Exploring Complexity Reduction in Deep Learning

Sourya Dey

PhD Candidate, University of Southern California
Advisors: Peter A. Beerel and Keith M. Chugg

February 18th, 2020

USC Viterbi
School of Engineering
Ming Hsieh Department of Electrical and Computer Engineering

Outline

Introduction

Pre-Defined Sparsity

Reduce complexity of neural networks with minimal performance degradation

Deep-n-Cheap

Automated search framework to explore performance-complexity tradeoffs in CNN design

Dataset Engineering

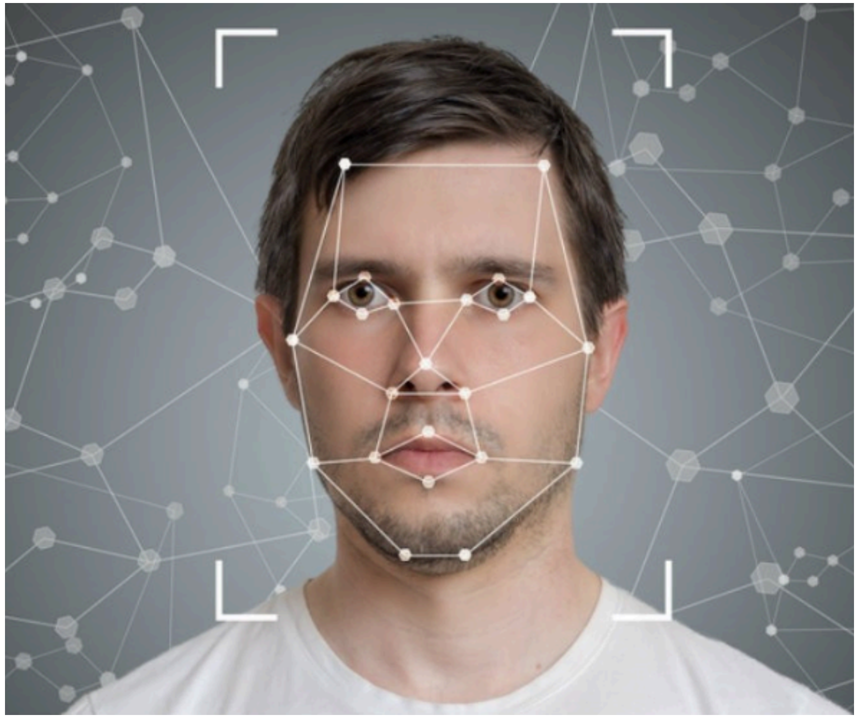
A family of synthetic datasets of customizable difficulty for ML classification problems

Introduction

Overview

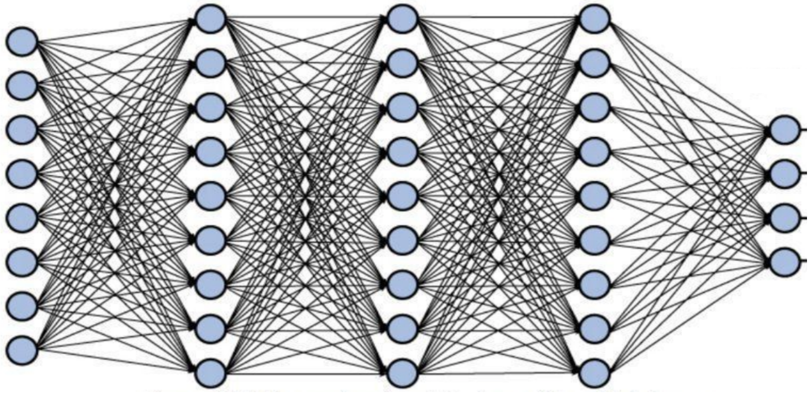
Neural networks (NNs) are key machine learning technologies

- Artificial intelligence
- Self-driving cars
- Speech recognition
- Face ID
- and more smart stuff …



Motivation behind complexity reduction

Modern neural networks suffer from parameter explosion



Fully connected **(FC)** Multilayer Perceptron **(MLP)**

Training can take weeks on CPU

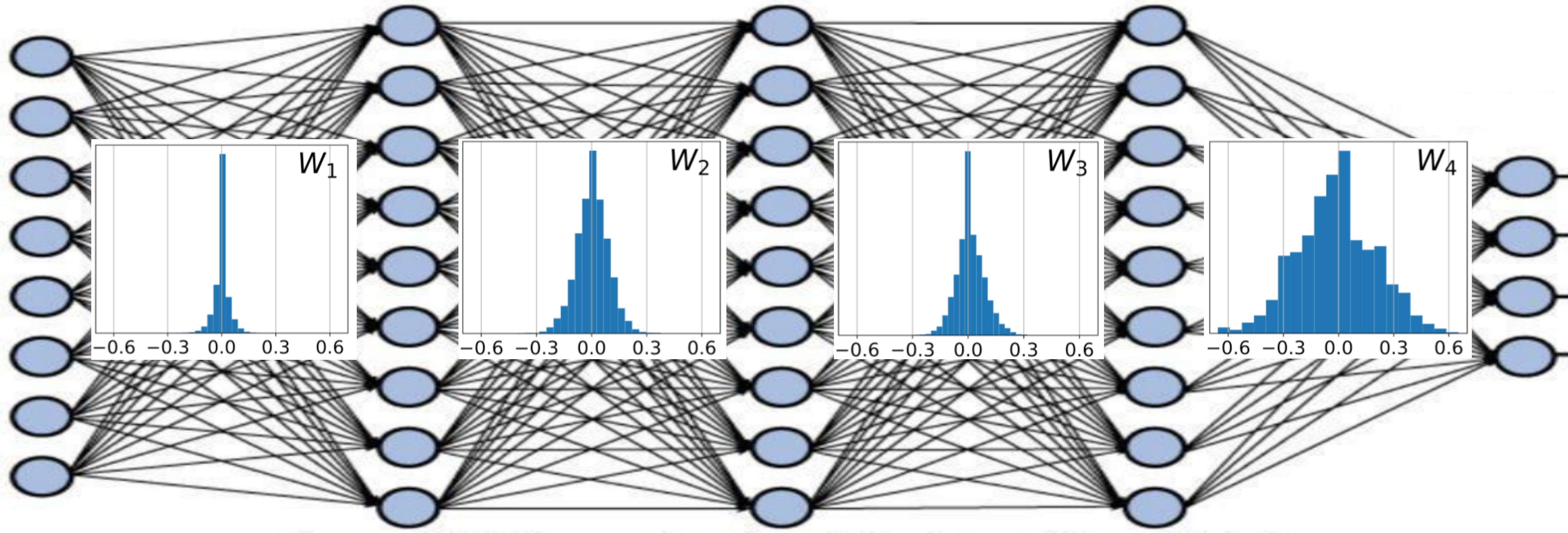
Cloud GPU resources are expensive

Typical deep **CNN**

Pre-Defined Sparsity

Reduce complexity of neural networks with minimal performance degradation

Motivation behind pre-defined sparsity



In a FC network, most weights are very small in magnitude after training

Pre-defined Sparsity

Pre-define a sparse connection pattern **prior to training**

Use this sparse network for both training and inference

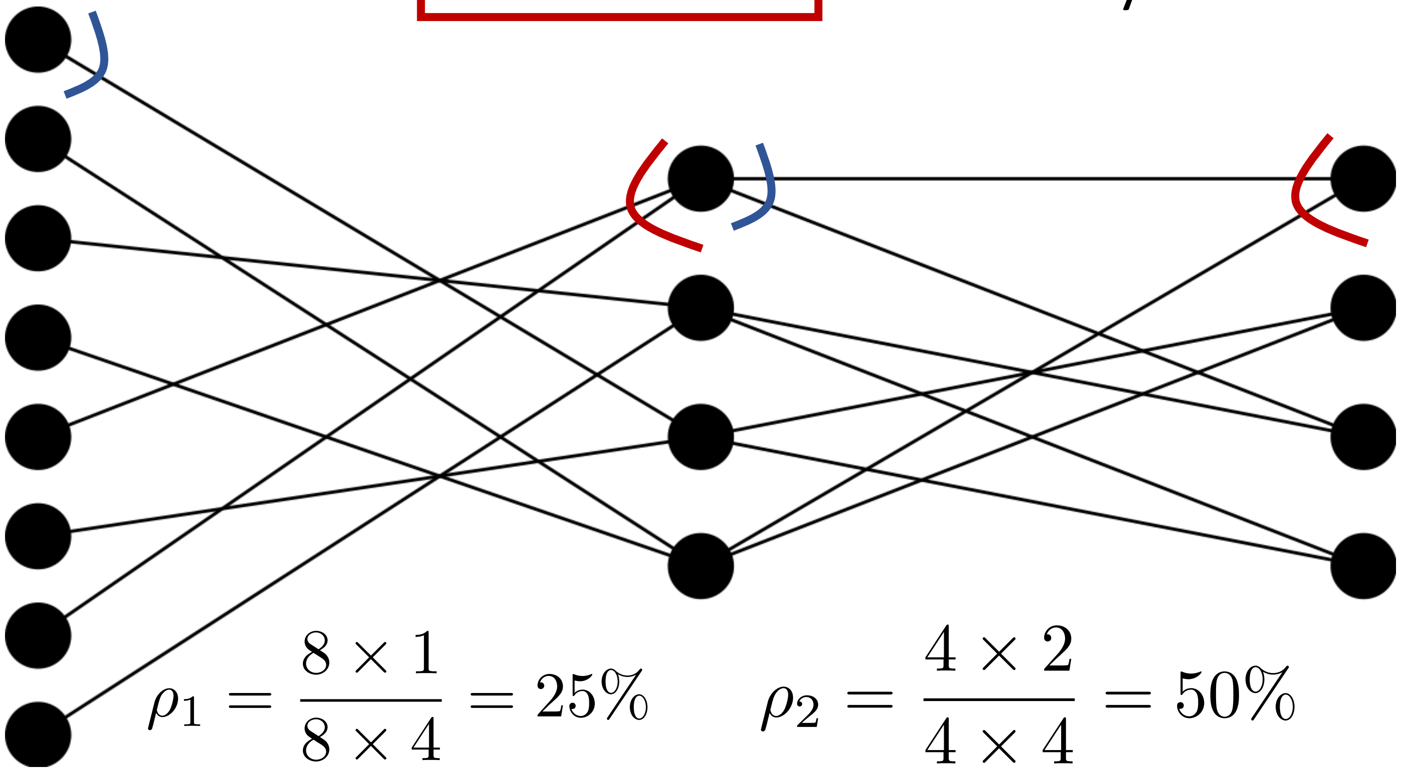
Reduced training *and* inference complexity

$$\boldsymbol{N}_{\text{net}} = (8,4,4)$$

$$\boxed{\boldsymbol{d}_{\text{net}}^{\text{out}} = (1,2)}$$

$$\boxed{\boldsymbol{d}_{\text{net}}^{\text{in}} = (2,2)}$$

Structured Constraints:
Fixed in-, out-degrees for every node

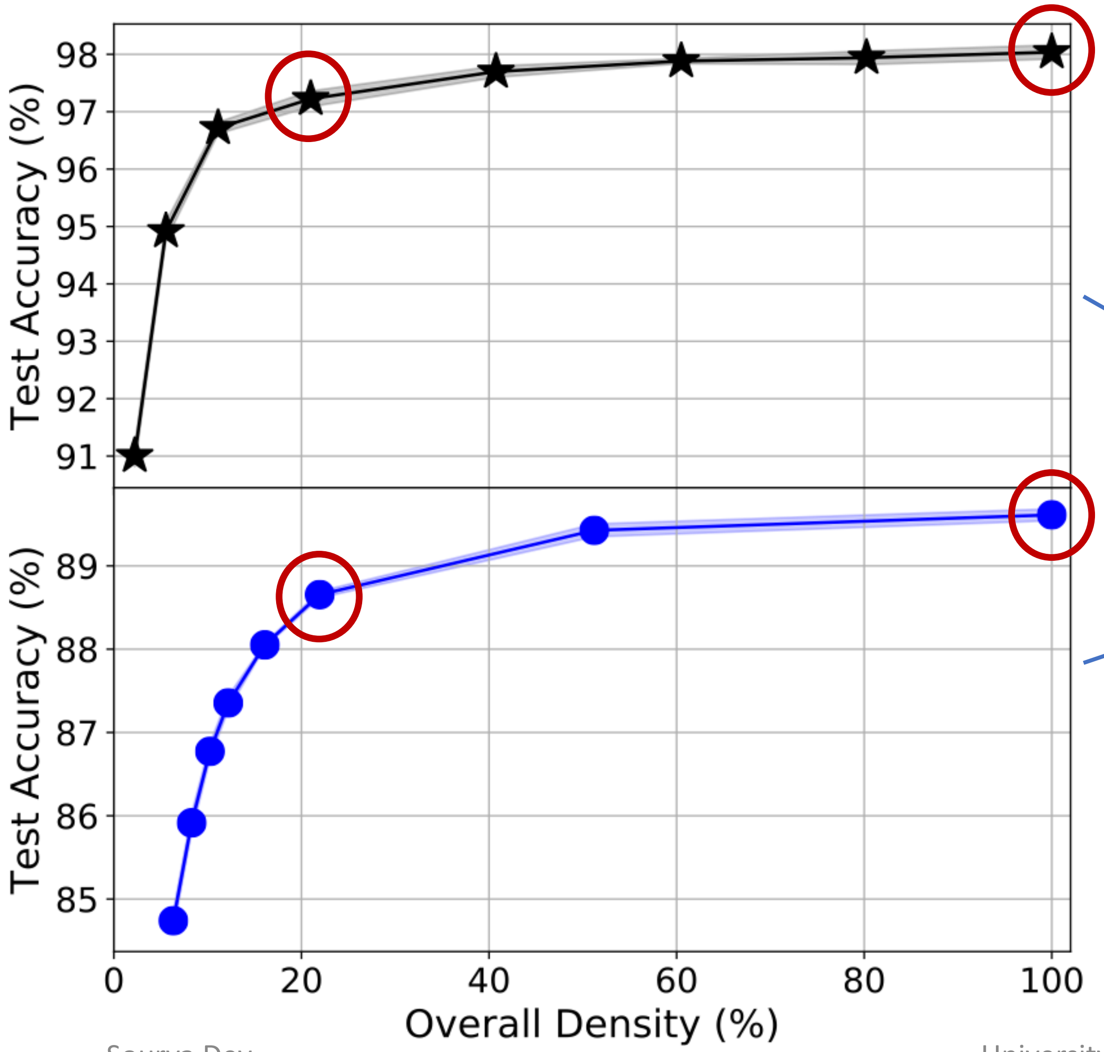


$$\rho_1 = \frac{8\times 1}{8\times 4} = 25\% \qquad \rho_2 = \frac{4\times 2}{4\times 4} = 50\%$$

$$\rho_{\text{net}} = \frac{8+8}{32+16} = 33\%$$

Overall Density compared to FC

Pre-defined sparsity performance on MLPs



Starting with only 20% of parameters reduces test accuracy by just 1%

MNIST handwritten digits

Reuters news articles

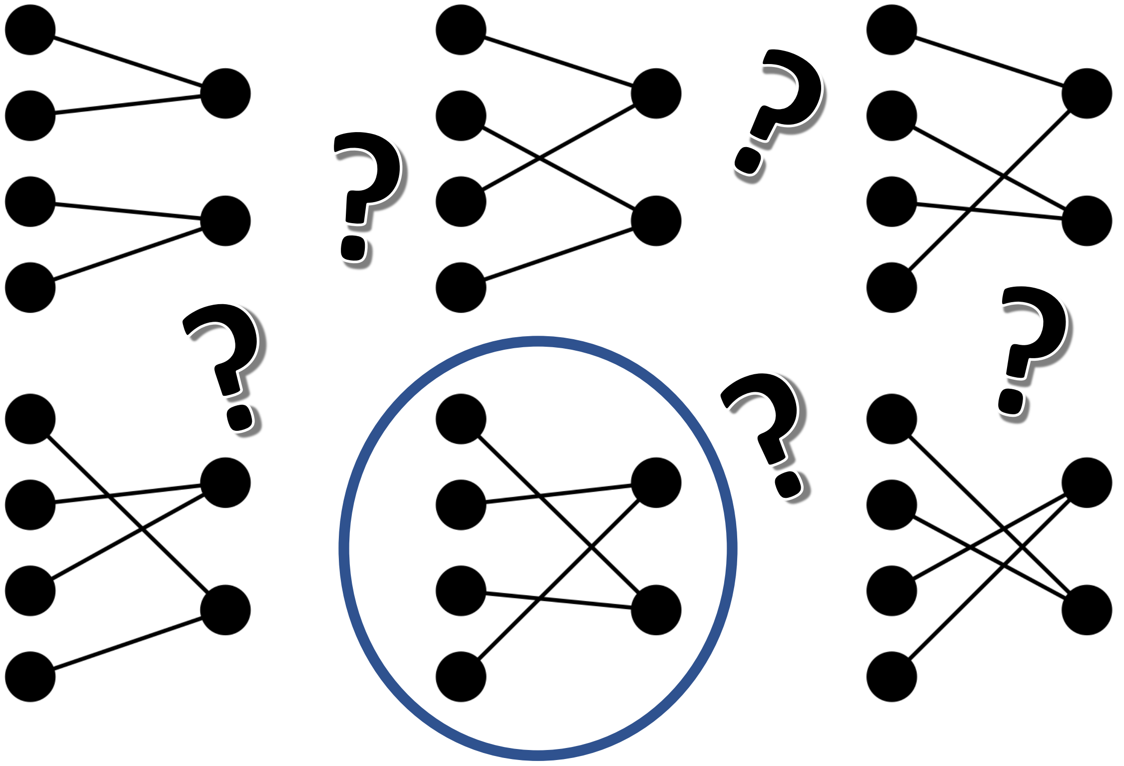
TIMIT phonemes

CIFAR images

Morse symbols

Designing pre-defined sparse networks

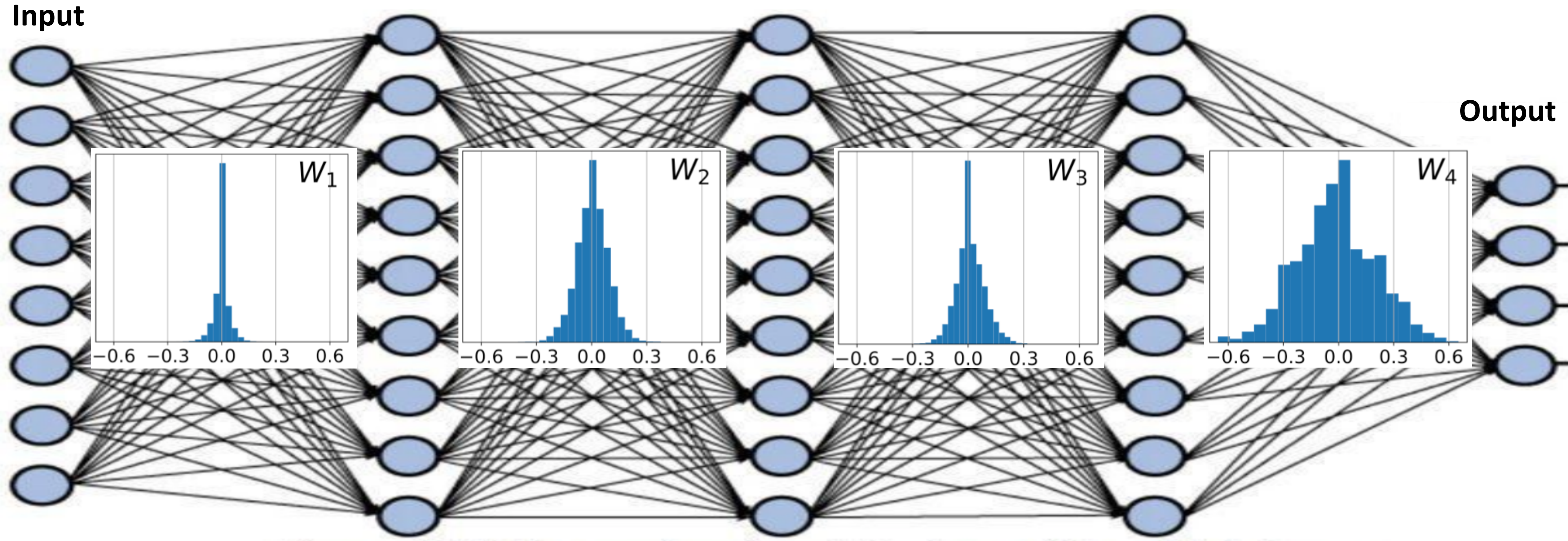
*A pre-defined sparse connection pattern is a **hyperparameter** to be set prior to training*



Find trends and guidelines to optimize pre-defined sparse patterns

S. Dey, K. Huang, P. A. Beerel and K. M. Chugg, "Pre-Defined Sparse Neural Networks with Hardware Acceleration," in *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 2, pp. 332-345, June 2019.

1. Individual junction densities



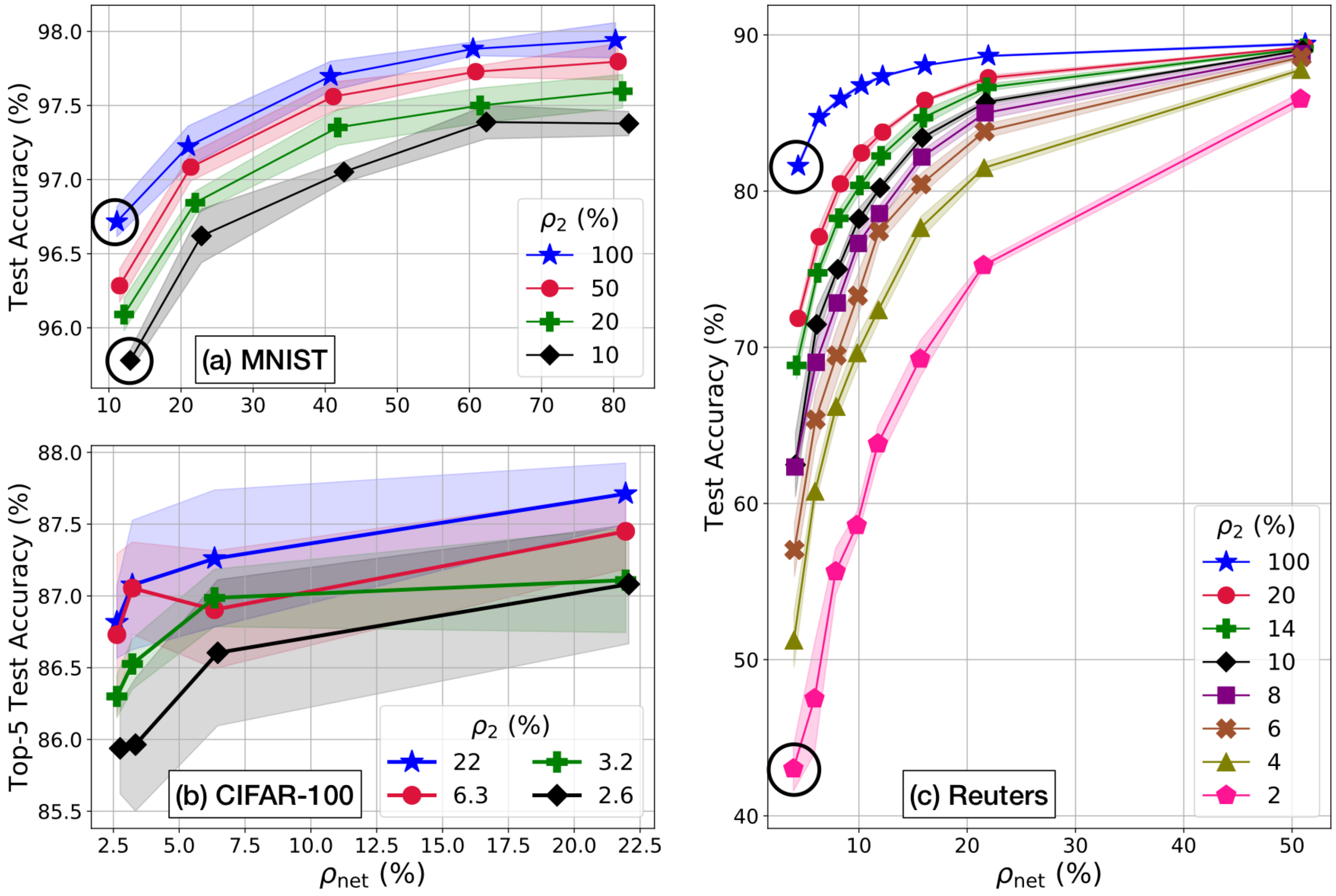
Latter junctions (closer to the output) learn higher-order, more complicated representations => They need to be denser

Results

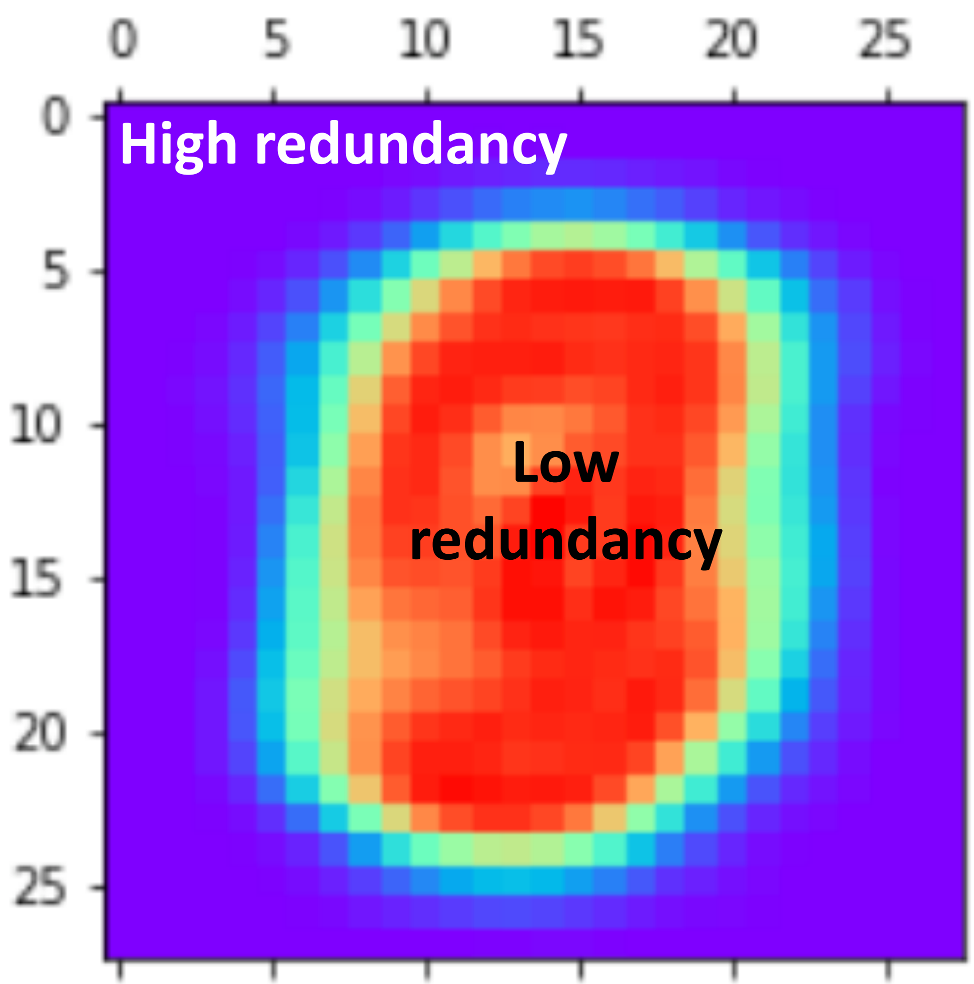
Each curve keeps ρ_2 fixed and varies ρ_{net} by varying ρ_1

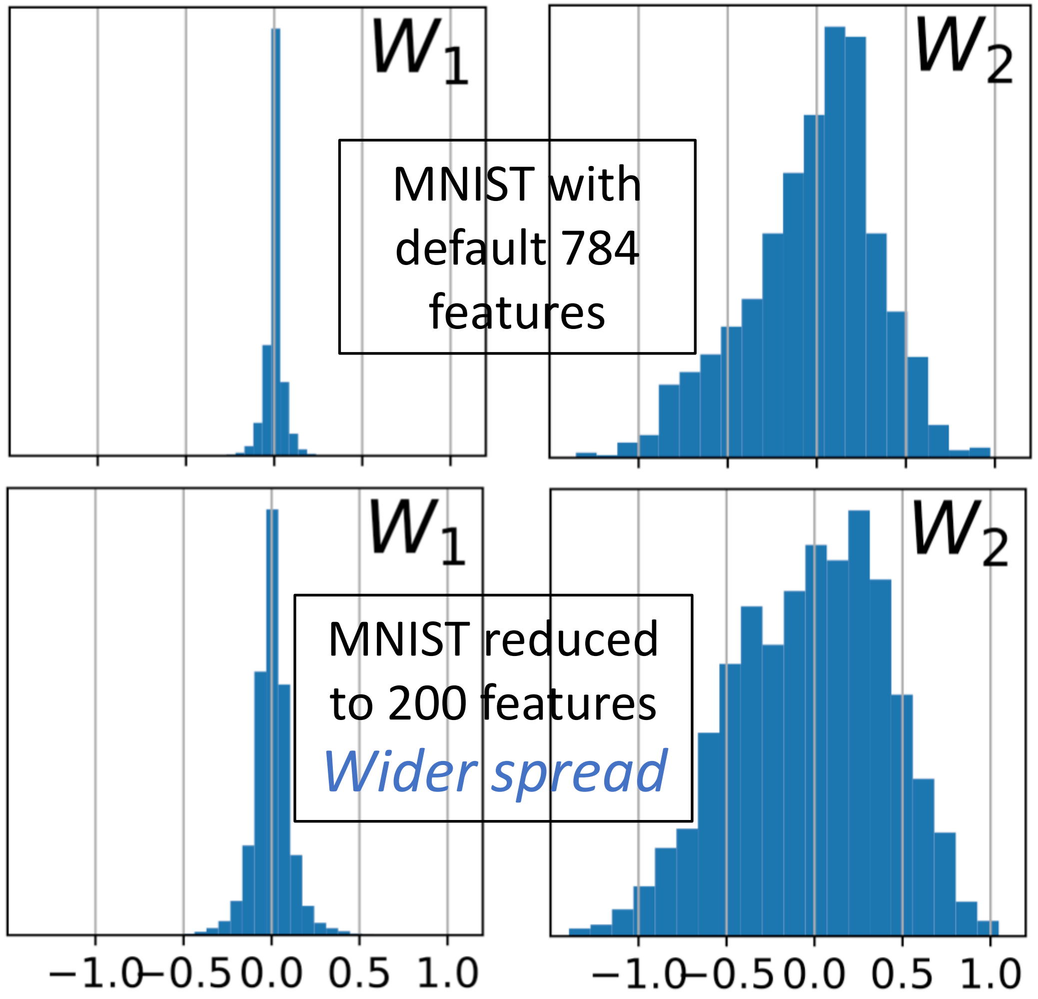
For the same ρ_{net}, $\rho_2 > \rho_1$ improves performance

Mostly similar trends observed for deeper networks



2. Dataset redundancy



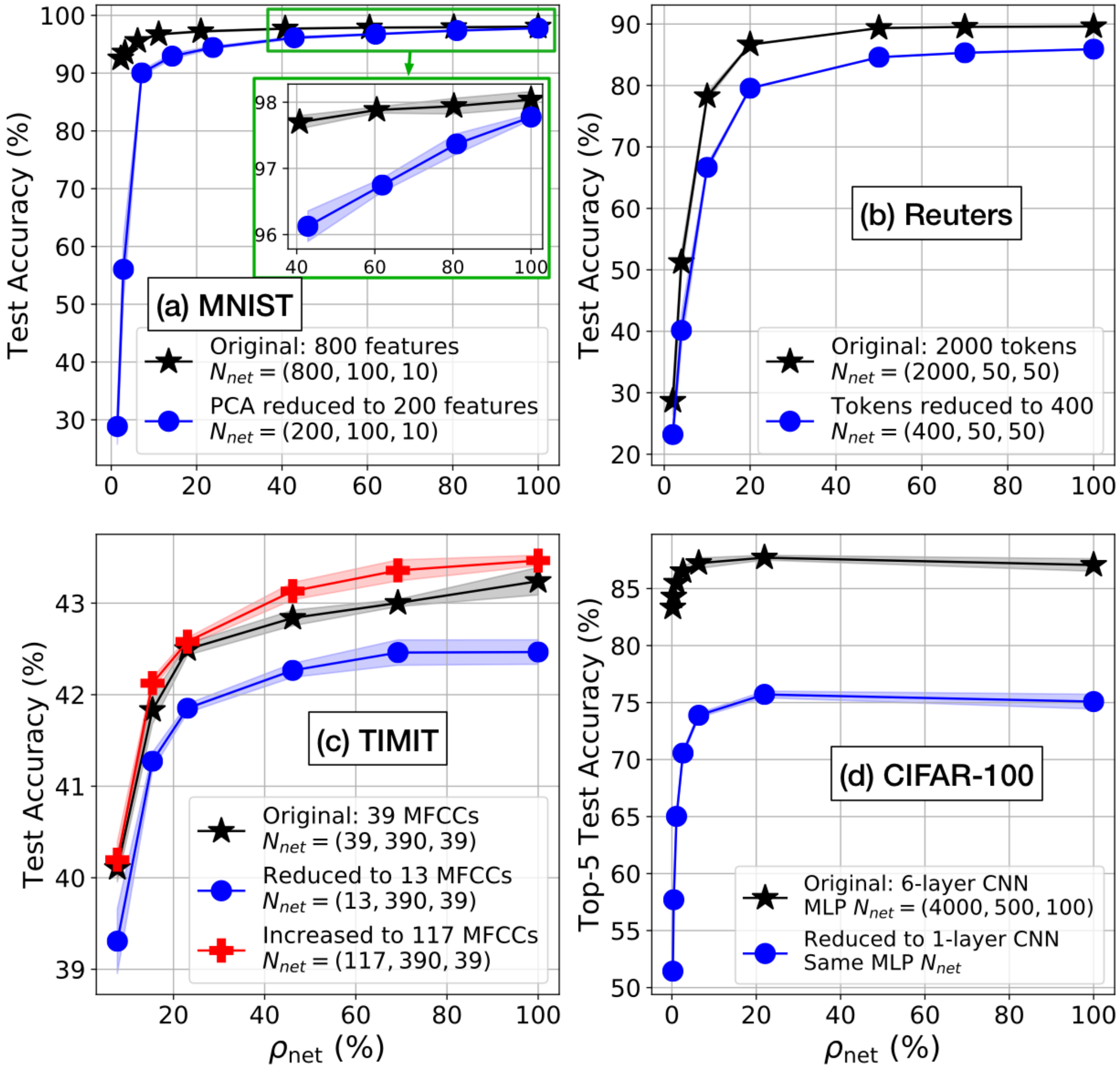


Less redundancy => Less sparsification possible

Results

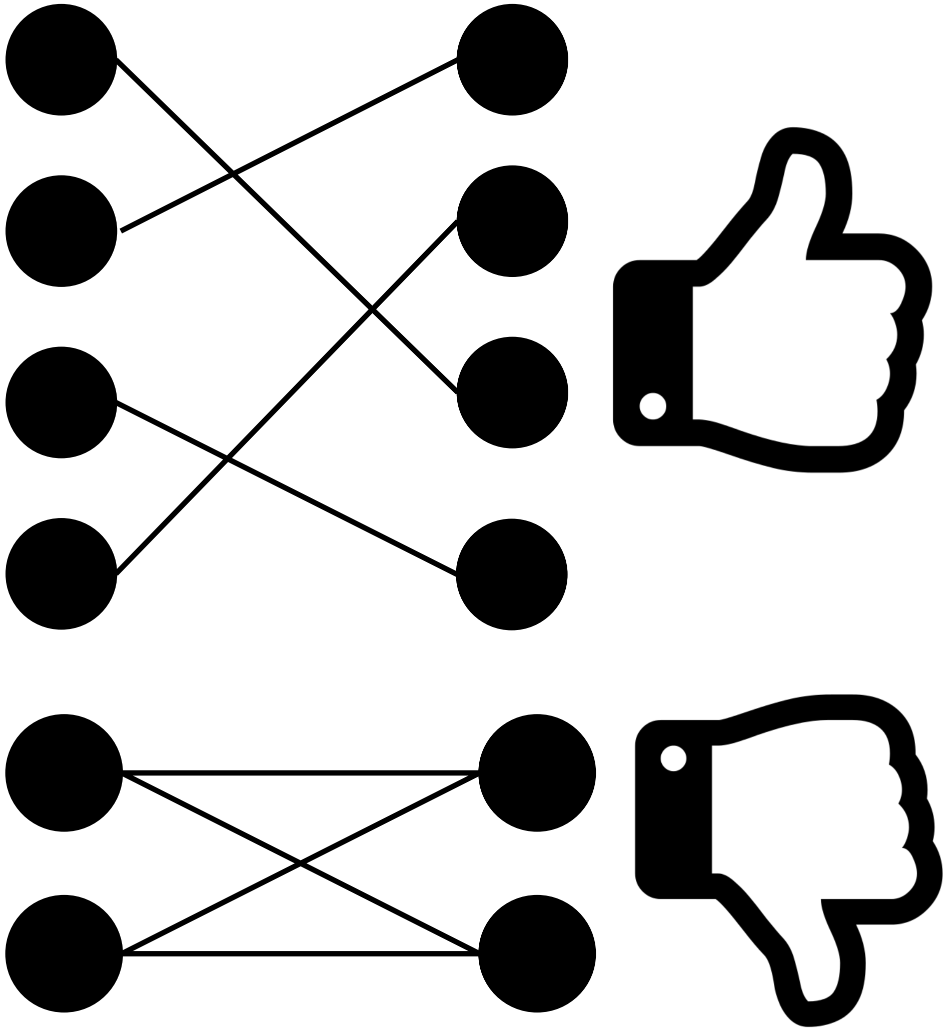
Reducing redundancy leads to increased performance degradation on sparsification

Pre-defined sparse design is problem-dependent



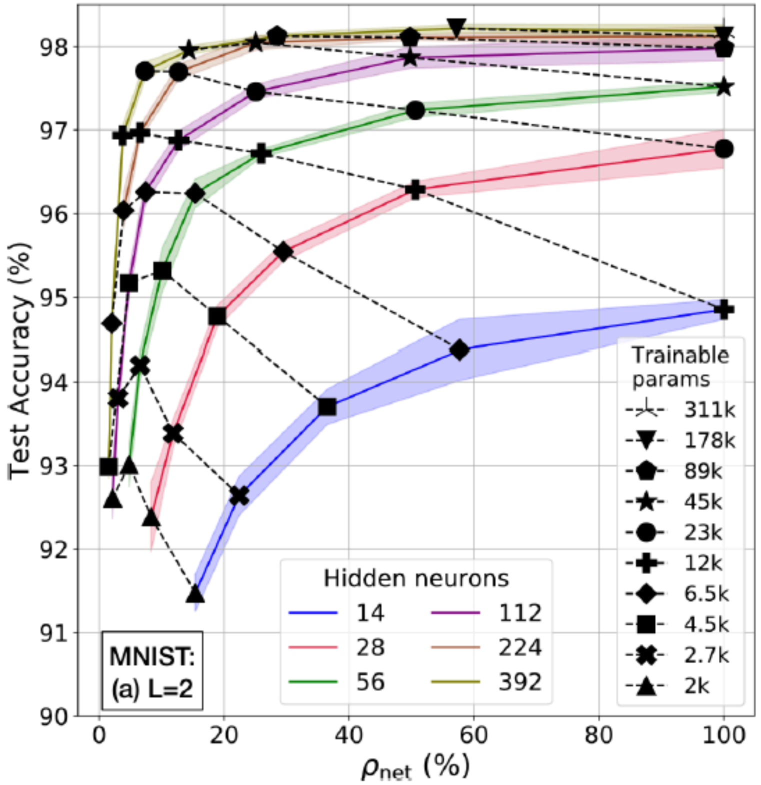
3. ‘Large sparse’ vs ‘small dense’ networks

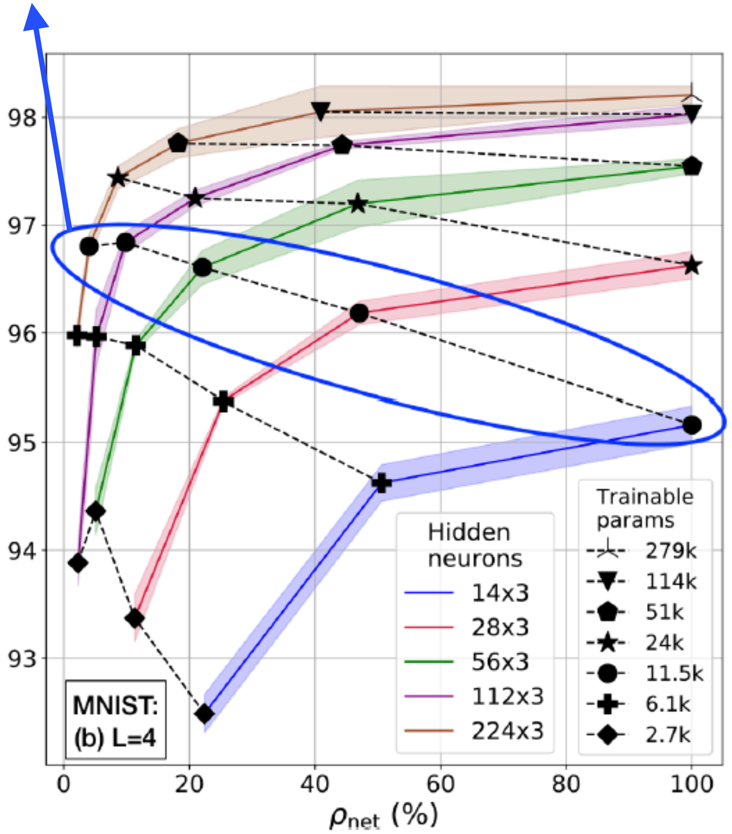
A sparser network with more hidden nodes will outperform a denser network with less hidden nodes, when both have same number of weights

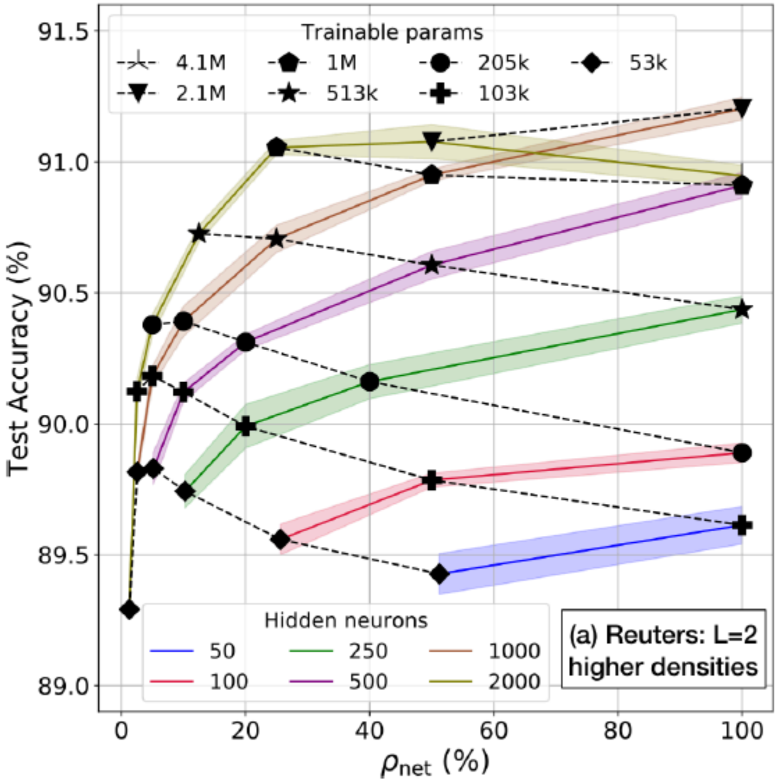


Results

Networks with same number of parameters go from bad to good as #nodes in hidden layers is increased







4. Regularization

$$C(\boldsymbol{w}) = C_0(\boldsymbol{w}) + \lambda \left\|\boldsymbol{w}\right\|_2^2$$

Regularized cost

Original unregularized cost (like cross-entropy)

Regularization term

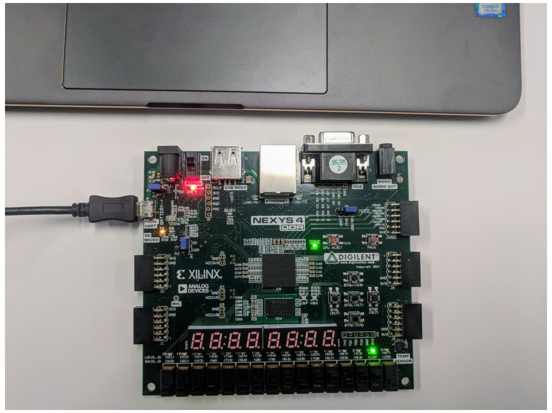
Pre-defined sparse networks need smaller λ (as determined by validation)

Overall Density	**λ**
100 %	1.1×10^{-4}
40 %	5.5×10^{-5}
11 %	0

Example for MNIST 2-junction networks

Pre-defined sparsity reduces the overfitting problem stemming from over-parametrization in big networks

Applications and extensions of pre-defined sparsity



A hardware architecture for on-device training and inference, prototype implemented on FPGA

S. Dey, Y. Shao, K. M. Chugg and P. A. Beerel, “Accelerating training of deep neural networks via sparse edge processing,” in *26th International Conference on Artificial Neural Networks (ICANN)* Part 1, pp. 273-280. Springer, Sep 2017.

Transferred to and currently being developed by team SAPIENT, in collaboration with DTRA and USC Information Sciences Institute (ISI).

W_{11}	W_{12}	W_{13}
W_{21}	W_{22}	W_{23}
W_{31}	W_{32}	W_{33}

Sparsifying kernels / filters in convolutional layers

S. Kundu, M. Nazemi, M. Pedram, K. M. Chugg and P. A. Beerel, “Pre-defined Sparsity for Low-Complexity Convolutional Neural Networks,” in *IEEE Transactions on Computers*, 2020.

Custom sparse libraries in software frameworks

- torch.sparse – Experimental API with room for improvement
- tensorflow.sparse

S. Dey, “Sparse Matrices in Pytorch,” in *Towards Data Science, Medium publication*, 2019. https://towardsdatascience.com/sparse-matrices-in-pytorch-be8ecaccae6

Deep-n-Cheap

Automated search framework to explore performance-complexity tradeoffs in CNN design

Objective

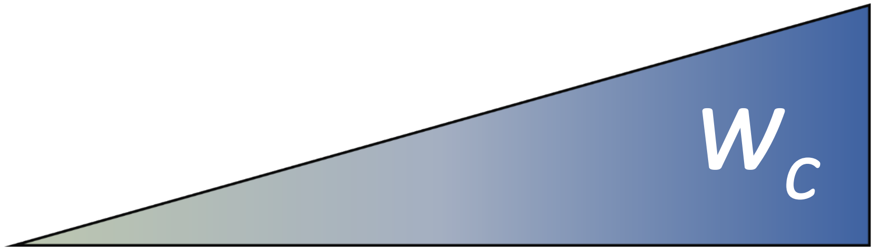
Find networks which optimize performance keeping in mind given complexity constraints

Loss function for search: $f = f_p(\text{Performance}) + w_c * f_c(\text{Complexity})$

1 – val accuracy

Wall clock time to train per epoch

w_c is like regularization



The optimization problem for *f*

Performance and complexity are both functions of overall config:

- Discrete architecture hyperparameters (#layers, type of layer, …), AND
- Continuous training hyperparameters (learning rate, weight decay, …)

Example: $\boldsymbol{x}_i$ = (10 conv layers, 6 batchnorm layers, lr=10^{-3}, weight_decay=10^{-4})

Approaches:

- **Bayesian optimization**
- *Grid search*

Bayesian optimization

Sample $f(\cdot)$ for n initial configs and model via a *Gaussian process*

$$f\left(\boldsymbol{X}_{1:n}\right) \sim \mathcal{N}\left(\underset{n\times 1}{\boldsymbol{\mu}}, \underset{n\times n}{\boldsymbol{\Sigma}}\right) \qquad \boldsymbol{\Sigma} = \begin{bmatrix} k(\boldsymbol{x}_1, \boldsymbol{x}_1) & \cdots & k(\boldsymbol{x}_1, \boldsymbol{x}_n) \\ \vdots & \text{Covariance kernel } k & \vdots \\ k(\boldsymbol{x}_n, \boldsymbol{x}_1) & \cdots & \kappa(\boldsymbol{x}_n, \boldsymbol{x}_n) \end{bmatrix}$$

Calculate *expected improvement* for new configs

$$EI\left(\boldsymbol{x}_{\text{new}}\right) = (\underbrace{f^*}_{\text{Current best}} - \mu)\, \underbrace{P}_{N(0,1)\text{ cdf}}\left(\frac{f^* - \mu}{\sigma}\right) + \sigma \underbrace{p}_{N(0,1)\text{ pdf}}\left(\frac{f^* - \mu}{\sigma}\right)$$

Expensive f evaluations are minimized

Designing the 'ramp' kernel function

Distance $$d\left(x_1, x_2\right) = \omega \left(\frac{|x_1 - x_2|}{u - l} \right)^r$$

Kernel $$k\left(x_1, x_2\right) = e^{-d^2/2}$$

<u>Example: batch_size</u>
Given: l=32 <= batch_size <= u=512

x_1 = 200, x_2 = 100, ω = 3, r = 1

=> k = 0.82

Kernel values for each search parameter are combined to get overall kernel between 2 configs

Search framework

- Step 1 – **Core architecture** using Bayesian optimization
  - Number of convolutional layers
  - Number of filters in each

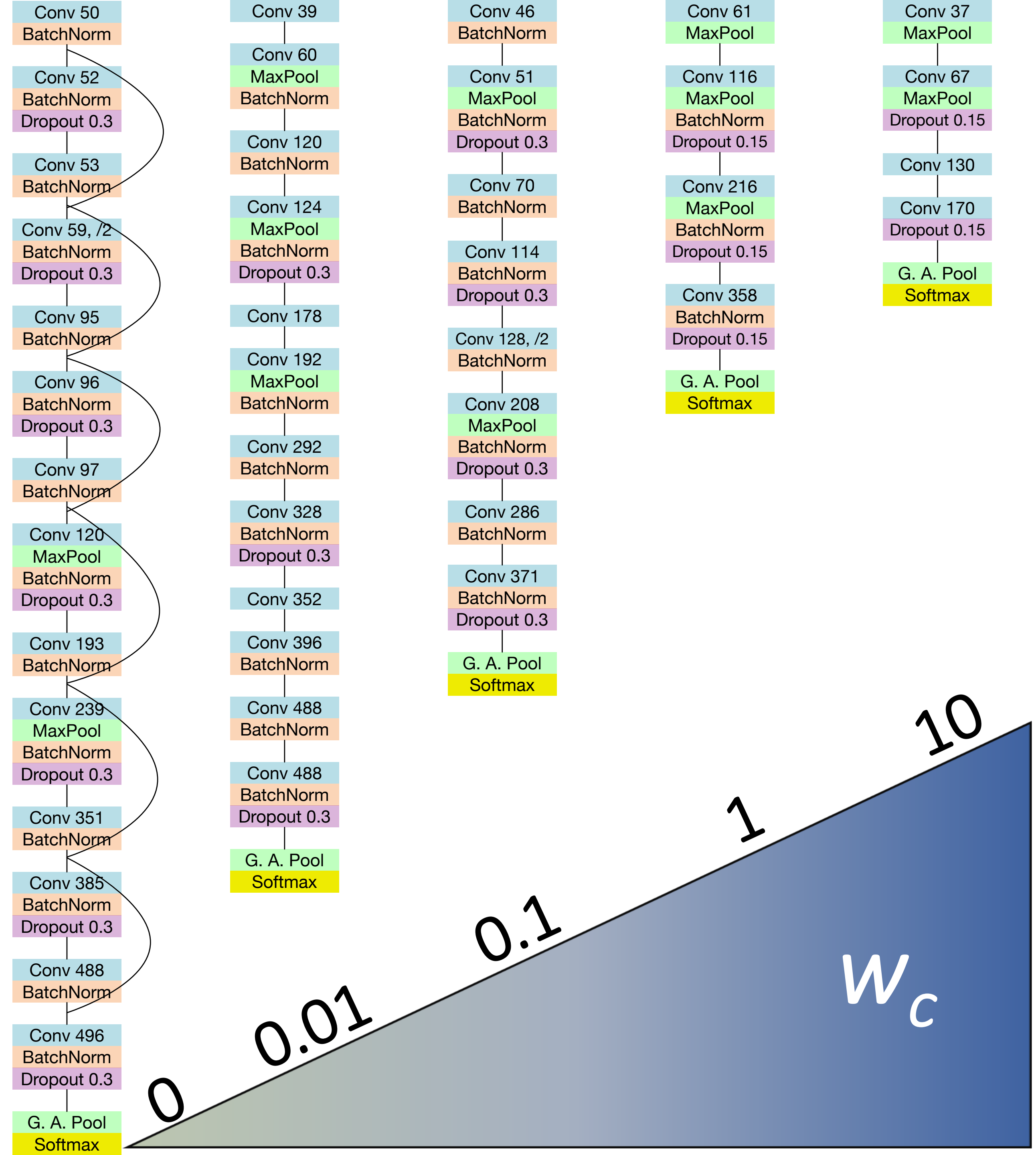
  Combined space

- Step 2 – **Remaining architecture** using grid search
  1. Strides vs max pooling
  2. Batch normalization locations
  3. Dropout locations and probabilities
  4. Shortcut connections

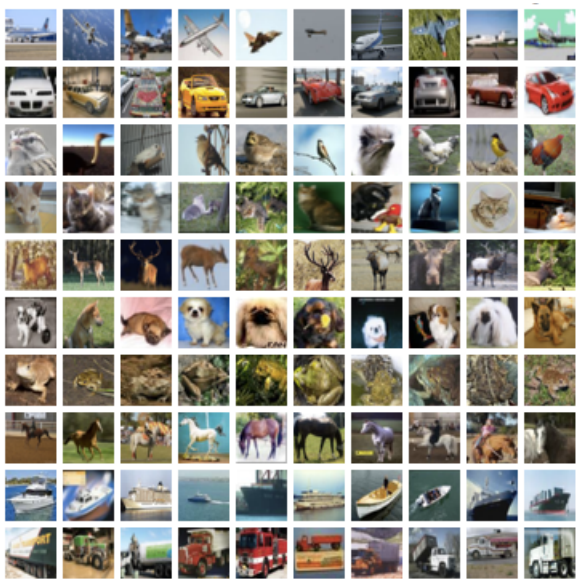
  Sequential, but order can be interchanged

- Step 3 – **Training hyperparameters** using Bayesian optimization
  - Initial learning rate
  - Weight decay
  - Batch size

  Combined space

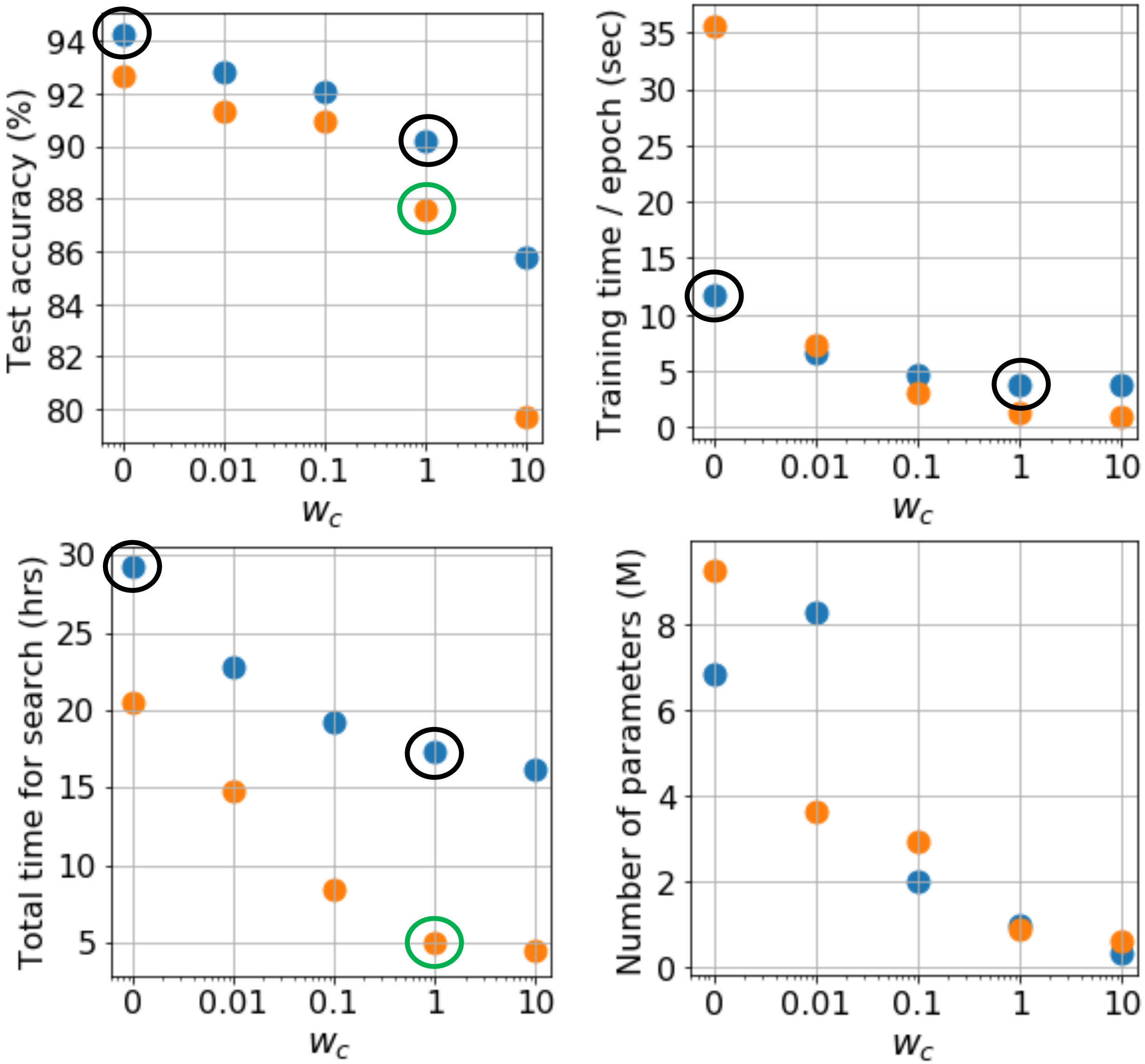


CIFAR-10 Results



w_c	Initial lr	Weight decay	Batch size
0	0.001	3×10^{-4}	195
0.01	0.001	8.3×10^{-5}	256
0.1	0.001	1.2×10^{-5}	459
1	0.003	0	452
10	0.001	0	256

Effect of w_c

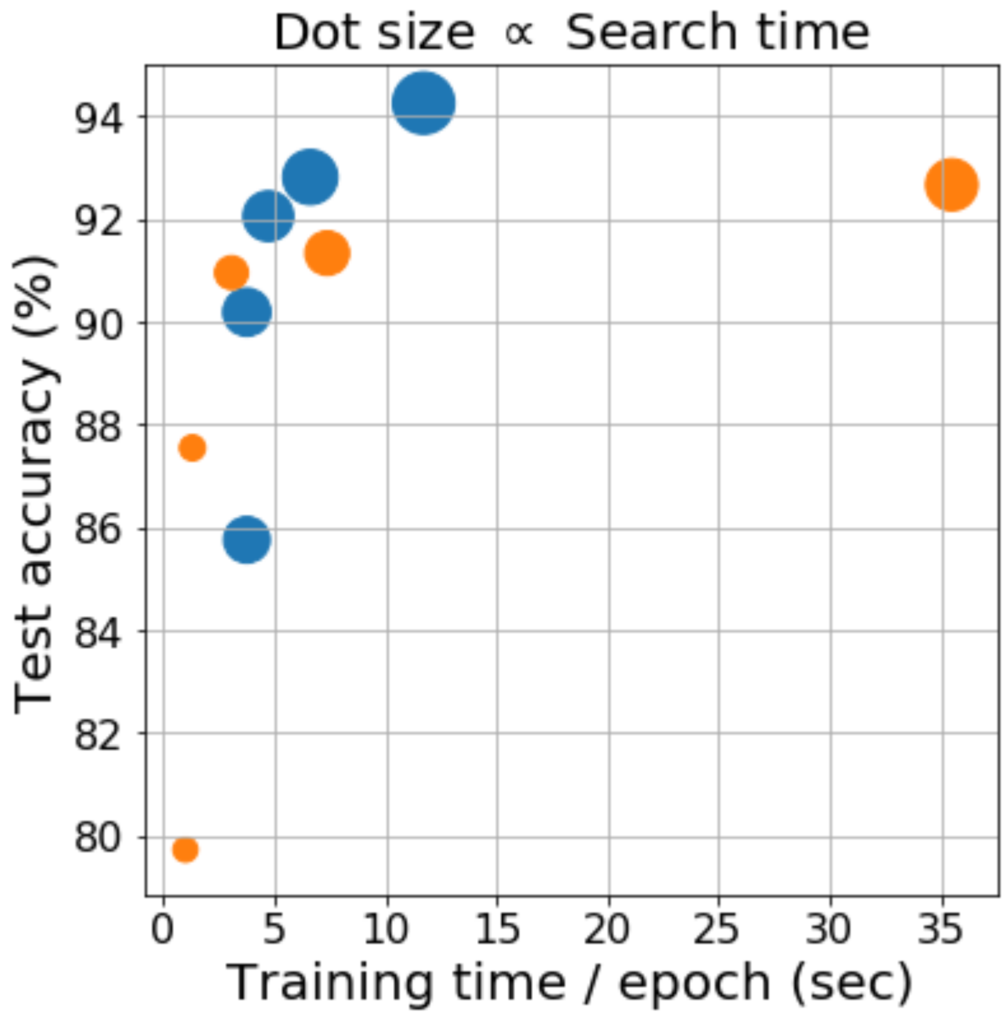


Legend:
CIFAR-10 with basic augmentation – normalization, flips, crops. Incurs data loading overheads!
CIFAR-10 without any preprocessing / augmentation. No data loading overheads!

Some observations:

- *Tradeoffs when switching from w_c = 0 (14 layers) to w_c = 1 (4 layers) :* Per epoch training time reduces by 3X and overall search time by >10 hrs at the cost of 4% performance
- A 5 hr search yields a net with 88% accuracy on CIFAR-10.

Performance - Complexity Tradeoff

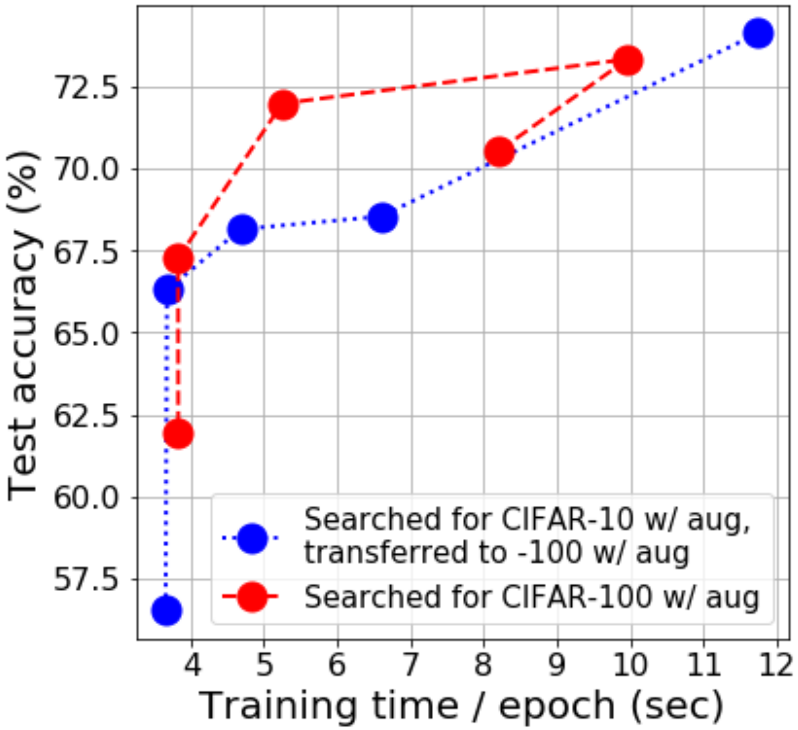


Spend a lot to get that last bit of performance!

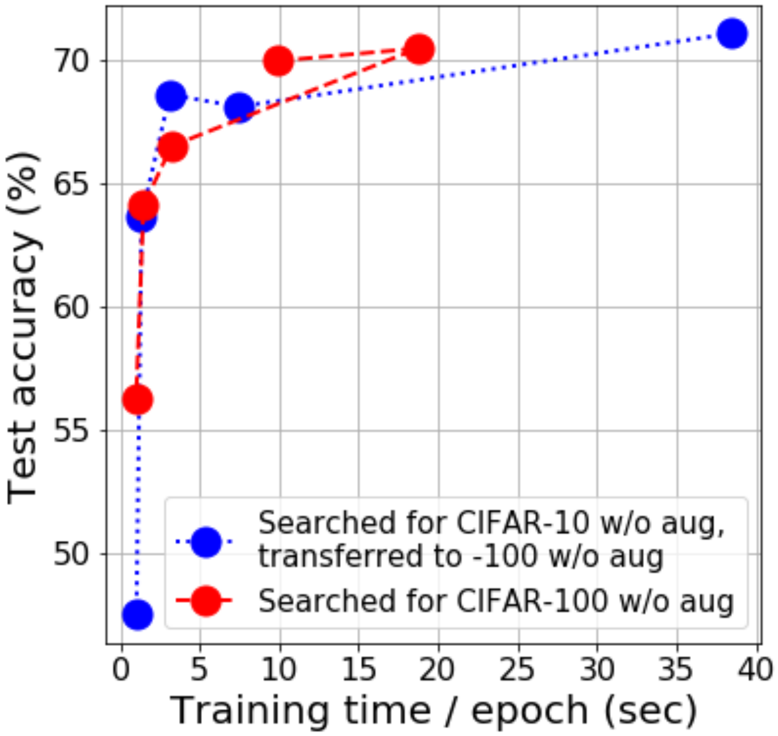
Transfer learning to CIFAR-100

Given a search process optimized for dataset A, how do the best configs perform on dataset B?

In other words, how do they compare with configs from a separate search optimized for B?



Best network for CIFAR-10 performs better on CIFAR-100 than the best network for CIFAR-100!



w_c varies through [10, 1, 0.1, 0.01, 0] along each line

Our upcoming work…

- We will release Deep-n-Cheap as an autoML framework for people to freely use to explore performance-complexity tradeoffs. (*ETA: On Github in April*)
    - Existing frameworks like AutoKeras and AutoGluon are limited in the architectures they search. For example, AutoGluon takes mins / epoch to try >50 layer networks on CIFAR-10.
    - Other existing frameworks like Auto-sklearn and Auto-PyTorch only support MLPs.

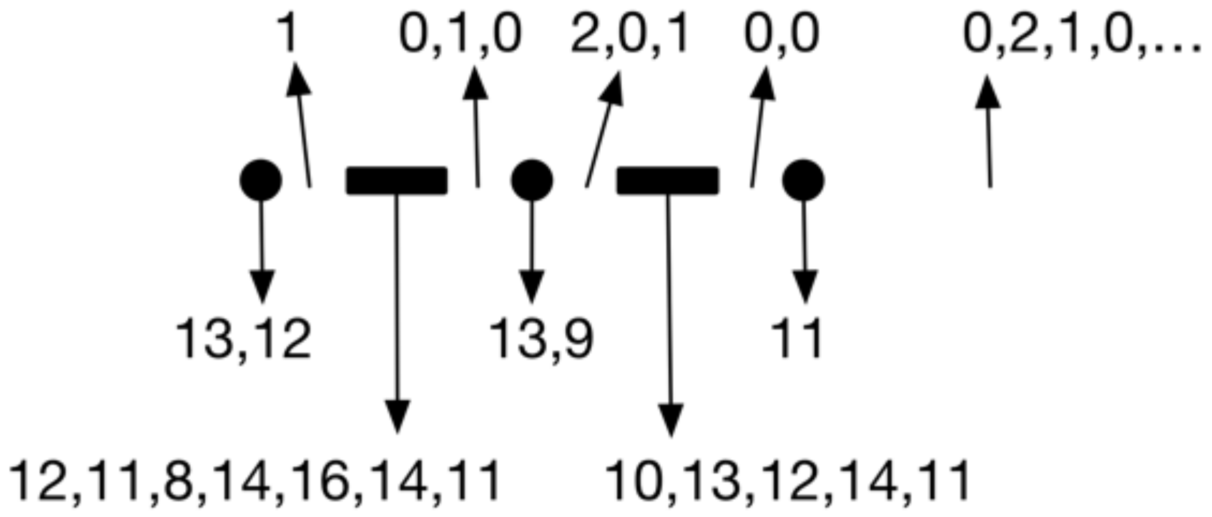
Dataset Engineering

A family of synthetic datasets of customizable difficulty for ML classification problems

Synthetic Datasets on Morse Code Classification

- Inputs: Intensity values for dots, dashes and spaces in Morse codewords
- Outputs: The actual symbol represented by the codeword
- Added features to customize difficulty such as noise and dataset size
- Cheaply generate large quantities of data

S. Dey, K. M. Chugg and P. A. Beerel, "Morse Code Datasets for Machine Learning," in ICCCNT 2018.
Won Best Paper award.
https://github.com/usc-hal/morse-dataset



Team Members

Peter Beerel
Professor

Keith Chugg
Professor

Leana Golubchik
Professor

Kuan-Wen Huang
PhD Student

Andrew Schmidt
Senior Computer Scientist, USC ISI

Souvik Kundu
PhD Student

Saikrishna C. Kanala
MS Student

... and others

Thank you!

I'll graduate soon and plan to conduct postdoctoral research

https://souryadey.github.io/