10-605/10-805: Machine Learning with Large Datasets

Fall 2022

Inference & Model Compression

Announcements

- HW4: please shut down all AWS resources!
- Mini-project Proposals due on 11/11 (this Friday!)
- Recitation this week: HW4 review
- **Guest lecture Thursday (11/17)**: Krishna Rangasayee SiMa.ai
  - Topic: ML on Embedded Edge Devices)
  - Virtual / Zoom
  - Will not be recorded — please show up!

WHERE ARE WE NOW?

Key course topics

Data preparation

- Data cleaning
- Data summarization
- Visualization
- Dimensionality reduction

Training

- Distributed ML
- Large-scale optimization
- Scalable deep learning
- Efficient data structures
- Hyperparameter tuning

Inference

- Hardware for ML
- Techniques for low-latency inference (compression, pruning, distillation)

Infrastructure / Frameworks

- Apache Spark
- TensorFlow
- AWS / Google Cloud / Azure

Advanced topics

- Federated learning
- Neural Architecture Search
- Productionizing ML

WHERE ARE WE NOW?

Key course topics

Data preparation
- Data cleaning
- Data summarization
- Visualization
- Dimensionality reduction

Training
- Distributed ML
- Large-scale optimization
- Scalable deep learning
- Efficient data structures
- Hyperparameter tuning

Inference
- Hardware for ML
- Techniques for low-latency inference (compression, pruning, distillation)

Infrastructure / Frameworks
- Apache Spark
- TensorFlow
- AWS / Google Cloud / Azure

Advanced topics
- Federated learning
- Neural Architecture Search

MOVING FORWARD

What makes deep learning expensive?

Training requires lots of computation

- Specialized hardware (GPUs, TPUs) can help with matrix computations
- Can use advanced iterative optimization methods
- Can parallelize training

Hyperparameter tuning and neural architecture search (NAS) make this worse

- Lots of knobs to tune!

Resulting models can be large!

- Can be expensive to store model, perform inference

MOVING FORWARD

What makes deep learning expensive?

Training requires lots of computation

- Specialized hardware (GPUs, TPUs) can help with matrix computations
- Can use advanced iterative optimization methods
- Can parallelize training

Hyperparameter tuning and neural architecture search (NAS) make this worse

- Lots of knobs to tune!

Resulting models can be large!

- Can be expensive to store model, perform inference

MOVING FORWARD

What makes deep learning expensive?

Training requires lots of computation

- Specialized hardware (GPUs, TPUs) can help with matrix computations
- Can use advanced iterative optimization methods
- Can parallelize training

Hyperparameter tuning and neural architecture search (NAS) make this worse

- Lots of knobs to tune!

Resulting models can be large!

- Can be expensive to store model, perform inference

MOVING FORWARD

What makes deep learning expensive?

Training requires lots of computation

- Specialized hardware (GPUs, TPUs) can help with matrix computations
- Can use advanced iterative optimization methods
- Can parallelize training

Hyperparameter tuning and neural architecture search (NAS) make this worse

- Lots of knobs to tune!

Resulting models can be large!

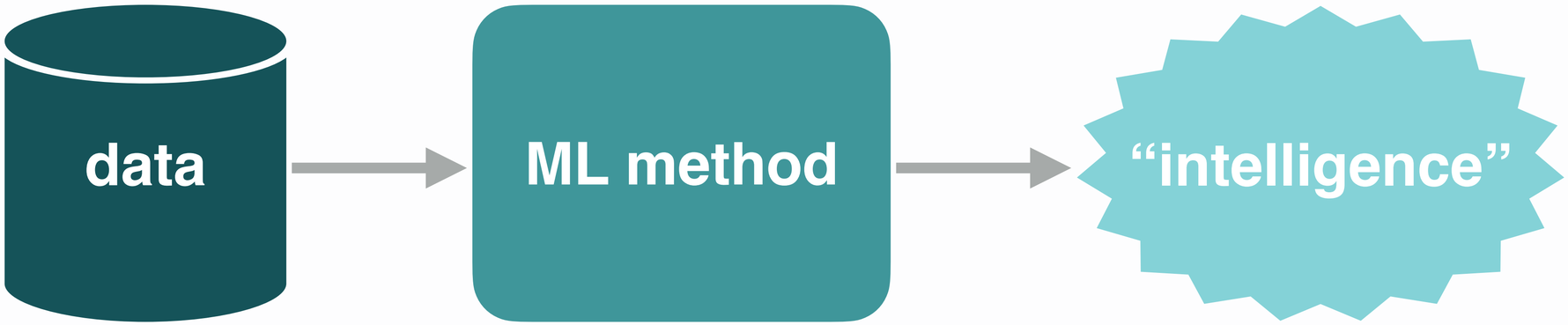
- Can be expensive to store model, perform inference

Outline

1. Why inference?
2. Techniques for model compression

A SIMPLIFIED

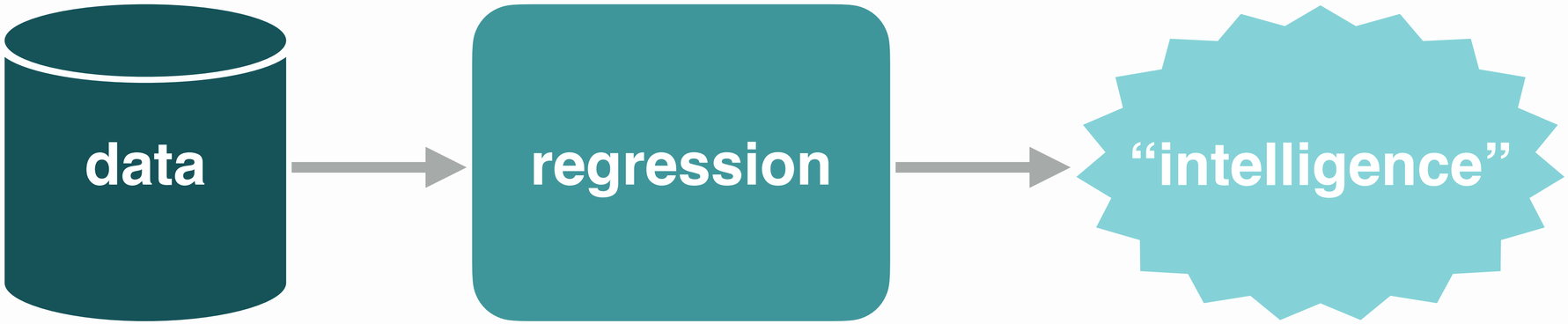
Machine learning pipeline

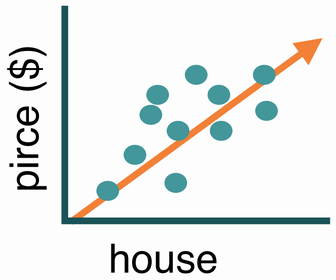


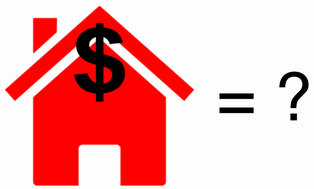
EXAMPLE 1:

Regression

How much should you sell your house for?







input: houses & features

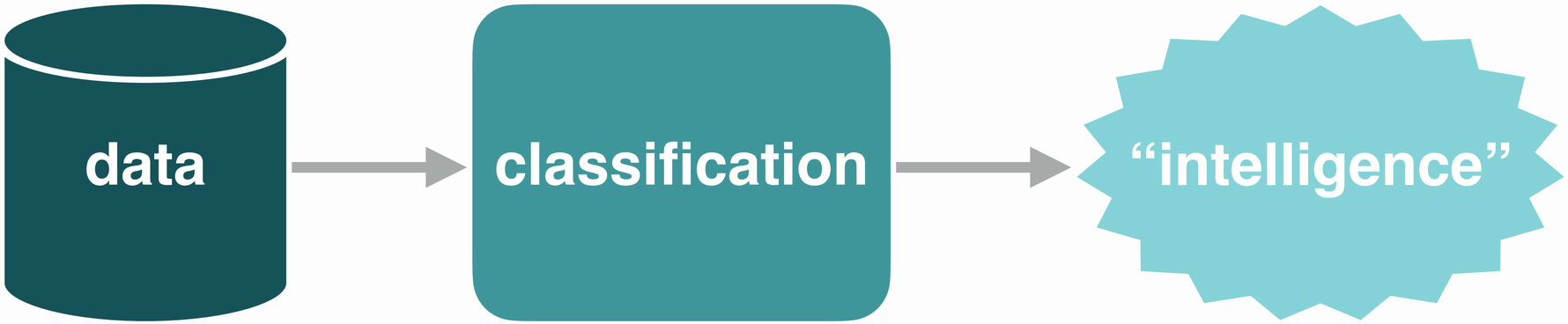
learn: x → y relationship

predict: y (continuous)

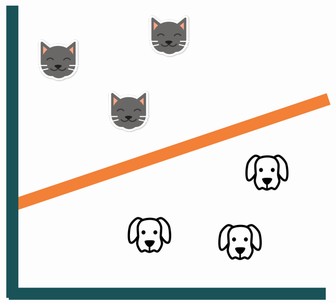
EXAMPLE 2:

Classification

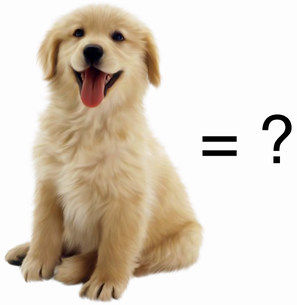
Cat or dog?



input: cats & dogs



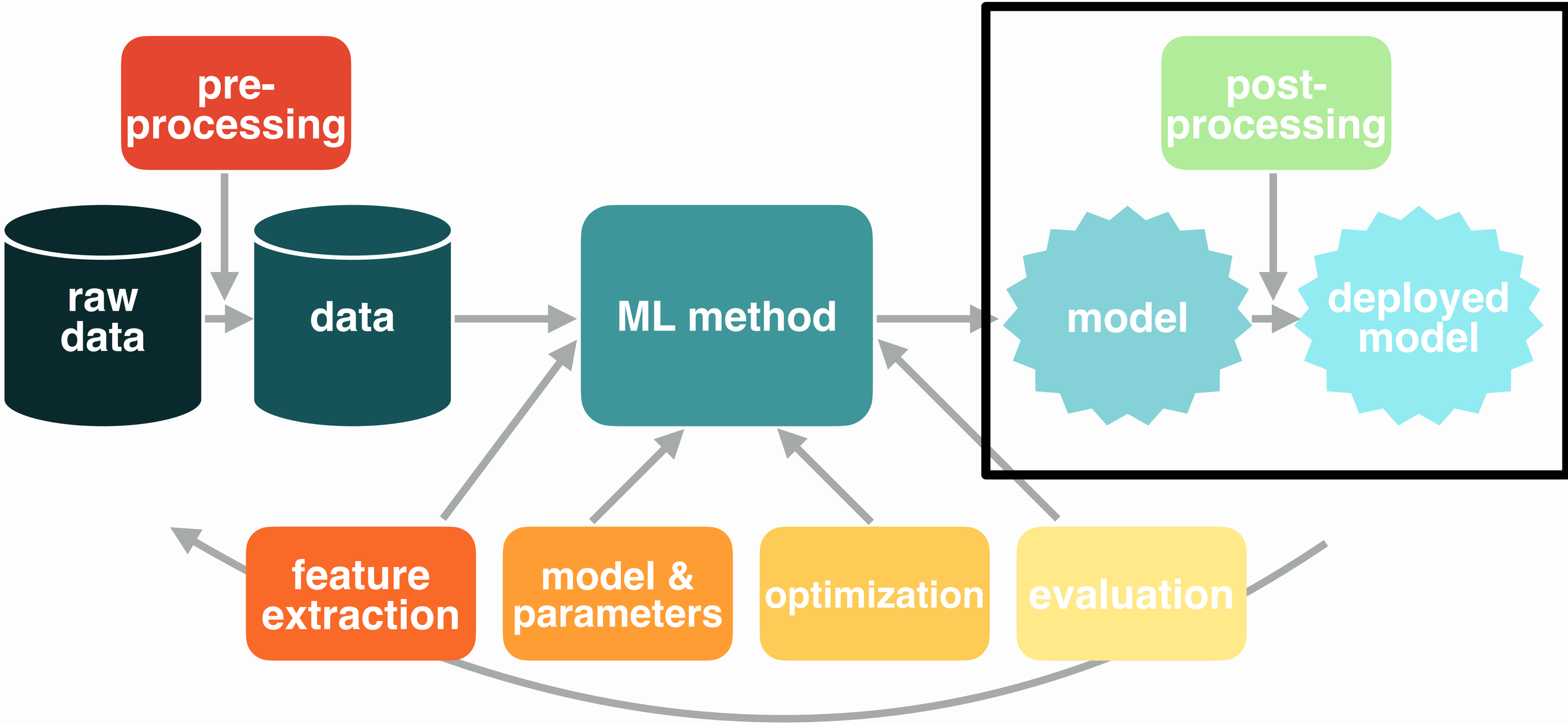
learn: x → y relationship



predict: y (categorical)

A MORE REALISTIC

Machine learning pipeline



Inference: Why should we care?

Train once, infer many times

- Many production machine learning systems just do inference

Often want to run inference on low-power edge devices

- Cell phones, wearable devices, security cameras
- Limited memory on these devices to store models

Need to predict quickly

- E.g., for web applications, want to respond to users in realtime

RECALL

Empirical risk minimization

$$\hat{f}_n = \arg\min_{f \in F} \frac{1}{n} \sum_{i=1}^{n} \ell(f(x_i), y_i)$$

- A popular approach in supervised ML
- Given a loss ℓ and data (x_1, y_1), … (x_n, y_n), we estimate a predictor *f* by minimizing the *empirical risk*
- We typically restrict this predictor to lie in some class, *F*
  - Could reflect our prior knowledge about the task
  - Or may be for computational convenience

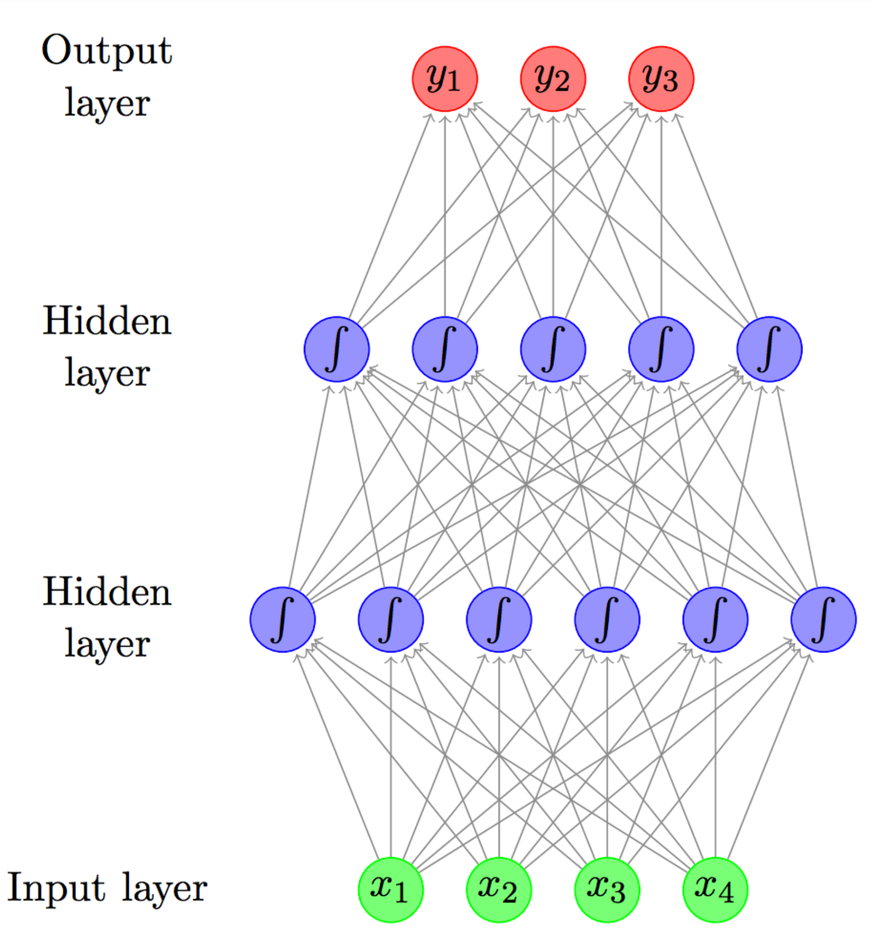
inference involves computing: $\hat{y} = f(\hat{x})$ *for some new observation* $\hat{x}$

COST OF INFERENCE

Feedforward neural networks

We can consider adding L hidden layers, each with their own activation functions σ_l and corresponding weights $\mathbf{W}_l$

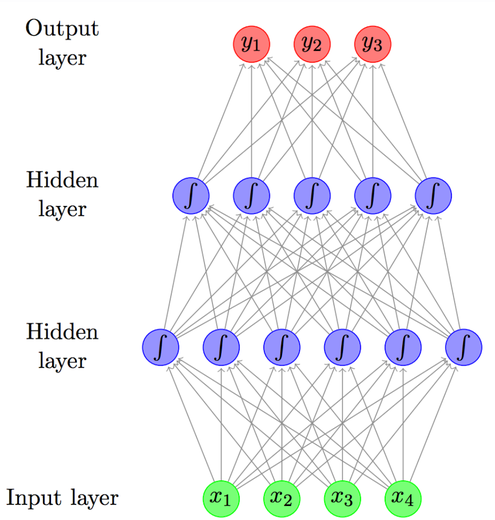
Using this layering technique, the resulting model becomes:



$$f(\mathbf{x}) = \mathbf{w}^\top \sigma_L(\mathbf{W}_L^\top(\sigma_{L-1}(\mathbf{W}_{L-1}^\top \ldots \sigma_2(\mathbf{W}_2^\top \sigma_1(\mathbf{W}_1^\top \mathbf{x}))\ldots)$$

COST OF INFERENCE

Feedforward neural networks



Can express NN model with the following recurrence:

$$\mathbf{o}_0 = \mathbf{x}$$

$$\forall l \in \{1,\ldots,L\},\ \mathbf{h}_l = \mathbf{W}_l^\top \mathbf{o}_l$$

$$\forall l \in \{1,\ldots,L\},\ \mathbf{o}_l = \sigma_l(\mathbf{h}_l)$$

$$f(\mathbf{x}) = \mathbf{o}_L$$

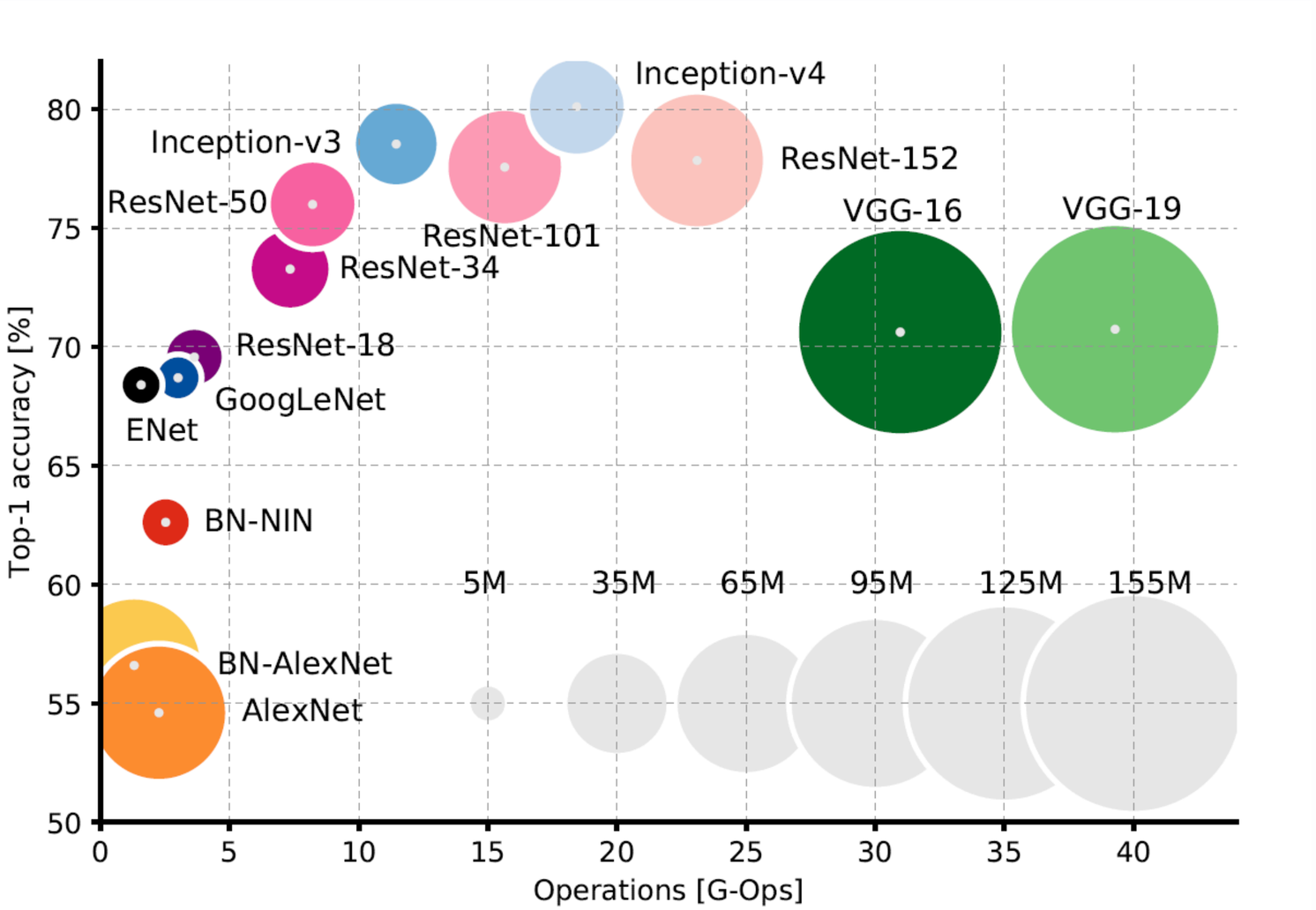
What dominates the cost?

Matrix(-vector) multiplies

O(nm) for n x m matrix, m x 1 vector

COST OF INFERENCE

Deep learning



- Operations required for one forward pass vs. accuracy
- Circle size proportional to the number of network parameters
- Correlation between operation count & accuracy

Canziani et al, 2016

Inference metrics

Accuracy

- How accurate is the deployed model at making predictions?

Model size

- How many bytes are needed to store or transmit the model?

Latency

- How much time does it take to make a prediction for a single observation?

Throughput

- How many observations we can predict within some amount of time?

Energy

- How much energy do we use to produce each prediction?

Cost

- How much money does it cost to deploy the model / perform inference?

Where do we perform inference?

The cloud (e.g., AWS, GCP)
Common concerns: latency, cost

User devices (e.g., phone, laptop)
Common concerns: memory/model size

IoT devices (e.g., wearable device, sensor)
Common concerns: energy, memory

Where do we perform inference?

Cortex M0	Cortex M4	Cortex M7	Raspberry Pi Zero 1Ghz ARM11	Raspberry Pi 3 4x1.2GHz Cortex A53	Smartphone 4x2.3 GHz Mongoose & 4x1.6 GHz Cortex A53
50 DMIPS	200 DMIPS	850 DMIPS	1250 DMIPS	11,000 DMIPS	50,000 DMIPS
32 KB	128 KB	384 KB	512 MB	1 GB	4 GB

Credit: Ofer Dekel

Outline

1. Why inference?
2. Techniques for model compression

Model compression

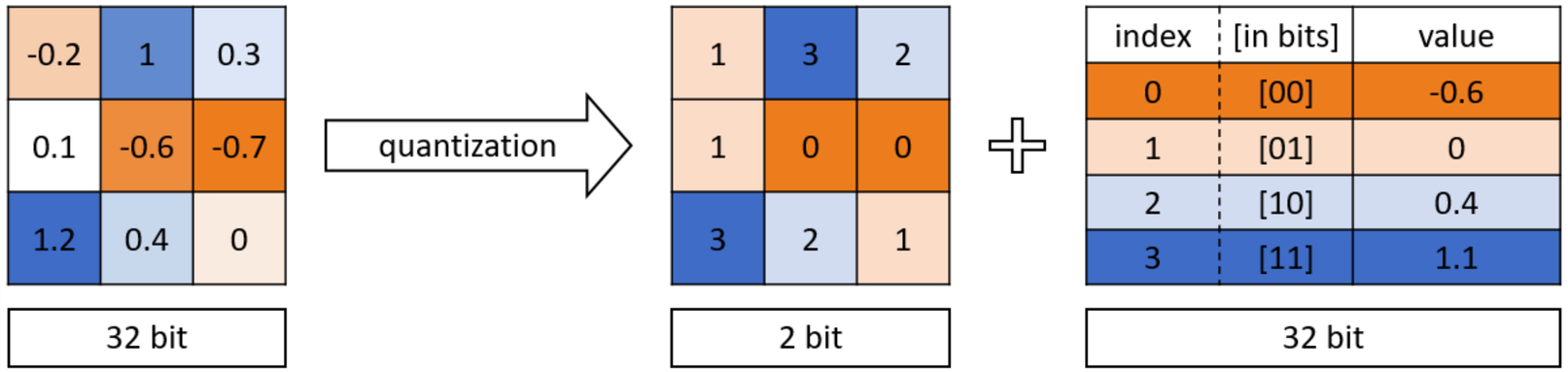
Goal: find a smaller, cheaper model with the same (or similar) accuracy

Many techniques:

- Quantization (low-precision inference)
- Pruning
- Knowledge distillation
- Efficient architectures (+ architecture search)

Low precision / Quantized NNs

- Simple technique: use low-precision arithmetic in inference
- Can make any signals in the model low precision
- How to determine the ‘right’ amount of precision?
  - One heuristic: keep lowering the precision of signals until the accuracy decreases
- Extreme example: binarization (1-bit {-1,1} or {0,1})



Low precision / Quantized NNs

“XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks”, Rastegari et al, ECCV 2016

Classification Accuracy(%)									
Binary-Weight				Binary-Input-Binary-Weight				Full-Precision	
BWN		BC[11]		XNOR-Net		BNN[11]		AlexNet[1]	
Top-1	Top-5	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5
56.8	**79.4**	35.4	61.0	**44.2**	**69.2**	27.9	50.42	56.6	80.2

binary weights → ~32x savings in memory

Low precision / Quantized NNs

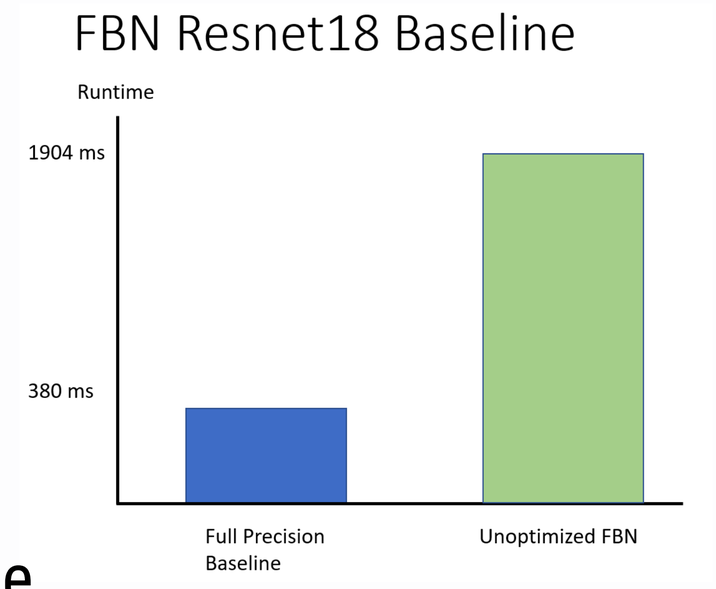
"XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks",
Rastegari et al, ECCV 2016

	ResNet-18		GoogLenet	
Network Variations	top-1	top-5	top-1	top-5
Binary-Weight-Network	60.8	83.0	65.5	86.1
XNOR-Network	51.2	73.2	N/A	N/A
Full-Precision-Network	69.3	89.2	71.3	90.0

binary weights → ~32x savings in memory

BUT WHAT ABOUT LATENCY?

Low precision



"*Riptide: Fast End-to-End Binarized Neural Networks*,"
Fromm, Cowan, Philipose, Ceze, and Patel, MLSys 2020

Motivation:

- Quantized NNs show promise for reducing inference time
- But, it can be difficult to achieve speedups on real-world processors

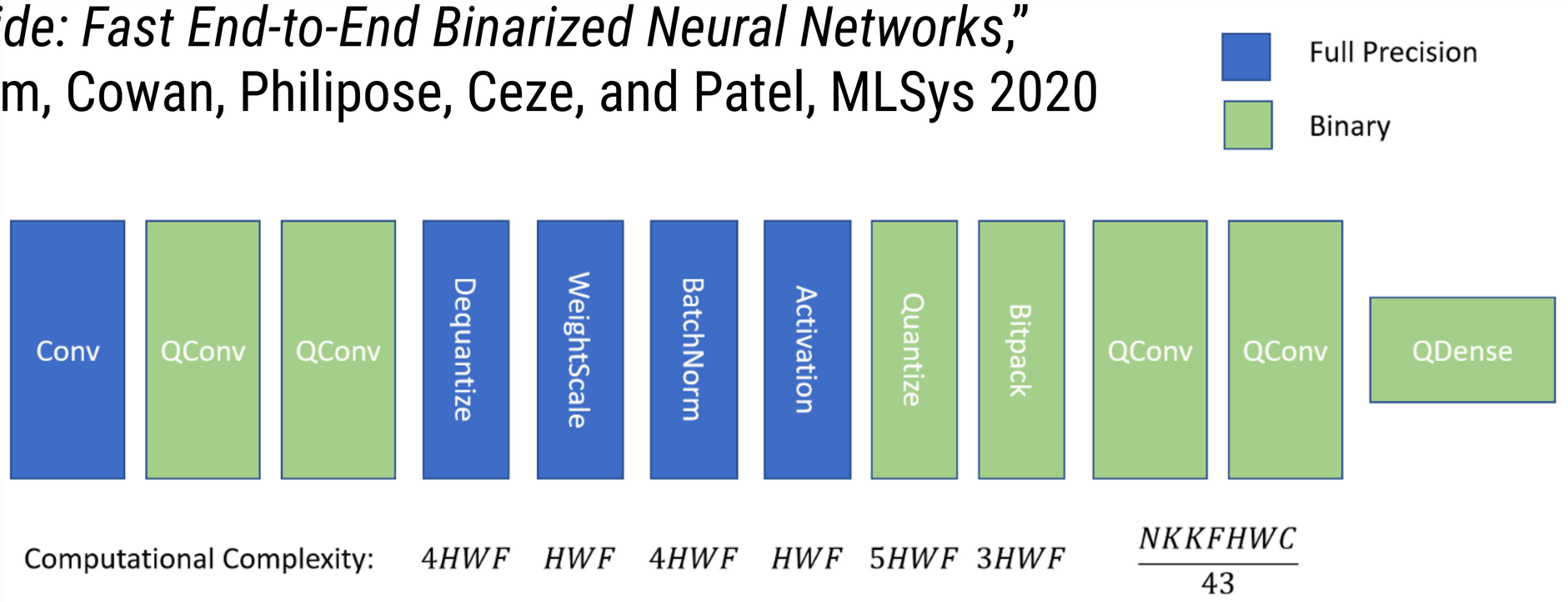
Why?

- Implementation matters: Current libraries like OpenBLAS have been engineered to work with full precision arithmetic
- Focus has also been on quantizing the main bottlenecks (e.g., convolutions), not the other operations ('glue layers')

BUT WHAT ABOUT LATENCY?

Low precision

"*Riptide: Fast End-to-End Binarized Neural Networks*,"
Fromm, Cowan, Philipose, Ceze, and Patel, MLSys 2020



- Important to reduce/remove these "glue layers", which become the bottleneck
- Leads to speedups of up to 12x using the DL compiler TVM

AN ASIDE…

Deep learning compilers

For training, we discussed a few common options for hardware:

- CPUs, GPUs, TPUs

This list becomes much longer when we additionally consider hardware for inference (e.g., mobile phones IoT devices, specialized accelerators)

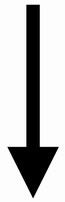
Q: How do we get our software (TensorFlow, PyTorch, etc) running on various hardware?

A: Need to compile it into hardware-specific language

AN ASIDE…

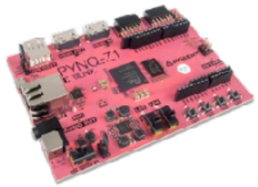
Deep learning compilers

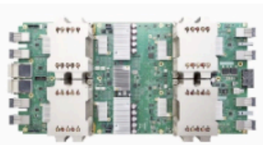
Frameworks express high-level computation



???

Hardware executes a series of low level operations





Credit: Tianqi Chen

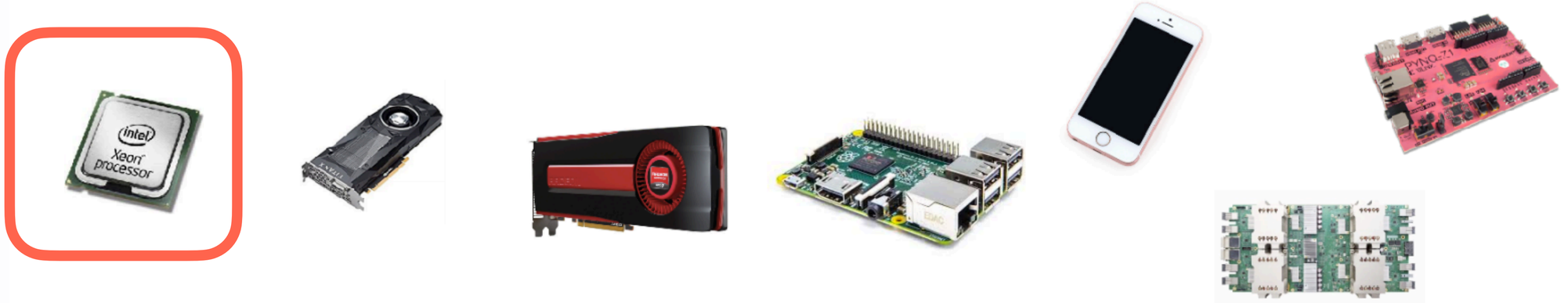
AN ASIDE…

Deep learning compilers

Frameworks express high-level computation

OpenBlas

Hardware executes a series of low level operations



Credit: Tianqi Chen

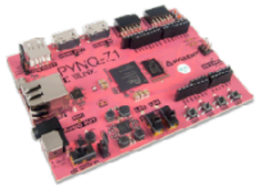
AN ASIDE…

Deep learning compilers

Frameworks express high-level computation

CUDA

Hardware executes a series of low level operations

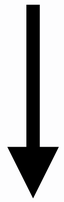


Credit: Tianqi Chen

AN ASIDE…

Deep learning compilers

Frameworks express high-level computation



TVM

Hardware executes a series of low level operations

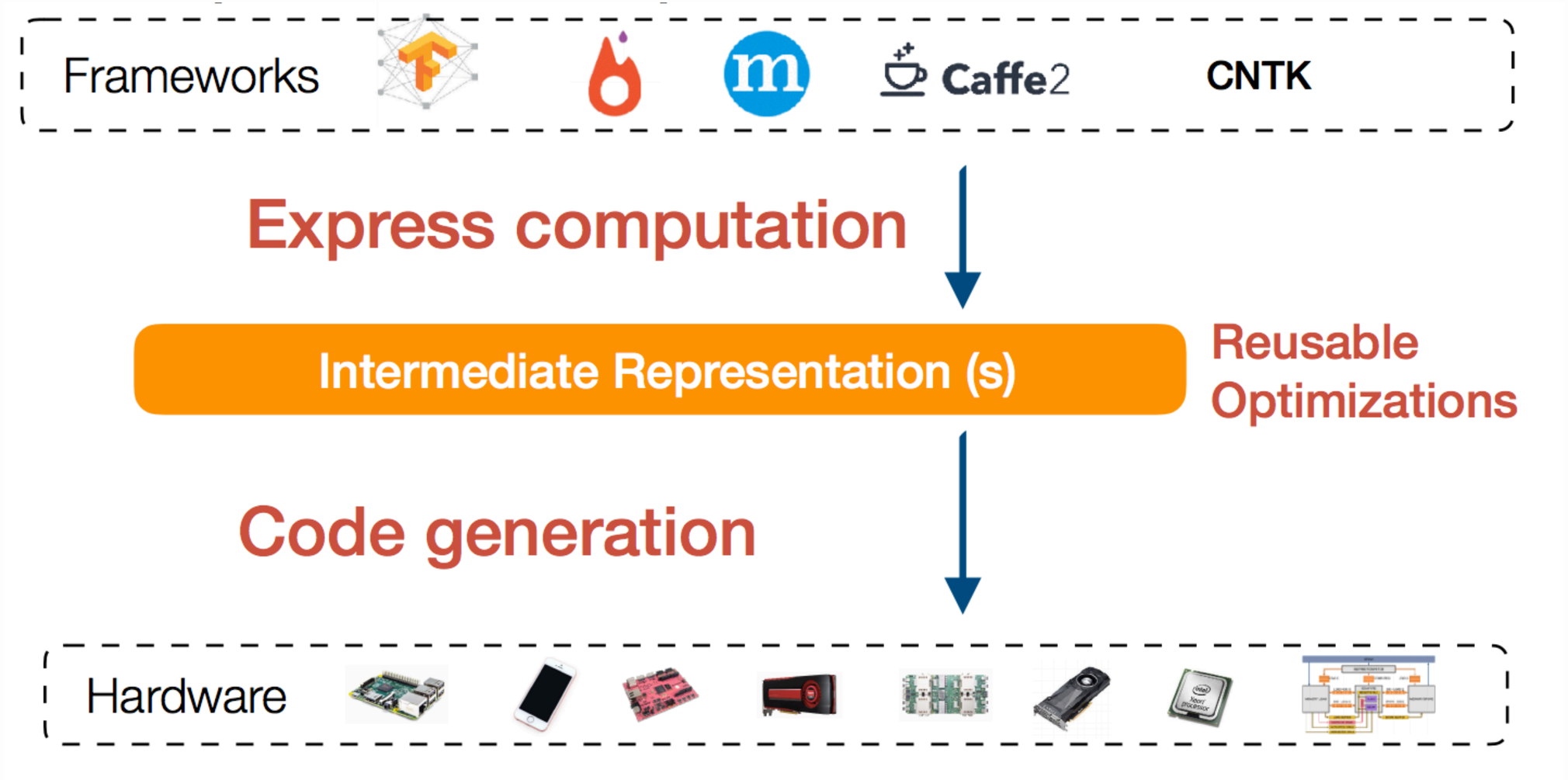
Credit: Tianqi Chen

AN ASIDE…

Deep learning compilers

OctoML

Automate efficient AI/ML ops through a unified software foundation.



Credit: Tianqi Chen

Model compression

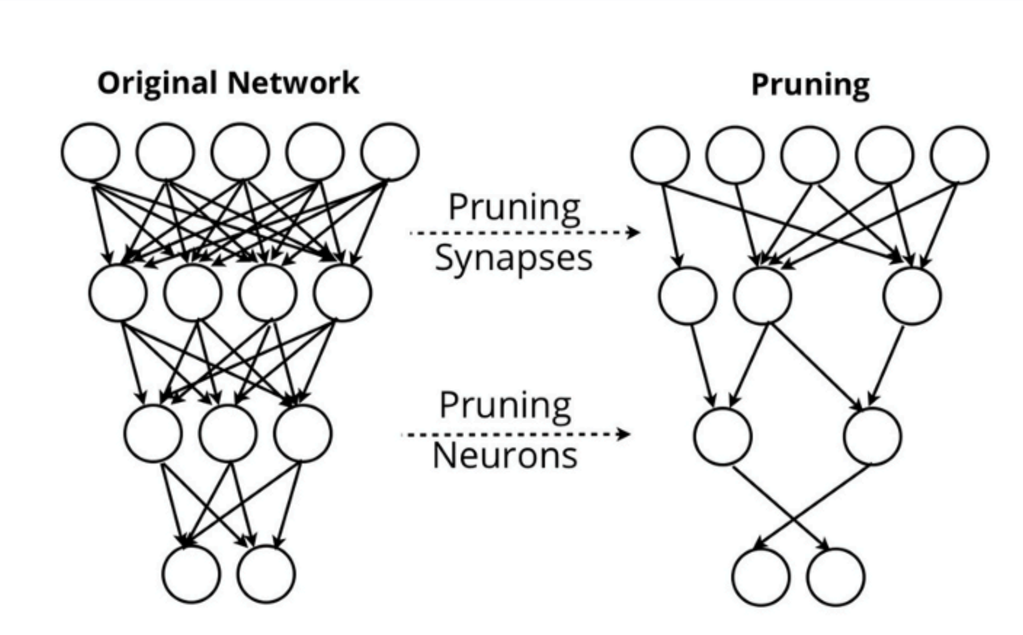
Goal: find a smaller, cheaper model with the same (or similar) accuracy

Many techniques:

- Quantization (low-precision inference)
- Pruning
- Knowledge distillation
- Efficient architectures (+ architecture search)

Pruning

- Remove ‘low impact’ weights or activations
- Effectively creates a smaller model (less weights to store)
- Makes it easy to retrain if necessary, since we’re training a smaller model
- Simple heuristic: remove all weights/ activations less than some threshold
- Will explore pruning heuristics in MP-A …



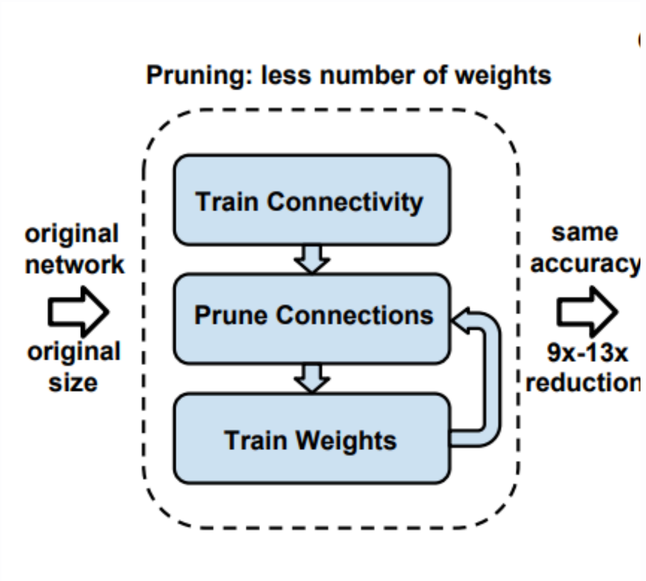
Pruning

Techniques may differ in terms of:

- What structures to prune?
  - unstructured: don’t consider relationships between weights
  - structured: remove entire neurons, filters, or channels
- How to rank which weights to prune?
  - magnitude, first or second order information
- When / how often prune?
  - one-shot pruning (at the end of training)
  - iterative pruning (train, prune, re-train multiple times)
  - after training vs during training (e.g., dropout)

Pruning

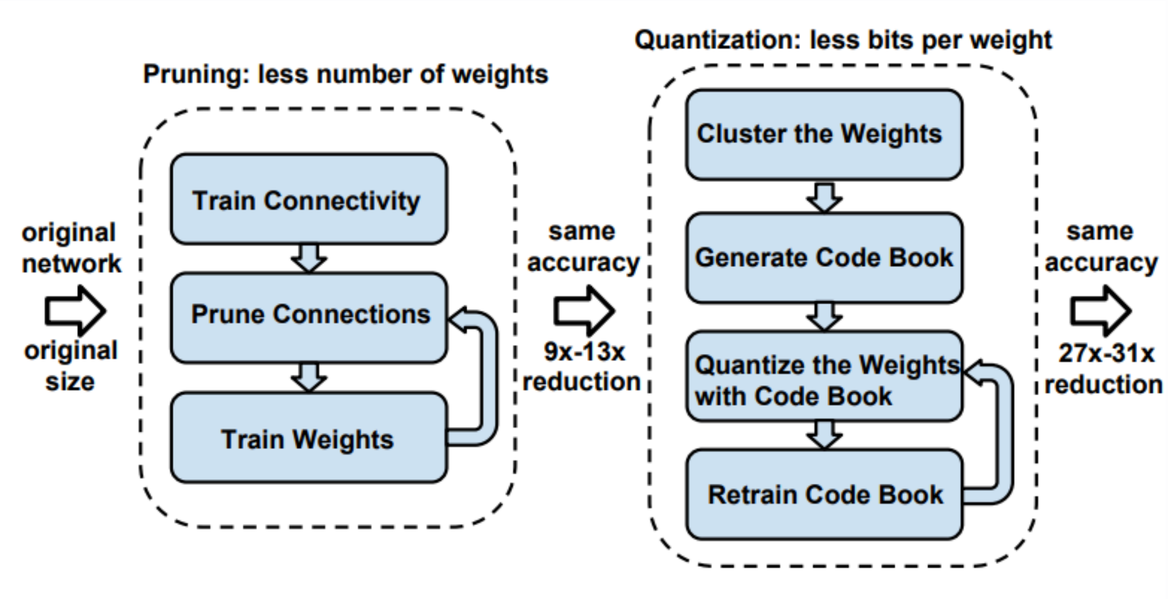
can also combine pruning and quantization



Han et al, ICLR 2016

Pruning

can also combine pruning and quantization



Han et al, ICLR 2016

Pruning

can also combine pruning and quantization

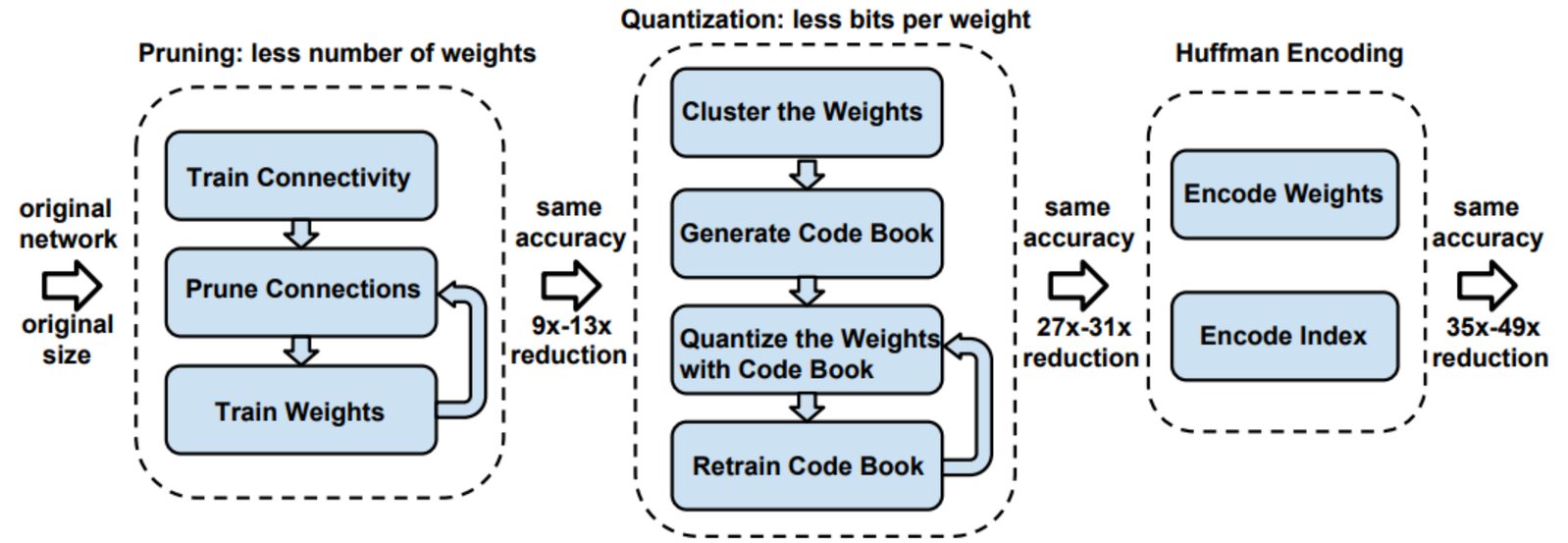
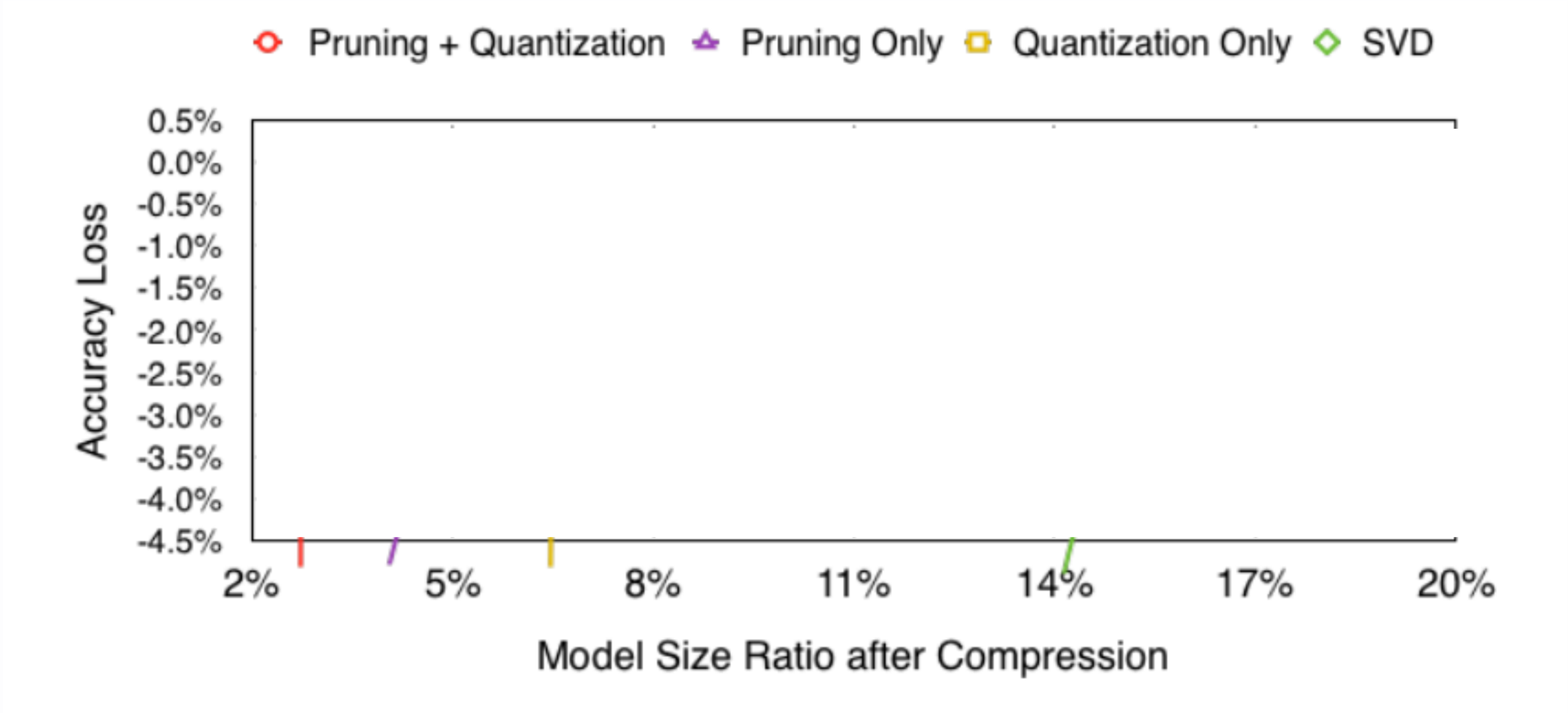


Figure 1: The three stage compression pipeline: pruning, quantization and Huffman coding. Pruning reduces the number of weights by $10\times$, while quantization further improves the compression rate: between $27\times$ and $31\times$. Huffman coding gives more compression: between $35\times$ and $49\times$. The compression rate already included the meta-data for sparse representation. The compression scheme doesn't incur any accuracy loss.

Han et al, ICLR 2016

Pruning

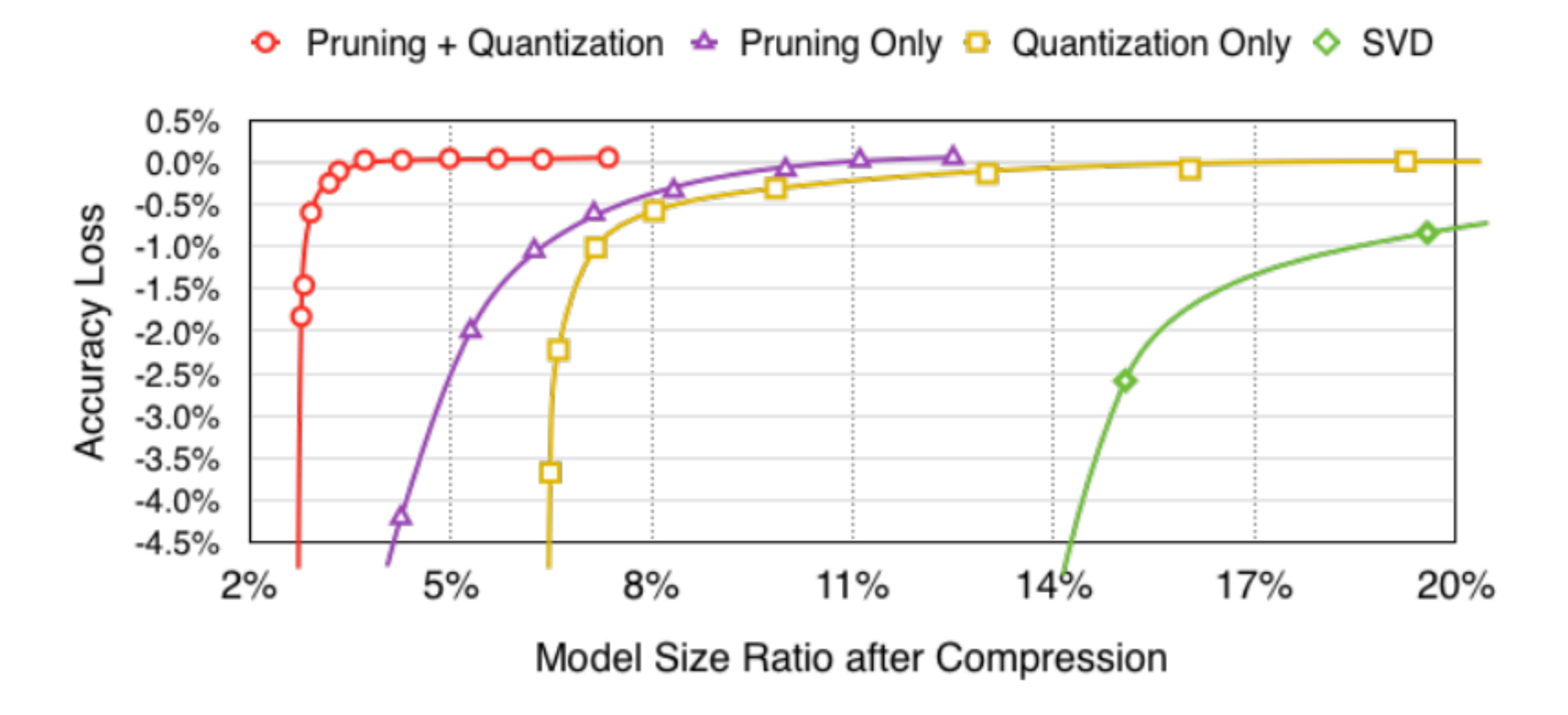
can also combine pruning and quantization



Han et al, ICLR 2016

Pruning

can also combine pruning and quantization



Han et al, ICLR 2016

Model compression

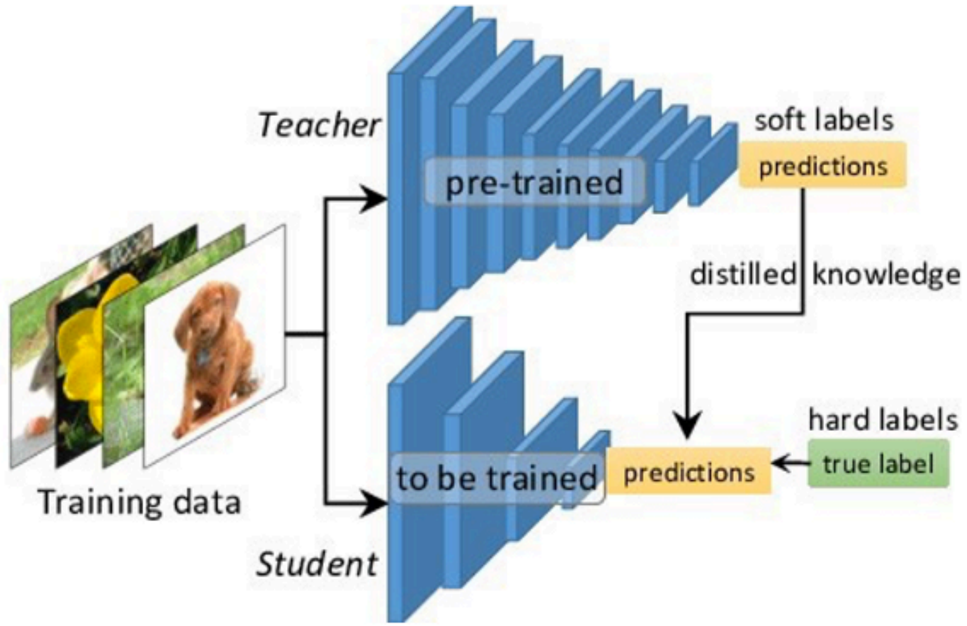
Goal: find a smaller, cheaper model with the same (or similar) accuracy

Many techniques:

- Quantization (low-precision inference)
- Pruning
- Knowledge distillation
- Efficient architectures (+ architecture search)

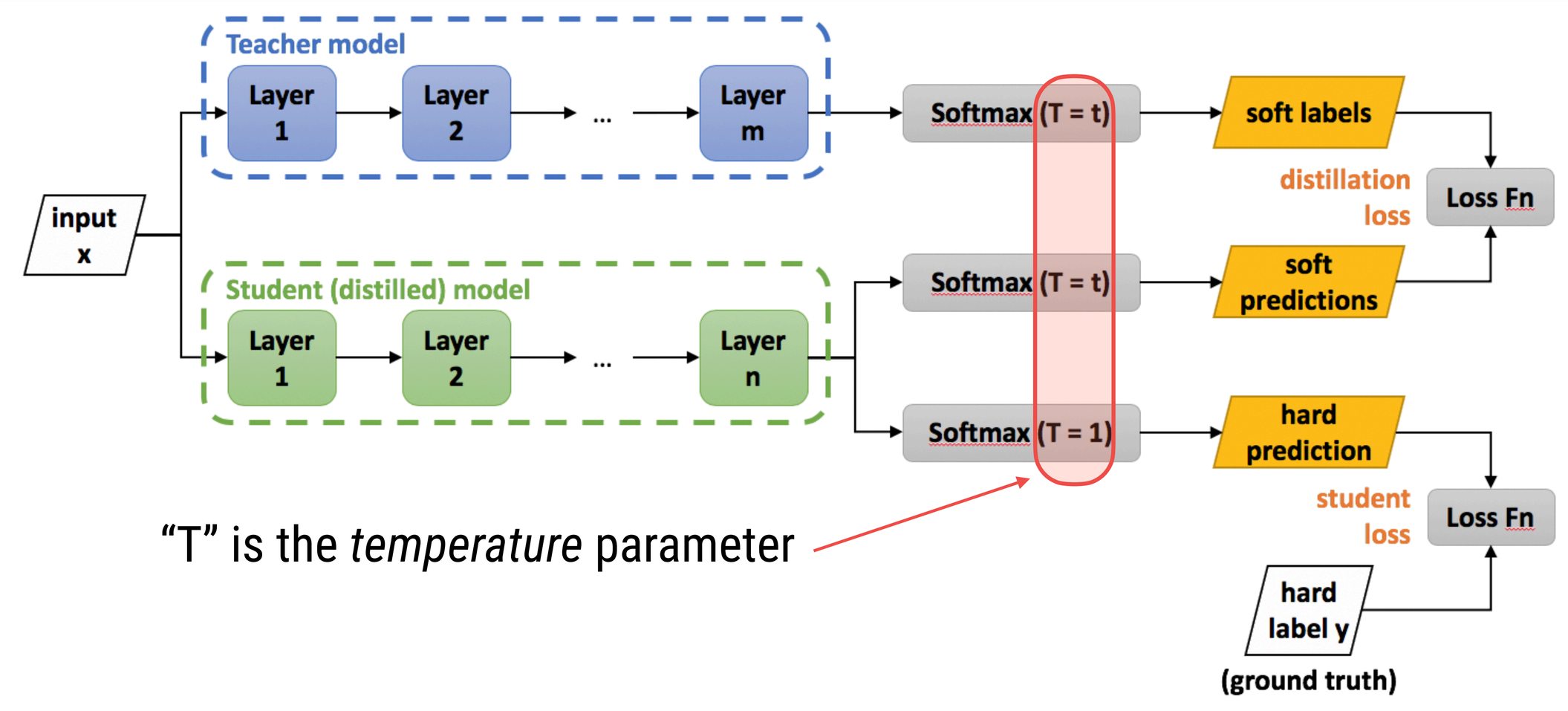
Knowledge distillation

- Idea: take a large/complex model and train a smaller network to match its output
- Possible benefit over pruning/quantization: Smaller architecture can differ significantly
- *Why might this heuristic work?*
  Intuition is that it's easier to learn from the model's output [the teacher] than the true labels



Bucila et al, 2006; Hinton et al, 2015

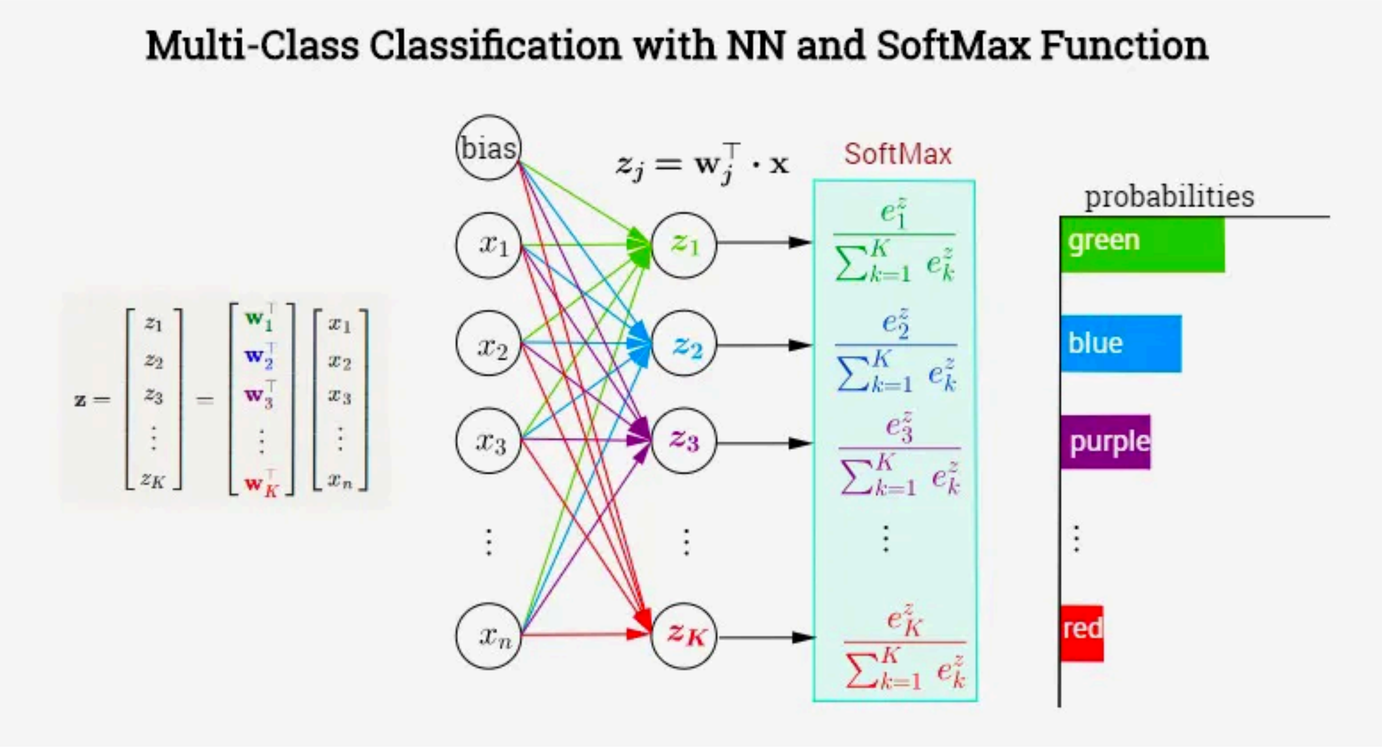
Knowledge distillation



Hinton et al, 2015

Softmax temperature

- Recall: softmax function is used to determine predicted probabilities for multiple classes

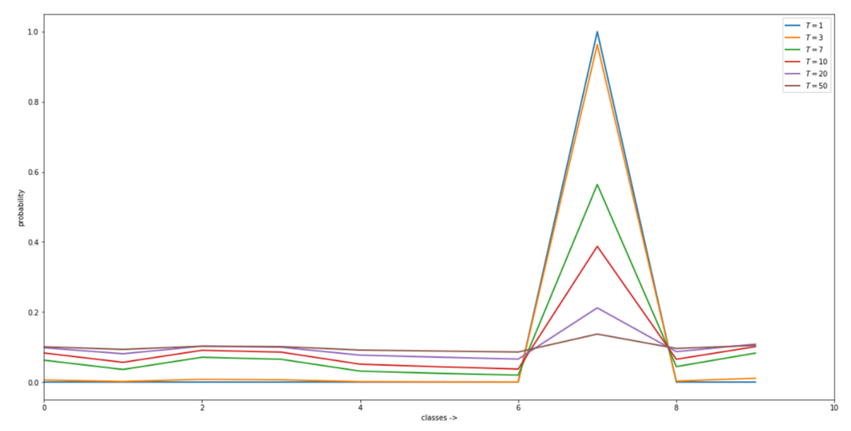


Hinton et al, 2015

Softmax temperature

- Potential issue: output of teacher's softmax will have very high probability on correct class, low probability on other classes → *doesn't provide much information beyond ground truth labels*
- Idea: use a slight variant of the usual softmax function with 'temperature' hyperparameter, T
- Provides a 'softer' distribution

$$p_i = \frac{\exp\left(\frac{z_i}{T}\right)}{\exp\left(\sum_j \frac{z_j}{T}\right)}$$



Hinton et al, 2015

PUTTING IT ALL TOGETHER …

Knowledge Distillation

- Train student network from scratch, using the following loss function:

$$\mathcal{L}(x; W) = \alpha * \mathcal{H}(y, \sigma(z_s; T = 1)) + \beta * \mathcal{H}(\sigma(z_t; T = \tau), \sigma(z_s, T = \tau))$$

student loss

teacher loss

- α, β, τ are all hyperparameters that must be set
- in initial experiments, $\tau \in [1,20]$ (use small τ for smaller students)
- generally set α to be smaller than β

Hinton et al, 2015

Model compression

Goal: find a smaller, cheaper model with the same (or similar) accuracy

Many techniques:

- Quantization (low-precision inference)
- Pruning
- Knowledge distillation
- Efficient architectures (+ architecture search)

Efficient architectures

- Some neural network architectures are designed to be efficient at inference time
  - e.g., mobilenet, shufflenet
- These networks are often based on sparsely connected units
- For efficiency, we can start by training one of these networks (in contrast to training a larger model and then compressing it)

Table 8. MobileNet Comparison to Popular Models

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
GoogleNet	69.8%	1550	6.8
VGG 16	71.5%	15300	138

Howard et al, 2017

ONE LAST OPTION …

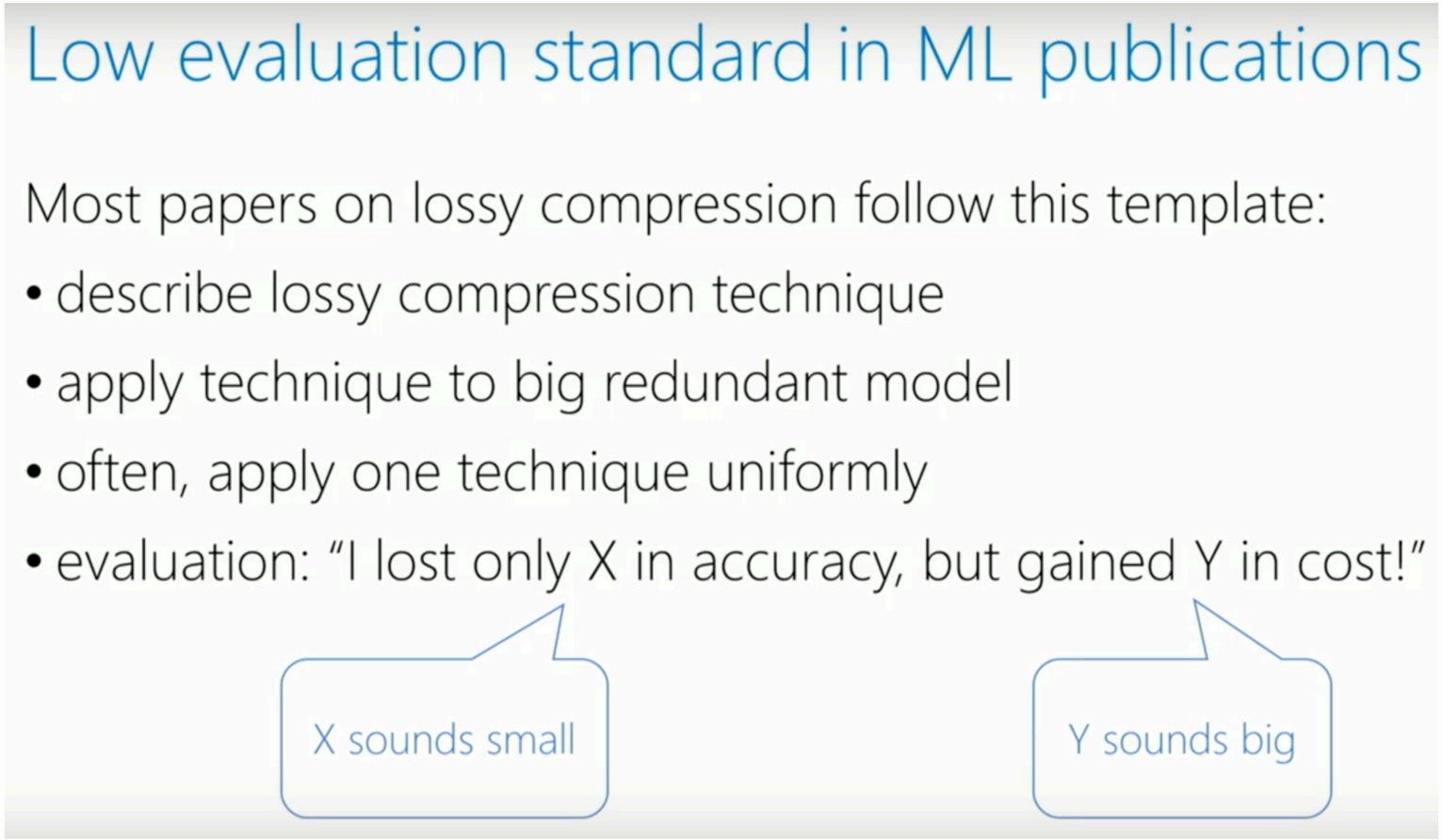
Pick a different model class!

- Finally, you can also train and use a model that is not a DNN …
- Linear models can be much faster and might be necessary in some applications

A WORD OF CAUTION

Performance on the Pareto frontier

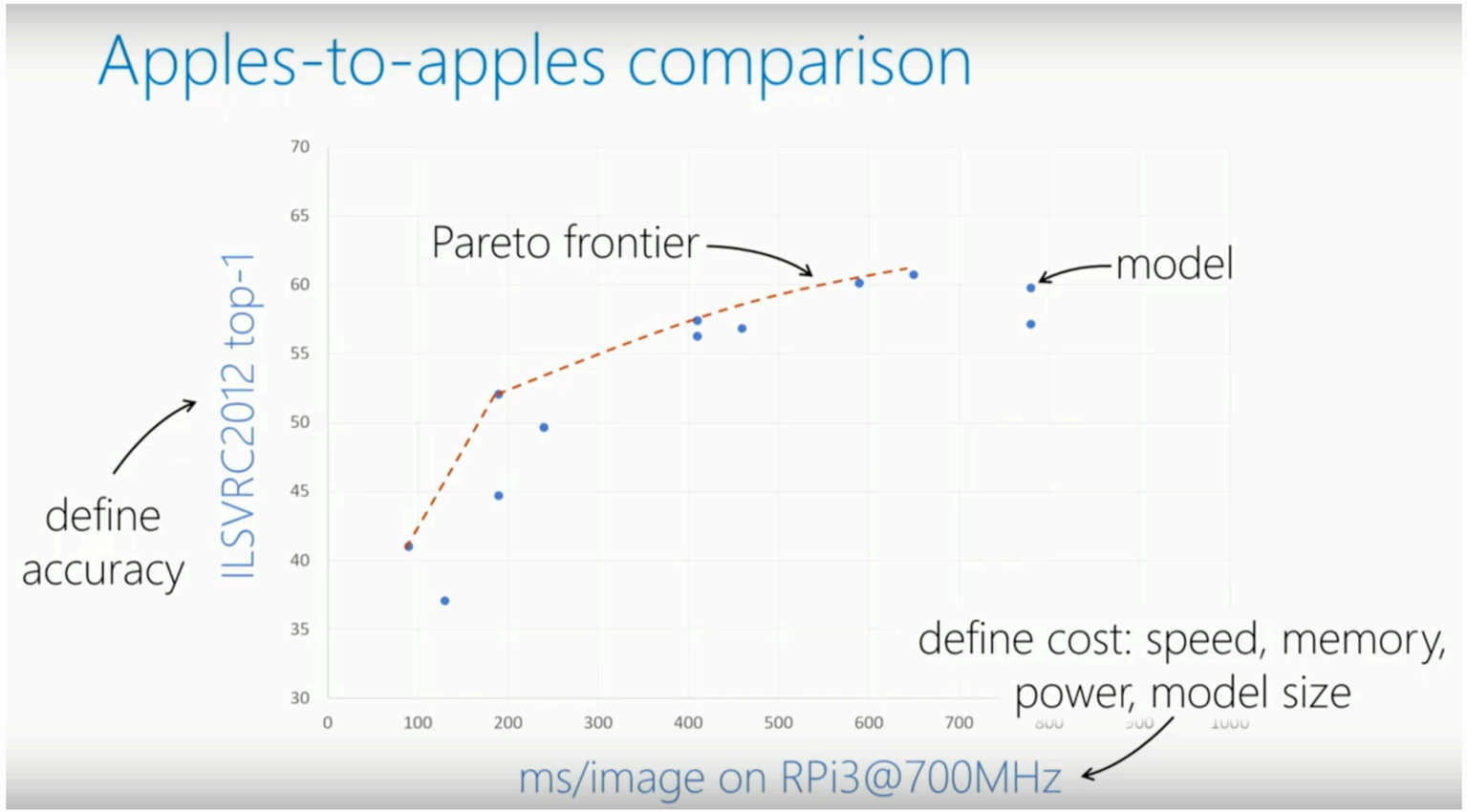
MLSys 2019 keynote: Compiling AI for the Edge, Ofer Dekel (MSR)



A WORD OF CAUTION

Performance on the Pareto frontier

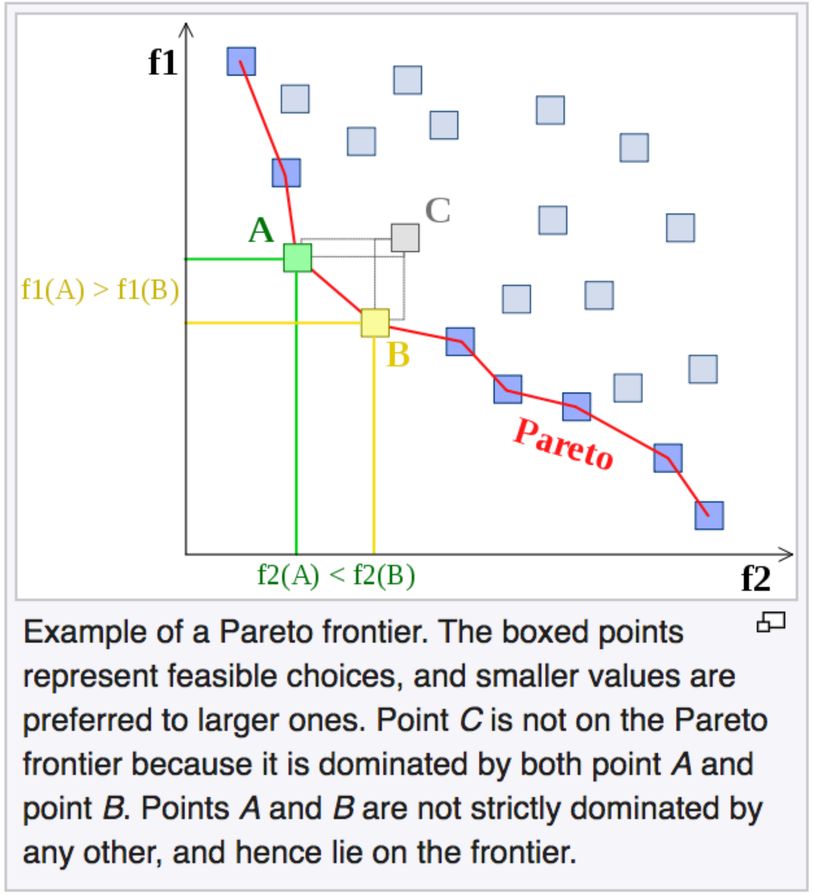
MLSys 2019 keynote: Compiling AI for the Edge, Ofer Dekel (MSR)



A WORD OF CAUTION

Performance on the Pareto frontier

MLSys 2019 keynote: Compiling AI for the Edge, Ofer Dekel (MSR)



Example of a Pareto frontier. The boxed points represent feasible choices, and smaller values are preferred to larger ones. Point *C* is not on the Pareto frontier because it is dominated by both point *A* and point *B*. Points *A* and *B* are not strictly dominated by any other, and hence lie on the frontier.

Issue: there are two metrics we care about:

- accuracy
- cost

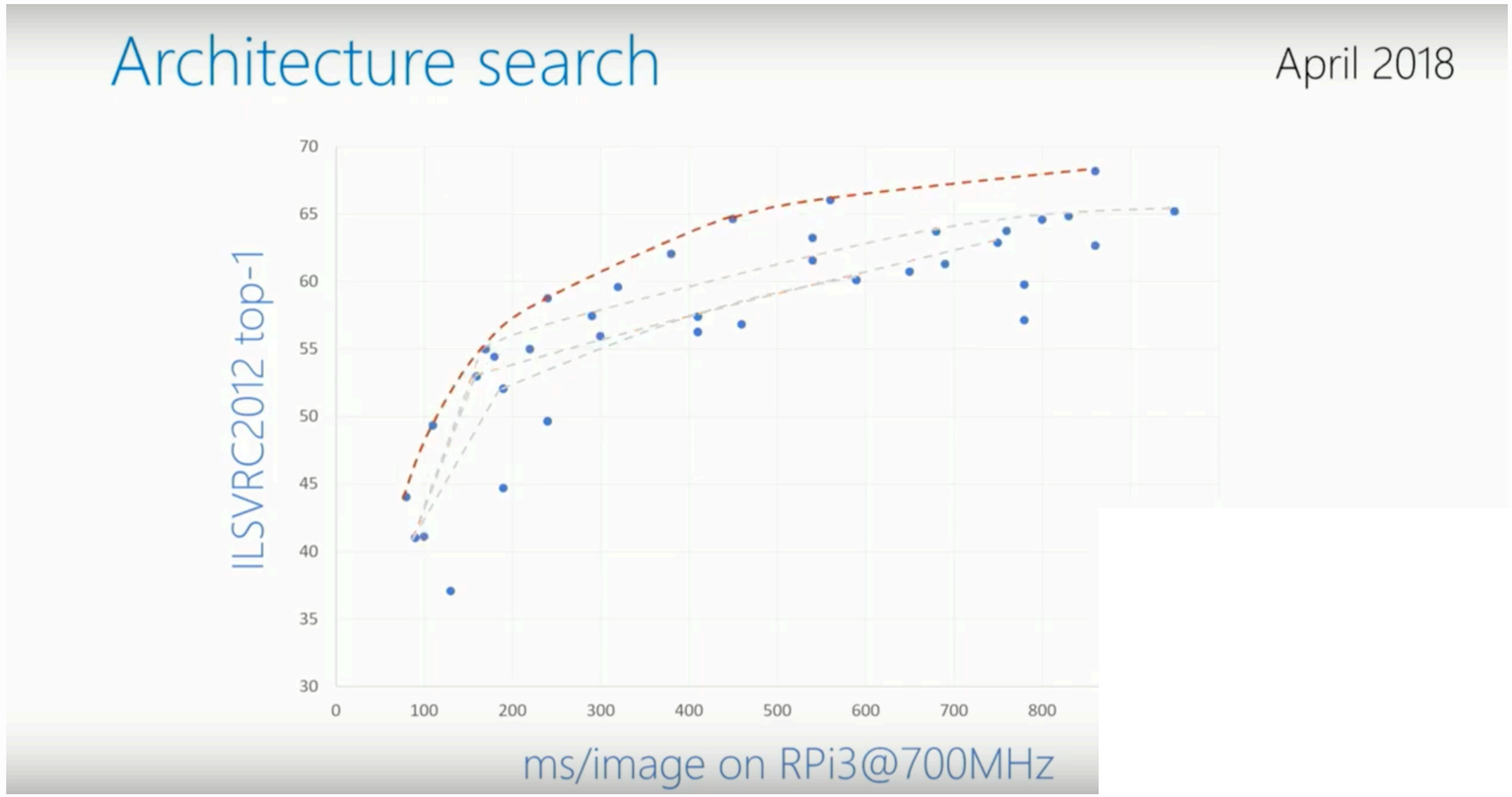
What models/techniques are the best?

- it depends on what accuracy / cost you care about
- goal: demonstrate optimality on the Pareto frontier
  - i.e., there are no other models with the same (or lower) cost that achieve the same (or higher) accuracy

A WORD OF CAUTION

Performance on the Pareto frontier

MLSys 2019 keynote: [Compiling AI for the Edge](), Ofer Dekel (MSR)

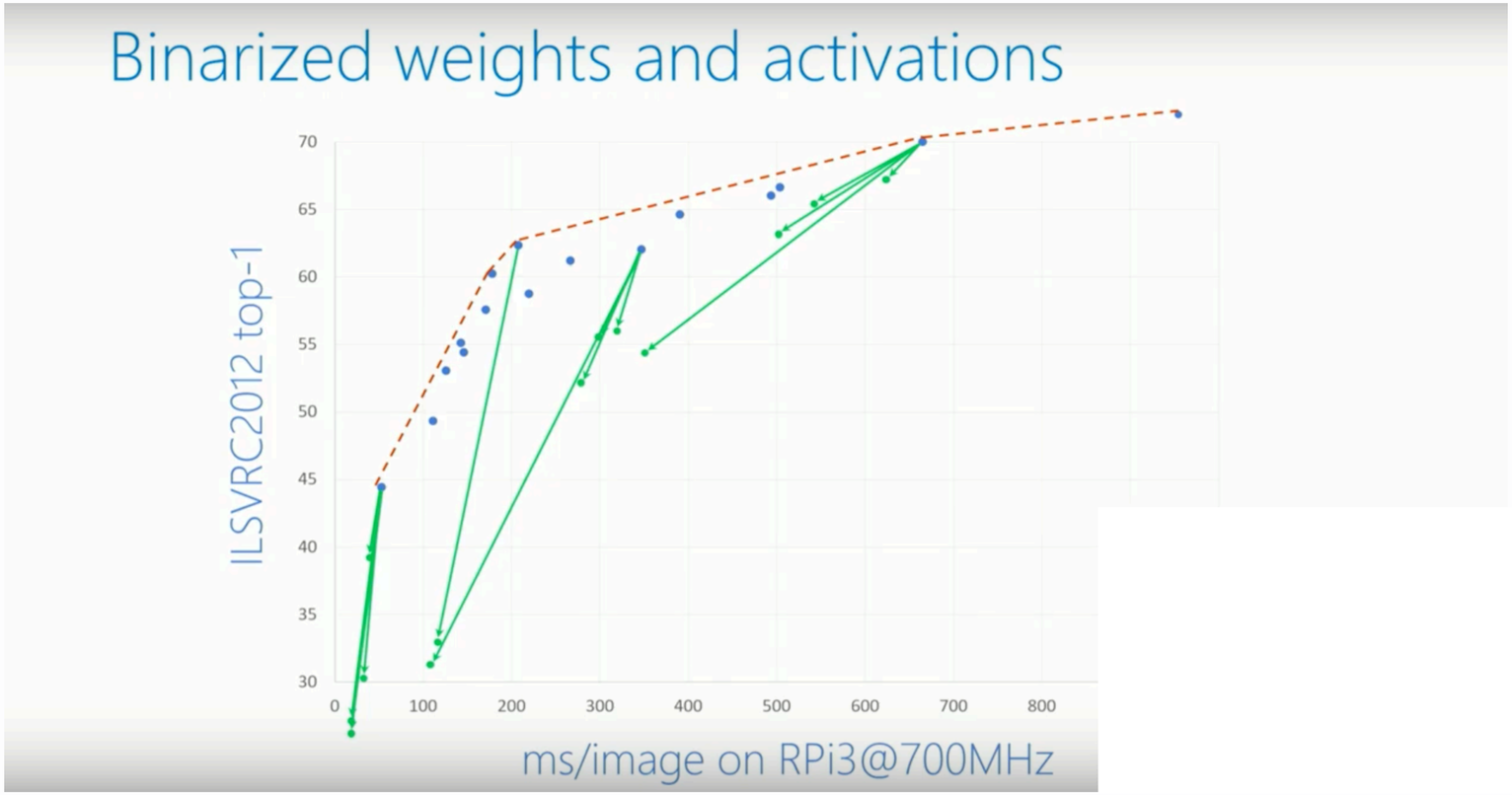


for this problem: architecture search pushed the Pareto frontier

A WORD OF CAUTION

Performance on the Pareto frontier

MLSys 2019 keynote: [Compiling AI for the Edge](), Ofer Dekel (MSR)

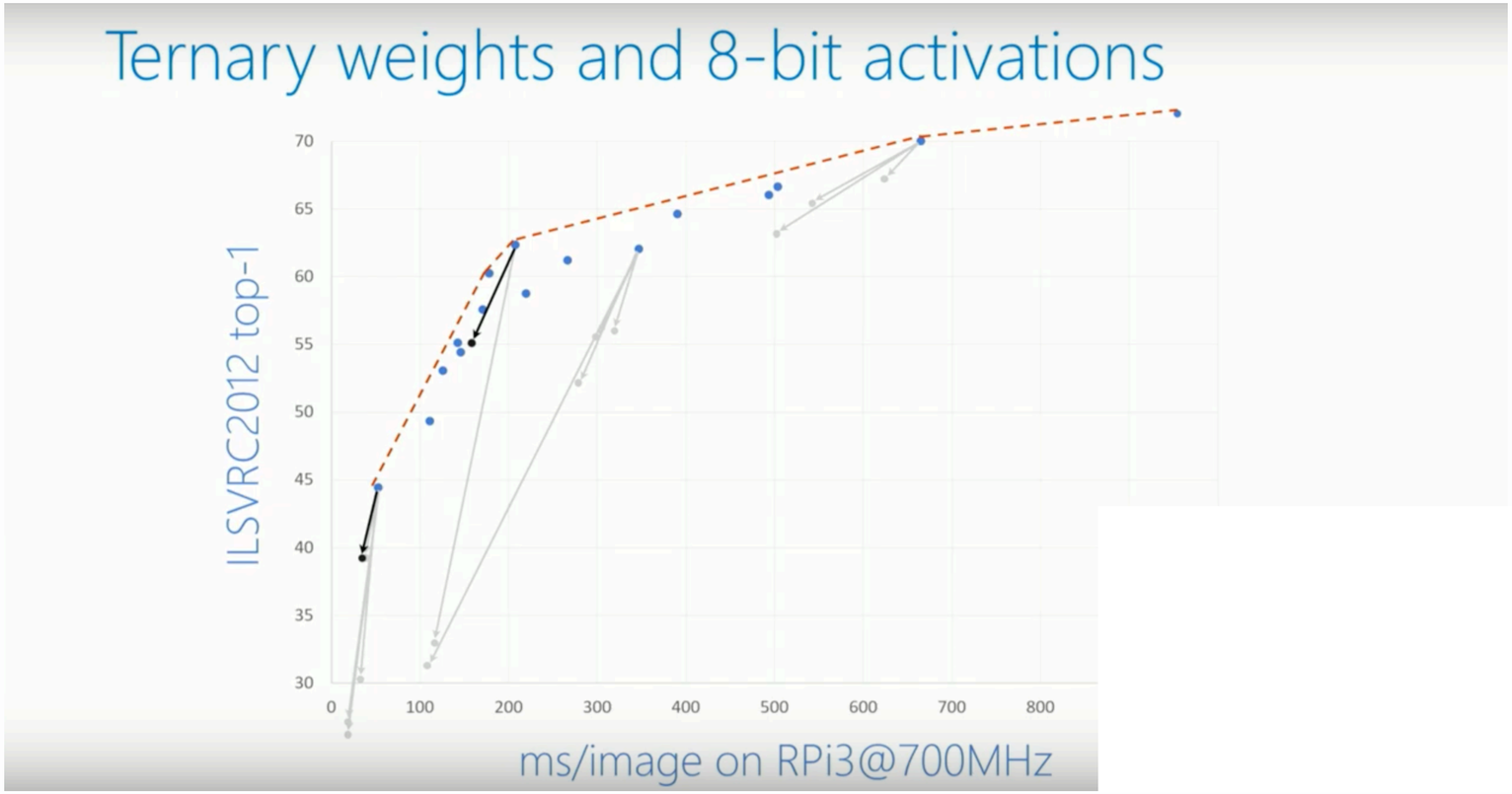


for this problem: quantization did not

A WORD OF CAUTION

Performance on the Pareto frontier

MLSys 2019 keynote: Compiling AI for the Edge, Ofer Dekel (MSR)

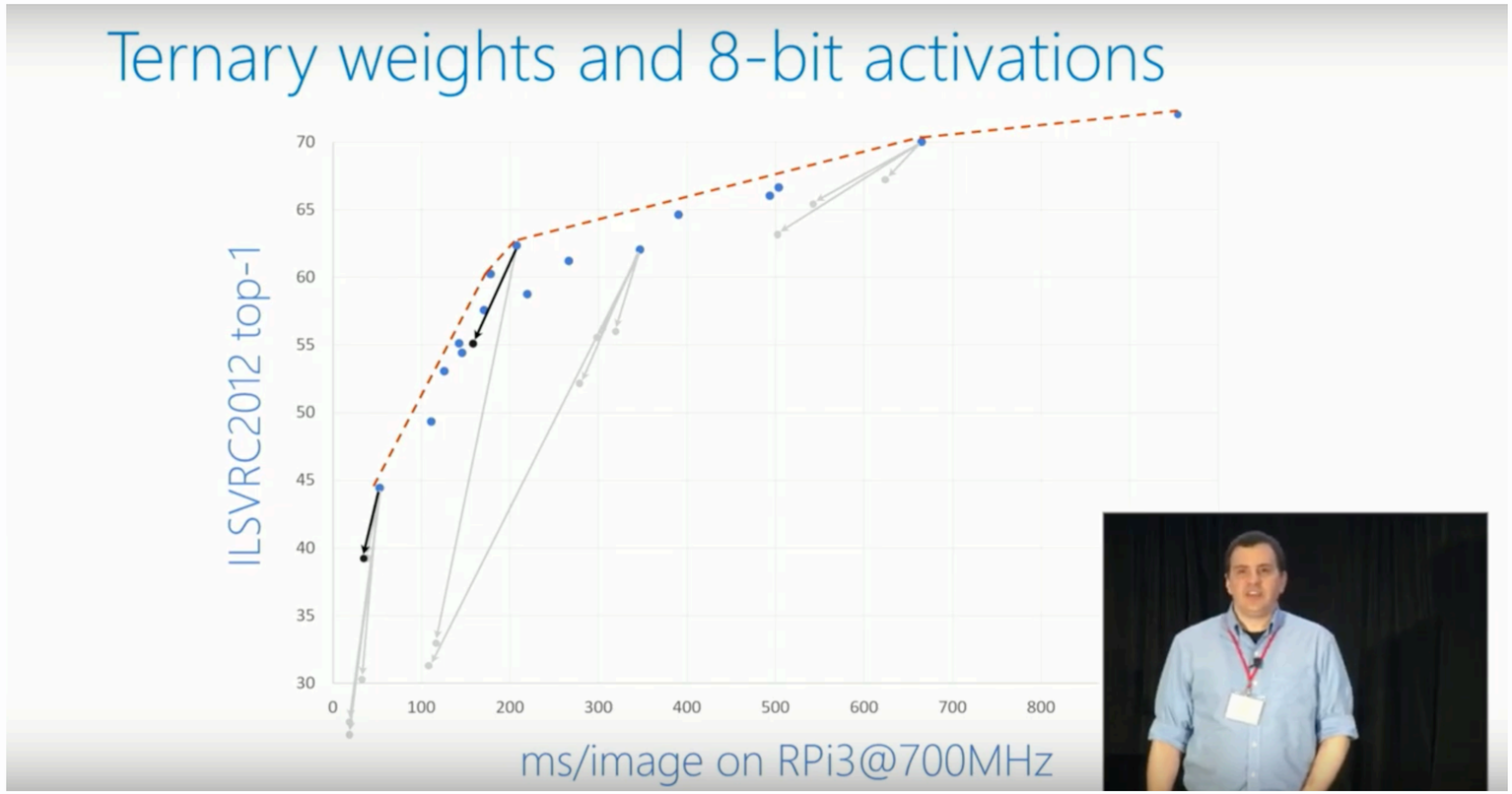


for this problem: quantization did not

A WORD OF CAUTION

Performance on the Pareto frontier

MLSys 2019 keynote: Compiling AI for the Edge, Ofer Dekel (MSR)



for this problem: quantization did not

(Virtual) Guest Lecture Next Thursday

Krishna Rangasayee, CEO of SiMa.ai

Will not be recorded: PLEASE SHOW UP!