Debugging and Visualization Tools of Deep Learning – Challenges and Chances

Abstract

In the following article we present a survey of ideas and related work that we think will become a new subfield: The visualization and debugging of deep neural networks using multimedia methods. The topic is new as most research treats deep learning as a black box and asks very little about the inner workings of deep neural networks. For the same reason the topic is also brave: Much is speculated but very little is actually known about how to introspect and debug neural network weights systematically with the goal of understanding outcomes. The topic is multimedia because we hypothesize that rather than visualizing number series, experimenting with perceptual data, such as audio, images or video will be the preferred way of conducting research in the field.

CCS Concepts: • **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability.

Keywords: datasets, neural networks, artificial intelligence, multimedia

ACM Reference Format:

. 2018. Debugging and Visualization Tools of Deep Learning – Challenges and Chances. In *Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/1122445.1122456

1 Introduction

The machine learning (ML) community currently suggests spending a large amount of time tuning parameters to find the best model. The traditional tuning process used by the community is one driven by trial and error, which mainly involves selecting the parameters that yield the best model performance through evaluation of different configurations. This "trial and error" tuning process is mostly guided by accuracy on a validation set. It is not uncommon for the time that researchers spend on tuning a model to exceed the time that they spend on developing a model's robustness by several orders of magnitude. Reducing model performance to a single number and otherwise relying on guessing definitely does not seem like an optimal process. We propose that the multimedia community can chip in here by implementing model-debugging tools and methods through extensive research. When a computer program does not exactly do what it should, we use debuggers to find the problem. Similarly, we propose that for machine learning, a model debugger should be able to examine the state at any time of the development process. We hypothesize that multimedia computing can play a large role here because rather than visualizing number series, experimenting with perceptual data such as audio files, images, or videos will be the preferred way to debug models like deep neural networks. The topic is new since most research treats deep learning as a black box and rarely questions the inner workings of deep neural networks. For the same reason the topic is also brave: much is speculated, but very little is actually known about how to examine and debug neural network weights systematically with the goal of understanding outcomes.

This article starts by discussing the state of the art and its limitations before presenting a few preliminary experiments that show the validity of this brave new topic.

2 The State of the Art

In the past few years, models have grown substantially in terms of both model size and complexity. The success of deep learning relies on the consistent growth of computation power provided by the Graphics Processing Unit (GPU). Thus people tend to make the neural network wider and deeper if the computation resources are sufficient. In the epochal AlexNet paper [10], the proposed model only has around ten layers and approximately 61 million parameters. It adopts a relatively simple neural network architecture. Its author had to specifically split the model carefully into two parts in order to fit into the GPUs. After AlexNet, GoogleLeNet [16], and VGGNet [14] marked the milestone of deep learning development. These models substantially extended the size of models and have much more complex network architectures, which are partially inspired by the human vision system. Later, the ResNet [7] model solves the problem of vanishing gradients when using deep models. With its skip connection, over-parameterized layers will not harm the model's performance. For the first time, the depth of the model reached unprecedented 152 layers.

Woodstock '18, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

However, too deep a model does not come with no cost. Potentially, over-parameterized models face the problem of overfitting to training data. It becomes more and more important to deal with problems emerging due to depth and size of the model. We consider such problems as bugs in the neural network context, although technically the model can still compile and run.. People then develop dropouts [15], ensemble training [13], distillation [8], and weight decays [11] in order to alleviate the overfitting. In addition, large models consume a lot of resources to train. In the image domain, this issue is especially prominent in generation models. This includes generative adversarial networks [6] and variational auto-encoder [9] because typically, these models need many parameters to model the distribution of the image space. For language tasks, models like the transformer [19] or BERT [4] need a lot of GPU resources due to its large query-key-value self-attention system. This problem becomes even worse for video tasks since videos consist of thousands of frames. Currently, widely used backbones such as I3D [3], contain a huge amount of parameters.

Also, another emerging problem of training complex models is hyper-parameters tuning. We consider incorrect hyper-parameters as another bug, as it leads to bad performance of the model. These bugs are hard to deal with. Hyper-parameters refer to the parameters that control the model's architecture or training scheme. They are not learned from the training data but need to be cross-validated in the validation set. Each cross-validation of the value of a single hyper-parameter requires training the model from scratch to converge. As a result, it is extremely difficult to choose the model's architecture and hyper-parameters. People hope to construct a system that can do this automatically. These systems are typically called auto-ML. For example, google's Cloud AutoML platform provides a lot of options for users to customize their model based on some well-designed templates. Facebook's FBNet [20] tries to embed its neural network system into cell phones, and it can automatically adjust to different cellphone's configuration. In addition, Nvidia, IBM, Facebook Research, and Google Research all maintain a large Github repository that contains many pre-trained models. Users can adapt these pre-trained models to their own usage. A lot of efforts have also been made to construct models that can explore appropriate training schemes on its own such as meta-learning [18].

Ad-hoc debugging tools for machine learning are available for specialized cases both, open source and commercial. Most, if not all, of these tools only focus on visualization of the training process. This means that the programmers still have to consciously track the process of training for the neural networks and ponder what might be the problems that are causing the bugs. In addition, these visualization tools only give feedback to the user but not trace back to the root problem and do not encode any knowledge inside their code. Also, most systems can only work on a predefined template. It is still unclear how users can leverage an automatic system in developing their own models. Given the user's own model, it would be ideal to have a debugger that can provide some advice to improve the model. The "bug" in a neural network context is no longer errors that cause compile errors or trigger exceptions. Commonly used neural network frameworks such as Keras, TensorFlow, and PyTorch all have a sophisticated mechanism to prevent such mistakes, and users can quickly locate and fix them. The bug neural network context refers to mistakes that lead to poor performance despite being able to compile and run. It includes using the wrong learning rate, forgetting to call eval() mode for dropout layers, or choosing an inappropriate model architecture. This motivates us to develop machine learning debuggers.

We propose debuggers to provide easy-to-use and generic methods to fix common problems that people might encounter during machine learning model training and validation, as well as in production. We want to see open source projects in the community.

3 Features of Possible Debugging Tools

The neural network has the best performance only when its network capacity "matches" with the complexity of the problem dataset [5]. "Match" means the network capacity is equal or smaller than the dataset complexity. To help speed up the machine learning training, we include the features in our debugger to calculate the capacity of the network and the dataset complexity. In this section, we will explain the fundamental features of our debugger. In *3.1*, we briefly describe the algorithm used to estimate the dataset complexity that is being used to train machine learning models. In *3.2*, we provide an explanation on how to calculate the deep network capacity. Additionally, we provide the methods to solve the dataset unbalanced problem using data resampling strategy in *3.3*. Then we introduce the neural network robustness check in *3.4*. Finally, We discuss our current works and future works to improve our project.

3.1 Dataset Complexity Estimation

Capacity, the ability to generalize information, is one important characteristic of neural networks which is largely used by the deep learning community to quantitatively analyze the dataset and the network. We use a heuristic method [5] to estimate the dataset complexity for a given network. By calculating the dataset complexity, our debugger can provide a theoretical upper bound limit for the network. To have optimal performance, the network capacity must match the dataset complexity. We rely on Mackay's information theory to find out the upper and lower limits of the dataset complexity. Based on Mackay's information model, a dataset with m data points contains 2^m bits of information [12]. The

Table 1. Results of eight different datasets' CSG Scores and expected complexity using heuristic method;

Dataset	Input Dimension	CSG Score	Exepcted Complexity
CIFAR-10	3072	1.396	30720
Seefood	3072	0.660	21504
Spiral	2	0.431	12
MNIST	784	0.428	7840
XOR	2	0.00609	10
Circle	2	0	10
Gauss	2	0	2
Moon	2	0	10

easiest way to solve this binary classification problem is to use a dummy neural network to memorize the dataset which has 2^m bits capacity. This implies that solving this binary problem by memorization, we need a dummy network of 2^m bits capacity. The 2^m bit would be the upper bound of the dataset complexity, and any network with a capacity that is larger than the upper bound would be considered the wrong model. However, in the real world, we expect the neural network to learn the hidden patterns of the dataset rather than memorize it. The upper bound provided by such dummy neural networks is useless and rarely used by researchers. We need to tighten up our upper limit by leveraging patterns that exist in the dataset using heuristic methods.

As the power of GPU/GPU increases dramatically in recent years, the number of complex datasets that are used by the deep learning community rise sharply. One challenge of training a deep network to classify these complex image datasets is to identify the dataset complexity. Although the heuristic measure provides an exact estimation of the dataset complexity, it fails to test the separability of the complex dataset. To measure the separability in the dataset, we are using the cumulative spectral gradient (CSG). Unlike the heuristic method, the CSG does not have prior assumptions on the dataset distribution, and it scales with the numbers of classes & images in the dataset [2]. The measure is built upon the probabilistic divergence between classes in a spectral clustering framework [2]. In other words, the CSG measures the speraralibity of the dataset and how likely the class labels would be correctly learned by the neural network. Additionally, the CSG could assist the research to reduce the dataset dimension, and assess the difficulty of disentangling the dataset.

3.1.1 Heuristic Method. We can squeeze the upper limit of the dataset complexity for the network by applying the heuristic method on a dataset for a given network. The intuition behind the method is that the data points share more features and should be considered as the same group and the dataset complexity should be proportional to the number of mislabeling of the group. The method defines similarity using the L1 norm. In order to squeeze the upper limit, we rely on the algorithm proposed by Gerald Friedland followed five steps [5]:

1. Calculate the dataset dimension d
2. Initialize threshold
3. Calculate the L1 norm for every data point and sort them based on the L1 norm
4. Find out the threshold which is the number of mislabeling data point
5. Estimate the dataset complexity: $expected\_capacity = log_2(\text{threshold} + 1) * d$

3.1.2 Cumulative Spectral Gradient(CSG). The CSG measure is used to capture the overall separability of the dataset. This CSG score characterizes the dataset complexity. High CSG score indicates a complex dataset, vise versa. The score is calculated by the following steps [2]:

1. Find the embedding for each image in the dataset by projecting the image into a new dimensional space. The images with similar content are closer
2. The embedding is used to calculate the class overlap of all pairs of classes
3. Use the probability distribution function to calculate the class overlap
4. Constructing a KxK similarity matrix S embodies the overall complexity of a dataset by means of class overlap
5. To extract a measure from S that summarizes the dataset complexity, we build an inner-class adjacency matrix
6. The CSG score is the cumulative maximum of the normalized eigen gaps of the above matrix

The results from Table 1 has shown that, in general, as the input dimension of the dataset increase, the CSG score and expected complexity also increase. In other word, we need a deep neural network to generalize the high input dimension dataset. Figure 1 provides visualizations of five different dataset which are considered as "easy". All of them have very low expected complexity and low CSG score, except for spiral dataset. Different from all other dataset, spiral dataset has a relatively higher CSG score and lower expected complexity. Figure 1 shows that drawing the boundary of spiral dataset is much more complex than others. CSG score correctly captures the overall separability of the dataset.

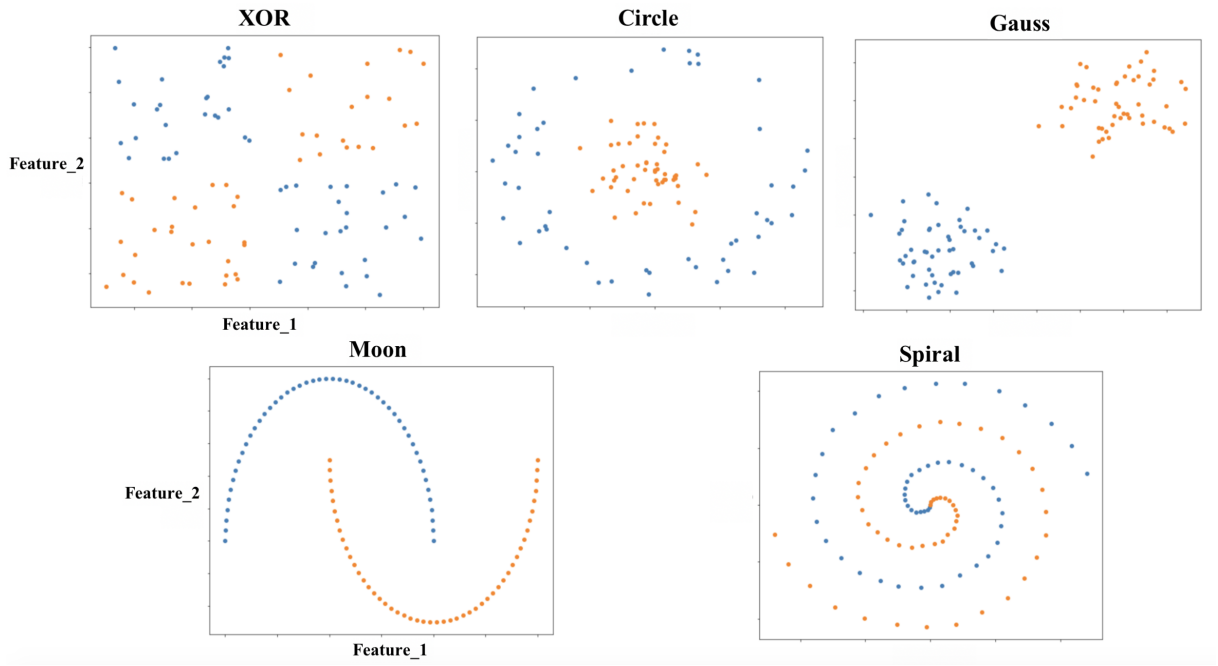

Figure 1. Visualizations of XOR, Circle, Gauss, Moon, Spiral dataset

3.2 Network Capacity Estimation

In addition to the dataset complexity, we need the network capacity to assess the performance of the network. If these two capacities do not match with each other, the network may fail to learn the dataset and solve the problem. In other words, the network capacity must match the dataset complexity for its best performance. We have chosen to implement and test the network estimation algorithms using AlexNet and ResNet due to their representative patterns and popularity. Our algorithm is built upon the four engineering rules to estimate the neural network capacity [5]:

1. The output of a single neuron yields maximally one bit of information
2. The capacity of a single neuron is the number of its parameters (weights and bias) in bits
3. The total capacity of M neurons in parallel is the capacity of each neuron at the current layer
4. For network contains multiple layers, the capacity of a subsequent layer cannot be larger than the output of previous layer

The typical deep network such as AlexNet comprises convolutional layers, max pooling layers, and forwarding layers. The memory capacity of each neurons is defined by the above rules. These neurons' capacities are added up to find the layer capacity. The network capacity calculation is the sum over each layer's capacity across the network, as Figure 2 illustrates. The pseudo-code for the network capacity estimation is shown as followed.

Algorithm 1: Model Capacity Estimation

```
Data: Model M
Result: Integer Capacity in bits totalCapacity

outputDic ← {} #The output capacity of each layer
bottleneckCapacity ← ∞
totalCapacity ← 0
for layer ∈ M.layers do
    #To store previous layers linked to the current
     layer
    connections=[]
    for inboundLayer ∈ M.inboundNodes do
        connections.append(inboundLayer.name)
    capacity ← 0
    if connections is not empty then
        for inboundLayer ∈ M.inboundNodes do
            #The input capacity is the maxium output
             capacity of previous layers
            capacity ←
             Max(capacity,outputDic[inboundLayer])
    else
        capacity ← ∞
    capacity ← Min(capacity,layer.countParams())
    outputBits ← Sum(layer.output.shape)
    bottleneckCapacity ←
     Min(outputBits,bottleneckCapacity)
    outputDic[layer.name] ← bottleneckCapacity
    totalCapacity ← totalCapacitycapacity
return totalCapacity
```

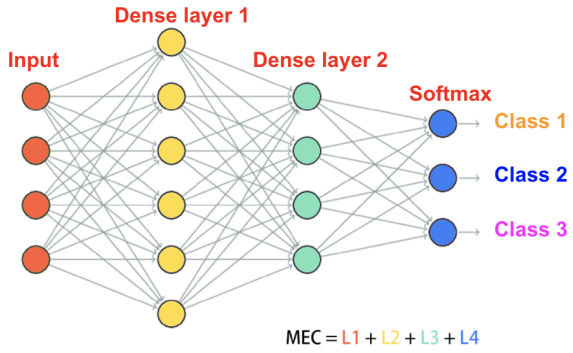

Figure 2. The maximum expected capacity for the AlexNet is acquired by counting the neuron layer by layer

3.3 Dataset Resampling

When it comes to the issue of classification, the best method to analyse it is to start from exploratory analysis, aka, Exploratory Data Analysis (EDA). During the process of analysing a dataset, the unbalanced category is a common problem. The unbalanced categories may cause the model to perform a biased training process. Tasks in real life that have an imbalanced issue are click-through rate prediction, fraud detection, disease diagnosis etc. For example, when it comes to click-through rate prediction. For example, the ratio between the unclicked items and clicked items is 57 : 1.

When facing an unbalanced dataset, trained machine learning model classifiers tend to have poor performance. For any unbalanced data, if the ratio of the class that we are trying to predict is less than 5%, then we call it a rare event. For the unbalanced datasets, although their positive samples are rare, they carry more useful information in the dataset. For example, when dealing with fraud detection, the regular transaction is very common while the fraud transaction is rare. However, it is important that we discover these minorities. Therefore, when we train the model, these important data could be underrepresented and therefore the model would have a poor performance. However, when we increase the ratio of the minority classes, the metrics of the model (like Area Under Curve) improves significantly. But most of the time, we could not acquire more minority real samples. Fortunately, there are two basic methods that could deal with these kinds of problems, oversampling and under-sampling.

3.3.1 Over Sampling - SMOTE. Synthetic Minority Oversampling Technique (SMOTE)[1] solves unbalanced problems by adding artificial minority samples to the original dataset. The core idea of SMOTE is to make synthetic samples in the minority classes. This is much better than random under sample method for it relieves model over-fitting. More specifically, we use K-Nearest Neighbor for a minority sample X_i, and we calculate the k nearest minority samples of X_i. Then we randomly choose one sample and generate a new sample in

$$X_{new} = X_i + (\hat{X}_i - X_i) \cdot \delta$$

where $\hat{X}_i$ is the K-Nearest point and $\delta \in [0, 1]$ is a random number.

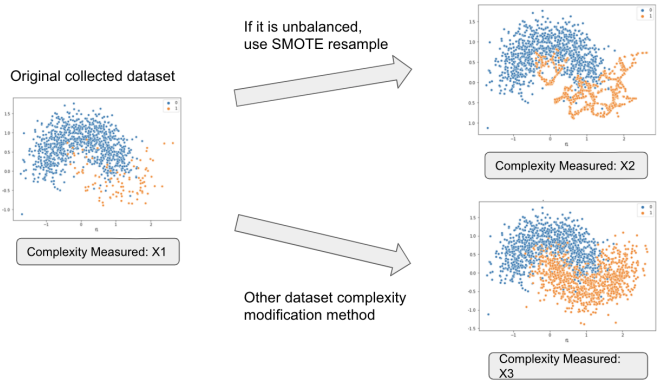

Figure 3. Visualization of unbalanced dataset using resampling techniques

3.3.2 Under Sampling - Tomek Link. Tomek Link[17] is a method to do under-sampling. Suppose sample X_i and X_j belong to different categories. $D(X_i, X_j)$ denotes the distances between these two. We name (X_i, X_j) to be a Tomek Link pair if there is not a third sample X_k that made $D(X_k, X_i) < D(X_i, X_j)$ or $D(X_k, X_j) < D(X_i, X_j)$. Apparently, when there is a Tomek Link pair, one or both of them are noise samples. For the under-sampling method, we simply delete the sample in the Tomek Link pair when it belongs to the majority category.

3.3.3 Sampling Combined with Data Capacity. We have discussed the two sampling methods before, in our tools, we proposed a new approach to do data resampling and it combines with our dataset complexity estimation algorithm. The algorithm could be described as given a dataset T, we first shuffle the dataset, and then do over-sampling or under-sampling methods, and we evaluate the capacity of the resampled dataset. We repeat this process until we get the minimum capacity. This is to ensure that our resampling method gets the local optimal result. The resampled dataset relieves the issue of unbalanced categories and provides a better generalization for the models.

3.4 Adversarial Attacks and Network Vulnerability

One of the important concepts to determine the performance of a Neural Network is its robustness. Neural Network models can be attacked easily when they are not robust enough. For example, one might modify a picture of a panda by one or two pixels so that the overall photo doesn't appear different to the human eye, but as a result, the neural network misidentifies the image as a gorilla or other animal. Such cases can

be dangerous in real world applications, if some details of signal lights are modified in some ways to attack the autonomous driving system, the system might think it should stop or accelerate when it supposes to do the opposite. These kinds of bugs in the neural network are dangerous and hard to find. Our professor has the theory about shrinking the neural networks to make it more robust while maintaining its accuracy when seeing the training data.

In the experiments, we take different sizes of two different models one is ResNet and the other one is AlexNet. We did the experiments on a simple dataset cifar 10. First we would use the methods above to calculate the complexity of the dataset. Then we would create ResNet and AlexNet from TensorFlow API that have about the same complexity as the dataset. Then we would train the model with the dataset and test the accuracy on the test sets. After the first round of training, we attack the model with another library called CleverHans with selected attack and calculate the difficulties to successfully attack the model in pixels. The more pixels we need to modify to attack the model, the harder the attack is. And the harder the attack can succeed means the more robust the model is. Then we shrink the models' capacity to be half of its original capacity by modifying the model structure and then we trained the model and attacked it. Then we would increase the capacity to be 75% of the original one. Overall, it was like a binary search to find the optimal results. We repeat these methods for the two different models with different capacities.

Some of the results were quite promising and others are not. In AlexNet it was quite clear that the smaller the capacity of the model, the more robust the model will be. After reaching some boundary, robustness will still decrease for AlexNet when the model becomes simpler and simpler. This shows that the theory of our professor might be correct about the correlation of robustness and the capacity and the method we use to find the optimal capacity number is useful. However, in ResNet the robustness of the model was changing as we tried to shrink the model's capacity. It might be the case that our dataset was too simple or the training methods we use was not optimal.

4 Discussion

We have introduced the heuristic method in *3.1.1*. It is used to estimate the dataset complexity using threshold would help lower the upper bound and provide a more efficient and effective way to analyze the dataset complexity for a given neural network. The heuristic method could be applied to estimate any classified problem dataset that needs a neural network to solve the problem. However, the heuristic method will fail if the dataset is very unbalanced.

To help balance the dataset so we could better estimate its complexity, we introduce data resampling. Although data resampling is not a silver bullet for all unbalanced issues, it performs well in most of the dataset. The main tasks are to determine whether the dataset is imbalanced and how the model is affected by the unbalanced problem. Resampling is one strategy used to accomplish the tasks. A good resampling would greatly facilitate the performance of the model, while a bad resampling may even introduce worse bias into the origin dataset. It should always be careful to decide which resample strategy we should take. There are many other brilliant methods for data resampling. In the future, we may try to build more data resampling features into our debugger and include more resampling techniques in each training iteration.

Network architecture can greatly affect the performance of our network capacity estimation algorithm. For specific architecture composition selections, there are no universal principles to follow. People usually follow consensus (such as convolution at the beginning, fully connected layers at the end, using batch normalization, dropouts, etc). Although our debugger provides a method to select the same architecture of different sizes, it cannot replace this empirical knowledge. That is why we offer several categories of the architecture of the same capacity in our model warehouse for users to select.

Additionally, the neural network capacity we calculate is only accurate in terms of VC dimension according to McKay's theory [12]. To have the best performance, the dataset complexity must match the model capacity. The matching process between these two capacities may introduce bias due to the limits of the estimation algorithm. We can add a correcting process to subtract such bias. Figure 3 shows the process to remove the bias which increases the model accuracy dramatically. In the future work, we may explore more statistical approaches to estimate and to remove such biase.

References

[1] Kevin W. Bowyer et al. "SMOTE: Synthetic Minority Over-sampling Technique". In: *CoRR* abs/1106.1813 (2011). arXiv: 1106.1813. URL: http://arxiv.org/abs/1106.1813.

[2] Frederic Branchaud-Charron, Andrew Achkar, and Pierre-Marc Jodoin. "Spectral Metric for Dataset Complexity Assessment". In: *CoRR* abs/1905.07299 (2019). arXiv: 1905.07299. URL: http://arxiv.org/abs/1905.07299.

[3] João Carreira and Andrew Zisserman. "Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset". In: *CoRR* abs/1705.07750 (2017). arXiv: 1705.07750. URL: http://arxiv.org/abs/1705.07750.

[4] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *CoRR* abs/1810.04805 (2018). arXiv: 1810.04805. URL: http://arxiv.org/abs/1810.04805.

[5] Gerald Friedland, Alfredo Metere, and Mario Michael Krell. "A Practical Approach to Sizing Neural Networks". In: *CoRR* abs/1810.02328 (2018). arXiv: 1810.02328. URL: http://arxiv.org/abs/1810.02328.

[6] Ian J. Goodfellow et al. "Generative Adversarial Networks". In: *ArXiv* abs/1406.2661 (2014).

[7] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

[8] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. "Distilling the Knowledge in a Neural Network". In: *ArXiv* abs/1503.02531 (2015).

[9] Diederik P. Kingma and Max Welling. "Auto-Encoding Variational Bayes". In: *CoRR* abs/1312.6114 (2014).

[10] Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Neural Information Processing Systems* 25 (Jan. 2012). DOI: 10.1145/3065386.

[11] Anders Krogh and John A. Hertz. "A Simple Weight Decay Can Improve Generalization". In: *Advances in Neural Information Processing Systems 4*. Ed. by J. E. Moody, S. J. Hanson, and R. P. Lippmann. Morgan-Kaufmann, 1992, pp. 950–957. URL: http://papers.nips.cc/paper/563-a-simple-weight-decay-can-improve-generalization.pdf.

[12] David J.C. MacKay. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2005. ISBN: 0521642981.

[13] Richard Maclin and David W. Opitz. "Popular Ensemble Methods: An Empirical Study". In: *CoRR* abs/1106.0257 (2011). arXiv: 1106.0257. URL: http://arxiv.org/abs/1106.0257.

[14] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *arXiv 1409.1556* (Sept. 2014).

[15] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[16] Christian Szegedy et al. "Going Deeper with Convolutions". In: *CoRR* abs/1409.4842 (2014). arXiv: 1409.4842. URL: http://arxiv.org/abs/1409.4842.

[17] Ivan Tomek. "Two Modifications of CNN". In: 1976.

[18] Joaquin Vanschoren. "Meta-Learning: A Survey". In: *CoRR* abs/1810.03548 (2018). arXiv: 1810.03548. URL: http://arxiv.org/abs/1810.03548.

[19] Ashish Vaswani et al. "Attention is All you Need". In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon et al. Curran Associates, Inc., 2017, pp. 5998–6008. URL: http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf.

[20] Bichen Wu et al. "FBNet: Hardware-Aware Efficient ConvNet Design via Differentiable Neural Architecture Search". In: *CoRR* abs/1812.03443 (2018). arXiv: 1812.03443. URL: http://arxiv.org/abs/1812.03443.