Distillation of Deep Learning Ensembles as a Regularisation method

Alan Mosca[1] and George D Magoulas

Department of Computer Science and Information Systems
Birkbeck, University of London
email: {a.mosca, gmagoulas}@dcs.bbk.ac.uk

Abstract. Ensemble methods are among the most commonly utilised algorithms that construct a group of models and combine their predictions to provide improved generalisation. They do so by aggregating multiple *diverse* versions of models learned using machine learning algorithms, and it is this diversity that enables the ensemble to perform better than any of its members taken individually. This approach can be extended to produce ensembles of deep learning methods that combine various good performing models, which are between them very diverse because they have reached different local minima and make different prediction errors. It has been shown that a large, cumbersome deep neural network can be approximated by a smaller network through a process of distillation, and that it is possible to approximate an ensemble of other learning algorithms by using a single neural network, with the help of additional artificially generated pseudo-data. We extend this work to show that an ensemble of deep neural networks can indeed be approximated by a single deep neural network with size and capacity equal to the single ensemble member, and we develop a recipe that shows how this can be achieved without using any artificial training data or any other special provisions, such as using the *soft output targets* during the distillation process. We also show that, under particular circumstances, the distillation process can be used as a form of regularisation, through its implicit reduction in learning capacity. We corroborate our findings with an experimental analysis on some common benchmark datasets in computer vision and deep learning.

Keywords: ensembles, deep learning, distillation, convolutional neural networks

1 Introduction

A typical trait among almost all of the many available machine learning ensemble methods, of which a survey can be found in [5], is that at the time of testing (sometimes also called "inference time"), the input is run through every single member of the ensemble, in order to obtain the combined result. This makes the operation of the ensemble at test–time expensive when compared to each base classifier taken individually, with both time and memory requirements increasing

Author produced PDF of a contribution published In
Hatzilygeroudis I., Palade V. (eds) Advances in Hybridization of
Intelligent Methods. Smart Innovation, Systems and Technologies,
vol 85. Springer.
The definitive authenticated version is available online via
https://doi.org/10.1007/978-3-319-66790-4_6.

linearly w.r.t. the number of members in the ensemble. Because of this cost relationship to the ensemble size, which is also mirrored in the training phase, ensembles have been usually only used with small base learner algorithms, or as a final addition to an independently fine–tuned, more complex, base learner. This means that deep learning ensembles have been largely overlooked as a field of research because of the high costs involved.

A process called *distillation* has been developed in [8], which shows how a reduced–complexity deep neural network is able to approximate the behaviour of a more *cumbersome* network. This is achieved by using the *soft output values* of the cumbersome network as a training target for the approximate network. In [3], the authors show that a simple feed-forward, fully-connected Artificial Neural Network is able to learn the distilled knowledge of an ensemble of other types of classifiers, demonstrating that the principle of distillation is also applicable to ensembles.

We show that it is possible to apply the distillation method to approximate an ensemble of deep convolutional neural networks with a single network. We do this for two reasons. First, a distilled ensemble will be more portable than the original ensemble, with smaller computational and memory requirements compared to the original version. Second, the improved generalisation achieved with the ensemble may be transferrable to the distilled version.

We provide guidelines that we derived from learning theory, as to what the optimal methodology for such a distillation is, and show that, where the original training data is sufficiently sized, the distillation process also serves as a regularizer of the ensemble as a whole. Our recipe for distillation of ensembles also simplifies some of the prerequisites imposed by existing methods: it is no longer necessary to use the "soft" output as a target for training the distilled network. Instead we can use the classification output as the target and the distillation process still works. Additionally, we reason that any additional artificial data used to cover the input space is only likely to reduce the regularization abilities of the distilled network.

The rest of the paper is structured as follows. Section 2 explores the background and existing literature in deep learning, specifically regarding the methods used in this paper. Section 3 covers the necessary background in ensemble theory and methods related to this work. Section 4 presents background on the distillation process. Section 5 derives and illustrates our recipe for the distillation of deep learning ensembles, and Section 6 shows how our methodology leads to an improvements over using just the ensemble method. Finally, Section 7 discusses our results, and considers how the use of an intermediate ensemble could become an integral part of training a deep neural network.

2 Supervised Deep Learning

Deep Learning is a branch of machine learning applied to the study of very large Artificial Neural Networks (ANNs). Such ANNs are usually termed "deep" because of the large number of hidden layers that they contain. In this paper, we

focus our efforts on Convolutional Neural Networks because of their popularity, but our recipe and experimentation can be easily extended to work with other supervised deep learning algorithms.

2.1 Convolutional Neural Networks

Convolutional Neural Networks [10] were introduced mostly for use in image recognition, or other similar problem domains, where applying a convolution operation to the input features makes sense (Video Analysis, Sound data, Natural Language Processing). These networks are usually trained using backpropagation and gradient descent, and most of the readily available update rules can be applied. Each convolutional layer is composed of the following functional elements, which for simplification of the backpropagation steps, can be considered as separate layers that usually come in sequence:

– Convolution of the input features across multiple kernels, sometimes also called filters, where the function of each kernel is learned. In some cases this is thought of as generating a volume of neurons by extrusion, because each kernel will perform a convolution across the input volume. However, the same kernel will be applied on each of the "depth" layers, which leads to weight-sharing and therefore improves the time required to compute and train the layer. If we consider an $N \times M$ rectangular input from layer $l-1$, to which a $n \times m$ rectangular kernel ω is applied, the forward pass of the convolution operation to yield the value of the output at index i, j would be:

$$z_{ij}^{(l)} = \sum_{a=0}^{n-1} \sum_{b=0}^{m-1} \omega_{ab} y_{(i+a)(j+b)}^{(l-1)} \tag{1}$$

An activation function is applied to calculate each output $y_{ij}^{(l)} = A(z_{ij}^{(l)})$, as with standard neurons. When it comes to the back-propagation of the error present at the output of the convolutional layer $E^{(l)}$, we need to compute the partial derivative with respect to each input $\frac{\partial E^{(l)}}{\partial y_{ij}^{(l-1)}}$. This is done using the chain rule, as per normal back-propagation

$$\begin{aligned} \frac{\partial E}{\partial \omega_{ab}} &= \sum_{a=0}^{n-1} \sum_{b=0}^{m-1} \frac{\partial E}{\partial z_{ij}^{(l)}} \frac{\partial z_{ij}^{(l)}}{\partial \omega_{ab}} \\ &= \sum_{a=0}^{n-1} \sum_{b=0}^{m-1} \frac{\partial E}{\partial z_{ij}^{(l)}} y_{(i+a)(j+b)}^{(l-1)} \\ &= \sum_{a=0}^{n-1} \sum_{b=0}^{m-1} \frac{\partial E}{\partial y_{ij}^{(l)}} A'(z_{ij}^{(l)}) y_{(i+a)(j+b)}^{(l-1)} \end{aligned} \tag{2}$$

then summing over all $z_{ij}^{(l)}$ where ω_{ab} appears. To propagate the error backwards to the previous layer, we utilize the chain rule again:

$$\begin{aligned}\frac{\partial E}{\partial y_{ij}^{(l-1)}} &= \sum_{a=0}^{n-1}\sum_{b=0}^{m-1}\frac{\partial E}{\partial z_{(i-a)(j-b)}^{(l)}}\frac{\partial z_{(i-a)(j-b)}^{(l)}}{\partial y_{ij}^{(l-1)}} \\ &= \sum_{a=0}^{n-1}\sum_{b=0}^{m-1}\frac{\partial E^{(l)}}{\partial z_{(i-a)(i-b)}^{(l)}}\omega_{ab}\end{aligned} \tag{3}$$

– Pooling, or Subsampling, is then applied to the output of the convolution, to reduce the dimensionality of the output. Most commonly the mean or the max functions are applied. If the original back-propagated error on the l-th layer is

$$E^{(l)} = ((W^{(l)})^T E^{(l+1)}) \cdot A'(z^{(l)}) \tag{4}$$

where $E^{(l)}$ is the error term for the l-th layer, W is the weights matrix and $A'(z^{(l)})$ is the derivative of the activation function, we need to first reverse the subsampling, by applying the reverse operation. For instance, for the mean we would just propagate back the error to all inputs to the subsampling:

$$E^{(l)} = upsample((W^{(l)})^T E^{(l+1)}) \cdot A'(z^{(l)}) \tag{5}$$

In practice, this also means that the error propagated backwards from a max-pooling layer is very sparse, because only one of the inputs will receive the error.

Typically, after a number of these convolutional layers, each connected into each other, a number of *fully connected* layers is added, with the specific goal of classifying the input features detected by the convolutional layers.

The "all convolutional neural network" [19], a variant of Convolutional Neural Networks that replaces the pooling layer for dimensionality reduction with a 1×1 convolution, and removes the fully connected layers, has shown improved results on the original method.

3 Ensembles

3.1 Bagging

Bagging (short for "bootstrap aggregating") is a technique that is based on the statistical bootstrapping method, originally introduced in [2], 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 dataset of size Z with re-sampling, and then using each of these bootstraps to train a separate identical base classifier. Ref [2] 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 randomized re-sampling, but because there will be significant overlap

in the training sets, all the members will still have positive correlation. If we consider a learner function $Y = \phi(X, T, c)$ that produces a candidate solution Y on class c for a training set T and input vector X, then the aggregate y_j of these bootstraps, by averaging, would be:

$$y_j(c_j) = \frac{\sum_{n=1}^{N} \phi(X, T_n, c_j)}{N} \tag{6}$$

and the final aggregate label is

$$Y = \underset{c_j \in C}{\operatorname{argmax}}\, y_j(c_j) \tag{7}$$

The reason why this works can be argued as follows, and is originally explained, in Ref. [2]. This argument is initially made for regression problems with a follow-up on classification. Given (x, y) pairs taken from the bootstrapped training set T_n, and $\phi(x, T_n)$ as the learned predictor, the aggregate predictor will be the average over all bootstraps:

$$\phi_A(x) = E_T \phi(x, T) \tag{8}$$

If we take x to be a fixed input value and y an output, then

$$E_T(y - \phi(x, T))^2 = y^2 - 2yE_T\phi(x, T) + E_T\phi^2(x, T) \tag{9}$$

Integrating both sides over the joint (x, y) distribution, we get that the MSE of $\phi_A(x)$ is lower than the MSE averaged over T of $\phi(x, T_n)$. The magnitude of this difference is dependent on the difference between the two sides of:

$$(E_T\phi(x, T))^2 \leqslant E_T\phi^2(x, T) \tag{10}$$

Breiman [2] goes on to highlight the importance of the instability between these two figures, and that there is a crossover point at which the bagged example has worse performance.

3.2 AdaBoost

Boosting is a technique first introduced in [17, 18], by which classifiers are trained sequentially, using a sample from the original dataset, with the prediction error from the previous algorithms 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.

In [18], 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, and they both follow roughly the high-level description of Algorithm 1. Each boosting variant builds a distribution of weights D_t, which is used

Algorithm 1 A generic Boosting meta-algorithm

$D_0 \leftarrow uniform()$
$t \leftarrow 0$
while $notstoppingConditionReached()$ **do**
  $currentSubset \leftarrow pickFromSet(wholeSet, D_t)$
  $h_t \leftarrow newClassifier(currentSubset)$
  $\epsilon_t \leftarrow getError(h_t, wholeSet)$
  $\alpha_t \leftarrow learnerCoefficient(\epsilon_t)$
  $D_{t+1} \leftarrow nextDistribution(D_t, h_t, \alpha_t, wholeSet)$
  $t \leftarrow t + 1$
end while
$H \leftarrow aggregate(h_{0..t}, \alpha_{0..t})$

to sample from the training set, and is updated at each iteration to increase the *importance* of the examples that are harder to classify correctly. The resampled dataset 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 are in how the *getError*, *learnerCoefficient*, *nextDistribution* and *aggregate* functions are implemented.

It has been shown that the requirement of AdaBoost.M1 on maximum error of the underlying weak learner is much stronger than just performing better than random[18]:

> "The main disadvantage of AdaBoost.M1 is that it is unable to handle weak hypotheses with error greater than 1/2. The expected error of a hypothesis which randomly guesses the label is $1 - 1/k$, where k is the number of possible labels. Thus AdaBoost.M1 requirement for k=2 is that the prediction is just slightly better than random guessing. However, when $k > 2$, the requirement of AdaBoost.M1 is much stronger than that, and might be hard to meet."

AdaBoost.M2 solves this problem by introducing additional communication between the weak learner and the boosting algorithm. For each sample x and label y a hypothesis function $h_t(x, y) \rightarrow [0, 1]$ is obtained such that the *pseudo-loss* of a hypothesis can be calculated. We use y to indicate the mislabelled class, and y_i to indicate the currently considered weak learner's output associated with x_i. D_t is the weight distribution on the training examples at iteration t.

$$\epsilon_t = \frac{1}{2} \sum_{(i,y) \in B} D_t(i, y)(1 - h_t(x_i, y_i) + h_t(x_i, y)) \tag{11}$$

to be used in place of the error, where B is the set of all mislabelled pairs (i, y):

$$B = (i, y) : i \in 1, ..., m, y \neq y_i \tag{12}$$

where m is the number of samples and y_i is the label for sample i. It should be noted that some slight modifications will need to be made to standard learners in

order to produce this hypothesis, so that instead of producing a single decision, they can output a probability $P(y|h_t, x) = h_t(x, y)$ of y being the correct label assignment [1]. Subsequently, we set $\beta_t = \epsilon_t/(1 - \epsilon_t)$ and use it to calculate the updated value of D_t:

$$D_{t+1}(i, y) = \frac{D_t(i, y)}{Z_t} \cdot \beta_t^{\frac{1}{2}(1+h_t(x_i,y_i)-h_t(x_i,y)} \tag{13}$$

where Z_t is the normalization constant that is also calculated in AdaBoost.M1.

3.3 Deep Incremental Boosting

Algorithm 2 Deep Incremental Boosting

$D_0(i) = 1/M$ for all i
$t = 0$
$W_0 \leftarrow$ randomly initialised weights for first classifier
while $t < T$ **do**
  $X_t \leftarrow$ pick from original training set with distribution D_t
  $u_t \leftarrow$ create untrained classifier with additional layer of shape L_{new}
  copy weights from W_t into the bottom layers of u_t
  $h_t \leftarrow$ train u_t classifier on current subset
  $W_{t+1} \leftarrow$ all weights from h_t
  $\epsilon_t = \frac{1}{2} \sum_{(i,y)\in B} D_t(i)(1 - h_t(x_i, y_i) + h_t(x_i, y))$
  $\beta_t = \epsilon_t/(1 - \epsilon_t)$
  $D_{t+1}(i) = \frac{D_t(i)}{Z_t} \cdot \beta^{(1/2)(1+h_t(x_i,y_i)-h_t(x_i,y))}$
  where Z_t is a normalisation factor such that D_{t+1} is a distribution
  $\alpha_t = \frac{1}{\beta_t}$
  $t = t + 1$
end while
$H(x) = \text{argmax}_{y\in Y} \sum_{t=1}^{T} log\alpha_t h_t(x, y)$

Deep Incremental Boosting (DIB), introduced in [13], is an example of a *white-box* ensemble method: instead of treating the base learner as a "black-box", characteristics and properties of such model are exploited to improve the results given by the ensemble method. DIB applies concepts from transfer of learning to improve the speed of convergence of boosting rounds by copying a subset of convolutional layers trained at round $t - 1$ as the initialisation for the network to be trained at round t. The new network is also given additional capacity to learn the new resampled dataset and the corrections from the previously trained layers. An illustration of the architecture of DIB is given in Figure 1, and the full algorithm is shown in Algorithm 2.

[1] Fortunately, in the case of Deep Learning algorithms, we find that in most cases the output layer is a SoftMax, which already gives us this value with no additional work.

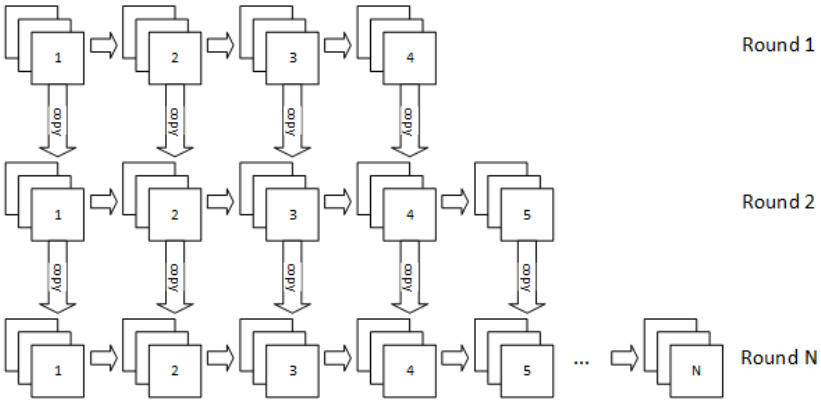

Fig. 1. Illusration of subsequent rounds of DIB

4 Related Work

4.1 Meta–Models

In the field of simulation, the concept of a *meta–model* is not a novel one. A meta–model is a model that is designed to reproduce the effects of another model.

This "second–order" modelling has several advantages. It may be the case that the original model cannot be used to make predictions, or it may be too complex to fully evaulate, so a meta–model can be developed to obtain similar results with lesser cost. It may be that some of the characteristics of the original model are not measurable, so that it cannot be completely reproduced.

A number of surveys has been produced on the subject of meta–modelling, for example [21]. The meta–models that are more closely related to the work in this paper are those that use Artificial Neural Networks as the second–order model [1, 6]. These meta–models typically have been trained on small datasets and the ensemble member is also usually a model with relatively small capacity. The distilled ANN cannot be considered deep enough to be representative of modern deep learning.

4.2 Approximating an ensemble with a Neural Network

In [22] it has been shown that it is possible to use a neural network to approximate an ensemble. However, whilst the results from the approximate network are better than training the network from scratch, and they approximate very well the results produced by utilising bagging, the ANNs used are also shallow and very small compared to today's deep learning models. This method makes use of a *pseudo-training set* which is generated from the original training set and the outputs of the ensemble.

An improvement to this methodology has been developed by [3]. Instead of using a sample of the original training set to generate the pseudo-training set, the authors sample the entire input space S. This approach is then evaluated on trying to approximate bagged ensembles of SVMs and numerous variants of decisions trees. The generation of artificial data points in the sample space appears to work well for small datasets with low complexity. However, as we will see, for high-dimensional datasets such as those used in deep learning, it poses a few problems of its own.

4.3 Distillation in Deep Learning

Work has been done towards showing that it is possible to distil the knowledge in an ensemble of relatively deep neural networks [8]. In the paper presenting this work, a similar argument as in [22] is made: the true function underpinning the data is not known and therefore it is impossible to learn to generalise it perfectly using only what is represented by the training data, so a *proxy* function has to be used – in this case the output of the cumbersome model. This gives credit to the idea of generating a pseudo-training dataset to extract the knowledge of said proxy function, which in this case is to be created using the inputs of the original training set (or a specially dedicated "transfer set") and the *soft* targets for those given inputs. For a single deep neural network, this equates to the output of the Softmax layer, before any argmax operation is applied. In the case of ensembles, this is extended as the geometric or algebraic mean of the soft outputs of each ensemble member. It is argued that doing so serves as a regulariser.

This research is focused on relatively small deep networks (up to 8 layers), and without any of the additional architectural characteristics that are typically found in contemporary deep learning systems. Additionally, two of the experiments provided are not reproducible, as they are conducted on ad-hoc, unpublished, proprietary datasets (Google JFT and and unnamed Android development set). On MNIST, the only results given are for a small network, used to distill a larger network. The large network reports an error of 0.67%, the small one 1.46% and the distilled version 0.74%, indicating that the process of distillation is useful to improve the performance of the smaller network, but it does not beat the performance of the cumbersome one. An attempt is then made to utilise ensembles in the distillation process. However, the authors make use of "specialist models", where each member of the ensemble has received specialised training on a specific class or task, and no consideration is made for the general case of traditional ensemble methods.

We therefore conclude that, although it is seminal work in proving that distillation of deep learning models is possible, it is still necessary to extend the experimentation to more modern supervised deep learning architectures, to network sizes that are similar to those currently being used, and, most importantly, to non-specialised ensembles.

Additional work has been done when looking at distillation as a tool, especially with regards to *adversarial examples* [7, 15, 16]. Such examples are generated in a way that a small perturbation of the input, which is normally imper-

ceptible to the human eye, causes a large change in the outputs of a network. It is shown that distillation reduces the magnitude and count of input gradients that create these adversarial examples, which serves as a good indication that the process of distillation is very effective at removing the superfluous perturbations of the learned function, by simplifying it and regularising it.

Another variant of distillation has been developed, called FitNet [**?**] , which specifically trains a distilled network that has a larger number of smaller layers than its teacher. In this case, we can see that a teacher network that has 9.84% error on CIFAR-10, can be used to train a FitNet network that reaches 8.39% error on the same test set.

5 Distillation of Deep Learning Ensembles

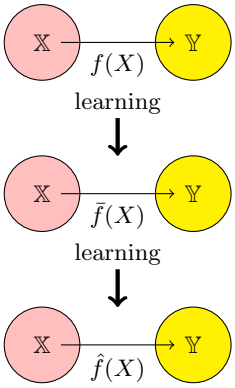

Fig. 2. A schematic representation of Distillation

We begin by formally defining the process of distillation.

Definition 1. *Let S be the (unknown) manifold of the feature space $\mathbb{S}$ where some data X exists.*

Let Y be the set of labels corresponding to the data X.

Let $\mathbb{Y}$ be the output space in which the data is labelled.

Let $Y = f(X) : \mathbb{S} \to \mathbb{Y}$ be the (unknown) ground truth *function that correctly labels the data, which is the target of the learning.*

Let D be the (unknown) distribution of the data X in the manifold S.

Let $\bar{D}$ be an approximate estimation of the distribution D.

Let $h(x)$ be a model hypothesis that is designed to learn the function $f(x)$.

Let $\bar{Y} = \bar{f}(X) : \mathbb{S} \to \mathbb{Y}$ be the approximate *function that is instead learned by the model $h(X)$.*

Let $\bar{X}$ be any additional synthetic training data sampled from $\bar{D}$ and $\bar{f}(\bar{X})$ its predicted labels by the model $h(X)$.

Definition 2. *Then we establish that it is possible to derive a* distilled *model* $h'(X)$ *which learns from data* $\left(\bar{X}, \bar{f}(\bar{X})\right)$ *a new function* $\hat{f}(X) : \mathbb{S} \rightarrow \mathbb{Y}$ *which is itself an approximation of* $\bar{f}(X)$*.*

Figure 2 represents graphically how each function is learned from its ancestor, mapping the same two spaces in different manners. Because the functions $f(X)$, $\bar{f}(X)$ and $\hat{f}(X)$ are by definition increased orders of approximation of one another, it is not possible to assume that they all cover the same manifold of the spaces $\mathbb{S}$ and $\mathbb{Y}$. We therefore identify the manifolds as:

- S and Y for $f(X)$
- $\bar{S}$ and $\bar{Y}$ for $\bar{f}(X)$
- $\hat{S}$ and $\hat{Y}$ for $\hat{f}(X)$

Proposition 1. *X is the best available representation of the feature space manifold S at training time.*

Because the distribution D is unknown, any estimation of it will be an approximation and inevitably

$$\bar{D} - D \neq \varnothing \tag{14}$$

$$\bar{D} - D \notin S \tag{15}$$

This implies that any new synthetic example $\bar{x}$ may be possibly drawn from $\bar{D} - D$, and that therefore there is a non-zero probability $P(\bar{x} \notin S) > 0$ of $\bar{x}$ being sampled outside of the manifold S.

Proposition 2. *Any additional training example $\bar{x}$ sampled from $\bar{D} - D$ does not provide any additional information to improve the generalisation from $\hat{f}(X) \rightarrow f(X)$.*

We know that $\bar{f}(X) \neq f(X)$, because otherwise the learning for the particular problem would be solved. It follows that any individual samples $\bar{x}$ taken outside of S will have no use for the representation of $f(X)$ when training $\hat{f}(X)$, because these regions are outside the domain of $f(X)$.

Proposition 3. *As the manifold S becomes smaller with respect to the feature space $\mathbb{S}$, the data is more sparse, and therefore $P(\bar{x} \notin S)$ increases.*

Paradoxically, for an infinitely small manifold S, we know that all the synthetic data will lie outside of S, and therefore represent values of $\bar{f}(X)$ that are not defined in $f(X)$. Therefore, although the synthetic dataset $\bar{X}$ will improve the ability of generalising $\bar{f}(X)$ from $\hat{f}(X)$, it will have a higher likelihood of containing information with a higher deviation from the original target function $f(X)$.

In the case of common problems in Deep Learning, we know S to be very small compared to $\mathbb{S}$, therefore making the additional purely synthetic data superfluous, and in some cases potentially harmful to the learning. This is in contrast with noise–injection and data–augmentation, which use existing training

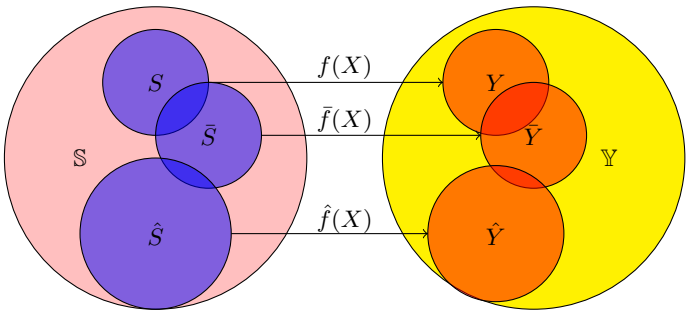

Fig. 3. A schematic depiction of how manifolds might be overlapping

examples as a starting point. In these cases, the augmented examples can also be considered as part of S because they represent classifiable data. In practical terms, it is easy to imagine that, when training a recognizer of hand-written digits, not all the values of the feature space of all the possible pixel value combinations in an image of the same dimension make sense as a training example. In fact, it makes no sense semantically to try and categorize most of the possible resulting images as a digit.

Moreover, if the new training set for the small network is generated solely from the examples on which the Ensemble is trained, other areas of the feature space $\mathbb{S}$ in which the distribution D does not exist, and the approximate function $\bar{f}(X)$ is overfitting, are explicitly avoided.

It is therefore possible to consider the smaller model as a regularised model, because it is not forced to learn these overfitted regions, allowing it to learn a *simpler* approximated function. Because of this, any additional sample $\bar{x}$ is superfluous and can even be harmful.

Because the given definition of distillation uses $h(X)$ as the target output, the proposed approach does not make use of soft target probabilities and does not create additional synthetic training data. This recipe relies solely on the assumption that the original Ensemble is likely to have overfit the training data. The soft target probabilities are not used as simplification to the procedure. We found empirically that with ensembles of deep convolutional neural networks the difference was small enough to justify dropping this practice in favour of simplifying the methodology.

By obtaining the labels produced by the Ensemble on the training set X_{train} and using them to train a new classifier of the same type and shape as those used as members of the Ensemble (Algorithm 3), it is possible to construct a regularised version of the approximated function $\bar{f}(X)$, which is learned by a model $h(X)$.

The validation and testing of the model is performed on the original validation and test sets. This ensures that the distilled model's learned function $\hat{f}(X)$ is evaluated on how well it approximates $f(X)$– the (unknown) function that

correctly labels the data, which is the target of the learning – rather than $\bar{f}(X)$, which was the original goal of the learning process.

Algorithm 3 Regularized Distillation

$H(X, Y)$ = generator of deep networks
$E(H, X, Y)$ = ensemble method
$h_E(X, Y) \leftarrow E(H, X_{train}, Y_{train})$
$y_h(X_{train}) \leftarrow h_E(X_{train}, Y_{train})$
$h_D(X, y_h(X)) \leftarrow H(X_{train}, y_h(X)$
evaluate $h_D(X, y_h(X))$ on the original validation and testing sets (X_{valid}, Y_{valid}) and (X_{test}, Y_{test})

Although in theory it is possible to utilise any architecture for the distilled network, it is practical to use the same architecture originally used for the base classifier. This avoids the need to fit additional hyperparameters and the architecture is already known to be suitable to the problem. It also serves to demonstrate the point that improved learning can be achieved with no additional capacity.

6 Experimental Study

We report the experimental results with convolutional neural networks on benchmark datasets in computer vision. We have compared the distillation of Bagging, AdaBoost and Deep Incremental Boosting. We have included a mixture of over– and under–fitting network, so as to illustrate that when a network is overfitting, distillation also acts as a regulariser.

Each time the experiment was run, the initialisations for each network were aligned by keeping a fixed random seed. This allows a direct comparison between the ensemble methods and their distilled counterparts, given that the single network is initialised identically each time. The values reported are for the median result from five separate runs. The goal of this experimental study is not to achieve a new state–of–the–art performance on the datasets under examination, but to demonstrate how our methodology can be applied to improve a well–performing network, that is close to state–of–the–art.

All experiments were run with the Toupee Deep Learning Ensemble experimentation toolkit. This is a set of libraries and tools, based on Keras [4], which allow the creation of repeatable experiments in deep learning, including ensemble methods. An experiment is described by a "model file", which contains the entire description of the architecture of the network, and an "experiment file", which contains all the information needed to train and test the network, and if applicable, the ensemble. Source code for Toupee is available at http://github.com/nitbix/toupee.

6.1 MNIST

MNIST [11] is a common computer vision dataset that associates 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 grayscale, and the outputs are the numerical value between 0 and 9.

MNIST is generally considered a solved problem, with some relatively simple deep learning models reaching 99.79% accuracy of the best model on the test set with the appropriate data augmentation [20] and longer training. However, we have used this dataset to show an example of how an ensemble of relatively small models that has already slightly overfit the data can be approximated by our regularised distillation. Figure 4 shows how even such a simple convolutional network is already able to start overfitting the training set, by reaching 0% classification error on the training set within four epochs, whilst the test error starts increasing after epoch 14.

We note from Table 1 that the distilled classification errors are better than the ensemble for each of the methods considered, and also better than the original single network (0.66%), further corroborating our hypothesis that the distillation process can serve as a regulariser.

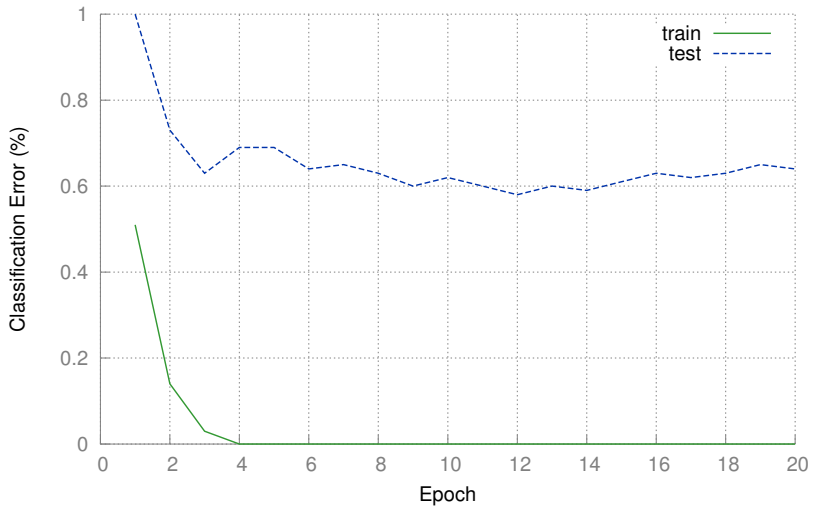

Fig. 4. Non-augmented training and test error on MNIST

The Convolutional Neural Network–CNN used for MNIST has the following structure. We used WAME [14] as the training method, and trained for 20 epochs. We found that training on MNIST for additional time did not improve generalization. We used the training, validation and test sets provided, and commonly used in the literature.

	AdaBoost	Bagging	DIB
Ensemble	0.63%	0.59%	0.51%
Distilled	0.52%	0.55%	0.49%

Table 1. Test misclassification error on MNIST (median of five experiments)

- An input layer of 784 nodes
- 64 5×5 convolutions
- 2×2 max-pooling
- 128 5×5 convolutions
- 2×2 max-pooling
- A fully connected layer of 1024 nodes
- Dropout with $P(D) = 0.5$
- a Softmax layer with 10 outputs (one for each class)

6.2 CIFAR-10

CIFAR-10 is a dataset that contains 60000 small images of 10 categories of objects. It was first introduced in [9]. 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.

The network structure that we used for CIFAR-10 is a variant of the network used in [19], with the following layers. We note that we replaced some of the all-convolutional layers for dimensionality reduction with traditional max-pooling layers, to reduce the required training times. We used WAME [14] as the training method. The training lasted 100 epochs, and, as is practice with the CIFAR datasets, did not use a validation set.

- An input layer of $3 \times 32 \times 32$ nodes
- 96 3×3 convolutions, with 1×1 padding
- 96 3×3 convolutions, with 1×1 padding
- 2×2 max-pooling
- Dropout with $P(D) = 0.5$
- 192 3×3 convolutions, with 1×1 padding
- 192 3×3 convolutions, with 1×1 padding
- 2×2 max-pooling
- Dropout with $P(D) = 0.5$
- 192 3×3 convolutions, with 1×1 padding
- 192 3×3 convolutions, with 1×1 padding
- 192 1×1 convolutions, with 1×1 padding
- 10 1×1 convolutions, with 1×1 padding
- Dropout with $P(D) = 0.5$
- Global average pooling
- a Softmax layer with 10 outputs (one for each class)

We first trained our model with no dataset augmentation, running the experiment five times. The single network obtained a median classification error of 11.15%. This is higher than the original all-convolutional network, which is reported as 9.08%. The principal reasons for the difference are:

- our dataset was not preprocessed with ZCA whitening and Global Contrast Normalization.
- the shorter and more aggressive training schedule, that we used in order to be able to run ensembles of 10 models in an acceptable time.
- the original paper is not reporting the median of five runs, but appears to just be reporting the best results

We were able to recreate a result that was similar to the original reported values when using the full training schedule, but given the long time required to train we were not able to incorporate this into the ensembles. This however, means that the network is underfitting the data, and therefore it is also likely that the ensemble will be doing the same. This is corroborated by the graph in Figure 5, which shows that even at the final epochs of training there is still improvement in the test error.

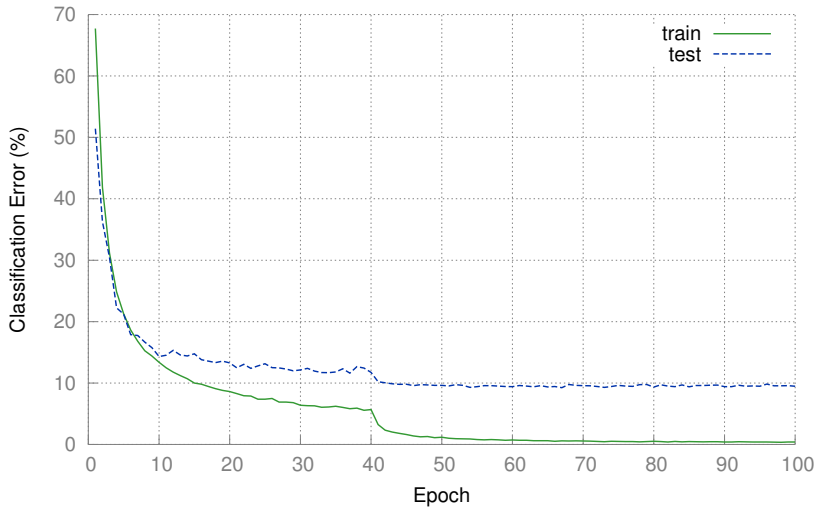

Fig. 5. Non-augmented training and test error on CIFAR-10

Each ensemble method was run with 10 members, then the final resulting ensemble was distilled according to the recipe provided in Section 5. The results without any data augmentation are reported in Table 2. We then repeated the same experiment with data augmentation, as prescribed in [12]. Results for the augmented data are reported in Table 3. In this case our single network reaches a median error of 10.07%

	AdaBoost	Bagging	DIB
Ensemble	9.62%	8.91%	9.39%
Distilled	9.15%	9.31%	9.34%

Table 2. Test misclassification error on non-augmented CIFAR-10 (median of five experiments)

	AdaBoost	Bagging	DIB
Ensemble	7.64%	6.69%	7.16%%
Distilled	7.14%	8.14%	7.11%%

Table 3. Test misclassification error on augmented CIFAR-10 (median of five experiments)

We believe that, while Bagging might still be able to improve the generalisation and is therefore slightly underfitting (despite achieving the best results of the three ensemble methods), Deep Incremental Boosting and AdaBoost are able to overfit the data, because of the increase in learning capacity added at each round and additional emphasis on "hard-to-classify" examples. This is corroborated by the graph in Figure 6, which shows how subsequent rounds of DIB are not improving the performance of the ensemble on the test set.

We note how the process of distillation, once again, has been able to improve the generalisation results for those ensembles that were overfitting, while, although it wasn't able to improve the results for bagging, it is still returning considerably improved results when compared to the single network.

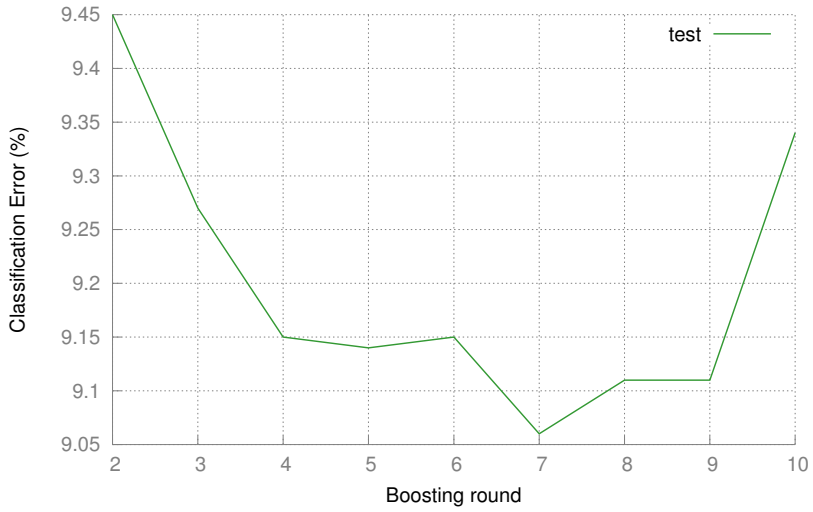

Fig. 6. Rounds of DIB on augmented CIFAR-10 show overfitting

6.3 CIFAR-100

CIFAR-100 is a dataset that contains 60000 small images of 100 categories of objects, grouped in 20 super-classes. It was first introduced in [9]. 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*).

The network used was identical to that used to classify the CIFAR-10 dataset, except for the number of filters in the last 1×1 convolution and the number of outputs of the Softmax, which were both increased to 100 to reflect the change in the number of output classes. We also used the same training parameters.

We first trained our model with no dataset augmentation. The single network obtained a median classification error of 39.48%. This is again higher than the original all-convolutional network, which is reported as 33.71%. We believe that the reason for not obtaining the same error results as reported in the original paper are the same as those we reported for CIFAR-10. Additionally, the shorter traning schedule allowed for a slight underfitting of the single network. This is corroborated by the graph in Figure 7, which shows that even at the final epochs of training there is still improvement in the test error.

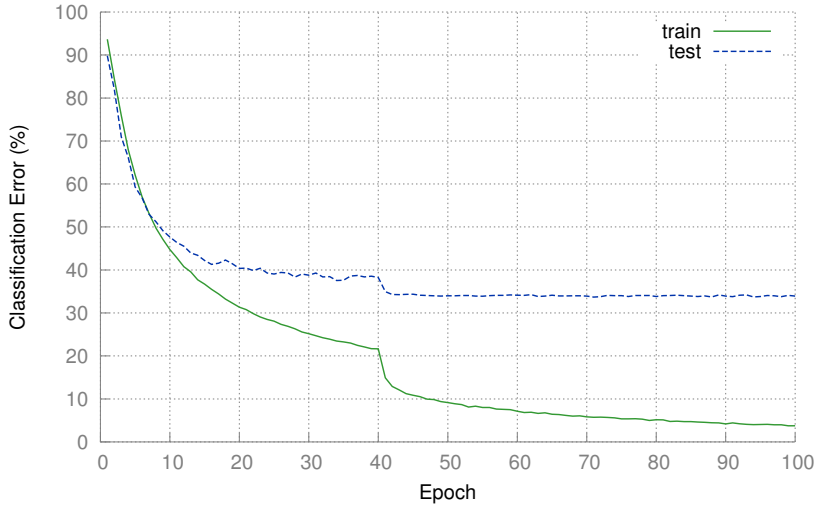

Fig. 7. Non-augmented training and test error on CIFAR-100

Each ensemble method was run for 10 rounds, then the final resulting ensemble was distilled according to the recipe provided in Section 5. The results without any data augmentation are reported in Table 4. We then repeated the same experiment with data augmentation, as prescribed in [12]. Results for the

augmented data are reported in Table 5. The single network with augmentation reaches an error of 32.76%

We believe that, unlike with CIFAR-10, the ensemble methods have not been able to overfit the data. This is corroborated by the graph in Figure 8, which shows how subsequent rounds of DIB are improving the performance of the ensemble on the test set, suggesting that if there were further rounds, the performance would continue to improve.

We note how, even though we are not improving on the results of the ensemble, the distillation process does improve the results when compared to the single network, capturing almost all of the gains given by the ensemble.

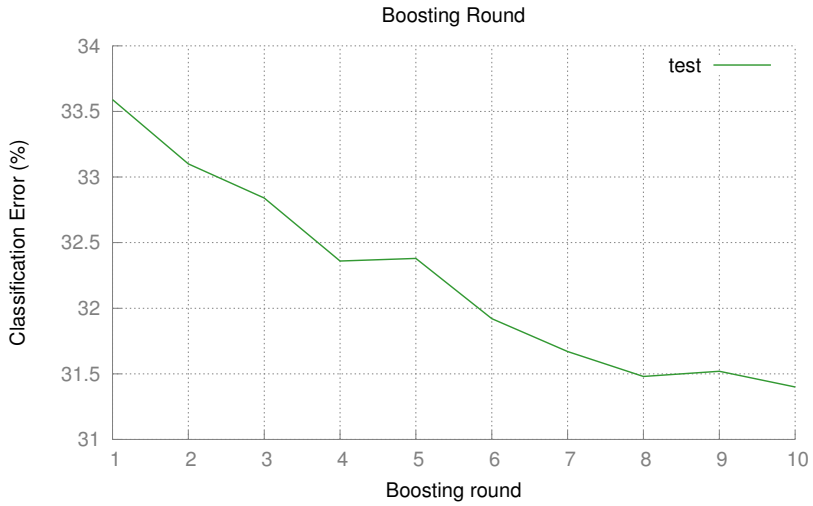

Fig. 8. Rounds of DIB on augmented CIFAR-100 show no overfitting

	AdaBoost	Bagging	DIB
Ensemble	32.23%	31.41%	31.40%
Distilled	33.66%	34.12%	33.69%

Table 4. Test misclassification error on non-augmented CIFAR-100 (median of five experiments)

We then repeated the same experiment with dataset augmentation: random horizontal flips, random zoom and tilt, but no random cropping. We still did not use ZCA whitening or Global Contrast Normalisation. The results in Table 5 show that a similar conclusion to the non-augmented version of the experiment can be made: because the Ensemble is not overfitting, the distillation process

	AdaBoost	Bagging	DIB
Ensemble	31.49%	28.58%	30.38%
Distilled	32.56%	31.02%	31.32%

Table 5. Test misclassification error on augmented CIFAR-100 (median of five experiments)

is regularising a function that doesn't need to be regularised. We note that, although the distilled network has worse performance than the ensemble methods, it still has better performance than te original network (33.71%).

7 Discussion and Conclusions

We have seen how the process of distillation of a cumbersome model can be extended to the representation of the function learned by an ensemble of deep neural networks with a single deep neural network. We have reported some guidelines that, when appropriate, also show how this process can be utilised as a regulariser for the function learned by the ensemble.

We have explored experimentally how our methodology works, both on overfitted and underfitted networks, showing that in all cases we are able to improve the performance of the single network by distilling the knowledge of the ensemble. Furthermore, when the ensemble is overfitting the training data, we are able to use the distillation process as a regulariser, improving the generalisation.

Based on the data from all experiments, a single-tailed Wilcoxon test confirms that, at the 5% confidence level, the improvements in performance are significant. The observed empirical evidence when comparing the distilled network to the original single network shows that the distilled network has lower error every time. A single-tailed Wilcoxon test also confirms that the distilled network is significantly better at the 5% confidence level. Because of the results of these two tests, we can conclude that, in the cases where the computational capacity is not sufficient to run the ensemble at test time (for example those situations where the model needs to be run at scale on commodity hardware), it is always better to use the distilled network.

In terms of the regularisation ability of the distillation process, our experiments provide evidence that distillation works best in cases where the original ensemble is overfitting the data. In such situations, applying distillation improves the accuracy even further. If we imagine that a single member network can learn a function of complexity C_n, we can assume that the overall learning capacity for an ensemble of size n will be $C_e \approx nC_n$. The process of distillation subsequently reduces the complexity of the function learned back to C_n. However, because of the changed learning process explained in Section 5, the function being learned is more effective and is able to reduce the overfitting learned by the ensemble.

Lastly, another finding of this work is that utilising a distilled ensemble, even in situations where the distillation does not provide good regularisation, can still produce improvements over the original single network. We therefore can easily

envision situations where training an ensemble and distilling its knowledge back to a single network could become an integral part of training the single network. In these cases, the single network remains the goal of the training process, and the ensemble becomes a training vehicle for the improvement of the generalisation.

References

1. Badiru, A.B., Sieger, D.B.: Neural network as a simulation metamodel in economic analysis of risky projects. European Journal of Operational Research 105(1), 130–142 (1998)
2. Breiman, L.: Bagging predictors. Machine Learning 24(2), 123–140 (1996)
3. Bucilu, C., Caruana, R., Niculescu-Mizil, A.: Model compression. In: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 535–541. ACM (2006)
4. Chollet, F.: keras. https://github.com/fchollet/keras (2015)
5. Dietterich, T.: Ensemble methods in machine learning. In: Multiple Classifier Systems, Lecture Notes in Computer Science, vol. 1857, pp. 1–15. Springer Berlin / Heidelberg (2000)
6. Fonseca, D., Navaresse, D., Moynihan, G.: Simulation metamodeling through artificial neural networks. Engineering Applications of Artificial Intelligence 16(3), 177–183 (2003)
7. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. arXiv preprint arXiv:1412.6572 (2014)
8. Hinton, G., Vinyals, O., Dean, J.: Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531 (2015)
9. Krizhevsky, A., Hinton, G.: Learning multiple layers of features from tiny images (2009)
10. LeCun, Y., Bengio, Y.: Convolutional networks for images, speech, and time series. The handbook of brain theory and neural networks 3361, 310 (1995)
11. Lecun, Y., Cortes, C.: The MNIST database of handwritten digits http://yann.lecun.com/exdb/mnist/
12. Lin, M., Chen, Q., Yan, S.: Network in network. arXiv preprint arXiv:1312.4400 (2013)
13. Mosca, A., Magoulas, G.: Deep incremental boosting. In: Benzmuller, C., Sutcliffe, G., Rojas, R. (eds.) GCAI 2016. 2nd Global Conference on Artificial Intelligence. EPiC Series in Computing, vol. 41, pp. 293–302. EasyChair (2016)
14. Mosca, A., Magoulas, G.D.: 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. (2017), (in press)
15. Papernot, N., McDaniel, P., Goodfellow, I., Jha, S., Celik, Z.B., Swami, A.: Practical black-box attacks against deep learning systems using adversarial examples. arXiv preprint arXiv:1602.02697 (2016)
16. Papernot, N., McDaniel, P., Wu, X., Jha, S., Swami, A.: Distillation as a defense to adversarial perturbations against deep neural networks. In: Security and Privacy (SP), 2016 IEEE Symposium on. pp. 582–597. IEEE (2016)
17. Schapire, R.E.: The strength of weak learnability. Machine Learning 5, 197–227 (1990)

18. Schapire, R.E., Freund, Y.: Experiments with a new boosting algorithm. Machine Learning: proceedings of the Thirteenth International Conference pp. 148–156 (1996)
19. Springenberg, J.T., Dosovitskiy, A., Brox, T., Riedmiller, M.: Striving for simplicity: The all convolutional net. arXiv preprint arXiv:1412.6806 (2014)
20. Wan, L., Zeiler, M., Zhang, S., Cun, Y.L., Fergus, R.: Regularization of neural networks using dropconnect. In: Proceedings of the 30th International Conference on Machine Learning (ICML-13). pp. 1058–1066 (2013)
21. Wang, G.G., Shan, S.: Review of metamodeling techniques in support of engineering design optimization. Journal of Mechanical design 129(4), 370–380 (2007)
22. Zeng, X., Martinez, T.R.: Using a neural network to approximate an ensemble of classifiers. Neural Processing Letters 12(3), 225–237 (2000)