ASPIRE

BERKELEY DeepDrive

Berkeley
Artificial Intelligence Research

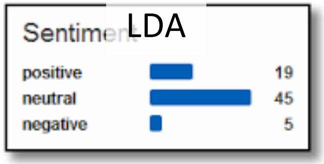
Dissertation Talk:

Exploring the Design Space of Deep Convolutional Neural Networks at Large Scale

Forrest Iandola
forresti@eecs.berkeley.edu

Machine Learning in 2012

Sentiment Analysis



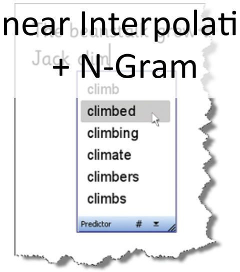
Sentiment

positive	🟦	19
neutral	🟦	45
negative	🟦	5

LDA

Word Prediction

Linear Interpolation + N-Gram



Jack clim

climb
climbed
climbing
climate
climbers
climbs
Predictor

Text Analysis

Audio Analysis

Computer Vision

Speech Recognition

Hidden Markov Model

Audio Concept Recognition

i-Vector + HMM

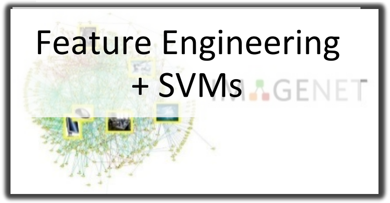
Object Detection

Deformable Parts Model

Semantic Segmentation

segDPM

Image Classification



Feature Engineering + SVMs

We have 10 years of experience in a broad variety of ML approaches …

[1] B. Catanzaro, N. Sundaram, **K. Keutzer.** Fast support vector machine training and classification on graphics processors. International Conference on Machine Learning (ICML), 2008.

[2] Y. Yi, C.Y. Lai, S. Petrov, **K. Keutzer**. Efficient parallel CKY parsing on GPUs. International Conference on Parsing Technologies, 2011.

[3] K. You, J. Chong, Y. Yi, E. Gonina, C.J. Hughes, Y. Chen, **K. Keutzer**. Parallel scalability in speech recognition. *IEEE Signal Processing Magazine*, 2009.

[4] **F. Iandola**, M. **Moskewicz**, K. **Keutzer**. libHOG: Energy-Efficient Histogram of Oriented Gradient Computation. ITSC, 2015.

[5] N. Zhang, R. Farrell, **F. Iandola**, and T. Darrell. Deformable Part Descriptors for Fine-grained Recognition and Attribute Prediction. ICCV, 2013.

[6] M. Kamali, I. Omer, **F. Iandola**, E. Ofek, and J.C. Hart. Linear Clutter Removal from Urban Panoramas International Symposium on Visual Computing. ISVC, 2011.

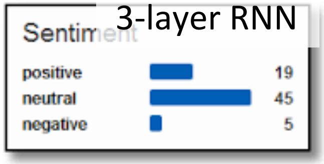
ASPIRE

Berkeley DeepDrive

Berkeley Artificial Intelligence Research

By 2016, Deep Neural Networks Give Superior Solutions in Many Areas

Sentiment Analysis

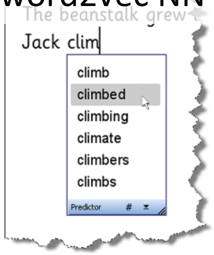

3-layer RNN

Sentiment

positive	19
neutral	45
negative	5

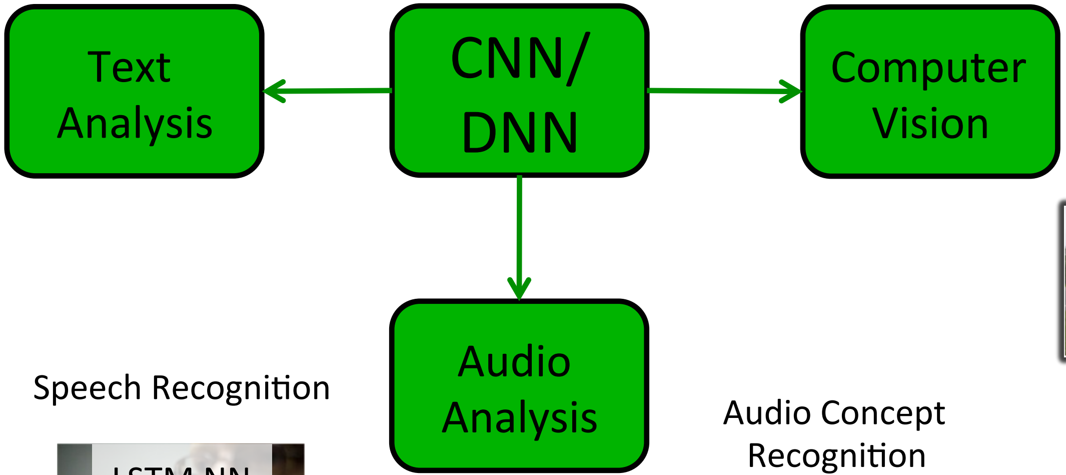
Word Prediction

word2vec NN


The beanstalk grew

Jack clim

- climb
- climbed
- climbing
- climate
- climbers
- climbs



Speech Recognition

LSTM NN

Audio Concept Recognition

4-layer DNN

Object Detection

16-layer DCNN

Semantic Segmentation

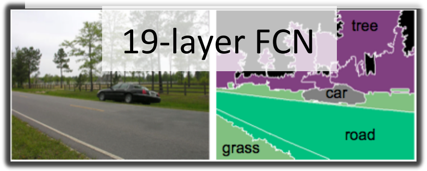
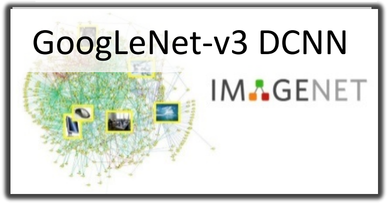

19-layer FCN

Image Classification


GoogLeNet-v3 DCNN

Finding the "right" DNN architecture is replacing broad algorithmic exploration for many problems.

[7] K. Ashraf, B. Elizalde, F. **Iandola**, M. **Moskewicz**, J. Bernd, G. Friedland, K. **Keutzer**. Audio-Based Multimedia Event Detection with Deep Neural Nets and Sparse Sampling. ACM ICMR, 2015.

[8] F. **Iandola**, A. **Shen**, P. Gao, K. **Keutzer**. DeepLogo: Hitting logo recognition with the deep neural network hammer. arXiv:1510.02131, 2015.

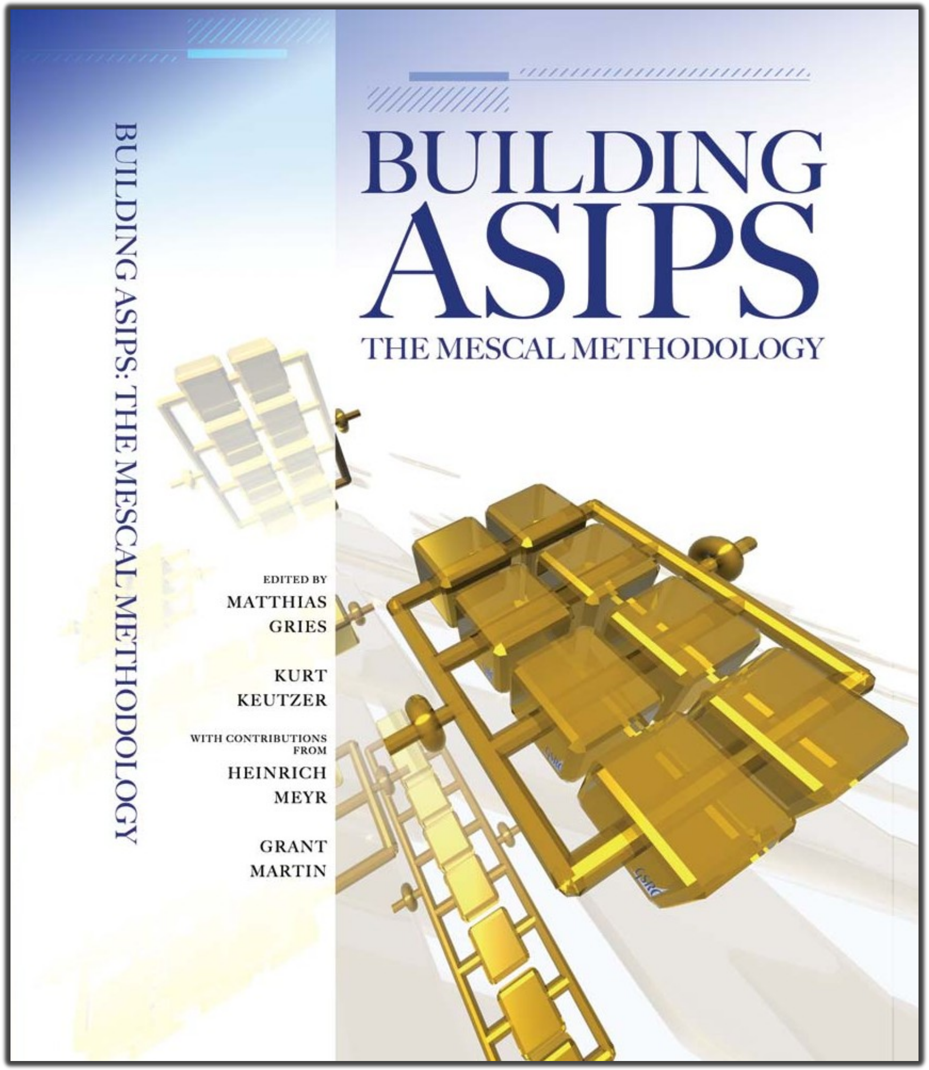
[9] F. **Iandola**, M. **Moskewicz**, S. Karayev, R. Girshick, T. Darrell, K. **Keutzer**. DenseNet: Implementing Efficient ConvNet Descriptor Pyramids. arXiv: 1404.1869, 2014.

[10] R. Girshick, F. **Iandola**, T. Darrell, J. Malik. Deformable Part Models are Convolutional Neural Networks. CVPR, 2015.

[11] **F. Iandola**, K. Ashraf, **M.W. Moskewicz, K. Keutzer**. FireCaffe: near-linear acceleration of deep neural network training on compute clusters. arXiv: 1511.00175, 2015. Also, CVPR 2016, pp. 2592–2600.

[12] K. Ashraf, B. Wu, **F.N. Iandola**, **M.W. Moskewicz, K. Keutzer**. Shallow Networks for High-Accuracy Road Object-Detection. arXiv:1606.01561, 2016.

The MESCAL Methodology for exploring the design space of computer hardware



The methodology includes a number of themes, such as…

- Judiciously using benchmarking
- Efficiently evaluate points in the design space
- Inclusively identify the architectural space
- Comprehensively explore the design space

Outline of our approach to exploring the design space of CNN/DNN architectures

- Theme 1: Defining benchmarks and metrics to evaluate CNN/ DNNs
- Theme 2: Rapidly training CNN/DNNs
- Theme 3: Defining and describing the CNN/DNN design space
- Theme 4: Exploring the design space of CNN/DNN architectures

Theme 1: Defining benchmarks and metrics to evaluate CNN/DNNs

What *exactly* would we like our neural network to accomplish?

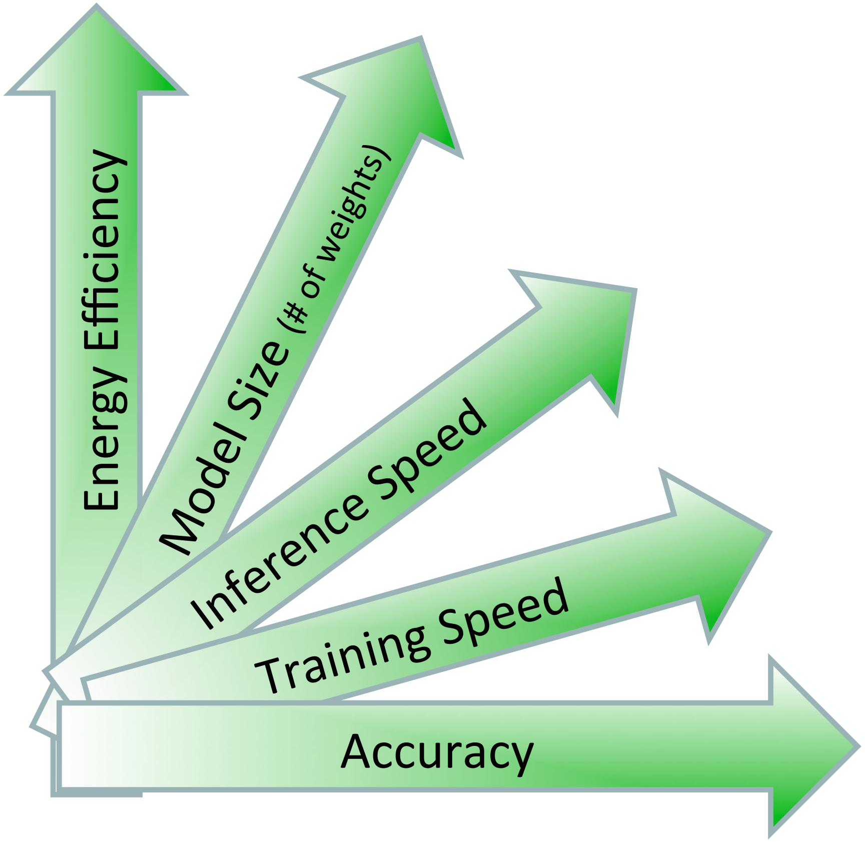
Key benchmarks used in four deep learning problem areas

Type of data	Problem area	Size of benchmark's training set	CNN/DNN architecture	Hardware	Training time
text [1]	word prediction (word2vec)	100 billion words (Wikipedia)	2-layer skip gram	1 NVIDIA Titan X GPU	6.2 hours
audio [2]	speech recognition	2000 hours (Fisher Corpus)	11-layer RNN	1 NVIDIA K1200 GPU	3.5 days
images [3]	image classification	1 million images (ImageNet)	22-layer CNN	1 NVIDIA K20 GPU	3 weeks
video [4]	activity recognition	1 million videos (Sports-1M)	8-layer CNN	10 NVIDIA GPUs	1 month

- High-dimensional data (e.g. images and video) tends to require more processing during both training and inference.
- One of our goals was to find the most computationally-intensive CNN/DNN benchmarks, and then go to work on accelerating these applications
  - Image/Video benchmarks meet these criteria
  - *Convolutional* Neural Networks (CNNs) are commonly applied to Image/Video data

[1] John Canny, et al., "Machine learning at the limit," IEEE International Conference on Big Data, 2015.
[2] Dario Amodei, et al., "Deep speech 2: End-to-end speech recognition in english and mandarin," arXiv:1512.02595, 2015.
[3] Sergio Guadarrama, "BVLC googlenet," https://github.com/BVLC/caffe/tree/master/ models/bvlc_googlenet, 2015.
[4] A. Karpathy, et al., "Large-scale video classification with convolutional neural networks," CVPR, 2014.

Key metrics for specifying CNN/DNN design goals



To achieve the optimal results on these metrics, it's important to design and/or evaluate:

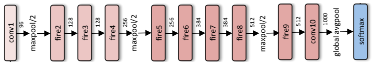
- CNN architectures
- Software/Libraries
- Hardware architectures

Strategies for evaluating team progress on full-stack CNN/DNN system development

Evaluating individual contributions

Evaluating the overall system

CNN Team



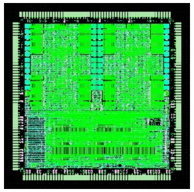
- Accuracy
- Quantity of computation
- Model Size

Software/Libraries Team

```
kernel<<< >>>
```

- Percent of peak throughput achieved on appropriate hardware

Hardware Team



- Power envelope
- Peak achievable throughput

- Energy per frame
- Inference speed per frame

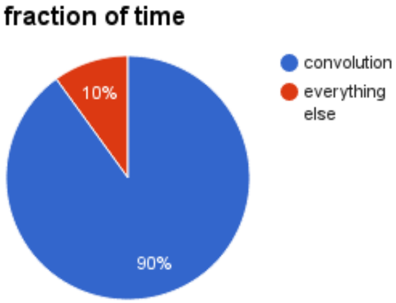
Theme 2: Rapidly training CNN models

Without exaggeration, training a CNN can take weeks

Train rapidly → More productively explore the design space

What are the options for how to accelerate CNN training?

- Accelerate convolution?
  - 90% of computation time in a typical CNN is convolution
  - 2008: Communication-avoiding GPU matrix-multiplication [1]
  - 2013: Communication-avoiding GPU 2D Convolution [2] **(our work!)**
  - 2014: Communication-avoiding GPU 3D Convolution [3]
    - 50-90% of peak FLOPS/s for typical DNN problem sizes
  - Not much juice left to squeeze here.



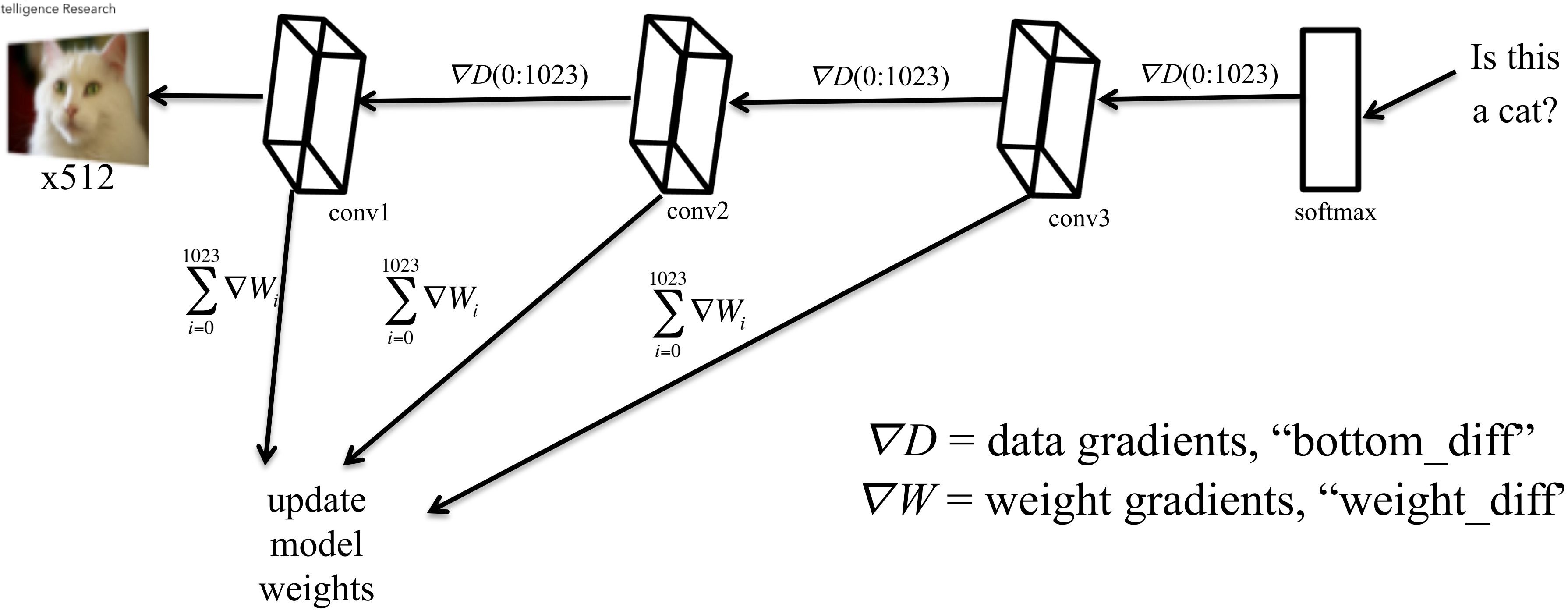
- Put more GPUs into my workstation?
  - Can fit up to 8 GPUs into a high-end workstation. Scale CNN training over these?
  - Facebook and Flickr have each been pretty successful at this

x8

- Scale CNN training across a cluster of GPU-enabled servers?
  - We enable this in our *FireCaffe* framework [4]

[1] V. Volkov and J. Demmel. Benchmarking GPUs to tune dense linear algebra. Supercomputing, 2008.
[2] **F.N. Iandola**, D. Sheffield, M. Anderson, M.P. Phothilimthana, K. Keutzer. Communication-Minimizing 2D Convolution in GPU Registers ICIP, 2013.
[3] S. Chetlur, et al. cuDNN: Efficient Primitives for Deep Learning. arXiv, 2014.
[4] **F.N. Iandola**, K. Ashraf, M.W. Moskewicz, and K. Keutzer. FireCaffe: near-linear acceleration of deep neural network training on compute clusters. CVPR, 2016.

Warmup: Single-Server CNN training



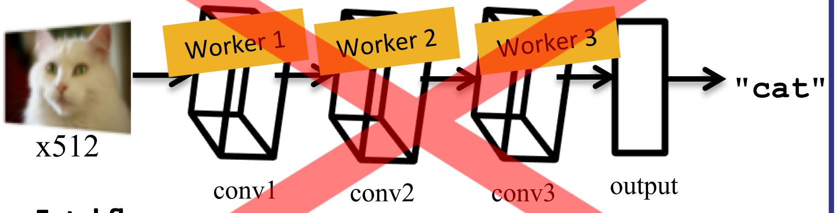
∇D = data gradients, "bottom_diff"

∇W = weight gradients, "weight_diff"

- Next, we discuss strategies for scaling up CNN training

Four approaches to parallelizing CNN training

Approach #1: Pipeline parallelism



Fatal flaws:

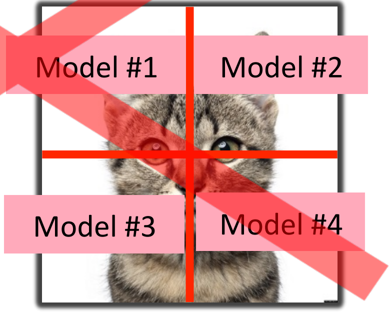
- Parallelism is limited by number of layers
- Communication is dominated by the layer w/ the largest output (which can be larger than all of the weights (W) combined)

Approach #2: Sub-image parallelism [1,2]

- Typical approach: one model per quadrant of the image

Fatal flaws:

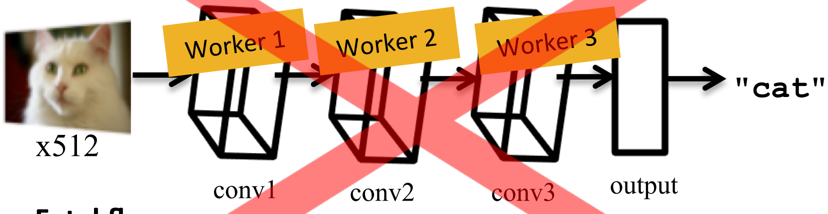
- 4x more FLOPS & 4x more parallelism, BUT:
- No decrease in training time over single-model
- No increase in accuracy over single-model



[1] T. Chilimbi, et al. Project Adam: Building an Efficient and Scalable Deep Learning Training System OSDI, 2014.
[2] J. Dean, et al. Large Scale Distributed Networks. NIPS, 2012.
[3] G. Fedorov, et al. Caffe Training on Multi-node Distributed-memory Systems Based on Intel Xeon Processor E5 Family, 2015.
[4] **F.N. Iandola**, K. Ashraf, M.W. Moskewicz, and K. Keutzer. FireCaffe: near-linear acceleration of deep neural network training on compute clusters. CVPR, 2016.

Four approaches to parallelizing CNN training

Approach #1: Pipeline parallelism



Fatal flaws:

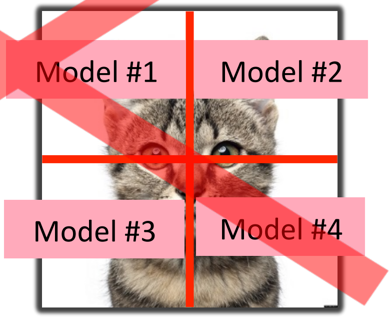
- Parallelism is limited by number of layers
- Communication is dominated by the layer w/ the largest output (which can be larger than all of the weights (W) combined)

Approach #2: Sub-image parallelism [1,2]

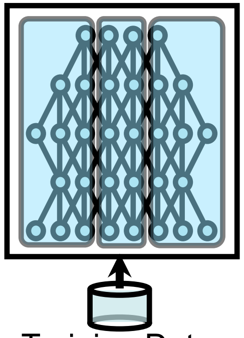
- Typical approach: one model per quadrant of the image

Fatal flaws:

- 4x more FLOPS & 4x more parallelism, BUT:
- No decrease in training time over single-model
- No increase in accuracy over single-model

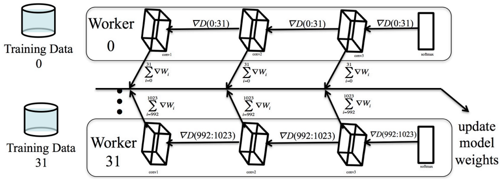


Approach #3: Model parallelism [1,2,3]



- Approach: Give a subset of the weights in each layer to each worker
- This is a scalable approach for some classes of CNNs (will discuss in detail shortly)

Approach #4: Data parallelism [4]

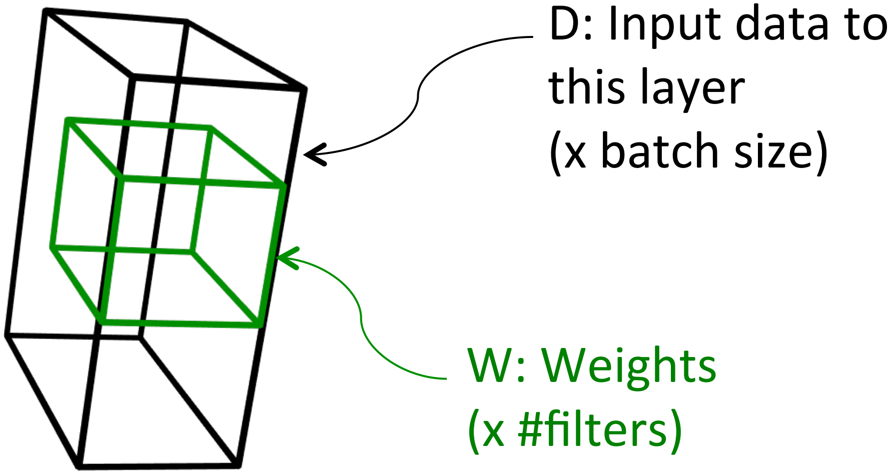


- Approach: Give each worker a subset of the batch
- This is a scalable approach for **convolutional** NNs (will discuss in detail shortly)

[1] T. Chilimbi, et al. Project Adam: Building an Efficient and Scalable Deep Learning Training System OSDI, 2014.
[2] J. Dean, et al. Large Scale Distributed Networks. NIPS, 2012.
[3] G. Fedorov, et al. Caffe Training on Multi-node Distributed-memory Systems Based on Intel Xeon Processor E5 Family, 2015.
[4] **F.N. Iandola**, K. Ashraf, M.W. Moskewicz, and K. Keutzer. FireCaffe: near-linear acceleration of deep neural network training on compute clusters. CVPR, 2016.

Data Parallelism vs. Model Parallelism

Anatomy of a CNN convolutional layer



D: Input data to this layer (x batch size)

W: Weights (x #filters)

- Model parallelism: give a subset of the weights to each worker
- Data parallelism: give a subset of the data to each worker

- Model parallelism is more scalable if: more weights (W) than data (D) per batch
- Data parallelism is more scalable if: more data (D) than weights (W) per batch
  - *GoogLeNet CNN: if we choose data parallelism instead of model parallelism, 360x less communication is required*

Our Approach:
Focus on Synchronous Data Parallelism

- Conventional wisdom: we just need to find enough parallelism dimensions (data parallel, model parallel, pipeline parallel, etc.)
  - Google [1], CMU [2], Microsoft [3], Intel [4], …
- **But,** our experience in scaling applications has taught us…
  - Simple computational/communication mechanisms scale better
  - The key is always to refactor/re-architect the problem to maximally harvest the underlying data parallelism

[1] J. Dean, et al. Large Scale Distributed Networks. NIPS, 2012.

[2] M. Li, et al. Parameter Server for Distributed Machine Learning. NIPSW, 2013.

[3] T. Chilimbi, et al. Project Adam: Building an Efficient and Scalable Deep Learning Training System OSDI, 2014.

[4] G. Fedorov, et al. Caffe Training on Multi-node Distributed-memory Systems Based on Intel Xeon Processor E5 Family, 2015.

Harvesting Data Parallelism

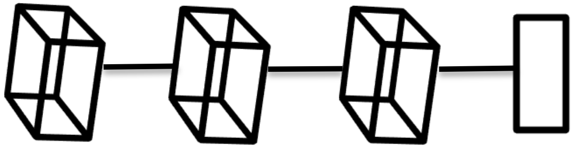
- Requires careful attention to all aspects of the deep learning problem (we've put our "intelligence beans" here):

We will give a taster of this next

Architecting Efficient Distributed Communication

- Hardware network and its logical topology
- Optimizing collective communication (e.g. reductions)
- Quantized/compressed communication of gradient updates

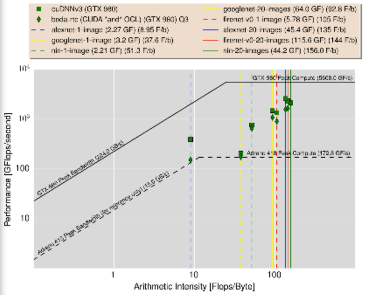
We will discuss in detail toward the end of the talk



Re-architecting CNNs

- CNN architecture
- Batch size and its relationship to other hyperparameters

In our group, Matt Moskewicz owns this

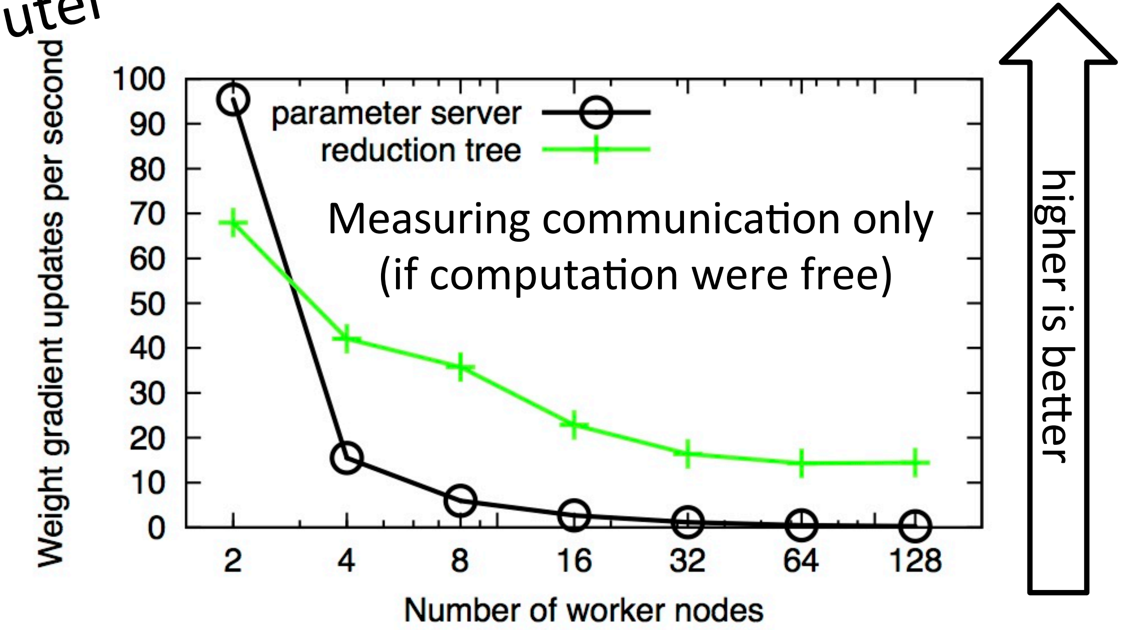


Architecting Efficient Computation

- Careful selection of computational HW/processors
- Code-generation of optimized kernels targeted to specific HW and specific CNN problem sizes
- http://github.com/forresti/convolution
- http://github.com/moskewcz/boda

Scaling up data parallel training: Parameter Server vs. Reduction Tree

Titan supercomputer

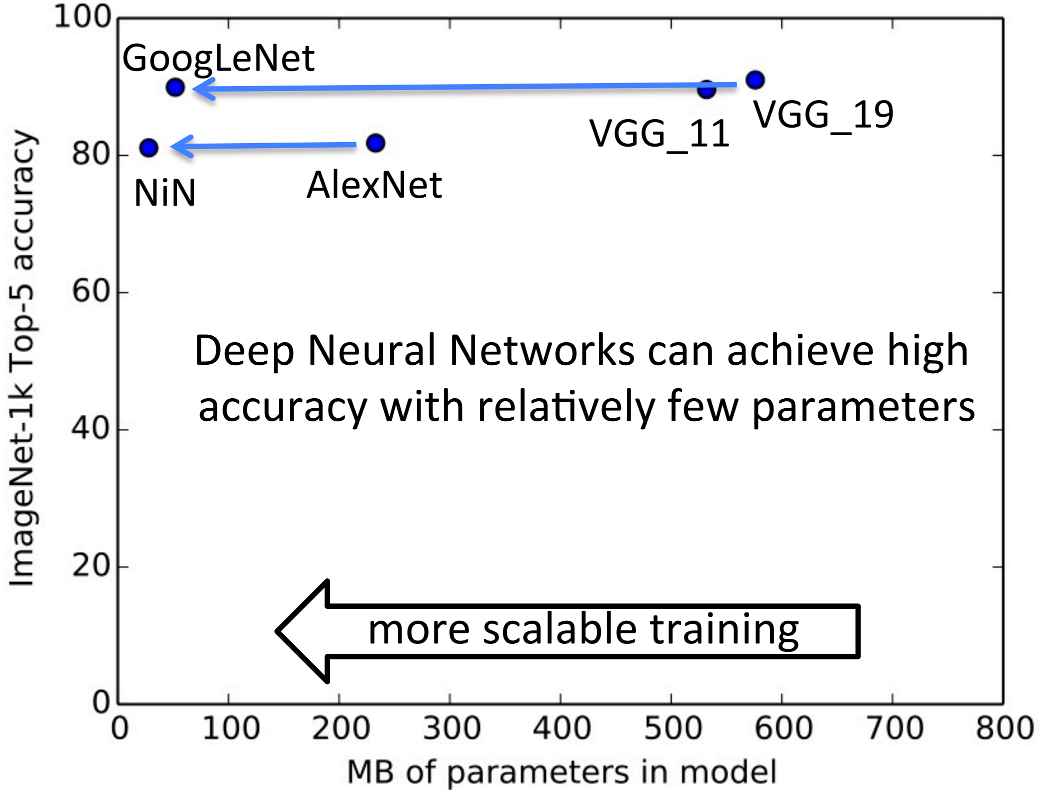


Measuring communication only (if computation were free)

higher is better

- Most related work (e.g. Google DistBelief [1]) uses a parameter server to communicate gradient updates
  - serialized communication: O(# workers) * 53MB for GoogLeNet
- We use an Allreduce (e.g. reduction tree)
  - serialized communication: O(**log**(# workers)) * 53MB for GoogLeNet
  - this is a collective communication operation → requires synchrony

Choosing CNN Architectures to Accelerate



- Data parallelism communication speed:
  - invariant to batch size
  - scales inversely with number of parameters in model
- **Prescriptively:** Fewer parameters → more scalability for data parallelism

Our Current Results

Enormous CNN models (GoogLeNet)

	Hardware	Net	Batch Size	Initial Learning Rate	Epochs	Train time	Speedup	Top-1 ImageNet Accuracy	Top-5 ImageNet Accuracy
Caffe	single K20 GPU	GoogLe-Net	32	0.01	64	**20.5 days**	1x	68.3%	88.7%
FireCaffe (ours)	32 K20 GPUs (Titan cluster)	GoogLe-Net	1024	0.08	72	**23.4 hours**	20x	68.3%	88.7%
FireCaffe (ours)	128 K20 GPUs (Titan cluster)	GoogLe-Net	1024	0.08	72	**10.5 hours**	47x	68.3%	88.7%

At an invitation-only deep learning event in January 2015, with key individuals from Google, Baidu, Facebook, Microsoft, and Twitter in the audience:

Forrest: "has anyone else trained GoogLeNet in 10.5 hours or less?"

Audience: (No.)

See our paper for more details:

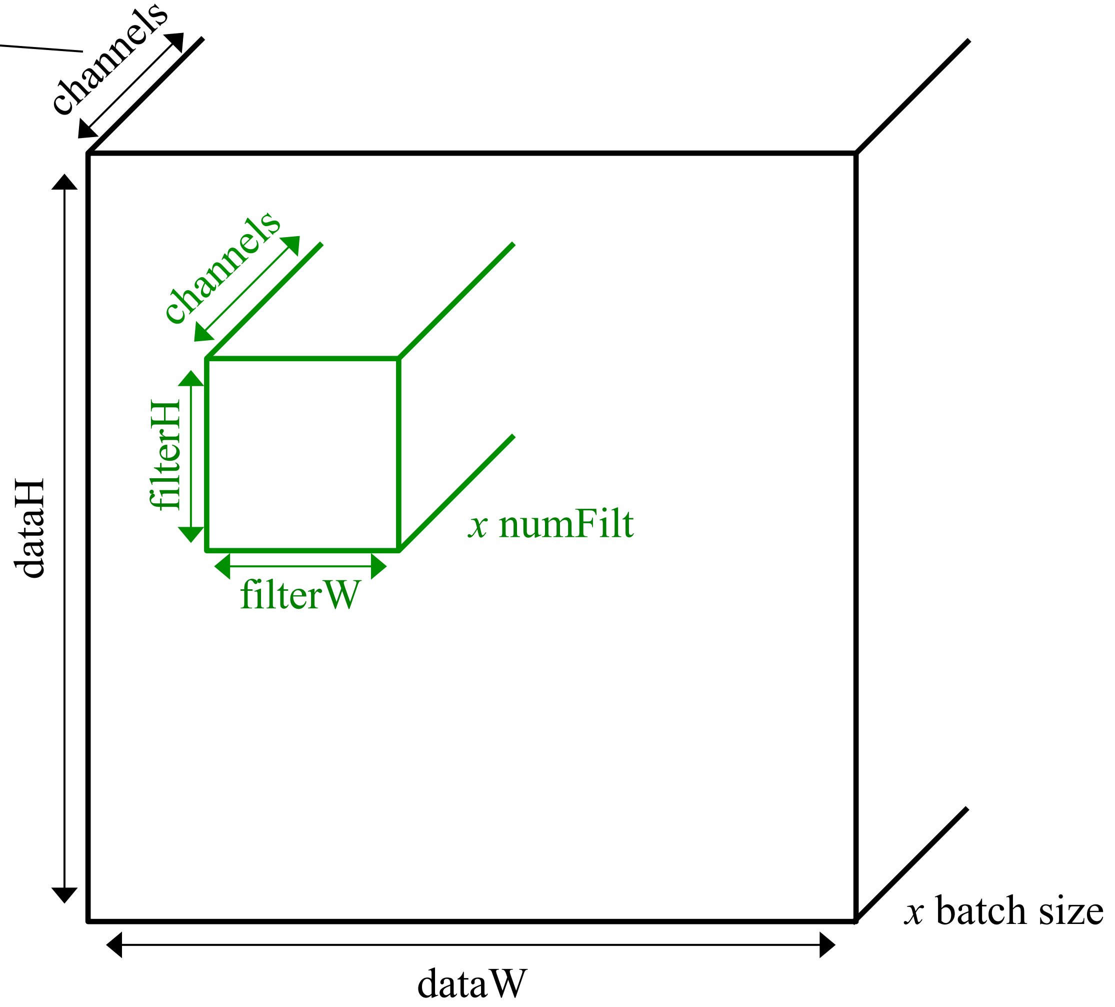
[1] F.N. Iandola, K. Ashraf, M.W. Moskewicz, and K. Keutzer. FireCaffe: near-linear acceleration of deep neural network training on compute clusters. CVPR, 2016.

Theme 3: Defining and describing the design space of CNNs

Highlighting some key tradeoffs in the design space of CNNs

Reminder: Dimensions of a convolution layer

The number of channels in the current layer is determined by the number of filters (numFilt) in the previous layer.



Bear in mind that CNNs are comprised of *many* layers, L_1 , …, L_n

Local and *Global* changes to CNN architectures

Examples of Local changes to CNNs

- Change the number of channels in the input data
- Change the number of filters in a layer
- Change the resolution of the filters in a layer (e.g. 3x3 → 6x6)
- Change the number of categories to classify

Effect of a Local change:

A Local change to layer L_i only affects the dimensions of layers L_i and L_{i+1}

Examples of Global changes to CNNs

- Change the strides or downsampling/upsampling in a layer
- Change the height and width of the input data (e.g. 640x480 → 1920x1440 image)

Effect of a Global change:

A Global change to layer L_i affects the dimensions of all downstream layers: L_{i+1} , …, L_n

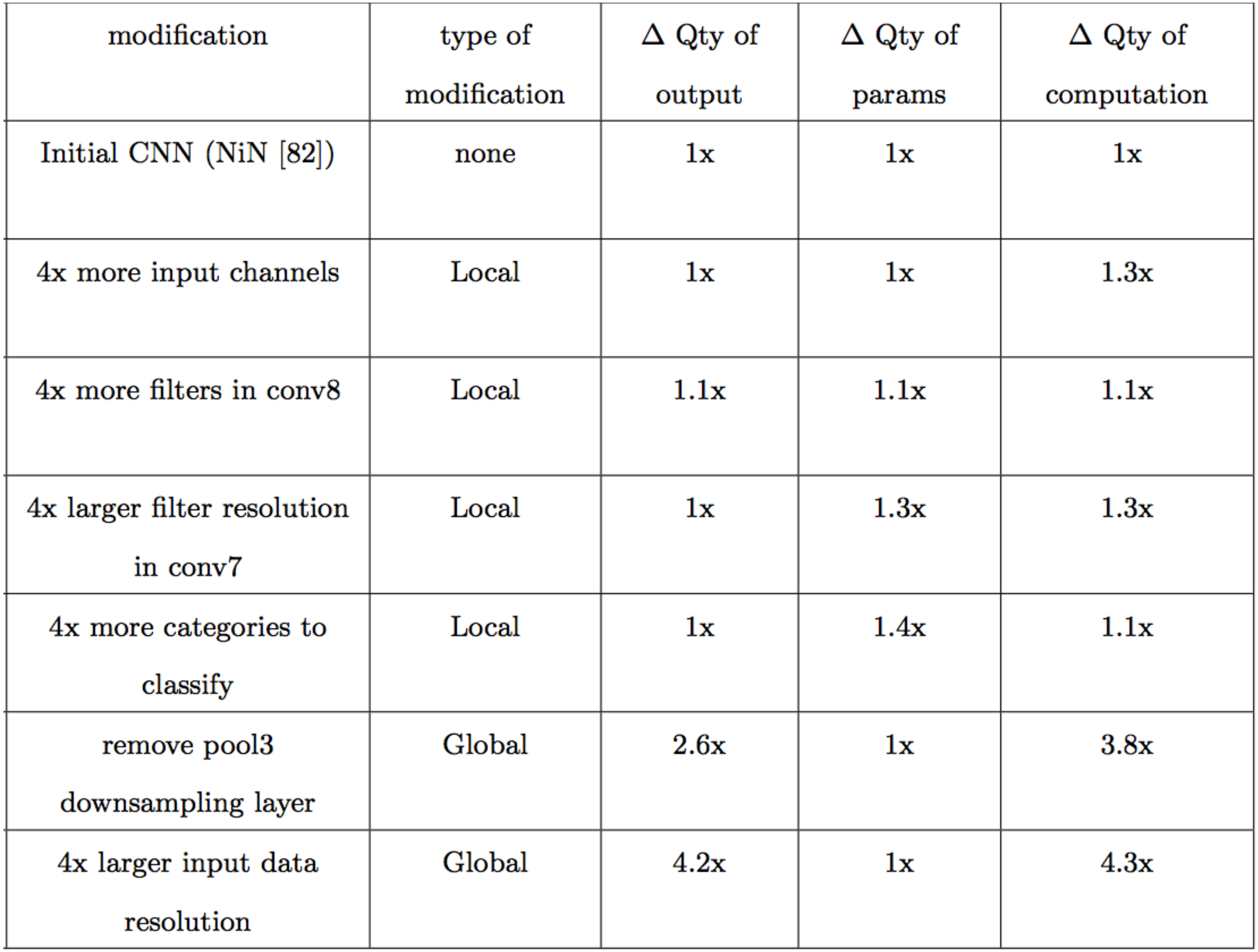
Effect of *Local* and *Global* changes to parallelism during CNN training

Recall the distributed data-parallel approach to training that we described earlier in the talk.

- Local changes involve modifying the dimensions of filters or channels.
  - Consider a local change where we increase the number of filters in a layer
  - Effect on training: This local change increases the number of parameters and therefore **increases the quantity of communication** required during training.

- Global changes involve modifying the dimensions of data or activations (i.e. the output data from the layers)
  - Consider a global change where we decrease the stride of a layer (increasing the number of activations it produces)
  - Effect on training: This global change does not affect the number of parameters and therefore **does not change the quantity of communication** required during training.

Effect of *Local* and *Global* changes to parallelism during CNN training



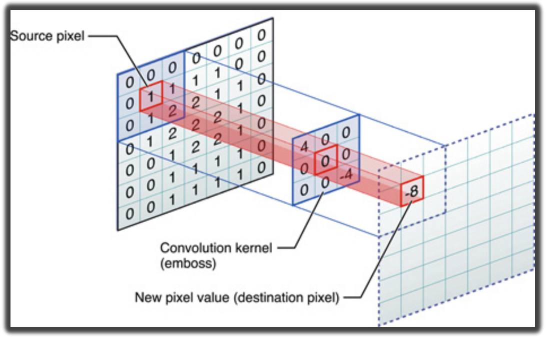
modification	type of modification	Δ Qty of output	Δ Qty of params	Δ Qty of computation
Initial CNN (NiN [82])	none	1x	1x	1x
4x more input channels	Local	1x	1x	1.3x
4x more filters in conv8	Local	1.1x	1.1x	1.1x
4x larger filter resolution in conv7	Local	1x	1.3x	1.3x
4x more categories to classify	Local	1x	1.4x	1.1x
remove pool3 downsampling layer	Global	2.6x	1x	3.8x
4x larger input data resolution	Global	4.2x	1x	4.3x

Theme 4: Exploring the design space of CNNs

Let's take what we've learned so far and *start exploring!*

FireNet: CNN architectures with few weights

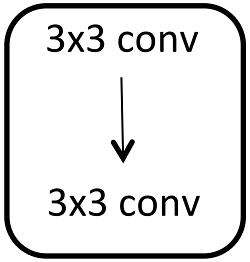
Image convolution in 2D



- Fewer weights in model → more scalability in training
- Observation: 3x3 filters contain 9x more weights and require 9x more computation than 1x1 filters.
- SqueezeNet is one example of a FireNet architecture

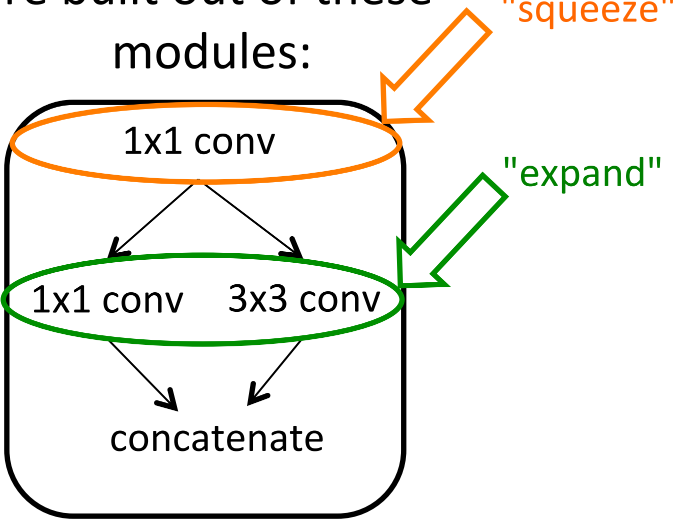
VGG [1] CNNs

are built out of these modules:



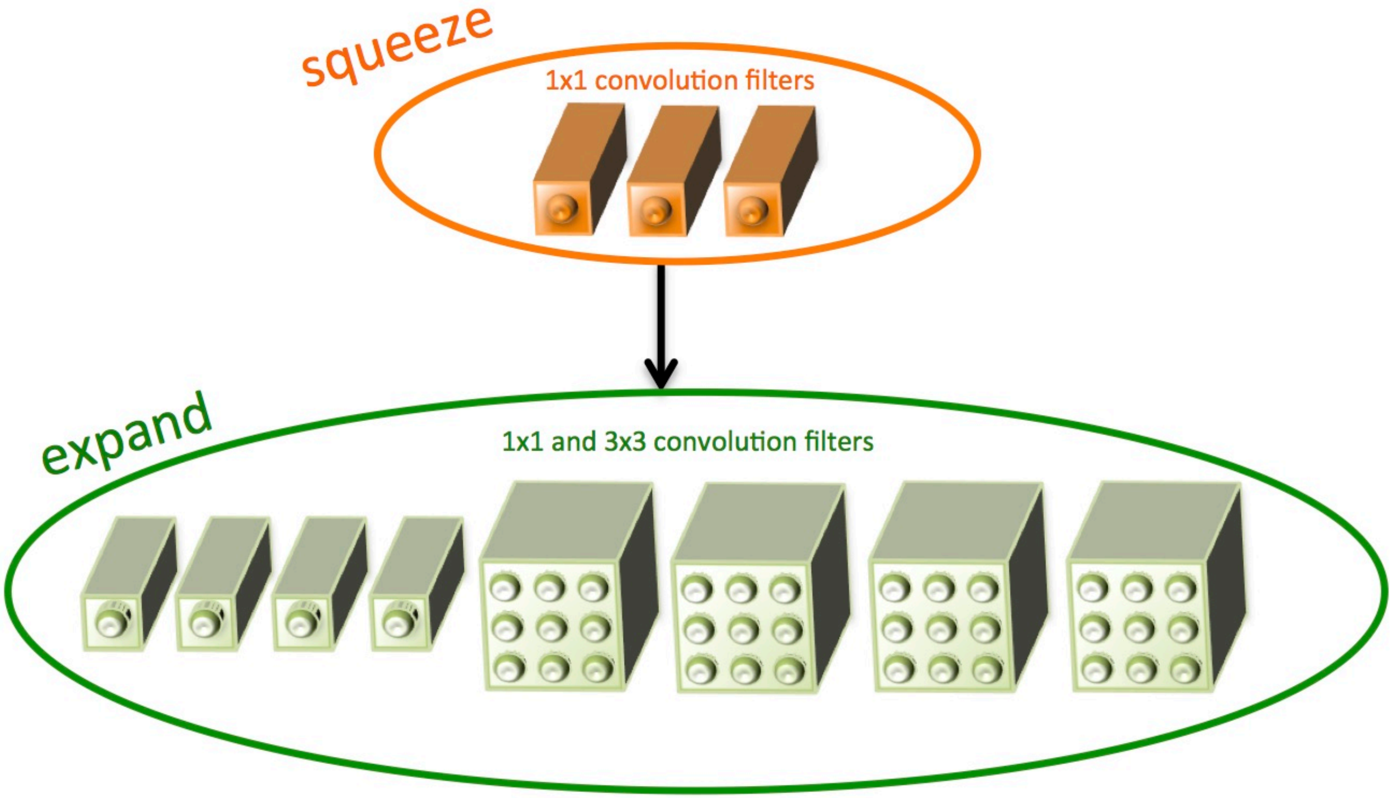
Our FireNet CNNs

are built out of these modules:

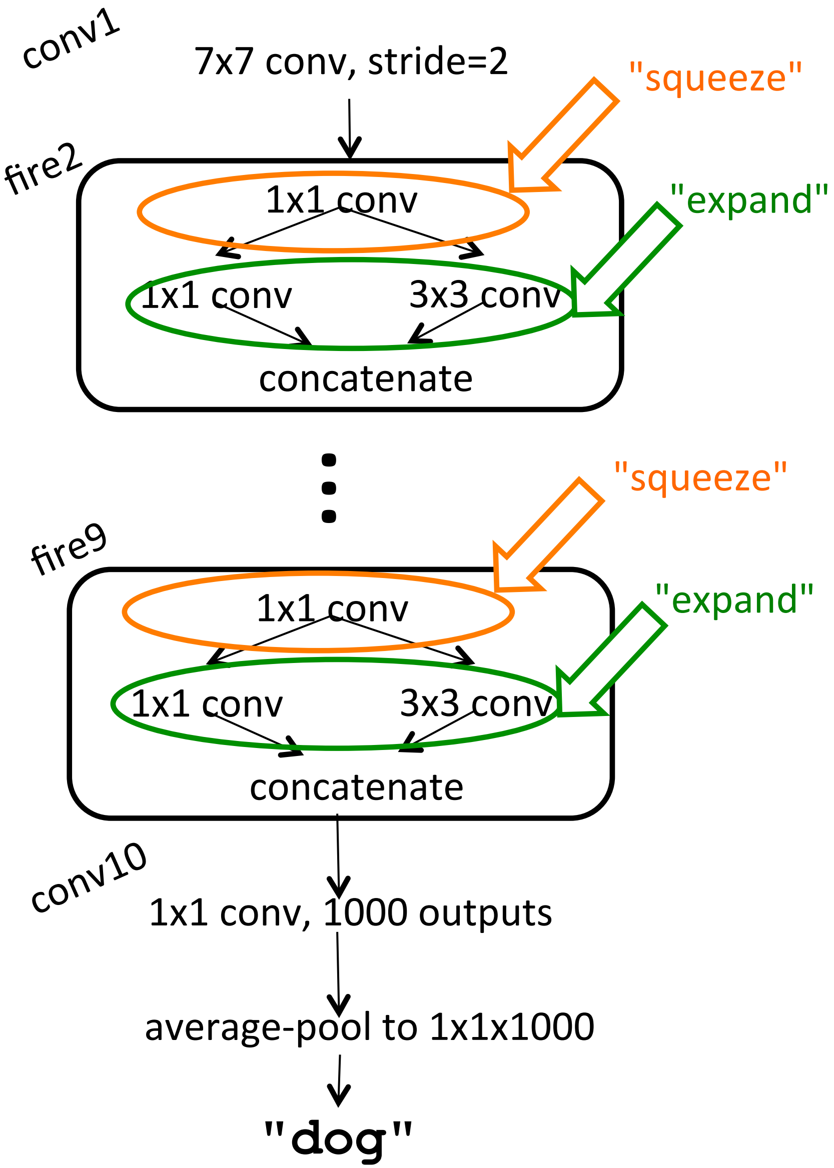


[1] K. Simonyan, A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition arXiv:1409.1556, 2014.

Fire Module in Detail



An Example FireNet CNN Architecture



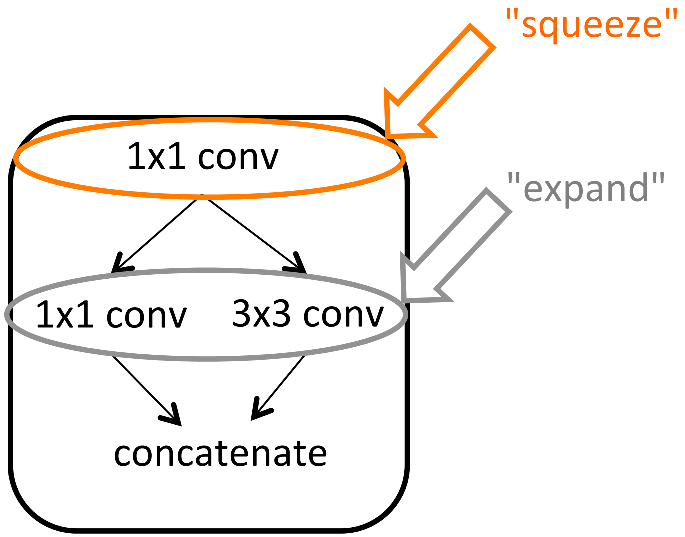
Convolution

- In "expand" layers half the filters are 1x1; half the filters are 3x3

Pooling

- Maxpool after conv1, fire4, fire8 (3x3 kernel, stride=2)
- Global Avgpool after conv10 down to 1x1x1000

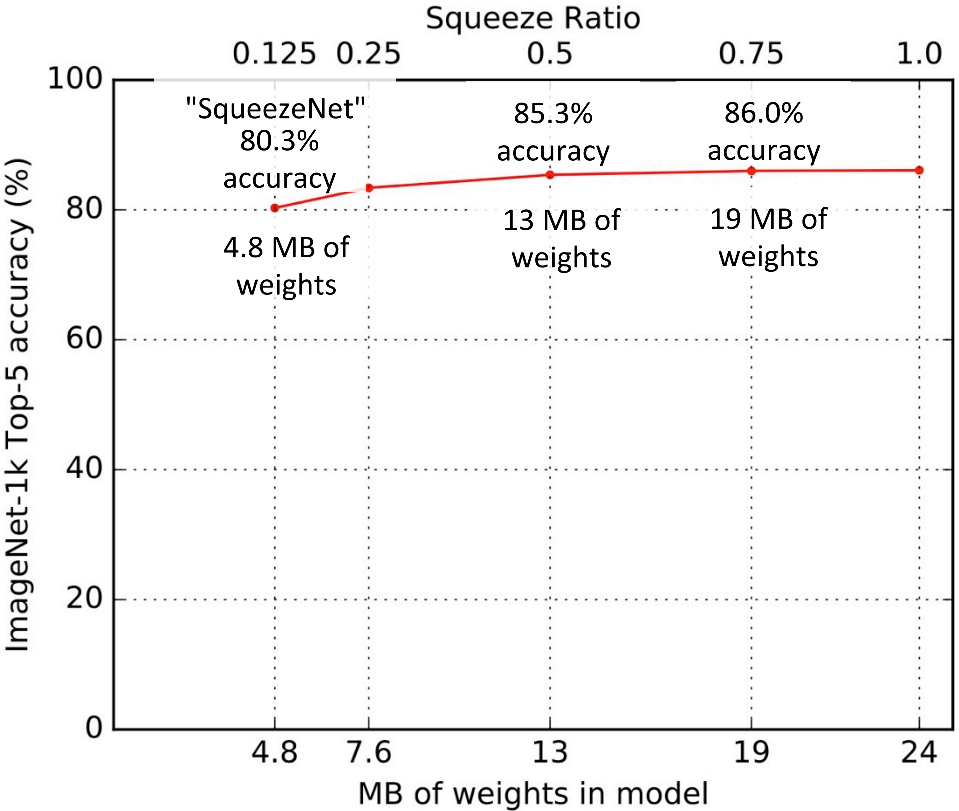
Tradeoffs in "squeeze" modules



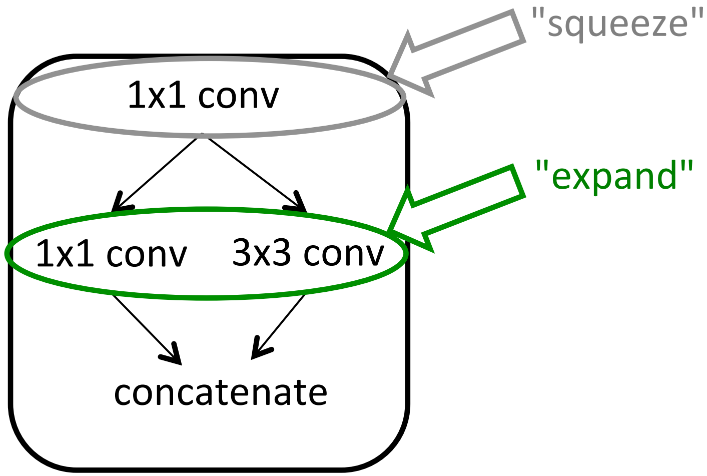
- The "squeeze" modules get their name because they have fewer filters (i.e. fewer output channels) than the "expand" modules
- A natural question: what tradeoffs occur when we vary the degree of squeezing (number of filters) in the "squeeze" modules?

S = number of filters in "squeeze"
E = number of filters in "expand"
***Squeeze Ratio** = S/E*
for a predetermined number of filters in E

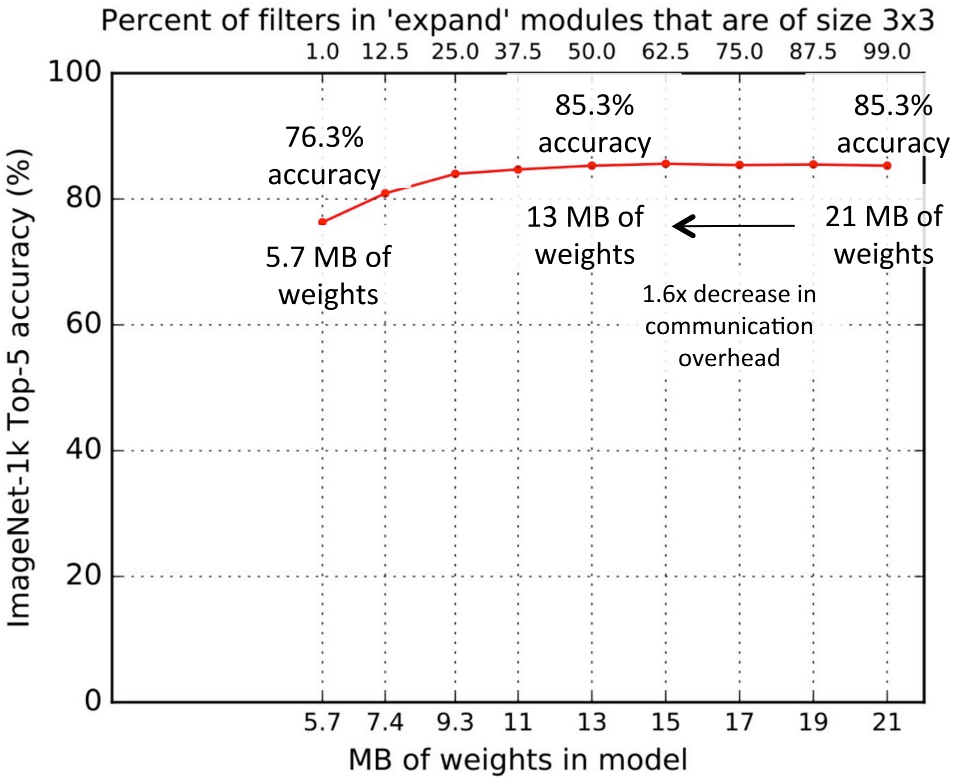
In these experiments: the *expand* modules have 50% 1x1 and 50% 3x3 filters



Judiciously using 3x3 filters in "expand" modules

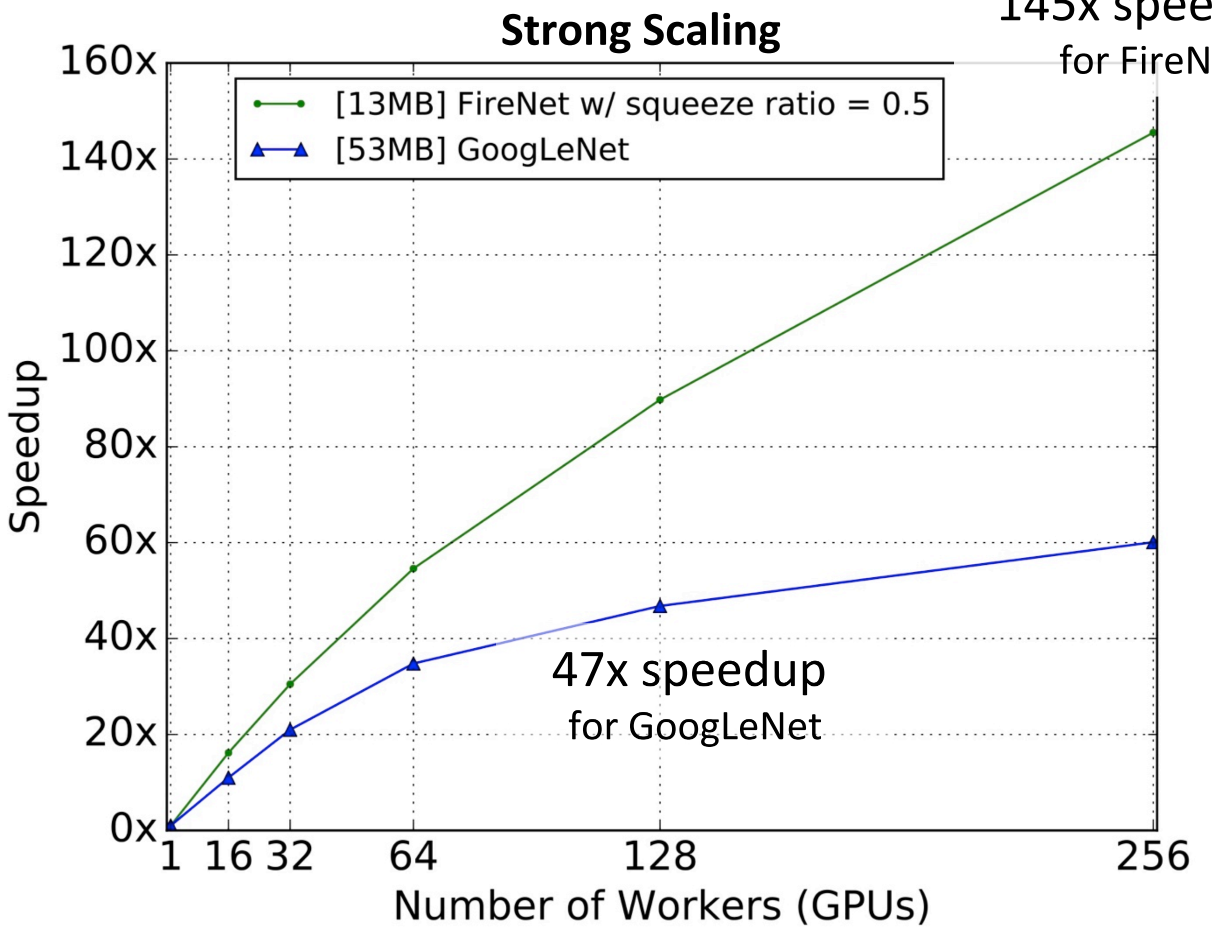


- In the "expand" modules, what are the tradeoffs when we turn the knob between mostly 1x1 and mostly 3x3 filters?
- Hypothesis: if having more weights leads to higher accuracy, then having *all* 3x3 filters should give the highest accuracy
- Discovery: accuracy plateaus with 50% 3x3 filters



Each point on this graph is the result of training a unique CNN architecture on ImageNet.

Fewer Weights in CNN→ More Scalability in Training



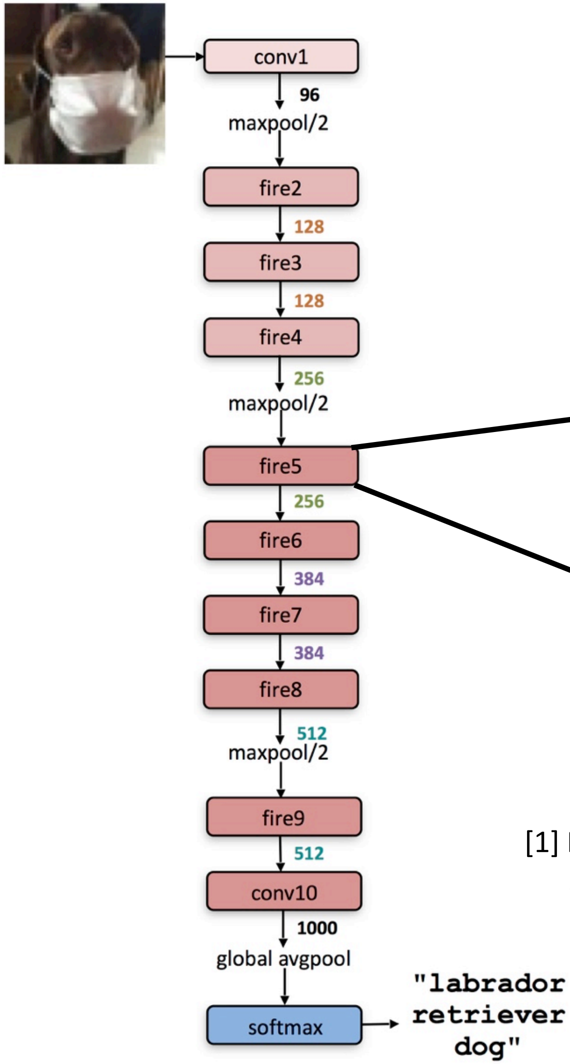
145x speedup
for FireNet

47x speedup
for GoogLeNet

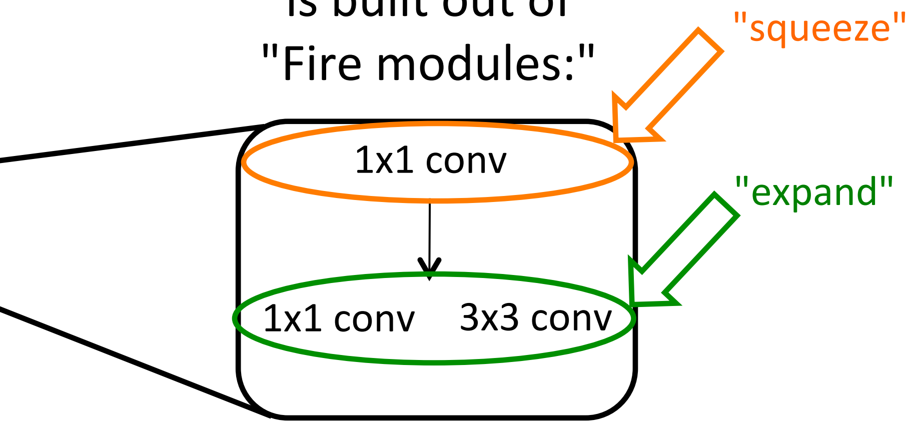
Using FireCaffe on
the Titan cluster

One of our new CNN architectures: *SqueezeNet*

The SqueezeNet Architecture [1]



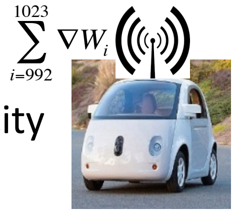
SqueezeNet is built out of "Fire modules:"



[1] **F.N. Iandola**, M. Moskewicz, K. Ashraf, S. Han, W. Dally, K. Keutzer. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <1MB model size. arXiv, 2016.

http://github.com/DeepScale/SqueezeNet

SqueezeNet vs. Related Work



- **Small DNN models** are important if you…
  - Are deploying DNNs on devices with limited memory bandwidth or storage capacity
  - Plan to push frequent model updates to clients (e.g. self-driving cars or handsets)
- The model compression community often targets AlexNet as a DNN model to compress

Compression Approach	DNN Architecture	Original Model Size	Compressed Model Size	Reduction in Model Size vs. AlexNet	Top-1 ImageNet Accuracy	Top-5 ImageNet Accuracy
None (baseline)	AlexNet [1]	240MB	240MB	1x	57.2%	80.3%
SVD [2]	AlexNet	240MB	48MB	5x	56.0%	79.4%
Network Pruning [3]	AlexNet	240MB	27MB	9x	57.2%	80.3%
Deep Compression [4]	AlexNet	240MB	6.9MB	35x	57.2%	80.3%
None	**SqueezeNet [5] (ours)**	**4.8MB**	**4.8MB**	**50x**	57.5%	80.3%
Deep Compression [4]	**SqueezeNet [5] (ours)**	**4.8MB**	**0.47MB**	**510x**	57.5%	80.3%

[1] A. Krizhevsky, I. Sutskever, G.E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. NIPS, 2012.

[2] E.L .Denton, W. Zaremba, J. Bruna, Y. LeCun, R. Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. NIPS, 2014.

[3] S. Han, J. Pool, J. Tran, W. Dally. Learning both Weights and Connections for Efficient Neural Networks, NIPS, 2015.

[4] S. Han, H. Mao, W. Dally. Deep Compression…, arxiv:1510.00149, 2015.

[5] **F.N. Iandola**, **M. Moskewicz**, K. Ashraf, S. Han, W. Dally, **K. Keutzer**. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <1MB model size. arXiv, 2016.

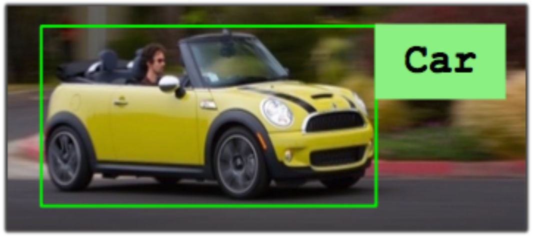
(Bonus!) Theme 5: Effectively deploying the new CNNs

Joint work with Bichen Wu. This will go in his dissertation!

Adapting SqueezeNet for object detection

Object *classification*

Object *detection*



- We originally trained SqueezeNet on the problem of *object classification*
- The highest-accuracy results on *object detection* involve pre-training a CNN on *object classification* and then fine-tuning it for *object detection*
    - Modern approaches (e.g. Faster R-CNN, YOLO) typically modify design of the CNN's final layers so that it can *localize* as well as *classify* objects

Next: Let's give this a try with SqueezeNet!

Adapting SqueezeNet for object detection

Method	Average Precision on KITTI [1] pedestrian detection	Mean Average Precison on KITTI [1] object detection dataset	Model Size	Speed (FPS) on Titan X GPU
MS-CNN [2]	85.0	78.5	-	2.5 FPS
Pie [3]	84.2	69.6	-	10 FPS
Shallow Faster R-CNN [4]	82.6	-	240 MB	2.9 FPS
SqueezeDet [5] (ours)	82.9	76.7	**7.90 MB**	**57.2 FPS**
SqueezeDet+ [5] (ours)	**85.4**	**80.4**	26.8 MB	32.1 FPS

[1] A. Geiger, P. Lenz, R. Urtasun. Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite. CVPR, 2012

[2] Z. Cai, Q. Fan, R. Feris, N. Vasconcelos. A unified multi-scale deep convolutional neural network for fast object detection. ECCV, 2016.

[3] Anonymous submission on the KITTI leaderboard.

[4] K. Ashraf, B. Wu, **F.N. Iandola**, M.W. Moskewicz, K. Keutzer. **Shallow Networks** for High-Accuracy Road Object-Detection. arXiv: 1606.01561, 2016.

[5] Bichen Wu, **Forrest Iandola**, Peter Jin, Kurt Keutzer, "**SqueezeDet**: Unified, Small, Low Power Fully Convolutional Neural Networks for Real-Time Object Detection for Autonomous Driving," In Review, 2016.

Summary

- Theme 1: Defining benchmarks and metrics to evaluate CNN/DNNs
  - Accuracy, speed, energy, model size, and other metrics can all play a critical role in evaluating whether a CNN is ready for practical deployment
- Theme 2: Rapidly training CNN/DNNs
  - 47x speedup on 128 GPUs
- Theme 3: Defining and describing the CNN/DNN design space
  - A condensed mental model for deciding how a modification to a CNN will impact distributed training time
- Theme 4: Exploring the design space of CNN/DNN architectures
  - Discovered a new CNN that is 500x smaller than a widely-used CNN with equivalent accuracy
- Bonus! Theme 5: Effectively deploying the new CNNs
  - Defining the state of the art on KITTI object detection in terms of *speed, model size, and accuracy!*

Summary

- Theme 1: Defining benchmarks and metrics to evaluate CNN/DNNs
  - Accuracy, speed, energy, model size, and other metrics can all play a critical role in evaluating whether a CNN is ready for practical deployment
- Theme 2: Rapidly training CNN/DNNs
  - 47x speedup on 128 GPUs
- Theme 3: Defining and describing the CNN/DNN design space
  - A condensed mental model for deciding how a modification to a CNN will impact distributed training time
- Theme 4: Exploring the design space of CNN/DNN architectures
  - Discovered a new CNN that is 500x smaller than a widely-used CNN with equivalent accuracy
- Bonus! Theme 5: Effectively deploying the new CNNs
  - Defining the state of the art on KITTI object detection in terms of *speed, model size, and accuracy!*

What am I doing next?

DEEPSCALE

Perception for autonomous vehicles

We're hiring… email me if you'd like to chat about this. ☺
`forrest@deepscale.ai`

Questions?