RegNet: Regularizing Deep Networks

Setareh Cohan
Department of Computer Science
University of British Columbia
setarehc@cs.ubc.ca

Saeid Naderiparizi
Department of Computer Science
University of British Columbia
saeidnp@cs.ubc.ca

Abstract

One of the most important issues in deep learning is the need for huge corpus of training samples to achieve desired results. Otherwise, any powerful model will overfit the training data. One way to overcome this problem is adding regularizers to the loss function in order to hold back the model from overfitting. Regularization is one of the crucial ingredients of model training. It plays an important role in preventing overfitting and reducing generalization error. Most of the efforts in regularizing deep networks has been focused on perturbing data and changing the structure of the network (e.g. Dropout). In this work, we focus on regularizing the network by adding a prior to the loss function based on similarities among labels. Particularly, we consider an image classification task where very few training samples are available for some classes and, study and expand a work[15] which adds a regularizer to push weight vectors corresponding to visually similar classes, to be similar.

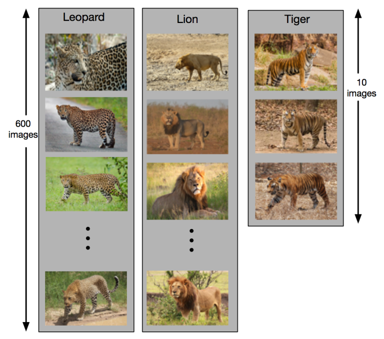


Figure 1. we have 100 classes of animals each with 600 image samples except for the "tiger" class which only contains 10 samples. If we train our network on the whole dataset, it will misclassify the test tiger samples. However, by transferring high level features from "leopard" and "lion" classes, the network has the general features of tigers(e.g., body shape and background format) and only needs to learn features specific to tigers (e.g., color and pattern of skin).

1. Introduction

Training a classifier that well generalizes with only a few training samples is a hard problem. As an example, having a large dataset with hundreds of classes where a few of them have less than 10 training samples, it will be difficult to correctly classify these rare samples. In this work, we try to tackle this problem by "transferring" information between relevant classes. The idea is that if a network borrows general features of a rare class from relevant classes, only the distinctive features specific to the rare class need to be learned. At the very least, the network should confuse the rare class with related classes and not with completely unrelated classes [15]. We show an example in Figure 1.

Our goal is to tackle the problem of image classification with few training samples for some classes by utilizing a task-specific regularizer. Looking back at common regularizing methods for deep networks, using class-dependent information seems a promising and not well-explored direction of possible development of a new regularization method [7]. To this end, we first need to have visually similar classes and then, we should regularize the network by encouraging weight vectors corresponding to these classes, to be similar.

The task of detecting visually similar classes is a hard problem itself. We either need some prior knowledge about classes or we need a good model to represent each class and their relations. The first approach we try is to simply use prior knowledge about the classes from other domains (more specifically, semantic similarity of classes). However, this is not promising to significant improvements, as visual similarity is not necessarily reflected in semantic space. Therefore, we try to make use of the model itself as our second approach. Initially, we use semantic similarity as our similarity measure. As the model learns how to classify images, we refine the similarities. This is discussed in more detail in section 3. We use hierarchies as a means to obtain similarity of classes. This is done by assuming that two classes in the same superclass (two leaves with the same parent in the hierarchy) are similar.

We propose a procedure for learning the class structure and parameters of the neural network with the goal of improving classification accuracy for rare classes. Similar to [15], we add a generative prior over the weights in the last layer of our network. To this end, we encourage the weight vectors of related classes to be similar. This shared prior will capture the common features across all children of a superclass. Consequently, rare classes have access to this information by being a member of a superclass. More generally, information will be transferred among related classes.

2. Related Work

The idea of using class similarities to regularize a classification deep neural network is proposed in [15]. They use a fixed hierarchy of classes and also propose a method for updating the hierarchy and show a slight improvement in the classification accuracy when number of training samples per class is small. We also use fixed and dynamic hierarchies in this work. For fixed hierarchy setting, we utilize the hierarchy provided in our dataset as [15]. However, we will use a different method for the dynamic hierarchy setting with the hope of improving the results of [15].

Hierarchies are a natural way to organize concepts and data [4]. Therefore, there has been an extensive amount of research on building hierarchies for a selection of data. For images, a good image hierarchy can serve as knowledge reference for end tasks such as classification.

Three types of hierarchies have been explored in computer vision: semantic hierarchies, visual hierarchies and hybrid hierarchies. Pure language-based hierarchies such as WordNet [11, 1], have been used in vision and multimedia communities for tasks such as image retrieval [5, 2] and object recognition [9, 16]. These hierarchies ignore important visual information that connect images together. For instance, *snowy mountain* and *skiing* are far in the WordNet hierarchy, while they are visually close [8]. On the other end, some purely visual feature-based hierarchies have also been represented [10, 14]. An advantage of these hierarchies is capturing visual relations between objects and concepts. But, they are difficult to interpret which makes the usage of them questionable. Motivated by having a meaningful visual hierarchy, methods have been proposed to construct hierarchies using both semantic and visual similarities [8, 18, 17].

We will try a solely semantic hierarchy in our fixed setting and then, learn the dynamic hierarchy by initializing it with a semantic hierarchy and updating it based on the weight vectors of last layer of the network. Thus, our dynamic hierarchy falls in the class of hybrid hierarchies.

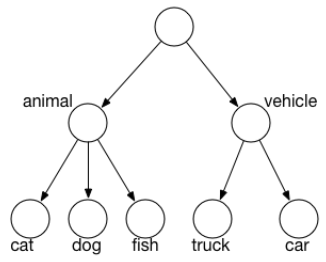


Figure 2. A 2-level hierarchy.

3. Detailed Approach

We generally follow the proposed method in [15] which uses a 2-level hierarchy of classes (figure 2) to impose a prior over the network's last layer parameters. Two cases are considered: first, a fixed hierarchy of classes will be utilized and second, the hierarchy will be learned as the network is being trained. We briefly explain these two cases in sections 3.1 and 4.4 and finally, we propose our own ideas about the potential alternatives to learning the hierarchy in section 3.3.

3.1. Fixed Hierarchy

In this setting and with the same notation as [15], it is assumed that the classes are organized in a fixed hierarchy which is available from some external resource. Consider the two level hierarchy as shown in figure 3b. There are K leaf nodes corresponding to K classes which are connected to S super-classes. Leaf node k is associated with a weight vector $\beta_k \in \mathbb{R}^D$ and each super class s is associated with a weight vector $\theta_s \in \mathbb{R}^D$. The following distributions are assumed for θ and β which show the relationship between classes and super classes:

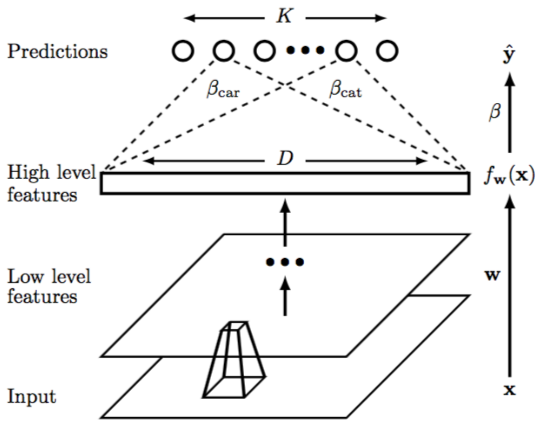
$$\theta_s \sim \mathcal{N}(0, \frac{1}{\lambda 1} I_D), \quad \beta_k \sim \mathcal{N}(\theta_{\text{parent}(k)}, \frac{1}{\lambda 2} I_D)$$

The general format of loss function for training the network (as shown in figure 3a) is:

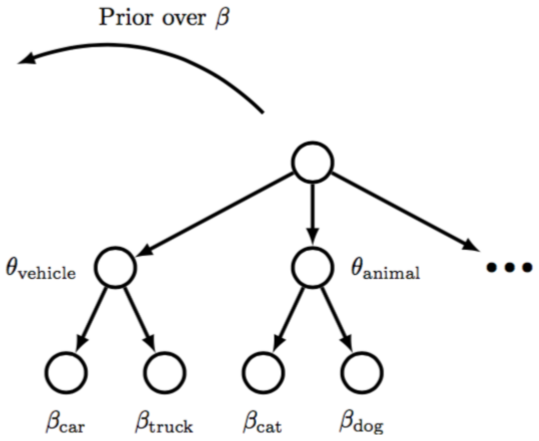
$$\begin{aligned}\mathcal{L}(\mathbf{w}, \beta, \theta) = &- \log P(\mathcal{Y}|\mathcal{X}, \mathbf{w}, \beta) \\ &+ \frac{\lambda_2}{2}\|\mathbf{w}\|^2 + \frac{\lambda_2}{2}\sum_{k=1}^{K}\|\beta_k - \theta_{\text{parent}(k)}\|^2 \\ &+ \frac{\lambda_1}{2}\|\theta\|^2\end{aligned} \tag{1}$$

which is derived from the following MAP estimate:

$$\begin{aligned}P(\mathcal{Y}|\mathcal{X}) = \int_{\mathbf{w},\beta,\theta} \Big[&P(\mathcal{Y}|\mathcal{X}, \mathbf{w}, \beta) \\ &P(\mathbf{w})P(\beta|\theta)P(\theta) d\mathbf{w} d\beta d\theta \Big]\end{aligned} \tag{2}$$



(a)



(b)

Figure 3. Model: A deep neural network with hierarchy-based priors over the last layer parameters [15].

The loss function in 1 is optimized by iteratively following the next two steps:

1. Minimize over $\mathbf{w}$ and β, keeping θ fixed which can be done by any standard optimizer such as stochastic gradient descent (SGD).
2. Minimize over θ, keeping β fixed which can be done in closed form as below. ($|C_s|$ is the number of nodes whose parent is s.)

$$\theta^* = \frac{1}{|C_s| + \lambda_1/\lambda_2} \sum_{k \in C_s} \beta_k$$

In our experiments, the second step which is almost instantaneous is only performed after every T gradient descent steps where T is set to 50. We perform the above steps L times and then update the hierarchy structure. We set L to 10000 in our experiments.[1]

3.2. Dynamic Hierarchy

In this setting, a fixed hierarchy is not presented to the model and the goal will be to learn the hierarchy while training the network. In [15], $\mathbf{z}$ is a K-length vector such that $z_k = s$ indicates class k is a child of super class s. Then, a non-parametric Chinese restaurant prior (CRP) is used over $\mathbf{z}$ which enables the model to have any number of super classes.
Using the CRP prior over $\mathbf{z}$, the MAP estimate is:

$$P(\mathcal{Y}|\mathcal{X}) = \sum_{\mathbf{z}} \Big(\int_{\mathbf{w},\beta,\theta} \Big[P(\mathcal{Y}|\mathcal{X},\mathbf{w},\beta)P(\mathbf{w}) \\ P(\beta|\theta,\mathbf{z})P(\theta)d\mathbf{w}d\beta d\theta \Big]\Big) P(\mathbf{z}) \quad (3)$$

[1]We set T and L according to suggested values in [15].

This, leads to maximizing the following expression:

$$\max_{\mathbf{w},\beta,\theta,\mathbf{z}} \log P(\mathcal{Y}|\mathcal{X},\mathbf{w},\beta) + \log P(\mathbf{w}) \\ + \log P(\beta|\theta,\mathbf{z}) + \log P(\theta) + \log P(\mathbf{z}) \quad (4)$$

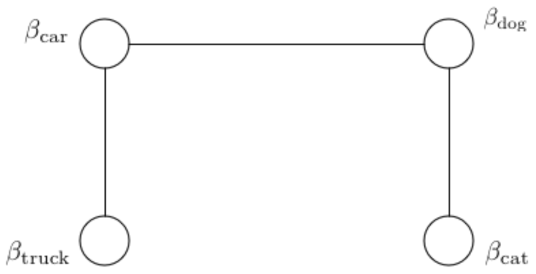
The hierarchy should be first initialized carefully either by hand or by extracting it from some external source such as WorldNet [11].

3.3. Dynamic Hierarchy Extension

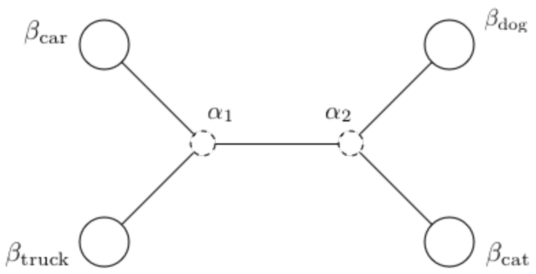
In this work, we propose other potential efficient and effective methods to learn the hierarchy and we hope to improve the results of [15]. We have four ideas for learning the hierarchy which will be explained later in this section.

3.3.1 Steiner Tree Problem Approach

One idea is to use a pre-trained network with normal L2-norm regularization over all weights (without taking hierarchy of classes into account) as the initial network and generate a hierarchy based on the weight vectors of classes in this pre-trained model and update the hierarchy after every few steps of updating the network's parameters. Let β_k^t be the the weight vector corresponding to class k after t updates which is the moment we want to update (or generate, in case of $t = 0$) the structure of the hierarchy. We consider β_k^t for all classes in a d-dimensional hyperspace (where d is the dimension of each β_k) and solve the "minimum length connection" problem which we define as the problem of drawing some straight lines so that all these points are connected together and the sum of length of lines is minimum. It is possible to add a set of auxiliary points, $\{\alpha_1, \alpha_2, \ldots, \alpha_m\}$, if needed, to reduce sum of line lengths. As an example, figure 4a shows 4 points in 2D space which are connected



(a) minimum length without auxiliary points.



(b) minimum length with adding auxiliary points.

Figure 4. Model: minimum length connection problem.

with the minimum length of lines but no auxiliary points is added. On the other hand, figure 4b shows the same 4 points which are connected by less amount of sum of lines lengths using the help of auxiliary points α_1 and α_2. Intuitively, to solve minimum length connection problem, points that are close to each other, should be connected to a single auxiliary point and we use that shared point as their parent in the hierarchy (for example, in case of figure 4, α_1 would be θ_{vehicle} and α_2 would be θ_{animal}).

This problem is equivalent to the Euclidean Steiner tree problem which is to find the tree with minimal Euclidean length spanning a set of fixed points in the plane, allowing the addition of auxiliary points to the set (Steiner points). Unfortunately, the Steiner tree problem is NP-hard. There are many heuristics that allow computing a locally optimal solution [3, 12, 13] to this problem. However, all these heuristics are either too vague in description or are inefficient for a hyper dimensional network (most methods are applicable for dimension up to a 100 while with neural networks, we deal with dimensions around 1000 or more).

3.3.2 Greedy Approach

A simpler approach would be to initialize the hierarchy using some external source again. For instance, we can greedily add each class to its nearest super-class over all the super-classes. Although this method does not guarantee improvements over [15], it is easier to implement and also, it is computationally cheap.

3.3.3 K-means Approach

Moreover, we could perform k-means clustering over the classes. One drawback of this method would be determining k. On the other hand, an advantage of this method is that it does not depend on an initial hierarchy like the previous methods, making it suitable for cases where there is no hierarchy of classes available.

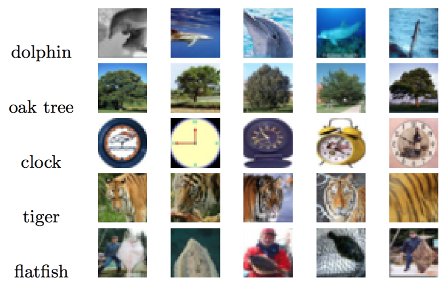


Figure 5. Examples from CIFAR-100. Five randomly chosen examples from 5 of the 100 classes are shown.

3.3.4 Expectation Maximization Approach

Besides the aforementioned ideas, another extension we think of is using expectation maximization (EM) algorithm. This approach of solving this problem has some hidden variables (i.e. θ_c) to learn, which can be extracted using EM.

After taking the drawbacks and time constraints of our work, we decided to implement the greedy approach and the k-means approach for learning the class hierarchy. We set the value of k to 25 empirically.

4. Experiments

4.1. Dataset

We will evaluate our method on CIFAR-100 dataset [6]. This dataset consists of 32×32 color images belonging to 100 classes. These classes are grouped into 20 super-classes each containing 5 classes. For example, super class **insects** contains *bee, beetle, butterfly, caterpillar, cockroach* and super class **trees** contains *maple, oak, palm, pine, willow*. There are 600 examples for each class of which 500 are in the training set and the remaining are in the test set. Some examples of this dataset are available in Fig 5.

4.2. Model Architecture

We use a convolutional neural network (CNN) as our learning model. It consists of three convolutional layers followed by 2 fully connected layers. Each convolutional layer is followed by a max-pooling layer. The convolutional layers have 96, 128 and 256 filters respectively. Each convolutional layer has a 5×5 receptive field applied with a stride of 1 pixel. Each max pooling layer pools 3×3 regions at strides of 2 pixels. The two fully connected hidden layers having 2048 units each. All units use the rectified linear activation function. Dropout was applied to all the layers of the network with the rate of p = (0.1, 0.25, 0.25, 0.5, 0.5, 0.5) for the different layers of the network, from input to convolutional layers to fully connected layers.

4.3. Experimental Setup Details

We use a learning rate of 0.01 and divide it by 10 every 50 epochs. We use a total of 150 epochs to train our network. The optimizer we use is SGD with Nesterov momentum.
We tuned the hyper parameters λ_1 and λ_2 for the fixed hierarchy setting, with grid search. After 20 hours of search, $\lambda_1 = 10^{-12}$ and $\lambda_2 = 10^{-8}$ were reported to be the best values.

4.4. Experiment on Few Examples for All Classes

In the first set of experiments, we consider the case where all classes have few training samples. We want to assess whether using the prior based on class hierarchy, improves the classification accuracy as we expected and was showed in [15]. We create 7 subsets of data by randomly choosing 5, 10, 20, 50, 70 and 100 percent of samples for each class. Then, we train four models on each subset - the baseline, our model with the CIFAR-100 hierarchy which we call `FixedTree`, our model where we learn the hierarchy using the greedy approach explained in 3.3.2 which we call `DynamicTree-Greedy`, and our model where we learn the hierarchy using the k-means approach explained in 3.3.3 which we call `DynamicTree-Kmeans`. The baseline model is a standard CNN with the same architecture explained in 4.2.

The performance of these four models is compared in Fig 6 and Fig 7. As we can see in both of the plots, accuracy of different approaches are almost identical. For the top-1 accuracy, we do not have any improvements after using the hierarchies. This was not completely unexpected since all classes are lacking samples and relative classes might not have sufficient number of sample to convey useful information to one another.
In our top-5 results, we have a small improvement by using the hierarchies over no hierarchy which complies with results of [15]. Learning the hierarchy with greedy approach improves the results of fixed hierarchy, but, learning it with

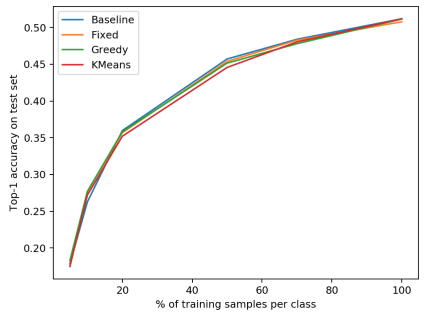


Figure 6. Top-1 classification accuracy on CIFAR-100 with few samples for all classes.

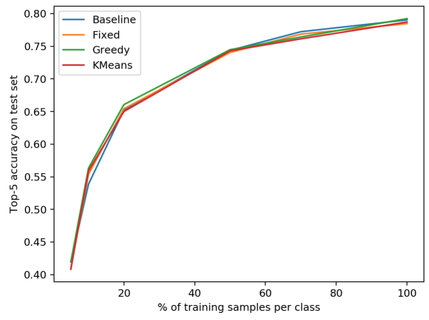


Figure 7. Top-5 classification accuracy on CIFAR-100 with few samples for all classes.

kmeans approach does not give much improvements over the fixed hierarchy setting. This could again be the result of all classes lacking samples and thus, being unable to borrow useful information from one another.

We believe that our method is specially effective if only a few number of rare classes exist in our dataset. These classes will be able to borrow information from their related classes and the classification accuracy over these rare classes will improve. We will test this hypothesis in 4.5.

4.5. Experiments on Few Examples for One Class

In this set of experiments, we consider the case where only one class has few training samples. Here, our goal is to see whether our model enables the rare class to borrow information from its related classes and thus, increase the classification accuracy of the rare class. We create datasets by randomly choosing 5, 10, 20, 50, 70 and 100 percent of samples for the "dolphin" class and all 500 samples for the other 99 classes[2]. Again, we train our four models on each subset - the baseline, the `FixedTree`, the `DynamicTree-Greedy`, and the `DynamicTree-Kmeans`.

The performance of these four models is compared in

[2]We chose the "dolphin" class as was done in [15]

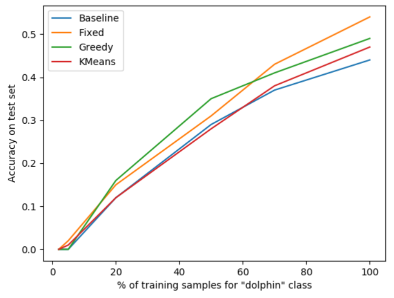


Figure 8. Classification results on CIFAR-100 with few samples for one class.

Figure 8. As expected, we see improvements using hierarchies over the baseline which is consistent with the results of [15]. Up to about 65%, we get the most improvement from `DynamicTree-Greedy` approach and then the fixed hierarchy and lastly, `DynamicTree-Kmeans` with and accuracy almost identical to the baseline. We expected that learning hierarchies would improve the results over the fixed hierarchy setting which can be seen from the results of `DynamicTree-Greedy`. However, using the hierarchy learned by `DynamicTree-Kmeans`, the accuracy is almost identical to the baseline and lower than the fixed hierarchy which is completely unexpected. This might be due to the fact that we do not use any data augmentations in our code.

5. Conclusion

We test a model that augments standard CNNs with a generative prior derived from a fixed hierarchy of classes over the classification parameters which was suggested in [15]. We also test a setting where the prior is derived from a dynamic hierarchy learned as the CNN is being trained. However, in this setting, we proposed four alternative methods to the one suggested in [15] and tested two of them.

Since [15] do not provide their implementation, we first tried to recreate their work as much as we could for the baseline and `FixedTree` setting. However, due to lack of specifications and details on their implementation, our results did not achieve the same exact performance as theirs did. Fortunately, we were still able to compare the results of the baseline, `FixedTree`, `DynamicTree-Greedy` and `DynamicTree-Kmeans` with one another in order to assess if and how each of them affects the classification accuracy.

Experiments show that we achieve some increase in the classification accuracy when hierarchy of classes is used. This is specially visible when we have a small number of rare classes. As expected, this suggests that classes do borrow information from one another. The future directions for this work would be to reach the same accuracy as [15]. Then, we could improve the results so that the learned hierarchy by `DynamicTree-Kmeans` would result in some improvements. Next, it would be interesting to see whether the Steiner tree and expectation maximization approaches can be implemented and tested.

References

[1] What is wordnet? https://wordnet.princeton.edu/. Accessed: 2018-03-09.

[2] R. Datta, W. Ge, J. Li, and J. Z. Wang. Toward bridging the annotation-retrieval gap in image search. *IEEE MultiMedia*, 14(3), 2007.

[3] D. R. Dreyer and M. L. Overton. Two heuristics for the euclidean steiner tree problem. *Journal of Global Optimization*, 13(1):95–106, 1998.

[4] T. L. Griffiths, M. I. Jordan, J. B. Tenenbaum, and D. M. Blei. Hierarchical topic models and the nested chinese restaurant process. In *Advances in neural information processing systems*, pages 17–24, 2004.

[5] Y. Jin, L. Khan, L. Wang, and M. Awad. Image annotations by combining multiple evidence & wordnet. In *Proceedings of the 13th annual ACM international conference on Multimedia*, pages 706–715. ACM, 2005.

[6] A. Krizhevsky. Learning multiple layers of features from tiny images. 2009.

[7] J. Kukačka, V. Golkov, and D. Cremers. Regularization for deep learning: A taxonomy. *arXiv preprint arXiv:1710.10686*, 2017.

[8] L.-J. Li, C. Wang, Y. Lim, D. M. Blei, and L. Fei-Fei. Building and using a semantivisual image hierarchy. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 3336–3343. IEEE, 2010.

[9] 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.

[10] M. Marszałek and C. Schmid. Constructing category hierarchies for visual recognition. In *European Conference on Computer Vision*, pages 479–491. Springer, 2008.

[11] G. A. Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[12] A. Olsen, S. Lorenzen, R. Fonseca, and P. Winter. Steiner tree heuristics in euclidean d-space. *Proc. of the 11th DIMACS Implementation Challenge*, 2014.

[13] T. Polzin and S. Vahdati Daneshmand. The steiner tree challenge: an updated study. *Unpublished manuscript at http://dimacs11. cs. princeton. edu/downloads. html*, 2014.

[14] 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.

[15] N. Srivastava and R. R. Salakhutdinov. Discriminative transfer learning with tree-based priors. In *Advances in Neural Information Processing Systems*, pages 2094–2102, 2013.

[16] A. Torralba, R. Fergus, and W. T. Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE transactions on pattern analysis and machine intelligence*, 30(11):1958–1970, 2008.

[17] C. Zhang, J. Cheng, and Q. Tian. Image-level classification by hierarchical structure learning with visual and semantic similarities. *Information Sciences*, 422:271–281, 2018.

[18] C. Zhang, R. Li, Q. Huang, and Q. Tian. Hierarchical deep semantic representation for visual categorization. *Neurocomputing*, 257:88–96, 2017.

6. Appendices

We have included the fixed hierarchy of CIFAR-100, the learned hierarchy using our `DynamicTree-Greedy` method, and the learned hierarchy using our `DynamicTree-Kmeans` method in tables 1,2 and 3 respectively. Note that the learned hierarchy obtained by `DynamicTree-Kmeans` has 25 superclasses since k of k-means is set to 25.
We can see that the `DynamicTree-Greedy` approach has not changed the fixed hierarchy and the learned hierarchy by this approach has remained the same as before. However, `DynamicTree-Kmeans` has updated the hierarchy. Some of these updates are meaningful. For instance, “forest” class is now a part of superclass 4 which contains all the tree classes. Another example would be putting “snake” and “worm” into a single superclass. But, some of these updates are not that natural.

Superclass	Classes
aquatic mammals	dolphin, whale, seal, otter, beaver
fish	aquarium fish, flatfish, ray, shark, trout
flowers	orchid, poppy, rose, sunflower, tulip
food containers	bottle, bowl, can, cup, plate
fruit and vegetables	apple, mushroom, orange, pear, sweet pepper
household electrical devices	clock, keyboard, lamp, telephone, television
household furniture	bed, chair, couch, table, wardrobe
insects	bee, beetle, butterfly, caterpillar, cockroach
large carnivores	bear, leopard, lion, tiger, wolf
large man made outdoor things	bridge, castle, house, road, skyscraper
large natural outdoor scenes	cloud, forest, mountain, plain, sea
large omnivores and herbivores	camel, cattle, chimpanzee, elephant, kangaroo
medium sized mammals	fox, porcupine, possum, raccoon, skunk
non insect invertebrates	crab, lobster, snail, spider, worm
people	baby, boy, girl, man, woman
reptiles	crocodile, dinosaur, lizard, snake, turtle
small mammals	hamster, mouse, rabbit, shrew, squirrel
trees	maple, oak, palm, pine, willow
vehicles 1	bicycle, bus, motorcycle, pickup truck, train
vehicles 2	lawn mower, rocket, streetcar, tank, tractor

Table 1. Given hierarchy for the CIFAR-100 dataset.

Superclass	Classes
superclass 1	beaver, dolphin, otter, seal, whale
superclass 2	aquarium fish, flatfish, ray, shark, trout
superclass 3	orchid, poppy, rose, sunflower, tulip
superclass 4	bottle, bowl, can, cup, plate
superclass 5	apple, mushroom, orange, pear, sweet pepper
superclass 6	clock, keyboard, lamp, telephone, television
superclass 7	bed, chair, couch, table, wardrobe
superclass 8	bee, beetle, butterfly, caterpillar, cockroach
superclass 9	bear, leopard, lion, tiger, wolf
superclass 10	bridge, castle, house, road, skyscraper
superclass 11	cloud, forest, mountain, plain, sea
superclass 12	camel, cattle, chimpanazee, elephant, kangaroo
superclass 13	fox, porcupine, possum, raccoon, skunk
superclass 14	crab, lobster, snail, spider, worm
superclass 15	baby, boy, girl, man, womean
superclass 16	crocodile, dinosaur, lizard, snake, turtle
superclass 17	hamster, mouse, rabbit, shrew, squirrel
superclass 18	maple, oak, palm, pine, willow
superclass 19	bicycle, bus, motorcycle, pickup truck, train
superclass 20	lawn mower, streetcar, tank, tractor, rocket

Table 2. Learned hierarchy by `DynamicTree-Greedy` method.

Superclass	Classes
superclass 1	raccoon, skunk, wolf
superclass 2	bicycle, caterpillar, lizard, rocket, spider
superclass 3	hamster, mouse, porcupine, possum, shrew
superclass 4	forest, maple, oak, palm, pine, willow
superclass 5	bowl, can, clock, plate
superclass 6	aquarium fish, orchid, poppy, rose, sunflower, tulip
superclass 7	crocodile, ray, trout, turtle
superclass 8	bus, lawn mower, motorcycle, pickup truck , streetcar, tank, tractor, train
superclass 9	bridge, castle, house, road
superclass 10	bottle, cup. lamp, television, wardrobe
superclass 11	bear, beaver, chimpanzee, otter, seal
superclass 12	bee, beetle, butterfly, cockroach
superclass 13	snake, worm
superclass 14	bed, chair, couch, table
superclass 15	baby, boy, girl, man, woman
superclass 16	camel, cattle, elephant, kangaroo
superclass 17	fox, leopard, lion, tiger
superclass 18	dolphin, shark, whale
superclass 19	crab, lobster
superclass 20	flatfish, rabbit
superclass 21	apple, orange, pear, sweet pepper
superclass 22	cloud, mountain, plain, sea
superclass 23	keyboard, telephone
superclass 24	dinosaur, mushroom
superclass 25	snail, squirrel

Table 3. Learned hierarchy by `DynamicTree-Kmeans` method.