Deep Neural Network Ensembles

Sean Tao
Department of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
shtao@andrew.cmu.edu

Abstract

Current deep neural networks suffer from two problems; first, they are hard to interpret, and second, they suffer from overfitting. There have been many attempts to define interpretability in neural networks, but they typically lack causality or generality. A myriad of regularization techniques have been developed to prevent overfitting, and this has driven deep learning to become the hot topic it is today; however, while most regularization techniques are justified empirically and even intuitively, there is not much underlying theory. This paper argues that to extract the features used in neural networks to make decisions, it's important to look at the paths between clusters existing in the hidden spaces of neural networks. These features are of particular interest because they reflect the true decision making process of the neural network. This analysis is then furthered to present an ensemble algorithm for arbitrary neural networks which has guarantees for test accuracy. Finally, a discussion detailing the aforementioned guarantees is introduced and the implications to neural networks, including an intuitive explanation for all current regularization methods, are presented. The ensemble algorithm has generated state-of-the-art results for Wide-ResNet on CIFAR-10 and has improved test accuracy for all models it has been applied to.

1 Introduction

Consider a simple feed forward neural network. Define a hidden space corresponding to a hidden layer of the neural network as simply the outputs of the hidden nodes at that hidden layer for some input. All hidden spaces are composed of perceptrons with respect to the previous layer, each of which has a hyperplane decision boundary. Points in the previous space are mapped to a constant function of their distance to this plane, and depending on the activation function, compressed or stretched. This naturally leads to clustering in hidden spaces. The process is repeated for each perceptron comprising the hidden space, where adding a perceptron adds a dimension to the hidden space by projecting the points into a new dimension depending on their distances from the hyperplane of the new perceptron and the activation function. Define a feature to be a measurable characteristic which a neural network uses to make its classification decision. Unfortunately, these clusters are not features in and of themselves but rather mixtures of features. To extract individual features, then, cluster paths should be examined, where a path is defined per individual input point as the sequence of clusters in the neural network the point belongs to, starting from cluster it belongs to in the input space, then the clusters it belongs to in each hidden space, and finally the cluster it belongs to in the output space. Intuitively, paths represent features because each path defines a region of the input space which will eventually end up at the same cluster in the output space via the same logic pathway used by the neural network. Thus, it immediately follows that a point on a path must be classified similarly as other points on that path, assuming all points in the output space are classified by cluster. This is another way of stating that points on the same path are indistinguishable from each other to the neural network, and since points in different paths were differentiated in some layer,

Preprint. Work in progress.

paths separate out features of neural networks. This process is formalized in the algorithm described below. However, besides merely finding the features in a neural network, the paths serve an even more important purpose–they separate the input into regions of confidence with respect to the output classification. In particular, paths represent specific features, some of which were found because they are truly useful and others due to overfitting. The process of determining "good" and "bad" data points, formally defined below, attempts to separate data into these two categories. Informally, "good" data come from paths that contain many points that are classified correctly, since these are likely not due to random chance and thus are real features. An ensemble algorithm can then be created to combine different models, where models only vote on their "good" data points. This is equivalent to querying neural networks for points where they are confident in their predictions. Points where the model is unsure can then be classified by other models.

2 Related Work

Much effort has been put into interpreting deep neural networks in the past. A few meta studies summarize the effort of the community as a whole rather well. There are three general types of methods for deep neural networks; namely, by discovering "ways to reduce the complexity of all these operations," "understand[ing] the role and structure of the data flowing through these bottlenecks," and "creat[ing] networks that are designed to be easier to explain" [4]. The difference between the algorithm presented here and the aforementioned attempts at interpretability is that this derives its logic entirely from the network–in particular, paths, by definition, represent the features used in classification.

In addition, clustering has been applied to the hidden spaces of neural networks in prior work, also in the context of interpretability [10]. However, cluster path analysis in deep neural networks could not be found.

Finally, ensembling algorithms of deep neural networks have been attempted before. For instance, algorithms already presented in different contexts have been applied to neural networks, such as the Super Learner and AdaBoost [7, 12, 19]. Nevertheless, "the behavior of ensemble learning with deep networks is still not well studied and understood" [7].

3 Algorithm

3.1 Training

Here, the algorithm to train the ensemble is presented; this also contains the process to generate paths. As clarification, the distinction in this paper between training data and a training set is that the training data is split into a training set and a validation set. First, partition the training data into training/validation sets. Repeat this such that there is minimal overlap between points in different training sets. Then, for each of the partitions, conduct the following steps.

Train a neural network on the training set, selecting the best model via validation accuracy. Add this neural network to the set of model 1's ("original models"). Determine the optimal number of means in each layer via the "elbow" technique [17] by plotting inertia against the number of means with k-means [15]. Run k-means again with the optimal number of means on the input layer, each of the hidden layers, and output layer for the training set. For each point in the training and validation sets, determine the path of clusters through the neural network. Find optimal values of the following three parameters via grid search, filter out validation points which do not meet the criteria, and calculate the validation accuracy on the remaining points. The three parameters are: maximum distance to a cluster center, minimum number of data points in a split, and minimum accuracy in a split. Define a split to be a partial path of length 2–in other words, how one cluster was split into different clusters in the subsequent layer. Thus, for each point in the validation set, if it is too far from its cluster in any layer, if it is in a split which has too few points, or if it is in a split with too low validation accuracy, filter out the point, and calculate the validation accuracy on the remaining points. Ideally, the parameters which represent the tightest restrictions that filter out the fewest points and achieve the desired validation accuracy should be chosen. This process identifies paths which represent features not generated by overfitting and contain points of high validation, and thus test, accuracy. The idea is to then train other models to focus specifically on the "bad" test points. Using the same parameters

found above, separate the training set into “good” and “bad” training points, where “good” data points are points which satisfy the parameters found above. Train another neural network, with the same partition of train and validation sets. However, repeat each “bad” training point in the training set, such that each “bad” point appears twice. Add this network to the set of model 2’s ("bad 1 models"). Repeat the analysis steps on this new neural network.

Algorithm 1 Training the Ensemble

```
Partition the training data into training/validation sets
for each partition do
    Train a neural network
    Add this network to the set of model 1's ("original models")
    Run k-means on all layers
    for each point in training and validation sets do
        Determine the path of clusters
    end for
    Find parameters for filtering clusters
    Separate the training set into "good" and "bad" points
    Train another neural network with oversampling on the "bad" points
    Add this network to the set of model 2's ("bad 1 models")
    Repeat the analysis steps for the new neural network
end for
```

3.2 Testing

To run this ensemble on the test set, for each test data point, conduct the following steps. Determine whether or not it’s a “good” data point in each of the models in the set of model 1’s by using the parameters for the respective model. If it’s a “good” data point in at least half of the models in the set of model 1’s, and any of these models agree on a label, return that label. Call these test points “original good” test points. Otherwise, determine whether or not it’s a “good” test point in each of the models in the set of model 2’s. If it’s a “good” test point in at least half of the models in the set of model 2’s, and if any of these models agree on a label, return that label. Call these test points “bad1” test points. Otherwise, add the output vectors of each of the models in the set of model 2’s, and return the label corresponding to the largest value in the resulting sum vector. Call these data points “bad2” data points.

Algorithm 2 Testing the Ensemble

```
for each test data point do
    if "good" in set of model 1's and agree on label then
        Classify via voting of "good" model 1's ("good" test point)
        continue
    end if
    if "good" in set of model 2's and agree on label then
        Classify via voting of "good" model 2's ("bad1" test point)
        continue
    end if
    Classify via majority voting of all model 2's ("bad2" test point)
end for
```

3.3 Larger Models

While this algorithm works to increase test accuracy in smaller, feed forward neural networks, it must be modified to so that it is effective for larger, state-of-the-art models. This is necessary for a few reasons. First, the architecture may not be able to be represented as a simple directed acyclic graph, and thus hidden spaces may not be defined. Second, even if some modern architectures avoid these problems, they often contain such high numbers of hidden nodes that clustering is not meaningful due to the curse of dimensionality. Fortunately, the concept of "good" and "bad" points seem to transcend model architectures and training processes. That is, models all seem to classify "good"

points with much higher accuracy than "bad" points, regardless of their architecture, methods of regularization applied, or other training techniques. In particular, larger models seem to be able to accurately classify points which smaller models can classify correctly when trained on any subset of the training data. Then it follows that only "bad2" test data are inaccurate for larger models. Thus, for larger models, to create an ensemble, conduct the following steps.

First, train one neural network normally ("original model"). This will be used to classify "good" and "bad1" test data. Then, train another neural network ("bad model"), oversampling all training points which were classified as "bad" in the majority of the set of "model 1's," as defined in the training section. Finally, use this model to classify all "bad2" test data.

Algorithm 3 Training and Testing on Larger Models

```
Train larger model normally ("original model")
Train larger model with oversampled "bad" training points as found in smaller models ("bad model")
for each test data point do
    if "good" or "bad1" test point then
        Classify using "original model"
        continue
    end if
    if "bad2" test point then
        Classify using "bad model"
    end if
end for
```

4 Results

The path analysis portion of the algorithm was run on a feed forward neural network with Dropout [14] with 4 hidden layers trained on MNIST [9]. Features were generated from all "good" splits in two manners: first, by averaging all the training points in that split; and second, by finding via backpropagation the input which best generates the cluster center at that layer while the hidden layers' weights remain fixed. No additional regularization is used, demonstrating an improvement over existing techniques, since typically this would just generate noise [11]. The former technique was used on the splits from the first to second hidden layer, since the previous splits had too many images to display, and subsequent layers only had 10 clusters, one per label. Due to lack of resources, cluster split means could not be found, but those could potentially generate clearer features. The latter technique was used on splits from the input to the first hidden layer. For both methods, visual inspection confirms that the features generated separate different digits and, when they exist, different manners of writing each digit. The features are presented below. The entire training algorithm was

Figure 1: All "good" features from the first to the second hidden layer via input averaging

run on CIFAR-10 [8] with an ensemble where the set of model 1's consisted of neural networks created by transfer learning of InceptionResNet [16] trained on ImageNet [2] and the set of model 2's consisted of VGG [13] models specific to CIFAR-10 [3]. The highest test accuracy of any individual of these neural networks when trained on CIFAR-10 is 93.56% [3]. With five validation sets ("Block Partitions"), where each validation set was created by partitioning the training data into contiguous blocks of size 10000, the ensemble achieves 94.63% test accuracy. With a different partition ("Stride 1 Partitions"), where the five validation sets were created by equivalence classes of the training data enumerated and taken modulo 5, the ensemble achieves 94.26% test accuracy. When these two are combined and 10 validation sets are used, the accuracy increases to 94.77%, a 1.21% increase over

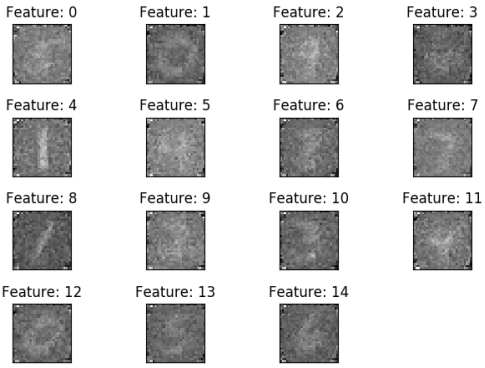

Figure 2: All "good" features from the input to the first hidden layer via backpropagation

any individual network trained on the entire training data. The specific breakdown over the different sets of points are presented below. The difference between the test accuracy of "original good" points, which were "good" on the set of model 1's, and "bad 1" points, which were "good" on the set of model 2's, and "bad 2" points is apparent.

Small Model Test Accuracy Breakdown				
Ensemble (Total Num Models)	Original Good	Bad 1	Bad 2	Overall
Block Partitions (10)	98.97%	99.37%	79.65%	94.63%
Stride 1 Partitions (10)	99.11%	99.33%	76.71%	94.26%
All Partitions (20)	99.06%	99.29%	77.80%	94.77%

Small Model Total Number of Points by Type Breakdown				
Ensemble (Total Num Models)	Original Good	Bad 1	Bad 2	Overall
Block Partitions (10)	5815	1900	2285	10000
Stride 1 Partitions (10)	5255	2555	2190	10000
All Partitions (20)	5853	2106	2041	10000

Larger models were also considered. The state-of-the-art results on CIFAR-10 were achieved via a model created with GPipe [6], but due to resource limitations, training this was impossible. The second best results were achieved by applying AutoAugment [1] on existing large models. Of the models used, Wide-ResNets (WRN) [22] were the simplest to train, so the larger model algorithm was applied to this, and this ensemble achieved 97.51% test accuracy, 0.18% better than the results for WRN's reported in the AutoAugment paper. The previous state-of-the-art results, now the second best, were achieved by applying AutoAugment to ShakeDrop [21], and the larger model algorithm was also applied here. This ensemble achieved 98.40% test accuracy, 0.12% worse than reported in the AutoAugment paper. However, the reported results could not be reproduced, and the ensemble performed better than the original model itself. Due to resource limitations, this experiment could not be repeated, and more trials are needed.

WRN Ensemble Test Accuracy Breakdown				
Ensemble (Total Num Models)	Original Good	Bad 1	Bad 2	Overall
WRN (2)	99.62%	99.57%	89.31%	97.51%
ShakeDrop (2)	99.71%	99.57%	93.39%	98.39%

Small Model Total Number of Points by Type Breakdown				
Ensemble (Total Num Models)	Original Good	Bad 1	Bad 2	Overall
WRN (2)	5853	2106	2041	10000
ShakeDrop (2)	5853	2106	2041	10000

5 Analysis

5.1 "Good" Paths

5.1.1 Bounds on Test Error

Assume that the training data and the test data are i.i.d., and the training data is sufficiently large. For neural networks, there exist potentially disjoint subspaces of the input space which correspond to "good" points in a path–namely, if a point were to fall into a particular subspace, then the point will be "good" in that is does not get filtered, and it will follow the corresponding path. Consider k neural networks each trained on disjoint and insignificant fractions f of the training data, and assume that each discovers some similar subspace. Due to the assumption that the training data is large, each of the k neural networks is effectively trained on some subset of the data sampled from the distribution of all (train and test) data. Moreover, because the samples are disjoint, there is no dependence introduced by overlapping data. Now, consider the intersection of all k previously discussed subspaces, one found per network; call this s. Assume n points from the respective validation set of each network reside in s, and the maximum classification error of the respective n points for each of the k networks is ϵ'. Define p to be the probability that a training set of size f will produce a neural network that will classify all points in this subspace from the training and test points with error at most ϵ'; in other words, the probability that a neural network will discover s. Noting that the maximum variance for a binary variable is $\frac{1}{4}$ and applying the central limit theorem, a confidence interval can be created to estimate p, since a sample mean is approximately normally distributed with variance at most $\frac{1}{4\sqrt{k}}$, and there exists a single sample mean with value 1. Define ϵ to be the average true error for all points in this subspace of any model trained on the fraction f of the training data. Due to the properties of the normal distribution, the true value of p is almost certainly within 6 standard deviations of this, or $1 - \frac{6}{4\sqrt{k}}$. Then with high confidence, and by applying similar logic as above, the probability ϵ' is approximately normally distributed with mean ϵ and variance at most $\frac{1}{4\sqrt{n}}$ is $(1 - \frac{6}{4\sqrt{k}})$, which converges to 1 as k grows large. More concisely, for any confidence level, by using a sufficiently large ensemble, ϵ' is normally distributed with mean ϵ and variance at most $\frac{1}{4\sqrt{n}}$. This is important because now a confidence interval can be created for ϵ, the average actual error of all points, train and test, in this subspace.

5.1.2 Implications

The only previous work comparable to this requires either a finite hypothesis space or restrictions on the capabilities of the model–specifically the VC dimension–and this is inapplicable to modern neural networks [20]. This lies in between–it provides bounds on test error for extremely complicated models, but only for certain data points. The intuitive explanation behind the previous section is thus: a neural network trained on training data should perform better than random on unseen test data is because the test data is expected to be drawn from the same distribution as the training data. Any given pattern, which is what a subspace represents, may not be applicable in the test set, but a pattern found in multiple disjoint subsets is most likely to be real, and the probability can be mathematically described via the central limit theorem. When applied on an ensemble of models trained with heavily overlapping training sets, the above analysis does not hold, and the ensemble does not perform too much better than the best model. On the other hand, the above algorithm provides a framework from which useful features can be extracted, and this allows larger models to differentiate between true patterns and overfitting.

The analysis above also serves as a justification of most regularization methods as well as an explanation for Occam's Razor in machine learning. "Simple" models are not preferred because they have a higher prior probability than a complicated one–there is no justification for this. "Simple" models are preferred to complicated ones because, in general, they tend to discover regions which are larger and not particularly convoluted and therefore more easily discovered by other models. This, in turn, implies a better bound on the test error. To see why regularization works, it's important to understand what regularization is doing. Regularization effectively forces models to find more similar solutions; for instance, by placing restrictions on weights [5, 18]. Thus, models, even when trained on different training sets or with different initializations, are more likely to find similar patterns, and this in turn implies that the patterns which are found are more likely to generalize.

Unfortunately, using regularization means imposing a bias on the model, as per the bias-variance trade-off. These trade-offs are often only backed by intuition and better test error but not justified mathematically. On the other hand, consider an algorithm which follows the general idea as presented above: it trains an ensemble of models; analyzes cluster paths; defines a set of "good" points; and handles "bad" points in some manner, whether by oversampling and training another model or by using some different method. Patterns which were found in smaller models are likely to be discovered by larger models, especially if they are true patterns and are not symptoms of overfitting. These patterns are identified in the larger models, so another model can be trained to focus on the truly bad (the "bad2") points. This is beneficial for three reasons. First, this ensemble is effectively a form of regularization on the larger models, since it forces the real patterns to be kept, and this increases test accuracy. There is no added bias since no new assumptions were made. Second, this method can be used iteratively, creating a process to continuously increase test accuracy. Third, this provides a framework for how models can collaborate with each other–they should yield when they are unsure and speak up when they are relatively certain. Combined with feature extraction techniques as outlined above, this could allow for an entirely new field of machine learning: continuous learning with neural networks.

5.2 Bounds on Validation Error of Ensembles

Consider an ensemble consisting of k neural networks, of which all incorrectly classify at most v of the "good" validation points. Then the number of incorrectly classified validation points when applied only to points such that at least a fraction f_1 of the models deem it "good" and of those, at least a fraction f_2 agree, is at most $\frac{v}{f_1 * f_2}$. This establishes a lower bound on the validation accuracy of the ensemble for "good" points. This is necessary because ensembling effectively trades a decrease in validation accuracy for expanding the number of "good" data points.

6 Discussion

6.1 Oversampling

The idea behind the "model 2's" is this: the model performs significantly better on “good” data points than “bad” data points because "good" data points represent features that the neural network is confident were not created as a result of overfitting. It is natural to continue by creating a new model to focus on the “bad” data points. Even if they were classified correctly in the training set, they may still have comparatively high loss. Thus, for any model used to classify “bad” data points, the idea is to increase the weight the model puts on these points in the training set to reduce this loss. Oversampling achieves this effect. On another note, it may be possible perform this process recursively; that is, to continue process for the models in the set of "model 2's" and create a set of "model 3's."

6.2 Partitions

For the training algorithm, the idea of partitioning the training data into a training and validation sets in multiple ways is crucial. This is because the multiple partitions are what create the ability to differentiate between real features and overfitting. Larger models seem to be able to emulate the effects of ensembles of smaller models trained on different portions of the training data. Intuitively, this makes sense–larger models are effectively just smaller models combined together, since neural networks are an iterative model, by design. This is further supported by the fact that for state-of-the-art models, "bad1" test points are classified with similar accuracy as "original good" test points. Indeed, the critical step occurs in identifying "bad2" data points, which are outliers in every model, and dealing with these specifically. While the algorithm presented oversamples the corresponding training data, this is not necessarily the optimal approach, and a better method of creating ensembles of models to handle these points should be considered in the future. Finally, it should be noted that the partitions should overlap as little as possible with each other, thus decreasing the probability that the same overfit features would be found by two different models.

6.3 Justification for Parameters Filtering Clusters

There exists an intuitive explanation behind the three parameters which were chosen when determining which data points are "good." The distance to a cluster center is considered because all data must belong to a cluster, and therefore clusters contain data which do not really belong to any single cluster. The idea is to filter out these outliers. The number of points in a split is considered because if this is too small, it's impossible to reason about the accuracy of points which follow that path. Intuitively, this is because points in small splits are most likely outliers which do not truly belong, and while they may be classified correctly in the training set, this is most likely due to overfitting. Finally, the accuracy of a split is taken into consideration because if the accuracy of the split is low in the training or the validation set, there's no reason to believe it will be better in the test set.

7 Conclusion

In conclusion, this paper presents an algorithm to analyze features of neural networks and differentiate between useful features and those found due to overfitting. This process is then applied to various models, resulting in improvements across the board and generating state-of-the-art results for Wide-ResNets on CIFAR-10. Lastly, an analysis concerning bounds to test accuracy for these ensembles is detailed, and the implications, including an intuitive understanding of regularization, are presented.

8 Acknowledgments

I like to call this project the Miracle Project because it's a true miracle that this project happened...there are too many people to thank. First, thanks to my mom, dad, and brother. Second, thanks to Professors Barnabás Póczos and Majd Sakr. Third, thanks to Theo Yannekis, Michael Tai, Max Mirho, Josh Li, Zach Pan, Sheel Kundu, Shan Wang, Luis René Estrella, Eric Mi, Arthur Micha, Rich Zhu, Carter Shaojie Zhang, Eric Hong, Kevin Dougherty, Catherine Zhang, Marisa DelSignore, Elaine Xu, David Skrovanek, Anrey Peng, Bobby Upton, Angelia Wang, and Frank Li. You guys are the real heroes of this project. And, lastly, thanks to you, my dear reader.

References

[1] Ekin Dogus Cubuk, Barret Zoph, Dandelion Mané, Vijay Vasudevan, and Quoc V. Le. Autoaugment: Learning augmentation policies from data. *CoRR*, abs/1805.09501, 2018.

[2] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[3] Yonatan Geifman. cifar-vgg. `https://github.com/geifmany/cifar-vgg`, 2018.

[4] Leilani Henrina Gilpin, David Bau, Ben Ze Yuan, Ayesha Bajwa, Michael Specter, and Lalana Kagal. Explaining explanations: An overview of interpretability of machine learning. *2018 IEEE 5th International Conference on Data Science and Advanced Analytics (DSAA)*, pages 80–89, 2018.

[5] Arthur E. Hoerl and Robert W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 42:80–86, 2000.

[6] Yanping Huang, Yonglong Cheng, Dehao Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V. Le, and Zhongqian Chen. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *CoRR*, abs/1811.06965, 2018.

[7] Cheng Ju, Aurélien Bibaut, and Mark J. van der Laan. The relative performance of ensemble methods with deep convolutional neural networks for image classification. *CoRR*, abs/1704.01664, 2017.

[8] Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009.

[9] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[10] Xuan Liu, Xiaoguang Wang, and Stan Matwin. Interpretable deep convolutional neural networks via meta-learning. *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–9, 2018.

[11] Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. Feature visualization. *Distill*, 2017. https://distill.pub/2017/feature-visualization.

[12] Holger Schwenk and Yoshua Bengio. Boosting neural networks. *Neural Computation*, 12:1869–1887, 2000.

[13] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2015.

[14] Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[15] Hugo Steinhaus. Sur la division des corps matériels en parties. *Bull. Acad. Pol. Sci., Cl. III*, 4:801–804, 1957.

[16] Christian Szegedy, Sergey Ioffe, and Vincent Vanhoucke. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, 2016.

[17] Robert L. Thorndike. Who belongs in the family? *Psychometrika*, 18(4):267–276, Dec 1953.

[18] Robert Tibshirani. Regression shrinkage and selection via the lasso. 1994.

[19] Mark J. van der Laan, Eric C. Polley, and Alan E. Hubbard. Super learner. *Statistical applications in genetics and molecular biology*, 6:Article25, 2007.

[20] Vladimir Naumovich Vapnik. The nature of statistical learning theory. In *Statistics for Engineering and Information Science*, 2000.

[21] Yoshihiro Yamada, Masakazu Iwamura, Takuya Akiba, and Koichi Kise. Shakedrop regularization for deep residual learning. 2018.

[22] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *CoRR*, abs/1605.07146, 2016.