Neural Computing and Applications (2019) 31:1713–1731
https://doi.org/10.1007/s00521-018-3922-2

EANN 2017

Customised ensemble methodologies for deep learning: Boosted Residual Networks and related approaches

Alan Mosca[1] · George D. Magoulas[1]

Received: 15 January 2018 / Accepted: 29 November 2018 / Published online: 11 December 2018

Abstract
This paper introduces a family of new customised methodologies for ensembles, called Boosted Residual Networks, which builds a boosted ensemble of residual networks by growing the member network at each round of boosting. The proposed approach combines recent developments in residual networks—a method for creating very deep networks by including a shortcut layer between different groups of layers—with Deep Incremental Boosting, a methodology to train fast ensembles of networks of increasing depth through the use of boosting. Additionally, we explore a simpler variant of Boosted Residual Networks based on bagging, called Bagged Residual Networks. We then analyse how the recent developments in ensemble distillation can improve our results. We demonstrate that the synergy of residual networks and Deep Incremental Boosting has better potential than simply boosting a residual network of fixed structure or using the equivalent Deep Incremental Boosting without the shortcut layers, by permitting the creation of models with better generalisation in significantly less time.

1 Introduction

Residual networks are a type of deep network recently introduced in [1], characterised by the use of *shortcut* connections (sometimes also called *skip* connections). These shortcuts link the input of a layer of a deep network to the output of another layer positioned a number of levels "above" it. As a result, each one of these shortcuts shows that networks can be built in *blocks*, which rely on both the output of the previous layer and the previous block. The advent of residual networks has allowed for the development of networks with many more layers than traditional deep networks, in some cases with over 1000 blocks, such as the networks in [2].

The authors gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan X Pascal GPUs used for this research.

✉ Alan Mosca
a.mosca@dcs.bbk.ac.uk

George D. Magoulas
gmagoulas@dcs.bbk.ac.uk

[1] Department of Computer Science and Information Systems Birkbeck, University of London, London, UK

Ensembles of machine learning models have been part of the field for a long time [3, 4] and have recently shown to be an efficient solution to adversarial learning [5] and as a vehicle for improving the single-model accuracy [6], as well as a method for creating better generalisation by consensus of models. Simultaneously, ensemble methods are often left as an *afterthought* in deep learning models: it is generally considered sufficient to treat the deep learning method as a "black-box" and use a well-known generic ensemble method to obtain marginal improvements on the original results. Whilst this is an effective way of improving on existing results without much additional effort, we find that it can amount to a waste of computations. Instead, it would be much better to apply an Ensemble method that is aware of, and makes use of, the underlying deep learning algorithm's architecture.

Such customised approaches for designing ensembles that are specific to a particular model allow us to improve on the generalisation and training speed compared to traditional ensembles, by making use of particular properties of the base classifier's learning algorithm and architecture. We follow this methodology to design a type of ensemble called Boosted Residual Networks (BRN), which makes use of developments in deep learning, previous other customised ensemble methodologies, and combines several ideas to achieve improved results on benchmark data sets. We then

build on these results to construct related variations of this method, to highlight how such customised ensemble methods can be created with particular specific properties.

The version of BRN presented in this paper presents some performance improvements over the previous version presented in [7]. The new version allows for a variant suitable for networks whose outputs are real-valued, called BRN.R, and we present a further derivation, based on bagging [8] instead of boosting, called BaRN.

Using a customised ensemble allows us to improve on the generalisation and training speed of other ensemble methods by making use of the knowledge of the base classifier's previous learning, structure, and architecture. Experimental results show that Boosted Residual Networks achieve improved results on benchmark data sets.

When compared to the existing customised ensemble methods such as DIB [9], BRN enables the creation of almost arbitrary length models, thanks to the ability of residual networks to not be affected by the common issues created by a large number of layers, such as vanishing or exploding gradients [1].

In Sects. 2–4 we present the prerequisite background to BRN. Section 5 presents the methodology itself. Section 6 explores an additional method based on bagging. Section 7 analyses the application of distillation to our methods and the chosen baselines. Section 8 shows the experimental results. Section 9 provides further analysis and explores potential future work.

2 Relevant techniques in deep learning

This section covers the existing literature on several techniques from deep learning that are necessary as a background to Boosted Residual Networks.

2.1 Shortcut connections in networks

The idea of adding shortcut connections in a network was introduced in the past in [10–14]. Work has been done, for example, to add a single linear layer between the input and the output of a network to simplify the learned function [14]. Other research utilises shortcut connections to address internal issues to network, such as vanishing gradients, layer responses, and propagated errors [15–18]. Highway networks [19, 20] are also a type of network that uses shortcut connections. In this case, the shortcut connection is guarded by a learned gating system, so it is no longer a simple identity function. The "information highways" created by this process are argued to enable the network to route information internally, enabling the training of deeper networks.

Dense convolutional neural networks [21] are another type of network that makes use of shortcuts, with the difference that each layer block is directly connected to all its ancestor layer blocks by a shortcut link. This increases significantly the computational complexity of the network, adding training time and memory requirements to the training process.

2.2 Residual networks

Residual networks [1] are a particular type of convolutional neural network built on the notion of connected *blocks* of layers. Each block in a residual network is composed of a combination of convolutional, pooling, or batch normalisation layers. These blocks are connected to each other both in a sequential feed-forward layout, as seen in standard convolutional networks, as well as via *skip connections*. Each skip connection provides a link between the output of the final layer of a block b_i to the input of a descendant b_j. A skip connection is then created for each of the descendants $b_{i+1} \ldots b_n$, where n is the total number of blocks in the network. These particular skip connections only connect forwards and do not form loops in the network. Residual networks have enabled the creation of very deep networks, in some cases in excess of 1000 layers [2]. This is because the technique has been explicitly created to solve the problems that are usually associated with the depth of a network.

The goal of the residual network is to explicitly let layers approximate a *residual function* $F(x) = H(x) - x$, where $H(x)$ is the true target function to be learned. The output is then recast as $F(x) + x$ to predict the original $H(x)$ again. This is based on the assumption that $H(X) - x$ is much easier to learn than just $H(X)$. Early work on residual networks has shown that they are very good at addressing the *degradation* problem: as a network gains additional layers, it becomes progressively harder to learn the target function, with accuracy degrading very rapidly. It is to be noted that this is not due to overfitting. (The increased error is observed on the training set as well.)

An observation made in Ref [1] is that if we construct a larger network by copying the layers from a smaller network and adding identity layers, we will obtain the same accuracy as the smaller network, with an indefinite amount of identity layers added. This is an important principle which sets the notion that a larger network can always be at least as good as a smaller one. This principle also supports the idea of Boosted Residual Networks, Deep Incremental Boosting, and residual networks, as it is crucial to be able to extend networks to an arbitrary number of layers.

2.3 Learning additive improvements

In the presentation of DIB [9], the notion is introduced that each new layer being added to the network is learning

corrections from the previous model. It has been shown that this principle is also applicable to residual networks and highway networks [22], where each additional block can be in fact equated to a further unrolling of an iterative learning procedure. Therefore, it is also shown that each new block in such networks is not necessarily learning increasingly higher-level representations, but additional refinements of the estimates of the previous layers. This principle partially justifies the empirical observation that at each round of BRN (and variants), the accuracy of the single classifier improves.

2.4 Transfer of learning in convolutional networks

Transfer of learning has also had an impact on deep learning. For example, for convolutional networks, certain sub-features in the lower layers of a trained network have been shown to be entirely transferrable to a new CNN. This leads to improved training results, and much faster training compared to having trained the entire network from scratch, as shown in [23]. Additionally, specific experimental work on computer vision data set shows that mid-level representations are transferrable between networks trained on different data sets [24].

An illustration of how the early and middle layers are copied between different architectures is shown in Fig. 1.

2.5 Comparison to approximate ensembles

Whilst both residual networks and Densely Connected Convolutional Networks may be unfolded into an equivalent ensemble, we note that there is a differentiation between an actual ensemble method and an ensemble "approximation". During the creation of an ensemble, one of the principal factors is the creation of *diversity*: each base learner is trained independently, on variations (re-samples in the case of boosting algorithms) of the training set, so that each classifier is guaranteed to learn a different function that represents a view of the original training data set. This is the enabling factor for the ensemble to perform better in aggregate.

2.5.1 Residual networks as an approximation of an ensemble

A recent study in [25] compares residual networks to an ensemble of smaller networks. This is done by unfolding the shortcut connections into the equivalent tree structure, which closely resembles an ensemble. An example of this is shown in Fig. 2.

2.5.2 Densely Connected Convolutional Networks as an approximation of an ensemble

In the case of Densely Connected Convolutional Networks (DCCN) specifically, one may argue that a partial unfolding of the network could be, from a schematic point of view, very similar to an ensemble of additively constructed residual networks. We make the observation that, although this would be correct, on top of the benefit of diversity, our method also provides a much faster training methodology: the only network that is trained for a full schedule is the network created at the first round, which is also the

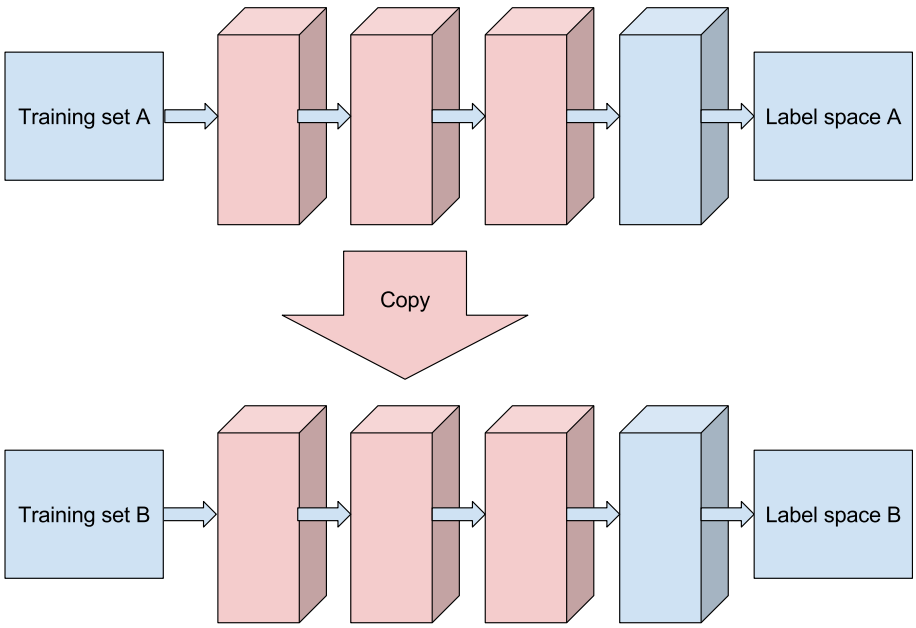


Fig. 1 Illustration of the transfer learning process in convolutional networks: a network trained on a data set in problem space *A* donates the weights from its lower and middle layers to initialise a new network. This is subsequently trained on a data set from a seemingly unrelated problem space *B*

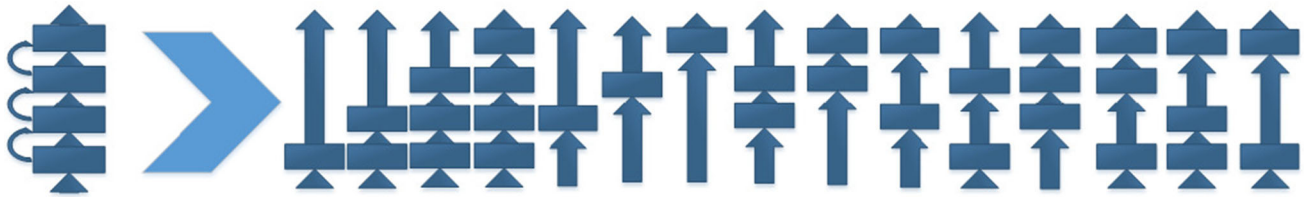


Fig. 2 A residual network of N blocks can be unfolded into an ensemble of $2^N - 1$ smaller networks

smallest one. All subsequent networks are trained for a much shorter schedule, saving a considerable amount of time. Additionally, whilst the schematic may seem identical, there is a subtle difference: each member network outputs a classification of its own, which is then aggregated by a weighted averaging determined by the errors on the test set. Instead, in a DCCN the input of the final aggregation layer is the output of each underlying set of layers. We conjecture that this aggressive dimensionality reduction before the aggregation has a regularising effect on the ensemble.

3 Traditional boosting methods

Boosting is a technique first introduced in [3, 26], by which classifiers are trained sequentially, using a subset of the original data set, with the prediction error from the previous classifiers affecting the sampling weight for the next round. After each round of boosting, the decision can be made to terminate and use a set of calculated weights to apply as a linear combination of the newly created set of learners.

3.1 AdaBoost

In [3], Freund and Schapire present two variants of boosting, called AdaBoost.M1 and AdaBoost.M2. The main difference between the two algorithms is in the way the final hypothesis is calculated and how multiple class problems are handled, with both variants shown in detail in Algorithms 1 and 2. Each boosting variant builds a distribution of training set resampling weights D_t. D_t is updated at each iteration to increase the *importance* of the examples that are harder to classify correctly. Each resampled data set is used to train a new classifier h_t, which is then incorporated in the group with a weight α_t, based on its classification error ϵ_t. The new D_t is then generated for the next iteration. The main differences between each AdaBoost variant lie in how the error ϵ_t, the classifier weight α_t, the data set distribution D_t, and the aggregation functions are designed and implemented.

Algorithm 1 AdaBoost.M1

Inputs: training set $\boldsymbol{X_0}$, an algorithm to create classifier hypotheses $h(X)$
Outputs: a trained ensemble classifier $H(\boldsymbol{X})$
$D_{0,i} = 1/M \forall i$
$t = 0$
$\boldsymbol{W}_0 \leftarrow$ randomly initialised weights for first classifier
while $t < t_{end}$ **do**
  $\boldsymbol{X}_t \leftarrow$ sample from $\boldsymbol{X}_0$ with distribution $\boldsymbol{D}_t$
  $h_t \leftarrow$ new classifier on current subset
  $\epsilon_t = \sum_{i:h_t(x_i) \neq y_i} D_t(i)$
  if $\epsilon_t > \frac{1}{2}$ **then**
    abort loop
  end if
  $\beta_t = \epsilon_t/(1-\epsilon_t)$
  $D_{t+1,i} = \frac{D_{t,i}}{Z_t} \cdot \begin{cases} \beta_t & if h_t(x_i) = y_i \\ 1 & otherwise \end{cases} | \forall i = 1 \cdot |x|$
  where Z_t is a normalisation factor such that $\boldsymbol{D}_{t+1}$ is a distribution
  $\alpha_t = \frac{1}{\beta_t}$
  $t = t + 1$
end while
$H(x) = \text{argmax}_{y \in Y} \sum_{t=1}^{T} log \alpha_t h_t(x, y)$

Algorithm 2 AdaBoost.M2

Inputs: training set $\boldsymbol{X_0}$, an algorithm to create classifier hypotheses $h(X)$
Outputs: a trained ensemble classifier $H(\boldsymbol{X})$
$D_{0,i} = 1/M$ for all i
$t = 0$
$\boldsymbol{W}_0 \leftarrow$ randomly initialised weights for first classifier
while $t < t_{end}$ **do**
  $\boldsymbol{X}_t \leftarrow$ sample from $\boldsymbol{X}_0$ with distribution $\boldsymbol{D}_t$
  $h_t \leftarrow$ new classifier on current subset
  $\epsilon_t = \frac{1}{2}\sum_{(i,y)\in B} D_{t,i}(1 - h_t(x_i, y_i) + h_t(x_i, y))$
  $\beta_t = \epsilon_t/(1 - \epsilon_t)$
  $D_{t+1,i} = \frac{D_{t,i}}{Z_t} \cdot \beta^{(1/2)(1+h_t(x_i,y_i)-h_t(x_i,y))} | \forall i = 1 \cdot |x|$
  where Z_t is a normalisation factor such that $\boldsymbol{D}_{t+1}$ is a distribution
  $\alpha_t = \frac{1}{\beta_t}$
  $t = t + 1$
end while
$H(x) = \text{argmax}_{y\in Y} \sum_{t=1}^{T} log\alpha_t h_t(x, y)$

3.2 SAMME

The original AdaBoost algorithm works very well in the binary classification setting. However, when the number of output classes $k > 2$ it suffers from problems with weak classifiers with error above $\frac{1}{2}$, which led the authors to create AdaBoost.M2. Another solution is presented as Stagewise Additive Modeling using a Multiclass Exponential loss function (SAMME) [27]. SAMME compensates for the fact that α would be negative for errors above $\frac{1}{2}$. SAMME is shown in Algorithm 3. An in-depth study of multiclass boosting is provided in [28].

Algorithm 3 SAMME

Inputs: training set $\boldsymbol{X_0}$, an algorithm to create classifier hypotheses $h(X)$
Outputs: a trained ensemble classifier $H(\boldsymbol{X})$
set k to the number of output classes in the problem
$D_{0,i} = 1/M$ for all i
$t = 0$
$\boldsymbol{W}_0 \leftarrow$ randomly initialised weights for first classifier
while $t < t_{end}$ **do**
  $\boldsymbol{X}_t \leftarrow$ sample from $\boldsymbol{X}_0$ with distribution $\boldsymbol{D}_t$
  $h_t \leftarrow$ new classifier on current subset
  $\epsilon_t = \frac{\sum_{i=1}^{n} D_t(i)\mathbb{I}(y_i \neq h_t(X_t)))}{\sum_{i=1}^{n} D_t(i)}$
  $\alpha_t = log\frac{1-\epsilon_t}{\epsilon_t} + log(k - 1)$
  $D_{t+1,i} = \frac{D_{t,i}}{Z_t} e^{\alpha_t \mathbb{I}(y_i \neq h_t(X_t)))} | \forall i = 1 \cdot |x|$
  where Z_t is a normalisation factor such that $\boldsymbol{D}_{t+1}$ is a distribution
  $t = t + 1$
end while
$H(x) = \text{argmax}_{y\in Y} \sum_{t=1}^{T} \alpha_t h_t(x, y)$

When the base classifier outputs a real-valued probability $P(k|\mathbf{x})$ rather than a one-hot encoded class decision, it may prove advantageous to utilise this additional information to calculate more precise sampling weights and improve the classifier's output. AdaBoost.M2 is an example of an algorithm the exploits this property. A variant of SAMME for classifiers that exploits this knowledge also exists, called SAMME.R [27], shown in Algorithm 4.

Algorithm 4 SAMME.R

Inputs: training set $\boldsymbol{X_0}$, an algorithm to create classifier hypotheses $h(X)$
Outputs: a trained ensemble classifier $H(\boldsymbol{X})$
set k to the number of output classes in the problem
$D_{0,i} = 1/M$ for all i
$t = 0$
$\boldsymbol{W}_0 \leftarrow$ randomly initialised weights for first classifier
while $t < t_{end}$ **do**
  $\boldsymbol{X}_t \leftarrow$ sample from $\boldsymbol{X}_0$ with distribution $\boldsymbol{D}_t$
  $h_t \leftarrow$ new classifier on current subset
  Obtain weighted class probability estimates $p_i(\boldsymbol{X}) = \mathbf{P}_t(y = c_i|\boldsymbol{X}_t, h_t), i = 1 \ldots k$
  replace $h_t(\boldsymbol{X_t}) \leftarrow (k-1)\left(log p_i(\boldsymbol{X}) - \frac{1}{k}\sum_{j=1}^{k} log p_j(\boldsymbol{X})\right), i = 1 \ldots k$
  $D_{t+1,i} = \frac{D_{t,i}}{Z_t} e^{-\frac{k-1}{k}\boldsymbol{y}^T log \boldsymbol{p}(\boldsymbol{X_t}a)} | \forall i = 1 \cdot |x|$
  $t = t + 1$
end while
$H(x) = \text{argmax}_{y \in Y} \sum_{t=1}^{T} h_t(x, y)$

effective way to learn the *corrections* introduced by the emphasisation of learning mistakes of the boosting process. The argument as to why this works effectively is based on the fact that the data sets at rounds t and $t+1$ will be *mostly similar*, and therefore, a classifier h_t that performs better than randomly on the resampled data set $\boldsymbol{X}_t$ will also perform better than randomly on the resampled data set $\boldsymbol{X}_{t+1}$. This is under the assumption that both data sets are sampled from a common ancestor set $\boldsymbol{X}_a$. It is subsequently shown that such a classifier can be retrained on the differences between $\boldsymbol{X}_t$ and $\boldsymbol{X}_{t+1}$.

This practically enables the ensemble algorithm to train the subsequent rounds for a considerably smaller number of epochs, consequently reducing the overall training time by a large factor. The original paper also provides a conjecture-based justification for why it makes sense to extend the previously trained network to learn the "corrections" taught by the boosting algorithm. A high-level description of the method is shown in Algorithm 5, and the structure of the network at each round is illustrated in Fig. 3.

4 An existing customised method: deep incremental boosting

Deep Incremental Boosting, introduced in [9], is an example of such customised ensemble methods developed for building ensembles of convolutional networks. The method makes use of principles from transfer of learning, for example those used in [23], applying them to conventional AdaBoost [26].

Deep Incremental Boosting increases the size of the network at each round by adding new layers at the end of the network. This, as discussed, is extremely unlikely to harm the learning process. In the original paper on Deep Incremental Boosting [9], this has been shown to be an

Algorithm 5 Deep Incremental Boosting

Inputs: training set $\boldsymbol{X_0}$, a modifiable algorithm to create classifier hypotheses $h(X)$
Outputs: a trained ensemble classifier $H(\boldsymbol{X})$
$D_{0,i} = 1/M$ for all i
$t = 0$
$\boldsymbol{W}_0 \leftarrow$ randomly initialised weights for first classifier
while $t < t_{end}$ **do**
  $\boldsymbol{X}_t \leftarrow$ sample from $\boldsymbol{X}_0$ with distribution $\boldsymbol{D}_t$
  $u_t \leftarrow$ create untrained classifier with additional layer of shape L_{new}
  copy weights from $\boldsymbol{W}_t$ into the bottom layers of u_t
  $h_t \leftarrow$ train u_t classifier on current subset
  $\boldsymbol{W}_{t+1} \leftarrow$ all weights from h_t
  $\epsilon_t = \frac{1}{2} \sum_{(i,y) \in B} D_{t,i}(1 - h_t(x_i, y_i) + h_t(x_i, y))$
  $\beta_t = \epsilon_t / (1 - \epsilon_t)$
  $D_{t+1,i} = \frac{D_{t,i}}{Z_t} \cdot \beta^{(1/2)(1+h_t(x_i,y_i)-h_t(x_i,y))} | \forall i = 1 \cdot |x|$
  where Z_t is a normalisation factor such that $\boldsymbol{D}_{t+1}$ is a distribution
  $\alpha_t = \frac{1}{\beta_t}$
  $t = t + 1$
end while
$H(x) = \text{argmax}_{y \in Y} \sum_{t=1}^{T} log \alpha_t h_t(x, y)$

5 Creating the Boosted Residual Network

In this section we propose a method for generating Boosted Residual Networks. This works by increasing the size of an original residual network by one residual block at each round of boosting. The method achieves this by selecting an *injection point* index p_i at which the new block is to be added. It is to be noted that p_i is not necessarily the last block in the network.

The boosting method performs an iterative re-weighting of the training set, which skews the resample at each round to *emphasise* the training examples that are harder to learn. Therefore, it becomes necessary to utilise the entire ensemble at test time, rather than just use the network trained in the last round. It is also possible to delete individual blocks from a residual network at training and/or testing time, as presented in [1]; however, this issue is considered out of the scope of this paper.

The iterative algorithm used in the paper is shown in Algorithm 6. At the first round, the entire training set is used to train a network of the original *base* architecture, for

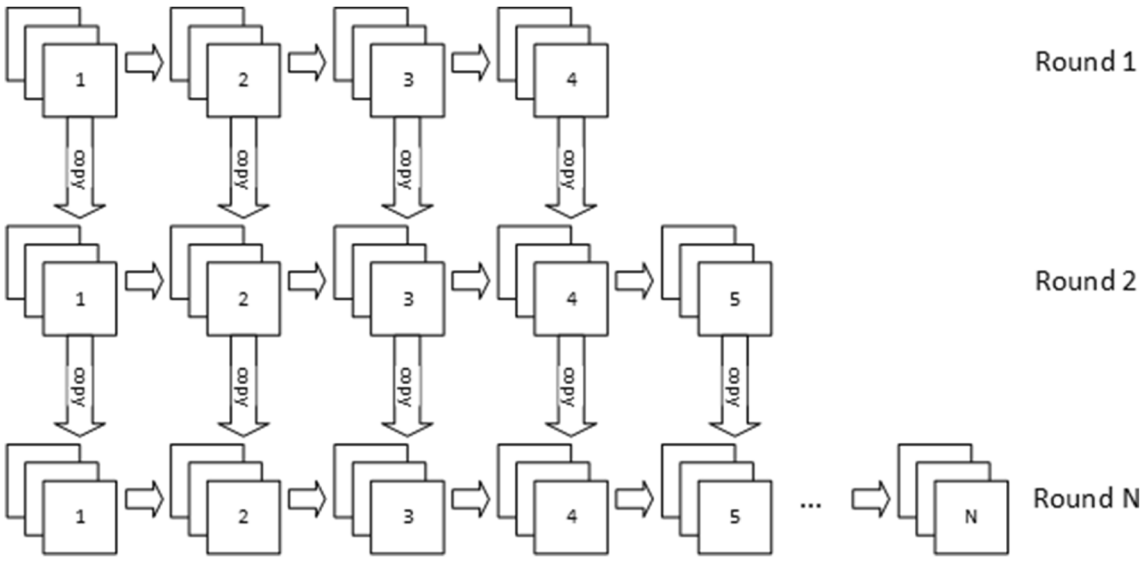


Fig. 3 Example illustration of how new members of the ensemble are created in each subsequent round of Deep Incremental Boosting. At each round a new layer is added to the previous network, starting at $p_0 = 4$. The weights of all layers below the newly inserted one are copied between rounds

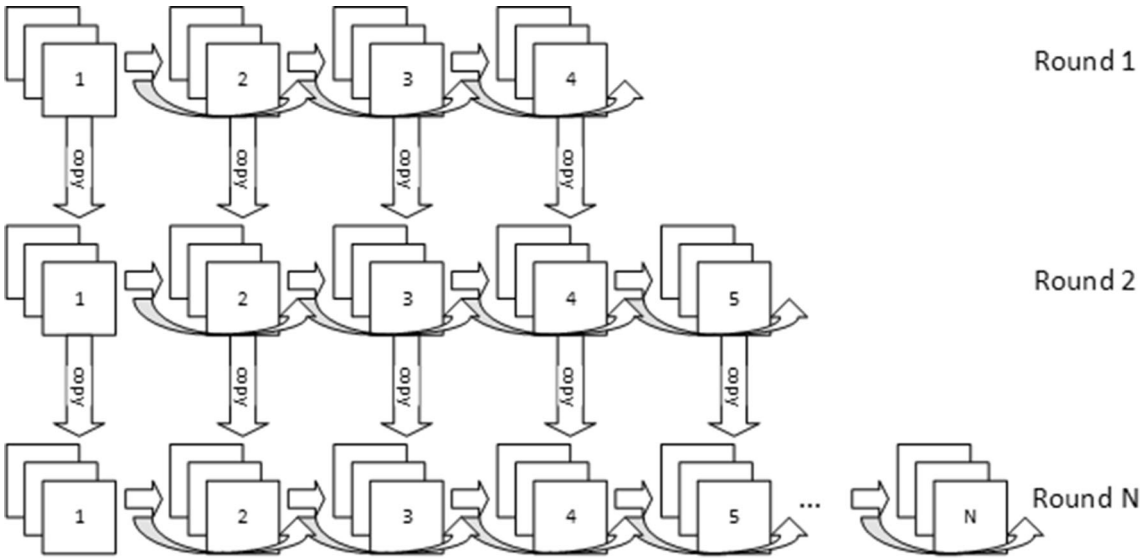


Fig. 4 Illustration of subsequent rounds of Boosted Residual Networks

a number of epochs n_0. After the first round, the following steps are taken at each subsequent round t:

- The ensemble constructed so far is evaluated on the training set to obtain the set errors ϵ, so that a new training set can be sampled from the original training set. This is a step common to all boosting algorithms.
- A new network is created with the same structure as that of the previous round. To this network, a new block of layers B_{new} is added immediately after position p_t, which is determined as an initial predetermined position p_0 plus an offset $\sum_{i=1\rightarrow p} \delta_i$ for all the blocks added at previous layers, where δ_i is chosen to be the size of the newly added layers at round i. This puts the new block of layers immediately after the block of layers added at the previous round, so that all new blocks are effectively added sequentially. B_{new} is not a residual block, but usually consists of a group of different layers (e.g. batch normalisation, convolution, and activation).
- The weights from the layers below p_t are copied from the network trained at round $t-1$ to the new network. This step allows to considerably shorten the training thanks to the transfer of learning shown in [23].
- The newly created network is subsequently trained for a reduced number of epochs n_t.
- The new network is added to the ensemble following the conventional rules and weight $\alpha_t = \frac{1}{\beta_t}$ used in AdaBoost. We did not see a need to modify the way β_t is calculated, as it has been performing well in both DIB and many AdaBoost variants [3, 9, 26, 29].

Figure 4 shows a diagram of how the ensemble is constructed by deriving the next network at each round of boosting from the network used in the previous round.

Algorithm 6 Boosted Residual Networks

Inputs: training set $\boldsymbol{X_0}$, a modifiable algorithm to train Residual Network hypotheses $h(X)$
Outputs: a trained ensemble classifier $H(\boldsymbol{X})$
$D_{0,i} = 1/M$ for all i
$t = 0$
$\boldsymbol{W}_0 \leftarrow$ randomly initialised weights for first classifier
$p_0 \leftarrow$ initial injection position
while $t < T$ **do**
  $\boldsymbol{X}_t \leftarrow$ sample from $\boldsymbol{X}_0$ with distribution $\boldsymbol{D}_t$
  $u_t \leftarrow$ create untrained classifier with an additional block $\boldsymbol{B}_{new}$ of pre-determined shape $\boldsymbol{N}_{new}$
  determine block injection position $p_t = p_{t-1} + |\boldsymbol{B}_{new}|$
  connect the input of $\boldsymbol{B}_{new}$ to the output of layer $p_t - 1$
  connect the output of $\boldsymbol{B}_{new}$ and of layer $p_t - 1$ to a merge layer m_i
  connect the merge layer to the remainder of the network
  copy weights from $\boldsymbol{W}_t$ into the bottom layers $l < p_t$ of u_t
  $h_t \leftarrow$ train u_t classifier on current subset
  $\boldsymbol{W}_{t+1} \leftarrow$ all weights from h_t
  $\epsilon_t = \sum_{i:h_t(x_i) \neq y_i} D_t(i)$
  if $\epsilon_t > \frac{1}{2}$ **then**
    abort loop
  end if
  $\beta_t = \epsilon_t/(1 - \epsilon_t)$
  $D_{t+1,i} = \frac{D_{t,i}}{Z_t} \cdot \begin{cases} \beta_t & if h_t(x_i) = y_i \\ 1 & otherwise \end{cases} |\forall i = 1 \cdot |x|$
  where Z_t is a normalisation factor such that $\boldsymbol{D}_{t+1}$ is a distribution
  $\alpha_t = \frac{1}{\beta_t}$
  $t = t + 1$
end while
$H(x) = \text{argmax}_{y \in Y} \sum_{t=1}^{T} log \alpha_t h_t(x, y)$

We identified a number of optional variations to the algorithm that may be implemented in practice, which we have empirically established as not having a significant impact on the overall performance of the network. We report them here for completeness.

- Freezing the layers that have been copied from the previous round and perform a round of "local learning" by only training the new layers, before performing an (optional) round of "global learning". This is common practice for many supervised and unsupervised transfer learning approaches and could provide a valuable improvement in performance for some data sets.
- Only utilising the weights distribution for the examples in the training set instead of resampling, as an input to the training algorithm.
- Inserting the new block always at the same position, rather than after the previously inserted block. (We found this to affect performance negatively.)

In the extreme cases where the base classifier learns the training set very well (or indeed perfectly), the value of α_t goes towards its asymptote of $+\inf$. This causes problems with both resampling weights and ensemble weights, so it is necessary to cap the value of α_t. Empirically, bounds of $(10^{-3}, 10^{3})$ have proven to contain the runaway effects whilst not affecting the learning in the non-degenerate case.

In a similar way to how SAMME.R extends SAMME, we present BRN.R as an extension of BRN, which derives its boosting procedure from SAMME.R to take advantage of the same *real-valued classifiers*. BRN.R is shown in Algorithm 7.

Algorithm 7 BRN.R

Inputs: training set $\boldsymbol{X_0}$, a modifiable algorithm to create classifier hypotheses $h(X)$
Outputs: a trained ensemble classifier $H(\boldsymbol{X})$
$D_{0,i} = 1/M$ for all i
$t = 0$
$\boldsymbol{W}_0 \leftarrow$ randomly initialised weights for first classifier
while $t < t_{end}$ **do**
  $u_t \leftarrow$ create untrained classifier with an additional block $\boldsymbol{B}_{new}$ of pre-determined shape $\boldsymbol{N}_{new}$
  determine block injection position $p_t = p_{t-1} + |\boldsymbol{B}_{new}|$
  connect the input of $\boldsymbol{B}_{new}$ to the output of layer $p_t - 1$
  connect the output of $\boldsymbol{B}_{new}$ and of layer $p_t - 1$ to a merge layer m_i
  connect the merge layer to the remainder of the network
  copy weights from $\boldsymbol{W}_t$ into the bottom layers $l < p_t$ of u_t
  $h_t \leftarrow$ train u_t classifier on current subset
  Obtain weighted class probability estimates $p_i(\boldsymbol{X}) = \mathbf{P}_t(y = c_i|\boldsymbol{X}_t, h_t), i = 1 \ldots k$
  replace $h_t(\boldsymbol{X_t}) \leftarrow (K-1)\left(logp_i(\boldsymbol{X}) - \frac{1}{k}\sum_{j=1}^{k} logp_j(\boldsymbol{X})\right), i = 1 \ldots k$
  $D_{t+1,i} = \frac{D_{t,i}}{Z_t} e^{-\frac{k-1}{k}\boldsymbol{y}^T log\boldsymbol{p}(\boldsymbol{X_t}a)|\forall i = 1 \cdot |x|}$
  $t = t + 1$
end while
$H(x) = \text{argmax}_{y \in Y} \sum_{t=1}^{T} h_t(x, y)$

5.1 The sensitivity of additional hyperparameters

BRN introduces a new set of hyperparameters that can be analysed. These hyperparameters can also be selected by search methods or evolutionary strategy, but given the computational requirements for training a large number of ensembles of deep networks we have not been able to conduct sufficient experiments to devise an optimal strategy.

First, we consider the position p_t at which a new residual block $\boldsymbol{B}_{\text{new}}$ is injected into the network. This governs the structure of the network at each boosting round, but more importantly, the number of layers that will have their weights initialised from a copy of the previous round. In our experiments we found that using the maximum possible value of p_t at each round produced the best results, in both generalisation ability and training speed up—we were able to reduce the number of training epochs for the subsequent rounds ($t > 1$) by a greater amount when the value of p_t was higher. Intuitively, this indicates that transferring a higher number of layers produces higher benefits.

Second, we consider the fact that a cut-off point $t_{\max}$ could be introduced for no longer adding new residual blocks. Our experiments indicated that, for ten rounds of boosting, adding such a cut-off point did not produce any further improvements. However, when generating much larger ensembles (for example 1000 members) it will likely be beneficial to provide an upper limit to the size of the networks being produced. Even though there have been residual networks with over 1000 layers [1, 30], it has not been guaranteed that adding an indefinite number of residual blocks will always produce better results. Adding this constraint will also help contain the amount of computation required and therefore the speed of training each member.

Third, the structure of the new residual block $\boldsymbol{B}_{\text{new}}$ has to be chosen appropriately. In residual networks, each block tends to belong to one of a few *families* of blocks defined in the structure of each network. Our experiments confirm that the best strategy is to create the new block $\boldsymbol{B}_{\text{new}}$ such that its structure is the same as its predecessor block $\boldsymbol{B}_{\text{p}_t}$. This results in each boosting round creating a "longer" version of the original network, without the addition of new families of blocks.

6 A related approach based on bagging

Bagging (short for "bootstrap aggregating") is a technique that is based on the statistical bootstrapping method, originally introduced in [8], where the original author also shows a number of applied use cases. A quantity N of bootstraps is created by randomly picking M elements from a training data set of size Z with resampling and then using each of these bootstraps to train a separate identical base classifier. Reference [8] introduces bagging with $M = Z$, and this practice seems to be observed in most of the

literature. This will create diverse members because of the randomised resampling, but because there will be significant overlap in the training sets, all the members will still have positive correlation.

The fact that boosting focuses the data set resampling on harder-to-classify examples has the effect that the Boosted Residual Networks cannot be used as a way to train a single residual network additively. However, it is possible to alleviate this situation by deriving an approach that uses bagging instead of boosting; therefore, removing the necessity to use the entire ensemble at test time.

The principle of additively creating an ensemble of progressively larger residual networks, when extended to bagging, generates a less complex process. We call this the Bagged Residual Network (BaRN). This method offers the same advantages and disadvantages that bagging offers over boosting. Based on the original bagging recipe [8], the algorithm is illustrated in Algorithm 8.

Algorithm 8 Bagged Residual Networks

$t = 0$
$p_0 \leftarrow$ initial injection position
while $t < T$ **do**
  $\boldsymbol{X}_t \leftarrow$ sample from $\boldsymbol{X}_0$ with uniform distribution
  $u_t \leftarrow$ create untrained classifier with an additional block $\boldsymbol{B}_{new}$ of pre-determined shape $\boldsymbol{N}_{new}$
  determine block injection position $p_t = p_{t-1} + |\boldsymbol{B}_{new}|$
  connect the input of $\boldsymbol{B}_{new}$ to the output of layer $p_t - 1$
  connect the output of $\boldsymbol{B}_{new}$ and of layer $p_t - 1$ to a merge layer m_i
  connect the merge layer to the remainder of the network
  copy weights from $\boldsymbol{W}_t$ into the bottom layers $l < p_t$ of u_t
  $h_t \leftarrow$ train u_t classifier on current subset
  $t = t + 1$
end while
$H(x) = \text{argmax}_{y \in Y} \sum_{t=1}^{T} log\alpha_t h_t(x, y)$

7 Distilled ensembles

It has been shown [31] that it is possible to approximate a deep neural network by using a more shallow one that is subsequently trained on its output, with the goal to emulate its output function. No restriction is mentioned with regard to generalising this approach to ensembles, and it should be theoretically possible to train a smaller model to perform like the larger one, as has been done, for example, in [32], where the authors have developed a new set of algorithms to approximate larger ensembles.

The process of distillation, introduced in Ref [33], produces small networks that emulate the behaviour of larger, more complex ones. It does so by utilising the output function $f'(X)$ of the *cumbersome* model as the target of the learning algorithm that generates the smaller one. It has been shown to be an effective process for regularising large ensembles of convolutional networks [6, 34]. By applying this principle to Boosted Residual Networks, we can create a new network, of the size of the network at the first round of boosting, that learns from the output of the ensemble. This improves the portability of the ensemble whilst not impacting the performance in any significant way, and in certain cases even improving it.

This distillation process has been applied to BRN, DIB, and BaRN, as it is possible to apply the same principle to all the ensemble learning algorithms. Figure 5 illustrates graphically the distillation process.

8 Experiments and discussion

In the experiments we used the MNIST, CIFAR-10, CIFAR-100, and TinyImagenet data sets. These are very common benchmark data sets in computer vision and have been used extensively to evaluate the performance of deep learning methods in the literature. A comprehensive list of experiments in the literature that have used these benchmarks can be found in Ref [35]. We compared Boosted Residual Networks (BRN) with an equivalent Deep Incremental Boosting without the skip connections (DIB), AdaBoost, and bagging with both the initial network as the base classifier (AdaBoost) and the single residual network equivalent to the last round of Boosted Residual Networks (ResNet), and Bagged Residual Networks (BaRN). All the parameters for training have been kept fixed for all experiments, and no further hyperparameter optimisation has been done on the base classifiers beyond that for improving the performance of the individual network

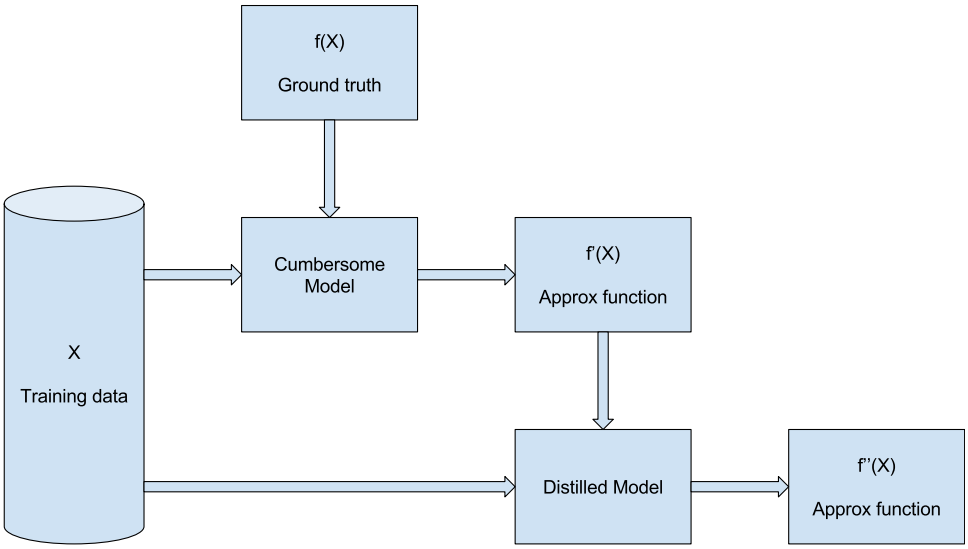


Fig. 5 Illustration of the distillation process: the cumbersome model creates an approximate function $f'(x)$ by learning from the training data and the ground truth function $f(x)$, whilst the distilled model learns a new second-order approximate function $f''(x)$ from the cumbersome approximate function

(ResNet). We performed a manual hyperparameter search for the individual residual network, before running the first experiment, on a small subset of each data set, using 10,000 images for training and 10,000 for testing. We then fixed the hyperparameters we found and used them for every experiment we ran for the data set in question.

In order to reduce noise in the results, we aligned the random initialisation of all network weights across experiments, by fixing the seeds for the random number generators. All experiments were repeated 10 times, and we report the mean accuracy values. This approach has guaranteed control over the variables that could have affected the learning, leaving only the ensemble method and its specific hyperparameters as the free variables being evaluated.

As already mentioned, MNIST [36] is a common computer vision data set that associates 70,000 pre-processed images of hand-written numerical digits with a class label representing that digit. The input features are the raw pixel values for the 28×28 images, in greyscale, and the outputs are the numerical value between 0 and 9. 50,000 samples are used for training, 10,000 for validation, and 10,000 for testing.

CIFAR-10 is a data set that contains 60,000 small images of 10 categories of objects. It was first introduced in [37]. The images are 32×32 pixels, in RGB format. The output categories are *airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck*. The classes are completely mutually exclusive so that it is translatable to a *1-vs-all* multiclass classification. 50,000 samples are used for training, and 10,000 for testing. This data set was originally constructed without a validation set.

CIFAR-100 is a data set that contains 60,000 small images of 100 categories of objects, grouped in 20 superclasses. It was first introduced in [37]. The image format is the same as CIFAR-10. Class labels are provided for the 100 classes as well as the 20 super-classes. A super-class is a category that includes 5 of the fine-grained class labels (e.g. "insects" contains *bee, beetle, butterfly, caterpillar, cockroach*). 50,000 samples are used for training, and 10,000 for testing. This data set was originally constructed without a validation set.

TinyImagenet is a simplified version of the Imagenet challenge data set [38]. It has 1,20,000 images, split into 1,00,000 for training, 10,000 for validation and 10,000 for testing, each 64×64 pixels in size. The data set comprises of 200 different classes, equally balanced through each split of the data set. It is derived completely from a small sample of the original Imagenet data set. Because the labels for the test set have not been released to the public, for this data set we had to use the validation set as the test set.

For the CIFAR-10 and CIFAR-100 data sets, we also report results with light data augmentation: we randomly rotated, flipped horizontally, and scaled images, but did not use any heavy augmentation, including random crops. For TinyImagenet no data set augmentation was used. Results are reported in Table 1. It is important to note that, except for TinyImagenet, these accuracy values are very close to the state of the art at the time of writing (99.79% for MNIST [39], 96.53% for CIFAR-10 [40], and 75.72% for CIFAR-100 [41]), but instead of using specially crafted methods and architectures, we have instead taken a general approach by using significantly smaller and less complex networks, with little effort dedicated to the search of optimal hyperparameters. It is also to be noted that the state-of-the-art methods make use of heavy data set augmentation, whilst our tests do not. There are multiple reasons why the performance on TinyImagenet is not close to state of the art. We used the same network architecture and hyperparameters as CIFAR-100, without any data set

Table 1 Mean test accuracy in the benchmark data sets for the methods compared

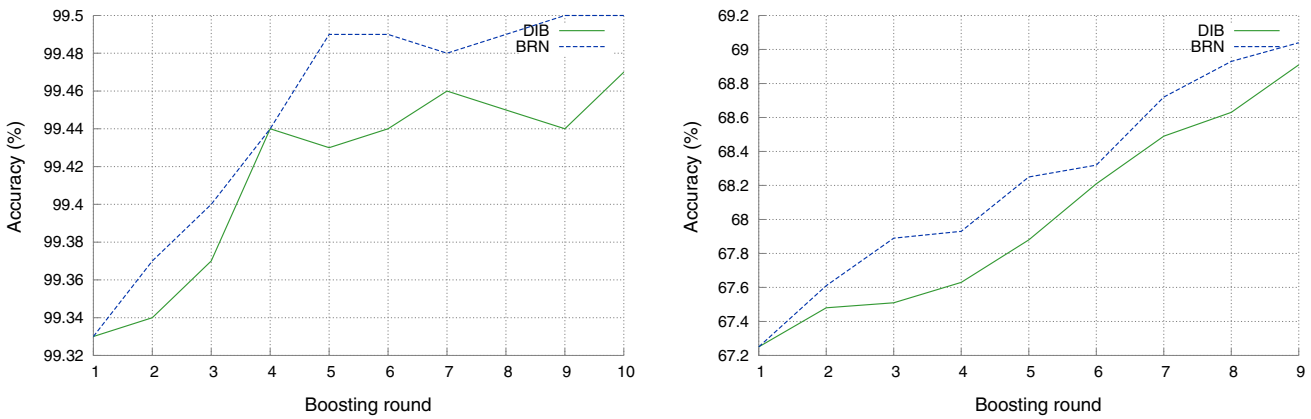
	ResNet (%)	Bagging (%)	AdaBoost (%)	DIB (%)	BRN (%)	BRN.R (%)	BaRN
MNIST	99.41	99.46	99.42	99.47	99.53	**99.55**	**99.55**
CIFAR-10	89.12	90.43	89.74	90.83	90.85	**91.04**	90.82
CIFAR-10 (aug)	92.14	92.61	92.47	92.51	92.94	**92.96**	92.80
CIFAR-100	67.25	68.15	69.11	69.16	70.79	**71.94**	69.42
CIFAR-100 (aug)	69.72	71.90	69.82	71.60	72.41	**73.52**	72.01
TinyImagenet	30.73	40.53	39.70	44.91	44.34	**45.68**	42.31

The best result is highlighted in bold

augmentation. The fact that we did not dedicate any time to hyperparameter search also contributed to the low accuracy. This resulted in a network that was not tailored to the data, constituting a "difficult" learning problem. This allows us to examine how the method behaves when there is plenty of margin for further generalisation on a problem. It is also important to note that as the accuracies for some of these data sets are high in absolute terms, the significance of small fluctuations on repeated experiments is also important to consider if they are consistent. We derive significance from the fact that these are average improvements over a number of experiments and that the majority of these experiments have improved results, as can be seen in Table 2. For example, although the mean improvement in MNIST from 99.47 to 99.55% accuracy appears small, it occurs in 9 out of 10 experiments. When focusing on the errors this represents an error reduction from 0.53 to 0.45%, which reflects a mean relative error reduction of 15%. Figure 6 shows a side-by-side comparison of accuracy levels at each round of boosting for both DIB and BRN on the MNIST and CIFAR-100 test sets. This figure illustrates how BRN are able to consistently outperform DIB at each intermediate value of ensemble size, and although such differences would still fall within a Bernoulli confidence interval of 95%, we make the note that this does not take account of the fact that all the random initialisations were aligned, so both methods started with the exact same network. In fact, an additional Friedman Aligned

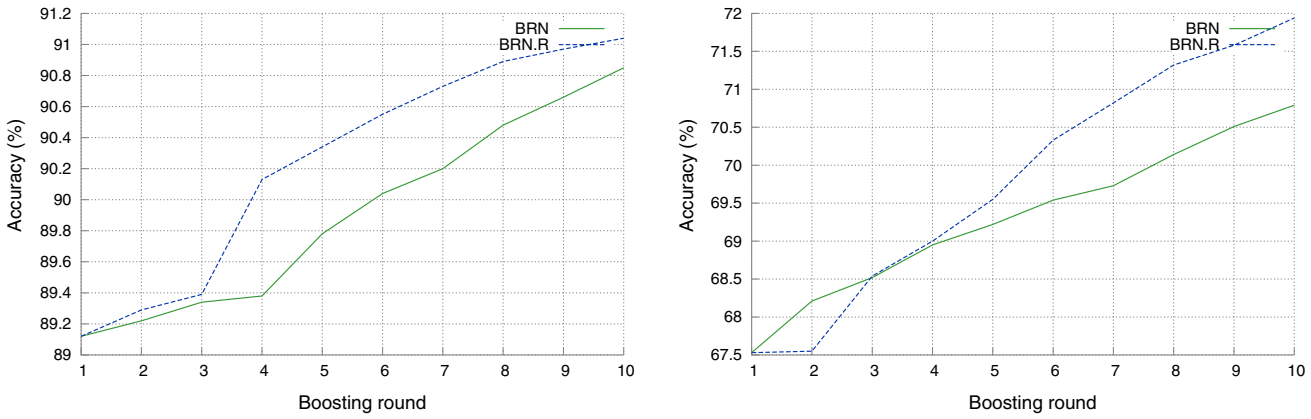
Table 2 The frequency of experimental runs where BRN has the best performance of all methods examined, both in generalisation and training time

	Number of improvements	Number of speed-ups
MNIST	9	10
CIFAR-10	8	10
CIFAR-10 (aug)	9	10
CIFAR-100	10	3
CIFAR-100 (aug)	10	10
TinyImagenet	10	4



(a) MNIST **(b)** CIFAR-100

Fig. 6 Round-by-round comparison of DIB vs BRN on the test set



(a) CIFAR-10 **(b)** CIFAR-100

Fig. 7 Round-by-round comparison of BRN vs BRN.R on the test set

Table 3 Training times comparison, in minutes

	ResNet	Base Net	Bagging	AdaBoost	DIB	BRN	BRN.R	BaRN
MNIST	217	62	437	442	202	**199**	207	209
CIFAR-10	1941	184	1193	1212	461	**449**	453	458
CIFAR-10 (aug)	2228	213	2138	2150	1031	**911**	943	955
CIFAR-100	2172	303	2762	2873	**607**	648	659	676
CIFAR-100 (aug)	2421	328	3044	3072	751	**735**	742	764
TinyImagenet	4804	619	6031	6288	**1591**	1613	1716	1645

Bold values indicate the lowest training times for each dataset

BRN and DIB are the fastest ensemble methods compared

The time to train the individual base network and a ResNet of comparable performance is reported for comparison

Ranks test on the entire group of algorithms tested shows that there is a statistically significant difference in generalisation performance, whilst a direct Wilcoxon test with a null hypothesis that BRN and DIB are sampled from the same distribution shows that BRN is significantly better. In both cases, the "sample" is the average of all experiments with the same characteristics (data set and method), rather than the single experiment run. This is also corroborated by the "number of wins" on each data set (Table 2), and the "number of data sets won" by BRN vs the other methods (Table 1).

Figure 7 shows how BRN.R generally achieves better performance at almost every boosting round. This may be partly because BRN.R is tailored more towards the type of data sets used as benchmarks—the use of the continuous probability output from the CNNs is a big factor.

Table 3 shows that this is achieved without significant changes in the training time.[1] The main speed increase is due to the fact that the only network being trained with a full schedule is the first network, which is also the smallest, whilst all other derived networks are trained for a much shorter schedule (in this case only 10% of the original training schedule). If we exclude the single network, which is clear from a different distribution and only mentioned for reference, a Friedman Aligned Ranks test [42] shows that there is a statistically significant difference in speed between the members of the group, but, as can be expected, a Wilcoxon test [43] between Deep Incremental Boosting and Boosted Residual Networks does not show a significant difference. This confirms what could be conjured from the algorithm itself for BRN, which is of the same complexity w.r.t. the number of ensemble members as DIB. The confirmation that the consistency of improvements is significant, combined with the fact that the method is significantly faster than training the equivalent network from the final round for the full number of epochs, presents an effective strategy for improving performance without requiring additional resources and in less time. The specific time improvement is highly dependent on the number of epochs chosen for the subsequent training rounds $e_t, \forall t > 0$, and the number of boosting rounds t; however, we find empirically that choosing a set of such parameters that keep the total training time low is feasible.

[1] In a few cases BRN is actually faster than DIB, but we believe this to be just noise due to external factors such as system load and affinity of some resulting computational graphs instead of others.

The hardware used to train each network was identical for every case, and because in all cases the ensemble members were trained sequentially, ours was the only work running on the system, providing a sufficiently controlled environment to justify using wall-clock time as a measurement of speed. Table 2 shows that BRN is the fastest method most of the time, whilst Table 3 shows the magnitude of the time improvements, which indicate that the speed improvement on regular ensemble methods is noteworthy and consistent.

Due to the limitations of the current hardware, the residual networks built in our experiments were comparatively smaller than those that achieve state-of-the-art performance, as our biggest residual network in the final round of BRN and BaRN is still orders of magnitude away from the 1001 layers in Ref [1]. The initial network architectures for the first round of boosting are shown in Table 4a for MNIST, and Table 4b for CIFAR-10 and CIFAR-100. It is to be noted that, because of the shortened training schedule and the differing architecture, the results on the augmented data sets are not the same as those reported in the original papers for residual networks. The single networks currently used to reach state of the art on these data sets are very cumbersome in terms of resources and training time [1]. Instead, we used relatively simpler network architectures that were faster to train whilst still performing well on the data sets at hand, with accuracy close to and almost comparable to the state of the art. This enabled us to test larger ensembles within an acceptable training time. Our intention is to demonstrate a methodology that makes it feasible to create ensembles of residual networks following a *customised* approach to significantly improve the training times and accuracy levels achievable with the current ensemble methods.

Training used the WAME method [44], which has been shown to be faster than Adam and RMSprop, whilst still achieving comparable generalisation. This is thanks to a specific weight-wise learning rate acceleration factor that is determined based only on the sign of the current and previous partial derivative $\frac{\partial E(x)}{\partial w_{ij}}$. For the single residual network, and for the networks in AdaBoost, we trained each member for 100 epochs. For Deep Incremental Boosting and all variants of Boosted Residual Networks, we trained the first round for 50 epochs, and every subsequent round for 10 epochs, and ran all the algorithms for 10 rounds of boosting (except for the single network). We chose to use less epochs for the first round because we found empirically that the additional epochs that fine-tuned the base network were not improving the performance at subsequent rounds in any significant way. Because our intention was to find an ensemble method that would train in significantly less time without loss of generalisation, we found that this was an effective strategy. Similarly, we found that above 10 rounds the time to train the ensemble was increasing without large improvements to generalisation.

Table 4 Initial network structures used in experiments

(a) MNIST
64 conv, 5×5
2×2 max-pooling
128 conv, 5×5
2×2 max-pooling*
64 conv, 3×3
Dense, 1024 nodes
50% dropout
(b) CIFAR-10, CIFAR-10, and TinyImagenet
2×96 conv, 3×3
96 conv, 3×3, 2×2 strides
96 conv, 3×3, 2×2 strides
96 conv, 3×3, 2×2 strides
2×2 max-pooling
2×192 conv, 3×3
192 conv, 3×3, 2×2 strides
192 conv, 3×3, 2×2 strides
192 conv, 3×3, 2×2 strides
2×2 max-pooling
192 conv, 4×3
192 conv, 3×3*
192 conv, 3×3
192 conv, 1×1
10 conv, 1×1
Global average pooling
10-way softmax[a]

The layers marked with "*" indicate the location after which we added the new residual blocks at each round of DIB and BRN

Batch normalisation and activation layers are omitted from this diagram for simplicity

[a]For CIFAR-100 this softmax was 100-way, and for TinyImagenet, this softmax was 200-way

The structure of the base network at the first round is shown in Table 4. This was created by taking the shape (strides and number of convolutions) of the existing blocks of ResNet-50 and making the network smaller to create a reasonable starting point that still performed well.

The structure of each additional block added to Deep Incremental Boosting and Boosted Residual Networks at each round is shown in Table 5a for MNIST, and in Table 5b for CIFAR-10, CIFAR-100, and TinyImagenet. The architecture of the ensemble at the Nth round of boosting is shown in Fig. 8.

The choice of additional block was based on the typical structure of a block in residual networks: convolution, followed by batch normalisation, followed by rectified linear units activation. For convenience, we chose to use

Table 5 Structure of blocks added at each round of DIB and BRN

(a) MNIST
64 conv, 3×3
Batch normalisation
ReLu activation
(b) CIFAR-10, CIFAR-100, and TinyImagenet
192 conv, 4×3
Batch normalisation
ReLu activation
192 conv, 3×3
Batch normalisation
ReLu activation

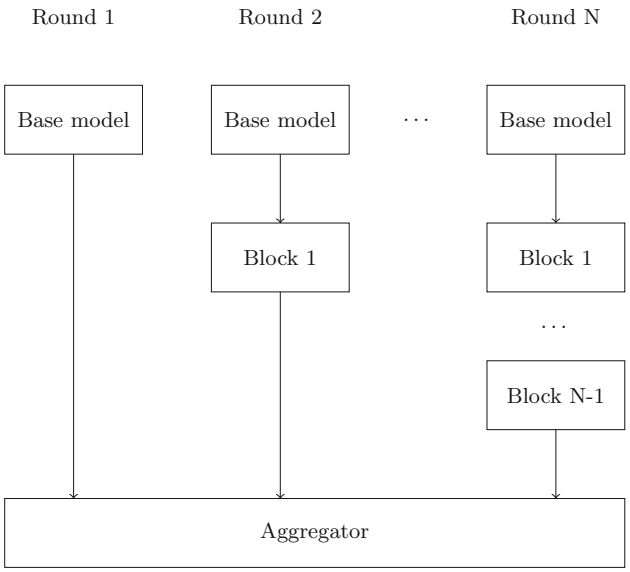


Fig. 8 Visualisation of the structure of the ensemble after N rounds of BRN. The structure of the "Base" blocks is illustrated in Table 4, whilst the size of each additional block is illustrated in Table 5

the same number of filters, shape, and stride as the convolutional layers that each block succeeds. All layers were initialised following the recommendations in [45]. Any additional network hyperparameters are reported in Table 6.

An additional experiment on TinyImagenet with BRN.R and 20 epochs at each round (instead of 10) has an even higher test accuracy of 46.78%, showing that it is possible to fine-tune the number of subsequent epochs as a hyperparameter to obtain better results. We only report this result for completeness, and it was not included in any statistical test.

8.1 Observing unrolled iterative estimation in BRN

Especially for the more complex data sets such as CIFAR-100 and TinyImagenet, the accuracy of the individual classifier improves considerably at each round. We attribute most of this to the fact that, by focusing the training on the newly added block, we are explicitly encouraging the layer-by-layer refinements discussed in the treaty of unrolled iterative estimation [22]. Figure 9 shows the observed accuracy on TinyImagenet at each round.

8.2 Special considerations about BaRN

In Sect. 6 we substituted the boosting algorithm with a simpler bagging algorithm [8] to evaluate whether it would be possible to only use the network from the final round of bagging as an approximation of the ensemble. We called this the Bagged Residual Networks (BaRN) method. When we compare our results to a bagged version of the same base ResNet used as a control for BRN, and the original bagging algorithm, we find that separate Wilcoxon tests refute the hypothesis that the results for BaRN and bagging are sampled from the same distribution, and that BRN and BaRN are sampled from the same distribution, meaning that the differences observed are statistically significant.

Despite the fact that BRN has better performance, the benefits of using BaRN are as follows:

- The reduction in sensitivity to highly imbalance data sets, a known issue for boosting algorithms.
- The potential to derive parallel and distributed implementations which approximate the final ensemble.
- The use of dynamic distortions and transformations of the original data.

Table 6 Hyperparameters used for each network

	Learning rate	Epochs	Batch size
MNIST	10^{-2}	100	64
CIFAR-10	10^{-3}, 10^{-4} after 40 epochs	100	128
CIFAR-10 (aug)	10^{-3}, 10^{-4} after 40 epochs	100	128
CIFAR-100	10^{-3}, 10^{-4} after 40 epochs	100	128
CIFAR-100 (aug)	10^{-3}, 10^{-4} after 40 epochs	100	128
TinyImagenet	10^{-3}, $5 * 10^{-4}$ after 40 epochs	100	128

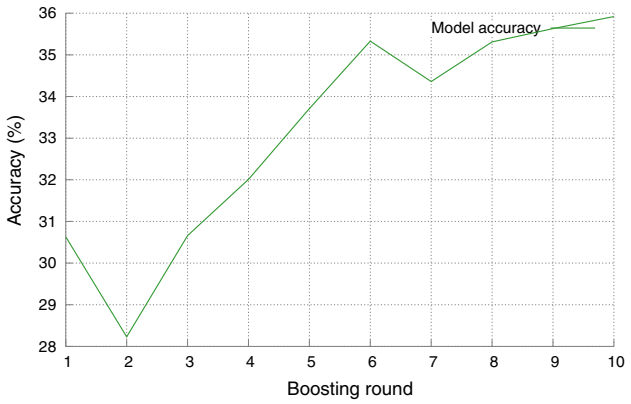


Fig. 9 Single-model test accuracy for each round of BRN on TinyImagenet

8.3 Additional experiments with distillation

In another set of experiments we tested the performance of a Distilled Boosted Residual Network (DBRN) and a Distilled Bagged Residual Network (DBaRN). For the structure of the final distilled network we used the same architecture as that of the residual network from the final round of boosting. Average accuracy results in testing over 10 runs are presented in Table 7, and for completeness of comparison we also report the results for the distillation of DIB, following the same procedure, as DDIB. DBRN does appear to improve results only for CIFAR-10, but it consistently beats DDIB on all data sets. These differences are too small to be deemed statistically significant with a Friedman Aligned Ranks test, confirming the hypothesis that the functions are sampled from the same distribution. It can therefore be said that the function learned by both BRN and DIB can be efficiently transferred to a single network, for the data sets taken under consideration.

Using only the network produced in the last round of BaRN instead of the distilled DBaRN is significantly worse. This is reported as BaRN-l. We therefore cannot simply replace the distillation process by utilising the network created in the last round of BaRN. This also refutes our hypothesis that BaRN could be used as a method for incrementally creating a large residual network.

9 Conclusions and future work

In this paper we introduced a customised methodology for creating ensembles of deep learning models and designed three algorithms that follow this approach, specifically tailored to convolutional networks to generate Boosted Residual Networks and Bagged Residual Networks, and looked at potential variants of those algorithms for real-valued classifiers. We have shown that this surpasses the performance of a single residual network equivalent to the one trained at the last round of boosting, of an ensemble of such networks trained with AdaBoost, and of the equivalent Deep Incremental Boosting on the MNIST, CIFAR-10, CIFAR-100, and TinyImagenet data sets, with and without using common data augmentation techniques.

We then derived and looked at distilled versions of the methods, and how this technique can serve as an effective way to reduce the test-time cost of running the ensemble. We analysed how this compares to the distilled version of the same baselines used in the preceding experiment.

The combination of such techniques has shown that it is possible to train a model that has slightly better generalisation with lower complexity in a significantly shorter amount of time.

Because of the limitations to the network size imposed in our experiments, it might be appealing in the future to evaluate the performance improvements obtained when creating ensembles of large, state-of-the-art, base networks, for example by using the 1001-layer networks found in [1] as a starting network architecture.

The BRN process builds a residual network block by block in additive steps. An investigation on whether this additive process enables the creation of deeper networks by virtue of the *unrolled iterative estimation* principle has been produced, concluding that, although the final classifier has a higher learning capacity than the one produced in the first round, and shows improved learning especially on large data sets, it is not sufficient on its own to replace the entire ensemble. This is likely due to the imbalanced resampling of the training set, and the fact that the contribution from the simpler networks at earlier rounds may serve as a control for overfitting. With BaRN and BRN.R, it is shown that the last classifier is indeed better than the

Table 7 Testing accuracy for distilled variants of the ensembles

	DBRN (%)	DBRN.R (%)	DDIB (%)	DBaRN (%)	BaRN-l (%)
MNIST	99.49	99.50	99.44	99.55	99.35
CIFAR-10	91.11	91.05	90.66	90.77	90.62
CIFAR-10 (aug)	93.28	92.76	92.43	92.68	92.73
CIFAR-100	68.99	68.86	65.91	67.42	66.16
CIFAR-100 (aug)	70.24	70.71	69.18	71.51	70.44
TinyImagenet	42.63	43.70	42.14	39.64	32.92

first. This is encouraging first evidence that the additive construction of large residual networks may be a valid approach, although the performance gained by using the whole ensemble instead is significant. We believe it is, however, still necessary to further investigate this approach and the behaviour of additive training in isolation.

Additional further investigation could also be conducted on the creation of Boosted Densely Connected Convolutional Networks, by applying the same principle to DCCN instead of residual networks.

Another very important property that has not been fully explored in this paper is the recent development of attack and defence methods for adversarial training using ensembles [5]. Whilst we do not investigate the effect of customised ensemble methods on adversarial learning, it is possible to speculate that, either with or without adaptations to the learning set-up, these methods could be used to improve on such a class of problems.

We also believe that there is additional work in exploring how such iterative methods like BRN may be extended to incorporate notions of differential computation in deep learning, such as LM-ResNet and LM-ResNeXt [46], and NAIS-Net [47].

Compliance with ethical standards

Conflict of interest The authors have received a hardware grant from NVIDIA for this research.

References

1. He K, Zhang X, Ren S, Sun J (2015) Deep residual learning for image recognition. arXiv:1512.03385
2. He K, Zhang X, Ren S, Sun J (2016) Identity mappings in deep residual networks. arXiv:1603.05027
3. Schapire RE, Freund Y (1996) Experiments with a new boosting algorithm. In: Machine learning: proceedings of the thirteenth international conference, pp 148–156
4. Dietterich TG (2000) An experimental comparison of three methods for constructing ensembles of decision trees: bagging, boosting, and randomization. Mach Learn 40(2):139–157
5. Tramèr F, Kurakin A, Papernot N, Goodfellow I, Boneh D, McDaniel P (2017) Ensemble adversarial training: attacks and defenses. arXiv:1705.07204
6. Mosca A, Magoulas GD (2018) Distillation of deep learning ensembles as a regularisation method. In: Advances in hybridization of intelligent methods, Springer, pp 97–118
7. Mosca A, Magoulas G (2017) Boosted residual networks. In: EANN. 18th international conference on engineering applications of neural networks
8. Breiman L (1996) Bagging predictors. Mach Learn 24(2):123–140
9. Mosca A, Magoulas G (2016) Deep incremental boosting. In: Benzmuller C, Sutcliffe G, Rojas R (eds) GCAI 2016. 2nd global conference on artificial intelligence. EPiC series in computing, EasyChair, vol 41, pp 293–302
10. Whitley D, Starkweather T, Bogart C (1990) Genetic algorithms and neural networks: optimizing connections and connectivity. Parallel Comput 14(3):347–361
11. Malakooti B, Zhou YQ (1994) Feedforward artificial neural networks for solving discrete multiple criteria decision making problems. Manag Sci 40(11):1542–1561
12. Płaczek S, Adhikari B (2014) Analysis of multilayer neural networks with direct and cross forward connection. Fundam Inf 133(2–3):227–240
13. Bishop C (1995) Neural networks for pattern recognition. Oxford University Press, Oxford
14. Ripley BD (2007) Pattern recognition and neural networks. Cambridge University Press, Cambridge
15. Raiko T, Valpola H, LeCun Y (2012) Deep learning made easier by linear transformations in perceptrons. In: Artificial intelligence and statistics, pp 924–932
16. Schraudolph N (1998) Accelerated gradient descent by factor-centering decomposition. Technical report/IDSIA 98
17. Schraudolph NN (2012) Centering neural network gradient factors. In: Montavon G, Orr GB, Müller KR (eds) Neural networks: tricks of the trade. Springer, Berlin, pp 205–223
18. Vatanen T, Raiko T, Valpola H, LeCun Y (2013) Pushing stochastic gradient towards second-order methods—backpropagation learning with transformations in nonlinearities. In: International conference on neural information processing, Springer, pp 442–449
19. Srivastava RK, Greff K, Schmidhuber J (2015) Highway networks. arXiv:1505.00387
20. Srivastava RK, Greff K, Schmidhuber J (2015) Training very deep networks. In: Advances in neural information processing systems, pp 2377–2385
21. Huang G, Liu Z, Weinberger KQ (2016) Densely connected convolutional networks. arXiv:1608.06993
22. Greff K, Srivastava RK, Schmidhuber J (2016) Highway and residual networks learn unrolled iterative estimation. arXiv:1612.07771
23. Yosinski J, Clune J, Bengio Y, Lipson H (2014) How transferable are features in deep neural networks? In: Advances in neural information processing systems, pp 3320–3328
24. Oquab M, Bottou L, Laptev I, Sivic J (2014) Learning and transferring mid-level image representations using convolutional neural networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 1717–1724
25. Veit A, Wilber MJ, Belongie S (2016) Residual networks behave like ensembles of relatively shallow networks. In: Advances in neural information processing systems, pp 550–558
26. Schapire RE (1990) The strength of weak learnability. Mach Learn 5:197–227
27. Hastie T, Rosset S, Zhu J, Zou H (2009) Multi-class adaboost. Stat Interface 2(3):349–360
28. Mukherjee I, Schapire RE (2013) A theory of multiclass boosting. J Mach Learn Res 14:437–497
29. Freund Y, Iyer R, Schapire RE, Singer Y (2003) An efficient boosting algorithm for combining preferences. J Mach Learn Res 4:933–969
30. Zagoruyko S, Komodakis N (2016) Wide residual networks. arXiv:1605.07146
31. Ba LJ, Caurana R (2014) Do deep nets really need to be deep? In: Advances in neural information processing systems, pp 2654–2662
32. Bucilu C, Caruana R, Niculescu-Mizil A (2006) Model compression. In: Proceedings of the 12th ACM SIGKDD international conference on knowledge discovery and data mining, ACM, pp 535–541
33. Hinton G, Vinyals O, Dean J (2015) Distilling the knowledge in a neural network. arXiv:1503.02531

34. Mosca A, Magoulas GD (2016) Regularizing deep learning ensembles by distillation. In: 6th international workshop on combinations of intelligent methods and applications (CIMA 2016), p 53
35. Benenson R What is the class of this image? http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html. Accessed 6 Dec 2018
36. Lecun Y, Cortes C The MNIST database of handwritten digits. http://yann.lecun.com/exdb/mnist/. Accessed 6 Dec 2018
37. Krizhevsky A, Hinton G (2009) Learning multiple layers of features from tiny images. vol 4, No. 4. Technical report, University of Toronto
38. Russakovsky O, Deng J, Su H, Krause J, Satheesh S, Ma S, Huang Z, Karpathy A, Khosla A, Bernstein M, Berg AC, Fei-Fei L (2015) ImageNet large scale visual recognition challenge. Int J Comput Vis (IJCV) 115(3):211–252
39. Wan L, Zeiler M, Zhang S, Cun YL, Fergus R (2013) Regularization of neural networks using dropconnect. In: Proceedings of the 30th international conference on machine learning (ICML-13), pp 1058–1066
40. Graham B (2014) Fractional max-pooling. CoRR arXiv:1412.6071
41. Clevert D, Unterthiner T, Hochreiter S (2015) Fast and accurate deep network learning by exponential linear units (elus). CoRR arXiv:1511.07289
42. Friedman M (1937) The use of ranks to avoid the assumption of normality implicit in the analysis of variance. J Am Stat Assoc 32(200):675–701
43. Wilcoxon F (1945) Individual comparisons by ranking methods. Biometrics 1(6):80–83
44. Mosca A, Magoulas GD (2017) Training convolutional networks with weight-wise adaptive learning rates. In: ESANN 2017 proceedings, European symposium on artificial neural networks, computational intelligence and machine learning. Bruges (Belgium), 26–28 April 2017, i6doc.com publ
45. He K, Zhang X, Ren S, Sun J (2015) Delving deep into rectifiers: surpassing human-level performance on imagenet classification. In: Proceedings of the IEEE international conference on computer vision, pp 1026–1034
46. Lu Y, Zhong A, Li Q, Dong B (2018) Beyond finite layer neural networks: bridging deep architectures and numerical differential equations. ICLR https://openreview.net/forum?id=ryZ283gAZ. Accessed 6 Dec 2018
47. Ciccone M, Gallieri M, Masci J, Osendorfer C, Gomez F (2018) NAIS-Net: stable deep networks from non-autonomous differential equations. CoRR arXiv:1804.07209