DnC

# Deep-n-Cheap

## An Automated Search Framework for Low Complexity Deep Learning

Sourya Dey

PhD, University of Southern California

May 29th, 2020

# Outline

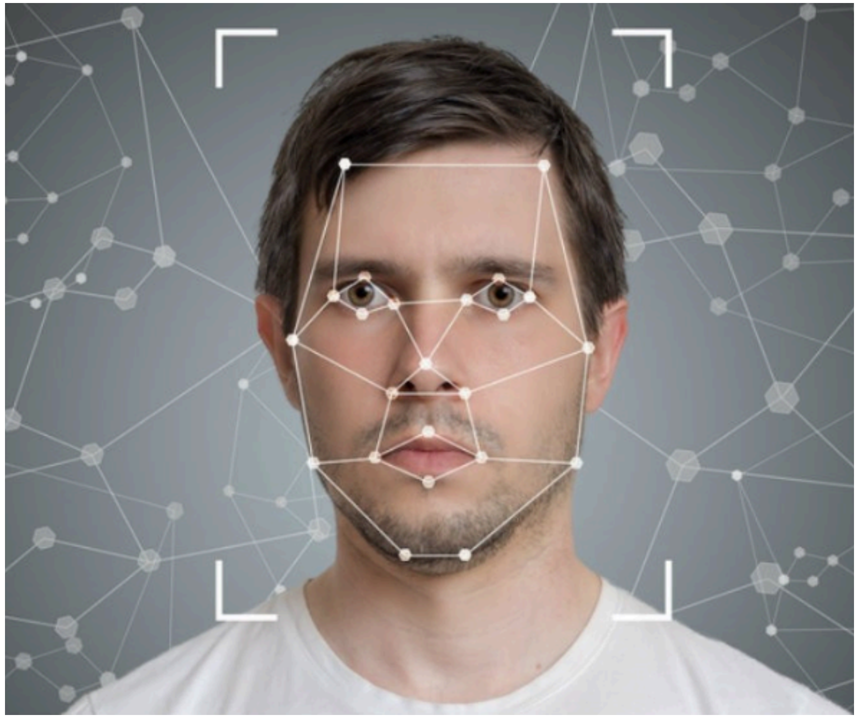


# Overview

# Overview

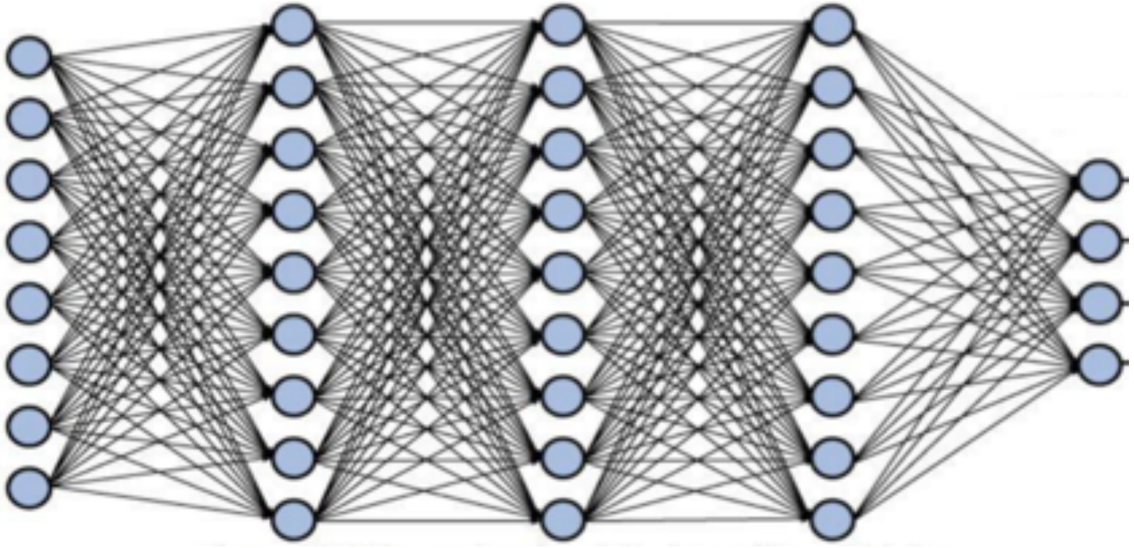
*Neural networks (NNs) are key machine learning technologies*

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



# The Complexity Conundrum…

## *Modern neural networks suffer from parameter explosion*



Training can take weeks on CPU

Cloud GPU resources are expensive

He 2016

# … and the Design Conundrum

- Deep neural networks have a lot of **hyperparameters**
  - How many layers?
  - How many neurons?

    *Architecture Hyperparameters*
  - Learning rate
  - Batch size

    *Training Hyperparameters*
  - and more…

- Our understanding of NNs is at best vague, at worst, zero!

# AutoML (Automated Machine Learning)

- Software frameworks that make design decisions
- Given problem specifications, **search** for NN models

AutoKeras

Jin 2019 – Auto-Keras

GLUON

AWsLabs 2020 – AutoGluon

AutoML.org
Freiburg-Hannover

Mendoza 2018 – Auto-PyTorch

# Our Work

DnC Deep-n-Cheap

Low Complexity AutoML framework

*Reduce training complexity*

*Target custom datasets and user requirements*

*Output complete training configs*

| Framework | Architecture search space | Training hyp search | Adjust model complexity |
|---|---|---|---|
| Auto-Keras | Only pre-existing architectures | No | No |
| AutoGluon | Only pre-existing architectures | Yes | No |
| Auto-PyTorch | Customizable by user | Yes | No |
| Deep-n-Cheap | Customizable by user | Yes | Penalize $t_{tr}$, $N_p$ |

*$t_{tr}$ = Training time / epoch*

*$N_p$ = # Trainable parameters*

# Relevant Details

DnC

- Development started in July 2019
- Supports Pytorch
- Supports classification via CNNs and MLPs

- Latest / ongoing work:
    - Support for Keras
    - Regression
    - Detection / segmentation
    - RNNs

S. Dey, S. C. Kanala, K. M. Chugg and P. A. Beerel, “Deep-n-Cheap: An Automated Search Framework for Low Complexity Deep Learning”, submitted to ECML-PKDD 2020.

https://arxiv.org/abs/2004.00974

# Approach

# Search Objective

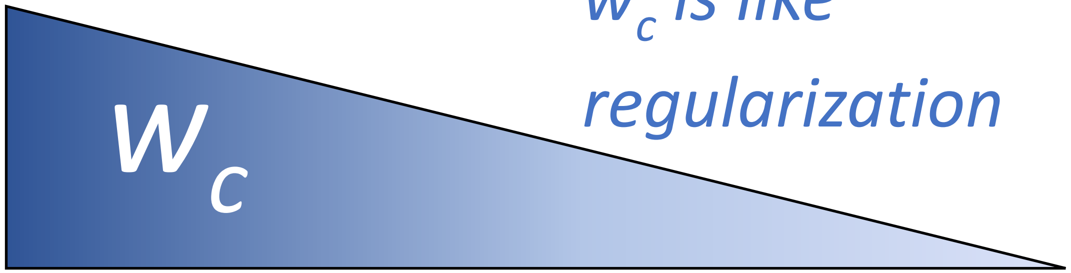
*Optimize performance and complexity*

Modified loss function: $f(\text{ NN Config } \mathbf{x} \text{ }) = \log( f_p + w_c * f_c )$

Example config ***x***:
[#layers, #channels] = [3, (29,40,77)]

$f_p$ = 1 - (Best Validation Accuracy)
$f_c$ = Normalized $t_{tr}$ or $N_p$
= $t_{tr}$(config) / $t_{tr}$(baseline)

*$w_c$ is like regularization*



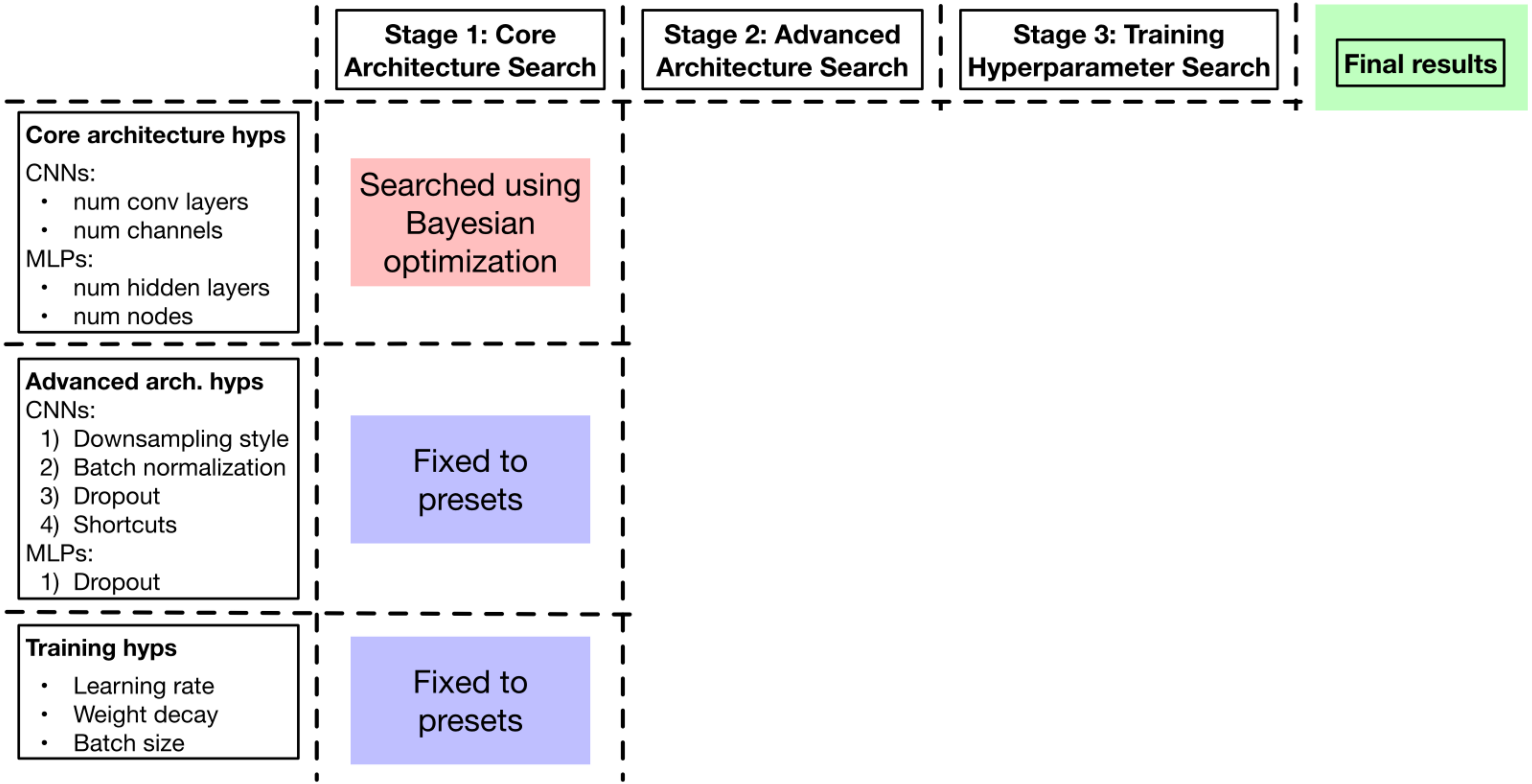

Quick to train
Sacrifice performance

Good performance
Slow to train
Slow search process

# Three-stage search process

| | **Stage 1: Core Architecture Search** | **Stage 2: Advanced Architecture Search** | **Stage 3: Training Hyperparameter Search** | **Final results** |
|---|---|---|---|---|
| **Core architecture hyps**<br>CNNs:<br>• num conv layers<br>• num channels<br>MLPs:<br>• num hidden layers<br>• num nodes | | | | |
| **Advanced arch. hyps**<br>CNNs:<br>1) Downsampling style<br>2) Batch normalization<br>3) Dropout<br>4) Shortcuts<br>MLPs:<br>1) Dropout | | | | |
| **Training hyps**<br>• Learning rate<br>• Weight decay<br>• Batch size | | | | |

# Three-stage search process



# Three-stage search process

| | Stage 1: Core Architecture Search | Stage 2: Advanced Architecture Search | Stage 3: Training Hyperparameter Search | Final results |
|---|---|---|---|---|
| **Core architecture hyps**<br>CNNs:<br>• num conv layers<br>• num channels<br>MLPs:<br>• num hidden layers<br>• num nodes | Searched using Bayesian optimization | Fixed to Stage 1 search results | | |
| **Advanced arch. hyps**<br>CNNs:<br>1) Downsampling style<br>2) Batch normalization<br>3) Dropout<br>4) Shortcuts<br>MLPs:<br>1) Dropout | Fixed to presets | Multiple grid searches in sequence | | |
| **Training hyps**<br>• Learning rate<br>• Weight decay<br>• Batch size | Fixed to presets | Fixed to presets | | |

# Three-stage search process

| | Stage 1: Core Architecture Search | Stage 2: Advanced Architecture Search | Stage 3: Training Hyperparameter Search | Final results |
|---|---|---|---|---|
| **Core architecture hyps**<br>CNNs:<br>• num conv layers<br>• num channels<br>MLPs:<br>• num hidden layers<br>• num nodes | Searched using Bayesian optimization | Fixed to Stage 1 search results | Fixed to Stage 1 search results | Stage 1 search results |
| **Advanced arch. hyps**<br>CNNs:<br>1) Downsampling style<br>2) Batch normalization<br>3) Dropout<br>4) Shortcuts<br>MLPs:<br>1) Dropout | Fixed to presets | Multiple grid searches in sequence | Fixed to Stage 2 search results | Stage 2 search results |
| **Training hyps**<br>• Learning rate<br>• Weight decay<br>• Batch size | Fixed to presets | Fixed to presets | Searched using Bayesian optimization | Stage 3 search results |

ARCHITECTURE + TRAINING

# Bayesian Optimization – Gaussian process

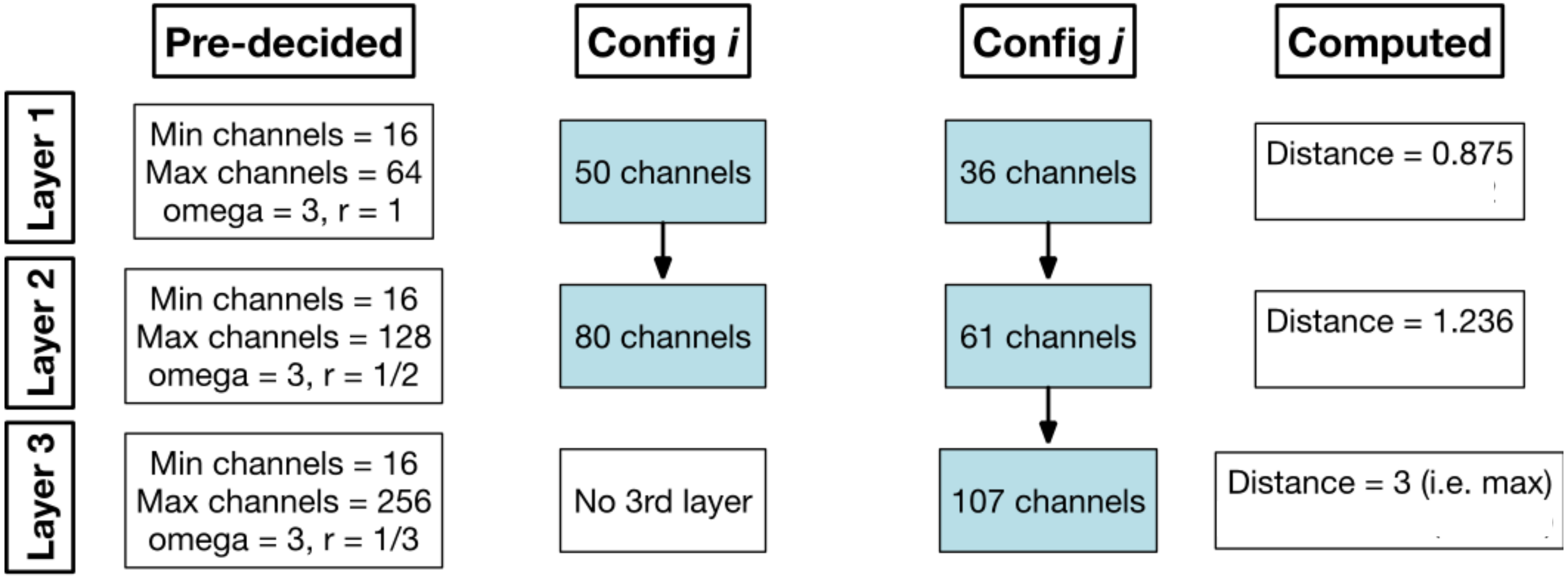
$$f\left(\boldsymbol{X}_{1:n}\right) \sim \mathcal{N}\left(\underset{n\times 1}{\boldsymbol{\mu}}, \underset{n\times n}{\boldsymbol{\Sigma}}\right)$$

$$\boldsymbol{\mu} = \begin{bmatrix} \mu\left(\boldsymbol{x}_1\right) \\ \vdots \\ \mu\left(\boldsymbol{x}_n\right) \end{bmatrix} \qquad \boldsymbol{\Sigma} = \begin{bmatrix} \sigma\left(\boldsymbol{x}_1, \boldsymbol{x}_1\right) & \cdots & \sigma\left(\boldsymbol{x}_1, \boldsymbol{x}_n\right) \\ \vdots & \ddots & \vdots \\ \sigma\left(\boldsymbol{x}_n, \boldsymbol{x}_1\right) & \cdots & \sigma\left(\boldsymbol{x}_n, \boldsymbol{x}_n\right) \end{bmatrix}$$

# Covariance kernel – Similarity between NN configs

Individual Distance

$$d\left(x_{ik}, x_{jk}\right) = \omega_k \left(\frac{|x_{ik} - x_{jk}|}{u_k - l_k}\right)^{r_k}$$



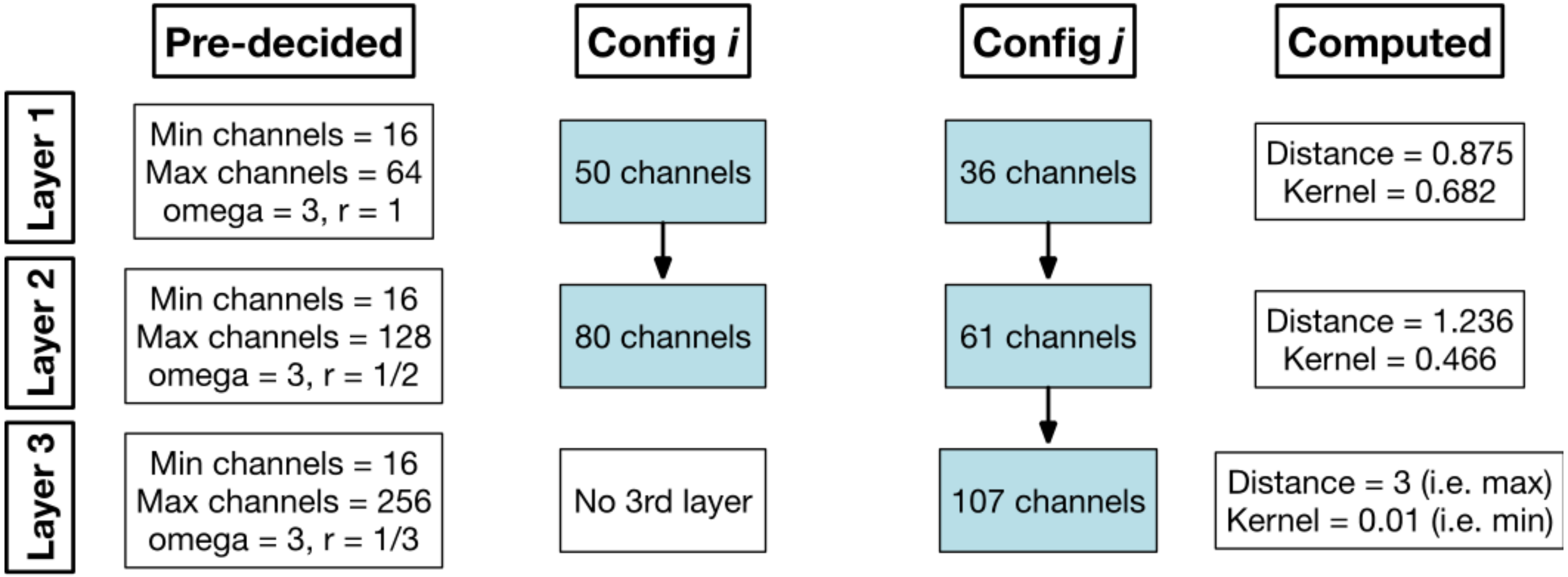
# Covariance kernel – Similarity between NN configs

Individual Distance

$$d\left(x_{ik}, x_{jk}\right) = \omega_k \left(\frac{|x_{ik} - x_{jk}|}{u_k - l_k}\right)^{r_k}$$

Individual Kernel

$$\sigma\left(x_{ik}, x_{jk}\right) = \exp\left(-\frac{d^2(x_{ik}, x_{jk})}{2}\right)$$



# Covariance kernel – Similarity between NN configs

Individual Distance

$$d\left(x_{ik}, x_{jk}\right) = \omega_k \left( \frac{\left|x_{ik} - x_{jk}\right|}{u_k - l_k} \right)^{r_k}$$

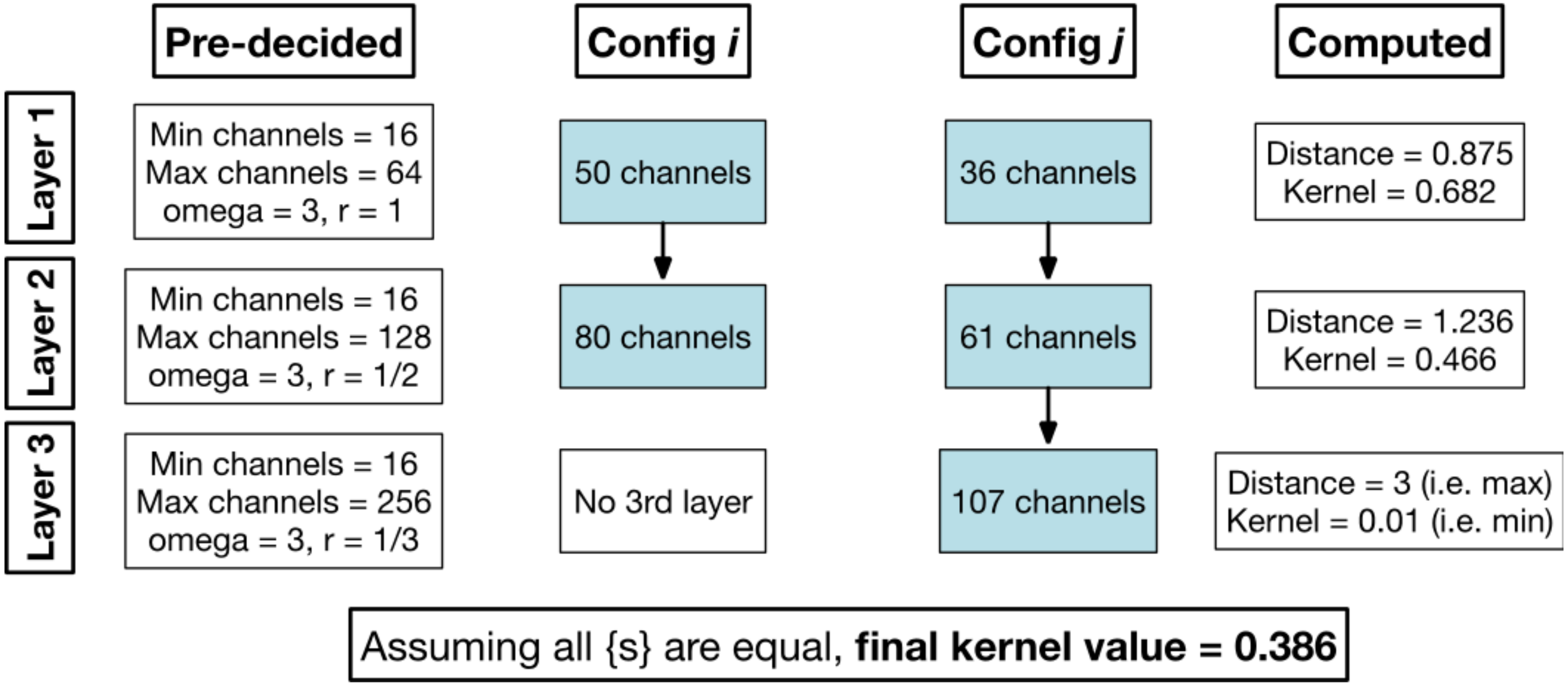
Individual Kernel

$$\sigma\left(x_{ik}, x_{jk}\right) = \exp\left( -\frac{d^2\left(x_{ik}, x_{jk}\right)}{2} \right)$$

Complete Kernel

$$\sigma\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right) = \sum_{k=1}^{K} s_k \sigma\left(x_{ik}, x_{jk}\right)$$

Convex combination



# Bayesian Optimization – Expected Improvement

- How much can a new point **x** in the search space improve over existing points?
- *Don't need to evaluate f(**x**) to find EI(**x**)*
- Can explore lots of points cheaply

$$EI(\boldsymbol{x}) = (f^* - \mu)P\left(\frac{f^* - \mu}{\sigma}\right) + \sigma p\left(\frac{f^* - \mu}{\sigma}\right)$$

# Results

# Deep-n-Cheap in action!

https://github.com/souryadey/deep-n-cheap

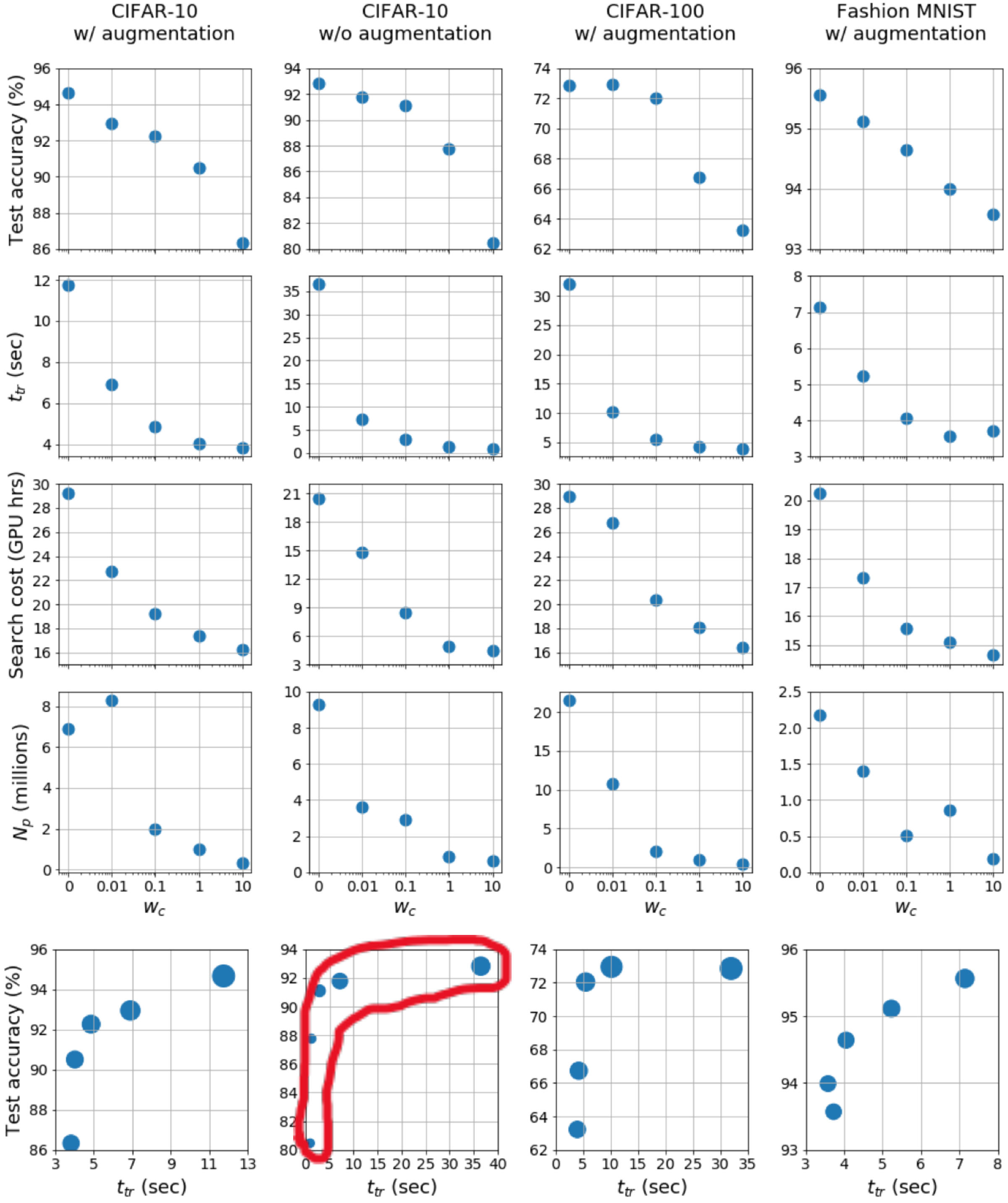
# CNN Results

*Complexity Penalty = Training time / epoch*

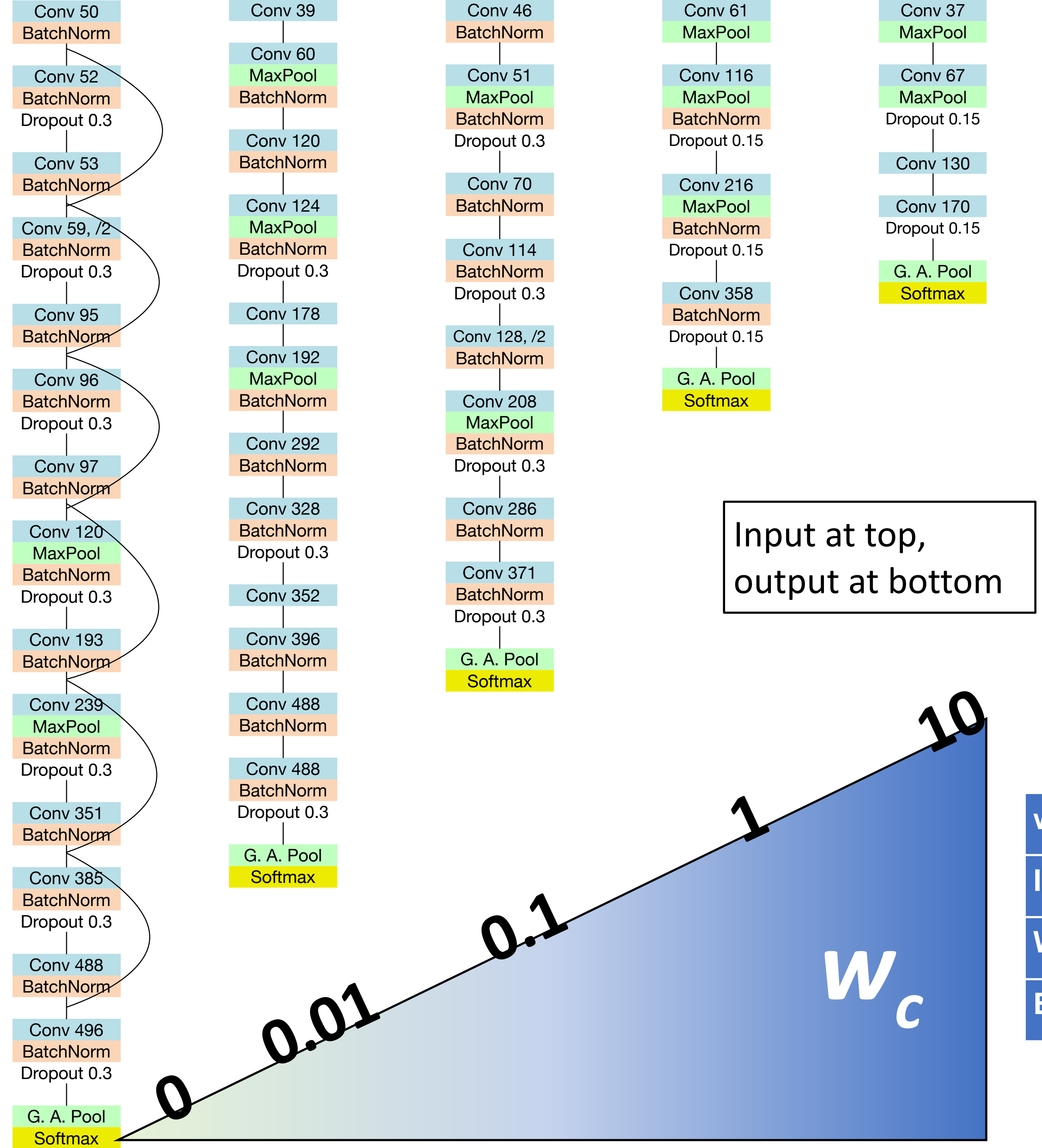
*AWS p3.2xlarge with 1 V100 GPU*

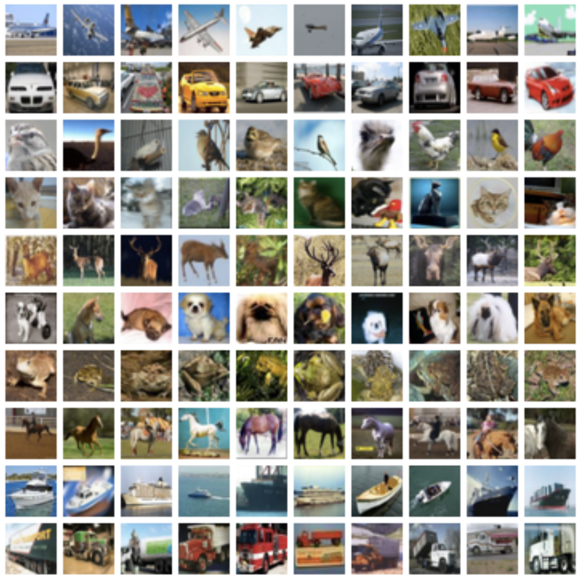
We are not penalizing this, but it's correlated

*Performance-complexity tradeoff*



# CIFAR-10 w/ aug



| $w_c$ | 0 | 0.01 | 0.1 | 1 | 10 |
|---|---|---|---|---|---|
| **Initial learning rate η** | 0.001 | 0.001 | 0.001 | 0.003 | 0.001 |
| **Weight decay λ** | $3.3 \times 10^{-5}$ | $8.3 \times 10^{-5}$ | $1.2 \times 10^{-5}$ | 0 | 0 |
| **Batch size** | 120 | 256 | 459 | 452 | 256 |

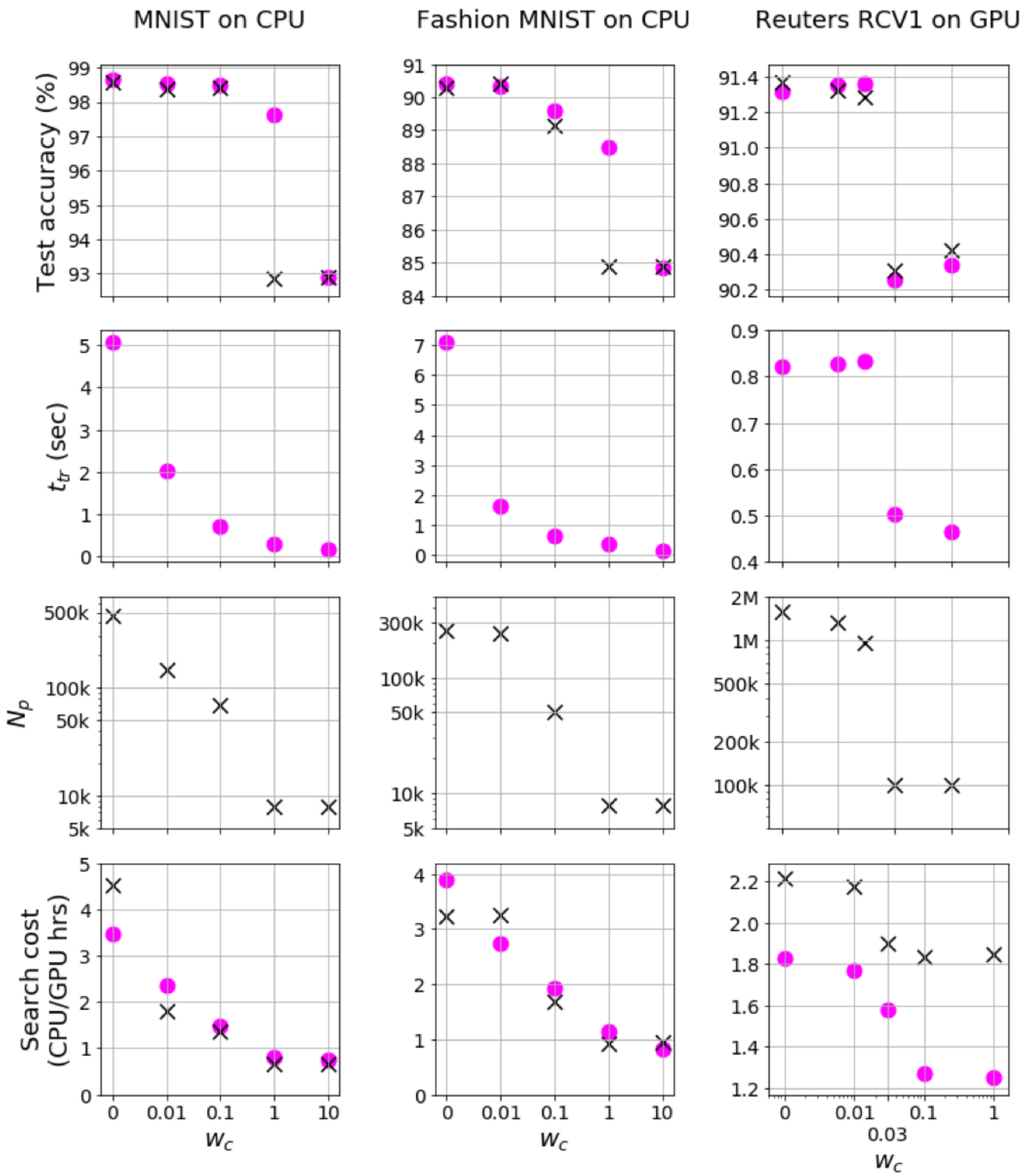
*λ strictly correlated with $N_p$*

# MLP Results

*Pink dots:*
*Complexity Penalty =*
*Training time / epoch*

*Black crosses:*
*Complexity Penalty =*
*# Trainable Params*

*CPU = Macbook Pro with*
*8GB RAM, no CuDA*
*GPU = (Same) AWS*
*p3.2xlarge with V100*



# Comparison (CNNs on CIFAR-10)

| Framework | Additional settings | Search cost (GPU hrs) | Best model found from search | | | |
|---|---|---|---|---|---|---|
| | | | Architecture | $t_{tr}$ (sec) | Batch size | Best val acc (%) |
| Proxyless NAS | Proxyless-G | 96 | 537 conv layers | 429 | 64 | 93.22 |
| Auto-Keras | Default run | 14.33 | Resnet-20 v2 | 33 | 32 | 74.89 |
| AutoGluon | Default run | **3** | Resnet-20 v1 | 37 | 64 | 88.6 |
| | Extended run | 101 | Resnet-56 v1 | 46 | 64 | 91.22 |
| Auto-Pytorch | 'tiny cs' | 6.17 | 30 conv layers | 39 | 64 | 87.81 |
| | 'full cs' | 6.13 | 41 conv layers | 31 | 106 | 86.37 |
| Deep-n-Cheap | $w_c = 0$ | 29.17 | 14 conv layers | 10 | 120 | **93.74** |
| | $w_c = 0.1$ | 19.23 | 8 conv layers | 4 | 459 | 91.89 |
| | $w_c = 10$ | 16.23 | 4 conv layers | **3** | 256 | 83.82 |

Penalizes inference complexity, <u>not</u> training

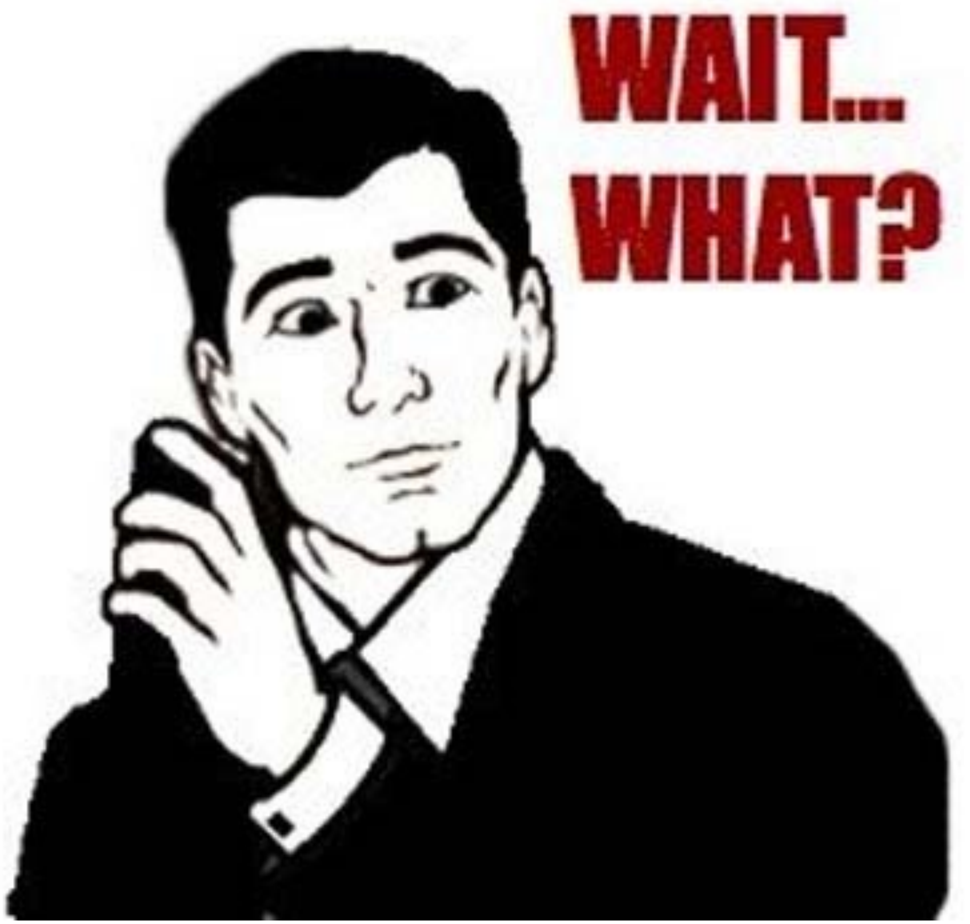
Auto Keras and Gluon don't support getting final model out, so we compared on best val acc found during search instead of final test acc

# Comparison (MLPs)

| Framework | Additional settings | Search cost (GPU hrs) | Best model found from search | | | | |
|---|---|---|---|---|---|---|---|
| | | | MLP layers | $N_p$ | $t_{tr}$ (sec) | Batch size | Best val acc (%) |
| **Fashion MNIST** | | | | | | | |
| Auto-Pytorch | 'tiny cs' | 6.76 | 50 | 27.8M | 19.2 | 125 | **91** |
| | 'medium cs' | 5.53 | 20 | 3.5M | 8.3 | 184 | 90.52 |
| | 'full cs' | 6.63 | 12 | 122k | 5.4 | 173 | 90.61 |
| Deep-n-Cheap (penalize $t_{tr}$) | $w_c = 0$ | 0.52 | 3 | 263k | 0.4 | 272 | 90.24 |
| | $w_c = 10$ | **0.3** | 1 | **7.9k** | **0.1** | 511 | 84.39 |
| Deep-n-Cheap (penalize $N_p$) | $w_c = 0$ | 0.44 | 2 | 317k | 0.5 | 153 | 90.53 |
| | $w_c = 10$ | 0.4 | 1 | **7.9k** | 0.2 | 256 | 86.06 |
| **Reuters RCV1** | | | | | | | |
| Auto-Pytorch | 'tiny cs' | 7.22 | 38 | 19.7M | 39.6 | 125 | 88.91 |
| | 'medium cs' | 6.47 | 11 | 11.2M | 22.3 | 337 | 90.77 |
| Deep-n-Cheap (penalize $t_{tr}$) | $w_c = 0$ | 1.83 | 2 | 1.32M | 0.7 | 503 | **91.36** |
| | $w_c = 1$ | **1.25** | 1 | **100k** | **0.4** | 512 | 90.34 |
| Deep-n-Cheap (penalize $N_p$) | $w_c = 0$ | 2.22 | 2 | 1.6M | 0.6 | 512 | **91.36** |
| | $w_c = 1$ | 1.85 | 1 | **100k** | 5.54 | 33 | 90.4 |

# Takeaway

*We may not need very deep networks!*




Also see Zagoruyko 2016 – WRN

Thank you!

Q&A ??

https://souryadey.github.io/