Exploiting Network Science for Feature Extraction and Representation Learning

Submitted in partial fulfillment of the requirements for
the degree of
Doctor of Philosophy
in
Electrical and Computer Engineering

Kartikeya Bhardwaj

Carnegie Mellon University
Pittsburgh, PA 15213

December 2019

To my father, mother, and brother

who taught me to never give up.

Acknowledgments

As we know, networks are everywhere. Indeed, the work in this thesis would not be possible without the support, guidance, and patience from an incredible *network* of researchers, friends, and family, that I had the pleasure of interacting with during the past few years at Carnegie Mellon. I would like to take this moment to sincerely thank them for their invaluable support.

First, I am deeply grateful to the guidance of my advisor Prof. Radu Marculescu, who introduced me to the very field of Network Science. He showed me the value of doing fundamental research by emphasizing the importance of *identifying new problems* throughout this journey. Thinking about new problems (instead of simply picking a problem from literature) is of course challenging, but is also significantly more rewarding as it can lead to completely new research directions. Moreover, he taught me that presenting a research properly is also as important as doing high quality research. Truly, I will carry these crucial lessons for the rest of my life. I am fortunate to work with Prof. Radu who gave me the opportunity to work on a very broad class of real-world problems and has always been there for any form of support.

Next, I would like to thank my dissertation committee members: Prof. Diana Marculescu, Prof. Shawn Blanton, and Dr. Frank Liu, for giving me consistent feedback on my work. Their excellent research also acted as a constant source of motivation and encouragement for my own research. Collaborations and discussions with their groups taught me essential things like how to formulate real-world problems (*e.g.*, how can one go from an English word problem to their mathematical formulation). These are the skills that any researcher must learn during their PhD, and are important for any profession. I am thankful that I got to learn all these things from useful collaborations/discussions. I also thank Prof. Virginia Smith for useful discussions.

I further thank my collaborators and mentors from my internship: Dr. Naveen Suda from Arm Inc., and Dr. Liangzhen Lai from Facebook, who introduced me

to new research directions. Working with these excellent researchers from industry broadened my horizons and gave me a context for making impact in the real-world.

I thank Nathan Snizaski, Judy Bandola, Holly Skovira, Shelley Phelps, and the Office of International Education for the continuous administrative support. I was never worried about any administrative issues throughout my five years.

I would be remiss if I did not mention the help from my fellow researchers and friends in the System-Level Design (SLD) group: Mr. Ching-Yi Lin and Dr. Anderson Sartor. It was a pleasure working with them on research projects. Moreover, I acknowledge the support of many Masters students who contributed to the experimental setup of various projects.

I acknowledge the funding from US National Science Foundation (NSF) under CyberSEES Grant CCF-1331804, Air Force Research Laboratory (AFRL) and Defense Advanced Research Projects Agency (DARPA) under agreement number FA-8650-18-2-7860, NVIDIA Corporation for donating a Titan Xp GPU, and Amazon support through the AWS ML Research Program.

Life, of course, would not be the same without my friends (and also some of the most brilliant researchers I have known): Prashanth Mohan, Susnata Mondal, Vignesh Balaji, Soumya Mittal, Sushant Kumar, Dipanjan Saha, Mark Blanco, and Brian Schwedock. Inventing a new variant of the game of pool with Prashanth and Susnata is a contribution of my PhD that is beyond the scope of this thesis! I also thank Dave Scheimer for being a great roommate and a friend.

Finally, I thank my father, mother, and my brother for their constant love, patience, and encouragement without which this thesis would not have been possible. When I was an undergrad, my brother Kshitij's outstanding research achievements motivated me to conduct research and pursue the PhD program. Throughout my life, my father and mother have taught me the value of putting progress before everything else; this gave me a penchant for creating new things, which I ultimately realized by enrolling into the PhD program at Carnegie Mellon. Indeed, my family has always been there for me through some difficult times. I am very fortunate to have a family which understands the effort needed for a PhD.

Abstract

Networks are ubiquitous for many real-world problems such as modeling information diffusion over social networks, transportation systems, understanding protein-protein interactions, human mobility, computational sustainability, among many others. Recently, due to the ongoing Big Data revolution, the fields of machine learning and Artificial Intelligence (AI) have also become extremely important, with AI mostly being dominated by representation learning techniques such as deep learning. However, research at the intersection of network science, machine learning and AI has been mostly unexplored. Specifically, most of the prior research focuses on how machine learning techniques can be used to solve "network" problems such as predicting information diffusion on social networks or classifying blogger interests in a blog network, *etc*. On the contrary, in this thesis, we answer the following key question: *How can we exploit network science to improve machine learning and representation learning models when addressing general problems?*

To answer the above question, we address several problems at the intersection of network science, machine learning, and AI. Specifically, we address four fundamental research challenges: (*i*) Network Science for Traditional Machine Learning, (*ii*) Representation Learning for Small-Sample Datasets, (*iii*) Network Science-Based Deep Learning Model Compression, and (*iv*) Network Science for Neural Architecture Space Exploration. In other words, we show that many problems are governed by latent network dynamics which must be incorporated into the machine learning or representation learning models.

To this end, we first demonstrate how network science can be used for traditional machine learning problems such as spatiotemporal timeseries prediction and application-specific feature extraction. More precisely, we propose a new framework called *Network-of-Dynamic Bayesian Networks* (NDBN) to address a complex probabilistic learning problem over networks with known but rapidly changing structure.

We also propose a new domain-specific network inference approach when the network structure is unknown and only the high-dimensional data is available. We further introduce a new network science-based, application-specific feature extraction method called *K-Hop Learning*. As concrete case studies, we show that both NDBN framework and K-Hop Learning significantly outperform traditional machine learning techniques for computational sustainability problems such as short-term solar energy and river flow prediction, respectively.

We then discuss how network science can be used to address general representation learning problems with high-dimensional and small-sample datasets. Here, we propose a new network community-based dimensionality reduction framework called *FeatureNet*. Our approach is based on a new correlations-based network construction technique that explicitly discovers hidden communities in high-dimensional raw data. We show the effectiveness of FeatureNet on many diverse small-sample problems as deep learning typically overfits for such problems. We demonstrate that our technique achieves significantly higher accuracy than ten state-of-the-art dimensionality reduction methods (up to 40% improvement) for the small-sample problems.

Since a simple correlations-based network alone cannot capture meaningful features for problems like image classification, we focus on deep learning models like Convolutional Neural Networks (CNN). Indeed, in the era of Internet-of-Things (IoT), computational costs of deep networks have become a critical challenge for deploying such models on resource-constrained edge devices. Towards this, model compression has emerged as an important area of research. However, when a computationally expensive CNN (or even a compressed model) cannot fit within the memory-budget of a single IoT-device, it must be distributed across multiple devices which leads to significant inter-device communication.

To alleviate the above problem, we propose a new model compression framework called the *Network-of-Neural Networks* (NoNN) which first exploits network science to partition a large "teacher" model's knowledge into disjoint groups and then trains individual "student" models for each group. This results in a set of student modules which satisfy the strict resource-constraints of individual IoT-devices. Extensive ex-

periments on five well-known image classification tasks show that NoNN achieves similar accuracy as the teacher model and significantly outperforms the prior art. We also deploy our proposed framework on real hardware such as Raspberry Pi's and Odroids to demonstrate that NoNN results in up to $12\times$ reduction in latency, and up to $14\times$ reduction in energy per device with negligible loss of accuracy.

Finally, since deep networks are essentially a network of (artificial) neurons, network science is a perfect candidate to study their architectural characteristics. Hence, we model deep networks from a network science perspective to identify which architecture-level characteristics enable models with different number of parameters and layers to achieve comparable accuracy. To this end, we propose new metrics called *NN-Mass* and *NN-Density* to study the architecture design space of deep networks. We further theoretically demonstrate that (*i*) For a given depth and width, CNN architectures with higher NN-Mass achieve lower generalization error, and (*ii*) Irrespective of number of parameters and layers (but same width), models with similar NN-Mass yield similar test accuracy. We then present extensive empirical evidence towards the above two theoretical insights by conducting experiments on real image classification tasks such as CIFAR-10 and CIFAR-100. Lastly, we exploit the latter insight to *directly* design efficient architectures which achieve comparable accuracy to large models ($\sim 97\%$ on CIFAR-10 dataset) with up to $3\times$ reduction in total parameters. This ultimately reveals how model sizes can be reduced directly from the architecture perspective.

In summary, in this thesis, we address several problems at the intersection of network science, machine learning, and representation learning. Our research comprehensively demonstrates that network science can not only play a significant role but also lead to excellent results in both machine learning and representation learning.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Networks have become ubiquitous to many applications such as social or biological networks, world wide web, urban systems, migration, among others. A major source of this rapid adoption of network science is the availability of enormous amounts of data for many different applications. However, the very same availability of data has also given rise to the recent *big data revolution* in the fields of machine learning and Artificial Intelligence (AI). Alas, the intersection between these three fields – network science, machine learning, and AI – remains sparse.

Many real-world systems are characterized by complex network dynamics [88]. For instance, information propagation among online users depends on their social network, the spread of epidemics can be modeled via how people interact, *etc*. Similarly, machine learning problems such as predicting short-term timeseries of, say, a river flowrate, can be better represented as a network of rivers, which can in turn improve our prediction accuracy. Hence, many systems often have an *inherent* or a *latent* network component which can be exploited to achieve highly accurate machine learning or AI models. Such problems do *not* consider a network science-based approach. However, we argue that since the behavior of these systems depends on latent interactions within their subsystems, a network science-based approach is imperative for more accurately capturing the system dynamics.

To this end, many machine learning problems traditionally have not even envisioned a network science perspective. For instance, we show that if we consider a network-based approach, we can predict cancer in patients from high-dimensional data with significantly higher accuracy than approaches which do not consider a network. Moreover, although AI is getting increasingly dominated by representation learning techniques such as deep neural networks, network science has surprisingly been absent from the deep learning literature. Therefore, we also show that network

science is effective at solving various deep learning problems. Hence, we exploit the major advantages of network science for addressing important machine learning and representation learning problems throughout this thesis, thereby demonstrating how network science can be used beyond traditional "network" problems such as social or biological networks.

Next, we give a brief overview of network science, machine learning and representation learning, and introduce the problems addressed in this thesis.

1.1 Network Science

Network science studies complex systems in which various subsystems or *actors* interact with each other. Key examples of such networks include social and biological networks, transportation networks, world wide web, internet, *etc*. The core idea of network science is to represent certain complex phenomena as networks and then create models that can be used to understand and predict the desired phenomena. A wide range of real-world phenomena have been modeled using networks, *e.g.*, problems such as information diffusion over social networks, modeling disease outbreaks, *etc*. Mathematically, the actors in the networks are represented as nodes or vertices $\mathcal{V}$, while links or edges $\mathcal{E}$ represent the interactions among the various actors (*e.g.*, people connected in a social network).

Real-world networks are characterized by a number of important characteristics. For instance, majority of users on a typical social network like Twitter are not very popular. However, most of these users follow a small number of very popular users (*e.g.*, sports figures, celebrities, politicians, *etc.*). This results in most users connected to a handful of popular users. Hence, social networks have a highly non-uniform *degree* distribution, where degree of a user refers to number of connections he/she has in the network. In turn, such connectivity patterns result in extremely complex dynamics of, say, information or opinion propagation. To model such dynamics, a number of concepts have been developed in network science such as scale-free networks [7], or community structure [86]. Fig 1.1(a) shows scale-free networks which are characterized by many low-degree nodes, and a few high-degree nodes. Similarly, often networks are organized into groups of tightly connected nodes called community structure (see Fig. 1.1(b)). Finally, other real-world systems can have fractal structure. Fig. 1.1(c) clearly illustrates the fractal structure of river networks.

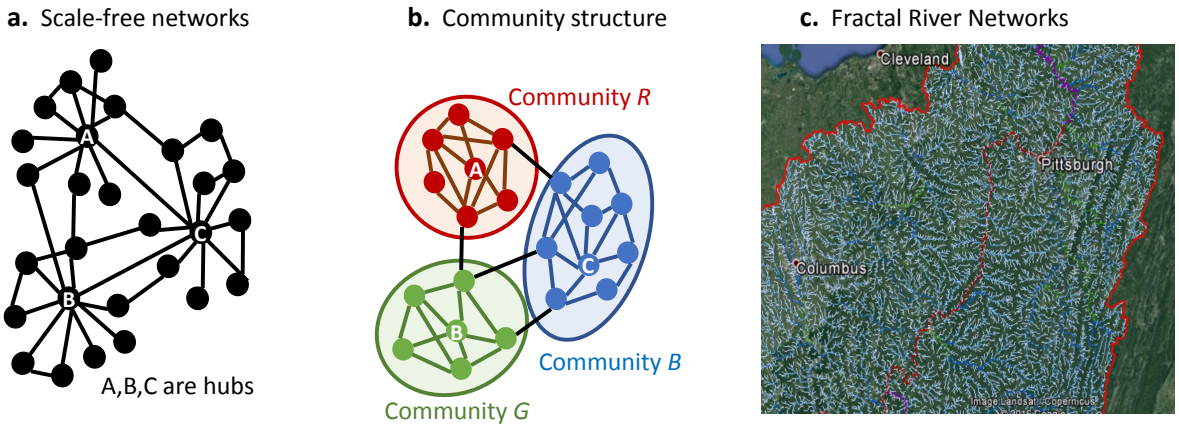


Figure 1.1: Network science (a) A scale-free network is characterized by presence of hubs (*i.e.*, high-degree nodes). (b) Real-world networks often contain community structure, *i.e.*, groups of tightly connected nodes. (c) Many complex systems such as River Networks exhibit fractal network structures.

Throughout this thesis, we argue that network characteristics such as communities can be exploited to solve many learning problems. Next, we discuss the problems in machine learning and representation learning that can particularly benefit from network science.

1.2 Machine Learning and Representation Learning

Two major categories of machine learning are described below:

1. **Supervised learning** problems have a *labeled dataset*. For instance, given a dataset containing both images of certain objects, as well as their explicit labels, the problem of "classifying the object in a given image" is an example of supervised learning.

2. **Unsupervised learning** problems have an *unlabeled dataset*. For instance, the problem of "clustering a set of observations into various groups" (when the label is not available) is an example of unsupervised learning.

There are several other machine learning problems such as reinforcement learning [17, 78] where the idea is to reward a learning agent based on correct actions (*e.g.*, a robot navigating through space towards some target). However, in this thesis, we will mainly focus on supervised learning.

There can be many kinds of supervised machine learning problems and the most notable are: (*i*) Regression problems aim to predict the value of a given target variable, given some fea-

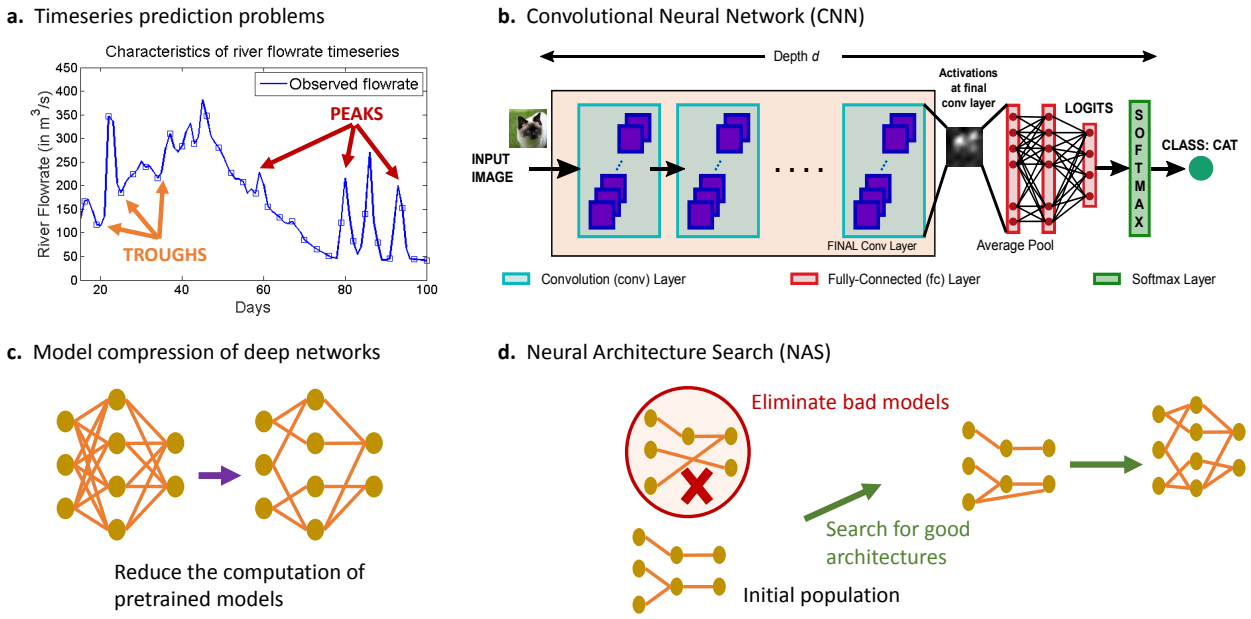


Figure 1.2: Machine Learning and Representation Learning: (a) Example of a traditional machine learning problem: River flow timeseries prediction – timeseries can contain sudden peaks and troughs. (b) Convolutional Neural Networks (CNN) for image classification. (c) Model compression aims to reduce the computational requirements of pretrained deep neural networks. (d) Neural Architecture Search (NAS) aims to automatically learn the architecture of deep learning models.

tures, (*ii*) Classification problems use the input features to classify them into multiple categories, (*iii*) Timeseries prediction problems specifically predict the value of the given variable at the next time step or next few time steps (see Fig. 1.2(a)). Timeseries prediction problems can further utilize additional features (on top of the given variable whose timeseries we are trying to predict).

Conventionally, designing a machine learning system first involves manually designing the features for the problem at hand, a process called *feature selection/engineering*. This is followed by *model selection*, where we select a linear or non-linear model (*e.g.*, linear regression, decision trees, artificial neural networks, *etc.* [17, 78]) which operates on the selected features. The performance of the machine learning system depends on both the quality of the features, and the selected model.

As mentioned above, traditional machine learning relies on designing hand-tailored application-specific features. However, in many other problems with *high-dimensional, small-sample datasets* like cancer prediction, digit-recognition, computational sustainability, *etc*., extracting a set of useful, *low-dimensional* features from the given dataset is a challenging task, and is often critical for

achieving high accuracy. Moreover, for several "big data" problems such as identifying objects in natural images, deep learning techniques such as Convolutional Neural Networks (CNN) have achieved state-of-the-art results (see Fig. 1.2(b)) [41, 48, 58, 108]. Hence, this has resulted in the era of *representation learning* where the objective is to learn features automatically from the data.

At present, representation learning techniques can be largely categorized into three classes: (*i*) Conventional dimensionality reduction for small-sample datasets (since deep learning-based techniques do not work well for small-sample problems) [46, 69], (*ii*) Deep Learning techniques for big data problems in vision, speech and natural language processing [41, 48, 108], and (*iii*) Network Representation Learning, where the idea is to learn representations for nodes in a network directly from its topological structure [35, 96, 116]. Of note, many exciting research problems in the field of deep learning have emerged recently. Notable problems include *Model Compression* for computationally expensive deep networks to deploy such models on hardware-constrained edge devices (see Fig. 1.2(c)) [43, 51, 63], and *Neural Architecture Search* (NAS), where the goal is to design learning models that can automatically create new deep learning models (see Fig. 1.2(d)) [100, 141, 142].

Both model compression and NAS research have seen enormous amount of success recently. For instance, model compression techniques such as pruning reduce the computational costs of the deep network by removing redundant and useless weights without sacrificing accuracy [63, 132]. Other techniques such as quantization try to maintain accuracy while reducing the number of bits used to represent weights/activations of the deep network [51]. Another class of techniques exploit a teacher-student learning paradigm called Knowledge Distillation, where we teach a significantly smaller "student" network to mimic a large "teacher" deep network [4, 43]. Similarly, models designed by recent NAS techniques have outperformed state-of-the-art human-designed deep networks for image classification and speech datasets [66, 97, 100, 141, 142]. NAS techniques are based on reinforcement learning [141, 142], evolutionary algorithms [100, 128], or gradient-based techniques [66]. While reinforcement learning-based techniques are computationally prohibitive (sometimes taking up to thousands of GPU-days and hundreds of GPUs [141, 142]), recent gradient-based techniques have drastically reduced the architecture search cost (*e.g.*, to a few GPU days or to even a few GPU hours [66]).

As we shall see shortly, many machine learning and representation learning problems described above can have an inherent or a latent network component. Hence, in this thesis, we address the following problems from a network science perspective: (*i*) Traditional machine learning problems such as application-specific feature extraction for timeseries prediction, (*ii*) Dimensionality reduction for small-sample problems, (*iii*) Model compression for big data image classification problems, and (*iv*) Neural Architecture Space Exploration via Network Science. But first, we must describe the existing research at the intersection of network science, machine learning, and representation learning, and must answer how our proposed research differs from the prior art.

1.3 Network Science vis-à-vis Machine Learning and Representation Learning

Most of the prior art has traditionally been on how machine learning or representation learning can be used to solve network problems in, say, social or biological networks. For instance, as illustrated in Fig. 1.3, Graphical Lasso [29] aims to infer network structure from data. Other problems include learning and optimization over the social networks [62]. Moreover, the network representation learning techniques [35] learn features for nodes in the network. These features are then used for *network classification problems* such as classifying user interests in a social network. Finally, other representation learning techniques such as deep neural networks do not exploit ideas from network science.

As evident, all of the above directions have one aspect in common: They focus on how machine learning, representation/deep learning, and big data (*e.g.*, social network data) can solve *network problems* (*e.g.*, inferring network structure, automatically extracting the features for nodes in the network, optimization over networks or network classification problems). These prior directions are shown as green arrows in Fig. 1.3. In contrast, in this thesis, we ask the opposite question: *How can network science help with general problems considered in machine or deep learning such as handwritten digit recognition, natural language processing (NLP), cancer prediction, sustainability problems, or image classification?* Can generic machine/deep learning models used for above problems benefit from the fundamental principles of network science? If so, what new models

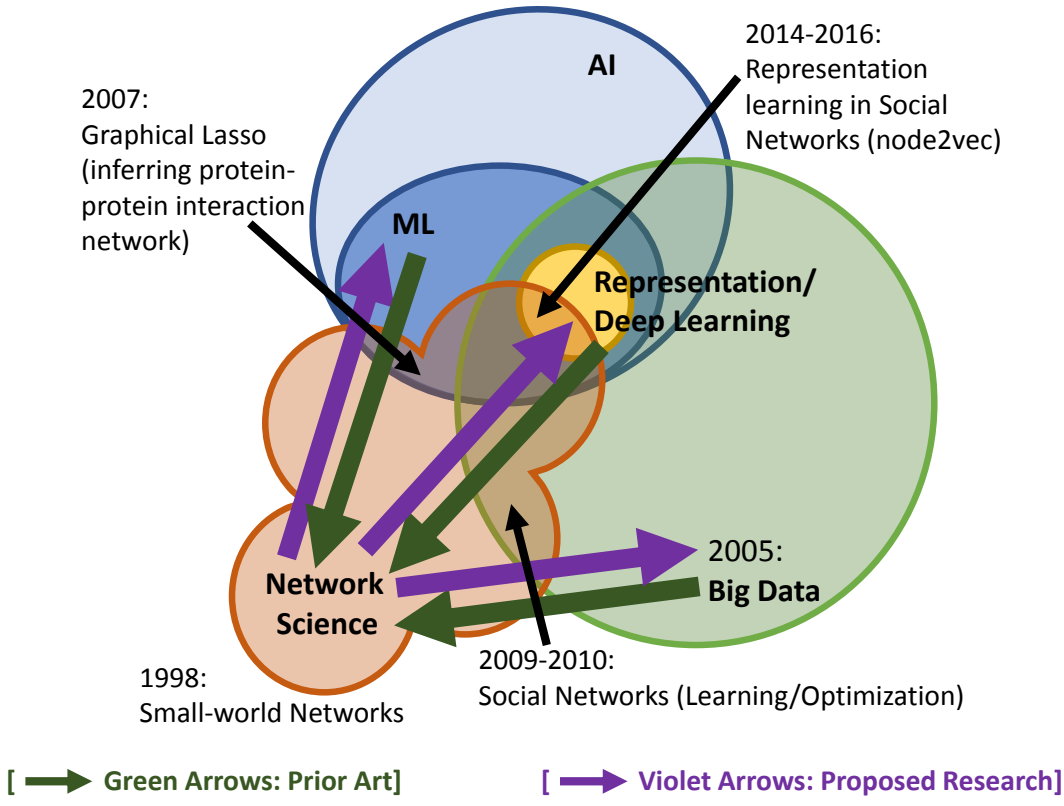


Figure 1.3: Big picture of the proposed research: Prior art (green arrows) focuses on how machine learning, representation learning and Big Data can be used to solve "network" problems in social and biological networks. On the other hand, we target a new network science perspective towards complex machine learning and representation/deep learning (AI) problems (violet arrows).

can be proposed to best integrate information from networks into machine/deep learning models? These questions remain unexplored and, hence, are a major focus of our work.

Given this significant interest in network science, machine learning and deep learning research, in this thesis, we explore how exactly network science concepts can be exploited to create effective solutions for various machine learning and deep learning problems. Specifically, since all systems involve some form of network dynamics, we believe that exploiting network science can truly ameliorate some of the challenges faced by machine/deep learning problems today. For example, in traditional machine learning problems, similarity between samples can be more accurately characterized by network science and, thus, lead to better machine learning problems. On the other hand, deep networks are essentially a network of filters or neurons. Hence, dynamics of information propagation within deep networks can be modeled via network science. Starting from these

overarching ideas, we describe the concrete research challenges addressed in this thesis below.

1.4 Research Challenges

We address four fundamental research challenges encompassing the following **key questions**:

1. **Network Science for Traditional Machine Learning:** Can network science be used to create effective spatiotemporal timeseries prediction models? How can we use network science to extract application-specific features for a given machine learning problem?
2. **Representation Learning on Small-Sample Problems:** How can we exploit network science ideas to automatically learn features for general problems with small datasets (100-1500 samples)? Are there any small-sample problems in the deep learning space?
3. **Network Science-Based Distributed Deep Learning Model Compression:** For big data problems such as image classification, how can we use network science to compress the size and computation of complex deep networks in order to deploy them on a network of resource-constrained IoT-devices?
4. **Network Science for Neural Architecture Space Exploration:** Finally, since deep networks are afterall networks of neurons or filters, what structural property of these networks results in models with high accuracy? Can we use network science to identify architecture-level characteristics that indicate which family of models (despite having different number of layers and parameters) achieve comparable accuracy? Understanding such characteristics can directly allow us to create compressed models with minimal loss of accuracy over the large deep networks.

Answering the above questions led to the new research directions shown as violet arrows in Fig. 1.3. Clearly, these directions run completely opposite to the existing research. We next describe each research challenge in detail.

1.4.1 Network Science for Traditional Machine Learning

We start with the traditional machine learning problems such as spatiotemporal timeseries prediction and application-specific feature extraction. This scenario occurs for problems in which some

kind of traffic or fluid is flowing through space, and we need to predict short-term timeseries at each location. Typically, when a traffic (or fluid) flows through the space, its dynamics can be described as a complex network and, hence, enable a network science-based prediction model. However, most existing machine learning-based timeseries prediction models do not consider the network dynamics. Here, we show that creating new learning models that take the underlying network dynamics into account lead to better prediction compared to the models which do not account for the networks.

To this end, we address two complex spatiotemporal timeseries prediction problems using network science: (*i*) Predicting short-term timeseries at multiple locations when the underlying network structure is known but rapidly changing (perhaps known from prior knowledge or available data), and (*ii*) Predicting short-term timeseries at multiple locations when the underlying network structure is unknown but fixed.

We model the first problem above as a Bayesian network whose structure is known but is dynamically and rapidly changing. Since the network structure is known, we propose a new model to learn parameters on such dynamically changing networks and use them for the underlying prediction tasks. To demonstrate a concrete example of a situation where this kind of learning problems can arise, we provide a case study on a computational sustainability task, *i.e.*, predicting the short-term solar energy and quantifying the solar energy interdependence across large regions.

Next, for many other real-world problems, the predefined network structure governing the dynamics is *not* known for the problem at hand, and only some raw high-dimensional timeseries data is available. Therefore, for such problems, we need to infer the network structure by exploiting the correlations in the given high-dimensional raw data. This leads to the second problem above, where we first infer the network structure from the raw data and then utilize this network structure for *application-specific feature extraction*. Again, we demonstrate this problem by considering a concrete case study on another computational sustainability problem, *i.e.*, use the high-dimensional timeseries river flowrate data to first infer the river network, and then predict short-term river flowrate at several locations in the given river-basin.

Both of the above problems are characterized by complex characteristics such as non-stationary data, changing or unknown underlying network dynamics, *etc*. Of note, the problems considered in this section are application-specific and, hence, different network-based machine learning models

are required to address different problems. However, many real-world problems can also benefit from general principles in network science. We discuss this important case below, where we use network science for representation learning on many general problems with high-dimensional raw data and small number of samples.

1.4.2 Representation Learning on Small-Sample Problems

A major goal in Artificial Intelligence (AI) is not to manually engineer application-specific features but rather to enable machines to learn them *automatically* for *any general problem*. Consequently, the domain of *representation learning* aims to automatically learn useful low-dimensional features from the data. Towards this low-dimensional feature learning, many dimensionality reduction techniques have been proposed in literature. For instance, dimensionality reduction can be performed via linear techniques such as Principal Component Analysis (PCA), or via non-linear neighborhood graph-based techniques such as Isomap, Stochastic Neighbor Embedding (SNE), t-SNE, among many others [1, 42, 69, 118]). Other techniques exploit deep neural networks such as Autoencoders [44].

In the real-world, networks constructed from raw data are often characterized by complex network characteristics (*e.g.*, groups of tightly connected nodes known as the community structure [84]). Existing dimensionality reduction techniques, however, do not take such network characteristics into account. Moreover, it has been theoretically established in [30, 52, 54, 99] that to obtain good classification performance in high-dimensional spaces, the number of samples must also be very large (*e.g.*, $\sim 10^5$ samples as in big data problems addressed by deep learning). Hence, for problems with relatively few samples (*e.g.*, 100-1500 samples), extracting useful low-dimensional features from the high-dimensional data is a very challenging problem [131]. Therefore, we propose FeatureNet [12], a new network science-based dimensionality reduction framework targeting the small-sample, high-dimensional problems. We demonstrate that our network-based approach outperforms traditional dimensionality reduction techniques on several, diverse applications like handwritten digit recognition, biology, physical sciences, NLP, and computational sustainability (sizes mostly between 100 and 1500 samples for each problem).

The small-sample problems have been addressed by conventional dimensionality reduction techniques since the deep learning-based solutions are known to overfit due to lack of sufficient

training data. Then, a natural question to ask is: *Are there any small-sample problems in deep learning space?* The answer to this question is *yes!* As mentioned earlier, with the growth of deep learning, model compression has emerged as a critical bottleneck for adoption of deep learning at the resource-constrained edge devices. However, we still need access to the original (large) training dataset for most model compression techniques. This data may not always be available due to privacy or regulatory concerns (*e.g.*, medical images, speech data, *etc*. can have significant privacy concerns). Hence, this leads to a new kind of small-sample problem in the deep learning space, where we need to compress a deep learning model without access to any real or original data (since collecting a new dataset can be expensive or infeasible). We call this new class of problems as *Data-Independent Model Compression* [16]. Towards this, we propose a new model compression technique which reduces the size and computation of a given deep learning model in absence of any real data, without compromising the accuracy of the compressed model. Such data-independent model compression techniques can play a major role in accelerating the adoption of AI on edge devices.

To summarize, in this section, we address representation learning for small-sample problems for both traditional dimensionality reduction, as well as for scenarios in which deep learning problems suffer from lack of data. We further demonstrate how using network science can improve the quality of features learned from small datasets. Beyond the small data problems, big data problems such as image classification are addressed by deep learning models such as Convolutional Neural Networks (CNN). Therefore, we next address how network science can play an important role in model compression of CNNs. In the following sections, we assume that the big data used for training the large deep network (which we want to compress) is available for model compression.

1.4.3 Distributed Deep Learning Model Compression via Network Science

For big data problems like image classification, deep learning models such as CNNs have achieved state-of-the-art results. Such models, however, not only require enormous training datasets, but also often utilize millions of parameters to work well. Due to this immense computational complexity, there is a fundamental need for highly efficient deep learning model architectures which can enable *faster and computationally inexpensive training, as well as inference* (*i.e.*, determining the class of an object in a new image after the training phase). This is particularly important

for *mobile applications* where the computational resources and memory are limited for image, speech, and natural language applications. As mentioned earlier, many techniques such as pruning [39, 132], quantization [51, 60], and Knowledge Distillation (KD) [4, 43, 135] have been proposed for deep learning model compression.

The prior approaches above *cannot* be used for *extremely memory-constrained* IoT scenarios (*e.g.*, microcontrollers with 500KB memory). Still, many smart home/cities applications have several such well-connected, but resource-constrained sensors. To achieve higher accuracy, pruning- or KD-based compressed models often grow in size due to which such models cannot fit on an individual IoT-device and, hence, must be distributed across multiple devices; this distribution of computation on multiple devices generates significant overhead in communication.

To alleviate this, we propose Network-of-Neural Networks (NoNN) [15], a new paradigm for compressing a given pretrained deep network into multiple disjoint modules which require minimal inter-device communication. Each module of NoNN satisfies strict memory- and computation-constraints (as measured by number of parameters and FLOPS per module). For instance, if a given module has 500K parameters, it can fit within a memory-budget of 500KB when quantized to 8-bits. Since our proposed NoNN is based on KD-like teacher-student learning, the pretrained deep network which we want to compress becomes our teacher, and each of the compressed modules serves as a disjoint student. Each of these student modules can then be deployed on a separate resource-constrained device, such that there is a minimal communication among devices. This results in a new kind of model compression, which we call as a *communication-aware model compression* problem. Of note, to the best of our knowledge, we are the first to address this problem.

As NoNN consists of a *network* of neural networks, this problem can naturally be viewed from a network science angle. Recall that KD aims to transfer knowledge from a teacher model to a student model. Since our objective is to distribute the knowledge from a single teacher network to multiple student models, we exploit network science for this knowledge partitioning problem. Specifically, we mine latent patterns of activation at teacher's final convolution layer to create a *filter activation network*. Then, we use network science techniques such as community detection [84] to partition this network into disjoint subsets, each of which is used to distill part of teacher's knowledge to a distinct student module. Finally, all students join together in the network

to make the final prediction. We demonstrate that our network science-based framework achieves higher accuracy than existing baselines for a given parameter/FLOP budget for many well-known image classification benchmarks. We further demonstrate the savings in latency and energy by deploying our models on real edge devices.

To summarize, we show how network science can play a fundamental role in the modeling of knowledge partitioning problem which results in effective model compression. We also deploy our models on real edge devices to demonstrate that there is good agreement between theory (*i.e.*, theoretical reduction in FLOPS) vs. practice (*i.e.*, energy reduction achieved on real devices). Next, since deep networks are ultimately a network of neurons and channels, we go even deeper into deep learning and see how network science can play an effective role in architecture design space exploration.

1.4.4 Network Science for Neural Architecture Space Exploration

So far, efficient deep networks have been designed using model compression techniques such as (*i*) Pruning [63, 132], (*ii*) Quantization [51, 60], (*iii*) Knowledge Distillation (KD) [15, 43, 135], or via (*iv*) Manually designed efficient networks and convolutions such as depth-wise separable convolutions [47, 103, 137], and (*v*) Efficient models resulting from automatic Neural Architecture Search (NAS) methods [66, 97, 100, 128, 141, 142]. All of the above directions do not answer a fundamental research question: *Are there any characteristics of a CNN architecture that can indicate which family of models (with different number of parameters and layers) can achieve similar accuracy?* Indeed, such characteristics can enable a new form of model compression in which we can directly design a novel architecture to reduce the number of parameters and layers without losing significant accuracy.

To address the above question, we first model deep CNNs as a network of channels and then propose two new network science-based metrics: (*i*) *NN-Density* quantifies how densely the channels within a deep network are connected, and (*ii*) *NN-Mass* captures the representational capacity of a given model. We merge, for the first time, the well-known Probably-Approximately-Correct (PAC)-Bayes theory [74, 75] for model generalization with theory of small-world networks [79, 87] in network science. We then prove that (*i*) The generalization error of models decreases as the NN-Mass increases, and (*ii*) Irrespective of number of parameters and layers, models with similar

NN-Mass achieve similar test accuracy. We then perform a large number of experiments on real image classification tasks such as CIFAR-10 and CIFAR-100 to empirically demonstrate the above relationship between NN-Mass and generalization of CNN architectures. Specifically, we found that models with similar NN-Mass indeed achieve comparable accuracy even if their number of parameters and depth are vastly different.

Finally, using the above theoretical/empirical results, NN-Mass can be exploited for model compression. Specifically, we directly design *new* models with significantly less parameters and layers but with mass comparable to (or higher than) that of a large CNN. We show that our newly designed models achieve very high accuracy (*e.g.*, up to 97% on CIFAR-10 dataset), while using $3\times$ fewer parameters than the large models. Hence, network science helps uncover latent architecture-level characteristics that enable us to build efficient deep networks.

1.5 Thesis Contributions

Overall, we make the following **core contributions** in this thesis:

1. **Network Science for Traditional Machine Learning.** We propose a new probabilistic learning framework called *Network of Dynamic Bayesian Networks* (NDBN) for problems with known but dynamically changing networks. This framework is proposed for spatiotemporal timeseries prediction. As a case study, we use our proposed model to handle a computational sustainability problem of solar energy prediction over large areas and also conduct knowledge discovery in this domain. To address problems with high-dimensional data but now unknown network structure, we propose a new correlations-based network inference method and an application-specific feature extraction technique called *K-Hop Learning*. As another case study, we show the effectiveness of our network-based feature extraction approach on a river flow prediction task.

2. **Representation Learning on Small-Sample Problems.** We further exploit complex network characteristics to address automatic feature learning, *i.e.* representation learning, and dimensionality reduction on many diverse small-sample problems (since deep learning typically overfits for this problem space). Specifically, we propose a new community-based dimensionality reduction framework called *FeatureNet* and show that our method outper-

forms ten state-of-the-art methods for the small-sample problems. Moreover, we show that lack of real data can pose significant challenges for deep network model compression. Towards this, we propose a new technique called *Dream Distillation* to perform deep learning model compression in limited-data setting.

3. **Model compression for Memory- and Communication-Aware Distributed Deep Learning Inference.** Next, since a simple correlations network-based representation learning approach alone cannot suffice for problems like image classification, we focus on Convolutional Neural Networks (CNN). When a computationally-expensive CNN (or even a compressed model) does not fit within the memory-budget of a single resource-constrained IoT-device, it must be distributed across multiple such devices, thus, generating significant communication among devices. Therefore, we propose a new model compression framework called the *Network-of-Neural Networks* (NoNN) which first exploits network science to partition teacher's knowledge into disjoint groups and then trains individual students for each group. This leads to a set of student modules which satisfy the strict memory-constraints of individual devices. We then deploy our proposed framework on real hardware such as Raspberry Pi's to demonstrate that NoNN results in up to $12\times$ reduction in latency, and up to $14\times$ reduction in energy per device.

4. **Network Science-Based Neural Architecture Space Exploration.** Finally, we model deep networks from a network science perspective to identify architecture-level characteristics that enable models with different number of parameters and layers to achieve comparable accuracy. To this end, we propose new metrics called NN-Mass and NN-Density to study the architecture design space of deep networks. We merge, for the first time, the PAC-Bayes theory with the theory of small-world networks to demonstrate provable relationship between NN-Mass and generalization of CNN architectures. Specifically, we show that (*i*) Higher the NN-Mass, the lower the generalization error, and (*ii*) Models with similar NN-Mass achieve similar generalization error. We present extensive empirical evidence towards the above theoretical insights. We further demonstrate that NN-Mass can be used to directly design efficient architectures which achieve comparable accuracy to large models ($\sim 97\%$ on CIFAR-10) with up to $3\times$ reduction in total parameters. This ultimately reveals a new

form of model compression which reduces the model size directly from the architecture perspective.

Despite the diversity of problems considered, they all revolve around the central theme of how learning can benefit from network science. Next, we describe the organization of this thesis.

1.6 Organization of Thesis

This thesis is organized in seven chapters. Chapter 2 briefly describes background on machine learning, representation learning, and network science. Next, in Chapter 3, we demonstrate how network science can be used for traditional machine learning problems such as timeseries prediction. Towards this, we address a complex problem of parameter learning over a network whose structure is known but is rapidly changing. For problems with high-dimensional data but no known network structure, we propose new network inference models and also formulate a network-based, application-specific feature extraction method to improve the performance of machine learning models.

Having demonstrated the importance of our proposed network science-based feature extraction for specific problems, we next move to automatic feature learning for *any* general problem (*i.e.*, representation learning) in Chapter 4. Here, we propose a new network communities-based dimensionality reduction framework and show its effectiveness for several small-sample datasets for which deep learning-based models typically overfit. In terms of small-sample problems for deep learning, we further demonstrate how to perform deep learning model compression in absence of real data.

Next, we address significantly more complex representation learning problems such as image classification, where correlations-based approaches are not sufficient. For such problems, computationally expensive deep learning models such as CNNs are used. In Chapter 5, we propose NoNN, a network science-based model compression framework for distributed deep learning inference on a network of IoT-devices. Chapter 5 presents the algorithmic details of NoNN as well as its complete validation on real resource-constrained hardware.

Following the distributed model compression, we model deep neural networks from a network science perspective to identify what architectural characteristics lead to highly accurate models.

Hence, in Chapter 6, we first propose new architectural characteristics and then theoretically prove their relationship with generalization of deep networks. We also exploit the newly proposes metrics to explore the architecture space of CNNs. We ultimately use the theoretical and empirical insights to directly design compressed models. Finally, we conclude the thesis in Chapter 7 with remarks on future work.

Chapter 2

Background

This chapter reviews the background on various topics in machine learning, representation learning and network science. To this end, we first discuss topics from machine learning such as regression, timeseries prediction, and classification problems. This is followed by background in representation learning techniques such as dimensionality reduction and deep convolutional neural networks (CNNs). Note that, the machine learning part focuses on traditional application-specific feature engineering, while the representation learning aims to automatically learn features from the available data. Following the discussion on CNNs, we then describe the background on model compression of deep networks and Neural Architecture Search (NAS). Finally, we will explain relevant topics from network science.

2.1 Machine Learning

As discussed in Chapter 1, there are mainly two categories of traditional machine learning (*i*) Supervised Learning, and (*ii*) Unsupervised Learning. Supervised learning consists of problems where two sets of variables are given: (a) Target (or response) variables are the ones that we are trying to predict for the problem-at-hand, and (b) Explanatory variables (or *features*) are the ones that influence the target. An example of supervised learning problem is to predict the stock price (*i.e.*, our target variable) using a set of variables such as quarterly revenues, *etc.*, as our features. The idea here is to learn the function between the features and the target, and then use this learned function to predict the target variable for new test data points. In contrast, unsupervised learning does not require target labels to be the part of the given dataset. A well-known example of unsupervised learning is the clustering problem, where we are given only the features for several

samples, and the task is to find clusters of samples with similar features. This can be very useful for understanding or analyzing the given dataset.

In this thesis, we will mostly focus on supervised learning problems. Specifically, supervised learning can be seen as a function approximation problem: Let y be the target variable, and $X \in \mathbb{R}^{n \times m}$ denote the given data matrix with n samples and m features. Then, the problem is to find the function f such that $\hat{y} = f(X)$ predicts the value of the target y. The precise form of the function f depends on our *hypothesis*. For instance, y can be a linear function of X, or it can be a non-linear function of X. For linear dependence between y and X, our function to be learned f must belong to the linear hypothesis; similar discussion holds for non-linear hypotheses.

Supervised learning can either be a regression problem or a classification problem. Regression also includes timeseries prediction problems in which at least one feature is the previous value of the target variable itself. Fig. 2.1 illustrates the regression, timeseries, and classification problems. With the help of well-known machine learning techniques in traditional supervised learning, we discuss these important categories below.

2.1.1 Regression and Timeseries Prediction Problems

Regression problems have a continuous target variable, whereas its features can be either continuous or categorical. That is, the function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ must map the features to continuous target values. As mentioned in Chapter 1, once the features are manually engineered for the problem-at-hand (*i.e.*, feature selection or engineering), the next step in any traditional supervised learning is to select an appropriate model (*i.e.*, model selection). Towards this, many linear and non-linear hypotheses for the function f can be used (again, this depends on the specific problem). Most common regression models include linear regression (ordinary least squares), lasso regression (which is linear regression with a regularizer term), k-nearest neighbor regression, support vector regression, and random forests [17, 78]. To understand how a typical machine learning problem is formulated, we next discuss the linear regression method. For other machine learning regression methods, please refer to [17, 78].

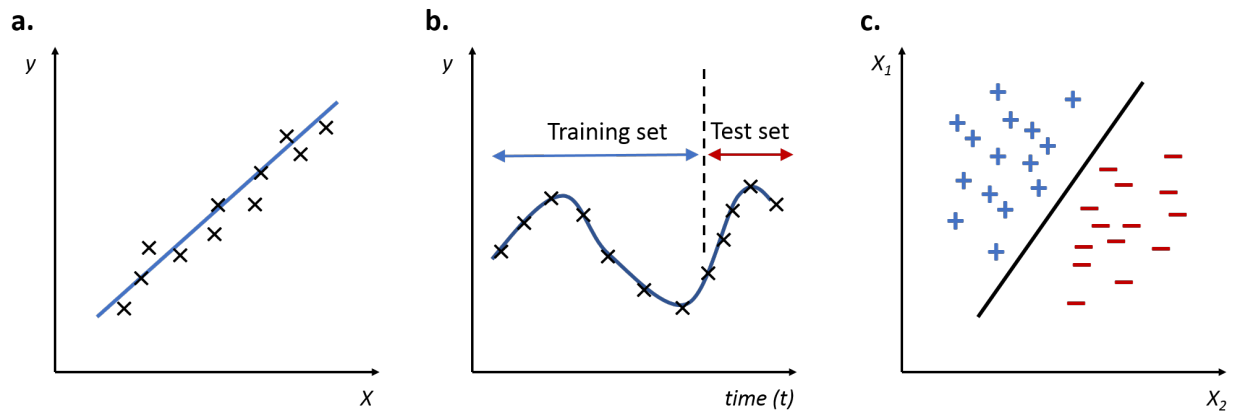


Figure 2.1: Supervised Learning problems: (a) Regression, (b) Timeseries Prediction, and (c) Classification. All supervised learning problems have a labeled dataset.

Linear Regression

To illustrate, a linear regression problem with one dimensional feature space is shown in Fig. 2.1(a). However, in general, we are given a dataset $\{X, y\}$ with n *training* samples, and m features, and a single target variable y for each of the training sample. Additionally, features for n_{test} samples (*i.e.*, X_{test}) are given (but not their true label). Then, in any supervised learning problem, the objective is to learn the function f on the training set, and use this function to predict $\hat{y}_{test} = f(X_{test})$, thereby obtaining predictions for unlabeled test samples.

For linear regression, the function f takes the following form: For a sample $x^{(i)} \in X$, $f(x^{(i)}) = w_0 + w_1 \cdot x_1^{(i)} + w_2 \cdot x_2^{(i)} + \ldots, + w_n \cdot x_n^{(i)}$. In matrix form, this can be expressed as $f(X) = Xw$. Since f must fit the training set well, the linear regression problem can be expressed as follows:

$$\min_{w} \quad ||y - Xw||_2^2 \tag{2.1}$$

Intuitively, problem (2.1) minimizes the error between y and $\hat{y} = Xw$. The loss function above can be minimized via stochastic gradient descent. Hence, the above procedure can be summarized as follows (*i*) manually engineer the features, (*ii*) come up with a hypothesis (*e.g.*, linear or non-linear models) for how the target variable depends on the features, and (*iii*) propose an appropriate loss function to minimize the error between the observed training data and the model's prediction. This general flow holds for many traditional machine learning problems.

Bias-Variance Tradeoff for Supervised Learning Problems

The complexity of a model's hypothesis should be decided based on the *bias-variance tradeoff*:

- Bias refers to the amount of error that occurs due to incorrect assumptions in our hypothesis. For instance, if the true underlying function (between the target and the features) to be learned is quadratic but our model's hypothesis is linear, then clearly, our hypothesis will miss the complete relationship between features and the target. This is called *underfitting*.
- Variance quantifies the error that occurs when our model fits even the smallest fluctuations in the training data; that is, it learns the random noise in the training data instead of the true function. For instance, if the true function is quadratic, a highly non-linear polynomial-based regression model can fit each and every training sample (even the random fluctuations); this can lead to lower generalization accuracy and is called *overfitting*.

Hence, a tradeoff between Bias and Variance must be achieved for effective learning. For a more concrete discussion on Bias-Variance, please refer to [17, 78]. Next, we consider the case of timeseries prediction problems in which there are no additional features other than the previous observed values of the target variable itself.

Timeseries Prediction

Training and prediction for timeseries problems is shown in Fig. 2.1(b). Specifically, at least one of the features for these problems is the timeseries of the target variable. When the timeseries of the target variable is the only feature available, the model is called *Autoregression* (AR). That is, the hypothesis for AR models is given as:

$$y_t = \sum_{i=1}^{p} w_i y_{t-i} + \epsilon_t \tag{2.2}$$

where, ϵ_t is Gaussian white noise ($(\epsilon_t \sim N_{iid}(0, \sigma_2)$), and p is the order of autoregressive model. Typically, the parameters of the AR are found by ordinary least squares method above.

Finally, in many cases, some exogenous features are also available in addition to the timeseries. In such cases, AR model can be extended to Autoregression-with-exogenous inputs (ARX) model [24]. Again, the parameters for ARX model are learned similar to those in linear regres-

sion. This completes our discussion of regression problems. Next, we describe the classification problems in detail.

2.1.2 Classification Problems

Classification problems have a discrete target variable and, similar to regression problems, can have continuous or categorical features. For simplicity, we explain below the most commonly used model, namely, the logistic regression for binary classification. Logistic regression is useful for classifying *linearly-separable* classes since the decision boundary generated by logistic regression is linear.

Note that, since binary classification problems have $y \in \{0, 1\}$, we can no longer use the same hypothesis that we used for linear regression (*i.e.*, $\hat{y} = Xw$). Therefore, in logistic regression, the hypothesis is given by the sigmoid function, $\hat{y} = \sigma_w(x) = \frac{1}{1+e^{-w^T x^{(i)}}}$, where x is a feature vector for the sample i, and w refers to the model weights.

From a probabilistic point-of-view, since $\hat{y} \in [0, 1]$, the following equations hold:

$$\begin{aligned} P(y = 1|x; w) &= \sigma_w(x) \\ P(y = 0|x; w) &= 1 - \sigma_w(x) \end{aligned} \tag{2.3}$$

The above two probabilities can be directly combined as:

$$P(y|x; w) = (\sigma_w(x))^y (1 - \sigma_w(x))^{1-y} \tag{2.4}$$

To compute the parameters that best explain the observed data ($X, y = \{x^{(i)}, y^{(i)}|i = 1, 2, \ldots, n\}$), we must maximize the *log-likelihood* (*i.e.*, find the parameters that make the observed data most likely):

$$\begin{aligned} l(w) &= \log(P(y|X; w)) \\ &= \prod_{i=1}^{n} \log(P(y^{(i)}|x^{(i)}; w)) \\ &= \prod_{i=1}^{n} \log((\sigma_w(x^{(i)}))^{y^{(i)}} (1 - \sigma_w(x^{(i)}))^{1-y^{(i)}}) \\ &= \sum_{i=1}^{n} y^{(i)} \log(\sigma_w(x^{(i)})) + (1 - y^{(i)}) \log(1 - \sigma_w(x^{(i)})) \end{aligned} \tag{2.5}$$

Maximizing the log-likelihood given in Eq. (2.5) via Stochastic Gradient Ascent generates the linear decision boundary for classification via logistic regression. We use logistic regression for binary or *multiclass classification* throughout this thesis. Multiclass classification can be conducted by using a *one-vs.-rest* logistic regression. For other classification methods (*e.g.*, non-linear classifiers such as Support Vector Machines with kernel trick), please refer to [17, 78].

So far, we have discussed the traditional machine learning, *i.e.*, Given the data for problem-at-hand, we first manually engineer the features, then select a hypothesis (*e.g.*, what model will best describe the relationship between the features and the target variable) and, finally, based on the above, we select an appropriate loss function to optimize. However, one of the main goals of AI is to allow machines to learn features automatically for the given problem. This leads us to the field of representation learning, largely dominated by deep neural networks, as described next.

2.2 Representation Learning

Representation learning is a sub-field in AI that refers to a broad set of models which engineer features automatically from the given data irrespective of the problem-at-hand. The hand-tailored feature engineering in traditional machine learning can indeed be time or labor intensive. Hence, via representation learning, we can learn the useful features automatically from the data without the manual, application-specific hand-tailoring. Towards this, we discuss two kinds of representation learning techniques: (*i*) Dimensionality Reduction which can learn features from small-sample and high-dimensional datasets, and (*ii*) Deep Neural Networks which are some of the most important representation learning methods and have achieved state-of-the-art results for many vision, speech and natural language processing tasks.

2.2.1 Dimensionality Reduction

In this section, we address the traditional dimensionality reduction problems that learn features from high-dimensional data and are particularly important for small-sample datasets in which deep learning-based methods lead to overfitting. We will discuss the two categories of dimensionality reduction: (*i*) Linear techniques such as Principal Component Analysis (PCA), and (*ii*) Non-linear nearest-neighbor graph-based techniques such as Isomap (see Fig. 2.2 for both methods).

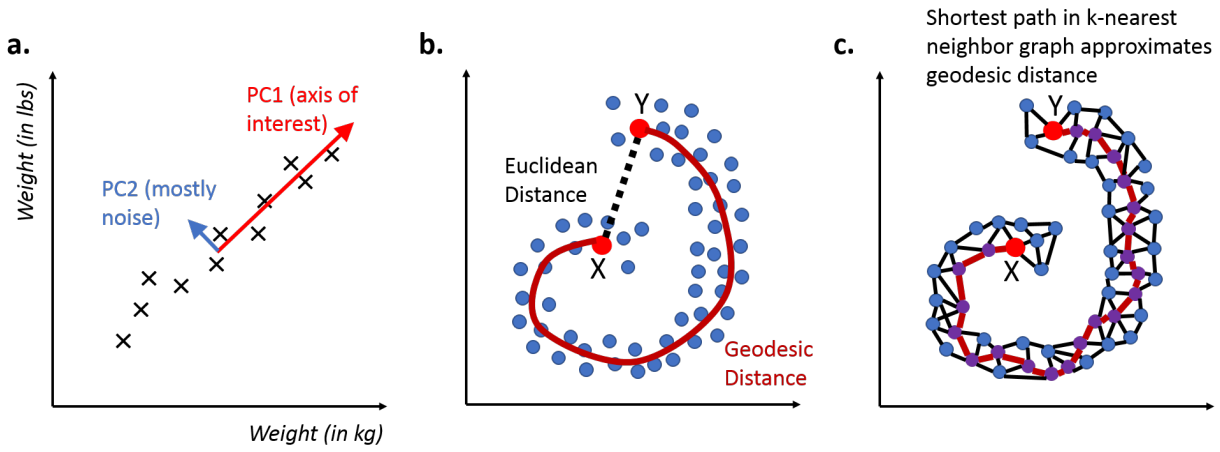


Figure 2.2: Dimensionality Reduction problems: (a) Linear Techniques – Principal Component Analysis (PCA): Find orthogonal principal components of the data and project the current dataset onto the subset of principal components (PCs). PCA reveals axes that do not contribute significantly to the overall variance of the data. (b) Non-linear techniques – Isomap: Euclidean distance in high-dimensional space may not accurately capture the true geodesic distance on the underlying manifold. (c) Isomap exploits the shortest distance on the k-nearest-neighbor graph as an approximation for true geodesic-distance to create low-dimensional representations. Illustrations (b,c) are adapted from [118].

Linear methods

Linear dimensionality reduction methods mainly consist of techniques such as Principal Component Analysis (PCA), Canonical Correlations Analysis (CCA), Linear Discriminant Analysis (LDA), Multi-Dimensional Scaling (MDS), *etc.* [23]. We next explain the linear models with the help of PCA.

In many problems, the features in the given dataset can be heavily correlated. Such correlated features are not useful for learning and exacerbate the learning problem due to the curse of dimensionality (*i.e.*, the more the dimensions, the bigger is the feature-space and, hence, we require exponentially higher number of samples to obtain higher classification accuracy) [30, 54, 99, 106]. Since these features do not contribute to the machine learning model, we must remove them before training a classification or a regression model. To this end, as illustrated in Fig. 2.2(a), PCA automatically finds a set of orthogonal axes along which the given data can be projected to obtain lower-dimensional embedding.

To understand the importance of PCA, consider the following example: Fig. 2.2(a) shows an example dataset where, say, the weight for a sample of population is measured using two separate

devices: Device 1 measures the weight in kilograms (kg), whereas Device 2 measures the weight in pounds (lbs). Since the devices can suffer from noise, as seen in Fig. 2.2(a), there can be small fluctuations between the measurement of two devices. As evident, the first principal component (PC) determined by PCA accounts for most of the variation in the dataset, whereas the second PC mostly accounts for noise (because the weights for each person must be a single true value on the line $y = x$). Hence, PCA can be used to remove the additional noise from the dataset which can otherwise confuse the learning algorithm. For detailed mathematical formulation of PCA, please refer to [46].

Non-linear graph-based methods

The underlying data may *not* always lie on a linear manifold (*e.g.*, see Fig. 2.2(b)). Therefore, for such problems, we need non-linear dimensionality reduction techniques that can take into account the (unknown) shape of the manifold while learning the low-dimensional representation. To account for this non-linear manifold, many techniques such as Stochastic Neighbor Embedding (SNE) [42], t-Distributed SNE (tSNE) [69], Isomap [118], *etc.*, follow a nearest neighbor graph-based approach. Below we explain this process with the help of Isomap.

The crux of dimensionality reduction is that if two points are close to each other in high-dimensional space, they must also be close together in the low-dimensional space. However, as shown in Fig. 2.2(b), euclidean distances in high-dimensional spaces are often not able to capture the true similarity between samples due to the non-linear manifold. Therefore, in Isomap, the true geodesic distance is approximated by the shortest path on a k-nearest neighbor graph. Clearly, as evident from Fig. 2.2(c), the network distance much better represents the true geodesic distance than the euclidean distance. Hence, the network distances are used to derive the final low-dimensional representations.

This ends our discussion of traditional dimensionality reduction which is particularly important for small-sample, high-dimensional datasets. For a thorough review of dimensionality reduction techniques, please refer to [70]. Next, we describe deep neural network-based representation learning including background on some of the recent problems in deep learning.

2.2.2 Deep Learning

To explain the key concepts in deep learning, we first discuss the building blocks of neural networks, namely, the multilayer perceptrons (MLP) and how to optimize them. Then, we move onto more complex Convolutional Neural Networks (CNNs) which have achieved state-of-the-art results in computer vision problems such as image classification, object detection, *etc*. Since the main focus of this thesis is also on deep learning for image classification problems, we later describe some of the latest challenges in deep learning, *e.g.*, model compression and neural architecture search (NAS) for vision tasks. Model compression and NAS for other applications such as speech recognition are out of scope of our work.

Multilayer Perceptron (MLP)

MLPs are the most fundamental deep learning models. As shown in Fig. 2.3(a), MLP consists of multiple "neurons" hierarchically organized into multiple layers. The idea behind MLPs is to somewhat mimic how neurons in the brain are organized layer-by-layer which learn hierarchical features.

The output of each hidden layer in MLP is a combination of two operations:

1. A linear model that operates on the output of previous layer. For instance, the output of the first hidden layer in Fig. 2.3(a) is given by:

$$h_{linear} = W_h^T X + b_h$$

   where, W_h is the weight matrix of hidden layer h, and b_h is the bias term.

2. A non-linear *activation* function σ that is applied on h_{linear}. Therefore, the output of the hidden layer is obtained as:

$$h = \sigma(W_h^T X + b_h) \tag{2.6}$$

   Popular activation functions include sigmoid function $(1/(1 + e^{-x}))$ and Rectified Linear Unit (ReLU): $\text{ReLU}(x) = \max(0, x)$.

The output of the next hidden layer can be computed in a similar fashion. We repeat this process until we reach the output of the network. Finally, a softmax function can be used if we are perform-

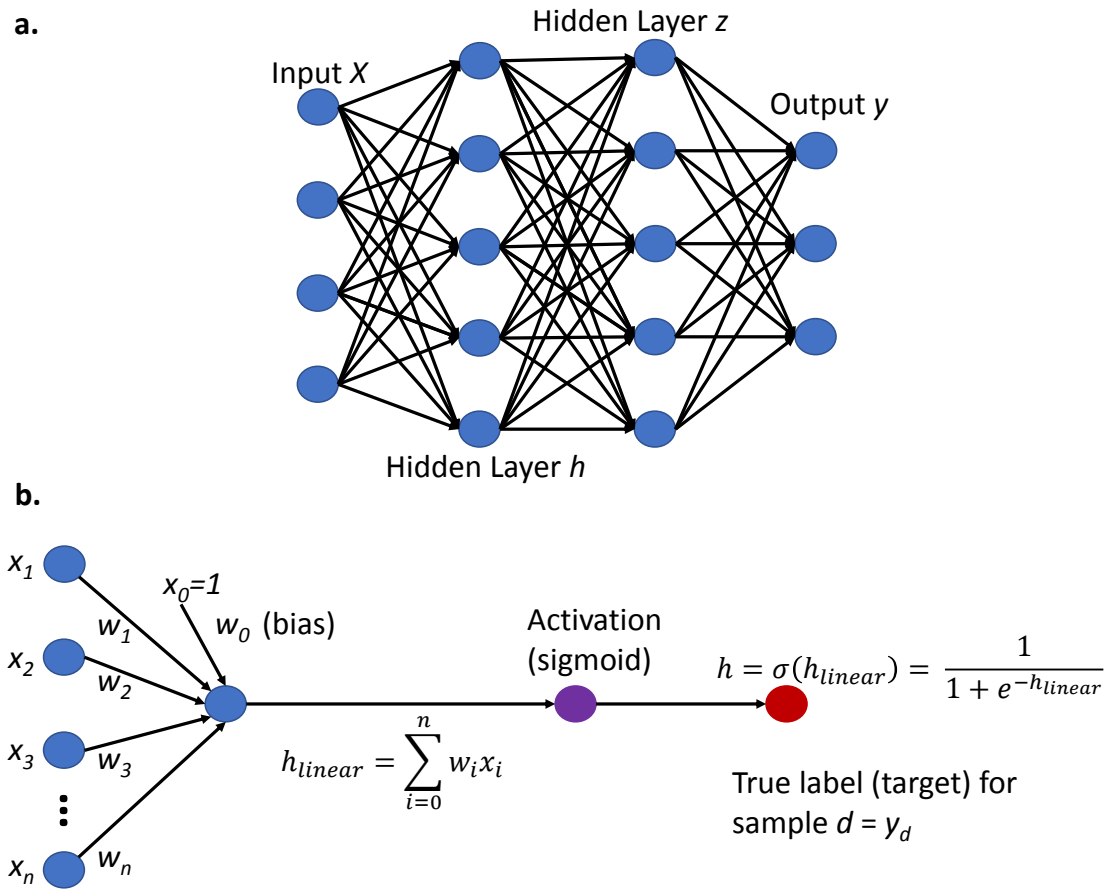


Figure 2.3: Deep Learning techniques. (a) Multilayer Perceptron (MLP): A neural network with two hidden layers At each hidden layer, the previous layer's output is weighted and is passed through a non-linear activation function. (b) Single neuron with sigmoid function as non-linear activation.

ing multiclass classification to convert unnormalized output of MLP (called *logits*) into prediction probabilities: $\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$.

We have described above the key elements in MLP and the process to go from initial input to output; this process is called *forward pass*. However, to train the neural network, we need to propagate the gradients through the network for stochastic gradient descent. This process, called *backward pass*, is described next.

Training Deep Networks via Backpropagation Algorithm (Backprop)

While training deep networks, Backprop algorthm exploits the chain rule for deriving efficient gradient updates for neural network parameters. Note that, the basic idea behind Backprop (*i.e.*,

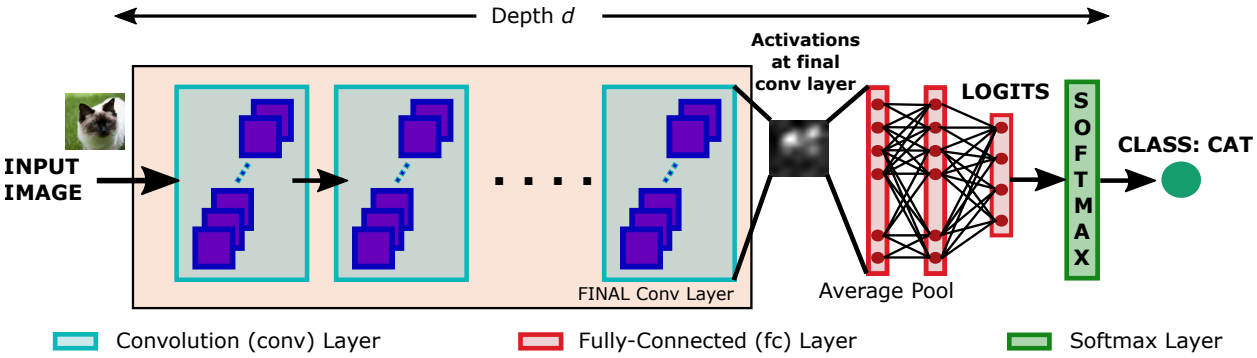


Figure 2.4: Convolutional Neural Networks (CNN): After an image passes through several convolutional layers, an activation map is obtained at the final layer. This activation map is averaged (average-pool layer) and passed through one or more fully-connected layers to generate logits (which are essentially unnormalized probabilities). The final prediction probability is obtained by passing the logits through the softmax layer.

the chain rule) holds not only for MLPs but also holds for CNNs and other kinds of deep networks. We explain Backprop with the help of an example given below.

Consider a single unit with sigmoid activation as shown in Fig. 2.3(b). Now let y_d, h_d, and x_i^d be the target output, output of the neural network, and the i-th feature for training example $d \in D$ (D is the entire training set), respectively. Also, suppose that w_i's are the model weights. Then, the error E is given as follows:

$$E = \frac{1}{2}\sum_{d \in D}(y_d - h_d)^2 \tag{2.7}$$

To compute the gradients to minimize E, we must derive the partial derivatives of E w.r.t. the parameters of the neural network:

$$\begin{aligned}\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i}\frac{1}{2}\sum_{d \in D}(y_d - h_d)^2 \\ &= -\sum_{d \in D}(y_d - h_d)\frac{\partial h_d}{\partial w_i} \\ &= -\sum_{d \in D}(y_d - h_d)\frac{\partial h_d}{\partial h_{linear,d}}\frac{\partial h_{linear,d}}{\partial w_i}\end{aligned} \tag{2.8}$$

However, since h_d is a sigmoid function of $h_{linear,d}$, $\frac{\partial \sigma(x)}{\partial x} = \sigma(x)(1 - \sigma(x))$. Therefore,

$$\frac{\partial E}{\partial w_i} = -\sum_{d \in D}(y_d - h_d) \cdot h_d \cdot (1 - h_d) \cdot x_i^d \tag{2.9}$$

Note how the chain rule is used in Eq. (2.8) to derive gradients efficiently. This is especially critical for deeper networks. This process is called Backprop since the error is propagated back-

wards through the network. Backprop is an important tool for optimizing the modern deep networks, and is used in state-of-the-art deep learning models.

This completes the fundamental neural network concepts. Next, we briefly discuss Convolutional Neural Networks which have established state-of-the-art results in complex computer vision problems.

Convolutional Neural Networks (CNN)

Many problems such as image classification, the data has a certain structure which can be exploited for better representation learning. This is why, CNNs have emerged for computer vision problem which aim to exploit the 3-D structure of images (*e.g.*, height, width, and RGB channels). Consequently, layers in a CNN are also arranged in a 3-D pattern shown in Fig. 2.4, where the neurons are organized into *filters*. Each filter has same number of channels as the layer's input and focuses on a certain region of the image called the *receptive field*. CNNs are very parameter-efficient since the receptive fields of filters are typically small (say, 3×3 or 5×5). To obtain the output of a given filter i, each channel of this filter traverses the entire corresponding input channel, and results from all input channels are added together to get the output channel i (corresponding to filter i).

Hence, each layer in a CNN consists of several 3-D filters and the number of such filters determines the number of output channels of the layer. The output of each filter is also called a *feature activation map*. Typically, the number of channels per layer are increased gradually, and to reduce the computation, the sizes of feature activation maps are reduced (using bigger strides in convolutions or things like max-pooling, *i.e.*, taking the maximum in a certain neighborhood). Finally, as shown in Fig. 2.4, a CNN consists of many such layers, followed by an average-pool layer, and a fully-connected layer. The main idea behind multiple layers of CNN is that initial layers capture some basic building block features (*e.g.*, edges, angles, *etc.*), and later layers detect more fine-grained features (*e.g.*, shapes, patterns, *etc.*). An average-pool layer simply takes the average of the feature maps from the final convolutional layer, and the fully-connected layer is simply a linear function of the average-pooled outputs. The results of fully-connected layer are called *logits* which are passed through a softmax layer to obtain the prediction probabilities of the network.

The above is a brief description of the CNNs. More details can be found in [34]. Next, we

move on to recent deep learning problems such as Model Compression and Neural Architecture Search (NAS).

2.3 Model Compression in Deep Neural Networks

Power- and memory-constrained hardware of IoT-devices continues to be the biggest challenge for rapid adoption of AI at the edge. Towards this, the field of model compression has received a lot of attention to make deep learning models more suitable for edge devices [43, 60, 132]. Besides model compression, another venue of active research is hardware accelerators and codesign of models and hardware architectures for efficient deep learning inference. However, we mainly focus on model compression in this thesis.

Model compression refers to a class of techniques that reduce the size (*i.e.*, number of parameters) and computations (*i.e.*, Floating Point Operations, FLOPS) of deep networks without losing accuracy. Most of the prior art in model compression literature focuses only on the computational aspects such as power, latency, memory, and energy consumption. Specifically, as summarized in Fig. 2.5, there are three main directions for model compression:

- **Pruning:** In this kind of model compression technique, the redundant or useless weights or even channels are completely removed [132] (see Fig. 2.5(a)). Moreover, pruning does *not* reduce the number of layers in the original model.
- **Quantization:** Conventionally, the deep networks are trained with 32-bit floating point weights and activations. Quantization techniques reduce the number of bits used for representing weights and activations, thereby reducing the memory footprint of models (see Fig. 2.5(b)) [60].
- **Knowledge Distillation (KD):** KD trains a significantly smaller student network to mimic a large teacher model (which we want to compress). KD allows us to directly reduce the number of layers compared to the teacher model [43] (see Fig. 2.5(c)). Of note, KD has also been shown to work with unlabeled datasets [4].

Since we focus heavily on KD in this thesis, we next discuss it in more detail.

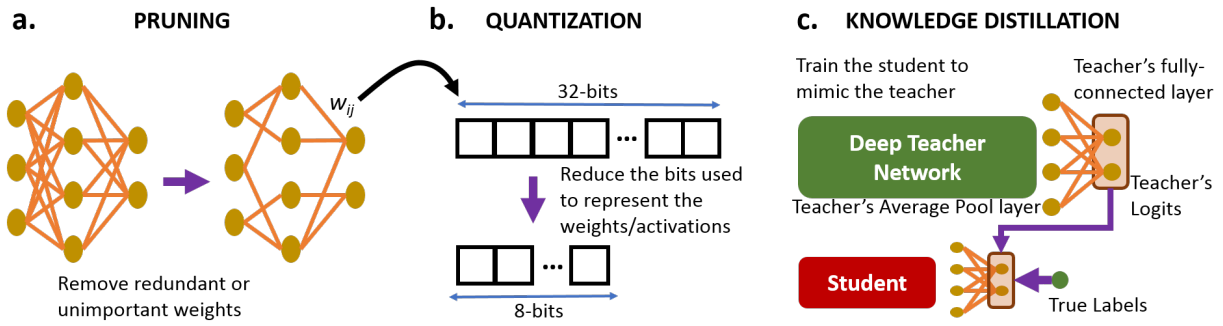


Figure 2.5: Key model compression techniques include (a) Pruning [132], (b) Quantization [60], and (c) Knowledge Distillation (KD) based teacher-student concepts [43].

Knowledge Distillation KD, as shown in Fig. 2.5(c), consists of two deep networks: (*i*) *Teacher* is the large deep network which we want to compress, and (*ii*) *Student* is a significantly smaller neural network which is trained to mimic the output of the teacher network. To mimic the teacher, the simplest way is to directly train a student model on teacher's *logits* (see Fig. 2.5(c)) – the unnormalized outputs of the teacher model before the softmax – instead of training on true labels from the dataset [4]. The basic idea here is that while training the student model, using logits instead of direct classification probabilities can transfer more information about the teacher to the student. Hinton *et al.* [43] argued that we must not only train the student model on the correct predictions made by the teacher, but must also quantify how wrong the teacher was about the incorrect classes. Hence, Hinton *et al.* introduced KD [43] which uses both the *hard-label loss* (based on true labels from the dataset), as well as the *soft-label loss* (based on logits) to train a significantly smaller student model.

Mathematically, let l_T and l_S be the logits of teacher and student respectively, and τ is a temperature parameter (see [43]), y be the true labels, and P_T^τ and P_S^τ respectively denote the softmax over relaxed logits l_T/τ and l_S/τ. Then, the KD loss ($\mathcal{L}^{kd}$) is given by:

$$\mathcal{L}^{kd}(\theta_S) = (1-\alpha)\mathbb{H}(y, P_S) + \alpha\mathbb{H}(P_T^\tau, P_S^\tau) \tag{2.10}$$

where, $\mathbb{H}$ is the standard cross-entropy loss, θ_S denotes the parameters of the student network, and α controls the weight of hard-label loss *vs.* soft-label loss. The temperature parameter in the second term of Eq. (5.1) improves knowledge transfer from the teacher to the student. Other variants of KD like Attention Transfer-based KD (ATKD) further use intermediate outputs of teacher's

convolution layers while training the student [135].

Model compression can indeed have major implication for deploying deep learning models at the edge. Other model compression techniques propose new types of efficient convolutions such as depth-wise separable convolutions [47, 103, 137], *etc*. Next, we describe the background on Neural Architecture Search, one of the newest problems in deep learning which aims to learn the architecture of the deep learning models and not just the model-weights.

2.4 Neural Architecture Search (NAS)

Designing highly-accurate deep networks has been largely driven by either trial-and-error [41, 47, 50, 58, 108], or via recently proposed NAS algorithms [66, 97, 100, 128, 141, 142]. Indeed, the manually-designed deep networks led to major improvements in architecture design practices which resulted in a carefully selected search space for NAS research. There are essentially three types of NAS: (*i*) Reinforcement learning with RNN controllers [97, 115, 141, 142], (*ii*) Evolution-based NAS [100, 128], and (*iii*) Gradient-based methods [20, 66, 111].

Fig. 2.6 summarizes the key NAS techniques. The reinforcement learning-based methods rely on a controller to sample architectures. The validation accuracy of the trained architectures is used as a reward function for the reinforcement learning-based controller (see Fig. 2.6(a)). Next, the evolution-based NAS techniques randomly select initial seed architectures and randomly train a subset of such models. Then, the worst models (in terms of validation accuracy) are eliminated and the remaining models become *parents*. The parents are then copied and *mutated* to become *child* networks. This mutation process is repeated until high-accuracy models are discovered (see Fig. 2.6(b)).

Both reinforcement learning- and evolution-based NAS techniques require a huge amount of computational power (*e.g.*, 800 GPUs were used in [141], and 500 GPUs were used in [142], 250 workers were used in [100]). Hence, a new gradient-based NAS approach has emerged recently, which is significantly more computationally-efficient than reinforcement learning or evolution-based NAS. As shown in Fig. 2.6(c), Differentiable Architecture Search (DARTS) [66] relaxes the continuous search space into a set of decisions that need to be made within each cell. To learn the probability distribution over architectures, DARTS first creates a super-network which trains both

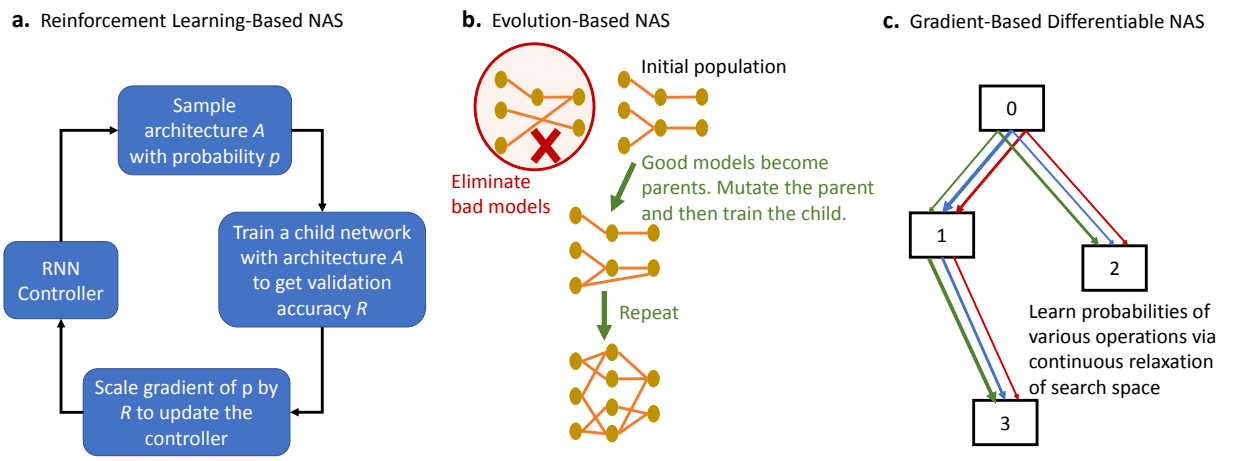


Figure 2.6: Major Neural Architecture Search (NAS) techniques (a) Reinforcement learning-based methods [97, 115, 141, 142]: The RNN-based controller samples architectures which are then trained. The validation accuracy of the trained model is used as a reward R for the controller so that models expected to obtain higher validation accuracy are more likely to be sampled. (b) Evolution-based NAS [100, 128]: Starting from an initial population, bad models are eliminated and good models are mutated and trained further (similar to an evolutionary process). Eventually, a highly accurate architecture gets discovered. (c) Gradient-based NAS [20, 66, 111]: Each edge is an operation (*e.g.*, separable convolution, skip connection, *etc.*) that connects two feature maps together. Gradient descent is used learn probabilities of high-accuracy architectures.

weights and architecture-decision-probabilities in a bi-level optimization framework. The final architecture is then sampled from the learned probabilities for all decisions in a cell (in Fig. 2.6(c), each color represents an operation such as depth-wise separable convolution or dilated convolution, *etc*., and thickness of arrow indicates how likely that decision is). This approach results in orders of magnitude faster search for good architectures.

Recently, many NAS techniques focus on finding efficient models for mobile applications. These techniques aim to optimize multiple objectives such as accuracy of the model, and latency constraints for a given edge device. Popular models include MNAS-Net [115], single-path NAS [111], Proxyless NAS [20], *etc*. Finally, despite the recent progress, a lot of open problems remain w.r.t NAS research. For instance, it has been shown that random search is competitive to NAS techniques [64, 66]; this has been attributed to the carefully designed search space for NAS such as depth-wise separable convolutions, dilated convolutions, *etc* [66]. Moreover, neither manually designed models nor NAS methods explore what architecture-level characteristics result in models with high accuracy. Therefore, such challenges present many new opportunities for

research in NAS.

This completes our discussion of representation learning topics. Next, we describe some background on Network Science.

2.5 Network Science

The field of network science has been used in a wide range of real-world problems such as information diffusion over social networks, modeling disease outbreaks, protein-protein interactions, transportation systems, power-grid, *etc*. Essentially, any problem in which different entities interact with each other can be represented as a network. Mathematically, the networks consist of nodes or vertices $\mathcal{V}$, and links or edges $\mathcal{E}$, where the different nodes are connected by links (*e.g.*, people connected with each other in a social network). This system can be represented by an adjacency matrix A, where $A_{ij} = w$ if there is link of weight w between nodes i and j. There are a number of interesting properties of networks such as degree distribution, community structure, centrality, *etc*. Below, we first discuss degree distribution and community structure. Then, we will explain network representation learning, a recent topic of interest which aims to learn features of nodes in a network directly from its structure.

2.5.1 Node Degree/Hubs

Given an undirected network $\mathcal{G}$ with nodes $\mathcal{V}$, links $\mathcal{E}$, and the adjacency matrix $\mathcal{A} = \{A_{ij}\}$ describing link weights between any two nodes $i, j \in \mathcal{V}$. Then, degree k_i of a node i refers to total number of links connected to node i. A network can have a *scale-free* structure where some nodes have many connections but most nodes have low degree [85]; such nodes with many connections in scale-free networks act as *hubs* of information. To illustrate, Fig. 5.5 shows nodes A, B, and C as examples of hubs in a network. As evident, these nodes are characterized by more connections than other nodes in the network. For example, in a social network, an important person with many connections will act as an information hub (i.e., information disseminated by this user will very quickly reach a large audience).

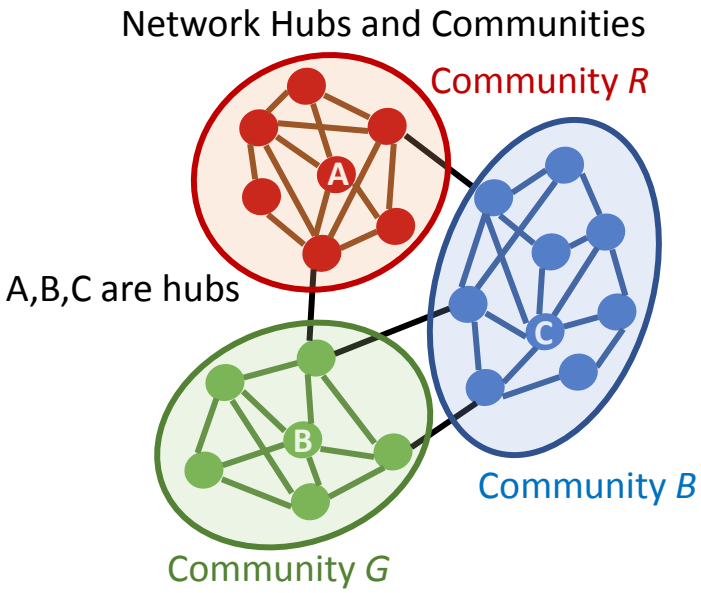


Figure 2.7: Network science concepts like degree distribution and community structure: Communities refer to groups of tightly connected nodes (shown in red, blue and green ovals), while degree refers to number of connections that a given node has. Nodes with a large number of connections are called hubs.

2.5.2 Community Structure

Many real world networks are organized into groups of tightly connected nodes known as the *community structure* [86]. For instance, in a social network, a community can refer to a group of users with common interests like sports or politics. Formally, a community can be defined as a group of nodes for which the number of connections within the group is significantly higher than what one would expect at random. Fig. 5.5 shows the community structure in a network. In contrast to older graph partitioning techniques, community detection automatically partitions the network into its natural groupings without requiring us to specify predefined parameters such as number/size of communities. Size and number of communities in a network instead depend on a user-specified *resolution* parameter γ.

As explained in the network science literature, communities are detected by maximizing a modularity function as follows [86]:

$$\max_{\mathbf{g}=\{g_0,g_1,\ldots,g_{l-1}\}} \frac{1}{2m}\sum_{ij}\left[A_{ij}-\frac{1}{\gamma}\cdot\frac{k_i k_j}{2m}\right]\delta(g_i,g_j), \tag{2.11}$$

where, m is the total number of edges, k_i refers to the degree (number of connections) of node

i, resolution γ controls the size/number of communities, and δ is Kronecker delta. The idea is to find groups of tightly connected nodes, $\mathrm{g} = \{g_0, g_1, \ldots, g_{l-1}\}$, which map the nodes $\mathcal{V}$ to l communities.

This completes the discussion of core network science concepts. As explained next, with the rise of machine learning and representation learning, a new problem space called *Network Representation Learning* has emerged in network science.

2.5.3 Network Representation Learning

Many problems in network science often consist of millions of nodes and billions of edges (*e.g.*, information diffusion on social networks). Such large-scale problems give rise to a new set of learning problems. For instance, *how can we automatically classify the interests of a blogger based on their social network?* Since the size of the network is huge, it is imperative that we *automatically* (and *not* manually) learn features of the nodes to perform such a classification. Hence, network representation learning techniques such as node2vec [35], LINE [116], DeepWalk [96], *etc*., aim to learn features by exploiting the network structure. Specifically, since the real-world networks are often characterized by community structure and structural equivalence (*e.g.*, nodes which are at structurally similar locations must share similar features), node2vec [35] learns the features while accounting for such network characteristics.

Note that, the above techniques exploit machine learning or representation learning for solving network problems (*e.g.*, classifying interests of a blogger in a social network). Hence, these techniques require the network structure as a starting point. In contrast, throughout this thesis, we exploit network science concepts for solving general machine learning and representation learning problems (*e.g.*, timeseries prediction, cancer prediction, digit recognition, learning from small data, model compression of deep networks, and neural architecture space exploration) without relying on network structure explicitly. The network for most of our problems is, in fact, unknown.

This ends the relevant background. In the next chapter, we propose new network science-based techniques for traditional machine learning problems such as timeseries prediction and application-specific feature extraction.

Chapter 3

Network Science for Spatiotemporal Timeseries Prediction and Feature Extraction

In this chapter, we exploit network science to address the conventional machine learning problems involving spatiotemporal timeseries prediction. Such problems arise in the scenario where some kind of traffic or fluid is flowing through space, and our objective is to predict short-term timeseries at each of the locations. The dynamics of traffic or fluid flowing through the space can be captured by a complex network. We argue that such dynamics should be a fundamental part of prediction models and, hence, the network must be taken into account while building machine learning models. In contrast, the existing machine learning models for timeseries prediction do not consider the network dynamics.

To this end, we demonstrate that new learning models that account for the underlying networks lead to better prediction compared to the models which do not consider such networks. Specifically, we address two spatiotemporal timeseries prediction problems from a network science perspective: (*i*) Learning over networks with known but rapidly changing structure for short-term timeseries prediction, and (*ii*) Application-specific feature extraction for predicting short-term timeseries when the underlying network structure is unknown but fixed. Of note, we will discuss the above two problems with concrete case studies from computational sustainability domain.

3.1 Learning over networks with known but dynamic structure

We start with a complex probabilistic learning problem in which the Bayesian network structure is known but is dynamically and rapidly changing. The class of dynamic Bayesian networks in which

the network structure changes rapidly are known as Non-stationary Dynamic Bayesian Networks. Towards this, we propose a new model to learn parameters on such dynamically changing networks and use them for the underlying prediction tasks. To give a concrete example of a situation where this kind of learning problems can arise, we provide a case study on a computational sustainability task, *i.e.*, to predict short-term solar energy and to quantify solar energy interdependence across a large river basin.

3.1.1 Non-Stationary Dynamic Bayesian Networks

Probabilistic graphical models such as Dynamic Bayesian Networks (DBN) have proven extremely useful for modeling cause-effect relationships, knowledge discovery, and short-term forecasting. Of particular importance are the non-stationary Dynamic Bayesian Networks (nsDBN) [101] in which the underlying network structure dynamically changes over time. Applications of nsDBN range from changing neural pathways in human brain, transcriptional regulatory networks during early development, all the way to traffic networks and sustainability/climate systems.

Like for any Bayesian network, there are two types of problems in nsDBN – parameter learning and structure learning. Prior art in nsDBN concentrates on structure learning on either piecewise stationary DBN [130] or on DBN with sparse structure changes [122]. However, dynamic Bayesian Networks with *rapidly* changing structures (*e.g.*, networks in which 50% of the structure changes within the next time step) have not been addressed. Therefore, in this chapter, we address a new problem in which the rapidly changing structure of Bayesian networks is known, but the parameters are unknown. Hence, the problem is to learn the parameters on Dynamic Bayesian Networks with rapidly changing structures. We propose a Network of Dynamic Bayesian Networks (NDBN) framework [11] to learn the parameters in such settings. The learned parameters can then be used for the underlying prediction problem.

This scenario, where the structure of DBN is known but rapidly changing can occur in a variety of problems with the most common case being the forecasting problems. For instance, suppose that there is a traffic (or some kind of fluid) flowing through a given space in known trajectories which are rapidly changing (perhaps known from the prior information or the available data). Then, the problem is to forecast the availability of this traffic/fluid at each node at the next time step (as illustrated in Fig. 3.1(a)). To give a more concrete example, we also consider a case study of

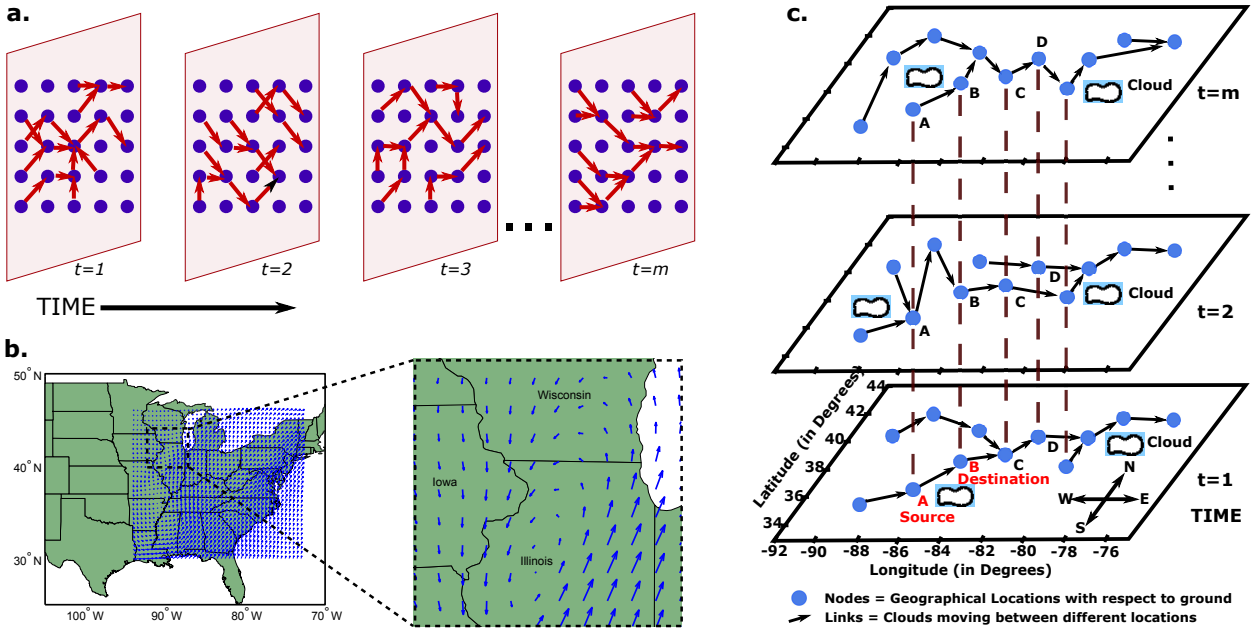


Figure 3.1: Applications of non-stationary dynamic Bayesian networks. (a) Schematic of any traffic/fluid flow problem in which flow trajectories are known but are rapidly changing over time. (b) Real-world example of wind and cloud movement over large regions. (c) Cloud movement can be shown as multilayer networks as shown in the schematic (a). Cloud and wind movement networks change rapidly (*e.g.*, up to 50% links change at every time step.

short-term solar energy prediction across a large river basin. As shown in Fig. 3.1(b) and (c), wind networks change rapidly with up to 50% links changing at every time step. Fig. 3.1(c) also shows how cloud movement can be represented as multilayer networks similar to Fig. 3.1(a). Towards this, we will use our proposed NDBN framework for scientific knowledge discovery, short-term solar energy forecasting, as well as quantifying solar energy interdependence (*i.e.*, who affects solar energy of whom) across large regions.

3.1.2 Problem Formulation

We now formulate the proposed parameter learning model for multilayer networks (Fig. 3.1(a)).

Learning over Multilayer Networks

Our model is based on nsDBN since the structure of our network is rapidly changing. Let the node attributes and link data at each of the n nodes for $m+1$ time steps be respectively given by $\mathbf{X}(t) = \{X_1^{(t)}, X_2^{(t)}, \ldots, X_n^{(t)}\}$ and $\mathbf{W}(t) = \{W_1^{(t)}, W_2^{(t)}, \ldots, W_n^{(t)}\}$, $t = 1, 2, \ldots, m+1$. While

the link data $\mathbf{W}(t)$ essentially characterizes the direction of flow of the traffic/fluid at each node, the node attributes $\mathbf{X}(t)$ represent the amount of traffic/fluid at each node at time t.

Fig. 3.2 shows the complete flow of our approach. First, using the above data, we construct the underlying network structures of our non-stationary DBN which are equivalent to network layers shown in Fig 3.1(a). These are schematically represented as flow patterns $P_1, \ldots, P_m$ in Fig. 3.2(a). We denote the network structure of flow pattern P_t as $G_X^{(t)}$. Hence, given this rapidly changing Bayesian Network structure, the problem is to (*i*) learn its unknown set of parameters $\mathbf{\Theta}$, and (*ii*) use the learned parameters to predict the node attributes at time $m+2$: $\mathbf{X}(m+2) = \{X_1^{(m+2)}, X_2^{(m+2)}, \ldots, X_n^{(m+2)}\}$[1]. Note that, Fig. 3.2(a) again looks like a multilayer network with each layer being a DBN. Therefore, following the nomenclature in [9, 32, 57], we call it a Network-of-Dynamic Bayesian Networks (NDBN) framework. Next, we discuss expansion-compression of node attributes as given below.

Expansion-Compression of Node Attributes We use the flow patterns and the node attribute data to observe three types of traffic/fluid movement: traffic/fluids moving "as they are", or "expanding" (*i.e.*, fluid amount increases as it moves from source to destination) or "compressing" (*i.e.*, the fluid decreases as it move from source to destination nodes). These source and destination pairs come from different links in the flow patterns as shown in Fig. 3.2(a)). Fig. 3.2(b) top panel illustrates expansion and compression cases of fluid movement. To take this into account, we introduce two additional nodes into the network: *Expansion node* ($\mathcal{E}$) and *Compression node* ($\mathcal{C}$). Next, we deduce the amount of expansion or compression occurring at each transition as illustrated in Fig. 3.2(b). This process gives rise to some additional nodes and links in the network (Fig. 3.2(b) lower panel). Finally, these Expansion and Compression nodes and links are deduced for all such source and destination pairs (given by each flow pattern P_t in Fig. 3.2(a)) and are appended to P_t to get a modified pattern matrix $\mathcal{D}_t$ (see Fig. 3.2(c)). Also, we assume that Expansion node has an infinite amount of traffic/fluids to supply. Of note, we use the terms "expansion" and "compression" as an analogy for how the clouds expand and compress as they move.

Main steps of our approach are summarized in Fig. 3.2. So far, we have discussed the steps (a),

[1]Note that, network structures $\{P_1, \ldots, P_m\}$ (Fig. 3.2(a)) use the data $\{\mathbf{X}(1), \ldots, \mathbf{X}(m+1)\}$ and $\{\mathbf{W}(1), \ldots, \mathbf{W}(m+1)\}$. Therefore, the prediction is carried out for $\mathbf{X}(m+2)$.

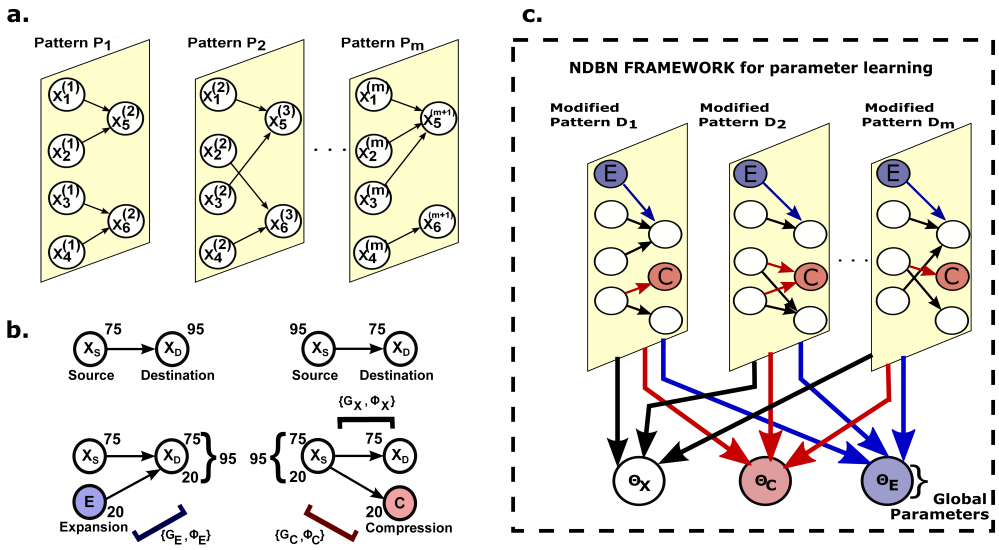


Figure 3.2: Complete flow of the proposed approach. (a) Traffic/fluid flow network structures at different times based on link data $\mathbf{W}(t)$. Pattern matrices at time t (P_t) capture the dynamics of the network, i.e., appearing and disappearing links. (b) Expansion (left) and compression (right) of the node attributes (*e.g.*, fluid amount) as they flow through the space. Expansion nodes supplies the residual fluid amount to the destination, while the compression node acts as a sink for excess fluid amount from source node. (c) Additional expansion and compression nodes are inferred and appended to the initial pattern matrices P_t to obtain modified pattern matrices (D_t). Parameter learning is performed on modified pattern matrices.

(b), and part of step (c) (*i.e.*, up to modified pattern matrices $\mathcal{D}_t$'s in Fig. 3.2(c)). These $\mathcal{D}_t$'s form the *training set* on which our machine learning model will be trained: $\boldsymbol{\mathcal{D}} = \{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_m\}$.

Parameter Learning for Non-Stationary Dynamic Bayesian Networks

Let us now describe the model parameters and the NDBN parameter learning framework. For all pairs of sources (S) and destinations (D), we define:

$$\begin{aligned} \theta_{SD} = &\quad P(X_D = d | \mathbf{Pa}(X_D) = s), \\ \theta_{\mathcal{E}D} = &\quad P(X_D = d | \mathbf{Pa}(X_D) = e), \\ \theta_{S\mathcal{C}} = &\quad P(X_D = c | \mathbf{Pa}(X_D) = s) \end{aligned} \tag{3.1}$$

where, X_D is the node attribute variable of the destination node toward which the traffic/fluids are moving, $\mathbf{Pa}(X_D)$ is total traffic/fluids at the set of parents of destination node and d, s, e, c are respectively the node attributes at destination, source, amount of expansion and compression. Intuitively, θ_{SD} represents the probability of traffic moving without expansion or compression, whereas

$\theta_{\mathcal{E}D}$ ($\theta_{S\mathcal{C}}$) represents the probability of expansion (compression) while moving from source node S to destination node D.

Since the Bayesian Network is completely observable, all the nodes are independent of their non-descendant nodes (given their parents). Therefore, the joint probability distribution for a single modified wind pattern matrix $\mathcal{D}_t$ is defined as:

$$P(\mathcal{D}_t) = \prod_{\{S,D\}} P(X_D^{(t)} = d' | \mathbf{Pa}(X_D^{(t)}) = s') = \prod_{\{S,D\}} \theta_{SD}^{(t)} \cdot \theta_{\mathcal{E}D}^{(t)} \cdot \theta_{S\mathcal{C}}^{(t)} \tag{3.2}$$

where, $t = 1, 2, \ldots, m$, $\{S, D\} \in G_X^{(t)}$ and $\theta_{SD}^{(t)}$, $\theta_{\mathcal{E}D}^{(t)}$ and $\theta_{S\mathcal{C}}^{(t)}$ are model parameters at time t.

The observed graph structures ($G_X^{(t)}$) are changing over time, giving rise to non-stationary DBN (nsDBN). We propose a Network of Dynamic Bayesian Networks (NDBN) framework to capture these changing structures. NDBN basically computes the maximum likelihood estimate of model parameters on time-varying Source- Destination pairs[2]. The proposed approach is shown in Fig. 3.2(c). We define three Global Parameters $\{\mathbf{\Theta}_{\mathcal{X}}, \mathbf{\Theta}_{\mathcal{E}}, \mathbf{\Theta}_{\mathcal{C}}\}$ which are complete sets of corresponding model parameters learned across different pattern matrices $\{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_m\}$:

$$\mathbf{\Theta}_{\mathcal{X}} = \{\theta_{SD}\},\ \mathbf{\Theta}_{\mathcal{E}} = \{\theta_{\mathcal{E}D}\},\ \mathbf{\Theta}_{\mathcal{C}} = \{\theta_{S\mathcal{C}}\}\ \forall\ \{S, D\} \tag{3.3}$$

Now let N_{SD} be the amount of traffic/fluids moving as they are from Source to Destination across time ($t = 1$ to $t = m$), $N_{\mathcal{E}D}$ be the amount of expansion and $N_{S\mathcal{C}}$ the amount of compression. For brevity, we define the following: $\mathbf{\Theta} = \{\mathbf{\Theta}_{\mathcal{X}}, \mathbf{\Theta}_{\mathcal{E}}, \mathbf{\Theta}_{\mathcal{C}}\}$. Then, the log-likelihood is given as:

$$l(\mathbf{\Theta}|\boldsymbol{\mathcal{D}}) = \log \left(\prod_{\{S,D\}} \theta_{SD}^{N_{SD}} \cdot \theta_{\mathcal{E}D}^{N_{\mathcal{E}D}} \cdot \theta_{S\mathcal{C}}^{N_{S\mathcal{C}}} \right) \tag{3.4}$$

Next, we learn the model parameters for a given training set by maximizing the log-likelihood:

$$\begin{aligned} \underset{\theta_{SD}, \theta_{\mathcal{E}D}, \theta_{S\mathcal{C}} \forall D}{\text{maximize}} \quad & l(\mathbf{\Theta}|\boldsymbol{\mathcal{D}}) \\ \text{subject to} \quad & \sum_D (\theta_{SD} + \theta_{S\mathcal{C}}) = 1 \\ & \sum_D \theta_{\mathcal{E}D} = 1 \end{aligned} \tag{3.5}$$

[2]The model parameters are defined in terms of Sources and Destinations and imply if the traffic/fluids move as they are or if they expand or compress. Therefore, using Maximum Likelihood Estimate (MLE) for parameter learning in this scenario basically computes MLE on time-varying Source and Destination combinations (determined by changing links in modified pattern matrices $\mathcal{D}_t$ for time $t = 1, 2, \ldots, m$) and hence takes changing structures into account.

The constraints in our problem (Eq. 3.5) come from the way Expansion and Compression nodes are defined. They basically indicate that the total traffic/fluid amount is conserved in the space. Solving the above problem by computing a Lagrangian gives the following Maximum Likelihood Estimates:

$$\hat{\theta}_{SD} = \frac{N_{SD}}{\sum_D N_{SD} + N_{S\mathcal{C}}}, \quad \hat{\theta}_{S\mathcal{C}} = \frac{N_{S\mathcal{C}}}{\sum_D N_{SD} + N_{S\mathcal{C}}}, \quad \hat{\theta}_{\mathcal{E}D} = \frac{N_{\mathcal{E}D}}{\sum_D N_{\mathcal{E}D}} \tag{3.6}$$

In short, Eq. 3.6 suggests that the parameters $\{\hat{\theta}_{SD}, \hat{\theta}_{S\mathcal{C}}, \hat{\theta}_{\mathcal{E}D}\}$ are given by the amount of traffic/fluid passing without change or compressing or expanding divided by the total amount at the source. Note that, MLE is used here for parameter learning since we do not know anything about the prior, making the use of Maximum *a posteriori* (MAP) unsuitable. Next, we discuss how to predict node attributes from learned parameters.

Prediction of Node Attributes from Learned Parameters

To predict the node attributes at all nodes for the next time step, it suffices to know which traffic/fluids move as the are from source to destination, which traffic/fluids expand, which compress, and the amount of expansion/compression occurring at each link. Fortunately, we are given link data $\mathbf{W}(m+1)$, *i.e.*, the information about the direction towards which the traffic/fluid was moving at the last time step. This can be used to construct the network structure at time $m+1$: $G_X^{(m+1)}$. This network structure shows in which direction the traffic/fluids will move but does not give any information on which they will expand or compress as they move from source to destination.

Note that, we need both $\mathbf{X}(t)$ and $\mathbf{X}(t+1)$ to deduce the additional Expansion and Compression links (Fig. 3.2(b)). Since we are given the node attribute data $\mathbf{X}(m+1)$ but not $\mathbf{X}(m+2)$ (which we want to predict), we cannot deduce Expansion/Compression links and, therefore, cannot generate the modified pattern $\mathcal{D}_{m+1}$. In order to find these Expansion/Compression links, we incrementally maximize the logarithm of estimate of joint probability:

$$\hat{\Pr}(\mathcal{D}_{m+1}) = \prod_{\{S,D\}} \hat{\theta}_{SD} \cdot \hat{\theta}_{\mathcal{E}D} \cdot \hat{\theta}_{S\mathcal{C}} \tag{3.7}$$

where, $\hat{\theta}_{SD}$, $\hat{\theta}_{S\mathcal{C}}$ and $\hat{\theta}_{\mathcal{E}D}$ are the parameters learned on the training dataset and $\{S, D\} \in G_X^{(m+1)}$. Maximizing Eq. 3.7 basically finds the *most likely* links from Expansion to Destination nodes ($\mathcal{E} \rightarrow D$) and those from Source to Compression nodes ($S \rightarrow \mathcal{C}$). These links tell us which traffic/fluids

will expand/compress while moving from source to destination. Furthermore, while training the model on $\mathcal{D} = \mathcal{D}_1, \ldots, \mathcal{D}_m$, we also compute Expected Expansion and Expected Compression for each Source-Destination pair. This quantifies the amount of expansion/compression taking place. Therefore, once we know which traffic/fluids will expand/compress and how much they will expand/compress, prediction simply follows from the observed $\mathbf{X}(m+1)$ data. We call this algorithm as Expansion-Compression prediction. For more details, please refer to [11].

This concludes the learning over multilayer networks. Next, we present a concrete case study on a computational sustainability problem which uses our proposed NDBN model. Specifically, since cloud movement across large regions results in rapidly and dynamically changing multilayer networks (see Fig. 3.1(c)), we use the proposed model to (i) quantify solar energy interdependence due to cloud movement (via NDBN parameter learning), and at the same time (ii) generate predictions at a large number of locations simultaneously (using the prediction procedure described above). To generate the global response towards sustainable energy production, both of these tasks are essential components and can help globally coordinate the actions of power-grid operators.

3.1.3 Case Study: Spatiotemporal Solar Irradiance Prediction Problem

Let us now use the proposed NDBN model for parameter learning and short-term prediction on a real-world computational sustainability problem: *Given spatiotemporal cloud and wind data, predict the solar irradiance simultaneously at all locations.*

Experimental Setup and Description of Data Sources

We use hourly wind and cloud cover data obtained from National Center for Atmospheric Research (NCAR) CISL Research Data Archive (CISL-RDA) [120]. The data is available on a regular grid (0.3125° latitude/longitude resolution) as shown in Fig. 3.1(b) (left). The zoomed-in portion towards the right shows a snapshot of wind movement from one location to other.

The amount of cloud at each grid-cell is given by cloud cover data (also called cloud fraction). *Cloud fraction* at a grid-cell is defined as percentage of area covered by clouds (varies from $0 - 100\%$). A sample of cloud fraction data is given in Table 3.1. Each location (given as latitude and longitude in the table) represents a grid-cell shown in Fig. 3.1(b). We have 3417 such grid-

Table 3.1: Cloud fraction data sample for the year 1997

Latitude (North)	Longitude (West)	Cloud fraction (in %) at time				
		$1h$	$2h$	...	$8759h$	$8760h$
45.7415	−80.3125	93	95	...	95	91
45.7415	−80	87	86	...	84	77
45.7415	−79.6875	82	79	...	75	67

cells in and around the Ohio Basin region (bounds: $[30°\text{N}, 46°\text{N}]$ and $[-94°\text{W}, -73°\text{W}]$) of more than $204,000$ sq. miles. The cloud fraction data that we use is for entire years 1997 and 2009 containing data for all $24 \times 365 = 8760$ hours[3] (as shown in Table 3.1). For our problem, the wind data serves as the link information $\mathbf{W}(t)$, whereas the cloud fraction data serves as node attributes $\mathbf{X}(t)$. Multilayer cloud networks (*e.g.*, Fig. 3.1(c)) are created using the wind data as the cloud movement is guided by the winds (for more details, please refer to [11]). A thorough analysis of cloud networks reveals that our problem is multivariate, has non-stationary data (with Detrended Fluctuation Analysis parameter $\alpha > 1$), and dynamically and rapidly changing network structure (around 50% edges change at every time step).

The prediction procedure described above yields one step forecasts at each of the n nodes in the network simultaneously. Therefore, the prediction can be carried out for 1-hour, 2-hours or 3-hours ahead of time by appropriately selecting the size of prediction time step (Δt) as 1, 2 or 3 hours, respectively. We call this step size (Δt) as time resolution. We conduct several experiments at varying number of nodes, training set sizes, and time resolutions to obtain 1, 2 and 3-hour predictions. Specifically, we run experiments on 400, 800, 1200, 2000 and 3417 nodes around the Ohio River Basin. Also, since the problem is non-stationary, the learned parameters keep changing with the training set.

Knowledge Discovery – Power Laws and Kolmogorov-Smirnov Tests

We now analyze the statistical properties of learned parameters for all Source-Destination pairs. Fig. 3.3 shows complementary cumulative distribution plots for $\mathbf{\Theta}_{\mathcal{X}}$ and $\mathbf{\Theta}_{\mathcal{C}}$.

[3]We express times as hour numbers (varying from 1 to 8760), *e.g.*, $t = 1h$ stands for Jan 1, 1997, 00:00:00 UTC. Similarly, $t = 1001h$ would stand for Feb 11, 1997, 16:00:00 UTC timezone.

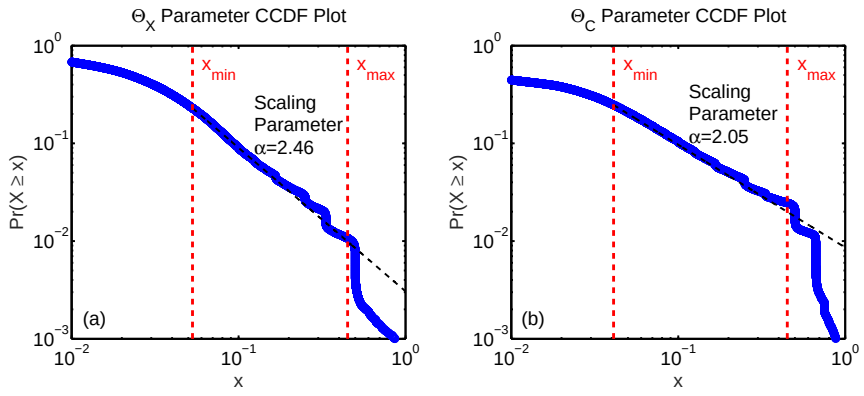


Figure 3.3: Complementary Cumulative Distribution functions (CCDF) for (a) $\Theta_{\mathcal{X}}$, and (b) $\Theta_{\mathcal{C}}$ parameters. The x_{min} and x_{max} limits are shown to mark the scaling region for both the parameters. Using a 2-sample KS test we prove that the linear region in the plots above is much more likely to be a power law than an exponential distribution.

Solar Energy Interdependence Like many natural (e.g. river systems *etc.*) and man-made (e.g. social networks) systems, the cloud movement patterns over time likely follow a power law distribution. Also, the density plot for $\mathbf{\Theta}_{\mathcal{E}}$ (not shown) is approximately Gaussian [11]. This may be due to the assumption that the Expansion node has infinite clouds to supply to its destinations. Note that, these learned parameters are basically the probabilities of cloud movement between various source and destination nodes in the network. Hence, these parameters can be seen as quantifying the spatiotemporal solar energy interdependence across the region considered (*i.e.*, how much solar energy harnessed at two locations will be affected by cloud movement between the locations).

Kolmogorov-Smirnov Hypothesis Tests We follow the methods given in [22] to prove that the linear region in the Fig. 3.3 is more likely to follow a power law than an exponential distribution. As shown in Fig. 3.3(a) and (b), we estimate the scaling parameter α for $\Theta_{\mathcal{X}}$ and $\Theta_{\mathcal{C}}$ parameters to be 2.4691 and 2.0515, respectively; both are typical values for power law type of behavior. The method in [22] also computes the value x_{min} where the scaling region begins. We obtain the x_{min} for $\Theta_{\mathcal{X}}$ and $\Theta_{\mathcal{C}}$ as 0.0530 and 0.0410 respectively. Further, in our case, clearly the cutoff region is not a power law. It is not exponential either, it is somewhat arbitrary (probably since our parameters are constrained to be in between 0 and 1). Therefore, using Fig. 3.3, we put a x_{max} cutoff at 0.45 for both $\Theta_{\mathcal{X}}$ and $\Theta_{\mathcal{C}}$ to remove the cutoff region.

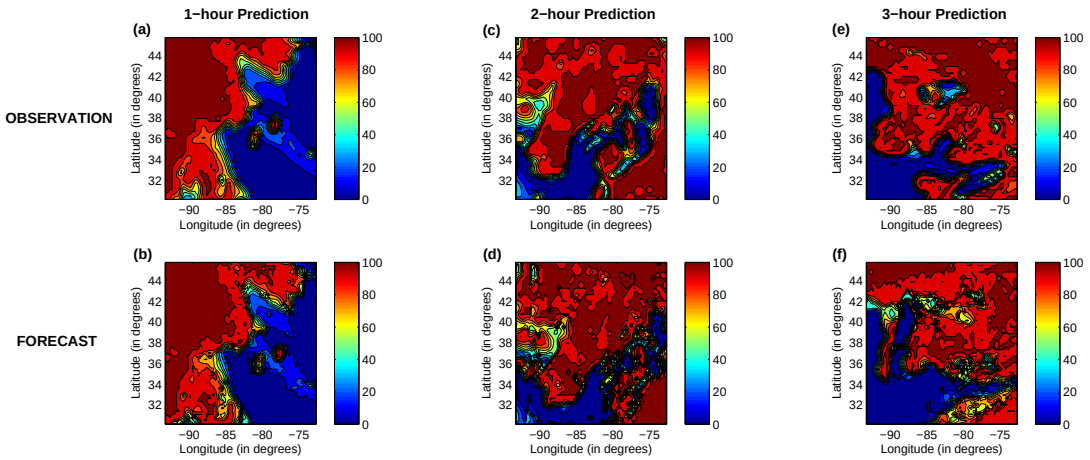


Figure 3.4: Cloud fraction prediction for 1h, 2h and 3h time resolutions for the 3417 node network that corresponds to the region in and around Ohio river basin. Prediction for $\Delta t = 1$ is at the 501 hour, for $\Delta t = 2$ is at the 1001 hour and for $\Delta t = 3$ is at the 1501 hour. (a) Observation of historic data, and (b) Forecast with our approach at time $t_{pred} = 501$ hour, (c) Observation, and (d) Forecast at time $t_{pred} = 1001$ hour, (e) Observation, and (f) Forecast at time $t_{pred} = 1501$ hour. Red (blue) stands for high (low) value of cloud fraction.

Next, we run the 2-sample Kolmogorov-Smirnov (KS) Test in the linear region for both power law and exponential hypothesis with (significance level, $\alpha = 0.001$ for each individual KS Test) 2500 times. We find that about 78% of the tests were positive for power law for $\Theta_{\mathcal{X}}$. On the other hand, for $\Theta_{\mathcal{C}}$, only about 25% of the tests were positive for power law. For the same settings, however, 0% of the tests came out positive for exponential distribution for both $\Theta_{\mathcal{X}}$ and $\Theta_{\mathcal{C}}$ parameters. This experiment, at the very least, suggests that in the linear region, both of these parameters show some evidence in favor of power law distribution and no evidence in favor of exponential distribution. Based on this experiment, therefore, we conclude that in the linear region, both of these parameters are more likely to follow a power law distribution than an exponential distribution.

Global Scale Predictions and Comparison with Null Models

Fig. 3.4 shows the cloud fraction prediction results of the proposed framework in the region covering the Ohio River Basin and beyond. Fig. 3.4(a), (c) and (e) are the actual observations of cloud fraction at 3417 different locations, while Fig. 3.4(b), (d) and (f) are the cloud fraction predictions produced by the model. As evident, the model predictions are fairly accurate.

Similar experiments for $\Delta t = 1, 2, 3$ were conducted for different training sets, prediction

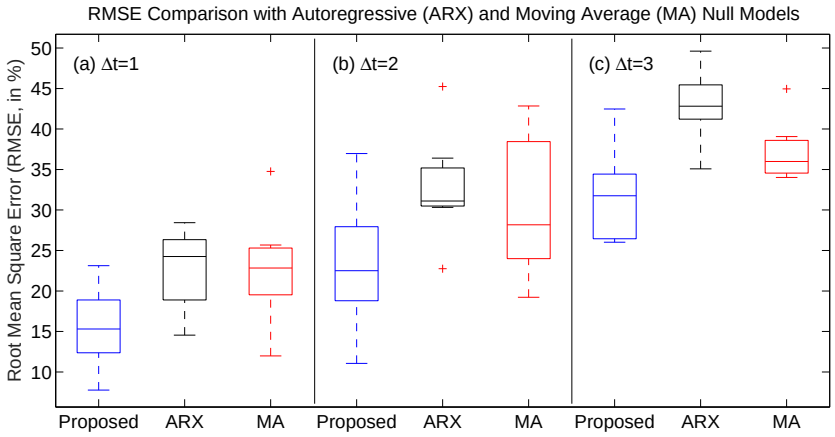


Figure 3.5: Comparison of the proposed model with two null modes: (i) Autoregressive with 8 exogenous inputs as the neighbors of a node (ARX), and (ii) Moving Average (MA) null model. The RMSE results show that our model performs better than both null models for all three time resolutions, (a) $\Delta t = 1$ hour, (b) $\Delta t = 2$ hours, and (c) $\Delta t = 3$ hours.

times and different network sizes. The results are summarized in the subsection below.

Comparison with null models We now compare our model against two null models to show that our results are good simply not because of good spatial correlations in data (*i.e.*, correlations between a node and its neighbors) and that learning over a network is important for better performance.

The two null models we consider are: (i) Moving Average Model (a standard null model), and (ii) Autoregression model with exogenous inputs as 8 nearest neighbors of a node (ARX). Null model (ii) is similar to the approach used by Dambreville *et al.* in [24] and it *explicitly* accounts for spatial correlation between a node and its neighbors. While Dambreville *et al.* apply their model only at a single location, we ran the model *at all* locations in the region considered (*i.e.*, this model does not consider a cloud network like in our approach).

Fig. 3.5 summarizes the results for (a) $\Delta t = 1$ hour, (b) $\Delta t = 2$ hours, and (c) $\Delta t = 3$ hours ahead predictions. Clearly, our results indicate that the proposed model performs better than both of these null models in all the cases. This proves that our model's performance is high not only because of spatial correlations in data since otherwise the null model (ii) would have done better than the proposed model. Next, we discuss how to estimate solar irradiance from cloud fraction.

From Cloud Fraction to Solar Irradiance We estimate the solar irradiance ($= solar\ power/area$) from cloud fraction values and Sun's position. The following formula is used to convert the cloud fraction to *approximate* solar irradiance [83]:

$$R = R_0 sin(\phi - 30)(1 - 0.75 \times (cf/100)^{3.4}) \tag{3.8}$$

where, R_0 is clear sky insolation (990 Watt/m^2), ϕ is mean of solar elevation angles at current and previous hour, and cf is cloud fraction (in %) at a certain location.

We conduct an experiment on the 3417 node network for $t_{pred} = 2301h$, $\Delta t = 1h$. Fig. 3.6(a) and (b) show the results for cloud fraction prediction, while Fig. 3.6(c) and (d) show the results for solar irradiance prediction. The RMSE for cloud forecast is 15.50% and that for solar irradiance is 41.34 Watt/m^2. Evidently, the prediction is quite accurate at most locations with only a few inaccurate forecasts. Of note, our model predicts solar irradiance at all locations *simultaneously* (which can be used to estimate solar supply at *all* locations). Combining this prediction information with knowledge obtained from parameter learning (*i.e.*, who impacts whom due to moving clouds) can allow policy-makers/engineers to optimally utilize the solar resources.

Discussion on Computational Sustainability Application Our model is the first step towards solving a global problem (*i.e.*, controlling the global availability of solar resources based on the dynamics of cloud movement across large regions) and this problem cannot be solved without a network that explicitly captures the spatiotemporal dependencies due to clouds moving across various locations. This is unlike many existing approaches [19, 21, 24, 25, 37, 73, 107] which aim to predict solar power only at a single location. On the other hand, our focus in this chapter is *not* to improve the accuracy of state-of-the-art solar prediction at a single location but rather to bring the global energy interdependencies into the picture. This interdependence information can promote power-grid operators to help out each other in case of energy deficit/excess at different locations and we capture these interdependencies using the proposed probabilistic framework for nsDBN.

So far, we have focused on learning problems for which the network structure is known and rapidly changing. Next, we will discuss learning on problems for which the network structure is not known. We will further demonstrate how network science can enable an effective application-specific feature extraction for highly accurate timeseries prediction.

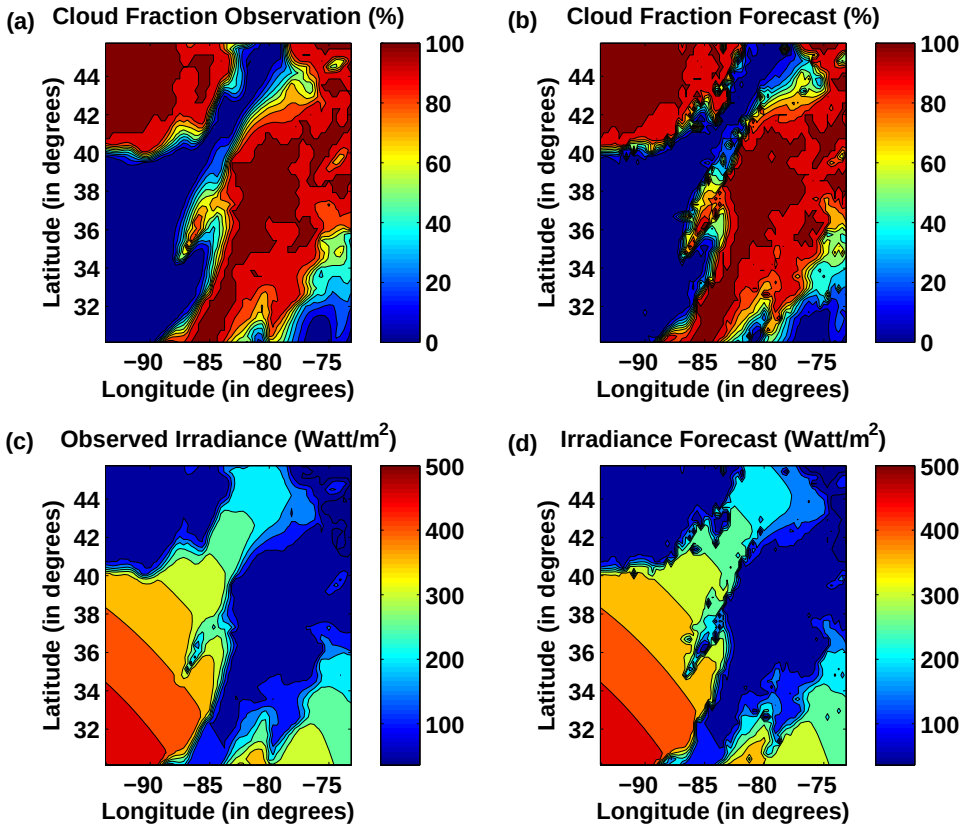


Figure 3.6: (a) Cloud fraction observation and, (b) prediction. (c) Solar irradiance observation and, (d) prediction. Red (blue) means high (low) cloud fraction and solar irradiance. Solar irradiance is derived from Eq. 5.

3.2 Exploiting Network Science for Application-Specific Feature Extraction

For many real-world machine learning problems, the underlying network structure that guides the dynamics is *not* known, and only raw high-dimensional timeseries data is available. Therefore, for such problems, we need to infer the network structure from the given high-dimensional raw data. Once the network is inferred, we can exploit it to reveal features that can improve the performance of the learning model. Hence, we must first infer the network structure from the raw data, and then use the network for application-specific feature extraction.

As discussed next, we demonstrate the above problem by considering a concrete case study on another computational sustainability problem for spatiotemporal river flow prediction.

3.2.1 Spatiotemporal River Flowrate Prediction

The timeseries data for river flowrates is hard to predict as it exhibits non-stationary behavior which, in turn, results in sudden peaks and troughs. An example of such hard-to-predict timeseries is shown in Fig. 3.7(a). These characteristics result from a complex river network which, as shown in Fig. 3.7(b), clearly consists of a fractal structure. Hence, the network structure must be taken into account while creating machine learning models for river flowrate prediction. Note that, only the raw timeseries data is available across a region and, hence, the network structure is unknown. Therefore, the problem is to *first infer the river network from the high-dimensional timeseries data, and then use the network to extract features for short-term river flowrate prediction at several locations*.

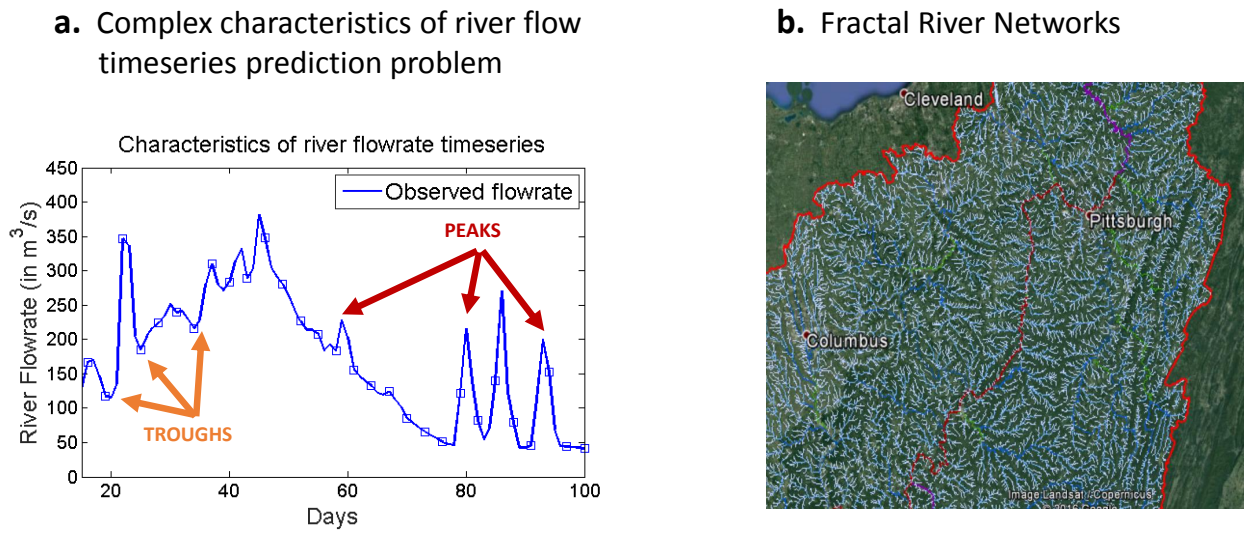


Figure 3.7: River flowrate and network characteristics: (a) The timeseries data for river flowrate exhibits non-stationarity which results in sudden peaks and troughs. (b) The river network clearly demonstrates its complex network characteristics such as fractal structure. Hence, the network must be taken into account while designing the machine learning models.

To this end, we first present the dataset sources for our problem, and our proposed approach for network inference. We will then propose the K-Hop Learning framework [10, 14] for application-specific feature extraction. Following this, we will demonstrate the effectiveness of features learned via K-Hop Learning for short-term river flow prediction.

Table 3.2: River Flowrate data sample for the year 1997

Node No.	Latitude (North)	Longitude (West)	River Flowrate (in m^3/s) at time			
			$1h$	$2h$	...	$8760h$
3029	40.3125	−79.8125	404.97	404.34	...	269.95
3185	40.5625	−79.8125	715.60	714.60	...	431.63
3107	40.4375	−79.8125	1131.3	1130.1	...	707.93

Data Sources In this work, we use two major datasets: *(i)* daily river flowrate [81] and *(ii)* daily precipitation [82] for the Ohio River Basin, USA (total area: $204,000$ sq. miles). The spatial resolution of both datasets is 0.125° latitude by 0.125° longitude. At this scale, Ohio River Basin contains a total of 3681 locations. Table 3.2 shows a sample of flowrate data for 1997 ($24h \times 365 = 8760h$). Here, each node number refers to a location with the given latitude and longitude. For the purpose of our experiments, we have used data for the years 1992-1997. Of note, we generate daily timeseries from the original hourly data. Next, we use this timeseries data to infer the river network.

3.2.2 Proposed Correlations-Based Network Inference

Let us now infer the approximate river network from the historic data. Suppose this network is represented by an adjacency matrix R and its nodes by Node IDs $\{1, 2, \ldots, 3681\}$ (see Table 3.2). Also, let $r_i(t)$ be the flowrate of a river at node i. The following two observations are critical to our approach: (*i*) Rivers combine to form larger rivers, and (*ii*) Average flowrate increases from smaller rivers towards larger rivers. The first observation above implies that, for a given node i, the sum of flowrates of its neighboring nodes (denoted $\mathcal{N}(i)$) must be highly correlated with the flowrate at node i. Here, $\mathcal{N}(i)$ consists of eight nearest neighbors of node i in the grid. Then, the directions in the river network can be obtained using the second observation.

Based on these observations and assuming that no more than 3 rivers meet at a single place, we infer the river network by maximizing the correlation between flowrate at node i and sum of

flowrates at its immediate neighbors. For each node i, therefore, the problem is to:

$$\begin{aligned} \underset{R(i,j)}{\text{maximize}} \quad & corr\left(r_i(t), \sum_{j\in\mathcal{N}(i)} r_j(t)\cdot R(i,j)\right) \\ \text{subject to} \quad & r_i^{avg} > \sum_{j\in\mathcal{N}(i)} r_j^{avg}\cdot R(i,j) \\ & \sum_{\forall j} R(i,j) \leq 3, \;\; R(i,j)\in\{0,1\}, \forall i,j \end{aligned} \tag{3.9}$$

where, r_i^{avg} refers to the average flowrate at node i. The final river network is obtained by solving Eq. 3.9 for all nodes. Once the network is obtained, we validated the river network by plotting a directed path from Pittsburgh in Google Earth. We found that the path not only ended at Cairo, IL, where the Ohio river meets the Mississippi river, but also follows the exact same trajectory as Ohio River. This verifies that the resulting network structure learned is correct.

3.2.3 Application-Specific Feature Extraction via K-Hop Learning

We now leverage the river network to reveal the features necessary for predicting the upcoming flowrate peaks 24-hours ahead. The schematic of river network is shown in Fig. 3.8(a), where nodes are locations, links show the rivers flowing between locations, and we have river flowrate data for each node. Then, the problem is to predict the river flowrate 24-hours ahead at node i as shown in Fig. 3.8(a).

Imagine the flowrates at each of these nodes as a signal flowing through the river network. Next, we consider the pair of orange and green nodes shown in Fig. 3.8(a). As the flowrate signal travels from the orange node (*i.e.*, the source) towards the green node (*i.e.*, the destination), there will be a delay as it moves from the source to the destination. Therefore, as the signal travels through the network, *at every hop*, it will experience and accumulate some *delay*. This delay is a function of physical features of various rivers such as elevation and topographical or geographical features. Consequently, the delay will be different not only for different nodes, but also for different rivers. Since the topography, geography and elevations of rivers do not change very quickly, we can easily learn such delays for all nodes using historic data. Hence, for every node i, we can *learn* the *optimal number of hops*, K, such that the total delay along those K hops is approximately 24 hours. In other words, as illustrated in Fig. 3.8(a), if the flowrate at node i ($r_i(t)$) peaks at times

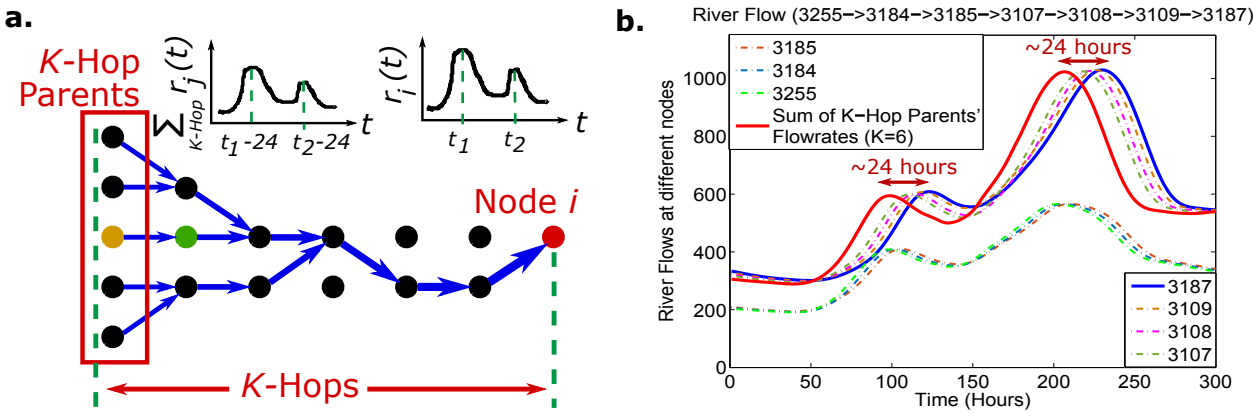


Figure 3.8: K-Hop Learning: (a) River network schematic. Upper panel (right): $r_i(t)$ plot shows the flowrate at node i with two peaks at times t_1 and t_2. Upper panel (left): the sum of K-Hop parents' flowrates plot – corresponding peaks appear almost 24 hours earlier. (b) Real data shows that the sum of K-Hop parents' flowrates detects peaks 24-hours ahead.

t_1 and t_2, we can detect these peaks 24 hours ahead using the sum of flowrates of its parent nodes, some K-Hops upstream in the network.

The above ideas can be formulated as an optimization problem. Let $\mathcal{P}_K^i$ be the K-Hop parents of a given node i. Then, the optimal K for each node can be found by maximizing the cross-correlation between flowrate at node i and sum of flowrates of its K-Hop Parents:

$$\begin{aligned} \underset{K}{\text{maximize}} \quad & xcorr\left(r_i(t), \sum_{j \in \mathcal{P}_K^i} r_j(t)\right) \\ \text{subject to} \quad & 21 \leq \tau_{xcorr}^{max} \leq 25 \\ & 3 \leq K \leq 12 \end{aligned} \tag{3.10}$$

where, τ_{xcorr}^{max} is the lag which maximizes the cross-correlation function. Intuitively, τ_{xcorr}^{max} corresponds to the total delay along the K hops between node i and its K-Hop parents. Therefore, the first constraint in Eq. 3.10 ensures that the K-Hop parents can detect the upcoming river flow peaks at node i, 24-hours earlier. Moreover, since we can predict such peaks only approximately, the above constraint aims to keep τ_{xcorr}^{max} between 23 ± 2 hours. The second constraint restricts K to be between 3 and 12 for efficiency purposes. Also, many very small rivers (average flowrate < 25 m^3/s) did not have parents more than 12 hops upstream in the network. Therefore, we do not focus on predictions for these rivers. The smallest rivers we consider have a flowrate of ≈ 150 m^3/s.

The above optimization problem is solved for *all nodes* in the river network for five different

years (1992-1996). Note that, the value of K for each node is almost the same for various years. Still, we take this small variation into account by averaging the K obtained for each node across the five years. Hence, we obtain a mean K for 844 nodes which correspond to significant rivers with flowrates ≥ 150 m^3/s. The rest 2837 nodes are very small rivers and are, therefore, insignificant.

Finding K-Hop parents for every node in the network can be implemented faster than a Depth First Search, since for each node, $K \in \{3, 4, \ldots, 12\}$ reduces the river network to a small tree with number of nodes $n << N$, the total number of nodes. Therefore, finding K-Hop parents takes $\mathcal{O}(N \cdot n)$. Next, computing cross-correlations for each node takes $\mathcal{O}(N \cdot p)$ (where, p is the length of timeseries). Therefore, the overall complexity of the problem is $\mathcal{O}(N \cdot (n+p))$ and since $n << p$, this is $\approx \mathcal{O}(N \cdot p)$. Practically, it takes about 15 seconds per year in MATLAB with 8 parallel workers to solve this problem for all nodes in the network. The code to find K-Hop away parents in the river network (for a specific K) can be implemented in a recursive fashion like a Depth First Search (see Algorithm 1).

Algorithm 1 `Compute_KHop_Parents`

Input: River Network R, Node i, $parentsSet$, $HopCount$, K
Output: $parentsSet$
$ImmediateParents \leftarrow$ find$(R(:, i) == 1)$ // Find immediate parents of node in river network i
if $HopCount == K$ **then**
  $parentsSet \leftarrow parentsSet \cup ImmediateParents$
  return
else
  for all j in $ImmediateParents$ **do**
    $HopCountTemp = HopCount + 1$
    $parentsSet \leftarrow$ `Compute_KHop_Parents`$(R, j, parentsSet, HopCountTemp, K)$
  end for
end if

Finally, we demonstrate the power of K-Hop Learning in detecting river flow peaks 24-hours ahead of time. As an example, we analyze the effectiveness of K-Hop parents for detecting

flowrate peaks of node 3187. Fig. 3.8(b) shows the flowrate of node 3187 (thick blue line) and sum of flowrates of its K-Hop parents (thick red line, $K = 6$). Note that, the river hierarchically flows from node 3255 to node 3184 to 3185, all the way to 3187. Looking closely at the river flow plots for each of the locations reveals that there is some delay between the peaks for each successive hop. Hence, going sufficiently upstream in the network (*e.g.*, $K = 6$ here) can enable us to detect the peaks at 3187 24-hours earlier. As evident, the peaks at node 3187 can be detected 24-hours ahead by looking at its K-hop parents (node 3255 shown in Fig. 3.8(b) is only one of the K-Hop parents). Therefore, K-Hop Learning can significantly improve the prediction accuracy.

Short-term Prediction Model and Experimental Setup

We now use the above K-Hop parents-based features in a machine learning model to predict the short-term flowrates for various rivers. Specifically, to predict the flowrate 24-hours (*i.e.*, 1-day) ahead at each node i, we use a simple linear model consisting of two features: (*i*) Cumulative 24-hours precipitation at node i, and (*ii*) Sum of daily flowrates of its K-Hop parents. To compare our model with other timeseries-based machine learning models, we replace the *second* feature above by timeseries of daily flowrate at node i itself; we refer to these models as *timeseries-based machine learning models*.

We account for non-stationarity in flowrate data using *rolling window*. In a rolling window method, the training period of a model is 'rolled' or moved forward, as soon as the next observation becomes available. Therefore, we train our model for 20 days, test on the next day, then move the training set forward by one day, test on the day after that, and so on. Also, we show the flowrate prediction experiments for four different sized rivers in Ohio River Basin, USA $-$ small (average flowrate $\sim$ 150 m^3/s), intermediate ($\sim$ 400 m^3/s), large ($>$ 1000 m^3/s), and very large ($\sim$ 10^4 m^3/s).

3.2.4 River Flowrate Prediction Results

We now present the flowrate prediction results using K-Hop Learning and compare them to four timeseries-based machine learning models: (*i*) Autoregression (AR), (*ii*) Random Forests (RF), (*iii*) SVM Regression (SVM-R), and (*iv*) K-Nearest Neighbors Regression (KNN-R).

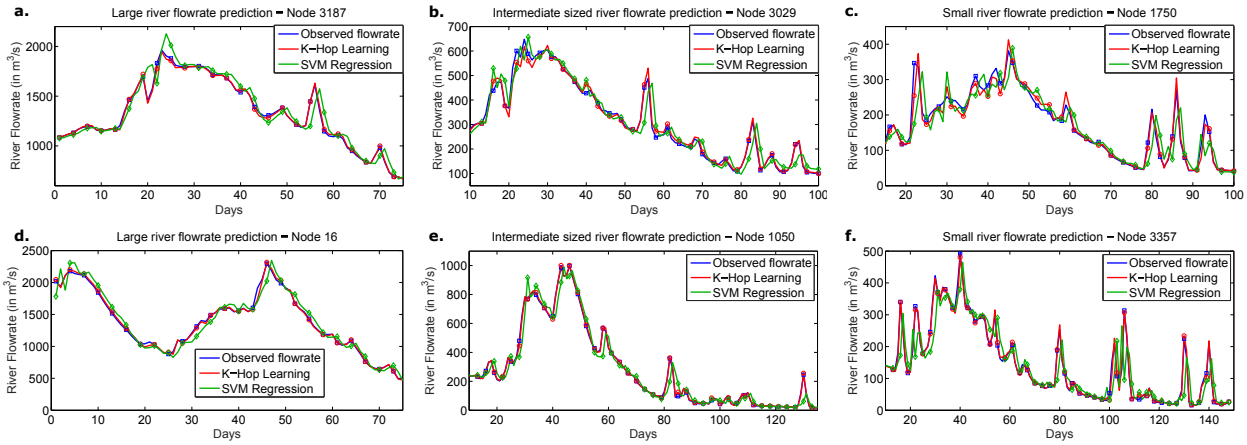


Figure 3.9: River flowrate prediction: (a,d) Large rivers (average flowrates $>$ 1000 m^3/s), (b,e) Intermediate sized rivers (average flowrates $\approx$ 400 m^3/s), and (c,f) Small rivers (average flowrates $\approx$ 150 m^3/s). Predictions based on the proposed K-Hop feature learning outperform the best timeseries-based model $-$ SVM-R.

Fig. 3.9 shows the flowrate prediction results for six different locations (two large (a,d), two intermediate (b,e), and two small (c,f) rivers) in Ohio River Basin. Each figure shows: (*i*) observed flowrate (blue), (*ii*) predicted flowrate using K-Hop features (red), and (*iii*) predicted flowrate using timeseries-based SVM-R (green) as it performs the best among prior models. Clearly, the prediction based on our proposed K-Hop Learning outperforms the timeseries-based SVM-R and, therefore, all other methods for all six rivers (see Fig. 3.9(a-f)).

Further, our model predicts the river flowrate *peaks* much more accurately than timeseries-based SVM-R. The peaks predicted by SVM-R and other models are generally one or two days after they actually occur. Our K-Hop Learning method, however, predicts these peaks on the correct day. Accurate prediction of these river flow peaks is critical for flood planning and run-of-river (ROR) hydropower management and, therefore, K-Hop Learning can play a fundamental role in this problem space.

For instance, consider 24-hours ahead flowrate prediction for a very large river (flowrates $\sim 10^4$ m^3/s) at node 1297. Fig. 3.10(a) illustrates the prediction results with K-Hop learning and SVM-R. The inset in Fig. 3.10(a) shows a zoomed-in portion of the results (between 112^{th}-116^{th} days) which reveals that the flowrates predicted by K-Hop learning (red) are very close to the observed flowrates (blue), while SVM-R (green) still predicts the peaks and troughs one day late. This can have a significant impact on run-of-river hydropower management. For instance, consider the

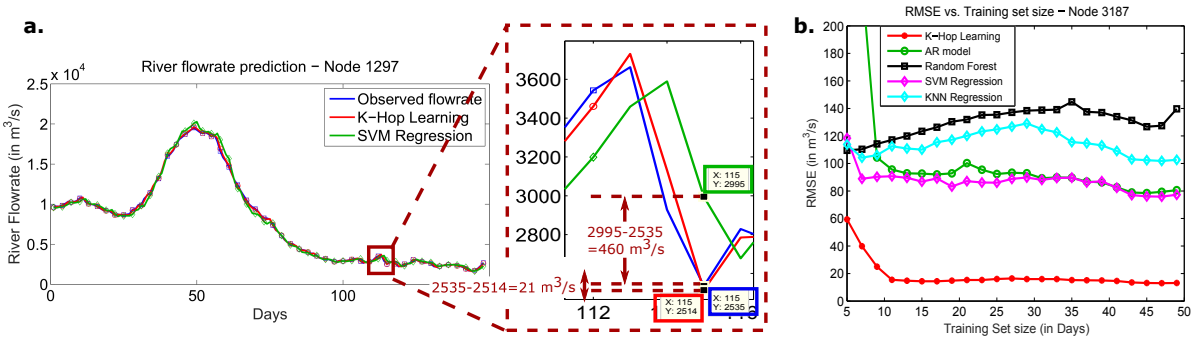


Figure 3.10: (a) River flowrate prediction for a very large river with flowrates of the order of 10^4 m^3/s. Inset: Observed and predicted flowrates zoomed-in between 112^{th}-116^{th} days. Clearly, the trough predicted on 115^{th} day by K-Hop Learning is much more accurate than the one predicted by SVM-R (with error of $21m^3/s$ for K-Hop Learning *vs.* $460m^3/s$ for SVM-R). (b) K-Hop Learning outperforms all other models for varying training set size for Ohio River at node 3187.

observed and predicted flowrates for the 115^{th} day (see $\{X, Y\}$ measurements shown as blue, red and green rectangles in Fig. 3.10(a) inset). Clearly, SVM-R overestimates the flowrate by 460 m^3/s while K-Hop Learning predicts only 21 m^3/s below the true value. As a result, if we were to depend on SVM-R for 1-day ahead predictions, we would not be able to prepare for this sudden hydropower deficit observed on 115^{th} day. Hence, K-Hop Learning can enable a significantly more effective ROR hydropower management.

Table 3.3: River Flowrate Prediction RMSE

Node & Size*	RMSE (in m^3/s)					% Gain
	AR	RF	SVM-R	KNN-R	K-Hop	
1750-S	33.28	39.69	33.27	39.49	**14.09**	57.64
3357-S	55.44	60.18	50.71	56.59	**9.00**	82.25
3029-M	57.65	61.21	48.20	60.86	**14.78**	69.33
1050-M	55.16	85.21	51.85	76.36	**8.17**	84.21
3187-L	95.80	132.88	85.61	117.9	**14.95**	82.53
16-L	88.71	180.24	86.48	143.70	**32.71**	62.17
1297-VL	341.95	1335.79	373.65	929.26	**86.16**	74.80**

*S− Small, M− Medium or Intermediate, L− Large, VL− Very Large River.

** % Gain reported with respect to AR since AR performs the best.

Next, Table 3.3 reports the Root Mean Square Error (RMSE) results for all flowrate prediction

experiments described above. The table clearly demonstrates that K-Hop Learning *significantly* reduces the RMSE for rivers *of all sizes*, with a net reduction of $57 - 84\%$ (*e.g.*, still 75% for very large river), *i.e.*, our model's performance does not degrade with increasing river sizes unlike all other models.

Finally, in Fig. 3.10(b), we analyze how the performance of various models varies with different training set sizes. As evident, among the timeseries-based machine learning models, SVM-R performs the best (magenta line in Fig. 3.10(b)). However, K-Hop Learning not only significantly reduces the RMSE but is also robust to varying training set sizes. Moreover, here we have compared a linear model using K-Hop features with advanced machine learning models using timeseries data at a given node, thereby showing the power of K-Hop feature learning and network-based features.

3.3 Summary

In this chapter, we have shown that certain machine learning problems such as spatiotemporal timeseries prediction are characterized by complex network dynamics. We have demonstrated that machine learning models must account for this underlying network dynamics in order to achieve higher accuracy.

To this end, we have first addressed a complex parameter learning problem over networks with known but dynamically changing structure. Specifically, our framework falls under non-stationary Bayesian learning. We have proposed a new model to learn parameters on such rapidly changing networks which can then be used for the underlying classification/prediction problems. As a concrete case study for this kind of learning scenarios, we have considered a computational sustainability task, *i.e.*, to predict short-term solar energy and to quantify solar energy interdependence across a large river basin. For this case study, our proposed model achieves 8-18% RMSE for one-hour cloud fraction predictions and outperforms standard models. Also, the spatiotemporal solar energy interdependence is explicitly quantified using the learned parameters.

Next, we have proposed a correlations-based network inference technique, and a new method called K-Hop Learning for feature extraction which improves the performance of machine learning models. Towards this direction, we have addressed another case study in the computational

sustainability domain: River network inference and highly accurate, short-term river flowrate prediction by exploiting network science-based features. Our proposed network-based feature extraction has resulted in significant improvements (57%-82%) in short-term flowrate predictions over the traditional models.

So far, we have demonstrated the importance of network science for spatiotemporal timeseries prediction and feature extraction, a traditional machine learning problem. However, for the river problem, our feature extraction method was largely application-specific. In the next chapter, we will demonstrate that network science concepts can also be used for automatic feature learning, *i.e.*, representation learning, for many general problems.

Chapter 4

Representation Learning for High-Dimensional and Small-Sample Problems

In the last chapter, we demonstrated how to use network science for spatiotemporal timeseries prediction and how network-based, application-specific feature extraction can result in significantly improved machine learning models. The feature extraction in the last chapter was indeed hand-tailored for a single application. However, a major goal in Artificial Intelligence (AI) and Machine Learning (ML) is not to engineer these features manually but rather to enable machines to learn them *automatically* for *any general problem*. This brings us to the domain of *representation learning* in which the low-dimensional features are learned automatically from the data.

Conventionally, representation learning has relied on *Big Data* to automatically learn features for any application [35, 58]. However, representation learning for small-sample datasets poses significant challenges due to curse of dimensionality [106]. Therefore, in this chapter, we first demonstrate how network science can be used for representation learning on many general, small-sample problems. We also show that deep learning problems such as model compression can suffer from lack of data. To alleviate this problem, we propose a new technique for model compression in absence of data.

4.1 Towards Representation Learning for Small Data

Due to curse of dimensionality [106], learning low-dimensional features from high-dimensional raw data is particularly challenging when the number of samples is small. Also, as seen in the last chapter, many machine learning problems have an underlying network which governs the

dynamics of the observed data. However, existing dimensionality reduction methods do not account for complex network characteristics (*e.g.*, community structure [86]) hidden within the raw data. Therefore, in this chapter, we propose a new community-based dimensionality reduction framework called *FeatureNet* which targets precisely the small-sample problems for learning low-dimensional features in many general problems such as digit recognition, Natural Language Processing (NLP), cancer prediction, computational sustainability, *etc*. Our proposed framework first creates a network from raw data to explicitly reveal complex network characteristics and then exploits network representation learning techniques to learn low-dimensional embedding.

The small-sample problems are not limited to merely the dimensionality reduction domain, and other well-known representation learning problems such as deep network model compression can also suffer from lack of data. Specifically, existing model compression techniques rely on access to the original or some alternate dataset [43, 51, 63, 132, 135]. However, for many applications, the original data may not be available (due to privacy or regulatory concerns, *e.g.*, speech data, medical images/data, *etc.*). Therefore, towards the end of this chapter, we address the model compression problem when no real data is available; we call this a *data-independent model compression* problem. To this end, we propose *Dream Distillation* framework, which first generates synthetic images from a small amount of metadata, and then uses them for model compression. However, we first explain the network science-based dimensionality reduction framework in the section given below.

4.2 Community-Based Dimensionality Reduction

As theoretically established in [30, 52, 54, 99], to obtain good classification performance in high-dimensional spaces, the number of samples must be very large (*e.g.*, $\sim 10^5$ samples). Hence, for problems with low number of samples (say, 100-1500 samples), extracting a set of useful features from the high-dimensional data is a very challenging task [131]. Towards this, many excellent dimensionality reduction techniques have been proposed in literature. For instance, there exist linear techniques such as Principal Component Analysis (PCA), Linear Discriminant Analysis (LDA), or non-linear neighborhood graph-based techniques such as Isomap, Stochastic Neighbor Embedding (SNE), t-SNE, among many others [1, 42, 44, 69, 118]. The neighborhood graph-based techniques largely rely on nearest-neighbor approach, where the size of the neighborhood is fixed

for all samples.

In real world, networks constructed from raw data often possess complex characteristics such as *communities* (*i.e.*, groups of tightly connected nodes) and *structural equivalence* (*i.e.*, nodes with similar roles in network, *e.g.*, hubs) [13, 14, 36, 112]. Therefore, such network characteristics must be accounted for while computing network neighborhoods for dimensionality reduction as they can lead to more accurate feature learning. Specifically, the neighborhood of a given node must depend on the community it belongs to. By contrast, prior techniques like Isomap [118] assume a rigid (fixed) neighborhood for all nodes in the network. Similarly, graph-based methods such as SNE [42]/t-SNE [69] use a fixed parameter called perplexity which measures effective number of neighbors. Hence, complex communities hidden within the raw data have *not* been explicitly taken into account in prior methods.

Recently, representation learning has been proposed in the context of learning features on networks while accounting for community structure, *e.g.*, node2vec [35], DeepWalk [96], community preserving embedding [121], LINE [116], *etc*. We refer to this problem space as "*Representation Learning on Networks*" throughout the chapter. However, the networks considered in this prior art do *not* come from high-dimensional raw data, but rather from social networks (*e.g.*, blogs, Youtube, Flickr), authorship networks or Wikipedia webpage networks. Hence, the prior representation learning on networks research does *not* directly address the problem of dimensionality reduction. However, by capturing communities and structural equivalence, ideas from "representation learning on networks" domain can have significant implications for dimensionality reduction. Therefore, in this chapter, we address the following **key questions**: *Can representation learning on networks have implications for dimensionality reduction by leveraging hidden communities in raw data? If so, how can we best construct a network from high-dimensional data to optimally capture its latent communities for dimensionality reduction?*

To answer these questions, we propose *FeatureNet*, a new community-based dimensionality reduction framework. We achieve this by contributing a new method that constructs a network directly from the raw data while explicitly revealing its hidden communities; this enables us to employ network representation learning ideas to learn *low-dimensional* community- and structural equivalence-based features from this network, thereby reducing the dimensions of the dataset. We demonstrate the effectiveness of our proposed framework on several small-sample problems

from diverse applications like handwritten digit recognition, biology, physical sciences, NLP, and computational sustainability (sizes mostly between 100 and 1500 samples).

4.3 Related Work

In this section, we discuss related work for FeatureNet. We first analyze the existing approaches from two perspectives: (*i*) Network Representation Learning techniques, and (*ii*) Dimensionality reduction methods. We then emphasize the key differences between our work and the prior research.

4.3.1 Network Representation Learning Perspective

Recently, representation learning on networks has focused on community-based feature learning. Our proposed FeatureNet differs in the following fundamental ways: (*i*) Prior techniques such as node2vec, DeepWalk, community-preserving embedding, LINE, *etc.* explicitly require a network as an input [35, 96, 116, 121]. (*ii*) The networks considered in this prior art are based on social networks (*e.g.*, Youtube, Flickr, blogs), or Wikipedia webpage networks which do *not* come from high-dimensional raw data. In contrast, we start with high-dimensional (raw) data that does not exist in such predefined network forms and learn the network structure directly from the raw data to explicitly reveal its latent communities. Hence, given high-dimensional raw data but no predefined network structure, none of these prior techniques can be used for dimensionality reduction.

For example, consider Arcene, a high-dimensional cancer benchmark dataset, where each sample comes from a patient and features specify the abundance of certain proteins[1]. This dataset does not have a predefined network structure like in social networks. To perform dimensionality reduction on such a dataset, methods like node2vec, LINE, *etc.* are *not* a natural choice simply because the initial dataset does not exist in a network form. This is where our key contribution lies – in representing *any kind of dataset* as a network which reveals hidden communities in raw data; we then use this network for dimensionality reduction using ideas from community-based feature learning.

To summarize, prior work in representation learning on networks constrains the community-based feature learning only to social networks or Wikipedia networks and performs only the

[1] https://archive.ics.uci.edu/ml/datasets/Arcene

network-based classification tasks (*e.g.*, classify interests of a blogger based on communities/homophily in a blog network). Our work, however, truly generalizes this "representation learning on networks" problem space to *any* classification problem which has high-dimensional data and does *not* restrict it only to network classification tasks.

4.3.2 Dimensionality Reduction Perspective

Neighborhood graphs created from initial dataset have been common in dimensionality reduction, *e.g.*, K-nearest neighbor graphs in Isomap [118], or fixed perplexity in t-SNE [69]. However, to the best of our knowledge, existing dimensionality reduction methods do *not* learn community-based features. Further, unlike prior methods, our proposed FeatureNet does not define a rigid neighborhood for each sample. Rather, the network neighborhood is *automatically* determined by the community structure. This way, communities hidden within the raw data can be exploited for dimensionality reduction. Of note, our newly proposed network construction approach is rooted in the field of network science, where correlation-based networks have been used for knowledge discovery [36, 112].

This completes the related work for FeatureNet. We next explain our proposed approach.

4.4 Proposed FeatureNet

Given a classification problem $\{X, y\}$, let $X \in \mathbb{R}^{n \times p}$ denote the original dataset with n samples and p features, while $y \in \mathbb{R}^{n \times 1}$ denotes the labels. Also, let $x^{(i)} \in \mathbb{R}^{p \times 1}$ be the i-th sample in X. Then, dimensionality reduction is a function $f : \mathbb{R}^{n \times p} \rightarrow \mathbb{R}^{n \times d}$, where d is the number of features in the reduced space ($d << p$).

Let $\mathcal{X}$ be the low-dimensional mapping of X, and $\mathbb{x}^{(i)} \in \mathbb{R}^{d \times 1}$ be the i-th sample of $\mathcal{X}$ (*i.e.*, the reduced representation of initial $x^{(i)}$). Then, the problem is to find $\mathcal{X}$ which accounts for the latent community structure and structural equivalences hidden within the raw data. Hence, unlike established techniques such as Isomap, the network neighborhood for each sample in our approach is *not* rigid, but rather takes communities and structural equivalences into account. To find such a mapping, therefore, we maximize the probability of observing a certain neighborhood $\mathcal{N}$ of sample $x^{(i)}$, conditional on its low-dimensional representation, as well as on its latent communities

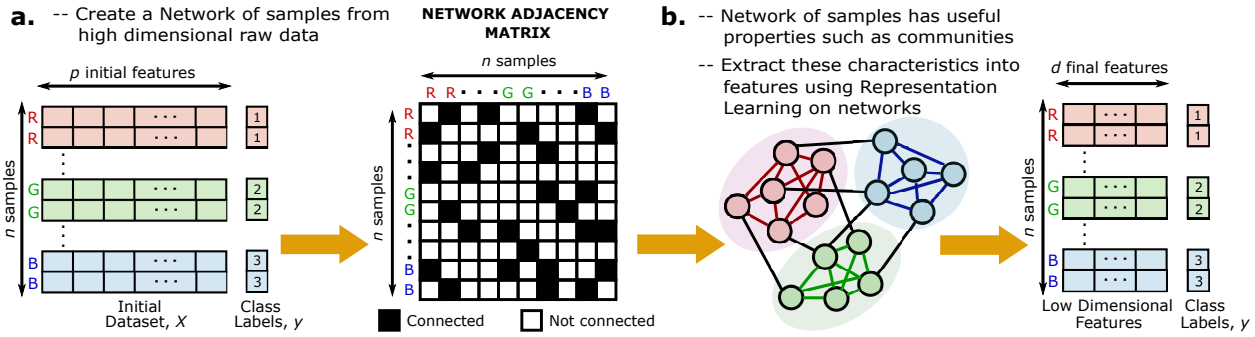


Figure 4.1: Complete flow of FeatureNet: (a) First, construct a network of samples using the proposed correlations-based method to explicitly reveal hidden communities in raw data (Section 4.3.1). (b) Next, use representation learning on this network to find the community-based low-dimensional features (Section 4.3.2).

and structural equivalences:

$$\max_{\mathcal{X}=\{\mathbb{x}^{(i)}|i=1:n\}} \sum_{x^{(i)}\in X} \log Pr(\mathcal{N}(x^{(i)})|\mathbb{x}^{(i)}, \mathcal{C}(x^{(i)}), \mathcal{S}(x^{(i)})) \tag{4.1}$$

where, $\mathcal{C}(x^{(i)})$ and $\mathcal{S}(x^{(i)})$ are latent variables containing information about communities and structural equivalence of sample $x^{(i)}$ hidden within the raw data.

To solve problem (4.1), we propose FeatureNet which: (*i*) transforms the raw data into a network space to *explicitly* reveal data's inherent communities, and (*ii*) performs representation learning on this network. Both of these steps are summarized in Fig. 5.4. Next, we present our proposed network construction technique to reveal hidden communities naturally.

4.4.1 Network Construction using Raw Data: Proposed K-τ Method

To construct a network directly from raw data, we again follow a two step process: (*i*) construct a correlation-based network (τ-step), and (*ii*) improve the density of network communities (K-step). For the τ-step, we transform the initial high-dimensional data into a correlation-based network of samples (*i.e.*, each sample now becomes a node). This first step is a mapping $l : \mathbb{R}^{n\times p} \rightarrow \mathbb{R}^{n\times n}$, which yields $\mathcal{G} = l(X)$. Here, $\mathcal{G} \in \mathbb{R}^{n\times n}$ is the adjacency matrix of the network of samples:

$$\mathcal{G}^{ij} = \begin{cases} c(x^{(i)}, x^{(j)}) & \text{if } c(x^{(i)}, x^{(j)}) \geq \tau \\ 0 & \text{if } c(x^{(i)}, x^{(j)}) < \tau \text{ or } i = j \end{cases} \tag{4.2}$$

where, $c(\cdot, \cdot)$ is a correlation function based on which the links are created, and τ is a threshold used on $c(\cdot, \cdot)$ to remove weakly correlated links from the network.

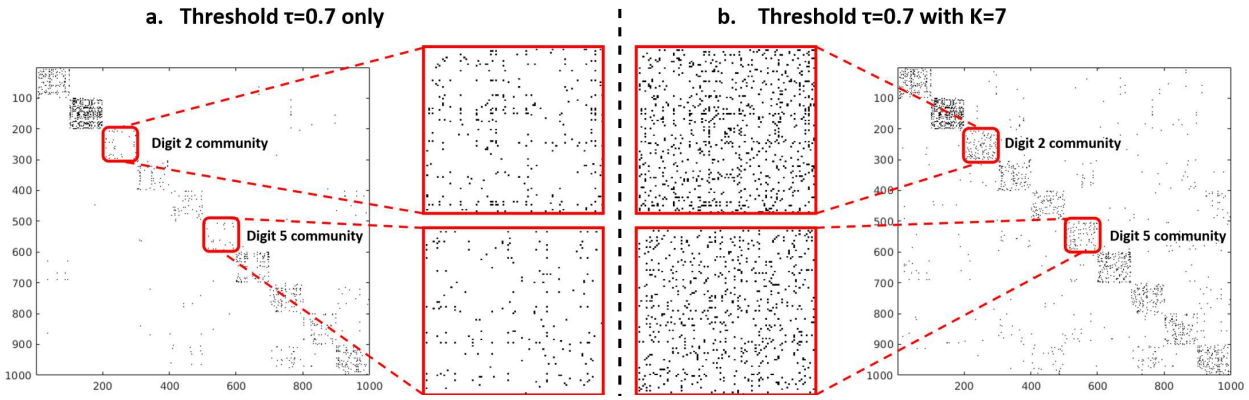


Figure 4.2: Adjacency Matrix for MNIST dataset network. (a) Threshold $\tau = 0.7$ removes noise from the network and reveals a clear community structure (*i.e.*, the diagonal clusters). (b) Introducing a density parameter K fixes the problem of sparse communities without adding significant noise and yields reliable low-dimensional representations.

Threshold (τ-step): Setting a higher τ removes the noise from the network by encouraging connections only among samples of the same class (*intra-class links*) and not among samples of different classes (*inter-class links*). To further elaborate, we consider MNIST handwritten digit dataset where each sample has 784 features. Since our focus is on relatively small datasets, we randomly select 1000 samples (100 images for each digit 0-9) from the MNIST database.

Next, we create a correlation-based network of samples from this 1000×784 dataset using Eq. 4.2 and a threshold $\tau = 0.7$. Fig. 4.2(a) illustrates the adjacency matrix of this network. Clearly, the ten diagonal clusters in Fig. 4.2(a) represent the intra-class links, thus revealing the hidden community structure of each digit. Moreover, using a high threshold of 0.7, most of the *noisy links* in the network (*i.e.*, the inter-class links) are removed. However, a too high threshold can also result in some samples getting completely disconnected from the network and very sparse communities (see digits 2 and 5 in Fig. 4.2(a) zoomed inset). To overcome this problem, we introduce a network density parameter, K.

Network Density (K-step): To connect the disconnected nodes and increase the density of network communities, we next connect each sample to its corresponding K highest correlated samples; *i.e.*, after the thresholding step, if a sample $x^{(i)}$ has less than K links, we connect it to samples $x^{(j)}$'s until it has K links. Here, samples $x^{(j)}$'s are selected based on the K-highest correlations. This step is a variant of the K-nearest neighbor approach to handle correlations rather than euclidean distances (*i.e.*, instead of K neighbors with minimum distances, we use K neighbors with

maximum correlations). As shown in Fig. 4.2(b), introducing a network density parameter of $K = 7$, (*i*) connects all disconnected nodes, and (*ii*) increases the density of diagonal clusters significantly without too much additional noise (see the zoomed-in inset of Fig. 4.2(b)). Hence, the threshold and density steps yield a K-τ method-based network of samples, $\mathcal{G}$.

To summarize, our proposed approach creates a network from raw data using two parameters: Threshold τ and Density K, which provide a tradeoff between the noise and the density of communities. Best K and τ can be selected via cross-validation and a simple grid search which generates an optimal neighborhood for each sample. Hence, in our approach, the neighborhood of each sample is not rigid, rather is determined *automatically* by its community structure. Explicitly revealing these hidden communities in raw data, therefore, enables the use of community-based representation learning methods in dimensionality reduction.

4.4.2 Community-Based Representation Learning

The network of samples, $\mathcal{G}$, often possesses many characteristics such as communities and structural equivalence. Once in the network space, therefore, the problem (4.1) reduces to:

$$\max_{\mathcal{X}=\{\mathbb{x}^{(i)}|i=1:n\}} \sum_{v\in\mathcal{V}(\mathcal{G})} \log Pr(\mathcal{N}_{\mathcal{G}}^{R}(v)|\mathbb{x}^{(i)}) \tag{4.3}$$

where, $\mathcal{V}(\mathcal{G})$ denotes the set of nodes in network $\mathcal{G}$, and sample $x^{(i)}$ is now represented by a node v in the network of samples. Finding the neighborhood of $x^{(i)}$, $\mathcal{N}(x^{(i)})$, now becomes the problem of finding the network neighborhood $\mathcal{N}_{\mathcal{G}}^{R}(v)$ of node v in $\mathcal{G}$. This network neighborhood can be found using a strategy R, which can account for the latent community structure $\mathcal{C}(x^{(i)})$ and structural equivalence $\mathcal{S}(x^{(i)})$. However, note that, this is precisely the skip-gram objective which the recent research on network representation learning aims to optimize [35, 77, 96]. Therefore, once the high-dimensional raw data is transformed into a network which *explicitly* reveals the community structure, the final low-dimensional representation can be learned using techniques such as node2vec [35]. Specifically, node2vec acts as a mapping $h : \mathbb{R}^{n\times n} \rightarrow \mathbb{R}^{n\times d}$, which yields $\mathcal{X} = h(\mathcal{G})$. The $n \times d$ matrix $\mathcal{X}$ contains the final low-dimensional features based on hidden communities in raw data. **Algorithm 2** shows these two stages of FeatureNet. Please refer to [35, 77] for more information on the classic word2vec skip-gram objective and node2vec search.

Algorithm 2 `FeatureNet` (X, τ, K)

Input: Raw Data $X \in \mathbb{R}^{n \times p}$, τ, K

Output: Low-dimensional representation $\mathcal{X} \in \mathbb{R}^{n \times d}$

——— ***K*-τ method to reveal communities** ———

$A \leftarrow \text{corr}(X')$ /* Pairwise correlations b/w samples */

$\mathcal{G} \leftarrow A$

$\mathcal{G}(\mathcal{G} < \tau) \leftarrow 0$ /* Remove links below threshold */

NODES $\leftarrow \mathcal{V}(\mathcal{G})$

for all i in NODES **do**

  $a^{(i)} \leftarrow A(i, :)$ /* i^{th} row of A */

  $g^{(i)} \leftarrow \mathcal{G}(i, :)$ /* i^{th} row of $\mathcal{G}$ */

  while $||g^{(i)}||_0 < K$ **do** /* # links for node $i < K$ */

    $\{m, j\} \leftarrow \max(a^{(i)})$

    $\mathcal{G}^{ij} \leftarrow m$

    $g^{(i)} \leftarrow \mathcal{G}(i, :)$

    $a^{ij} \leftarrow \phi$ /* Remove max element from $a^{(i)}$ */

  end while

end for

———————————————————————————

$\mathcal{X} \leftarrow h(\mathcal{G})$ /* Learn low-dimensional representation by solving problem (4.3). Use node2vec search strategy to explicitly account for communities revealed by $\mathcal{G}$ */

4.5 Experimental Setup and Results

In this section, we present our setup and results in detail.

4.5.1 Experimental Setup

We use *one-vs-rest* logistic regression with L2 regularization for multiclass classification. Node2vec parameters (return parameter p, and in-out parameter q) which control a trade-off between communities and structural equivalence, are optimized via a grid search on $p, q \in \{0.25, 0.75, 0.9, 1.5, 2, 4\}$.

Table 4.1: Characteristics of the datasets

Area	Dataset	X $(n \times p)$	#classes	Area	Dataset	X $(n \times p)$	#classes
Cancer (Bio.)	Arcene	100×10000	2	Comp. Sust.	CE-GDP 1980	1015×1095	17
Phys. Sci.	Musk1	476×166	2	Comp. Sust.	CE-GDP 1990	1420×1095	16
Hand Digits	MNIST	1000×784	10	Comp. Sust.	CE-GDP 2000	1448×1095	11
NLP	CNAE-9	1079×856	9	NLP	Reuters (subset)	5946×18933	65

Finally, FeatureNet parameters are also optimized using a grid search. τ is varied in steps of 0.05 from 0.6 to 0.95 while K is varied from 1 to 9. Best parameter values are given by 10-fold cross-validation (CV).

To show the effectiveness of FeatureNet for *any* general problem, we conduct experiments on eight datasets coming from five very different application areas as summarized in Table 4.1. Since our focus in this chapter is on dimensionality reduction for relatively small datasets, the sample sizes are mostly between 100 and 1500 in Table 4.1. Reuters subset data is used for analyzing scalability of our approach. Table 4.1 contains five benchmarks from UCI ML repository[2].

Table 4.1 also shows three datasets from the computational sustainability domain. Here, we make a twofold contribution: First, we propose the following new computational sustainability problem: "*Given multiple years of daily carbon emissions (CE) data across the world, can we correctly classify the GDP growth of different regions?*". Second, we contribute three new datasets to further benchmark dimensionality reduction problems. The datasets are compiled using the same carbon emissions database from Chapter 3 [2] and the world bank [119] data. The above CE-GDP problem is solved for three years – 1980, 1990, and 2000. For instance, for 1980, we use 3 years (1978-1980) of daily CE data ($p = 3 \times 365 = 1095$) and 1015 locations with available 1980 GDP growth data. GDP growth is categorized as follows: $<$-20% is class 0, -20% to -18% is class 1, -18% to -16% is class 2, *etc*.

Finally, we compare our approach against ten[3] well-established dimensionality reduction techniques: (1) PCA, (2) PPCA, (3) Polynomial Kernel PCA (KPCA - Poly.), (4) KPCA – Gaussian kernel, (5) Linear Discriminant Analysis (LDA), (6) SPE, (7) Deep Autoencoders, (8) SNE, (9) t-SNE, and (10) Isomap. We used a dimensionality reduction toolbox [70] for these techniques.

[2] `http://archive.ics.uci.edu/ml/index.php`

[3] For the ease of presentation, we will report only the top six performers.

Table 4.2: 10-fold CV F_1-Macro and F_1-Micro (Accuracy) for UCI benchmarks ($d = 16$): Best six prior methods shown.

Dimension	Arcene		Musk1		MNIST		Dimension	CNAE-9	
Reduction	F_1-Macro	F_1-Micro	F_1-Macro	F_1-Micro	F_1-Macro	F_1-Micro	Reduction	F_1-Macro	F_1-Micro
PCA	0.7596	0.76	0.7502	0.7563	0.8069	0.808	PCA	0.855	0.852
PPCA	0.7681	0.7699	0.7502	0.7563	0.805	0.806	PPCA	0.855	0.8535
KPCA (Poly.)	0.7493	0.75	0.7034	0.7037	0.6655	0.679	SNE	0.8728	0.8721
SPE	0.6124	0.62	0.7354	0.7436	0.7872	0.789	SPE	0.8527	0.8535
t-SNE	0.7312	0.7399	0.709	0.7142	0.8837	0.884	t-SNE	0.8068	0.81
Isomap	0.6386	0.64	0.726	0.7331	0.8338	0.8349	Isomap	0.8317	0.8331
FeatureNet	**0.8164**	**0.82**	**0.829**	**0.834**	**0.9128**	**0.913**	**FeatureNet**	**0.9229**	**0.923**

4.5.2 Results

Below we present the results to demonstrate the effectiveness of our approach.

UCI Machine Learning Repository Benchmarks

In our experiments, we reduce the dimensions of each dataset from initial p features to $d = 16$ features. We then conduct logistic regression on the reduced features and report its 10-fold CV F_1-Macro and F_1-Micro scores. Note that, F_1-Micro scores have the same interpretation as classification accuracy for multiclass classification problems. Table 4.2 presents these results for FeatureNet and the best six traditional techniques for all UCI datasets. As shown, our proposed FeatureNet *significantly* outperforms all six (and, hence, all ten!) prior dimensionality reduction methods.

For Arcene, FeatureNet achieves a F_1-Micro of 0.82 improving over the best performing PPCA method by 6.5%. Arcene is a challenging dataset because 3000 out of its 10000 features are 'probes' with no predictive power. This shows that our proposed FeatureNet is able to handle such noisy datasets. Next, for Musk1, we achieve an improvement of 10.27% in F_1-Micro scores over the best traditional methods – PCA and PPCA. Similarly, for MNIST, we observe an improvement of 3.28% in F_1-Micro over the best performing t-SNE technique. Recall that, we are only using 1000 samples for MNIST and not all 60,000 images for training. In fact, all datasets used in the present work are "relatively small" with samples mostly between 100 and 1500. This is why, deep learning-based techniques such as autoencoders do not perform very well and, as expected, overfit

the data.

Finally, for CNAE-9 dataset (NLP), we improve the F_1-Micro by 5.83% over the best performing SNE method. CNAE-9 is a business description text data for certain companies classified according to economic sectors. Each document is processed using standard NLP techniques (*e.g.*, stop-word removal, stemming, *etc.*) and is converted to a term frequency vector. This results in a very sparse dataset wherein 99.22% of the raw data is all zeros.

In summary, we demonstrate that FeatureNet can handle diverse dimensionality reduction problems and significantly outperforms prior methods. Similar improvements are observed for F_1-Macro scores.

Empirical Evaluation of FeatureNet in the K-τ Parameter Space

Fig. 4.3 shows the impact of varying density K (y-axis) and threshold τ (x-axis) for each UCI dataset. As shown, FeatureNet outperforms the traditional methods for several different combinations of K and τ (see orange/red portions in Fig. 4.3). For instance, especially for Musk1, MNIST and CNAE-9 (Fig. 4.3(b-d)), almost any combination of parameters gives a high classification accuracy. Indeed, for Arcene (Fig. 4.3(a)), we observe that only a few parameter combinations give high performance (*e.g.*, for $\tau = 0.95$ and all K values). A possible reason for FeatureNet's behavior for Arcene could be due to the additional noise in this dataset. We leave the theoretical analysis of stability of FeatureNet as a future work (*e.g.*, theoretically analyzing impact of addition of noise, *etc.*).

Why we achieve performance gain?

As mentioned before, the parameters τ and K control the tradeoff between noise in the network and density of communities. Consider the case $\tau = 0.85$ and varying K's for MNIST (*i.e.*, the rightmost column of Fig. 4.3(c)). For a high threshold of 0.85, the diagonal communities are even more sparse than those shown in Fig. 4.2 where the threshold used was only 0.7. Now, if we increase the density K, the F_1-Micro score increases from 0.873 for $K = 2$, to 0.904 for $K = 5$, to 0.902 for $K = 9$ (probably too much noise for $K = 9$ case which reduces the performance). This clearly demonstrates the tradeoff between the noise and density, and how it can affect the model performance. Therefore, our K-τ method successfully captures the best tradeoff and thus yields a

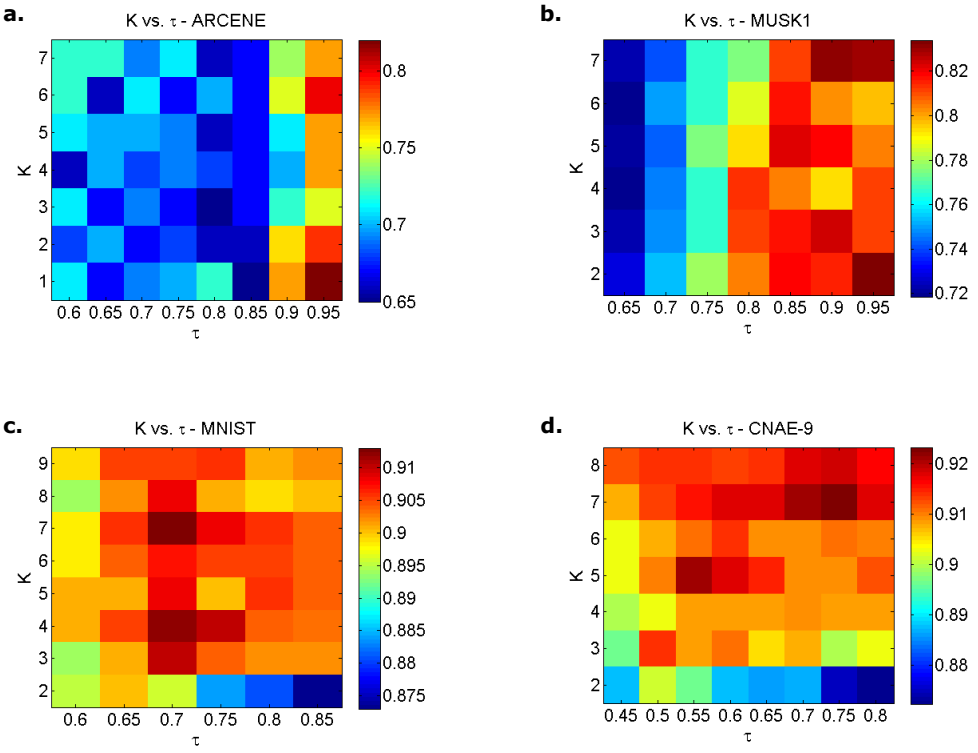


Figure 4.3: F_1-Micro for varying K and τ: (a) Arcene, (b) Musk1, (c) MNIST, and (d) CNAE-9. Red (blue) indicates higher (lower) accuracy. For all datasets, FeatureNet outperforms prior methods for many combinations of K and τ.

high classification accuracy.

Computational sustainability – A case study and new AI datasets

Table 4.3 shows F_1-Micro for the competitive methods across the three years for the CE-GDP datasets. As evident, FeatureNet significantly outperforms the best PPCA method by 40.13%, 22.51%, and 27.26% for 1980, 1990, and 2000, respectively. We also observed results like Fig. 4.3 for the CE-GDP datasets and also conducted experiments for varying the number of target dimensions from $d = 16$ to 32. Again, FeatureNet outperforms other techniques (the top three methods for CE-GDP, *i.e.*, PPCA, Isomap and PCA) for all d. Therefore, we propose that our CE-GDP datasets can also be used by the AI community to further benchmark dimensionality reduction problems.

Finally, K-τ network shown in Fig. 4.4(a) for CE-GDP 2000 dataset demonstrates that FeatureNet models the hidden communities with significantly different sizes very accurately, thus explaining the excellent performance of FeatureNet. Hence, fixed neighborhood size or perplexity

Table 4.3: 10-fold CV F_1-Micro (Accuracy) for CE-GDP Problems ($d = 16$): Best six prior methods are shown.

Years	PCA	PPCA	KPCA (Poly.)	KPCA (Gauss.)	t-SNE	Isomap	**FeatureNet**
1980	0.5123	0.6137	0.4108	0.4059	0.398	0.5241	**0.86**
1990	0.6098	0.6661	0.483	0.4845	0.5366	0.6443	**0.8161**
2000	0.625	0.6609	0.4544	0.4171	0.4682	0.6153	**0.8411**

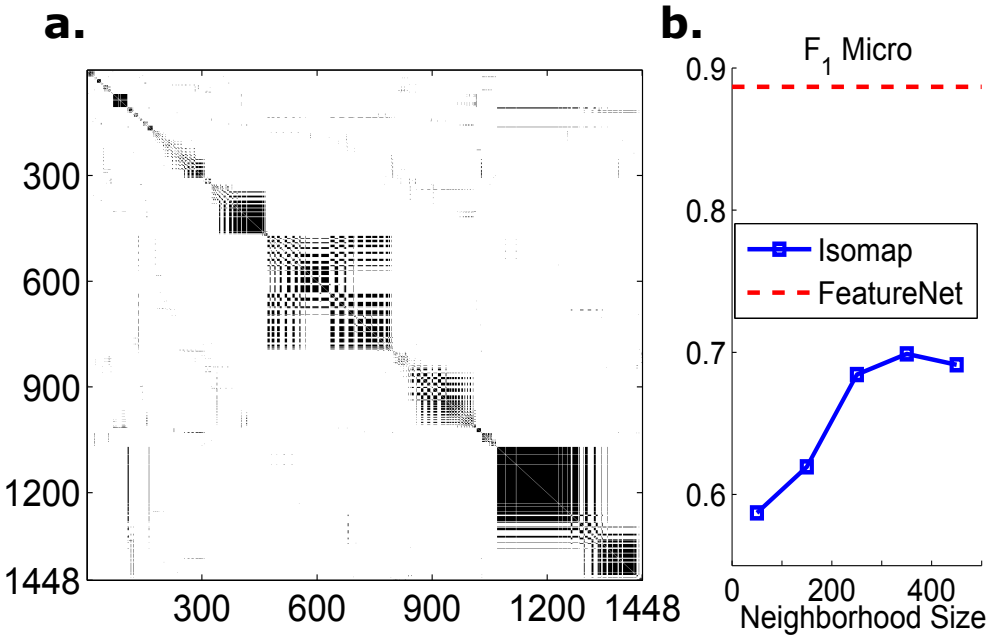


Figure 4.4: (a) K-τ network for CE-GDP 2000 shows communities with very different sizes that are accurately modeled by FeatureNet. (b) Varying fixed neighborhood size in Isomap and other methods cannot capture such variable size communities ($d = 32$).

methods (*e.g.*, Isomap, t-SNE) cannot capture such massive heterogeneity in raw data's community structure. To show this, we vary the *fixed* neighborhood size for Isomap in Fig. 4.4(b). As shown, FeatureNet is far superior to Isomap for all neighborhood sizes (FeatureNet's F_1-Micro approaches 0.9 for $d = 32$).

Note on scalability

To analyze the scalability of FeatureNet, we consider a subset of Reuters-21578 dataset in which documents with multiple category labels were removed. This yielded 8293 documents from 65 classes with 18933 distinct terms[4]. Of the total 8293 documents, we focus on the given training

[4]See details at `http://www.cad.zju.edu.cn/home/dengcai/Data/TextData.html`.

Table 4.4: Network characteristics

Case	Dataset, Parameters	#nodes	#links
(a)	MNIST, $\tau = 0.7$, $K = 7$	1000	6669
(b)	Reuters, $\tau = 0.7$, $K = 30$	5946	$719,080$
(c)	Reuters, $\tau = 0.5$, $K = 50$	5946	> 2.1 million

dataset of 5946 documents and report 10-fold CV classification F_1-Micro after reducing its dimensions from 18933 to 16. We compare FeatureNet with the top performers from above experiments – SPE, PCA, and t-SNE as these were amongst the only few techniques that were able to finish execution in a reasonable time (*e.g.*, about 2-4 hours) using reasonable computational resources (e.g., an 8-core Intel i7 desktop computer).

Table 4.4 shows the number of links for networks created using K-τ method for MNIST (Table 4.4(a)) and two different cases for Reuters dataset (Table 4.4(b,c)). As shown, for relatively small datasets, the number of links is not very big. However, for larger datasets, depending on the hidden community structure for that dataset, the number of links can increase rapidly. Indeed, the diagonal communities of the Reuters $\tau = 0.7$ and $K = 30$ case ($\approx 700,000$ links in Table 4.4(b)) are significantly more sparse than those of the $\tau = 0.5$ and $K = 50$ case (> 2.1 million links in Table 4.4(c)). Consequently, our proposed FeatureNet successfully reduced the dimensions and finished executing for the former but not for the latter. In terms of the classification accuracy, F_1-Micro for SPE, PCA, and t-SNE were 0.725, 0.82 and 0.823 respectively, whereas FeatureNet again significantly outperformed these techniques with a F_1-Micro score of 0.867 (5.34% improvement).

These results demonstrate that currently FeatureNet can indeed scale up to large datasets provided their networks contain several hundreds of thousands of links. However, optimizing FeatureNet to handle datasets which result in several million links, provides an excellent venue for future research.

So far, we have discussed the representation learning scenario for generic small-sample prob-

lems using our proposed network science-based dimensionality reduction framework. Other well-known representation learning techniques such as deep convolutional neural networks (CNNs) have traditionally been used for Big Data problems (e.g., image classification) [58]. However, the small-sample problems can still exist in deep learning space. For instance, model compression of deep networks, which usually relies on the original dataset, can suffer from lack of data. We next discuss this important problem in detail and propose a new technique to perform model compression in absence of data.

4.6 Data-Independent Model Compression

Complex deep neural networks with millions of parameters have achieved breakthrough results for many vision, and speech recognition applications. In the IoT-era, however, the edge devices are heavily hardware-constrained. Therefore, model compression of deep networks has now emerged as an important problem. Towards this end, many state-of-the-art model compression techniques such as pruning [132], quantization [68] and Knowledge Distillation (KD) [43, 135] have been proposed. Pruning aims at removing redundant or useless weights from deep networks, while quantization reduces the number of bits used to represent weights and activations. On the other hand, KD trains a significantly smaller student model to mimic the outputs of a large pretrained teacher model.

The existing model compression techniques above rely on access to the original training data or some unlabeled dataset. Hence, most model compression research implicitly assumes access to the original dataset. However, for many applications, the original data may not be available due to privacy or regulatory reasons (*e.g.*, private medical images, speech data, *etc*.). Consequently, the industries deploying large deep learning models at the edge must compress them *without* access to any original, private, or alternate datasets[5] [67]. Therefore, in this chapter, we address the following **key question**: *How can we perform model compression when the original or unlabeled data for an application is not available?* We call this problem as *data-independent model compression*.

To answer this question, we propose a new framework called *Dream Distillation* [16]. Our

[5]Collecting alternate datasets for model compression may not always be possible, or can be very expensive/time-consuming and, hence, infeasible.

framework uses ideas from the field of deep network interpretability [94] to distill the relevant knowledge from the teacher to the student, in absence of access to the original training data. Specifically, our approach consists of two steps: (*i*) We first exploit a small amount of metadata and the pretrained teacher model to generate a dataset of synthetic images, and (*ii*) We then use these synthetic images for KD. To this end, our **key goal** is to generate synthetic data while preserving the features from the original dataset such that the teacher can transfer the knowledge about these features to the student. This effective transfer of knowledge via synthetic data can make the student model learn characteristics about original classification problem without actually training on any real data! By allowing users to deploy a model on IoT-devices without access to the private third-party datasets, data-independent model compression techniques can truly accelerate the adoption of AI on edge devices.

4.6.1 Preliminaries

We first briefly review Knowledge Distillation (KD) and feature visualization, both of which are necessary for Dream Distillation.

Knowledge Distillation

KD refers to the teacher-student paradigm, where the teacher model is a large deep network we want to compress [43, 135]. In KD, we train a significantly smaller student neural network to mimic this large teacher model (see Fig. 4.5). KD has also been shown to work with unlabeled datasets [59]. Of note, since the term "model compression" usually refers to pruning and quantization, we assume KD to be a part of model compression, as it also leads to significantly compressed models. For more details on KD, please refer to Section 2.3.

Feature Visualization

Feature visualization domain aims to visualize and understand which patterns activate various neurons in deep networks [94]. Towards this, tools such as DeepDream [80] and Tensorflow Lucid [105] can generate an image that maximizes a given objective. For example, Tensorflow Lucid can be used to generate an image that maximally activates a given hidden unit (say, a neuron or a

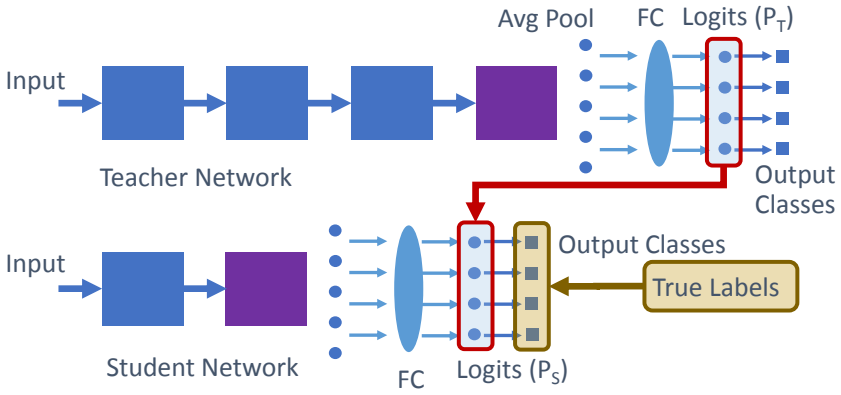


Figure 4.5: Knowledge Distillation (KD): A significantly smaller student model mimics the outputs of a large teacher network, thereby compressing the model without losing significant accuracy.

channel). These generated synthetic images are called as the *Dreams* of a neural network. Since our work is inspired by KD and feature visualization, we call our approach *Dream Distillation*.

Prior Art on Data-Independent Model Compression

Despite its significance, the literature on model compression in absence of real data is very sparse. A relevant prior work is [67] where the authors propose a Data-Free KD (DFKD) framework. However, there are major differences between DFKD and the present work:

1. DFKD requires significantly more metadata than our approach. Specifically, [67] argue that using metadata from only the final layer under-constrains the image generation problem, and results in very poor student accuracy. Consequently, DFKD assumes access to metadata at *all* layers. In contrast, Dream Distillation assumes that metadata is available only at one layer of the teacher network. Hence, in this chapter, we precisely demonstrate that metadata from a *single* layer is sufficient to achieve high student accuracy, something that DFKD failed to accomplish.
2. When using metadata from only one layer, DFKD achieves only 68-77% accuracy on MNIST dataset [67]; this means the accuracy of DFKD will be even lower for significantly more complex, *natural* image classification datasets like CIFAR-10. On the other hand, we demonstrate 81-88% accuracy on CIFAR-10 dataset without training on any real data.
3. DFKD also proposes a spectral method-based metadata for synthetic image generation.

However, both spectral methods and all-layer metadata can be computationally very expensive and do not scale for larger networks. Compared to these, we follow a clustering-based approach which helps generate diverse images while using significantly less computation.

Finally, [110] focus on data-free finetuning for pruning, and show the effectiveness of their approach for fully-connected layers. In comparison, our work is much more general, as we do not focus on just the finetuning of a compressed model, but rather on training a compressed student model from scratch.

4.6.2 Proposed Dream Distillation

We propose Dream Distillation to address the following research question: *In absence of the original training dataset (or any alternate unlabeled datasets), how can we perform model compression without compromising on accuracy?* Specifically, for KD with teacher model trained on CIFAR-10, datasets such as CIFAR-100 and tinyImagenet have been shown to be effective alternatives for distilling relevant knowledge [59]. However, since alternate datasets may not always be available, our focus here is to generate a synthetic dataset which can be just as effective at distilling knowledge as these alternate data. Hence, we assume that the alternate/original data is *not* available; rather, a small amount of metadata is given for model compression.

Metadata

We will use CIFAR-10 dataset throughout this chapter. To generate our metadata, we start with the *activation vectors* generated by passing 10% real CIFAR-10 images through the given pretrained teacher model. The activation vectors are simply the output of average-pool layer of the teacher model (*i.e.*, average output of final convolution layer; see Fig. 4.5). Then, we cluster these vectors via k-means clustering. Finally, we use the *cluster-centroids* and the orthogonal principal components (PCs) of clusters as our metadata. Fig. 4.6(a) illustrates our metadata for the airplane class for CIFAR-10 dataset and 2-D visualization[6] of activation vector clusters. By definition, centroids refer to mean activations of clusters. Hence, using centroids reduces the privacy-concerns since we do not use activations (or any other identifying information) directly from real images.

[6]Two-Dimensional Visualization via tSNE: `https://bit.ly/2FUmCzj`

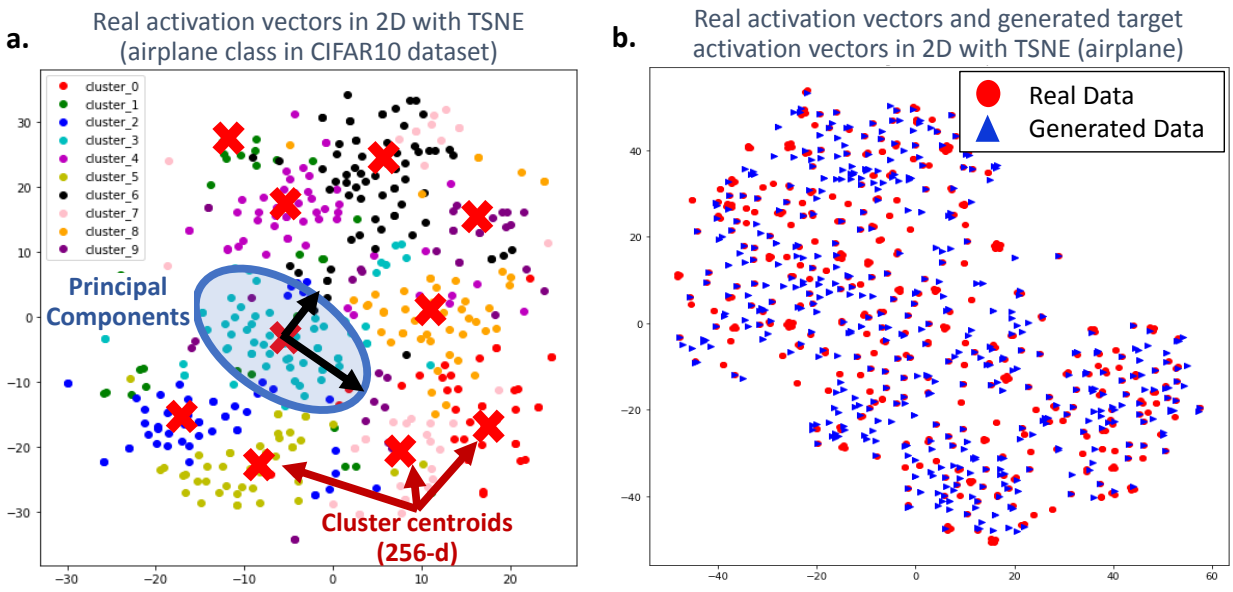


Figure 4.6: Dream Distillation: A data-independent knowledge distillation framework. (a) Metadata used by our method consists of cluster centroids and principal components, and (b) tSNE visualization [69] of real data activations and the generated target activations demonstrates that they belong to the same data distribution.

For WRN40-4 teacher model, our metadata merely amounts to 0.58MB which is about $100\times$ smaller than the size of even 10% real CIFAR-10 images (around 58MB). We next use this metadata (centroids and PCs) and teacher model to generate synthetic images. Of note, image generation techniques such as Generative Adversarial Networks (GANs) cannot be used for our problem since GANs also rely on availability of real data [33].

Dream Generation and Distillation

Let c_k be the centroid for cluster k, and $p_j^k, j \in \{1, \ldots, m\}$ denote its m PCs. We first create *objectives* from metadata, and then optimize them to generate images. Specifically, we add a small noise to the centroids in the direction of PCs to generate new *target activations*: $t_i = c_k + \sum_j \epsilon_j p_j^k, i \in \{1, \ldots, n\}$. Here, n is the number of images to be generated, and ϵ_j is Gaussian noise for j^{th} PC. To compute ϵ_j, explained variance of j^{th} PC is used as the variance in the Gaussian noise. Adding noise proportional to the explained variance of corresponding PCs makes our target activations mimic the real activations.

Therefore, by adding small noise to mean activations (*i.e.*, cluster-centroids), the target activa-

tions emulate the behavior of real samples at teacher's average-pool layer. In Fig. 4.6(b), we show a two-dimensional visualization of target activations (blue triangles) and real data activations (red circles) at the average pool layer of the teacher network. As evident, since the target activations and real data activations overlap, their distributions are quite similar. Hence, the generated target activations can be reliably used for generating synthetic images. As a result, to generate the images, we must find an image X_i whose average-pool activations are as close as possible to t_i. Therefore, we generate the synthetic images X_i as follows:

$$\min_{X_i} ||g(X_i) - t_i||_2^2 \quad i \in \{1, \ldots, n\} \tag{4.4}$$

where, the function g refers to the average-pool output of the teacher network. We used about $m = 50$ PCs per cluster and generated a total of $n = 50,000$ synthetic images for CIFAR-10 classes. To generate the synthetic images, we minimize the objective in (4.4) by using Adam optimizer with a learning rate of 0.05, $\beta_1 = 0.9$, and $\beta_2 = 0.999$. We optimize the image X_i for 500 iterations. Finally, these synthetic images are used to train the student via KD.

The main advantage of our clustering-based approach is that it enables more diversity among the generated synthetic images and, hence, achieves high accuracy. To summarize, the idea is to generate synthetic images, and then use them to distill knowledge about real data to the student.

4.6.3 Experimental Results

For the CIFAR-10 dataset, our teacher is a large Wide Resnet (WRN) [135] WRN40-4 (8.9M parameters, 95% accuracy) model, and our student model is WRN16-1 (100K parameters). Training the WRN16-1 student via Attention Transfer KD [135] on WRN40-4 teacher results in 91% accuracy and $89\times$ fewer parameters than the teacher.

Fig. 4.7(a) shows samples generated by our approach for different CIFAR-10 classes. As evident, for classes such as car, truck, deer, *etc.*, key features like distorted-animal-faces/wheels are visible. On the other hand, images from classes such as cat, frog, ship are hard to interpret. For instance, to generate cat samples, the teacher network generally creates a striped pattern which may not be the most distinguishing feature of a cat (although many cats do have a striped pattern!). Therefore, the generated images *do contain key features* learned by the teacher network for various classes (*e.g.*, stripes for cats, *etc.*) even though the images look far from real. Hence, these

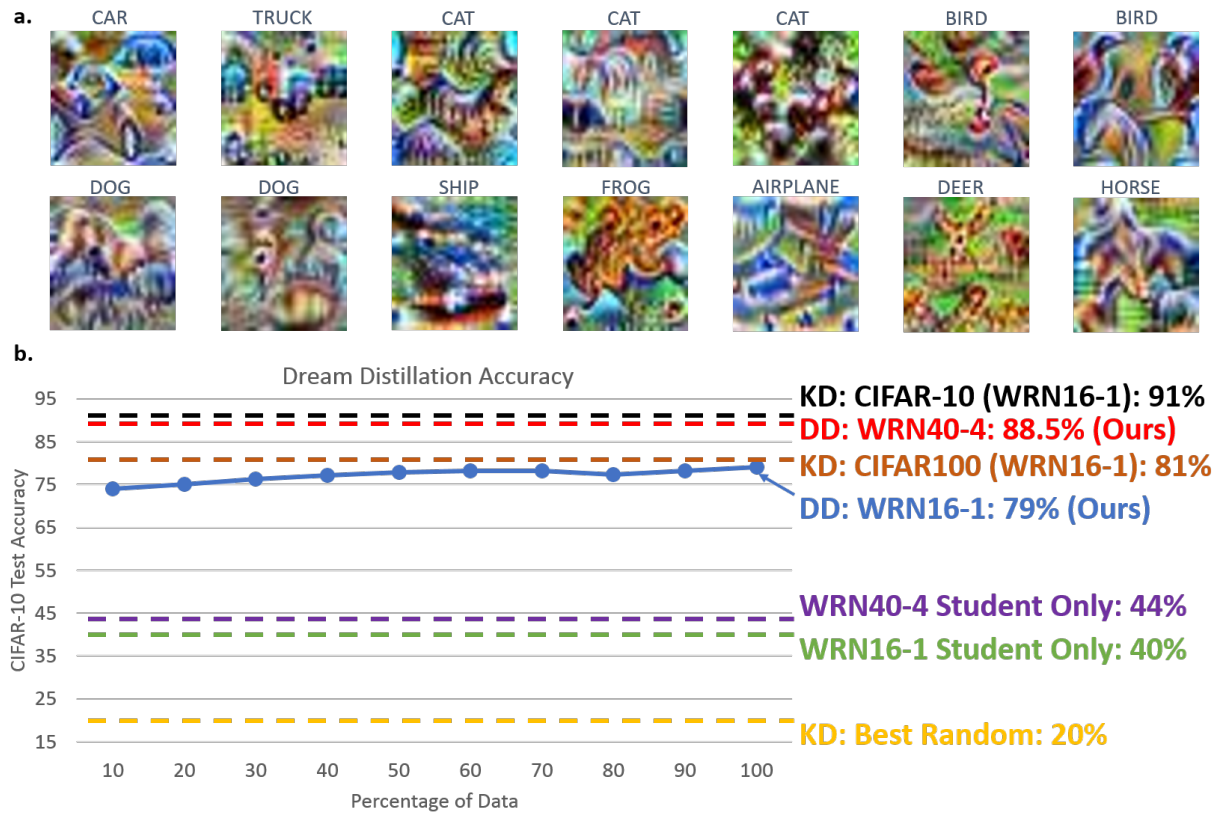


Figure 4.7: (a) Generated synthetic data for CIFAR-10 classes, and (b) Accuracy of student models trained on random, synthetic, alternate CIFAR-100, and real CIFAR-10 images.

synthetic images can transfer relevant knowledge about the real data.

Next, in Fig. 4.7(b), we compare student models trained via KD on four datasets: (*i*) random noise images, (*ii*) images generated via Dream Distillation, (*iii*) CIFAR-100 as the alternate data, and (*iv*) CIFAR-10 training set. The accuracy is reported for CIFAR-10 test set. The solid blue line shows how the accuracy of Dream Distillation varies with the number of synthetic images used for training (*e.g.*, 10% data means 5000 synthetic images since we generated total $50,000$ images). Fig. 4.7(b) demonstrates that the accuracy of Dream Distillation is comparable to that of CIFAR-100 (around 80%), both being around 10% lower accuracy than the student trained on real CIFAR-10 data. Further, we demonstrate that the WRN40-4 student model (which is the same as the teacher) trained via Dream Distillation achieves 88.5% accuracy on CIFAR-10 test set without training on *any* real data! Again, for metadata available only at one layer, the prior DFKD model achieves merely 68-77% accuracy even for MNIST dataset [67]. Hence, it would achieve much

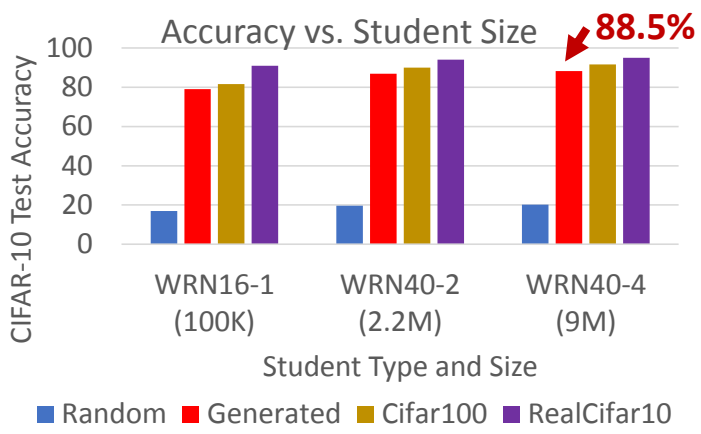


Figure 4.8: Varying the size of the compressed student model from 100K parameters to 2.2M parameters, all the way to 9M parameters (when teacher and student models are the same WRN40-4 architectures): Student models are trained using random data (blue), generated data (red), alternate real data: CIFAR100 (yellow), and real CIFAR-10 dataset (violet).

lower accuracy for CIFAR-10.

If the above generated sythetic images are used to train a model without the teacher, WRN40-4 model achieves only 44% accuracy, whereas the same model achieves $2\times$ better accuracy with the pretrained teacher model (88.5%). This shows that the generated images can very effectively transfer knowledge *from the teacher* to the student. Hence, the images generated via Dream Distillation can transfer significant knowledge about the real data without accessing any real or alternate datasets.

Finally, we vary the student model from WRN16-1 (100K parameters) to WRN40-2 (2.2M parameters) to WRN40-4 (9M parameters) and see how the accuracy changes as the model size increases. The results are shown in Fig. 4.8. Clearly, for the images generated via Dream Distillation, the accuracy of the compressed models gradually increases from 79% for 100K parameters to about 86% for WRN40-2 model, all the way to 88.5% for WRN40-4 model. In all three cases, the accuracy of Dream Distillation is comparable to that of alternate real data.

This completes Dream Distillation. Overall, we have demonstrated that data-independent model compression techniques can greatly increase the scale of deep learning at the edge since industries can quickly deploy compressed models without the need for proprietary datasets.

4.7 Summary

In this chapter, we have addressed representation learning for Small Data problems. We have first shown that network science concepts such as community structure can be exploited to learn useful low-dimensional features for many general problems. Specifically, such network characteristics are hidden within the raw data and can be used for dimensionality reduction. Next, we have shown that deep learning problems such as model compression can also suffer from lack of data.

To this end, we have first proposed FeatureNet, a new community-based dimensionality reduction framework for small sample problems. We have proposed a new technique to construct a network from any general raw data while revealing its hidden communities. Community-based low-dimensional features are then learned using a representation learning framework. We have demonstrated the effectiveness of FeatureNet across five very different application domains ranging from handwritten digit recognition, biology, physical science, NLP, to computational sustainability. We have further shown that FeatureNet significantly outperforms many well-known dimensionality reduction techniques such as PCA, PPCA, deep autoencoders, t-SNE and Isomap. This ultimately shows how network science ideas can have huge implications for dimensionality reduction.

Finally, we have proposed Dream Distillation, a new approach to address model compression in absence of real data. Towards this, we have used a small amount of metadata to generate synthetic images, and then used those images for model compression. Our experiments have shown that models trained via Dream Distillation can achieve up to 88.5% accuracy on the CIFAR-10 test set without ever training on any real data!

In the next chapter, as a natural extension of our current theme of model compression and network science for representation learning, we will explicitly address how network science can lead to new research directions in model compression.

Chapter 5

Network-of-Neural Networks: Memory- and Communication-Aware Model Compression

In the last chapter, we addressed *general* representation learning and correlations-based networks for small datasets for which deep learning is not suitable due to overfitting (*e.g.*, for many problems in biological systems, sustainability, *etc.*). We further showed how model compression problems can suffer from lack of data. However, we did not explicitly explore the advantages of network science for deep network model compression. As we know, for problems such as image classification, deep learning methods such as Convolutional Neural Networks (CNN) are used to accurately classify the image data. Such models, however, not only require enormous training datasets, but also often utilize millions of parameters to correctly work.

Motivated by this immense computational complexity, there is a fundamental need for highly efficient deep learning model architectures which can enable *faster and computationally inexpensive training, as well as inference*. Towards this end, many model compression techniques exist such as pruning [63, 132], quantization [51, 60], and Knowledge Distillation (KD) [43, 135]. Most of this prior art aims to compress the given deep networks in order to fit them on a *single* device. However, since Internet-of-Things (IoT) naturally implies a *network* of connected devices, an important question is *why restrict the intelligence to a single device, when a network of connected devices is available instead?* Therefore, in this chapter, we propose new directions in model compression for an efficient inference across a network of edge devices. Indeed, since a network is involved in our problem, memory and communication pose critical constraints. Hence, we demonstrate how network science can enable an effective memory- and communication-aware model compression.

5.1 Memory- and Communication-Aware Model Compression

Even though deep learning has gained significant importance, it is challenging to implement these models on resource-constrained devices. Such devices are commonly used in the ever-growing *Internet-of-Things* (IoT) domain. For instance, microcontrollers such as Arm Cortex-M which is used in many IoT applications such as Smart Healthcare, Keyword Spotting, *etc.*, has only 500KB available memory [138]. For such devices, prominent deep networks like AlexNet/Resnets that use up to 60M parameters are unsuitable. Therefore, there is a fundamental need for highly-efficient compressed deep learning models to enable *fast and computationally inexpensive inference* on the resource-constrained IoT-devices.

To this end, several deep learning model compression techniques exist in the literature. The most common approaches are pruning [39, 132], quantization [51, 60], and Knowledge Distillation (KD) [4, 43] and its variants such as Attention Transfer (AT) [135]. Since deep models can learn a large number of redundant or useless weights/channels, pruning aims to remove such parameters without sacrificing accuracy [39, 132]. Moreover, quantization reduces the number of bits required to represent weights and/or activations in deep networks [51, 60]. Also, KD-based approaches rely on teacher-student training, where a teacher model is a large deep network we want to compress. KD trains a significantly smaller student model with far fewer layers/width, to mimic the teacher network (see Fig. 5.1a). Further, most prior art on model compression refers to pruning and quantization. However, since KD can lead to significantly compressed models, we assume that KD is also a model compression technique throughout this chapter. Of note, the present work considers model compression and distributed inference for Convolutional Neural Networks (CNNs) used in image classification problems. However, distributed inference for other deep learning models such as Recurrent Neural Networks (RNNs) for speech/natural language applications can also be explored in future work.

The existing approaches mentioned above *cannot* be used for *extremely memory-constrained* IoT scenarios (*e.g.*, microcontrollers with a *total* memory of 500KB [113, 138]). Yet, a lot of smart home/cities applications can have several such connected but resource-limited sensors. To achieve higher accuracy, pruning- or KD-based student models often grow in size due to which such models cannot fit on an individual IoT-device and must be distributed across multiple devices; this

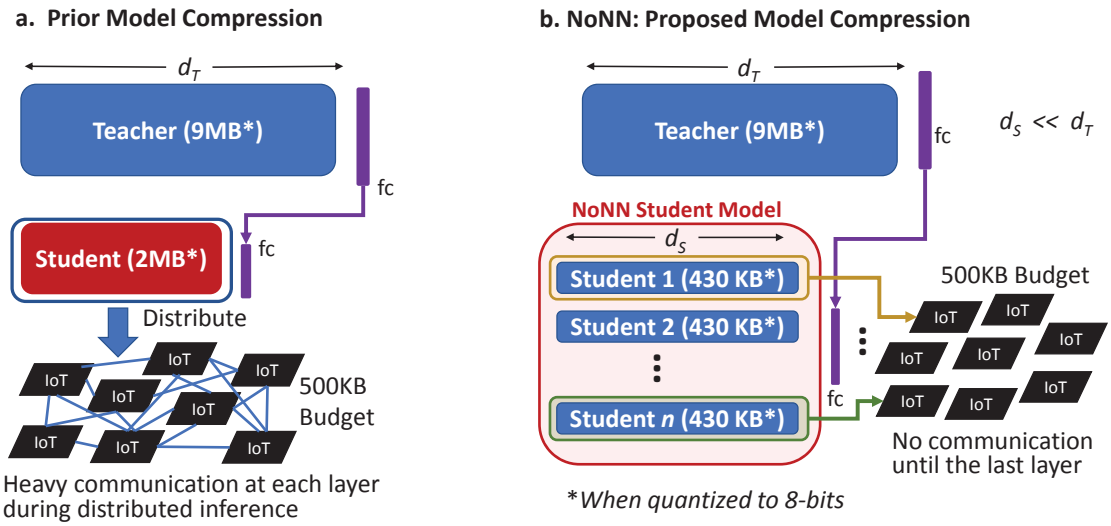


Figure 5.1: (a) Prior art: Distributing large student models that do not fit on a single memory-limited IoT device leads to heavy communication at each layer. (b) Proposed NoNN results in several disjoint students that can fit on individual IoT-devices: No communication until the very final layer.

distribution of computation on multiple devices generates significant overhead in communication. For example, Fig. 5.1a illustrates KD with a teacher model containing around 9 million parameters, and a large student model containing about 2 million parameters. Also, suppose that we are trying to deploy the student model (which will require 2MB memory after quantization) on IoT-devices with only 500KB memory budget. Distributing this student model on multiple such IoT-devices will incur heavy communication at each of its intermediate layers. Consequently, in addition to the computation cost, a major (and largely ignored) impediment for widespread deployment of deep learning on IoT is this *communication* cost during distributed inference, which occurs due to extremely limited memory in IoT environments.

Starting from the above ideas, we aim to answer the following **key question**: *How can we compress a given deep learning model into multiple separate modules that: (i) fit within the memory and performance budgets per device, (ii) minimize the communication latency when distributed across a network of edge devices, and (iii) achieve high accuracy?*

To address this question, we propose *Network of Neural Networks* (NoNN), a paradigm to derive low memory-, computation-, as well as communication-cost student architectures from a single (powerful) but much larger teacher model. As shown in Fig. 5.1b, a NoNN consists of multiple, *memory-limited* student networks which individually learn only a specific part of teacher's

function. Indeed, training an individual student to mimic only a part of teacher's function is a significantly simpler problem than mimicking its entire function. Since this highly-parallel architecture of NoNN allows us to effectively increase the overall model-size, we obtain higher accuracy without significantly increasing the communication costs among the disjoint students. These individual students can then be deployed on separate resource-constrained IoT-devices to perform the distributed inference (see Fig. 5.1b).

Finally, to distribute knowledge from a teacher to multiple students, we propose a new approach based on network science [85, 86]. Specifically, as our objective is to obtain individual student models below certain memory-constraints, network science allows for significant flexibility in terms of how the knowledge from the teacher is distributed across multiple students. For instance, with our proposed network-theoretic approach, we uniformly distribute knowledge from the teacher across the students to achieve high accuracy.

Overall, we make the following **key contributions**:

1. Since there has been limited research on performing efficient distibuted inference on highly memory-constrained IoT devices, we propose a new NoNN paradigm to compress a large teacher into a set of independent and highly-parallel student networks. In prior works, student models that cannot fit on a single device can lead to significant communication at all layers, whereas NoNN communicates only at the last layer. This makes NoNN an ideal candidate for distributed deep learning. To the best of our knowledge, we are the *first* to introduce a communication-aware model compression for distributed inference on IoT-devices.
2. We are also the first to formulate the model compression problem from a network science angle. By building a network of filter activation patterns, we exploit principles from network science such as community detection [86] to model how to split teacher's knowledge into multiple disjoint partitions which, in turn, results in disjoint and compressed student modules.
3. Extensive experiments on five well-known image classification tasks demonstrate that NoNN achieves significantly lower memory ($2.5\times$-$24\times$ reduction w.r.t. teacher) and computation ($2\times$-$15\times$ fewer FLOPS w.r.t. teacher), with minimal communication costs and similar accuracy as the teacher model. Also, NoNN achieves higher accuracy than prior art.

Table 5.1: Comparison to Prior Art

Area	Model Compression	Communication-Aware	Distributed Inference	Complements our work
Quantization [51, 60]	✓	×	×	✓
Pruning [38, 53, 63, 132]	✓	×	×	✓
Separable convolutions [31, 103, 137]	✓	×	×	✓
KD [4, 43, 135]	✓	×	×	×
SplitNet [55]	×	✓	✓	×
MoDNN, DeepThings [71, 140]	×	✓	✓	✓
Proposed NoNN	✓	✓	✓	✓

4. We further deploy the proposed models for CIFAR-10 dataset on Raspberry Pi, and more powerful Odroid devices to evaluate several scenarios with homogeneous and heterogeneous devices. We demonstrate $6.22\times$-$12.22\times$ gain in performance and $12.99\times$-$14.36\times$ gain in per node energy w.r.t. teacher. Moreover, for distributed inference on multiple edge devices, NoNN results in up to $33\times$ speedup in total latency w.r.t. a prior model compression baseline. We also show that the proposed approach is robust and achieves high accuracy even with a reduced number of devices.

In the next section, we discuss the existing literature and show how our problem is different from the prior art. We also review key concepts that will be used throughout this chapter.

5.2 Related Work

We first discuss prior art on model compression and distributed inference. We then briefly review Knowledge Distillation (KD), and network science later in this section.

5.2.1 Model Compression/Distributed Inference

Table 5.1 summarizes the key differences between the present work and the existing literature from model compression and distributed inference. Specifically, existing approaches for model compression such as quantization [51, 60], pruning [38, 63, 132], KD [4, 43, 135], and separable convolutions [31, 103, 137] are *not* communication-aware and do *not* address the distributed

inference problem. For instance, although pruning is effective at reducing parameters/FLOPS, it does *not* result in highly-parallel model architectures. Clearly, pruned models that cannot fit on a single IoT-device will need to be distributed across multiple devices which will incur heavy communication costs (see Section 5.3).

On the other hand, existing distributed inference techniques such as SplitNet [55], MoDNN [71], and DeepThings [140] do not exploit *ideas* from model compression (*e.g.*, pruning/distillation, or memory-constraints). Specifically, SplitNet [55] splits the network into disjoint parts during training without any constraints on individual network partitions. This unconstrained splitting can result in large partitions which may not conform to memory-budgets of individual IoT devices. Hence, taking specific memory budgets into account is more effective (and necessary) in order to meet hard memory-constraints of IoT-devices; such constraints are not considered by SplitNet [55]. Moreover, by partitioning the feature activation maps of convolutions, MoDNN [71] and DeepThings [140] successfully reduce the FLOPS and the feature activation map memory during distributed inference. However, both of these works assume that the mobile device is big enough to fit the entire deep learning model, which is a strong assumption and, hence, they do not address the memory due to model weights. In contrast, we take the distributed inference one step further to an even more constrained IoT environment (*e.g.*, 500KB memory) where the physical IoT-devices cannot fit a single model. In such cases, the model itself must be split in non-intuitive ways which can result in heavy communication cost at every layer. Therefore, in this chapter, our proposed NoNN compresses the model (to reduce the memory and computation costs such that individual modules fit on the devices) while accounting for communication among the student nodes.

Note that, most of the prior techniques mentioned above are complementary to our work so they can be used synergistically on top of our approach. For instance, another recent work called MeDNN [72] extends MoDNN with a pruning technique to reduce the per-node computation. Since pruning can still be performed on top of our proposed NoNN and because both MeDNN and MoDNN partition the feature maps (and not the weights), our work is complementary to MeDNN [72]. We also show that we can achieve higher accuracy than several KD baselines and SplitNet in our experiments, *i.e.*, the only two techniques in Table 5.1 that do not complement our work.

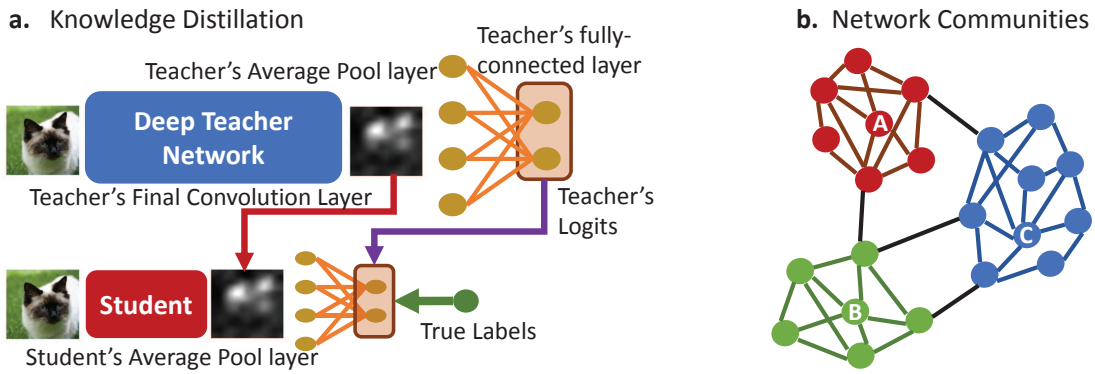


Figure 5.2: (a) Knowledge Distillation (KD) is based on a significantly smaller student model trained to mimic a large teacher network. (b) Network communities and hub nodes.

5.2.2 Knowledge Distillation

Recall that KD [43] consists of two deep networks: (*i*) *Teacher* is the large deep network which we want to compress, and (*ii*) *Student* is a significantly smaller neural network which is trained to mimic the output of the teacher network. This is illustrated in Fig. 5.2(a). KD uses both the *hard-label loss* (based on true labels from the dataset), as well as the *soft-label loss* (based on logits) to train a significantly smaller student model. Mathematically, let l_T and l_S be the logits of teacher and student respectively, and τ is a temperature parameter (see [43] for more details), y be the true labels, and P_T^τ and P_S^τ respectively denote the softmax over relaxed logits l_T/τ and l_S/τ. Then, the KD loss ($\mathcal{L}^{kd}$) is given by:

$$\mathcal{L}^{kd}(\theta_S) = (1-\alpha)\mathbb{H}(y, P_S) + \alpha\mathbb{H}(P_T^\tau, P_S^\tau) \tag{5.1}$$

where, $\mathbb{H}$ is the standard cross-entropy loss, θ_S denotes the parameters of the student network, and α controls the weight of hard-label loss *vs.* soft-label loss. The temperature parameter in the second term of Eq. (5.1) improves knowledge transfer from the teacher to the student. Finally, variants of KD like Attention Transfer-based KD (ATKD) additionally use intermediate outputs of teacher's convolution layers while training the student [135]. Of note, KD-based techniques have also been proposed for RNNs [56, 117]. However, our focus in this chapter is on KD-based distributed inference for CNNs.

This completes our review of KD. More details on KD are given in Section 2.3. We next briefly describe the network science concepts used in this work.

5.2.3 Network Science Concepts for Model Compression

To our knowledge, we are the first to utilize network science concepts for model compression. Below, we review the core ideas from network theory used in this chapter.

Node degree and hubs in a network Node degree refers to total number of connections for a given node in the network. Specifically, for an undirected network $\mathcal{G}$ with nodes $\mathcal{V}$, links $\mathcal{E}$, and the adjacency matrix $\mathcal{F} = \{F_{ij}\}$ describing link weights between any two nodes $i, j \in \mathcal{V}$, degree k_i of a node i refers to total number of links connected to node i. Moreover, a network can have a *scale-free* structure where some nodes have many connections but most nodes have low degree [85]; these nodes with a large number of connections in scale-free networks act as *hubs* of information. Fig. 5.2(b) shows nodes A, B, and C as examples of hubs in a network. As evident, these nodes have significantly higher number of connections than other nodes in the network.

Community Structure Recall that many real world networks are characterized by groups of tightly connected nodes known as the community structure [86] of a network. For example, social networks have several communities of users, where a community can refer to a group of users with common interests like sports or politics. Formally, a community can be defined as a group of nodes for which the number of connections within the group is significantly higher than what one would expect at random. Fig. 5.2(b) shows the community structure in a network.

We will use the concept of hubs and communities in the context of deep networks throughout our approach. Next, we provide concrete evidence towards why memory- and communication-aware model compression is needed.

5.3 Motivation

We now discuss the impact of horizontally splitting a Convolutional Neural Network (CNN) on distributed inference performance. When a CNN does not fit on a single memory-constrained device, it must be split horizontally for parallel execution on multiple devices (for better utilization of all resources). However, as mentioned in Section 5.1, this can lead to heavy communication

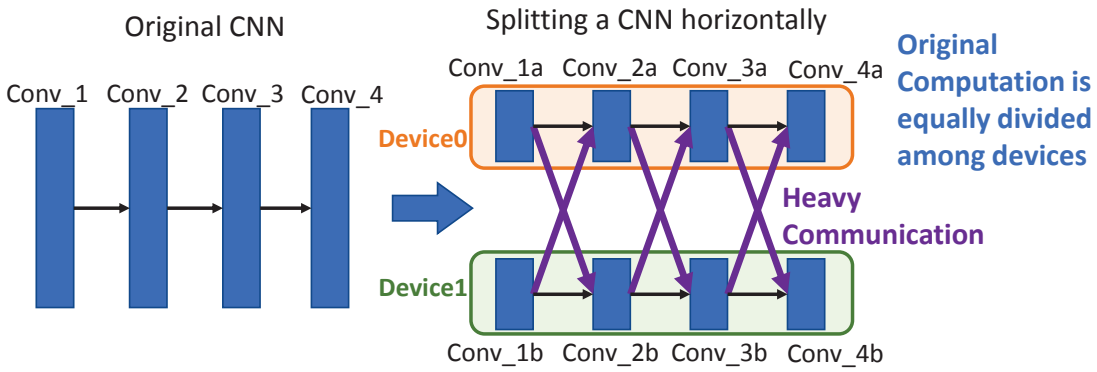


Figure 5.3: Splitting a deep network horizontally leads to huge communication cost at every step since the next layer convolutions require access to all the input channels. Original computation is equally divided between the two devices.

when each split is deployed on a separate device. To verify this assumption, we conduct the following experiment.

We start with a large Wide Resnet[1] (WRN40-4) teacher model with 8.9 million parameters, trained on CIFAR-10 dataset [134]. This model takes about 86ms for inference on a single powerful x86 machine. Next, we split the WRN40-4 model horizontally (*e.g.*, if a layer has 32 channels, we split it into two parts with 16 channels each). We then deploy the individual horizontal splits on two powerful x86 machines connected via a wired point-to-point connection. As shown in Fig. 5.3, since each successive convolution layer requires input from all the output channels from the previous layer, we need to make the output from all devices available to all other devices. Clearly, this results in significant communication overhead. This is why, even though the computation happens on two powerful x86 machines, the inference time increases **from 86ms to 1006ms** ($> 10\times$ increase in latency). Hence, this experiment shows that, when a deep network does not fit on an IoT-device, splitting it directly is not an option. Obviously, the problem gets exacerbated for edge devices which often operate on low frequencies and, hence, take even longer for computation.

As a result, a new memory- and communication-aware model compression technique is imperative for distributed inference on IoT-devices. Therefore, we next describe our proposed approach.

[1]The base WRN architecture consists of three groups: G_0, G_1, and G_2 with width (*i.e.*, #channels) in each group as $[16, 32, 64]$ respectively. Width multiplier is used to increase the number of channels per group [134]; WRN40-4 implies 40 layers, and a width multiplier of 4 which results in $[64, 128, 256]$ channels per group (see Fig. 5.5a).

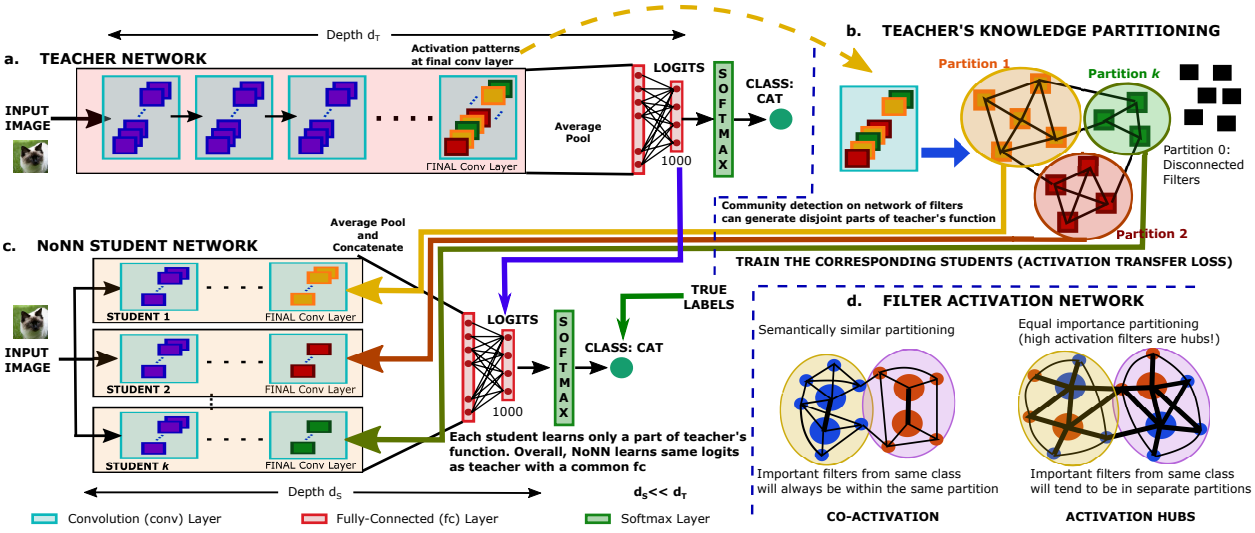


Figure 5.4: Complete flow of our approach. (a) The pretrained teacher model exhibits certain activation patterns for the validation set. (b) We use these activation patterns to build a network of filter activity. Community detection on the filter network partitions teacher's knowledge into simpler functions. (c) Train individual students in NoNN to mimic each partition. (d) Teacher's knowledge can be partitioned in different ways depending on how the filter activation network is built. Node colors represent filters relevant to a given class, whereas node sizes represent importance of a filter (determined by activation value).

5.4 Proposed Approach

We first formulate our problem (Section 5.4.1), followed by our proposed solution which consists of two stages: (1) Network science-based knowledge partitioning of the teacher: We find disjoint partitions of teacher's knowledge which lead to independent students with minimal communication costs (Section 5.4.2, Fig. 5.4b). (2) Proposed NoNN architecture and the training process: We first select an efficient model architecture for each student that reduces its memory and FLOPS, and then jointly train individual NoNN students on disjoint partitions from teacher (Section 5.4.3; Fig. 5.4c).

5.4.1 Problem Formulation

In this section, we formulate teacher's knowledge partitioning subject to the resource-constraints of individual students.

As shown in Fig. 5.4a, the output of final convolution (fconv) layer of teacher (T_{fconv}) gets average-pooled and passes through the fully-connected (fc) layer to yield *logits*. The logits then pass through the softmax layer to generate probabilities. Since our goal is to break teacher's learned function into multiple disjoint parts, we focus on teacher network's fconv. Specifically, we partition the teacher's fconv layer by looking at the *patterns of activation of filters* as validation images pass through the teacher (see Section 5.4.2).

Partitioning a teacher's fconv often uncovers a partition P_0 which does not contribute to teacher's validation accuracy. Therefore, we should find and remove the largest set of filters $P_0 \subset T_{fconv}$ that does not contribute to teacher's accuracy since such filters only transfer noise to the students during distillation (Section 5.4.2). Moreover, the sizes of partitions of T_{fconv} should be almost equal so that the resulting student networks fit within a memory budget $\mathcal{B}_{mem}$, while their computation costs fit a FLOP budget $\mathcal{B}_{FLOP}$. Let $h : \mathbb{R} \rightarrow \mathbb{R}$ compute the memory of student i depending on its partition size $|P_i|$, and $f : \mathbb{R} \rightarrow \mathbb{R}$ compute the resulting FLOPS. Then, the *knowledge-partitioning problem* is mathematically expressed as follows:

$$
\begin{aligned}
\min_{\mathcal{P}=\{P_0,P_1,\ldots,P_k\}} \quad & |\Delta\text{val}| - |P_0| \\
\text{subject to} \quad & |P_1| \approx |P_2| \ldots \approx |P_k|,\ P_i \sim \text{Rule } R,\ P_i \subset T_{fconv} \\
& \qquad\qquad\qquad\qquad\qquad\qquad \forall i \in \{0, \ldots, k\} \\
& |\Delta\text{val}| < \epsilon \\
& \max(h(|P_1|), h(|P_2|), \ldots, h(|P_k|)) < \mathcal{B}_{mem} \\
& \max(f(|P_1|), f(|P_2|), \ldots, f(|P_k|)) < \mathcal{B}_{FLOPS}
\end{aligned}
\tag{5.2}
$$

where, $|\Delta\text{val}|$ denotes the absolute value of change in validation accuracy due to removal of P_0 filters. Also, $|P_i|$ does not denote the absolute value but rather the number of elements in the set P_i (*i.e.*, the cardinality of P_i). The first constraint in (5.2) aims to keep the sizes of all partitions (except P_0) almost equal since the sizes of student models must be below the *fixed* memory budget $\mathcal{B}_{mem}$. The objective function and the second constraint aim to minimize change in teacher's validation accuracy while removing as many useless filters from T_{fconv} as possible. The last two constraints in problem (5.2) specify the per-device memory- and FLOP-budgets that must be satisfied by each student (which will be deployed on individual devices). Note that, while it is always possible to satisfy memory- and FLOP-constraints, there will be a tradeoff between model-size or

computation and the NoNN accuracy. For instance, for fixed number of students, larger individual student modules can lead to higher overall accuracy. Finally, the rule R to determine how the partitions are formed is described below.

5.4.2 Network Science-Based Knowledge Partitioning via the Proposed Filter Activation Network

We propose a network science-based solution to the knowledge partitioning problem. The idea is to uniformly distribute knowledge from teacher's fconv using network science in order to jointly train individual students in the NoNN.

To partition teacher's fconv, we first build a network $\mathcal{F} = \{F_{ij}\}$ of filter activation patterns, where i and j are two filters. Now, let a_i denote the *average activity* of a filter i for a given image in the validation set (val). Average activity of a filter i is defined as the averaged output of the corresponding output channel of teacher's fconv. For instance, suppose a teacher network's fconv has 256 output channels, and height and width of its activation map is 8×8 pixels. Then, to obtain average activity of filter i, we average the value of its 8×8 output channel activation map. Essentially, we use this average activity metric as a measure of *importance* of a filter for a given class of images; that is, the higher the average activity of a filter, the more important it is for classification for some class of images. In Fig. 5.4d, each circle depicts a filter at teacher's fconv, and filters shown in same color activate for similar classes. Also, the bigger the size of the circle, the more important is the filter for the corresponding class (*i.e.*, it has higher average activity).

To create the filter activation network, there exist multiple strategies (*i.e.*, rule R's) in which the filters can be connected. Fig. 5.4d illustrates two such strategies: **(*i*) Activation Hubs (AH):** Encourage connections between very important and less important filters (*i.e.*, less number of high-importance filters get surrounded by many low-importance filters and thus act as hubs), and **(*ii*) Co-Activation (CA):** Filters that activate together for similar classes get connected. The former rule encourages partitions of roughly equal importance (since each partition will have a few important filters). The latter partitions the filters into semantically similar parts. Formally,

$$F_{ij} = \begin{cases} \sum_{\text{val}} a_i a_j |a_i - a_j| & \textbf{AH Rule} \\ \sum_{\text{val}} \frac{a_i a_j}{|a_i - a_j| + 1} & \textbf{CA Rule} \end{cases} \tag{5.3}$$

Let us analyze what happens during the AH case. If i and j are two filters with average activities a_i and a_j, respectively, the F_{ij} link weight for AH case will be high if, for a given image, either a_i is high and a_j is low, or vice versa. Note that, if either a_i or a_j is close to zero, that link is not created. Further, when a_i and a_j are both very high for the same image (*i.e.*, both filters are important for same class), then that link will also not be encouraged (since the link weight will be low due to the $|a_i - a_j|$ part). Therefore, important filters from the same class will be discouraged from connecting together. Consequently, for AH Rule, highly important filters (with high a_i) will be encouraged to connect with not-so-important filters (from same or other classes). Important filters from same classes will be forced to occupy separate communities; hence, the knowledge is distributed uniformly across the students. Similar discussion holds for CA Rule where the communities represent semantically-similar features. Of note, this flexibility in partitioning teacher's fconv comes from network science ideas.

Of course, there can be other ways filters may be connected. Indeed, if the dataset contains too many semantically similar classes, CA can lead to heterogeneous student models (because many filters from T_{fconv} will go into a few partitions). Hence, to obtain student models of similar sizes, we only explore AH in this chapter. In matrix form, adjacency matrix of AH network can be written as:

$$\mathcal{F}^{AH} = \sum_{n \in \text{val}} a_n a_n^T \odot |\mathbf{D}_n - \mathbf{D}_n^T|, \tag{5.4}$$

where, a_n is the vector of average filter activities (a_i's) for each image $n \in$ val, matrix $\mathbf{D}_n$ contains all columns as a_n, and $\odot$ denotes element-wise multiplication. To partition this AH network, we need to detect communities by maximizing a modularity function as explained in the network science literature [86]:

$$\max_{\mathbf{g}=\{g_0,g_1,\ldots,g_{l-1}\}} \frac{1}{2m} \sum_{ij} \left[F_{ij}^{AH} - \frac{1}{\gamma} \cdot \frac{k_i k_j}{2m} \right] \delta(g_i, g_j), \tag{5.5}$$

where, m is #edges, k_i is degree (number of connections) of node i, resolution γ controls the size/number of communities, and δ is Kronecker delta. The idea is to find groups of tightly connected nodes, $\mathbf{g} = \{g_0, g_1, \ldots, g_{l-1}\}$, which map the nodes $\mathcal{V}$ to l communities. Finally, these l communities are converted to k partitions P_i by combining the communities such that the partition sizes are almost equal (so that resulting students stay within the budget). This partitioning $\mathcal{P}$ is then used to train individual student networks. Of note, removing community g_0

(which consists of all disconnected nodes in the filter network) often has very little impact on teacher's accuracy. Hence, partition $P_0 = g_0$ is obtained directly by community detection. Consequently, we solve problem (5.2) by detecting and removing the largest community g_0 that does not change teacher's validation accuracy, and then combine the rest of the communities to form the remaining partitions. For instance, say we have four remaining communities after removing g_0: $\{|g_1| = 12, |g_2| = 14, |g_3| = 25, |g_4| = 30\}$ ($|x|$ indicates size of group x), and we want to combine them into two partitions of nearly equal sizes. Then, we can combine them as $\mathcal{P} = \{|P_1| = |g_2 \cup g_3| = 39, |P_2| = |g_1 \cup g_4| = 42\}$. Hence, we obtain the required k almost equal partitions from l communities.

5.4.3 Network of Neural Networks (NoNN)

Once we have partitioned the teacher's fconv, we jointly train our student networks on individual partitions from T_{fconv} as shown in Fig. 5.4c. However, we must first make the following design decisions: (*i*) Select the deep network architecture for individual students, and (*ii*) Select how students are connected together in NoNN.

Student Architecture Selection

Similar to prior art in KD [43, 135], we select our individual student networks to be significantly reduced versions of the teacher (*i.e.*, far fewer layers and lower width). Specifically, we pick our individual student models based on user-defined memory- and FLOP-constraints for IoT-devices. For instance, for CIFAR-10 dataset, our teacher is Wide Resnet model WRN40-4 with 40 layers and width multiplier of 4. That is, as shown in Fig. 5.5a (left), it has 64 channels in first group, 128 in second group, and 256 in third group. Also, suppose that our each individual student must have less than 500K parameters. Then, we select our individual student (see Fig. 5.5a (right)) with 16 layers and channel-width appropriately adjusted so that each student has less than 500K parameters. Note that, student networks are not exactly the same as they mimic different partitions. Therefore, we need an additional 1×1 convolution layer to make the dimensions equal between the teacher's knowledge partition and individual student's output layer. Next, we describe the architecture of NoNN containing multiple such students, and the training process.

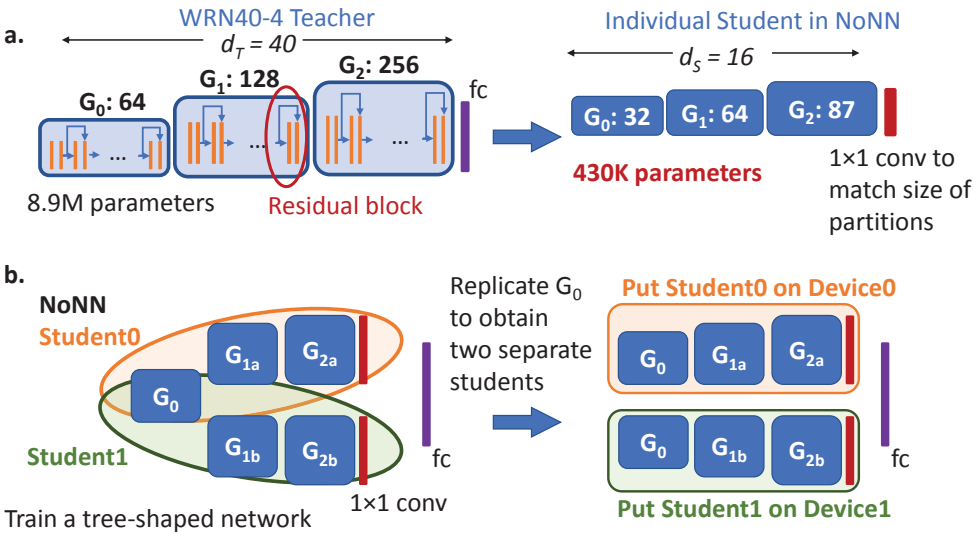


Figure 5.5: (a) Selecting an individual NoNN student architecture: Reduce depth and channel-width so that it fits within the desired budgets. (b) Overall, NoNN trains a tree-shaped network (where possible) with some initial layers common (denoted as G_0). At inference time, replicate the common group G_0 and put individual students on separate devices.

NoNN Architecture

A NoNN consisting of two students is shown in Fig. 5.5b. As evident, wherever possible, we make the initial few layers from the students common (since the knowledge learned at initial few layers will be common for all students). Moreover, at inference time, the common layers between all students can be simply replicated across multiple devices without significant increase in memory per student (since the common group is not too big). Once the common group is replicated, the individual students are completely independent and do not communicate until the final fc layer (see Fig. 5.5b). Hence, we obtain a final NoNN architecture which consists of multiple disjoint students. The output from all students is concatenated and passed through a fc to yield logits.

Training Loss

To train the overall NoNN, we use the KD loss (see Eq. (5.1)) and, as given below, we additionally propose a new loss function called *activation-transfer* for transferring partitioned knowledge from the teacher to individual students. The main motivation behind the activation-transfer loss function is that each individual student must mimic the corresponding partition of teacher's knowledge.

Specifically, for each partition of teacher's knowledge, we want to minimize the error between activations of the teacher's filters (that belong to the given partition) and activations of filters in the corresponding student. We define the activation-transfer loss function as given below:

$$\mathcal{L}^{Act}(\theta_S) = \sum_{p \in \mathcal{P}} \left\| \frac{v_T^F(p)}{||v_T^F(p)||} - \frac{v_S^F(p)}{||v_S^F(p)||} \right\|_2^2 \tag{5.6}$$

where, $v_Z^F(p) = vec(A_Z^F(p))$, $Z \in \{T, S\}$ is the vectorized fconv activations for partition $p \in \mathcal{P}$ of teacher or for individual student network in NoNN, and the $||v_Z^F(p)||$, $Z \in \{T, S\}$ terms denote the partition-wise normalization. Hence, the total loss is given by $\mathcal{L} = \mathcal{L}^{kd}(\theta_S) + \beta \mathcal{L}^{Act}(\theta_S)$ which can be minimized via stochastic gradient descent.

This completes our proposed NoNN. We next present detailed experimental evaluation for NoNN.

5.5 Experimental Setup and Results

In this section, we first present our experimental setup and then results for different datasets and NoNN architectures.

5.5.1 Experimental Setup

We compress various deep networks for five well-known image classification tasks: CIFAR-10, CIFAR-100, Scene [98], Caltech-UCSD-Birds (CUB) [125], and Imagenet. Scene and CUB datasets belong to the transfer learning domain where a pretrained model is finetuned on a different problem. For comparison, we consider strong teacher-student baselines like KD [43], Attention-transfer with KD (ATKD) [135]. For KD and ATKD, we use models from WRN family [134] as single, large students. Also, *total parameters* in NoNN (*i.e.*, parameters in all students combined) are comparable to the parameters in KD/ATKD baseline models. We also show that we outperform prior models like Splitnet [55]. Further, we show experiments with up to 8 student networks for CIFAR-10/100. With these experiments, we thoroughly demonstrate that NoNN compresses the teacher model into disjoint subsets which do not communicate until the fc layer.

We first train the teacher model on 90% of training dataset while saving 10% for validation. This validation set is used for building the filter activation network to partition the teacher's knowl-

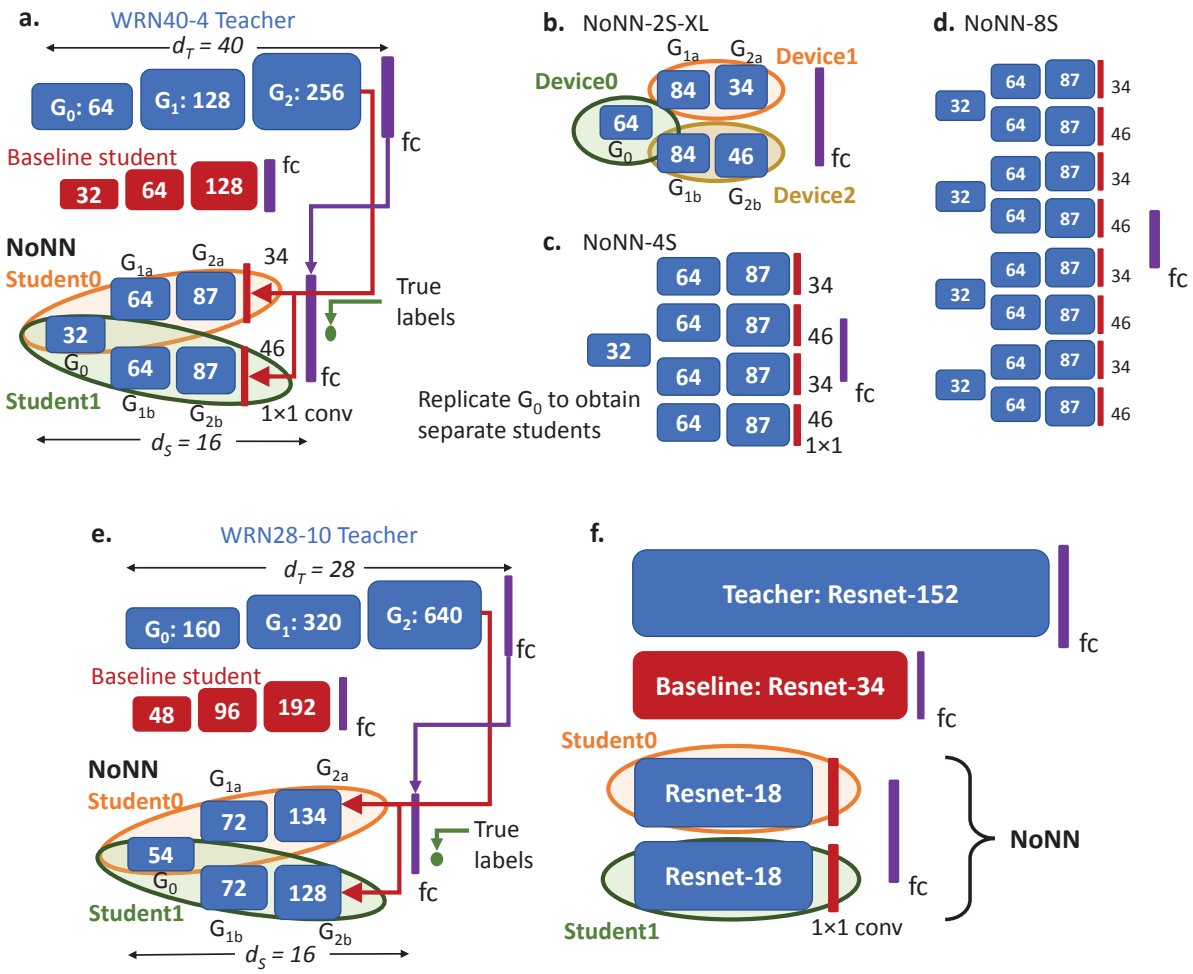


Figure 5.6: Teacher, baseline student, and NoNN models for various datasets. (a) WRN-40-4 teacher, WRN-16-2 baseline, and 2-student NoNN for CIFAR-10. (b) A larger 2-student NoNN (NoNN-2S-XL) can be distributed on three separate devices to keep FLOPS under a budget. (c, d) 4-student and 8-student NoNNs. (e) CIFAR-100 models, (f) Transfer learning models. After replicating the common groups (if any), the individual students in our proposed NoNN will communicate only at fc layer.

edge. Then, we use the Activation Transfer loss given in Eq. (5.6) and KD-loss to train the compressed NoNN model. Moreover, all the accuracies reported in the chapter are on test set for the respective datasets. We set the number of partitions, $k = 2$. To build more than two students, since our objective is to keep the individual student architectures as similar as possible, we simply shuffle the filters in these two partitions while transferring knowledge to rest of the students. This ensures that the size of students is not too diverse and limits the possible factors that can contribute to the improvement in accuracy of our model. Also, we set $\alpha = 0.9$ and selected the best

$\beta = \{10^1, 10^2, \ldots, 10^4\}$ for both ATKD [135] and NoNN. For training NoNN, we set the initial learning rate as 0.1 and used a momentum of 0.9. Finally, our entire framework is implemented in Pytorch which is run on NVIDIA GTX 1080-Ti and Titan Xp GPUs.

5.5.2 Results

We first show results on CIFAR-10/100 and transfer learning datasets (Scene, CUB), and then our preliminary results for Imagenet.

CIFAR-10

Fig. 5.6a depicts our teacher model for CIFAR-10 as WRN40-4 and our baseline student for KD and ATKD as WRN16-2. As evident, knowledge from partitions of only 34 and 46 filters from teacher's fconv is transferred to student-0 and student-1, respectively (via 1×1 convolutions). The rest 176 out of 256 filters at teacher's fconv do not contribute to teacher's accuracy and, hence, form partition P_0 (or community g_0) in $\mathcal{F}^{AH}$; this partition is not used for transferring knowledge to NoNN students. We set $\mathcal{B}_{FLOPS} \approx 200$M which is close to baseline FLOPS used by WRN16-2 model. Further, based on our initial motivation w.r.t. Arm Cortex-M, $\mathcal{B}_{mem} = 0.5$M parameters (since for models quantized to 8-bits, parameters < 0.5M means less than 500KB of memory).

As shown in Table 5.2, our NoNN-2S model (Fig. 5.6a) achieves higher accuracy than the baseline student model. Note that, each student in NoNN utilizes fewer FLOPS (*i.e.*, $167M$ FLOPS) than the allowed budget. Moreover, although *each* student has 0.43M parameters, due to common G_0, total parameters for NoNN-2S in Fig. 5.6a is 0.82M (comparable to 0.7M in WRN16-2). Further, *after replicating* G_0, student-0 and student-1 will not communicate with each other until the final fc layer. Clearly, when quantized, our individual student models of 430K parameters can fit on a device with 500KB memory (430K parameters are after replicating G_0). Indeed, if a device cannot fit more than 500K parameters, prior models such as WRN16-2 (with 700K parameters) will need to be distributed across multiple devices which will lead to communication at every layer; that is, when distributed, the baseline WRN16-2 will communicate at all 16 layers, whereas NoNN communicates only at the final layer.

This situation gets exacerbated by higher accuracy baseline student models (*e.g.*, 40-layer

Table 5.2: CIFAR-10 Teacher-Student Results*

Model	#parameters (largest student)	#FLOPS (largest student)	Accuracy
Teacher WRN40-4	8.9M	2.6G	95.49%
KD WRN16-2 [43]	0.7M	202M	$93.79 \pm 0.15\%$
ATKD WRN16-2 [135]	0.7M	202M	$93.83 \pm 0.08\%$
NoNN-2S	**0.43M****	**167M**	$\mathbf{94.32 \pm 0.17\%}$
KD: NoNN-2S-XL	0.46M	245M	$93.96 \pm 0.09\%$
NoNN-2S-XL	**0.46M****	245M***	$\mathbf{94.53 \pm 0.19\%}$

*For statistical significance, results are reported as mean $\pm$ standard deviation of five experiments.

**All NoNN #parameters/FLOPS are reported for one student only. Complete NoNN has very similar #parameters as baselines for fair comparison. Our contribution is to show that NoNN can be broken down into disjoint parts that stay below certain budgets and do not communicate.

***NoNN-2S-XL can be distributed onto three devices as shown in Fig. 5.6b.

WRN40-2) for which communication costs at every layer grow rapidly. In contrast, as we shall see shortly, our student models can be very flexible: we can have larger students (see Fig. 5.6b), or many disjoint students without significantly increasing the communication costs (*e.g.*, see NoNN-4S and NoNN-8S in Fig. 5.6(c, d)). Concrete energy and latency for CIFAR-10 experiments are reported in the hardware deployment described in Section 5.6.

For an even higher per-device FLOP budget (say, 250M), a NoNN-2S-XL student architecture can be distributed on three devices (see Fig. 5.6b). Although the larger student in this case has 0.46M parameters ($G_0 \rightarrow G_{1b} \rightarrow G_{2b}$), due to a bigger common G_0 group, total parameters for NoNN-2S-XL is 0.77M (again, comparable to 0.7M in WRN16-2). Also, each individual student in this model has higher FLOPS because the common group G_0 is wider and has 245M FLOPS. Since it is impractical to replicate this group, as shown in Fig. 5.6b, we can assign a completely new device (device-0) to this group and put $G_{1a} \rightarrow G_{2a}$ ($G_{1b} \rightarrow G_{2b}$) on device-1 (device-2).

Table 5.2 demonstrates that our NoNN-2S-XL achieves even better accuracy (¡1% away from the teacher). Moreover, this increase in accuracy is not only due to an increased FLOP budget. To demonstrate, we show that filter network community-based knowledge transfer from teacher plays a more significant role in accuracy improvement. If we simply train our NoNN-2S-XL student

Table 5.3: CIFAR-100 Teacher-Student Results*

Model	#para (largest student)	#FLOPS (largest student)	Accuracy
Teacher WRN28-10	36.5M	10.48G	79.28%
KD WRN16-3 [43]	1.5M	446M	$73.99 \pm 0.26\%$
ATKD WRN16-3 [135]	1.5M	446M	$74.90 \pm 0.17\%$
NoNN-2S	**0.85M**	**345M**	$\mathbf{75.74 \pm 0.31\%}$

*For statistical significance, results are reported as mean $\pm$ standard deviation of five experiments.

model via KD, *i.e.*, without the $\mathcal{L}^{Act}$ loss defined in Eq. (5.6), the accuracy increases only slightly over the baseline WRN16-2 (*e.g.*, 93.96% *vs.* 93.79%). Therefore, community-based knowledge partitioning is extremely important for successfully training our proposed NoNN.

Next, we show that NoNN is scalable to many datasets and many model sizes for different memory and FLOP budgets.

CIFAR-100 and Comparison with SplitNet

Table 5.3 shows results for CIFAR-100. Here, the teacher is a WRN28-10 model with 36.5M parameters and our baseline student is WRN16-3 model containing 1.5M parameters (see Fig. 5.6e). Therefore, we create a NoNN with overall 1.5M parameters as shown in Fig. 5.6e; this NoNN can be split into two students where the larger student has 0.85M parameters (after replicating G_0). Also, the FLOPS for our larger student are lower than the WRN16-3 baseline. Again, NoNN-2S model is found to be more accurate than KD and ATKD, while creating disjoint students that can fit within some memory budget (say, 1MB). Once our disjoint students fit within the 1MB budget, techniques like MoDNN [71] can further distribute our model's computation on more devices with similar memory-constraints to reduce the FLOPS. MoDNN assumes that the model fits on the device, and our work makes it possible to meet the memory-constraints.

We next compare our model to SplitNet [55] that splits a deep network into multiple disjoint parts (albeit without any model compression ideas such as resource-constraints). Kim *et al.* obtained two CIFAR-100 models via SplitNet: (*i*) A 7.42M parameter WRN model with 2 sub-groups which achieves 76.04% accuracy, and (*ii*) A 4.12M parameter model with 4 sub-groups which achieves 75.2% accuracy. For fair comparison with SplitNet, we create two corresponding models:

First with 2-students (7.42M parameters), and the second with 4-students (4.13M parameters), respectively. While keeping the experimental setup same as SplitNet (*e.g.*, same validation set size, *etc.*), we achieve $\mathbf{79.07}\%$ **accuracy** for the 7.42M parameter model, and $\mathbf{77.42}\%$ **accuracy** for the 4.13M parameter model. Therefore, the proposed NoNN significantly outperforms SplitNet for similar model-sizes. Moreover, the NoNN-2S model shown in Table 5.3 which has merely 1.5M total parameters (*i.e.*, 0.85M parameters per student) achieves higher accuracy than the 4.13M parameter SplitNet model.

Varying Number of Students

Now, we vary students from two to eight for CIFAR datasets. As evident from Table 5.4, NoNN models achieve close to teacher's accuracy while reducing memory/FLOPS by orders of magnitude (*i.e.*, $2.5\times$-$24\times$ reduction in $\#$parameters and $2\times$-$15\times$ reduction in FLOPS w.r.t. teacher). Specifically, the NoNN-8S model in Fig. 5.6d achieves 95.02% accuracy (*i.e.*, less than 0.5% away from teacher) while using eight separate students (after G_0 is replicated), each of which can fit within 167M FLOPS and 430K parameters (*i.e.*, < 500KB when quantized to 8-bits), and do not communicate until the final fc layer. Overall, this results in $2600/(167 \times 8) \approx 2\times$ reduction in FLOPS, and $8.9/(0.43 \times 8) \approx 2.5\times$ reduction in parameters over the teacher.

Finally, to compare NoNN-8S for CIFAR-10, we also experimented with a larger WRN40-2 student (40 layers, 2.2M parameters, and 655M FLOPS) with ATKD [135]. Although, this model achieves similar accuracy of 95.03%, it clearly does *not* result in an architecture that can be distributed on multiple devices (since it will require 2.2MB storage when quantized to 8-bits). As a result, distributing this single model on multiple devices will lead to heavy inter-device communication at each of the 40 layers (see Section 5.3), whereas our model will communicate only at the final layer. This clearly highlights the significance of our proposed NoNN.

Transfer Learning Datasets

We now demonstrate the effectiveness of NoNN for transfer learning tasks where the idea is to finetune a model pretrained on Imagenet w.r.t. a new dataset. Table 5.5 shows results for two such datasets: (*i*) Indoor Scene Classification, (*ii*) Caltech-UCSD Birds. For both datasets, we used Resnet-152 as teacher model, Resnet-34 as baseline student, and our NoNN consists of two

Table 5.4: Accuracy for more students*

#S	CIFAR-10						CIFAR-100		
	NoNN	ΔM**	ΔF**	**NoNN-XL**	ΔM	ΔF	**NoNN**	ΔM	ΔF
2	$94.32 \pm 0.17\%$	10×	7.7×	$94.53 \pm 0.19\%$	11×	5×	$75.74 \pm 0.31\%$	24×	15×
4	$94.64 \pm 0.16\%$	5.1×	3.9×	$94.87 \pm 0.14\%$	6.3×	3.3×	$77.07 \pm 0.18\%$	12×	7.6×
6	$94.90 \pm 0.06\%$	3.5×	2.6×	$95.02 \pm 0.10\%$	3.9×	1.7×	$\mathbf{77.25 \pm 0.24}\%$	7.7×	5×
8	$\mathbf{95.02 \pm 0.08}\%$	2.5×	2×	$\mathbf{95.28 \pm 0.07}\%$	2.9×	1.3×	$77.03 \pm 0.17\%$	5.7×	3.8×
T	95.49%	1×	1×	95.49%	1×	1×	79.28%	1×	1×

*For statistical significance, results are reported as mean ± standard deviation of five experiments.

** ΔM (ΔF) shows compression rate, *i.e.*, ΔMemory (ΔFLOPS) for *complete* NoNN (*i.e.*, parameters of all students combined) w.r.t. teacher.

Table 5.5: Transfer Learning Results*

Model	#para (1 student)	#FLOPS (1 student)	Accuracy		ΔM**	ΔF**
			Scene	CUB		
Teacher Resnet-152	58M	11G	79.44%	80.94%	1×	1×
KD Resnet-34 [43]	22M	4G	$76.92 \pm 0.39\%$	$78.59 \pm 0.22\%$	2.5×	2.8×
ATKD Resnet-34 [135]	22M	4G	$\mathbf{77.63 \pm 0.94}\%$	$79.10 \pm 0.56\%$	2.5×	2.8×
NoNN-2S	**11M**	**2G**	$\mathbf{77.48 \pm 0.76}\%$	$\mathbf{79.81 \pm 0.33}\%$	2.5×	2.8×

*For statistical significance, results are reported as mean ± standard deviation of five experiments.

** ΔM (ΔF) shows compression rate, *i.e.*, ΔMemory (ΔFLOPS) for *complete* NoNN (*i.e.*, parameters of all students combined) w.r.t. teacher.

separate Resnet-18 models with a common fc layer (see Fig. 5.6f). As evident from Table 5.5 and Fig. 5.6f, our NoNN-2S model (total 23M parameters) consists of disjoint parts of 11M parameters each, which do not communicate until the last layer. Also, for Scene dataset, NoNN-2S achieves slightly lower (but still comparable) accuracy than ATKD on Resnet-34. However, NoNN-2S achieves much higher accuracy than prior approaches for CUB.

Preliminary Imagenet Results

We used the fastai setup [49] to obtain imagenet results in a fast and inexpensive way. Our teacher model is Resnet-34 with 73% top-1 and 91% top-5 accuracy. We used Resnet-18 model (11.68M parameters) as a baseline ATKD student which achieved 69.98% top-1 and 89.50% top-5 accuracy. Our NoNN consists of two students (each with around 6M parameters; total 11.69M parameters). For training our NoNN, we use attention transfer losses [135] on *each* student since activation

transfer loss led to some underflow problems[2]. Even with this setup, our NoNN achieved *comparable* accuracies, *i.e.*, 69.82% top-1, and 89.61% top-5 while ensuring that the two students do not communicate until the last layer. With availability of more resources, we should be able to improve these results even further.

So far, we have shown that NoNN achieves higher accuracy than several baselines, and minimal communication among students. We have also demonstrated that NoNN achieves close to teacher's accuracy with $2.5\times$-$24\times$ lower memory, and $2\times$-$15\times$ fewer FLOPS than the teacher model. Next, we deploy our models on real edge devices and show significant gain in performance and energy.

5.6 Case Study: Hardware Deployment

In this section, as a case study, we deploy our NoNN models on two hardware-constrained devices to quantify their effectiveness in terms of latency and energy on real hardware.

5.6.1 Hardware Setup

We implement NoNN on edge devices such as Raspberry Pi (RPi) and Odroid-XU4S. We demonstrate an extensive exploration of two scenarios that can benefit from our approach: (*i*) Homogeneous case in which all devices are identical, and (*ii*) Heterogeneous case, where devices have different resources and computing power. We perform detailed evaluation of accuracy, performance, and energy for each scenario. We show these results for our teacher (WRN40-4), NoNN-2S, and NoNN-8S models trained on the CIFAR-10 dataset.

We use eight Raspberry Pi-3 Model-B and eight Odroid-XU4S boards. Each RPi has an Arm Cortex-A53 quad-core processor with nominal frequency of 1.2GHz and 1GB of RAM. We scaled the frequency of RPi's down to 400MHz as the RPi's running at maximum frequency demonstrated unstable behavior and would often crash due to high temperature. The XU4S boards have a Samsung Exynos5422 SoC, which has an Arm big.LITTLE architecture with 4 big Cortex-A15 and 4 little Cortex-A7 cores, and 2GB of RAM. The Odroid is executed at the nominal frequency: 2GHz

[2]Fastai uses a floating point-16 (fp16) format for computation on 8 NVIDIA Volta GPUs. Due to an underflow problem on fp16 (which is well-known to happen with fp16 format), we couldn't use activation transfer $\mathcal{L}^{Act}$ losses for training our NoNN.

for the big cores and 1.4GHz for the little cores. To maximize performance, we run the inference on the big cores in the Odroids.

In order to measure voltage, current, and power, we use the Odroid Smart Power 2 for the Odroid-XU4S and AVHzY USB Power Meter for the RPi's. In addition, we use an x86 machine (Core i7), further referred as *server*, to send the images to the devices, concatenate the results, and apply the fc layer. Of note, the fc is small and could also be placed in another edge/mobile device. We perform all of our experiments with a point-to-point wired local network connecting all devices together.

Since the NoNN models were built with PyTorch, we use TVM[3], a deep learning compiler that optimizes NN models for several hardware architectures, to deploy the trained model into the RPi's and Odroid boards. To achieve this, the PyTorch models are converted into an intermediate representation with Open Neural Network Exchange (ONNX) [27]. Then the ONNX representation is used by TVM to generate the binary that is deployed in the target device. In addition, we integrate these binaries into a TCP-wrapper so that each student can be distributed to separate devices and the communication is performed through the TCP protocol.

5.6.2 Hardware Results

We first show the results for NoNN when all devices are identical (*i.e.*, the homogeneous case). Then, we address the heterogeneous scenario, where one device is more powerful than the rest. We also discuss the case when some devices are unavailable for inference.

Homogeneous Devices

Table 5.6 and Table 5.7 present the evaluation of NoNN-2S, NoNN-8S, and teacher models for RPi's and Odroids respectively. In these experiments, each student is deployed on a different device, so in the 8-student case we have eight separate devices (either RPi or Odroid). From the experiment in Section 5.3, since splitting the teacher model horizontally leads to more than $10\times$ increment in latency (even on powerful x86 machines), we executed the teacher on a single edge device, as otherwise it would take even longer.

[3]TVM Compiler: `https://tvm.ai/`

Table 5.6: Accuracy, Performance, and Energy Results for CIFAR-10 on both Raspberry Pi

	Raspberry Pi				
	Teacher	2S	8S	Improvement	
				2S	8S
Accuracy	95.49%	94.32%	95.02%	-1.17%	-0.47%
Latency (ms)	1405	115	150	12.22×	9.37×
Energy per node (mJ)	3430.67*	238.98	249.15	14.36×	13.77×
Theoretical FLOPS per student	2.6G*	167M	167M	15.56×	15.56×

*Teacher model is executed on a single node since splitting the teacher model onto multiple devices incurs heavy communication latency (see Section 5.3). NoNN energy and FLOP numbers are reported per student because we are mostly concerned with per node budgets for memory and FLOPS.

Table 5.7: Accuracy, Performance, and Energy Results for CIFAR-10 on both Odroid-XU4S

	Odroid-XU4S				
	Teacher	2S	8S	Improvement	
				2S	8S
Accuracy	95.49%	94.32%	95.02%	-1.17%	-0.47%
Latency (ms)	224	25	36	8.96×	6.22×
Energy per node (mJ)	3084.08*	219.07	237.45	14.08×	12.99×
Theoretical FLOPS per student	2.6G*	167M	167M	15.56×	15.56×

*Teacher model is executed on a single node since splitting the teacher model onto multiple devices incurs heavy communication latency (see Section 5.3). NoNN energy and FLOP numbers are reported per student because we are mostly concerned with per node budgets for memory and FLOPS.

On the RPi's, NoNN-2S improves the performance and energy per node by $12.22\times$ and $14.36\times$ w.r.t. teacher. Also, since NoNN-8S synchronizes eight devices, it achieves $9.37\times$ better latency and $13.77\times$ better energy per node. The same trend is observed for the Odroid boards, having $8.96\times$ and $14.08\times$ improvement for NoNN-2S in performance and energy per node, respectively, and $6.22\times$ and $12.99\times$ for NoNN-8S. Note that, the reduction in theoretical FLOPS for NoNN w.r.t. the teacher is about $15\times$, while energy per node reduction for both RPi's and Odroids is 13-14×. This shows very good agreement between theory and practice. Further, the latency and energy per node do *not* increase significantly between NoNN-2S and NoNN-8S for both devices. Therefore, our proposed NoNN leads to significant gains in performance and per node energy,

Table 5.8: Average Latency Breakdown Per Inference

Latency (ms)	Raspberry Pi		Odroid-XU4S		x86
	2S	8S	2S	8S	2-Split Teacher
Computation	101.91	106.24	19.01	20.60	86
Communication	13.10	43.76	6.00	15.40	920

while maintaining accuracy within 0.5%-1.17% compared to the teacher.

Table 5.8 presents the average latency breakdown for computation and communication per image, considering the 2S and 8S cases. The computation time for one image increases by merely 4% and 8% for RPi and Odroid, respectively, when going from 2S to 8S, because each device operates in parallel. But when communication is considered, the overhead of sending and receiving data for 8S increases the average latency by $3.34\times$ for RPi's, and $2.57\times$ for Odroid boards (compared to the 2S-case). Again, this communication cost is very small compared to the cost incurred from directly splitting the single, large models. For instance, we have 43ms communication latency for distributing NoNN-8S on RPi's. In contrast, splitting the teacher on x86 incurs $1006 - 86 = 920$ms communication latency (see Section 5.3), which is $21\times$ higher than NoNN-8S.

Comparison of Latency for Distributed Inference w.r.t. a Model Compression Baseline

Next, we present a concrete, side-by-side comparison between the distributed inference latency incurred by our proposed NoNN and ATKD [135], a traditional model compression baseline. Note that, we compare the latency of NoNN against ATKD [135] as it achieves the best baseline accuracy results. Consequently, for this experiment, we use the ATKD model WRN40-2 (2.2M parameters, 95.03% accuracy on CIFAR-10 dataset) distilled from the same teacher model as above (*i.e.*, WRN40-4, 8.9M parameters, 95.49% accuracy). To achieve higher accuracy, the size of the model compressed via ATKD grows significantly and, therefore, it must be distributed across multiple resource-constrained devices as it may not fit within the given per-device memory budget (henceforth known as the *split-ATKD* experiment); similar to the split-teacher experiment above (see Section 5.3 and Table 5.8), ATKD baseline incurs communication at each intermediate layer. Since TVM generates a single binary file for the entire deep network, models compiled in TVM cannot be used to perform such split-teacher or split-ATKD experiments on RPi's (as we need to

Table 5.9: NoNN and Split-ATKD Latency on RPi's (Pytorch)

	Split-ATKD (WRN40-2)		NoNN-8S
	4 RPi's	8 RPi's	8 RPi's
Accuracy	95.03%	95.03%	95.02%
Parameters per device	550K	275K	430K
FLOPS per device	163M	82M	167M
Total latency per inference (s)	23	28.5	**0.85**
Speedup with NoNN	**27×**	**33×**	–

extract the intermediate outputs at each layer and communicate them across devices). Hence, for the following split-ATKD experiments, we install Pytorch 0.4.1 on all eight RPi's to run distributed inference for both split-ATKD and NoNN-8S.

Table 5.9 shows the accuracy, parameters per device, FLOPS executed per device, and total latency for NoNN-8S and split-ATKD models in pytorch on RPi's. We distribute the ATKD baseline in two different ways:

1. **ATKD split on four RPi's (split-ATKD-4S):** Each RPi for this case stores about 550K (2.2M/4) parameters and executes about 163M (655M/4) FLOPS. This experiment simulates the scenario where each device has, say, close to 500K-parameter memory-budget (similar to the 430K parameter-budget considered in NoNN for the CIFAR-10 dataset).
2. **ATKD split on eight RPi's (split-ATKD-8S):** Each RPi for this case stores around 275K (2.2M/8) parameters and executes merely 82M (655M/8) FLOPS. Similar to NoNN-8S, this case compares the performance of split-ATKD on eight separate devices.

Note that, the split-ATKD-4S case has slightly more parameters than the single NoNN-8S student (550K *vs.* 430K) but executes slightly fewer FLOPS than NoNN-8S (163M *vs.* 167M). On the other hand, the split-ATKD-8S case executes far fewer FLOPS than a single NoNN-8S student (82M *vs.* 167M) and also stores fewer parameters than NoNN-8S (275K vs. 430K). As evident from Table 5.9, despite the lower computation involved in both the split-ATKD cases, the heavy communication at each layer incurred by split-ATKD-4S (split-ATKD-8S) results in a total latency of 23s (28.5s). In contrast, NoNN-8S takes only 0.85s (in pytorch) to perform the distributed inference[4], which is 27× (33×) better than the split-ATKD-4S (split-ATKD-8S) baseline, while

[4]The total latency for distributed inference via NoNN-8S increases from 150ms in TVM (see Table 5.6) to 850ms in

achieving nearly the same accuracy. Hence, this result emphasizes the importance of our proposed NoNN for distributed deep learning inference on a network of IoT-devices.

Heterogeneous Devices

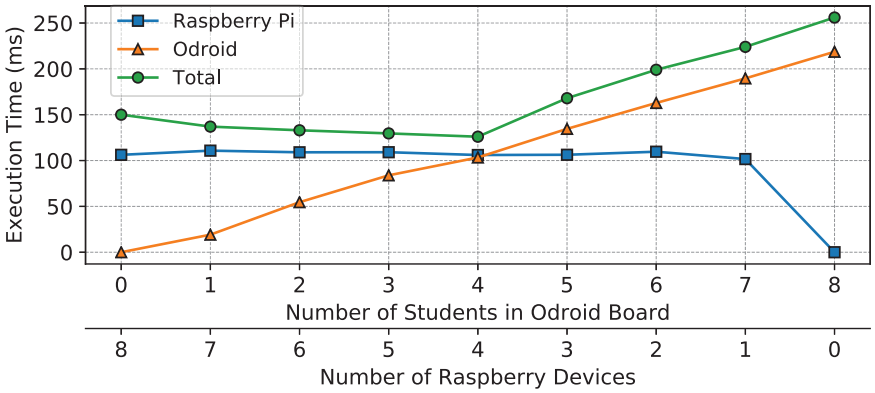
To demonstrate NoNN's flexibility, we perform the following experiment: we start with a homogeneous scenario for the RPi's, then, gradually move each student to an Odroid board, depicting an environment where we have several small devices (RPi's) and one slightly more powerful device (Odroid), the latter running on the big cores to maximize performance. Each small device (RPi) executes only one student, while the more powerful device (Odroid) executes multiple students.

Fig. 5.7a breaks down the latency into three parts: latency of RPi's, the single Odroid, and the *total latency* including the server. The total latency represents the execution time, starting when the server transmits the first image and ending when fc layer is applied (after receiving and concatenating results from all students). The RPi performance represents the latency (*i.e.*, both computation and communication time) for the *slowest* device.

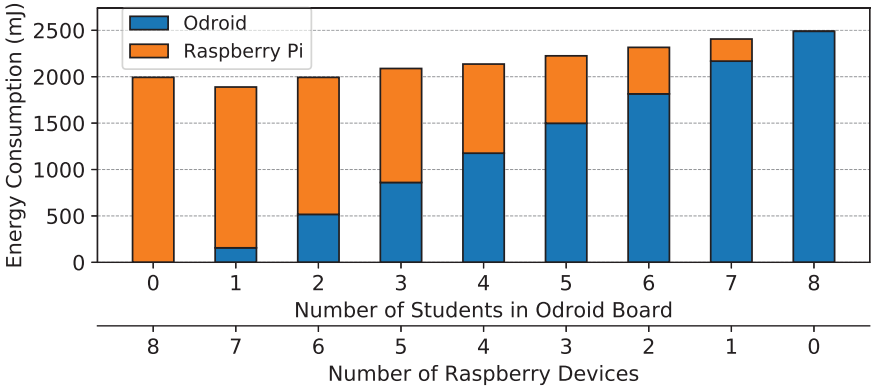
As the Odroids are considerably faster than the RPi's in the current setup, a single Odroid is able to execute four students in the same amount of time a RPi takes to compute one student (Fig. 5.7a). As the number of students in the Odroid increases even further, its latency becomes a bottleneck and the total latency increases. The difference between the total latency and the slowest device is the communication time between the server and the devices, which is 30.65ms on average. The server (x86 machine) takes 0.1ms to concatenate and apply the fc layer, which is negligible. Note that, until four students are deployed in Odroid, the total latency in Fig. 5.7a keeps on reducing since the server can process the Odroid results while the RPi's are still running the individual students. Hence, performing this task in parallel reduces the critical path.

Fig. 5.7b depicts the energy consumption for *all* RPi devices and the single Odroid device. As evident, we have a trade-off among performance, energy, and number of devices. For instance, when the Odroid runs just one student, the total energy gets minimized because Odroid is more energy-efficient than the RPi for the one student case. However, as we increase the workload for the Odroid (*i.e.*, number of students), the total energy increases even though the number of

pytorch. This shows that the device-level optimizations performed by TVM are important for deep learning inference on edge devices.



(a) Performance per Inference



(b) Energy Consumption per Inference

Figure 5.7: Performance and energy as the number of Raspberry devices is reduced and the workload for the Odroid board is increased, always executing a total of eight students.

devices (RPi's) decreases. In the current setup, inference on eight separate RPi's consumes the same amount of energy as using six RPi's plus two students on the Odroid. Comparing the two extreme cases (eight students in eight RPi's, or all eight students in a single Odroid), the Odroid consumes 25% more energy.

Therefore, this evaluation demonstrates three important aspects: (*i*) NoNN approach demonstrates good performance in both homogeneous and heterogeneous environments, (*ii*) In terms of energy, distributing the students on several devices instead of using a single more powerful device can reduce the total energy consumption for inference. (*iii*) Finally, to maximize the energy savings for the entire system, deploying NoNN on a combination of several devices can trade-off between energy dissipation and number of devices.

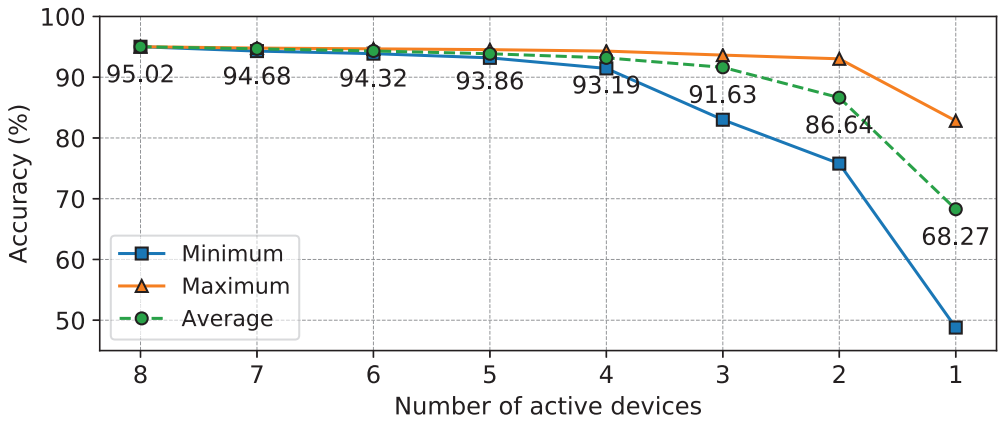


Figure 5.8: Accuracy as some devices become unavailable due to device failures (*e.g.*, due to processor or network failures, and battery depletion). NoNN achieves more than 91% accuracy, on average, when only three devices are available. When four devices are available, NoNN achieves more than 93% accuracy.

Robustness of NoNN

We now evaluate a scenario where some devices may become unavailable due to a variety of reasons such as power depletion or network failure. We evaluate all possible cases in which such devices may fail. For instance, for six active devices, any two devices may fail, resulting in 28 possible scenarios, and so on for the remaining cases.

Fig. 5.8 shows the accuracy drop as we remove some devices. On an average, NoNN is still able to achieve more than 91% accuracy when only three devices are available. Moreover, if a minimum of four devices are available, the average accuracy is more than 93%. As more devices become unavailable, the difference between the minimum and the maximum accuracy increases since the final decision depends on output from all devices. When only a few devices are available, the final result depends on the characteristics of each student, some being able to deliver higher accuracy than others. For instance, one of the students is able to achieve 82.82% accuracy alone (*i.e.*, the maximum point when only one student is active in Fig. 5.8). Therefore, deploying this student on the primary device (which will always be available) will guarantee at least 82.82% accuracy. Other devices can then be used to further improve the accuracy of the system. Hence, NoNN is robust to random device failures. Of note, robustness can also be formulated as part of the partitioning problem itself; this, however, is left as a future work.

5.7 Summary

In this chapter, we have proposed a new NoNN paradigm to compress a large teacher model into multiple highly-compressed student modules that can be distributed across a network of edge devices with minimal communication overhead. To this end, we have first proposed a network science-based partitioning of teacher's knowledge, and then trained individual students on the corresponding partitions. With extensive experiments, we have demonstrated that NoNN achieves close to teacher's accuracy with significantly lower memory ($2.5\times$-$24\times$ gain w.r.t. teacher) and computation ($2\times$-$15\times$ fewer FLOPS w.r.t. teacher), while guaranteeing that individual modules of NoNN fit within some given memory/FLOP budget. We have also shown that NoNN achieves higher accuracy than several baselines with no communication among students until the last layer. Finally, we have deployed the proposed models for CIFAR-10 dataset on Raspberry Pi and Odroid, and have demonstrated $6.22\times$-$12.22\times$ improvement in performance and $12.99\times$-$14.36\times$ in energy per node w.r.t. teacher. We have further shown that NoNN model results in up to $33\times$ reduction in total latency for distributed inference on multiple edge devices w.r.t. a state-of-the-art model compression baseline. Hence, the proposed communication-aware model compression can ultimately lead to effective deployment of deep networks on multiple memory-constrained IoT-devices.

This completes our proposed network science-based model compression. We demonstrated that network science can be used to target new directions in model compression by actually targeting a network of devices (instead of model compression for a single device like in the prior art). In the current work, the individual compressed student architectures were selected such that they satisfy certain resource-constraints. An important question then is *what characteristics result in efficient yet highly accurate architectures?* We exploit network science to address this critical question in the next chapter.

Chapter 6

Network Science for Neural Architecture Space Exploration: Theory and Practice

Given a large Convolutional Neural Network (CNN) which achieves high accuracy, can we *directly* design a significantly compressed model with minimal loss of accuracy over the initial model? We believe the above question is one of the most critical problems to address in the modern deep learning research. So far, deep networks have been compressed using model compression techniques such as (*i*) Pruning [63, 132], (*ii*) Quantization [51, 60], (*iii*) Knowledge Distillation (KD) [15, 43, 135], or via (*iv*) Manually designed efficient networks and convolutions [47, 103, 137], and (*v*) Efficient models resulting from automatic Neural Architecture Search (NAS) methods [66, 97, 100, 128, 141, 142]. However, all of the above directions do not answer the following fundamental research question: *Are there any characteristics of a CNN architecture that can indicate a priori (i.e., without training) which family of models (with different number of parameters and layers) achieve similar accuracy?* Indeed, answering this question can enable new model compression methods where we can directly design novel architectures to reduce the number of parameters without losing significant accuracy.

In this chapter, we propose an architecture-level metric called *NN-Mass* to model deep learning architectures from a complex networks perspective. We first demonstrate a theoretical relationship between NN-Mass and generalization of CNN architectures. Then, we conduct extensive empirical validation of our ideas by exploiting the proposed metrics for *Neural Architecture Space Exploration*. These new insights enable us to directly design compressed models which reduce parameters by up to $3\times$, while losing minimal accuracy compared to the initial, large CNN (*e.g.*, the compressed model reaches 96.82% test accuracy *vs.* $\sim 97\%$ for large CNN on CIFAR-10).

6.1 Towards Neural Architecture Space Exploration

Are there any characteristics of a CNN *architecture* that can indicate *a priori* (*i.e.*, without training) which family of models achieve similar (test) accuracy, despite having vastly different number of parameters or layers? Even though, there has been significant progress in architecture design practices (both manual [41, 47, 50] as well as automated via Neural Architecture Search (NAS) [66, 100, 142]), the above question remains unanswered, thereby making it one of the most fundamental problems in modern deep learning research. Clearly, answering this question can help us *directly design* efficient CNN architectures. The above question is related to three important areas:

Neural Architecture Search (NAS). Search for *efficient* architectures that result in high accuracy [66, 100, 142]. The models designed by recent NAS algorithms usually surpass manually designed architectures [41, 47, 50]).

Model Compression. Reduce the computational costs of existing deep networks without losing significant accuracy. Towards this, the existing model compression techniques mainly focus on Pruning, Quantization, and Knowledge Distillation (KD) [15, 43, 51, 60, 63, 132, 135].

Generalization of deep networks. Traditional wisdom suggests that if a model has high-enough capacity (*e.g.*, when the number of parameters is much larger than the number of samples in the dataset, which happens in practice), it would lead to overfitting (*i.e.*, very low training error and high test error). However, deep networks achieve low generalization error despite having a large number of parameters. Hence, generalization aims to *theoretically* understand why deep networks work well in practice by exploring properties of weight-norms/initializations, stability of deep networks to noise, optimization characteristics such as sharpness of the minima, *etc.* [3, 8, 89, 90, 91, 136].

Although there has been extensive research in the above three areas separately, to our knowledge, there is no research at the intersection of all three directions. Specifically, while NAS has been used to generate efficient architectures for mobile applications [20, 115, 127], existing NAS research does not theoretically explain *why* the discovered architecture performs better than other models containing similar number of parameters. Conversely, NAS also does *not theoretically explain why architectures with significantly different number of parameters sometimes achieve similar accuracy*. Similarly, a few generalization studies (*e.g.*, [3]) exploit compression to prove

generalization error bounds. However, none of the generalization studies provide any theory that can explicitly guide efficient architecture design.

In view of the above, in this chapter, we propose a new, *theoretically-grounded*, architecture-level metric called *NN-Mass*. We first model a CNN as a complex network and then derive NN-Mass from a network science perspective. Specifically, since a CNN is a set of channels connected via filters at each convolutional layer, we can express any CNN as a network of channels (see Fig. 6.1(a)). Then, we define *NN-Density* to quantify how densely the channels of a CNN are connected to each other. Next, we exploit NN-Density to define our proposed NN-Mass metric that quantifies the link between CNN architectures and their generalization capabilities. We use our proposed metrics for *Neural Architecture Space Exploration (NASE)*. NASE is defined as the process of studying the design space of deep networks via theoretically-grounded metrics such as NN-Mass.

To this end, we combine, for the very first time, Probably Approximately Correct (PAC)-Bayes theory of generalization [74, 75] with network science [79, 87, 123] to theoretically prove that architectures with higher NN-Mass achieve lower generalization error. We further theoretically demonstrate that models with similar NN-Mass lead to similar test accuracy. Then, we provide extensive empirical evidence towards these theoretical insights. Towards this, we empirically demonstrate that models with similar NN-Mass indeed achieve comparable accuracy despite having vastly different number of parameters and layers (see Fig. 6.1(b)). Finally, given a large, highly accurate CNN, we demonstrate how NN-Mass can be used to directly design efficient models without sacrificing significant accuracy. Hence, our proposed metric can be used to directly discover a family of efficient models without training individual models (see Fig. 6.1(c)). Of note, given the complexity of deep learning systems such as CNNs, an alternative, *system-level* approach is imperative for understanding the architectural properties of enormous deep networks. This is why we systematically model deep networks from a network science perspective. Our complete approach is shown in Fig. 6.1.

Overall, we make the following **key contributions** in both theory and practice:

1. **Modeling CNNs as complex networks and Neural Architecture Space Exploration.** To our knowledge, we are the first to exploit network science for *theoretically* studying the properties of the architecture design space. Next, we exploit network science to build novel

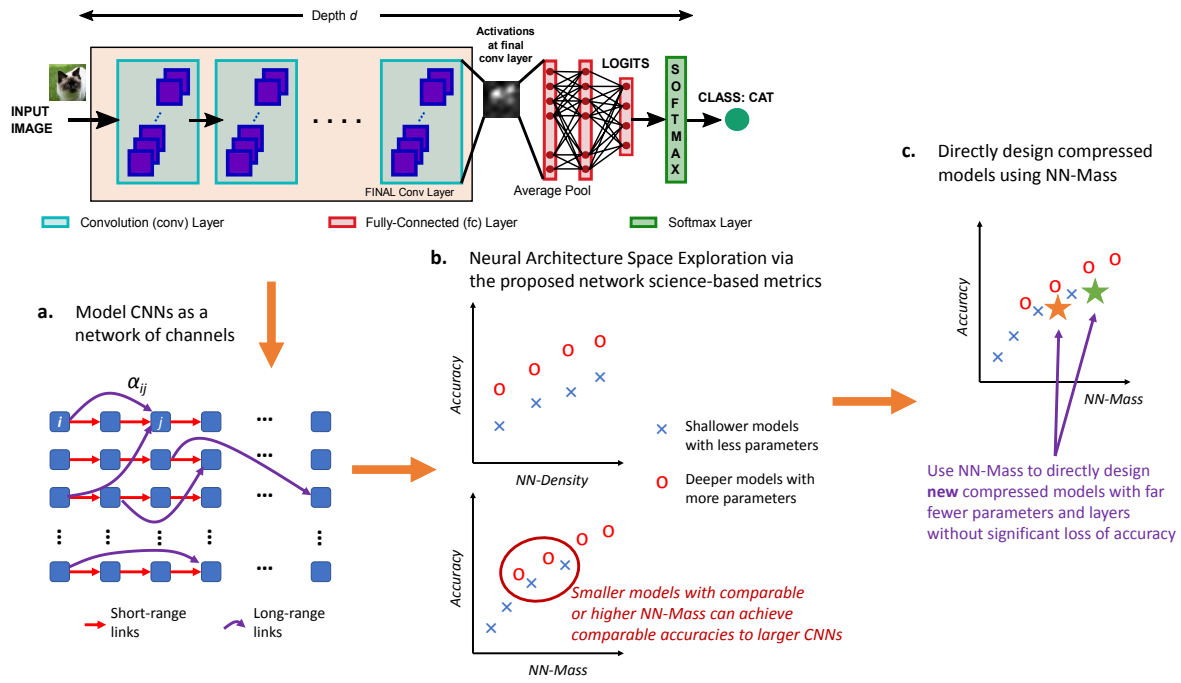


Figure 6.1: Complete flow of our approach: (a) First model a given CNN as a network of channels. Each layer can get contributions from the last layer (short-range links), as well as from all other previous layers (long-range links). (b) Next, we exploit network science to propose NN-Mass and NN-Density, where NN-Mass is a theoretically-grounded metric that can indicate generalization capability. (c) Since NN-Mass can indicate which smaller models can obtain comparable accuracy to large CNNs, we can exploit it to directly design significantly compressed models.

and intuitive metrics for NASE. Specifically, we define *NN-Mass* to quantify the link between various architectures and their generalization capabilities.

2. **Theory for Modeling Architectural Aspects of Generalization.** We merge, for the very first time, the PAC-Bayes theory with the network theory to demonstrate provable relationship between NN-Mass and generalization of CNN architectures. We show two key properties of NN-Mass: (*i*) For a given depth and width, a CNN with higher mass results in lower generalization error, and (*ii*) Irrespective of total number of parameters and depth but same width, models with similar NN-Mass yield similar generalization performance.

3. **Validation on real world image classification tasks.** We further provide extensive empirical evidence for the above theoretical insights. Towards this, we conduct experiments with CNNs of different depths, parameters, and long-range links for CIFAR-10/100 datasets. We found that models with fewer layers and fewer parameters often achieve comparable accu-

racy to deeper networks with more number of parameters. We also observed that models with similar NN-Mass often achieve similar test accuracy even though the number of parameters are vastly different. To quantify the relationship between NN-Mass and generalization, we fit a linear model and demonstrate that the goodness-of-fit parameter R^2 achieves high values (*e.g.*, between 0.74-0.90). We also show that NN-Mass can be used to predict the test accuracy of the models which were never trained before.

4. **Model Compression via NN-Mass** Finally, we demonstrate that we can design a highly-compressed model with minimal loss of accuracy over a large CNN. Specifically, given a large CNN which achieves high accuracy (*e.g.*, $\sim 97\%$ on CIFAR-10), we directly use our proposed NN-Mass metric to design new models with significantly fewer parameters and layers, but with NN-Mass close to that of the large model. We show that the accuracy of this newly designed model reaches close to that of the large model (*e.g.*, 96.82% on CIFAR-10 test set) while reducing the total parameters and FLOPS by more than $3\times$. Of note, since we do not use pruning, quantization or KD to reduce the model-size, NN-Mass can be seen as a new way of model compression, directly from the architecture standpoint.

The rest of the chapter is organized as follows: Section 6.2 covers related work on model compression, NAS and generalization, while Section 6.3 describes our proposed network science-based NASE approach including the theoretical relationship between CNN architectures and their generalization performance. Next, Section 6.4 presents extensive experimental results and their analysis. Finally, the chapter is concluded in Section 6.5.

6.2 Related Work

We now discuss the related work on model compression, NAS, and generalization of deep networks. We also discuss prior work on long-range links in deep learning and network science.

6.2.1 Model Compression

As explained earlier, model compression techniques can be broadly classified into pruning [63, 132], quantization [51, 60], and KD [15, 43, 135]. Pruning aims to remove redundant and/or

useless weights from a trained deep network, whereas quantization reduces the number of bits required to represent weights and activations. On the other hand, KD aims to train a significantly compressed student model to mimic a large teacher network (which we want to compress).

Although all of these techniques are effective at reducing parameters and computation without losing significant accuracy, most of the prior art does not address what *architectural characteristics* make the compressed models nearly as accurate as the original, large model. For instance, quantization does not talk about architecture since it does not lead to a different architecture. Moreover, although student models in KD can result in significantly different architecture (with different layers and width), designing such compressed students is usually a manual process. Hence, KD also does not shed any light on which student models might lead to better accuracy. Further, even though pruning can also result in novel architectures, the pruning techniques do not completely remove entire layers and, therefore, do not result in compressed models with significantly fewer layers. Finally, unlike our proposed work, existing model compression literature does not give architecture-level *metrics* that can allow direct identification of highly-accurate models.

Note that, a few works attempt to understand the relationship between training hyper-parameters of deep networks and the architecture. For instance, Frankle *et al.* analyze the impact of initialization on pruned networks in their lottery ticket hypothesis [28]. Similarly, Park *et al.* empirically understand the relationship between batch size, learning rate and network width [95]. However, to the best of our knowledge, complex network characteristics have not been exploited to model deep networks explicitly. Moreover, architecture-level metrics – that relate number of parameters, depth, width and long-range links – are not addressed in prior art to indicate the generalization power of CNNs.

6.2.2 Neural Architecture Search (NAS)

The process of designing deep networks that perform very well in practice has been largely driven by trial-and-error [41, 47, 50, 58, 108] or via recently proposed NAS algorithms [66, 97, 100, 128, 141, 142]. More recently, standard network-generation models such as Barabasi-Albert (BA) model [6] or Watts-Strogatz (WS) model [123] were exploited for NAS [129]. Moreover, a framework to discover dynamic network wiring has also been proposed [126]. However, like the rest of the NAS research, [126, 129] did not explore what specific theoretical characteristics of the

architecture make different models (with different number of parameters) achieve similar generalization performance. Hence, to our knowledge, no theoretical attempt has been made to better understand the link between architecture design and generalization. On the other hand, our objective is twofold: (*i*) *Theoretically* understand architectural aspects of generalization, and (*ii*) *Practically* design efficient architectures without searching for them (*e.g.*, by directly exploiting our proposed metric).

It has been further shown that random search is competitive to NAS [64]; this has been attributed to the carefully designed search space for NAS such as depth-wise separable convolutions, dilated convolutions, *etc* [66]. In contrast, our work only uses the standard convolutions and concatenation-type long-range links and none of the specialized convolutions which are known to reduce parameters and improve accuracy. Therefore, we demonstrate that our designed architectures achieve state-of-the-art accuracy without *any* carefully selected search space, and that regular convolutions can be just as effective. This highlights the importance of efficient *architectures* that can lead to high accuracy with less parameters.

6.2.3 Generalization of deep networks

The field of generalization has recently gained attention to understand why deep learning works well in practice [3, 8, 18, 65, 89, 90, 91, 92, 104]. However, these generalization studies either look at the loss landscapes of deep networks and properties of Stochastic Gradient Descent algorithm, or analyze the norms and spectral properties of model weights. Specifically, existing generalization studies explore why deep networks generalize well on real data despite having sufficient capacity to overfit.

Traditional machine learning theory indicates that if the complexity of a model is high, it can lead to overfitting (*i.e.*, the model can fit perfectly on training data but cannot generalize well on the testing data). To test this traditional theory, Zhang *et al.* conducted several kinds of experiments [136] such as (*i*) Randomizing labels, (*ii*) Randomizing pixels. The idea is to fit a deep network to such randomized datasets. Many other variants of such experiments have been conducted (*e.g.*, taking the negative of images, adding noise to weights, activations, inputs, *etc.* [3]). The objective in these experiments is to demonstrate that deep networks easily fit such randomized datasets, thus, showing that the networks have enough capacity to overfit (since test error on cor-

rect labels would clearly be very low in such scenarios). Consequently, the generalization papers attempt to understand why deep networks generalize on real datasets (even though they can overfit on random data). Hence, generalization does *not* explicitly study what characteristics make good deep network *architectures*.

In contrast, the main objective of our work is to specifically identify what architectural characteristics of deep CNNs lead to better generalization and how we can use those characteristics to directly design efficient models. By integrating the PAC-Bayes theory with the theory of small-world networks for the first time, we further present a new way of modeling generalization of complex CNN architectures. Hence, our contributions are completely orthogonal to the existing research: Prior art seeks answers about overfitting, whereas we seek answers about the contribution of the architecture itself towards generalization.

6.2.4 Long-range links in CNNs and Network Science

Many recent innovations in deep learning architecture design have resulted in state-of-the-art deep networks that achieve excellent prediction accuracy on complex vision and natural language applications. One of the most important innovations is the concept of *shortcut connections* in deep networks which enable complex architectures and have pushed the accuracy far beyond the traditional CNNs. For instance, Resnets [41] introduced residual blocks which add feature maps at alternate convolution layers. Similarly Densenets [50] have dense blocks that contain all-to-all connections. In contrast to Resnets, Densenets do not add feature maps but instead concatenate them together. The core idea in both of these models is to improve the information flow through the deep networks with the help of such shortcut connections between various layers.

Indeed, shortcut connections have long been a subject of study in the field of network science, *e.g.*, small world networks [124] that specifically deal with networks with long-range and short-range links (*e.g.*, in social, biological, transportation networks [85], and even multicore networks [93]). In this chapter, we view the shortcut connections in deep networks as being analogous to the long-range links in generic networks such as social and transportation networks. Hence, network science can be a good choice for studying the dynamics of long-range links in deep networks. Since Densenets have been shown to achieve higher accuracy than Resnets [50], we focus on concatenation-type long-range links.

This completes the related work. Next, we model CNNs from a network science perspective and also define our proposed NN-Mass and NN-Density metrics.

6.3 Proposed Approach

In this section, we first explain how CNNs can be modeled via network science. Then, we mathematically derive the proposed NN-Mass and NN-Density metrics that are needed for architecture design space exploration. Next, we demonstrate the theoretical relationship between NN-Mass and generalization of CNN architectures. Finally, we describe how NN-Mass can be exploited for model compression.

6.3.1 Modeling CNNs via Network Science

In the present work, we assume that CNN consists of multiple cells, with each cell consisting of a fixed number of convolutional layers (similar to existing works such as Densenets [50], Resnets [41], Wide-Resnets [134]). As shown in Fig. 6.2(a), each cell can have a different width (*i.e.*, the number of channels per layer). Following the standard practice [41, 50, 108, 134], the width is increased by a factor of two at each cell as the feature map is reduced to half (see Fig. 6.2(a)). After the convolutions, the final feature map is average-pooled and passed through a fully-connected layer to generate logits.

We now illustrate a single convolution layer of our proposed model in Fig. 6.2(b). Note that, in a standard CNN, a convolutional layer with n input channels and m output channel consists of m filters, each with $[k \times k \times n]$ dimensions. That is, as shown in Fig. 6.2(b), red kernel in the filter convolves with red input channel, green filter convolves with green input channel, and so on. The output of all such *channel-wise convolutions* are added together to obtain a *single* output channel (*e.g.*, violet output channel in Fig. 6.2(b)). However, this implicitly assumes that all input channels contribute *equally* to all output channels. Clearly, this assumption is counter-intuitive since our overall objective in an image classification problem is to separate out the identifying features of various classes at the final convolution layer (*i.e.*, output channels at the final layer must activate for *different* features). Therefore, adding the outputs of channel-wise convolutions at each intermediate layer can make the training process of CNN inherently hard.

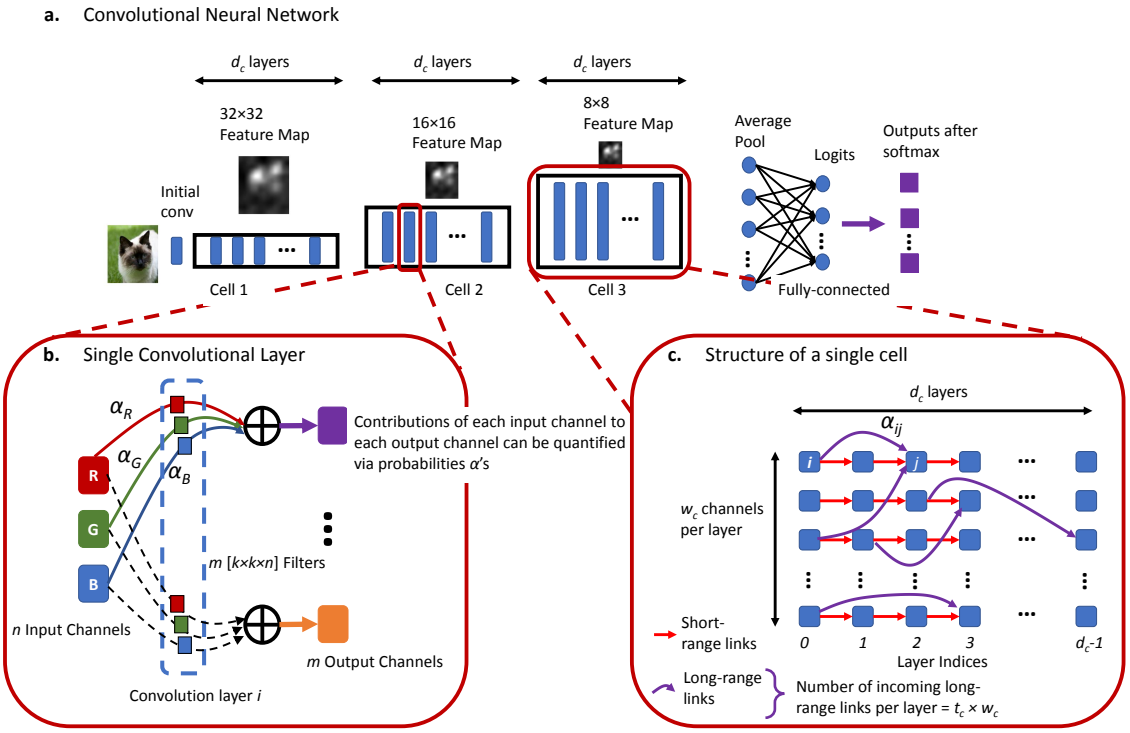


Figure 6.2: (a) CNN consisting of three cells of d_c layers each. (b) Each convolutional layer in a cell contains m filters of size $[k \times k \times n]$, where m (n) is the number of output (input) channels. Each input channel contributes to a given output channel with a probability α. In a traditional CNNs, all input channels contribute equally to all output channels, *i.e.*, $\alpha = 1$ for all input-output channel pairs. (c) Each cell contains d_c layers with w_c channels per layer. Output channels at each layer get contributions from output channels of last layer and additional long-range links from previous layers (via concatenation of feature maps). For simplicity, not all links are shown. Contribution of an input channel i towards an output channel j is given by probability α_{ij}. These probabilities are used to define the weighted adjacency matrix of the CNN.

To alleviate the above problem, we explicitly assign *different* contributions from each input channel i to each output channel j as probabilities α_{ij}. Specifically, for each output channel, we create a vector $\boldsymbol{q_j} = \{q_{1j}, q_{2j}, \ldots, q_{nj}\}$ with Kaiming-normal initialization [40], where each element q_{ij} of this vector denotes an unnormalized contribution from an input channel i to the given output channel j. Then, to generate the probabilities α_{ij}, we simply compute the softmax of $\boldsymbol{q_j}$. The probabilities thus obtained are used as contributions from input channels to this output channel (*e.g.*, $\{\alpha_R, \alpha_G, \alpha_B\}$ in Fig. 6.2(b)). We call the unnormalized weight vector $\boldsymbol{q_j}$ as *contribution-weights*, and the probabilities α_{ij}'s as *contribution-probabilities* throughout this chapter. Hence, in contrast to a standard convolution where individual channel-wise convolutions are directly added

to obtain one output channel (*i.e.*, in a traditional CNN, $\alpha_{ij} = 1$ for all connections), the final convolution in our proposed model is computed as a weighted sum of channel-wise convolutions (where, weights are given by α_{ij}). Of note, throughout the training as well as inference, we keep these probabilities fixed.

Next, we define the structure of individual cells in our proposed model. Fig. 6.2(c) shows a cell with d_c convolution layers, and w_c channels per layer. Output channels at each layer i receive contribution from all output channels of layer $i-1$; we call these contributions *short-range links* since they connect consecutive layers (see red links in Fig. 6.2(c))[1]. In addition to short-range links, output channels at layer i can also receive *long-range* contributions from t_c channels present at layers $l \in \{0, 1, \ldots, i-2\}$ within the given cell. By definition, a link is long-range if it connects channels across two or more layers. In practice, to create long-range links, the feature maps from previous layers are concatenated at the current convolution layer (similar to Densenets [50]). Of note, in our current setup, long-range links are confined only within the cell and do not extend between different cells.

To create long-range links at each layer i within a given cell, we *randomly* select $\min\{w_c(i-1), t_c\}$ previous channels (out of all channels until layer $i-2$). As it has been established in recent NAS research [64], our rationale behind selecting random links is that random architectures are often competitive to carefully designed models. Hence, throughout the chapter, we look only at architectures with randomly chosen long-range links (after fixing the random seed). Our experiments further reinforce the observation that random architectures are quite competitive to state-of-the-art models.

Note that, *all* channel contributions (both long-range and short-range) are quantified by probabilities α_{ij}'s. Specifically, instead of defining the contribution-weight vectors ($\boldsymbol{q_i}$) for each output channel, we can directly initialize a contribution-weight matrix $Q = [\boldsymbol{q_1}^T, \boldsymbol{q_2}^T, \ldots, \boldsymbol{q_n}^T]$ for all channels in a cell. Then, the contribution-probabilities α_{ij}'s are obtained by taking column-wise softmax of Q. Therefore, the contribution-probabilities and the cell structure can be directly used to define an adjacency matrix of a CNN: $\mathcal{A}_{ij} = \alpha_{ij}$, $i, j \in \{0, 1, 2, \ldots, w_c \cdot d_c\}$. Hence, we can

[1]Not all links are shown in Fig. 6.2(c) for simplicity. In fact, all output channels at layer i will receive short-range contributions from all channels at layer $(i-1)$, and all output channels at layer i will receive long-range contributions from $\min\{w_c(i-1), t_c\}$ channels.

now represent a CNN as a network of channels connected via convolutions. Next, we exploit ideas from network science to propose two new metrics for architecture space exploration, *i.e.*, the mass and density of CNNs.

6.3.2 Neural Architecture Space Exploration: Mass and Density of CNNs

We now exploit the above network formulation to systematically study the architecture space for deep networks. Specifically, our problem has following objectives:

1. Theoretically quantify the architectural aspects of the generalization problem.
2. Exploit the above theory to *directly* design efficient architectures in practice.

To address the above goals, we propose two new network science-based metrics called NN-Mass and NN-Density, as defined below.

Definition 1 (Density of Deep Networks)**.** *Density of a CNN quantifies how densely a given neural network is connected in terms of long-range links. Formally, for a given cell c, density ρ_c is expressed as follows:*

$$\rho_c = \frac{\textit{Number of long-range links in cell } c}{\textit{Total possible long-range links in cell } c} = \frac{l_c}{L} \tag{6.1}$$

Note that, the maximum number of channels contributing long-range links at each layer in cell c is given by t_c. Also, for a layer i, possible candidates for long-range links = all channels up to layer $(i-2) = w_c(i-1)$ (see Fig. 6.2(c)). Indeed, if t_c is sufficiently large, initial few layers may not have t_c channels that can supply long-range links. For these layers, we use all available channels for long-range links. Therefore, for a given layer i, number of long-range links (l_i) is given by:

$$l_i = \begin{cases} w_c(i-1) \times w_c & \text{if } t_c > w_c(i-1) \\ t_c \times w_c & \text{otherwise} \end{cases} \tag{6.2}$$

where, both cases have been multiplied by w_c because once the channels are selected to supply long-range links, they supply long-range links to all w_c channels at the current layer i. Hence, for an entire cell, total number of channels contributing long-range links (l_c) is as follows:

$$l_c = w_c \sum_{i=2}^{d_c-1} \min\{w_c(i-1), t_c\} \tag{6.3}$$

On the other hand, total number of possible long-range links within a cell (L) is simply the sum of possible candidates at each layer:

$$\begin{aligned} L &= \sum_{i=2}^{d_c-1} w_c(i-1) \times w_c \\ &= w_c^2 \sum_{i=2}^{d_c-1} (i-1) \\ &= w_c^2[1+2+\ldots+(d_c-2)] \\ &= \frac{w_c^2(d_c-1)(d_c-2)}{2} \end{aligned} \tag{6.4}$$

Using Eq. (6.3) and Eq. (6.4), we can rewrite Eq. (6.1) as:

$$\rho_c = \frac{2\sum_{i=2}^{d_c-1} \min\{w_c(i-1), t_c\}}{w_c(d_c-1)(d_c-2)} \tag{6.5}$$

Finally, we use the density for individual cells to define the average density of a CNN (ρ_{avg}). Say, we have N_c cells in the complete CNN. Then, average density ρ_{avg} is given as follows:

$$\rho_{avg} = \frac{1}{N_c}\sum_{c=1}^{N_c} \rho_c = \frac{1}{N_c}\sum_{c=1}^{N_c} \frac{2\sum_{i=2}^{d_c-1} \min\{w_c(i-1), t_c\}}{w_c(d_c-1)(d_c-2)} \tag{6.6}$$

This completes the derivation for average density of a CNN. We will use the term NN-Density to denote average density throughout this chapter. Next, we derive the NN-Mass. Both of these metrics will be used to explore the architecture space of a CNN.

Definition 2 (Mass of Deep Networks)**.** *We define NN-Mass to reflect the representational capacity of a CNN. Intuitively, for a given width, models with same NN-Mass but different depth, long-range links and number of parameters should achieve similar accuracy.*

Now, recall that density is basically mass divided by volume. Analogously, we can derive NN-Mass of a CNN by multiplying the density with total number of channels in each cell. Hence, the NN-Mass (m) is given by:

$$\begin{aligned} m &= \sum_{c=1}^{N_c} w_c d_c \rho_c \\ &= \sum_{c=1}^{N_c} w_c d_c \frac{2\sum_{i=2}^{d_c-1} \min\{w_c(i-1), t_c\}}{w_c(d_c-1)(d_c-2)} \\ &= \sum_{c=1}^{N_c} \frac{2d_c\sum_{i=2}^{d_c-1} \min\{w_c(i-1), t_c\}}{(d_c-1)(d_c-2)} \end{aligned} \tag{6.7}$$

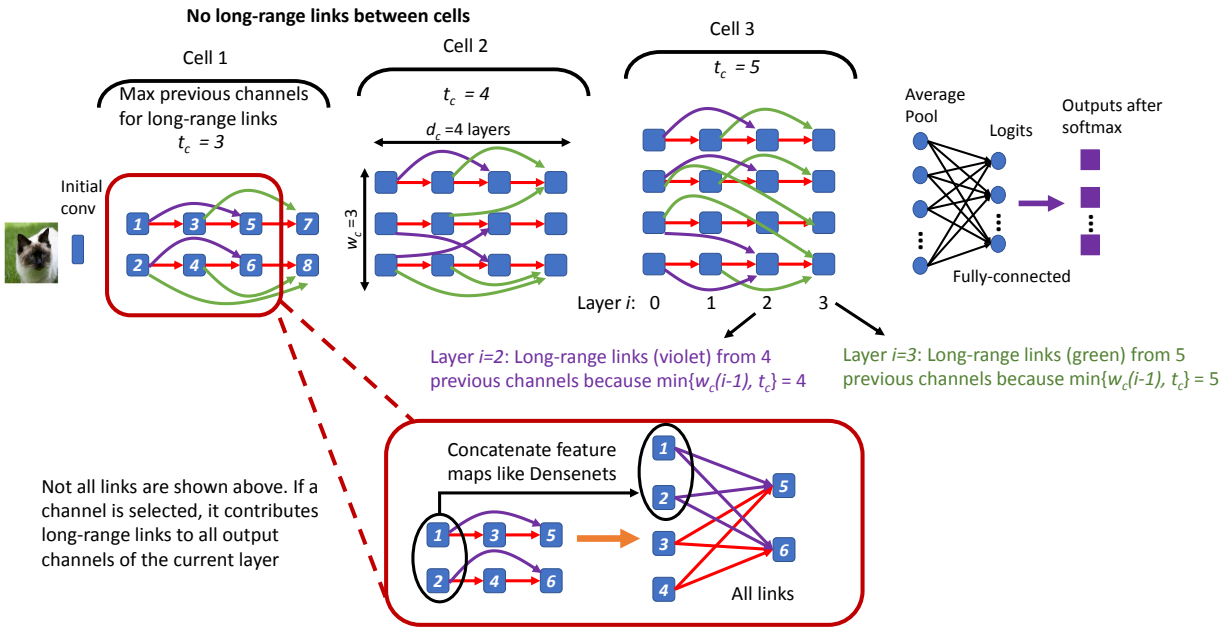


Figure 6.3: An example CNN to calculate NN-Density and NN-Mass. Not all links are shown in the main figure for simplicity. The inset shows the contribution from all long-range and short-range links: The feature maps for randomly selected channels are concatenated at the current layer (similar to Densenets [50]). At each layer in a given cell, the maximum number of channels that can contribute long-range links is given by t_c.

Both NN-Density and NN-Mass relate network width, depth, and number of long-range links. Now that we have derived both of these metrics, we next present a concrete example to illustrate how NN-Mass and NN-Density can be calculated for a given architecture.

Example 1. Given a CNN architecture shown in Fig. 6.3, we now calculate its NN-Density and NN-Mass. This CNN consists of three cells, each containing $d_c = 4$ convolutional layers. The three cells have a width, (*i.e.*, the number of channels per layer) of 2, 3, and 4, respectively. We denote the network width as $w_c = [2, 3, 4]$. Finally, the maximum number of channels that can supply long-range links is given by $t_c = [3, 4, 5]$. That is, first cell can have a maximum of three long-range links per layer, second cell can have a maximum of four long-range links per layer, and so on. Note that, the contribution-probabilities ($\alpha_i j$'s) are computed using the procedure in Section 6.3.1 (*i.e.*, we first generate contribution-weights via Kaiming initialization, and then take the softmax to get probabilities). Moreover, as mentioned before, we randomly choose $\min\{w_c(i-1), t_c\}$ channels for long-range links at each layer. The inset of Fig. 6.3 shows how long-range links are created by concatenating the feature maps from previous layers.

Hence, using $d_c = 4$, $w_c = [2, 3, 4]$, and $t_c = [3, 4, 5]$ for each cell c, we can directly use Eq. (6.6) and Eq. (6.7) to compute the NN-Density and NN-Mass values. Putting the values in the equations, we obtain $\rho_{avg} = 0.78$ and $m = 28$.

Both NN-Density and NN-Mass relate network width, depth, and number of long-range links. For a fixed number of cells, a CNN architecture can be completely specified by {depth per cell, width per cell, maximum long-range link candidates per cell} = $\{d_c, w_c, t_c\}$. Hence, to explore the architecture space of the CNNs (henceforth called the *Neural Architecture Space Exploration* or NASE), we vary $\{d_c, w_c, t_c\}$ to obtain random architectures with varying NN-Density and NN-Mass values. We next show the theoretical link between NN-Mass (*i.e.*, a property of CNN architectures) and generalization.

6.3.3 Provable Relationship between NN-Mass and Generalization

In this section, we theoretically prove the relationship between generalization and our proposed NN-Mass metric, a property of CNN architectures. We further show that models with similar NN-Mass values achieve similar test accuracy. To this end, we start with the PAC-Bayes theory which is used to bound generalization error of a given (not necessarily a neural network-based) classifier. We will also integrate the network theory with PAC-Bayes theory to derive our results.

Theorem 1 (McAllester Bound for Generalization Error [61, 74, 75])**.** *Given any data generating distribution $\mathcal{D}$, any hypothesis class of predictors $\mathcal{H}$, any prior distribution P over the predictors, and any $\delta \in (0, 1]$, with probability at least $1 - \delta$ and for all distributions Q over $\mathcal{H}$ for a randomly drawn training set $\mathcal{S}$ of N examples, the generalization bound is given by:*

$$\mathbb{E}_{\mathcal{D}}(L(f_Q)) \leq \mathbb{E}_{\mathcal{S}}(\hat{L}(f_Q)) + \sqrt{\frac{KL(Q||P) + \log \frac{2\sqrt{N}}{\delta}}{2N}},$$

where, f_Q is a classifier drawn from Q, $\mathbb{E}_{\mathcal{D}}(L(f_Q))$ denotes the expected error, $\mathbb{E}_{\mathcal{S}}(\hat{L}(f_Q))$ is the empirical error over the training set $\mathcal{S}$, and $KL(Q||P)$ denotes the Kullback-Leibler (KL) divergence between the distributions P and Q.

The above PAC-Bayes theorem bounds the generalization error of any classifier. Next, we prove generalization bounds for CNN architectures with long-range links in terms of NN-Mass. For simplicity, we will assume for the rest of this section that the CNN architecture consists of a

single cell containing d_c convolutional layers, each containing w_c channels (*i.e.*, the width of the CNN is w_c).

Theorem 2 (Generalization Bound for CNN Architectures in terms of NN-Mass)**.** *Consider single-cell CNN models with d_c layers and w_c channels per layer. Also, let t_c be the number of channels contributing long-range links. If Q denotes a probability distribution over CNNs with long-range links and an architecture $f_Q \sim Q$ has a NN-Mass of m, then, with probability at least $1-\delta$ the generalization error for this architecture is bounded as follows:*

$$\mathbb{E}_{\mathcal{D}}(L(f_Q)) \leq \mathbb{E}_{\mathcal{S}}(\hat{L}(f_Q)) + \sqrt{\frac{\frac{1}{2\sigma}(\frac{1}{6m} - \frac{1}{2}\log(\pi e m)) + \log\frac{2\sqrt{N}}{\delta}}{2N}},$$

where, σ is the maximum number of connections a channel can have in a given CNN architecture.

Proof. Note that, to prove the above theorem, it only suffices to express the KL-divergence term in Theorem 1 as a function of m. Consequently, we must model the distributions Q and P for various CNNs. Since we are dealing with the hypothesis class of CNN *architectures*, network science can be used to explicitly model the *distributions of the CNN topology*. Hence, in order to study the distributions over CNN architectural topology (*i.e.*, how various channels are connected together), network science concepts such as random networks, small-world networks, *etc.*, can provide valuable insights. Towards this, *the problem of computing distributions over CNN architectures can be equivalently seen as the problem of modeling connectivity in various network topologies*. The connectivity of a network can be quantified using its degree distribution (recall that degree of a node in a network is given by total number of links connected to it). Hence, degree distribution can be used for modeling the topology of CNN architectures.

Starting from the above ideas, we assume *a priori* that our CNN architecture does *not* have any long-range links. In other words, our prior distribution P for CNN architecture consists only of models with short-range links and no shortcut connections[2]. Therefore, any architecture drawn from prior distribution P will look like a lattice network $\mathcal{G}$ with $w_c \times d_c$ total channels, and each channel at layer i is connected to w_c channels from the previous layer. Let this prior distribution P

[2]Note that, choosing a prior without any long-range links makes sense even from an "evolution of CNN architectures" perspective since, initially, CNN architectures such as AlexNet [58] and VGG-16 [108] were developed (which do not have any shortcut connections), and complex architectures with long-range connections such as NASNET [142], DenseNets [50], *etc.* were proposed only later.

be given by a distribution $P(\mathcal{G})$ over lattice networks $\mathcal{G}$.

Next, we model the distribution Q for the proposed CNN architectures with long-range links. Note that, the CNNs considered in our work have both short-range and long-range links (see Fig. 6.2(c) and Fig. 6.3(inset)). This kind of topology typically falls into the category of small-world networks which can be represented as a lattice network $\mathcal{G}$ (containing short-range links) superimposed with a random network $\mathcal{R}$ (to account for long-range links) [79, 87]. Hence, the distribution over connectivities c in the small-world network can be written as:

$$Q \sim P(\mathcal{G}, \mathcal{R}) = P(\mathcal{G}) \cdot P(\mathcal{R}|\mathcal{G}) \tag{6.8}$$

Since $P(\mathcal{R}|\mathcal{G})$ represents the random long-range links created on top of the lattice network $\mathcal{G}$, the connectivity of long-range links due to $\mathcal{R}|\mathcal{G}$ follows a Poisson Distribution. This is because the degree distribution of random networks has been shown to be Poisson Distribution [5]. The λ parameter (*i.e.*, the mean) of this Poisson Distribution is given by the average degree of the random network. Therefore, the average degree for $\mathcal{R}$ superimposed on $\mathcal{G}$ is given by:

$$\begin{aligned}
\lambda = \bar{k}_{\mathcal{R}|\mathcal{G}} &= \frac{\textit{Number of long-range links added by } \mathcal{R}}{\textit{Number of nodes}} \\
&= \frac{w_c \sum_{i=2}^{d_c-1} \min\{w_c(i-1), t_c\}}{w_c d_c} \\
&= \frac{m(d_c-1)(d_c-2)}{2d_c^2} && \text{(using Eq. (6.7) for one cell)} \\
&\approx \frac{m}{2} && \text{(when } d_c >> 2\text{, } \textit{e.g.}\text{, for deep CNNs)}
\end{aligned} \tag{6.9}$$

Then, the probability $Q(c) = P(\mathcal{G}) \cdot P(\mathcal{R}|\mathcal{G}))$ of observing connectivity c in the small-world network can be written as follows:

$$\begin{aligned}
Q(c) &= P(\mathcal{G}) \cdot \frac{\lambda^c e^{-\lambda}}{c!} \\
&= P(\mathcal{G}) \cdot \frac{(\bar{k}_{\mathcal{R}|\mathcal{G}})^c e^{-\bar{k}_{\mathcal{R}|\mathcal{G}}}}{c!},
\end{aligned} \tag{6.10}$$

where, $c \geq \bar{k}_{\mathcal{G}}$, the average degree of the lattice network $\mathcal{G}$. Since average degree is given by total number of links divided by number of nodes, $\bar{k}_{\mathcal{G}} = (w_c^2(d_c-1))/(w_c d_c) = (w_c(d_c-1))/d_c \approx w_c$ when $d_c >> 2$ (and implicitly $d_c >> 1$). Now, let σ be the upper bound on the connectivity c. Also, we assume that the prior distribution P of drawing a lattice network is a uniform distribution

for support $c \in \{\bar{k}_{\mathcal{G}} - \sigma + 1, \bar{k}_{\mathcal{G}} - \sigma + 2, \ldots, \bar{k}_{\mathcal{G}} + \sigma\}$. Then, the probability of observing a lattice network $\mathcal{G}$ with connectivity c is given by:

$$P(\mathcal{G}) = \begin{cases} \frac{1}{2\sigma} & \text{if } c \in \{\bar{k}_{\mathcal{G}} - \sigma + 1, \bar{k}_{\mathcal{G}} - \sigma + 2, \ldots, \bar{k}_{\mathcal{G}} + \sigma\} \\ 0 & \text{otherwise} \end{cases} \tag{6.11}$$

Note that, since our proposed CNN architecture (drawn from distribution Q) is also based on a lattice network, the probability of drawing a lattice $P(\mathcal{G})$ is same between P (*i.e.*, the prior) and Q (*i.e.*, our hypothesis).

We next use the above equations to simplify the KL-divergence term in Theorem 1. For deep CNNs with $d_c >> 2$, we have

$$\begin{aligned} KL(Q||P) &= -\sum_{c=\bar{k}_{\mathcal{G}}}^{\sigma} Q(c) \log\left(\frac{P(c)}{Q(c)}\right) \\ &= -\sum_{c=\bar{k}_{\mathcal{G}}}^{\sigma} P(\mathcal{G}) \cdot P(\mathcal{R}|\mathcal{G}) \log\left(\frac{P(\mathcal{G})}{P(\mathcal{G}) \cdot P(\mathcal{R}|\mathcal{G})}\right) \\ &= \sum_{c=\bar{k}_{\mathcal{G}}}^{\sigma} P(\mathcal{G}) \cdot P(\mathcal{R}|\mathcal{G}) \log(P(\mathcal{R}|\mathcal{G})) \\ &= \sum_{c=\bar{k}_{\mathcal{G}}}^{\sigma} \frac{1}{2\sigma} \cdot P(\mathcal{R}|\mathcal{G}) \log(P(\mathcal{R}|\mathcal{G})) \\ &= \frac{1}{2\sigma} \sum_{c=\bar{k}_{\mathcal{G}}}^{\sigma} P(\mathcal{R}|\mathcal{G}) \log(P(\mathcal{R}|\mathcal{G})) \\ &= \frac{1}{2\sigma}(-\mathbb{H}_{\text{Poisson}(\bar{k}_{\mathcal{R}|\mathcal{G}})}), \end{aligned} \tag{6.12}$$

where, $\mathbb{H}_{\text{Poisson}(\bar{k}_{\mathcal{R}|\mathcal{G}})})$ is the Entropy of $\mathcal{R}|\mathcal{G}$ which follows a Poisson Distribution (Eq. 6.10). For a large λ, the Entropy of a Poisson Distribution can be approximated as [26]:

$$\mathbb{H}_{\text{Poisson}(\lambda)} = \frac{1}{2}\log(2\pi e\lambda) - \frac{1}{12\lambda} + O(\frac{1}{\lambda^2}) \tag{6.13}$$

From Eq. (6.9), Eq. (6.13), and Eq (6.12) and by ignoring the higher order terms of λ (since it is large), we get the following expression:

$$KL(Q||P) = \frac{1}{2\sigma}\left(\frac{1}{6m} - \frac{1}{2}\log(\pi e m)\right) \tag{6.14}$$

Putting the above expression into the bound in Theorem 1, we immediately get the generalization bound in terms of NN-Mass as shown in Theorem 2. □

Remark 1. *Note that, the KL-divergence term (Eq.* (6.14)*) is a decreasing function of* m*. Hence, for a given family of models with depth* d_c *and width* w_c*, Theorem 2 indicates that the test error should reduce as NN-Mass increases. We show extensive empirical results for this observation.*

We next present another result which follows as a natural consequence of Theorem 2. The next corollary shows that, for two CNNs with a given width but different depths, models with similar NN-Mass values are expected to achieve similar generalization performance, even if the two models have vastly different number of parameters and layers!

Corollary 1 (Models with Similar Mass Achieve Similar Generalization Performance)**.** *Given two models of same width* w_c*, let* f_Q^L *be a deeper model with NN-Mass* m_L*, and let* f_Q^S *be a shallower model with NN-Mass* m_S *such that* $m_L \leq m_S$*. Then, the difference in expected test error of the two models is bounded with probability at least* $1-\delta$ *as follows:*

$$\mathbb{E}_{\mathcal{D}}(L(f_Q^L)) - \mathbb{E}_{\mathcal{D}}(L(f_Q^S)) \leq \mathbb{E}_{\mathcal{S}}(\hat{L}(f_Q^L)) - \mathbb{E}_{\mathcal{S}}(\hat{L}(f_Q^S)) + \sqrt{\frac{\frac{1}{2\sigma}\left[\frac{m_S - m_L}{6 m_L m_S} - \frac{1}{2}\log\frac{m_L}{m_S}\right] + \log\frac{4N}{\delta^2}}{2N}}$$

In other words, irrespective of number of parameters, as the NN-Mass for the two models becomes similar, their test error is also expected to become similar.

Proof. The corollary follows directly from the KL-divergence term in Theorem 2 (see Eq. (6.14)). To obtain the closed form bound above, we first compute the difference between the expected test errors for the deeper model f_Q^L and the shallower model f_Q^S. Then, according to Theorem 1, we have:

$$\begin{aligned}\mathbb{E}_{\mathcal{D}}(L(f_Q^L)) - \mathbb{E}_{\mathcal{D}}(L(f_Q^S)) \leq \mathbb{E}_{\mathcal{S}}(\hat{L}(f_Q^L)) - \mathbb{E}_{\mathcal{S}}(\hat{L}(f_Q^S)) \\ + \sqrt{\frac{KL(Q_L||P) + \log\frac{2\sqrt{N}}{\delta}}{2N}} - \sqrt{\frac{KL(Q_S||P) + \log\frac{2\sqrt{N}}{\delta}}{2N}}\end{aligned} \tag{6.15}$$

To simplify the difference of square roots, we exploit the following useful lemma.

Lemma 1 (Two Related Triangles [45])**.** *If three non-negative numbers* a*,* b*, and* c *satisfy the triangle inequality (i.e.,* $a+b \geq c$*,* $b+c \geq a$*, and* $a+c \geq b$*), then the following also holds:*

$$|\sqrt{b} - \sqrt{c}| \leq \sqrt{a} \leq \sqrt{b} + \sqrt{c}$$

Proof for this lemma is given in [45].

Let $b = \frac{KL(Q_L||P)+\log\frac{2\sqrt{N}}{\delta}}{2N}$, and let $c = \frac{KL(Q_S||P)+\log\frac{2\sqrt{N}}{\delta}}{2N}$. Then, all we need to do is to find a such that the triangle inequality between a, b, and c is satisfied. Once we find such an a, the difference of square roots in (6.15) can be bounded by $\sqrt{a}$. Towards this, we begin by finding a lower bound for $b + c$:

$$\begin{aligned} b + c &= \frac{1}{2N}(KL(Q_L||P) + KL(Q_S||P)) + \frac{1}{2N}\log\frac{4N}{\delta^2} \\ &\geq \frac{1}{2N}|KL(Q_L||P) - KL(Q_S||P)| + \frac{1}{2N}\log\frac{4N}{\delta^2} \\ &= \frac{1}{2N}(KL(Q_L||P) - KL(Q_S||P)) + \frac{1}{2N}\log\frac{4N}{\delta^2} = a, \end{aligned} \tag{6.16}$$

where, the first inequality follows from the fact that the sum of two non-negative real numbers will always be greater than or equal to the absolute value of their difference. The second equality follows from Eq. (6.14). Specifically, since KL-divergence is a decreasing function of NN-Mass, and since $m_L \leq m_S$, it follows that $KL(Q_L||P) \geq KL(Q_S||P)$.

Now that we have a lower bound on $b + c$, we use this value as a. In order to use Lemma 1, we only need to make sure that the chosen values for a, b, and c satisfy the triangle inequality. Note that, by construction, $b + c \geq a$. So, we only need to prove that $a + c \geq b$ and $a + b \geq c$. For the former, it is easy to see that $a+c = b+\frac{1}{2N}\log\frac{4N}{\delta^2} \implies a+c \geq b$ (since $\frac{1}{2N}\log\frac{4N}{\delta^2}$ is non-negative as the size of training set N is a large number and $\delta \in (0, 1]$). For the latter, we have:

$$\begin{aligned} a + b &= \frac{1}{2N}KL(Q_L||P) - \frac{1}{2N}KL(Q_S||P) + \frac{1}{2N}\log\frac{4N}{\delta^2} + \frac{1}{2N}KL(Q_L||P) + \frac{1}{2N}\log\frac{2\sqrt{N}}{\delta} \\ &= \frac{2}{2N}KL(Q_L||P) - \frac{1}{2N}KL(Q_S||P) \qquad \text{(add and subtract } KL(Q_S||P)/2N\text{)} \\ &\quad - \frac{1}{2N}KL(Q_S||P) + \underbrace{\frac{1}{2N}KL(Q_S||P) + \frac{1}{2N}\log\frac{2\sqrt{N}}{\delta}}_{c} + \frac{1}{2N}\log\frac{4N}{\delta^2} \\ &= c + \underbrace{\frac{2}{2N}(KL(Q_L||P) - KL(Q_S||P))}_{\geq 0 \text{ since } m_L \leq m_S \implies KL(Q_L||P) \geq KL(Q_S||P)} + \underbrace{\frac{1}{2N}\log\frac{4N}{\delta^2}}_{\geq 0} \geq c \end{aligned} \tag{6.17}$$

Hence, a, b, and c satisfy the triangle inequality. Therefore, we can use Lemma 1 to bound the difference of square roots in (6.15). Putting KL-divergence from Eq. (6.14) into a in (6.16), we get the following expression:

$$a = \frac{\frac{1}{2\sigma}\left[\frac{m_S - m_L}{6m_L m_S} - \frac{1}{2}\log\frac{m_L}{m_S}\right] + \log\frac{4N}{\delta^2}}{2N} \tag{6.18}$$

Finally, using Lemma 1 on (6.15) with the above a, we get the corollary statement. Clearly, the difference in expected error for the two models reduces as their NN-Mass values become similar. Therefore, irrespective of the number of parameters, if two models have similar NN-Mass, they are expected to yield similar test accuracies. We will present extensive empirical evidence to verify this corollary. □

This completes the theoretical analysis of our proposed NN-Mass metric. Next, we briefly discuss how NN-Mass can be used for model compression.

6.3.4 NN-Mass for directly designing compressed architectures

From the above theoretical insights, NN-Mass is a reliable indicator for models which achieve similar accuracy despite having different number of layers and parameters. Therefore, this observation can be used for model compression as follows:

- First, train a reference big CNN (with a large number of parameters and layers) which achieves very high accuracy on the target dataset. Calculate its NN-Mass (denoted m_L).
- Next, create a *completely new and significantly compressed model* using far fewer parameters and layers, but with a NN-Mass comparable to or higher than the large CNN. This process is very fast as the new model is created without any *a priori* training. For instance, to design a compressed CNN of width w_c and depth per cell d_c and NN-Mass $m_S \approx m_L$, we only need to find how many long-range links to add in each cell. Since, NN-Mass has a closed form equation, a simple search over the number of long-range links can directly determine NN-Mass of various architectures. Then, we select the architecture with the NN-Mass close to that of the reference CNN. Unlike current manual or NAS-based methods, our approach does not require training of individual architectures during the search.
- Since NN-Mass of the compressed model is similar to that of the reference CNN, Corollary 1 suggests that the newly generated model will lose only a small amount of accuracy, while significantly reducing the model size. To validate this, we train the newly compressed model and compare its test accuracy against that of the original large CNN.

Note that, our work is agnostic to what dataset is used since we rely on the properties of the CNN architecture. That is, if we train the large CNN on a different dataset, our compressed model

should still give accuracy close to that of the large CNN on the new dataset. Of course, the range of accuracy of different models will vary when the dataset is changed, but different architectures with similar NN-Mass should still yield a similar test accuracy. We explicitly show this observation for CIFAR-10 and CIFAR-100 datasets in experiments. Next, we present detailed experimental evidence towards our theoretical insights.

6.4 Experimental Setup and Results

In this section, we first describe our experimental setup, followed by results.

6.4.1 Experimental Setup

We conduct numerous experiments to validate our theory (Theorem 2 and Corollary 1). Towards this, we systematically analyze the impact of NN-Mass and NN-Density on the test accuracy of CNN architectures. Specifically, we perform NASE by varying $\{d_c, w_c, t_c\}$ to generate random architectures with different NN-Mass and NN-Density values. We perform experiments for CIFAR-10 and CIFAR-100 datasets.

We conduct the following kinds of experiments for NASE: (*i*) We show that NN-Density is not a useful metric to answer our core question: *What characteristics of a CNN architecture indicate that different models (with different number of parameters and layers) will achieve similar test accuracy?* (*ii*) We next show that NN-Mass can indeed answer this question by providing extensive empirical evidence towards Theorem 2 and Corollary 1. (*iii*) We further show that our findings hold across models with different widths. (*iv*) We then quantitatively demonstrate that our metric is better for predicting generalization performance than *parameter counting*, a regularly used indicator of model generalization. (*v*) We predict test accuracy of completely unknown architectures. (*vi*) We also show that our findings hold for CIFAR-100 dataset which is significantly more complex than the CIFAR-10 dataset. All experiments are repeated three times with different random seeds. Complete details of various t_c values for different networks is given in Table. 6.1.

Finally, to demonstrate the practical implications of our work, we exploit NN-Mass for model compression: We show that NN-Mass can enable us to directly create efficient CNNs which achieve accuracy comparable to larger networks with similar NN-Mass. The learning rate for all

Table 6.1: Details of Experiments for varying α_{ij}'s and Average Densities

Experiment Type	Number of Cells	Long-Range Links (t_c)	α_{ij}'s	Depth	Width Multiplier
Impact of α_{ij}'s	1	200	Constant(1/N)	46	2
	1	200	Ones (Traditional CNN)	46	2
	1	200	Random Probabilities	46	2
Impact of Average Density	3	[10,35,50] [20,45,75] [30,50,100] [40,60,120] [50,70,145]	Random Probabilities	31	2
Impact of Average Density	3	[20,40,70] [30,50,100] [40,80,125] [50,105,150] [60,130,170]	Random Probabilities	40	2
Impact of Average Density	3	[25,50,90] [35,80,125] [50,105,150] [70,130,170] [90,150,210]	Random Probabilities	49	2
Impact of Average Density	3	[30,80,117] [50,110,150] [70,140,200] [90,175,250] [110,215,300]	Random Probabilities	64	2

models is initialized to 0.05 and follows a cosine-annealing schedule at each epoch. The minimum learning rate is 0.0. Similar to setup in NAS prior works, cutout is used for data augmentation. All models are trained in Pytorch on NVIDIA 1080-Ti, Titan Xp, and 2080-Ti GPUs. This completes the experimental setup. Next, we describe our results for the above classes of experiments.

6.4.2 Results

We begin by evaluating the impact of random probabilities on CNN accuracy. We then perform NASE by varying NN-Density, NN-Mass, and model-width to thoroughly validate Theorem 2 and Corollary 1. Unless stated otherwise, the results are for CIFAR-10 dataset. Towards the end of this section, the main results are also corroborated for CIFAR-100 dataset.

Impact of Input-to-Output Contributions

To evaluate the impact of various α_{ij}'s on CNN generalization, we train three separate models (see Table 6.1(top) for details of the architecture): (a) For the first model, we initialize the contribution-weights for each layer to zeros and then take the softmax. Therefore, in this case, all contribution probabilities (α_{ij}'s) are constant (1/N, N being the number of input channels per layer). (b) In the next model, we directly initialize α_{ij}'s to all ones, which is the traditional CNN case where all channel-wise convolutions are directly added to obtain each output channel. The above two cases are *equal-contribution cases*. (c) Finally, to account for the proposed *unequal-contribution case*, we follow the process described in Section 6.3.1 by fixing the contribution-weights to Kaiming initialization [40] and then take the softmax. As evident from Table 6.1, all models have 46-layers, $t_c = 200$, and width-multiplier of 2, which amounts to about 8M parameters. The models are trained for 350 epochs.

Fig. 6.4 demonstrates the training and test accuracy for the three models. As shown, the model with proposed unequal contributions (via random probabilities) achieves about 96.6% accuracy, which is about 1% higher than the corresponding equal-contributions cases. This is a significant result because achieving accuracy beyond 96% is very hard for CIFAR-10 models [50, 66, 134, 141]. Hence, this clearly demonstrates that not all input channels should contribute equally to all output channels, and that even fixing the contributions to random probabilities significantly

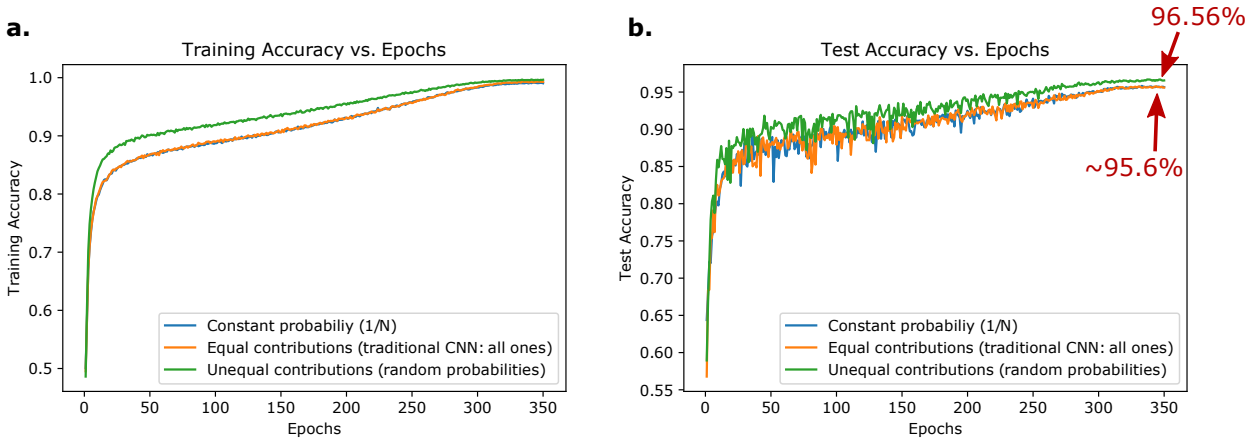


Figure 6.4: Not all input channels contribute equally to all output channels. (a) Training accuracy for three cases: (*i*) Set α_{ij}'s to constant probabilities by initializing the input-to-output contribution-weights to zero at every layer and then taking the softmax. This results in constant probabilities as contributions from input to output channel at each layer (N is the number of input channels). (*ii*) A traditional CNN case where all channel-wise convolutions are directly added (*i.e.*, contributions from all inputs to all outputs (α_{ij}'s) are all ones). (*iii*) Channel-wise contributions are fixed to random probabilities, *i.e.*, α_{ij}'s are random. (b) Test accuracy for the above three cases demonstrates that the unequal contributions from input-to-output channels achieves significantly higher accuracy.

improves the generalization and convergence of the model. Note that, the α_{ij} contributions do not come at any additional cost in terms of number of parameters since these values are fixed and can be directly incorporated into convolution weights by element-wise multiplication. Therefore, this simple observation can be used to improve the accuracy of CNNs without any overhead in terms of number of parameters.

Neural Architecture Space Exploration and NN-Mass as an Indicator of Generalization

Impact of Varying NN-Density In this section, we first fix the width of the model and vary the average density for CNNs with different depths. In Fig. 6.5, we analyze the relationship between NN-Mass and NN-Density for various deep networks. As evident, NN-Mass grows faster for deeper networks than shallower networks as the NN-Density increases (see Fig. 6.5(a)). We next analyze the relationship between the NN-Mass and total number of model parameters. The results are shown in Fig. 6.5(b). We observe that, as the number of parameters grows, the NN-Mass increases at a faster rate for shallower networks than the deeper networks. Therefore, shallower networks can attain NN-Mass comparable to deeper networks with significantly lower number of

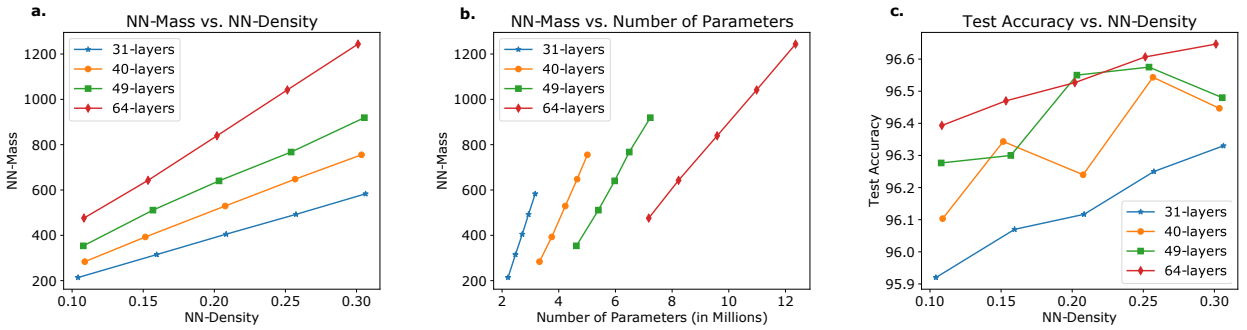


Figure 6.5: (a) NN-Mass *vs.* NN-Density for networks of different depth (and, hence, parameters): NN-Mass increases more rapidly for deeper networks with higher NN-Density. (b) NN-Mass *vs.* Number of parameters: NN-Mass increases rapidly (gradually) for shallower (deeper) networks. (c) Accuracy as a function of density for networks of different depths. As density increases, the accuracy generally increases. The accuracy eventually saturates for different networks. Shallower models with higher density can reach comparable accuracy to deeper models with lower density. Results are reported as mean of three runs. Width multiplier is fixed to 2 for all experiments.

parameters.

Next, we train different deep networks with varying NN-Density and record their test accuracy. Fig. 6.5(c) shows that shallower models with higher density can reach accuracy comparable to deeper models with lower density. Note that, this observation may be argued as being obvious. For instance, one might say that, for a fixed width, if a shallower model is more densely connected than a deeper model, it might lead to accuracy comparable to that of the deeper CNN. However, this insight does *not* help us with our original question: What characteristics help us identify *family of models* that yields similar generalization despite having significantly different number of parameters/layers? Specifically, while we can say that for a given width, a shallower model might outperform a deeper model provided it is connected densely-enough, NN-Density does not tell us how densely the shallower model must be connected. Moreover, the NN-Density value (that determines when two models of different depths perform comparably) changes for different depths (*e.g.*, although a 31-layer model with $\rho_{avg} = 0.3$ performs close to 64-layer model with $\rho_{avg} = 0.1$, a 49-layer model with $\rho_{avg} = 0.2$ already outperforms the test accuracy of the above 64-layer model). Therefore, NN-Density *cannot* be used to identify which architectural characteristics result in CNNs with similar generalization performance. Hence, we next move to NN-Mass and analyze its relationship with generalization.

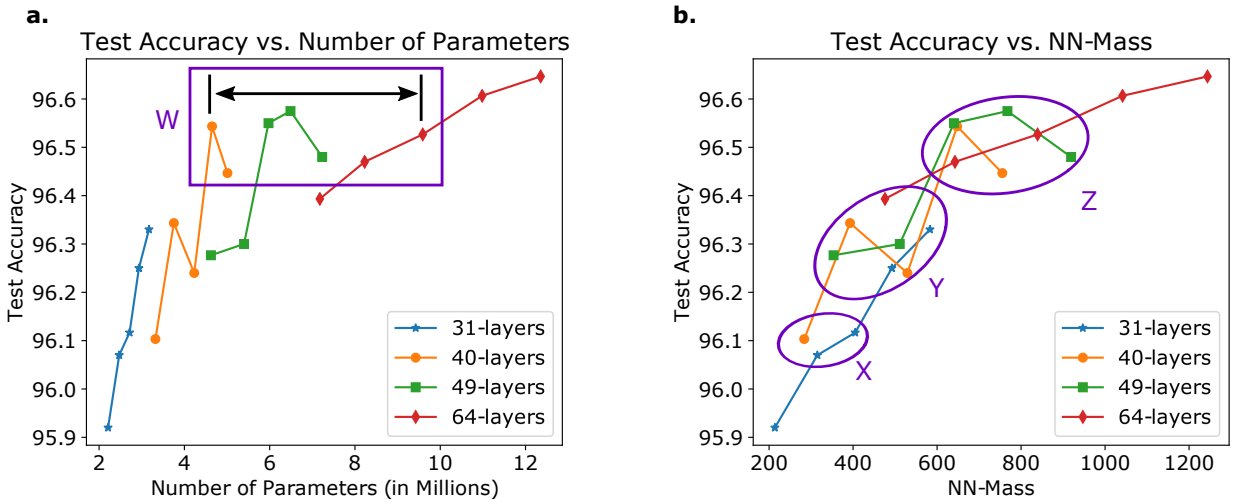


Figure 6.6: CIFAR-10 Neural Architecture Space Exploration for Width Multiplier $wm = 2$: (a) Accuracy *vs.* Number of parameters: Shallower models with less parameters achieve comparable accuracy to deeper models with more parameters. (b) Models achieving similar accuracy often have similar NN-Mass. Width multiplier is fixed to 2 for all experiments. Results are reported as mean of three runs.

Impact of Varying NN-Mass on Generalization In this section, we present concrete empirical evidence towards Theorem 2 and Corollary 1. To this end, we first analyze the architecture space from the perspective of total number of parameters and the proposed network science-based NN-Mass metric. Fig. 6.6(a) shows the test accuracy of the trained models *vs.* total parameters. As evident, some of the 40-layer models with about 5M parameters perform comparably to several 64-layer models with more than 8M parameters. Hence, smaller models (with both lower depth and fewer parameters!) can achieve accuracy close to larger models.

To explain the above phenomenon, Fig. 6.6(b) shows test accuracy *vs.* NN-Mass. Two observations are worth noting:

1. **The higher the NN-Mass, the higher the test accuracy.** This observation reinforces Remark 1 and Theorem 2 that higher NN-Mass should result in lower generalization error.
2. **Irrespective of number of layers or parameters, models with similar NN-Mass achieve similar accuracy.** This gives empirically evidence towards our Corollary 1. Specifically, we observe that 40-layer models with around 5M parameters have similar NN-Mass as 8M-parameter, 64-layer models and, therefore, achieve comparable accuracy. Moreover, all models shown within the box W in Fig. 6.6(a) in fact cluster into bucket Z shown in Fig. 6.6(b).

Same argument holds for the models with similar NN-Mass within buckets X and Y which obtain similar test accuracy.

The above observations clearly emphasize the importance of the proposed NN-Mass metric as an indicator of generalization performance and to identify *family of models* that obtain similar test accuracy.

The above results are for a width multiplier of 2 and vary NN-Mass indirectly by changing the NN-Densities. Since it is clear that NN-Mass is highly correlated with generalization, we now directly vary NN-Mass for models with different depths. Further, we vary the width multiplier ($wm \in \{1, 3\}$) in order to ensure that the above observations hold true for CNNs with different widths. More specifically, in Fig. 6.7(a), we plot test accuracy *vs.* number of parameters for $wm = 1$ and observe that models in boxes U and V have significantly different number of parameters and, yet, they achieve very similar test accuracy. Again, when plotted against NN-Mass (see Fig. 6.7(b)), models within the boxes U and V in Fig. 6.7(a) concentrate into buckets W and Z, respectively (see also other clusters).

Note that, for $wm = 1$, the 31-layer model does not fall within the buckets. We hypothesize that this could be because of the tradeoff between the two terms in Corollary 1. Specifically, since Corollary 1 states that the difference in test errors is bounded by the sum of (*i*) the difference in training errors, and (*ii*) the difference between NN-Mass values, the former term might dominate for low-capacity models (and, thus, the difference in test errors would increase). For instance, the training accuracy of 31-layer models is found to be much (*e.g.*, 0.66-0.9%) lower than that of 64-layer models. In contrast, 40-layer models have only 0.27-0.4% lower training error than the 64-layer CNNs. Hence, this suggests that a tradeoff between training accuracy difference and NN-Mass values should affect the difference in test accuracies of various architectures. We next show that as the width multiplier increases further, the shallower models perform much more similar to deeper models.

Fig. 6.8(a) shows the test accuracy *vs.* number of parameters for four different depths. As evident, models with 6-7M parameters achieve comparable test accuracy as models with up to 16M parameters (*e.g.*, bucket Y in Fig. 6.8(b) contains models ranging from {31 layers, 6.7M parameters}, all the way to {64 layers, 16.7M parameters}). Further, 31-layer models consistently perform comparable to deeper models. In general, we observe that as the width increases, the

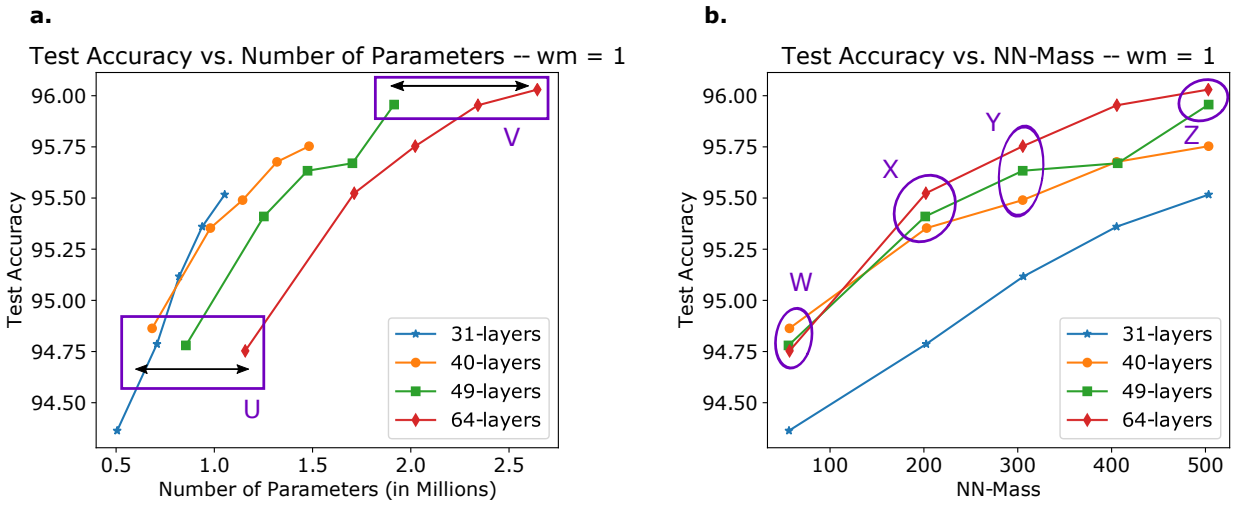


Figure 6.7: CIFAR-10 Neural Architecture Space Exploration for Width Multiplier $wm = 1$:(a) Accuracy *vs.* Number of parameters: Shallower models with less parameters achieve comparable accuracy to deeper models with more parameters. (b) Models achieving similar accuracy often have similar NN-Mass. Width multiplier is fixed to 1 for all experiments. Results are reported as mean of three runs.

capacity of the CNNs increases and, hence, the curves on Accuracy *vs.* NN-Mass plot come closer to each other. We next quantify the above observation by fitting a linear model to predict Accuracy (target variable) using log(NN-Mass) (explanatory variable). The results for $wm = 1, 2, 3$ are shown in Fig. 6.9(a,b,c), respectively. As evident, the goodness-of-fit R^2 value increases from 0.74 to 0.84 to 0.90, as the width multiplier increases. This clearly demonstrates that as the width of the CNN increases, NN-Mass becomes better indicator of generalization performance.

Comparison between NN-Mass and Parameter Counting Naive parameter counting is often used as an indicator of generalization [3]. Here, we quantitatively demonstrate that while parameter counting can be a useful indicator of generalization for models with low width (but still not as good as NN-Mass), as the width increases, parameter counting cannot predict generalization. In contrast, we show that NN-Mass consistently predicts generalization performance with high accuracy. Specifically, in Fig. 6.10(a), we fit a linear model between test accuracy and log(#parameters) and found that the R^2 for this model is 0.76 which is slightly lower than that obtained for NN-Mass (see Fig. 6.10(b)). When the width multiplier of CNNs increases to three, parameter counting completely fails to fit the test accuracies of the models ($R^2 = 0.14$). In contrast, NN-Mass significantly outperforms parameter counting for $wm = 3$ as it achieves an $R^2 = 0.90$. This demonstrates that

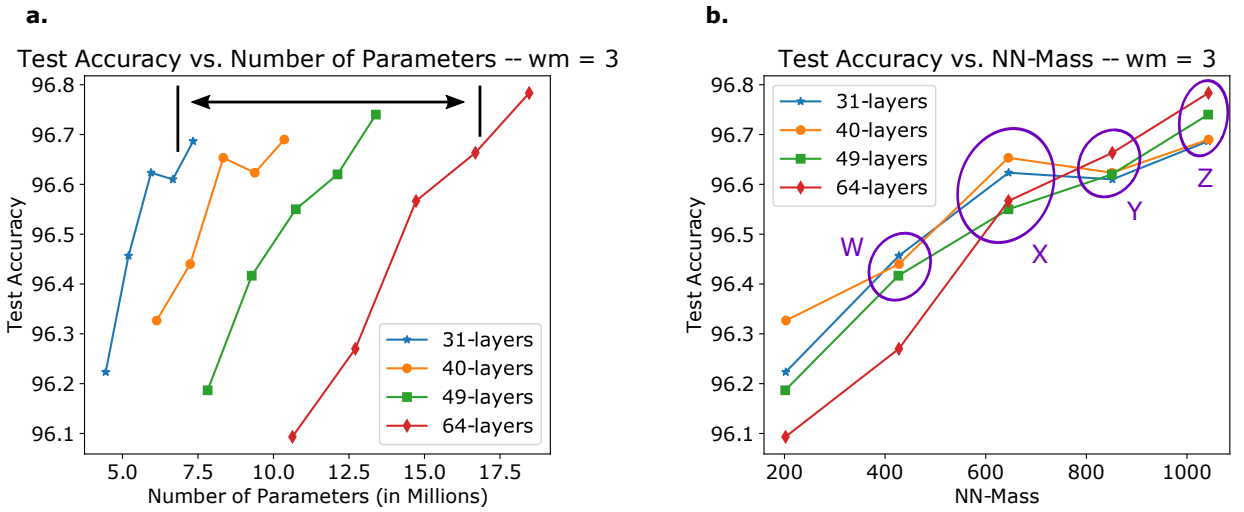


Figure 6.8: CIFAR-10 Neural Architecture Space Exploration for Width Multiplier $wm = 3$: (a) Accuracy *vs.* Number of parameters: Shallower models with less parameters achieve comparable accuracy to deeper models with more parameters. (b) Models achieving similar accuracy often have similar NN-Mass. Width multiplier is fixed to 3 for all experiments. Results are reported as mean of three runs.

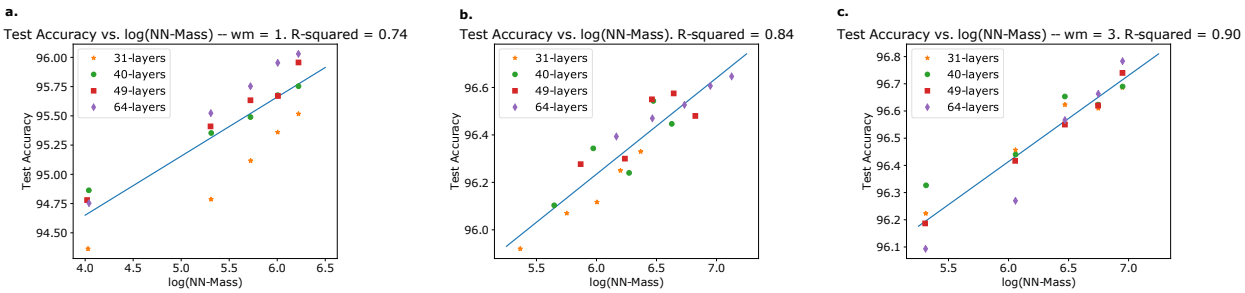


Figure 6.9: Impact of varying width on accuracy *vs.* log(NN-Mass) relationship. (a) Width multiplier, $wm = 1$, (b) $wm = 2$, and (c) $wm = 3$. As width increases, capacity of small (shallower) models increases and, therefore, the accuracy-gap between models of different depths reduces. Hence, the R^2 for linear fit increases as width of the CNNs increases.

NN-Mass is indeed a significantly stronger indicator of generalization than parameter counting for models with long-range links.

We note that our work cannot be compared against generalization indicators presented in [3, 8, 89, 90, 91], *etc*., because these works do *not* consider architectural aspects of the generalization problem and do *not* deal explicitly with CNN architectures. The objective of this prior art is to understand how optimization properties (*e.g.*, sharpness of minima), noise-stability, weight-norms, *etc* affect generalization. Hence, the prior research does not explicitly provide any insights into the

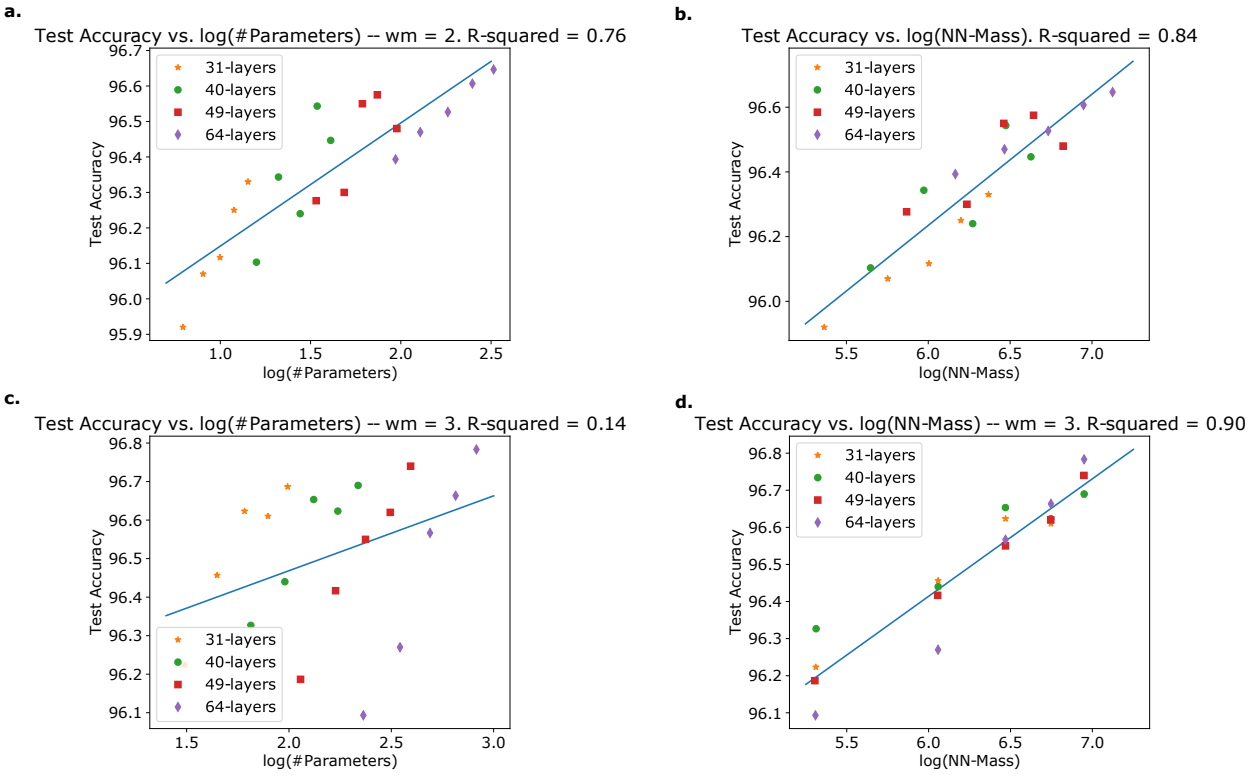


Figure 6.10: NN-Mass as an indicator of generalization performance compared to parameter counting. (a) For $wm = 2$, log(#parameters) fits the test accuracy with an $R^2 = 0.76$. (b) For the same $wm = 2$ case, log(NN-Mass) fits the test accuracy with a higher $R^2 = 0.84$. For lower width, parameter counting is a decent indicator of generalization performance. (c) For higher width ($wm = 3$), parameter counting completely fails to fit the test accuracy of various models ($R^2 = 0.14$). (d) In contrast, NN-Mass still fits the accuracies with a high $R^2 = 0.9$.

architecture itself. On the other hand, our problem is to explicitly understand the impact of CNN architecture on generalization.

Exploiting NN-Mass to Predict Test Accuracy on Unknown Architectures Next, we demonstrate that NN-Mass can be used to predict the test accuracy of unknown architectures that have not been trained before. Towards this, we create a *testing set of new architectures* by training 20 different models with $wm = 2$, and $\{28, 43, 52, 58\}$ layers. For these models, we vary the NN-Density between $\{0.125, 0.175, 0.225, 0.275, 0.325\}$ which is different from the initial architecture space exploration setting (*i.e.*, $\{0.10, 0.15, 0.20, 0.25, 0.30\}$). We use the linear model trained on the test accuracy of models shown in Table 6.1 (see Fig. 6.9(b) for the linear fit). Note that, our testing set consists of models with both different number of layers and different densities compared to the

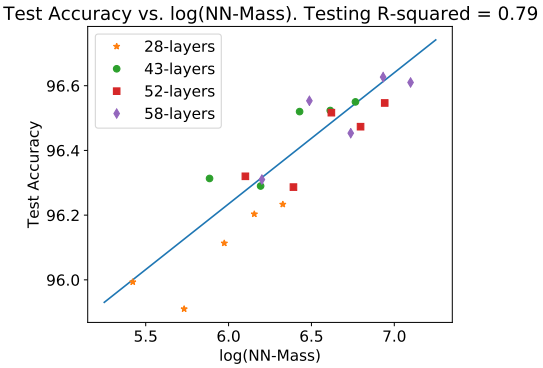


Figure 6.11: Linear modeled trained in Fig. 6.9(b) is used to predict the test accuracy of completely new architectures. The resulting $R^2 = 0.79$ is still high and is comparable to the training $R^2 = 0.84$. The linear model was trained on the test accuracies and NN-Mass of models with $\{31, 40, 49, 64\}$ layers, and densities varying as $\{0.10, 0.15, 0.20, 0.25, 0.30\}$. To create the testing set, we trained completely new models with $\{28, 43, 52, 58\}$ layers, and densities varying as $\{0.125, 0.175, 0.225, 0.275, 0.325\}$.

training set.

Fig. 6.11 shows that the testing $R^2 = 0.79$ (*i.e.*, the R^2 obtained by predicting the accuracy of models in the testing set) which is close to the training $R^2 = 0.84$ (see Fig. 6.9(b)). Hence, NN-Mass can be used to predict test accuracy of models which were never trained before.

Results for CIFAR-100 Dataset We now corroborate our main findings on CIFAR-100 dataset which is significantly more complex than the CIFAR-10 dataset. To this end, we retrain the models shown in Table 6.1 on CIFAR-100. Fig. 6.12(a) shows the test accuracy of various models as a function of number of parameters. As evident, several models achieve similar accuracy despite having highly different number of parameters (*e.g.*, see models within box W in Fig. 6.12(a)). Again, these models get clustered together when plotted against NN-Mass. Specifically, models within box W in Fig. 6.12(a) fall into buckets Y and Z in Fig. 6.12(b). Hence, models that got clustered together for CIFAR-10 dataset, also get clustered for CIFAR-100.

To quantify the above results, we fit a linear model between test accuracy and log(NN-Mass) and, again, obtain a high $R^2 = 0.84$. Therefore, our observations for NN-Mass hold true across multiple image classification datasets.

This completes the NASE with NN-Density and NN-Mass metrics. We learned that (*i*) NN-Mass is a reliable indicator for generalization performance: For a given width, as NN-Mass in-

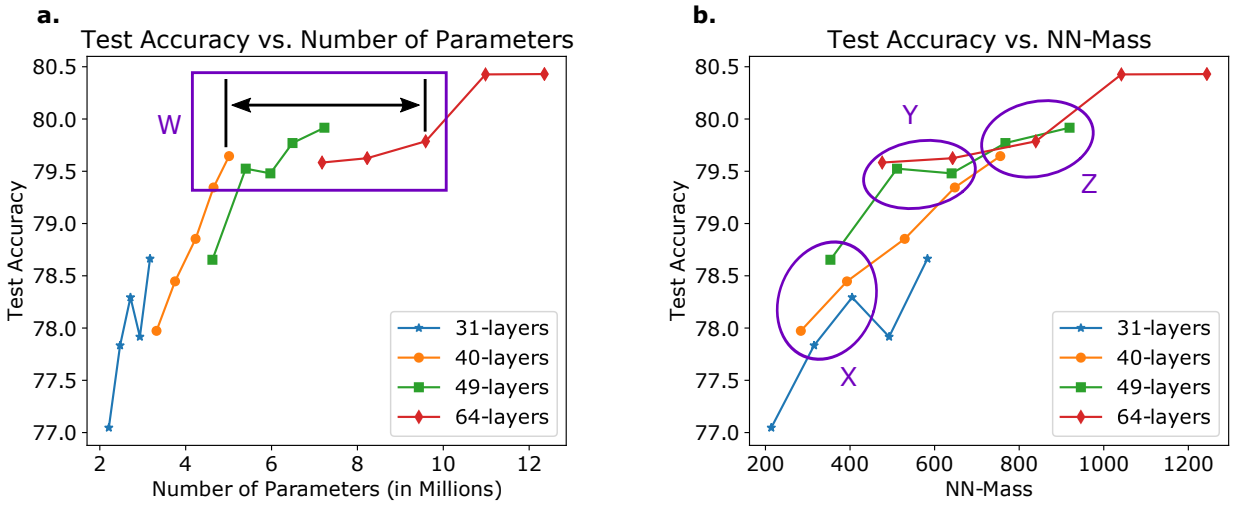


Figure 6.12: Similar results are obtained for the CIFAR-100 dataset. (a) Accuracy *vs.* Number of parameters: Shallower models with less parameters achieve comparable accuracy to deeper models with more parameters. (b) Models achieving similar accuracy often have similar NN-Mass. Width multiplier is fixed to 2 for all experiments. Results are reported as mean of three runs.

creases, the test error of CNNs reduces (Theorem 2, Remark 1), and (*ii*) NN-Mass can be used to identify family of models which yield similar test accuracy without any *a priori* training (Corollary 1). We next exploit the latter insight to directly design highly compressed architectures which achieve as high accuracy as large, overparameterized models.

Directly Designing Compressed Models with NN-Mass

In this section, we directly exploit the NN-Mass to design significantly compressed models with minimal loss of accuracy compared to a large CNN. We demonstrate that NN-Mass can enable a new form of model compression from the architecture standpoint. Specifically, given a large CNN, we can now directly create new models with far fewer layers and parameters but with comparable (or slightly higher) NN-Mass as the large CNN. Hence, such models should achieve high accuracy, while reducing the number of parameters and layers. The amount of accuracy loss generally depends on depth of the compressed models: for a fixed width, shallower models tend to lose more accuracy than the deeper models. We next present the results for this new form of model compression. Following the setup in recent NAS works like DARTS [66], we train our models for 600 epochs and report their test accuracy.

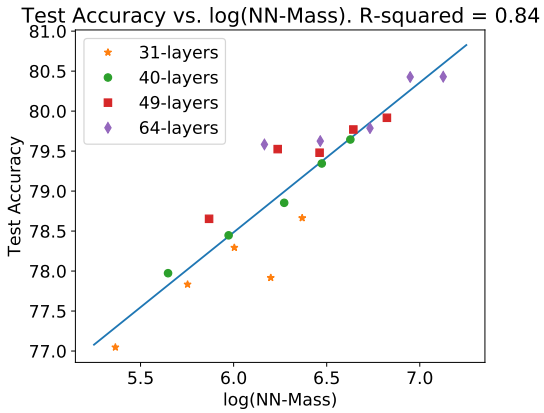


Figure 6.13: Test accuracy *vs.* log(NN-Mass) for the CIFAR-100 dataset. The R^2 value for fitting a linear regression model is 0.84 which shows that NN-Mass is a good predictor of test accuracy. All results are reported as a mean of three runs.

Table 6.2 summarizes the number of parameters and test accuracy of various models. We first train two large CNN models of about 8M and 12M parameters with a NN-Mass of 622 and 1126, respectively; both of these models achieve around 97% accuracy as shown in Table 6.2. Next, we train three compressed models: (*i*) A 5M parameter model with 40 layers and a NN-Mass of 755 (this model is the same as the highest density, 40-layer model shown in Fig. 6.5(c)), (*ii*) A 4.6M parameter model with 37 layers and a NN-Mass of 813, and (*iii*) A 31-layer, 3.82M parameter model with a NN-Mass of 856. Note that, unlike the 5M parameter model (which we partially explored in the previous section by training it for 200 epochs), we never trained the second and third compressed models before for Neural Architecture Space Exploration. In fact, we never trained any 37-layer models at all throughout this work. Hence, these models are completely new and we designed them directly based on the NN-Mass!

As evident from Table 6.2, our 5.02M parameter model reaches a test accuracy of 97.00%, while the 4.6M (3.82M) parameter model obtains 96.93% (96.82%) accuracy on the CIFAR-10 test set. Clearly, all these accuracies are either comparable to, or slightly lower ($\sim 0.2\%$) than the large baseline models, while reducing total parameters by up to $3\times$ compared to the 11.89M parameter baseline CNN. Moreover, the improvement in number of FLOPS is also up to $3\times$. Of note, since we accomplish model compression without any pruning [63, 132], quantization [51, 60], or knowledge distillation [43, 135], our proposed NN-Mass metric allows for a novel model compression technique which operates directly at the architecture-level.

Table 6.2: Exploiting NN-Mass for Model Compression on CIFAR-10 Dataset. All of our experiments are reported as mean $\pm$ standard deviation of three runs. DARTS results are reported from [66] which uses a similar setup for training.

Model	Architecture design method	#Parameters/ #FLOPS	Number of layers/ cells/long-range links (t_c)	Specialized search space?	NN-Mass	Test Accuracy
DARTS (first order) [66]	NAS	3.3M/–	–/20 cells/–	Yes	–	$97.00 \pm 0.14\%$
DARTS (second order) [66]	NAS	3.3M/–	–/20 cells/–	Yes	–	$\mathbf{97.24 \pm 0.09}\%$
Train large models to be compressed	Manual	11.89M/3.63G	64/3 cells/ [90, 170, 300]	No	1126	$97.02 \pm 0.06\%$
	Manual	8.15M/2.54G	64/3 cells/ [50,100,150]	No	622	$96.99 \pm 0.07\%$
Proposed	Directly via NN-Mass	**5.02M/1.59G**	40/3 cells/ [60,130,170]	No	755	$\mathbf{97.00 \pm 0.06}\%$
Proposed	Directly via NN-Mass	**4.69M/1.51G**	37/3 cells/ [70,140,180]	No	813	$\mathbf{96.93 \pm 0.10}\%$
Proposed	Directly via NN-Mass	**3.82M/1.2G**	31/3 cells/ [70,140,200]	No	856	$\mathbf{96.82 \pm 0.05}\%$

Finally, Table 6.2 shows CIFAR-10 results for DARTS [66], a competitive NAS baseline which first searches for a model and then trains the searched model for 600 epochs. As shown, with slightly lower 3.3M parameters, the first order DARTS achieves comparable accuracy to our proposed NN-Mass-based compressed models. Moreover, DARTS second order achieves slightly higher accuracy ($\sim 0.2\%$ higher). However, it should be noted that the search space of DARTS (like all other NAS techniques) is very specialized and utilizes many state-of-the-art innovations such as depth-wise separable convolutions [47], dialated convolutions [133], *etc.*, to reduce the number of parameters while obtaining high accuracy. In contrast, we use the regular convolutions with only concatenation-type long-range links in our work. This clearly demonstrates that NN-Mass can help us directly discover novel *architectures* with low number of parameters that achieve high accuracy; that is, *it is not just the specialized convolutions that result in models that attain high accuracy with less parameters, but also certain CNN architectures are inherently more accurate*. Therefore, the proposed characteristics such as NN-Mass can be used to design computationally-efficient architectures that achieve high accuracy even when the search space is limited to regular convolutions.

In future, NN-Mass and NN-Density can be extended to cover more architectures which have

say, branches of convolutions (similar to GoogleNet or Inception [114] *etc.*), or that explicitly have depth-wise separable convolutions (*e.g.*, MobileNets [47, 103], DARTS [66]). Hence, improved NN-Mass and NN-Density metrics that directly give us insights on which family of models (in the extended search space including separable convolutions or branches) performs better than others can be used to significantly reduce the search space of future NAS techniques.

6.5 Summary

In this chapter, we have proposed new architecture-level metrics called *NN-Mass* and *NN-Density* by modeling deep learning architectures from a complex network perspective. By merging PAC-Bayes theory with the theory of small-world networks (for the very first time), we have also theoretically proved two key properties of NN-Mass: (*i*) For a given depth and width, higher the NN-Mass, the lower the generalization error, and (*ii*) Irrespective of total number of parameters, models with similar NN-Mass yield similar test accuracy.

We have further presented extensive empirical evidence for the above theoretical insights. Specifically, we have used the proposed NN-Mass and NN-Density for *Neural Architecture Space Exploration* of deep networks. We found that for a given depth and width, models higher NN-Density achieve higher accuracy. With experiments on real datasets such as CIFAR-10/100, we have demonstrated that CNNs with similar NN-Mass indeed achieve comparable test accuracy, despite having significantly different number of parameters and layers. Finally, we have exploited these new insights to directly design compressed models which reduce parameters by up to $3\times$, while losing minimal accuracy compared to the large CNN (*e.g.*, the compressed model reaches 96.82% test accuracy *vs.* $\sim 97\%$ for large CNN on CIFAR-10 dataset).

The present work provides new directions in deep network generalization with implications for architecture search and model compression. As a future work, we plan to extend, both in theory as well as in practice, NN-Mass for networks with branches and depth-wise separable convolutions. We further plan to bridge the theory between architectural aspects of generalization presented in this work *vs.* the generalization ideas discussed in prior art (*e.g.*, we can quantify the role of architecture in overfitting by conducting experiments with randomized labels, noise-added inputs, activations, *etc.*).

Chapter 7

Conclusion and Future Work

In this chapter, we summarize the key contributions of this thesis. To this end, we present the major takeaways from the four research challenges addressed in our work: (*i*) Network Science for Traditional Machine Learning (Feature Extraction), (*ii*) Representation Learning on Small-Sample Problems, (*iii*) Network Science-Based Deep Network Model Compression, and (*iv*) Network Science for Neural Architecture Space Exploration. Finally, since our work breaks new ground in many different topics, we will describe novel future research directions towards the end of this chapter.

7.1 Conclusion

In this thesis, we have addressed several problems at the intersection of network science, machine learning, and Artificial Intelligence. We next briefly summarize the key contributions in each of the research challenges described above.

7.1.1 Network Science for Traditional Machine Learning

Towards the first research direction, we have answered the following questions:

1. How can we exploit network science to achieve effective spatiotemporal timeseries prediction, a traditional machine learning problem?
2. Can we use network science for application-specific feature extraction?

To answer these questions, we have shown that certain spatiotemporal timeseries prediction problems are characterized by complex network dynamics. Therefore, to achieve higher accuracy, machine learning models must account for this underlying network dynamics. Consequently, we have

proposed Network-of-Dynamic Bayesian Networks (NDBN) [11], a new probabilistic framework for learning over networks with known but rapidly changing structure. The fundamental research problem here was to learn parameters on rapidly changing networks which can then be used for the underlying classification/prediction problems. To demonstrate a scenario where such a learning problem can occur, we have considered a concrete case study from the computational sustainability domain, *i.e.*, to predict short-term solar energy and to quantify solar energy interdependence across a large river basin. We have shown that our proposed model achieves 8-18% RMSE for one-hour cloud fraction predictions and outperforms standard models.

The above learning problem assumed that even though the network is changing rapidly, we still know its structure. Next, we have addressed learning and prediction on a problem where the network structure is unknown and only high-dimensional raw data is available. Towards this, we have proposed a correlations-based network inference technique and a new method called K-Hop Learning [10, 14] for application-specific feature extraction. We have demonstrated the importance of our feature extraction technique on another case study in the computational sustainability domain: River network inference and highly accurate, short-term river flowrate prediction by exploiting network science-based features. Our results have shown that the proposed network-based feature extraction leads to significant improvements (57%-82%) in short-term flowrate predictions over the traditional models.

Having demonstrated the importance of network science for traditional machine learning problems such as application-specific feature extraction, we have next targeted automatic feature learning for general problems with small number of samples. Specifically, we have shown that network science concepts can also be quite useful for representation learning on many problems.

7.1.2 Representation Learning on Small-Sample Problems

Here, we have addressed representation learning for Small Data problems. Specifically, we have targeted the following questions:

1. Can network science concepts be used to automatically learn features for problems with high-dimensions and small number of samples (say, 100-1500 samples)?
2. How does lack of data impact deep learning model compression?

We have answered the above questions by first showing that *general* network science concepts such as community structure can be exploited to learn useful low-dimensional features for many general problems. More specifically, these network characteristics are hidden within the high-dimensional raw data and, hence, can be used reducing the dimensions of the initial dataset. Next, since many existing model compression techniques rely on the original training dataset, we have proposed a new technique to perform model compression in absence of real data.

In view of the above, we have first proposed FeatureNet [12], a new community-based dimensionality reduction framework for small sample problems. We have also proposed a new technique to construct a network from any general high-dimensional raw data to reveal its hidden communities. Final low-dimensional features are then learned using a representation learning framework that can account for community structure. We have demonstrated the effectiveness of FeatureNet across five, diverse application domains such as handwritten digit recognition, biology, physical science, NLP, and computational sustainability. Our experiments have shown that FeatureNet significantly outperforms many well-known dimensionality reduction techniques such as PCA, t-SNE, Isomap, *etc*.

Next, we have proposed Dream Distillation [16], a new approach to address deep learning model compression when the original training data is not available. We have called this problem space as data-independent model compression. To this end, we have used a small amount of metadata to generate synthetic images (which capture key features learned by the teacher), and then used those images for distilling knowledge to the compressed student. We have shown that models trained via Dream Distillation can achieve up to 88.5% accuracy on the CIFAR-10 without ever seeing any real data!

Overall, we have shown the importance of network science for representation learning on Small Data problems. We have next addressed network science for new directions in model compression.

7.1.3 Distributed Deep Learning Model Compression via Network Science

Towards model compression, we have focused on an important research problem – Compress a large Convolutional Neural Network (CNN) into multiple separate modules such that:

1. Compressed modules fit within the per-device memory/performance budgets.

2. Communication latency is minimized when the individual modules are distributed across a network of edge devices.
3. Loss of accuracy is minimal compared to the large CNN model.

To accomplish the above objectives, we have proposed a new paradigm called Network-of-Neural Networks [15] which compresses a large teacher model into multiple, disjoint student modules that can be distributed across a network of edge devices with minimal inter-device communication and accuracy loss over the teacher model. Towards this, we have first partitioned teacher's knowledge by proposing a network science-based technique. Then, we have used these partitions to train individual students. Extensive experiments on five well-known image classification tasks have demonstrated that NoNN achieves close to teacher's accuracy with significantly lower memory ($2.5\times$-$24\times$ gain w.r.t. teacher) and computation ($2\times$-$15\times$ fewer FLOPS w.r.t. teacher), while guaranteeing that individual modules of NoNN fit within some given memory/FLOP budget. Further, we have shown that NoNN achieves higher accuracy than several baselines. For a complete validation of our proposed approach on real hardware, we have also deployed our NoNN models for CIFAR-10 dataset on Raspberry Pi and Odroid devices. We have demonstrated $6.22\times$-$12.22\times$ improvement in performance and $12.99\times$-$14.36\times$ in energy per node w.r.t. teacher. Finally, for distributed inference on multiple edge devices, we have shown that NoNN outperforms a state-of-the-art model compression baseline by up to $33\times$ in total latency for distributed inference; this highlights the importance of communication costs when existing compressed models are distributed across multiple devices to meet memory-constraints. Ultimately, our proposed communication-aware model compression can lead to effective deployment of deep networks on multiple memory-constrained IoT-devices connected together.

This concludes our proposed network science-based model compression, where we demonstrated that network science can be used to target new directions in model compression, namely, the communication aspects involved when memory-constrained IoT-devices are connected together in a network. So far, designing the individual compressed student architectures has been largely dependent on the resource-constraints of edge devices (*e.g.*, how many layers, width, parameters to choose for individual student modules). However, many deep learning architectures yield comparable accuracies despite having different number parameters and layers. We have next addressed how network science can be exploited to identify characteristics that can lead to efficient model

architectures.

7.1.4 Network Science for Neural Architecture Space Exploration

In the final research challenge, we have addressed the following key questions:

1. What structural property of CNNs results in models with high accuracy?
2. Can we use network science to reveal architectural characteristics that indicate which family of models (with different number of layers and parameters) achieve comparable accuracy?

Indeed, such characteristics can directly allow us to design compressed models with minimal loss of accuracy over the large deep networks. Therefore, we have proposed new architecture-level metrics called *NN-Mass* and *NN-Density* by modeling deep learning architectures from a complex network perspective. Specifically, NN-Density quantifies how densely the channels in a CNN are connected to each other. On the other hand, NN-Mass quantifies the representational capacity of CNNs. We have then merged PAC-Bayes theory with the theory for small-world networks to *provably* show that (*i*) The higher the NN-Mass, the lower the generalization error, and (*ii*) Irrespective of the number of parameters and layers, models with similar NN-Mass yield similar generalization performance.

Next, we have provided extensive empirical evidence towards the above theoretical results. Specifically, we have used the proposed NN-Mass and NN-Density metrics for *Neural Architecture Space Exploration*. More precisely, we found that for a given depth and width, models higher NN-Density achieve higher accuracy. Moreover, as theoretically suggested above, models with similar NN-Mass indeed achieve comparable accuracies irrespective of number of layers and parameters. Finally, we have exploited these new network science-based insights to directly design compressed models which reduce parameters by up to $3\times$, while losing minimal accuracy compared to the large CNN. For instance, our directly designed compressed model reaches 96.82% test accuracy on CIFAR-10 dataset which is close to $\sim 97\%$ accuracy achieved by a large CNN.

As evident from our contributions, our work revolves around a central theme of how learning can benefit from network science principles. This ultimately opens up new directions of research at the intersection of representation learning and network science.

7.2 Future Work

The current work opens up several new research directions. We first discuss short-term plans followed by long-term research directions.

7.2.1 Short-Term Plans

The contributions made in this thesis can naturally be extended to address many more applications. We describe some of these ideas below.

First, our network science for traditional machine learning can target other applications such as traffic prediction. For instance, our current technique in network science-based feature extraction focuses only on river networks, which are tree-shaped. Indeed, we can extend this technique to networks with more complex topologies, *e.g.*, traffic networks; this can potentially improve the prediction of traffic flowing through a city. Similar extensions can be thought for NDBN framework.

Next, in data-independent model compression, we can validate our Dream Distillation approach on applications with medical, bio-metric, or speech data, where the data may be more privacy-sensitive. Note that, for such datasets, representations for different classes may not be fully-separable. We can also conduct more ablation studies to analyze the impact of using more real data to generate metadata (*e.g.*, instead of 10% data, what if we use 20% data, *etc.*). Further, an in-depth theoretical analysis of FeatureNet can be done to analyze its stability w.r.t. the network construction parameters.

For communication-aware model compression, we plan to address problems like how to accurately map deep network partitions to heterogeneous devices. Specifically, when the edge devices have highly variable computational budgets, what is the optimal way to distribute teacher network's knowledge across multiple students. Moreover, we can also make robustness as a primary objective of NoNN. We can further explore NoNN for other types of deep learning models such as Recurrent Neural Networks (RNNs), which can ultimately extend NoNN to other applications in speech and language domains.

Finally, for network science-based Neural Architecture Space Exploration, NN-Mass and NN-

Density can be directly extended for CNNs with branches and depth-wise separable convolutions. Similar to NoNN, we can also potentially explore NN-Mass and NN-Density for RNNs.

7.2.2 Long-Term Plans

Since this thesis introduced many novel problems, our contributions can also have long-lasting impact on the research community. We next describe how our work on data-independent model compression, communication-aware model compression, and network science-based Neural Architecture Space Exploration can be used for long-term research.

Federated Learning Federated learning is one of the most exciting new learning paradigms where the idea is to train machine learning models locally on each device (with their local data) [76, 102, 109]. Clearly, this approach has advantages related to privacy. One of the fundamental problems in Federated Learning is that the data is not independent and identically distributed (non-IID) [139]. Specifically, since the local data on each device may be skewed towards certain classes, locally training a copy of the model can result in same set of weights updating in different directions.

Currently, NoNN works only for distributed inference on edge devices. However, since NoNN involves partitioning the knowledge from a deep network, it can be easily modified to work in the federated learning setting. Specifically, if we can find a way to constrain which parts of the model get updated for which classes, this could clearly alleviate the non-IID issue. Also, an approach similar to Dream Distillation's synthetic image generation could be used to generate data from various classes to make the distribution more IID. Hence, these two ideas can be future research directions.

Neural Architecture Search In this thesis, we concluded the Neural Architecture Space Exploration problem by demonstrating how the proposed NN-Mass metric can be used to identify family of models with similar accuracy (even though they have different number of parameters and layers). We also demonstrated how, given a large CNN, NN-Mass can be used for directly designing an efficient model with minimal loss of accuracy. Once NN-Mass is extended to comprehensively cover other state-of-the-art convolutions (*e.g.*, depth-wise separable convolutions), it can

have many more applications in Neural Architecture Search (NAS) techniques [66, 100, 141, 142]. For instance, based on previous models searched, NN-Mass could potentially be used to directly identify that a given candidate model may not achieve high accuracy. Therefore, such "bad candidates" can be removed from the pool of searched models without the need to train them. Clearly, such an approach can significantly accelerate the process of architecture search.

This concludes this thesis. Overall, we have demonstrated several new problems where learning is significantly improved by using network science. We hope this comprehensive work further encourages more collaboration between network science, machine learning, and Artificial Intelligence in the future.

Bibliography

[1] Dimitris K Agrafiotis. Stochastic Proximity Embedding. *Journal of computational chemistry*, 24(10):1215–1221, 2003.

[2] R.J. Andres, T.A. Boden, and G. Marland. Monthly Fossil-Fuel CO2 Emissions: Mass of Emissions Gridded by One Degree Latitude by One Degree Longitude. *CDIAC, Oak Ridge National Laboratory, U.S. Department of Energy, U.S.A.*, 2013.

[3] Sanjeev Arora, Rong Ge, Behnam Neyshabur, and Yi Zhang. Stronger generalization bounds for deep nets via a compression approach. *arXiv preprint arXiv:1802.05296*, 2018.

[4] Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? In *Advances in neural information processing systems*, pages 2654–2662, 2014.

[5] Albert-Laszlo Barabasi. *Network Science (Chapter 3: Random Networks)*. Cambridge University Press, 2016. URL `https://bit.ly/2ONAUqQ`.

[6] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *science*, 286(5439):509–512, 1999.

[7] Albert-László Barabási and Eric Bonabeau. Scale-free networks. *Scientific american*, 288 (5):60–69, 2003.

[8] Peter L Bartlett, Dylan J Foster, and Matus J Telgarsky. Spectrally-normalized margin bounds for neural networks. In *Advances in Neural Information Processing Systems*, pages 6240–6249, 2017.

[9] M. Bazzi, M. A. Porter, and et al. Community Detection in Temporal Multilayer Networks, with an Application to Correlation Networks. *Multiscale Model. Simul.*, 14(1):1–41, Jan. 2016.

[10] Kartikeya Bhardwaj and Radu Marculescu. K-hop learning: a network-based feature extraction for improved river flow prediction. In *Proceedings of the 3rd International Workshop*

on Cyber-Physical Systems for Smart Water Networks, pages 15–18. ACM, 2017.

[11] Kartikeya Bhardwaj and Radu Marculescu. Non-stationary bayesian learning for global sustainability. *IEEE Transactions on Sustainable Computing*, 2(3):304–316, 2017.

[12] Kartikeya Bhardwaj and Radu Marculescu. Dimensionality reduction via community detection in small sample datasets. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, pages 102–114. Springer, 2018.

[13] Kartikeya Bhardwaj, HingOn Miu, and Radu Marculescu. Discovering hidden knowledge in carbon emissions data: A multilayer network approach. In *International Conference on Discovery Science*, pages 223–238. Springer, 2017.

[14] Kartikeya Bhardwaj, Dimitrios Stamoulis, Ruizhou Ding, Diana Marculescu, and Radu Marculescu. *Computational Approaches for Incorporating Short- and Long-Term Dynamics in Smart Water Networks*. CRC Press, Smart Water Grids: A Cyber-Physical Approach, Eds. P. Tsakalides, A. Panousopoulou, G. Tsagkatakis, L. Montestruque, 2018.

[15] Kartikeya Bhardwaj, ChingYi Lin, Anderson Sartor, and Radu Marculescu. Memory- and Communication-Aware Model Compression for Distributed Deep Learning Inference on IoT. In *International Conference on Hardware/Software Codesign (CODES). To appear in ACM Transactions on Embedded Computing Systems (TECS) Special Issue*, pages 1–22, 2019.

[16] Kartikeya Bhardwaj, Naveen Suda, and Radu Marculescu. Dream Distillation: A Data-Independent Model Compression Framework. *ICML 2019 Joint Workshop on On-Device Machine Learning and Compact Deep Neural Network Representations (ODML-CDNNR),* ***Oral Presentation***, pages 1–4, Jun. 2019.

[17] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[18] Alon Brutzkus, Amir Globerson, Eran Malach, and Shai Shalev-Shwartz. Sgd learns over-parameterized networks that provably generalize on linearly separable data. *arXiv preprint arXiv:1710.10174*, 2017.

[19] C. V. T. Cabral, D. O. Filho, and et. al. A stochastic method for stand-alone photovoltaic system sizing. *Solar Energy*, 84(9):1628–1636, 2010.

[20] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332*, 2018.

[21] Y. Chu, H. Pedro, M. Li, and C. Coimbra. Real-time forecasting of solar irradiance ramps with smart image processing. *Solar Energy*, 114:91–104, 2015.

[22] A. Clauset, C. R. Shalizi, and M. E. J. Newman. Power-law distributions in empirical data. *SIAM Rev.*, 51(4):661–703, 2007.

[23] John P Cunningham and Zoubin Ghahramani. Linear dimensionality reduction: Survey, insights, and generalizations. *The Journal of Machine Learning Research*, 16(1):2859–2900, 2015.

[24] Romain Dambreville and et. al. Very short term forecasting of the Global Horizontal Irradiance using a spatio-temporal autoregressive model. *Renewable Energy*, 72:291–300, 2014.

[25] H. Escrig, F. J. Batlles, and et. al. Cloud detection, classification and motion estimation using geostationary satellite imagery for cloud cover forecast. *Energy*, 55:853–859, 2013.

[26] Ronald J Evans and J Boersma. The entropy of a Poisson distribution (C. Robert Appledorn). *SIAM Review*, 30(2):314–317, 1988.

[27] Facebook. ONNX: Open Neural Network Exchange Format, 2017. URL `https://onnx.ai/`.

[28] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

[29] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Sparse inverse covariance estimation with the graphical lasso. *Biostatistics*, 9(3):432–441, 2008.

[30] Keinosuke Fukunaga and Raymond R. Hayes. Effects of sample size in classifier design. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(8):873–885, 1989.

[31] Hongyang Gao, Zhengyang Wang, and Shuiwang Ji. Channelnets: Compact and efficient convolutional neural networks via channel-wise convolutions. In *Advances in Neural Information Processing Systems*, pages 5203–5211, 2018.

[32] J. Gao, D. Li, and S. Havlin. From a single network to a network-of-networks. *National*

Science Review, July 2014.

[33] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[34] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[35] Aditya Grover and Jure Leskovec. node2vec: Scalable Feature Learning for Networks. In *KDD*, pages 855–864. ACM, 2016.

[36] O. Guez and et al. Global climate network evolves with North Atlantic Oscillation phases: Coupling to Southern Pacific Ocean. *Europhysics Letters*, 103:1–5, 2013.

[37] A. Hammer and et. al. Short-term forecasting of solar radiation: A statistical approach using satellite data. *Solar Energy*, 67(1-3):139–150, 1999.

[38] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[39] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *NIPS*, pages 1135–1143, 2015.

[40] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[41] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[42] Geoffrey Hinton and Sam Roweis. Stochastic neighbor embedding. In *NIPS*, volume 15, pages 833–840, 2002.

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

[44] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[45] V. E. Hoggatt, Chih yi Wang, R. T. Hood, J. L. Brown, and C. H. Cunkle. E1366: Two related triangles. *The American Mathematical Monthly*, 67(1):82–84, 1960. ISSN 00029890, 19300972. URL `http://www.jstor.org/stable/2308942`.

[46] Harold Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of educational psychology*, 24(6):417, 1933.

[47] Andrew G Howard and et al. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv:1704.04861*, 2017.

[48] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[49] Jeremy Howard. Imagenet in 18 minutes. 2018. Accessed: 2018-10-01.

[50] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[51] Itay Hubara and et al. Quantized neural networks: Training neural networks with low precision weights and activations. *JMLR*, 18(1):6869–6898, 2017.

[52] Gordon Hughes. On the mean accuracy of statistical pattern recognizers. *IEEE transactions on information theory*, 14(1):55–63, 1968.

[53] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[54] Anil K Jain, Robert P. W. Duin, and Jianchang Mao. Statistical pattern recognition: A review. *IEEE Transactions on pattern analysis and machine intelligence*, 22(1):4–37, 2000.

[55] Juyong Kim, Yookoon Park, Gunhee Kim, and Sung Ju Hwang. Splitnet: Learning to semantically split deep networks for parameter reduction and model parallelization. In *International Conference on Machine Learning*, pages 1866–1874, 2017.

[56] Yoon Kim and Alexander M Rush. Sequence-level knowledge distillation. *arXiv preprint arXiv:1606.07947*, 2016.

[57] M. Kivela, A. Arenas, and et al. Multilayer Networks. *Journal of Complex Networks*, 2: 203–271, 2014.

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

[59] Mandar Kulkarni, Kalpesh Patil, and Shirish Karande. Knowledge distillation using unlabeled mismatched images. *arXiv preprint arXiv:1703.07131*, 2017.

[60] Liangzhen Lai, Naveen Suda, and Vikas Chandra. Deep convolutional neural network inference with floating-point weights and fixed-point activations. *arXiv preprint arXiv:1703.03073*, 2017.

[61] Francois Laviolette. A tutorial on pac-bayesian theory. *NeurIPS 2017 Tutorial (NeurIPS 2017 Workshop on (Almost) 50 shades of Bayesian Learning: PAC-Bayesian trends and insights)*. URL `https://bit.ly/2ySQPsU`.

[62] Jure Leskovec and Julian J. Mcauley. Learning to discover social circles in ego networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 539–547. Curran Associates, Inc., 2012. URL `http://papers.nips.cc/paper/4532-learning-to-discover-social-circles-in-ego-networks.pdf`.

[63] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv:1608.08710*, 2016.

[64] Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. *arXiv preprint arXiv:1902.07638*, 2019.

[65] Yuanzhi Li and Yingyu Liang. Learning overparameterized neural networks via stochastic gradient descent on structured data. In *Advances in Neural Information Processing Systems*, pages 8157–8166, 2018.

[66] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.

[67] R Lopes, S Fenu, and T Starner. Data-free knowledge distillation for deep neural networks. *arXiv preprint arXiv:1710.07535*, 2017.

[68] Christos Louizos and et al. Relaxed quantization for discretized neural networks. *ICLR*, 2019.

[69] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9(Nov):2579–2605, 2008.

[70] Van Der Maaten. Dimensionality Reduction: A Comparative Review. *Journal of Machine Learning Research*, 10:66–71, 2009.

[71] Jiachen Mao and et al. Modnn: Local distributed mobile computing system for deep neural network. In *2017 DATE Conference*, pages 1396–1401. IEEE, 2017.

[72] Jiachen Mao, Zhongda Yang, Wei Wen, Chunpeng Wu, Linghao Song, Kent W Nixon, Xiang Chen, Hai Li, and Yiran Chen. Mednn: A distributed mobile system with enhanced partition and deployment for large-scale dnns. In *Proceedings of the 36th International Conference on Computer-Aided Design*, pages 751–756. IEEE Press, 2017.

[73] R. Marquez and et. al. Hybrid solar forecasting method uses satellite imaging and ground telelmetry as inputs to ANNs. *Solar Energy*, 92:176–288, 2013.

[74] David A McAllester. Some PAC-Bayesian Theorems. *Machine Learning*, 37(3):355–363, 1999.

[75] David A McAllester. PAC-Bayesian model averaging. In *COLT*, volume 99, pages 164–170. Citeseer, 1999.

[76] H Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, et al. Communication-efficient learning of deep networks from decentralized data. *arXiv preprint arXiv:1602.05629*, 2016.

[77] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[78] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997. ISBN 0070428077, 9780070428072.

[79] Remi Monasson. Diffusion, localization and dispersion relations on small-world lattices.

The European Physical Journal B-Condensed Matter and Complex Systems, 12(4):555–567, 1999.

[80] Alexander Mordvintsev, Christopher Olah, and Mike Tyka. Deepdream. 2015. Accessed: 2019-04-01.

[81] NASA. NLDAS-2 Noah Streamflow Dataset. *Accessed: 2016. http://ldas.gsfc.nasa.gov/nldas/NLDASnews.php*, 1979-2008.

[82] NASA. NLDAS-2 Forcing File A Dataset. *Accessed: 2016. http://ldas.gsfc.nasa.gov/nldas/NLDAS2forcing.php*, 1979-2016.

[83] NASA. Cloud Cover to Solar Radiation. *http://scool.larc.nasa.gov*, 2014.

[84] M. Newman. Modularity and community structure in networks. *PNAS*, 103(23):8577–8582, Jun. 2006.

[85] Mark Newman, Albert-Laszlo Barabasi, and Duncan J Watts. *The structure and dynamics of networks*, volume 19. Princeton University Press, 2011.

[86] Mark EJ Newman. Modularity and community structure in networks. *Proceedings of the national academy of sciences*, 103(23):8577–8582, 2006.

[87] Mark EJ Newman and Duncan J Watts. Renormalization group analysis of the small-world network model. *Physics Letters A*, 263(4-6):341–346, 1999.

[88] M.E.J. Newman. *Networks - An Introduction*. Oxford University Press, Oxford, 2010.

[89] Behnam Neyshabur, Ryota Tomioka, and Nathan Srebro. Norm-based capacity control in neural networks. In *Conference on Learning Theory*, pages 1376–1401, 2015.

[90] Behnam Neyshabur, Srinadh Bhojanapalli, David McAllester, and Nati Srebro. Exploring generalization in deep learning. In *Advances in Neural Information Processing Systems*, pages 5947–5956, 2017.

[91] Behnam Neyshabur, Srinadh Bhojanapalli, and Nathan Srebro. A pac-bayesian approach to spectrally-normalized margin bounds for neural networks. *arXiv preprint arXiv:1707.09564*, 2017.

[92] Maxwell Nye and Andrew Saxe. Are efficient deep representations learnable? *arXiv*

preprint arXiv:1807.06399, 2018.

[93] Umit Y Ogras and Radu Marculescu. " it's a small world after all": Noc performance optimization via long-range link insertion. *IEEE Transactions on very large scale integration (VLSI) systems*, 14(7):693–706, 2006.

[94] Chris Olah. The building blocks of interpretability. *Distill*, 3(3):e10, 2018.

[95] Daniel S Park, Jascha Sohl-Dickstein, Quoc V Le, and Samuel L Smith. The effect of network width on stochastic gradient descent and generalization: an empirical study. *arXiv preprint arXiv:1905.03776*, 2019.

[96] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. DeepWalk: Online Learning of Social Representations. In *KDD*, pages 701–710. ACM, 2014.

[97] Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*, 2018.

[98] Ariadna Quattoni and Antonio Torralba. Recognizing indoor scenes. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 413–420. IEEE, 2009.

[99] Sarunas J Raudys, Anil K Jain, et al. Small sample size effects in statistical pattern recognition: Recommendations for practitioners. *IEEE Transactions on pattern analysis and machine intelligence*, 13(3):252–264, 1991.

[100] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V Le, and Alexey Kurakin. Large-scale evolution of image classifiers. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2902–2911. JMLR. org, 2017.

[101] J. Robinson and A. Hartemink. Learning Non- Stationary Dynamic Bayesian Networks. *Journal of Machine Learning Research*, 11:3647–3680, 2010.

[102] Anit Kumar Sahu, Tian Li, Maziar Sanjabi, Manzil Zaheer, Ameet Talwalkar, and Virginia Smith. On the convergence of federated optimization in heterogeneous networks. *arXiv preprint arXiv:1812.06127*, 2018.

[103] Mark Sandler and et al. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. *arXiv:1801.04381*, 2018.

[104] Andrew M Saxe, James L McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013.

[105] Ludwig Schubert, Christopher Olah, Alexander Mordvintsev, Ian Johnson, Arvind Satyanarayan, and et al. Tensorflow lucid. 2018. Accessed: 2019-04-01.

[106] B.M. Shahshahani and D.A. Landgrebe. The effect of unlabeled samples in reducing the small sample size problem and mitigating the Hughes phenomenon. *IEEE Transactions on Geoscience and Remote Sensing*, 32(5):1087–1095, Sept. 1994.

[107] W. X. Shen. Optimal sizing of solar array and battery in a standalone photovoltaic system in Malaysia. *Renewable Energy*, 34(1):348–352, 2009.

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

[109] Virginia Smith, Chao-Kai Chiang, Maziar Sanjabi, and Ameet S Talwalkar. Federated multi-task learning. In *Advances in Neural Information Processing Systems*, pages 4424–4434, 2017.

[110] Suraj Srinivas and R Venkatesh Babu. Data-free parameter pruning for deep neural networks. *arXiv preprint arXiv:1507.06149*, 2015.

[111] Dimitrios Stamoulis, Ruizhou Ding, Di Wang, Dimitrios Lymberopoulos, Bodhi Priyantha, Jie Liu, and Diana Marculescu. Single-path nas: Designing hardware-efficient convnets in less than 4 hours. *arXiv preprint arXiv:1904.02877*, 2019.

[112] Steinhaeuser, et al. Multivariate and multiscale dependence in the global climate system revealed through complex networks. *Clim Dyn*, 39:889–895, 2012.

[113] STMicro. Datasheet for Arm-Based Microcontroller with up to 512KB total storage (including FLASH memory). Product Page: `https://bit.ly/2I5ZSMR`. Datasheet, 2018. URL `https://bit.ly/2Kz8ehD`.

[114] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[115] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2820–2828, 2019.

[116] Jian Tang and et al. LINE: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*, pages 1067–1077, 2015.

[117] Zhiyuan Tang, Dong Wang, and Zhiyong Zhang. Recurrent neural network training with dark knowledge transfer. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5900–5904. IEEE, 2016.

[118] Joshua B Tenenbaum and et al. A Global Geometric Framework for Nonlinear Dimensionality Reduction. *Science*, 290(5500):2319–2323, 2000.

[119] The World Bank. GDP Data for Suriname and Sweden. World Development Indicators, Retrieved from http://data.worldbank.org, 1960-1980.

[120] UCAR. Research Data Archive, Computational and Information Systems Laboratory. *National Center for Atmospheric Research, http://rda.ucar.edu/*, 2015.

[121] Xiao Wang, Peng Cui, Jing Wang, Jian Pei, Wenwu Zhu, and Shiqiang Yang. Community preserving network embedding. In *AAAI*, pages 203–209, 2017.

[122] Z. Wang and et. al. Time Varying Dynamic Bayesian Network for Non-Stationary Events Modeling and Online Inference. *IEEE Transactions on Signal Processing*, 2011.

[123] Duncan J Watts and Steven H Strogatz. Collective dynamics of small-worldnetworks. *nature*, 393(6684):440, 1998.

[124] Duncan J Watts and Steven H Strogatz. Collective dynamics of small-worldnetworks. *nature*, 393(6684):440, 1998.

[125] Peter Welinder, Steve Branson, Takeshi Mita, Catherine Wah, Florian Schroff, Serge Belongie, and Pietro Perona. Caltech-ucsd birds 200. 2010.

[126] Mitchell Wortsman, Ali Farhadi, and Mohammad Rastegari. Discovering neural wirings. *arXiv preprint arXiv:1906.00586*, 2019.

[127] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong

Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 10734–10742, 2019.

[128] Lingxi Xie and Alan Yuille. Genetic cnn. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1379–1388, 2017.

[129] Saining Xie, Alexander Kirillov, Ross Girshick, and Kaiming He. Exploring randomly wired neural networks for image recognition. *arXiv preprint arXiv:1904.01569*, 2019.

[130] X. Xuan and K. Murphy. Modeling changing dependency structure in multivariate time series. *24th International Conference on Machine Learning*, pages 1055–1062, 2007.

[131] Makoto Yamada and et al. High-dimensional feature selection by feature-wise kernelized lasso. *Neural computation*, 26(1):185–207, 2014.

[132] Tien-Ju Yang and et al. Designing energy-efficient convolutional neural networks using energy-aware pruning. *arXiv:1611.05128*, 2016.

[133] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015.

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

[135] Sergey Zagoruyko and Nikos Komodakis. Improving the performance of convolutional neural networks via attention transfer. *ICLR*, 2017.

[136] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016.

[137] Xiangyu Zhang and et al. Shufflenet: An extremely efficient convolutional neural network for mobile devices. *CoRR*, abs/1707.01083, 2017.

[138] Yundong Zhang, Naveen Suda, Liangzhen Lai, and Vikas Chandra. Hello edge: Keyword spotting on microcontrollers. *arXiv preprint arXiv:1711.07128*, 2017.

[139] Yue Zhao, Meng Li, Liangzhen Lai, Naveen Suda, Damon Civin, and Vikas Chandra. Federated learning with non-iid data. *arXiv preprint arXiv:1806.00582*, 2018.

[140] Zhuoran Zhao, Kamyar Mirzazad Barijough, and Andreas Gerstlauer. Deepthings: Distributed adaptive deep learning inference on resource-constrained iot edge clusters. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(11):2348–2359, 2018.

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

[142] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018.