



*DnC*

# Deep-n-Cheap

An Automated Search Framework  
for Low Complexity Deep Learning

Sourya Dey

PhD, University of Southern California

April 24<sup>th</sup>, 2020


# Outline



Overview



Approach

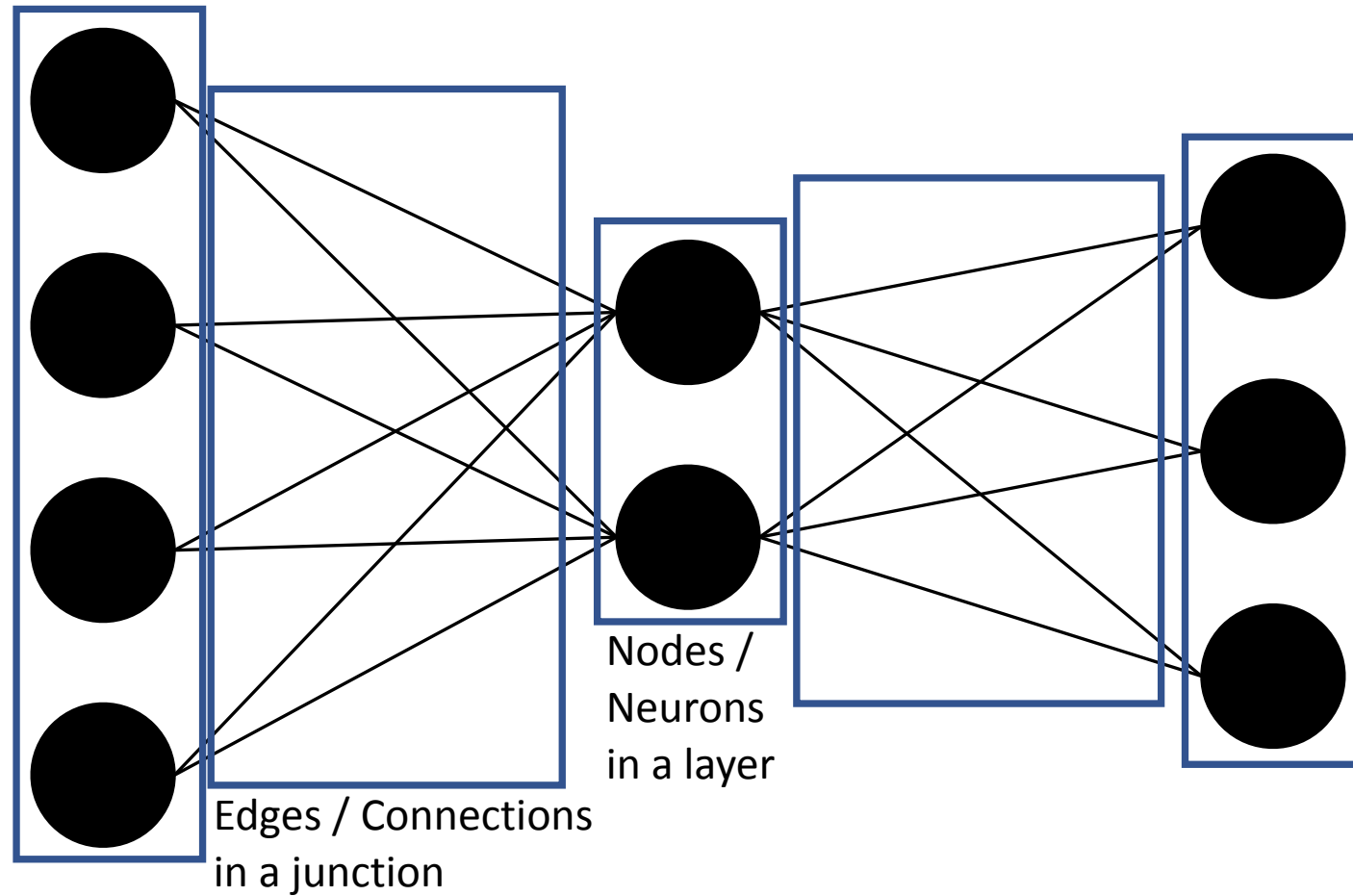


Results

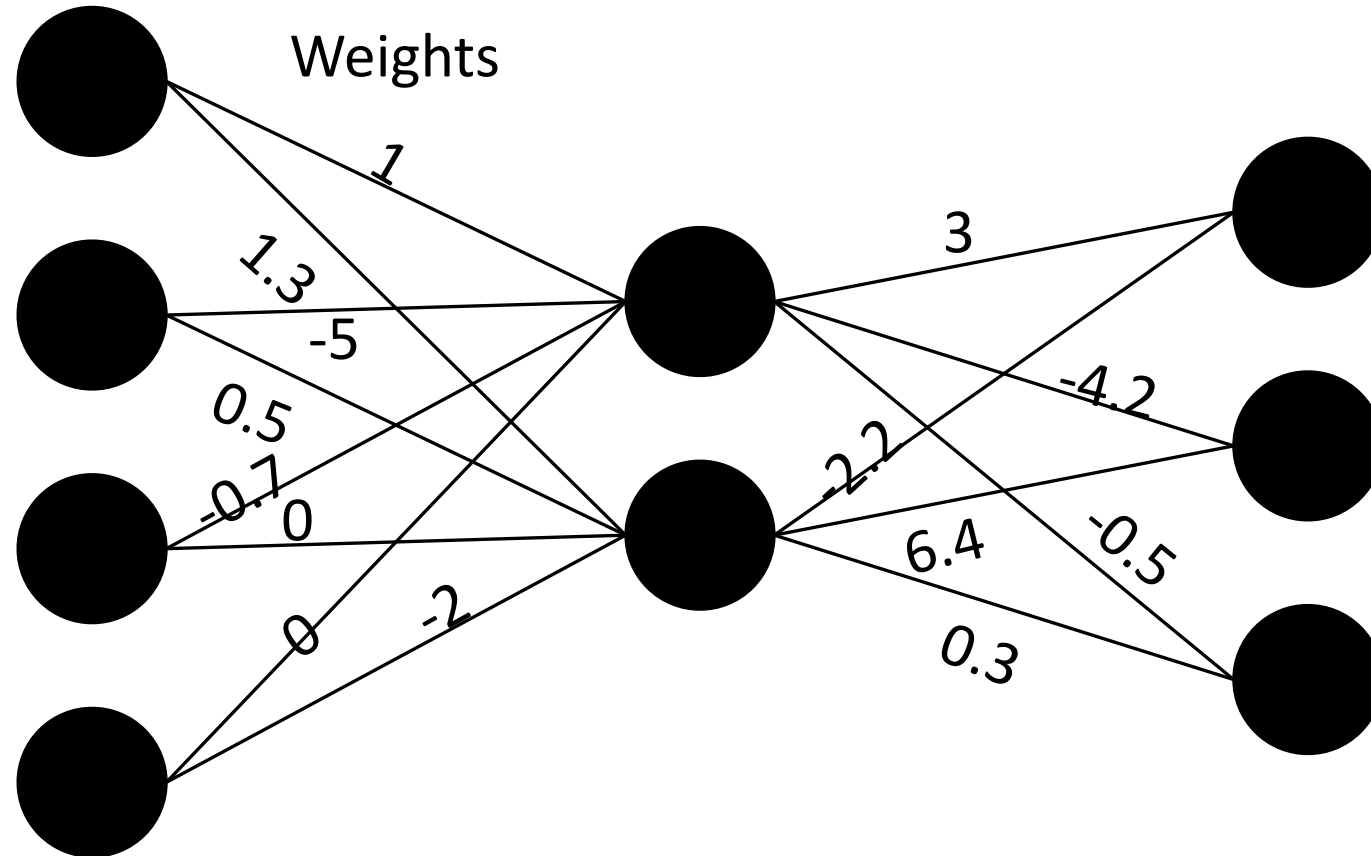


# Overview

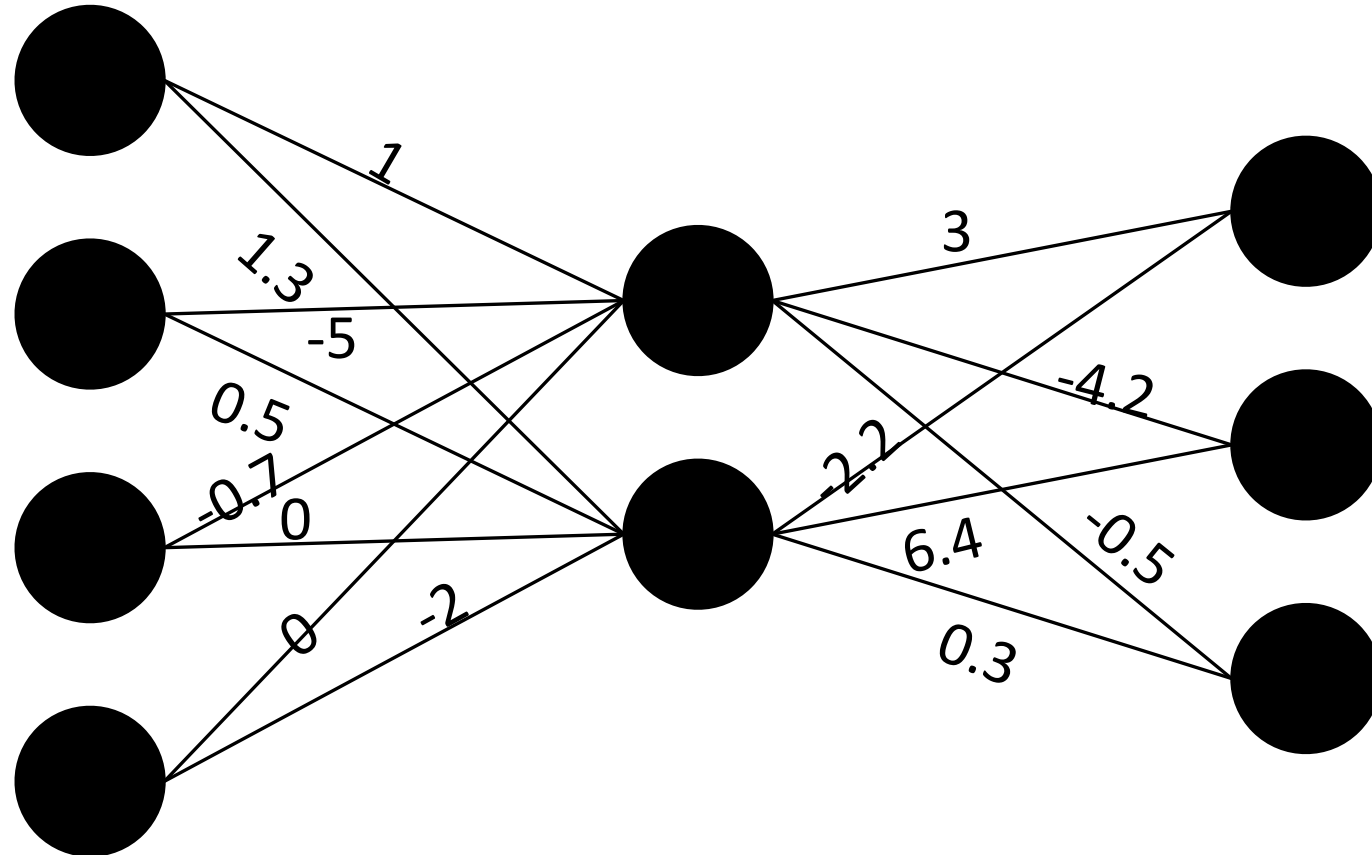
# A Quick Primer on Neural Networks (NN101)



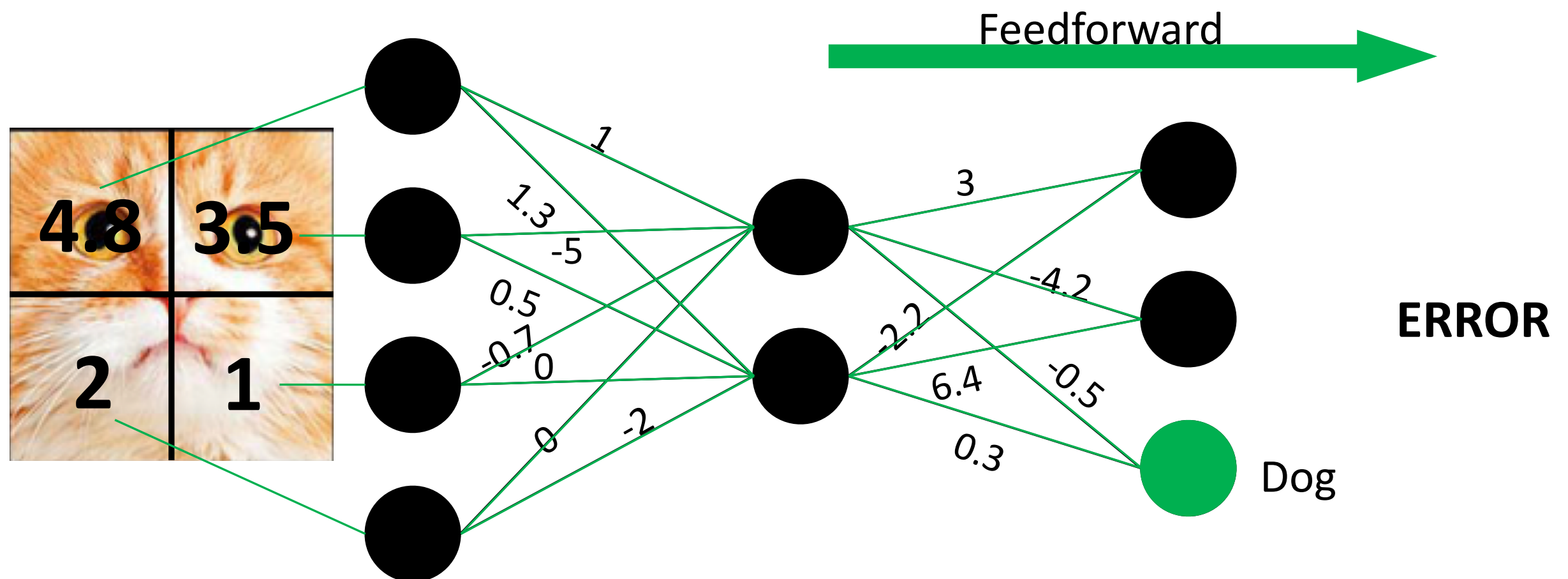
# A Quick Primer on Neural Networks (NN101)



# A Quick Primer on Neural Networks (NN101)



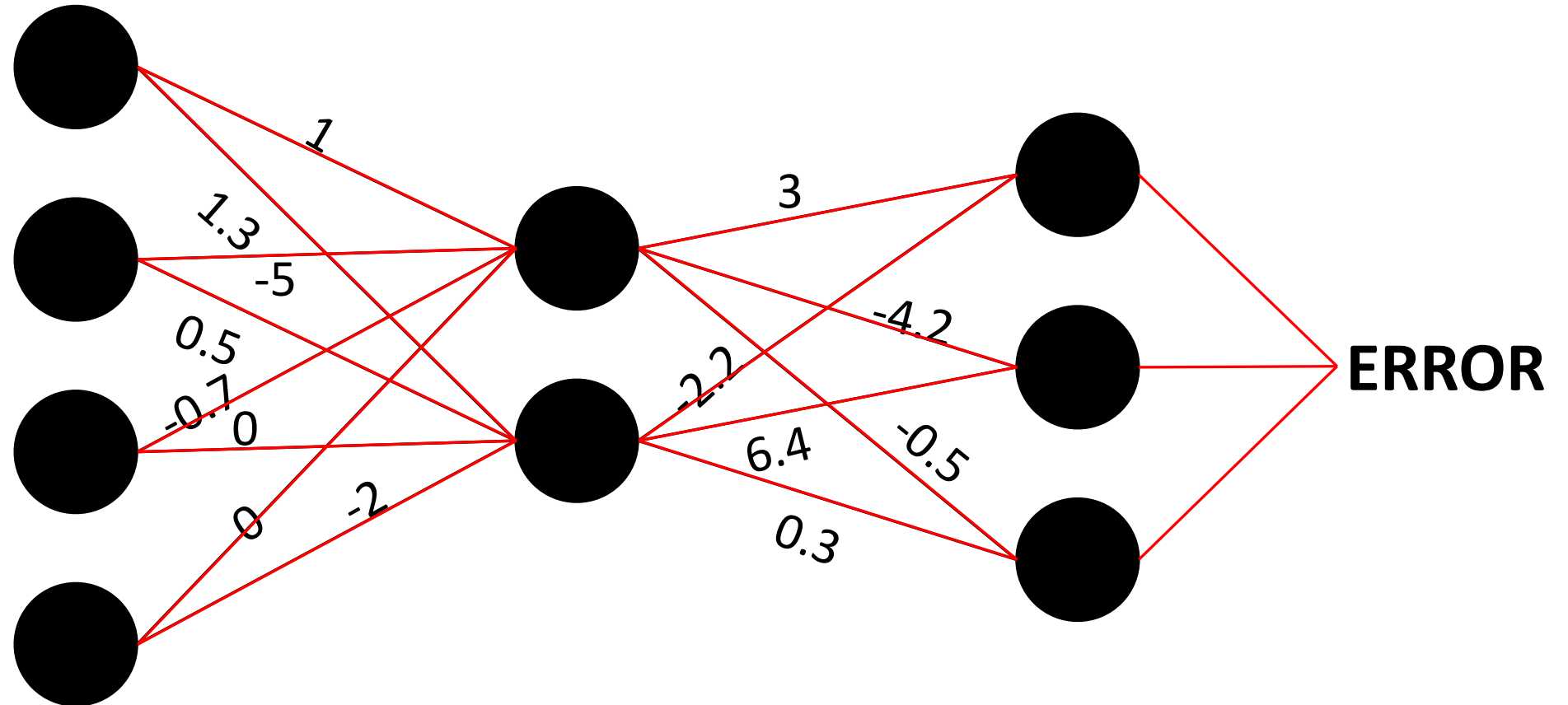
# A Quick Primer on Neural Networks (NN101)



**TRAINING**

*Learn network parameters — weights*

# A Quick Primer on Neural Networks (NN101)



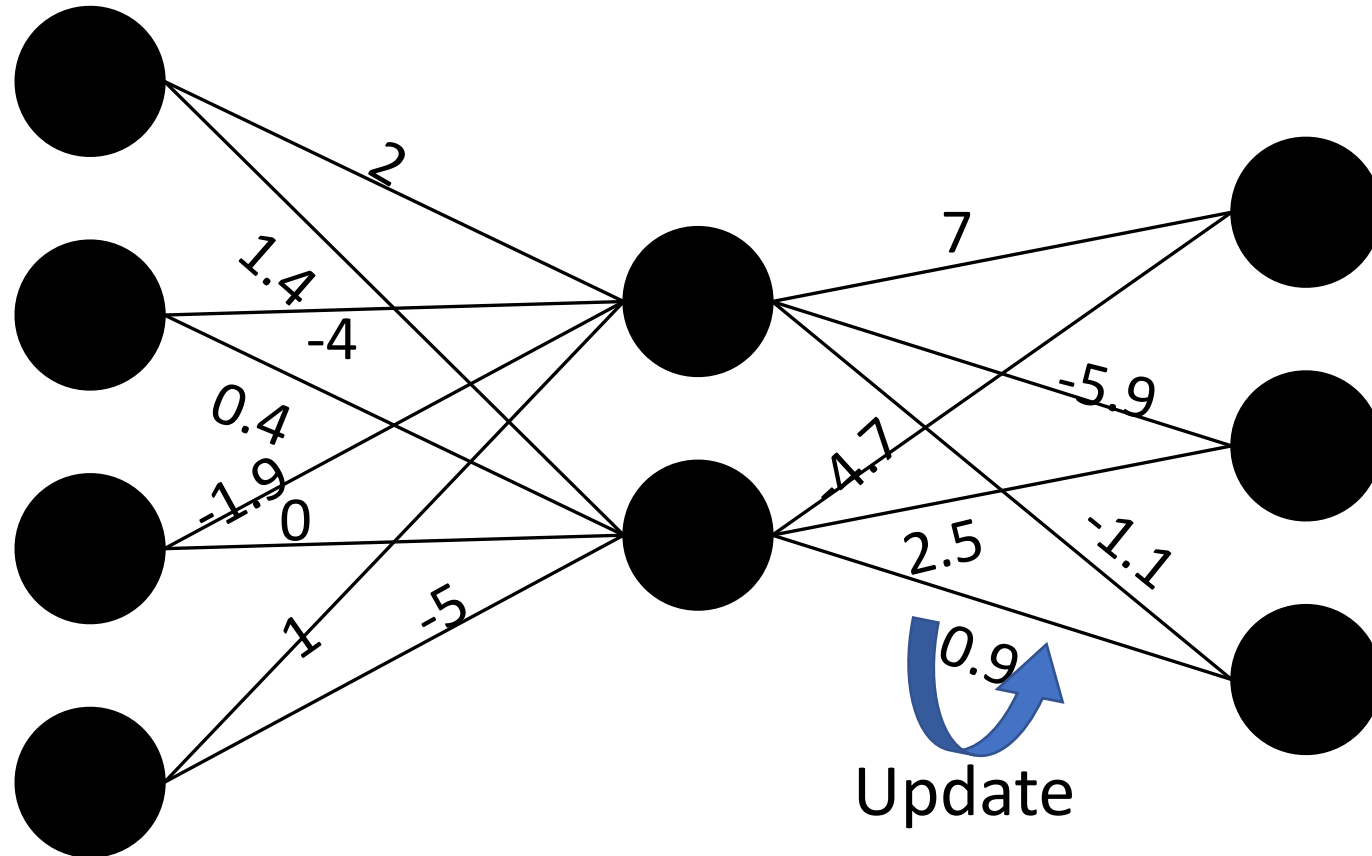
**TRAINING**

*Learn network parameters — weights*

Backpropagation



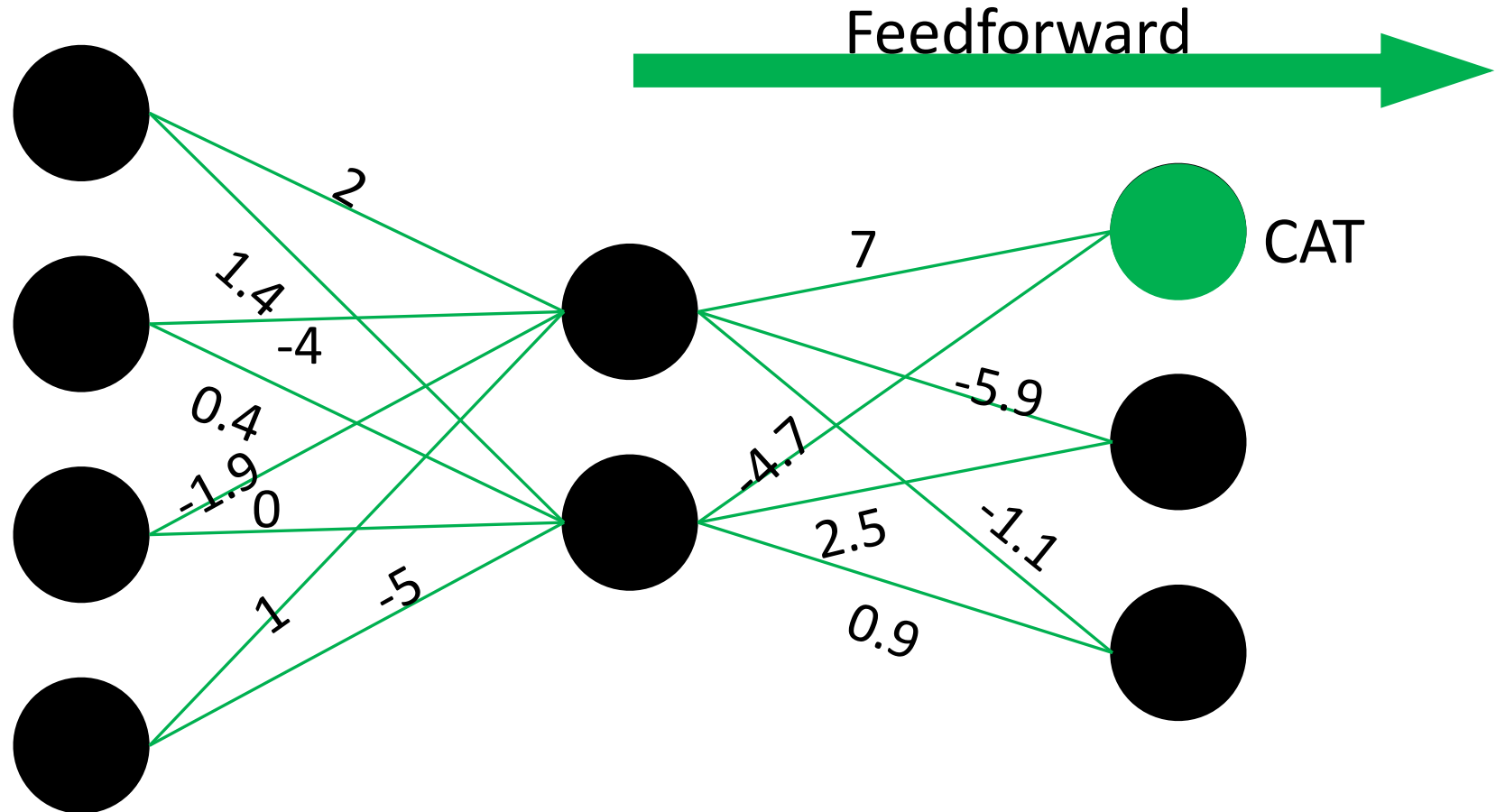
# A Quick Primer on Neural Networks (NN101)



**TRAINING**

*Learn network parameters — weights*

# NNs can be used for classification

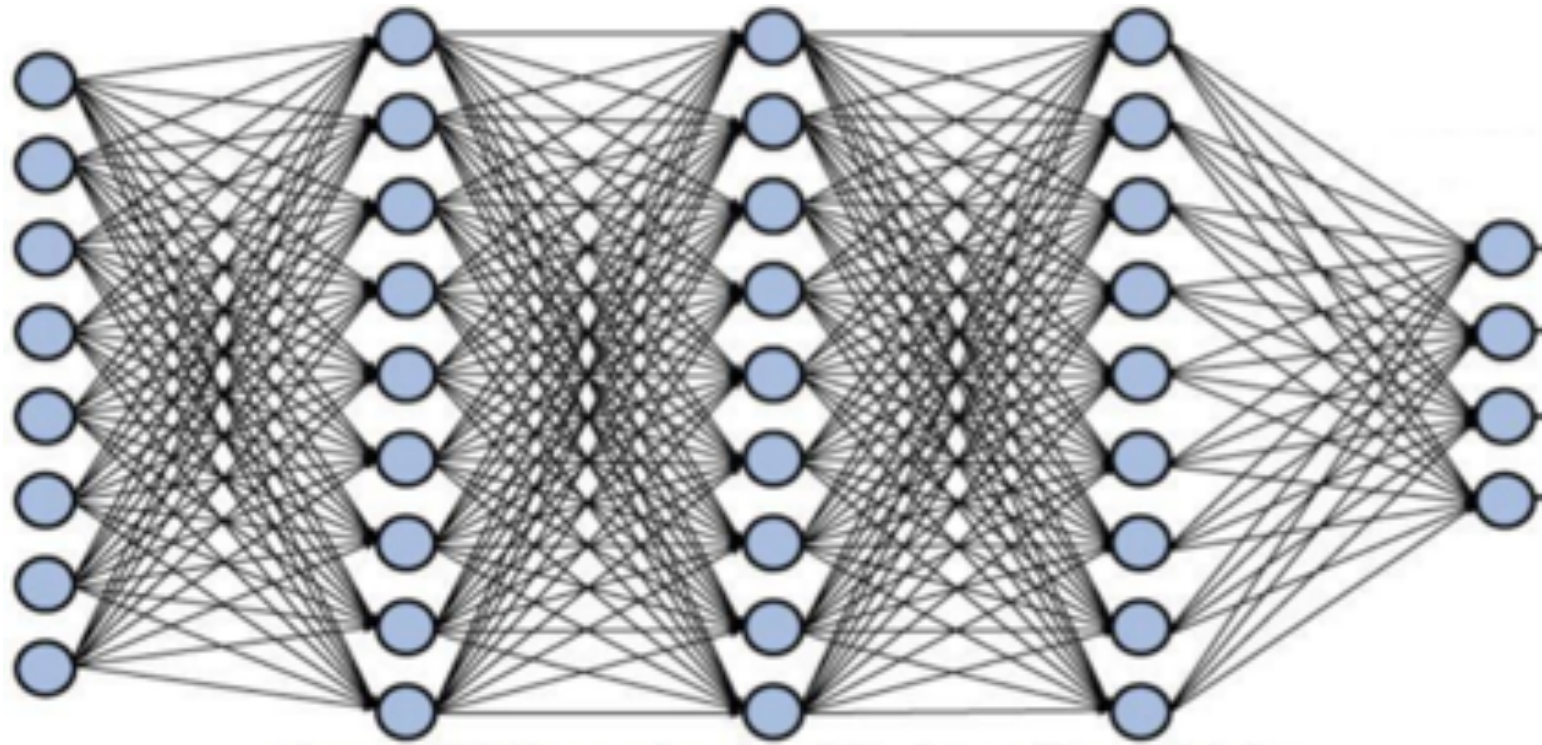


**TESTING / INFERENCE**

*Use learned network parameters*

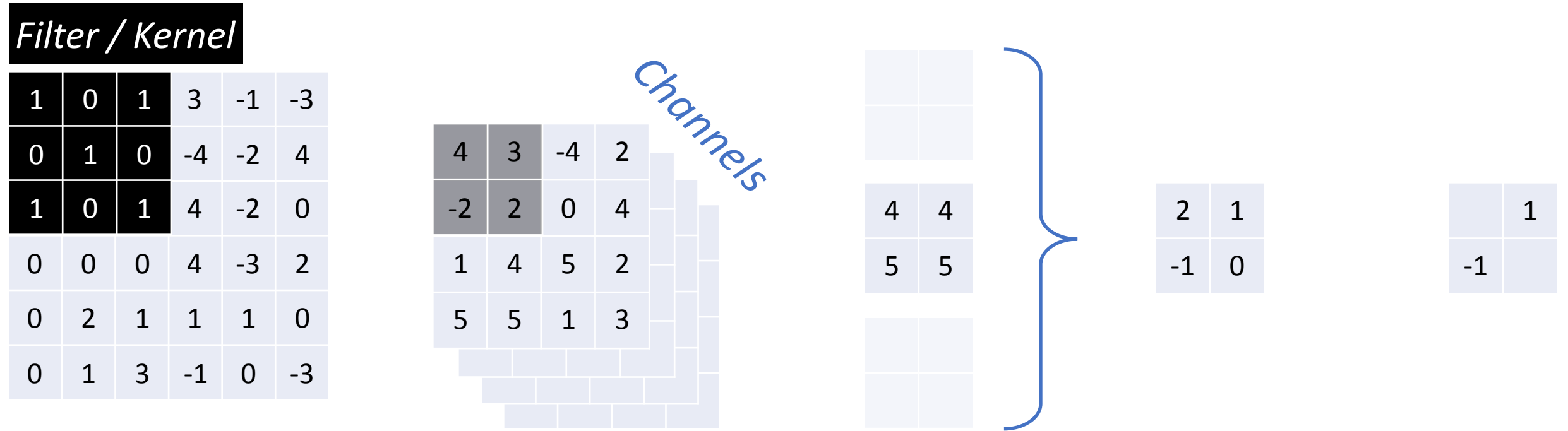
*Measure accuracy performance — % of correctly classified test samples*

# Types of NNs – Multilayer Perceptron (MLP)



*Fully connected – every node connects to every adjacent node*

# Types of NNs – Convolutional Neural Network (CNN)



*Convolution*

technically it's correlation... but since when have scientists been good at nomenclature?

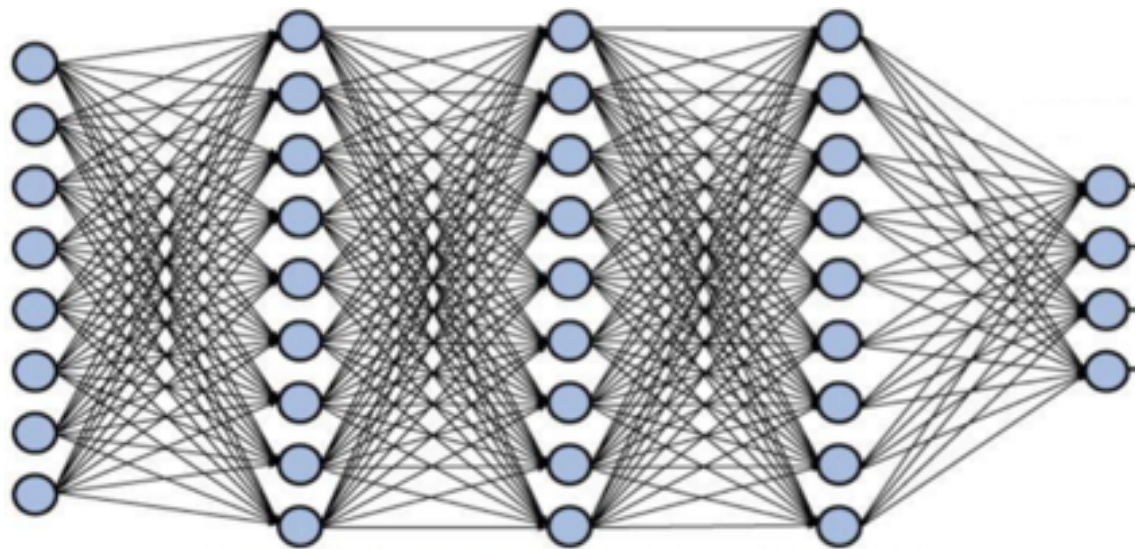
*Pooling  
(Downsampling)*

*Batch  
Normalization*

*Dropout*

# The Complexity Conundrum...

*Modern neural networks suffer from parameter explosion*



Training can take weeks on CPU  
Cloud GPU resources are expensive



Google Cloud Platform

He 2016



# ... and the Design Conundrum

- Deep neural networks have a lot of **hyperparameters**

- How many layers? *Architecture*
- How many neurons? *Hyperparameters*
- Learning rate *Training*
- Batch size *Hyperparameters*
- and more...



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

# AutoML (Automated Machine Learning)

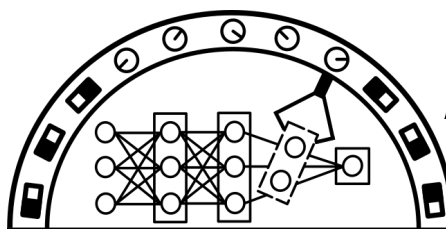
- Software frameworks that make design decisions
- Given problem specifications, **search** for NN models



Jin 2019 – Auto-Keras



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



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

Development cycle: July 2019 – April 2020

Currently built using Pytorch

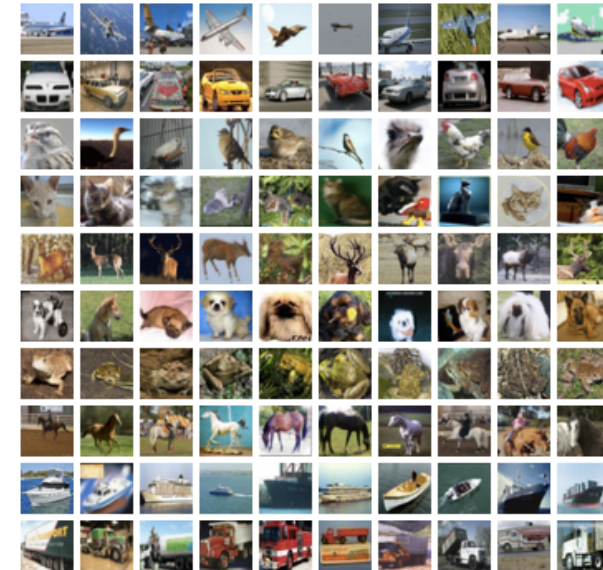
Ongoing extension to Keras (Tensorflow)

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>

Tested on image and textual datasets:

- CIFAR-10, CIFAR-100
- MNIST, Fashion MNIST
- Reuters RCV1 news corpus





Approach

# Search Objective

*Optimize performance and complexity*

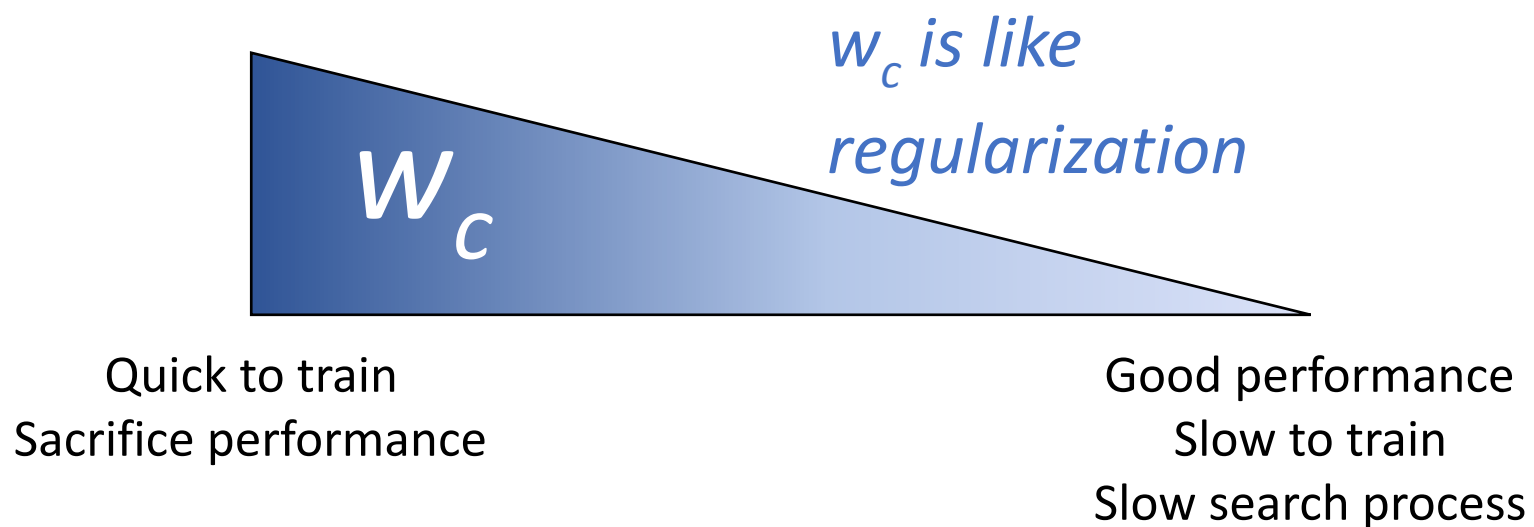
Modified loss function:  $f(\text{NN Config } \mathbf{x}) = \log(f_p + w_c * f_c)$

Example config  $\mathbf{x}$ :

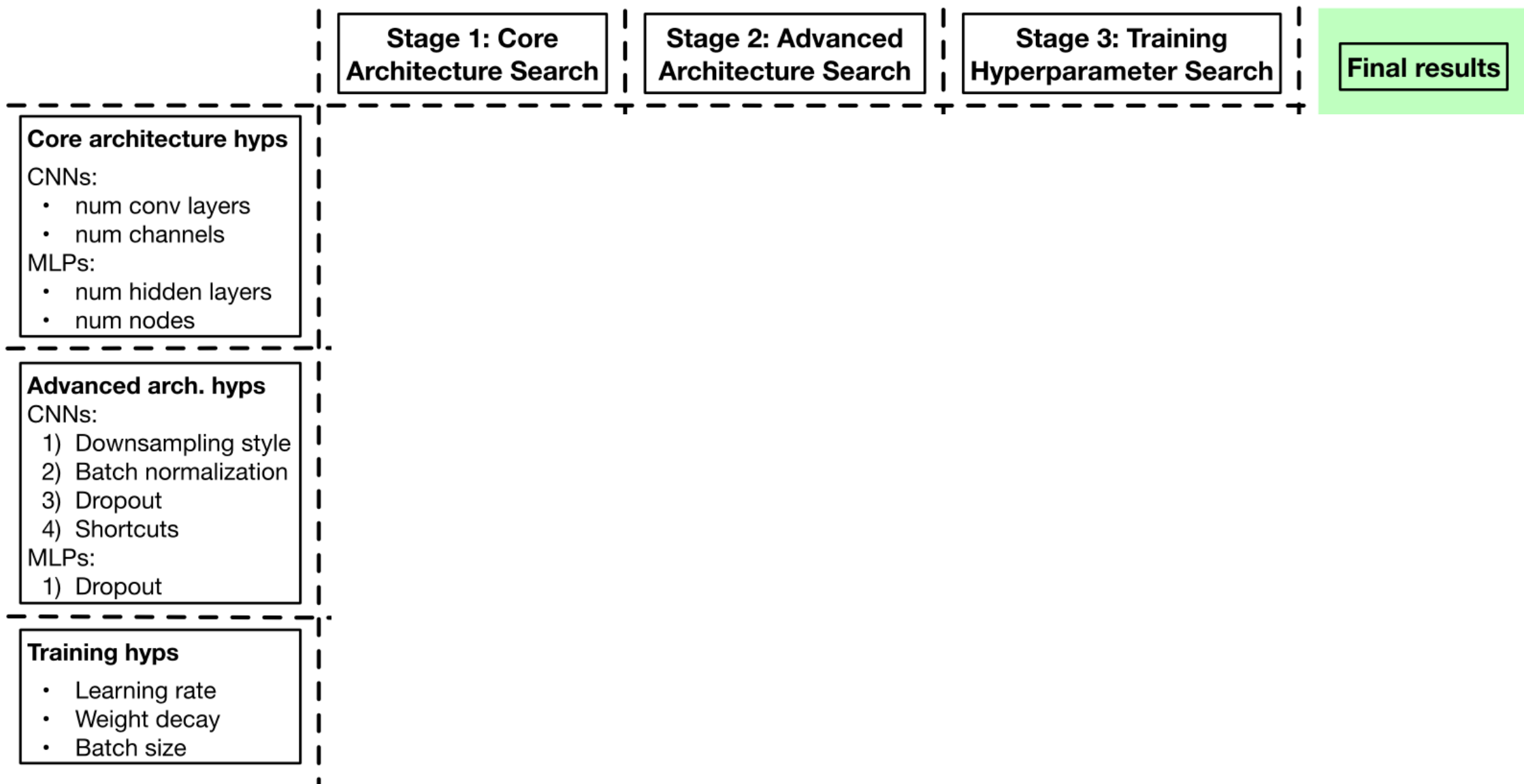
[#layers, #channels] = [3, (29,40,77)]

$f_p = 1 - (\text{Best Validation Accuracy})$

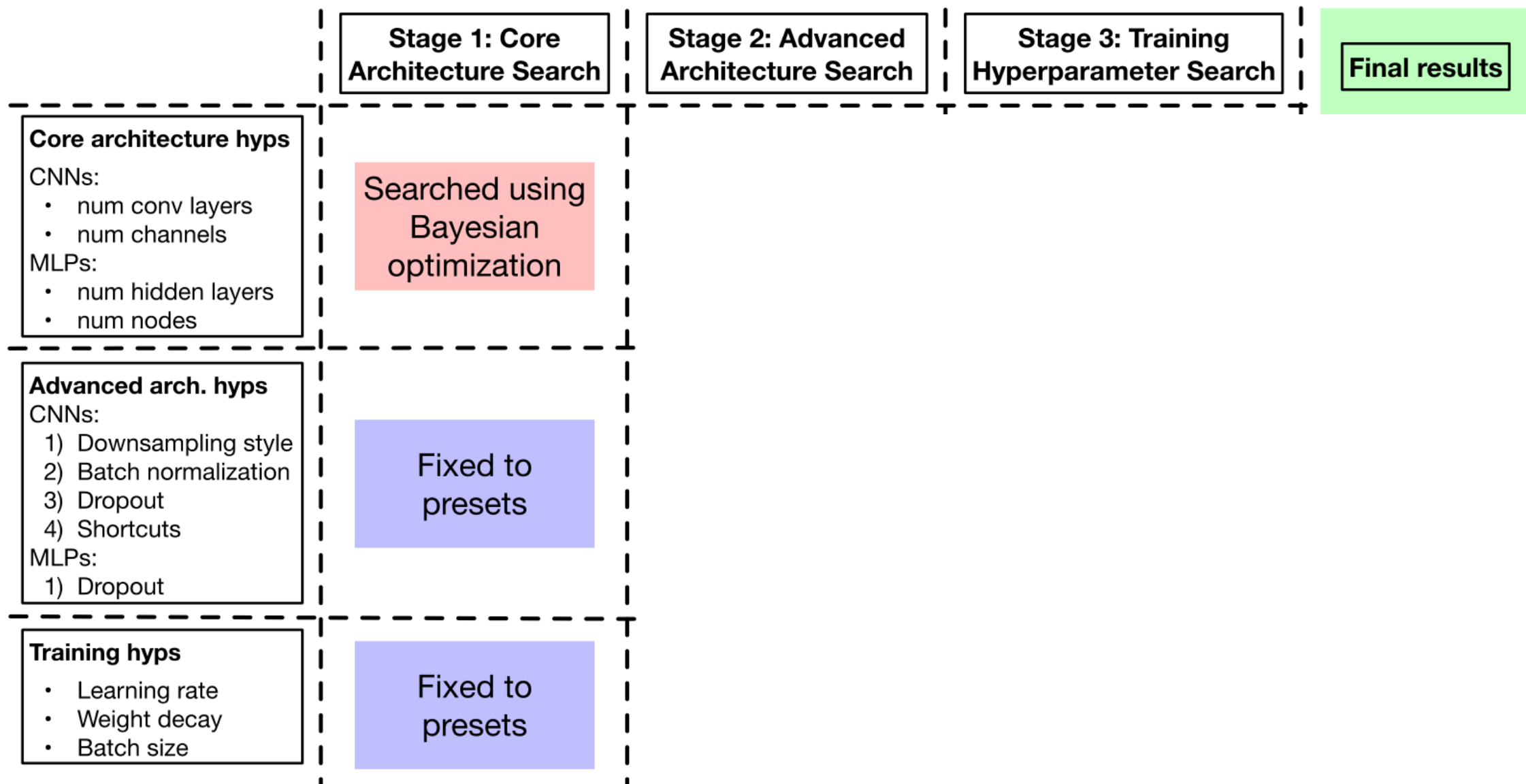
$f_c = \text{Normalized } t_{tr} \text{ or } N_p$   
 $= t_{tr}(\text{config}) / t_{tr}(\text{baseline})$



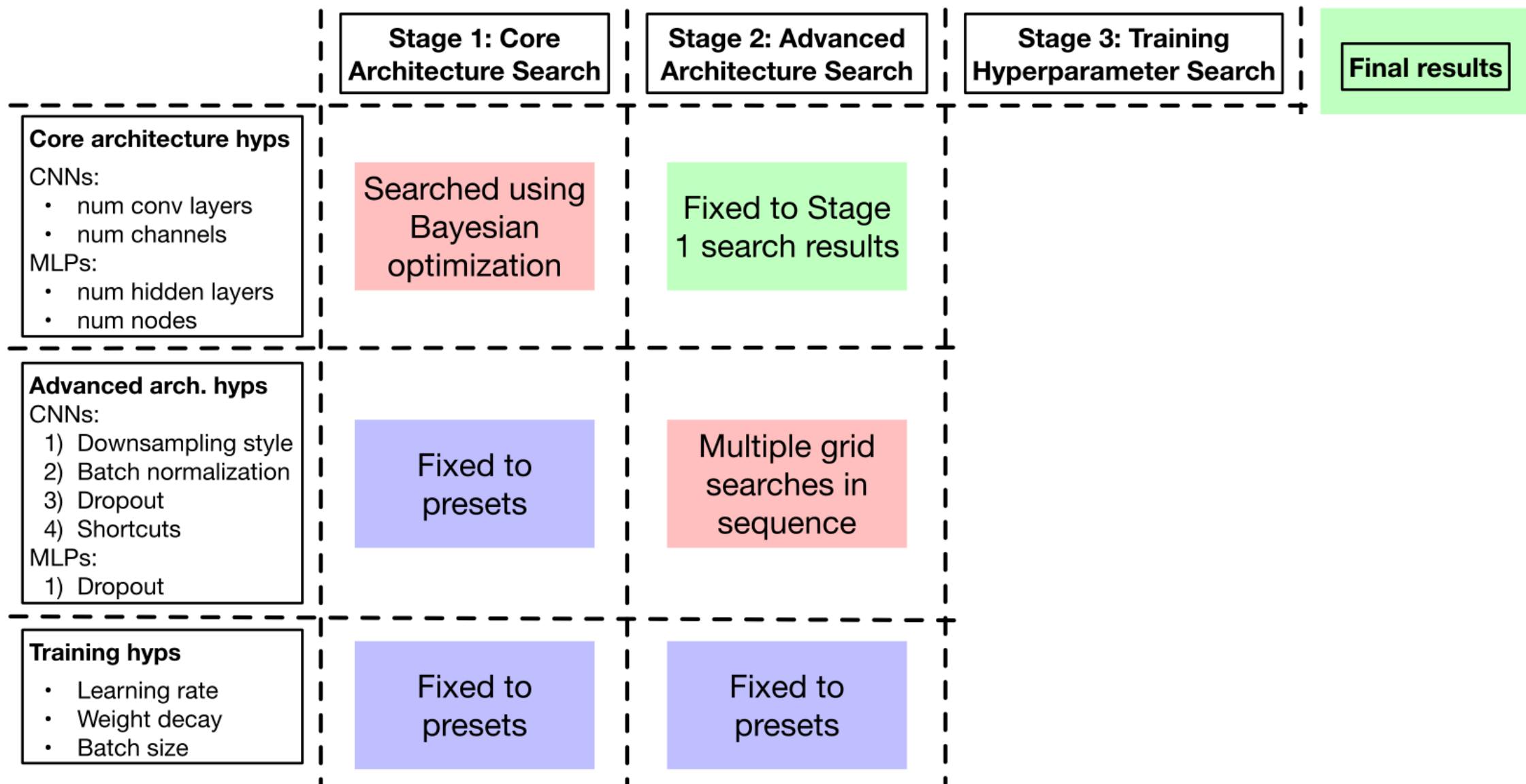
# Three-stage search process



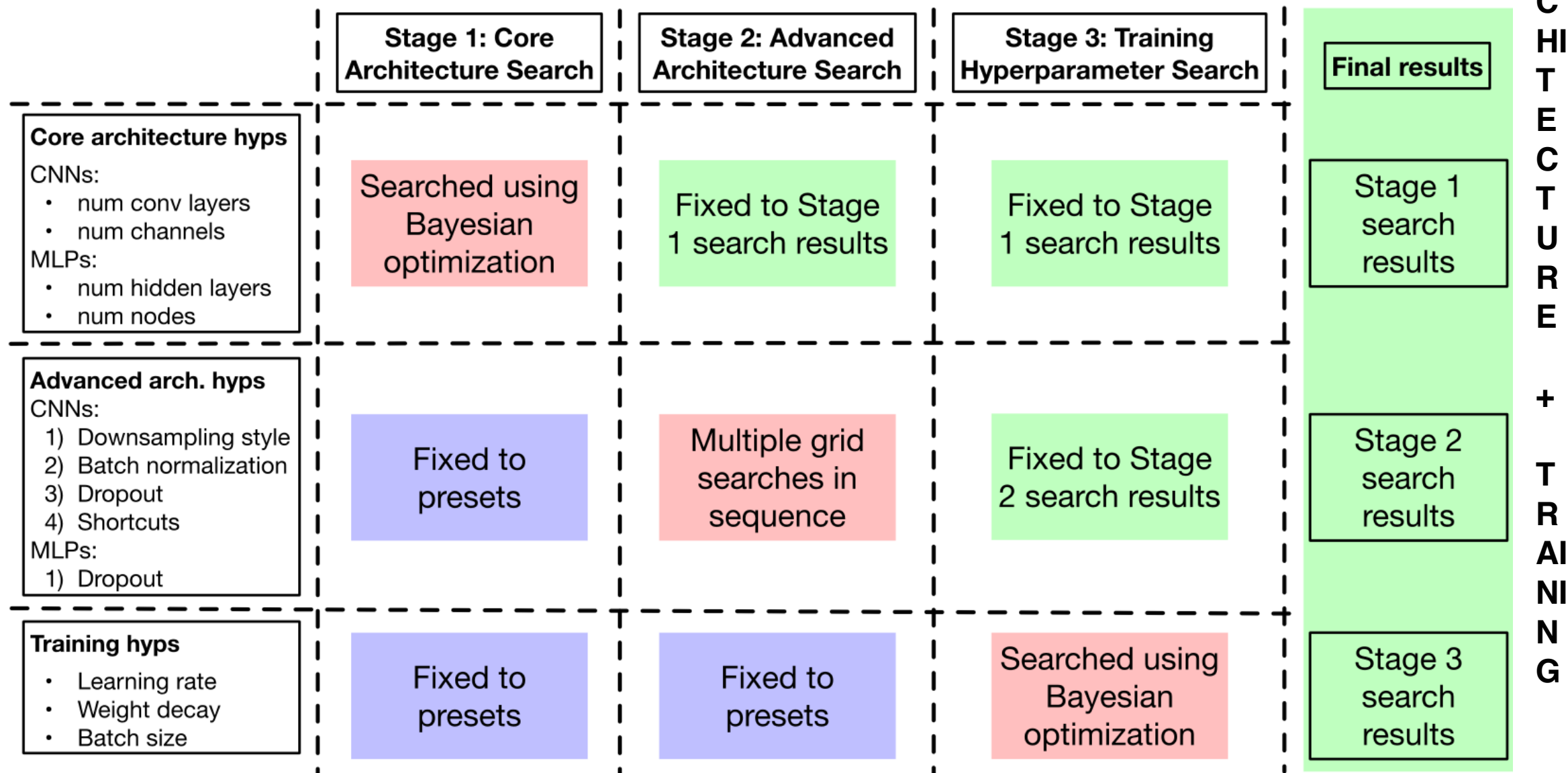
# Three-stage search process



# Three-stage search process



# Three-stage search process



# Bayesian Optimization – Gaussian process

$$f(\mathbf{X}_{1:n}) \sim \mathcal{N} \left( \begin{array}{c} \boldsymbol{\mu} \\ n \times 1 \end{array}, \begin{array}{c} \boldsymbol{\Sigma} \\ n \times n \end{array} \right)$$

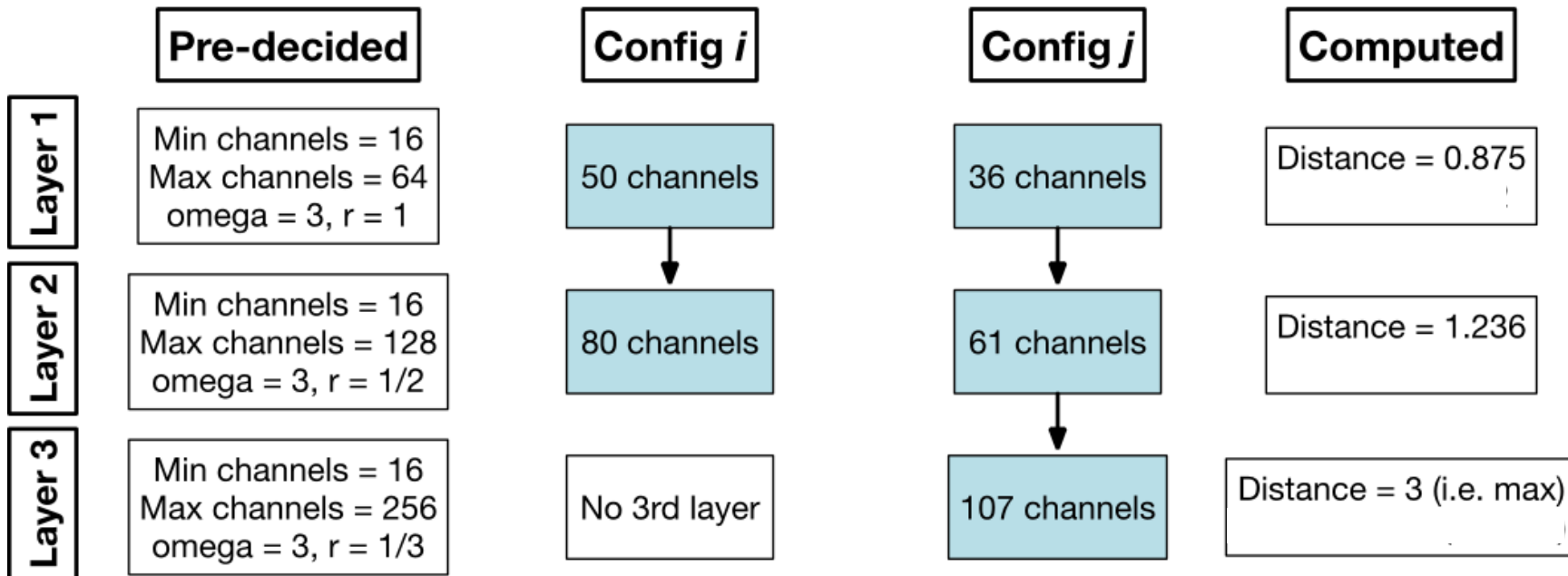
$$\boldsymbol{\mu} = \begin{bmatrix} \mu(\mathbf{x}_1) \\ \vdots \\ \mu(\mathbf{x}_n) \end{bmatrix} \quad \boldsymbol{\Sigma} = \begin{bmatrix} \sigma(\mathbf{x}_1, \mathbf{x}_1) & \cdots & \sigma(\mathbf{x}_1, \mathbf{x}_n) \\ \vdots & \ddots & \vdots \\ \sigma(\mathbf{x}_n, \mathbf{x}_1) & \cdots &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



# Covariance kernel – Similarity between NN configs

Individual  
Distance

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



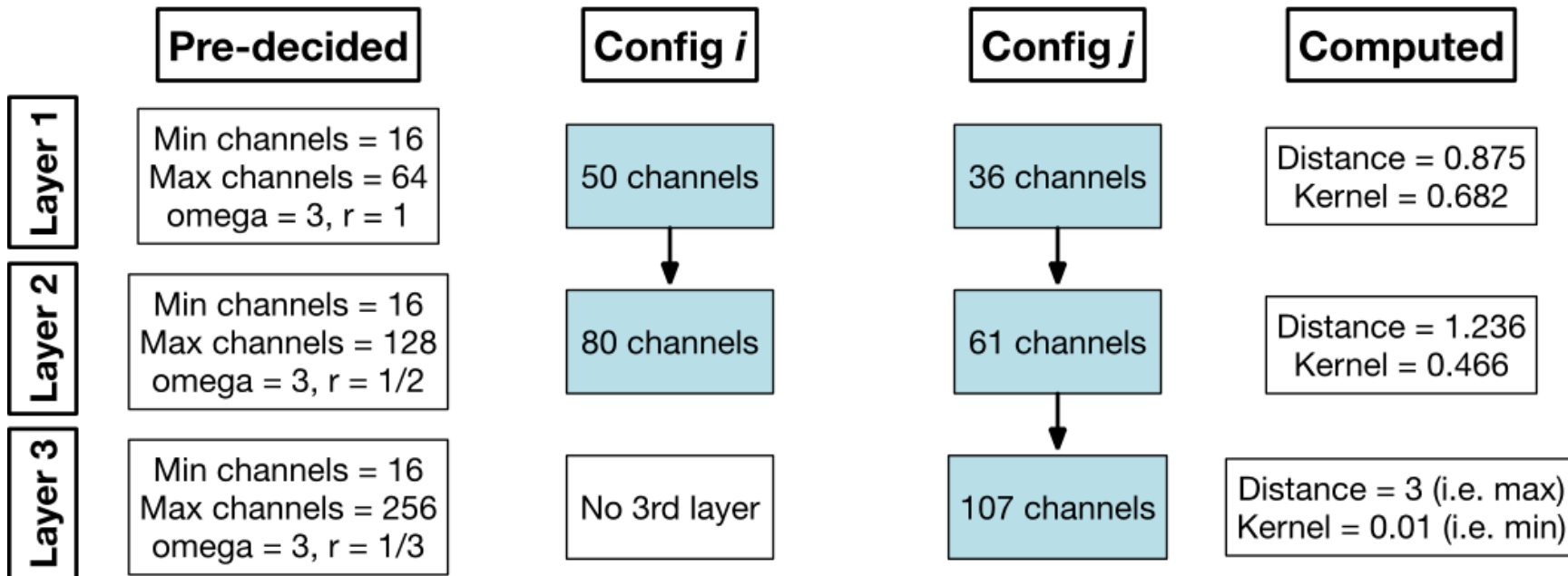
# Covariance kernel – Similarity between NN configs

Individual  
Distance

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

Individual  
Kernel

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



# Covariance kernel – Similarity between NN configs

Individual Distance

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

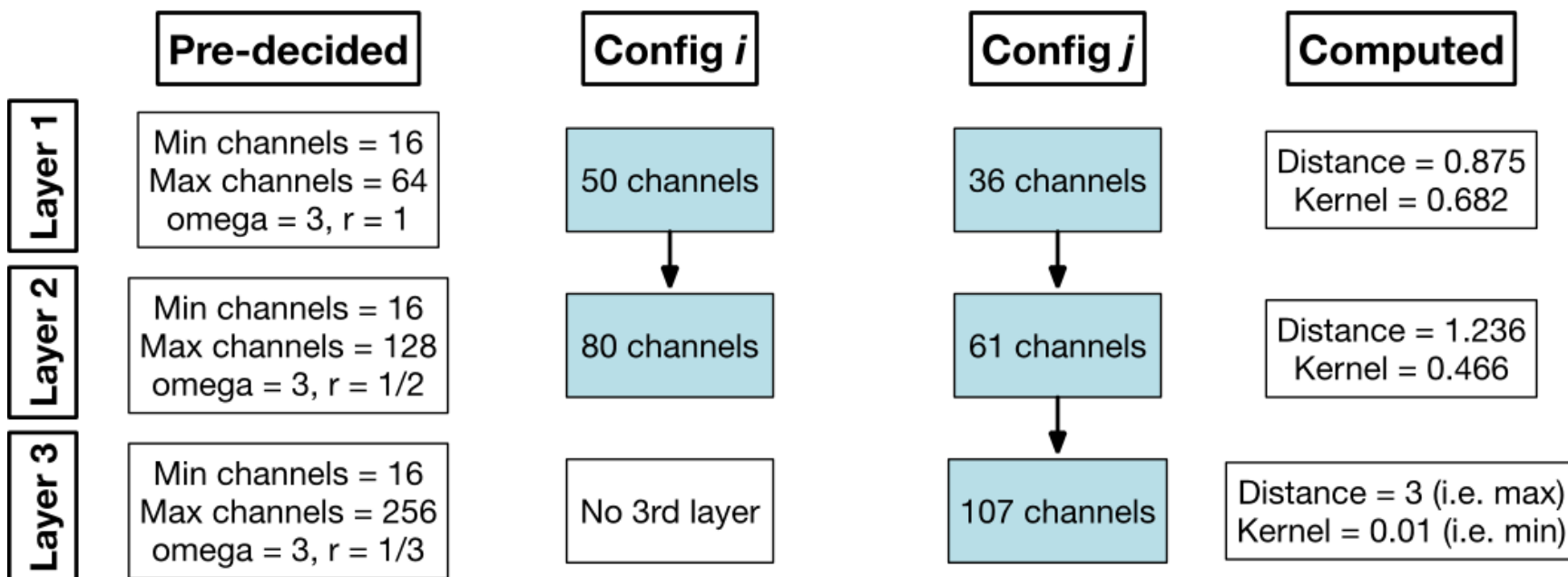
Individual Kernel

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

Complete Kernel

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

Convex combination



Assuming all {s} are equal, **final kernel value = 0.386**

# Bayesian Optimization – Expected Improvement

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

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



# Results

# Deep-n-Cheap in action!

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

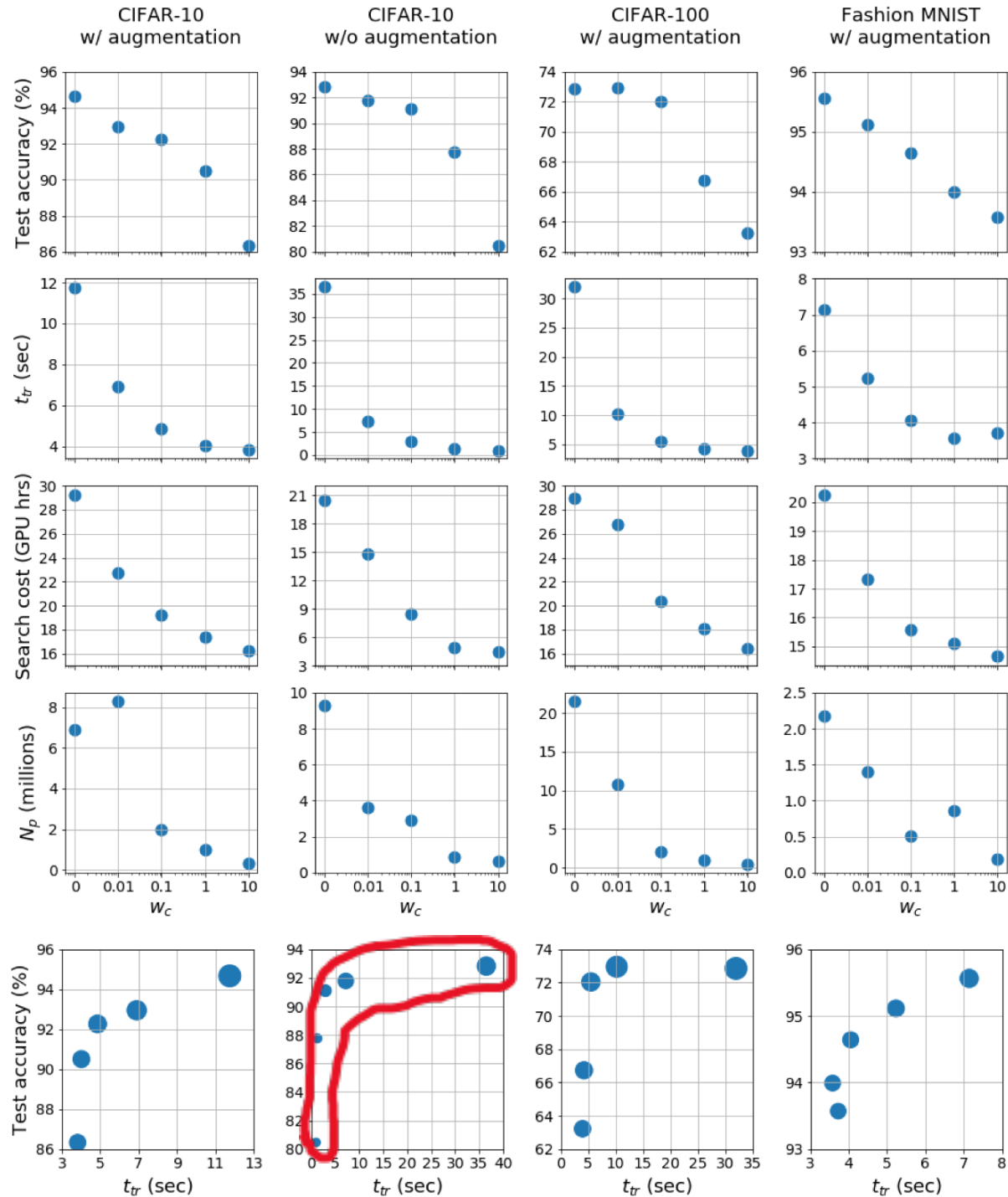
# CNN Results

*Complexity Penalty =  
Training time / epoch*

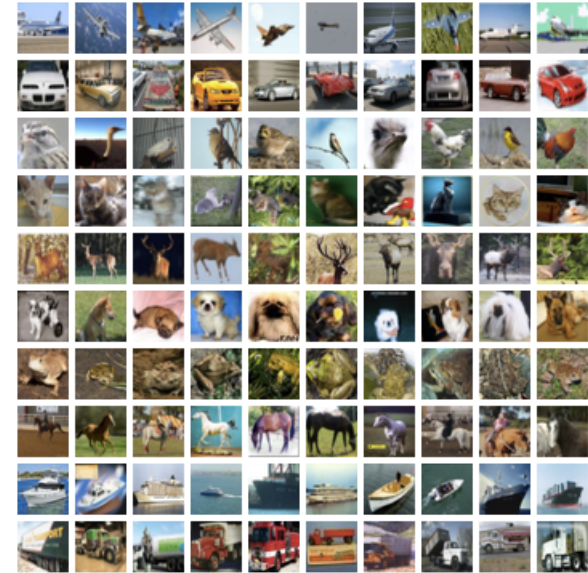
*AWS p3.2xlarge  
with 1 V100 GPU*

We are not penalizing  
this, but it's correlated

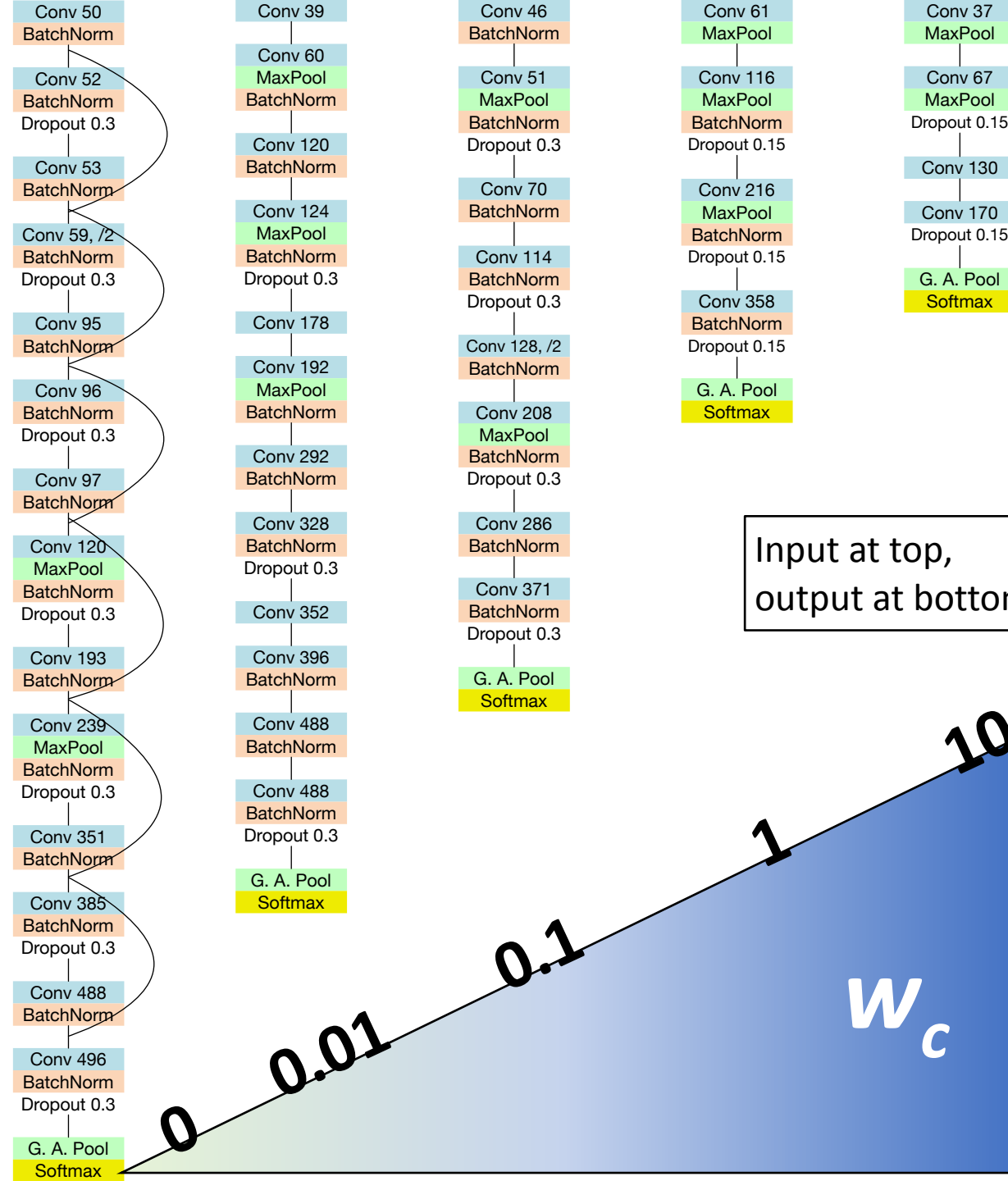
*Performance-  
complexity  
tradeoff*



# CIFAR-10 w/ aug



Input at top,  
output at bottom



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

$\lambda$  strictly correlated with  $N_p$

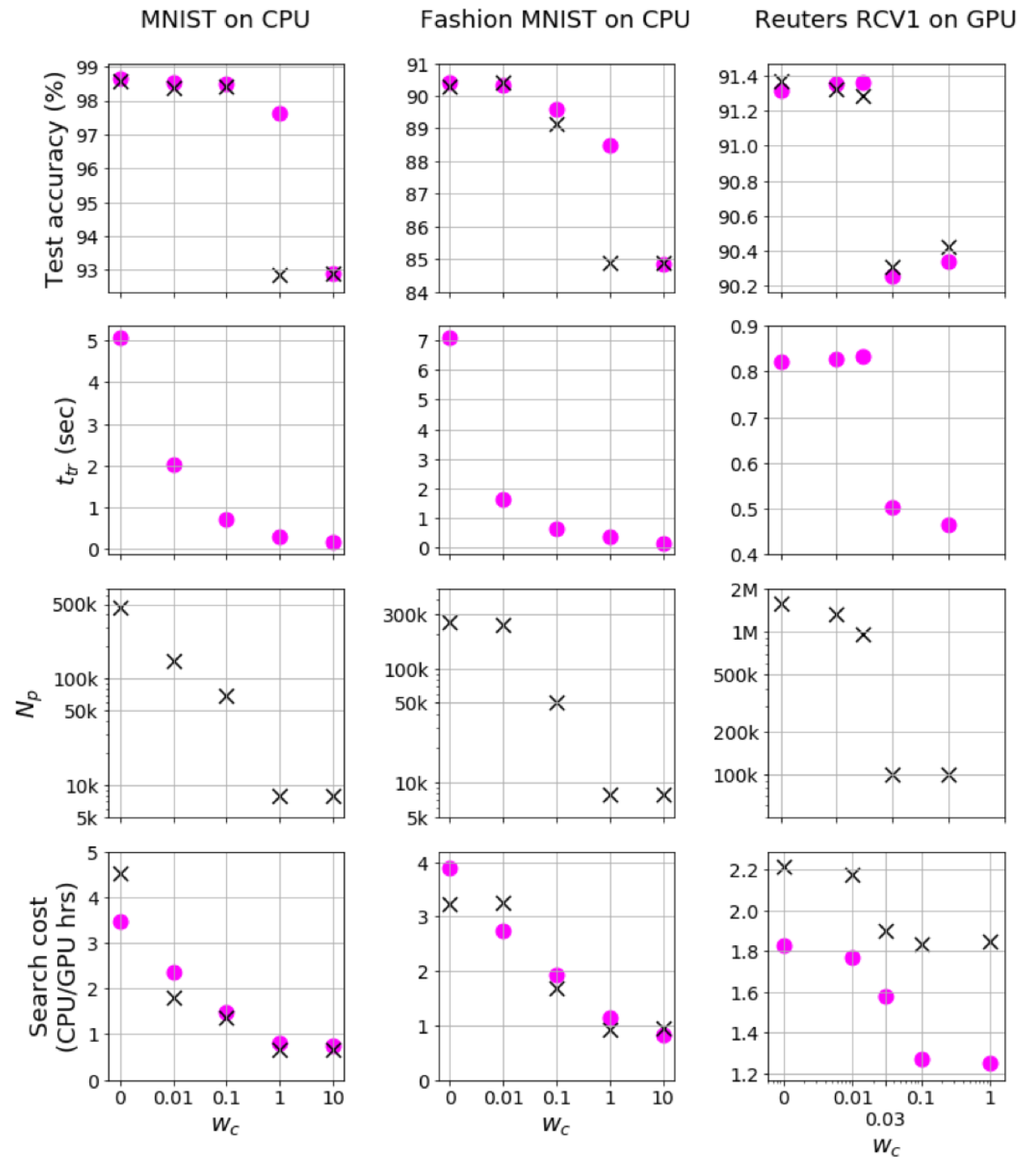


# MLP Results

*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	<b>3</b>	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	<b>93.74</b>
	$w_c = 0.1$	19.23	8 conv layers	4	459	91.89
	$w_c = 10$	16.23	4 conv layers	<b>3</b>	256	83.82

Penalizes inference complexity, not 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

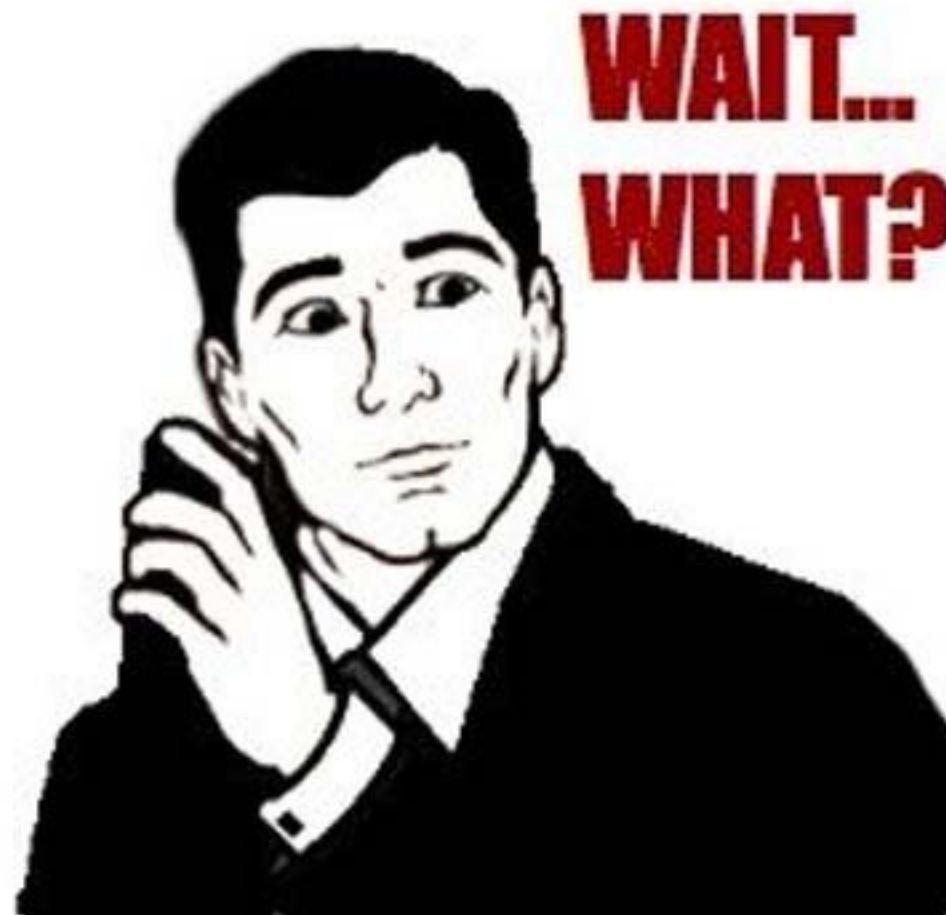
# Comparison (MLPs)

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	<b>91</b>
	‘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
	$w_c = 10$	<b>0.3</b>	1	<b>7.9k</b>	<b>0.1</b>	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	<b>7.9k</b>	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	<b>91.36</b>
	$w_c = 1$	<b>1.25</b>	1	<b>100k</b>	<b>0.4</b>	512	90.34
Deep-n-Cheap (penalize $N_p$ )	$w_c = 0$	2.22	2	1.6M	0.6	512	<b>91.36</b>
	$w_c = 1$	1.85	1	<b>100k</b>	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/>

