Constructing Deep Neural Networks by Bayesian Network Structure Learning

Raanan Y. Rohekar
Intel AI Lab
raanan.yehezkel@intel.com

Shami Nisimov
Intel AI Lab
shami.nisimov@intel.com

Yaniv Gurwicz
Intel AI Lab
yaniv.gurwicz@intel.com

Guy Koren
Intel AI Lab
guy.koren@intel.com

Gal Novik
Intel AI Lab
gal.novik@intel.com

Abstract

We introduce a principled approach for unsupervised structure learning of deep neural networks. We propose a new interpretation for depth and inter-layer connectivity where conditional independencies in the input distribution are encoded hierarchically in the network structure. Thus, the depth of the network is determined inherently. The proposed method casts the problem of neural network structure learning as a problem of Bayesian network structure learning. Then, instead of directly learning the discriminative structure, it learns a generative graph, constructs its stochastic inverse, and then constructs a discriminative graph. We prove that conditional-dependency relations among the latent variables in the generative graph are preserved in the class-conditional discriminative graph. We demonstrate on image classification benchmarks that the deepest layers (convolutional and dense) of common networks can be replaced by significantly smaller learned structures, while maintaining classification accuracy—state-of-the-art on tested benchmarks. Our structure learning algorithm requires a small computational cost and runs efficiently on a standard desktop CPU.

1 Introduction

Over the last decade, deep neural networks have proven their effectiveness in solving many challenging problems in various domains such as speech recognition (Graves & Schmidhuber, 2005), computer vision (Krizhevsky et al., 2012; Girshick et al., 2014; Szegedy et al., 2015) and machine translation (Collobert et al., 2011). As compute resources became more available, large scale models having millions of parameters could be trained on massive volumes of data, to achieve state-of-the-art solutions. Building these models requires various design choices such as network topology, cost function, optimization technique, and the configuration of related hyper-parameters.

In this paper, we focus on the design of network topology—structure learning. Generally, exploration of this design space is a time consuming iterative process that requires close supervision by a human expert. Many studies provide guidelines for design choices such as network depth (Simonyan & Zisserman, 2014), layer width (Zagoruyko & Komodakis, 2016), building blocks (Szegedy et al., 2015), and connectivity (He et al., 2016; Huang et al., 2016). Based on these guidelines, these studies propose several meta-architectures, trained on huge volumes of data. These were applied to other tasks by leveraging the representational power of their convolutional layers and fine-tuning their deepest layers for the task at hand (Donahue et al., 2014; Hinton et al., 2015; Long et al., 2015; Chen et al., 2015; Liu et al., 2015). However, these meta-architectures may be unnecessarily large and require large computational power and memory for training and inference.

The problem of model structure learning has been widely researched for many years in the probabilistic graphical models domain. Specifically, Bayesian networks for density estimation and causal discovery (Pearl, 2009; Spirtes et al., 2000). Two main approaches were studied: score-based and constraint-based. Score-based approaches combine a scoring function, such as BDe (Cooper & Herskovits, 1992), with a strategy for searching in the space of structures, such as greedy equivalence search (Chickering, 2002). Adams et al. (2010) introduced an algorithm for sampling deep belief networks (generative model) and demonstrated its applicability to high-dimensional image datasets.

Constraint-based approaches (Pearl, 2009; Spirtes et al., 2000) find the optimal structures in the large sample limit by testing conditional independence (CI) between pairs of variables. They are generally faster than score-based approaches (Yehezkel & Lerner, 2009) and have a well-defined stopping criterion (e.g., maximal order of conditional independence). However, these methods are sensitive to errors in the independence tests, especially in the case of high-order CI tests and small training sets.

Motivated by these methods, we propose a new interpretation for depth and inter-layer connectivity in deep neural networks. We derive a structure learning algorithm such that a hierarchy of independencies in the input distribution is encoded in a deep generative graph, where lower-order independencies are encoded in deeper layers. Thus, the number of layers is automatically determined, which is a desirable virtue in any architecture learning method. We then convert the generative graph into a discriminative graph, demonstrating the ability of the latter to mimic (preserve conditional dependencies) of the former. In the resulting structure, a neuron in a layer is allowed to connect to neurons in deeper layers skipping intermediate layers. This is similar to the shortcut connection (Raiko et al., 2012), while our method derives it automatically. Moreover, neurons in deeper layers represent low-order (small condition sets) independencies and have a wide scope of the input, whereas neurons in the first layers represent higher-order (larger condition sets) independencies and have a narrower scope. An example of a learned structure, for MNIST, is given in Figure 1 (X are image pixels).
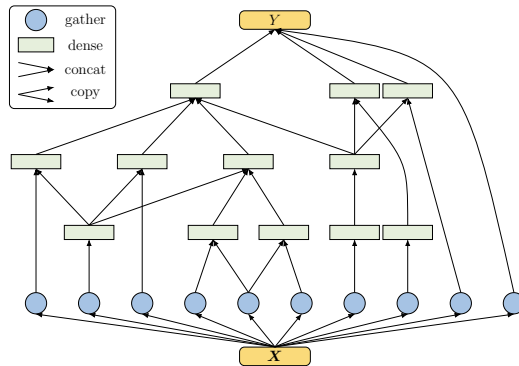

Figure 1: An example of a structure learned by our algorithm (classifying MNIST digits, 99.07% accuracy). Neurons in a layer may connect to neurons in any deeper layer. Depth is determined automatically. Each gather layer selects a subset of the input, where each input variable is gathered only once. A neural route, starting with a gather layer, passes through densely connected layers where it may split (copy) and merge (concatenate) with other routes in correspondence with the hierarchy of independencies identified by the algorithm. All routes merge into the final output layer.

The paper is organized as follows. We discuss related work in Section 2. In Section 3 we describe our method and prove its correctness in supplementary material Sec. A. We provide experimental results in Section 4, and conclude in Section 5.

2   Related Work

Recent studies have focused on automating the exploration of the design space, posing it as a hyper-parameter optimization problem and proposing various approaches to solve it. Miconi (2016) learns the topology of an RNN introducing structural parameters into the model and optimizing them along with the model weights by the common gradient descent methods. Smith et al. (2016) take a similar approach incorporating the structure learning into the parameter learning scheme, gradually growing the network up to a maximum size.

A common approach is to define the design space in a way that enables a feasible exploration process and design an effective method for exploring it. Zoph & Le (2016) (NAS) first define a set of hyper-parameters characterizing a layer (number of filters, kernel size, stride). Then they use a controller-RNN for finding the optimal sequence of layer configurations for a "trainee network". This is done using policy gradients (REINFORCE) for optimizing the objective function that is based on the accuracy achieved by the "trainee" on a validation set. Although this work demonstrates capabilities to solve large-scale problems (Imagenet), it comes with huge computational cost. In a following work, Zoph et al. (2017) address the same problem but apply a hierarchical approach. They use NAS to design network modules on a small-scale dataset (CIFAR-10) and transfer this knowledge to a large-scale problem by learning the optimal topology composed of these modules. Baker et al. (2016) use reinforcement learning as well and apply Q-learning with epsilon-greedy exploration strategy and experience replay. Negrinho & Gordon (2017) propose a language that allows a human expert to compactly represent a complex search-space over architectures and hyper-parameters as a tree and then use methods such as MCTS or SMBO to traverse this tree. Smithson et al. (2016) present a multi objective design space exploration, taking into account not only the classification accuracy but also the computational cost. In order to reduce the cost involved in evaluating the network's accuracy, they train a Response Surface Model that predicts the accuracy at a much lower cost, reducing the number of candidates that go through actual validation accuracy evaluation. Another common approach for architecture search is based on evolutionary strategies to define and search the design space. Real et al. (2017) and Miikkulainen et al. (2017) use evolutionary algorithm to evolve an initial model or blueprint based on its validation performance.

Common to all these recent studies is the fact that structure learning is done in a supervised manner, eventually learning a discriminative model. Moreoever, these approaches require huge compute resources, rendering the solution unfeasible for most applications given limited compute and time.

3   Proposed Method

Preliminaries. Consider $\boldsymbol{X} = \{X_i\}_{i=1}^N$ a set of observed (input) random variables, $\boldsymbol{H}$ a set of latent variables, and Y a target (classification or regression) variable. Each variable is represented by a single node, and a single edge connects two distinct nodes. The parent set of a node X in G is denoted $\mathtt{Pa}(X; G)$, and the children set is denoted $\mathtt{Ch}(X; G)$. Consider four graphical models, G, G_{inv}, G_{dis}, and g_X. Graph G is a generative DAG defined over $\boldsymbol{X} \cup \boldsymbol{H}$, where $\mathtt{Ch}(X; G) = \emptyset, \forall X \in \boldsymbol{X}$. Graph G can be described as a layered deep Bayesian network where the parents of a node can be in any deeper layer and not restricted to the previous layer[1]. In a graph with m latent layers, we index the deepest layer as 0 and the layer connected to the input as $m - 1$. The root nodes (parentless) are latent, $\boldsymbol{H}^{(0)}$, and the leaves (childless) are the observed nodes, $\boldsymbol{X}$, and $\mathtt{Pa}(\boldsymbol{X}; G) \subset \boldsymbol{H}$. Graph G_{inv} is called a stochastic inverse of G, defined over $\boldsymbol{X} \cup \boldsymbol{H}$, where $\mathtt{Pa}(X; G_{\mathrm{inv}}) = \emptyset, \forall X \in \boldsymbol{X}$. Graph G_{dis} is a discriminative graph defined over $\boldsymbol{X} \cup \boldsymbol{H} \cup Y$, where $\mathtt{Pa}(X; G_{\mathrm{dis}}) = \emptyset, \forall X \in \boldsymbol{X}$ and $\mathtt{Ch}(Y; G_{\mathrm{dis}}) = \emptyset$. Graph g_X is a CPDAG (a family of Markov equivalent Bayesian networks) defined over $\boldsymbol{X}$. Graph g_X is generated and maintained as an internal state of the algorithm, serving as an auxiliary graph. The order of an independence relation between two variables is defined to be the condition set size. For example, if X_1 and X_2 are independent given X_3, X_4, and X_5 (d-separated in the faithful DAG $X_1 \perp\!\!\!\perp X_2 | \{X_3, X_4, X_5\}$), then the independence order is $|\{X_3, X_4, X_5\}| = 3$.

3.1   Key Idea

We cast the problem of learning the structure of a deep neural network as a problem of learning the structure of a deep (discriminative) probabilistic graphical model, G_{dis}. That is, a graph of the form $\boldsymbol{X} \rightsquigarrow \boldsymbol{H}^{(m-1)} \rightsquigarrow \cdots \rightsquigarrow \boldsymbol{H}^{(0)} \rightarrow Y$, where "$\rightsquigarrow$" represent a sparse connectivity which we learn, and "$\rightarrow$" represents full connectivity. The joint probability factorizes as $P(\boldsymbol{X})P(\boldsymbol{H}|\boldsymbol{X})P(Y|\boldsymbol{H}^{(0)})$ and the posterior is $P(Y|\boldsymbol{X}) = \int P(\boldsymbol{H}|\boldsymbol{X})P(Y|\boldsymbol{H}^{(0)})d\boldsymbol{H}$, where $\boldsymbol{H} = \{\boldsymbol{H}^{(i)}\}_0^{m-1}$. We refer to the $P(\boldsymbol{H}|\boldsymbol{X})$ part of the equation as the *recognition network* of an unknown "true" underlying generative model, $P(\boldsymbol{X}|\boldsymbol{H})$. That is, the network corresponding to $P(\boldsymbol{H}|\boldsymbol{X})$ approximates the posterior (e.g., as in amortized inference). The key idea is to approximate the latents $\boldsymbol{H}$ that

---

[1]This differs from the common definition of deep belief networks (Hinton et al., 2006; Adams et al., 2010) where the parents are restricted to the next layer.

generated the observed $\boldsymbol{X}$, and then use these values of $\boldsymbol{H}^{(0)}$ for classification. That is, avoid learning G_{dis} directly and instead, learn a generative structure $\boldsymbol{X} \leftsquigarrow \boldsymbol{H}$, and then reverse the flow by constructing a stochastic inverse (Stuhlmüller et al., 2013) $\boldsymbol{X} \rightsquigarrow \boldsymbol{H}$. Finally, add Y and modify the graph to preserve conditional dependencies (G_{dis} can mimic G; G_{dis} does not include sparsity that is not supported by G). Lastly, G_{dis} is converted into a deep neural network by replacing each latent variable by a neural layer. We call this method B2N (Bayesian to Neural), as it learns the connectivity of a deep neural network through Bayesian network structure.

3.2 Constructing a Deep Generative Graph

The key idea of constructing G, the generative graph, is to recursively introduce a new latent layer, $\boldsymbol{H}^{(n)}$, after testing n-th order conditional independence in $\boldsymbol{X}$, and connect it, as a parent, to latent layers created by subsequent recursive calls that test conditional independence of order $n+1$. To better understand why deeper layer represent smaller condition independence sets, consider an ancestral sampling of the generative graph. First, the values of nodes in the deepest layer, corresponding to marginal independence, are sampled—each node is sampled *independently*. In the next layer, nodes can be sampled independently given the values of deeper nodes. This enables gradually factorizing ("disentangling") the joint distribution over $\boldsymbol{X}$. Hence, approximating the values of latents, $\boldsymbol{H}$, in the deepest layer provides us with statistically *independent* features of the data, which can be fed in to a single layer linear classifier. Yehezkel & Lerner (2009) introduced an efficient algorithm (RAI) for constructing a CPDAG over $\boldsymbol{X}$ by a recursive application of conditional independence tests with increasing condition set sizes. Our algorithm is based on this framework for testing independence in $\boldsymbol{X}$ and updating the auxiliary graph g_X.

Our proposed recursive algorithm for constructing G, is presented in Algorithm 1 (DeepGen) and a flow chart is shown in the supplementary material Sec. B. The algorithm starts with condition set $n = 0$, g_X a complete graph (defined over $\boldsymbol{X}$), and a set of exogenous nodes, $\boldsymbol{X}_{\mathrm{ex}} = \emptyset$. The set $\boldsymbol{X}_{\mathrm{ex}}$ is exogenous to g_X and consists of parents of $\boldsymbol{X}$. Note that there are two exit points, lines 4 and 14. Also, there are multiple recursive calls, lines 8 (within a loop) and 9, leading to multiple parallel recursive-traces, which will construct multiple generative flows rooted at some deeper layer.

The algorithm starts by testing the exit condition (line 2). It is satisfied if there are not enough nodes in $\boldsymbol{X}$ for a condition set of size n. In this case, the maximal depth is reached and an empty graph is returned (a layer composed of observed nodes). From this point, the recursive procedure will trace back, adding latent parent layers.

Algorithm 1: $G \longleftarrow \mathrm{DeepGen}(g_X, \boldsymbol{X}, \boldsymbol{X}_{\mathrm{ex}}, n)$

---

1 **DeepGen** $(g_X, \boldsymbol{X}, \boldsymbol{X}_{\mathrm{ex}}, n)$
    Input: an initial CPDAG g_X over endogeneous $\boldsymbol{X}$ & exogenous $\boldsymbol{X}_{\mathrm{ex}}$ observed nodes, and a desired resolution n.
    Output: G, a latent structure over $\boldsymbol{X}$ and $\boldsymbol{H}$

2   **if** *the maximal indegree of $g_X(\boldsymbol{X})$ is below $n + 1$* **then**   ▷ exit condition
3      $G \longleftarrow$ an empty graph over $\boldsymbol{X}$ ▷ create a gather layer
4   **return** G

5    $g_X' \longleftarrow \mathtt{IncSeparation}(g_X, n)$ ▷ n-th order independencies
6    $\{\boldsymbol{X}_{\mathrm{D}}, \boldsymbol{X}_{\mathrm{A}1}, \ldots, \boldsymbol{X}_{\mathrm{A}k}\} \longleftarrow \mathtt{SplitAutonomous}(\boldsymbol{X}, g_X')$ ▷ identify autonomies
7   **for** $i \in \{1 \ldots k\}$ **do**
8      $G_{\mathrm{A}i} \longleftarrow \mathtt{DeepGen}(g_X', \boldsymbol{X}_{\mathrm{A}i}, \boldsymbol{X}_{\mathrm{ex}}, n + 1)$ ▷ a recursive call
9    $G_{\mathrm{D}} \longleftarrow \mathtt{DeepGen}(g_X', \boldsymbol{X}_{\mathrm{D}}, \boldsymbol{X}_{\mathrm{ex}} \cup \{\boldsymbol{X}_{\mathrm{A}i}\}_{i=1}^k, n + 1)$ ▷ a recursive call
10    $G \longleftarrow (\bigcup_{i=1}^k G_{\mathrm{A}i}) \cup G_D$ ▷ merge results
11   Create in G, k latent nodes $\boldsymbol{H}^{(n)} = \{H_1^{(n)}, \ldots, H_k^{(n)}\}$ ▷ create a latent layer

12   Let $\boldsymbol{H}_{\mathrm{A}i}^{(n+1)}$ and $\boldsymbol{H}_{\mathrm{D}}^{(n+1)}$ be the sets of parentless nodes in $G_{\mathrm{A}i}$ and G_{D}, respectively.
13   Set each $H_i^{(n)}$ to be a parent of $\{\boldsymbol{H}_{\mathrm{A}i}^{(n+1)} \cup \boldsymbol{H}_{\mathrm{D}}^{(n+1)}\}$ ▷ connect
14   **return** G

The procedure `IncSeparation` (line 5) disconnects (in g_X) conditionally independent variables in two steps. First, it tests dependency between $\boldsymbol{X}_{\text{ex}}$ and $\boldsymbol{X}$, i.e., $X \perp\!\!\!\perp X'|\boldsymbol{S}$ for every connected pair $X \in \boldsymbol{X}$ and $X' \in \boldsymbol{X}_{\text{ex}}$ given a condition set $\boldsymbol{S} \subset \{\boldsymbol{X}_{\text{ex}} \cup \boldsymbol{X}\}$ of size n. Next, it tests dependency within $\boldsymbol{X}$, i.e., $X_i \perp\!\!\!\perp X_j|\boldsymbol{S}$ for every connected pair $X_i, X_j \in \boldsymbol{X}$ given a condition set $\boldsymbol{S} \subset \{\boldsymbol{X}_{\text{ex}} \cup \boldsymbol{X}\}$ of size n. After removing the corresponding edges, the remaining edges are directed by applying two rules (Pearl, 2009; Spirtes et al., 2000). First, v-structures are identified and directed. Then, edges are continually directed, by avoiding the creation of new v-structures and directed cycles, until no more edges can be directed. Following the terminology of Yehezkel & Lerner (2009), we say that this function increases the graph d-separation resolution from $n-1$ to n.

The procedure `SplitAutonomous` (line 6) identifies autonomous sets, one descendant set, $\boldsymbol{X}_{\text{D}}$, and k ancestor sets, $\boldsymbol{X}_{\text{A}1}, \ldots, \boldsymbol{X}_{\text{A}k}$ in two steps. First, the nodes having the lowest topological order are grouped into $\boldsymbol{X}_{\text{D}}$. Then, $\boldsymbol{X}_{\text{D}}$ is removed (temporarily) from g_X revealing unconnected sub-structures. The number of unconnected sub-structures is denoted by k and the nodes set of each sub-structure is denoted by $\boldsymbol{X}_{\text{A}i}$ ($i \in \{1 \ldots k\}$).

An autonomous set in g_X includes all its nodes' parents (complying with the Markov property) and therefore a corresponding latent structure can be further learned independently, using a recursive call. Thus, the algorithm is called recursively and independently for the k ancestor sets (line 8), and then for the descendant set, treating the ancestor sets as exogenous (line 9). This recursive decomposition of $\boldsymbol{X}$ is illustrated in Figure 2. Each recursive call returns a latent structure for each autonomous set. Recall that each latent structure encodes a generative distribution over the observed variables where layer $\boldsymbol{H}^{(n+1)}$, the last added layer (parentless nodes), is a representation of some input subset $\boldsymbol{X}' \subset \boldsymbol{X}$. Thus, latent variables, $\boldsymbol{H}^{(n)}$, are introduced as parents of the $\boldsymbol{H}^{(n+1)}$ layers (lines 11–13).

It is important to note that conditional independence is tested only between input variables, $\boldsymbol{X}$, and condition sets do not include latent variables. Conditioning on latent variables or testing independence between them is not required by our approach. A 2-layer toy-example is given in Figure 3.
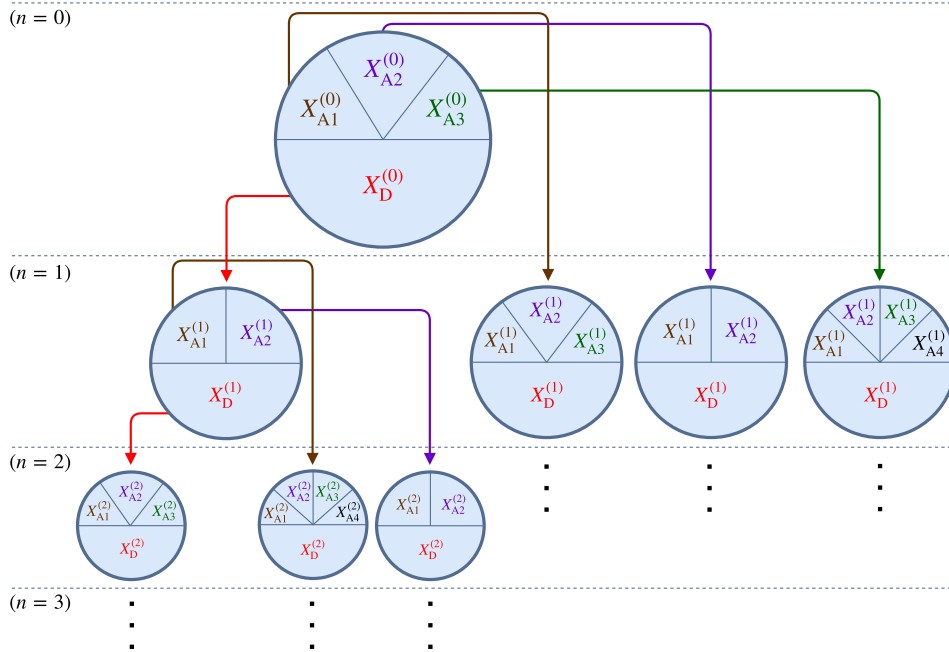

Figure 2: An example of a recursive decomposition of the observed set, $\boldsymbol{X}$. Each circle represents a distinct subset of observed variables (e.g., $\boldsymbol{X}_{\text{A}1}^{(1)}$ in different circles represents different subsets). At $n = 0$, a single circle represents all the variables. Each set of variables is split into autonomous ancestors $\boldsymbol{X}_{\text{A}i}^{(n)}$ and descendent $\boldsymbol{X}_{\text{D}}^{(n)}$ subsets. An arrow indicates a recursive call. Best viewed in color.
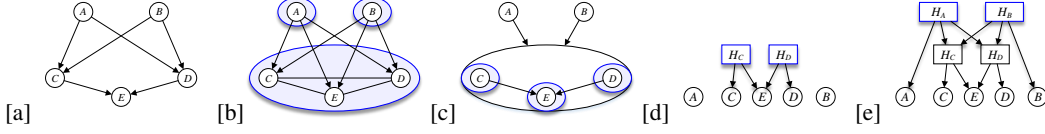
5

Figure 3: An example of learning a 2-layer generative model. [a] An example Bayesian network encoding the underlying independencies in X. [b] g_X after marginal independence testing ($n = 0$). Only A and B are marginally independent ($A \perp\!\!\!\perp B$). [c] g_X after a recursive call to learn the structure of nodes $\{C, D, E\}$ with $n = 2$ ($C \perp\!\!\!\perp D | \{A, B\}$). Exit condition is met in subsequent recursive calls and thus latent variables are added to G at $n = 2$ [d], and then at $n = 0$ [e] (the final structure).

3.3 Constructing a Discriminative Graph

We now describe how to convert G into a discriminative graph, G_{dis}, with target variable, Y (classification/regression). First, we construct G_{inv}, a graphical model that preserves all conditional dependencies in G but has a different node ordering in which the observed variables, X, have the highest topological order (parentless)—a stochastic inverse of G. Stuhlmüller et al. (2013) and Paige & Wood (2016) presented a heuristic algorithm for constructing such stochastic inverses. However, limiting G_{inv} to a DAG, although preserving all conditional dependencies, may omit many independencies and add new edges between layers. Instead, we allow it to be a projection of a latent structure (Pearl, 2009). That is, we assume the presence of *additional* hidden variables Q that are not in G_{inv} but induce dependency (for example, "interactive forks" (Pearl, 2009)) among H. For clarity, we omit these variables from the graph and use bi-directional edges to represent the dependency induced by them. G_{inv} is constructed in two steps:

1. Invert the direction of all the edges in G (invert inter-layer connectivity).
2. Connect each pair of latent variables, sharing a common child in G, with a bi-directional edge.

These steps ensure the preservation of conditional dependence.

Proposition 1. *Graph G_{inv} preserves all conditional dependencies in G (i.e., $G \preceq G_{\text{inv}}$).*

Note that conditional dependencies among X are not required to be preserved in G_{inv} and G_{dis} as these are observed variables (Paige & Wood, 2016).

Finally, a discriminative graph G_{dis} is constructed by replacing the bi-directional dependency relations in G_{inv} (induced by Q) with explaining-away relations, which are provided by adding the observed class variable Y. Node Y is set in G_{dis} to be the common child of the leaves in G_{inv} (latents introduced after testing marginal independencies in X). See an example in Figure 4. This ensures the preservation of conditional dependency relations in G_{inv}. That is, G_{dis}, given Y, can mimic G_{inv}.
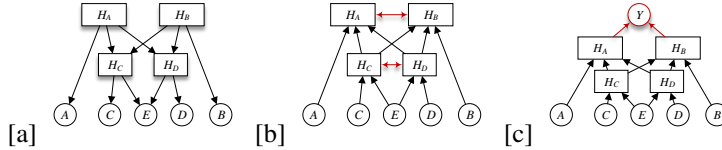

Figure 4: An example of the three graphs constructed by our algorithm: [a] a generative deep latent structure G, [b] its stochastic inverse G_{inv} (Stuhlmüller et al., 2013; Paige & Wood, 2016), and [c] a discriminative structure G_{dis} (target node Y is added).

Proposition 2. *Graph G_{dis}, conditioned on Y, preserves all conditional dependencies in G_{inv} (i.e., $G_{\text{inv}} \preceq G_{\text{dis}} | Y$).*

It follows that $G \preceq G_{\text{inv}} \preceq G_{\text{dis}}$ conditioned on Y.

Proposition 3. *Graph G_{dis}, conditioned on Y, preserves all conditional dependencies in G (i.e., $G \preceq G_{\text{dis}}$).*

Details and proofs for all the propositions are provided in supplementary material Sec. A.

6

3.4 Constructing a Feed-Forward Neural Network

We construct a neural network based on the connectivity in G_{dis}. Sigmoid belief networks (Neal, 1992) have been shown to be powerful neural network density estimators (Larochelle & Murray, 2011; Germain et al., 2015). In these networks, conditional probabilities are defined as logistic regressors. Similarly, for G_{dis} we may define for each latent variable $H' \in \boldsymbol{H}$, $p(H' = 1|\boldsymbol{X}') = \mathrm{sigm}\left(\boldsymbol{W}'\boldsymbol{X}' + b'\right)$ where $\mathrm{sigm}(x) = 1/(1 + \exp(-x))$, $\boldsymbol{X}' = \boldsymbol{Pa}(H'; G_{\mathrm{dis}})$, and $(\boldsymbol{W}', b')$ are the parameters of the neural network. Nair & Hinton (2010) proposed replacing each binary stochastic node H' by an infinite number of copies having the same weights but with decreasing bias offsets by one. They showed that this infinite set can be approximated by $\sum_{i=1}^{N} \mathrm{sigm}(v - i + 0.5) \approx \log(1 + e^v)$, where $v = \boldsymbol{W}'\boldsymbol{X}' + b'$. They further approximate this function by $\max(0, v + \epsilon)$ where ϵ is a zero-centered Gaussian noise. Following these approximations, they provide an approximate probabilistic interpretation for the ReLU function, $\max(0, v)$. As demonstrated by Jarrett et al. (2009) and Nair & Hinton (2010), these units are able to learn better features for object classification in images.

In order to further increase the representational power, we represent each H' by a set of neurons having ReLU activation functions. That is, each latent variable H' in G_{dis} is represented in the neural network by a fully-connected layer. Finally, the class node Y is represented by a softmax layer.

4 Experiments

Our structure learning algorithm is implemented using BNT (Murphy, 2001) and runs efficiently on a standard desktop CPU (excluding neural network parameter learning). For the learned structures, all layers were allocated an equal number of neurons. Threshold for independence tests, and the number of neurons-per-layer were selected by using a validation set. In all the experiments, we used ReLU activations, ADAM (Kingma & Ba, 2015) optimization, batch normalization (Ioffe & Szegedy, 2015), and dropout (Srivastava et al., 2014) to all the dense layers. All optimization hyper-parameters that were tuned for the vanilla topologies were also used, without additional tuning, for the learned structures. In all the experiments, parameter learning was repeated five times where average and standard deviation of the classification accuracy were recorded. Only test-set accuracy is reported.

4.1 Learning the Structure of the Deepest Layers in Common Topologies

We evaluate the quality of our learned structures using five image classification benchmarks and seven common topologies (and simpler hand-crafted structures), which we call "vanilla topologies". The benchmarks and vanilla topologies are described in Table 1. Similarly to Li et al. (2017), we used the VGG-16 network that was previously modified and adapted for the CIFAR-10 dataset. This VGG-16 version contains significantly fewer parameters than the original one.

Table 1: Benchmarks and vanilla topologies used in our experiments. MNIST-Man and SVHN-Man topologies were manually created by us. MNIST-Man has two convolutional layer (32 and 64 filters each) and one dense layer with 128 neurons. SVHN-Man was created as a small network reference having reasonable accuracy (Acc.) compared to Maxout-NiN.

| Dataset | Vanilla Topology | | | | |
	Id.	Topology	Description	Size	Acc.
MNIST (LeCun et al., 1998)	A	MNIST-Man	32-64-FC:128	127K	99.35
SVHN (Netzer et al., 2011)	B	Maxout NiN	(Chang & Chen, 2015)	1.6M	98.10
	C	SVHN-Man	16-16-32-32-64-FC:256	105K	97.10
CIFAR 10 (Krizhevsky & Hinton, 2009)	D	VGG-16	(Simonyan & Zisserman, 2014)	15M	92.32
	E	WRN-40-4	(Zagoruyko & Komodakis, 2016)	9M	95.09
CIFAR 100 (Krizhevsky & Hinton, 2009)	F	VGG-16	(Simonyan & Zisserman, 2014)	15M	68.86
ImageNet (Deng et al., 2009)	G	AlexNet	(Krizhevsky et al., 2012)	61M	57.20

In preliminary experiments we found that, for SVHN and ImageNet, a small subset of the training data is sufficient for learning the structure. As a result, for SVHN only the basic training data is used (without the extra data), i.e., 13% of the available training data, and for ImageNet 5% of the training data is used. Parameters were optimized using all of the training data.

Convolutional layers are powerful feature extractors for images exploiting spatial smoothness properties, translational invariance and symmetry. We therefore evaluate our algorithm by using the first convolutional layers of the vanilla topologies as "feature extractors" (mostly below 50% of the vanilla network size) and then learning a deep structure, "learned head", from their output. That is, the deepest layers of the vanilla network, "vanilla head", is removed and replaced by a structure which is learned, in an unsupervised manner, by our algorithm[2]. This results in a new architecture which we train end-to-end. Finally, a softmax layer is added and the entire network parameters are optimized.

First, we evaluate the accuracy of the learned structure as a function of the number of parameters and compare it to a densely connected network (fully connected layers) having the same depth and size (Figure 5). For SVHN, we used the Batch Normalized Maxout Network-in-Network topology (Chang & Chen, 2015) and removed the deepest layers starting from the output of the second NiN block (MMLP-2-2). For CIFAR-10, we used the VGG-16 and removed the deepest layers starting from the output of conv.7 layer. It is evident that accuracies of the learned structures are significantly higher (error bars represent 2 standard deviations) than those produced by a set of fully connected layers, especially in cases where the network is limited to a small number of parameters.
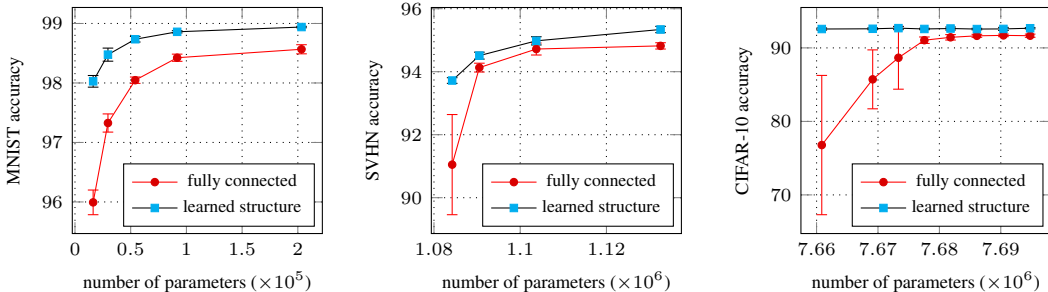

Figure 5: Classification accuracy of MNIST, SVHN, and CIFAR-10, as a function of network size. Error bars indicate two standard deviations.

Next, in Figure 6 and Table 2 we provide a summary of network sizes and classification accuracies, achieved by replacing the deepest layers of common topologies (vanilla) with a learned structure. In all the cases, the size of the learned structure is significantly smaller than that of the vanilla topology.
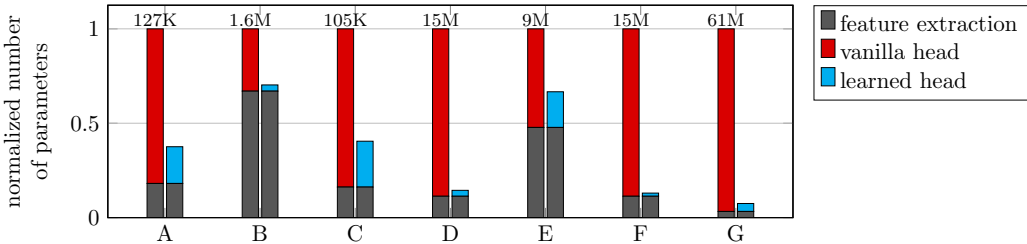

Figure 6: A comparison between the vanilla and our learned structure (B2N), in terms of normalized number of parameters. The first few layers of the vanilla topology are used for feature extraction. Stacked bars refer to either the vanilla or our learned structure. The total number of parameters of the vanilla network is indicated on top of each stacked bar.

---

[2]We also learned a structure for classifying MNIST digits directly from image pixels, without using convolutional layers for feature extraction. The resulting network structure (Figure 1), achieves an accuracy of 99.07%, whereas a network with 3 fully-connected layers achieves 98.75%.

4.2 Comparison to Other Methods

Our structure learning algorithm runs efficiently on a standard desktop CPU, while providing structures with competitive classification accuracies and network sizes. First, we compare our method to the NAS algorithm (Zoph & Le, 2016). NAS achieves for CIFAR-10 an error rate of 5.5% with a network of size 4.2M. Our method, using the feature extraction of the WRN-40-4 network, achieves this same error rate with a 26% smaller network (3.1M total size). Using the same feature extraction, the lowest classification error rate achieved by our algorithm for CIFAR 10 is 4.58% with a network of size 6M whereas the NAS algorithm achieves an error rate of 4.47% with a network of size 7.1M. Recall that the NAS algorithm requires training thousands of networks using hundreds of GPUs, which is impractical for most real-world applications.

When compared to recent pruning methods, which focus on reducing the number of parameters in a pre-trained network, our method demonstrates state-of-the-art reduction in parameters. Recently reported results are summarized in Table 3. It is important to note that although these methods prune all the network layers, whereas our method only replaces the network head, our method was found significantly superior. Moreover, pruning can be applied to the feature extraction part of the network which may further improve parameter reduction.

Table 2: Parameter reduction ratio (vanilla size/learned size) and difference in classification accuracy (Acc. Diff.=learned−vanilla, higher is better). "Full"=feature extration+head.

Id.	Acc. Diff.	Param. Reduc.	
		Full	Head
A	$+0.10 \pm 0.04$	$2.7\times$	$4.2\times$
B	-0.40 ± 0.05	$1.4\times$	$10.0\times$
C	-0.86 ± 0.05	$2.5\times$	$3.5\times$
D	$+0.29 \pm 0.14$	$7.0\times$	$28.3\times$
E	$+0.33 \pm 0.14$	$1.5\times$	$2.8\times$
F	$+0.05 \pm 0.17$	$7.7\times$	$53.2\times$
G	$+0.00 \pm 0.03$	$13.3\times$	$23.0\times$

Table 3: Parameter reduction ratio (vanilla/learned size) compared to recent pruning methods (reducing the size of a pre-trained network with *minimal* accuracy degradation). Results indicated by "acc. deg." correspond to accuracy degradation after pruning.

Network	Method	Reduction
VGG-16 (CIFAR-10)	Li et al. (2017)	$3\times$
	Ayinde & Zurada (2018)	$4.6\times$
	Ding et al. (2018)	*(acc. deg.)* $5.4\times$
	Huang et al. (2018)	*(acc. deg.)* $6\times$
	B2N (our)	$\mathbf{7\times}$
AlexNet (ImageNet)	Denton et al. (2014)	$5\times$
	Yang et al. (2015)	$3.2\times$
	Han et al. (2015, 2016)	$9\times$
	Manessi et al. (2017)	*(acc. deg.)* $12\times$
	B2N (our)	$\mathbf{13.3\times}$

5 Conclusions

We presented a principled approach for learning the structure of deep neural networks. Our proposed algorithm learns in an unsupervised manner and requires small computational cost. The resulting structures encode a hierarchy of independencies in the input distribution, where a node in one layer may connect to another node in any deeper layer, and network depth is determined automatically.

We demonstrated that our algorithm learns small structures, and maintains classification accuracies for common image classification benchmarks. It is also demonstrated that while convolution layers are very useful at exploiting domain knowledge, such as spatial smoothness, translational invariance, and symmetry, in some cases, they are outperformed by a learned structure for the deeper layers. Moreover, while the use of common topologies (meta-architectures), for a variety of classification tasks is computationally inefficient, we would expect our approach to learn smaller and more accurate networks for each classification task, uniquely.

As only unlabeled data is required for learning the structure, we expect our approach to be practical for many domains, beyond image classification, such as knowledge discovery, and plan to explore the interpretability of the learned structures. Casting the problem of learning the connectivity of deep neural network as a Bayesian network structure learning problem, enables the development of new principled and efficient approaches. This can lead to the development of new topologies and connectivity models, and can provide a greater understanding of the domain. One possible extension to our work which we plan to explore, is learning the connectivity between feature maps in convolutional layers.

References

Adams, Ryan, Wallach, Hanna, and Ghahramani, Zoubin. Learning the structure of deep sparse graphical models. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pp. 1–8, 2010.

Ayinde, Babajide O. and Zurada, Jacek M. Building efficient convnets using redundant feature pruning. In *Workshop Track of the International Conference on Learning Representations (ICLR)*, 2018.

Baker, Bowen, Gupta, Otkrist, Naik, Nikhil, and Raskar, Ramesh. Designing neural network architectures using reinforcement learning. *arXiv preprint arXiv:1611.02167*, 2016.

Chang, Jia-Ren and Chen, Yong-Sheng. Batch-normalized maxout network in network. *arXiv preprint arXiv:1511.02583*, 2015.

Chen, Tianqi, Goodfellow, Ian, and Shlens, Jonathon. Net2net: Accelerating learning via knowledge transfer. *arXiv preprint arXiv:1511.05641*, 2015.

Chickering, David Maxwell. Optimal structure identification with greedy search. *Journal of machine learning research*, 3(Nov):507–554, 2002.

Collobert, Ronan, Weston, Jason, Bottou, Léon, Karlen, Michael, Kavukcuoglu, Koray, and Kuksa, Pavel. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(Aug): 2493–2537, 2011.

Cooper, Gregory F and Herskovits, Edward. A Bayesian method for the induction of probabilistic networks from data. *Machine learning*, 9(4):309–347, 1992.

Deng, Jia, Dong, Wei, Socher, Richard, Li, Li-Jia, Li, Kai, and Fei-Fei, Li. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 248–255. IEEE, 2009.

Denton, Emily L, Zaremba, Wojciech, Bruna, Joan, LeCun, Yann, and Fergus, Rob. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in Neural Information Processing Systems*, pp. 1269–1277, 2014.

Ding, Xiaohan, Ding, Guiguang, Han, Jungong, and Tang, Sheng. Auto-balanced filter pruning for efficient convolutional neural networks. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI)*, 2018.

Donahue, Jeff, Jia, Yangqing, Vinyals, Oriol, Hoffman, Judy, Zhang, Ning, Tzeng, Eric, and Darrell, Trevor. Decaf: A deep convolutional activation feature for generic visual recognition. In *International Conference on Machine Learning*, volume 32, pp. 647–655, 2014.

Germain, Mathieu, Gregor, Karol, Murray, Iain, and Larochelle, Hugo. Made: Masked autoencoder for distribution estimation. In *ICML*, pp. 881–889, 2015.

Girshick, Ross, Donahue, Jeff, Darrell, Trevor, and Malik, Jitendra. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 580–587, 2014.

Graves, Alex and Schmidhuber, Jürgen. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks*, 18(5):602–610, 2005.

Han, Song, Pool, Jeff, Tran, John, and Dally, William. Learning both weights and connections for efficient neural networks. In *Advances in Neural Information Processing Systems*, pp. 1135–1143, 2015.

Han, Song, Mao, Huizi, and Dally, William J. Deep compression: Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

Hinton, Geoffrey, Vinyals, Oriol, and Dean, Jeff. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Hinton, Geoffrey E, Osindero, Simon, and Teh, Yee-Whye. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

Huang, Gao, Liu, Zhuang, Weinberger, Kilian Q, and van der Maaten, Laurens. Densely connected convolutional networks. *arXiv preprint arXiv:1608.06993*, 2016.

Huang, Qiangui, Zhou, Kevin, You, Suya, and Neumann, Ulrich. Learning to prune filters in convolutional neural networks. *arXiv preprint arXiv:1801.07365*, 2018.

Ioffe, Sergey and Szegedy, Christian. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pp. 448–456, 2015.

Jarrett, Kevin, Kavukcuoglu, Koray, LeCun, Yann, et al. What is the best multi-stage architecture for object recognition? In *Computer Vision, 2009 IEEE 12th International Conference on*, pp. 2146–2153. IEEE, 2009.

Kingma, Diederik and Ba, Jimmy. Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.

Krizhevsky, Alex and Hinton, Geoffrey. Learning multiple layers of features from tiny images. 2009.

Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

Larochelle, Hugo and Murray, Iain. The neural autoregressive distribution estimator. In *AISTATS*, volume 1, pp. 2, 2011.

LeCun, Yann, Bottou, Léon, Bengio, Yoshua, and Haffner, Patrick. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Li, Hao, Kadav, Asim, Durdanovic, Igor, Samet, Hanan, and Graf, Hans Peter. Pruning filters for efficient convnets. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

Liu, Baoyuan, Wang, Min, Foroosh, Hassan, Tappen, Marshall, and Pensky, Marianna. Sparse convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 806–814, 2015.

Long, Mingsheng, Cao, Yue, Wang, Jianmin, and Jordan, Michael. Learning transferable features with deep adaptation networks. In *International Conference on Machine Learning*, pp. 97–105, 2015.

Manessi, Franco, Rozza, Alessandro, Bianco, Simone, Napoletano, Paolo, and Schettini, Raimondo. Automated pruning for deep neural network compression. *arXiv preprint arXiv:1712.01721*, 2017.

Miconi, Thomas. Neural networks with differentiable structure. *arXiv preprint arXiv:1606.06216*, 2016.

Miikkulainen, Risto, Liang, Jason, Meyerson, Elliot, Rawal, Aditya, Fink, Dan, Francon, Olivier, Raju, Bala, Navruzyan, Arshak, Duffy, Nigel, and Hodjat, Babak. Evolving deep neural networks. *arXiv preprint arXiv:1703.00548*, 2017.

Murphy, K. The Bayes net toolbox for Matlab. *Computing Science and Statistics*, 33:331–350, 2001.

Nair, Vinod and Hinton, Geoffrey E. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814, 2010.

Neal, Radford M. Connectionist learning of belief networks. *Artificial intelligence*, 56(1):71–113, 1992.

Negrinho, Renato and Gordon, Geoff. Deeparchitect: Automatically designing and training deep architectures. *arXiv preprint arXiv:1704.08792*, 2017.

Netzer, Yuval, Wang, Tao, Coates, Adam, Bissacco, Alessandro, Wu, Bo, and Ng, Andrew Y. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, volume 2011, pp. 5, 2011.

Paige, Brooks and Wood, Frank. Inference networks for sequential Monte Carlo in graphical models. In *Proceedings of the 33rd International Conference on Machine Learning*, volume 48 of *JMLR*, 2016.

Pearl, Judea. *Causality: Models, Reasoning, and Inference*. Cambridge university press, second edition, 2009.

Raiko, Tapani, Valpola, Harri, and LeCun, Yann. Deep learning made easier by linear transformations in perceptrons. In *Artificial Intelligence and Statistics*, pp. 924–932, 2012.

Real, Esteban, Moore, Sherry, Selle, Andrew, Saxena, Saurabh, Suematsu, Yutaka Leon, Le, Quoc, and Kurakin, Alex. Large-scale evolution of image classifiers. *arXiv preprint arXiv:1703.01041*, 2017.

Simonyan, Karen and Zisserman, Andrew. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Smith, Leslie N, Hand, Emily M, and Doster, Timothy. Gradual dropin of layers to train very deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4763–4771, 2016.

Smithson, Sean C, Yang, Guang, Gross, Warren J, and Meyer, Brett H. Neural networks designing neural networks: Multi-objective hyper-parameter optimization. In *Computer-Aided Design (ICCAD), 2016 IEEE/ACM International Conference on*, pp. 1–8. IEEE, 2016.

Spirtes, P., Glymour, C., and Scheines, R. *Causation, Prediction and Search*. MIT Press, 2nd edition, 2000.

Srivastava, Nitish, Hinton, Geoffrey, Krizhevsky, Alex, Sutskever, Ilya, and Salakhutdinov, Ruslan. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15: 1929–1958, 2014. URL `http://jmlr.org/papers/v15/srivastava14a.html`.

Stuhlmüller, Andreas, Taylor, Jacob, and Goodman, Noah. Learning stochastic inverses. In *Advances in neural information processing systems*, pp. 3048–3056, 2013.

Szegedy, Christian, Liu, Wei, Jia, Yangqing, Sermanet, Pierre, Reed, Scott, Anguelov, Dragomir, Erhan, Dumitru, Vanhoucke, Vincent, and Rabinovich, Andrew. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–9, 2015.

Yang, Zichao, Moczulski, Marcin, Denil, Misha, de Freitas, Nando, Smola, Alex, Song, Le, and Wang, Ziyu. Deep fried convnets. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1476–1483, 2015.

Yehezkel, Raanan and Lerner, Boaz. Bayesian network structure learning by recursive autonomy identification. *Journal of Machine Learning Research*, 10(Jul):1527–1570, 2009.

Zagoruyko, Sergey and Komodakis, Nikos. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

Zoph, Barret and Le, Quoc V. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

Zoph, Barret, Vasudevan, Vijay, Shlens, Jonathon, and Le, Quoc V. Learning transferable architectures for scalable image recognition. *arXiv preprint arXiv:1707.07012*, 2017.