COMS E6998 010

# Practical Deep Learning Systems Performance

**Lecture 7 10/29/20**

# Logistics

- Homework 3 due Nov 7 by 11:59 PM
- Homework 4 will be posted on Nov 5; due Nov 20
- Submit seminar recording by Sunday Nov 1

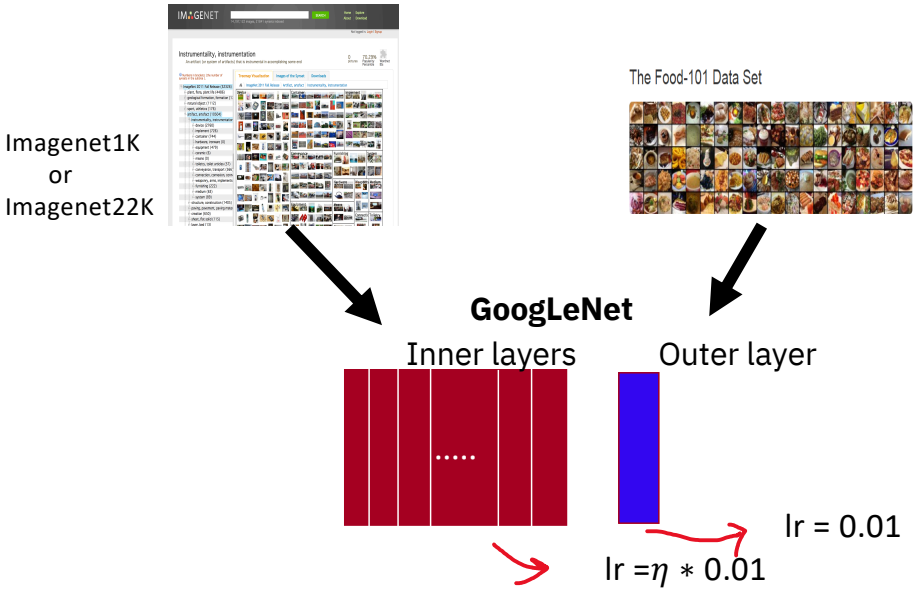
# Recall from last lecture

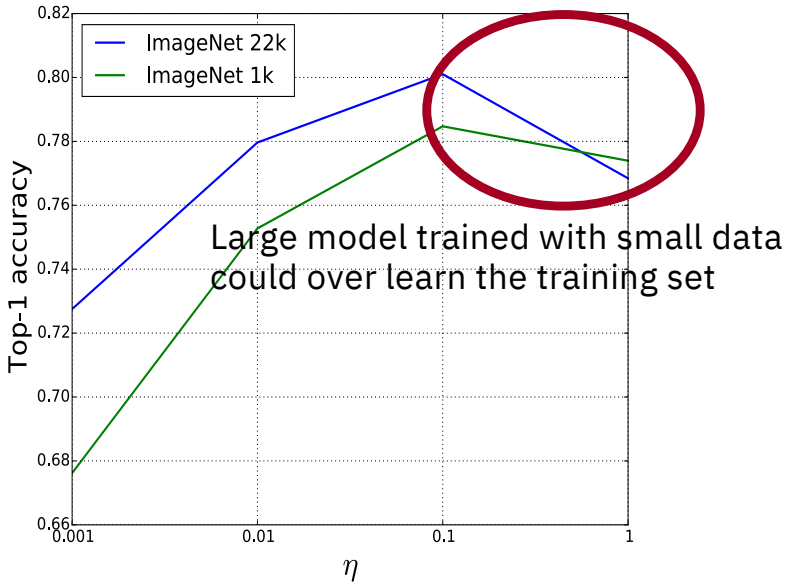
- LeNet, Alexnet, VGG, Resnet; ILSVRC challenge
- Performance comparison of different CNNs
- Object localization and detection
- R-CNN and Fast R-CNN
- Transfer Learning and Transferability of features
- Predict to Learn (P2L) based source model selection
- Finetuning and its degree

# 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?
  - Imagenet1K (1000 classes ,~1.3M images,) -> Food101 (101 classes, 101000 images) -> Greek Food
  - Imagenet22K (22K classes ,~15M images,) -> 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

Bhattacharjee et al, "Distributed learning of deep feature embedding for visual recognition tasks", IBM J of R & D, 2017

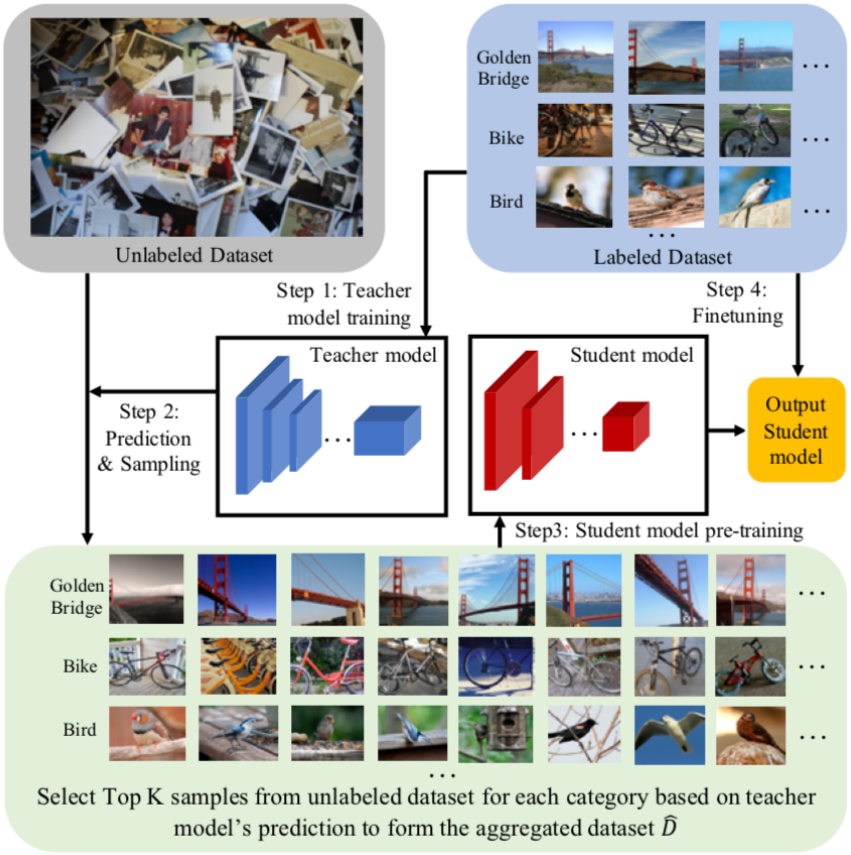
# Exploiting Web Data for Training

- Web and social media are most important sources of data for vision research
- Vision datasets such as ImageNet (classification), PASCAL VOC (object detection), and MS COCO (object detection) created from Google or Flickr by harnessing human intelligence to filter out the noisy images and label object locations
- 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 ?
  - Semi-supervised learning: using supervised learning to train teacher model and use it to classify unlabeled images
  - Weakly supervised learning, e.g., using hashtags, captions of images from social media sites without any supervised learning

# Semi-supervised Learning




(1) Train a teacher model on labeled data $\mathcal{D}$;

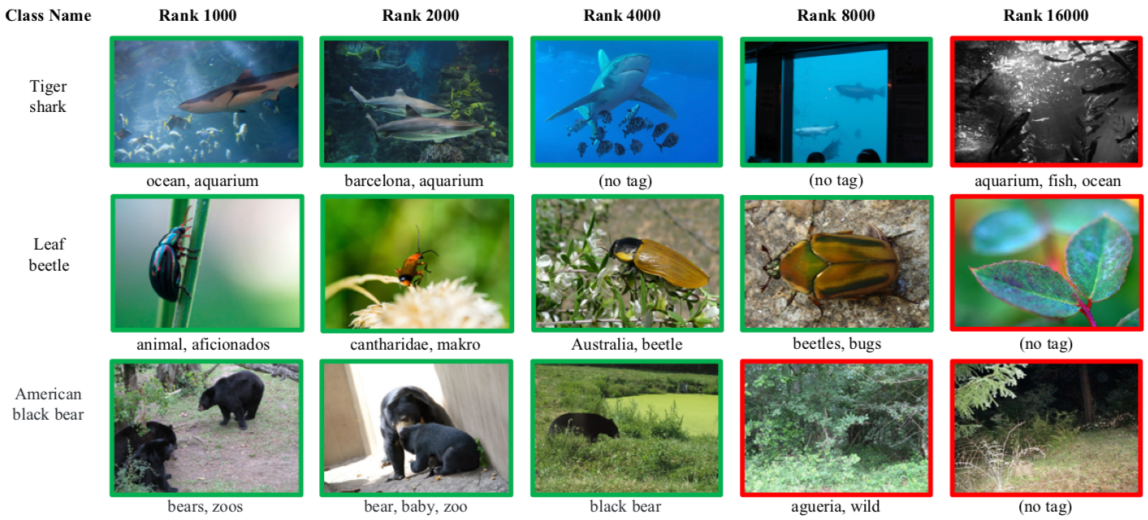
(2) Run the trained model on unlabeled data $\mathcal{U}$ and select relevant examples for each label to construct a new labeled dataset $\hat{\mathcal{D}}$;

(3) Train a new student model on $\hat{\mathcal{D}}$;

(4) Fine-tune the trained student on the labeled set $\mathcal{D}$.

Yalniz et al. Billion-scale semi-supervised learning for image classification. 2019

# Labeling unlabeled images




- Teacher model is run on each example in unlabeled dataset to obtain the softmax prediction vector
- For each image, the classes associated with the **P** highest scores are retained
- For each class rank the images based on the corresponding classification scores
- Top-**K** images from each class is taken to create a new dataset

- K: Number of top samples from unlabeled dataset to keep for each class in the new dataset
- Probability of introducing false positive becomes higher as K increases

# Performance of teacher-student learning

Teacher model: ResNext-101-32x48

Supervised baseline

| Student Model | Ours: Semi-supervised Pre-training | Ours: Semi-supervised Fine-tuned | Fully Supervised |
|---|---|---|---|
| ResNet-18 | 68.7 | **72.6** | 70.6 |
| ResNet-50 | 75.9 | **79.1** | 76.4 |
| ResNext-50-32x4 | 76.7 | **79.9** | 77.6 |
| ResNext-101-32x4 | 77.5 | **80.8** | 78.5 |
| ResNext-101-32x8 | 78.1 | **81.2** | 79.1 |
| ResNext-101-32x16 | 78.5 | **81.2** | 79.6 |

Table 2: ImageNet1k-val top-1 accuracy for students models of varying capacity before and after fine-tuning compared to corresponding fully-supervised baseline models.

| Teacher Model | Teacher # Params | Teacher top-1 | Student top-1 | Gain (%) |
|---|---|---|---|---|
| ResNet-18 | 8.6M | 70.6 | 75.7 | -0.7 |
| ResNet-50 | 25M | 76.4 | 77.6 | +1.2 |
| ResNext-50-32x4 | 25M | 77.6 | 78.2 | +1.8 |
| ResNext-101-32x4 | 43M | 78.5 | 78.7 | +2.3 |
| ResNext-101-32x8 | 88M | 79.1 | 78.7 | +2.3 |
| ResNext-101-32x16 | 193M | 79.6 | 79.1 | +2.7 |
| ResNext-101-32x48 | 829M | 79.8 | 79.1 | +2.7 |

Table 3: Varying the teacher capacity for training a ResNet-50 student model with our approach. The gain is the absolute accuracy improvement over the supervised baseline.

# Performance of teacher-student learning

Sensitivity to size of unlabeled dataset

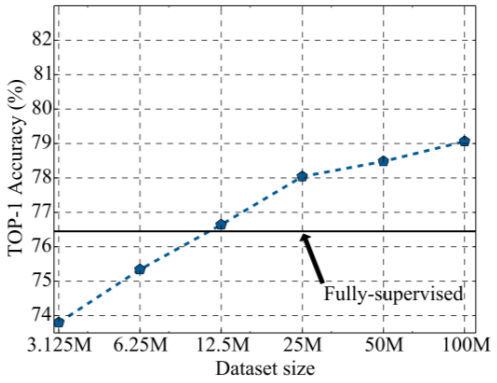



Figure 3: ResNet-50 student model accuracy as a function of the size of the unlabeled dataset $\mathcal{U}$.

Sensitivity to training iterations

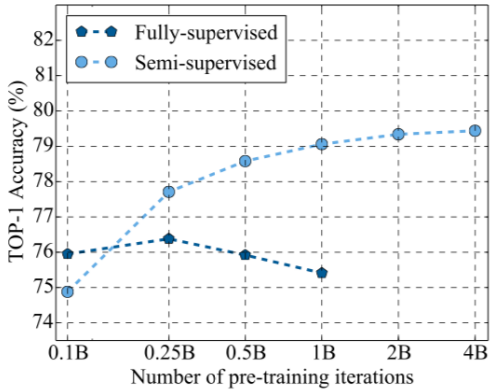



Figure 4: Effect of number of training iterations on the accuracy of fully-supervised and semi-supervised ResNet-50 student models.

Sensitivity to K

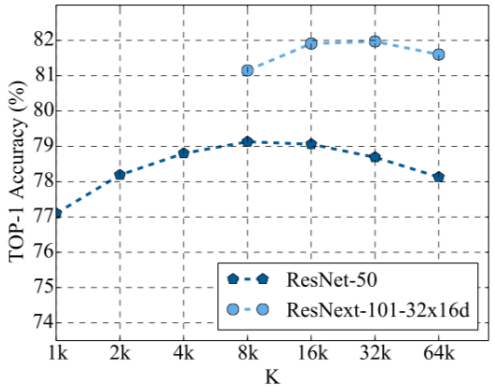
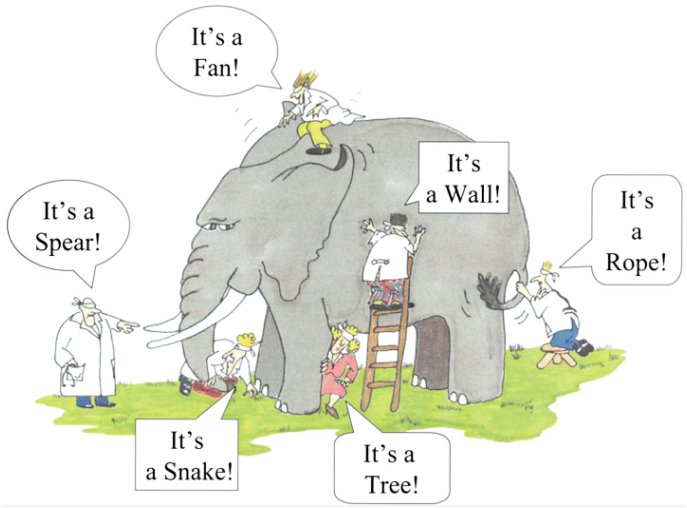
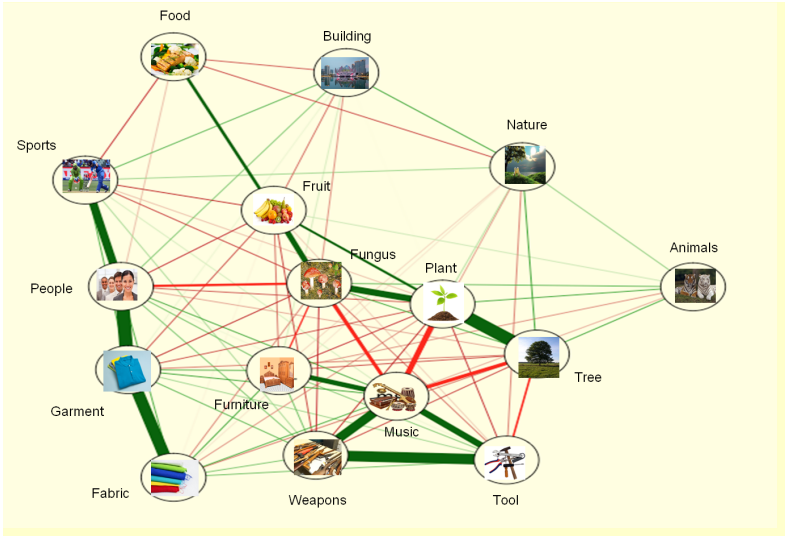



Figure 5: Student model accuracies as a function of the sampling hyper-parameter $K$.

# Divergence based labeling

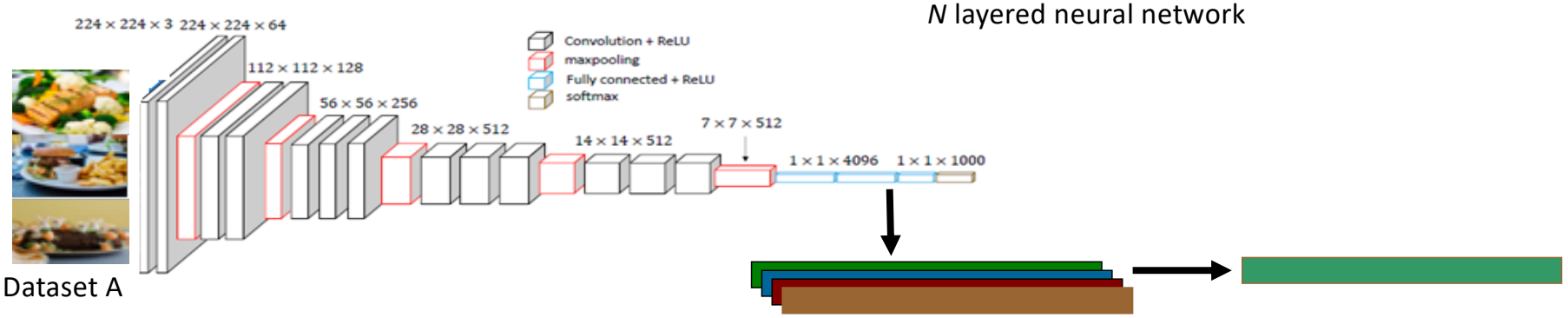
Divergence in feature space between an image with existing datasets
Measure of how close the image is to the dataset in feature space



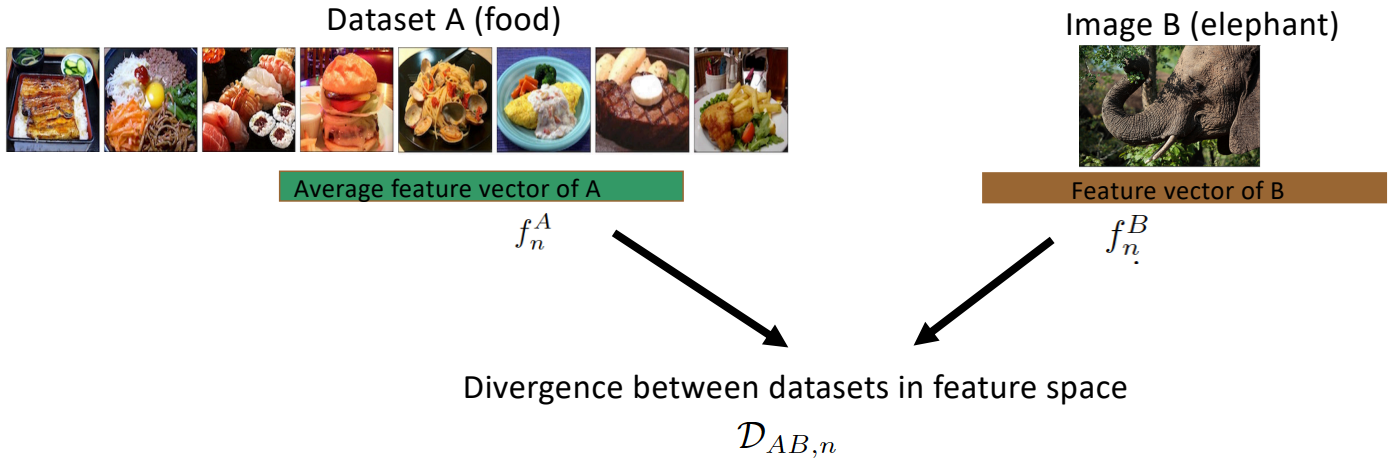

Dube et al. Automatic Labeling of Data for Transfer Learning. Deep Vision 2019

# Feature Vector Embeddings




- Average over the activations from layer $n$
- For $n = N - 1$, vector of 4096 dimension

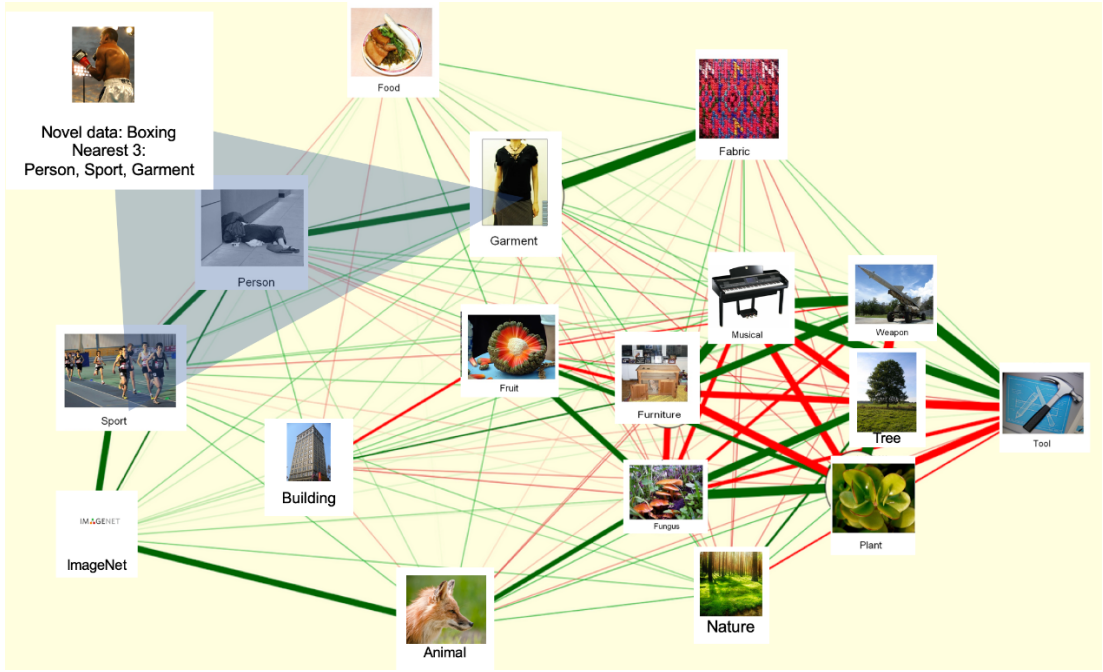
# Divergence in Feature Space



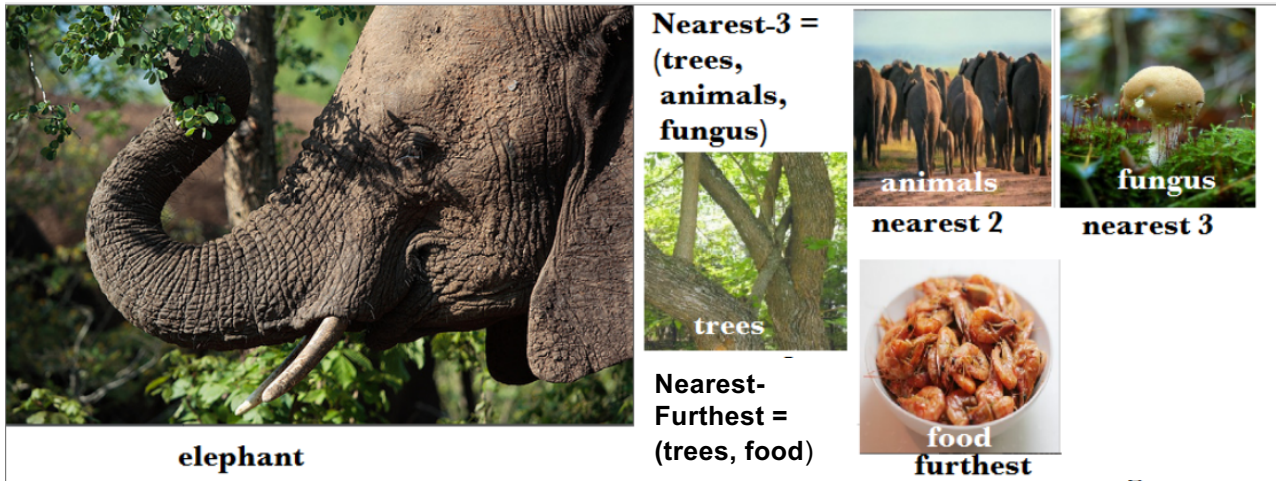

- Different metrics to measure divergence
  - Cosine similarity
  - Kullback-Leibler divergence

# Divergence based pseudo-labeling

Nearest-N
(Nearset-1: closest , Nearest-2: closest two)



# Visual Example: Nearest-3 and Nearest-Furthest



# Why Performance modeling ?

- Helps in studying sensitivity of system performance to different workload related parameters (algorithmic and platform)
- Study tradeoffs between different choices (compute units, hyperparameters etc.)
- Early estimate for making correct design choices
- Helps identify bottlenecks and what-if analysis

# Performance Modeling Techniques

- White Box modeling
  - Model the internal flow and dynamics of the system
  - Analytical models (equation based) : 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<br>Number of layers<br>Neurons per layer<br>Interconnection topology<br>Compute intensity of functions | AlexNet<br>Inception3<br>ResNet50<br>VGG16 |
| Framework | Neural network libraries<br>Inter-gpu communication<br>Native distribution support | Tensorflow<br>Pytorch<br>Caffe/Caffe2 |
| Dataset | Modality<br>Size<br>Encoding<br>Training and testing datasets | Imagenet<br>Cifar<br>Places<br>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<br>Type of accelerators<br>Number of accelerators<br>Shared resources | 2, 4, 8, 16, 32, 64<br>NVIDIA K80, P100, V100<br>1, 2, 4, 8, 16, 32<br>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
- DL on cloud:
  - Cloud providers support different frameworks and hardware resources
  - Need a generic methodology to works across different cloud-based DL platform services

# DL Resource Requirement on Mobile Platforms

- Resource requirement of the forward path of CNNs
- Deployed several Caffe based CNN models mobile platforms
- NVIDIA TK1 and TX1 developer kits
- TX1 is more powerful than TK1 (256 cuda cores Maxwell vs 192 cuda cores Kepler)

compute bound

memory bound

**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 | 619.5±0.2 |
| | | **51.42%** | 0.99% | 16.74% | 0.75% | 30.05% | | |
| | GPU | 24.6±3.5 | 2.3±0.6 | 2.4±0.5 | 5.2±1.2 | 35.1±5.9 | 73.3±10.7 | 54.7±2.4 |
| | | 33.53% | 3.15% | 3.22% | 7.11% | **47.95%** | | |
| TX1 | 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 | 892.7±2.3 |
| | | 7.48% | 0.85% | 19.28% | 0.27% | **72.09%** | | |
| | GPU | 24.2±8.3 | 1.3±2.6 | 2.7±3.0 | 5.9±5.9 | 15.2±4.7 | 52.8±15.7 | 29.3±6.5 |
| | | **45.79%** | 2.51% | 5.12% | 11.23% | 28.76% | | |
| FLOPs | | 666M | 1M | 2M | 0.7M | 59M | 729M | |
| | | **91.36%** | 0.14% | 0.27% | 0.10% | 8.09% | | |

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 | 2072.2±0.3 |
| | | **88.31%** | 0.42% | 4.68% | 3.09% | 2.03% | 1.20% | 0.26% | | |
| | 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 | 149.4±4.9 |
| | | 36.53% | 0.81% | **37.08%** | 5.75% | 11.29% | 6.98% | 0.58% | | |
| TX1 | 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 | 566.8±9.7 |
| | | **63.83%** | 2.41% | 14.7% | 5.86% | 5.62% | 3.59% | 3.92% | | |
| | 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 | 104.4±14.0 |
| | | **42.03%** | 0.45% | 29.80% | 9.57% | 12.01% | 5.24% | 0.27% | | |
| 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.

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

# Observations

- Overhead of time measurement on GPUs is very high
- CUDA supports asynchronous kernel execution
- For timing measurement explicit synchronization is required
  - Call to cudaDeviceSynchronize API to make sure that all cores have finished their tasks

# 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 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

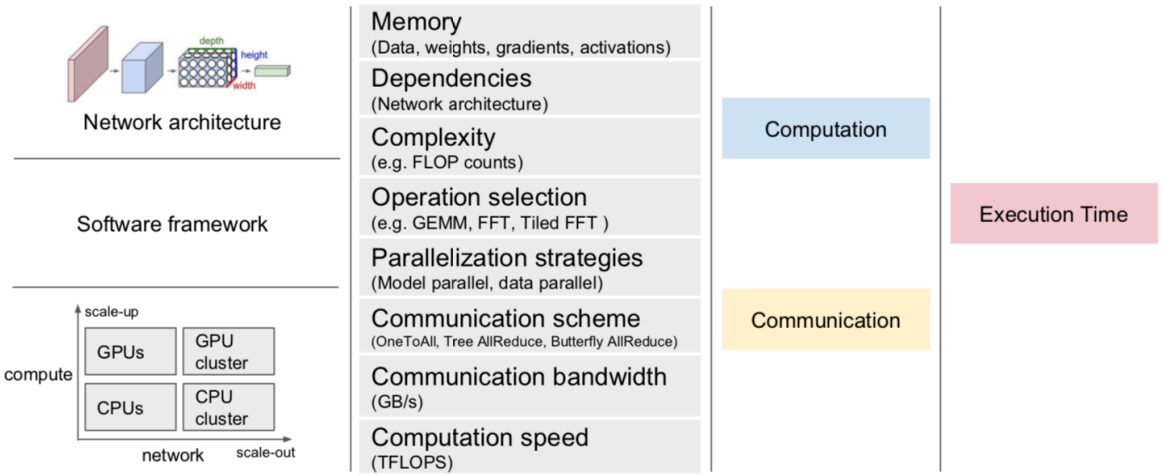



Figure 1: Overview of the PALEO modeling approach. PALEO decomposes execution time into computation time and communication time, which can be derived from various factors implicitly specified by network architectures and hardware configurations.

# PALEO Computation Model

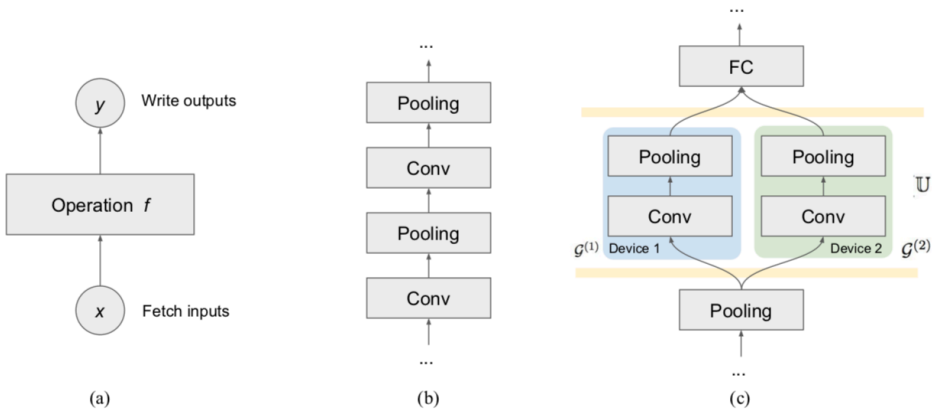



Figure 2: (a) The execution time of a node in the computation graph consists of the time for fetching input, computing results, and writing results to memory. (b) An example of a sequential computation graph segment. (c) An example of a parallel computation graph segment.

## Computation Node

- Expressed as a directed graph $\mathcal{N} = \langle \{u^{(i)}\}_{i=1}^{n}, \{(u^{(i)}, u^{(j)})\} \rangle$

- Each node in the graph is associated with an operation and a device on which it is executed

# PALEO Computation Model

- Computation time for a single layer

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

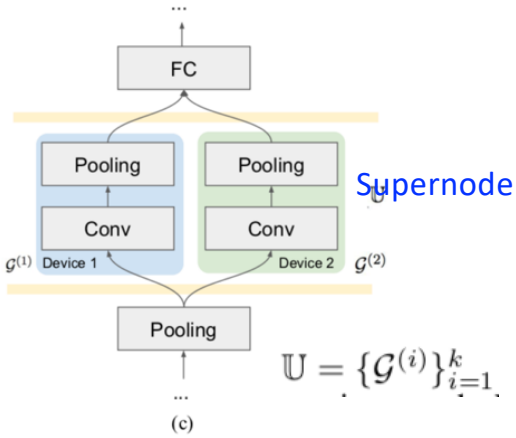
Time to fetch input from parent layer

Time to compute operation f on device d

Time to write output to local memory

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

- 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
  - Example: Inception module




$\mathbb{U} = \{\mathcal{G}^{(i)}\}_{i=1}^{k}$

# FLOPs calculation

- Several optimized implementations to perform convolution
- FLOPs depend on the choice of algorithm for convolution
- Choice of algorithm – matrix multiplication or FFT – is problem specific
  - Depends on the filter size, strides, input size of the convolutional layers, and memory workspace.
- Two common approaches are employed in existing DNN software frameworks and libraries to choose between these algorithms:
  1. Using predefined heuristics based on offline benchmarks;
  2. Autotuning to empirically evaluate available algorithms on the given specification

# PALEO Communication Model

- PALEO considers three communications schemes: OneToAll, Tree AllReduce, and Butterfly AllReduce.
- Time to transfer |D| data between two workers with a communication bandwidth B

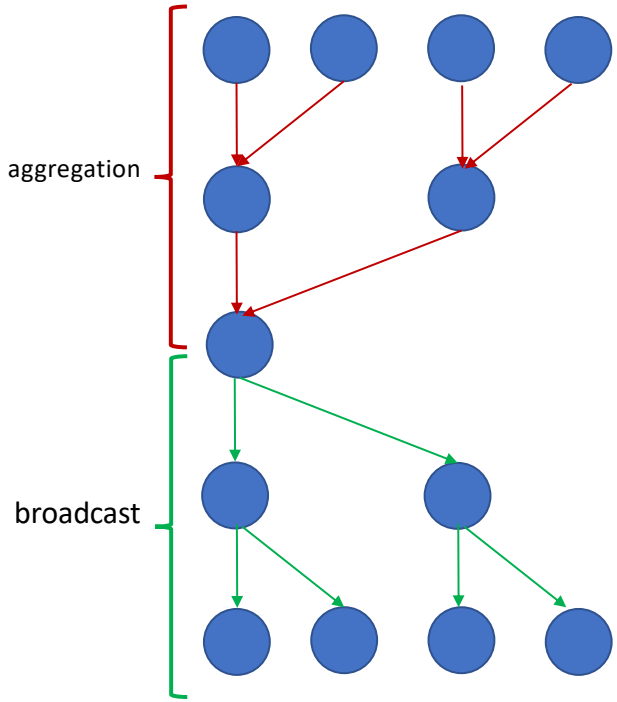
$$T_{\text{comm}} = |D|/B$$

- Depending on the communication scheme the communication time can have different scalability with number of learners (K)

# Communication schemes and time

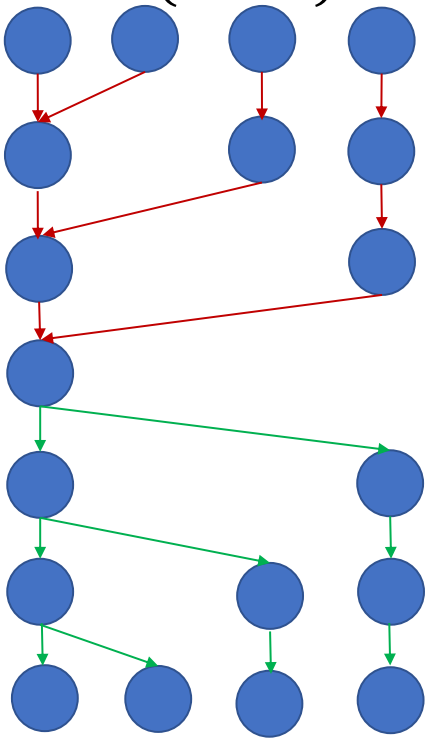
TreeAllReduce

$2log_2K$



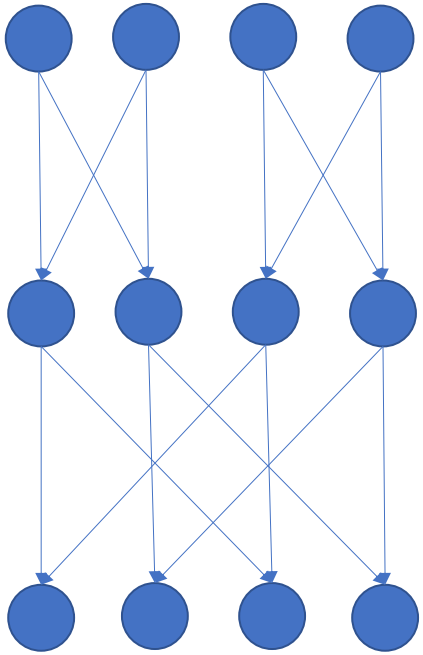

OneToAll

$2(K-1)$



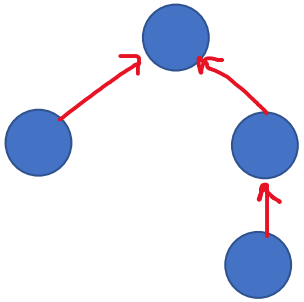
Butterfly AllReduce

$log_2K$

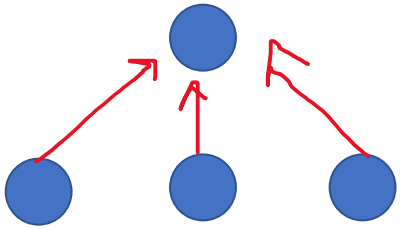


# Alternative representation

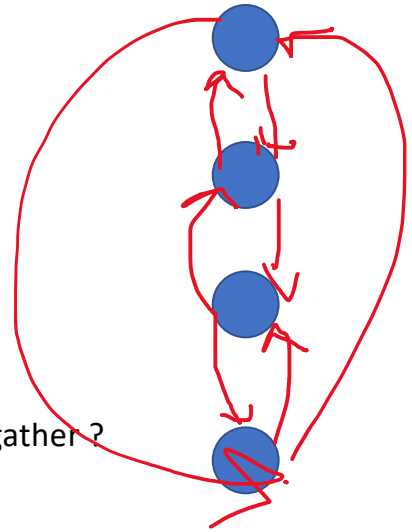
TreeAllReduce



OneToAll



Butterfly AllReduce



- What about number of communication rounds in AllReduce with scatter-gather ?
- Is it same as which scheme on this slide ?
- What about communication time ?
- Can you use any of the above scheme and achieve same communication time scaling with K as AllReduce ?

# 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 (Math Kernel Library), NVIDIA cuDNN (CUDA Deep Neural Network) library.
- 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 benchmark consisting of a set of representative DL workloads
- Given observed total throughput and estimated total throughput on this benchmark we fit a scaling constant to estimate a *platform percent of peak* (PPP) parameter which captures the average relative inefficiency of the platform compared to peak FLOPS
- Calculate PPP for both compute and communication

# 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 Time**[6] | |
| PALEO Estimation | 402 ms |
| Reported Time[7] | 418 ms |

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

| | **One Weird Trick** | | **PALEO Estimation** | | | |
|---|---|---|---|---|---|---|
| | Hybrid parallelism | | Hybrid parallelism | | Data parallelism | |
| Workers | Train Time (h) | Speedup | Train Time (h) | Speedup | Train Time (h) | 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.
  - 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

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

# 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

$M(i)$ type of layer 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 | Cloud | 3584 | 1303 MHz | 16 GB HBM2 | 732 GB/s | PCIe |
| GTX1080Ti | Local | 3584 | 1582 MHz | 11 GB GDDR5X | 484 GB/s | PCIe |
| M60 | Cloud | 4096 | 1178 MHz | 16 GB GDDR5 | 320 GB/s | PCIe |
| K80 | Cloud | 2496 | 875 MHz | 12 GB GDDR5 | 240 GB/s | PCIe |
| K40 | Local | 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

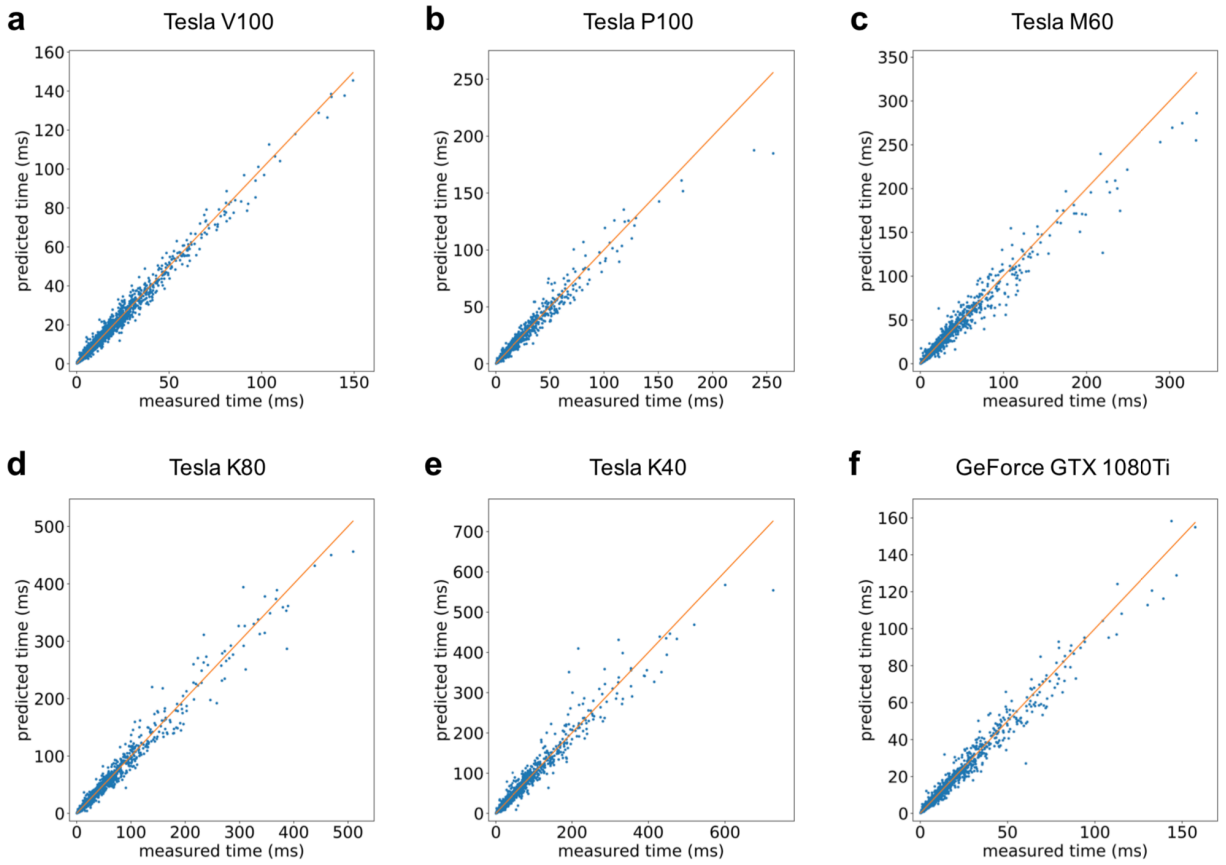


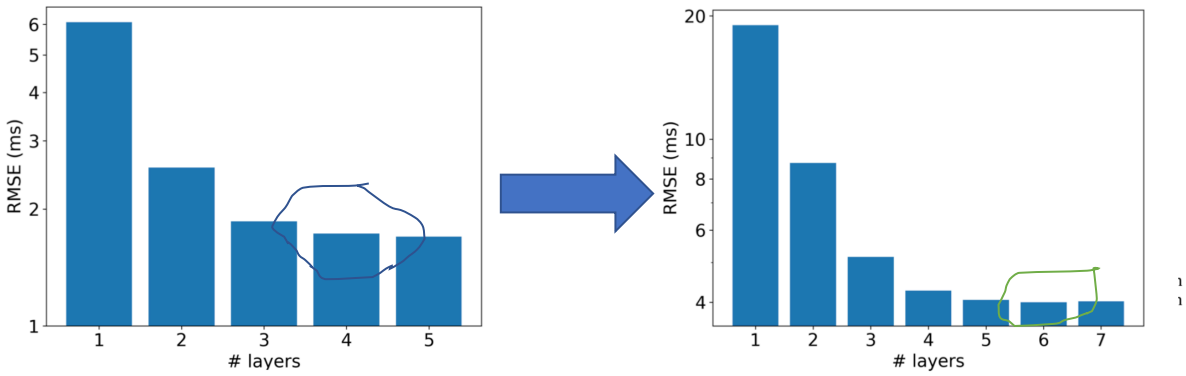

Fig. 3. Predicted time for convolutions versus measured time. **a)** Tesla V100, RMSE of prediction 1.73 ms. **b)** Tesla P100, RMSE 3.32 ms. **c)** Tesla M60, RMSE 6.07 ms. **d)** Tesla K80, RMSE 7.84 ms. **e)** Tesla K40, RMSE 11.90 ms. **f)** Geforce GTX1080Ti, RMSE 2.55 ms

- Separate model for each GPU type

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

# 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

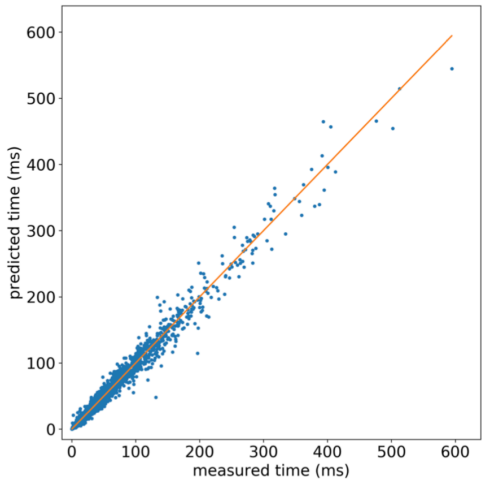
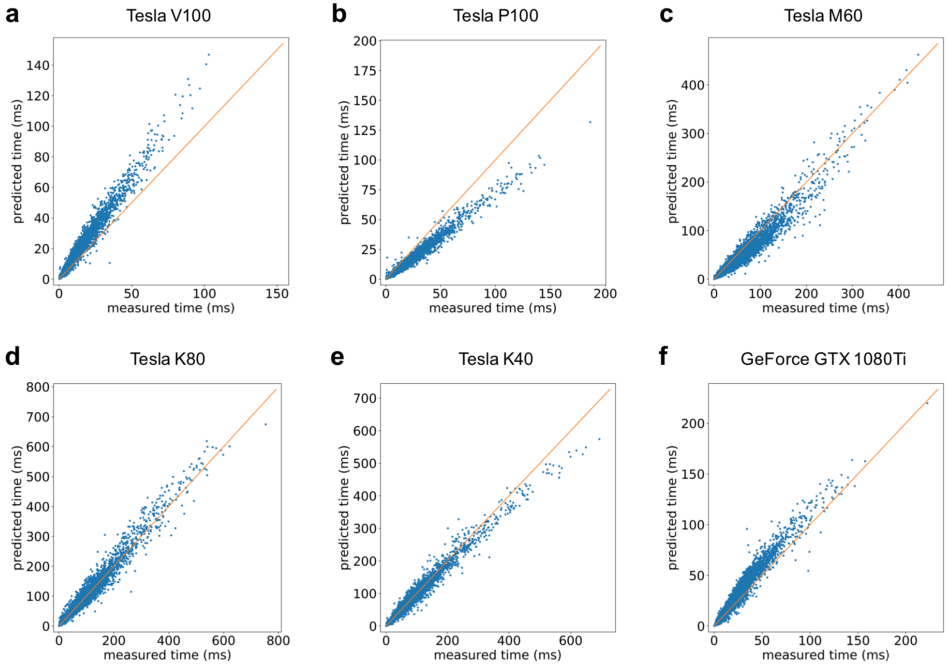



Fig. 6. Predicted time for convolutions versus measured time for a model that has been trained on data from all available GPUs. The RMSE is 3.88 ms.

# 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.

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

# Batch Execution Time Prediction

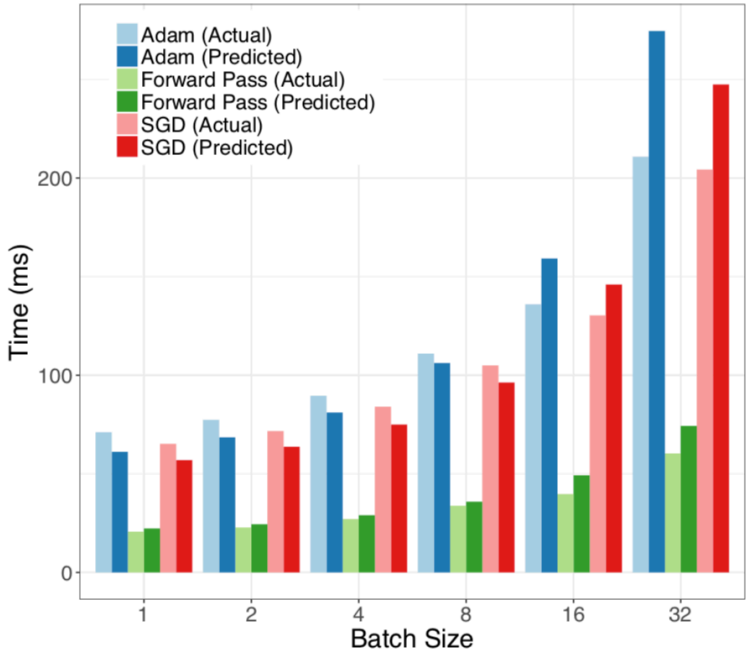



Fig. 9. Predicted versus actual execution time for VGG-16.

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

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

# 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

Dube et al. AI Gauge: AI Gauge: Runtime Estimation for Deep Learning in the Cloud. SBAC-PAD 2019

# 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

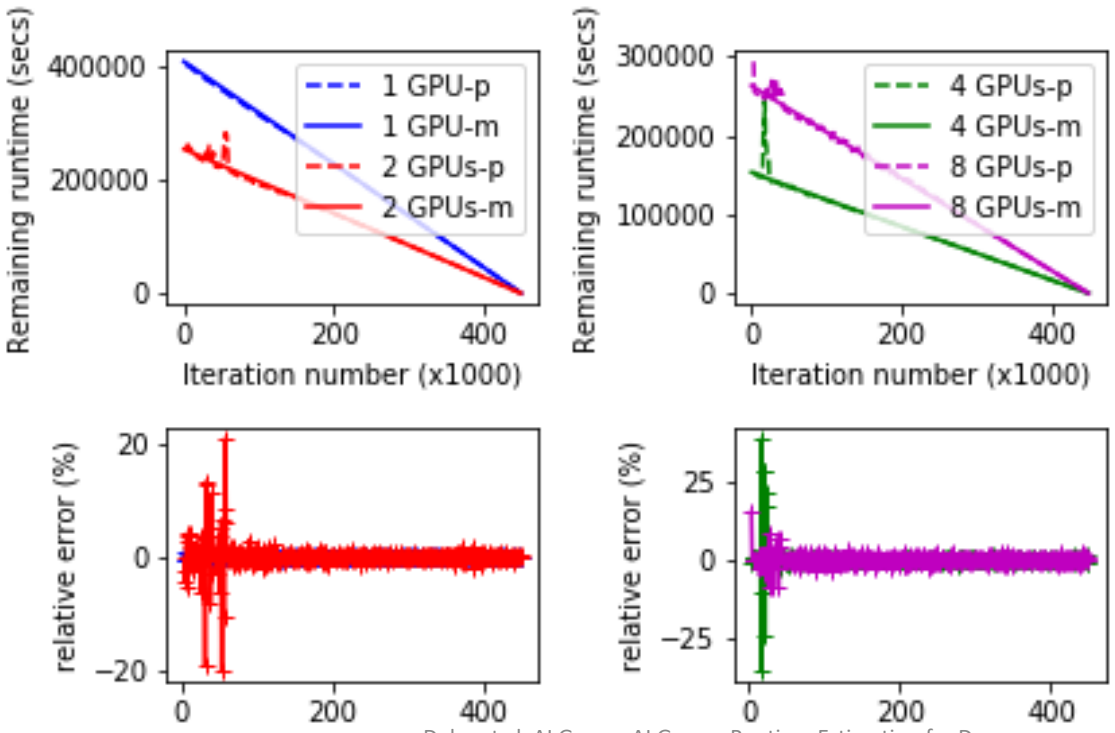
# AI Gauge: Offline Runtime Estimation

- 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

# Performance Evaluation: Online Estimation




Dube et al. AI Gauge: AI Gauge: Runtime Estimation for Deep Learning in the Cloud. SBAC-PAD 2019

# Offline Estimation

- 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.<br>(max depth=5) | 7 %<br>(max depth=13) |
| Random Forest | 4.00 sec.<br>(# of estimators=800) | 8 %<br>(# estimators=1,000) |
| Gradient Boosting | 4.00 sec.<br>(# of estimators=100) | 11 %<br>(# estimators=150) |
| 3-layer NN | 4.00 sec.<br>(nodes in hidden layers: [250, 200, 200]) | 14 %<br>(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/