

COMS E6998 012/15099

Practical Deep Learning Systems Performance

Lecture 7 10/17/19

Logistics

- Homework 2 due Oct 22 by 11:59 PM
- Project proposals due Today 11:59 PM (5%)
- Seminar reading list due Today 11:59 PM (5%)
- Homework 1 has been graded; grades conveyed soon
- Start working on your 10% for technical community participation

Recall from last lecture

- LeNet, Alexnet, VGG, Resnet; ILSVRC challenge
- Performance comparison of different CNNs
- Object localization and detection
- R-CNN and Fast R-CNN
- Transfer Learning and Transferability of features
- Predict to Learn (P2L) based source model selection
- Finetuning and its degree

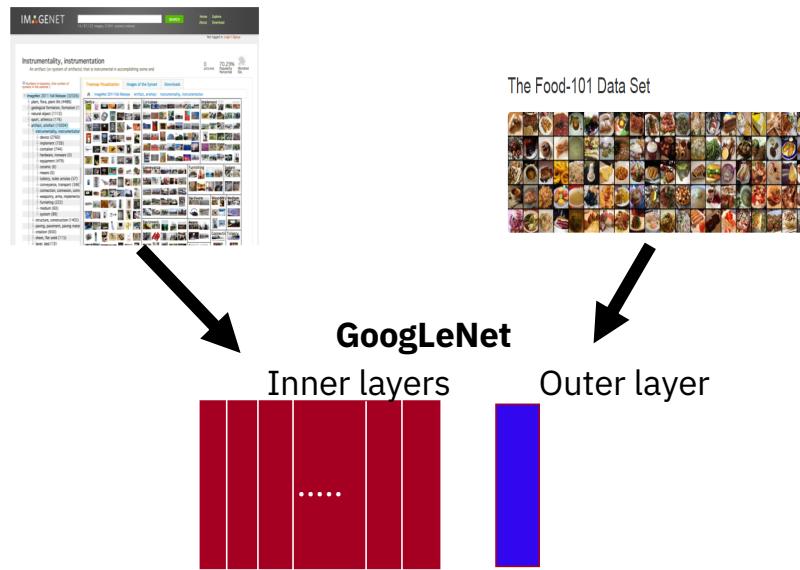
Fine tuning example using Caffe

- Show an example: solver, network file
- Pretrained model
- Finetuning command
- Go over a convolution neural network in Caffe
 - Different layers and their parameters
 - Show filter specifications and learning rate

Learn to Transfer

- Which source model to use ?
 - Caffe Model Zoo has 44 entries
 - IBM AI Vision Model Catalog has 95 entries
- Which method to use ?
 - Shallow learning driving SVMs
 - Fine tuning
 - Single source or ensemble
- Which layers to freeze and finetune ?
 - ResNet101 has 101 layers
 - GoogLeNet has 22 layers
 - VGG16 has 16 layers
- What is the performance requirement?
 - Latency
 - Memory Footprint
 - Target domain accuracy
- What curriculum should I follow?
 - Phalanx2 -> Food101 -> Greek Food
 - Imagenet1K -> Greek Food

Impact of base model data size on transfer learning



Fine-tuning results on Food-101 dataset with varying numbers of learning rate multipliers for layers with pre-trained weights

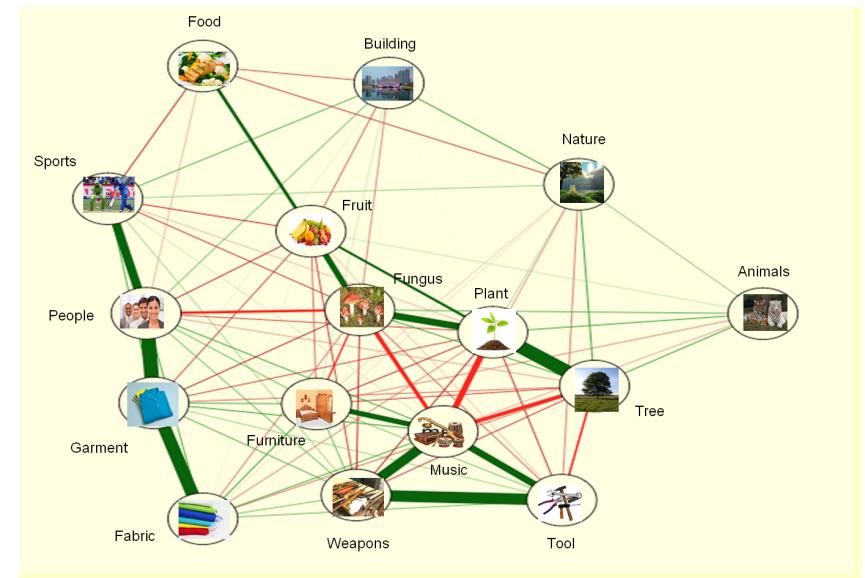
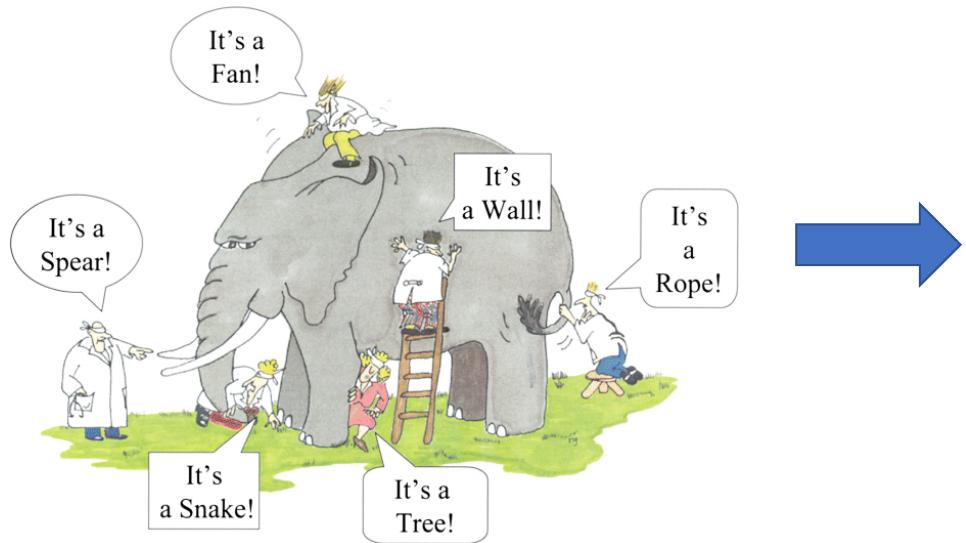
Exploiting Web Data for Training

- Web and social media are most important sources of data for vision research
- Vision datasets such as ImageNet, PASCAL VOC, and MS COCO created from Google or Flickr by harnessing human intelligence to filter out the noisy images and label object locations
- However human labeling is costly
- Can we learn visual representations and object detectors from the web data directly ?
 - Without the need for any human intervention

Automated Labeling of Images

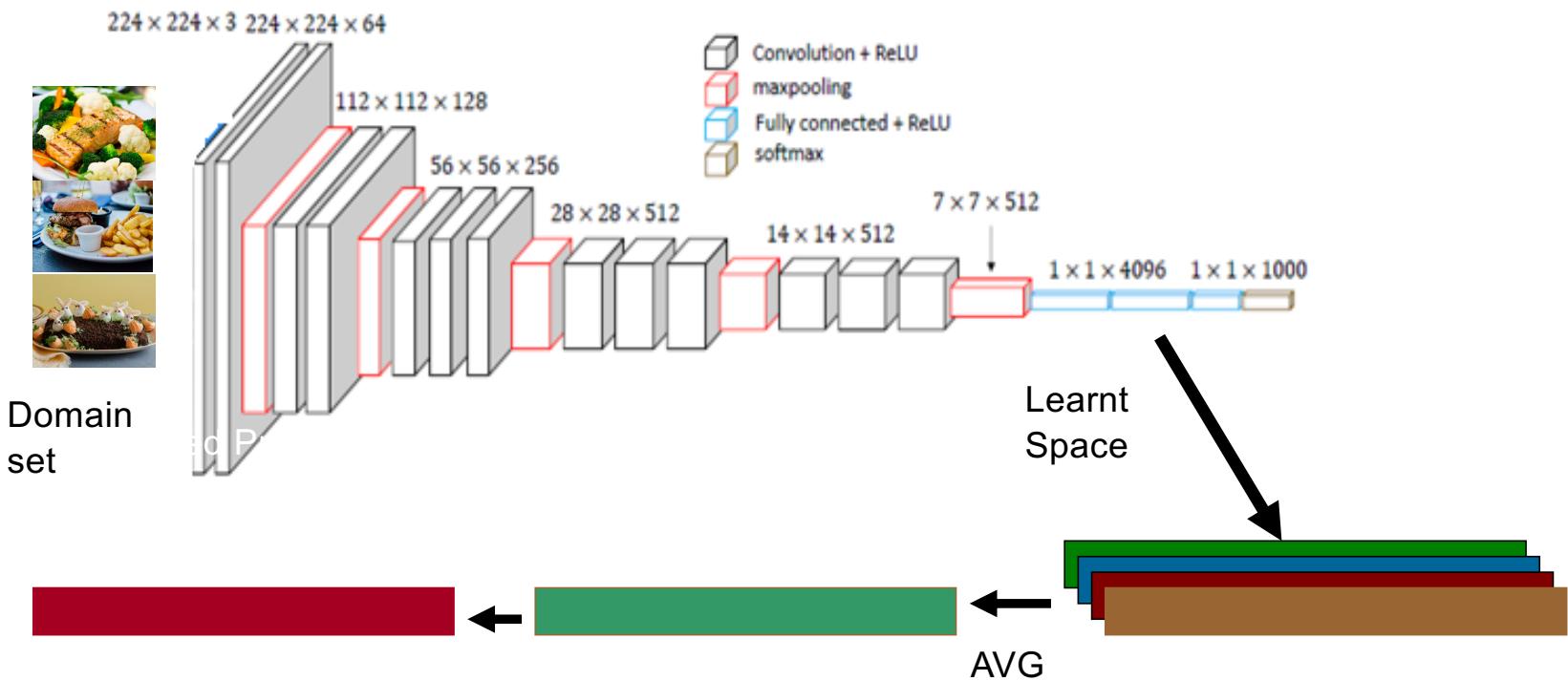
- Pseudo labeling of unlabeled Data for Transfer Learning
- Goal: Exploit data from the wild to capture rich representations in source models
- Can we automatically label data and build base models for transfer learning ?

KL divergence based labeling



Dube et al. Automatic Labeling of Data for Transfer Learning. Deep Vision 2019

Extracting Feature Vectors For Distance Measure



Using KL Divergence For Distance Measure

Target Image

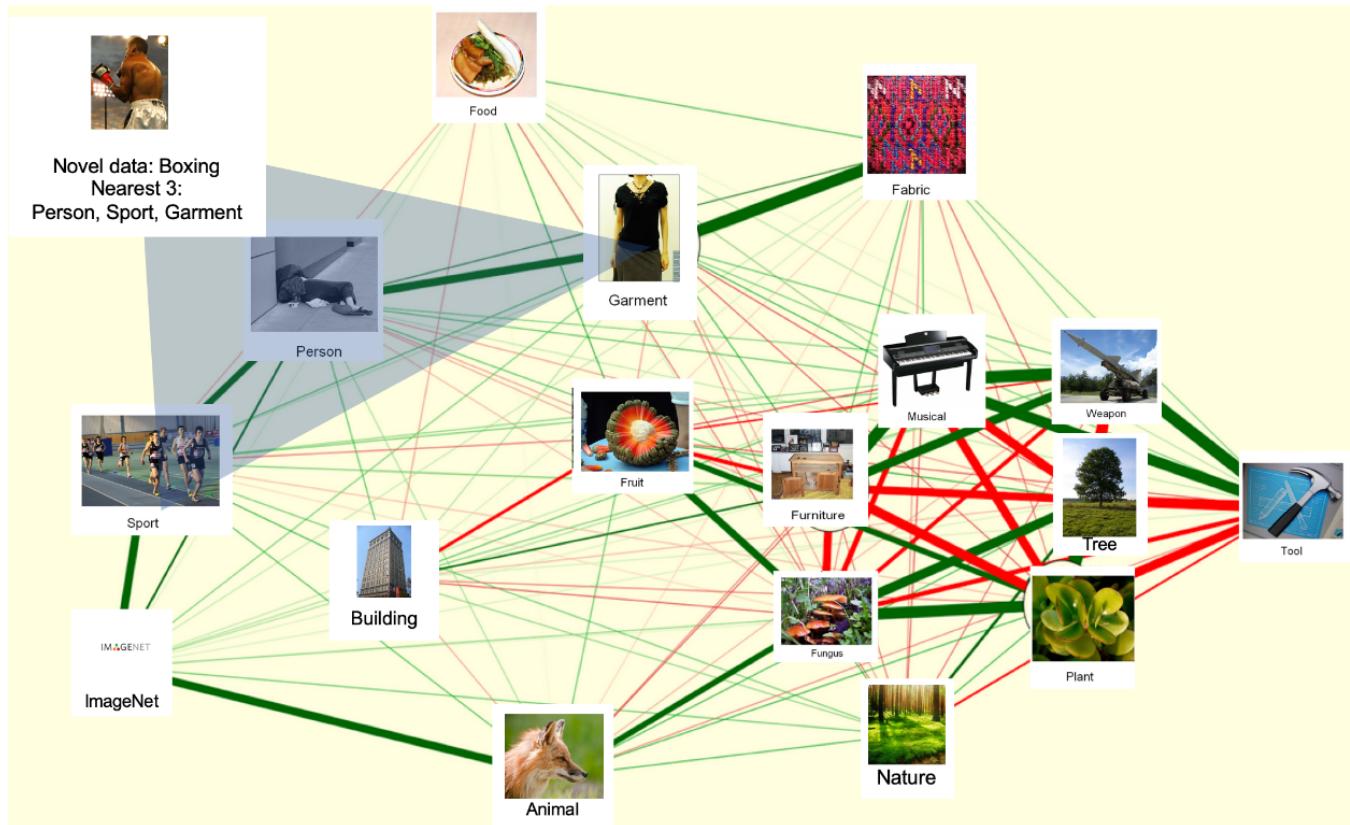


A Source Dataset

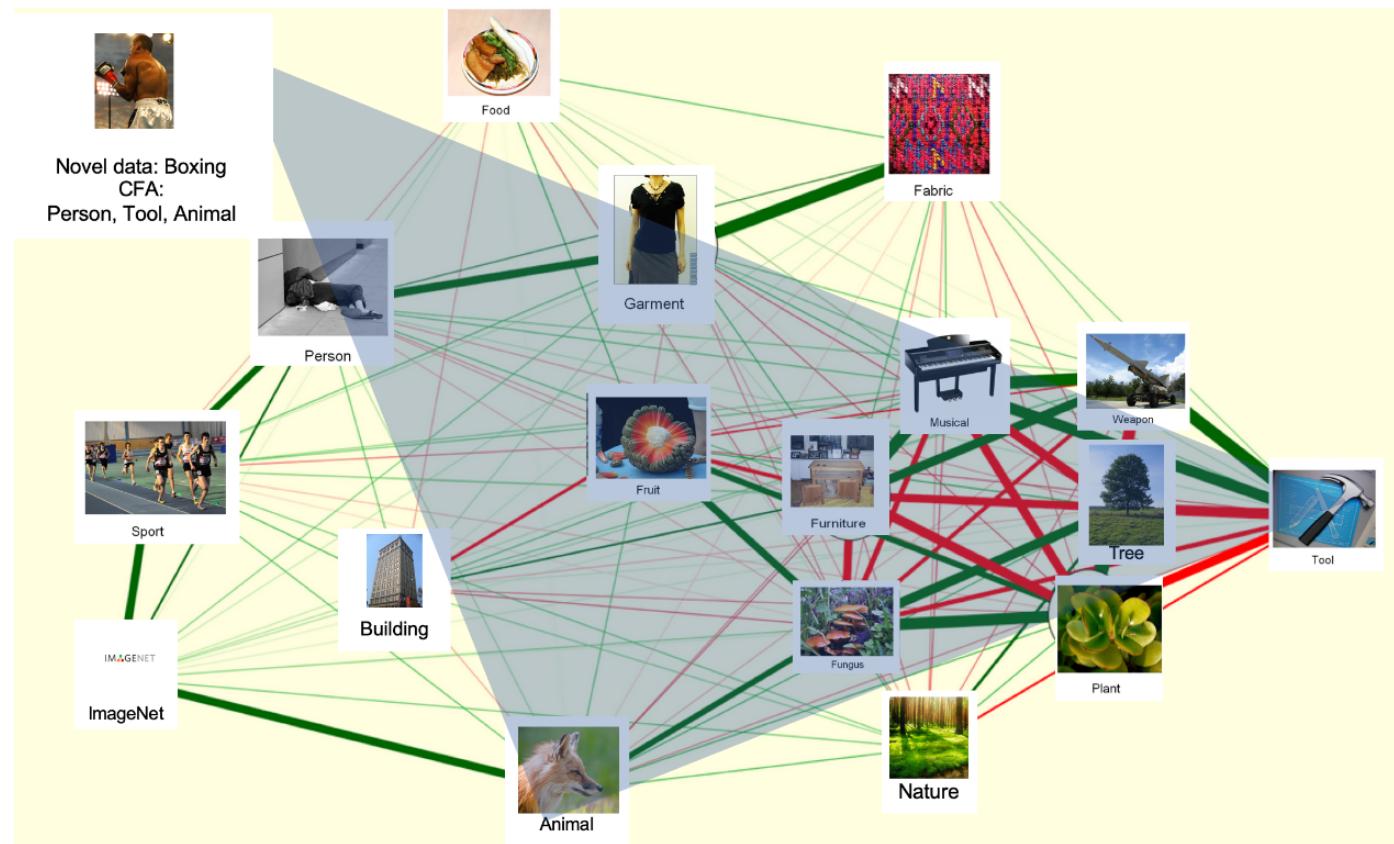


Kullback-Leibler (KL) Divergence

Nearest N



CFA (closest, farthest, area maximizer)



Visual Example: Nearest 3 and CFA



elephant

Nearest-3 =
**(trees,
animals,
fungus)**



nearest 1
CFA = (trees,
food,
fungus)



nearest 2 animals



fungus
nearest 3



**food
furthest**



fungus
largest area

Evaluation Results: Imagenet1K

Target	Nearest-3	CFA	Imagenet1K	Err_Nearest-3	Err_CFA
food	10.70%	10.45%	11.75%	8.92%	11.06%
weapon	31.81%	31.37%	33.01%	3.65%	4.97%
animal	19.31%	19.32%	22.51%	14.22%	14.17%
oxford	80.14%	81.16%	86.15%	6.98%	5.79%
garment	21.36%	20.87%	24.23%	11.85%	13.88%
person	8.74%	8.47%	9.32%	6.26%	9.16%
sport	34.68%	34.54%	38.39%	9.67%	10.01%
tool	42.58%	42.01%	43.67%	2.51%	3.80%
fungus	23.28%	22.77%	23.63%	1.46%	3.62%
music	45.80%	45.56%	47.54%	3.67%	4.17%
fruit	21.54%	22.23%	25.75%	16.35%	13.70%
plant	13.12%	13.31%	13.52%	3.02%	1.57%

- Average error (across 12 targets): 7.38% (Nearest-3); 7.99% (CFA)

Why Performance modeling ?

- Estimate for make correct design choices
- Study tradeoffs between different choices
- Helps in understanding of system dynamics
- Helps identify bottlenecks and what-if analysis

Performance Modeling Techniques

- White Box modeling
 - Model the internal flow and dynamics of the system
 - Analytical models: computation graph, queueing models
 - Make assumptions for tractability which might not be true in practice
 - Easier to understand
- Black box modeling
 - Model the system as a black box by only observing the input and system's response
 - Machine learning models
 - Linear and non-linear regression models
 - Lacks explainability
- Grey box modeling
 - In between white and black box in terms of modeling of system internals
 - Partial observability of system

Performance of Deep Learning Jobs

Factors affecting performance

Parameter	Dimension	Example Choices
DL Model	Model size Number of layers Neurons per layer Interconnection topology Compute intensity of functions	AlexNet Inception3 ResNet50 VGG16
Framework	Neural network libraries Inter-gpu communication Native distribution support	Tensorflow Pytorch Caffe/Caffe2
Dataset	Modality Size Encoding Training and testing datasets	Imagenet Cifar Places TREC
Batch Size	Number of data samples used in one iteration of the training job	32, 64, 218, 256, 512
Job Resources	Number of CPU threads Type of accelerators Number of accelerators Shared resources	2, 4, 8, 16, 32, 64 NVIDIA K80, P100, V100 1, 2, 4, 8, 16, 32 network, cloud object storage

- Training times can vary from several minutes to days
 - Nature of job: training vs inference
 - Type of resources:
 - Compute: GPUs, TPUs, CPUs
 - Network: Ethernet, Infiniband
- Cloud-providers follow a per-unit hour pricing model for GPU enabled platform

Performance Estimation Challenges

- Both algorithmic and system related challenges
- Performance depends on different software and hardware factors
 - Interdependent, e.g., batch size governed by GPU type, network type
 - Distributed training performance tied to learner communication topology
- DL workloads are domain specific and lack standard representations.
 - Convolutional (computer vision) vs Recurrent neural networks (NLP)
 - Standard log format in Caffe; User specified format in Tensorflow, Pytorch, and Keras
 - Space of DL frameworks and hardware resources is continuously evolving
- Need a generic methodology to works across different cloud based DL platform services
 - Cloud providers support different frameworks and hardware resources

DL Resource Requirement on Mobile Platforms

- NVIDIA TK1 and TX1 developer kits
- TX1 is more powerful than TK1 (256 cuda cores Maxwell vs 192 cuda cores Kepler)

Table 2: Timing benchmarks on AlexNet

Platform	Layerwise Pass (ms)					Total (ms)	Forward Pass (ms)
	CONV	POOL	LRN	ReLU	FC		
TK1	CPU	318.7±0.2	6.1±0.1	103.8±0.0	4.6±0.0	186.3±0.1	619.8±0.2
	GPU	51.42%	0.99%	16.74%	0.75%	30.05%	619.5±0.2
TX1	CPU	24.6±3.5	2.3±0.6	2.4±0.5	5.2±1.2	35.1±5.9	73.3±10.7
	GPU	33.53%	3.15%	3.22%	7.11%	47.95%	54.7±2.4
FLOPs	CPU	66.9±5.3	7.6±0.0	172.4±0.3	2.4±0.0	644.7±5.3	894.3±4.8
	GPU	7.48%	0.85%	19.28%	0.27%	72.09%	892.7±2.3
FLOPs		24.2±8.3	1.3±2.6	2.7±3.0	5.9±5.9	15.2±4.7	52.8±15.7
		45.79%	2.51%	5.12%	11.23%	28.76%	29.3±6.5
						729M	
FLOPs							
91.36%							

Lu et al. Modeling the Resource Requirements of Convolutional Neural Networks on Mobile Devices. 2017

Layerwise time addition does not work

Table 6: Timing benchmarks on ResNet

Platform	Layerwise Pass (ms)							Total (ms)	Forward Pass (ms)
	CONV	POOL	BatchNorm	ReLU	Scale	Eltwise	FC		
TK1	CPU	1830.4±0.4	8.8±0.0	97.1±0.1	64.0±0.1	42.0±0.1	24.8±0.1	5.4±0.0	2072.7±0.4
		88.31%	0.42%	4.68%	3.09%	2.03%	1.20%	0.26%	2072.2±0.3
TX1	GPU	245.8±16.3	5.5±0.6	249.5±11.6	38.7±2.0	76.0±3.3	47.0±2.7	3.9±0.1	673.0±33.4
		36.53%	0.81%	37.08%	5.75%	11.29%	6.98%	0.58%	149.4±4.9
	CPU	362.3±5.4	13.7±0.2	83.5±0.3	33.2±0.1	31.9±3.6	20.4±4.2	22.2±0.1	567.6±7.6
		63.83%	2.41%	14.7%	5.86%	5.62%	3.59%	3.92%	566.8±9.7
	GPU	279.4±42.6	3.0±2.7	198.1±36.8	63.6±31.3	79.8±24.2	34.8±12.9	1.8±2.4	664.7±116.5
		42.03%	0.45%	29.80%	9.57%	12.01%	5.24%	0.27%	104.4±14.0
FLOPs		3866M	2M	32M	9M	11M	6M	2M	3922M
		98.59%	0.05%	0.81%	0.23%	0.27%	0.14%	0.05%	

For GPUs summation of layerwise pass time is much higher than the actual forward pass time. For CPUs Layerwise addition is a good estimate.

PALEO

- An analytical model to estimate the scalability and performance of deep learning systems.
- PALEO decomposes the total execution time into computation time and communication time
- Computation time is calculated from factors including
 - the size of the computation inputs imposed by the network architecture
 - the complexity of the algorithms and operations involved in the network layers
 - the performance of the hardware to be used
- Communication time is estimated based on
 - the computational dependencies imposed by the network
 - the communication bandwidth of the hardware
 - the assumed parallelization schemes
- Once the network architecture and design space choices are fixed, all of the key factors in PALEO can be derived, and we can estimate execution time without actually implementing the entire network and/or an underlying software package.
- Paleo code <https://github.com/TalwalkarLab/paleo>
- Live demo <https://talwalkarlab.github.io/paleo/>

H. Qi, E. Sparks, and A. Talwalkar. Paleo: A performance model for deep neural networks. ICLR 2017

PALEO Modeling Approach

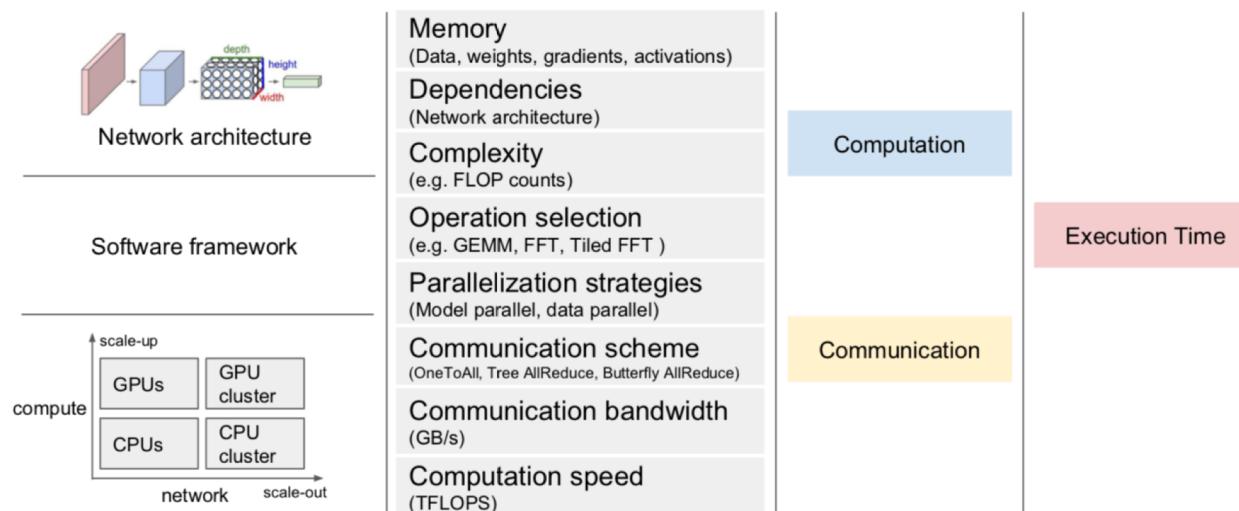


Figure 1: Overview of the PALEO modeling approach. PALEO decomposes execution time into computation time and communication time, which can be derived from various factors implicitly specified by network architectures and hardware configurations.

PALEO Computation Model

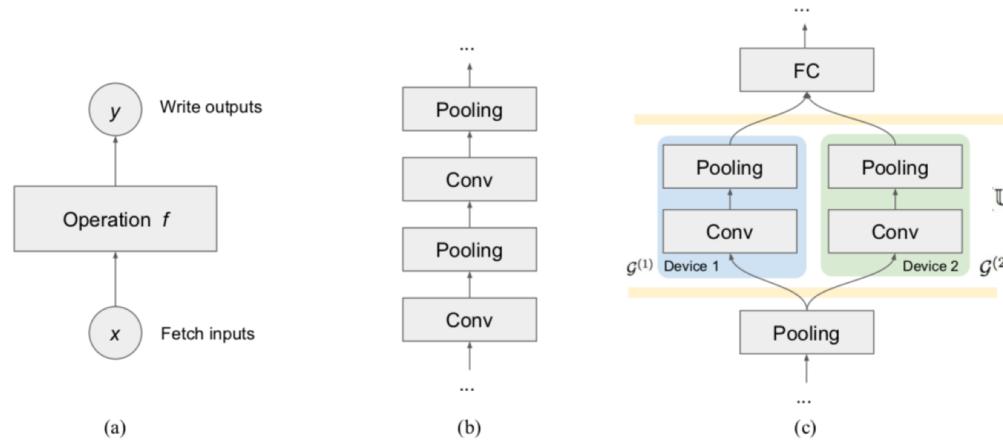


Figure 2: (a) The execution time of a node in the computation graph consists of the time for fetching input, computing results, and writing results to memory. (b) An example of a sequential computation graph segment. (c) An example of a parallel computation graph segment.

Computation Node

- Expressed as a directed graph

$$\mathcal{N} = \langle \{u^{(i)}\}_{i=1}^n, \{(u^{(i)}, u^{(j)})\} \rangle$$

PALEO Computation Model

- Computation time for a single layer

$$T(u) = \mathcal{R}(\text{Pa}(u)) + \mathcal{C}(f, d) + \mathcal{W}(f, d).$$

Fetch input from parent layer Perform computation on device d Write output to local memory

$$\mathcal{C}(f, d) = \text{FLOPs}(f)/\text{speed}(d)$$

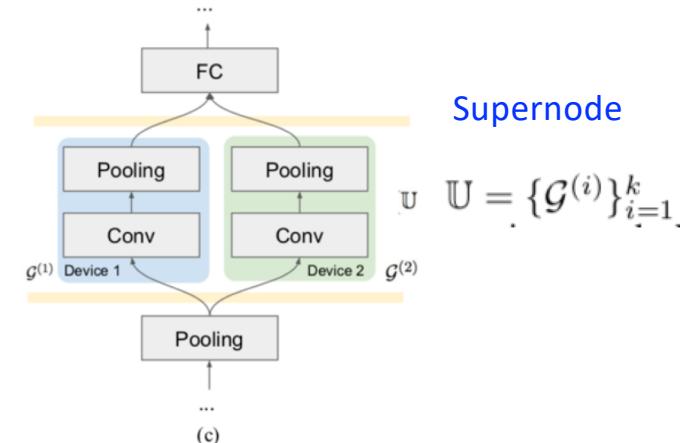
- Computation time for networks

- Sequential structures: $T(\mathcal{N}) = \sum_{i=1}^n T(u^{(i)})$.

- Parallel structures: $T(\mathbb{U}) \in [\max_i T(\mathcal{G}^{(i)}), \sum_i T(\mathcal{G}^{(i)})]$
 the lower bound corresponds to perfect parallelization
 the upper bound corresponds to sequential execution

- Computation modeling for layer operations

- Depends on the implementation used
- Convolution operation: matrix multiplication, Fast Fourier Transform
- Choice of algorithm is problem specific



PALEO Communication Model

- PALEO considers three communications schemes: OneToAll, Tree AllReduce, and Butterfly AllReduce.

$$T_{\text{comm}} = |D|/B.$$

Platform Percent of Peak (PPP)

- Achieving peak FLOPS is a difficult proposition, usually requiring customized libraries developed by organizations with intimate knowledge of the underlying hardware, e.g., Intel's MKL, NVIDIA cuDNN.
- Captures average relative inefficiency of the platform (hardware+framework) compared to peak theoretical FLOPS
- Calculated using observed total throughput and estimated total throughput on a set of workloads

PALEO: Layerwise Evaluation

Table 1: Full pass time of TensorFlow and PALEO estimation on AlexNet and VGG-16.

		Forward pass (ms)	Backward pass (ms)
AlexNet	TensorFlow	44.00	155.10
	PALEO Estimation	45.96	118.44
VGG-16	TensorFlow	400.46	1117.48
	PALEO Estimation	435.46	1077.27

PALEO: Training time estimation

Case 3	
Net	AlexNet
Device	NVIDIA K20
Workers	Up to 8
Bandwidth	6 GB/s
Communication	Various
Parallelization	Hybrid
Platform	cuda-convnet2
One Step Time⁶	
PALEO Estimation	402 ms
Reported Time ⁷	418 ms

Table 4: Comparison between PALEO estimation and Krizhevsky (2014b) for training AlexNet.

Workers	One Weird Trick		PALEO Estimation			
	Hybrid parallelism	Speedup	Hybrid parallelism	Speedup	Data parallelism	Speedup
1	98.95	1×	96.31	1×	96.31	1×
2	50.24	1.95×	49.57	1.94×	55.90	1.72×
4	26.20	3.74×	25.42	3.79×	32.82	3.03×
8	16.68	6.25×	14.37	6.70×	23.65	5.40×

Deep Learning for Predicting Execution Time

- Predicting the execution time for a deep learning network through the use of deep learning.
 - To capture non-linear dependency between different features derived from the computational resource used, the network being trained and the data used for training
 - Learn a representation of the training execution time which is more complex than the linear models
- Approach:
 - Predict execution time for commonly used layers in deep neural networks
 - Execution time required for one training step (forward and backward pass) for processing an individual batch of data.
 - Overall execution time can then be calculated as the time per batch multiplied by the number of batches

Features of a Training Job

- Layer Specific Features
 - Multi-Layer Perceptron
 - Number of inputs
 - Number of neurons
 - Convolutional features
 - Input size and depth
 - Kernel size and output depth
 - Stride size and input padding
 - Pooling features
 - Kernel size
 - Stride size
 - Input padding

Features of a Training Job

- Layer Features
 - Activation function
 - Optimizer
 - Batch size
- Hardware Features
 - GPU type
 - GPU count
 - GPU memory
 - GPU clock speed
 - GPU memory bandwidth
 - GPU core count
 - GPU peak performance
 - Card connectivity

Layerwise Performance Modeling

- Benchmark execution time of individual layer operations (atomic operations)
- The feature set for the atomic operation along with the execution timings are then used to train a fully connected feed forward network. This network can then be used for predicting the execution time for a new operation based just on the feature set.
- Once a prediction has been made for an individual operation these can be combined together and across layers in order to provide a prediction for the overall performance of the deep learning network.
- Working with atomic operations:
 - Reduces the computational time to train whilst also maximising the range of layer types that can be predicted

Full Model Prediction

- Per batch time

$$T_b = \sum_{i=0}^l b_{M(i)}$$

l is the number of layers in the deep neural network

$b_{M(i)}$ is the batch execution time estimate

$M(\bar{i})$

- Execution time of a single epoch

$$E = pT_b$$

p is the number of batches required to process the data

Experimental Setup

Name	Provisioning	Cuda cores	Clock (boost)	Memory	GPU memory bandwidth	Bus
V100	Local	5120	1455 MHz	16 GB HBM2	900 GB/s	NVLink
P100		3584	1303 MHz	16 GB HBM2	732 GB/s	PCIe
GTX1080Ti		3584	1582 MHz	11 GB GDDR5X	484 GB/s	PCIe
M60		4096	1178 MHz	16 GB GDDR5	320 GB/s	PCIe
K80		2496	875 MHz	12 GB GDDR5	240 GB/s	PCIe
K40		2880	875 MHz	12 GB GDDR5	288 GB/s	PCIe

TABLE I
SPECIFICATION OF HARDWARE TESTED

- Training space is very large (~ 10^9 combinations for fully connected and 10^{12} combinations for convolutional neural network)
 - Random subsampling of training space to generate experimental data
- Test Case: Fully Connected Layer
 - *tensorflow.layers.dense* to generate fully connected layers
 - 25K randomly chosen training data points
- Test Case: Convolutional Layer
 - *tensorflow.layers.conv2d*
 - 50K randomly chosen training data points

Convolution layer runtime estimation

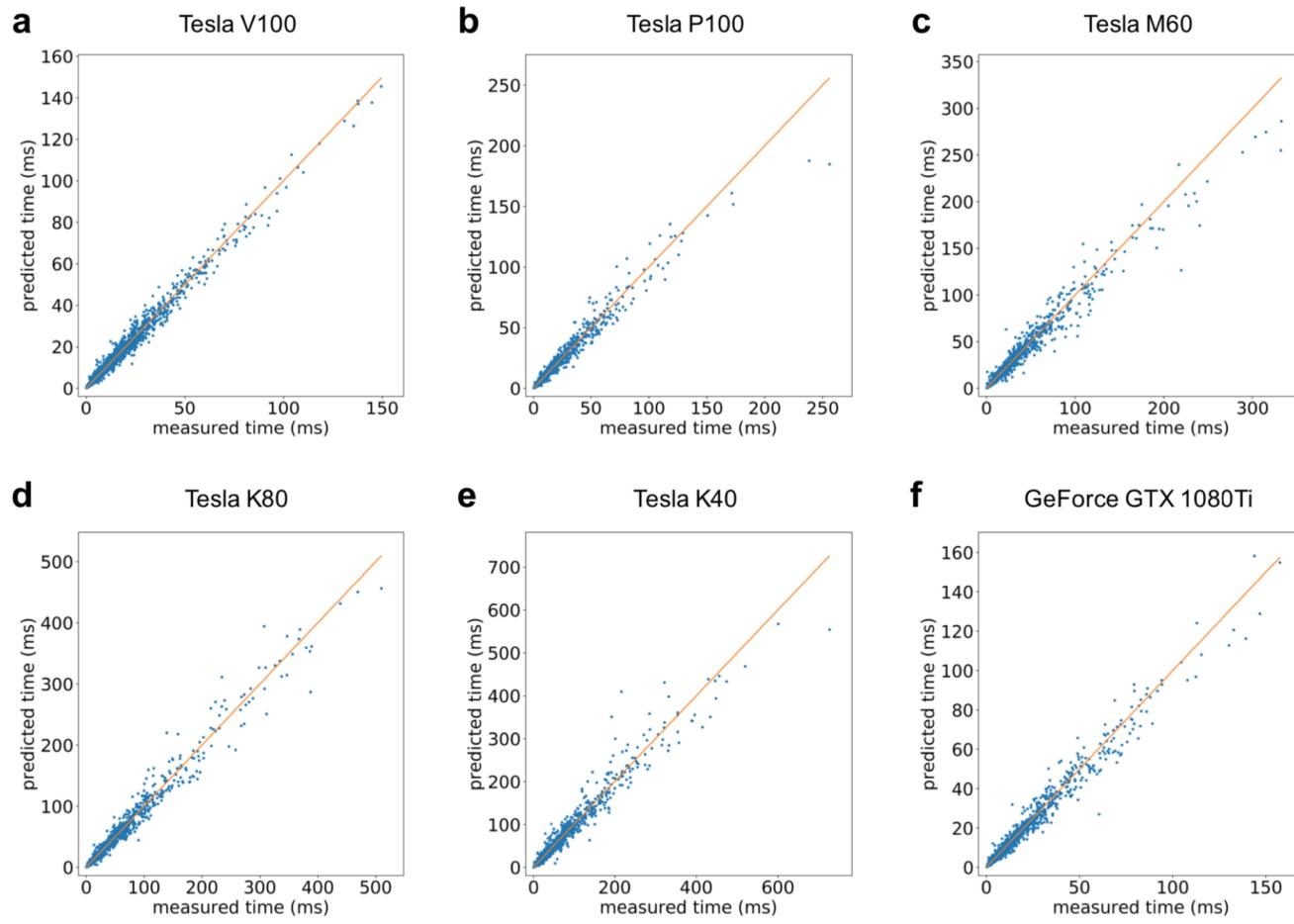
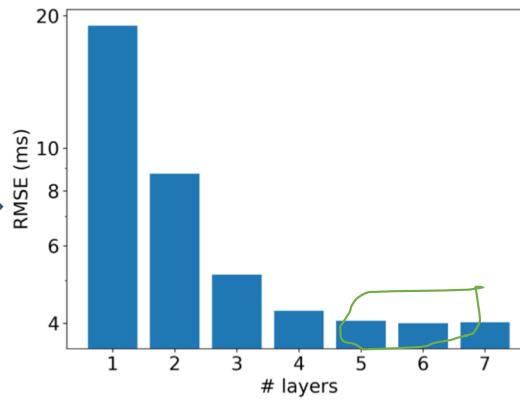
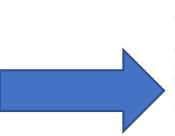
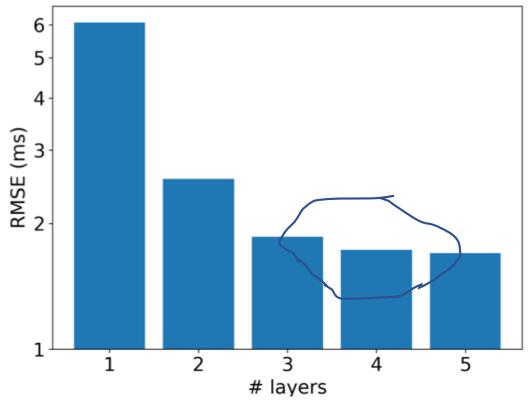


Fig. 3. Predicted time for convolutions versus measured time. **a)** Tesla V100, RMSE of prediction 1.73 ms. **b)** Tesla P100, RMSE 3.32 ms. **c)** Tesla M60, RMSE 6.07 ms. **d)** Tesla K80, RMSE 7.84 ms. **e)** Tesla K40, RMSE 11.90 ms. **f)** Geforce GTX1080Ti, RMSE 2.55 ms

- Separate model for each GPU type
- Older GPUs show longer execution times, and accordingly larger absolute deviations in the predictions

Single performance model for all GPU types

- Features of the model were expanded to include GPU memory bandwidth, GPU clock frequency and the number of CUDA cores.
- More features → more complex prediction model
- Merge data from all different GPU types → bigger size training data
- Can be used to predict performance on a new, unseen hardware



Justus et al. Predicting the Computational Cost of Deep Learning Models. 2018

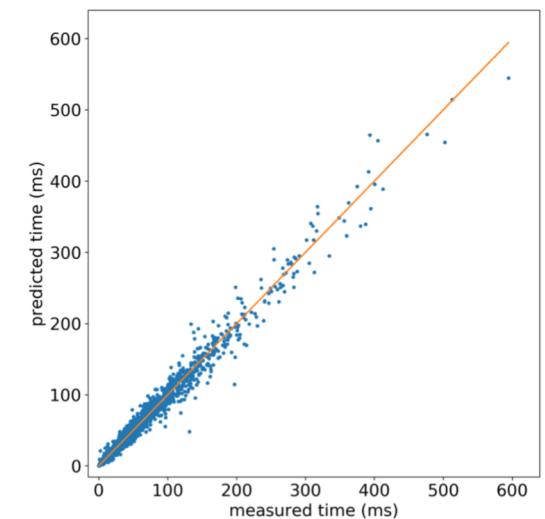
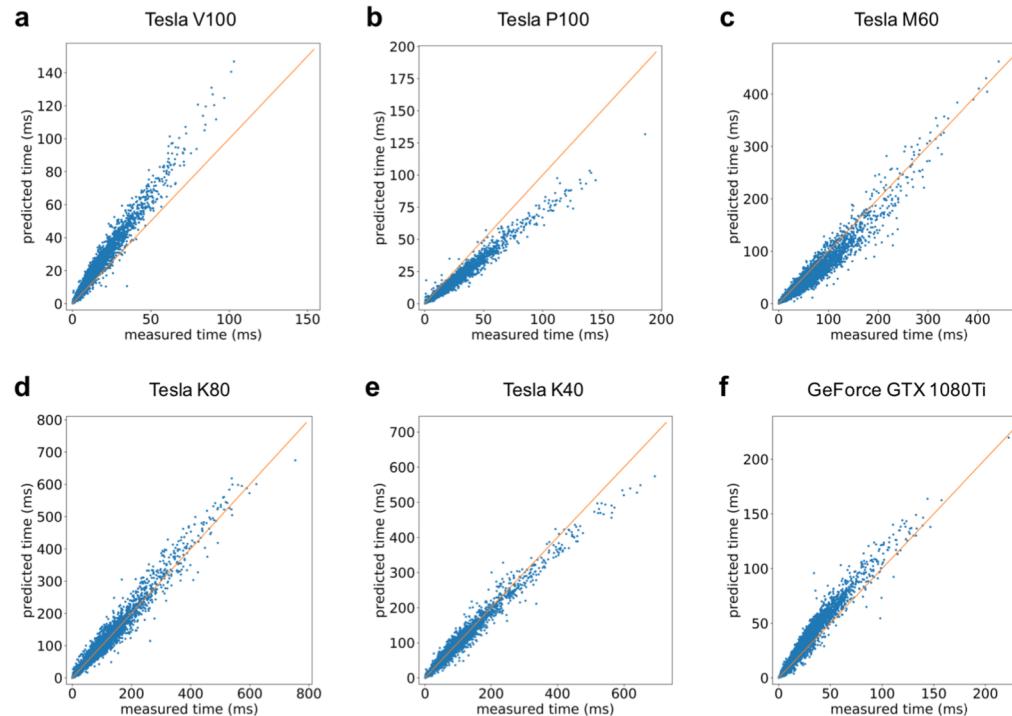


Fig. 6. Predicted time for convolutions versus measured time for a model that has been trained on data from all available GPUs. The RMSE is 3.88 ms.

Single performance model for all GPU types



- Predicted time for convolutions plotted against measured time for a model that has been trained on data from a set of GPUs excluding the one tested
- The uncertainty of predictions can be expected to be particularly large for GPUs with specifications that don't fall into the range of known hardware.

Batch Execution Time Prediction

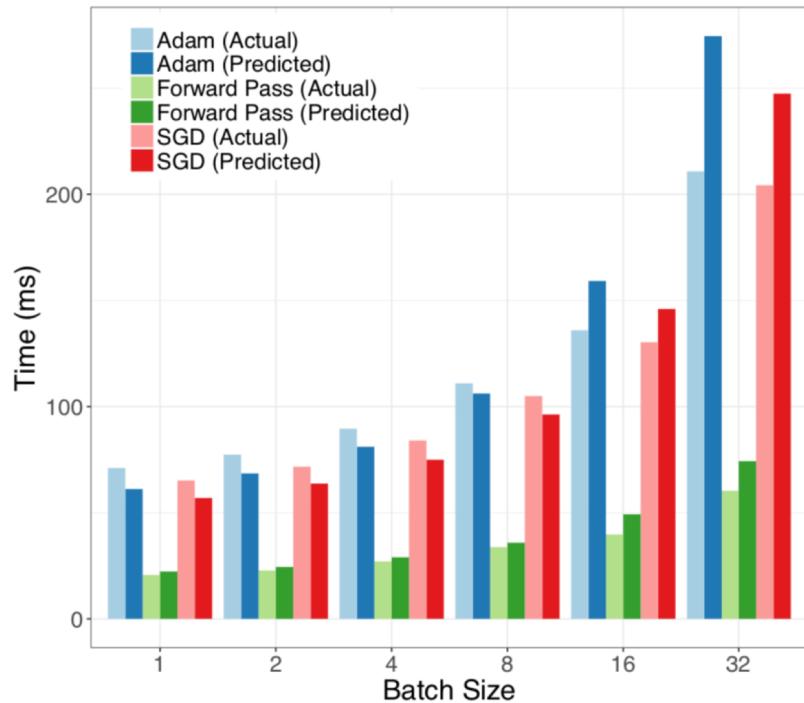


Fig. 9. Predicted versus actual execution time for VGG-16.

Predictive performance modeling of DL training and inference time is promising

- Estimate training time of an epoch and hence the total training time execution
- Study effect of hardware, optimizers etc. on runtime
- Runtime translates to cost
- Cost-benefit tradeoff

AI Gauge

- **Goal**

- Provide runtime estimation service for Deep Learning (DL) training jobs in cloud based DL services
- AI Gauge provides online (for ongoing training jobs) and offline (for new training jobs) runtime estimates

- **Value**

- *End User*

- Explore cost-performance tradeoff for different resource configurations options
 - Estimate remaining runtime of ongoing job alongside accuracy (enhanced user experience)

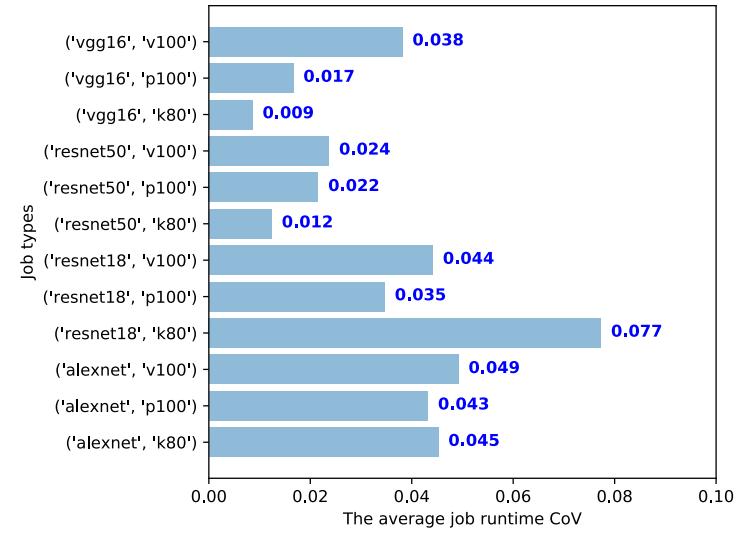
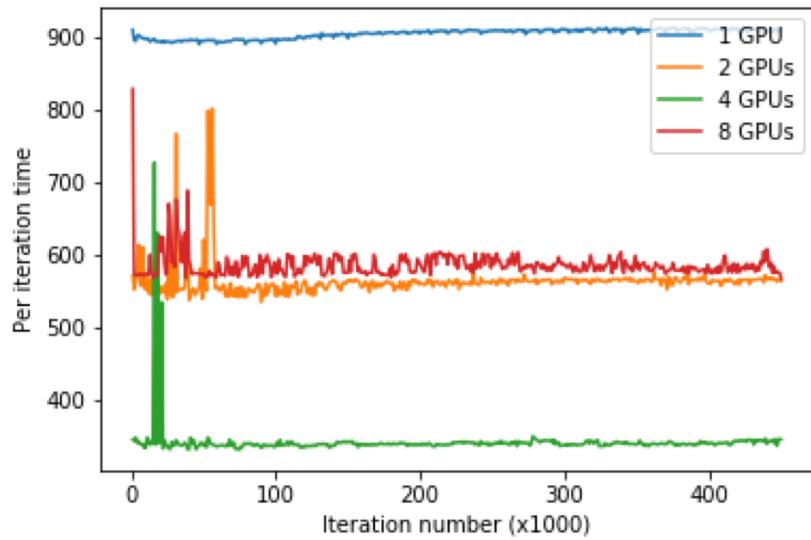
- *Cloud Platform*

- Priority based scheduling and preemption of DL jobs
 - Elastic scaling of DL jobs

AI Gauge: Online Runtime Estimation

- Remaining runtime estimation of an ongoing job:
 - For current hardware configuration
 - For a different hardware configuration (number and type of GPUs)
- Main components
 - Job Logger: get logs, parses, stores in InfluxDB database
 - Runtime estimation computed using metrics from logs and window based moving average
- Service also provides Pareto optimal set of configurations
 - Best configurations in terms of time and cost
 - Cost computed accordingly to the cloud provider's plan for the DL service

Runtime of Deep Learning Jobs



- Very low coefficient of variation (<0.1) in per iteration time series
- Augmented Dicky-Fuller (ADF) test for stationarity: 100% for GANs, 88.9% for RNNs and 80% for ARs.

AI Gauge: Offline Runtime Estimation

- Runtime estimation of a new job:
 - For job features: network, framework, ...
 - For hardware configurations: number of GPU, types of GPU, ...
- Main components
 - Performance data for historical jobs in database
 - Runtime estimation computed using metrics from regression model: linear/nonlinear
- Performance metric for predictive model and hyperparameter tuning

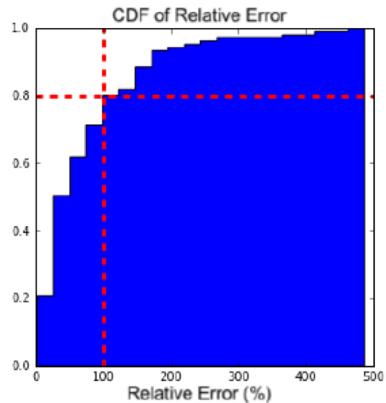
$$\frac{1}{|T|} \sum_{(x,y) \in T} \left[\frac{|y - \phi(x)|}{y} \right]$$

Mean Relative Error

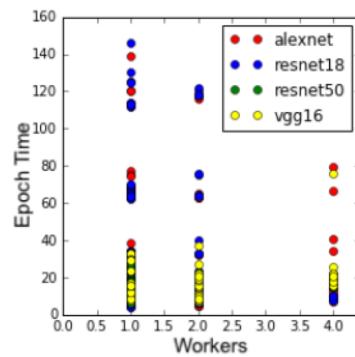
Regression Models for Offline Estimation

- Linear Regression (Optimus 2018):

$$\tilde{y} = \beta_0 + \sum_{i=1}^4 \beta_i I_i + \beta_5 W + \beta_6 \frac{B}{T+1} + \beta_7 \frac{B}{W \times F}$$



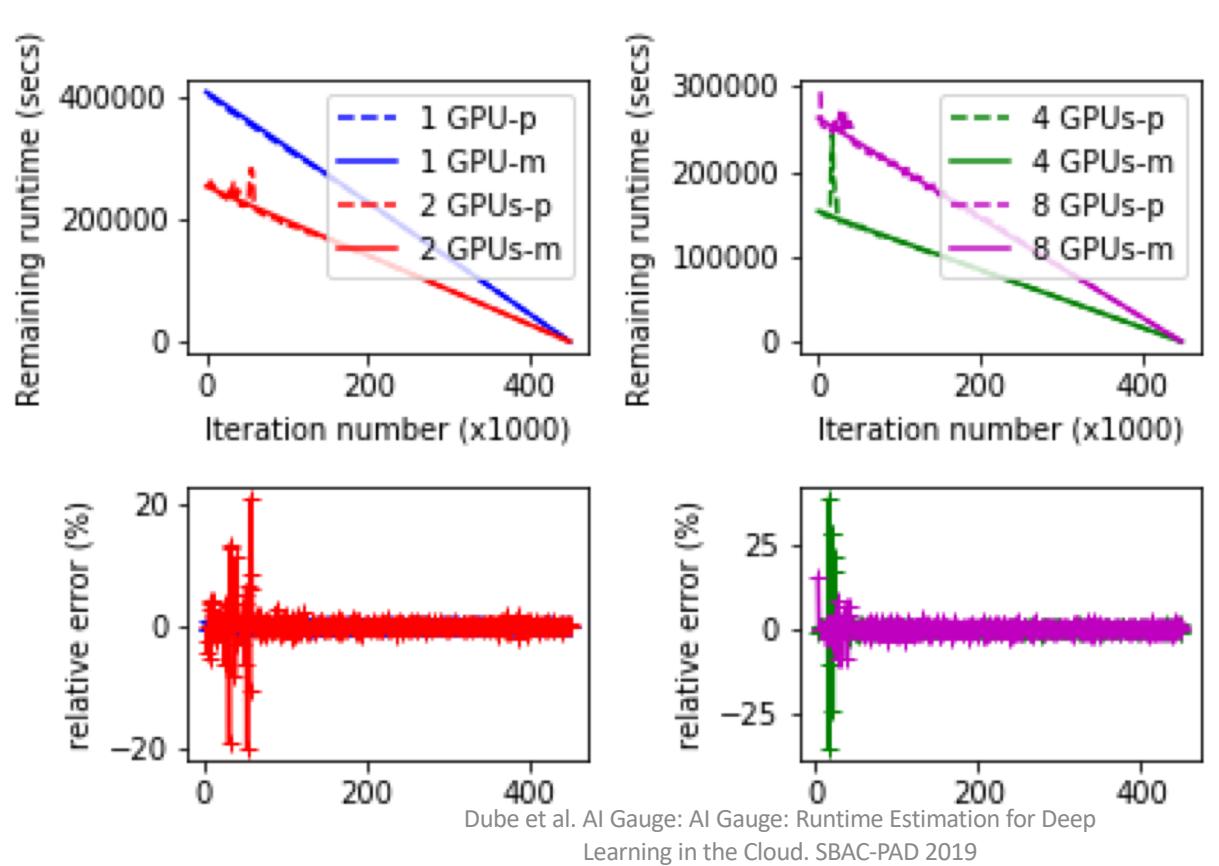
(a) CDF of relative error
in regression model



(b) # of Workers vs Epoch
Time Scatter plot

- Non-linear models: Decision Tree, Random Forest, Gradient Boosting, Neural network

Performance Evaluation: Online Estimation



Offline Estimation: Results

- Nonlinear models: *Decision tree, Random forest, Gradient boosting, 3-layer Neural Network*
 - Independent variables:
 - Categorical (Network, GPU Type)
 - Numerical (Workers, Batch size, Threads)
 - Dependent variable: *Epoch time*
- Model hyperparameters tuned using 5-fold cross-validation and mean relative error as performance metric

Regression Model	Root-Mean-Square Error	Mean of Relative Error
Decision Tree	3.39 sec. (max depth=5)	7 % (max depth=13)
Random Forest	4.00 sec. (# of estimators=800)	8 % (# estimators=1,000)
Gradient Boosting	4.00 sec. (# of estimators=100)	11 % (# estimators=150)
3-layer NN	4.00 sec. (nodes in hidden layers: [250, 200, 200])	14 % (nodes in hidden layers: [250, 200, 200])
Linear Regression	21.711 sec.	84 %

Seminar Reading List

- **Transfer Learning in Vision**
 - Bhattacharjee et al. [Distributed learning of deep feature embedding for visual recognition tasks](#). IBM J of R & D, 2017
- **Weakly supervised learning in vision**
 - Chen et al. [Webly Supervised Learning of Convolutional Networks](#). 2015
 - Joulin et al. [Learning Visual Features from Large Weakly Supervised Data](#). 2015
 - Dube et al. [Automatic Labeling of Data for Transfer Learning](#). Deep Vision, CVPR 2019
 - Yalniz et al. [Billion-scale semi-supervised learning for image classification](#). 2019
- **DL resource estimation**
 - Justus et al. [Predicting the Computational Cost of Deep Learning Models](#). 2018
 - Qi et al. [PALEO: A PERFORMANCE MODEL FOR DEEP NEURAL NETWORKS](#). ICLR 2017
 - Lu et al. [Modeling the Resource Requirements of Convolutional Neural Networks on Mobile Devices](#). 2017
 - Dube et al. [AI Gauge: Runtime Estimation for Deep Learning in the Cloud](#). SBAC-PAD 2019

Blogs, Code Links

- Yalniz et al. [Billion-scale semi-supervised learning for state-of-the-art image and video classification](#). Facebook AI Blog. Oct 2019
- Paleo code repository <https://github.com/TalwalkarLab/paleo>
- Paleo Live demo <https://talwalkarlab.github.io/paleo/>