ACCURATE, LARGE MINIBATCH SGD: TRAINING IMAGENET IN 1 HOUR (ARXIV 2017)

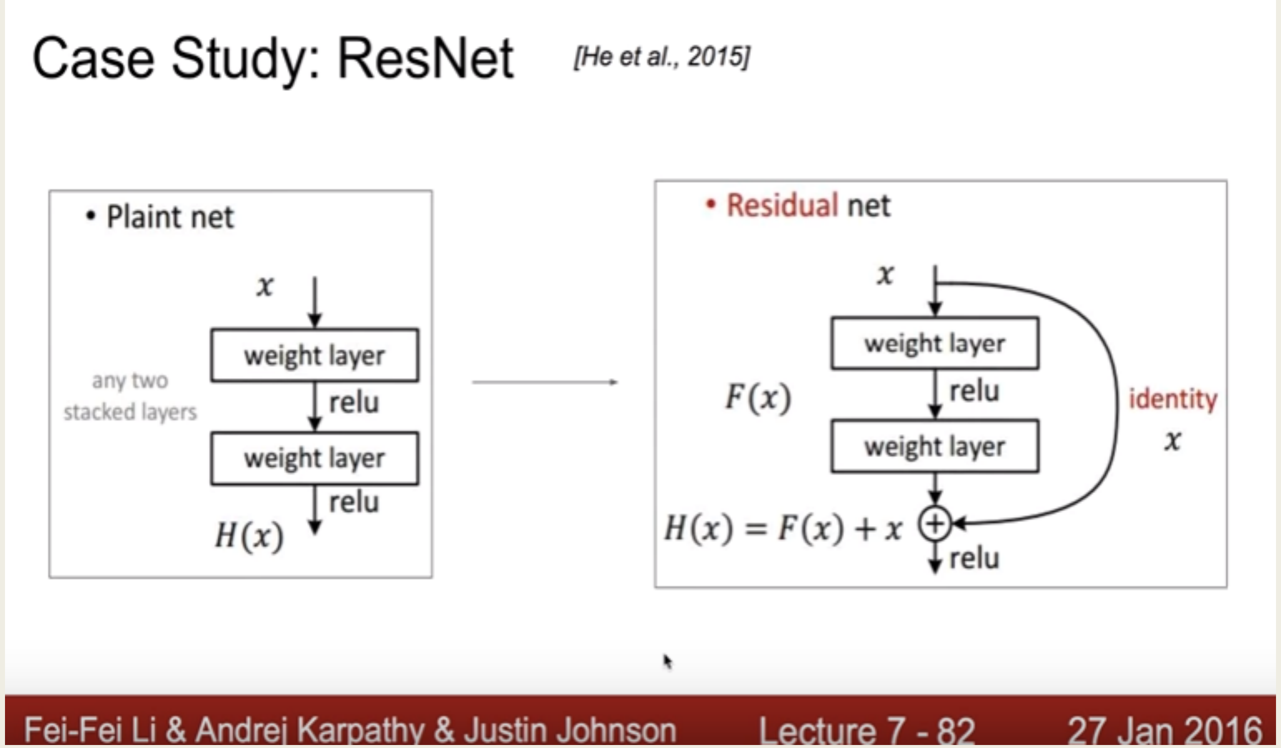
Priya Goyal

서상우
인공지능 연구실
2018103382

Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour

- Arxiv 2017.06
- Facebook
- Priya Goyal
  - Focal Loss for Dense Object Detection (*Arxiv)*
- Ross Girshick
  - Fast R-CNN
- Kaiming He
  - he initialization

Resnet-50



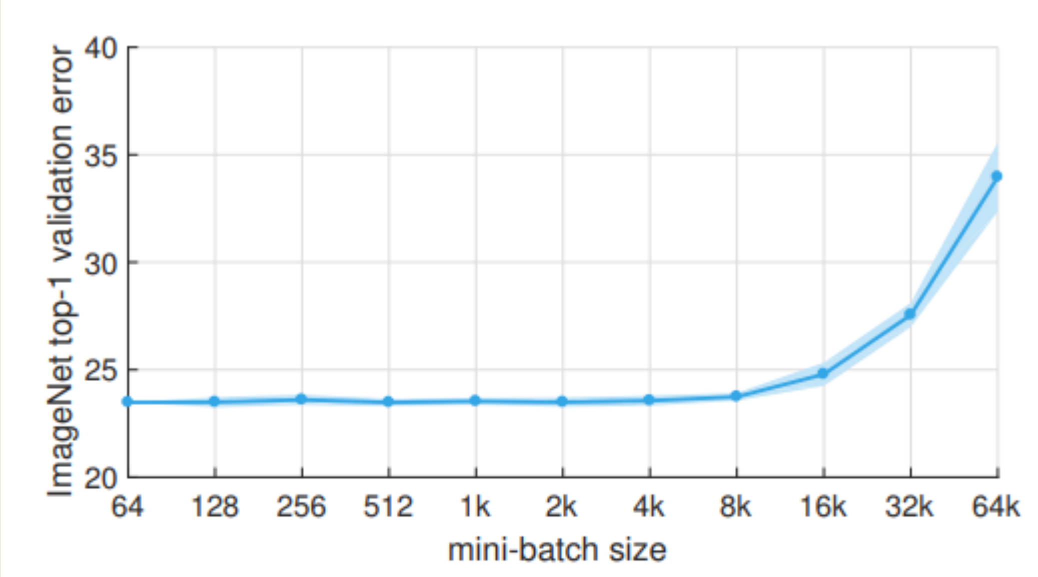


Figure 1. **ImageNet top-1 validation error *vs*. minibatch size.**

Motivation

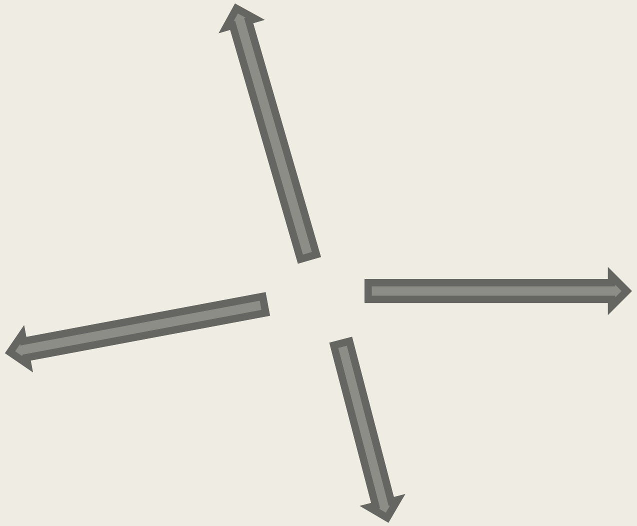
- Motivation of scaling up deep learning:
  - *데이터 셋이 크면 성능적인 향상이 생기나 느리다.*
  - *스케일 업을 통해 더 빠르게 학습 하고 싶다*
    - ResNet50 on P100/caffe2: 1GPU/10d-> 8GPUs/29h -> 256GPUs/1h
  - *Train visual models on internet-scale data*
  - *Generalize to object detection and segmentations*

Introduction

- Difficulty
  - *기존의* distributed synchronous SGD*는 결과에 대한 보장이 없다.*
  - *generalization accuracy can be maintained with minibatches as large as 8192*
  - *high-accuracy models can be trained in such short time.*

Introduction

- Difficulty:
  - *poor generalization (at the end of training)*
  - *optimization difficulty (at the beginning of training)*



1. Introduction

- Method: (for Distributed Synchronous SGD)
  - *Gradient aggregation*
  - *Learning rate linear scaling + warmup*
  - *Some tricks to overcom optimization difficulty*

Distributed Synchronous SGD

- grad on local batch i
- grad := grad aggregation
- weight := weight + f(grad)

Distributed Synchronous SGD

- grad on local batch i
- grad := grad aggregation
- weight := weight + f(grad)

$$l(x, w) = \frac{\lambda}{2}\|w\|^2 + \varepsilon(x, w)$$

$$\frac{1}{n}\sum_{x\in\mathcal{B}} \nabla l(x, w) = \lambda w + \frac{1}{n}\sum_{x\in\mathcal{B}} \nabla \mathcal{E}(x, w)$$

$$\frac{1}{kn}\sum_{j<k}\sum_{x\in\mathcal{B}_j} \nabla l(x, w_t)$$

$$\hat{w}_{t+1} = w_t - \hat{\eta}\frac{1}{kn}\sum_{j<k}\sum_{x\in\mathcal{B}_j} \nabla l(x, w_t).$$

2. Large Minibatch SGD

$$L(w) = \frac{1}{|X|} \sum_{x \in X} l(x, w)$$

$$w_{t+1} = w_t - \eta \frac{1}{n} \sum_{x \in \mathcal{B}} \nabla l(x, w_t)$$

Large Minibatch SGD

- Why are we interested in large minibatch SGD?
  - *The larger mini-batches, the higher per-worker workload, the lower the relative communication overhead (or easier to hide communication overhead) and the easier to scale up.*
  - *We want to use large mini-batches in place of small mini-batches.*
  - *However, using large mini-batches will sacrifice model accuracy in recent literature or simply won't converge.*

Learning rates for Large Minibatch

Linear Scaling Rule: *When the minibatch size is multiplied by k, multiply the learning rate by k.*

Learning rates for Large Minibatch: Interpretation

- assume:

$$\nabla l(x, w_t) \approx \nabla l(x, w_{t+j})$$

- then setting: $\hat{\eta} = k\eta$ gives $\widehat{W}_{t+k} \approx W_{t+k}$

Learning rates for Large Minibatch: Interpretation

K iterations

$$w_{t+k} = w_t - \eta \frac{1}{n} \sum_{j<k} \sum_{x \in \mathcal{B}_j} \nabla l(x, w_{t+j}).$$

Single iterations

$$\hat{w}_{t+1} = w_t - \hat{\eta} \frac{1}{kn} \sum_{j<k} \sum_{x \in \mathcal{B}_j} \nabla l(x, w_t).$$

Learning rates for Large Minibatch: condition

- Except:
  - *1. In the beginning few epochs of training, this does not hold. (optimization difficulty.)*
  - *2. Using too many workers (k is too large),this does not hold. Minibatch size cannot be scaled up indefinitely*

Learning rate Warmup

- Constant warmup

- Gradual warmup

	k	n	kn	η	top-1 error (%)
baseline (single server)	8	32	256	0.1	23.60 ±0.12
no warmup, Figure 2a	256	32	8k	3.2	24.84 ±0.37
constant warmup, Figure 2b	256	32	8k	3.2	25.88 ±0.56
gradual warmup, Figure 2c	256	32	8k	3.2	23.74 ±0.09

3. Subtleties and Pitfalls of Distributed SGD

- weight decay
- momentum correction
- data shuffling

weight decay

- Weight decay is originally a part of loss function called L2-regularization.
- After we take the gradient of the loss function, it appears as the weight decay term here.

$$w_{t+1} = w_t - \eta \frac{1}{n} \sum_{x \in \mathcal{B}} \nabla l(x, w_t)$$

$$l(x, w) = \frac{\lambda}{2} \|w\|^2 + \varepsilon(x, w)$$

$$w_{t+1} = w_t - \eta \lambda w_t - \eta \frac{1}{n} \sum_{x \in \mathcal{B}} \nabla \varepsilon(x, w_t)$$

Momentum Correction

$$u_{t+1} = mu_t + \frac{1}{n}\sum_{x\in\mathcal{B}} \nabla l(x, w_t)$$
$$w_{t+1} = w_t - \eta u_{t+1}.$$

- Substituting v_t for ηv_t in yield

$$v_{t+1} = mv_t + \eta\frac{1}{n}\sum_{x\in\mathcal{B}} \nabla l(x, w_t)$$
$$w_{t+1} = w_t - v_{t+1}.$$

Momentum Correction

$$w_{t+1} = w_t - \eta_{t+1} u_{t+1}$$

$$= w_t - \eta_{t+1}(m u_t + \frac{1}{n}\sum \nabla l(x, w_t)$$

$$= w_t - \eta_{t+1}(m \frac{v_t}{\eta_t} + \frac{1}{n}\sum \nabla l(x, w_t))$$

$$= w_t - m\frac{\eta_{t+1}}{\eta_t} v_t - \eta_{t+1}\frac{1}{n}\sum \nabla l(x, w_t)$$

Momentum Correction

$$
\begin{aligned}
w_{t+1} &= w_t - \eta_{t+1} u_{t+1} \\
&= w_t - \eta_{t+1}\left(m u_t + \frac{1}{n}\sum \nabla l(x, w_t)\right. \\
&= w_t - \eta_{t+1}\left(m \frac{v_t}{\eta_t} + \frac{1}{n}\sum \nabla l(x, w_t)\right) \\
&= w_t - m \frac{\eta_{t+1}}{\eta_t} v_t - \eta_{t+1} \frac{1}{n} \sum \nabla l(x, w_t)
\end{aligned}
$$

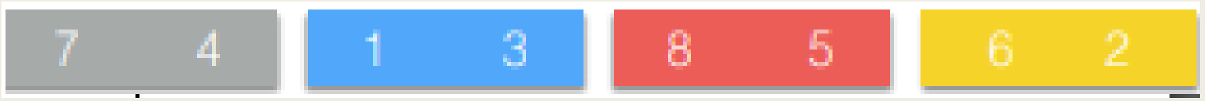
So the correct v_{t+1} should be

$$
v_{t+1} = m \frac{\eta_{t+1}}{\eta_t} v_t + \eta_{t+1} \frac{1}{n} \sum \nabla l(x, w_t)
$$

Data shuffling

- Single-worker data shuffling:

- 4-worker data shuffling:



O

X

Communication

- gradient aggregation
- software
- hardware

Gradient Aggregation

- within a server:
  - *if data>256kb, use NCCL*
  - *else, GPU->host + reduction*
- between servers:
  - *recursive halving and doubling algorithm*
- Non-power-of-two servers:
  - *binary blocks algorithm*

Gradient Aggregation : allreduce

- (1) buffers from the 8 GPUs within a server are summed into a single buffer for each server
- (2) the results buffers are shared and summed across all servers
- (3) the results are broadcast onto each GPU.

Gradient Aggregation : interserver allreduce

- For interserver allreduce, we implemented two of the best algorithms for bandwidth-limited scenarios

- the recursive halving and doubling algorithm and the bucket algorithm (also known as the ring algorithm).
  - *For both, each server sends and receives $2 *(p-1)/p*b$ bytes of data, where b is the buffer size in bytes and p is the number of servers.*
  - *b is the buffer size in bytes and p is the number of servers.*

- This generally makes the halving/doubling algorithm faster in latency-limited scenarios

Gradient Aggregation : interserver allreduce

- The halving/doubling algorithm consists of a reduce scatter collective followed by an allgather.
  - *In the first step of reduce-scatter, servers communicate in pairs sending and receiving for different halves of their input buffers (rank 0 with 1, 2 with 3, etc.)*

Software

- Gloo
  - *intra-node wraps NCCL operations*
  - *inter-node self-implemented*
  - *can use GPU-Direct and Infiniband API*
- Caffe2
  - *multi-threaded execution of subgraphs*
  - *parallel communication with training*

Hardware

- Big Basin
  - *Intel-based*
  - *8 P100 GPUs with NVLink*
  - *3.2T NVMe SSDs*
  - *Mellanox 50G Ethernet*

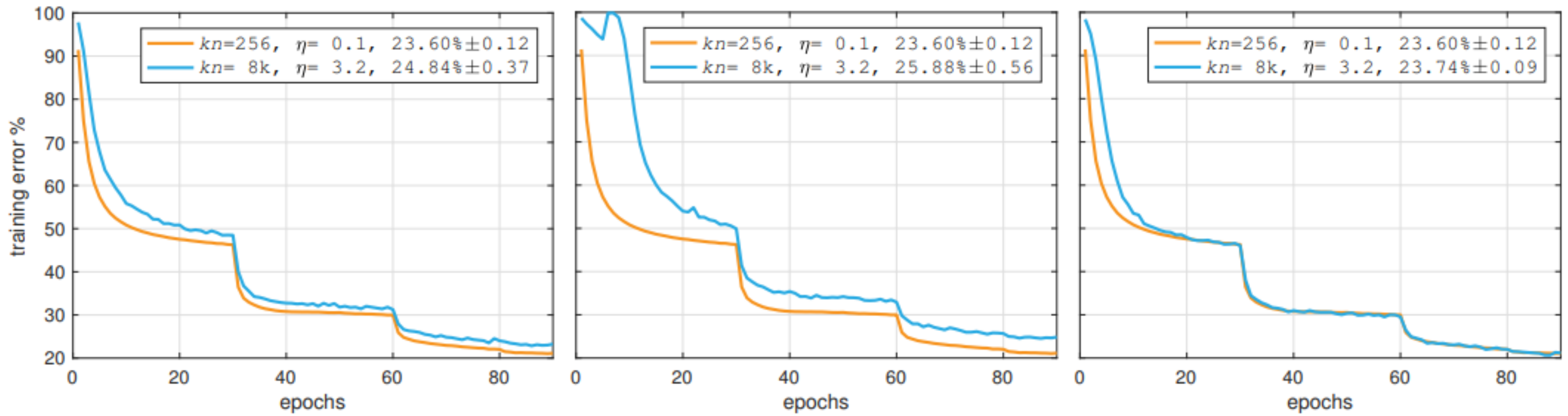
5. Results and Analysis

- Experimental Settings
- Optimization or Generalization Issues
- Analysis Experiments

Experimental Settings

- ResNet50 train on ImageNet-1k (1.28 million images)
- Momentum SGD, batch size n=32
- learning rate (linear scaling rule)
- Baseline: k=8GPU, n=32, top-1 validation error=23.6%
- k ranges from 8 to 256 (1 to 32 Big Basins)
- model's error rate as the median error of the final 5 epochs
- report the mean and standard deviation (std) of the error from 5 independent runs

Optimization or Generalization Issues



(a) no warmup (b) constant warmup (c) gradual warmup

Analysis Experiments

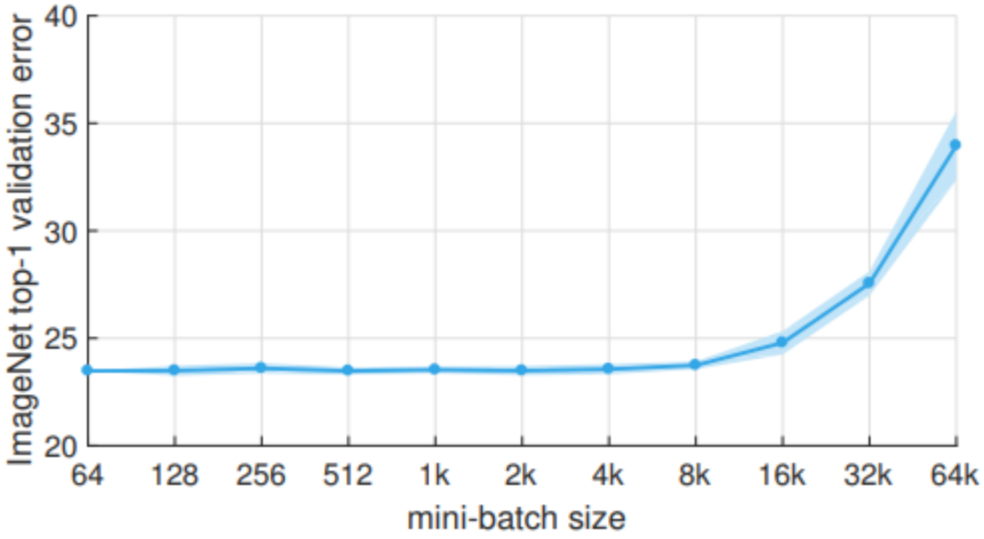


Figure 1. **ImageNet top-1 validation error *vs*. minibatch size.**

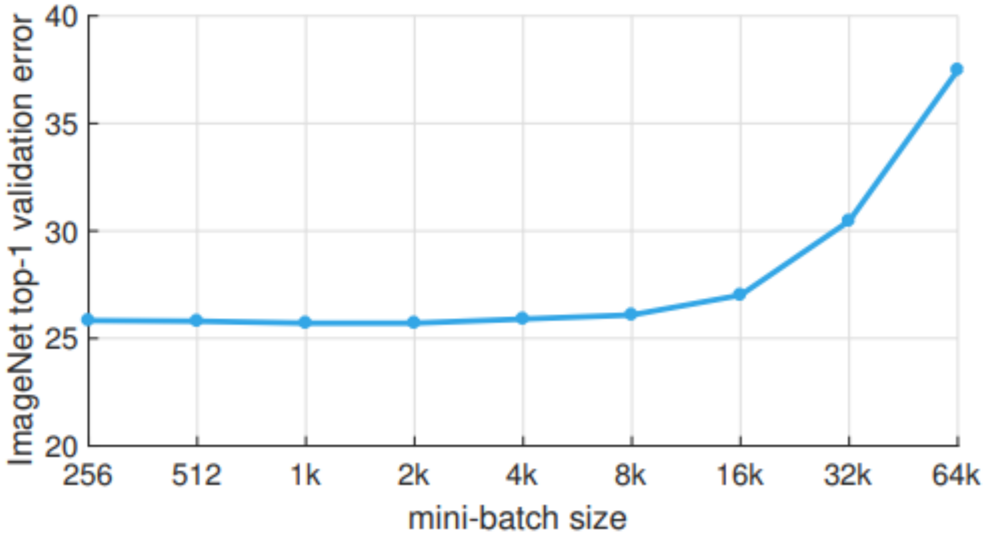
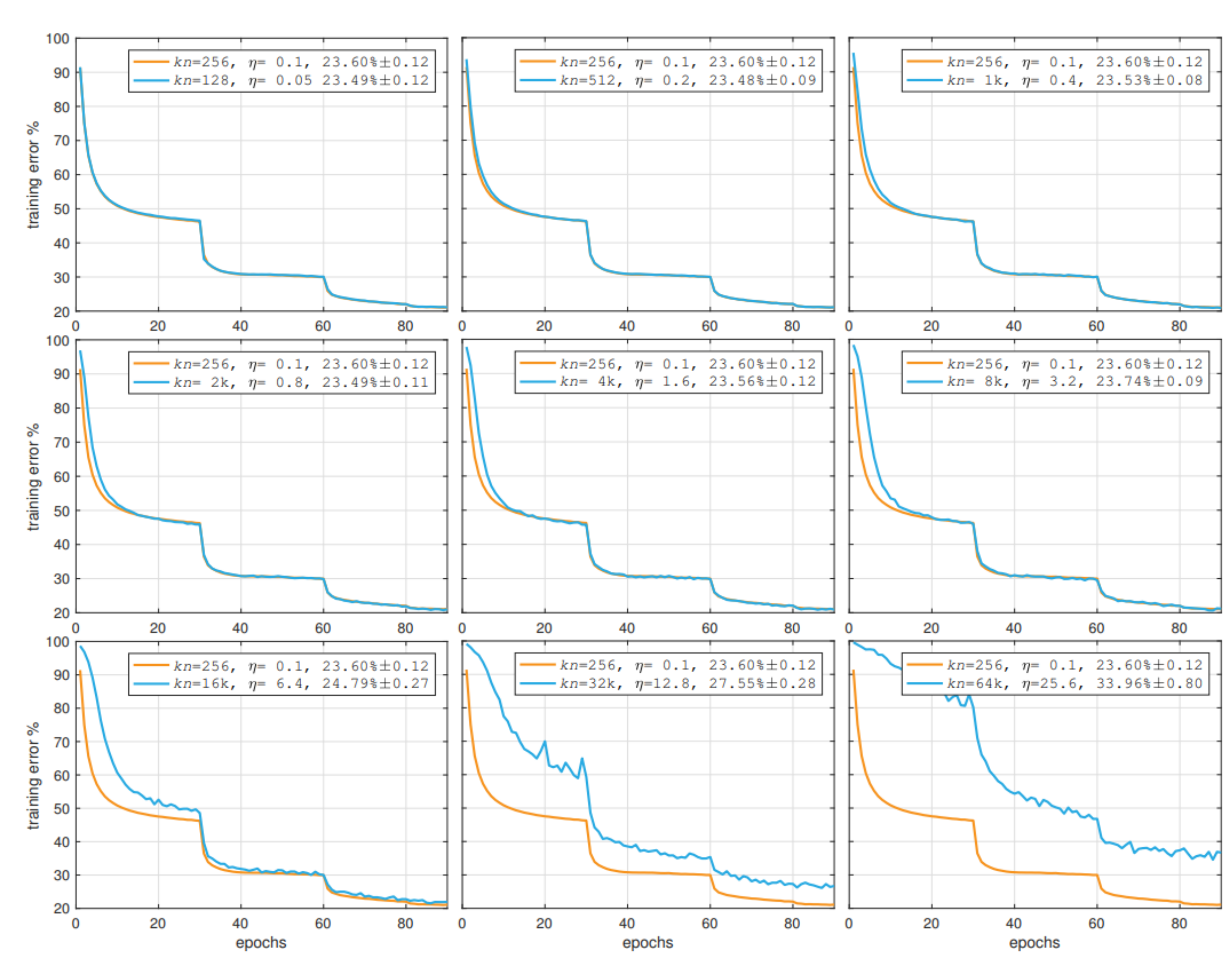


Figure 6. **ImageNet-5k** top-1 validation error *vs*. minibatch size



training error %
epochs
kn=256, η= 0.1, 23.60%±0.12
kn=128, η= 0.05 23.49%±0.12
kn=512, η= 0.2, 23.48%±0.09
kn= 1k, η= 0.4, 23.53%±0.08
kn= 2k, η= 0.8, 23.49%±0.11
kn= 4k, η= 1.6, 23.56%±0.12
kn= 8k, η= 3.2, 23.74%±0.09
kn=16k, η= 6.4, 24.79%±0.27
kn=32k, η=12.8, 27.55%±0.28
kn=64k, η=25.6, 33.96%±0.80

Analysis Experiments

- 32 Big Basin servers
  - *32*8 GPUs*
  - *32*8*32 = 8192 batch size*
- To simulate higher batch sizes: 16k, 32k, 64k

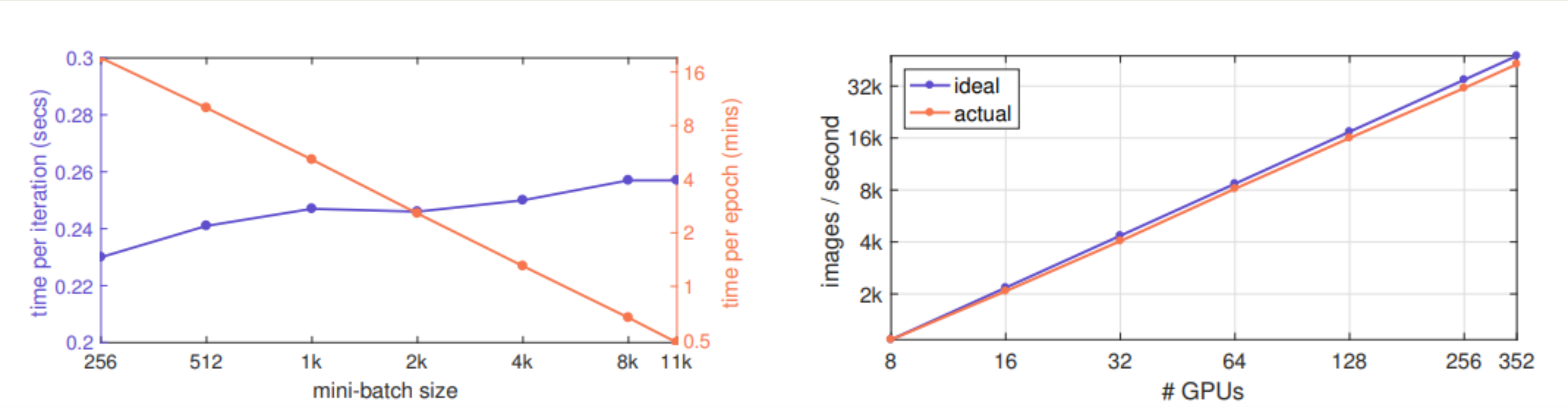
Generalization to Detection and Segmentation

ImageNet pre-training			COCO	
kn	η	top-1 error (%)	box AP (%)	mask AP (%)
256	0.1	23.60 ±0.12	35.9 ±0.1	33.9 ±0.1
512	0.2	23.48 ±0.09	35.8 ±0.1	33.8 ±0.2
1k	0.4	23.53 ±0.08	35.9 ±0.2	33.9 ±0.2
2k	0.8	23.49 ±0.11	35.9 ±0.1	33.9 ±0.1
4k	1.6	23.56 ±0.12	35.8 ±0.1	33.8 ±0.1
8k	3.2	23.74 ±0.09	35.8 ±0.1	33.9 ±0.2
16k	6.4	24.79 ±0.27	35.1 ±0.3	33.2 ±0.3

(a) **Transfer learning of large minibatch pre-training to Mask R-CNN.**

# GPUs	kn	$\eta \cdot 1000$	iterations	box AP (%)	mask AP (%)
1	2	2.5	1,280,000	35.7	33.6
2	4	5.0	640,000	35.7	33.7
4	8	10.0	320,000	35.7	33.5
8	16	20.0	160,000	35.6	33.6

Run Time



time per iteration (secs)
0.3
0.28
0.26
0.24
0.22
0.2
16
8
4
2
1
0.5
time per epoch (mins)
256
512
1k
2k
4k
8k
11k
mini-batch size
images / second
32k
16k
8k
4k
2k
ideal
actual
8
16
32
64
128
256
352
GPUs

Summary

- Deep learning can be scaled up and accelerated accurately to some limits.
- Efforts needed on both convergence and speed.