Pattern Recognition Letters 112 (2018) 184–190

Contents lists available at ScienceDirect

Pattern Recognition Letters

journal homepage: www.elsevier.com/locate/patrec

Improving multiclass classification by deep networks using DAGSVM and Triplet Loss☆

Nakul Agarwal[a,*], Vineeth N Balasubramanian[b], C.V. Jawahar[a]

[a] Centre for Visual Information Technology, IIIT Hyderabad, Gachibowli, Hyderabad, Telangana 500032, India
[b] Department of Computer Science and Engineering, Indian Institute of Technology, Kandi, Sangareddy, Hyderabad, Telangana 502285, India

ARTICLE INFO

Article history:
Received 31 August 2017
Available online 7 July 2018

Keywords:
Multiclass classification
Deep networks
DAGSVM
Triplet loss

ABSTRACT

With recent advances in the field of computer vision and especially deep learning, many fully connected and convolutional neural networks have been trained to achieve state-of-the-art performance on a wide variety of tasks such as speech recognition, image classification and natural language processing. For classification tasks however, most of these deep learning models employ the softmax activation function for prediction and minimize cross-entropy loss. In contrast, we demonstrate a consistent advantage by replacing the softmax layer by a set of binary SVM classifiers organized in a tree or DAG (Directed Acyclic Graph) structure. The idea is to not treat the multiclass classification problem as a whole but to break it down into smaller binary problems where each classifier acts as an expert by focusing on differentiating between only two classes and thus improves the overall accuracy. Furthermore, by arranging the classifiers in a DAG structure, we later also show how it is possible to further improve the performance of the binary classifiers by learning more discriminative features through the same deep network. We validated the proposed methodology on two benchmark datasets, and the results corroborated our claim.

1. Introduction

Multiclass classification problems are fundamental in computer vision. Many vision tasks, e.g. object recognition, person recognition and scene classification, require the classifier to discriminate between multiple categories. Over the years, deep learning methods have claimed state-of-the-art performances in such tasks [14,16,23]. All of these deep architectures use the softmax activation function (also known as multinomial logistic regression) for classification. Although effective, these deep networks lack the scope for improvement without a major change in architecture for increase in classification accuracy.

In this paper, our goal is to design a multiclass classification method that improves upon the accuracy of the existing deep learning models without bringing about a change in the architecture of the earlier layers (focusing only on the output layer). The key contribution of our work is that we break down the N-class classification problem into $N(N-1)/2$ binary problems by replacing the softmax layer of a deep architecture by binary classifiers for each pair of classes arranged in a DAG (Directed Acyclic Graph) structure. Such a breakdown enables the network to focus on solving individual binary problems which are easier to solve. It also enables the network to further improve the performance of the binary classifiers by learning more pairwise discriminative features, which is especially effective since each classifier deals with only two classes. Another advantage of our method is that it can be easily applied to different deep architectures, since the method focuses only on the output layer. Fig. 1 illustrates the overall idea of the proposed approach.

The rest of the paper is organized as follows: in Section 2, we review related work. This is followed in Section 3 by the detailed description of our methodology. Section 4 presents the experimental results on two benchmark datasets. We finally conclude with the summary of our approach and future extensions in Section 5.

2. Related work

The family of multiclass classification algorithms that reduces multiclass problems to binary can broadly be categorized into two kinds, depending on whether the methods take advantage of the hierarchical structure in the label space. The first category of meth-

☆ The authors whose names are listed immediately below certify that they have NO affiliations with or involvement in any organization or entity with any financial interest (such as honoraria; educational grants; participation in speakers' bureaus; membership, employment, consultancies, stock ownership, or other equity interest; and expert testimony or patent-licensing arrangements), or non-financial interest (such as personal or professional relationships, affiliations, knowledge or beliefs) in the subject matter or materials discussed in this manuscript.

* Corresponding author.
E-mail address: nagarwal2@ucmerced.edu (N. Agarwal).

https://doi.org/10.1016/j.patrec.2018.06.034
0167-8655/

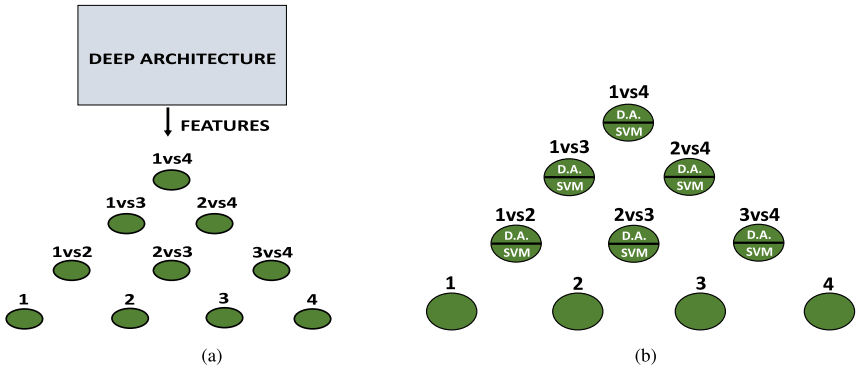

Fig. 1. Two variants of multiclass classification using binary classifiers arranged in a DAG structure. (a) shows the architecture of the model for a 4-class problem where features learned by the deep network are used to train all the classifiers. Here each node is a binary classifier for a pair of classes. In (b), the architecture for the same 4-class is shown but here each node represents a combination of the deep architecture (D.A.) and the Support Vector Machine (SVM) classifier. This makes it possible to learn more pairwise discriminative features to further improve the performance of the classifiers.

ods is generic and treats the label space as flat. It includes one-vs-all, one-vs-one with maximum wins (all pairs of classes are compared and the class with the most votes wins), error correcting output codes [1,6], output-coding based multiclass boosting [15,21], and DAGSVM [20] (one-vs-one classifiers are organized in a DAG to discard classes sequentially). The second class of methods exploits the hierarchical structure [2,9,26]. The main intuition behind these methods is to reduce the test time by finding some measure of separability to partition the classes into two subsets rather than pairwise.

In this work, we focus on adapting a DAG-based approach for two reasons: (i) it allows us to use a combination of binary classifiers where it is possible to effectively learn more discriminative pairwise features; and (ii) it has a lower time complexity than the corresponding one-vs-one method. Using convolutional nets in combination with SVMs (especially linear) have been proposed in the past as part of a multistage process. In particular, a deep convolutional net is first trained using supervised/unsupervised objectives to learn good invariant hidden latent representations. The corresponding representations of data samples are then treated as input and fed into linear (or kernel) SVMs [3,12,18]. Few other efforts in the past have also proposed similar models where the output layer of standard neural nets as well as convolutional neural nets is replaced by SVMs but with joint training of weights at lower layers [4,17,30]. Vinyals et al. [25] proposed a method to learn a recursive representation using linear SVMs at every layer, but without joint fine-tuning of the hidden representation. Weston et al. [28] proposed a semi-supervised embedding algorithm for deep learning where the hinge loss is combined with the contrastive loss from Siamese networks [10]. Softmx layer is replaced by linear SVMs in [24] where the whole network used the loss from the L2-SVM instead of the standard hinge loss for optimization. All of the above efforts however, treat multiclass classification as one single problem whilst we break it down into smaller binary problems and exploit the advantage of pairwise classifiers.

We now present the proposed methodology.

3. Methodology

3.1. Overview

Our goal is to design a model that is able to improve upon the accuracy of existing deep networks for classification tasks. To achieve this, we remove the softmax layer after training the deep network and extract features from the last layer to train a set of binary classifiers organized in a tree or DAG structure. Each classifier is trained to differentiate between only two classes, which is what is best done by popular classification methods such as support vector machines (SVMs) and logistic regression. By doing so, we break down the problem of multiclass classification into smaller binary problems where each classifier acts as an expert when it comes to choosing a decision path based on two classes, which in turn helps in improving the overall accuracy. As an example, Fig. 1(a) shows the DAG structure for a 4-class problem. The deep architecture is trained to perform classification using softmax, which in turn makes it learn good features for classification. We then take those features and train each of the 6 classifiers individually. The classifiers are based on SVMs and use linear kernel. At test time, an instance traverses the hierarchy from top to bottom until a leaf node where the final decision is made.

We also propose a variant of the same methodology where adding a set of binary classifiers arranged in a DAG like structure at the end of the deep architecture makes it possible to tweak the classifiers individually. For instance, to further improve the performance of the individual classifiers, we use the same original deep architecture to learn more discriminative features. This is done by fine-tuning the pre-trained deep network using the triplet loss [22] instead of the cross-entropy loss. Its corresponding structure can be seen for the same 4-class problem in Fig. 1(b). Each node in the DAG is now a combination of the fine-tuned deep architecture and a SVM classifier. The deep network inside each node has been specifically fine-tuned to find more discriminative features corresponding to the two classes of its classifier counterpart. At training time, we want the test accuracy of the classifiers to go up when compared to the case in Fig. 1(a). So, we use the newly learned discriminative features to train the individual SVM classifiers. At test time, an instance first passes through the deep network to get transformed into a new feature space and then passes through the classifier.

3.2. DAG

The DAG is a arrangement of nodes organized in a hierarchical structure where each node represents a binary SVM classifier, a configuration popularly known as DAGSVM. For a N-class problem, we have $N(N-1)/2$ binary SVM classifiers in the DAG. Support Vector Machines were originally proposed and work best for binary classification, and we make use of that fact. The performance of the DAGSVM can depend on the sequence order of the decision list, which is controlled by the arrangement of the binary classifiers in the DAG. For our experiments, we consider a DAG which has a fixed structure. There are several efforts in the past [7,19] that have focused on learning the hierarchy of the DAG for multiclass classification for computational efficiency, where the

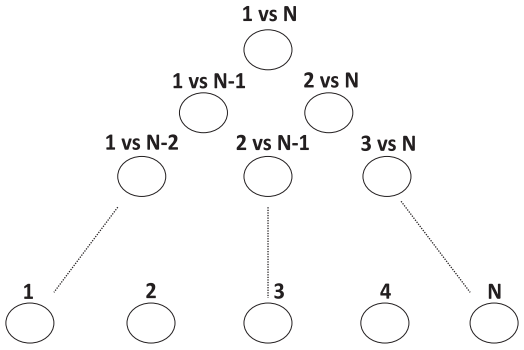

Fig. 2. Arrangement of the different binary classifiers in the DAG for a N-class problem.

nodes still represent a binary classifier but actually consider two subsets of classes. We maintain a fixed hierarchy in this work to show a proof-of-concept of our idea; besides, a separate classifier for a pair of classes makes learning of discriminative features more effective, as will be explained in Section 3.3.

Now we will define the structure of the DAG used in this paper. Let the problem be a N-class classification problem. The root node then is a classifier between classes i and j where i represents 1 and j represents N. Its left child node represents a classifier between classes i and $j-1$ and right child node between classes $i+1$ and j. The rest of the DAG is designed in a similar way, as can be seen in Fig. 2. We adhere to this configuration for purposes of simplicity and convenience, without any loss of generality. We do note that the performance of the proposed methodology can be improved if an optimal configuration can be identified for a given application.

3.3. Learning more discriminative features

In the second variant of the proposed methodology (Fig. 1(b)), we make use of the triplet loss [22] to improve the discriminability of the pairwise features. The triplet loss tries to enforce a margin between each pair of instances belonging to one class to instances from all the other classes. Since our model considers only two classes at a time, this makes the triplet loss even more effective in enforcing this discriminability. In particular, we strive for an embedding $f(x)$, from an image x into a feature space in $\mathbb{R}^d$, such that the squared distance between instances of the same class is small, whereas the squared distance between a pair of instances from different classes is large. $f(x)$ in our case is the learned descriptor embedding. The triplet loss is motivated in [27] in the context of nearest-neighbor classification. Basically we want to ensure that an image x_i^a (anchor) belonging to a particular class is closer to all other images x_i^p (positive) of the same class than it is to any image x_i^n (negative) of the other class. The loss L being minimized can then be given as:

$$L = \sum_{(a,p,n)\in T} \max\{0, ||f(x_i^a) - f(x_i^p)||_2^2 + margin - ||f(x_i^a) - f(x_i^n)||_2^2\} \quad (1)$$

where T is a collection of training triplets and $margin \geq 0$ is a scalar quantity which is empirically chosen.

Triplets in [22] were chosen based on a strategy where all possible anchor positive pairs were picked in a mini-batch while still selecting the *semi-hard* negatives. Since we're dealing with only two classes at a time, picking all possible anchor positive pairs in a mini batch is equivalent to randomly selecting any anchor positive pairs. *Semi-hard* negatives were picked in [22] for faster convergence. As we generate triplets online, we empirically find that in our case, we lose more time in mining *semi-hard* negatives than we gain by faster convergence, without any significant change in the final result. So, we choose to adopt a strategy where the negatives are picked at random but we make sure that the anchors in a mini-batch have equal contribution from both the classes. An important point to mention here is that although the deep networks are not *directly* being fine-tuned using a triplet loss to give features that improve overall classification accuracy (for all classes), we find the newly learned features to perform well regardless.

Table 1
Performance of the different models on CIFAR-10 dataset. Base model* indicates model after being fine-tuned with triplet loss.

Base model	Method	Accuracy (%)
A	A + Softmax	89.07
	A + DAGSVM	90.76
	A* + DAGSVM	92.50
B	B + Softmax	87.97
	B + DAGSVM	90.17
	B* + DAGSVM	90.95
C	C + Softmax	90.98
	C + DAGSVM	92.52
	C* + DAGSVM	93.69

4. Experimental results

In this section, we present our experimental results on two benchmark datasets to validate the proposed idea. The popularly used CIFAR-10 [13] and STL-10 [3] datasets are used for this purpose. We use different deep learning architectures as our base models to show that our methodology can be applied under different scenarios. Our implementation is based on Torch [5] and linked to the NVIDIA CuDNN libraries to accelerate training. We perform experiments using a single Nvidia Titan Xp GPU and Intel(R) Core(TM) i7-7700K CPU @ 4.20 GHz. For preprocessing, the images in the datasets are converted from RGB to YUV and are mean-std normalized. We also do data cropping and augmentation using horizontal flipping in our experiments.

4.1. CIFAR-10

CIFAR-10 is a dataset with 10 object classes, 50,000 images for training and 10,000 for testing. Each image is an RGB image of size 32×32. We make use of three different architectures as our base models. The first one is similar to the VGG-D network [23] but with 2 fully connected layers in the end having 512 and 10 neurons respectively and batch normalization after every convolutional layer. We refer to this network as 'A'. The second network is similar to the NIN model [16] but with a linear layer at the end along with batch normalization after every convolutional layer. We refer to this network as 'B'. The third is similar to the Resnet-20 model used for training on CIFAR-10 dataset in [11], but with number of filters as {64, 128, 256} for the feature maps of sizes {32, 16, 8} respectively. We refer to this network as 'C'. We train these base models over softmax layer using SGD with a batch size of 128 and a learning rate of 0.1 which decays by a factor of 2 after every 50 epochs for 200 epochs. We use a weight decay of 0.0001 and momentum of 0.9. The respective training curves can be seen in Fig. 3a–c where eventually A achieves a test accuracy of 89.07%, B of 87.97% and C of 90.98%. We then remove the softmax layer and take the learned features from the last layer of these deep architectures to train the DAGSVM, which improves the accuracy significantly (as shown in Table 1). To further improve the performance of the individual binary classifiers in the DAG, we fine-tune these deep architectures specific to each classifier to learn more pairwise discriminative features using a triplet loss with a margin value of

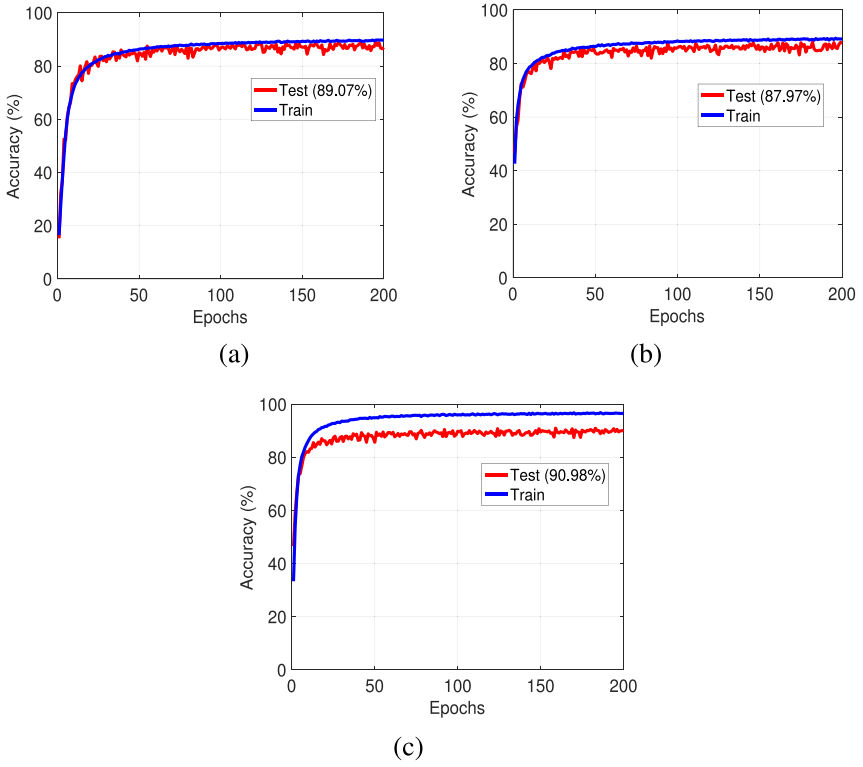

Fig. 3. Training curves corresponding to three different models with softmax on CIFAR-10 dataset. Curve for model A is represented in (a). (b) shows the plot for model B and (c) represents the plot for model C. Train and test accuracies are represented by blue and red curves respectively. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

0.5. We use SGD for fine-tuning the network over 4 k iterations with a batch size of 100 triplets and weight decay and momentum values of 0.0005 and 0.9 respectively. The results can be seen in Table 1. With the improved features, which is our best model, the reduction in overall error is even better.

The test accuracies of the individual binary classifiers before the features are improved using a triplet loss can be seen in Fig. 4 a,b,c. For A and C, apart from 1 classifier, rest all have a test accuracy greater than 96%. And for B, more than 75% of the classifiers have an accuracy greater than 96%. This makes it extremely difficult to further improve their performance. But even then, out of the ${}^{10}C_2$ i.e. 45 classifiers, we are able to improve the accuracies of 40, 32 and 43 classifiers using triplet loss with A, B and C respectively. As can be gleaned from the histograms in Fig. 4a–c the percentage improvement in test accuracies of the individual classifiers is not huge, which is understandable given the high reference scale. But even then the impact on overall classification error reduction is considerable.

4.2. STL-10

STL-10 is also a 10-class image recognition dataset with 5000 images for training and 8000 images for testing. Each image in the dataset is an RGB image of size 96×96. But for the experiments, we resize them to 32×32 so that the scale of depicted objects matches that of CIFAR-10. Overfitting is a significant challenge for this dataset, so we start by showing results on 2 'small' networks and then later move on to a 'large' network to show that our methodology can be applied to different architectures. All networks are trained from scratch. In the following we will refer to the small networks as 'S1', 'S2' and large network as 'L1'. S1 consists of two convolutional layers with 64 and 128 filters followed by two fully-connected layers with 256 and 10 neurons each. S2 consists of four convolutional layers with 64, 128, 256 and 512 filters respectively followed by two fully-connected layers with 512 and 10 neurons each. In both S1 and S2, the convolutional layers are followed by rectification non-linearity (ReLU) layer along with max-pooling layers. Max-pooling is performed over a 2×2 pixel window, with stride 1. L1 is the same as network A used in Section 4.1.

Table 2
Performance of the different models on STL-10 dataset. Base model* indicates model after being fine-tuned with triplet loss.

Base model	Method	Accuracy (%)
S1	S1 + Softmax	60.05
	S1 + DAGSVM	63.33
	S1* + DAGSVM	63.78
S2	S2 + Softmax	60.78
	S2 + DAGSVM	63.83
	S2* + DAGSVM	64.58
L1	L1 + Softmax	66.54
	L1 + DAGSVM	69.16
	L1* + DAGSVM	70.28

We train the base models over softmax layer using the same parameters used in the training for CIFAR-10 in Section 4.1. The respective training curves can be seen in Fig. 6. The networks S1, S2 and L1 eventually achieve an accuracy of 60.05%, 60.78% and 66.54% with softmax layer respectively. We then remove the softmax layer and extract features from the last layer to train the respective DAGSVMs. As can be seen in Table 2, with all three base models, we get around 3% improvement in overall accuracy. We then further improve the performance by fine-tuning the deep networks for each binary classifier. Using the triplet loss, we are able to improve the performance of 20, 35 and 36 classifiers in the DAG using base models S1, S2, and L1 respectively. When comparing our

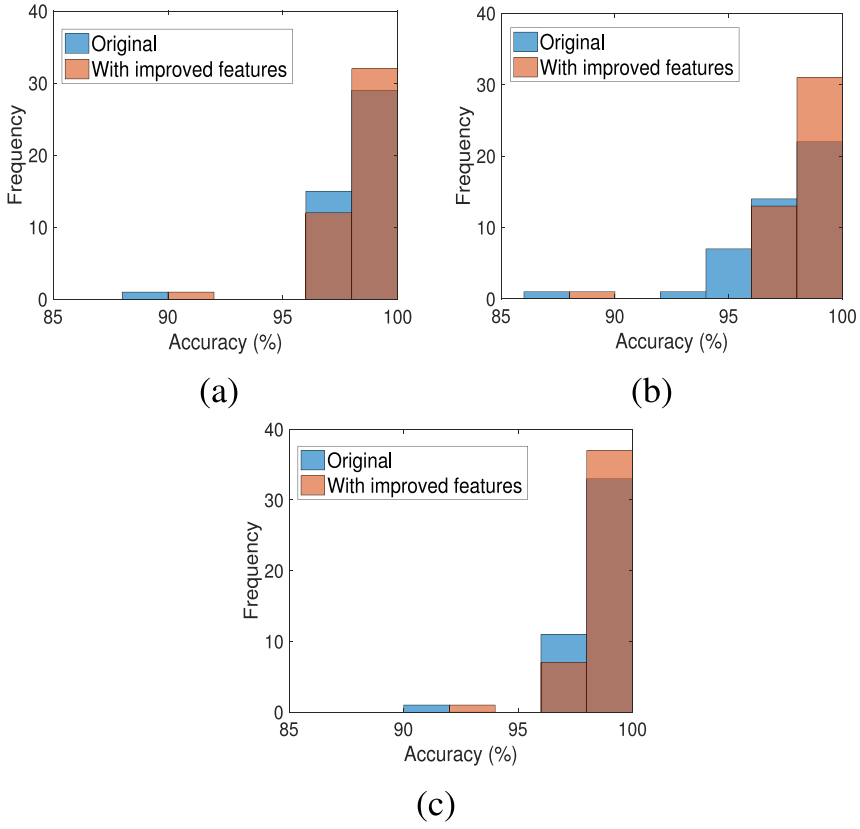

Fig. 4. Histogram plots of test accuracies of the individual binary classifiers in the DAG before (Original) and after fine-tuning the base model A, B and C are shown in (a), (b) and (c) respectively. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

best method with softmax, the improvement in overall accuracy reaches around 4% for all three base models.

4.3. Discussion

An interesting observation is that S1, having the smallest architecture, is only able to improve the performance of 20 classifiers after fine-tuning whereas L1, with the largest architecture, improves the performance of highest number of classifiers (36). A similar trend can also be seen in Section 4.1 when models A, B and C are compared. This fact is also naturally reflected in the percentage improvement of accuracy in Tables 1 and 2. Since we fine-tune over the full network in the experiments, we believe that in a smaller network, which has lesser number of weights to refine, it is difficult to bring about a significant change in the features for improving classifier performance. Besides improving accuracy over softmax layer, our method is also fast during runtime. This is because of the DAG structure introduced by this work, where we extract the features corresponding to the test images prior to classification. For instance, method C + DAGSVM in Table 1 runs at about 1ms per frame. The same runtime applies for C* + DAGSVM as well since we extract the features corresponding to each and every improved deep network prior to classification. On the other hand, C + Softmax runs at about 200 ms per frame, which also includes the time for feature extraction.

In order to explain the behavior of the solution obtained when a multiclass optimization problem is transformed into several binary problems, we also show the relationship between the performance of the binary classifiers and the overall solution in Fig. 5. We show this for one setting where base model A is being fine-tuned using triplet loss to improve the performance of the corresponding binary classifiers, as shown in Table 1. As you can see in Fig. 5, the accuracy for the overall solution monotonically increases as the average accuracy of the binary classifiers increases, which shows that improving the performance of the individual binary classifiers only has a positive impact on the overall solution.

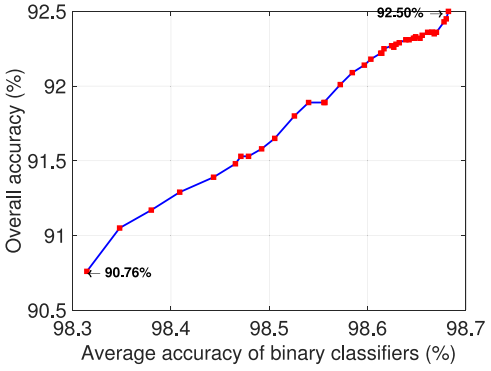

Fig. 5. Relationship between the performance of binary classifiers and overall solution when fine-tuning using triplet loss with base model A. The red points correspond to every 100th iteration. (For interpretation of the references to color in this figure, the reader is referred to the web version of this article.)

Although we deal with datasets having small number of classes in this work, our method can be extended to efficiently handle large number of classes as well. In [7], the authors learn a relaxed hierarchy of binary classifiers organized in a DAG structure to solve the multiclass classification problem and show results on two datasets, Caltech-256 [8] and SUN dataset [29], both having large number of classes. To achieve a good trade-off between speed and

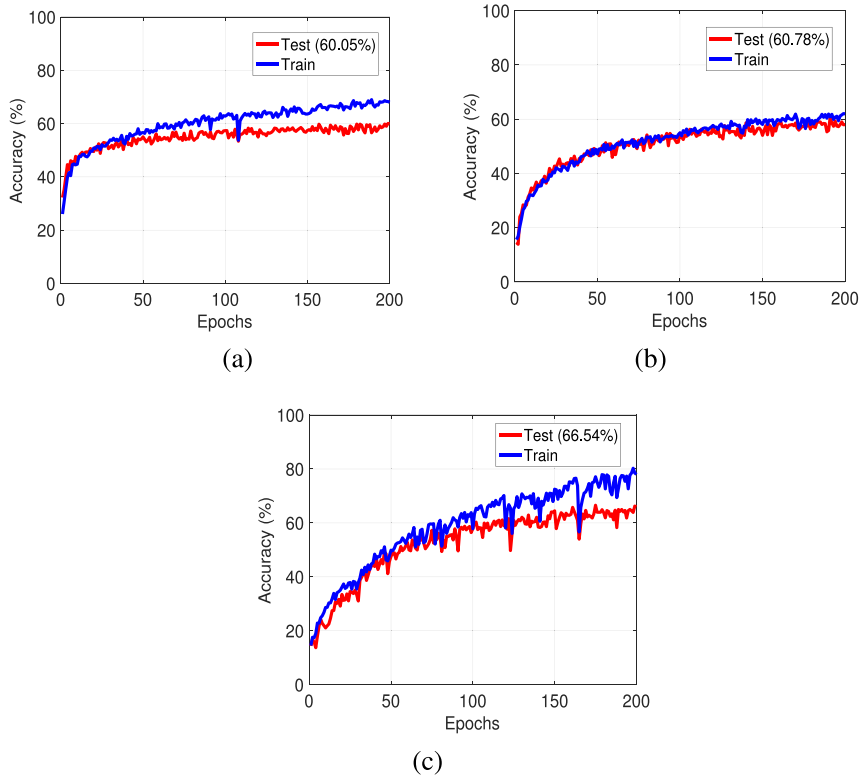

Fig. 6. Training curves corresponding to three different models with softmax on STL-10 dataset. Curve for model S1 is represented in (a). (b) shows the plot for model S2 and (c) represents the plot for model L1. Train and test accuracies are represented by blue and red curves respectively. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

accuracy, they learn two subsets of classes for each binary classifier which can be separated as well as possible, using a unified and principled max-margin optimization. Instead of dealing with only two classes at each node, we can similarly learn such a hierarchy which will significantly reduce the number of classifier evaluations for large number of classes. In addition, since the two subset of classes are learned in a way in which they can be separated as well as possible, we can further use the triplet loss to learn more pairwise discriminative features. Only in this case, the pair would consist of a subset of classes. While this may result in additional time for offline training, the runtime performance will continue to be fast and realtime. As this thread is beyond the scope of this work, we leave it to be explored in future work.

5. Conclusion and future work

In this paper, we present a multiclass classification method to improve upon the classification accuracies of existing deep architectures which use a softmax layer. Primarily, we remove the softmax layer after training the deep network and replace it with a DAGSVM. Such a breakdown enables the network to focus on solving multiple binary problems, which are easier to solve. Such a design also helps to further improve the performance of the individual classifiers by learning more discriminative features through the triplet loss. Experimental results on CIFAR-10 and STL-10 datasets show that our methodology can be applied to different deep architectures and that combining DAGSVM with fine-tuned base model gives best results which is an improvement over the combination of base model with softmax.

One major limitation of our proposed method is that it is not computationally feasible to be applied to problems with large number of classes, since we consider ${}^{N}C_{2}$ binary classifiers in the DAG where N is the number of classes. Future work will deal with proposing an end-to-end framework and learning a hierarchy for the DAG to reduce the number of classifiers in such a way so that a good trade-off is achieved between computational complexity and accuracy.

References

[1] E.L. Allwein, R.E. Schapire, Y. Singer, Reducing multiclass to binary: a unifying approach for margin classifiers, J.Mach.Learn.Res. 1 (December) (2000) 113–141.
[2] S. Bengio, J. Weston, D. Grangier, Label embedding trees for large multi-class tasks, in: Advances in Neural Information Processing Systems, 2010, pp. 163–171.
[3] A. Coates, A. Ng, H. Lee, An analysis of single-layer networks in unsupervised feature learning, in: Proceedings of the fourteenth international conference on artificial intelligence and statistics, 2011, pp. 215–223.
[4] R. Collobert, S. Bengio, A gentle Hessian for efficient gradient descent, in: Acoustics, Speech, and Signal Processing, 2004. Proceedings.(ICASSP'04). IEEE International Conference on, 5, IEEE, 2004, pp. V–517.
[5] R. Collobert, K. Kavukcuoglu, C. Farabet, Torch7: a matlab-like environment for machine learning, BigLearn, NIPS Workshop, EPFL-CONF-192376, 2011.
[6] T.G. Dietterich, G. Bakiri, Solving multiclass learning problems via error-correcting output codes, J.Artif.Intell.Res. 2 (1995) 263–286.
[7] T. Gao, D. Koller, Discriminative learning of relaxed hierarchy for large-scale visual recognition, in: Computer Vision (ICCV), 2011 IEEE International Conference on, IEEE, 2011, pp. 2072–2079.
[8] G. Griffin, A. Holub, P. Perona, Caltech-256 object category dataset. Technical Report 7694, California Institute of Technology, 2007. URL http://authors.library.caltech.edu/7694.
[9] G. Griffin, P. Perona, Learning and using taxonomies for fast visual categorization, in: Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on, IEEE, 2008, pp. 1–8.
[10] R. Hadsell, S. Chopra, Y. LeCun, Dimensionality reduction by learning an invariant mapping, in: Computer vision and pattern recognition, 2006 IEEE computer society conference on, 2, IEEE, 2006, pp. 1735–1742.
[11] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.
[12] F.J. Huang, Y. LeCun, Large-scale learning with SVM and convolutional for generic object categorization, in: Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on, 1, IEEE, 2006, pp. 284–291.

[13] A. Krizhevsky, G. Hinton, Learning multiple layers of features from tiny images. Vol. 1. No. 4. Technical report, University of Toronto, 2009.
[14] A. Krizhevsky, I. Sutskever, G.E. Hinton, ImageNet classification with deep convolutional neural networks, in: Advances in neural information processing systems, 2012, pp. 1097–1105.
[15] L. Li, Multiclass boosting with repartitioning, in: Proceedings of the 23rd international conference on Machine learning, ACM, 2006, pp. 569–576.
[16] M. Lin, Q. Chen, S. Yan, Network in network, arXiv preprint arXiv:1312.4400(2013).
[17] J. Nagi, G.A. Di Caro, A. Giusti, F. Nagi, L.M. Gambardella, Convolutional neural support vector machines: hybrid visual pattern classifiers for multi-robot systems, in: Machine Learning and Applications (ICMLA), 2012 11th International Conference on, 1, IEEE, 2012, pp. 27–32.
[18] J. Ngiam, Z. Chen, D. Chia, P.W. Koh, Q.V. Le, A.Y. Ng, Tiled convolutional neural networks, in: Advances in Neural Information Processing Systems, 2010, pp. 1279–1287.
[19] P. Panda, K. Roy, Attention tree: learning hierarchies of visual features for large-scale image recognition, arXiv preprint arXiv:1608.00611(2016).
[20] J.C. Platt, N. Cristianini, J. Shawe-Taylor, Large margin dags for multiclass classification, in: Proceedings of the 12th International Conference on Neural Information Processing Systems, MIT press, 1999, pp. 547–553.
[21] R.E. Schapire, Using output codes to boost multiclass learning problems, in: ICML, 97, 1997, pp. 313–321.
[22] F. Schroff, D. Kalenichenko, J. Philbin, FaceNet: a unified embedding for face recognition and clustering, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 815–823.
[23] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, arXiv preprint arXiv:1409.1556(2014).
[24] Y. Tang, Deep learning using linear support vector machines, arXiv preprint arXiv:1306.0239(2013).
[25] O. Vinyals, Y. Jia, L. Deng, T. Darrell, Learning with recursive perceptual representations, in: Advances in Neural Information Processing Systems, 2012, pp. 2825–2833.
[26] V. Vural, J.G. Dy, A hierarchical method for multi-class support vector machines, in: Proceedings of the twenty-first international conference on Machine learning, ACM, 2004, p. 105.
[27] K.Q. Weinberger, L.K. Saul, Distance metric learning for large margin nearest neighbor classification, J. Mach. Learn. Res. 10 (February) (2009) 207–244.
[28] J. Weston, F. Ratle, H. Mobahi, R. Collobert, Deep learning via semi-supervised embedding, in: Neural Networks: Tricks of the Trade, Springer, 2012, pp. 639–655.
[29] J. Xiao, J. Hays, K.A. Ehinger, A. Oliva, A. Torralba, Sun database: large-scale scene recognition from abbey to zoo, in: Computer vision and pattern recognition (CVPR), 2010 IEEE conference on, IEEE, 2010, pp. 3485–3492.
[30] S. Zhong, J. Ghosh, Decision boundary focused neural network classifier, in: Intelligent Engineering Systems Through Artifcial Neural Networks, 2000.