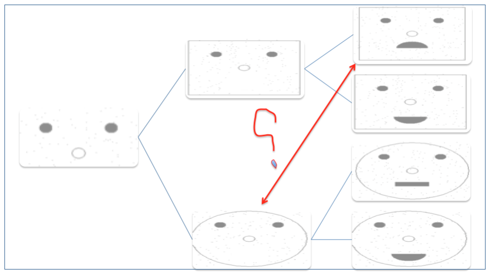

Figure 5. Example of caption. It is set in Roman so that mathematics (always set in Roman: $B \sin A = A \sin B$) may be included without an ugly clash.

4.3.2 Fine Grained Data Set

Due to limited time we just could learn for two hours. We use 10k pixels images for 8 classes and two domains. We reach the accuracy of 70

For checking that if one class belong to one of the leaves we compute

$$predictedClass = \underset{i \in leavesofTree}{\arg\max} \frac{\frac{1}{(1+exp(-\beta_i' . \phi(x)))}}{\|\beta_i\|^2}$$

and find the maximum as a desired class.

References

[1] Hierarchical classification source code, December 2014.

[2] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[3] T. Deselaers, B. Alexe, and V. Ferrari. Localizing objects while learning their appearance. In *Computer Vision–ECCV 2010*, pages 452–466. Springer, 2010.

[4] C. Fellbaum. *WordNet*. Wiley Online Library, 1998.

[5] M. Marszalek and C. Schmid. Semantic hierarchies for visual object recognition. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–7. IEEE, 2007.

[6] S. J. Pan and Q. Yang. A survey on transfer learning. *Knowledge and Data Engineering, IEEE Transactions on*, 22(10):1345–1359, 2010.

[7] A. Quattoni, M. Collins, and T. Darrell. Transfer learning for image classification with sparse prototype representations. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.

[8] M. T. Rosenstein, Z. Marx, L. P. Kaelbling, and T. G. Dietterich. To transfer or not to transfer. In *NIPS 2005 Workshop on Transfer Learning*, volume 898, 2005.

[9] R. Salakhutdinov, J. B. Tenenbaum, and A. Torralba. Learning with hierarchical-deep models. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(8):1958–1971, 2013.

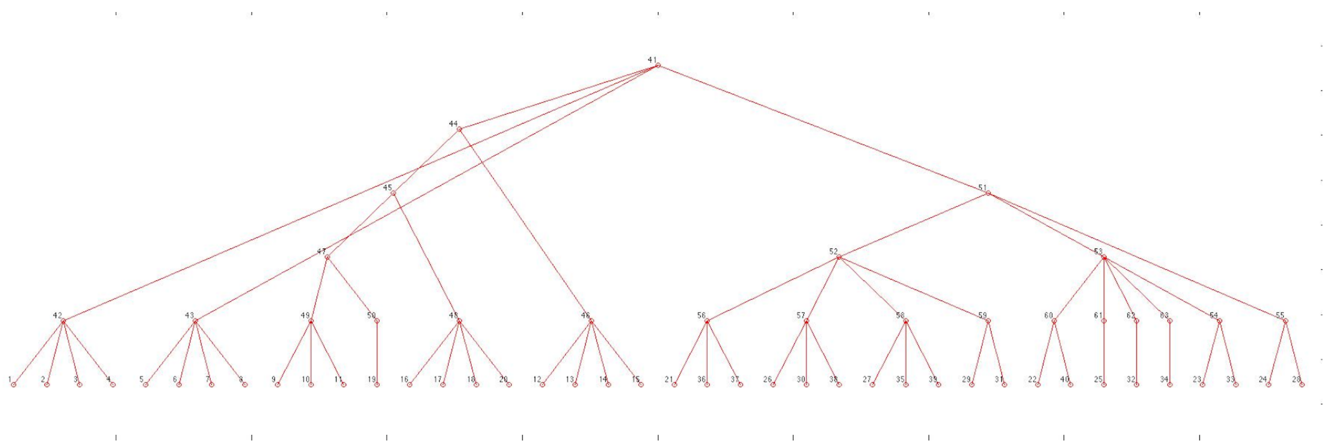

Figure 6. Example of a short caption, which should be centered.

[10] R. Salakhutdinov, A. Torralba, and J. Tenenbaum. Learning to share visual appearance for multiclass object detection. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1481–1488. IEEE, 2011.

[11] J. Sivic, B. C. Russell, A. Zisserman, W. T. Freeman, and A. A. Efros. Unsupervised discovery of visual object class hierarchies. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.

[12] C. Vondrick, A. Khosla, T. Malisiewicz, and A. Torralba. Hoggles: Visualizing object detection features. In *Computer Vision (ICCV), 2013 IEEE International Conference on*, pages 1–8. IEEE, 2013.