SN Computer Science (2023) 4:85
https://doi.org/10.1007/s42979-022-01505-2

ORIGINAL RESEARCH

Combining Deep Learning with Good Old-Fashioned Machine Learning

Moshe Sipper[1]

Received: 4 March 2022 / Accepted: 10 November 2022 / Published online: 8 December 2022

Abstract
We present a comprehensive, stacking-based framework for combining deep learning with good old-fashioned machine learning, called Deep GOld. Our framework involves ensemble selection from 51 retrained pretrained deep networks as first-level models, and 10 machine-learning algorithms as second-level models. Enabled by today's state-of-the-art software tools and hardware platforms, Deep GOld delivers consistent improvement when tested on four image-classification datasets: Fashion MNIST, CIFAR10, CIFAR100, and Tiny ImageNet. Of 120 experiments, in all but 10 Deep GOld improved the original networks' performance.

Keywords Machine learning · Deep learning · Image analysis · Pattern recognition

Introduction

The rapid rise of artificial intelligence (AI) in recent years has been accompanied (and enabled) by staggering advances both in software and hardware technologies. Tools, such as PyTorch [1] for deep learning (DL), scikit-learn for machine learning (ML) [2, 3], and graphics processing unit (GPU) hardware, all enable faster and better prototyping and deployment of AI systems than was possible a mere half-decade ago.

While deep learning has taken the world by storm, often—it would seem—at the expense of other computational paradigms, these (plentiful) latter are still quite alive and kicking. We propose herein to revisit stacking-based modeling [4], but within a comprehensive framework enabled by modern state-of-the-art software packages and hardware platforms.

As previously argued by Ref. [5, 6], significant improvement can be attained by making use of models we are already in possession of anyway, through what they termed "conservation machine learning": conserve models across runs, users, and experiments—and make use of all of them. Herein, focusing on image-classification tasks, we ask whether, given a (possibly haphazard) collection of deep neural networks (DNNs), can the tools at our disposal—specifically, "good old-fashioned" ML algorithms, many of which have been around for quite some time—help improve prediction accuracy.

To wit, can we combine DL and ML in a manner that improves DL performance? We answer positively, with a major novelty being the use of the most DL and ML models to date within a single, comprehensive framework.

The section "Previous Work" discusses related previous work. The section "Deep GOld: Algorithmic Setup" describes Deep GOld—Deep Learning and Good Old-Fashioned Machine Learning—which employs 51 deep networks and 10 ML algorithms. The section "Results" presents the results of 120 experiments over four image-classification datasets: Fashion MNIST, CIFAR10, CIFAR100, and Tiny ImageNet. We end with a discussion in the section "Discussion" and concluding remarks in the section "Concluding Remarks".

This article is part of the topical collection "Advances in Applied Image Processing and Pattern Recognition" guest edited by K. C. Santosh.

✉ Moshe Sipper
sipper@bgu.ac.il

[1] Department of Computer Science, Ben-Gurion University, Beer Sheva 84105, Israel

Previous Work

There are many works that involve some form or other of ensembling several models, and this section does not serve as a full review, but focuses on those papers found to be most relevant to our topic.

In an early work, [7] presented a technique called Addemup that uses a genetic algorithm to search for an accurate and diverse set of trained networks. Addemup works by creating an initial population of networks, then evolving new ones, keeping the set of networks that are as accurate as possible while disagreeing with each other as much as possible. They tested these on three DNA datasets of about 1000 samples.

A few years later, [8] presented an approach named Genetic Algorithm-based Selective ENsemble (GASEN) to select some neural networks from a pool of candidates, and assign weights to their contributions to the resultant ensemble. The networks had one hidden layer with five hidden units. The efficacy of this method was shown for regression and classification problems over structured (non-image) datasets of a few thousand samples. Another work by [9] studied financial-decision applications, wherein a neural-network ensemble prediction was similarly reached by weighting the decision of each ensemble member.

A more recent example (one of many) of straightforward ensembling is given in [10], who presented an ensemble neural-network model for real-time prediction of urban floods. Their ensemble approach used a number of artificial neural networks with identical topology, trained with different initial weights. The final result of maximum water level was the ensemble mean. Ensemble sizes examined were 1, 5, and 10.

In a similar vein, [11] trained multiple neural networks and combined their outputs using three combination strategies: simple average, weighted average, and what they termed a meta-learner, which applied a Bayesian regulation algorithm to the network outputs. The application field considered was real-time production monitoring in the oil and gas industry, specifically, virtual flow meters that infer multiphase flow rates from ancillary measurements, and are attractive and cost-effective solutions to meet monitoring demands, reduce operational costs, and improve oil recovery efficiency.

Ref. [12] trained five convolutional neural networks (CNNs) to detect ankle fractures in radiographic views. Model outputs were evaluated using both one and three radiographic views. Ensembles were created from a combination of CNNs after training. They implemented a simple voting method to consolidate the output from the three views and ensemble of models.

Ref. [13] presented a malware detection method called MalNet, which uses a stacking ensemble with two deep neural networks—CNN and LSTM—as first-level learners, and logistic regression as a second-level learner.

Ref. [14] examined neural-network ensemble classification for lung cancer disease diagnosis. They proposed an ensemble of Weight Optimized Neural Network with Maximum-Likelihood Boosting (WONN-MLB), which essentially seeks to find optimal weights for a weighted (linear) majority vote. Ref. [15] applied a neural-network ensemble to intrusion detection, again using weighted majority voting.

Ref. [16] recently presented a cogent case for the use of XGBoost for tabular data, demonstrating that it outperformed deep models. They also showed that an ensemble comprising four deep models and XGBoost, predicting through weighted voting, worked best for the tabular datasets considered.

Ref. [17] proposed an ensemble DNN for tumor detection in colorectal histology images. The mechanism consists of weights that are derived from individual models. The weights are assigned to the ensemble DNN based on their metrics and the ensemble model is then trained. The model is again retrained by freezing all the layers, except for the fully connected and dense layers.

Ref. [18] presented an ensemble DL method to detect retinal disorders. Their method comprised three pretrained architectures—DenseNet, VGG16, InceptionV3—and a fourth Custom CNN of their own design. The individual results obtained from the four architectures were then combined to form an ensemble network that yielded superb performance over a dataset of retinal images.

Ref. [19] examined Deep Q-learning, presenting an ensemble approach that improved stability during training, resulting in improved average performance.

As noted above, [5, 6] presented conservation machine learning, which conserves models across runs, users, and experiments, and makes use of them. They showed that significant improvement could be attained by employing ML models already available anyway.

Deep GOld: Algorithmic Setup

Stacking (or Stacked Generalization) [4] is an ensemble method that uses multiple models to tackle classification or regression problems. The main idea is to first train different models on the original problem. The outputs of these models are considered to be a first level, which are then passed on to a second level to perform the final prediction. The inputs to the second-level model are thus the outputs of the first-level models.

Our framework involves deep networks as first-level models and ML methods as second-level models. For the former we used PyTorch, one of the top-two most popular and performant deep-learning software tools [1]. The module `torchvision.models` contains 59 deep-network models that were pretrained on the large-scale (over 1 million images), 1000-class ImageNet dataset [20].

Of the 59 models, we retained 51 (8 models proved somewhat unwieldy or evoked a "not implemented" error). Each of the models was first retrained over the four datasets we experimented with in this paper: Fashion MNIST, CIFAR10, CIFAR100, and Tiny ImageNet. As seen in Table 1, these datasets contain between 50,000 and 100,000 greyscale or color images in the training set, 10,000 images in the test set, with number of classes ranging between 10 and 200. Retraining was necessary, since the datasets contain images that differ in size and number of classes from ImageNet.

For retraining, we replaced the last fully connected (FC) 1000-class layer with a sequence of blocks comprising three layers: {FC, batchnorm, leaky ReLU}, denoted FBL. The final number of features of the original network was reduced to the dataset's number of classes through halving the number of nodes at each layer, starting with the closest power of 2. Consider en example: If the original network ended with 600 features, and the dataset contains 100 classes, then our modified network's final layers comprised a 512-node, 3-layer FBL block (512 being the closest power of 2 to 600), followed by a 256-node FBL, followed by a 128-node FBL, and ending with the 100 classes. In addition, the first convolutional layer of the original network needed adjustment in some cases. The retraining phase is detailed in Algorithm 1.

Table 1 Datasets

Dataset	Images	Classes	Training	Test
Fashion MNIST	28 × 28 grayscale	10	60,000	10,000
CIFAR10	32 × 32 color	10	50,000	10,000
CIFAR100	32 × 32 color	100	50,000	10,000
Tiny ImageNet	64 × 64 color	200	100,000	10,000

Algorithm 1 Retrain 51 pretrained models

```
Input:
    dataset ← dataset to be used
    pretrained ← {alexnet, densenet121, densenet161, densenet169, densenet201,
    efficientnet_b0, efficientnet_b1, efficientnet_b2, efficientnet_b3, efficientnet_b4,
    efficientnet_b5, efficientnet_b6, efficientnet_b7, mnasnet0_5, mnasnet1_0,
    mobilenet_v2, mobilenet_v3_large, mobilenet_v3_small, regnet_x_16gf,
    regnet_x_1_6gf, regnet_x_32gf, regnet_x_3_2gf, regnet_x_400mf, regnet_x_800mf,
    regnet_x_8gf, regnet_y_16gf, regnet_y_1_6gf, regnet_y_32gf, regnet_y_3_2gf,
    regnet_y_400mf, regnet_y_800mf, regnet_y_8gf, resnet101, resnet152, resnet18,
    resnet34, resnet50, resnext101_32x8d, resnext50_32x4d, shufflenet_v2_x0_5,
    shufflenet_v2_x1_0, vgg11, vgg11_bn, vgg13, vgg13_bn, vgg16, vgg16_bn, vgg19,
    vgg19_bn, wide_resnet101_2, wide_resnet50_2}  # Networks pretrained over
    ImageNet dataset
Output:
    Retrained models and their test scores
Load training set and test set of dataset
for net ∈ pretrained do
    Replace net final layer, and possibly adjust first convolutional layer
    Train entire net for 20 epochs over training set  # mini-batch size: 64 (8 for
    Tiny ImageNet), optimizer: SGD
    Save trained net and test set score
end for
```

Once Algorithm 1 is run for all four datasets, we are in possession of 51 trained models per dataset. We can now proceed to perform the two-level prediction, as detailed in Algorithm 2. Our interest herein was to study what one can do with models one has at hand. Towards this end, we first selected from the 51 retrained models three random ensembles of networks, of sizes 3, 7, and 11. Each network of an ensemble was then run over both the training and test sets of the dataset in question (without any training—only feed-forward output computation). These first-level outputs were then concatenated to form an input dataset for the second level. For example, if the ensemble contains seven networks, and the dataset in question is CIFAR100, then the first level creates two datasets: a training set with 50,000 samples and 701 features, and a test set with 10,000 samples and 701 features (701: 7 networks × 100 classes + 1 target class).

After the first level produced output datasets, we passed these along to the second level, wherein we employed ten ML algorithms:

1. `sklearn.linear_model.SGDClassifier`: Linear classifiers with SGD training.
2. `sklearn.linear_model.PassiveAggressiveClassifier`: Passive Aggressive Classifier [21].
3. `sklearn.linear_model.RidgeClassifier`: Classifier using Ridge regression.
4. `sklearn.linear_model.LogisticRegression`: Logistic Regression classifier.
5. `sklearn.neighbors.KNeighborsClassifier`: Classifier implementing the k-nearest neighbors vote.

```
Algorithm 2 Two-level prediction
Input:
    dataset ← dataset to be used
    ml_algs ← {SGDClassifier, PassiveAggressiveClassifier, RidgeClassifier, LogisticRegression, KNeighborsClassifier, RandomForestClassifier, MLPClassifier, XGBClassifier, LGBMClassifier, CatBoostClassifier}
Output:
    Test scores for majority prediction, and for all ML algorithms
# level 1: generate datasets from outputs of retrained networks
for $i \in \{3,7,11\}$ do
    networks ← pick $i$ networks at random from retrained networks of Algorithm 1
    for net ∈ networks do
        Run net over training set and test set
        Accumulate generated outputs along with (known) targets
    end for
    Generate 2 datasets, respectively: train-i, test-i
end for
# level 2: run ML algorithms over datasets generated by level 1
for $i \in \{3,7,11\}$ do
    for alg ∈ ml_algs do
        Load train-i, test-i
        Run alg to fit model to train-i
        Test fitted model on test-i
    end for
end for
```

Table 2 Hyperparameter value ranges and sets used by Optuna

Algorithm	Parameter	Values
`SGDClassifier`	alpha	$[1e-05, 1]$
	penalty	{'l2', 'l1', 'elasticnet'}
`PassiveAggressiveClassifier`	C	$[1e-02, 10]$
	fit_intercept	{True, False}
	shuffle	{True, False}
`RidgeClassifier`	alpha	$[1e-3, 10]$
	solver	{'auto', 'svd', 'cholesky', 'lsqr', 'sparse_cg', 'sag', 'saga'}
`LogisticRegression`	penalty	{'l1', 'l2'}
	solver	{'liblinear', 'saga'}
`KNeighborsClassifier`	weights	{'uniform', 'distance'}
	algorithm	{'auto', 'ball_tree', 'kd_tree', 'brute' }
	n_neighbors	[2, 20]
`RandomForestClassifier`	n_estimators	[10, 1000]
	min_weight_fraction_leaf	[0, 0.5]
	max_features	{'auto', 'sqrt', 'log2'}
`MLPClassifier`	activation	{'identity', 'logistic', 'tanh', 'relu'}
	solver	{'lbfgs', 'sgd', 'adam'}
	hidden_layer_sizes	{(64,64), (64,64,64), (64,64,64,64), (64,64,64,64)}
`XGBClassifier`	n_estimators	[10, 1000]
	learning_rate	[0.01, 0.2]
	gamma	[0, 0.4]
`LGBMClassifier`	n_estimators	[10, 1000]
	learning_rate	[0.01, 0.2]
	bagging_fraction	[0.5, 0.95]
`CatBoostClassifier`	iterations	[2, 10]
	depth	[2, 10]
	learning_rate	[1e-2, 10]

6. `sklearn.ensemble.RandomForestClassifier`: A random forest classifier.
7. `sklearn.neural_network.MLPClassifier`: Multi-layer Perceptron classifier, with five hidden layers of size 64 neurons each.
8. `xgboost.XGBClassifier`: XGBoost classifier [22].
9. `lightgbm.LGBMClassifier`: LightGBM classifier [23].
10. `catboost.CatBoostClassifier`: CatBoost classifier [24].

Results

Unsurprisingly, we found significant differences in the runtime of the level-2 ML algorithms (Algorithm 2). While some methods, such as RidgeClassifier and KNeighborsClassifier, were very fast, usually finishing within minutes, others proved slow (notably, XGBClassifier and CatBoostClassifier, which took several hours). While the number of samples of the generated ML datasets for the four problems studied is similar (identical to the original datasets—Table 1), the number of features differs by an order of magnitude: with ten classes for Fashion MNIST and CIFAR10, 100 classes for CIFAR100, and 200 classes for Tiny ImageNet, the latter two have 10 and 20 times more features than the former two, respectively. Some ML methods are known to scale less well with number of features.

ML runtimes for Fashion MNIST and CIFAR10 proved sufficiently fast to afford the use of hyperparamater tuning. Towards this end, we used Optuna, a state-of-the-art, automatic, hyperparameter optimization software framework [25], which we previously used successfully [26, 27]. Optuna offers a define-by-run style user API where one can dynamically construct the search space, and an efficient sampling algorithm and pruning algorithm. Moreover, our experience has shown it to be fairly easy to set up. Optuna formulates the hyperparameter optimization problem as a process of minimizing or maximizing an objective function given a set of hyperparameters as an input. The hyperparameter ranges and sets are given in Table 2. With CIFAR100

Table 3 Results for ensembles of 3, 7, and 11 random networks

Dataset	3 networks			7 networks			11 networks		
	Net	Maj	ML	Net	Maj	ML	Net	Maj	ML
Fashion MNIST	91.97%	92.38%	**92.40% (KN)**	92.23%	93.12%	**93.38% (KN)**	94.22%	93.79%	**94.40% (RG)**
	91.97%	92.05%	**92.50% (KN)**	92.40%	**93.26%**	93.25% (KN)	94.22%	93.62%	**94.57% (KN)**
	92.32%	**92.95%**	92.80% (KN)	93.24%	**93.69%**	93.59% (KN)	93.24%	93.57%	**94.11% (RG)**
	92.05%	92.39%	**92.61% (KN)**	94.01%	93.15%	**94.54% (RG)**	93.24%	93.71%	**93.92% (KN)**
	91.67%	91.40%	**92.00% (KN)**	92.95%	93.57%	**93.95% (KN)**	94.01%	93.19%	**94.50% (KN)**
	91.98%	**92.93%**	92.74% (KN)	92.27%	93.16%	**93.37% (KN)**	**93.86%**	93.75%	93.84% (KN)
	92.14%	92.69%	**93.19% (KN)**	93.86%	93.24%	**94.00% (RG)**	93.63%	93.83%	**94.07% (KN)**
	91.82%	92.27%	**92.51% (RF)**	92.27%	93.09%	**93.48% (KN)**	93.86%	93.81%	**94.25% (KN)**
	93.86%	92.79%	**93.94% (KN)**	92.95%	93.14%	**93.82% (RF)**	94.01%	93.78%	**94.66% (RG)**
	91.82%	**92.81%**	92.42% (RG)	93.86%	92.85%	**94.18% (KN)**	94.01%	93.53%	**94.39% (RG)**
CIFAR10	74.72%	71.00%	**75.87% (KN)**	75.60%	79.78%	**80.42% (KN)**	87.82%	80.94%	**88.29% (RG)**
	71.67%	72.10%	**75.05% (RG)**	86.90%	80.54%	**87.53% (RG)**	83.57%	80.09%	**84.82% (RG)**
	82.48%	82.03%	**83.30% (KN)**	87.33%	85.50%	**89.15% (RG)**	87.33%	80.82%	**89.12% (RG)**
	74.82%	74.68%	**75.05% (KN)**	74.72%	75.76%	**79.29% (KN)**	**86.60%**	83.28%	86.43% (RG)
	74.95%	**76.48%**	76.26% (KN)	86.90%	82.72%	**87.57% (RG)**	86.60%	79.93%	**87.24% (RG)**
	75.60%	75.28%	**76.79% (KN)**	83.57%	81.31%	**83.95% (KN)**	87.22%	81.54%	**88.33% (RF)**
	76.21%	77.77%	**78.27% (KN)**	74.00%	74.30%	**77.96% (KN)**	**86.90%**	84.44%	86.67% (RG)
	86.90%	85.96%	**88.16% (LR)**	**86.90%**	84.74%	86.72% (RG)	86.60%	82.26%	**87.71% (RG)**
	86.60%	81.76%	**86.92% (KN)**	86.90%	82.54%	**88.00% (RG)**	87.82%	83.36%	**89.56% (RG)**
	76.43%	72.62%	**77.42% (SG)**	86.60%	77.97%	**87.08% (RG)**	87.82%	85.02%	**89.59% (RG)**
CIFAR100	48.86%	51.02%	**54.69% (KN)**	55.70%	55.11%	**59.50% (RG)**	60.76%	60.10%	**66.02% (RG)**
	48.74%	44.95%	**51.68% (KN)**	60.76%	60.11%	**65.82% (LR)**	61.66%	62.50%	**67.30% (RG)**
	48.74%	49.47%	**54.08% (KN)**	46.38%	51.97%	**54.61% (KN)**	9.58%	57.51%	**66.07% (LR)**
	60.08%	55.30%	**63.67% (SG)**	61.30%	54.45%	**64.14% (RG)**	9.58%	58.43%	**64.86% (RG)**
	47.06%	47.46%	**52.48% (RG)**	61.30%	57.22%	**64.08% (RG)**	60.08%	59.00%	**64.55% (RG)**
	47.55%	50.94%	**53.64% (RG)**	60.08%	56.86%	**64.46% (RG)**	60.76%	56.07%	**64.25% (LR)**
	46.55%	43.81%	**51.19% (KN)**	47.55%	47.91%	**52.79% (SG)**	9.58%	58.29%	**64.89% (RG)**
	61.30%	57.47%	**63.86% (RG)**	48.86%	50.85%	**54.93% (RG)**	61.30%	57.09%	**65.07% (RG)**
	46.30%	44.65%	**50.30% (SG)**	9.58%	53.56%	**56.81% (KN)**	9.58%	59.39%	**66.34% (RG)**
	9.58%	33.88%	**44.23% (SG)**	61.66%	57.84%	**64.84% (SG)**	61.48%	58.78%	**65.20% (LR)**
Tiny ImageNet	55.77%	54.32%	**59.97% (RG)**	57.40%	56.06%	**63.32% (RG)**	67.30%	65.33%	**70.17% (RG)**
	53.50%	54.83%	**59.61% (LR)**	54.83%	53.33%	**59.26% (RG)**	57.40%	63.72%	**66.00% (RG)**
	58.34%	48.45%	**61.60% (RG)**	58.34%	60.59%	**64.30% (RG)**	57.23%	58.45%	**64.31% (RG)**
	58.34%	59.91%	**63.05% (RG)**	55.77%	54.83%	**60.10% (RG)**	58.62%	60.42%	**65.19% (RG)**
	53.50%	50.97%	**58.10% (RG)**	55.47%	53.11%	**60.11% (RG)**	56.20%	60.78%	**63.83% (RG)**
	57.23%	47.54%	**59.88% (RG)**	58.62%	61.26%	**62.14% (RG)**	67.30%	64.33%	**69.88% (RG)**
	58.62%	59.98%	**64.69% (RG)**	58.62%	62.41%	**66.24% (RG)**	56.20%	60.02%	**64.09% (RG)**
	57.40%	56.14%	**62.76% (LR)**	56.16%	60.85%	**63.46% (RG)**	58.62%	62.44%	**65.94% (RG)**
	54.76%	53.81%	**60.57% (RG)**	56.61%	57.70%	**63.76% (RG)**	58.34%	63.05%	**65.98% (RG)**
	57.23%	48.52%	**60.31% (RG)**	58.34%	62.00%	**63.85% (RG)**	58.62%	62.90%	**66.46% (RG)**

Accuracy scores shown are over test sets. *Net* score of best network, *Maj* score of majority prediction, *ML* score of best ML method. For the latter, the ML method producing the best score is given in parentheses. *RG* Classifier using Ridge regression, *KN* k-nearest neighbors classifier, *SG* Linear classifier with SGD training, *PA* Passive Aggressive classifier, *LR* Logistic regression, *RF* Random Forest classifier, *MP* Multi-layer perceptron, *LG* LightGBM, *XG* XGBoost, *CB* Catboost

and Tiny ImageNet, we did not use Optuna, but rather ran the ML algorithms with their default values.

Table 3 presents our results (we set a 10-h limit on an ML algorithm's run of a row in the table, i.e., the level-2 loop of Algorithm 2.) A total of 120 experiments were performed: 4 datasets × ensembles of size 3, 7, and 11 × 10 complete runs per dataset. In each experiment, we generated level-1 datasets and then executed the ML algorithms, as delineated in Algorithm 2. We then compared three values: (1) the test score of the top network amongst the random ensemble (known from Algorithm 1); (2) the test score of majority prediction, wherein the predicted class is determined through a majority vote amongst the ensemble's networks' outputs; (3) the test score of the top ML method. The code is available at https://github.com/moshesipper.

Discussion

As observed in Table 3, of the total of 120 experiments, an ML algorithm won in all but ten experiments (four were won by the retrained network, and six by majority prediction).

We note that classical algorithms, notably Ridge regression and k-nearest neighbors, worked best (they account for 104 of the wins). They are also fast, scalable, and amenable to quick hyperparameter tuning. If one wishes to focus on a smaller batch of ML algorithms, these two seem like an excellent choice.

As noted in the section "Introduction", we often find ourselves in possession of a plethora of models, either collected by us through many experiments, or by others (witness our use of pretrained models herein). Benefiting from current state-of-the-art technology, Deep GOld leverages this wealth of models to attain better performance. One can of course tailor the framework to available deep networks and to a personal predilection for any ML algorithm(s).

Concluding Remarks

We presented Deep GOld, a comprehensive, stacking-based framework for combining deep learning with machine learning. Our framework involves ensemble selection from 51 retrained pretrained deep networks as first-level models, and 10 machine-learning algorithms as second-level models. We demonstrated the unequivocal benefits of the approach over four image-classification datasets.

We suggest a number of paths for future research:

- Further analysis of ML algorithms whose inputs are the outputs of deep networks. Do some ML methods inherently work better with such datasets?
- Currently, the features for level 2 comprise only the level-1 outputs. We might enhance this setup through automatic feature construction.
- Train (or retrain) the level-1 networks *alongside* a level-2 ML model: (1) After each training epoch of the networks in the ensemble, generate a dataset from the network outputs; (2) a level-2 ML algorithm then fits a model to the level-1 dataset; (3) the ML model generates class probabilities, which are used to ascribe loss values to the networks-in-training.

Acknowledgements I would like to thank Raz Lapid for the helpful discussions.

Declarations

Conflict of Interest M. Sipper declares that he has no conflict of interest.

Research Involving Human Participants and/or Animals This article does not contain any studies with human participants or animals performed by any of the authors.

Informed Consent Not applicable.

References

1. Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, Killeen T, Lin Z, Gimelshein N, Antiga L, et al. PyTorch: an imperative style, high-performance deep learning library. arXiv preprint; 2019. arXiv:1912.01703.
2. Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, Vanderplas J, Passos A, Cournapeau D, Brucher M, Perrot M, Duchesnay E. Scikit-learn: machine learning in Python. J Mach Learn Res. 2011;12:2825–30.
3. Scikit-learn: machine learning in python; 2022. https://scikit-learn.org/. Accessed: 2022-1-12.
4. Wolpert DH. Stacked generalization. Neural Netw. 1992;5(2):241–59.
5. Sipper M, Moore JH. Conservation machine learning. BioData Min. 2020;13(1):9.
6. Sipper M, Moore JH. Conservation machine learning: a case study of random forests. Nat Sci Rep. 2021;11(1):3629.
7. Opitz D, Shavlik J. Generating accurate and diverse members of a neural-network ensemble. In: Touretzky D, Mozer MC, Hasselmo M, editors. Advances in neural information processing systems, vol. 8. Cambridge: MIT Press; 1996.
8. Zhou Z-H, Wu J, Tang W. Ensembling neural networks: many could be better than all. Artif Intell. 2002;137(1–2):239–63.
9. West D, Dellana S, Qian J. Neural network ensemble strategies for financial decision applications. Comput Oper Res. 2005;32(10):2543–59.
10. Berkhahn S, Fuchs L, Neuweiler I. An ensemble neural network model for real-time prediction of urban floods. J Hydrol. 2019;575:743–54.
11. Al-Qutami TA, Ibrahim R, Ismail I, Ishak MA. Virtual multiphase flow metering using diverse neural network ensemble and adaptive simulated annealing. Expert Syst Appl. 2018;93:72–85.
12. Kitamura G, Chung CY, Moore BE. Ankle fracture detection utilizing a convolutional neural network ensemble implemented with a small sample, de novo training, and multiview incorporation. J Digit Imaging. 2019;32(4):672–7.
13. Yan J, Qi Y, Rao Q. Detecting malware with an ensemble method based on deep neural network. Secur Commun Netw. 2018;2018:7247095. https://doi.org/10.1155/2018/7247095.
14. Alzubi JA, Bharathikannan B, Tanwar S, Manikandan R, Khanna A, Thaventhiran C. Boosted neural network ensemble

classification for lung cancer disease diagnosis. Appl Soft Comput. 2019;80:579–91.

15. Ludwig SA. Applying a neural network ensemble to intrusion detection. J Artif Intell Soft Comput Res. 2019;9:177–88.
16. Shwartz-Ziv R, Armon A. Tabular data: deep learning is not all you need. Inf Fusion. 2022;81:84–90.
17. Ghosh S, Bandyopadhyay A, Sahay S, Ghosh R, Kundu I, Santosh KC. Colorectal histology tumor detection using ensemble deep neural network. Eng Appl Artif Intell. 2021;100: 104202.
18. Paul D, Tewari A, Ghosh S, Santosh KC. OCTx: Ensembled deep learning model to detect retinal disorders. In: 2020 IEEE 33rd International Symposium on Computer-Based Medical Systems (CBMS). 2020; p. 526–31.
19. Elliott DL, Santosh KC, Anderson C. Gradient boosting in crowd ensembles for Q-learning using weight sharing. Int J Mach Learn Cybern. 2020;11(10):2275–87.
20. Deng J, Dong W, Socher R, Li L-J, Li K, Fei-Fei L. ImageNet: a large-scale hierarchical image database. In: 2009 IEEE conference on computer vision and pattern recognition. 2009; p. 248–55.
21. Crammer K, Dekel O, Keshet J, Shalev-Shwartz S, Singer Y. Online passive-aggressive algorithms. J Mach Learn Res. 2006;7(19):551–85.
22. Chen T, Guestrin C. XGBoost: a scalable tree boosting system. In: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining. KDD '16. 2016; pp. 785–94, New York, NY, USA.
23. Ke G, Meng Q, Finley T, Wang T, Chen W, Ma W, Ye Q, Liu T-Y. LightGBM: a highly efficient gradient boosting decision tree. Adv Neural Inf Process Syst. 2017;30:3146–54.
24. Prokhorenkova L, Gusev G, Vorobev A, Dorogush AV, Gulin A. CatBoost: unbiased boosting with categorical features. arXiv preprint. 2017; arXiv:1706.09516.
25. Akiba T, Sano S, Yanase T, Ohta T, Koyama M. Optuna: a next-generation hyperparameter optimization framework. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2019; p. 2623–31.
26. Sipper M. Neural networks with à la carte selection of activation functions. SN Comput Sci. 2021;2(470). https://doi.org/10.1007/s42979-021-00885-1.
27. Sipper M, Moore JH. AddGBoost: a gradient boosting-style algorithm based on strong learners. Mach Learn Appl. 2022;7: 100243.

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.