

Deep-n-Cheap

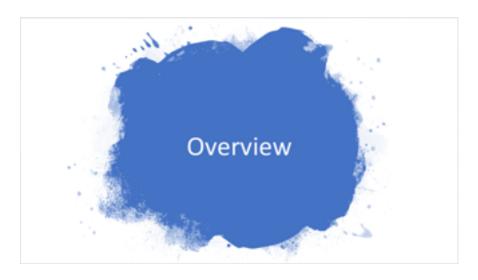
An Automated Search Framework for Low Complexity Deep Learning

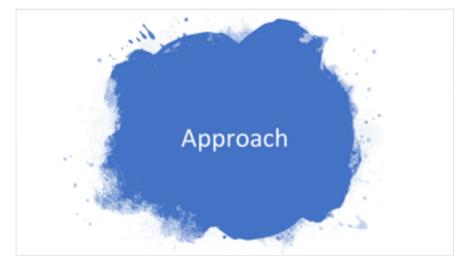
Sourya Dey

PhD, University of Southern California

April 24th, 2020

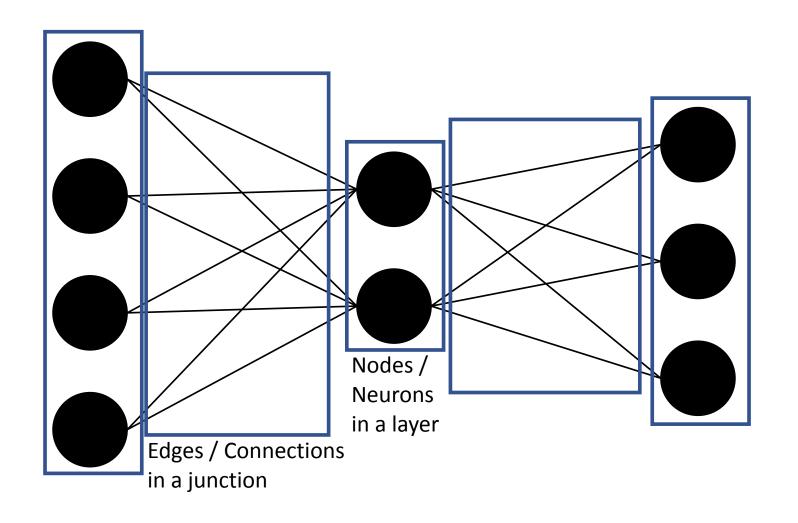
Outline

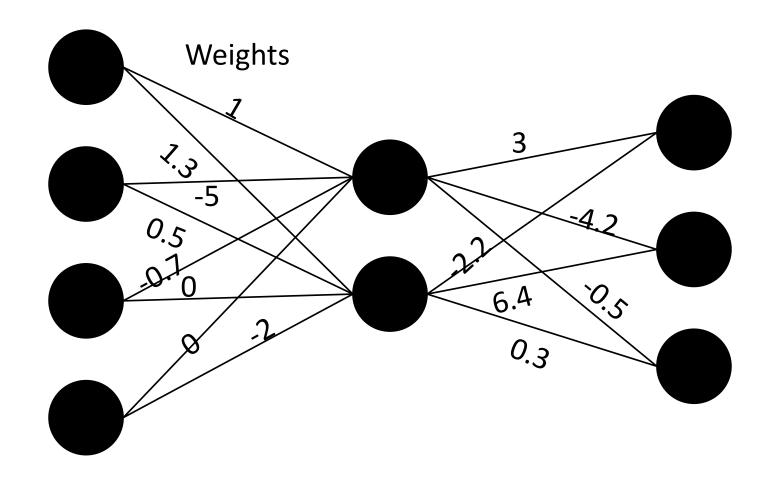


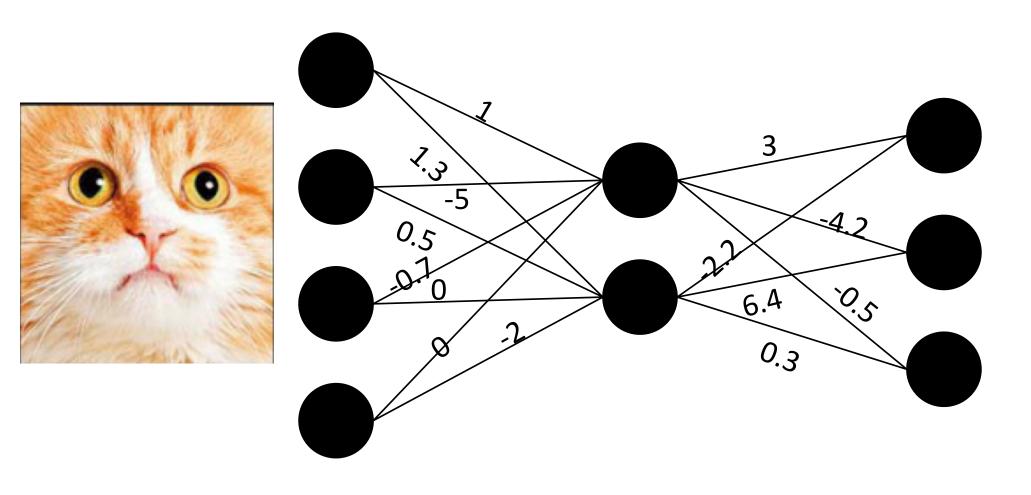


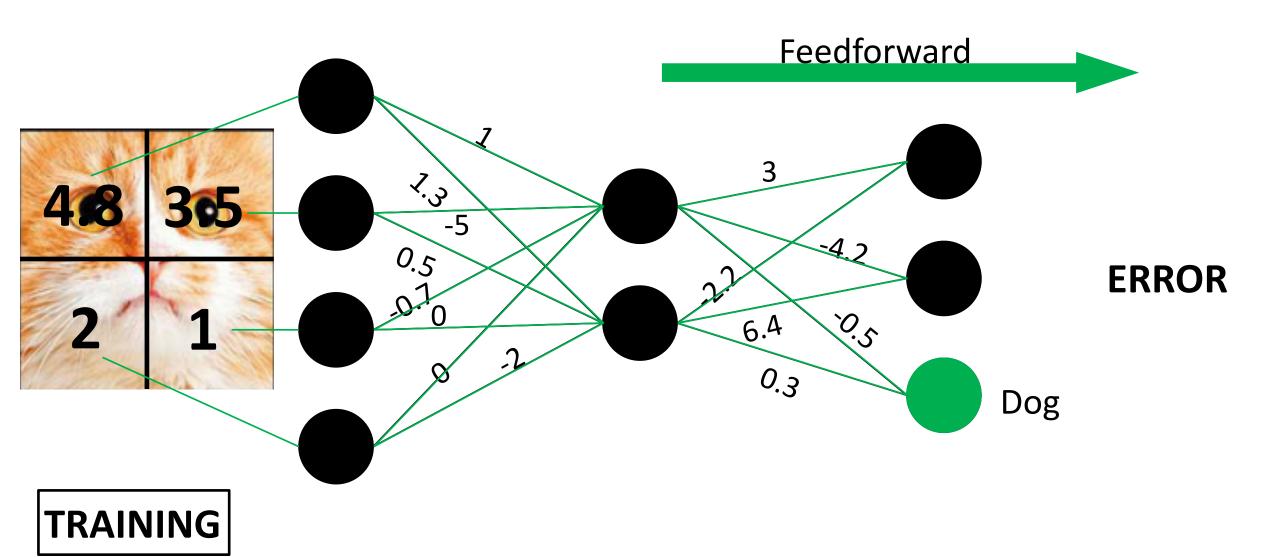




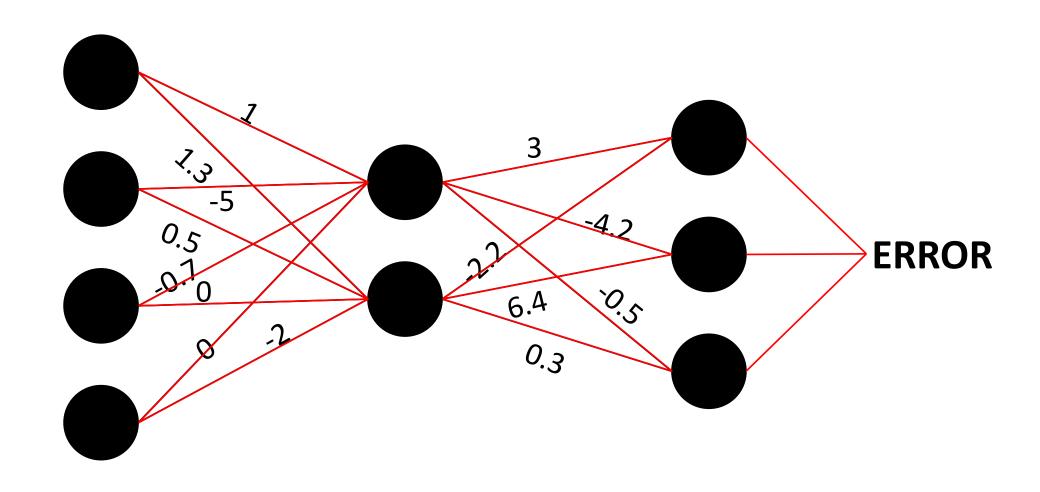






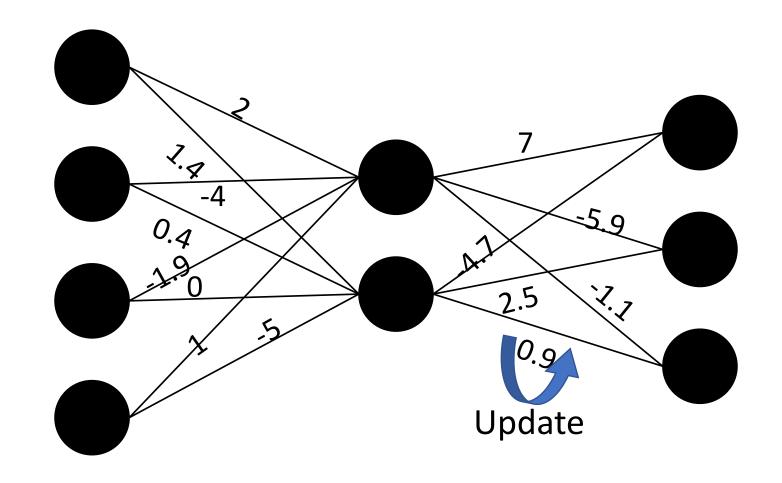


Learn network parameters — weights



TRAINING

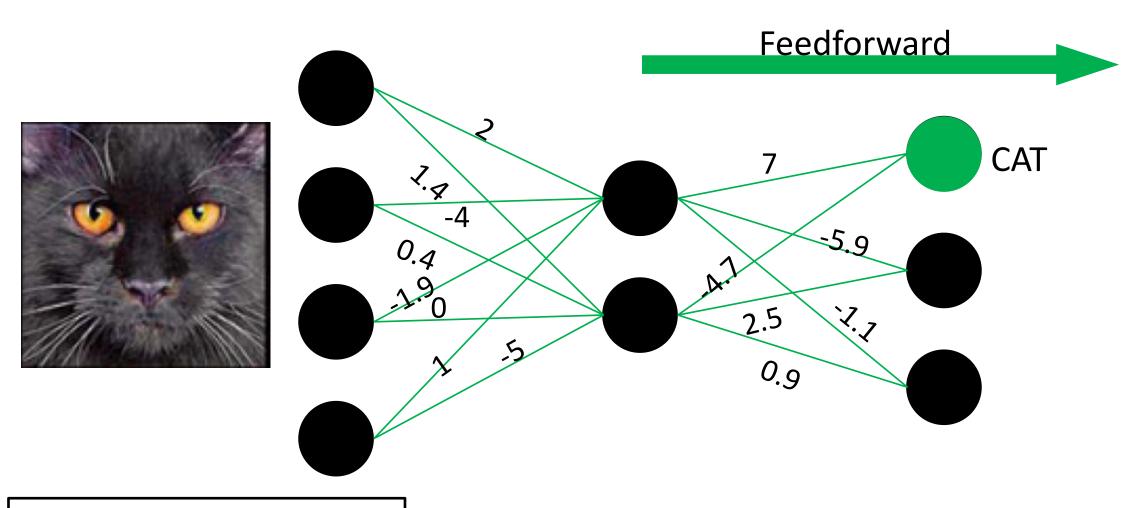
Backpropagation



TRAINING

Learn network parameters — weights

NNs can be used for classification

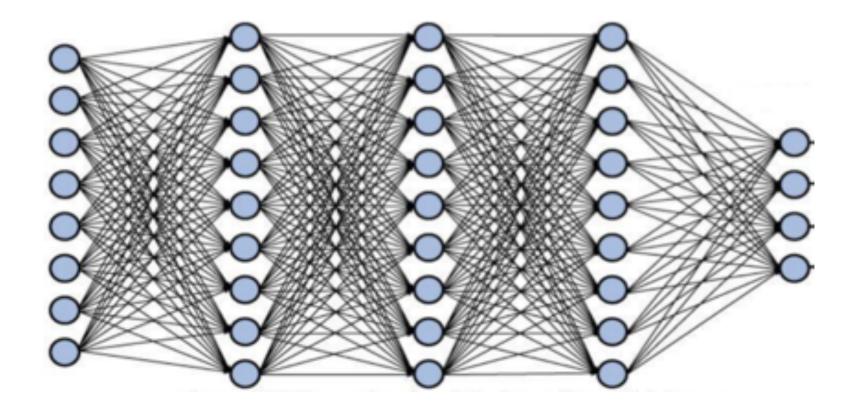


TESTING / INFERENCE

Use learned network parameters

Measure accuracy performance — % of correctly classified test samples

Types of NNs – Multilayer Perceptron (MLP)

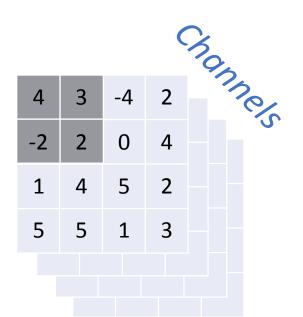


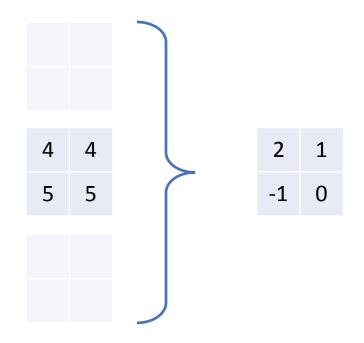
Fully connected – every node connects to every adjacent node

Types of NNs – Convolutional Neural Network (CNN)

Filter / Kernel

1		1	2		2
1	0	1	3	-1	-3
0	1	0	-4	-2	4
1	0	1	4	-2	0
0	0	0	4	-3	2
0	2	1	1	1	0
0	1	3	-1	0	-3





1

Convolution

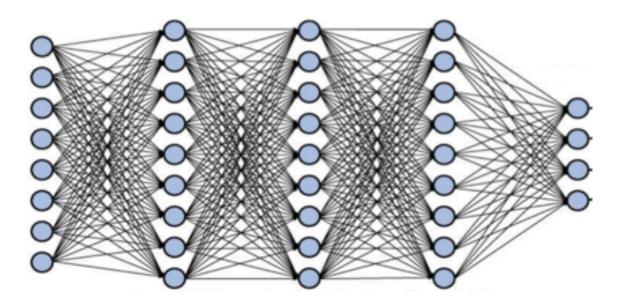
technically it's correlation... but since when have scientists been good at nomenclature?

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	Hynernarameters



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





Our Work



Low Complexity AutoML framework

Reduce training complexity

Target custom datasets and user requirements

Output complete training configs

Framework	Architecture search space	Training	Adjust model	
Framework	Architecture search space	hyp search	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 $t_{\rm tr}, N_p$	

 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)

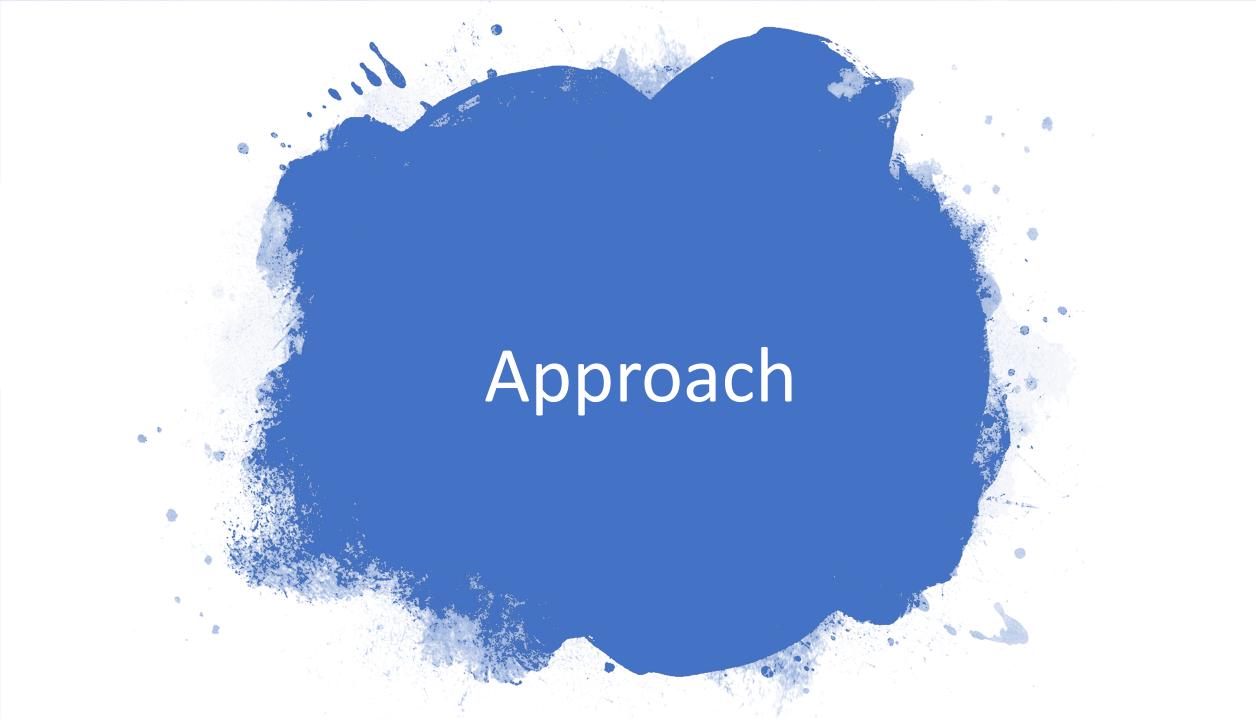
S. Dey, S. C. Kanala, K. M. Chugg and P. A. Beerel, "Deep-n-Cheap: An Automated Search Framework for Low Complexity Deep Learning", submitted to ECML-PKDD 2020.

https://arxiv.org/abs/2004.00974

Tested on image and textual datasets:

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





Search Objective

Optimize performance and complexity

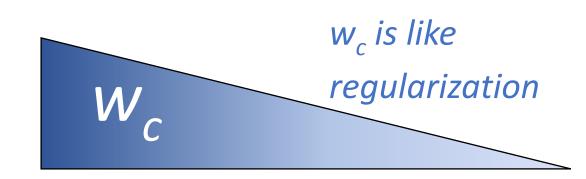
Modified loss function: $f(NN Config x) = log(f_p + w_c * f_c)$

Example config x:

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

$$f_p = 1$$
 - (Best Validation Accuracy)

$$f_c$$
 = Normalized t_{tr} or N_p



Quick to train Sacrifice performance Good performance
Slow to train
Slow search process

Stage 1: Core
Architecture Search

Stage 2: Advanced Architecture Search Stage 3: Training
Hyperparameter Search

Final results

Core architecture hyps

CNNs:

- · num conv layers
- num channels

MLPs:

- num hidden layers
- num nodes

Advanced arch. hyps

CNNs:

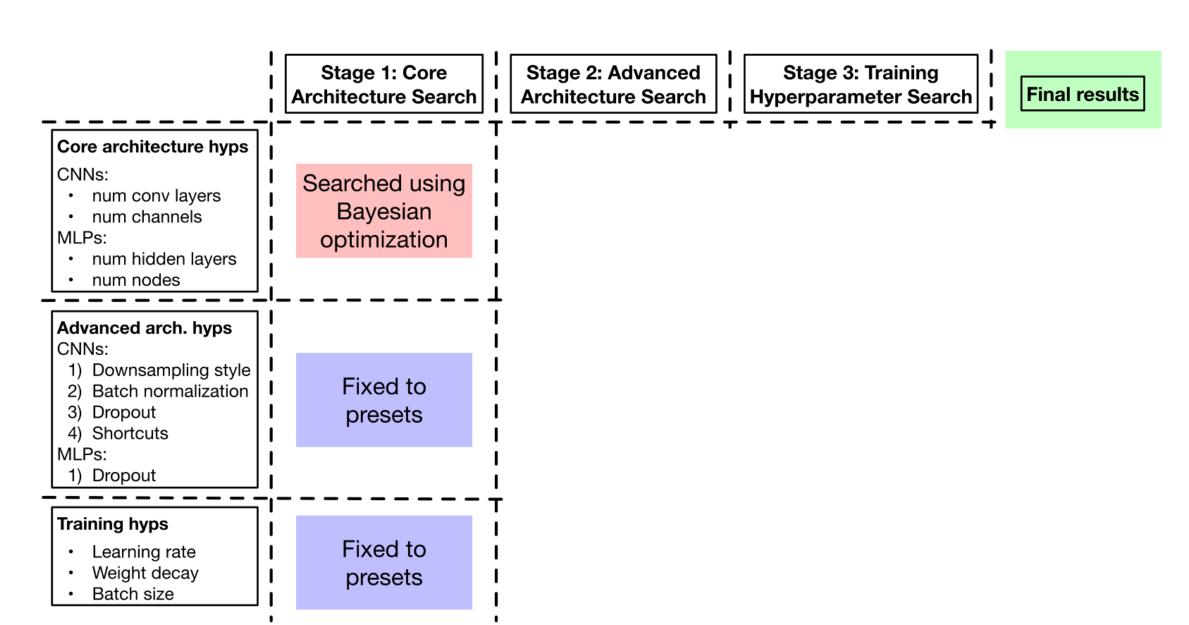
- 1) Downsampling style
- 2) Batch normalization
- 3) Dropout
- 4) Shortcuts

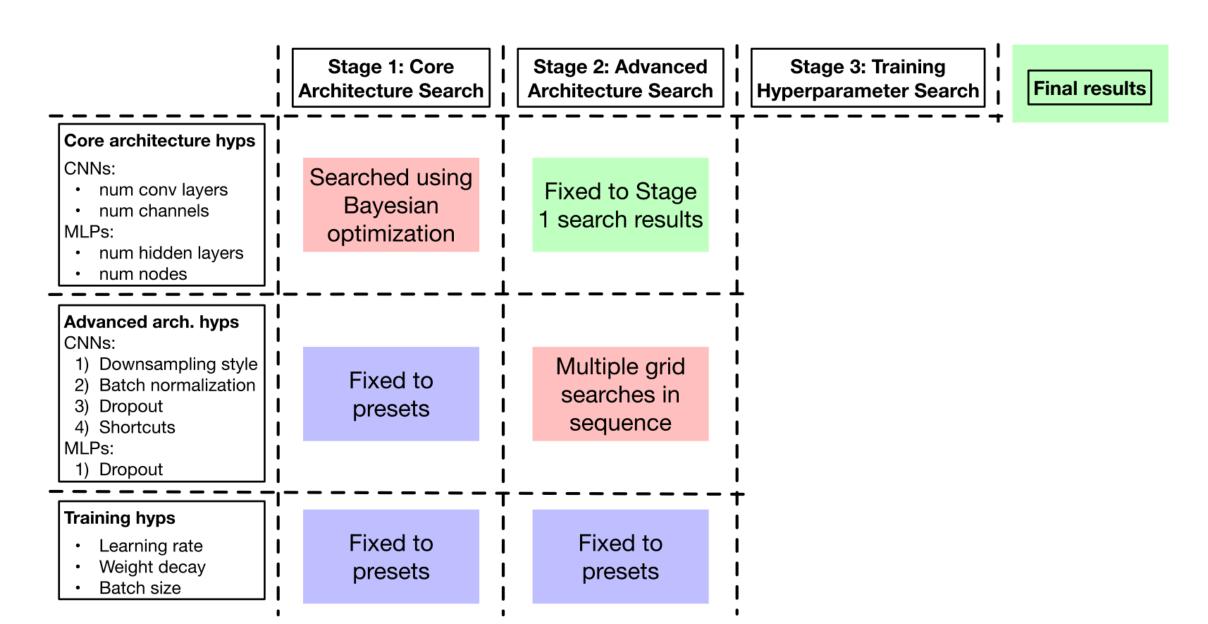
MLPs:

1) Dropout

Training hyps

- · Learning rate
- Weight decay
- Batch size





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

Bayesian Optimization – Gaussian process

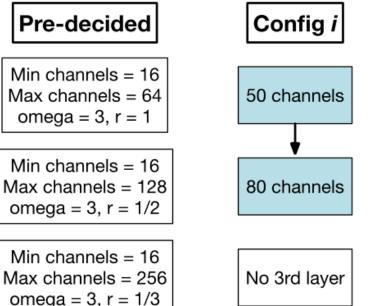
$$f\left(\boldsymbol{X}_{1:n}\right) \sim \mathcal{N}\left(\boldsymbol{\mu}, \boldsymbol{\Sigma}_{n \times 1}, \boldsymbol{n} \times n\right)$$

$$oldsymbol{\mu} = egin{bmatrix} \mu\left(oldsymbol{x}_1
ight) \ dots \ \mu\left(oldsymbol{x}_n
ight) \end{bmatrix}$$

Covariance kernel – Similarity between NN configs

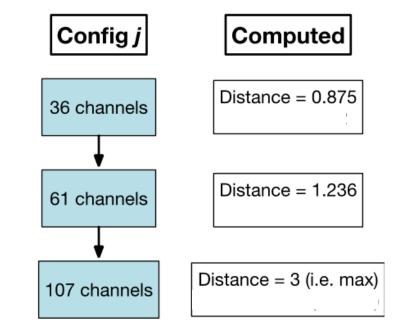
Individual Distance

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



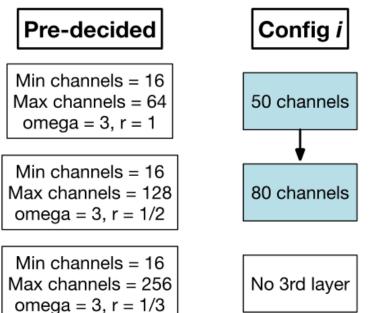
Layer 2

Layer 3



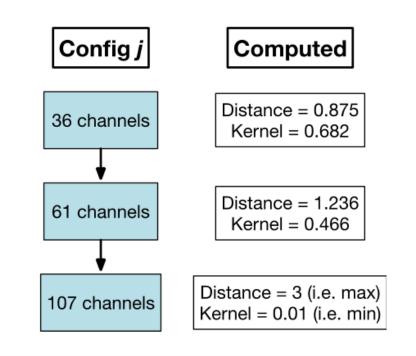
Covariance kernel – Similarity between NN configs

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



Layer 2

Layer 3



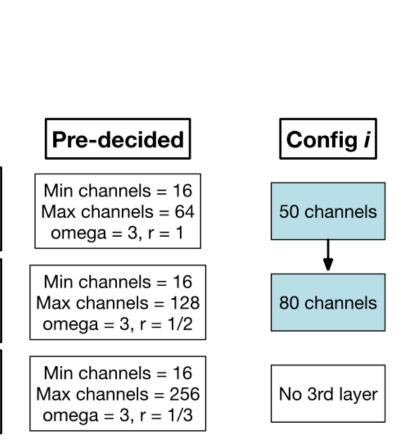
Kernel

Covariance kernel – Similarity between NN configs

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

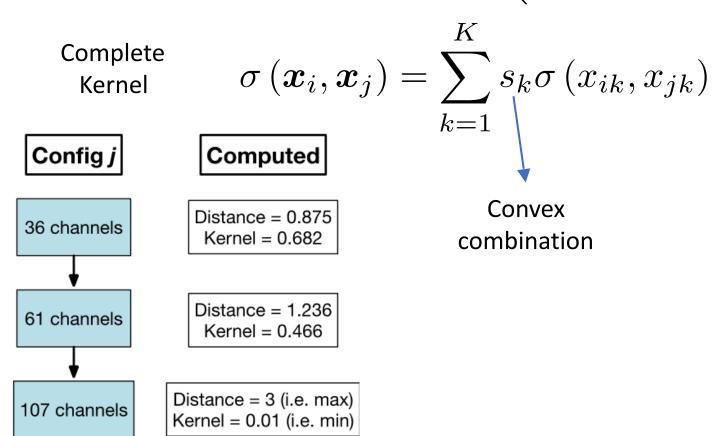
Individual Kernel

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



Layer 2

Layer 3



Convex combination

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

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 El(x)
- Can explore lots of points cheaply

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



Deep-n-Cheap in action!

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

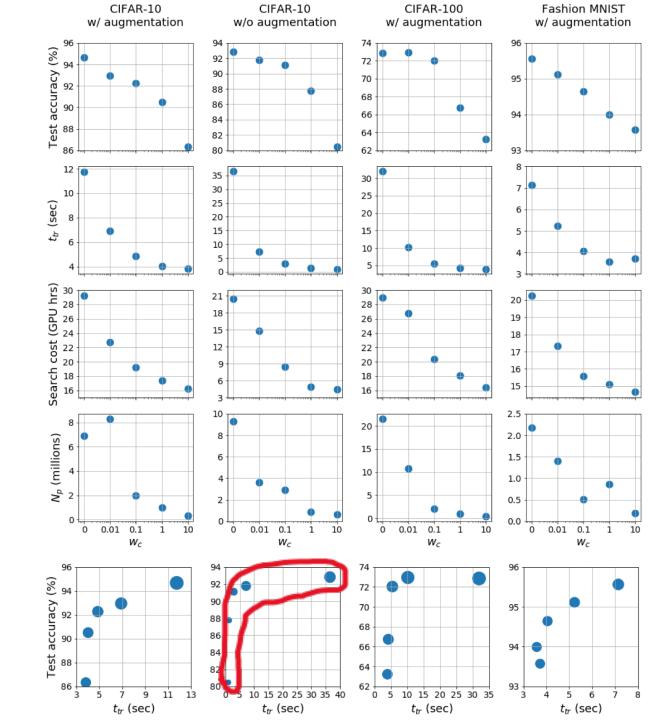
CNN Results

Complexity Penalty = Training time / epoch

AWS p3.2xlarge with 1 V100 GPU

We are not penalizing this, but it's correlated

Performancecomplexity tradeoff



Conv 39 Conv 46 Conv 61 Conv 50 Conv 37 MaxPool BatchNorm BatchNorm MaxPool Conv 60 Conv 52 MaxPool Conv 51 Conv 116 Conv 67 MaxPool MaxPool MaxPool BatchNorm BatchNorm Dropout 0.3 BatchNorm BatchNorm Dropout 0.15 Conv 120 Dropout 0.3 Dropout 0.15 BatchNorm Conv 53 Conv 130 Conv 70 BatchNorm Conv 216 Conv 124 BatchNorm MaxPool Conv 170 MaxPool Conv 59, /2 BatchNorm Dropout 0.15 BatchNorm Conv 114 BatchNorm Dropout 0.15 G. A. Pool Dropout 0.3 Dropout 0.3 BatchNorm Dropout 0.3 Conv 358 Softmax Conv 178 Conv 95 BatchNorm BatchNorm Conv 128, /2 Dropout 0.15 Conv 192 BatchNorm MaxPool G. A. Pool Conv 96 BatchNorm Softmax BatchNorm Conv 208 MaxPool Dropout 0.3 Conv 292 BatchNorm BatchNorm Dropout 0.3 Conv 97 BatchNorm Conv 328 Conv 286 Input at top, BatchNorm Conv 120 BatchNorm MaxPool Dropout 0.3 Conv 371 BatchNorm output at bottom Dropout 0.3 Conv 352 BatchNorm Dropout 0.3 Conv 193 Conv 396 BatchNorm BatchNorm G. A. Pool Softmax Conv 488 Conv 239 MaxPool BatchNorm BatchNorm Conv 488 Dropout 0.3 BatchNorm Dropout 0.3 Conv 351 BatchNorm G. A. Pool Softmax Conv 385 BatchNorm Dropout 0.3 Conv 488 BatchNorm Conv 496 BatchNorm Dropout 0.3 G. A. Pool Softmax

CIFAR-10 w/ aug



$\mathbf{w_c}$	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 x 10 ⁻⁵	8.3 x 10 ⁻⁵	1.2 x 10 ⁻⁵	0	0
Batch size	120	256	459	452	256

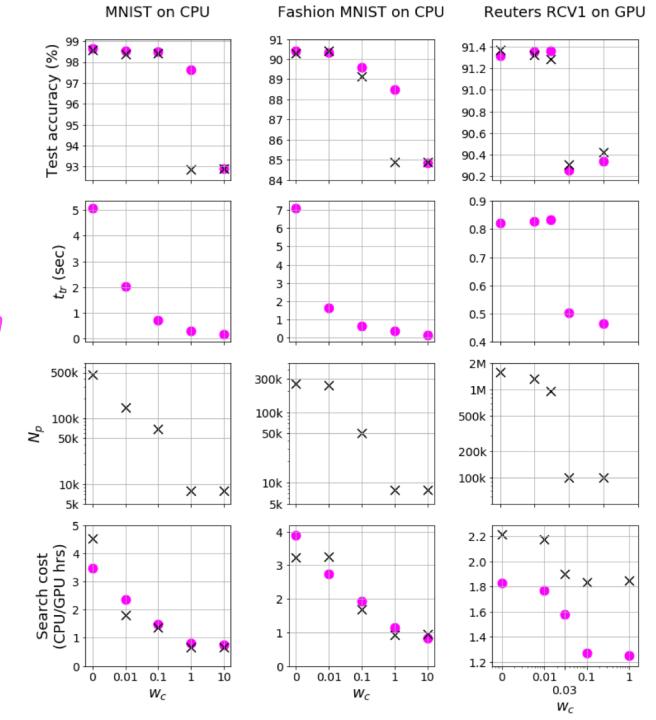
 λ strictly correlated with N_p

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	Search cost	Best model found from search			
Framework	settings	(GPU hrs)	Architecture	$t_{ m tr}~({ m sec})$	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 actoring during search instead of final test acc

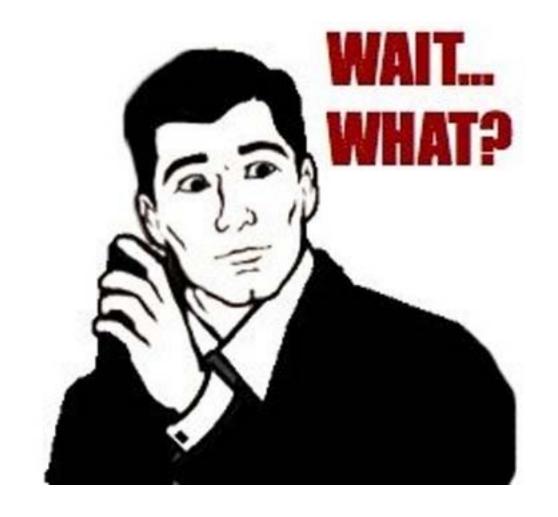
Comparison (MLPs)

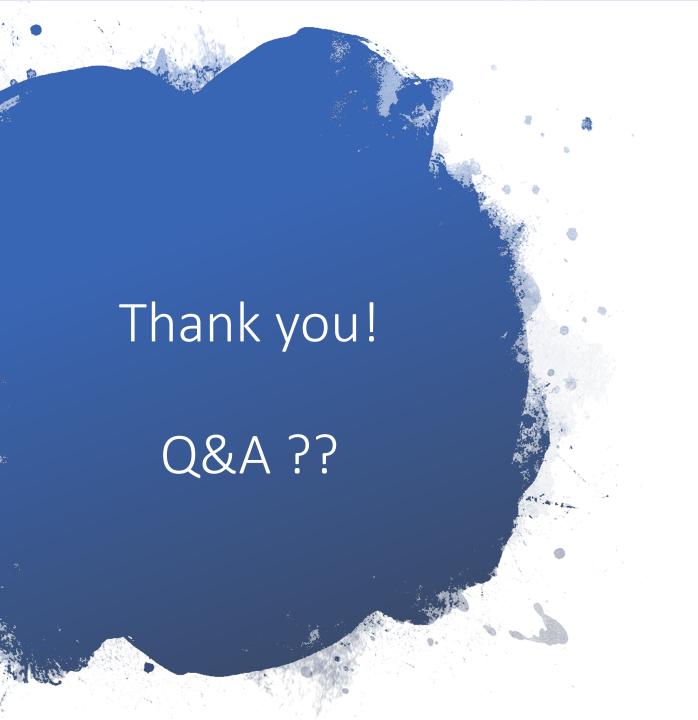
Duo mo orregula	Additional	Search cost		Best n	nodel for	ind from se	arch	
Framework	settings	(GPU hrs)	MLP layers	N_p	$t_{\rm tr}~({ m sec})$	Batch size	Best val acc (%)	
Fashion MNIST								
	'tiny cs'	6.76	50	27.8M	19.2	125	91	
Auto-Pytorch	'medium cs'	5.53	20	3.5M	8.3	184	90.52	
	'full cs'	6.63	12	122k	5.4	173	90.61	
Deep-n-Cheap	$w_c = 0$	0.52	3	263k	0.4	272	90.24	
(penalize $t_{\rm tr}$)	$w_c = 10$	0.3	1	7.9k	0.1	511	84.39	
Deep-n-Cheap	$w_c = 0$	0.44	2	317k	0.5	153	90.53	
(penalize N_p)	$w_c = 10$	0.4	1	7.9k	0.2	256	86.06	
			Reuters RC	V1				
Auto-Pytorch	'tiny cs'	7.22	38	19.7M	39.6	125	88.91	
Auto-i ytorch	'medium cs'	6.47	11	11.2M	22.3	337	90.77	
Deep-n-Cheap	$w_c = 0$	1.83	2	1.32M	0.7	503	91.36	
(penalize $t_{\rm tr}$)	$w_c = 1$	1.25	1	100k	0.4	512	90.34	
Deep-n-Cheap	$w_c = 0$	2.22	2	1.6M	0.6	512	91.36	
(penalize N_p)	$w_c = 1$	1.85	1	100k	5.54	33	90.4	

Takeaway

We may not need very deep networks!

Also see Zagoruyko 2016 – WRN





https://souryadey.github.io/

