PROCEEDINGS OF SCIENCE

A Convolutional Hierarchical Neural Network Classifier

Ismail Gadzhiev[a,*] **and Sergey Dolenko**[a]

[a] *D.V.Skobeltsyn Institute of Nuclear Physics, M.V. Lomonosov Moscow State University, 1(2) Leninskie gory, Moscow 119991, Russian Federation*

E-mail: ismailgadzhievff@gmail.com, dolenko@srd.sinp.msu.ru

The report presents an algorithm for constructing a convolutional hierarchical neural network classifier, which is a modification of the algorithm for constructing hierarchical neural network classifiers suggested before. The original algorithm was designed to exploit intrinsic class hierarchy to build a class tree with a neural network in each node classifying groups of initial classes (in a non-terminal node) or a subset of original classes (in a terminal node). The convolutional modification utilizes convolutional neural networks instead of regular fully connected networks in order to apply the model to image classification tasks. Use of class hierarchy for image classification should reduce the number of adjusted neural network parameters compared to deep convolutional neural networks, and therefore it should reduce training and inference time. In this context the algorithm may be compared with some pruning techniques. The convolutional hierarchical neural network classifier inherits some hyperparameters of a conventional hierarchical neural network classifier, like the activation threshold and the threshold by the share of voting patterns. The goal of this study was to explore different strategies of choosing these hyperparameters. To test these strategies, we used the CIFAR-10 dataset. Also, for demonstration purposes we apply the convolutional hierarchical neural network classifier to the CIFAR-100 dataset.

The 5th International Workshop on Deep Learning in Computational Physics
28-29 June, 2021
Moscow, Russia

*Speaker

PoS(DLCP2021)014

1. Introduction

In this paper we present a novel algorithm for image classification. It is a modification of Hierarchical Neural Network classifier (HNNC) presented in paper [1]. HNNC was designed to utilize adaptively formed class hierarchy in multi-class classification tasks.

In this new modification we use convolutional neural networks instead of regular multi-layer perceptrons with fully-connected layers only to apply the model to image classification tasks. We call this modification Convolutional Hierarchical Neural Network Classifier (CHNNC).

The main hope of the algorithm development is to build a classifier of acceptable quality that is cheap in terms of computational complexity. Usage of class hierarchy allows one to train fewer number of parameters compared to deep convolutional neural networks that now show the best results in image classification. However, recent studies show that most of the weights' values optimized during training do not affect model's prediction much. In the paper [2], a "lottery ticket" hypothesis is formulated, according to which fully connected networks contain subnetworks comparable to the original network by classification metrics after several iterations of training.

In the context of reducing the number of parameters, our algorithm can be compared to pruning techniques, which allow reducing the number of weights by dropping useless ones.

To demonstrate the performance of CHHNC, we use CIFAR-10 and CIFAR-100 datasets [3] for image classification tasks. Besides that, we explore the effect of some hyperparameters tuning on the hierarchical classifier quality. To demonstrate this effect, we use the CIFAR-10 dataset.

2. Convolutional Hierarchical Neural Network classifier

CHNNC, like HNNC, is a tree, at each node of which there is a neural network that takes as input patterns from some subset G of the set of all classes K. This network classifies patterns into several groups corresponding to subsets $S \subseteq G$ of the original subset of all classes. The patterns divided to be mapped into the resulting groups are recursively classified by the neural networks in the child nodes. So, for example, at the root node of the tree there is a neural network that accepts the entire set as input and classifies it into several groups of classes. At the leaf nodes there are networks that classify the patterns that have reached the node by their original labels. An example of the hierarchical class tree is shown in Fig. 1.

This section discusses the details of training CHNNC and also its tuning hyperparameters.

CHNNC is trained in two stages - the first is the construction of a class hierarchy, at the second stage we replace the models at the nodes with new ones, and train them again with the fixed class hierarchy. The first stage is the same as the training algorithm for the original HNNC model. The necessity of the two-stage training is explained in the next section.

This article discusses hyperparameters that allow one to control only the first stage of training - the threshold for the activation of the output neurons.

2.1 Constructing class hierarchy

Let us describe the algorithm for constructing a class tree for CHNNC - the first stage of training, which is the same as the training algorithm for HNNC. As mentioned earlier, at each node

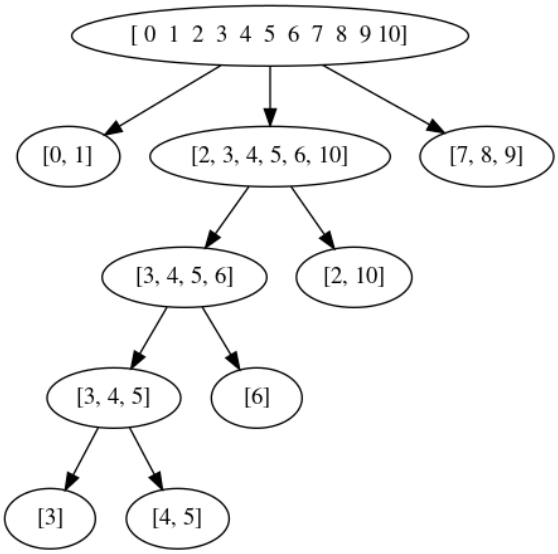


Figure 1: An example of class hierarchy.

of the tree there is a neural network that learns to recognize patterns from a subset of all classes. In the process of training the network, the classes that the network most often confuses are merged.

To determine which classes the network confuses with each other, the procedure of "voting" of the patterns of each class is used; those classes are merged, most of the patterns in which "voted" for the same class. (We consider that a pattern has voted to some class if the maximum amplitude among the outputs of the network when this pattern is fed to its inputs corresponds to this class; some possible additional conditions will be discussed below in Section 2.6) After voting, the desired answers for the merged classes are made the same, and the training of the network continues with the examples re-labeled in this way. In cases where further merging of classes is impossible, or if the specified maximum number of epochs has passed without merging, or if the increase in classification metrics is insignificant, the training procedure stops.

As the result, the classes are merged into groups in accordance with how the network at the node confuses them; the efficiency and quality of training after the classes are merged significantly increase. Then, for each of the obtained groups, a new neural network is trained - in this way a class tree is recursively built.

To force the neural networks at the nodes to confuse classes even when those networks are trained enough (and they show no significant improvement of classification metrics after next iteration of training), we use “weak” neural networks, i.e., networks with a small number of hidden neurons (and therefore a small number of weights).

2.2 Use of convolutional neural networks

CHNNC is a modification of HNNC designed for image classification. Therefore, convolutional neural networks are used as models in the nodes at both stages of training.

In particular, the following architecture is used as a "weak" neural network: a convolutional layer with 2-3 neurons, then fully connected cascades with 2-3 neurons in a single hidden layer. The architecture of the model is shown schematically in Fig. 2.

Figure 2: Architecture of the weak convolutional neural network for the first stage.

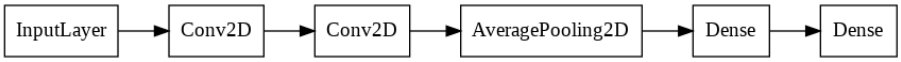


Figure 3: Architecture of a "strong" convolutional neural network for the second stage.

2.3 Inference mode

Before moving to the second stage of CHNNC training, let us explain how the prediction is made, as it is crucial to understand the necessity of the second stage. This procedure is common for CHNNC and HNNC.

At the first step, a pattern is classified by the neural network in the root node of the class hierarchy as belonging to one of the groups of classes that represent its child nodes. The node N with maximum output activation is chosen, and the pattern is classified by the network in N. This process is repeated recursively, and the pattern follows a path down the tree, until it reaches a node preceding the leaf nodes. Such nodes make final predictions, classifying the pattern into one of the original classes that are represented by the leaf nodes.

2.4 The second stage of training

The purpose of using weak neural networks at the first stage of the algorithm is that they confuse similar classes; this allows us to build a class hierarchy with classes of visually close objects recursively grouped at the nodes. However, the classification quality of networks in the nodes is poor, and that implies even worse results of the tree as the whole. As every pattern is classified sequentially by the networks in the nodes following some path in a tree, the accuracy of all the models is multiplied, and the poor classification quality of the model in the top node worsens the total accuracy of the tree.

To address the issue and to improve the quality of classification of the CHNNC, we first replace the "weak" networks in the nodes with "stronger" ones, i.e., with additional layers and weights. Then we train those networks with regular neural network training algorithm to classify patterns in each node N into the groups corresponding to the child nodes of N. To put it simply, we fix the class hierarchy obtained at the first stage and train new stronger networks in the nodes so that their classification quality is better than that of original ones.

A neural network with two convolutional layers of 32 neurons in each and fully connected cascades with 10 neurons in a single hidden layer was used as the "strong" neural network. The network architecture is shown schematically in Fig. 3.

2.5 Overall training details

We split the training data into two parts in a $m : n$ ratio ($m < n$) and use these parts for the first and second stage of CHNNC training. This is done for several reasons listed below.

Firstly, "weak" networks at the first stage have fewer parameters and therefore need less data to train. That is why we use only a fraction of data for the first stage. This also makes the first stage faster.

Secondly, we use separate parts of the original training set to avoid possible overfitting. If we perform first and second stages of training on the same set, the second stage will use class hierarchy that has already "seen" the training set.

The proportion of the split and its necessity is the subject for further discussion, but it is outside the scope of the paper. In this paper we choose $m : n$ ratio to be 1:4.

2.6 Thresholds for activation and for portion of voting patterns

The main questions that immediately arise are how do we know that the "weak" network at the first stage is trained enough to confuse classes in the right way, i.e., to confuse similar classes with each other, and how much confusion do we actually need (generally, the greater is the number of confused classes that are merged, the deeper grows the class hierarchy). Let us take the extreme case for clarity – the network after zero training iterations. Obviously, the voting procedure results will be determined by randomness (as the weight initialization is random). The effect of random weight initialization is decreased after every epoch of training. To perform a correct voting procedure, we should exclude this random effect by imposing some additional constraints. As the definition of the effect of randomness in the voting procedure is fuzzy and unclear, the methods proposed are rather heuristics than universal recipes.

The first attempt is to constraint the number of epochs after which the voting could be performed, but it might be very task-dependent. In the paper [1] in which the original HNNC is described, two hyperparameters were proposed – thresholds for maximum output activation θ and for portion of voting patterns γ.

Only votes of those patterns are taken into account for which the maximum output neuron activation is greater than θ. This is done to cut off the localization area for output activations after random initialization of the weights. If the activation is greater than θ then the network is believed to be trained enough for this pattern to participate in the voting.

It is shown in [1] that use of γ allows one to improve classification quality of HNNC.

After voting, the class i is merged with the class j if the portion of patterns from i that voted for j is greater than γ. It allows one to exclude voting results caused by small random fluctuations that would produce unnecessary nodes in the tree.

In this study we test an adaptive method for selecting θ at each node. We model θ as an upper bound for activation localization area. Consider a network with a normalized output activation like softmax. Obviously, regardless of the distribution used to initialize the weights of the network, the mathematical expectation of the output activation is $\frac{1}{N}$, where N is number of classes in the node. It follows that the localization area should generally decrease with N.

We define the localization area as

$$\theta = \mu + k\sigma, \tag{1}$$

where μ and σ are the mean and the standard deviation of the output activations over the patterns of the training set during random initialization; coefficient k is chosen by grid search.

In this paper we test both fixed and adaptive method for the activation threshold selection.

Table 1: CHNNC accuracy values for the CIFAR-10 and CIFAR-100 sets after the first and second stages of the model training

Dataset	Stage 1	Stage 2
CIFAR-10	33%	63.1%
CIFAR-100	1.4%	11%

2.7 Other hyperparameters of the algorithm

Let us briefly mention other hyperparameters thar remain fixed in this study.

We use the maximum number of epochs without merging and the maximum number of all epochs to terminate training at each node at the first stage. Also, for training of the original HNNC, a timeout between mergings was set. In this study we set this timeout to 1 epoch.

Other parameters are the parameters of the weight optimization (training) algorithm, for example, the learning rate, as well as the batch size. To solve the optimization problem in this paper, we used the Adam algorithm.

3. Results

For the testing purposes, we used CIFAR-10 and CIFAR-100 well-known datasets for image classification [3]. To test the dependence of the accuracy on the values of the thresholds, we used CIFAR-10 only.

To estimate the accuracy in each case, 5 runs of training were carried out with various values of weight initialization random seed. For each run we calculate the accuracy, and average this value across all runs.

To implement CHNNC, we used Tensorflow [4] library.

3.1 Results on CIFAR-10 and CIFAR-100

Table 1 shows the CHNNC accuracy values for the CIFAR-10 and CIFAR-100 sets after the first and second stages of the model training. It can be seen that the second stage improves the quality of the CHNNC classification.

We also conclude that accuracy values on CIFAR-100 are still unsatisfactory even after the second stage. This is subject for further research.

3.1.1 Class hierarchy and merge diagrams

Fig. 4 shows a class hierarchy tree obtained for CIFAR-10. Fig. 5 visualizes the process of class merging in the nodes.

As one can see, the class hierarchy obtained adaptively follows natural intuition about class similarity.

Interestingly, the obtained hierarchy may be potentially reused for different tasks. For example, YOLO9000 [5] model for object detection exploits semantic hierarchy graph obtained from WordNet database to construct complex loss function. However, semantic similarity does not accurately

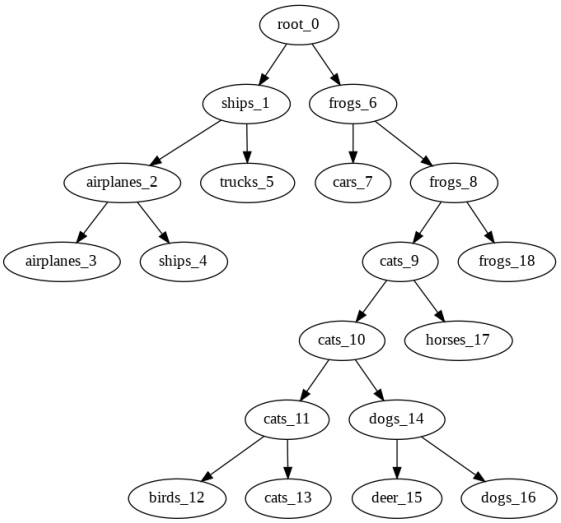


Figure 4: Class hierarchy obtained for CIFAR-10. Indices represent node numbers.

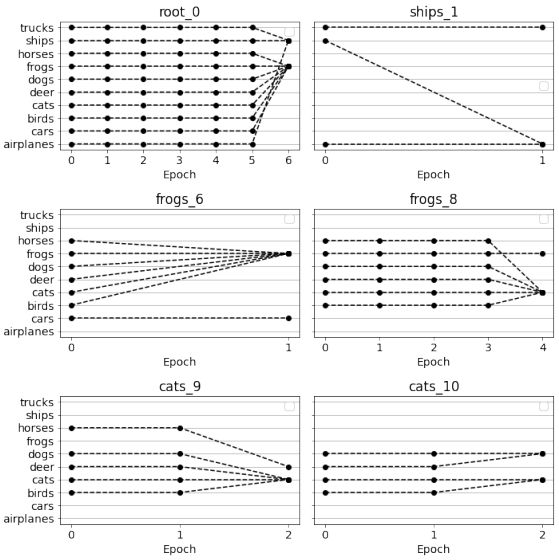


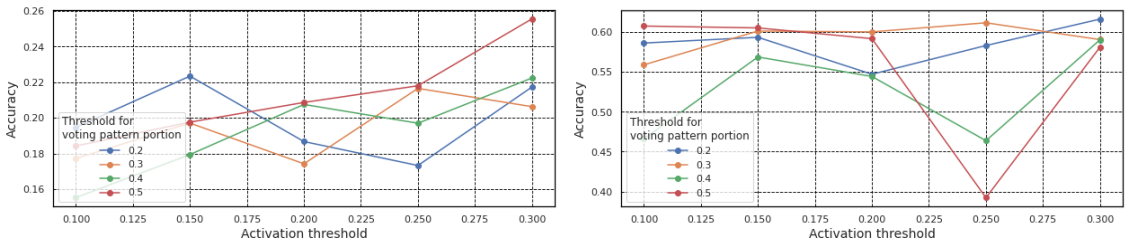
Figure 5: Class merge diagrams for CIFAR-10.

represent visual similarity, and the hierarchy from the first stage of CHNNC seems more suitable for this task.

3.2 Effect of thresholds

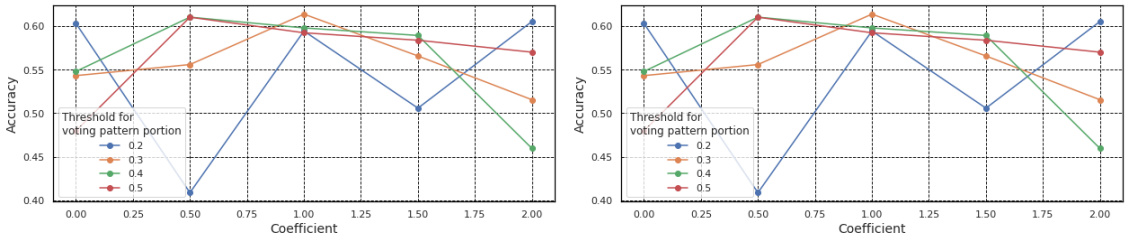
Fig. 6a shows the dependence of accuracy on the thresholds for the first stage. It can be concluded that the quality of the model without repeated training is unsatisfactory. Fig. 6b shows the same dependency for the second stage. The optimal value of the threshold for γ - $0.2 - 0.3$, while no pronounced dependence on θ is observed.

Fig. 7a shows the dependence of accuracy on the thresholds for the first and second stages. We again conclude that the quality of the model without repeated training is unsatisfactory. Fig. 7b



(a) **(b)**

Figure 6: Dependency of accuracy on activation threshold for different values of the threshold for voting patterns portion for the first (a) and second (b) stages.



(a) **(b)**

Figure 7: Dependency of accuracy on k for different values of the threshold for voting patterns portion for the first (a) and second (b) stages.

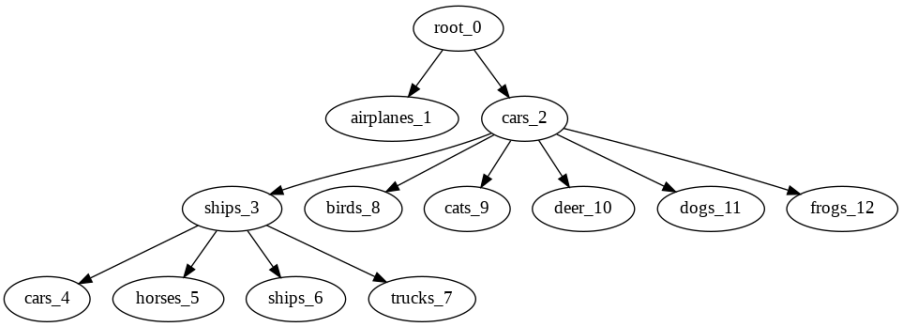


Figure 8: The tree view for $\gamma = 0.3$ and $k = 1$.

shows the same dependency for the second stage. The optimal value for γ is $0.3 - 0.4$, while the best result of 63.4% is achieved at $\gamma = 0.3$ and $k = 1$.

As we see, for convolutional modification of HNNC, the choice of the thresholds is not of such importance as for the original HNNC model, but it could also slightly improve the CHNNC performance.

Fig. 8 shows the view of the CHNNC tree for $\gamma = 0.3$ and $k = 1$.

4. Conclusions

We present a novel algorithm for constructing convolutional hierarchical neural network classifier that is a modification of the hierarchical neural network classifier. We test the model on CIFAR-10 and CIFAR-100 image classification problems. The necessity of two-stage learning is demonstrated. The first stage is constructing a class hierarchy tree and the second stage is retraining of more powerful models in the nodes. The dependence on the thresholds for output neuron activations and for voting patterns portion is tested. It is shown, that the threshold setup is not sufficient for CHNNC even though it can slightly improve performance.

5. Further development

Let us briefly list the potential fields for further research.

Firstly, testing the dependence of accuracy on threshold on CIFAR-100 is needed. Prior to such testing, optimal parameters for the weak and strong networks for CIFAR-100 problem should be determined by grid search.

Secondly, the algorithm should be tested on other datasets with high number of classes (from tens to first hundreds) different from classical CIFAR-10 and CIFAR-100.

Thirdly, we are interested in applying class hierarchy that is obtained at the first stage of CHNNC training to construct loss function for YOLO9000.

References

[1] V. Svetlov, I. Persiantsev, J. Shugay and S. Dolenko, *A new implementation of the algorithm of adaptive construction of hierarchical neural network classifiers*, *Optical Memory and Neural Networks (Information Optics)* **24** (2015) 288.

[2] J. Frankle and M. Carbin, *The lottery ticket hypothesis: Training pruned neural networks*, *CoRR* **abs/1803.03635** (2018) [1803.03635].

[3] A. Krizhevsky, *Learning multiple layers of features from tiny images*, Tech. Rep. (2009).

[4] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro et al., *TensorFlow: Large-scale machine learning on heterogeneous systems*, 2015.

[5] J. Redmon and A. Farhadi, *YOLO9000: better, faster, stronger*, *CoRR* **abs/1612.08242** (2016) [1612.08242].