Shared Roots: Regularizing Deep Neural Networks through Multitask Learning

Piotr Teterwak
Dartmouth College

Shared Roots: Regularizing Neural Networks through Multitask Learning

Piotr Teterwak
Department of Computer Science
Dartmouth College
Hanover, NH
piotr.teterwak@dartmouth.edu

Lorenzo Torresani
Department of Computer Science
Dartmouth College
Hanover, NH
lorenzo@cs.dartmouth.edu

Abstract

In this paper, we propose to regularize deep neural nets with a new type of multi-task learning where the auxiliary task is formed by agglomerating classes into super-classes. As such, it is possible to jointly train the network on the class-based classification problem AND super-class based classification problem. We study this and show that , concurrently with a regularization scheme of randomly reinitializing weights in deeper layers, this leads to competitive results on the ImageNet and Caltech-256 datasets and state-of-the-art results on CIFAR-100.

1 Introduction

In the past several years, Convolutional Neural Networks have demonstrated exceptional performance on complex visual tasks. They have shown state of the art results on the CIFAR-10, CIFAR-100, and ImageNet datasets, among others [15, 20]. Unfortunately, the discriminative power of large convnets is also their weakness. They overfit to the training data easily, and are prone to getting stuck in local minima. This problem is less pronounced on large datasets such as Imagenet; and it has been circumvented on some datasets by pre-training using ImageNet and fine-tuning on the final task [20].

Nevertheless, sometimes either a large and related task is unavailable, or the several days it takes to pre-train on that task is too costly. This leaves the discriminative power provided by convnets impossible to harness. In this paper, we introduce novel ways of regularizing the convnet. We explore different ways of forming related tasks to the original task by agglomerating classes into super-classes, which then can be trained jointly with the original task in a form of multi-task learning [4]. We see significant improvement in generalization performance.Furthermore, we have found that part-way through training, a random re-initialization of later layers in the network also significantly boosts test-time performance.

This paper proposes a novel way of defining secondary tasks for multi-task learning, introduces the idea of re-initialization of deep layers in the networks, and studies the effect of these regularizations on learning.

1.1 Previous work

Because neural networks are so prone to over-fitting and susceptible to finding themselves in local minima, a wide range of techniques for regularization have been developed. A simple approach called weight decay is an l_2 penalty on the network weights [14]. Other simple approaches include early stopping and data augmentation, where random crops, flips, and rotations provide the network with a larger dataset [5, 13]. These are heuristics that are both beneficial and useful for almost all network architectures, and are complimentary to our regularization methods.

More recently, Hinton *et al.* introduced Dropout [10]. By deleting a fraction of the activations in each layer, and backpropagating through the remaining activations, the networks weights are prevented from collaborating in memorization of the training set. This work was generalized and extended with Wan *et al.*'s DropConnect [18], where weights instead of activations were deleted. Both Dropout and DropConnect are effective in nearly all settings, but they slow down learning fairly substantially. Once again, these methods are complimentary to the ones introduced in this paper.

Different forms of pooling have also been explored. Most notably, Zeiler and Fergus introduced Stochastic Pooling as an alternative to max or average pooling [19]. In this approach, the deterministic pooling procedure is replaced by a stochastic procedure where an activation is sampled according to a multinomial distribution. Experiments show that Stochastic Pooling achieves better results than standard pooling procedures.

Srivastava and Salakhutdinov's work with Tree-based priors [17] is closely related to our work, in the sense that auxiliary tasks are defined using a class hierarchy. Unlike our work, though, their regularization is achieved by assuming that related classes use similar deepest-level features, regularizing output weights so related classes in the soft-max layer have similar output weight values. This limits the ability of a network to find differences between similar classes, so the method is somewhat limiting, especially because similar classes are already the ones most difficult to discriminate.

Most related to our proposed method, however, is the general concept of multi-task learning in neural networks. Ahmed *et al.* generate pseudo-tasks for each input image, which the network is asked to perform jointly with its main classification task [1]. It is shown that this improves classification performance. Similarly, Collobert and Weston [6] share deep layers among several different NLP tasks, showing improvement on all of them. As will be discussed in the methods section, sometimes it is difficult to find a task that is both related but different, or data is unavailable.

2 Model Description

Our multi-task learning regularization is meant to be used with a neural network architecture. Several tasks share the same network weights until a bifurcation in the network at level l. At this point, the features extracted by the network through level l are used as input for several networks with different target tasks. One of these tasks is the primary task, in our case the original classification problem. The others are related, but different, task. In our case, these are the classification super-classes, or unions of classes in the original classification problem. How many auxiliary tasks are used is up to the network architect, though we use only one additional tasks to conserve memory. This model is trained through standard backpropagation, though the error propagating back is scaled by half right before bifurcations join to have a consistent learning rate. Our neural network models are implemented using Caffe, a software developed at Berkeley [11]

2.1 Multi-task learning, bifurcations

The idea of multi-task, or inductive transfer learning, has been around for quite some time. The idea is simple: if there are two related tasks then learned features should be related as well. This is especially true of deep models like deep Convolutional Neural Networks, because there is a hierarchy of features [20]. Even if the highest-level features are task-specific, it is likely that lower-level features can be shared. This gives the benefit of being able to augment the data-set for those lower-level features by training those jointly on the related tasks. This idea was explored by [6]. In this case, several related NLP tasks share embeddings.

Unfortunately, this means that one must find similar, yet different classification. Usually, this is accomplished by finding related classification data-sets. Sometimes, though, this is not an option. In those cases, it is sometimes possible to develop auxiliary tasks from the same data, forcing the network to develop features good for both tasks. We explore this idea by grouping classes together into so-called super-classes in different ways, such that the new groupings are related enough to not distract from the learning of the primary task but is different enough to inject new information into the network.

Of course, it is not enough to find similar tasks for effective multi-task learning. One must also find a way to share features. In neural networks, it is natural to simply bifurcate the network. All features before the bifurcation are shared and all features after the bifurcation are not.

2.2 Learning Super-Classes

We try several schemes for choosing super-classes. As a baseline, we use the fixed hierarchies given by the datasets themselves in ImageNet, Caltech-256, and CIFAR-100. Image hierarchies group similar classes, or classes which are difficult to tell apart, together. This is done recursively, in a tree-like fashion. Because during training networks may be pushed away from the global minimum by trying to discern classes that are challenging to discriminate (and therefore don't provide a uniform gradient for learning), grouping similar classes together eliminates that distraction. In order to form these super-classes, we go up one level in the tree from the original classes and use that to group classes together.

We also learned the super-classes, using the label tree learning described in [2]. Although this algorithm was originally used to speed up object-categorization in situations where the number of categories is much larger than practical, we use the algorithm to learn a set of super-classes in which the sub-classes are hard to tell apart. Below we review the algorithm, and then show how it is adapted to our problem.

Each node in the label tree is a set of distinct classes from the training set, and children define a partitioning of the set. Let us now describe how the tree is generated. Starting from the root node, which is the set of all distinct training classes, we want to find a partition in which each subset contains classes which are easily distinguished from each other. This is accomplished by training a copy of the model without an auxiliary objective (so, in essence, just a standard CNN), and computing its confusion matrix on a separate validation set. Interpreting the confusion matrix with classes as vertices and the probability of confusion as edge weights, we solve a min-cut problem on the graph. This leaves with two partitions of the original graph which preserve the most confused classes. This is done recursively until all clusters have only one class.

Hypothesizing that there may also be situations where the closest minimum to the initialization is the one which discriminates only the easy classes well, we inverted the confusion matrix (by subtracting each value in the confusion matrix from one), to form a graph where high edge values mean low-confusion rates. Performing the above method on this modified confusion matrix gives us clusters of classes that are easy to discriminate, but super-classes that are hard to discriminate. Learning with this auxiliary task could pull the learning away from the 'easy' minimum and bring it to a global minimum.

2.3 Random Restart

We have also found, that in the case of heavily overfit networks, it is beneficial to re-initialize later layers, keeping the learned weights of the earlier ones. There is no freezing of the weights in this scenario.

Intuitively this makes sense, heavily over-fitting networks run themselves into local minima quite quickly. These local minima are usually not terrible: the network still performs many times better than random chance alone. Therefore, the lower-level filters are semantically meaningful and pre-initialize the network to a place of learning that is most likely relatively close to a better optimum. Re-initilizing later values injects some stochasticity into the training of the network that knocks it out of its global minimum.

3 Experiments on ImageNet

ImageNet is a dataset of about 15 million-images, belonging to over 20,000 categories. The images were collected from the web and labeled by Amazon Turk according to the nouns of the word-net hierarchy [7].

The word-net hierarchy is a tree-like structure, where the root is the set of all words. Each node is called a synset, and each child of a node represents a super-subordinate relationship, where the child is a more specific synset than the parent, where the child synset is an instance of the parent. Although the word-net hierarchy is tree like, in the strictest sense it is not a tree since each child may have several parents.

The abundance of data within ImageNet as well as a principled and pre-defined hierarchichal structure made it a sensible choice for our experiments.

3.1 Sampling from ImageNet

Because we are interested in exploring the capablilties of large neural networks in scenarios where little data is present, we subsample from the 15 million images of ImageNet. In order to do this, we transform-the tree-like structure into a tree through a breadth-first search. We then select some percent of the classes in the third level in the hierarchy.

We then randomly split the dataset into a training and a testing set, with about 90 percent of data in the training set and 10 percent in the testing set. We are then left with some 240 classes, 142 super-classes, 171,631 training images, and 15,821 testing images. For the sake of clarity, we call this ImageNet-240.

The images in ImageNet are of variable size, so we had to transform them to a fixed size. In this, we followed [13] and given an image we rescaled the shorter side to 256 pixels and then cropped the center.

There was no other pre-processing other than zero-centering of the values. All accuracies reported are overall accuracies.

3.2 Model Architechture and Training Details

Because the network architecture originally presented by Alex Krizhevsky has proven to be so general, we decided to use it. As such, we have a model with 5 convolutional layers followed by 2 fully-connected layers and one softmax-layer with 240 outputs (for the 240 classes in our dataset), just as in [13]. However, because the problem is of a much smaller magnitude than that of the ImageNet Challenge , we scaled down the number of kernel mappings in all of the convolutional layers to half.The first,second, and fifth convolutional layers are followed by max-pooling with a kernel size of 3x3 and stride of two. The first convolutional layers has a kernel size of of 11x11 and a stride of 4. The second convolutional layer has a kernel size of 5x5 and a stride of 1. The third, fourth, and fifth convolutional layers have kernel sizes of 3x3 and strides of 1. The convolutional layers have (48,128,192,192,128) kernel maps going from the first convolutional layer to the fifth.

Response normalization, as described in [13], is applied after the first and second convolutional layers.

Weight decay was applied to all layers, with a value of 0.0005. Weight decay was not added to the biases.

The learning rate on all biases was twice the learning rate of the weights.

Dropout was applied to the fully-connected layers with a value of 0.5.

In our models, when there is a bifurcation the two splits are identical except for the softmax layer, and the backpropagated error is split evenly so that effective learning rate stays constant in the earlier layers. The bifurcation location was determined by experimentation.

All images were presented as a randomly cropped and randomly flipped image.

We experiment with two variants of training and architecture. One is the baseline, with the architecture and learning rate schedule described above. Another is the bifurcated model, splitting at the 4th convolutional layer. One branch has a 240-way softmax relating to the primary objectice. The other branch has 142-way softmax relating to the auxillary objective, or classifying into the super-classes defined by the ImageNet hierarchy.

The learning rate is initially set to 0.01, but is decreased by a factor of 10 each time testing accuracy falls below that of the previous test. Testing occurs every 1,000 iterations. A mini-batch size of 256 is used.

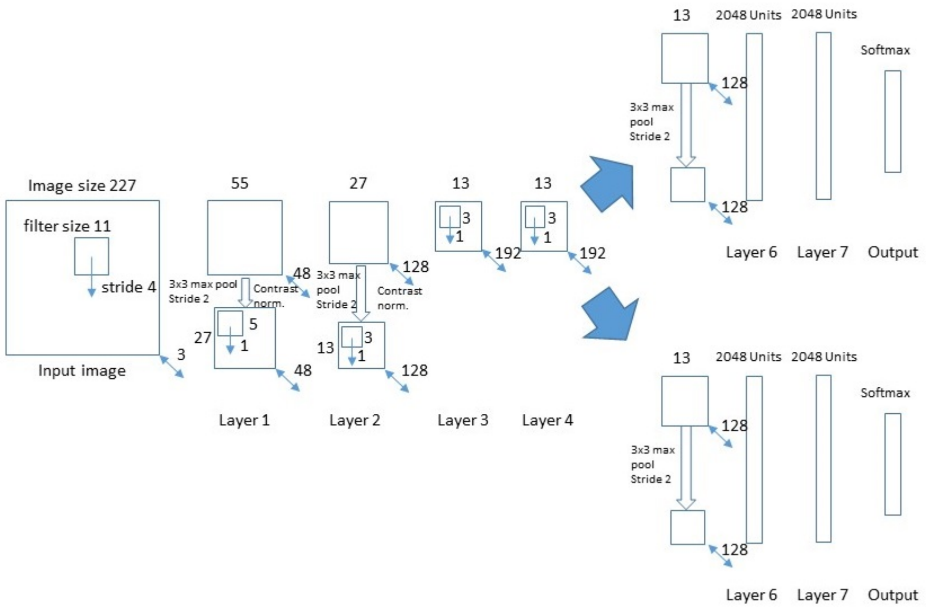

Figure 1: . An schematic of the network used for our ImageNet-240 subset. A very similar network is used for the Caltech-256 experiments. Notice the bifurcation after the 4th convolutional layer.

3.3 Experiments on few images per class

It has already been shown that this general architecture works well for large subsets of ImageNet [13]. However, as data set get smaller and smaller, large network architectures tend to overfit. In order to investigate our success with dealing with this, we artificially scarcify the dataset. We randomly sample 25%, 50%, 75%, and 100% of the training set, and train both models described in the previous section. It is clear that the bifurcation adds generalization power to the network.

Table 1: Classifiction Results: ImageNet-240

Method	25%	50%	75%	100%
Baseline	34.80%	41.6%	46.57%	49.22%
Bifurcated Network	36.91%	44.82%	46.45%	52.3%

A baseline network and network bifurcated at the 4th convolutional layer trained on subsets of the dataset. These subsets are 25%, 50%, 75%, and 100% of the size of our ImageNet-240 dataset. The auxiliary task in the bifurcated network is trained on super-classes that are just the parents of the original classes in the ImageNet hierarchy.

4 Experiments on Caltech-256

Caltech-256 is a relatively challenging dataset with relatively few data samples. This training data scarcity makes successful models highly problem specific, with architectures that incorporate dataset knowledge, or by generating features that are highly-discriminative but hand-engineered, limiting problem generalization. Demonstrating that through regularization, deep neural net models can in fact perform competitively on Caltech-256 is an important demonstration of their power. All training samples were warped to 256x256 images, but otherwise no pre-processing was done. All experiments were done with 60 training images per class and all accuracies are per-class accuracies.

4.1 Model Architecture and Training Details

The base architecture is identical to that of that for ImageNet-240, with the exception of having half as many units in the fully-connected layers and 256-way softmax for the final classification task. The convolutional layers have (48,128,192,192,128) kernel maps going from the first convolutional layer to the fifth. The two fully connected layers that follow have 2048 units each.

As in the ImageNet experiment, the learning rate is initially set to 0.01 and is decreased by a factor of 10 each time testing accuracy falls below that of the previous test. Testing occurs every 1,000 iterations. A mini-batch size of 256 is used.

4.2 Classification Results

On Caltech-256, we experiment with two schemes. One scheme is just a baseline, trained with the architecture presented in the previous section but without a bifurcation. Another is the architecture, with a bifurcation at the 4th convolutional layer, with the auxiliary task being the meta-classes of the classes as defined in [9].

Table 2: Classifiction Results: Caltech 256

Method	Test Accuracy
Baseline	35.04%
Bifurcated	38.67%

The Bifurcated model splits at the 4th Convolutional Layer

5 Experiments with Class Groupings

In this section we explore results from different agglomerations of classes for the auxiliary task. We generate the agglomerations using spectral clustering as described in the methods section. First we explore grouping similar classes together, and travel up and down the label tree. Second we group different classes together using spectral clustering, and travel up and down the label tree. Clearly, no matter the agglomeration, there is a significant improvement over the baseline with no bifurcations. In the following table, the super-class agglomerations are keyed as follows: D stands for super-classes of very different classes, and S represents super-classes of like classes. The number that follows is the number of clusters. The size of the clusters approximates 25%, 50%, and 75% of the total number classes, but the trees do not allow for perfectly neat partitioning.

Table 3: Varying Auxiliary Tasks: Caltech 256

Method	Test Accuracy
Fixed Hierarchy	38.67%
D-54	37.9%
D-119	38.178%
D-188	38.88%
S-61	39.24%
S-144	38.87%
S-191	39.96%
Duplicate Bifurcations	39.58%

D stands for clusters containing very different (easily discriminated classes). S stands for clusters containing similar classes (challenging to discriminate). The number that follows is the number of clusters in the auxiliary task.

5.1 Averaging bifurcations

Since there is no significant advantage to any particular, it is prudent to simply replicate the 256-way soft-max in the auxiliary task. This way, one can average the output of the two softmax's, both of which are strong models. This pseudo-model ensembling comes for free with this training, and boosts performance. What's more, as can be seen in the table below, this model performs better than averaging the predictions of two baseline models and is computationally cheaper: training the bifurcated model takes just over half the memory of ensembling two separate models.

Table 4: Averaging outpus: Caltech 256

Method	Test Accuracy
Ensembling 2 Baseline Architectures	40.11%
Averaging Duplicate Bifurcations	41.15%

Having an auxiliary task that is a copy of the primary task, and averaging the outputs of the bifurcated network, works better and consumes less memory than averaging the outputs of 2 baseline networks.

5.2 Bifurcation Placement

A natural questions is: does where you place the bifurcation matter? Intuitively, it would makes sense that the closer to the soft-max the bifurcation is placed, the more regularized the network is. Similarly, if one is averaging the outputs of the two bifurcations, it is also beneficial for the two bifurcations to be diverse in their predictions: if two networks share too many weights, they will predict similar things, and combining their outputs will give little information gain.

Table 5: Bifurcation placement: Caltech 256

Method	Test Accuracy
Bifurcated-1st Convolutional Layer	40.45%
Bifurcated-4th Convolutional Layer	41.15%
Bifurcated - 1st Fully Connected Layer	38.9%

There is a clear optimum for bifurcation placement when averaging outputs.

6 Experiments on CIFAR-100

The CIFAR-100 image dataset is a subset of the Tiny Images dataset [12]. It has 50000 training images and 10000 testing images, divided equally into 100 categories. These classes are then grouped into 20 super-classes.

As in in [8], we pre-processed the images doing global contrast normalization followed by ZCA whitening.

6.1 Model Architecture and Training Details

Srivastava and Salakhutdinov's work in [17] provides a very strong baseline for our work, so that is the basis of the model that we use. There are three convolutional layers, and then subsequently two fully connected layers. The convolutional layers have a kernel size of 5 x 5 and are applied with a stride of 1. All of the convolutional layers are associated with max-pooling, with pooling regions of size 3 x 3 and applied at a stride of 2. Dropout is applied to all the layers with the probability of setting the units activation to 0 being 0.1, 0.25, 0.25, 0.5, 0.5, 0.5, from input to the fully connected layers. Weight decay of 0.002 was applied to all weights, but not biases, and the learning rate for the fully connected layers is 10 times that of the convolutional layers. Weights are initialized from a zero mean normal distribution with a standard deviation of 0.01.

There is no data augmentation in these experiments. A decision on where to bifurcate was based on a simple grid-search.

The learning rate is initially set to 0.001, and is decreased by a factor of 10 every 35,000 iterations. We train for 140,000 iterations.

6.2 Classification Results

Applying what was learned on the Caltech-256 and ImageNet to a completely different architecture and dataset demonstrates the generality of our approach. Bifurcating the network improves original results, and averaging the results of the two bifurcations gives state-of-the-art results.

Table 6: Averaged Classifiction Results: CIFAR-100

Method	Test Accuracy
Baseline	62.47%
Bifurcated-1st Convolutional Layer	63.07%
Bifurcated - 1st Convolutional Layer, Averaged	65.01%

7 Adding Random Restart

Although the networks we have are heavily regularized with Dropout, they still heavily overfit. For instance, both the ImageNet and Caltech-256 models have a training loss of near 0 after a while. This is because the network has enough parameters to memorize the training set instead of learning visually meaningful filters for discrimination. Nevertheless, the network achieves reasonable testing accuracy even though it runs itself into minima that are not optimally meaningful. Intuitively this means that many parameters do convey good information, but some do not. In order to knock the learning out of these bad minima, we re-initialize some layers to weights sampled as they were when the network was first initialized.

We do this in two flavors. The first we call Random Restart, and is when we apply this re-initialization to the layers after the bifurcation in the network. Seeing that this improved classification on both ImageNet and Caltech-256 significantly (Figure 2 and Table 7), we also introduce Recurring Restart. In Recurring Restart we re-initialize the network at layer l after 10000 iterations. l starts at the first layer, and after each restart is moved forward one layer. Once the last-layer is restarted for the last time, the network is trained for 30000 iterations to maximize testing accuracies. Recurring Restart brings the Caltech-256 problem to 2nd only to Hierarchichal Matching Pursuit [3] for the 60-image training scheme. Figures 3 and 4 show the testing accuracy and training loss as a function of training iterations.

Table 7: Classifiction Results: Caltech 256

Method	Test Accuracy
Baseline	35.04%
Bifurcated	38.67%
Bifurcated w/ Random Restart	42.35%
Bifurcated w/ Recurring Restart	49.37%
Convolutional Restricted Boltzmann Machines [16]	47.94%
Hierarchichal Matching Pursuit [3]	58%

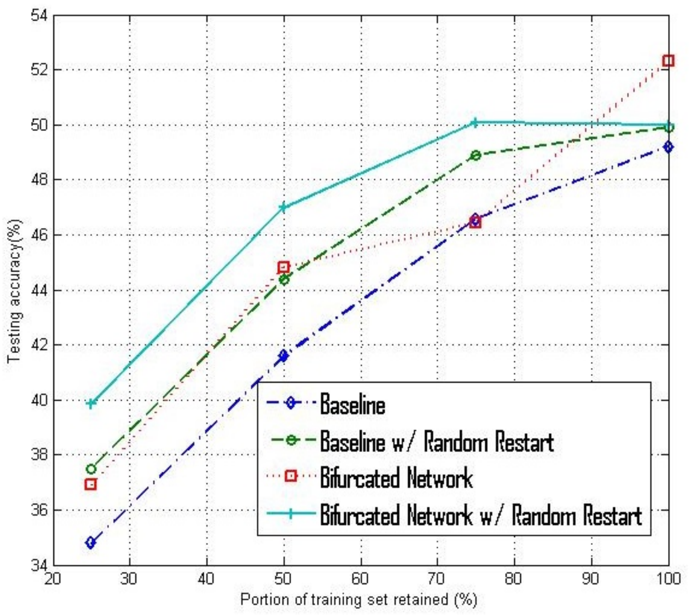

Figure 2: . Comparison of different architectures for ImageNet classification. The bifurcated architecture combined with Random Restart performs best, especially for small subsets of the training set.

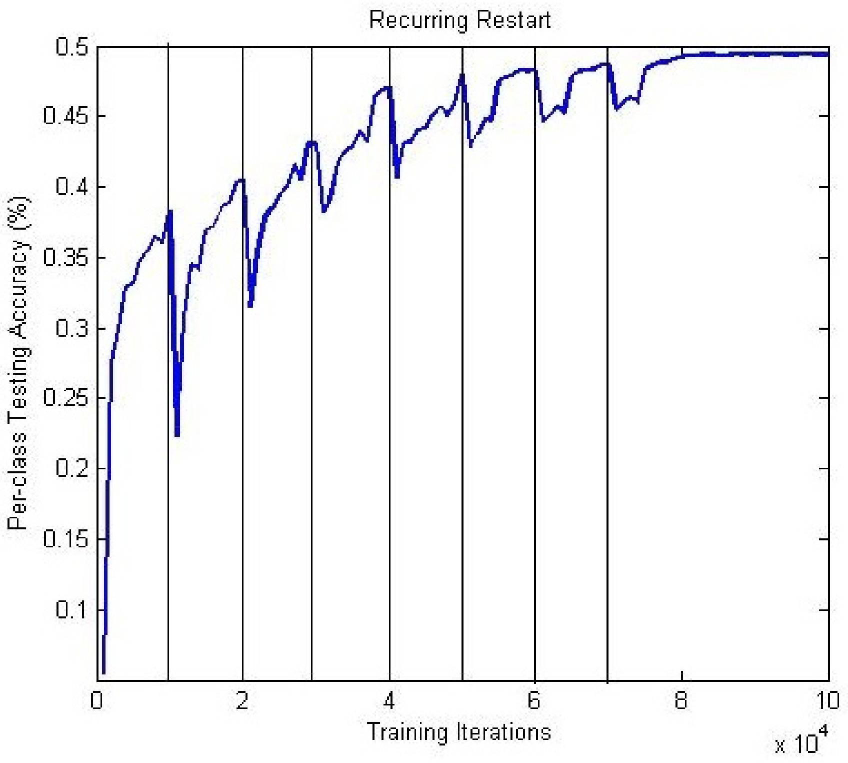

Figure 3: . Testing accuracies as a function of iterations. Each vertical line indicates a re-initialization of later layers in the network. Since there are 8 layers, there are 7 reinitializations.

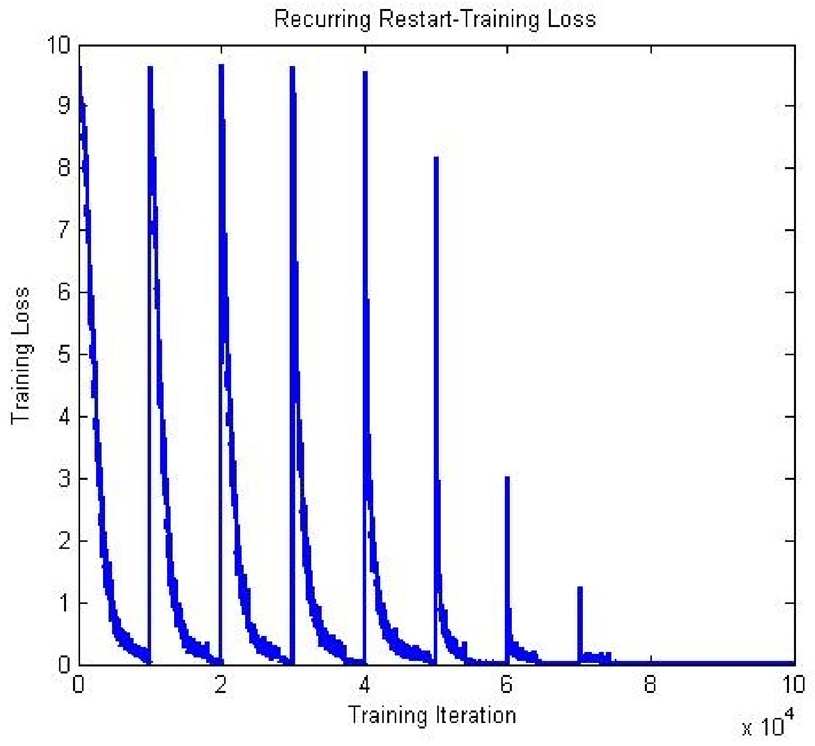

Figure 4: . The training loss in Recurring Restart. Before each restart, the training runs itself into a local minimum that memorizes the training set.

8 Conclusion

In this paper, we explored Convolutional Neural Networks in the context of little data. First, we explored using fixed image hierarchies as an auxilary task in a form of multi-task learning. This improved classification results on both subsets of ImageNet and on the per-class accuracies of Caltech-256. We then explored different auxiliary tasks, and found that if one simply replicated the original form as an auxiliary task, one could average the outputs of the two tasks together in a form of pseudo-ensembling. This memory-conscious technique further improved classification results on Caltech-256, and gave state-of-the-art results on CIFAR-100.

Lastly, we added Random Restart, further improving results on both the subsets of ImageNet and Caltech-256 datasets, making our Caltech-256 results second only to one method in the literature when trained on 60 examples per class.

References

[1] A. Ahmed, K. Yu, W. Xu, Y. Gong, and E. P. Xing. Training hierarchical feed-forward visual recognition models using transfer learning from pseudo-tasks. In *ECCV (3)*, pages 69–82, 2008.

[2] S. Bengio, J. Weston, and D. Grangier. Label embedding trees for large multi-class tasks. In J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, editors, *NIPS*, pages 163–171. Curran Associates, Inc., 2010.

[3] L. Bo, X. Ren, and D. Fox. Multipath sparse coding using hierarchical matching pursuit. In *CVPR*, pages 660–667, 2013.

[4] R. Caruana. Multitask learning. *Machine learning*, 28(1):41–75, 1997.

[5] D. C. Ciresan, U. Meier, and J. Schmidhuber. Multi-column deep neural networks for image classification. *CoRR*, abs/1202.2745, 2012.

[6] R. Collobert and J. Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning, 2008.

[7] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[8] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. C. Courville, and Y. Bengio. Maxout networks. In *ICML (3)*, pages 1319–1327, 2013.

[9] G. Griffin, A. Holub, and P. Perona. Caltech-256 object category dataset. Technical Report 7694, California Institute of Technology, 2007.

[10] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012.

[11] Y. Jia. Caffe: An open source convolutional architecture for fast feature embedding. `http://caffe.berkeleyvision.org/`, 2013.

[12] A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, volume 1, page 4, 2012.

[14] A. Krogh and J. A. Hertz. A simple weight decay can improve generalization. In *ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS 4*, pages 950–957. Morgan Kaufmann, 1992.

[15] M. Lin, Q. Chen, and S. Yan. Network in network. *CoRR*, abs/1312.4400, 2013.

[16] K. Sohn, D. Y. Jung, H. Lee, and A. O. Hero. Efficient learning of sparse, distributed, convolutional feature representations for object recognition. In *Proceedings of the 2011 International Conference on Computer Vision*, ICCV '11, pages 2643–2650, Washington, DC, USA, 2011. IEEE Computer Society.

[17] N. Srivastava and R. Salakhutdinov. Discriminative transfer learning with tree-based priors. In *NIPS*, pages 2094–2102, 2013.

[18] L. Wan, M. D. Zeiler, S. Zhang, Y. LeCun, and R. Fergus. Regularization of neural networks using dropconnect. In *ICML (3)*, volume 28 of *JMLR Proceedings*, pages 1058–1066. JMLR.org, 2013.

[19] M. D. Zeiler and R. Fergus. Stochastic pooling for regularization of deep convolutional neural networks. *CoRR*, abs/1301.3557, 2013.

[20] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. *CoRR*, abs/1311.2901, 2013.