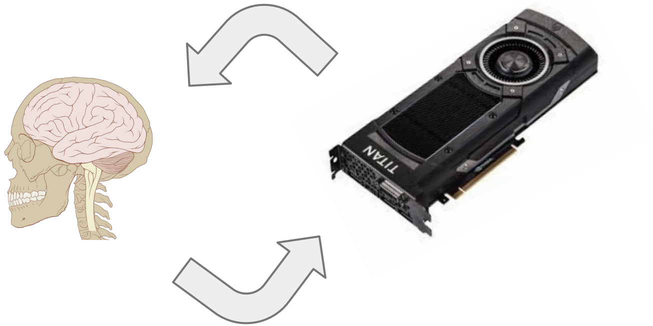
MorphNet

Elad Eban

Faster Neural Nets with Hardware-Aware Architecture Learning

Google AI

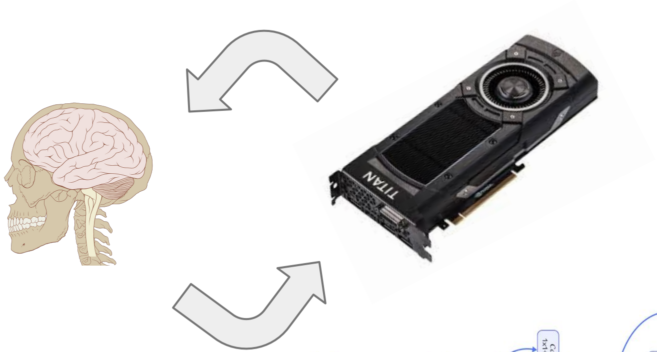
Where Do Deep-Nets Come From?

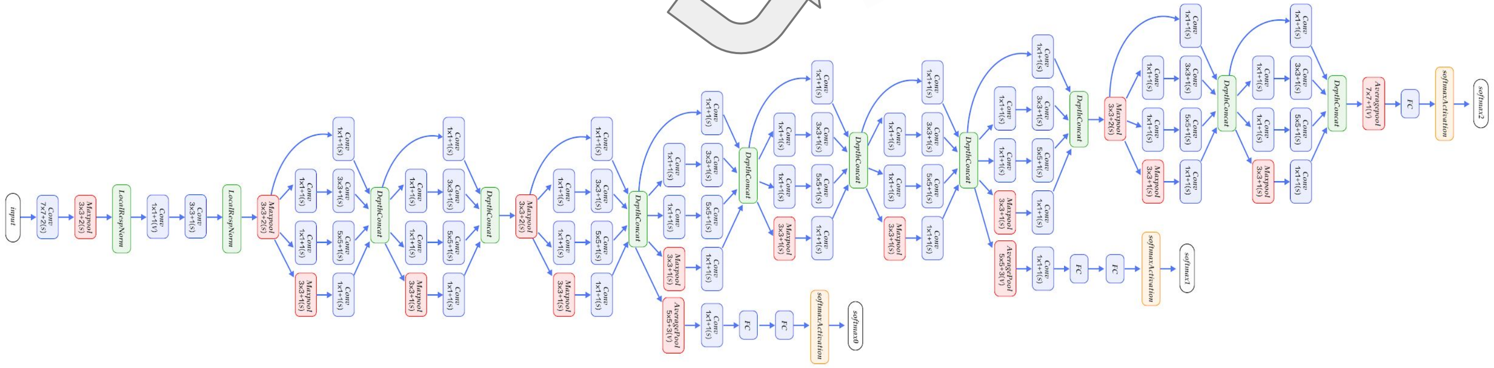


VGG: Chatfield et al. 2014

Image from: http://www.paddlepaddle.org/

How Do We Improve Deep Nets?

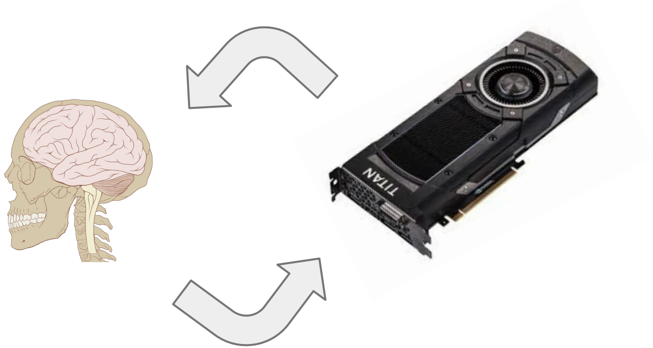


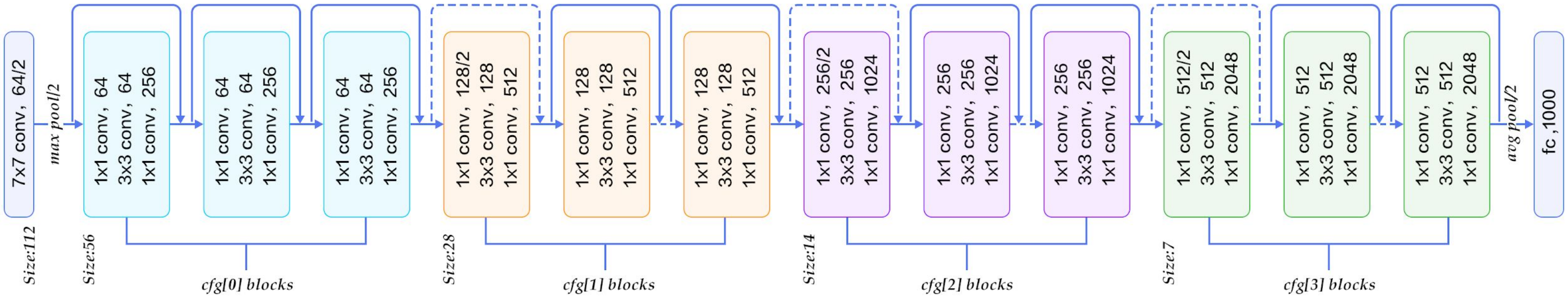


Inception - Szegedy et al. 2015

Image from: http://www.paddlepaddle.org/

How Do We Improve? Speed? Accuracy?

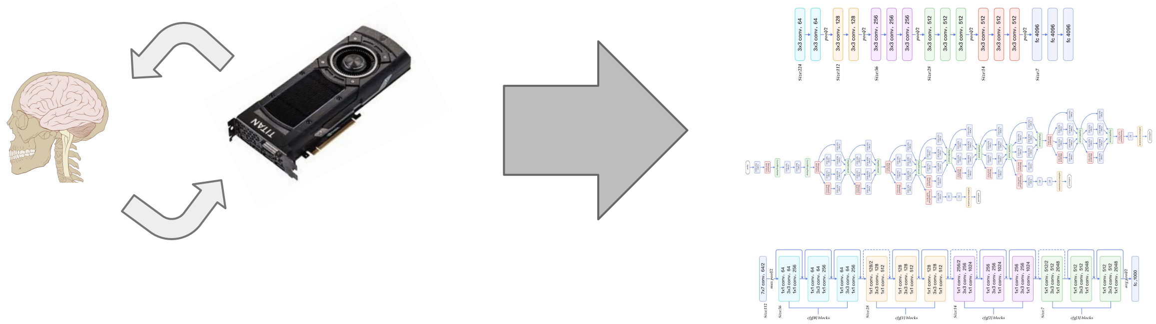




ResNet - K. He, et al. 2016.

Image from: http://www.paddlepaddle.org/

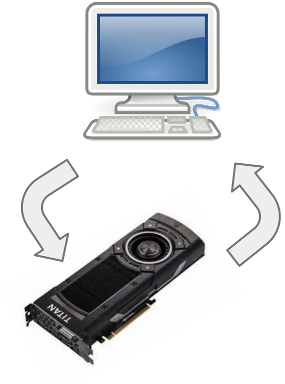
Classical Process of Architecture Design

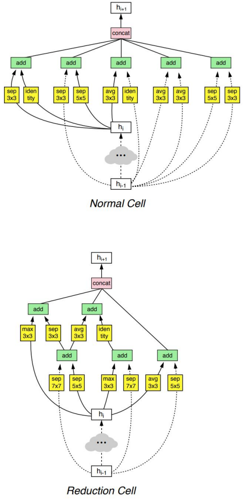


- **Not** scalable
- **Not** optimal
- **Not** customized to YOUR data or task
- **Not** designed to YOUR resource constraints

Rise of the Machines: Network Architecture Search

- identity
- 1x7 then 7x1 convolution
- 3x3 average pooling
- 5x5 max pooling
- 1x1 convolution
- 3x3 depthwise-separable conv
- 7x7 depthwise-separable conv
- 1x3 then 3x1 convolution
- 3x3 dilated convolution
- 3x3 max pooling
- 7x7 max pooling
- 3x3 convolution
- 5x5 depthwise-seperable conv





Neural Architecture Search with Reinforcement Learning **22,400 GPU days!**

Learning Transferable Architectures for Scalable Image Recognition - **RNN 2000 GPU days**

Efficient Neural Architecture Search via Parameter Sharing **~ 2000 training runs**

Figures from: Learning Transferable Architectures for Scalable Image Recognition

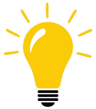
MorphNet: Architecture Learning

Efficient & scalable architecture learning for everyone

- Resource constraints guide customization

- Requires handful of training runs

- Trains on your data
- Start with your architecture
- Works with your code



Idea: Continuous relaxation of combinatorial problem

Simple & effective tool: **weighted sparsifying regularization.**

Learning the Size of Each Layer

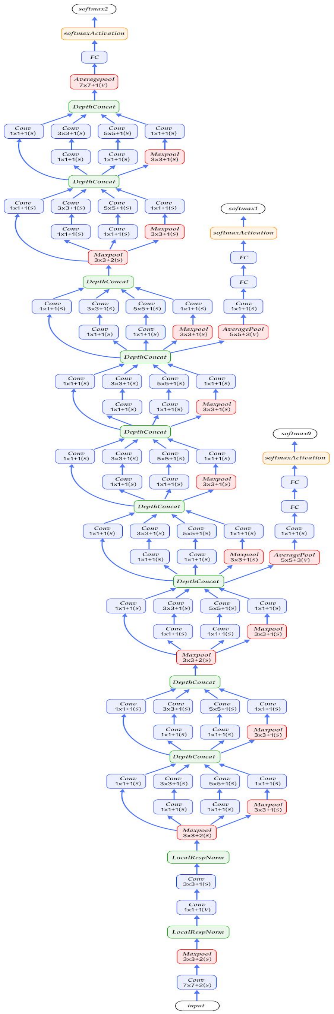
Architecture search

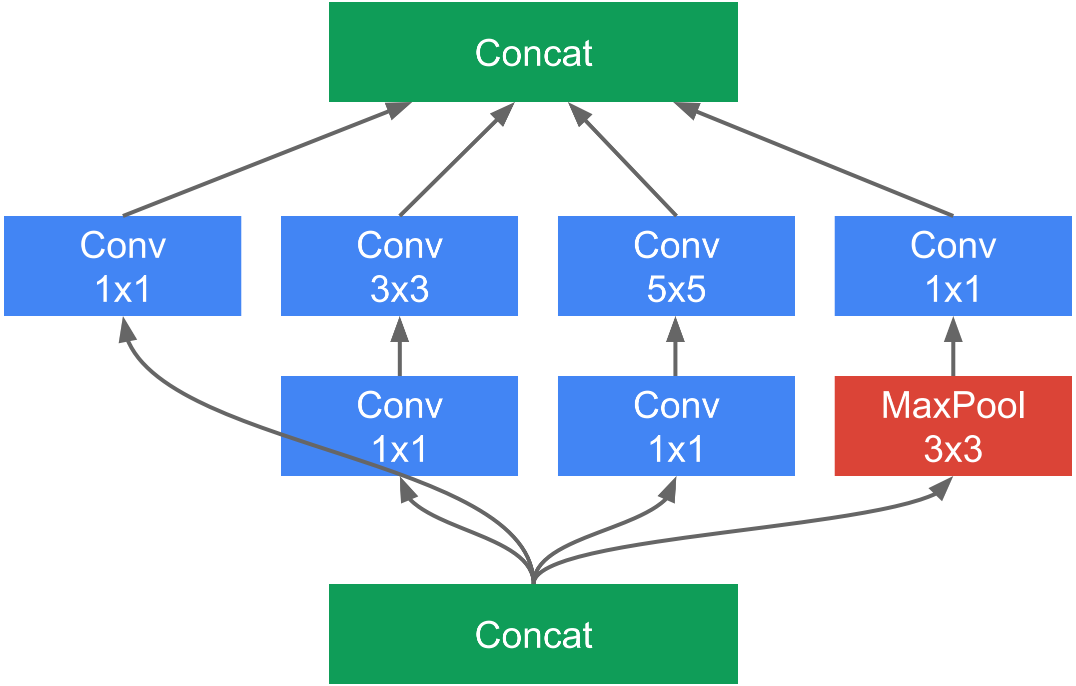
Topology

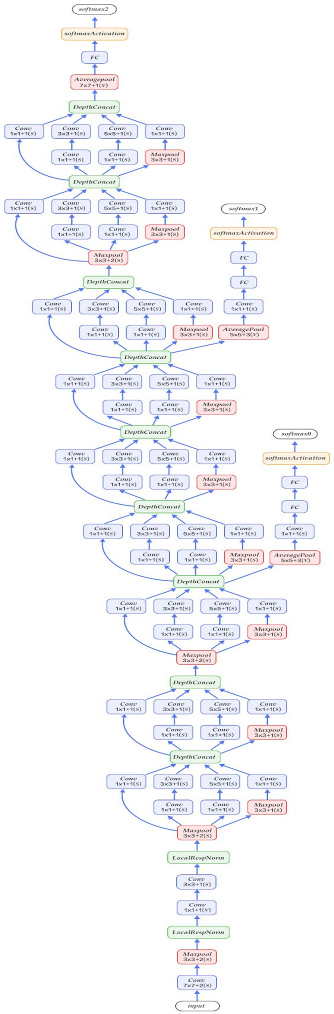
type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj
convolution	7×7/2	112×112×64	1						
max pool	3×3/2	56×56×64	0						
convolution	3×3/1	56×56×192	2		64	192			
max pool	3×3/2	28×28×192	0						
inception (3a)		28×28×256	2	64	96	128	16	32	32
inception (3b)		28×28×480	2	128	128	192	32	96	64
max pool	3×3/2	14×14×480	0						
inception (4a)		14×14×512	2	192	96	208	16	48	64
inception (4b)		14×14×512	2	160	112	224	24	64	64
inception (4c)		14×14×512	2	128	128	256	24	64	64
inception (4d)		14×14×528	2	112	144	288	32	64	64
inception (4e)		14×14×832	2	256	160	320	32	128	128
max pool	3×3/2	7×7×832	0						
inception (5a)		7×7×832	2	256	160	320	32	128	128
inception (5b)		7×7×1024	2	384	192	384	48	[illegible]	[illegible]

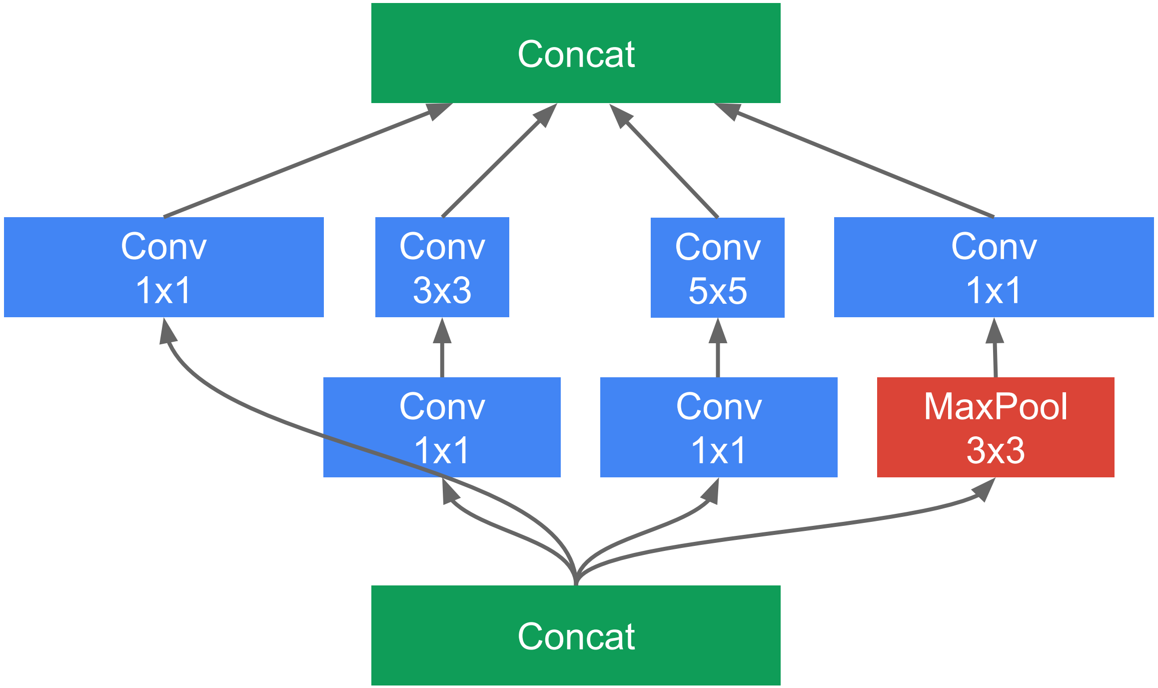
Sizes

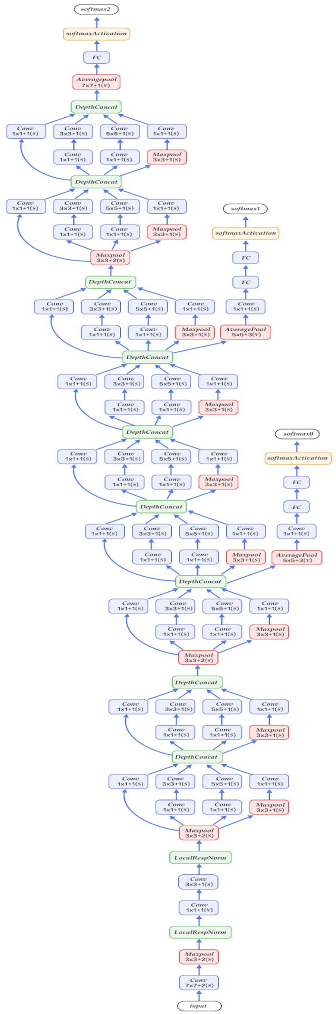
We focus on

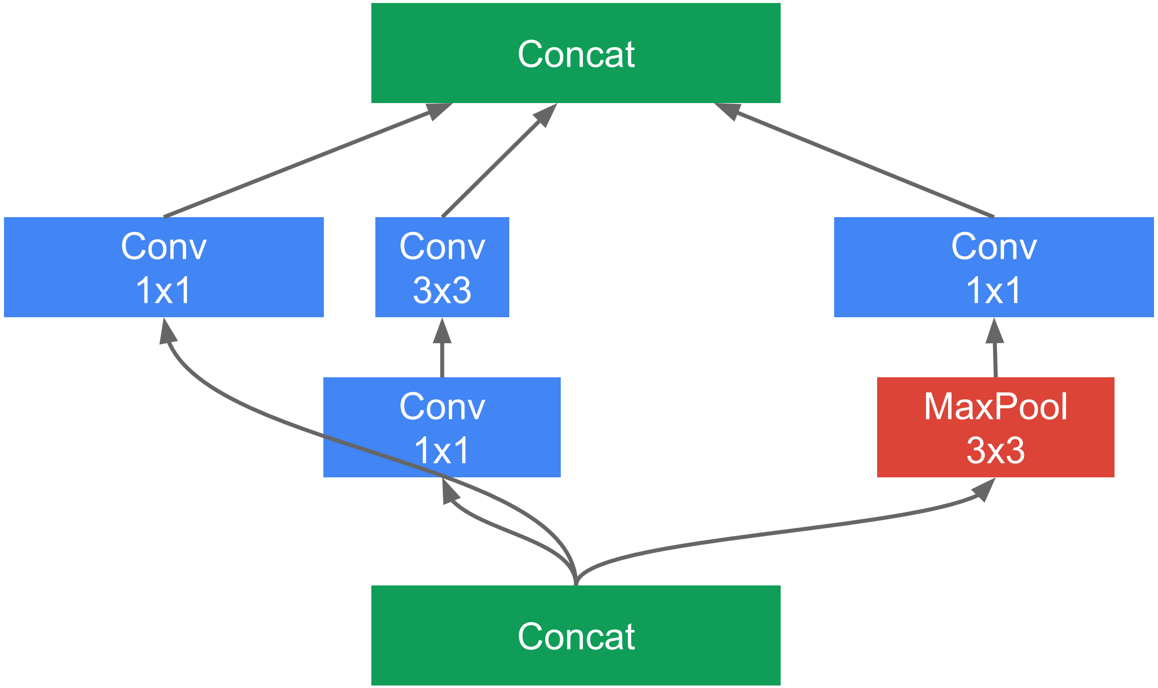



Concat
Conv 1x1
Conv 3x3
Conv 5x5
Conv 1x1
Conv 1x1
Conv 1x1
MaxPool 3x3
Concat




Concat
Conv
1x1
Conv
3x3
Conv
5x5
Conv
1x1
Conv
1x1
Conv
1x1
MaxPool
3x3
Concat




Concat
Conv
1x1
Conv
3x3
Conv
1x1
Conv
1x1
MaxPool
3x3
Concat

Main Tool: Weighted **sparsifying** regularization.

Sparsity Background

Sparsity is just - few non zeros

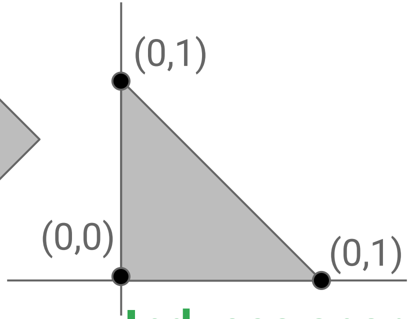
$$B_{L_0}(k) = \left\{ x \in \{0,1\}^n , \sum_i x_i \leq k \right\}$$

$$\{(0,0),(0,1),(1,0)\}$$

Hard to work with in neural nets

Continuous relaxation

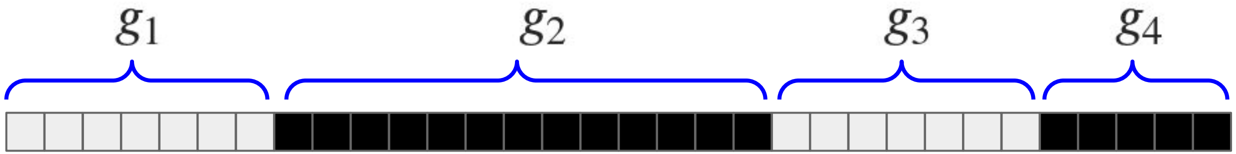
$$B_{L_1}(k) = \left\{ \sum_i |x_i| \leq k \right\}$$



Induces sparsity

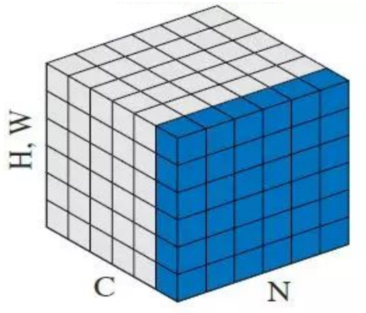
(Group) LASSO: Sparsity in Optimization

$$\min_{w} \ell(X, Y, w) + \lambda |w|_1$$



$$\min_{w} \ell(X, Y, w) + \lambda \sum_{g \in G} \sqrt{w_{g_1}^2 + \ldots w_{g_k}^2}$$

Weight matrix



MorphNet Algorithm

Main Tool: Good-old, simple sparsity

Optional 1.1: Uniform expansion

Stage 1: Structure learning

$$\min_W \sum_i \ell(x_i, y_i, W) + \lambda R(W)$$

Export learned structure

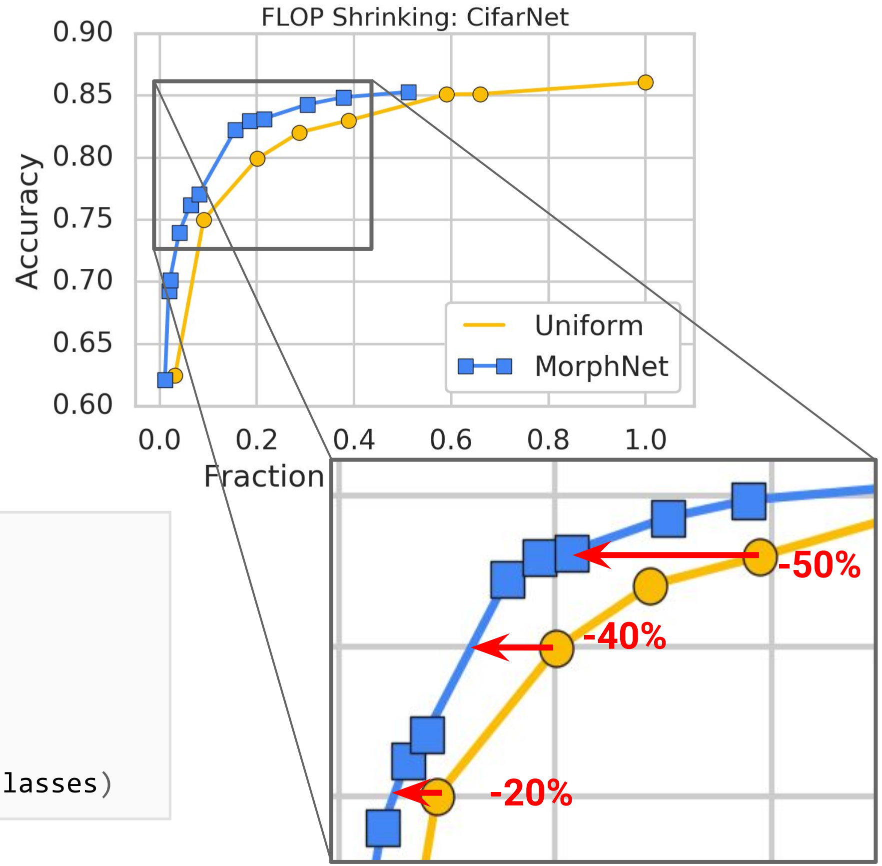
Stage 2: Finetune or retrain weights of learned structure

$$\min_W \sum_i \ell(x_i, y_i, W)$$

Shrinking CIFARNet

```
net = conv2d(images, 64, [5, 5])
net = max_pool2d(net, [2, 2], 2)
net = conv2d(net, 64, [5, 5])
net = max_pool2d(net, [2, 2], 2)
net = fully_connected(net, 384)
net = fully_connected(net, 192)
logits = fully_connected(net, num_classes)
```

```
net = conv2d(images, 28, [5, 5])
net = max_pool2d(net, [2, 2], 2)
net = conv2d(net, 18, [5, 5])
net = max_pool2d(net, [2, 2], 2)
net = fully_connected(net, 205)
net = fully_connected(net, 88)
logits = fully_connected(net, num_classes)
```



Can This Work in Conv-nets?

What do Inception, resnet, dense-net, NAS-net, Amoeba-Net have in common?

```
def batch_norm(x):
  return (x - mean(x)) / std(x)
```

Problem: The weight matrix is scale invariant.

```
batch_norm(W*x) == batch_norm((W/2.0)*x)
```

L1-Gamma regularization

Actually batch norm has a learned scale parameter:

```
def batch_norm(x, gamma):
  return (x - mean(x)) / std(x) * gamma
```

Problem: Still scale invariant.

Solution: The scale parameter **gamma** is the perfect substitute.
Zeroing **gamma** is effectively removing the filter!

$$\min_{w} \ell(X, Y, w) + \lambda|\gamma|_1$$

Main Tool: **Weighted** sparsifying regularization.

What Do We Actually Care About?

We can now control on the number of filters.

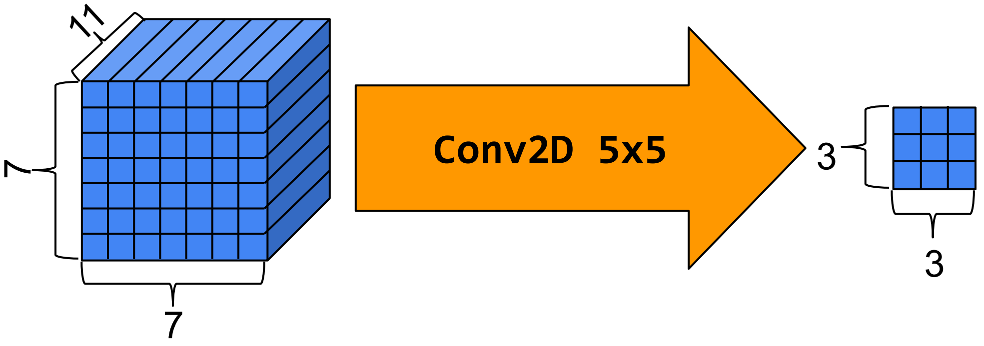
But, what we actually care about is: model size, FLOPs and inference time.

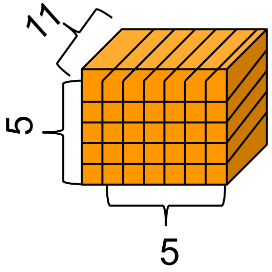
Notice: FLOPs and model size are a simple function of the number of filters.

Solution: Per-layer coefficient that captures the cost.

$$\min_{w} \ell(X, Y, w) + \lambda \sum_{l} C^{cost}(l)|\gamma_l|_1$$

What is the Cost of a Filter?





Model-size coefficient: `filter_size * num_inputs`

(5 * 5) * 11

FLOP coefficient: `filter_size * num_inputs * output_size`

(5 * 5) * 11 * (3 * 3)

Inception V2 Based Networks on ImageNet

Baseline: Uniform shrinkage of all layers (width multiplier).

FLOP Regularizer: Structure learned with FLOP penalty.

Expanded structure: Uniform expansion of learned structure.

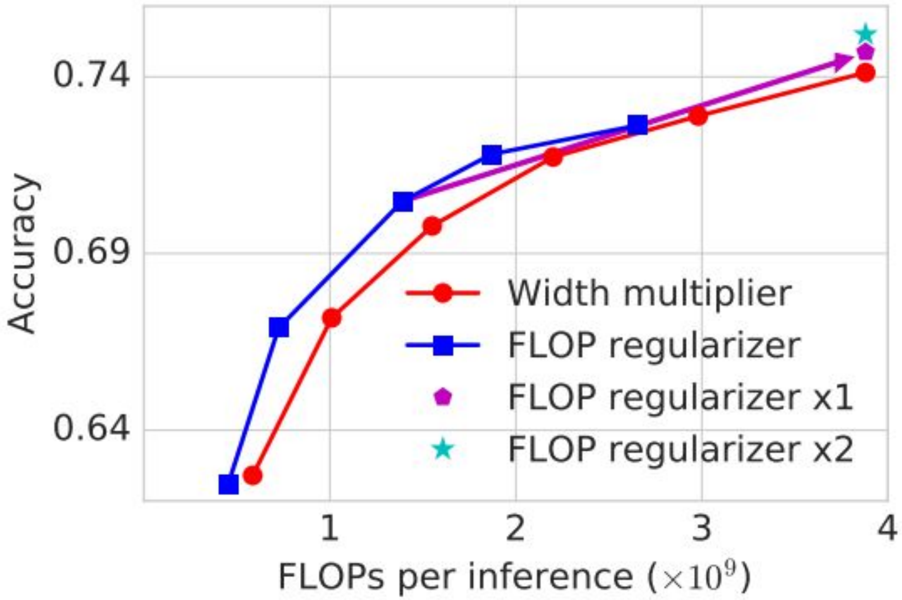


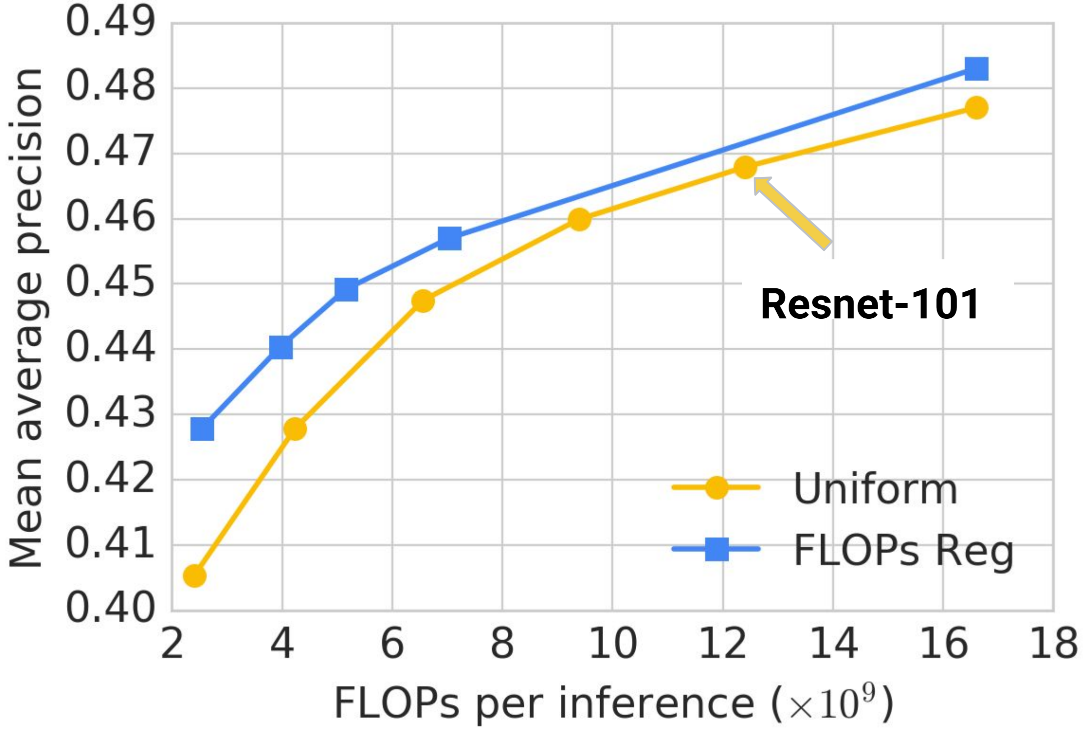
Figure from: MorphNet: Fast & Simple Resource-Constrained Structure Learning of Deep Networks

JFT: Google Scale Image Classification

Image classification with 300M+ Images, >20K classes.

Started with a ResNet-101 architecture.

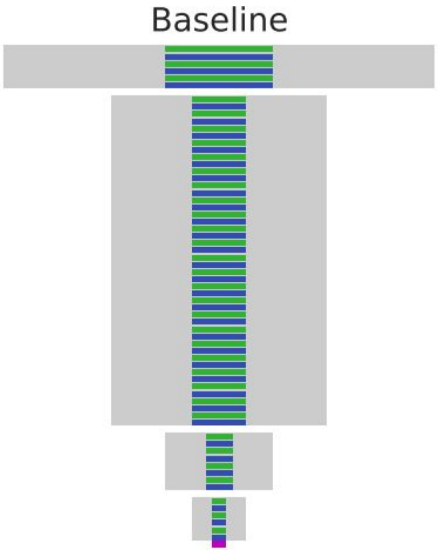
The first model with algorithmically learned architecture serving in production.



Google

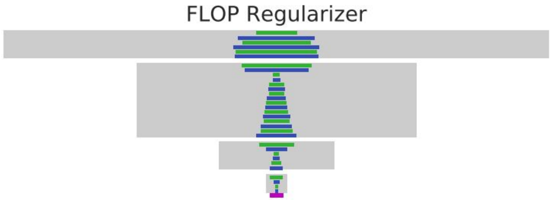
Figure adapted from: MorphNet: Fast & Simple Resource-Constrained Structure Learning of Deep Networks

ResNet101-Based Learned Structures

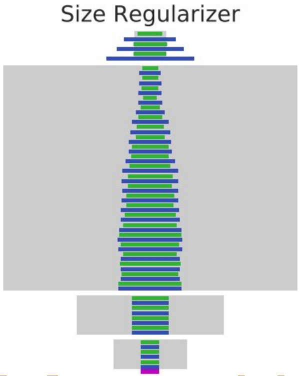


FLOP Regularizer

Model Size Regularizer



40% fewer FLOPs



All models have the same performance.

43% fewer weights

Figure adapted from: MorphNet: Fast & Simple Resource-Constrained Structure Learning of Deep Networks

A Custom Architecture Just For You!

Partnered with Google OCR team which maintains models for dozens of scripts which differ in:

- **Number of characters,**
- **Character complexity**,
- **Word-length**,
- **Size of data.**

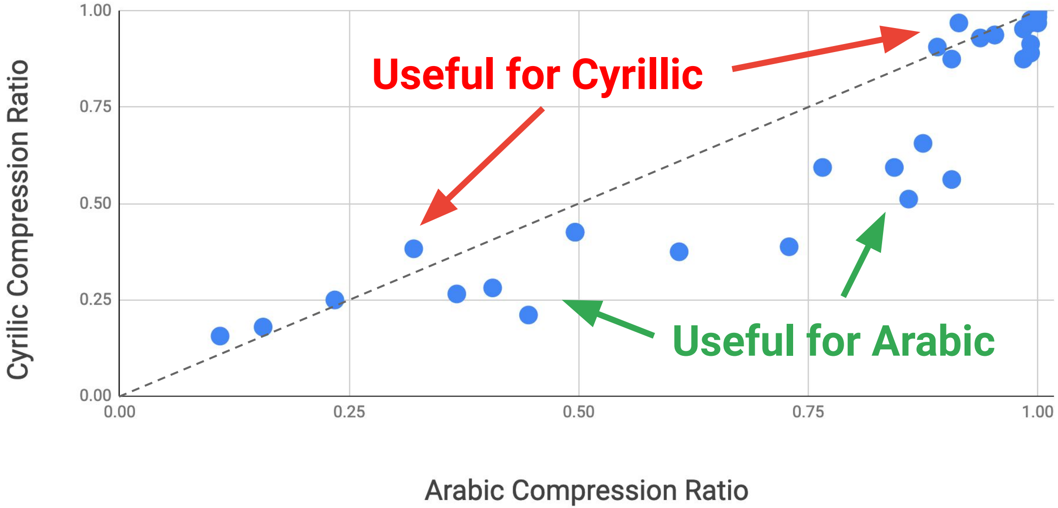
A single fixed architecture was used for all scripts!

A Custom Architecture Just For You!

А	Б	В	Г	Д	Е
Ё	Ж	З	И	Й	К
Л	М	Н	О	П	Р
С	Т	У	Ф	Х	Ц
Ч	Ш	Щ	Ъ	Ы	Ь
Э	Ю	Я			

ا	ب	ت	ث	ج	ح	خ
د	ذ	ر	ز	س	ش	ص
ض	ط	ظ	ع	غ	ف	ق
ك	ل	م	ن	ه	و	ى

Models with 50% of FLOPs
(with same accuracy)



Zooming in On Latency

Brute force

#Activation

#FLOPs

?

Latency is device specific!

Latency Roofline Model

Each op needs to read inputs, perform calculations, and write outputs.

Evaluation time of an op depends on the **compute** and **memory** costs.

Compute time = FLOPs / **compute_rate**.

Memory time = tensor_size / **memory_bandwidth**.

Latency = max(**Compute time, Memory time**)

Device Specific

Example Latency Costs

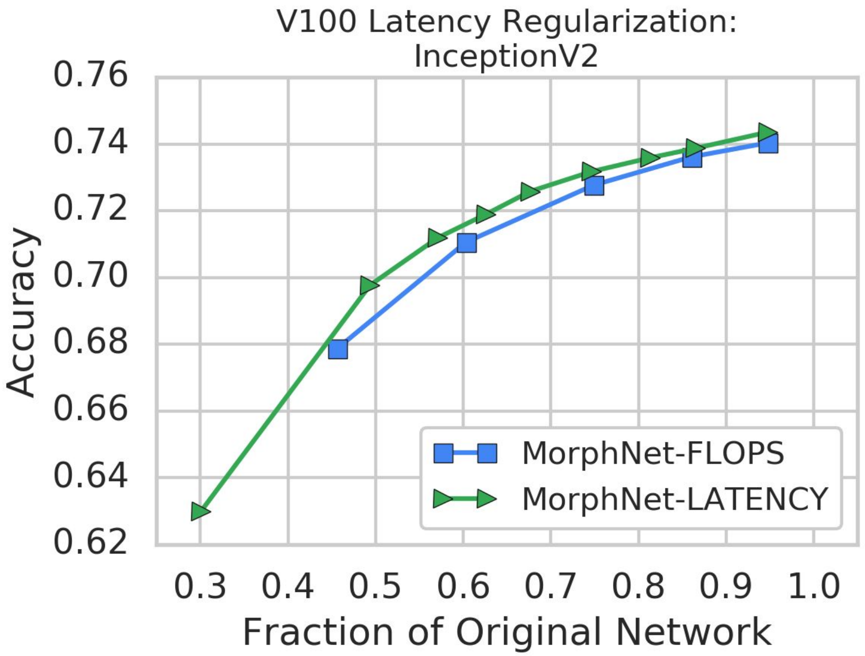
Different platforms have different cost profile

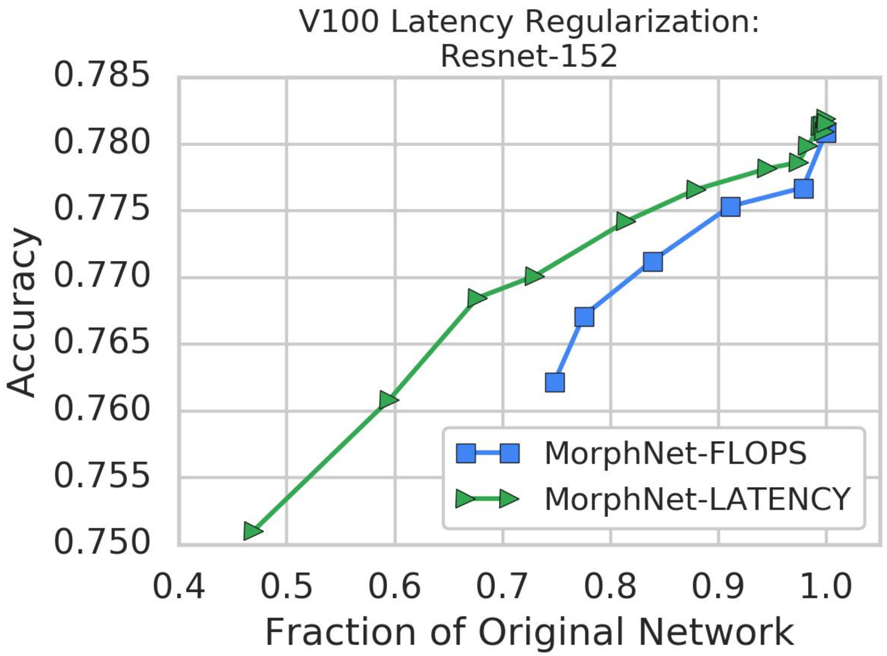
Platform	Peak Compute	Memory Bandwidth
P100	9300 GFLOPs/s	732 GB/s
V100	125000 GFLOPs/s	900 GB/s

Leads to different relative cost

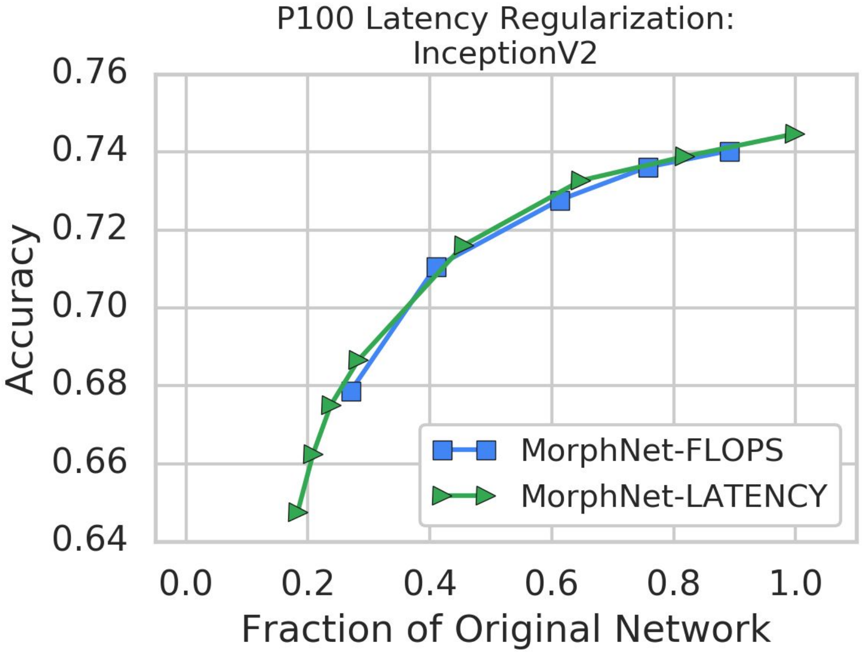
Inception V2 Layer Name	P100 Latency	V100 Latency	Ratio
Conv2d_2c_3x3	74584	5549	7%
Mixed_3c/Branch_2/Conv2d_0a_1x1	2762	1187	43%
Mixed_5c/Branch_3/Conv2d_0b_1x1	1381	833	60%

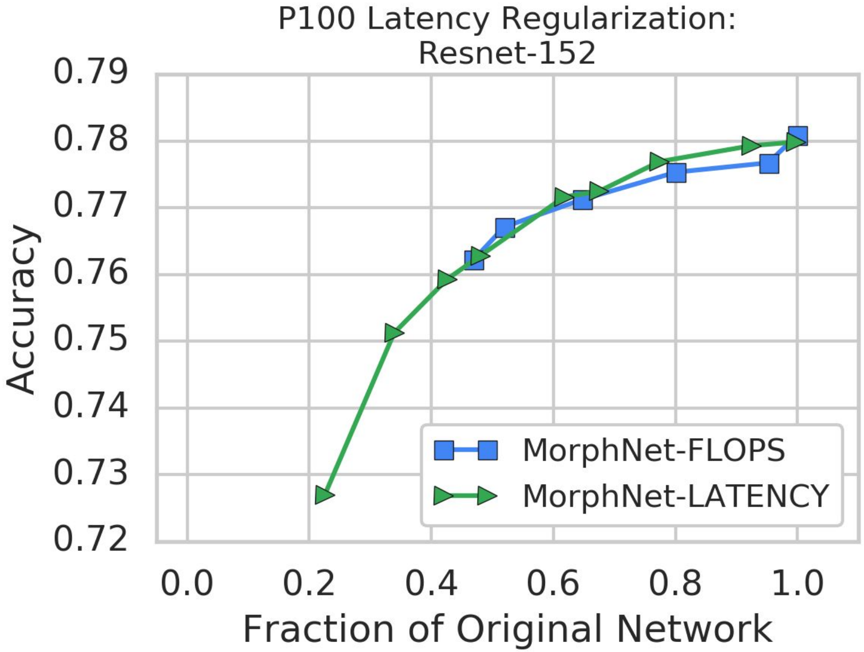
Tesla V100 Latency





Tesla P100 Latency



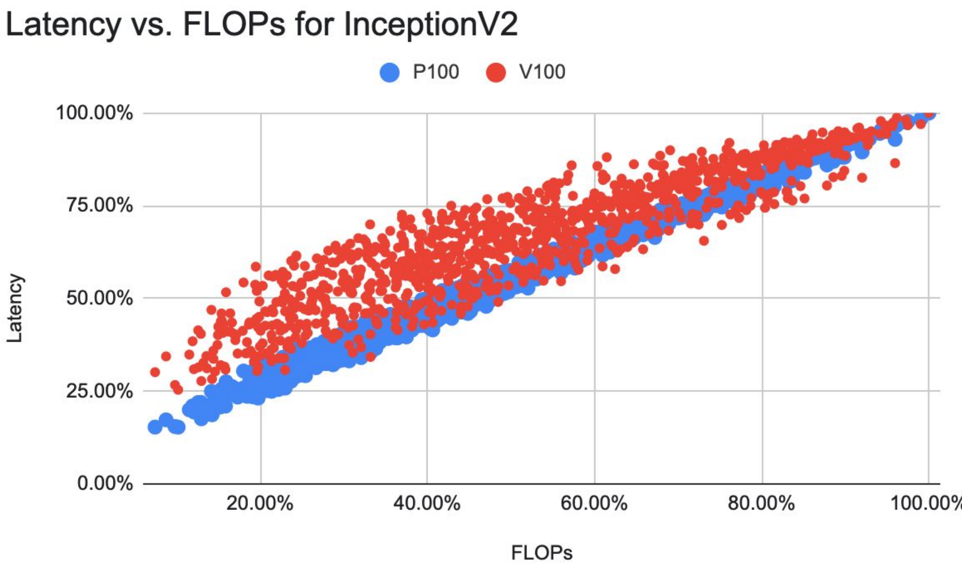


When Do FLOPs and Latency Differ?

- Create 5000 *sub-Inception V2* models with a random number of filters.
- Compare FLOPs, V100 and P100 Latency.

P100 - is compute bound, tracks FLOPs “too” closely

V100 - gap between FLOPs and Latency is looser



What Next

If you want to

- Algorithmically speedup or shrink your model,
- Easily improve your model

You are invited to use our open source library

https://github.com/google-research/morph-net

Quick User Guide

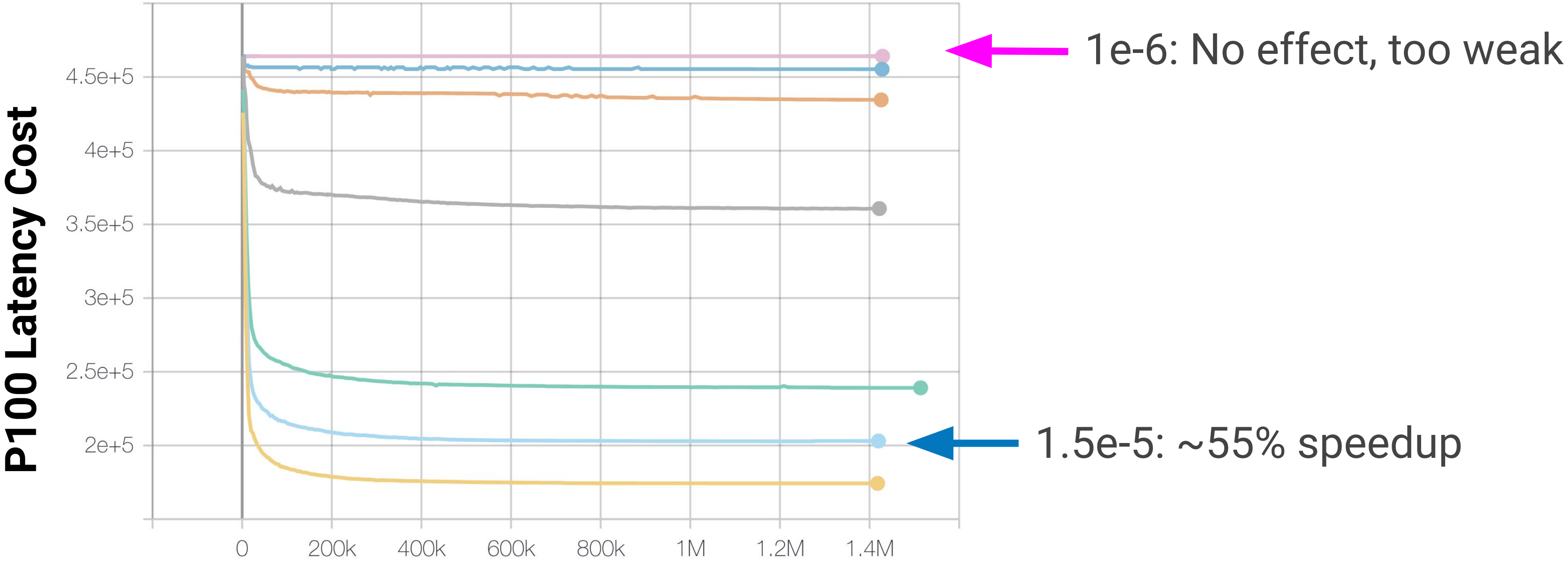
```
from morph_net.network_regularizers import flop_regularizer
from morph_net.tools import structure_exporter

logits = build_model()

network_regularizer = flop_regularizer.GammaFlopsRegularizer(
    [logits.op], gamma_threshold=1e-3)
regularization_strength = 1e-10
regularizer_loss = (network_regularizer.get_regularization_term() * regularization_strength)

model_loss = tf.nn.sparse_softmax_cross_entropy_with_logits(labels, logits)

optimizer = tf.train.MomentumOptimizer(learning_rate=0.01, momentum=0.9)

train_op = optimizer.minimize(model_loss + regularizer_loss)
```

Exact same API works for different costs and settings:

GroupLassoFlops, GammaFlops, GammaModelSize, GammaLatency

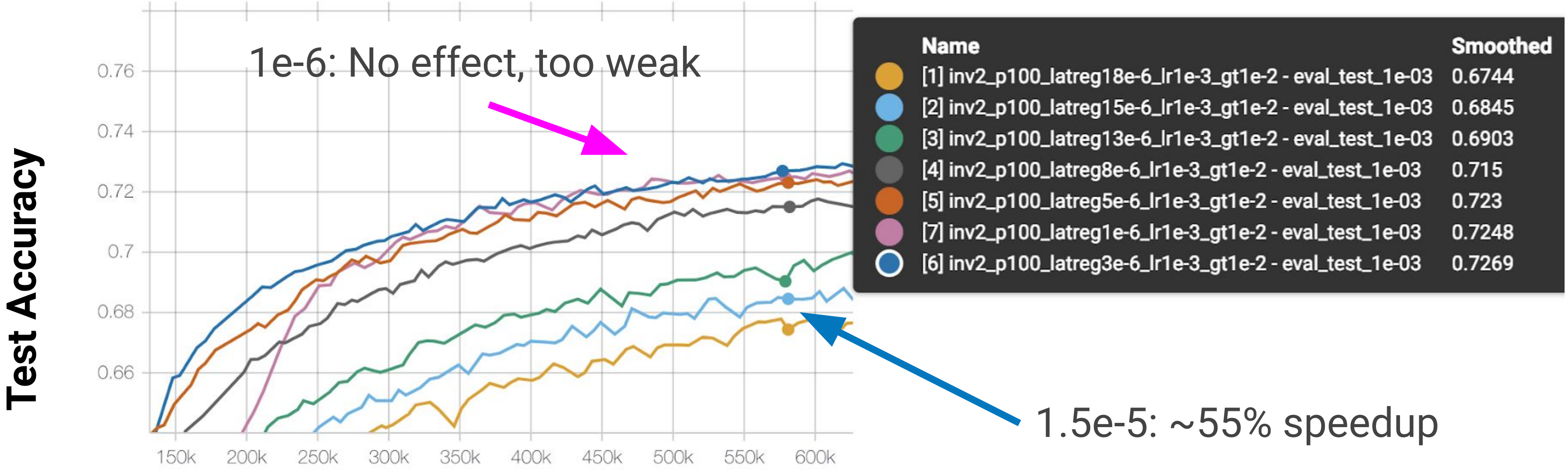
Structure Learning: Regularization Strength

Pick a few regularization strengths.



Structure Learning: Accuracy Tradeoff

Of course there is a tradeoff

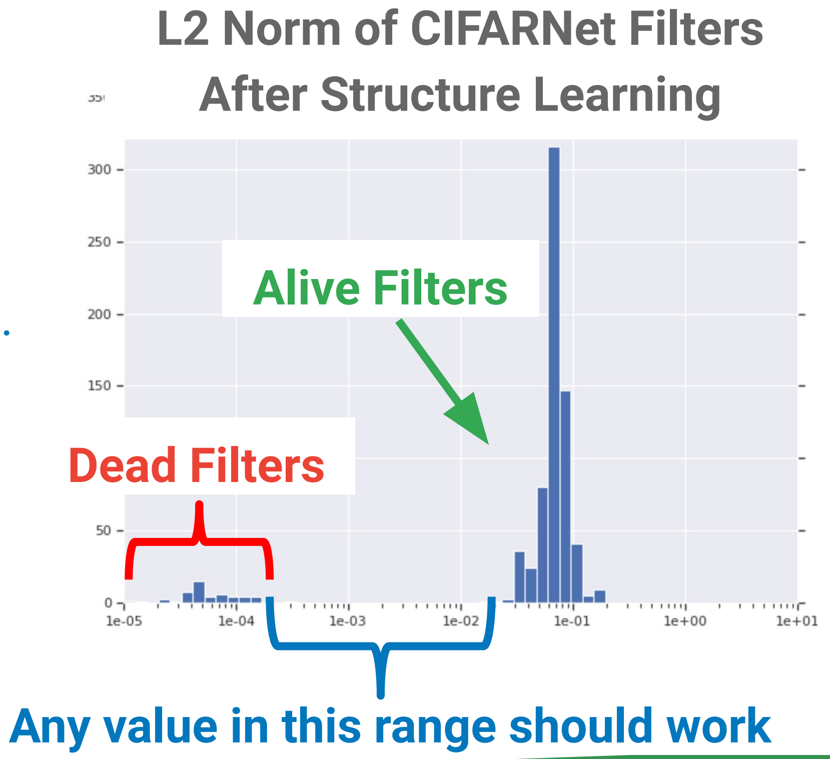


Structure Learning: Threshold

Value of gamma, or group LASSO norms usually don't reach 0.0 so a threshold is needed.

Plot regularized value: L2 or abs(gamma).

Usually easy to determine, often the distribution is bimodal.



Structure Learning: Exporting

```
network_regularizer = ...
net = conv2d(images, 64, [5, 5])
net = max_pool2d(net, [2, 2], 2)
net = conv2d(net, 64, [5, 5])
net = max_pool2d(net, [2, 2], 2)
net = fully_connected(net, 384)
net = fully_connected(net, 192)
logits = fully_connected(net, num_classes)

se = structure_exporter.StructureExporter(
                network_regularizer.opreg_manager)
f = file.Open(filename, 'w')
se.save_alive_counts(f)
```

```
with file.Open(filename, 'r') as f:
  parameterization = json.loads(f.read())

net = conv2d(images, parameterization['conv1'], [5, 5])
net = max_pool2d(net, [2, 2], 2)
net = conv2d(net,  parameterization['conv2'], [5, 5])
net = max_pool2d(net, [2, 2], 2)
net = fully_connected(net, parameterization['fc3'])
net = fully_connected(net, parameterization['fc4'])
logits = fully_connected(net, num_classes)
```

Retraining/Fine Tuning

Problem

- Extra regularization hurts performance.
- Some filters are not completely dead.

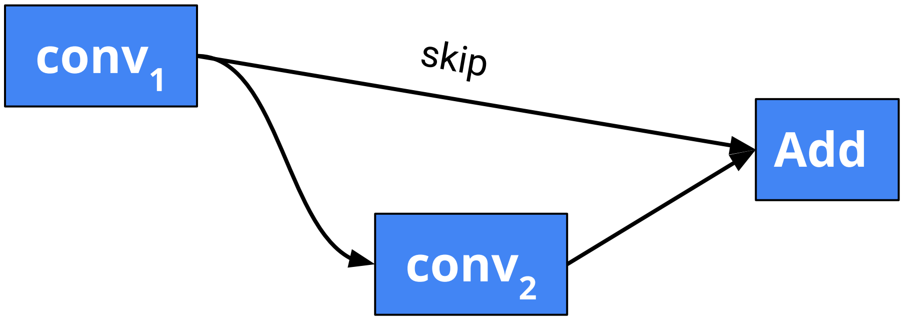
Options

- Zero dead filters and finetune.
- **Train learned structure from scratch.**

Why

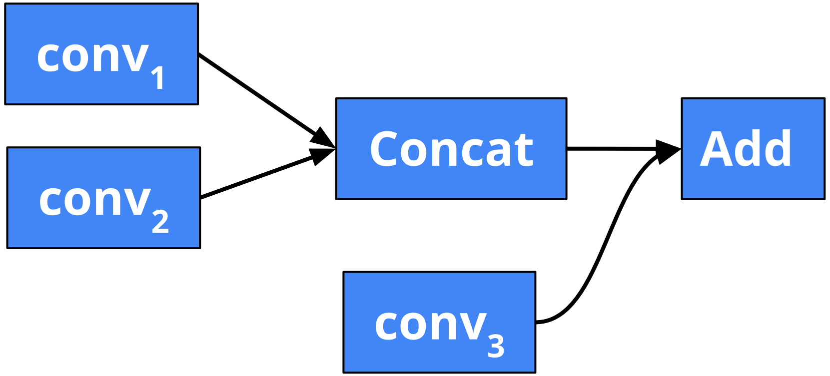
- Ensures learned structure is stand-alone and not tied to learning procedure.
- Stabilizes downstream pipeline.

Under the Hood: Shape Compatibility Constraints



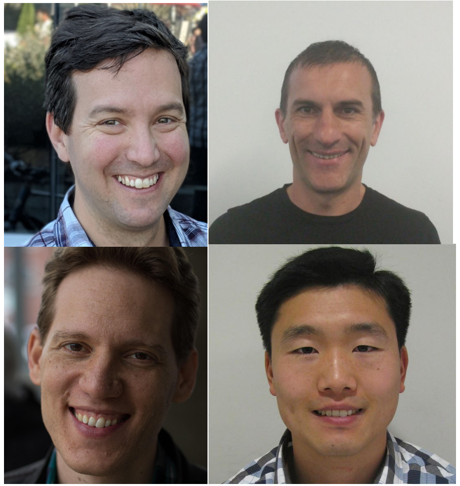
NetworkRegularizers figures out structural dependence in the graph.

Under the Hood: Concatenation (as in Inception)



Things can get complicated, but it is all handled by the MorphNet framework.

Team Effort



Elad Eban, Max Moroz

Yair Movshovitz-Attias, Andrew Poon

Contributors & collaborators:
Ariel Gordon, Bo Chen, Ofir Nachum, Hao Wu, Tien-Ju Yang, Edward Choi, Hernan Moraldo, Jesse Dodge, Yonatan Geifman, Shraman Ray Chaudhuri.

Thank You

Elad Eban

Contact: morphnet@google.com

https://github.com/google-research/morph-net