Unlocking the potential of neural networks in resource and data constrained environments

by

Otkrist Gupta

S.M., Massachusetts Institute of Technology (2012)

Submitted to the Program in Media Arts & Sciences,
School of Architecture and Planning,
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Media Arts & Sciences

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2018

Signature redacted

Author .....
Program in Media Arts & Sciences,
School of Architecture and Planning,
May 4, 2018

Signature redacted

Certified by .....
Ramesh Raskar
Associate Professor
Program in Media Arts & Sciences
Thesis Supervisor

Signature redacted

Accepted by .....
Tod Machover
Academic Head, Program in Media Arts and Sciences

Unlocking the potential of neural networks in resource and data constrained environments

by

Otkrist Gupta

Submitted to the Program in Media Arts & Sciences,
School of Architecture and Planning,
on May 4, 2018, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Media Arts & Sciences

Abstract

Data driven methods based on deep neural networks (DNNs) have ushered in a new era in the field of machine learning computer vision. Conventional algorithmic approaches are being replaced by end-to-end deep learning systems that can leverage big data. Deep learning has begun revolutionizing human centric fields such as health-care and finance, finding its way into automated screening and diagnoses. At present, developing and training artificial neural network architectures requires both human expertise and labor, requiring millions of labeled data-points to train and hours of engineering effort to develop best performing architectures.

In this dissertation, my goal is to make deep learning more ***accessible*** by developing algorithms for low shot learning (learning from a few examples). This work includes new semi-supervised approaches to learn from unlabeled datasets with only a fraction of labeled examples, deep learning methods to learn from generated data using simulation based techniques, and learning to optimize neural networks for smaller data sets. Specifically, this dissertation focuses on two proposed directions which will contribute towards both technical and conceptual advances in literature.

- How can we use **invariant-based** approaches when training from small datasets ?
- How to enable training from **multiple** data sources carrying very small amounts of data ?
- How to use **meta-modeling** approach to automatically generate high-performing DNNs ?

To address these questions, this dissertation describes machine learning algorithms as follows (a) an action recognition autoencoder which learns over very small datasets; (b) an algorithm to train deep neural networks over multiple entities; (c) a meta-modeling approach to automatically generate high-performing architectures. We also provide a dataset of neural network topologies used for predicting accuracy of a deep neural network.

Thesis Supervisor: Ramesh Raskar
Title: Associate Professor
Program in Media Arts & Sciences

Unlocking the potential of neural networks in resource and data constrained environments

by

Otkrist Gupta

The following person served as a reader for this thesis:

Signature redacted

Reader: _____

Dan Raviv
School of Electrical Engineering, Faculty of Engineering
Tel-Aviv University, Israel

Unlocking the potential of neural networks in resource and data constrained environments

by

Otkrist Gupta

The following person served as a reader for this thesis:

Signature redacted

Reader: _____ _____

David Cox

Assistant Professor of Molecular & Cellular Biology & Computer Science

Center for Brain Science, Harvard University

Acknowledgments

I would like to express my exceeding gratitude to the following people: my advisor, Ramesh Raskar, for his support and guidance; to Dan Raviv for being an amazing mentor and helping me learn machine learning from fundamentals; to David Cox for valuable career advice and suggestions on the thesis; to Nikhil Naik who was a great friend and source of valuable advice.

I greatly appreciate my group members and collaborators for their invaluable support and feedback. Thanks to Dan McDuff for helping me enter in the field of affect recognition and Hisham Bedri for his invaluable help in gesture recognition over WiFi. Abhimanyu Dubey was a great person to take advice on about the caffe infrastructure. Tristan Swedish and Guy Satat were invaluable collaborators and friends. In addition, I would like to express my thanks to Ayush Bhandari for being there for discussions on scientific merits and impact of work; to Anshuman Das for a great collaboration in hyperspectral imagery. Thank you to Barmak Heshmat and Micha Feign for their invaluable feedback on signal processing and hardware projects; to Pratik Shah for guidance on health related projects. Thank you to Greg Yauney, Aman Rana, Matt Tancik, Bowen Baker and Mrinal Mohit for being lively collaborators on some of these projects. I would like to express my thanks to Pratik Kapasi, Vruddhi Shah, Leo Pauly and Rohit Bhaskar for their invaluable input in facial video annotation tools.

I wish to express my gratitude to my parents, Ashok Kumar Gupta and Suman Gupta, for their considerable support and invaluable advice without which I would have never finished. I would also like to acknowledge and express my thanks to my wife, Marcia Gupta, for her partnership and support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The fields of Artificial Intelligence and Computer Vision have long aspired to synthetically create human levels of perceptive and cognitive abilities. Defining intelligence remains a challenging task and computer science researchers often rely upon indirect methods such as the *Turing Test*[1] when identifying what it means to be a human level AI. In hopes of matching the diversity and creativity of a human brain, we have resorted to borrowing from biology, genetics and human physiology when designing AI algorithms. In fact, biologically inspired methods such as *perceptrons* and connectionist architectures have proven to be effective at solving basic artificial intelligence tasks[2, 3]. Similarly, experiential techniques such as decision trees[4] have been shown to be quite effective in modeling the complex nuances of real world phenomenon.

A *perceptron*[5] forms a singular biologically inspired computational unit which is capable of activating based on several inputs. Over the years, many flavors of perceptron algorithm have emerged, aiding in the design and creation of *Support Vector Machines*[6]. Support Vector Machines (or SVMs for short) have well defined convergence characteristics and can greatly enhance classification accuracy by introducing nonlinearities into the training and test procedures. Some of the theoretical limitations when using a singular perceptron are discussed in [7], arguing that some classes of problems remain intractable under the perceptron framework. This fundamental issue in the *generalization* power of perceptrons was one of their greatest hurdles – solved only recently when multi-layer connectionist architectures were introduced as a viable alternative to them. Adding multiple layers in a network with multiple neurons in each layer, greatly enhances the representative power

of neural networks, allowing them to represent almost any complex function with sufficient ease[8]. Such multi-layered perceptron architectures are one of the closest counterpart to the human brain in both form and functioning and have proven to be dramatically robust and efficient in dealing with real world AI problems.

1.1 Advent of deep neural networks

Modern improvements in computing technology have helped accelerate research in the fields of artificial intelligence and machine learning. The rise of the Internet has enabled creation of very large datasets from which machines can learn – there would be no ImageNet[9] without the Internet coming first. Smartphones have become ubiquitous and have changed the way we produce and consume data. Looking at the hardware advancements over last decade, computing speeds have grown exponentially. Volatile memory and hard drive have shown similar growth with persistent storage expanding by several orders of magnitudes. Advances in sensing technologies has made a huge impact in making very small and low power sensors for measuring physiology and biometrics. Advent of such advancements has enabled us to study human behavior and societal problems at a large scale and follow the trajectories of evolution of people at micro and macro levels. Analyzing data from such sources might be crucial in devising impactful interventions that can help improve human lives.

Training of multi-layered neural networks (DNNs), was surmised to be a computationally infeasible task - until recently when [10] introduced a new method for fast training of *Restricted Boltzmann Machines* using energy based methods. *Contrastive divergence learning* and *convolutional architectures* have greatly simplified computational burden when training multi-layered perceptrons; such models have become a strong driving force in modern computer vision and excel at object classification[9], segmentation and recognition[11, 12]. In most cases DNNs have proven to be more successful than algorithms engineered specifically to solve such problems. Success of deep neural networks has begun influencing fields such as finance, biomedicine and health-care, while mustering the possibility of *better-than-human* accuracies at daunting tasks in these fields.

Learning a deep neural network can be data and resource intensive, because it involves learning the conditional distributions of multiple hidden layers given its input and output

data vectors. Training convolutional neural networks using a large labeled dataset has become the standard methodology when learning neural network layers [13]. Modern algorithms learn deep neural networks using greedy strategies involving training one layer at a time while using unsupervised learning and RBMs [10], requiring specialized software and hardware resources.

1.2 Data and resource constraints in DNN training

Conventional supervised training methods may suffer from multiple challenges requiring human intervention at large scale. The first challenge involves accumulation of labeled datasets at a large scale. Accumulation of large amounts of labeled data can be expensive and may suffer from risks and responsibilities associated with storing training data in a centralized location. For example, the original ImageNet[9] dataset included over one million images spanning one thousand classes, which were collected and labeled using crowdsourced labeling techniques. The data and resources required to create such massive datasets continues to be a challenge when training deep neural networks on other problem domains.

Large expert labeled dataset, can be hard to obtain due to privacy converns. Publicly available datasets tend to be very small, making it harder to use them for deep learning, thereby creating a need to develop specialized AI algorithms that can learn from multiple data sources without the need for data and resource aggregation. When deploying, real world models need to work under varying conditions of camera and lighting requiring development of invariant architectures. Deploying such architectures requires development of methods for easy scalability and distributed computation.

These methodologies also involve expert intervention when designing topologies and selecting hyperparameters, especially when engineering algorithms over new problem domains and data modalities. While neural networks trained on large amounts of labeled data generalize well to randomly selected validation sets, they may overfit to the data distribution leading to poor real world performance under real world scenarios. Removing these challenges continues to be an area of active research in machine learning, as we describe in the upcoming chapters.

1.3 Thesis roadmap

Over the upcoming chapters we present algorithms that will aid in training of neural networks while reducing the resource burden. Our second chapter includes techniques that reduce expert intervention on data labeling side. It describes *low shot* learning techniques involving new topologies to learn invariants over the video data, and semi-supervised approaches while learning from very small number of examples. The third chapter describes *meta-modeling* techniques that reduce expert intervention on the algorithm side, reducing effort required when hand crafting topologies. We will also introduce a method that enables distributed learning over multiple entities, without direct data sharing. We demonstrate how to incorporate *distributed learning* in the network itself using a specialized layer that connects the local neural network to other nodes, while jointly optimizing the neural network over several entities. We present an overview of these methods in following sections.

Chapter 2: Learning invariants and semi-supervised application[1]

Semi-supervised methods are one of the most well studied approaches for learning from unlabeled data in absence of labeled examples[14, 15]. In this thesis we will demonstrate how to use a semi-supervised approach for facial expression recognition using a deep neural network, by combining an autoencoder with a classification loss function; and training both of them in parallel. We develop a multilayer *stacked* autoencoder which is trained hierarchically, by adding layers in an iterative fashion. Such hierarchical contrastive divergence minimization techniques have been shown to work well when training autoencoders over large datasets[16]. While training the stacked autoencoder, we adaptively add layers to the autoencoder, train the resulting neural network, and use the produced weights as initializations for the next step. We fine tune all of the layers once the neural network weights have converged to an acceptable value.

We also collected a diverse dataset comprising of 162 million facial images over 6.5 million video clips with 25 frames each (1 second duration). We used public sources such as local broadcast television networks and YouTube for data acquisition. We used Viola-Jones

[1] An abridged version of this work appeared as "Multi-velocity neural networks for facial expression recognition in videos", Gupta, Otkrist and Raviv, Dan and Raskar, Ramesh. IEEE Transactions in Affective Computing (2017).

face detector to find and segment out the faces and isolated clips which contained more than 25 overlapping facial frames. We localized landmarks for each frame using a deformable model for the face and detected the facial pose by fitting a 3D model to the landmarks, which allowed us to restrict the dataset to videos which contain faces tilted less than 30 degrees and remove any faces looking sideways. We removed clips with static gestures or those where the faces were rapidly altering, either due to some high speed movement or simply due to appearance of a different face. We achieved this by blurring the clips and calculating the difference between consecutive frames. To our knowledge this is the biggest facial dataset reported in literature, and we plan to make it public[2].

Since video data is extremely high dimensional we rely on a deep *convolutional* autoencoder to extract meaningful features from this data by embedding it into $\mathbb{R}^{4096}$. Our action autoencoder comprises of stacked *convolutional* layers with spatio-temporal convolutions for learning deep features and reducing the dimensionality of the data. We extended the convolution layers in the time domain and use slow fusion model which slowly combines temporal information in successive layers[17]. The deconvolution layers are extended in time as well and reverse the slow fusion generating temporal features successively. Once trained, the middle layers of autoencoder are combined with a softmax loss function for classification purposes i.e. we propose a semi-supervised approach for gesture classification using a deep neural network, by combining an autoencoder with a classification loss function, and training both of them in parallel. This semi-supervised methodology allows us to learn from dramatically fewer labeled data points.

Learning Invariants[3]

Invariants in computer science refer to conditions that continue to be true during the course of execution of a program. Learning invariants in computer vision and machine learning refers to developing features which are robust to modification in translation or rotation or other image properties. One of the main challenges in action recognition is related to assigning similar classification to objects at different velocities. In this thesis we propose to learn the velocity of the sequence in parallel to its classification by adaptive temporal inter-

[2]An abridged version of this work appeared in "Real-Time Physiological Measurement and Visualization Using a Synchronized Multi-Camera System", Gupta, Otkrist and McDuff, Dan and Raskar, Ramesh. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops (2016).

[3]An abridged version of this work appeared as "Illumination invariants in deep video expression recognition", Otkrist Gupta, Dan Raviv, Ramesh Raskar, The Journal of Pattern Recognition Society (2017)

polation. Our multi-velocity autoencoder consists of three action autoencoders combined together to access temporal features for different velocities. We achieve this by adding a convolution layer as the first layer which uses cubic b-spline interpolation to *slow down* the video and generate intermediate frames inside the neural network itself.

We will also demonstrate how to induce scale invariance to pixel intensities by adding additional layers as an illumination invariant neural network in the beginning of the semi-supervised learner. The illumination invariant layers include a convolutional layer, an absolute value layer, a reciprocal layer followed by a Hadamard product layer combined to achieve a normalization function $(\frac{\bar{x}}{(\alpha|\bar{F}(\bar{x})|+\beta)^{\gamma}})$. Scale invariance is achieved by applying element wise multiplication between the output layers of proposed architecture and the original input layer. The convolutional layer is used to emulate various types of filters and select the best performing functions, the rest of hyperparameters are selected using standard grid search.

While semi-supervised methods and transfer learning still dominate low shot learning approaches, a new set of challenges apply to problems which lack high quality data samples. A way around this involves using generative approaches to *virtually up-sample* input space[18] thereby generating fresh training samples. In this thesis, we demonstrate how to use only *synthesized* data to train a CNN that is both invariant to changes in forward model physical parameters and is able to correctly classify hidden objects behind scattering media. Resulting neural network robustness depends on the search space of underlying generative models using Markov Chain Monte Carlo (MCMC) model that simulates any real world action. Each data point in the dataset corresponds to a specific instance of the target measured by a system which is defined by a set of random model parameters. Varying model parameters in the training data allows the CNN to be invariant to changes in those parameters, while generating more diverse data. The simulated data points are used to train topology designed specifically for 3D space time tensor to learn target, calibration and noise related invariants and help learn in absence of real world data.

Chapter 3: Generating neural network topologies[4]

In this thesis, we will demonstrate an algorithmic learning agent that selects optimal neural network topologies on small and medium sized datasets. We model the layer selection process as a Markov Decision Process with the assumption that a well-performing layer in one network should also perform well in another network, based on the hierarchical nature of the feature representations learned by neural networks with many hidden layers[19]. We define the search space for optimal hypothesis using compressed notations for neural network topologies and use a context free grammar to generate possible topological instances. Each sample of topology selected at random is trained for a fixed duration and its validation accuracy is used to make informed guess about which topology should be chosen next.

Since the topology space thus defined is potentially infinite, we discretize and prune the search space using our learning from layer engineering from earlier chapters. We then use the context free grammar to define a directed acyclic graph or a tree. We pose the problem of selecting each layer as a probabilistic process and use standard reinforcement learning based methods to further optimize the selection of optimal topologies. We use the space thus defined to traverse a wide array of topologies and present a dataset of trained topologies to allow for further inference and meta learning[5].

Chapter 4: Distributed learning approaches

To address the issue of data scarcity in training and deployment of neural network-based systems, we will develop a new technique to train deep neural networks from multiple data sources. Specifically we address the problem of training a deep neural network over several data entities (Alice(s)) and one supercomputing resource (Bob). While each data source (Alice) has only a tiny amount of data, we want to efficiently train neural networks using several such sources. As a research plan I will test out the method for which part of layers can go with Alice and which part can go with Bob and attempt to train a U shaped architecture when Bob has middle part and Alice has outer parts. I will also demonstrate

[4] An abridged version of this work appeared as "Designing Neural Network Architectures using Reinforcement Learning", Baker, Bowen and Gupta, Otkrist and Naik, Nikhil and Raskar, Ramesh. International Conference on Learning Representations (2017).

[5] An abridged version of this work is being reviewed as "Accelerating Neural Architecture Search using Performance Prediction", Gupta, Otkrist* and Baker, Bowen* and Naik, Nikhil and Raskar, Ramesh. Under Review International Conference on Learning Representations (2018).

	Invariants	Architecture Selection	Distributed Learning
Training with finite labeled data	✓	×	✓
Manual engineering of topology	✓	✓	×
Computational and data sharing constraints	×	✓	✓

Table 1.1: Summary of methods and applications we will address through this thesis.

that the method works on datasets such as CIFAR, MNIST and ImageNet[6].

Chapter 5: Conclusions

We conclude this thesis with a discussion on making machine learning more accessible and human centric while reducing the resources required. We will also discus how generative approaches can help improve machine learning pipelines in real world. We summarize the key insights on how to introduce invariants in machine learning, reduce labeled data requirements and automate machine learning pipelines. We discuss the potential implications of this work in the advancement of the field of artificial intelligence and machine learning.

1.4 Summary of thesis contributions

While multi-layered perceptrons have become the new state-of-the-art in most visual understanding tasks, training a neural network can be challenging because of multiple resource constraints. These include engineering constraints that involve need for massive amounts of labeled data, and engineering effort required to build and tweak neural network configurations. These also involve operational constraints which include need to build neural networks that are invariant to real world conditions, and resource considerations when sharing data from several data repositories.

In this thesis I will focus on learning optimizing deep neural architectures to work under data and resource constraints. Broadly speaking my work has 3 different aspects described as follows - specialized neural ***network layers*** for training from few examples and learning invariants[20, 21], using ***generative methods*** for architectures [22, 23] and data[24] to increase accuracy and distributed learning methods to allow training from multiple entities [25].

[6] An abridged version of this work has been filed as a patent in "Secure Training of Deep Neural Networks", Gupta, Otkrist and Raskar, Ramesh. Patent Filed 18864T MIT (2017).

Key contributions of this thesis include

1. Learning invariants with small amounts of labeled data
2. Meta-modeling algorithms to automate neural network design
3. Distributed learning approaches to allow training from multiple entities

In my work I demonstrate how to use velocity and intensity invariants to improve accuracy when training with small datasets of video clips [20, 26]. I develop neural networks which are trained on simulated data alone and can be used to perform real world inference. I will describe custom techniques for generating neural network architectures when training over small datasets. I also develop methods which enable distributed training of neural networks, allowing us to pool datasets from multiple sources.

Finally, I will make the following new datasets publicly available to researchers through interactive online resources: (i) a dataset for facial video clips containing millions of face images annotated for facial landmark locations, mined using public sources, and (ii) a dataset of neural network topologies used for predicting accuracy of a topology before training.

Chapter 2

Learning with fewer examples using deep invariant learning

With advances in convolutional neural networks, we have seen neural networks being applied to video classification [17, 27] and even facial expression recognition [28, 29]. In this chapter we apply convolutional neural networks for recognizing and classifying human gestures, using a diverse array of several different neural network architectures[1]. We obtain high level information in both space and time by implementing 4$\mathbb{D}$ convolutional layers and training an autoencoder on videos. We tackle the problem of a small size labeled dataset using generative means such as autoencoders, and present several new layers with invariance properties in temporal and illumination domains. We compare our methods to multiple techniques and datasets, as well as on our own collected data, and we report competitive results in almost every category. We also hope to empower researchers in this area by providing them with a huge dataset which can help them build even bigger and better deep neural networks, while eliminating the need to spend several months required to acquire a dataset of such proportions.

Over the past decade, algorithms for training deep neural networks have dramatically evolved, allowing us to train multi layered perceptron architectures more efficiently [10, 30]. Recently, deep neural networks have been shown to perform well on classification tasks on images and videos, outperforming most traditional learning systems [9]. Such

[1] An abridged version of this work appeared as "Multi-velocity neural networks for facial expression recognition in videos", Gupta, Otkrist and Raviv, Dan and Raskar, Ramesh. IEEE Transactions in Affective Computing (2017).

models have become a strong driving force in modern computer vision and excel at object classification [9], segmentation and facial recognition [11]. However these applications in gesture recognition were not deep enough or relied upon complex preprocessing involving other feature extraction techniques like PCA or Fisherface. In this work we hope to remove some of these preprocessing steps by building end to end pipelines driven entirely by the neural networks.

2.1 Related work

Deep neural networks have proven to be an effective tool to classify and segment high dimensional data such as images[9], audio and videos[17, 27]. With advances in convolutional neural nets, we have seen neural nets applied to a wide variety of specialized computer vision tasks such as object instance retrieval, scene image retrieval, classification of buildings, bird categorization[13]. Deep neural nets have triumphed over traditional vision algorithms, thereby dominating fields which involve high dimensional data classification, to the point that given enough data its almost always possible to train a deep neural network which will perform better than model driven techniques.

Richness of data is probably one of the main reasons why neural nets report such impressive predictive results in almost every field, but it is also extremely hard to collect and label such datasets. While human beings learning new concepts can often generalize successfully from just a single example, machine learning algorithms typically require tens or hundreds of examples to perform with similar accuracy[2]. Human beings can also use learned concepts in richer ways than conventional algorithms for action, imagination, and explanation unlike machines[31]. In the following sections we discuss some of these methods useful in learning from a few labeled data points bridging the gap between man and machine.

Learning from a few examples

While most neural network applications rely on separate data and labels for learning, it is possible to learn from the data itself by using *unsupervised learning* methodologies[32]. Unsupervised learning in neural networks can be achieved using an *autoencoder*, a neural network capable of learning by setting its output values equal to its inputs. A trivial autoencoder could just contain one hidden layer and learn the identity function. However more

interesting arrangement of neurons can be used for compressing data in lower dimensional manifolds[33], and learning hidden sparse parametric representations to generate highly non-linear *embeddings*. An autoencoder can have several layers, with each layer projecting data in a different dimension until we reach the output layer which has the same dimension as the input layer[10].

There are several types of autoencoders, for example convolutional autoencoders contain convolutional and de-convolutional neural network layers to learn *sparse* hierarchical representations[34]. Stacked denoising autoencoders (SDEs) which attempt to reconstruct data from partial or noisy data, are capable of learning complex gabor filters[35]. Stacked autoencoders can be trained layer by layer for better results, faster convergence and ease of training[10]. In this thesis we will use convolutional neural networks with layer-by-layer training for initializing weights, please refer to Hinton et. al. [32] for a detailed mathematical explanation on how autoencoders work.

Learning invariants

Even though deep neural nets are notorious for high quality results, training a deep neural network can be challenging because of well known data requirements. A way around this is to use learn invariants for feature extraction or weights initialization[33], followed by fine tuning over a smaller labeled dataset. This issue can also be solved using embeddings in lower dimensional manifold[36, 37] or pre-training using pseudo labels[38] thereby requiring fewer number of labeled samples. Approaches based on semi-supervised learning have shown to work for smaller labeled datasets[39] and techniques using deep neural nets to combine labels and unlabeled data in the same architecture[40, 41] have emerged victorious. Simulated and hand drawn data has also been shown to perform well when training deep neural nets for pose estimation [42]. Similar approaches have been shown to work well for learning invariants while performing video classification [43].

Recognition of gesture and physiology

Human beings, as social animals, rely on a vast array of methods to communicate with each other in the society. Non-verbal communication, that includes body language and expressions, is an essential aspect of interpersonal communication. In fact, studies have shown that non-verbal communication accounts for more than half of all societal interactions

[44]. Studying facial expressions is therefore of vital importance in fields like sociology, psychology and automated recognition of expressions can be applied towards creating more user affable software and user agents in these fields.

Automatic expression recognition has wide implications in the field of human computer interaction. As technology progresses, we spend large amounts of our time looking at screens, interacting with computers and mobile phones. In spite of their wide usage, majority of software interfaces are still non-verbal, impersonal, primitive and terse. Adding emotion recognition and tailoring responses towards users emotional state can help improve human computer interaction drastically [45, 46] and help keep users engaged. Last two decades have seen some innovation in this area [47, 48, 49] such as humanoid robots for example *Pepper* which can both understand and mimic human emotions.

Modeling and parameterizing human faces is one of the most fundamental problems in computer graphics [50]. Understanding and classification of expressions from videos can have applications towards better modeling of human faces in computer graphics and human computer interaction. Accurate characterization of face geometry and muscle motion can be used for both expression identification and synthesis [51, 52] with applications towards computer animation [53]. Such approaches combine very high dimensional facial features from facial topology and compress them to lower dimensions using a series of parameters or transformations [54, 55]. This chapter demonstrates how to use deep neural networks to reduce dimensionality of high information facial videos and recover the embedded temporal and spatial information by utilizing a series of stacked autoencoders.

Machine learning techniques such as Support Vector Machines have been used for facial expression recognition given the movement of facial fiducial points [56, 57] achieving real time performance [58]. Many of these techniques involve a pipeline with multiple phases - face detection and alignment, feature extraction/landmark localization and classification as the final step. Other interesting approaches [59, 60] we should mention are based on temporal features [61, 52], and multiple kernels [62], action units [63, 64], as well as emotion recognition from speech [65, 66]. We will compare our method against some of those approaches in section 2.4.

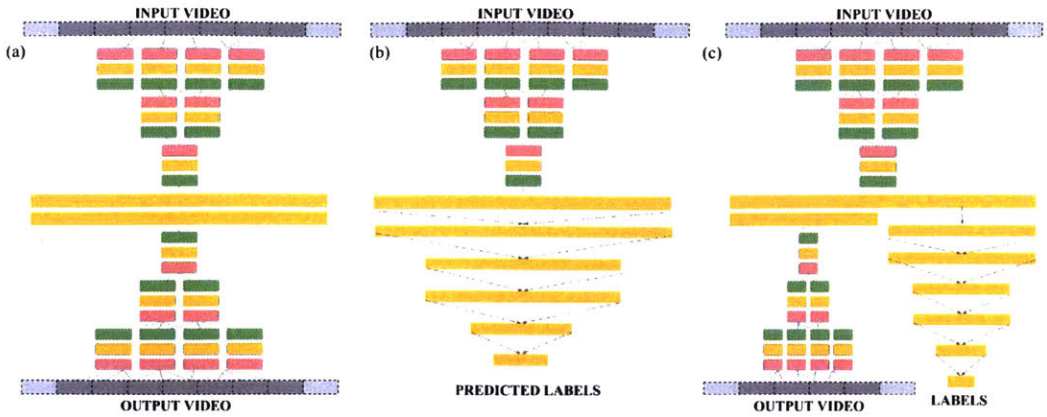


Figure 2-1: Schematic representation of deep neural networks for supervised and unsupervised learning. We use pink boxes to denote convolutional layers, yellow boxes denote *rectified linear unit* layers and green boxes indicate normalization layers. Our technique combines unsupervised learning approaches (a) with labeled prediction (b) to predict expressions using massive amounts of unlabeled data and few labeled samples.

2.2 Neural networks on video data

Our facial expression recognition pipeline comprises of Viola-Jones algorithm for face detection followed by a deep convolutional neural network for predicting expressions. The deep convolutional network includes an autoencoder combined with a predictor which relies on the semi-supervised learning paradigm. The autoencoder neural network takes videos containing 9 frames of size 145×145 as input and produces $145 \times 145 \times 9$ tensor as output. Predictor neural net sources innermost hidden layer of autoencoder and uses a cascade of fully connected layers accompanied by the softmax layer to classify expressions. Since videos can have different sizes and durations they need to be resized in temporal and spatial domain using standard interpolation techniques. The network topologies and implementation are describe henceforth.

2.2.1 Action autoencoder

Stacked autoencoders can be used to convert high dimensional data into lower dimensional space which can be useful for classification, visualization or retrieval [67]. Since video data is extremely high dimensional we rely on a deep convolutional autoencoder to extract meaningful features from this data by embedding it into $\mathbb{R}^{4096}$. The autoencoder topology is inspired by ImageNet [9] and comprises of convolutional layers gradually reducing data dimensionality until we reach a fully connected layer. Central fully connected layers are followed by

a cascade of deconvolutional layers which essentially invert the convolutional layers thereby reconstructing the input tensor ($\mathbb{R}^{145\times145\times9}$). The complete autoencoder architecture can be described in following shorthand $C(96,11,3) - N - C(256,5,2) - N - C(384,3,2) - N - FC(4096) - FC(4096) - DC(96,11,3) - N - DC(256,5,2) - N - DC(384,3,2)$. Here $C(96,11,3)$ is a convolutional layer containing 96 filters of size 11×11 in spatial domain and spanning 3 frames in temporal domain. N stands for local response normalization layers, DC stands for deconvolutional layers and $FC(4096)$ stands for fully connected layers containing 4096 neurons. This is only a subset of a more comprehensive vocabulary which is capable of generating most modern state-of-the-art architectures, as covered in detail in section 3.2.

In the same way that spatial convolutions consolidate nearby spatial characteristics of an image, we use the slow fusion model described in [17] to gradually combine temporal features across multiple frames. We implement slow fusion by extending spatial convolution to the temporal domain and adding representation of filter *stride* for both space and time domains. This allows us to control filter size and stride in both temporal and spatial domains leading to a generalized $3\mathbb{D}$ convolution over spatio-temporal input tensor followed by $4\mathbb{D}$ convolutions on intermediate layers. The first convolutional layer sets temporal size and stride as 3 and 2 respectively whereas the subsequent layer has both size and stride of 2 in temporal domain. Finally the third convolutional layer merges temporal information from all frames together, culminating in a lower dimensional vector of size 4096 at the innermost layer.

Since weight initialization is critical for convergence in a deep autoencoder, we use pre-training for each convolutional layer as we add the layers on. Instead of initializing all weights at once and training from there, we train the first and last layer first, followed by the next convolutional layer and so on. We stack new layers on top of previous layers and fix the weights of earlier layers for faster convergence. We discuss this in detail in section 2.4.1.

2.2.2 Semi-supervised learner

We propose a semi-supervised approach using a deep neural network, by combining an autoencoder with a classification loss function, and training both of them in parallel. The input for the first layer is a short sequence of facial gestures composed of 9 frames cropped

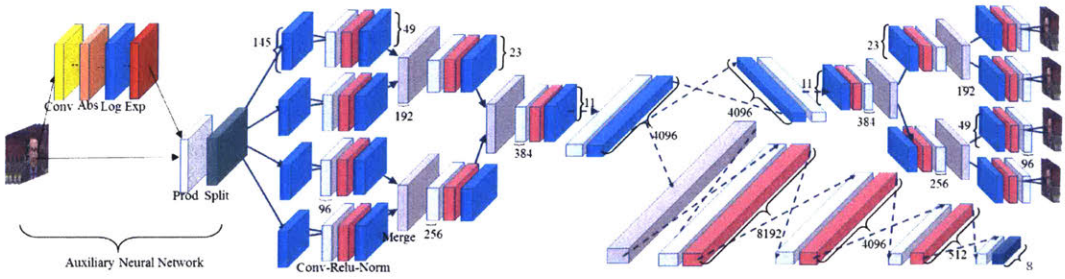


Figure 2-2: We learn 7 different facial emotions from short (about 1 sec, 25 frames) video clips. The Illumination invariant aspect is achieved by adding an illumination invariant neural network to induce illumination invariance on original input. Our prediction system is based on slow temporal fusion neural network, trained by hybridization of autoencoding a huge collected dataset and a loss prediction on a small set of labeled expressions.

to 145×145 pixels window. The loss function is evaluated by combining a predictive loss from 7 different pre-labeled gestures (for the labeled part of the dataset), and autoencoder Euclidean loss for the entire (labeled and un-labeled) collection. The weights of each layer are dynamically altered such that the importance of the autoencoder loss decreases with relation to the predictive loss as the training progresses. While generating the data, we use Viola-Jones face detection [68] for cropping the faces. We use slow fusion based convolutional neural network with convolutions in both space and time (see figure 2-3 for a detailed overview).

Our predictor neural net consists of a combination of several convolutional layers followed by multiple fully connected layers ending in a softmax logistic regression layer for prediction. Architecture can be described as $C(96, 11, 3) - N - C(256, 5, 2) - N - C(384, 3, 2) - N - FC(4096) - FC(8192) - FC(4096) - FC(1000) - FC(500) - FC(8)$ using shorthand notation described in section 2.2.1. Notice that our autoencoder architecture is overlaid on top of the predictor architecture by adding deconvolutional layers after the first fully connected layer to create a semi-supervised topology which is capable of training both autoencoder and predictor together (see Figure 2-2). We use autoencoder to initialize weights for all convolutional layers, all deconvolutional layers and central fully connected layers and we initialize any remaining layers randomly. We use stochastic gradient descent to train weights by combining losses from both predictor and autoencoder while training, this combined loss function for the semi-supervised learner is described in the equation 2.1.

$$L = -\beta \sum_j y_j log\left(\frac{e^{o_j}}{\sum_k e^{o_k}}\right) + \alpha||\bar{x} - \bar{x}_o||_2 \tag{2.1}$$

The protocol we suggest for training the net is as important as the topology itself. We begin by training the autoencoder as a sole learner from the outer layer to the inner ones. Meaning, we adaptively add layers to the autoencoder, train the neural net, and use the produced weights as initialization for the next step. This is one of the traditional approaches used to train autoencoders [10, 16]. Next, we use the weights for initialization of the semi-supervised net, allowing the entire net to fine tune. A key factor in training is the learning rate of the two matched learners. We begin the training using a higher learning rate for the autoencoder (with predictor layers staying fixed using zero learning rate) and end the process with increased importance to the labeled loss function. While training on the labeled data, ratio between the two varies from a factor of 10^3 to a factor of 10^5 favoring the loss layer.

2.2.3 Multi-velocity semi-supervised learner

One of the main challenges in action recognition is related to assigning similar classification to objects at different velocities. In this work we propose to learn the velocity of the sequence in parallel to its classification by adaptive temporal interpolation. Our multi-velocity autoencoder consists of 3 action autoencoders combined together to access temporal features for different velocities. We achieve this by adding a convolution layer as the first layer which uses cubic b-spline interpolation to *slow down* the video and generate intermediate frames. Piece-wise cubic b-spline interpolation is preferred over polynomial techniques as it can minimize interpolation error for fewer points and lower degree polynomials [69]. For initialization a sampling factor of 1, 2/3 and 1/3 is chosen, which is later refined as a part of the learning.

Next we show how to generate the required weights for interpolation and encode them as a neural network layer. Cubic b-splines are continuous *piecewise-polynomial* functions containing polynomials of degree 3 or less. A cubic b-spline spanning $N+1$ points comprises of N cubic polynomials $\left(\mathbf{S_k}(x)_{k=1}^{N}\right)$ which can be uniquely defined using $4N$ coefficients. These coefficients can be recovered by applying linear constraints arising from continuity and differentiability of the function on the break points (or *knots*). We represent input video

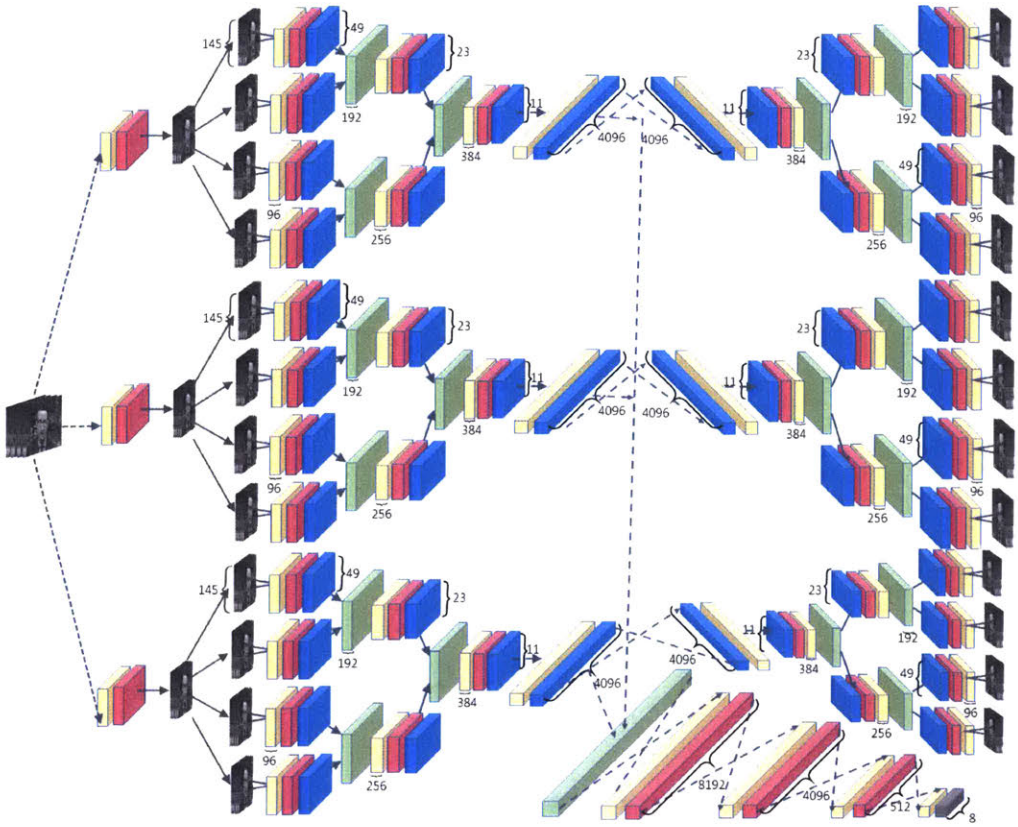


Figure 2-3: Complete architecture of multi-video semi supervised learner comprises of temporal filters for generating video frames at multiple velocities serving as input to 3 separate autoencoders. The predictor merges deep features from all 3 autoencoders and learns classification labels using deep neural net on top of these.

at each pixel as a function of time and use cubic b-splines to approximate intermediate values. We represent intermediate polynomials between $N+1$ frames as a coefficient vector $\bar{p}$ containing coefficients for all N polynomials. We use algorithm 1 to create 3 different weight matrices which interpolate sampling factors of 1, 2/3 and 1/3. Here *nSplines* refers to number of spline curves which is one less than the number of frames. If sequence was represented by 9 frames $0, 3...24$, the fastest velocity interpolant would have frames $0, 1, 2...8$ and the second fastest would have frames indexed $0, 2, 4...16$.

Finally we attach the new proposed Multi-Velocity layers as the first structure of the semi-supervised neural net. Each sub-structure (See Figure 2-3 and 2-5), has its own autoencoder, all of which are concatenated after the inner most convolution layer into a feature vector (size 12288), later used by the labeled loss function. The learner loss function can be expressed as a weighted sum of autoencoder and predictor loss given in equation 2.1.

Algorithm 1 Generate convolution layer spline weights.

Input: Frame numbers $\bar{x}$, new temporal locations $\bar{u}$
Output: Caffe Weight Matrix **W**

function SPLINEWEIGHTS(c)
  $nSplines \leftarrow length(\bar{x}) - 1$
  for $(i \leftarrow 0; i < nSplines; i++)$ **do**
    $p \leftarrow 4i$
    $\mathbf{T_{p,i}} \leftarrow 1$
    $\mathbf{T_{p+1,i+1}} \leftarrow 1$
    for $(h \leftarrow 0; h <= 1; h++)$ **do**
      $s \leftarrow p - 4h$
      $\mathbf{A_{p+h,p:p+3}} \leftarrow [h^3, h^2, h, 1]$
      $\mathbf{A_{i+2,s+4:s+8}} \leftarrow -1^{h+1}[3h^2, 2h, 1, 0]$
      $\mathbf{A_{i+3,s+4:s+8}} \leftarrow -1^{h+1}[6h, 2, 0, 0]$
  $\mathbf{A_{i+3,i-4:i+3}} \leftarrow [6, 0, 0, 0, -6, 0, 0, 0]$
  $\mathbf{A_{i+4,0:7}} \leftarrow [6, 0, 0, 0, -6, 0, 0, 0]$
  for $(i \leftarrow 0; i < length(\bar{u}); i++)$ **do**
    $p \leftarrow find(\bar{x}, \lfloor \bar{u}(i) \rfloor)$
    for $(h \leftarrow 0; h < 4; h++)$ **do**
      $\mathbf{R_{i,4p+h}} \leftarrow (\bar{u}(i) - \bar{x}(p))^h$
  $\mathbf{W} \leftarrow \mathbf{RA^{-1}T}$
return W

2.2.4 Illumination invariant learner

We introduce scale invariance[2] to pixel intensities by adding additional layers as an illumination invariant neural network in the beginning of semi-supervised learner. The illumination invariant layers include a convolutional layer, an absolute value layer, a reciprocal layer followed by a Hadamard product layer. Scale invariance is achieved by applying element wise multiplication between the output layers of proposed architecture and the original input layer. This normalization can be written as $C(9,1,9) - Abs - Log(\alpha,\beta) - Exp(-\gamma,\delta) - Prod(x_1,x_2)$ (please refer to shorthand notation in section 2.2.1). Here $C(9,1,9)$ refers to the first convolutional layer containing 9 filters with size 1×1 in spatial domain and a size of 9 in time domain. *Abs* is a fixed layer to compute absolute value, $Log(\alpha,\beta)$ layer computes the function $\bar{ln}(\alpha * \bar{x} + \beta)$ and $Exp(\gamma,\delta)$ layer gives us $e^{\gamma * \bar{x} + \delta}$. In the end $Prod(x_1,x_2)$ layer takes two inputs (x_1,x_2) and multiplies the output of exponential layer (x_2) with the original input tensor (x_1). If $\bar{F}(\bar{x})$ denotes function emulated by first convolution layer, we can write the transfer function of this sub-net as follows (equation 2.2).

[2]An abridged version of this work appeared as "Illumination invariants in deep video expression recognition", Otkrist Gupta, Dan Raviv, Ramesh Raskar, The Journal of Pattern Recognition Society (2017)

Algorithm 2 Generate scale invariant convolution layers

Input: Frame numbers $\bar{x}$, new temporal locations $\bar{u}$
Output: Caffe Weight Matrix **W**

```
function AUTOENCODERWEIGHTS(nFrames, wSize)
    r ← (wSize − 1)/2
    A ← zeros(nFrames, nFrames)
    for (i ← 0; i < nFrames; i++) do
        n ← min(i, r)
        n ← min(n, nFrames − i)
        A_{i,i−n:i+n} ← 1/(2n + 1)
    W ← A
return W
```

$$\bar{H}(\bar{x}) = \bar{x}e^{-\gamma log(\alpha|\bar{F}(\bar{x})|+\beta)+\delta} = \frac{e^{\delta}\bar{x}}{(\alpha|\bar{F}(\bar{x})| + \beta)^{\gamma}} \tag{2.2}$$

Log and *Exp* layers are used to generate a reciprocal layer by setting meta-parameters γ to 1 and δ to zero. We can also *"switch off"* this sub-net by setting both of these parameters to zero. Transfer function meta parameters α (*scale*) and β (*shift*) can be tuned as well for optimal performance. We perform a grid search to find optimal values for these after re-characterizing the transfer function parameters as a global multiplicative factor τ and a proportion factor η (see equation 2.3). Table 2.1 shows results for various choices of α and β. We can reformulate equation 2.2 as given below:

$$H(\bar{x}) = \frac{e^{0}\bar{x}}{(\alpha|\bar{F}(\bar{x})| + \beta)^{1}} = \frac{\frac{1}{\beta}\bar{x}}{1 + \frac{\alpha}{\beta}|\bar{F}(\bar{x})|} = \frac{\tau\bar{x}}{1 + \eta|\bar{F}(\bar{x})|} \tag{2.3}$$

The output from scale invariant neural net is a $145 \times 145 \times 9$ tensor which is used as input in the autoencoder and predictor neural networks. The convolution layer can be parametrized using a $9 \times 1 \times 1 \times 9$ tensor and changes during fine tuning while α and β are fixed constants greater than zero. In our experiments we initialized convolutional filter of scale invariant sub-net using several approaches, such as partial derivatives, Mellin transform, moving average and laplacian kernel and found that it performed best when using neighborhood averaging. Algorithm 2 demonstrates initialization of convolutional layer at the beginning of illumination invariant neural net.

$\eta \rightarrow$ / $\tau \downarrow$	0.1			1			10		
0.2	scale (η/τ)	0.5	0.486	scale (η/τ)	5	0.472	scale (η/τ)	50	0.243*
	shift $(1/\tau)$	5		shift $(1/\tau)$	5		shift $(1/\tau)$	5	
0.5	scale (η/τ)	0.2	0.50	scale (η/τ)	2	0.5135	scale (η/τ)	20	0.45*
	shift $(1/\tau)$	2		shift $(1/\tau)$	2		shift $(1/\tau)$	2	
1	scale (η/τ)	0.1	0.499	scale (η/τ)	1	0.51	scale (η/τ)	10	0.47
	shift $(1/\tau)$	1		shift $(1/\tau)$	1		shift $(1/\tau)$	1	
5	scale (η/τ)	0.02	-*	scale (η/τ)	0.2	0.44	scale (η/τ)	2	0.50
	shift $(1/\tau)$	0.2		shift $(1/\tau)$	0.2		shift $(1/\tau)$	0.2	

Table 2.1: Accuracies for various values of scale(α) and shift(β) for illumination invariant neural net. We do a grid search for τ varying from 0.5 to 5 and η varying from 0.1 to 10. Yello columns show corresponding values scale and shift and blue columns show test accuracies. Cells marked with asterisk(*) indicate configurations that did not converge during training.

2.3 Datasets

In order to evaluate the proposed architecture we use two known datasets from literature as well as present two additional datasets collected by us; The first dataset contains more than 160 million images combined into 6.5 million short (25 frames) clips, used by us to train our autoencoders. The second dataset is comprised of 2777 short clips labeled for seven emotions. In the following section we elaborate on the four datasets.

2.3.1 Autoencoder dataset

In order to train very deep neural nets we must obtain a huge collection of data. Here we collected 6.5 million video clips containing 25 frames each, adding up to more than 162 million face images. We use public sources such as local broadcast television networks (FOX, CSPAN, NBC etc.) and YouTube for data collection. We used Viola-Jones face detector to find and segment out the faces. Only clips which contained more than 25 overlapping facial frames were selected. Next, we localized landmarks for each frame using a deformable model for the face [70] and detected the facial pose by fitting a 3D model to the landmarks. This process allowed us to restrict the dataset to videos which contain faces tilted less than 30 degrees and remove any faces looking sideways.

In order to extract only meaningful video clips we removed clips with static gestures or those where the faces were rapidly altering, either due to some high speed movement or simply due to appearance of a different face. We achieved this by blurring the clips and

calculating the difference between consecutive frames. To our knowledge this is the biggest facial dataset reported in literature, and we plan to make it public.

2.3.2 Asevo dataset

The database contains facial clips from 160 subjects both male and female, where gestures were artificially generated according to a specific request, or genuinely given due to a shown stimulus. We collected a total of 2777 clips out of which 1745 were captured after providing the stimulus while 1032 were generated artificially. To create natural facial expressions we selected a bank of YouTube videos for each facial expression and showed them to subjects, capturing their reaction to the visual stimulus. Clips had a uniform distribution accross all seven facial expressions, and contained both posed and non-posed expressions.

2.3.3 Cohn Kanade Dataset

The Cohn Kanade dataset [71] is one of the oldest and well known dataset containing facial expression video clips. It contains a total of 593 video clip sequences from which 327 clips are labeled for seven basic emotions (most of these are *posed*). Clips contain the frontal view of face performing facial expression varying from neutral expression to maximum intensity of emotion. While the dataset contains a lot of natural smile expressions it lacks diversity of induced samples for other facial expressions. Along with posed facial expressions, the dataset also contains non-posed smile expressions. However the dataset lacks depth in having other non-posed expressions and is not extensive as Asevo dataset in capturing naturally expressed emotions. Each video clip contains facial expression going from baseline neutral to peak of expressed emotion.

2.3.4 Man Machine Interaction Dataset

Man Machine Interaction (MMI) facial expression dataset [72] involves an ongoing effort for representing both enacted and induced facial expressions. The dataset comprises of 2894 video samples out of which around 200 video clips are labeled for six basic emotions. The clips contain faces going from blank expression to the peak emotion and then back to neutral facial expression. MMI which originally contained only posed facial expressions, was recently extended to include natural versions of happiness, disgust and surprise [73].

Figure 2-4: Results from reconstruction using temporal convolutional autoencoder on a face video. (a) Input video sequence. (b) Reconstruction after using 4 convolutional layers. (c) Reconstruction after using 8 layers. (d) Reconstruction after using 12 layers.

2.4 Experiments and Results

2.4.1 Deep autoencoder

Since deep autoencoders can show slow convergence when trained from randomly initialized weights [67], we used contrastive divergence minimization to train stacked autoencoder layers *iteratively* [16]. Initially, we pre-trained the beginning and end convolutional layers by creating an intermediate neural network ($C(96,11,3)-N-C(256,5,2)-N-DC(256,5,2)-N-DC(384,3,2)$) and training it on facial video clips. Inner layers were trained successively by adding them to the intermediate neural network and keeping pre-trained layers fixed until the convergence of weights. To yield best results, we also fine tuned the entire network at the end of each iteration. This process was repeated until the required number of layers had been added and final architecture was achieved. Training of the entire autoencoder typically required 3 days and a million data inputs.

Our neural network was implemented using the *Caffe* framework [74] and trained using NVIDIA Tesla K40 GPUs. The trained weights used to initialize next phase were stored as Caffe *model* files and each intermediate neural network was implemented as a separate *prototxt* file. Weights were shared using shared parameter feature and transferred across neural networks using the resume functionality provided in *Caffe*. Our deep autoencoder took $145 \times 145 \times 9$ clips as input, the spatial resolution was achieved by down-sampling all clips to a fixed size using bi-cubic interpolation. 9 frames were obtained by extracting

Figure 2-5: Results from reconstruction using multi velocity encoders, bottom 3 images are output from autoencoder ensemble. (a) Input video sequence from *MMI dataset* ([72]). (b) Reconstruction using encoder with sampling factor of 1/3. (c) Reconstruction using sampling factor of 2/3. (d) Reconstruction at original velocity.

every third frame from video clips. All videos were converted into 1305×145 image clips containing consecutive input frames placed horizontally and we used the *Caffe "imagedata", "split"* and *"concat"* layers to isolate individual frames for autoencoder input and output. We trained with a batch size of 22 images using a base learning rate of 10^{-6}. The learning rate was reduced by a factor of 10 every 10000 iterations. We used a momentum of 0.9 and used stochastic gradient descent based optimization when training the network. The network requires 4-5 hours to train on a single NVIDIA Tesla K40 GPU.

Please see Figure 2-4 to visualize results obtained from intermediate autoencoders using different number of layers.

2.4.2 Multi-velocity video autoencoder

Multi velocity semi-supervised learner comprises of an array of three independent autoencoders and a predictor net. We initialize the autoencoders using the weights from the video autoencoder and add a convolution layer as described in section 2.2.3. We fine tune the multi-velocity layers by creating 3 datasets containing video clips at different velocities. We achieve that by selecting every third frame to create set 1 *(speed = 3x)*, selecting every second frame to generate set 2 *(speed = 2x)* and taking first 9 frames for set 3 *(speed = 1x)*. The weights from this step are used for initialization of our multi-velocity predictor which described next.

2.4.3 Multi-velocity predictor

For training, testing and validation we divide each dataset into 3 parts randomly. We select 50% inputs for training, 30% of dataset for testing and use 20% of dataset for validation. After the dataset was split, we further increased the size of the training dataset by shifting each video along both axes, rotating images and taking their mirror.

We train our proposed semi-supervised learner and the multi-velocity semi-supervised learner on the three datasets (MMI, CK and Asevo), and compare our results against multiple kernel methods [62] and expression-lets base approaches [61]. We used sources downloaded from *Visual Information Processing and Learning Resources* [75] as a reference to compare to our methods. Note that we made the same data partitioning scheme (train, validation, test) for all methods to show a fair comparison.

We outperform all the methods compared on all the datasets used, by a substantial gap, in almost all cases. We summarize our findings in Table 2.4, and show confusion matrices per facial expression in Table 2.5. For baseline comparison against other deep neural architectures, we compare our methods against [9] and GoogleNet [76]. We further verified our results against prior state of the art methods discussed in [50] by performing **10 fold cross validation**. On MMI we get 66.15 (vs 63.4) % and on CK+ we get 94.18 (vs 92.4) %. Because of extreme parallelism in neural networks, we are able to run our system in real time at 20.7 fps.

2.4.4 Illumination-invariant semi-supervised predictor

Another approach to induce scale invariance can involve using standardized Local Response Normalization (LRN) based layers in the neural network right after the first input layer. This approach is similar to pre-normalizing the data before testing. We compare our method to this approach as well and found that adaptive normalization performed better than plain LRN based learner. Our results are summarized in Table 2.2.

Our scale-invariant neural network prefixes semi-supervised learner with an axillary neural net to induce scale invariance (see 2.2.4). We test our method on three datasets (MMI, CK and Asevo) by randomly dividing each of them into non-intersecting train, test and validation subsets. Our training dataset contains 50% inputs while testing and validation datasets contain 30% and 20% of inputs. After the split we increase the size of

	MMI	*CKPlus*	*Asevo*
Semi-Supervised Learner	55.7	68.42	38.66
LRN @(0.5)	54.09	69.47	35
LRN @(0.75)	55.73	69.47	35.84
Scale Invariant Learner	59.03	73.68	48

Table 2.2: Comparison of illumination invariant learner to plain semi supervised learner with Local Response Normalization layers in the beginning. We try changing coeffecients of Local Response Normalization and got good results when setting β at 0.75. Our method continued to win for both small and large datasets (winning method is shown in blue and the leading method is showed using yellow).

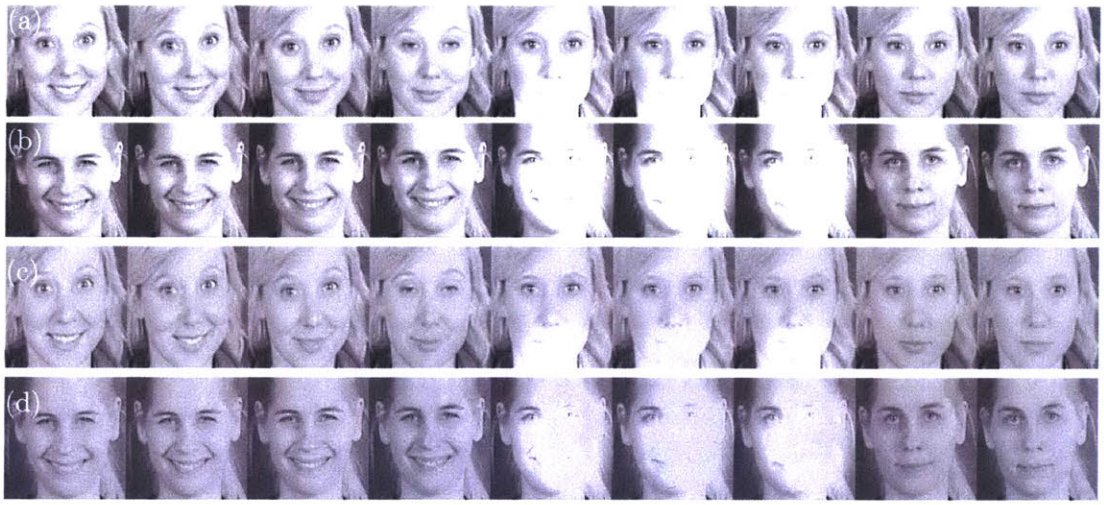


Figure 2-6: Results from autoencoder reconstruction while using scale invariant autoencoder. (a), (b) Input video sequence. (c), (d) Output video sequence from illumination invariant neural network.

training dataset by adding rotation, translation or flipping the image.

For quantitative analysis we compare our results against expression-lets base approaches [61] and multiple kernel methods [62]. We utilize sources downloaded from *Visual data transforming and taking in Resources* [75] as a reference to contrast with our strategies. For reasonable comparison we use same partitioning techniques while comparing our techniques with external methods. While we cannot compare against methods such as [50] because of absence of publicly available code our method still wins on MMI dataset.

We test our method with and without varying illumination on external datasets, results of our findings can be summarized in Table 2.3. Please see tables 2.5 for confusion matrices demonstrating results for each expression. We outperform all external methods on datasets in almost all cases. Our method also shows large margin of improvement over

	MMI	CKPlus	Asevo
Main Gaussian Riemann [62]	40.9	67	46.92
Main Grassman [62]	9.09	17.9	44.99
Main Covariance Reimann [62]	40.9	79	51.05
Expressionlets [61]	52.91	82.7	48.6
Interval Temporal Bayesian Network [52]	59.7	86.3	-
Hidden Markov Models [52]	51.5	83.5	-
Our topology without pre-training	20.96	26.31	24.43
Our method without semi-supervised learning	51.61	83.17	43.36
Vanilla Semi-Supervised Learner	59.01	87.36	51.11
Scale Invariant Learner	65.57	90.52	51.35

Table 2.3: Comparison of results using Illumination Invariant Techniques for datasets under standard conditions. Results from both scale invariant and simple(vanilla) architecture are compared.

	MMI	CKPlus	Asevo
Main Gaussian Riemann [62]	39.39	65	43.9
Main Grassman [62]	9.09	11	43.91
Main Covariance Reimann [62]	36.36	65	46.44
Expressionlets [62]	55.38	70	46.02
Vanilla Semi-Supervised Learner	55.7	68.42	38.66
Scale Invariant Learner	59.03	73.68	48

Table 2.4: Comparison of results from various techniques on CKPlus, MMI and Asevo datasets. The dataset was divided into 3 parts test, train and val randomly. Training set was 50%, test and validation were 30% and 20% respectively. We compare the performance with and without scale invariant architecture. The table on the top shows results on original data while the one on the bottom shows results after we added illumination changes. Our method consistently won for both small and large datasets (winning method is shown in blue and the leading method is showed using yellow).

plain semi-supervised approaches. Both autoencoder and predictor network topologies are implemented as *Caffe prototxt* files [74] and they will be made available for public usage. As we demonstrate in Table 2.4, pretraining with large amount of unlabeled data helps greatly by providing very good initialization of weights. Without pretraining the network overfits by a huge extent converging to values around 20-26% which are only slightly above random accuracy. We demonstrate an accuracy gain of more than 30% on MMI and more than 60% on CKPlus when using large unlabeled dataset to pretrain the network.

Confusion matrix using our methods on Asevo Dataset

	Anger	*Contempt*	*Happy*	*Disgust*	*Fear*	*Sadness*	*Surprise*
Anger	0.54	0.13	0.01	0.12	0.03	0.12	0.04
Contempt	0.07	0.47	0.14	0.06	0.04	0.14	0.07
Happy	0.01	0.15	0.78	0.04	0.00	0.00	0.00
Disgust	0.13	0.21	0.15	0.31	0.07	0.05	0.08
Fear	0.04	0.14	0.08	0.08	0.28	0.04	0.32
Sadness	0.18	0.22	0.07	0.10	0.02	0.27	0.13
Surprise	0.04	0.07	0.07	0.05	0.21	0.05	0.51

Confusion matrix using multiple kernel methods [62] on Asevo dataset

	Anger	*Contempt*	*Happy*	*Disgust*	*Fear*	*Sadness*	*Surprise*
Anger	0.61	0.08	0.06	0.07	0.04	0.06	0.08
Contempt	0.09	0.44	0.27	0.06	0.02	0.06	0.05
Happy	0.01	0.07	0.85	0.01	0	0.01	0.06
Disgust	0.19	0.14	0.25	0.22	0.01	0.02	0.16
Fear	0.21	0.09	0.06	0.07	0.12	0.06	0.39
Sadness	0.26	0.20	0.19	0.02	0.05	0.13	0.16
Surprise	0.15	0.03	0.09	0.01	0.06	0.05	0.61

Confusion matrix using our methods on Cohn-Kanade

	Anger	*Contempt*	*Happy*	*Disgust*	*Fear*	*Sadness*	*Surprise*
Anger	0.92	0.08	0	0	0	0	0
Contempt	0	0.80	0	0	0	0	0.20
Happy	0	0	1.00	0	0	0	0
Disgust	0.06	0	0	0.94	0	0	0
Fear	0	0	0.14	0	0.71	0	0.14
Sadness	0.38	0	0	0	0	0.50	0.12
Surprise	0	0	0	0	0	0	1.00

Confusion matrix using expressionlets [61] on Cohn-Kanade

	Anger	*Contempt*	*Happy*	*Disgust*	*Fear*	*Sadness*	*Surprise*
Anger	0.73	0	0.07	0	0	0.20	0
Contempt	0	0.86	0	0	0	0.14	0
Happy	0	0	0.95	0	0.05	0	0
Disgust	0.25	0.12	0	0.38	0	0.12	0.12
Fear	0	0	0	0	1.00	0	0
Sadness	0.33	0	0	0.11	0	0.44	0.11
Surprise	0	0	0	0.05	0	0	0.95

Table 2.5: Confusion matrices over test results for Cohn Kanade and Asevo datasets using our methods and best performing external method which uses *Expressionlets* for CKPlus [61] and *covariance Riemann kernel* for Asevo [62]. On the left we show results for the proposed illumination invariant semi-supervised approach across various facial expressions, while on the right we present confusion matrix from external methods. Highest accuracy in each category is marked using blue color. For CKPlus we outperform competing method in 5 verticals by getting 100% accuracy on happiness, 100% on surprise, 94% on disgust, 92% in anger and 50% in sadness. For both methods misclassification occur when emotions like sadness get recognized as anger and *vice-versa*.

2.4.5 Learning calibration invariant sensing

The video convolutional layers discussed above can be used to learn other invariants in CNNs as well. In this section we describe how we can use a data driven approach and leverage convolutional neural networks (CNN) to learn a model that is invariant to calibration parameters. The CNN is trained with a large synthetic dataset generated with a Markov Chain Monte Carlo (MCMC) model that contains random realizations of major calibration parameters. The synthetic random dataset generated with the MCMC forward model is used to train a CNN for classification of hidden objects behind a diffuser. The topology is modified by extension of convolution filters into time domain (3D space-time filters) using spacial convolutional filters we described previously. Filters were resized to $3 \times 3 \times 10$ where the last index denotes the time dimension (see further details in the Discussion). The training time on $60,000$ data points is approximately two hours on an NVIDIA Titan XP GPU.

To evaluate our approach, we used the well-known MNIST dataset of handwritten digits. The targets are placed behind a regular paper sheet (diffuser) and measured with the SPAD camera. The goal is to evaluate the CNN ability to classify hidden objects while being invariant to changes in calibration parameters. To that end, $60,000$ training samples and $10,000$ test samples are synthesized with the MCMC forward model. Each data point is a realization of a different set of target and calibration parameters. The result is an overall classification accuracy of 74% (compared to 10% random guess accuracy). These simulations demonstrate the ability to classify objects hidden behind a scattering layer without calibration. As a proof of concept experiment, we cut the zero and one digits from cardboard and placed them behind a paper sheet. The network described above correctly classified the digits.

Additional test sets were generated to evaluate the extent of the network invariance to changes in calibration parameters in a controlled experiment. In each test set, all calibration parameters are held fixed (on the mean), except for one parameter that is randomly sampled from distributions with different variances. Thus, the CNN's sensitivity to variations in different parameters is probed independently. Specifically, for each calibration parameter to be investigated, multiple test sets are generated, each one with a different distribution variance. Fig. 2-7 demonstrates results for three parameters (other parameters demonstrate

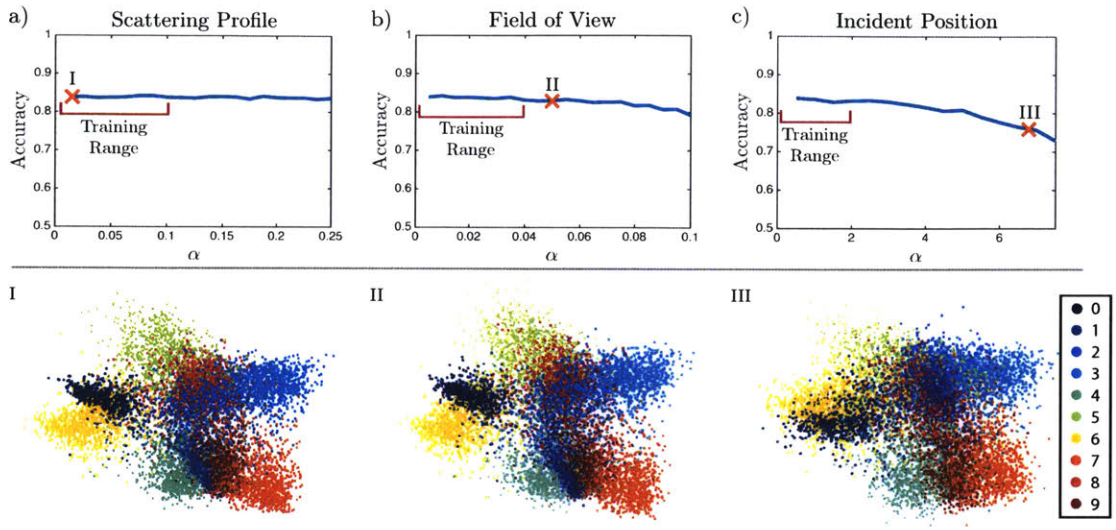


Figure 2-7: CNN learns to be invariant to model parameters. The CNN is trained with the complete random training set (based on the MNIST dataset), and evaluated with test sets in which all model parameters are fixed except for one that is randomly sampled from distributions with growing variance. Three parameters are demonstrated (other parameters show similar behavior). a) Diffuser scattering profile variance $D_D \sim N(0, \sigma)$, $\sigma \sim U(1-\alpha, 1+\alpha)$ radians, b) Camera field of view $C_{FV} \sim U(0.15-\alpha, 0.15+\alpha)$ radians, and c) Illumination source position $L_P \sim U(-\alpha, \alpha)$ cm. The top plots shows the classification accuracy as a function of the parameter distribution variance in the test set. Red lines show the ranges used for training. The 'X' marks point to specific locations sampled for PCA projections in the bottom part of the figure. PCA projections show a color map where each digit has different color. Performance is maintained beyond the training range and starts to slowly degrade further from it, as can be observed in PCA projection III where more mixing is apparent.

similar behavior). As can be seen from the test accuracies, performance is maintained within the variance range used for training and extended beyond that range. This demonstrates the network ability to learn an invariant model to changes in the calibration parameters. For example, in Fig. 2-7c the network was trained with data that had the illumination position distributed uniformly within $5cm$ from the mean. Yet, the test performance starts to slightly drop only after the illumination position may be found within $10cm$ of the mean. Qualitative evaluation of these results are also presented in Fig. 2-7 with PCA projections of the activations from the penultimate layer of the CNN, these demonstrate sustained performance beyond the training range.

2.5 Concluding Remarks

This chapter uses semi-supervised paradigms in convolutional neural nets for classification of facial expressions in video sequences. Our topologies are trained on millions of facial video clips and use spatio-temporal convolutions to extract transient features in videos. We developed a new scale-invariant sub-net which showed superior results for expression recognition under variable lighting conditions. We demonstrate effectiveness of our approach on both publicly available datasets and samples collected by us.

In this chapter we introduce a framework for facial expression recognition which combines semi-supervised learning approaches with carefully designed neural network topologies. We demonstrate how to induce illumination invariance by including specialized layers and use spatio-temporal convolutions to extract features from multiple image frames. Currently, our system relies on utilization of Viola-Jones to distinguish and segment out the faces and is limited to analyzing only the front facing views. Emotion recognition in the wild still remains an elusive problem with low reported accuracies which we hope will be addresses in future work.

In this work we only considered video frames but other, richer, modalities could be taken into account. Sound, for example, has a direct influence on the emotional status and can improve our current system. Higher refresh rates, multi-resolution in space and time, or interactions between our subjects are just few of many possibilities which can to enrich our data and can lead to better classification or inference. Methods such as RNNs could be used to work with variable number of temporal frames. Future methods could benefit from addition of more frames at higher FPS, leading to better classification accuracy due to higher resolution temporal information.

Deep neural networks have proven to be extremely effective in solving computer vision problems even though training them at large scale continues to be both CPU and memory intensive. Our system tries to make best use of resources available and further improvements in hardware and software can help us build even larger and deeper neural networks while enabling us to train and test them on portable devices. Computational complexity of neural networks continues to be a major hurdle towards their ubiquitous application in such areas. Further research using methods such as binarized and compressed neural networks [77, 78] can help alleviate such issues in future. Over here, we introduce layers which learn invariance

adaptively and can be fine tuned to get best results. In this work, we emphasize on scale invariance for illumination, velocity and camera parameters, in future we hope to explore induction of other invariants, which continues to be an area of rapid research in neural networks.

Chapter 3

Optimizing neural network topologies

As the previous chapter demonstrates, convolutional neural networks are a very powerful tool for real world imaging and inference. Deep convolutional architectures comprise of hierarchical organization of several convolutional, normalization, pooling and fully connected layers, while combining simple linear operations with non-linearities like sigmoid and rectified linear units. Engineering such layers continues to be a challenging task driven primarily by expert experience and intuition while requiring large amount of resources. This difficulty in developing new topologies has led to creation and popularization of successful topologies, namely VGG[79], AlexNet[9], GoogleNet[8] and ResNet[80]. Deep learning topologies such as AlexNet and VGG have been commoditized to a large extent, with most research focusing on direct application of such topologies to a new real world problem domain.

In this chapter we explore the space of convolutional neural network architectures using context free grammars and develop methods to automatically select architectures that perform well on a given dataset. We begin by describing the topology space using a context free grammar and a succinct vocabulary. We use our learnings from layer engineering to restrict the generation space to a finite number of possibilities. We demonstrate how our context free grammar formulation leads to a conception of a finite state automata describable using a directed acyclic graph. Each traversal of this graph generates a possible topology thereby sampling the space of topologies. We can reformulate the problem of selecting good

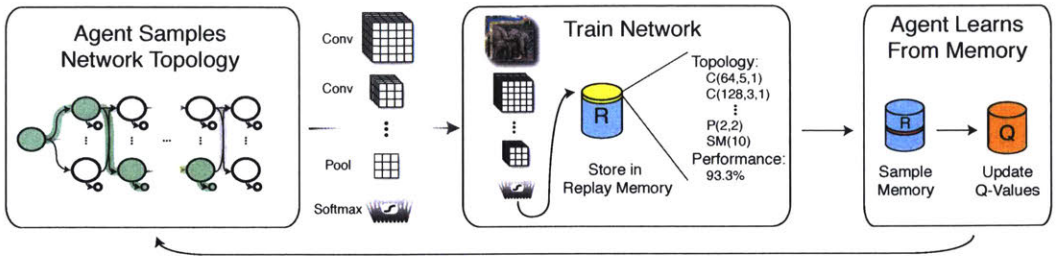


Figure 3-1: Our system involves exploration of neural network topology space by sampling and training of topologies. The validation accuracies obtained from training is used to derive statistics behind layer selection and neural network accuracies. This information about expected *reward* is used to influence the layer distribution when selecting next batch of topologies. The agent learns by *exploring* the state of topologies and then *exploits* the information it acquired to sample well performing architectures.

topologies as finding the most *rewarding* path when traversing the graph. There can be several ways to optimize such graphs, in our formulation we optimize it using reinforcement learning. In this formulation we will pose layers as states, layer connections as actions and validation accuracy as reward function. We can then use model free techniques such as Q learning to learn reward distribution while selecting best performing architectures.

We traverse the space of model architectures using standard convolutional, pooling and fully connected layers and demonstrate competitive results on the datasets including MNIST, CIFAR-10 and SVHN. We discover that for the same depth, learning agent is able to beat the existing human generated topologies. We are successful in generating architectures that generalize well, and demonstrate that the topologies learned on one dataset can be transferred robustly to other datasets. We also find that less deep *shallower* architectures can sometimes even outperform much more deeper architectures. Our methods also perform better than previously automated hyperparameter selection algorithms (e.g., [81, 82]), producing topologies which are significantly better than the previous automated topology selection techniques [83, 84]. Surprisingly, we discover that the topology space is in fact richly filled with topologies that perform well.

3.1 Related work

The impressive performance of CNN architectures has been driven by manual tuning of network designs and hyper-parameters, along with new design ideas or training procedures

introduced on top of successful architectures. Research on automating neural network design goes back to the 1980s when genetic algorithm-based approaches were proposed to find both architectures and weights[85]. Interesting methods include evolutionary algorithms[83] (NEAT), using networks of networks for searching optimal configurations[86], Bayesian optimization[87] or utilizing tree of parzen estimators to design feed-forward networks [84]. Other biologically inspired ideas have also been explored; motivated by screening methods in genetics,[88] proposed a high-throughput network selection approach where they randomly sample thousands of architectures and choose promising ones for further training.

In the context of neural networks, *hyperparameter* optimization refers to an algorithmic approach for finding optimal values of design-independent constants such as learning rate, weight decay, step size and batch size, along with a limited search through the network design space, usually through the space of filter types and sizes. While random or grid search is still the most commonly used method[89], a variety of Bayesian optimization methods have been proposed for hyperparameter optimization, including methods based on sequential model-based optimization (**SMAC**)[90], Gaussian processes (**GP**)[91], and tree of parzen estimators (**TPE**)[84]. To improve on the scalability of Bayesian methods, [92] utilize neural networks to efficiently model distributions over functions. Recently, [93] introduced ***Hyperband***, a multi-armed bandit-based efficient random search technique that outperforms state-of-the-art Bayesian optimization methods.

3.2 Generating architectures using context free grammars

In this section we will describe the context free grammar based sampling of the space of neural network architectures. Our aim is to find a suitable parametrization for neural network topologies [88]. The space of neural network topologies consists of both acyclic and cyclic directed graphs, which can be infinitely large, making the problem of sampling it quite challenging. We simplify this problem by making some fundamental assumptions about the networks being sampled. We assume that the networks comprise of hierarchically organized neural network *layers* arranged as a directed acyclic graph. We also assume that these neural network layers layers can be sampled from a small set of finitely many possibilities. We remove any topologies that may have skip connections and restrict to only one outgoing edge per vertex. While our method can work with higher number of

layers, in this chapter we assume that sampled networks have a finite maximum depth less than or equal to 20 layers. Our assumptions are based on the topological configurations of modern state-of-the-art architectures such as AlexNet [9], VGG [79] and the neural network topologies described in section 2.2.3.

We will describe topologies using a shorthand notation consisting of characters and numbers to define the layer specific hyperparameters. For example we use $C(n, f, s)$ to denote a *convolutional* layer containing n filters of size $f \times f$ in spatial domain with a stride of s pixels. N denotes local response *normalization* layers, $P(f, s)$ is used to describe *pooling* layers with filters size $f \times f$ and stride s. Similarly $FC(n)$ is used to describe *fully connected* layers containing n neurons. Finally $L(n)$ denotes the loss layer composed of n neurons (which can be a *softmax* logistic regression layer for classification problems). Note that this shorthand representation is very similar to one used to describe the layers in section 2.2.1. The entire topology can be written as a succinct string by representing each layer using its shorthand notation laid out in a sequence. For example, the topology AlexNet[9] can be written as the sequence of following layers – C(96, 11, 3)-N-P(2,2)-C(256, 5, 1)-N-P(2,2)-C(384, 3, 1)-C(384, 3, 1)- C(256, 3, 1) -P(2,2)- FC(4096)- FC(4096) - L(1000).

We can now define the context free grammar for topology generation ($G = (V, \Sigma, R, s)$). The definition of a context free grammar G will include the set of non terminals V, set of terminals Σ, production rules R and starting symbol s. Set of terminals (Σ) comprises of the layer symbols $\{C, FC, P, L, N\}$, the natural number alphabet $\{0, 1...9\}$ and some syntactic sugar to simplify parsing $\{(, -,)\}$. Set of non terminals V will include the symbol to generate layers (s) and a non terminal for generating natural numbers (z). The production rules R can be divided into layer generation rules and rules to generate layer specific hyperparameters.

```
s -> C(z,z,z)-s | FC(z)-s | N-s | P(z,z)-s | L(z)
z -> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 ...
       | 0z | 1z | 2z | 3z | 4z | 5z | 6z | 7z | 8z | 9z
```

While the context free grammar formulation greatly simplifies topology generation, it is still capable of generating infinitely many topologies. We can further restrict the filter sizes and strides to fixed values and focus on topologies lesser than a maximum depth of 20. We

can achieve this by including layer depth specific rules when generating topologies. Section 3.5.2 describes how to restrict the search space and make it finite.

3.3 Application of Reinforcement Learning

We can pose the problem of selecting neural network layers as individual actions taken by an artificially intelligent agent. We use reinforcement learning based methods to optimize the directed acyclic graph generated from our context free grammar. The final objective of this reinforcement learning can be calculated using the validation accuracy of the generated topology (obtained after training). We impose restrictions on the layers by fixing number of filters, stride, size and depth, restricting our state space to a finite set of actions.

Over here we give an overview of the theoretical foundations behind reinforcement learning. Reinforcement learning is an area of machine learning which focuses on optimizing the actions of an artificially intelligent agent while maximizing a reward function. The actions of such intelligent agents are usually modeled using Markov Decision Processes. Our reinforcement learning methods will include application of approximate dynamic programming techniques when characterizing and optimizing the actions taken by intelligent agents.

A basic Markov Decision Process characterization comprises of a set of agent states (S) and a set of actions (A). It also includes probabilistic rules governing transitions from one state to another ($P_a(s, s')$) and expected reward obtained from transitioning between states ($R_a(s, s')$). Combination of agents action selections is defined as the policy function π.

$$\pi_\theta : S \times A \rightarrow [0, 1] \tag{3.1}$$

$$\pi_\theta(a|s) \leftarrow P(a_t = a|s_t = s) \tag{3.2}$$

Notice that this is analogous to context free grammar characterization when we map states to layers (or *terminals*) and production rules to state transition probabilities (with each transition probability set to 0 or 1). Every sequence of actions that an agent takes leads to generation of a new neural network topology. The total reward (R) can then be determined by training and evaluating a topology on a given dataset. In Markov Decision Formulation the reward is obtained by discounted summation of individual rewards (taken

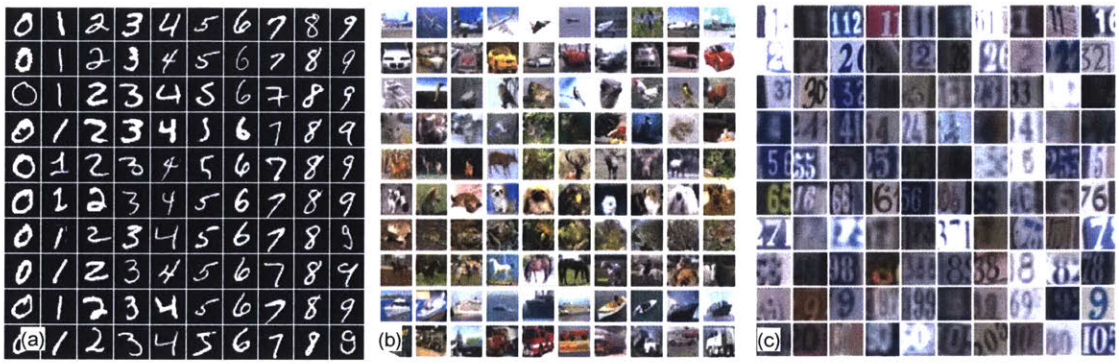


Figure 3-2: We demonstrate results on three separate datasets: (a) MNIST dataset composed of 28x28 handwritten digits, (b) CIFAR-10 contains 32x32 RGB images of real world objects and (c) SVHN contains 32x32 images of digits taken from Google Street View images.

from each state transition):

$$R = \sum_{t=0}^{\infty} \gamma^t r_t \tag{3.3}$$

Various methods in literature demonstrate how to solve the problem of finding optimal policy [94]. We further simplify the notion of policy to represent finite number of actions before generating the *terminal* loss layer and solve the problem to find more optimal architectures using reinforcement learning. Next, we will describe the datasets used in our experiments.

3.4 Datasets

In this section we describe various image classification datasets used in the experimental evaluation. The datasets have varying levels of difficulty and include tasks ranging from hand written digit recognition to object classification. The dataset sizes vary from 50000 to 700000 images. The color channels and image sizes are varied as well, demonstrating the empirical strengths in our method.

3.4.1 Mixed NIST

Mixed NIST (MNIST) database [95] contains handwritten digits sampled from postal codes and is a subset of a much larger dataset available from the National Institute Science and Technology. MNIST comprises of a total of 70,000 samples divided into 60,000 training samples and 10,000 testing samples. Original binary images were reformatted and spatially normalized to fit in a 20×20 bounding box. Anti-aliasing techniques were used to convert black and white (bilevel) images to grey scale images. Finally the digits were placed in a 28×28 grid, by computing the center of mass of the pixels and shifting and superimposing images in the center of a 28×28 image.

3.4.2 Canadian Institute For Advanced Research

The Canadian Institute For Advanced Research (CIFAR-10) dataset is a labeled subset of tiny images dataset (containing 80 million images). It is composed of 60,000, 32×32 color images distributed over 10 different class labels. The dataset consists of 50,000 training samples and 10,000 testing images. Images are uniformly distributed over 10 classes with training batches containing exactly 6000 images for each class. The classes are mutually exclusive and there are no semantic overlaps between the images coming from different labels. We normalized the images using GCA whitening and applied global mean subtraction before training.

3.4.3 Street View House Numbers

The Street View House Numbers (SVHN) is a real world dataset for developing digit recognition algorithms in the wild. The dataset is comprised of over 600,000 images of digits taken from Google Street View images. The dataset is divided into three parts, a smaller more difficult training set comprising of 73,257 samples, a testing set which contains a total of 26,032 samples and finally a bigger set of 531,131 images containing less difficult samples. The data comes in a CIFAR-10 format of 32×32 RGB images centered around a single digit, and contains several images with noisy distractors around the edges. The dataset was preprocessed by mean subtraction and GCA whitening techniques described in [82]. We obtained best results by first training on the original more difficult subset and then finetuning on the extended training set.

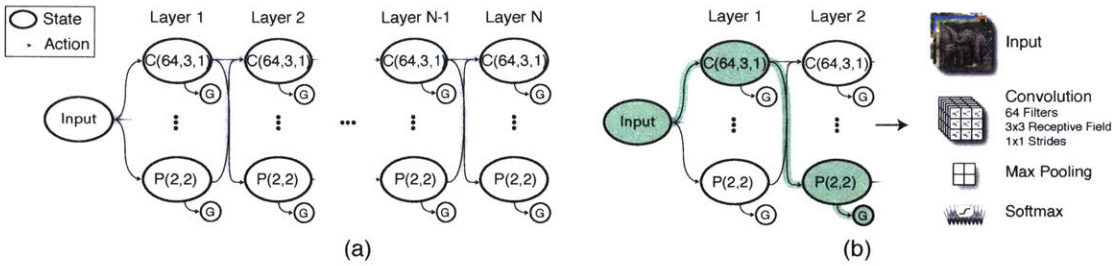


Figure 3-3: Here we show state and action space associated with the context free grammar based reinforcement learning agent. States are denoted in circles and actions are depicted using arrows. The agents starts from initial state and samples layers until it reaches the termination state. $C(n, f, l)$ represents a convolutional layer with n outputs, filter size f, and stride l. $P(f, l)$ denotes a pooling layer with filter size of f and stride l. L denotes a termination state which can be softmax or global average pooling. An example topology is sampled (denoted by yellow arrows) leading to topological configuration shown on the right.

3.5 Experimental Details

In this section we describe how to train reinforcement learning agents to sequentially select neural network layers while optimizing neural network topologies. As described earlier, we will model the layer selection process using a Markov Chain formulation with the assumption that convolutional layers learn *hierarchically*, with varying feature complexities based on their depths[19]. The learning agent sequentially selects layers that maximize the reward for the topology generation graph until it reaches a termination state. Figure 3-3 gives a description of state and action spaces and demonstrates example trajectory observed when sampling topologies. We will represent the expected reward from selecting individual layers by using the $\mathbb{Q}$ function (see equation 3.4).

While we can observe the cumulative reward after sampling an entire topology, rewards from sampling individual layers remain unknown. For this reason, we will apply the Q Learning based formulation when sampling and training layers using the reinforcement learning agent. Q Learning is a *model free* reinforcement learning technique that can learn the optimal action selection for any given Markov Decision Process, without explicit specification of action reward function. We can achieve this by learning the expected utility when selecting any given layer. The $\mathbb{Q}$ values represent the expected *quality* of a state-action combination (as described in equation 3.4).

$$\mathbb{Q} : S \times A \rightarrow \mathbb{R} \tag{3.4}$$

Q value based reinforcement learning involves iteratively sampling the state action space, and updating expected utilities from state-action pairs using the observed rewards. Equation 3.5 describes the iterative approach when computing the Q values. The algorithm begins by initializing dictionary Q with arbitrary random values. With each iteration we sample a new trajectory, observe a reward r_t (validation accuracy) and a new state s_{t+1}. The algorithm computes the $\mathbb{Q}$ values for a state s_t recursively, by combining the current reward with maximum obtainable reward from all possible state-action pairs producing s_{t+1}.

$$\mathbb{Q}(s_t, a_t) \leftarrow (1 - \alpha)\mathbb{Q}(s_t, a_t) + \alpha(r_t + \gamma \max_a \mathbb{Q}(s_{t+1}, a)) \tag{3.5}$$

We assume a learning rate of α and a discounting factor of γ. The algorithm is implemented using a simple dynamic programming based formulation while iteratively updating the reward function. Each action involves generation step in context free grammar, adding a new convolutional, softmax, fully connected or pooling layer. We will describe the state and action spaces in the updatingcoming sections.

3.5.1 The state space

As described in section 3.2 we represent each state using an alphanumeric identifier and a tuple. The alphanumeric identifier represents the *type* of layers, for our experiments these are set to convolution (C), fully connected (FC), pooling (P), softmax (SM) and global average pooling (GAP). The corresponding tuple is used to represent numerical hyperparameters associated with each layer and contain information about the layer specific properties. For example convolutional layer can have hyperparameters that include layer depth, filter size, stride and number of filters, while a fully connected layer may be described using only its depth and number of output neurons. We can also add an additional hyperparameter to pooling and convolutional layers used for representing their input representation tensor dimensions. This allows us to restrict action space to transitions which make sense, for

Layer type	Layer specific parameters	Parameter values
Convolution (C)	$i \sim$ Layer depth	< 12
	$f \sim$ Filter size	Square. $\in \{1, 3, 5\}$
	$\ell \sim$ Filter stride	Square. Always equal to 1
	$d \sim$ Number of filters	$\in \{64, 128, 256, 512\}$
	$n \sim$ Input tensor dimension	$\in \{(\infty, 8]$, $(8, 4]$, $(4, 1]\}$
Pooling (P)	$i \sim$ Layer depth	< 12
	$(f, \ell) \sim$ (Filter size, Filter strides)	Square. $\in \{(5, 3), (3, 2), (2, 2)\}$
	$n \sim$ Input tensor dimension	$\in \{(\infty, 8]$, $(8, 4]$ and $(4, 1]\}$
Fully Connected (FC)	$i \sim$ Layer depth	< 12
	$n \sim$ Number of consecutive FC layers	< 3
	$d \sim$ Number of neurons	$\in \{512, 256, 128\}$
Termination state (L)	$s \sim$ Previous state	
	$t \sim$ Type	Global Avg. Pooling/Softmax

Table 3.1: We describe the various types of layers and corresponding hyperparameters used when describing them.

example a pooling of size 3×3 cannot be applied to representation stacks of size 2×2.

3.5.2 The action space

Context free grammar formulation can be used to create a generalized set of actions taken by reinforcement learning agent. However such set is potentially infinite and needs to be limited in the state-action space for computational feasibility. We make reinforcement learning agent formulation much more tractable by defining state space to a finite set of values (see 3.1) and restricting actions to produce a finite directed acyclic graph. We limit actions to include transitions from layers with depth i to a state with layer depth $i + 1$, removing the possibility of cycles. We further restrict the maximum layer depths to 20 and allow any layer to terminate by addition of softmax or global average pooling layers (loss layers).

Transitions to convolutional layers are allowed only when the previous layer is either the input layer, a convolutional layer or a pooling layer. Similarly, pooling layers are allowed only when the preceding layer is a convolutional layer. We reduce the memory and computational footprint by restricting transitions to fully connected layers when input tensor dimensions exceed $8 \times 8 \times N$. Convolutional layers are allowed to transition to another convolutional, pooling, fully connected or loss layer, whereas a fully connected layer may only transition to another fully connected layer or loss layer. Pooling layers may only transition to a fully connected or a convolutional layer. Additions of these constraints

ϵ	1.0	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1
# Models Trained	1500	100	100	100	150	150	150	150	150	150

Table 3.2: We summarize the training schedule when learning topologies using ϵ greedy descent. The learning agent trains the specified number of unique models at each ϵ.

ensures reasonable computation times while keeping the search space rich of well performing topologies.

3.5.3 Training procedure

We implemented Q Learning using epsilon greedy gradient descent when sampling the state-action space. Under the epsilon greedy policy, the agent selects greedy outcomes with ϵ probability while selecting a uniformly random action with probability $1 - \epsilon$. We begin by setting ϵ to 1, leading to purely random topologies and slowly reduce ϵ in steps of 0.1. Higher values of ϵ correspond to higher randomness in selected topologies leading to more *exploration*, whereas smaller epsilon values select better performing topologies by *exploitation* of knowledge learned during the exploration phase. We sample a large number of topologies in the beginning when ϵ is set to 1. We slowly populate $\mathbb{Q}$ values with better approximations of expected rewards when selecting each layer.

We also maintain a *replay dictionary*, a key value hash map containing previously trained topologies and their corresponding validation accuracies. Every time a topology is sampled, it gets compared against the replay dictionary keys. In case of repetitions, instead of retraining the topology, we use the validation accuracy stored in our replay dictionary. New topologies are sampled and trained in batches of 32. After sampling and training is done, we randomly select 100 topologies from replay dictionary and used them to update the Q values dictionary (see equation 3.5). We obtain faster convergence by applying iterative updates in their reversed temporal order [96].

3.6 Results

Topologies generated by reinforcement learning agent were trained on the three datasets described in section 3.4. The training data was split into training and validation sets with validation set consisting of 5000 samples selected uniformly from each class. Dropout layers were added after every two layers to prevent overfitting, the dropout ratio was varied uni-

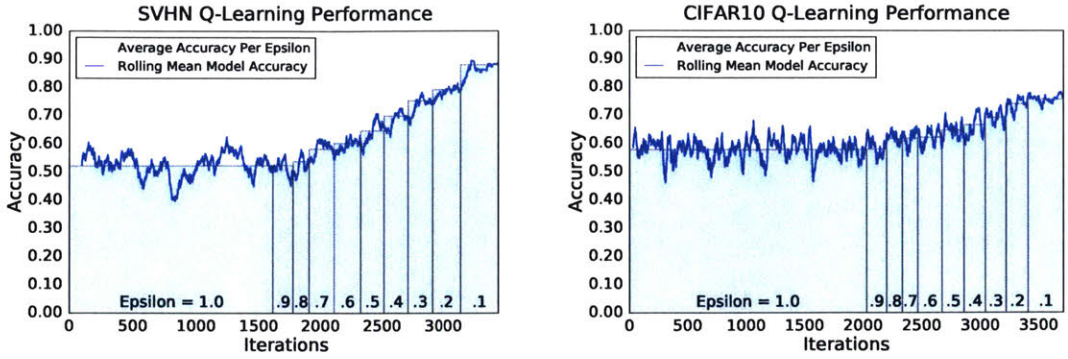


Figure 3-4: In the plots, the blue line shows a rolling mean of model accuracy versus iteration, where in each iteration of the algorithm the agent is sampling a model. Each bar (in light blue) marks the average accuracy over all models that were sampled during the exploration phase with the labeled ϵ. As ϵ decreases, the average accuracy goes up, demonstrating that the agent learns to select better-performing CNN architectures.

formly between 0 and 0.5. Each topology was trained for 20 epochs, using Adam optimizer [97]with momentum of 0.9, weight decay of $5e^{-4}$ and gamma of 0.5. Initial learning rate was set to 0.1, the learning rate reduction was set to 0.2, the batch size was set to 128 and the step size was set to 5 epochs. If the model failed to learn over its first epoch, we reduced the learning rate by a factor of 0.1 and restarted the training with a new random initialization of weights. We used Xavier weight initialization [98] combined with Adam optimizer for faster learning and convergence. We used an array of 10 Nvidia GPUs to reduce computation time and sample topologies much more quickly. Experiments involved sampling of 2500 topologies and took a period of 15 days over each dataset.

Top 10 models obtained after topology selection were finetuned using a longer training schedule. After longer training, top five topologies were used to produce an ensemble. We used simple voting scheme to ensemble our models, see table 3.2 for description of epsilon schedule and table 3.4 for a summary of our results.

Topology selection

We find that our algorithm steadily improves the average validation accuracy of its selected models. The mean validation accuracy shows an increase with every step of ϵ as it is varied from 1 to 0.1, indicating the balance between exploration and exploitation. We show a rolling mean of validation accuracy over 100 models in figure 4-5 for CIFAR-10 and SVHN experiments. The average validation accuracy remains constant when ϵ is fixed, for example

Method	CIFAR-10	SVHN	MNIST	CIFAR-100
Maxout [99]	9.38	2.47	0.45	38.57
HighWay [100]	7.72	-	-	-
NIN [101]	8.81	2.35	0.47	35.68
VGGnet [79]	7.25	-	-	-
FitNet [102]	8.39	2.42	0.51	35.04
All-CNN [103]	7.25	-	-	33.71
AutoML (top model)	**6.92**	2.28	0.44	**27.14***
AutoML (ensemble)	7.32	**2.06**	**0.32**	-

Table 3.3: We compare our performance with CNNs that only use convolution, pooling, and fully connected layers. We report results for CIFAR-10 and CIFAR-100 with moderate data augmentation and results for MNIST and SVHN without any data augmentation.

CIFAR-10 validation accuracy stays at 0.58 % during the exploration phase with $\epsilon = 1$ and it steadily increases to 0.78 as ϵ is reduced to 0.1.

We find interesting motifs in topological hyperparameters discovered by the reinforcement learning agent. The agent routinely selects convolutional layers with 1×1 filters, capable of learning linear transformation on the color space, very similar to the color space transformations such as mean subtraction, whitening and RGB to YUV (see [104, 105]). Consecutive 1×1 filters can also be used to emulate network in network topologies which have be shown produce high quality results[101]. Even though more parameters are important for learning more complex representations, model sizes can vary by huge amounts. For example top five topologies in CIFAR-10 experiments have sizes ranging from 11.26 million to 1.10 million parameters and demonstrate only 2.32% variation in test error.

Analysis of test accuracy

We compare the results from automated architecture selection with current state-of-the-art topologies, that were engineered by humans. For performance evaluation, we report the test errors over a single topology combined with an ensemble comprising of five different topologies. We will split comparisons among two different tables (see tables 3.3 and 3.4). Table 3.3 shows the comparisons when human generated topologies are restricted to the layers available to the reinforcement learning agent. We find that when comparing with topologies containing only convolution, pooling, dropout, fully connected, normalization, softmax and global average pooling, our method beats all existing state-of-the-art topologies. Table 3.4 performs a more general comparison, including topologies containing skipped connections, highway networks, residual connection, densely connected networks and specialized pooling

Method	CIFAR-10	SVHN	MNIST	CIFAR-100
DropConnect [108]	9.32	1.94	0.57	-
DSN [109]	8.22	1.92	0.39	34.57
R-CNN [110]	7.72	1.77	**0.31**	31.75
AutoML (ensemble)	7.32	2.06	0.32	-
AutoML (top model)	6.92	2.28	0.44	27.14*
Resnet(110) [80]	6.61	-	-	-
Resnet(1001) [111]	**4.62**	-	-	**22.71**
ELU [81]	6.55	-	-	24.28
Tree+Max-Avg [82]	6.05	**1.69**	**0.31**	32.37

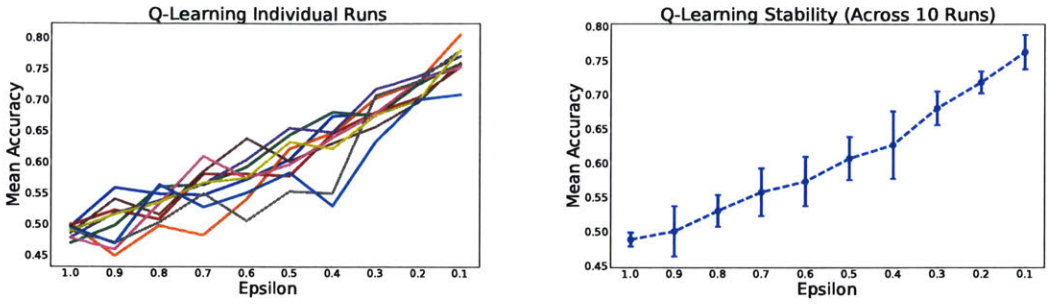
Table 3.4: We compare our error rate with state-of-the-art methods with complex layer types. We report results for CIFAR-10 and CIFAR-100 with moderate data augmentation and results for MNIST and SVHN without any data augmentation.

layers. We demonstrate competitive results when comparing to all existing modern human engineered topologies.

We further compare our topology selection technique with existing automated hyper-parameter optimization methods and demonstrate massive improvements over the test accuracy reported in prior-art. On CIFAR-10 dataset, we see a reduction in test error from 21.2% reported by [106] to 6.92%. We demonstrate similar improvement from 7.9% reported by [107] to 0.32% on the MNIST dataset. Finally an ensemble of top 10 models on MNIST dataset accomplished an error rate of **0.28%** beating the current state-of-the-art results, demonstrating the power of automated architecture selection techniques.

Reinforcement Learning Stability

Since AutoML algorithm involves randomized iterative methods for topology optimization, multiple executions are needed to establish its robustness and efficacy. In this section we analyze the variance in topology accuracies obtained from automated architecture selection. We rerun the ϵ greedy descent 10 different times using 10% of SVHN dataset. A smaller subset of SVHN is used to reduce computational overhead from 100 GPU days to merely 10 GPU days. We induct an array of 10 NVIDIA GPUs to further speed up the process, reducing the execution time to one single day. Figure 3-5 shows accuracy over individual runs and their average accuracies combined with corresponding variances. We observe that for each individual iteration, reinforcement learning agent consistently improves the average accuracies over its selected models. Variance in average accuracy is very small at beginning $\epsilon = 1$ due to similar random distribution and large selection of models. Other stages show



(a) (b)

Figure 3-5: Figure 3-5a shows the mean model accuracy at each ϵ for each independent experiment. Figure 3-5b shows the mean model accuracy and standard deviation at each ϵ over 10 independent runs of the Q-learning procedure on 10% of the SVHN dataset. Despite some variance due to a randomized exploration strategy, each independent run successfully improves architecture performance.

Dataset	CIFAR-100	SVHN	MNIST
Training from scratch	27.14	2.48	0.80
Finetuning	34.93	4.00	0.81
State-of-the-art	24.28 [81]	1.69 [82]	0.31 [82]

Table 3.5: We summarize our prediction errors for the top AutoML (CIFAR-10) model trained for other tasks. Finetuning refers to initializing training with the weights found for the optimal CIFAR-10 model.

higher variance possibly due to fewer models and different parent distributions at each step. We observe a mean accuracy of 88.25% and standard deviation of 0.58% at $\epsilon = 0.1$. Even though the parent distributions can diverge with each iteration of ϵ, we demonstrate that the average variance remains bounded demonstrating the robustness of our method.

Transfer Learning Ability: Topologies including VGG [79] and AlexNet [9] have been demonstrated to perform well at a large selection of computer vision tasks. We test if this is true for automated topology generation, by training the topology selected for one dataset on other datasets. Table 3.5 summarizes the results when training the best topology selected for CIFAR-10 dataset on CIFAR-100, SVHN and MNIST datasets. We find that topologies designed over one dataset perform remarkably well on other datasets, demonstrating transfer learning ability in AutoML algorithm.

[1]Results in this column obtained with the top AutoML architecture for CIFAR-10, trained from random initialization with CIFAR-100 data.

3.7 Concluding Remarks

In this chapter we demonstrate how to automate layer selection process when engineering neural network topologies. Earlier work in this domain primarily focused on selecting only a few hyperparameters or engineering of much smaller topologies. We are one of the first to demonstrate end-to-end optimization of deep neural network layers at performance close to human engineered topologies. However further research is needed in making this process more computationally feasible and robust. We explore a reasonable subset of layers which was further discretized to reduce the computational burden. Further exploration into more diverse set of layers could lead to more interesting insights.

Chapter 4

Distributed learning approaches

While deep neural networks have become the new state of art in classification and prediction on high dimensional data, training of deep neural nets can be extremely data intensive requiring preparation of large scale datasets collected from multiple entities. A deep neural network typically contains millions of parameters and requires tremendous computing power for training, making it difficult for individual data repositories to train them. Emerging technologies in domains such as biomedicine and health, stand to benefit from methods which can allow training of neural networks without requiring data or resource aggregation at one single place.

In domains such as health care and finance, lack of data is a critical issue while developing machine learning algorithms. To address the issue of data scarcity in training and deployment of neural network-based systems, we develop a methods and api to train deep neural networks using distributed means. Our method allows for deep neural networks to be trained using data from multiple entities. Sufficiently deep neural architectures needing large supercomputing resources and engineering oversight may be required for optimal accuracy in real world applications. Furthermore, application of deep learning to such domains can sometimes be challenging because of scarcity of data and lack of absence of platforms that allow data sharing. This chapter attempts to solve these problems by proposing methods that enable training of neural network using multiple data sources and a single supercomputing resource.

In this chapter we propose new techniques that can be used to train multi layer perceptrons over a computer network in a distributed fashion. Specifically we address the problem

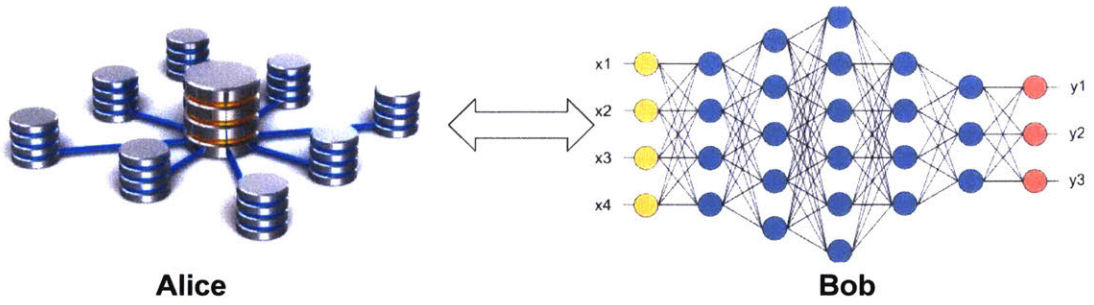


Figure 4-1: We are interested in distributed learning approaches bridging the gap between data sources (Alice) and supercomputing resources (Bob).

of training a deep neural network over several data entities (Alice(s)) and one supercomputing resource (Bob). In upcoming sections we will show how to train neural networks between multiple data entities (Alice(s)) and supercomputing resource (Bob). These techniques will include methods which allow several data repositories to train a neural network with one supercomputing resource training a large section of the neural network topology.

4.1 Related work

Distributed and secure inference continues to be a challenging problem in computer vision. One category of solutions to this problem involve adopting oblivious transfer protocols to perform secure dot product over multiple entities in polynomial time [112]. While this method is secure, its somewhat impractical when considering large scale datasets. A more practical approach proposed in [112] involves sharing only SIFT and HOG features instead of the actual raw data. However as shown in [113], such feature vectors can be inverted very accurately using prior knowledge of methods used to create them. Neural networks have been shown to be extremely robust to addition of noise and their denoising and reconstruction properties make it difficult to compute them securely [35]. Neural networks have also been shown to be able to recover entire image from only a partial input [114] rendering simple obfuscation methods inert.

Multi layer neural networks have been applied to generate secure one-way hashes [115] with high key sensitivity. They have also been used to encrypt JPEG images [116] and generate public and private keys more efficiently [117]. Secure one way hashing schemes can be used for non-linear dimensionality reduction, thereby enabling transmission of information over networks [118]. Combined with homomorphic encryption, such schemes can enable

cloud based feature learning in big data applications [119, 120].

Widespread application of neural networks in sensitive areas such as finance and health, has created a need to develop methods for both secure training [121] and classification in neural networks [122]. Under secure processing paradigms, owner of neural network doesn't have access to actual raw data used to train the neural network [123]. The secure paradigms may also extend to the neural activations and (hyper)parameters. Such algorithms form a subset inside the broader realm of multi-party protocol problems involving secure computation over several parties [124, 125]. Some interesting solutions include using Ada-boost to jointly train classifier ensembles [126], using random rotation perturbations for homomorphic pseudo-encryption [127] and applying homomorphic cryptosystem to perform secure computation [128, 129].

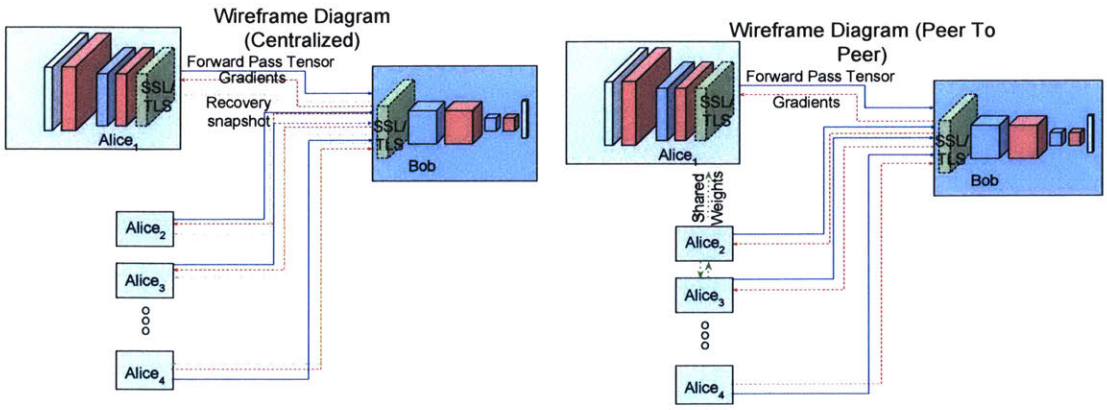
We will compare our method against modern distributed learning methodologies used in data center settings. We compare our method against federated learning and large batch synchronous SGD methods, shown to be state of the art in data center settings [130, 131]. Our baseline involves comparison with large-batch synchronous SGD which is a state-of-the-art method in the data center, shown to outperform asynchronous approaches (proposed in [131, 132]). We will also compare against modern federated SGD methods [130] which modifies the global batch size and relies on asynchronous weight updates followed by model averaging to produce more optimal results. These methods rely on localized optimization by the clients followed by global averaging of the optimized models [133, 132, 134].

4.2 Theoretical underpinnings

In this section we consider the task of training a deep neural network using data from Alice and using computing resources of Bob. Our algorithm can be run using one or multiple data entities, and can be run in peer to peer or centralized mode. Please see Figure 5-2 for schematic depiction of algorithm modalities.

4.2.1 Algorithm for training over a single entity

We will start by describing the algorithm in its simplest form which considers training a neural network using data from a single entity and supercomputing resource. Let us define a deep neural network as a function F, topologically describable using sequence of layers



(a) Centralized distributed neural network training.

(b) Peer-to-peer training for distributed learning.

Figure 4-2: Two modalities of our algorithm. In centralized mode (4-2a) we use a central server to save encrypted weights. In peer to peer (4-2b) data entities (Alice(s)) share weights and download them from last data entity which trained with supercomputing resource (Bob).

$\{L_0, L_1, ...L_N\}$. For a given input $(data)$, the output of this function is given by $F(data)$ which is computed by sequential application of layers $F(data) \leftarrow L_N(L_{N-1}...(L_0(data)))$.

Let $G_{loss}(output, label)$ denote the customized loss function used for computing gradients for final layer. Gradients can be backpropagated over the each layer to generate gradients of previous layer and update current layer. We will use $L_i^T(loss)$ to denote the process of backpropagation over one layer and $F^T(loss)$ to denote backpropagation on entire Neural Network. Similar to forward propagation, backpropagation on the entire neural network is comprised of sequential backward passes $F^T(loss) \leftarrow L_1^T(L_2^T...(L_N^T(loss)))$. Finally, $Send(X, Y)$ represents process of sending data X securely to entity Y. In the beginning, Alice and Bob initialize their parameters randomly. Alice then iterates over its dataset and transmits representations to Bob. Bob computes losses and gradients and sends the gradients back to Alice. Algorithm 3 describes how to securely train a deep neural classifier using a single data source.

Correctness

Here we analyze if training using our algorithm produces same results as a normal training procedure. Under normal training procedure we would first compute forward pass

Algorithm 3 Secure Neural Network training using single data source.

Initialize:
  $\phi \leftarrow$ Random Initializer (Xavier/Gaussian)
  $F_a \leftarrow \{L_0, L_1, ...L_n\}$
  $F_b \leftarrow \{L_{n+1}, L_{n+2}, ...L_N\}$
Alice randomly initializes the weights of F_a using ϕ
Bob randomly initializes the weights of F_b using ϕ
while Alice has new data to train on **do**
  Alice uses standard forward propagation on data
  $\triangleright\ X \leftarrow F_a(data)$
  Alice sends n^{th} layer output X and label to Bob
  $\triangleright\ Send((X, label), Bob)$.
  Bob propagates incoming features on its network
  $\triangleright\ output \leftarrow F_b(X)$
  Bob generates gradients for its final layer
  $\triangleright\ loss \leftarrow G(output, label)$
  Bob backpropagates the error in F_b until L_{n+1}
  $\triangleright\ F_b', loss' \leftarrow F_b^T(loss)$
  Bob sends gradient of L_n to Alice
  $\triangleright\ Send(loss', Alice)$
  Alice backpropagates gradients received
  $\triangleright\ F_a', \_ \leftarrow F_a^T(loss')$

$output \leftarrow F(data)$ followed by computation of loss gradients $G(output, label)$. This loss will be backpropagated to refresh weights $F' \leftarrow F^T(loss)$. Since forward propagation is nothing more but sequential application of individual layers $F(data)$ is same as $F_b(F_a(data))$. Similarly backpropagation $F^T(loss)$ is functionally identical to sequential application of $F_a^T(F_b^T(data))$. Therefore, we can conclude that our algorithm will produce identical results to a normal training procedure.

4.2.2 Generalization for training over multiple entities

Now we demonstrate how to extend algorithm describe in 4.2.1 to train using multiple data entities. We will use the same mathematical notations as used in 4.2.1 when defining neural network forward and backward propagation. Additionally, there are N data entities, each of them is denoted by $Alice_i$.

As an initialization step, Bob sends Alice_1 topological description of first N layers. Alice and Bob use standard libraries for random initialization of their parameters. Bob sets Alice_1 as the last Alice it trained with. We modify 3 and add a step which uses data from multiple entities in a round robin fashion, however Alice_j may be required to update weights before beginning training. Alice(s) can update weights in a peer to peer or centralized fashion,

Algorithm 4 Secure Neural Network over N+1 agents.

Initialize:
  $\phi \leftarrow$ Random Initializer (Xavier/Gaussian)
  $F_{a,1} \leftarrow \{L_0, L_1, ...L_n\}$
  $F_b \leftarrow \{L_{n+1}, L_{n+2}, ...L_N\}$
Alice$_1$ randomly initializes the weights of $F_{a,1}$ using ϕ
Bob randomly initializes the weights of F_b using ϕ
Bob sets Alice$_1$ as last trained
while Bob waits for next Alice$_j$ to send data **do**
  Alice$_j$ requests Bob for last Alice$_o$ that trained
  Alice$_j$ updates its weights
  $\triangleright$ $F_{a,j} \leftarrow F_{a,o}$
  Alice$_j$ uses standard forward propagation on data
  $\triangleright$ $X \leftarrow F_{a,j}(data)$
  Alice$_j$ sends n^{th} layer output and label to Bob
  $\triangleright$ $Send((X, label), Bob)$.
  Bob propagates incoming features on its network
  $\triangleright$ $output \leftarrow F_b(X)$
  Bob generates gradients for its final layer
  $\triangleright$ $loss \leftarrow G(output, label)$
  Bob backpropagates the error in F_b until L_{n+1}
  $\triangleright$ $F'_b, loss' \leftarrow F_b^T(loss)$
  Bob sends gradient of L_n to Alice$_j$
  $\triangleright$ $Send(loss', Alice_j)$
  Alice$_j$ backpropagates the gradients it received
  $\triangleright$ $F'_{a,j}, \_ \leftarrow F_{a,j}^T(loss')$
  Bob sets Alice$_j$ as last trained

please refer to section 4.3.3 for detailed description of both. Once the weights are updated, Alice$_j$ continues its training, please refer to algorithm 4 for detailed pseudo-code describing algorithm.

Correctness

We analyze if training using our algorithm produces identical results as a normal training procedure would (under assumption that the data arriving at multiple entities is used in same sequence and random weights use same initialization). The algorithm correctness stems from the fact that Bob and at least on of Alice$_o$ have identical neural network parameters at iteration$_k$. We use inductive techniques to prove that this is indeed the case.

Lemma 1 *The neural network currently being trained is identical to neural network if it was trained by just one entity.*

Base Case: Alice$_{1...N}$ have same weights at beginning of first iteration.

Proof: Alice_1 randomly initialized weights and transmitted them to $\text{Alice}_{2...N}$ in beginning making them identical to training from just one entity.

Recursive Case: Assertion: If Alice_j has correct weights at beginning of iteration_i it will have correct weights at beginning of iteration $i+1$.

Proof: Alice_j performs backpropagation as the final step in iteration i. Since this backpropagation is functionally equivalent to backpropagation applied over entire neural network at once, Alice_j continues to have correct parameters at the end of iteration. ($F^T(loss)$ is functionally identical to sequential application of $F^T_{a,j}(F^T_b(data))$).

4.2.3 Online learning

An additional advantage of using our algorithm is that the training can be performed in an online fashion, by providing Bob output of forward propagation whenever there is new annotated data. Further in the beginning instead of transmitting the entire neural net, Alice_i can initialize the weights randomly using a seed and just send the seed to $\text{Alice}_{1...N}$ preventing further network overhead. When Alice is requested for weights in peer to peer mode, it can simply share the *weight updates*, which it adds to its parameters during the course of training. The combined value of weight updates can be computed by subtracting weights at beginning of training from current weights. Weights can be refreshed by Alice by combining its initial weights with subsequent weight updates it downloads from centralized weight server (or Alice(s) depending on mode). To facilitate centralized modality, we can modify step 6 of 4 and 5, replacing it with a request to download encrypted weights from weight server. Once training is over Alice_j can upload the new encrypted weights to weight server (step 15 in 4 and step 16 in 5).

4.2.4 Semi-supervised application over multiple entities

In situations with fewer labeled data-samples, a reasonably approach includes learning hierarchical representations using unsupervised learning [135]. Compressed representations generated by *autoencoders* can be used directly for classification [15]. Additionally, we can combine the losses of generative and predictive segments to perform semi supervised learning, adding a regularization component while training on fewer samples [36].

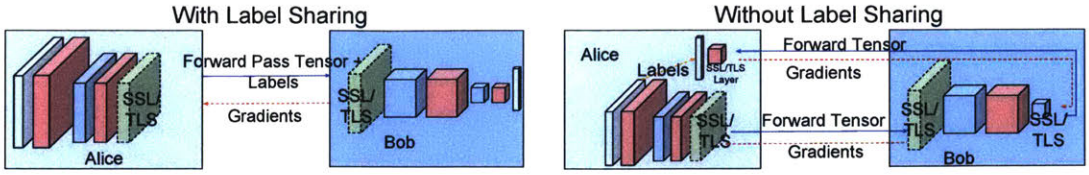
In this section we describe how to modify split neural network algorithm to incorporate semi-supervised learning and generative losses. We assume that out of n layers for

Algorithm 5 Training Split Neural Network with an Autoencoder over N+1 agents.

Initialize:
  $\phi \leftarrow$ Random Initializer (Xavier/Gaussian)
  $F_{e,1} \leftarrow \{L_0, L_1, ...L_m\}$
  $F_{d,1} \leftarrow \{L_m, L_{m+1}, ...L_n\}$
  $F_b \leftarrow \{L_{n+1}, L_{n+2}, ...\}$
Alice$_1$ randomly initializes the weights of $F_{a,1}$ using ϕ
Bob randomly initializes the weights of F_b using ϕ
Alice$_1$ transmits weights of $F_{a,1}$ to Alice$_{2...N}$
while Bob waits for next feature vector from Alice$_j$ **do**
  Alice$_j$ requests Bob for last Alice$_o$ that trained
  Alice$_j$ updates its weights
  ▷ $F_{a,j} \leftarrow F_{a,o}$
  Alice$_j$ uses standard forward propagation on data
  ▷ $X_m \leftarrow F_{e,j}(data)$
  ▷ $X \leftarrow F_{d,j}(X_m)$
  Alice$_j$ sends m^{th} layer output and label to Bob
  ▷ $Send((X_m, label), Bob)$.
  Bob propagates incoming features on its network F_b
  ▷ $output \leftarrow F_b(X_m)$.
  Bob generates loss for its final layer
  ▷ $loss \leftarrow G(output, label)$
  Bob backpropagates the error in F_b until L_{n+1}
  ▷ $F'_b, loss' \leftarrow F_b^T(loss)$
  Bob sends gradient for L_n to Alice$_j$
  ▷ $Send(loss', Alice_j)$
  Alice$_j$ generates autoencoder loss for its decoder
  ▷ $F'_{d,j}, loss'_{enc} = F_{d,j}^T(X)$
  Alice$_j$ backpropagates combined gradients
  ▷ $F_{a,-} \leftarrow F_a^T(\eta(loss', loss'_{enc}))$
  Bob sets Alice$_j$ as last trained

Alice, first m layers are encoder and the remaining $n - m$ layers belong to its decoder. $F_{e,i}$ denotes the forward propagation over encoder (computed by sequential application $L_m(L_{m-1}...(L_0(data))))$. $F_{d,i}$ denotes application of decoder layers. During forward propagation Alice propagates data through all n layers and sends output from m^{th} layer to Bob. Bob propagates the output tensor from alice through $L_{n...N}$ and computes the classifier loss (logistic regression).

Let $loss$ define the logistic regression loss in predictive segment of neural network (last $N - n$ layers owned by Bob), and let $loss_{enc}$ define the contrastive loss in autoencoder (completely owned by Alice(s)). Bob can compute $loss$ using its softmax layer and can backpropagate this loss to layer L_{n+1} giving gradients from classifier network $[loss' \leftarrow F_b^T(loss)]$. Alice$_i$ can compute the autoencoder loss and can backpropagate it through it's *decoder* network $[F_{d,i}^T(loss_{enc})]$. We can facilitate semi-supervised learning by combining weighted



(a) Training with label propagation

(b) Training without label sharing

Figure 4-3: Figure (4-3a) shows the normal training procedure while figure (4-3b) demonstrates how to train without transmitting labels, by wrapping the network around at its last layers.

sum of two losses. The weight α is an added hyperparameter which can be tuned during training.

$$\eta \leftarrow F_b^T(loss) + \alpha * F_{d,i}^T(loss_{enc}) \tag{4.1}$$

After the initialization steps, Alice propagates its data through her network and sends output from the encoder part to Bob. Bob does a complete forward and backward to send gradients to Alice. Alice then combines losses from its decoder network with gradients received from Bob and uses them to perform backpropagation (please see 5 for detailed description).

4.2.5 Training without label propagation

While the algorithm we just described doesn't require sharing of raw data, it still does involve sharing of labels. We can mitigate this problem by presenting a simple adjustment to the training framework. In this topographical modification, we wrap the network around at its end layers and send those back to Alice. While Bob still retains majority of its layers, it lets Alice generate the gradients from the end layers and uses them for backpropagation over its own network. We can use a similar argument as one used in 1 to prove that this method will still work after the layers have been wrapped around. Please see figure 4-4 for schematic description of our training methodology without label sharing.

4.3 Network implementation for distributed learning

We ensure safety against man-in-middle attacks, by using RSA cryptography and SSL encrypted sockets for network communication. Each of Alice is given a unique user id and password which is used to establish identity of Alice while connecting. When communicating over SSL, client and server initiate a hand shake using the asymmetric public key provided by the server. After a successful handshake, client and server agree upon a temporary key used during session to perform symmetric cryptography. We add additional safeguards by adding timeouts when either side is idle and we use trusted sources to generate SSL certificates.

We use standard *json* communication libraries for asynchronous RPC for implementation. On top of those, we implement a custom protocol for training once a secure connection is established using SSL. Our protocol defines several network primitives (implemented as remote functions) which we broadly divide in 3 parts (1) Training request, (2) Tensor transmission and (3) Weight update. Please refer to table 5.3 for a complete list of network primitives. We describe these three network primitives categories below.

4.3.1 Training request

Training request is the first message sent by Alice to Bob in the beginning of a session. Training request message carries information to establish identity, SHA-2 256 bit *checksum* (a cryptographically secure hash function) of latest weights available along with the number of training batches. Bob uses this to verify identity of Alice and ensure that the weights available to Alice are the most recent weights. Once bob has established identity it responds if Alice needs to refresh its weights. In case the checksum is older and Alice needs to refresh, she is provided with the identity of Alice which trained latest with Bob. Bob simply disconnects after transmission in cases when message is malformed or identity was mismatched.

4.3.2 Tensor transmission

Tensor transmission is the message used by both Alice and Bob to transmit forward propagated activations and backpropagated losses. Message carries the actual shape of tensors, raw data of tensors in a serialized form along with a checksum for transmitted tensor to

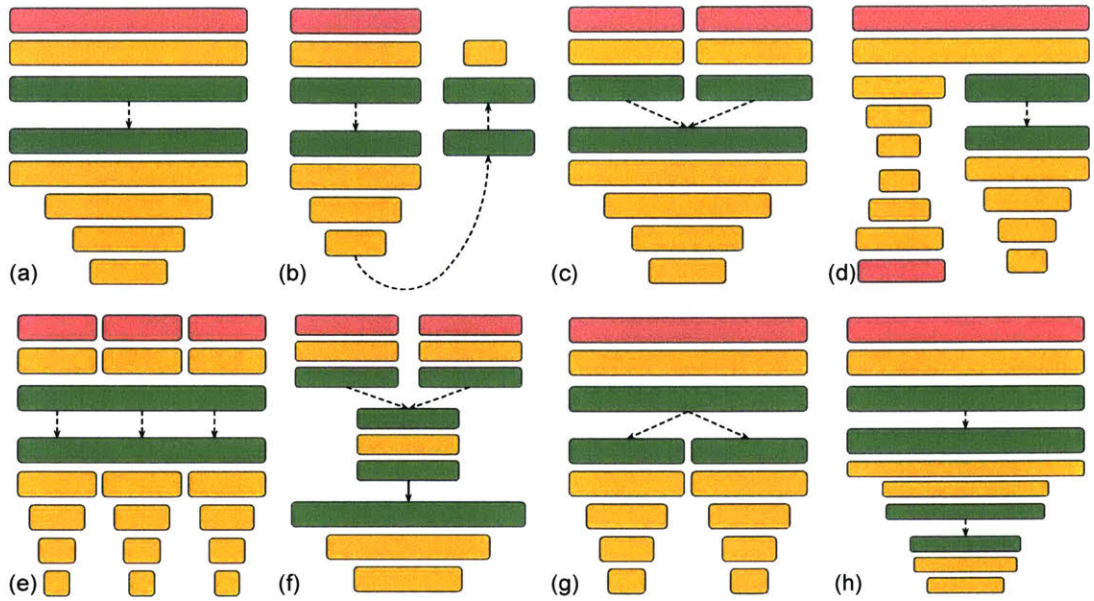


Figure 4-4: We explore several interesting extensions of distributed learning platform. The figure above shows the eight different configurations we implemented and tested. These include (a) Simple vanilla configuration (b) Wrap around configuration with labels (c) Multi-agent learner (d) Semi-supervised learner (e) Ensemble learner (f) Splitting data in space or time over different agents (g) Multi-task learner (h) Tor like configuration involving several nodes.

ensure data integrity. We combine together tensors from a single batch and transmit them as a single blob. For testing purposes tensor transmission from Alice can carry a mode string to specify if data used was training or validation.

4.3.3 Weight update

Network primitives for weight update differ based on if we are operating in *peer-to-peer* or *centralized* mode. In *peer-to-peer* mode, Bob sends last trained Alice's information to current training party and Alice use this to connect and download encrypted weights. This method has a limitation that the data nodes cannot go offline after the training has finished.

In the *centralized* mode, Alice uploads encrypted weights file to a weight server using an "Encrypted weight upload" primitive. When a new party wants to train, it downloads and decrypts these weights. Encryption and decryption is performed using existing methods for PSK. While this removes the need for Alice to stay online, it makes it harder to add new nodes since they need to establish identity and obtain the PSK before attempting to train.

Dataset	Topology	Accuracy (Single Agent)	Accuracy using our method
MNIST	LeNet	99.18 %	99.20 %
CIFAR 10	VGG	92.45 %	92.43 %
CIFAR 100	VGG	66.47 %	66.59 %
ILSVRC 12	AlexNet	57.1 %	57.1 %

Table 4.1: We observe same accuracies when training using multi-agent algorithm vs when training on a single machine. MNIST dataset is verified using LeNet topology. We use modified VGG to verify accuracy on CIFAR 10 and CIFAR 100. Finally we verify our method on very large dataset (ILSVRC 12) using AlexNet topology.

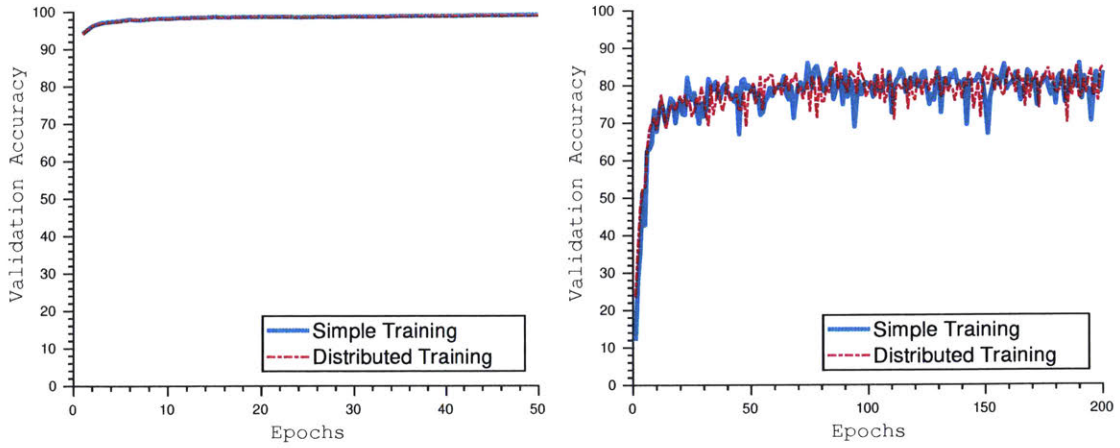
Another limitation arises from shared keys being susceptible to attacks and therefore need recycling and updating.

4.4 Experimental evaluation and comparison

We implement our algorithm and protocol using python bindings for *caffe*. We test our implementation on datasets of various sizes (50K - 1M) and classes (10, 100 or 1000 classes). We demonstrate that our method works across different topologies and experimentally verify identical results when training securely over multiple agents.

We compare our method against the large-batch global SGD [131] and federated averaging approaches [130]. Federated averaging techniques require definition of 4 different hyperparameters – B refers to the local mini batch, hyperparameter C refers to the fraction of clients used in each iteration, E refers to number of training passes made by a client in each round and K refers to total number of clients. Large-batch global SGD (also referred as federated SGD) involves gradient and loss computation over all the data held by clients (setting C equal to 1 and B equal to ∞). Selecting best hyperparameters can be critical to obtain competitive performance, as described in [130].

We perform several different comparisons using the best hyperparameter selections for federated averaging and federated SGD. For our first comparison we analyze data the transmission cost of state-of-the-art deep networks including ResNet and VGG on CIFAR-10 and CIFAR-100 (please refer to 4-7). We demonstrate higher validation accuracy and faster convergence when considering large number of clients. We also compare client side computational costs when using deep models and demonstrate significantly lower computational burden on clients when training using our algorithm (see figure 4-6). We summarize our findings in table 4.3 and 4.4. Overall analyses can be found in the schematic depicted in



(a) Accuracy when training on MNIST

(b) Accuracy when training on CIFAR-10

Figure 4-5: Convergence characteristics with iteration count for MNIST (4-5a) and CIFAR-10 (4-5b). We observe same convergence rate using multi agent algorithm v/s when training using a single machine.

Figure 4-8.

4.4.1 Empirical verification of algorithm

In 4.2.2 we show why our algorithm should give results identical to a normal training procedure. We verify our methods correctness by implementing it and training it on a wide array of datasets and topologies including MNIST, ILSVRC 12 and CIFAR 10. Table 4.1 lists datasets and topologies combined with their test accuracies.

4.4.2 Accuracy validation

We verify that our algorithm takes similar number of iteration to converge, proving that the method doesn't add inefficiencies at an algorithmic level. We plot the validation accuracies with every iteration on different datasets. Figure 4-5 shows plots from 2 different datasets. The convergence characteristics from split neural network algorithm coincide well with the training trajectory when training over a single agent. While the overall convergence characteristics match, there are small variations observed because of different parameter initializations at the beginning of training.

Dataset	Accuracy using 1 agent (10 %)	Accuracy using 5 agents (50 % of data)	Accuracy using all agents
MNIST	97.54	98.93	99.20
CIFAR 10	72.53	89.05	92.45
CIFAR 100	36.03	59.51	66.59

Table 4.2: We show significant improvements in accuracy as more data-sources are added.

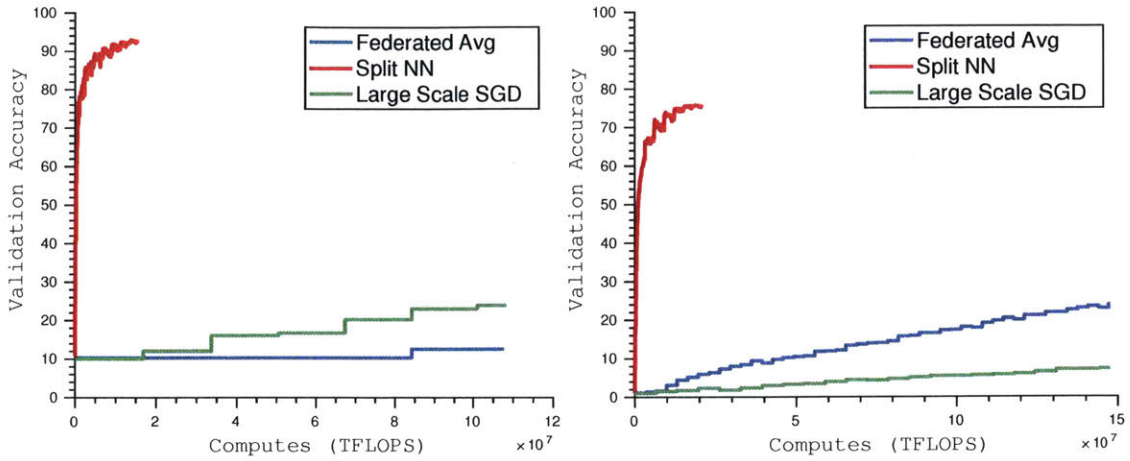
4.4.3 Performance analysis

An important benefit of our method lies in its ability to combine multiple data-sources. When using deep neural networks, larger datasets have been shown to perform significantly better than smaller datasets. We experimentally demonstrate the benefits of pooling several agents by uniformly dividing dataset over 10 agents and training topologies using 1, 5 or 10 agents. As shown in table 4.2, we observe that adding more agents causes accuracy to improve significantly. We also compare our method against federated learning and large batch stochastic gradient descent for communication and computational overhead. In figure 4-7, we plot validation accuracy with bytes transferred using our method and observe that we obtain higher validation accuracies using lesser communication bandwidth. Similarly figure 4-6 shows that we require significantly lower computations on each client to achieve state-of-the-art validation accuracy.

4.5 Concluding Remarks

In this chapter we present new methods to train deep neural networks over several data repositories. We also present algorithms on how to train neural networks without revealing actual raw data, while reducing computational requirements on individual data sources. We describe how to modify this algorithm to work in semi-supervised modalities, greatly reducing number of labeled samples required for training. We provide mathematical guarantees for correctness of our algorithm.

We devise a new protocol for easy implementation of our distributed training algorithm. We use popular computer vision datasets such as CIFAR-10 and ILSVRC12 for performance validation and show that our algorithm produces identical results to standard training procedures. We also show how this algorithm can be beneficial in low data scenarios by combining data from several resources. Such method can be beneficial in training using



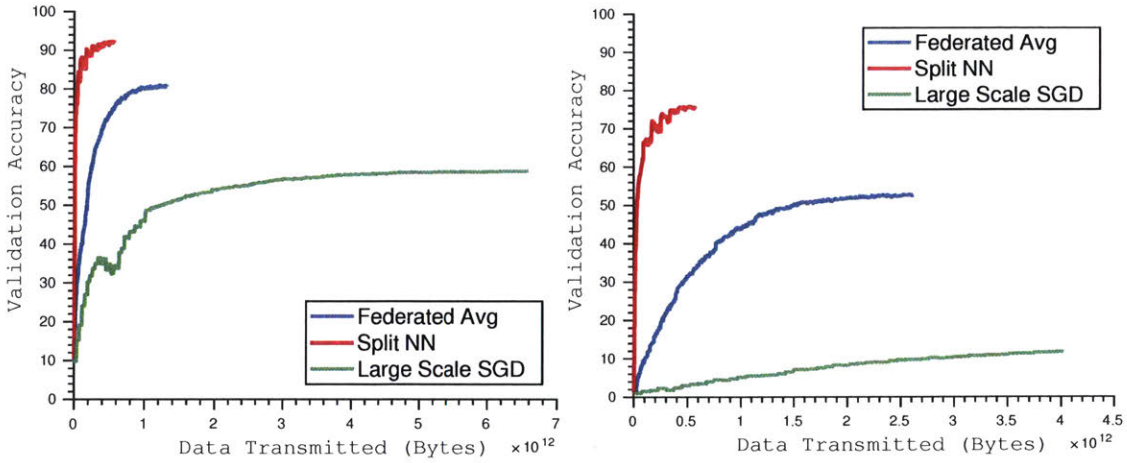
(a) Validation accuracy with client side flops when training 100 clients (VGG).

(b) Validation accuracy with client side flops when training 500 clients (Resnet-50).

Figure 4-6: We compare client side computational cost of our method against existing state of the art methods when training with multiple clients. Red line denotes distributed learning using our method, blue lines indicate federated averaging and green line indicates large batch stochastic gradient descent. As shown above, we reduce the computational burden on clients dramatically while maintaining higher accuracies when training over large number of clients.

proprietary data sources when data sharing is not possible. It can also be of value in areas such as biomedical imaging, when training of deep neural network without revealing personal details of patients and minimizing computation resources required on devices.

In this chapter we describe a method to train a single network topology over several data repositories and a computational resource. A reasonable extension to this approach can be to train an ensemble of classifiers by transmitting forward and backward tensors for all classifiers every iteration. A deep neural network classifier ensemble can comprise of several individual deep neural network topologies which perform classification. The network topologies are trained individually by computing forward and backward functions for each neural network, and during the testing phase the results are combined using majority vote to produce classification. We can train such an ensemble by generating separate forward and backward propagation tensors for each neural network and transmitting them during each training iteration. This is equivalent to training individual networks one by one, but it saves time by combining iterations of various networks together. Ensemble classifiers have also been shown to be more secure against network copy attacks and have also been shown to perform better in real world applications [136].



(a) Validation accuracy with transmitted data when training 500 clients using VGG over CIFAR-10.

(b) Validation accuracy with transmitted data when training 500 clients using Resnet-50 over CIFAR-100.

Figure 4-7: We compare data transmission costs of our method against existing state of the art methods when training with multiple clients. Red line denotes distributed learning using our method, blue lines indicate federated averaging and green line indicates large batch stochastic gradient descent. As shown above, the validation accuracy for our method remains higher with same number of bytes transferred, making our method overall a better choice when training over large number of clients.

We demonstrate significant reduction in computation and communication bandwidth when comparing against federated SGD and federated averaging [130]. Reduced computational requirements can be explained by the fact that while federated averaging requires forward pass and gradient computation for entire neural network on the client, our method requires these computations for only the first few layers, significantly reducing the computational requirements (as shown in table 4-6). Even though federated averaging requires a lot fewer iterations than large-scale SGD, it is still outperformed by our method requiring only a fraction of computations on the client.

Reduction in communication bandwidth can be attributed to the fact that federated averaging involves transmitting the gradient updates for entire neural network from all clients to a central server, accompanied by transmission of updated weights to every single client. While federated averaging algorithm is able to converge in fewer transmission cycles, each transmission cycle requires huge amounts of data download and upload to client and server. Split neural network algorithm reduces data transmitted by restricting size of client neural network to only first few layers, thereby greatly reducing the total amount of data transmitted during training. Additionally federated averaging fails to achieve optimal accuracy

Method	100 Clients	500 Clients
Large Scale SGD	29.4 TFlops	5.89 TFlops
Federated Learning	29.4 TFlops	5.89 TFlops
Our Method	0.1548 TFlops	0.03 TFlops

Table 4.3: Computation resources consumed per client when training CIFAR 10 over VGG (in teraflops).

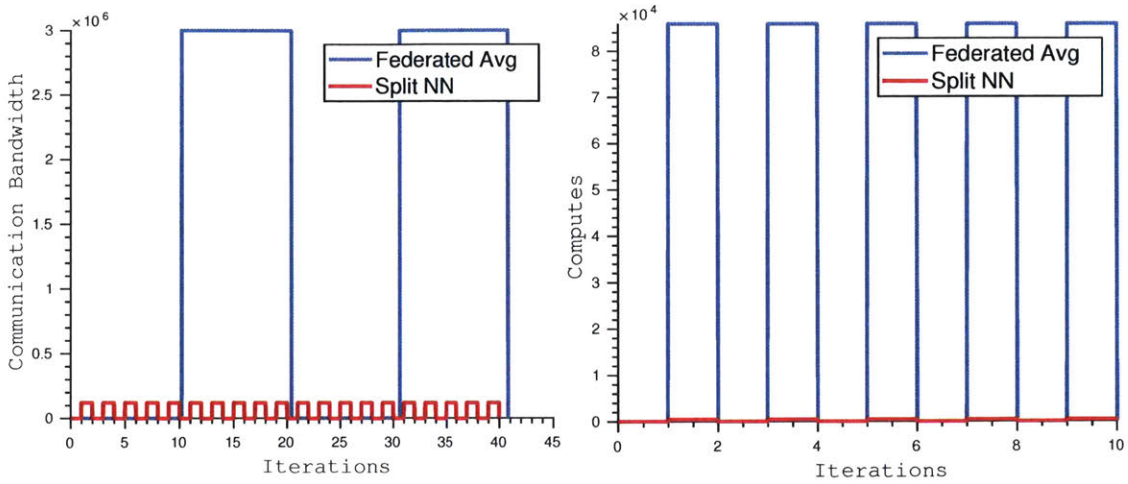
Method	100 Clients	500 Clients
Large Scale SGD	13 GB	14 GB
Federated Learning	3 GB	2.4 GB
Our Method	6 GB	1.2 GB

Table 4.4: Computation bandwidth required per client when training CIFAR 100 over ResNet (in gigabytes).

for higher number of clients since for general non-convex optimization averaging models in parameter space could produce an arbitrarily bad model (phenomenon described in [137]).

Learned neural network could be shared using student-teacher methods for transferring information learned by neural network [138]. After the training phases are over, Alice and Bob can use any publicly available dataset to train secondary (student) neural network using outputs from primary (teacher) neural network. Alice can propagate the same training sample from public dataset through the layers from previously trained network and Bob can propagate them through its network. Bob can use the output of its layers to train student network by doing forward-backward for same data sample. This way, knowledge of distributed trained network can be transferred to another network which can be shared for public use. Such algorithms can help in introducing deep learning in several areas such as health, products and finance where user data is an expensive commodity and needs to remain anonymized.

Tor like layer-by-layer computation could allow for training this network over multiple nodes with each node carrying only a few layers. Such method could help protect not just the data but identity of person sharing the data and performing classification. In Tor like setup, additional entities $Eve_{0...M}$ are added which do not have access to data or complete network topology. Each Eve is provided with a few of network layers $F_k^{eve} \leftarrow L_q, L_{q+1}...L_r$. During forward propagation Alice computes F_a and passes it to Eve_0, which then passes it to Eve_1 and so on until it reaches Eve_M. Eve_M is analogous to the exit node in Tor network and it passes the tensor to Bob. Similarly, when backpropagating, Bob computes



(a) Data resources required with iterations

(b) Computational resources with iterations

Figure 4-8: We summarize the computational and data bandwidth requirements using schematic diagrams in Figure (4-8a, 4-8b).

loss and sends it to Eve_M, which sends it to Eve_{M-1} and so on until it reaches Eve_0 and then Alice. The onion like organization of network layers can be used to keep the identity of Alice confidential.

We can also apply our algorithm on not just classification tasks but also on regression and segmentation tasks. We can also use this over LSTMs and Recurrent Neural Networks. Such neural networks can be easily tackled by using a different loss function (euclidean) on Bob's side when generating gradients.

Chapter 5

Conclusions

Traditionally, problem solving on imaging and sensor data tended to be model driven and involved developing physical and mathematical models when describing system characteristics. Over the last decade, emphasis has shifted from model driven to data driven methodologies. Deep multi-layered perceptrons have become the new gold standard when solving real world problems using machine learning. Deep neural networks have demonstrated amazing capability at solving problems in segmentation, detection and classification while surpassing human level capabilities. However, training deep neural networks continues to be a challenging task because of lack of labeled data, data sensitivity issues and shortage of human expertise. Deep neural networks typically require large amounts of human intervention involving user input in labeling of data and manual optimization of neural network layers.

In this thesis we explore methods which reduce requirement of labeled data when training using deep neural networks. We explore how generated data combined with new layers can be used to learn effective classification models. We make our neural networks more robust by introducing invariance to changes in illumination, velocity and other calibration parameters. We introduce methods for automated layer and hyperparameter selection, thereby reducing the expert intervention involved in neural network deployment. We also introduce a new layer for distributed learning over the network, which aids in learning neural networks in data sensitive applications. Our method aids in developing data driven techniques by reducing the friction associated with accessing data when training neural networks over multiple parties.

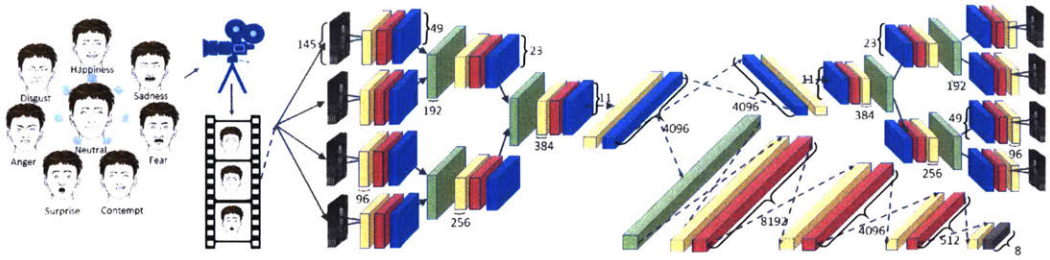


Figure 5-1: We introduce semi-supervised learning to video clips when learning facial expressions. We train an autoencoder of facial videos and learn the facial expressions using a semi-supervised predictor. Additionally we introduce topological modifications to aid in learning invariants.

5.1 Key Results

In this section, we will summarize the key results and findings of this thesis. These will include the results in learning invariants using neural network layers, using generative methods such as autoencoders and Markov Decision Processes to improve machine learning pipelines, optimizing neural network layers over imaging, video and other data for real world applications and specialized layers that aid in distributed learning.

5.1.1 Neural network layers to learn invariants

We began by introducing a framework for gesture and pose recognition which combines semi-supervised learning approaches with carefully designed neural network topologies. We introduced a new layer which can learn on video data adaptively and demonstrate how to learn invariants using these layers. We demonstrated how to induce illumination invariance by including specialized layers and use spatio-temporal convolutions to extract features from multiple image frames. In this work, we emphasize on learning invariants on velocity, illumination and other camera parameters, in future we hope to explore induction of even more invariants, which continues to be an area of rapid research in neural networks.

5.1.2 Optimization of neural network layers

Topology engineering continues to be a challenging task involving days of expert intervention and human intuition when selecting best layers. We explored how layer engineering can be automated by defining the search space using context free grammars and applying

Model Architecture	Test Error (%)	# Params (10^6)
[C(512,5,1), C(256,3,1), C(256,5,1), C(256,3,1), P(5,3), C(512,3,1), C(512,5,1), P(2,2), SM(10)]	6.92	11.18
[C(128,1,1), C(512,3,1), C(64,1,1), C(128,3,1), P(2,2), C(256,3,1), P(2,2), C(512,3,1), P(3,2), SM(10)]	8.78	2.17
[C(128,3,1), C(128,1,1), C(512,5,1), P(2,2), C(128,3,1), P(2,2), C(64,3,1), C(64,5,1), SM(10)]	8.88	2.42
[C(256,3,1), C(256,3,1), P(5,3), C(256,1,1), C(128,3,1), P(2,2), C(128,3,1), SM(10)]	9.24	1.10
[C(128,5,1), C(512,3,1), P(2,2), C(128,1,1), C(128,5,1), P(3,2), C(512,3,1), SM(10)]	11.63	1.66

Table 5.1: We summarize the top 5 model architectures when training over CIFAR-10 using CFG based topology generation pipeline. We observe that number of parameters in top performing deep neural architectures may vary widely.

reinforcement learning driven approaches to select the best topologies. We show how to apply this practically by intelligently pruning the search space thereby restricting the action space taken by reinforcement learning agent. We demonstrated that automated engineering of layers can be used to produce architectures that are competitive to human engineered topologies. We explored the trends followed by layers and analyze which layers perform best when selecting the architectures with higher validation accuracy. In table 5.1, we show the top 5 topologies selected using our method.

5.1.3 Distributing layers over the network

We also presented new methods to train deep neural networks in a distributed fashion over several data repositories. We presented algorithms on how to train neural networks without sharing actual raw data or labels, while reducing computational requirements on individual data sources. We described how to modify this algorithm to work in semi-supervised modalities, greatly reducing number of labeled samples required for training. We analyzed mathematical guarantees for correctness of our algorithm. We demonstrated how this method can be extended to incorporate techniques such as multi-task learning, distributed learning over several concurrent nodes (like *tor*) and ensemble learning.

5.1.4 Datasets and collection methodologies

Training the unsupervised component of our neural net required a large amount of data to ensure that the deep features were general enough to represent any face expression. We trained the deep convolutional autoencoder using a massive collection of unlabeled data

Emotion	Posed	Non-Posed	Cumulative
Anger	132	318	450
Sadness	118	148	266
Contempt	153	301	454
Fear	137	96	233
Surprise	188	232	420
Joy	172	503	675
Disgust	132	147	279
Total	1032	1745	2777

Table 5.2: Data distribution for Asevodataset for various emotions. Posed clips refer to the artificially generated clips, while non-posed refer to those captured using the stimulus activation procedure.

points comprising of 6.5 million video clips with 25 image frames per clip. The clips were generated by running Viola-Jones face algorithm to detect and isolate face bounding boxes on public domain videos. We further enhanced the data quality by removing any clips which showed high variation in-between consecutive frames. This eliminated video clips containing accidental appearance of occlusions, rapid facial motions or sudden appearance of another face. As an additional step we obtained the facial pose information by using active appearance models and generating facial landmarks [70]. We fitted the facial landmarks to a $3D$ deformable model and restricted our dataset to clips containing less than 30 degrees of yaw, pitch or roll, thereby eliminating faces looking sideways. Collection of this dataset required development of an automated system to mine video clips, segment faces and filter meaningful data and it took us more than 6 months to collect the entire dataset. To our knowledge this is the largest dataset containing facial video clips and we plan to share it with scientific community by making it public.

We developed specialized video recording and annotation tools to collect and label all of video data (first presented in [20]). The application was developed in Python programming language and we used well known libraries such as OpenCV for video capture and annotation. We further extended the tools to create a secure web-based annotation platform (named *pubmed*) which can be used to create and annotate expertly labeled data. The database contains facial clips from 160 subjects (both male and female), where expressions were artificially generated according to a specific request, or genuinely given due to a shown stimulus. We captured 1032 clips for posed expressions and 1745 clips for induced facial ex-

pressions amounting to a total of 2777 video clips. Genuine facial expressions were induced in subjects using visual stimuli, i.e. videos selected randomly from a bank of YouTube videos to generate a specific emotion. Please refer to Table 5.2 to see the distribution of database, where posed clips refers to the artificially generated expressions and non-posed refers to the stimulus activation procedure.

We also generated massive number of neural networks trained on datasets including CIFAR-10, CIFAR-100 and MNIST. We plan to share the trained models and autoencoder dataset with rest of scientific community.

5.2 Limitations and Future Work

In this section we explore limitations of our system and discuss where our system may fail or be of less value. While we used scale-invariant methods for gesture recognition, it will be interesting to examine how much improvement these could provide over general video classification techniques. Physiological signals such as heart rate and respiratory rate can be extracted from color [139] and motion variation in the video data [140, 141] and can be used to improve emotion recognition in non-posed expressions [142]. In the future, neural nets could be pre-trained to extract these signals and we could use the deep features from these nets to improve the classification accuracy. Currently, our system relies on utilization of Viola-Jones to distinguish and segment out the faces and is limited to analyzing only the front facing views. Emotion recognition in the wild still remains an elusive problem with low reported accuracies which we hope will be addresses in future work.

Learning for deep neural networks can be extremely computationally intensive and can impose massive constraints on systemic space-time complexity. Our neural network invariant layers are no different and require specialized hardware (NVIDIA Tesla^{TM}or K40^{TM}Grid GPUs) with a minimum of 12 GB of VRAM on the graphics card for lowest of batch sizes. Deep autoencoders can be data intensive and require millions of unlabeled samples to train. Further the stacked autoencoder we train takes over 3 days to train requiring an additional day to fine tune predictor weights for larger labeled datasets. Even though the system supports 7 emotions and 1 neutral face state, it was not trained to detect neutral emotions - a constraint which can be fixed by adding more labeled data for neutral facial expressions. The pipeline only recognizes 7 facial emotions but recent research shows that there is a

much wider range of emotions.

Limitations of automated architecture selection

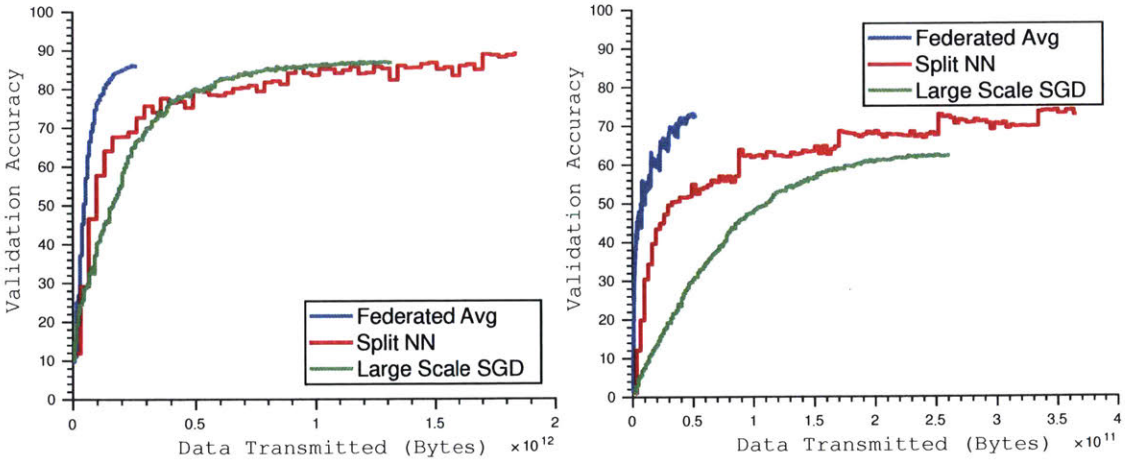
Our system on automated architecture generation is also resource intensive and requires multiple GPUs to search over the space of hyperparameters. Even after application of methods such as early stopping and search space pruning, we required tens of GPUs and several days to select best architectures. More research in both faster hardware and better machine learning techniques will be required to make architecture selection an attractive alternative to downloading standardized topologies. We demonstrate our system on small and medium sized datasets, however far greater amount of resources are required to adapt such systems on datasets such as ImageNet. We were able to demonstrate the method efficacy on classification pipelines, and further research is needed to demonstrate the full potential on other applications in fields like natural language processing.

Limitations of the distributed learning approach

The distributed learning framework requires high network bandwidth to send and receive data. While our method is able to produce comparable and better results to state-of-the-art techniques, it may require larger computational bandwidth when training over smaller number of clients. Advanced compression methods can be used to reduce the network footprint and make the method less data intensive. The comparative performance overhead gets reduced as we add more clients, producing a much more scalable distributed learning methodology. The current implementation also requires constant network connection, which can be fixed by adding asynchronous communication with automatic reconnection techniques. We rely on TCP socket communication which has several added overheads, some of these can be eliminated by introducing protocols such as UDP allowing for errors in network transmission.

5.3 Concluding Remarks

Deep neural networks have revolutionized modern computer vision, excelling at tasks such as segmentation, object classification [9] and facial recognition [11]. Such models have been shown to perform at levels higher than human accuracy, enabling automation in human



(a) Validation accuracy when training 10 clients using VGG topology on CIFAR 10 dataset.

(b) Validation accuracy when training 10 clients using ResNet-50 on CIFAR 100.

Figure 5-2: Limitations of out method when using fewer clients. In this figure we demonstrate how split neural networks can have higher communication overhead when fewer clients are being used to train. Red line denotes distributed learning using our method, blue lines indicate federated averaging and green line indicates large batch stochastic gradient descent.

centric areas such as manufacturing, health care and finance. Application of machine learning in such areas reduces the cost and increases the quality of life by reducing the level of human input required in such tasks. While inference of machine learning models reduces human intervention, building machine learning models still continues to be challenging and human input intensive. Building neural network topologies requires research communities spending days on tweaking machine learning pipelines. Building large datasets requires expert input at large scale, and solving of ethical and bureaucracy problems in data sensitive domains.

In this thesis we present new methods which aid in reducing resource requirement and removing human intervention while building new machine learning pipelines. We present new topologies for inference on video data and learn invariants using semi-supervised learning. We used generative means such as autoencoders and Markov decision processes as viable alternatives in absence of data while training with very small datasets. Our topologies used spatio-temporal convolutions and learned invariants, making neural networks more robust under real world variations.

We further aid in reducing expert intervention by developing new methods for architecture selection. This thesis is one of the first to apply data driven techniques to neural

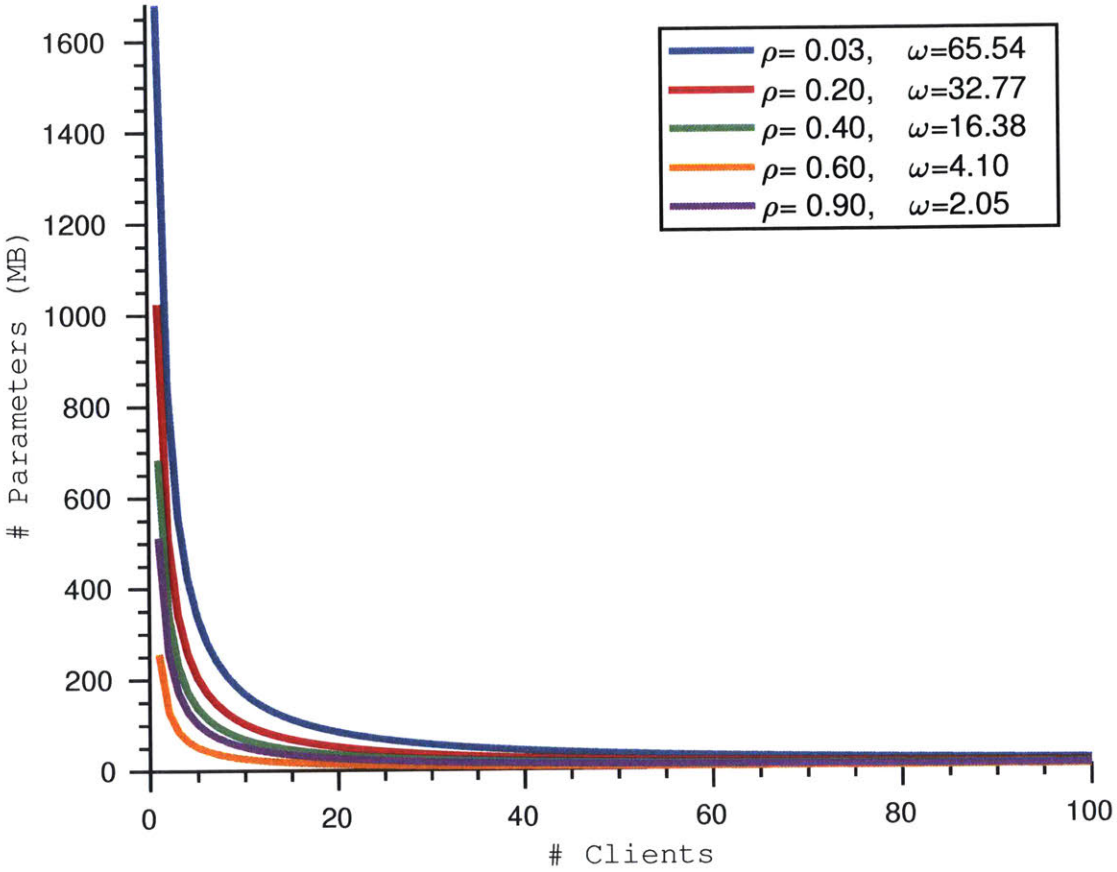


Figure 5-3: Iso curves for data transmission when using different values of ρ and ω. We compare data transmission requirements between our method and federated learning and plot iso curves for when both are equal. Hyperparameter ρ represents fraction of network on client side, and ω represents feature vector size in kilobytes. Our method beats federated learning for all points above the graph, demonstrating the scalability of our method.

network architecture selection. We use context free grammar driven Markov decision process to select and optimize topology selection while achieving competitive accuracies for image classification tasks. Our methods for state space pruning, and early stopping reduce the computational burden making architecture selection a more feasible alternative to expert level optimization of neural network layers.

Finally we build new layers which aid in data sharing in data sensitive applications. Our distributed learning system helps in training neural networks without sharing of actual raw data or labels. We demonstrate how this system can be used to learn from several agents in distributed fashion while achieving accuracies similar to copying the data at one centralized location. Our system aims at reducing the friction when sharing data between

different organizations which may not want to share their raw data because of intellectual property, ethical or bureaucratic issues.

Appendix

Table of network primitives

Mode	Alice (Request)	Bob (Response)	Meaning
Training request	mode: "training request" checksum: "<weight checksum>" nIter: <number iterations> client id: "<client identifier>"	response: "allowed" token: "<token>"	Bob is ready to train and alice has most recent weights. Alice must initiate training using token within next 20 seconds or token will expire!
Training request	mode: "training request" checksum: "<weight checksum>" nIter: <number iterations> client id: "<client identifier>"	response: "denied"	Bob is currently training with another alice or waiting for another alice to start training.
Training request	mode: "training request" checksum: "<weight checksum>" nIter: <number iterations> client id: "<client identifier>"	response: "refresh" client id: ["Alice$_k$", "xxx.xxx.xxx.xxx"]	Checksum is old, alice needs to refresh weights (most recent alice ip is provided for peer to peer case).
Tensor transmission	mode: "training" checksum: "<tensor checksum>" shape: <tensor dimensions> raw_data: <serialized tensor>	response: "success/-failure"	
Encrypted weight Upload (Centralized)	mode: "weight upload" checksum: "<weight checksum>" weights: <encrypted weight file> client id: "<client identifier>"	response: "successful"	Bob has registered checksum and stored Alicejweights. It has also snapshotted its weights.

Mode	Alice (Request)	Bob (Response)	Meaning
Encrypted weight Upload (Centralized)	mode: "weight upload" checksum: "<weight checksum>" weights: <encrypted weight file> client id: "<client identifier>"	response: "failed"	Bob was not training with this alice currently or lastly.
Encrypted weight request (Centralized)	mode: "weight request" client id: "<client identifier>"	weight: <encrypted weights file>	Bob allows for download of encrypted file.
Encrypted weight request (Centralized)	mode: "weight request" client id: "<client identifier>"	response: "denied" reason: "<string>"	Bob was not started in centralized mode, or never trained with an alice.
Snapshot request (Peer to peer)	mode: "snapshot" checksum: "<weight checksum>" client id: "<client identifier>"	response: "successful"	Bob has registered checksum. It has also snapshotted its weights to match Alice_j.
Snapshot request (Peer to peer)	mode: "snapshot" checksum: "<weight checksum>" client id: "<client identifier>"	response: "failed"	Bob was not training with Alice_j currently or lastly.
Encrypted weight request (Peer to peer)	[To Alice_k] mode: "weight request" client id: "<client identifier>"	weight: <encrypted weights file>	Alice provides most recent encrypted weights used in training with Bob.
Encrypted weight request (Peer to peer)	[To Alice_k] mode: "weight request" client id: "<client identifier>"	response: "denied"	This Alice never trained with bob.

Table 5.3: Message specification for communication between multiple parties in split neural network algorithm.

Top topologies selected by algorithm

In Tables 5.1 through 5.5, we present the top five model architectures selected with Q-learning for each dataset, along with their prediction error reported on the test set, and their total number of parameters.

Model Architecture	Test Error (%)	# Params (10^6)
[C(128,3,1), P(2,2), C(64,5,1), C(512,5,1), C(256,3,1), C(512,3,1), P(2,2), C(512,3,1), C(256,5,1), C(256,3,1), C(128,5,1), C(64,3,1), SM(10)]	2.24	9.81
[C(128,1,1), C(256,5,1), C(128,5,1), P(2,2), C(256,5,1), C(256,1,1), C(256,3,1), C(256,3,1), C(256,5,1), C(512,5,1), C(256,3,1), C(128,3,1), SM(10)]	2.28	10.38
[C(128,5,1), C(128,3,1), C(64,5,1), P(5,3), C(128,3,1), C(512,5,1), C(256,5,1), C(128,5,1), C(128,5,1), C(128,3,1), SM(10)]	2.32	6.83
[C(128,1,1), C(256,5,1), C(128,5,1), C(256,3,1), C(256,5,1), P(2,2), C(128,1,1), C(512,3,1), C(256,5,1), P(2,2), C(64,5,1), C(64,1,1), SM(10)]	2.35	6.99
[C(128,1,1), C(256,5,1), C(128,5,1), C(256,5,1), C(256,5,1), C(256,1,1), P(3,2), C(128,1,1), C(256,5,1), C(512,5,1), C(256,3,1), C(128,3,1), SM(10)]	2.36	10.05

Table 5.4: Top 5 model architectures: SVHN. Note that we do not report the *best* accuracy on test set from the above models in Tables 3.3 and 3.4 from the main text. This is because the model that achieved 2.28% on the test set performed the best on the validation set.

Model Architecture	Test Error (%)	# Params (10^6)
[C(64,1,1), C(256,3,1), P(2,2), C(512,3,1), C(256,1,1), P(5,3), C(256,3,1), C(512,3,1), FC(512), SM(10)]	0.35	5.59
[C(128,3,1), C(64,1,1), C(64,3,1), C(64,5,1), P(2,2), C(128,3,1), P(3,2), C(512,3,1), FC(512), FC(128), SM(10)]	0.38	7.43
[C(512,1,1), C(128,3,1), C(128,5,1), C(64,1,1), C(256,5,1), C(64,1,1), P(5,3), C(512,1,1), C(512,3,1), C(256,3,1), C(256,5,1), C(256,5,1), SM(10)]	0.40	8.28
[C(64,3,1), C(128,3,1), C(512,1,1), C(256,1,1), C(256,5,1), C(128,3,1), P(5,3), C(512,1,1), C(512,3,1), C(128,5,1), SM(10)]	0.41	6.27
[C(64,3,1), C(128,1,1), P(2,2), C(256,3,1), C(128,5,1), C(64,1,1), C(512,5,1), C(128,5,1), C(64,1,1), C(512,5,1), C(256,5,1), C(64,5,1), SM(10)]	0.43	8.10
[C(64,1,1), C(256,5,1), C(256,5,1), C(512,1,1), C(64,3,1), P(5,3), C(256,5,1), C(256,5,1), C(512,5,1), C(64,1,1), C(128,5,1), C(512,5,1), SM(10)]	0.44	9.67
[C(128,3,1), C(512,3,1), P(2,2), C(256,3,1), C(128,5,1), C(64,1,1), C(64,5,1), C(512,5,1), GAP(10), SM(10)]	0.44	3.52
[C(256,3,1), C(256,5,1), C(512,3,1), C(256,5,1), C(512,1,1), P(5,3), C(256,3,1), C(64,3,1), C(256,5,1), C(512,3,1), C(128,5,1), C(512,5,1), SM(10)]	0.46	12.42
[C(512,5,1), C(128,5,1), C(128,5,1), C(128,3,1), C(256,3,1), C(512,5,1), C(256,3,1), C(128,3,1), SM(10)]	0.55	7.25
[C(64,5,1), C(512,5,1), P(3,2), C(256,5,1), C(256,3,1), C(256,3,1), C(128,1,1), C(256,3,1), C(256,5,1), C(64,1,1), C(256,3,1), C(64,3,1), SM(10)]	0.56	7.55

Table 5.5: Top 10 model architectures: MNIST. We report the top 10 models for MNIST because we included all 10 in our final ensemble. Note that we do not report the *best* accuracy on test set from the above models in Tables 3.3 and 3.4 from the main text. This is because the model that achieved 0.44% on the test set performed the best on the validation set.

Bibliography

[1] Alan Turing, Richard Braithwaite, Geoffrey Jefferson, and Max Newman. Can automatic calculating machines be said to think?(1952). *B. Jack Copeland*, page 487, 1952.

[2] Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

[3] Joshua B Tenenbaum, Vin De Silva, and John C Langford. A global geometric framework for nonlinear dimensionality reduction. *science*, 290(5500):2319–2323, 2000.

[4] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[5] Frank Rosenblatt. *The perceptron, a perceiving and recognizing automaton Project Para.* Cornell Aeronautical Laboratory, 1957.

[6] V Vapnik. The nature of statistical learning theory, 1995.

[7] ML Minsky, SA Papert, and First Perceptrons. The mit press: Cambridge. *Mass.(Rev. Edition, 1988)*, 1969.

[8] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *arXiv preprint arXiv:1409.4842*, 2014.

[9] 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.

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

[11] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lars Wolf. Deepface: Closing the gap to human-level performance in face verification. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1701–1708. IEEE, 2014.

[12] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *arXiv preprint arXiv:1611.08097*, 2016.

[13] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. CNN features off-the-shelf: an astounding baseline for recognition. *CVPR*, 2014.

[14] Daniel Holden, Jun Saito, Taku Komura, and Thomas Joyce. Learning motion manifolds with convolutional autoencoders. In *SIGGRAPH Asia 2015 Technical Briefs*, page 18. ACM, 2015.

[15] Adam Coates, Andrej Karpathy, and Andrew Y Ng. Emergence of object-selective features in unsupervised feature learning. In *Advances in Neural Information Processing Systems*, pages 2681–2689, 2012.

[16] Miguel A Carreira-Perpinan and Geoffrey E Hinton. On contrastive divergence learning. In *Proceedings of International Workshop on Artificial Intelligence and Statistics*, pages 33–40, 2005.

[17] Andrej Karpathy, George Toderici, Sachin Shetty, Tommy Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1725–1732. IEEE, 2014.

[18] Li Fei-Fei, Rob Fergus, and Pietro Perona. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. *Computer vision and Image understanding*, 106(1):59–70, 2007.

[19] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[20] Otkrist Gupta, Dan Raviv, and Ramesh Raskar. Multi-velocity neural networks for facial expression recognition in videos. *IEEE Transactions of Affective Computing*, 2017.

[21] Abhimanyu Dubey, Otkrist Gupta, Pei Guo, Ramesh Raskar, Ryan Farell, and Nikhil Naik. Training with confusion for fine-grained visual classification. *Under Review Advances in Neural Information Processing Systems*, 2017.

[22] Otkrist Gupta, Bowen Baker, Nikhil Naik, and Ramesh Raskar. Practical neural network performance prediction for early stopping. *Under Review Advances in Neural Information Processing Systems*, 2017.

[23] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. *International Conference on Learning Representations*, 2017.

[24] Guy Satat, Mat Tancick, Otkrist Gupta, Barmak Heshmat, and Ramesh Raskar. Calibration free imaging through scattering media. *Optics Express*, 2017.

[25] Otkrist Gupta and Ramesh Raskar. Secure training of deep neural networks. *Patent Filed 18864T*, 2017.

[26] Otkrist Gupta, Dan Raviv, and Ramesh Raskar. Illumination invariants in deep video expression recognition. *The Journal of Pattern Recognition*, 2017.

[27] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. Learning spatiotemporal features with 3d convolutional networks. *arXiv preprint arXiv:1412.0767*, 2014.

[28] Zaenal Abidin and Agus Harjoko. A neural network based facial expression recognition using fisherface. *International Journal of Computer Applications*, 59(3):30–34, 2012.

[29] M Gargesha and P Kuchi. Facial expression recognition using artificial neural networks. *Artificial Neural Computer Systems*, pages 1–6, 2002.

[30] Heechul Jung, Sihaeng Lee, Junho Yim, Sunjeong Park, and Junmo Kim. Joint fine-tuning in deep neural networks for facial expression recognition. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2983–2991, 2015.

[31] Li Fei-Fei, Rob Fergus, and Pietro Perona. One-shot learning of object categories. *IEEE transactions on pattern analysis and machine intelligence*, 28(4):594–611, 2006.

[32] Geoffrey E Hinton and Richard S Zemel. Autoencoders, minimum description length, and helmholtz free energy. *Advances in neural information processing systems*, pages 3–3, 1994.

[33] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. *ICML*, 2008.

[34] Jonathan Masci, Ueli Meier, Dan Cireşan, and Jürgen Schmidhuber. Stacked convolutional auto-encoders for hierarchical feature extraction. In *International Conference on Artificial Neural Networks*, pages 52–59. Springer, 2011.

[35] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11(Dec):3371–3408, 2010.

[36] Jason Weston, Frédéric Ratle, Hossein Mobahi, and Ronan Collobert. Deep learning via semi-supervised embedding. In *Neural Networks: Tricks of the Trade*, pages 639–655. Springer, 2012.

[37] Diederik P Kingma, Shakir Mohamed, Danilo Jimenez Rezende, and Max Welling. Semi-supervised learning with deep generative models. In *Advances in Neural Information Processing Systems*, pages 3581–3589, 2014.

[38] Dong-Hyun Lee. Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks. In *Workshop on Challenges in Representation Learning, ICML*, volume 3, 2013.

[39] George Papandreou, Liang-Chieh Chen, Kevin Murphy, and Alan L Yuille. Weakly-and semi-supervised learning of a dcnn for semantic image segmentation. *arXiv:1502.02734*, 2015.

[40] Ping Liu, Shizhong Han, Zibo Meng, and Yan Tong. Facial expression recognition via a boosted deep belief network. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1805–1812. IEEE, 2014.

[41] Samira Ebrahimi Kahou, Christopher Pal, Xavier Bouthillier, Pierre Froumenty, Çaglar Gülçehre, Roland Memisevic, Pascal Vincent, Aaron Courville, Yoshua Bengio, and Raul Chandias Ferrari. Combining modality specific deep neural networks

for emotion recognition in video. In *Proceedings of the 15th ACM on International conference on multimodal interaction*, pages 543–550. ACM, 2013.

[42] Markus Oberweger, Gernot Riegler, Paul Wohlhart, and Vincent Lepetit. Efficiently creating 3d training data for fine hand pose estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4957–4965, 2016.

[43] Will Y Zou, Andrew Y Ng, Shenghuo Zhu, and Kai Yu. Deep learning of invariant features via simulated fixations in video. In *NIPS*, volume 3, page 6, 2012.

[44] Chris Frith. Role of facial expressions in social interactions. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 364(1535):3453–3458, 2009.

[45] Roddy Cowie, Ellen Douglas-Cowie, Nicolas Tsapatsoulis, George Votsis, Stefanos Kollias, Winfried Fellenz, and John G Taylor. Emotion recognition in human-computer interaction. *Signal Processing Magazine, IEEE*, 18(1):32–80, 2001.

[46] Zhanpeng Zhang, Ping Luo, Chen-Change Loy, and Xiaoou Tang. Learning social relation traits from face images. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3631–3639, 2015.

[47] T Klein and W Picard. Computer response to user frustration. *MIT Media Laboratory Vision and Modelling Group Technical Reports, TR 480*, 1999.

[48] Eva Cerezo, Sandra Baldassarri, and Francisco Seron. Interactive agents for multimodal emotional user interaction. *Multi Conferences on Computer Science and Information Systems*, pages 35–42, 2007.

[49] Elisabeth André, Martin Klesen, Patrick Gebhard, Steve Allen, and Thomas Rist. Exploiting models of personality and emotions to control the behavior of animated interactive agents. In *Workshop on Achieving Human-Like Behavior in Interactive Animated Agents*, pages 3–7, 2000.

[50] Mengyi Liu, Shaoxin Li, Shiguang Shan, Ruiping Wang, and Xilin Chen. Deeply learning deformable facial action parts model for dynamic expression analysis. In *Computer Vision–ACCV 2014*, pages 143–157. Springer, 2014.

[51] Frédéric Pighin, Jamie Hecker, Dani Lischinski, Richard Szeliski, and David H Salesin. Synthesizing realistic facial expressions from photographs. In *ACM SIGGRAPH 2006 Courses*, page 19. ACM, 2006.

[52] Ziheng Wang, Shangfei Wang, and Qiang Ji. Capturing complex spatio-temporal relations among facial muscles for facial expression recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3422–3429, 2013.

[53] Justine Cassell, Catherine Pelachaud, Norman Badler, Mark Steedman, Brett Achorn, Tripp Becket, Brett Douville, Scott Prevost, and Matthew Stone. Animated conversation: rule-based generation of facial expression, gesture & spoken intonation for multiple conversational agents. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 413–420. ACM, 1994.

[54] Keith Waters. A muscle model for animation three-dimensional facial expression. In *Proceedings of the annual conference on Computer graphics and interactive techniques*, volume 21, pages 17–24. ACM, 1987.

[55] Hyewon Pyun, Yejin Kim, Wonseok Chae, Hyung Woo Kang, and Sung Yong Shin. An example-based approach for facial expression cloning. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 167–176. Eurographics Association, 2003.

[56] Irene Kotsia and Ioannis Pitas. Facial expression recognition in image sequences using geometric deformation features and support vector machines. *Transactions on Image Processing*, 16(1):172–187, 2007.

[57] Abhinav Dhall, Akshay Asthana, Roland Goecke, and Tom Gedeon. Emotion recognition using PHOG and LPQ features. In *International Conference on Automatic Face & Gesture Recognition*, pages 878–883. IEEE, 2011.

[58] Shaoqing Ren, Xudong Cao, Yichen Wei, and Jian Sun. Face alignment at 3000 fps via regressing local binary features. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1685–1692. IEEE, 2014.

[59] Hui Chen, Jiangdong Li, Fengjun Zhang, Yang Li, and Hongan Wang. 3d model-based continuous emotion recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1836–1845, 2015.

[60] Robert Walecki, Ognjen Rudovic, Vladimir Pavlovic, and Maja Pantic. Variable-state latent conditional random fields for facial expression recognition and action unit detection. In *Automatic Face and Gesture Recognition (FG), 2015 11th IEEE International Conference and Workshops on*, volume 1, pages 1–8. IEEE, 2015.

[61] Mengyi Liu, Shiguang Shan, Ruiping Wang, and Xilin Chen. Learning expressionlets on spatio-temporal manifold for dynamic facial expression recognition. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1749–1756. IEEE, 2014.

[62] Mengyi Liu, Ruiping Wang, Shaoxin Li, Shiguang Shan, Zhiwu Huang, and Xilin Chen. Combining multiple kernel methods on riemannian manifold for emotion recognition in the wild. In *Proceedings of the 16th International Conference on Multimodal Interaction*, pages 494–501. ACM, 2014.

[63] Kaili Zhao, Wen-Sheng Chu, Fernando De la Torre, Jeffrey F Cohn, and Honggang Zhang. Joint patch and multi-label learning for facial action unit detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2207–2216, 2015.

[64] Thibaud Senechal, Daniel McDuff, and Rana Kaliouby. Facial action unit detection using active learning and an efficient non-linear kernel approximation. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 10–18, 2015.

[65] Tin Lay Nwe, Say Wei Foo, and Liyanage C De Silva. Speech emotion recognition using Hidden Markov Models. *Speech communication*, 41(4):603–623, 2003.

[66] Björn Schuller, Gerhard Rigoll, and Manfred Lang. Speech emotion recognition combining acoustic features and linguistic information in a hybrid support vector machine-belief network architecture. In *International Conference on Acoustics, Speech, and Signal Processing*, volume 1, pages I–577. IEEE, 2004.

[67] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[68] Paul Viola and Michael J Jones. Robust real-time face detection. *International Journal of Computer Vision*, 57(2):137–154, 2004.

[69] Hsieh S Hou and H Andrews. Cubic splines for image interpolation and digital filtering. *Transactions on Acoustics, Speech and Signal Processing*, 26(6):508–517, 1978.

[70] Akshay Asthana, Stefanos Zafeiriou, Shiyang Cheng, and Maja Pantic. Incremental face alignment in the wild. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1859–1866. IEEE, 2014.

[71] Patrick Lucey, Jeffrey F Cohn, Takeo Kanade, Jason Saragih, Zara Ambadar, and Iain Matthews. The extended cohn-kanade dataset (ck+): A complete dataset for action unit and emotion-specified expression. In *Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 94–101. IEEE, 2010.

[72] Maja Pantic, Michel Valstar, Ron Rademaker, and Ludo Maat. Web-based database for facial expression analysis. In *International Conference on Multimedia and Expo*, pages 5–pp. IEEE, 2005.

[73] Michel Valstar and Maja Pantic. Induced disgust, happiness and surprise: an addition to the mmi facial expression database. In *Workshop on EMOTION: Corpora for Research on Emotion and Affect*, page 65, 2010.

[74] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[75] Sources. Visual information processing and learning. [Online; accessed 10-July-2015].

[76] 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.

[77] Yunhe Wang, Chang Xu, Shan You, Dacheng Tao, and Chao Xu. CNNpack: packing convolutional neural networks in the frequency domain. In *Advances in Neural Information Processing Systems*, pages 253–261, 2016.

[78] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In *Advances in neural information processing systems*, pages 4107–4115, 2016.

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

[80] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[81] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (ELUs). *arXiv preprint arXiv:1511.07289*, 2015.

[82] Chen-Yu Lee, Patrick W Gallagher, and Zhuowen Tu. Generalizing pooling functions in convolutional neural networks: Mixed, gated, and tree. *International Conference on Artificial Intelligence and Statistics*, 2016.

[83] Kenneth O Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, 2002.

[84] James Bergstra, Daniel Yamins, and David D Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. *ICML (1)*, 28:115–123, 2013.

[85] J David Schaffer, Darrell Whitley, and Larry J Eshelman. Combinations of genetic algorithms and neural networks: A survey of the state of the art. *International Workshop on Combinations of Genetic Algorithms and Neural Networks*, pages 1–37, 1992.

[86] Shreyas Saxena and Jakob Verbeek. Convolutional neural fabrics. In *Advances in Neural Information Processing Systems 29*, pages 4053–4061. 2016.

[87] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando de Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016.

[88] Nicolas Pinto, David Doukhan, James J DiCarlo, and David D Cox. A high-throughput screening approach to discovering good forms of biologically inspired visual representation. *PLoS Computational Biology*, 5(11):e1000579, 2009.

[89] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *JMLR*, 13(Feb):281–305, 2012.

[90] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International Conference on Learning and Intelligent Optimization*, pages 507–523. Springer, 2011.

[91] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. *NIPS*, pages 2951–2959, 2012.

[92] Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Mostofa Patwary, Mr Prabhat, and Ryan Adams. Scalable bayesian optimization using deep neural networks. In *International Conference on Machine Learning*, pages 2171–2180, 2015.

[93] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *International Conference on Learning Representations*, 2017.

[94] Dimitri P Bertsekas. *Convex optimization algorithms*. Athena Scientific Belmont, 2015.

[95] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[96] Long-Ji Lin. Reinforcement learning for robots using neural networks. Technical report, DTIC Document, 1993.

[97] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[98] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. *AISTATS*, 9:249–256, 2010.

[99] Ian J Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron C Courville, and Yoshua Bengio. Maxout networks. *ICML (3)*, 28:1319–1327, 2013.

[100] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.

[101] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

[102] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550*, 2014.

[103] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

[104] Pierre Sermanet, Soumith Chintala, and Yann LeCun. Convolutional neural networks applied to house numbers digit classification. *ICPR*, pages 3288–3291, 2012.

[105] Pierre Sermanet, Koray Kavukcuoglu, Soumith Chintala, and Yann LeCun. Pedestrian detection with unsupervised multi-stage feature learning. *CVPR*, pages 3626–3633, 2013.

[106] James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. *NIPS*, pages 2546–2554, 2011.

[107] Phillip Verbancsics and Josh Harguess. Generative neuroevolution for deep learning. *arXiv preprint arXiv:1312.5355*, 2013.

[108] Li Wan, Matthew Zeiler, Sixin Zhang, Yann L Cun, and Rob Fergus. Regularization of neural networks using dropconnect. *ICML*, pages 1058–1066, 2013.

[109] Chen-Yu Lee, Saining Xie, Patrick Gallagher, Zhengyou Zhang, and Zhuowen Tu. Deeply-supervised nets. *AISTATS*, 2(3):6, 2015.

[110] Ming Liang and Xiaolin Hu. Recurrent convolutional neural network for object recognition. *CVPR*, pages 3367–3375, 2015.

[111] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European Conference on Computer Vision*, pages 630–645. Springer, 2016.

[112] Shai Avidan and Moshe Butman. Blind vision. *European Conference on Computer Vision*, pages 1–13, 2006.

[113] Alexey Dosovitskiy and Thomas Brox. Inverting visual representations with convolutional networks. *arXiv preprint arXiv:1506.02753*, 2015.

[114] Deepak Pathak, Philipp Krahenbuhl, Jeff Donahue, Trevor Darrell, and Alexei A Efros. Context encoders: Feature learning by inpainting. *arXiv preprint arXiv:1604.07379*, 2016.

[115] Shiguo Lian, Jinsheng Sun, and Zhiquan Wang. Secure hash function based on neural network. *Neurocomputing*, 69(16):2346–2350, 2006.

[116] Shiguo Lian, Guanrong Chen, Albert Cheung, and Zhiquan Wang. A chaotic-neural-network-based encryption algorithm for jpeg2000 encoded images. *International Symposium on Neural Networks*, pages 627–632, 2004.

[117] Khalil Shihab. A backpropagation neural network for computer network security. *Journal of Computer Science*, 2(9):710–715, 2006.

[118] Lokesh Jain. Preserving security in routing in mobile ad-hoc environment through non-linear dimension reduction technique. *International Journal of Advanced Research in Computer Science*, 2(6), 2011.

[119] Qingchen Zhang, Laurence T Yang, and Zhikui Chen. Privacy preserving deep computation model on cloud for big data feature learning. *IEEE Transactions on Computers*, 65(5):1351–1362, 2016.

[120] Ankur Bansal, Tingting Chen, and Sheng Zhong. Privacy preserving back-propagation neural network learning over arbitrarily partitioned data. *Neural Computing and Applications*, 20(1):143–150, 2011.

[121] Jimmy Secretan, Michael Georgiopoulos, and Jose Castro. A privacy preserving probabilistic neural network for horizontally partitioned databases. *2007 International Joint Conference on Neural Networks*, pages 1554–1559, 2007.

[122] Weiru Wang, Chi-Man Vong, Yilong Yang, and Pak-Kin Wong. Encrypted image classification based on multilayer extreme learning machine. *Multidimensional Systems and Signal Processing*, pages 1–15, 2016.

[123] Mauro Barni, Claudio Orlandi, and Alessandro Piva. A privacy-preserving protocol for neural-network-based computation. *Proceedings of the 8th workshop on Multimedia and security*, pages 146–151, 2006.

[124] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game. *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 218–229, 1987.

[125] Andrew Chi-Chih Yao. How to generate and exchange secrets. *Foundations of Computer Science, 1986., 27th Annual Symposium on*, pages 162–167, 1986.

[126] Yuan Zhang and Sheng Zhong. A privacy-preserving algorithm for distributed training of neural network ensembles. *Neural Computing and Applications*, 22(1):269–282, 2013.

[127] Keke Chen and Ling Liu. A random rotation perturbation approach to privacy preserving data classification. 2005.

[128] Claudio Orlandi, Alessandro Piva, and Mauro Barni. Oblivious neural network computing via homomorphic encryption. *EURASIP Journal on Information Security*, 2007:18, 2007.

[129] Yan-Cheng Chang and Chi-Jen Lu. Oblivious polynomial evaluation and oblivious neural learning. *International Conference on the Theory and Application of Cryptology and Information Security*, pages 369–384, 2001.

[130] 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.

[131] Jianmin Chen, Rajat Monga, Samy Bengio, and Rafal Jozefowicz. Revisiting distributed synchronous sgd. *arXiv preprint arXiv:1604.00981*, 2016.

[132] Martin Zinkevich, Markus Weimer, Lihong Li, and Alex J Smola. Parallelized stochastic gradient descent. In *Advances in neural information processing systems*, pages 2595–2603, 2010.

[133] Yossi Arjevani and Ohad Shamir. Communication complexity of distributed convex learning and optimization. In *Advances in neural information processing systems*, pages 1756–1764, 2015.

[134] Yuchen Zhang, Martin J Wainwright, and John C Duchi. Communication-efficient algorithms for statistical optimization. In *Advances in Neural Information Processing Systems*, pages 1502–1510, 2012.

[135] Hoo-Chang Shin, Matthew R Orton, David J Collins, Simon J Doran, and Martin O Leach. Stacked autoencoders for unsupervised feature learning and multiple organ detection in a pilot study using 4d patient data. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1930–1943, 2013.

[136] Pablo M Granitto, Pablo F Verdes, and H Alejandro Ceccatto. Neural network ensembles: evaluation of aggregation algorithms. *Artificial Intelligence*, 163(2):139–162, 2005.

[137] Ian J Goodfellow, Oriol Vinyals, and Andrew M Saxe. Qualitatively characterizing neural network optimization problems. *arXiv preprint arXiv:1412.6544*, 2014.

[138] Nicolas Papernot, Martín Abadi, Úlfar Erlingsson, Ian Goodfellow, and Kunal Talwar. Semi-supervised knowledge transfer for deep learning from private training data. *arXiv preprint arXiv:1610.05755*, 2016.

[139] Ming-Zher Poh, Daniel J McDuff, and Rosalind W Picard. Non-contact, automated cardiac pulse measurements using video imaging and blind source separation. *Optics Express*, 18(10):10762–10774, 2010.

[140] Giovanni Cennini, Jeremie Arguel, Kaan Akşit, and Arno van Leest. Heart rate monitoring via remote photoplethysmography with motion artifacts reduction. *Optics Express*, 18(5):4867–4875, 2010.

[141] Ganesh Balakrishnan, Frederic Durand, and John Guttag. Detecting pulse from head motions in video. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3430–3437. IEEE, 2013.

[142] Richard D Lane, Kateri McRae, Eric M Reiman, Kewei Chen, Geoffrey L Ahern, and Julian F Thayer. Neural correlates of heart rate variability during emotion. *Neuroimage*, 44(1):213–222, 2009.