# An Empirical Analysis of Boosting Deep Networks

Sai Saketh Rambhatla
*Electrical and Computer Engineering*
*University of Maryland*
College Park, USA
rssaketh@umd.edu

Michael Jones
*Mitsubishi Electric Research Laboratories*
Cambridge, USA
mjones@merl.com

Rama Chellappa
*Electrical and Computer Engineering*
*Johns Hopkins University*
Baltimore, USA
rchella4@jhu.edu

***Abstract*—Boosting is a method for finding a highly accurate hypothesis by linearly combining many "weak" hypotheses, each of which may be only moderately accurate. Thus, boosting is a method for learning an ensemble of classifiers. While boosting has been shown to be very effective for decision trees, its impact on neural networks has not been extensively studied. Using standard object recognition datasets, we verify experimentally the well-known result that a boosted ensemble of decision trees usually generalizes much better on testing data than a single decision tree with the same number of parameters. In contrast, using the same datasets and boosting algorithms, our experiments show the opposite to be true when using neural networks (both convolutional neural networks (CNNs) and multilayer perceptrons (MLPs)). We find that a single neural network usually generalizes better than a boosted ensemble of smaller neural networks with the same total number of parameters.**

***Index Terms*—Neural Networks, Boosting, AdaBoost, Ensembles**

## I. Introduction

We are motivated to study the application of boosting [1], [2] to neural networks (especially convolutional neural networks (CNNs)) because of the great success boosting has had in conjunction with decision trees and other classifiers. Before the recent explosion of research on deep neural networks, boosted decision trees were considered to be state of the art. In fact, the well-known statistician, Leo Breiman, once called boosted decision trees "the best off-the-shelf classifier in the world" [3].

The basic idea of boosting is to form a "strong" classifier (one with high accuracy) using a linear combination (an ensemble) of "weak" classifiers. The only requirement of a weak classifier is that it has accuracy slightly better than chance. AdaBoost [4] is probably the best known boosting algorithm and has been widely used by machine learning researchers. AdaBoost maintains a set of weights per training example and requires that any weak learning algorithm returns a classifier that has a weighted accuracy better than chance on the training examples. On each round of boosting, the weight on each example is updated with a specific equation that gives less weight to examples the weak classifier got right and more weight to examples it got wrong. The next weak classifier will be forced to classify more of the incorrect examples correctly. AdaBoost has been proven to reduce the training error as more weak classifiers are added to the ensemble [4].

Since boosting can be applied to any classifier, it makes sense to try to combine the power of neural networks with the power of boosting. The idea of boosting many small CNNs to create a very accurate classifier while avoiding the trial-and-error search for better network architectures is an exciting possibility. Other researchers have had the same motivation [5]–[8]. In the past, researchers have shown that a boosted ensemble of decision trees or neural networks improves accuracy as the number of decision trees or neural networks increases. An examination of past Kaggle challenge winners shows the success of neural network ensembles. Unlike past work, however, we look at the accuracy of an ensemble of classifiers compared to the accuracy of a single classifier of the same type with the *same number of total parameters*. When comparing classifiers of the same type with different numbers of parameters and trained with different algorithms, it is not possible to know whether any accuracy advantages are due to the difference in number of parameters or the difference in training algorithm. Furthermore, classifiers with more parameters are computationally more expensive so it makes sense to study how to maximize accuracy within a certain parameter budget. In the case of decision trees, we show that a boosted ensemble is usually much more accurate than a single decision tree with the same number of total parameters. Surprisingly, however, the same may not hold for neural networks. The main contribution of this paper is to present empirical evidence that a single large neural network is usually more accurate than a boosted ensemble of neural networks with the same number of total parameters. In terms of accuracy and assuming sufficient training data, it appears that one is better off training a single large network than an ensemble of small networks.

The remainder of the paper is organized as follows. Past work on boosting both using decision trees and neural networks is discussed in Section II. An overview of the basic boosting method is given in Section III as well as details on the specific version of multi-class boosting that we use. In Section IV, we present experiments on boosted decision trees using three well-known object recognition datasets. Analogous experiments are presented in Section V for boosted neural networks. These experiments confirm that while boosting makes sense for decision trees, it may not be a win for neural networks. Finally, in Section VI we give some conclusions and speculate on why boosting is not as effective for neural networks.

## II. Background and Related Work

Boosting was first developed in the early to mid 1990's by Schapire and Freund [1], [2], [4], [9] as a method of creating more accurate classifiers from linear combinations (ensembles) of simpler classifiers. The AdaBoost algorithm [9] was the first practical boosting algorithm to come from this initial work. This early work opened up a rich line of further research that explored the potential of boosting and tried to better understand its success. Some of the key follow-on papers are by Friedman, Hastie and Tibshirani [3], [10] who tried to explain AdaBoost in terms of well-understood techniques in statistics. Friedman proposed gradient boosting [10] as a method of learning a regression function in a stagewise fashion by fitting a function that approximates the residual errors of the training data at each stage and then using a weighted sum of the functions learned at each stage. Mason et al. [11] further showed the connections between boosting and gradient descent function optimization. Friedman et al. [3] derived the LogitBoost algorithm as a way of understanding AdaBoost in terms of additive logistic regression. Another notable paper by Zhu et al. [12] proposed a multi-class boosting algorithm called SAMME (Stagewise Additive Modeling using a Multi-class Exponential loss function) that is similar to AdaBoost but differs in how it computes the weight of each weak classifier and how it updates the example weights in each round of boosting.

In the early work on boosting, decision trees were typically used as weak classifiers despite the fact that boosting applies to any type of classifier. Some examples of early work on boosting decision trees include [9], [12]–[14]. These and many more works demonstrated the interesting result that larger and larger ensembles with growing numbers of parameters usually did not overfit the training data. In other words, testing error continued to decrease along with training error as the number of weak classifiers increased. In contrast, it was well known that as the size (number of parameters) of a single decision tree increased, it would eventually overfit, i.e. the testing error would eventually increase as the training error decreased to zero [15]. Combating this problem is a major reason why boosted decision trees are so useful - they allow the training of classifiers with large numbers of parameters, usually without overfitting.

There have also been many papers on applying boosting to neural networks. Early papers applied boosting to multilayer perceptrons (MLPs) such as [5], [16]–[18] and showed that boosted ensembles of MLPs got significantly lower errors rates than a single MLP on problems such as character recognition and the UCI classification datasets. More recently, boosting has been applied to deep convolutional neural networks [6]–[8], [19], [20]. Many of these papers have proposed different boosting algorithms that are tailored for CNNs, but they all result in a linear combination of CNNs. For example, Moghimi et al. [8] derived a boosting algorithm for CNNs for multi-class classification problems in which the CNN weak learner was trained to estimate the example weights since this was shown to be equivalent to taking a step in the direction of the negative gradient of the error. Mosca and Magoulas [20] use the multiclass AdaBoost.M2 algorithm of Freund and Schapire [9] but initialize each subsequent CNN weak classifier using the weights learned in the previous boosting round which leads to both higher accuracy and faster network training times.

Recent advances in boosting are mainly focused on very fast and efficient implementations of gradient boosting, for example XGBoost [21] and CatBoost [22] are popular gradient boosting packages. For a more comprehensive survey of various boosting algorithms, please see [23], [24].

Past work on boosting neural networks has not analyzed whether an ensemble of MLPs or CNNs is a "win" in terms of decreasing the testing error below what is achievable with a single network with the same number of total parameters as in an ensemble. Unlike decision trees which are very susceptible to overfitting, this is less of a problem with highly over-parameterized CNNs [25], [26]. Typically, training CNNs with more parameters does not result in worse generalization error. Although it is certainly true that CNNs can overfit especially when trained on very small datasets (such as hundreds of examples), one of the reasons they have revolutionized machine learning is because they usually generalize well even when only trained on thousands of examples with networks containing millions of parameters. While this phenomenon may be counter-intuitive, there is a growing body of research attempting to explain this surprising behavior [25]–[28]. Despite our incomplete understanding, the fact remains that networks with more parameters tend to generalize better (see Figures 3, 5, 6, 7, for example).

So the incentive to use boosting to avoid overfitting may not be so important for neural networks. Then the question becomes whether boosted ensembles of neural networks can achieve better accuracy than simply using standard deep learning methods on a single, large neural network. This is the main question we will address here.

## III. Boosting/AdaBoost

Boosting is a meta-learning algorithm that uses a base learning algorithm (also called a weak learner) to build an ensemble of (weak) classifiers such that the ensemble classifier (also called a strong classifier) is more accurate than any of the weak classifiers. The main idea is to assign each training example a weight and use the weak learner to return a weak classifier that minimizes the *weighted* error. The example weights are initially uniform but are updated on each round of boosting so as to give more weight to examples that the previous weak classifier got wrong and less weight to examples that it gets right. This forces the next weak classifier to get more of the examples right that the previous weak classifier got wrong. The final strong classifier gives more weight to the more accurate weak classifiers. Many versions of this basic boosting method have been published with the main differences being how the margin for each example is computed and the weight assigned to each weak classifier in the ensemble.

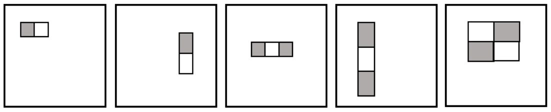


Fig. 1: Examples of 2, 3 and 4 rectangle Haar-like filters. The sum of pixel values in shaded rectangles are subtracted from the sum of pixel values in white rectangles. Each Haar-like filter has two versions: one with absolute value and one without. A Haar-like filter can be applied to any color channel in the case of color input images (CIFAR-10 and CIFAR-100).

**Algorithm 1** A Multi-class Version of AdaBoost

```
1: Given: $(x_1, y_1), (x_2, y_2), \cdots, (x_m, y_m)$; $x_i \in \mathcal{X}$, $y_i \in \mathcal{Y}, \mathcal{Y} = \{1, ..., C\}$ where $C$ is the number of classes
2: Initialize $D_1(i) = \frac{1}{m}$
3: for $t = 1, 2, \cdots, T$ do
4:     Train weak classifier using distribution $D_t$ which yields weak classifier $\mathbf{h_t} : \mathcal{X} \to \mathbb{R}^C$ where $\mathbb{R}^C$ is a vector of $C$ reals representing a probability distribution over $C$ classes
5:     Compute margins, $m_i$, for each training example $(x_i, y_i)$ according to equation 1
6:     Choose $\alpha_t \in \mathbb{R}$
7:     Update:
           $D_{t+1}(i) = \frac{D_t(i)e^{-\alpha_t m_i}}{Z_t}$
       where $Z_t$ is a normalization factor (chosen to make $D_{t+1}$ a distribution).
8: end for
9: Output the final hypothesis
       $H(x) = \text{argmax}(\sum_{t=1}^{T} \alpha_t \mathbf{h_t}(x))$
```

In our experiments, we tested different multi-class versions of boosting, which all worked similarly, but we found the following version (Algorithm 1) which is slightly modified from the AdaBoost.M1 algorithm of Freund and Schapire [4] to perform best. We follow the notation used in the generalized version of AdaBoost described in Schapire and Singer ( [14]).

In our multi-class version, weak classifiers output a probability distribution over output classes represented as a $C$ element real-valued vector. On each round, $t$, of boosting, a weak classifier is found that minimizes a weighted error over examples based on the current weights of examples, $D_t$. In our case, the weak learning algorithm is either the decision tree training algorithm or a neural network training algorithm. For training neural networks, we sample batches according to the distribution $D_t$ so that the neural network training algorithm does not have to use example weights directly.

Once a weak classifier is trained, we compute the margin of each example which is essentially how correct the weak classifier is on each example. We use a margin, $\mathcal{M}$, similar to the one defined in [29], which is equal to the probability of the correct class minus the average probability of the other classes on the training examples.

$$m_i = \mathcal{M}(\mathbf{h_t}(x_i), y_i) = \mathbf{p_i}[y_i] - \frac{1}{C-1}(1 - \mathbf{p_i}[y_i]) \tag{1}$$

where $y_i$ is the correct label for input sample $x_i$, $C$ is the number of classes and $\mathbf{h_t}(x_i) = \mathbf{p_i} = [p_1, p_2, \cdots p_C] \in \mathbb{R}^C$ is the probability over the classes output by the weak classifier for input $x_i$.

In step 6 of the algorithm, a weight $\alpha_t$ is chosen as the weight of weak classifier $\mathbf{h_t}$. In [14], theoretical justification is given for choosing $\alpha_t = \frac{1}{2}\log(\frac{1+r_t}{1-r_t})$ where $r_t$ is the weighted average of the margin computed over the training set. In our experiments, we found that the sigmoid function, $\alpha_t = \frac{1}{1+e^{-r_t}}$, worked better for both neural networks and decision trees.

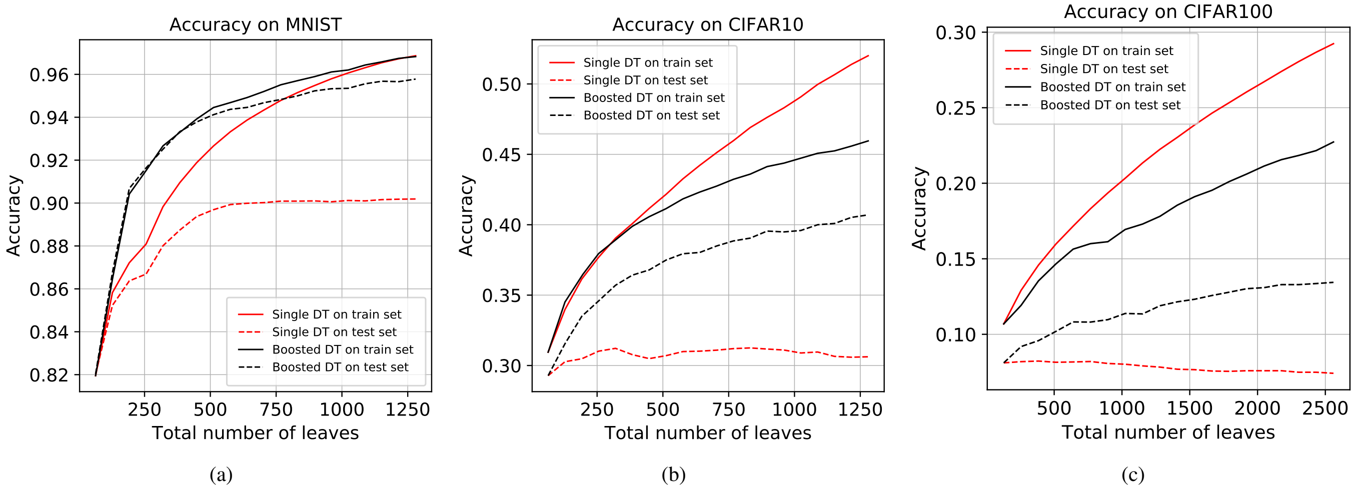
Finally, the weights on the examples are updated according to the negative exponential of the margin for each example. The whole process is repeated to train the next weak classifier.

## IV. Decision Tree Experiments

In this section we present some experiments on image classification datasets comparing boosted ensembles of decision trees to single decision trees with the same number of total leaves. There have been very many papers published that plot the error rate of a boosted ensemble of decision trees as a function of the number of weak classifiers. The vast majority of these plots show the error rate continuing to decrease as more weak classifiers are added to the ensemble. We also examine the error rates of single decision trees as the number of leaves increases. A comparison of the error rates of boosted decision tree ensembles and single decision trees with the same number of leaves is interesting here because it confirms that single decision trees often overfit as the number of leaves increases while boosted decision tree ensembles often do not, and as a consequence that boosted ensembles generalize much better than single decision trees. We will show in Section V that boosted neural network ensembles exhibit very different behavior. We use binary decision trees with multi-class outputs. The output is a probability distribution over all possible output classes. Each node computes a Haar-like filter on the input image and thresholds the filter value to determine whether the input goes to the left or right child node. When a leaf node is reached, the output class is the class with the highest probability. The probability distribution for any leaf node is computed from the training examples that fall into that leaf as the count of each class divided by the number of training examples in the leaf.

We use a set of Haar-like filters that are very similar to the Haar-like filters used in [30]. We use 2-rectangle, 3-rectangle and 4-rectangle Haar-like filters as illustrated in Figure 1. The value of a Haar-like filter applied to an image is the sum of the pixel values (in a single color channel) in the white rectangles minus the sum of the pixel values in the gray rectangles.

For the CIFAR-10 and CIFAR-100 datasets in which each example is 32x32 pixels, we used 2912 Haar-like filters sampled from all possible 2-rectangle, 3-rectangle and 4-rectangle filters that can fit within a 32x32 pixel image. The



(a)

(b)

(c)

Fig. 2: Classification accuracy on MNIST [2a], CIFAR-10 [2b] and CIFAR-100 [2c] training and testing sets for single decision tree and boosted decision tree versus total number of leaves in the tree/ensemble.

minimum rectangle size within any filter was 5 pixels. Because a filter can be applied to any of the three color channels independently, there were effectively $3 \times 2912 = 8736$ filters used for training.

For the MNIST dataset with 28x28 pixel examples, we used 3512 Haar-like filters with a minimum size of 4 pixels for any single rectangle within a filter. There is only a single gray-level channel for MNIST.

We use the following decision tree learning algorithm to build a tree with a fixed number of leaf nodes. Initially, a leaf node is created containing all of the training examples. On each iteration of the learning algorithm, the leaf node with the minimum ”peak” is chosen to split where ”peak” is defined as the probability of the highest probability class minus the average probability of all other classes. The peak value is similar to negative entropy. To split a node, all Haar-like filters in the given filter set are used to find the optimal filter and threshold that splits the examples in the node so that the sum of the peaks for the examples going into the children nodes is maximized. The Haar-like filter and threshold that maximize the sum of peaks of the children nodes is chosen as the decision function for the node. Splitting of nodes continues until the desired number of leaves is reached.

We use the multi-class boosting algorithm described in Section III. We conducted experiments on three standard image classification datasets: MNIST, CIFAR-10 and CIFAR-100.

**MNIST**: The MNIST dataset [31] is a handwritten digit (from 0 to 9) recognition task consisting of $28 \times 28$ pixel gray scale images. The dataset contains 60,000 training images (6000 images of each digit) and 10,000 testing images (1000 images of each digit).

**CIFAR-10**: The CIFAR-10 dataset [32] represents an object recognition task. It contains 60,000 $32 \times 32$ color images with each image containing one of 10 different classes. The 10 different classes represent airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. There are 6,000 images of each class from which 5,000 are used for training and 1,000 are used for testing.

**CIFAR-100**: This dataset [32] represents a similar object recognition task as the CIFAR-10 dataset, except there are 100 object classes instead of 10. Each class contains 600 color images. Each image is $32 \times 32 \times 3$, and the 600 images are divided into 500 training, and 100 test for each class.

For each dataset, we trained boosted decision trees for 20 rounds with a fixed number of leaves (64 for MNIST and CIFAR-10, 128 for CIFAR-100) in each weak classifier. We also trained a series of single decision trees with the same number of total leaves as in each ensemble. To train a single decision tree with 128 leaves, for example, we start with one already trained with 64 leaves and continue the decision tree training algorithm until 128 leaves are reached. Since there is no randomness in the training algorithm, each experiment is run only once. (Rerunning the same experiment would yield the exact same classifier.)

In each of the Figures 2a-2c, the red curves are for single decision trees and the black curves are for boosted decision trees. The solid curves show accuracy on the training set and the dashed curves show accuracy on the testing set.

Figure 2a shows that on the MNIST dataset, the boosted decision trees are significantly more accurate on the testing set than the single decision trees with the same number of leaves. On the training set, the boosted and single trees are more comparable, with the boosted trees being a little more accurate until the number of leaves becomes greater than 1000 or so.

On CIFAR-10 (Figure 2b) and CIFAR-100 (Figure 2c), we again see that the boosted decision trees are significantly more accurate on the testing sets, but not on the training sets as the number of leaves grows. The fact that the single decision trees continue to improve accuracy on the training set while worsening accuracy on the testing sets is clear evidence of overfitting. The boosted decision trees, on the other hand, avoid overfitting as both training and testing accuracy continue to increase with increasing numbers decision trees.

These experiments confirm that boosted decision trees usually generalize better than single decision trees with the same number of total leaves. They also confirm that training single large decision trees is prone to overfitting while boosted ensembles of decision trees are resistant to overfitting.

## V. Neural Network Experiments

In this section we explore empirically whether boosted neural networks are similar to boosted decision trees and result in better accuracy than simply using a single network with an equivalent number of parameters. We experiment with three network architectures described below and three datasets. As with the experiments using decision trees, we use the CIFAR-10 and CIFAR-100 object recognition datasets. Because even the small base networks we use achieve nearly 100% training accuracy on MNIST, we instead use the Street View House Numbers (SVHN) dataset for our third dataset. SVHN [33] is a real-world image dataset obtained from house numbers in Google Street View images. The dataset contains over $600,000$ training images, and about $20,000$ test images, each of size $32 \times 32$ pixels.

**CNN**: The first architecture is a CNN base classifier. We adopt a similar architecture as LeNet [31], as a base learner, for our experiments. Boosting such a model for $N$ rounds makes the total number of parameters $N$ times the number of parameters of the base classifier. For training a single model with an equal number of parameters, we increase the number of filters in each hidden layer until the total number of parameters is less than or equal to the required number. To increase the number of parameters in the CNN for creating larger single networks, we add convolutions (width) to each existing layer but do not add new layers (depth).

For a single CNN for which we want $n$ times more parameters than the base CNN, one way to increase the number of parameters, for a network with odd number of layers (which is true in our case), is to multiply the number of convolutional filters in every other layer by $n$. This increases the number of parameters by exactly $n$. However for higher values of $n$, this method results in a tight bottleneck in the intermediate layers. To alleviate this issue, we use this method only for smaller values of $n$ ($n \in \{1, \cdots, 5\}$) and for larger values of $n$, we multiply the number of filters in each layer of the base CNN by $\sqrt{n}$ (rounding to the nearest integer). Multiplying the number of filters in layers $L$ and $L+1$ by $\sqrt{n}$, multiplies the number of *parameters* in layer $L+1$ by $n$ (except for the first and last layers). Since the number of parameters in the first and last layers only increases by a factor of $\sqrt{n}$, we multiply the number of neurons in the penultimate layer by a factor $c\sqrt{n}$ to make the total number of parameters about the same as in the boosted counterpart. We use $c = 1.35$ in this work for CNN base classifiers. It is likely that the architectures could be optimized to produce even higher accuracy in our experiments, but this is not the point of the current work.

| SVHN & CIFAR-10 | | | CIFAR-100 | | |
|---|---|---|---|---|---|
| Ensemble | Single CNN | | Ensemble | Single CNN | |
| # params. | hid dims | # params | # params | hid dims | # params |
| 5954 | $[6, 16, 32]$ | 5954 | 8834 | $[6, 16, 32]$ | 8834 |
| 11908 | $[12, 16, 64]$ | 11908 | 17668 | $[12, 16, 64]$ | 17668 |
| 17862 | $[18, 16, 96]$ | 17862 | 26502 | $[18, 16, 96]$ | 26502 |
| 23816 | $[24, 16, 128]$ | 23816 | 35336 | $[24, 16, 128]$ | 35336 |
| 29770 | $[30, 16, 160]$ | 29770 | 44170 | $[30, 16, 160]$ | 44170 |
| 35724 | $[14, 39, 82]$ | 34894 | 53004 | $[14, 39, 105]$ | 52647 |
| 41678 | $[15, 42, 88]$ | 40219 | 61838 | $[15, 42, 114]$ | 60567 |
| 47632 | $[16, 45, 95]$ | 46337 | 70672 | $[16, 45, 122]$ | 68522 |
| 53586 | $[18, 48, 100]$ | 52462 | 79506 | $[18, 48, 129]$ | 76890 |
| 59540 | $[18, 50, 106]$ | 57346 | 88340 | $[18, 50, 136]$ | 83386 |

TABLE I: Table enumerating hidden dimensions and number of trainable parameters of the ensemble and single CNN classifier on CIFAR-10/SVHN and CIFAR-100.

| SVHN & CIFAR-10 | | | CIFAR-100 | | |
|---|---|---|---|---|---|
| Ensemble | Single MLP | | Ensemble | Single MLP | |
| # params. | hid dims | # params | # params | hid dims | # params |
| 410880 | $[128, 128]$ | 410880 | 422400 | $[128, 128]$ | 422400 |
| 821760 | $[246, 246]$ | 818688 | 844800 | $[247, 247]$ | 844493 |
| 1232640 | $[358, 358]$ | 1231520 | 1267200 | $[358, 358]$ | 1263740 |
| 1643520 | $[463, 463]$ | 1641335 | 1689600 | $[464, 464]$ | 1687104 |
| 2054400 | $[563, 563]$ | 2052135 | 2112000 | $[565, 565]$ | 2111405 |
| 2465280 | $[658, 658]$ | 2460920 | 2534400 | $[661, 661]$ | 2533613 |
| 2876160 | $[750, 750]$ | 2874000 | 2956800 | $[753, 753]$ | 2955525 |
| 3287040 | $[838, 838]$ | 3284960 | 3379200 | $[841, 841]$ | 3374933 |
| 3697920 | $[923, 923]$ | 3696615 | 3801600 | $[927, 927]$ | 3799773 |
| 4108800 | $[1005, 1005]$ | 4107435 | 4224000 | $[1010, 1010]$ | 4223820 |

TABLE II: Table enumerating hidden dimensions and number of trainable parameters of the ensemble and single MLP classifier on CIFAR-10/SVHN and CIFAR-100.

The CNN base classifier has $5954$ trainable parameters for CIFAR-10, SVHN ($c = 1.05$) and $8834$ trainable parameters for CIFAR-100 ($c = 1.35$). Refer to Table I for exact details of the architecture.

**MLP**: We adopt a four-layer MLP with two hidden layers. To increase the number of parameters in the MLP to approximately match the number of parameters in each boosted ensemble of MLPs, we assume the same number, $n$, of neurons in both the MLP hidden layers and solve for $n$ analytically. The total number of parameters in the MLP is $p(n) = 3072 * n + n^2 + Cn = n^2 + (3072 + C) * n$, where $C$ is the number of output neurons. We do not consider the bias terms in our experiments. To obtain a network with $N$ parameters, we solve the quadratic equation $p(n) = N$ and round the solution to the lowest integer. Each MLP base classifier has $410880$ trainable parameters for CIFAR-10 and SVHN and $422400$ trainable parameters for CIFAR-100 (due to more output classes). Refer to Table II for exact details of the architecture and number of trainable parameters.

**VGG-8**: To demonstrate results using a deeper architecture, we adopt a VGG [34] style architecture as the base learner. We follow a similar strategy to increase the number of parameters as with the CNN mentioned above. VGG-8 classifier has $87234$ and $98844$ trainable parameters for CIFAR-10 ($c = 1.48$) and CIFAR-100 ($c = 1.70$), respectively. We show the exact details of the architecture in Table III.

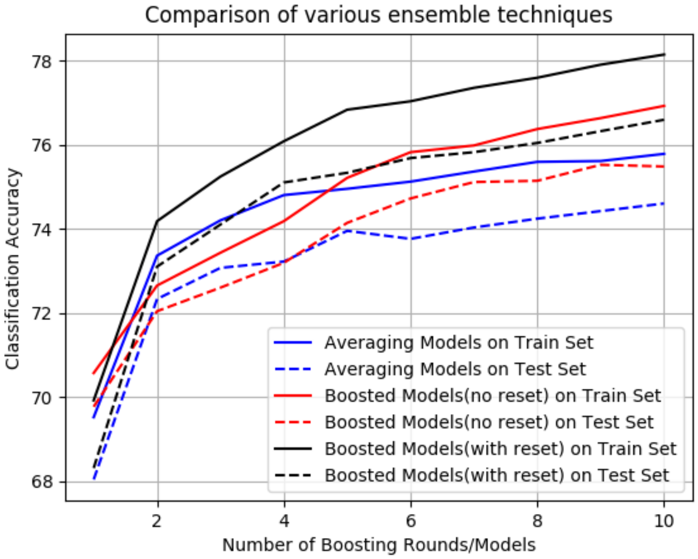


Fig. 3: Comparison of averaging and boosting with/without reset of a CNN base classifier on CIFAR-10 train & test sets

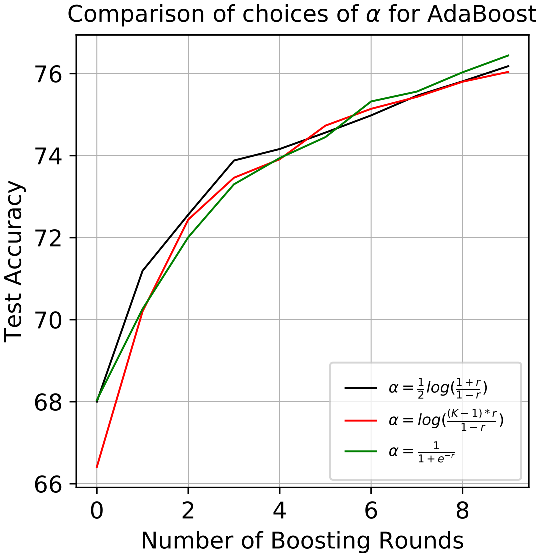


Fig. 4: Comparison of different choices of $\alpha$ for AdaBoost on CIFAR-10 test set.

| CIFAR-10 | | | CIFAR-100 | | |
|---|---|---|---|---|---|
| Ensemble | Single VGG-8 | | Ensemble | Single VGG-8 | |
| # params. | hid dims | # params | # params | hid dims | # params |
| 87234 | $[6, 16, 32, 64, 64]$ | 87234 | 98844 | $[6, 16, 32, 64, 64]$ | 98844 |
| 174648 | $[8, 22, 45, 90, 133]$ | 190142 | 197688 | $[8, 22, 45, 90, 153]$ | 220532 |
| 261972 | $[10, 27, 55, 110, 164]$ | 272163 | 296532 | $[10, 27, 55, 110, 188]$ | 310629 |
| 349296 | $[12, 32, 64, 128, 189]$ | 356215 | 395376 | $[12, 32, 64, 128, 217]$ | 403693 |
| 436620 | $[13, 35, 71, 143, 211]$ | 435156 | 494220 | $[13, 35, 71, 143, 243]$ | 492078 |
| 523944 | $[14, 39, 78, 156, 232]$ | 516055 | 593064 | $[14, 39, 78, 156, 266]$ | 579787 |
| 611268 | $[15, 42, 84, 169, 250]$ | 596331 | 691908 | $[15, 42, 84, 169, 287]$ | 668991 |
| 698592 | $[16, 45, 90, 181, 267]$ | 677620 | 790752 | $[16, 45, 90, 181, 307]$ | 759550 |
| 785916 | $[18, 48, 96, 192, 284]$ | 761294 | 889596 | $[18, 48, 96, 192, 326]$ | 850898 |
| 873240 | $[18, 50, 101, 202, 299]$ | 838108 | 988440 | $[18, 50, 101, 202, 344]$ | 937333 |

TABLE III: Table enumerating hidden dimensions and number of trainable parameters of the ensemble and single VGG-8 classifier on CIFAR-10/SVHN and CIFAR-100.

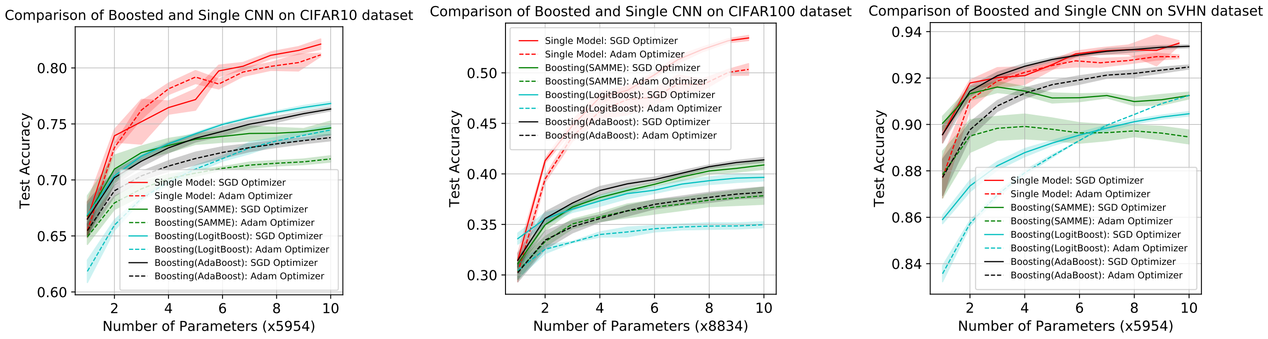
**Network Training Parameters**: For all experiments using the CNN base classifier, we use a batch size of 128 with an initial learning rate of 0.1 (0.0001) with SGD (ADAM) optimizer trained for 300 epochs. The learning rate is decreased by a factor of ten after 95 epochs for each optimizer. We do not perform any hyperparameter optimization. We boost the base classifier for ten rounds. We report results using two optimizers (SGD and Adam) for the CNN classifier and only SGD (which yields better accuracy) for MLP and VGG networks. The MLP classifiers are trained with a batch size of 128 for 300 epochs with an initial learning rate of 0.01 using an SGD optimizer. The learning rate is decreased by a factor of 10 after 95 epochs. VGG-8 classifiers are trained for 200 epochs with a batch size of 256 using an SGD optimizer. We begin training with a learning rate of 0.1 and decrease by a factor of 5 after 60, 120 and 160 epochs. We repeat all experiments in Figure 5, 6 and 7 for 5 rounds and plot the mean and standard deviation.

**Boosting vs Averaging Models:** We start by comparing training and testing set performance of averaging and boosting with/without reset on the CIFAR-10 dataset. Boosting with reset, similar to the one in [8], means initializing the models in every round of boosting from scratch. Boosting without reset initializes the model with the weights of the first model. From Figure 3, it is clear that boosting with reset is most accurate, and that boosting offers improvement over naive averaging of models which is often used as a quick and simple way of improving accuracy.

**Choice of $\alpha$:** [14] prove that choosing $\alpha = \frac{1}{2}\ln(\frac{1+r_t}{1-r_t})$ in step 6 of Algorithm 1 minimizes the training error. However, in our experiments, we observe empirically that a different choice of $\alpha$ works better for neural networks. In Fig. 4, we compare the CIFAR-10 test set performance of a CNN base classifier boosted using three choices of $\alpha$ for 10 rounds using the multi-class Adaboost described in Algorithm 1. We experiment with 1) $\alpha = \frac{1}{2}\log(\frac{1+r_t}{1-r_t})$ originally proposed in [14], 2) $\alpha = \log(\frac{(K-1)*r_t}{1-r_t})$ ($K$ is the number of classes) as proposed in [35] for another multi-class boosting algorithm SAMME and 3) $\alpha = \frac{1}{1+e^{-r_t}}$. From the figure, choosing $\alpha = \frac{1}{1+e^{-r_t}}$ performs better than $\frac{1}{2}\log(\frac{1+r_t}{1-r_t})$ for higher rounds of boosting. Unless otherwise mentioned, we use $\frac{1}{1+e^{-r_t}}$ for AdaBoost for the rest of the experiments.

### A. Main Experiments

**Boosting Algorithms:** In this section, we experiment with three boosting algorithms namely AdaBoost, SAMME [35] and LogitBoost [3]. While we re-implement the exact algorithm of SAMME [35] we had to make a few modifications to the LogitBoost algorithm [3] to work with a Neural Network base classifier. LogitBoost [3] requires training a regression model. We use the Mean Squared Error (MSE) loss for CIFAR-10 and SVHN datasets. However, for more than 10 classes, in the case of CIFAR-100, this training loss didn't converge. For CIFAR-100, we train a classification network using soft labels instead of one-hot-encoded targets. This can be achieved by minimizing the KL-Divergence between the soft labels and the network outputs. Additionally, for rounds



(a)

(b)

(c)

Fig. 5: Classification accuracy on CIFAR-10 [5a], CIFAR-100 [5b] and SVHN [5c] testing sets for single and boosted CNN classifiers versus number of parameters.

greater than 1, we clamp the output of the networks to be no greater than 3.

*1) CNN experiments:* Figure 5 shows results comparing a series of single CNNs with their Adaboost, SAMME and LogitBoost ensemble counterparts on CIFAR-10, CIFAR-100 and SVHN using the SGD and ADAM optimizers. The solid lines are results with the SGD optimizer while the dotted lines are for ADAM. Red lines show results for a single network and black, green and cyan lines depict results of boosting using AdaBoost, SAMME and LogitBoost algorithms respectively. Using AdaBoost, a single CNN classifier (with equal number of parameters) outperforms the boosted ensemble after 10 rounds of boosting by 5.8, 12 and 0.1 percentage points on CIFAR-10, CIFAR-100 and SVHN, respectively, using an SGD optimizer. Using ADAM, the single classifier is better than the boosted ensemble by 7.5, 14 and 0.5 percentage points on CIFAR-10, CIFAR-100 and SVHN, respectively.

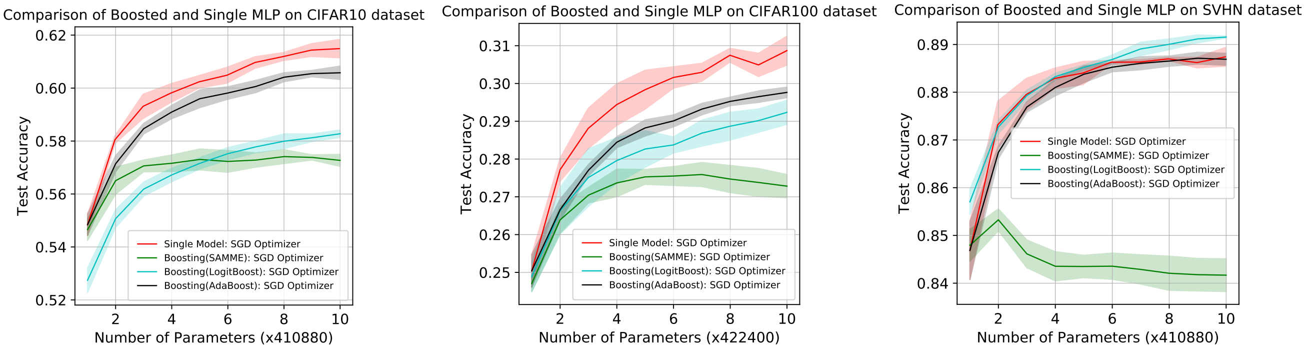
Using SAMME, a single CNN classifier (with equal number of parameters) outperforms the boosted ensemble after 10 rounds of boosting by 7.49 and 12.57 percentage points on CIFAR-10 and CIFAR-100 respectively using an SGD optimizer. Using ADAM, the single classifier is better than the boosted ensemble by 9.28 and 12.50 percentage points on CIFAR-10 and CIFAR-100 respectively. On the SVHN dataset, the SAMME algorithm fails after two rounds because neural network training fails to find a classifier with weighted error better than chance. We think this is due to the nature of the discrete margin computed in SAMME which puts almost all of the weight on the most difficult examples.

Using LogitBoost, a single CNN classifier (with equal number of parameters) outperforms the boosted ensemble after 10 rounds of boosting by 5.30, 13.79 and 3.04 percentage points on CIFAR-10, CIFAR-100 and SVHN, respectively, using an SGD optimizer. Using ADAM, the single classifier is better than the boosted ensemble by 6.70, 15.37 and 1.66 percentage points on CIFAR-10, CIFAR-100 and SVHN, respectively.

*2) MLP experiments:* Figure 6 shows results of single MLPs and their boosted counterparts, trained using Adaboost, SAMME and LogitBoost, on each of the three test sets. The single networks are more accurate than the equivalent boosted ensemble of networks for almost all values of the number of parameters. After 10 rounds of boosting, using Adaboost, the equivalent single networks are about 1%, 1.2% and 0.2% more accurate on CIFAR-10, CIFAR-100 and SVHN, respectively. Using SAMME, the single networks are about 4.22% and 3.59% more accurate than the boosted counterparts on CIFAR-10 and CIFAR-100 test sets respectively. Similar to the CNN architecture, the boosted MLP ensemble failed to converge on SVHN. Finally, using LogitBoost, the single networks are about 3.22% and 1.64% more accurate than the boosted counterparts on CIFAR-10 and CIFAR-100 test sets respectively. On SVHN, we observe that the ensemble trained using LogitBoost outperforms the single MLP architecture by 0.46%. This is the only case out of 3 architectures and 3 datasets where a boosted ensemble outperformed a single neural network. We believe this is the case because the inductive bias of MLPs is not as suitable for structured data like images resulting in lower performance than the boosted ensembles. This can also be seen in the comparison of MLP and CNN single models on SVHN (88.74% vs 93.50%).

**VGG experiments:** Figure 7 shows results comparing single VGG-8 classifiers with the boosted ensembles on CIFAR-10 and CIFAR-100. We do not show results for VGG-8 on SVHN as the single base classifier attains 100% training accuracy, leaving little room for improvement for either boosting or larger networks. It is clear from the plots that the single models outperform the boosted ensembles, trained using Adaboost, by a significant margin (4 and 12 percentage points after 10 boosting rounds on CIFAR-10 and CIFAR-100, respectively). Using LogitBoost, the single models outperform the boosted ensemble by 4.83% and 16.14% points on CIFAR-10 and CIFAR-100 test sets. We observe a similar pattern with the boosted ensembles trained using SAMME algorithm as with the other two architectures above.

**Boosting rounds:** Finally, to verify that boosting a larger ensemble would not change the results, we boost a CNN base classifier for 50 rounds (297700 parameters) on CIFAR-10 and compare it with a single VGG-8 classifier enlarged to 297680 parameters. The boosted ensemble achieves 77.00% accuracy

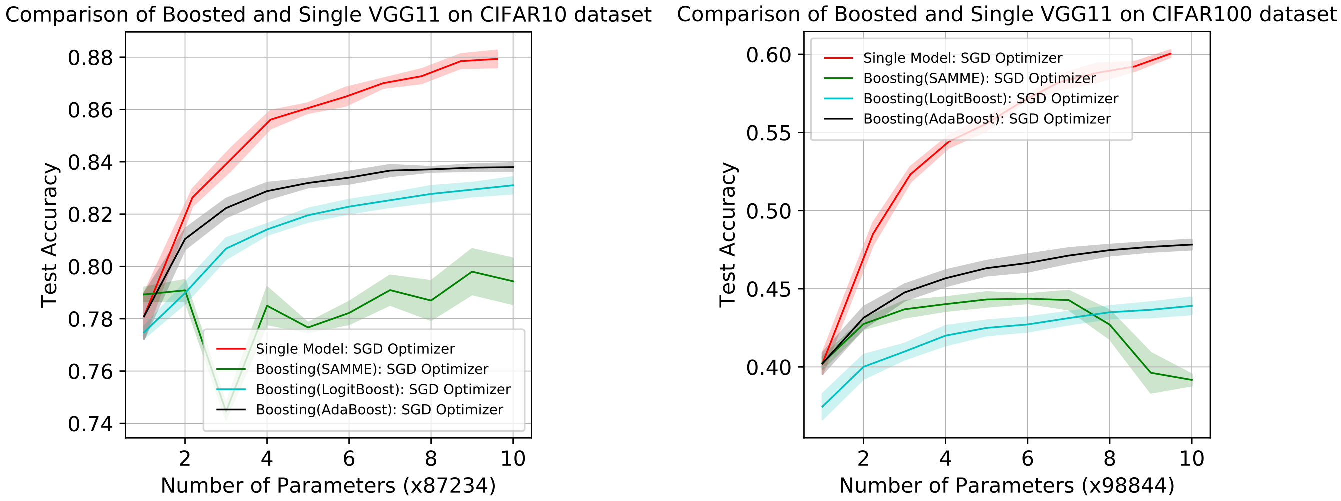


(a)

(b)

(c)

Fig. 6: Classification accuracy on CIFAR-10 [6a], CIFAR-100 [6b] and SVHN [6c] testing sets for a single and boosted MLP classifiers versus number of parameters.



(a)

(b)

Fig. 7: Classification accuracy on CIFAR-10 [7a] and CIFAR-100 [7b] testing sets for a VGG-8 single and boosted classifiers versus number of parameters.

on the test set while the single network achieves $85.81\%$. We compare an ensemble of CNN classifiers to a single VGG-8 network because of the difficulty of increasing the number of parameters in the CNN base classifier by 50 times to create a single network.

**Final thoughts:** These experiments show that boosting neural networks, unlike boosting decision trees, does not lead to better accuracy compared to simply using a larger neural network. One caveat to this statement that "bigger is better" for neural networks is that training set size and difficulty also matter. In cases where a single neural network is able to achieve near 100% accuracy on the training set, we observed that adding more parameters to the single network can lead to either overfitting or test error just not improving as much as it does using boosting. In these cases, boosting is not a big improvement over training a single larger network, but neither is it true that a single large network is an improvement over a boosted ensemble.

## VI. Discussion and Conclusions

Through empirical evidence, we have demonstrated some limits of boosted ensembles of neural networks. In particular, this research suggests that larger networks often yield higher accuracy than ensembles of smaller networks with the same total number of parameters. This does not imply that boosting does not work with neural nets. It does. Boosted ensembles of neural nets have higher accuracy than any single neural net in the ensemble. However, it is likely that a single, large neural net would have even greater accuracy.

An important reason for why boosted CNNs do not produce greater accuracy gains is because linear combinations of CNNs do not add any new expressive power over a single (larger) CNN, unlike boosted decision trees. Another way to view this fact is to think of a single CNN as an ensemble in itself. To see this, consider a convolution layer with $K$ input channels, and $D$ filters ($w_d$, $d \in \{1, 2, \cdots, D\}$). The output of the layer can be written as

$$O[d] = \sum_{k=1}^{K} w_d[k] * I[k] \quad d \in \{1, 2, \cdots, D\} \tag{2}$$

where $*$ is the convolution operator, $[.]$ is the channel slicing, $w_d$ is the $d^{th}$ filter and $O$, $I$ are the output and input of the layer respectively. This is an ensemble over feature channels analogous to the score ensemble used in boosting. In a similar fashion, the matrix vector product in multilayer perceptrons (MLP) (or a fully connected layer in a CNN) can be viewed as an ensemble over features. This view of neural networks as ensembles may provide another way of understanding the robustness of overparameterized networks against overfitting. The same theory that has been developed for understanding the robustness of classifier ensembles against overfitting can be applied to single neural networks.

## References

[1] R. Schapire, "The strength of weak learnability," *Machine Learning*, vol. 5, no. 2, 1990.

[2] Y. Freund, "Boosting a weak learning algorithm by majority," *Information and Computation*, vol. 12, no. 2, 1995.

[3] J. Friedman, T. Hastie, and R. Tibshirani, "Additive logistic regression: a statistical view of boosting," *Annals of Statistics*, vol. 28, no. 2, 2000.

[4] Y. Freund and R. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of Computer and System Sciences*, vol. 55, 1997.

[5] H. Schwenk and Y. Bengio, "Boosting neural networks," *Neural Computation*, vol. 12, no. 8, p. 1869–1887, Aug. 2000.

[6] D. Medera and S. Babinec, "Incremental learning of convolutional neural networks," in *International Joint Conference on Computational Intelligence (IJCCI)*, 2009.

[7] A. Mosca and G. Magoulas, "Deep incremental boosting," in *2nd Global Conference on Artificial Intelligence (GCAI)*, ser. EPiC Series in Computing, vol. 41, 2016, pp. 293–302.

[8] M. Moghimi, M. Saberian, J. Yang, L.-J. Li, N. Vasconcelos, and S. Belongie, "Boosted convolutional neural networks," in *Proceedings of the British Machine Vision Conference (BMVC)*, September 2016.

[9] Y. Freund and R. Schapire, "Experiments with a new boosting algorithm," in *International Conference on Machine Learning (ICML)*, 1996.

[10] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Annals of Statistics*, vol. 29, no. 5, 1999.

[11] L. Mason, J. Baxter, P. Bartlett, and M. Frean, "Boosting algorithms as gradient descent," in *Advances in Neural Information Processing Systems (NIPS)*, 1999.

[12] J. Zhu, S. Rosset, H. Zou, and T. Hastie, "Multi-class boosting," *Statistics and its Interface*, vol. 2, no. 3, 2006.

[13] R. Quinlan, "Bagging, boosting and c4.5," in *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI)*, 1996.

[14] R. Schapire and Y. Singer, "Improved boosting algorithms using confidence-rated predictions," *Machine Learning*, vol. 37, p. 297–336, 1999.

[15] L. Breiman, J. Friedman, R. Olshen, and C. Stone, *Classification and Regression Trees*. Monterey, CA: Wadsworth & Brooks/Cole Advanced Books & Software, 1984.

[16] H. Drucker, R. Schapire, and P. Simard, "Improving performance in neural networks using a boosting algorithm," in *Proceedings of the International Conference on Neural Information Processing Systems (NIPS)*, 1993.

[17] H. Schwenk and Y. Bengio, "Adaboosting neural networks: Application to on-line character recognition," in *International Conference on Artificial Neural Networks (ICANN)*, 1997.

[18] R. Banfield, L. Hall, K. Bowyer, and W. P. Kegelmeyer, "A comparison of decision tree ensemble creation techniques," *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, vol. 29, 2007.

[19] S. Lee, S. Purushwalkam, M. Cogswell, D. Crandall, and D. Batra, "Why m heads are better than one: Training a diverse ensemble of deep networks," *arXiv preprint arXiv:1511.06314 [cs.CV]*, 2015.

[20] A. Mosca and G. D. Magoulas, "Boosted residual networks," in *Engineering Applications of Neural Networks*. Springer International Publishing, 2017, pp. 137–148.

[21] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *22nd SIGKDD Conference on Knowledge Discovery and Data Mining*, 2016.

[22] A. V. Dorogush, V. Ershov, and A. Gulin, "Catboost: gradient boosting with categorical features support," in *Workshop on ML Systems at NIPS*, 2017.

[23] Z. He, T. Lin, Danchen ad Lau, and M. Wu, "Gradient boosting machine: A survey," *arXiv preprint arXiv:1908.06951 [stat.ML]*, 2019.

[24] R. Schapire, "The boosting approach to machine learning: An overview," *Nonlinear Estimation and Classification, Lecture Notes in Statistics*, vol. 171, 2003.

[25] Z. Allen-Zhu, Y. Li, and Z. Song, "A convergence theory for deep learning via over-parameterization," in *International Conference on Machine Learning (ICML)*, 2019.

[26] M. Belkin, D. Hsu, S. Ma, and S. Mandal, "Reconciling modern machine learning practice and the classical bias-variance trade-off," *Proceedings of the National Academy of Sciences*, 2019.

[27] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, "Understanding deep learning requires rethinking generalization," in *International Conference on Learning Representations (ICLR)*, 2017.

[28] T. Poggio, K. Kawaguchi, Q. Liao, B. Miranda, L. Rosasco, X. Boix, J. Hidary, and H. Mhaskar, "Theory of deep learning iii: explaining the non-overfitting puzzle," *MIT Center for Brains, Minds and Machines, Memo No. 073*, 2018.

[29] M. J. Saberian and N. Vasconcelos, "Multiclass boosting: Theory and algorithms," in *Proceedings of the 24th International Conference on Neural Information Processing Systems (NIPS)*, 2011, p. 2124–2132.

[30] P. Viola and M. Jones, "Robust real-time face detection," *International Journal of Computer Vision*, 2004.

[31] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," in *Proceedings of the IEEE*, 1998, pp. 2278–2324.

[32] A. Krizhevsky, "Learning multiple layers of features from tiny images," Tech. Rep., 2009.

[33] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.

[34] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *International Conference on Learning Representations (ICLR)*, 2015.

[35] T. Hastie, S. Rosset, J. Zhu, and H. Zou, "Multi-class adaboost ," *Statistics and Its Interface*, vol. 2, pp. 349–360, 2009.