

Lecture 11: Training Neural Networks (Part 2)

Reminder: A3

Due this Friday 10/9 at 11:59pm EST

Reminder: 100% on autograder does not mean 100% score! Some components of each assignment will be hand-graded.

Midterm

- Monday, October 19
- Will be online via <https://crabster.org/>
- Exam is 90 minutes
- You can take it any time in a 24-hour window
- We will have 3-4 “on-call” periods during the 24-hour window where GSIs will answer questions within ~15 minutes
- Open note
- True / False, multiple choice, short answer
- For short answer questions requiring math, either write LaTeX or upload an image with handwritten math

Overview

1. One time setup

Activation functions, data preprocessing, weight initialization, regularization

Last Time

2. Training dynamics

Learning rate schedules;
hyperparameter optimization

Today

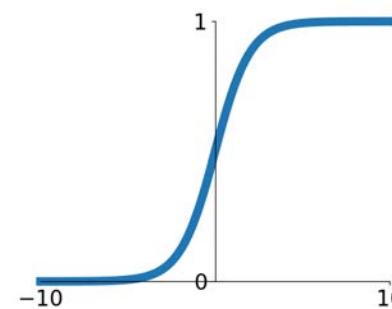
3. After training

Model ensembles, transfer learning,
large-batch training

Last Time: Activation Functions

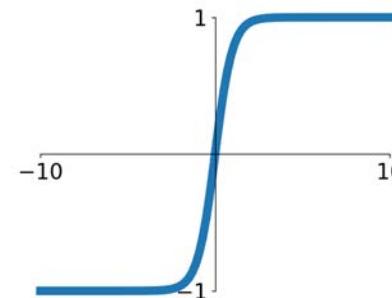
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



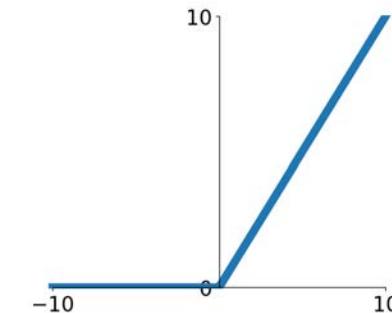
tanh

$$\tanh(x)$$



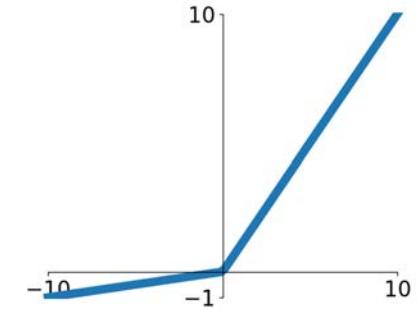
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

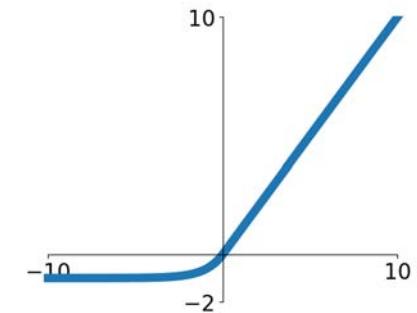


Maxout

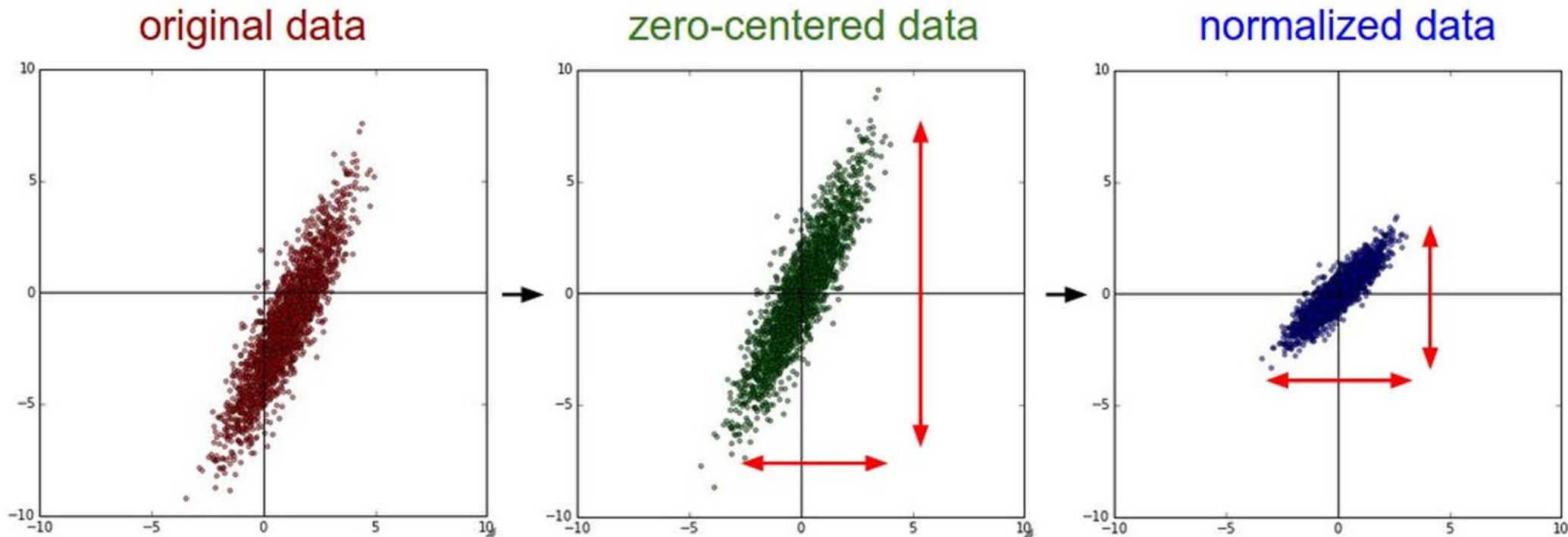
$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



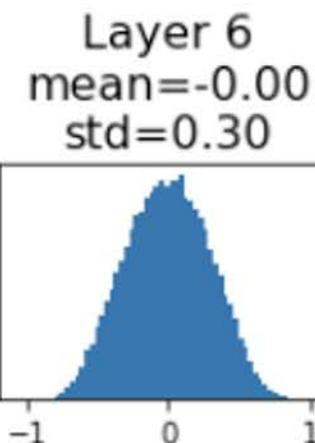
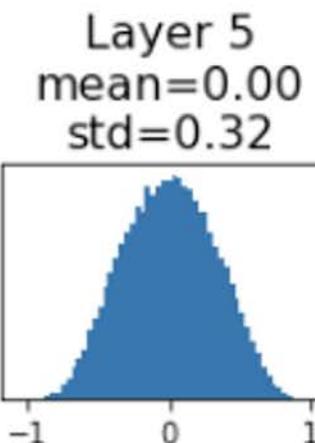
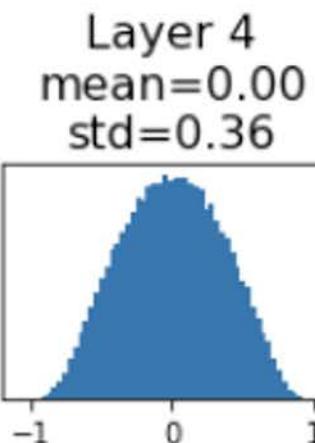
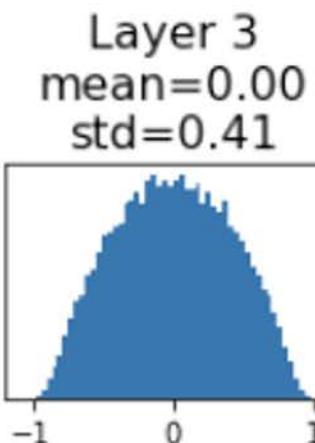
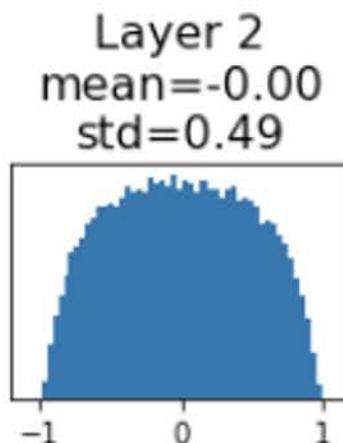
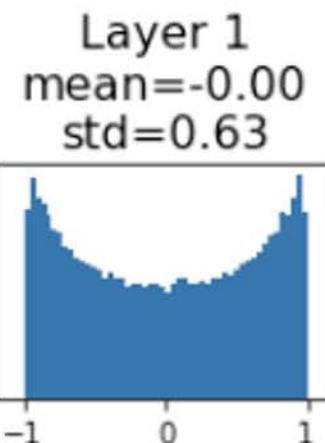
Last Time: Data Preprocessing



Last Time: Weight Initialization

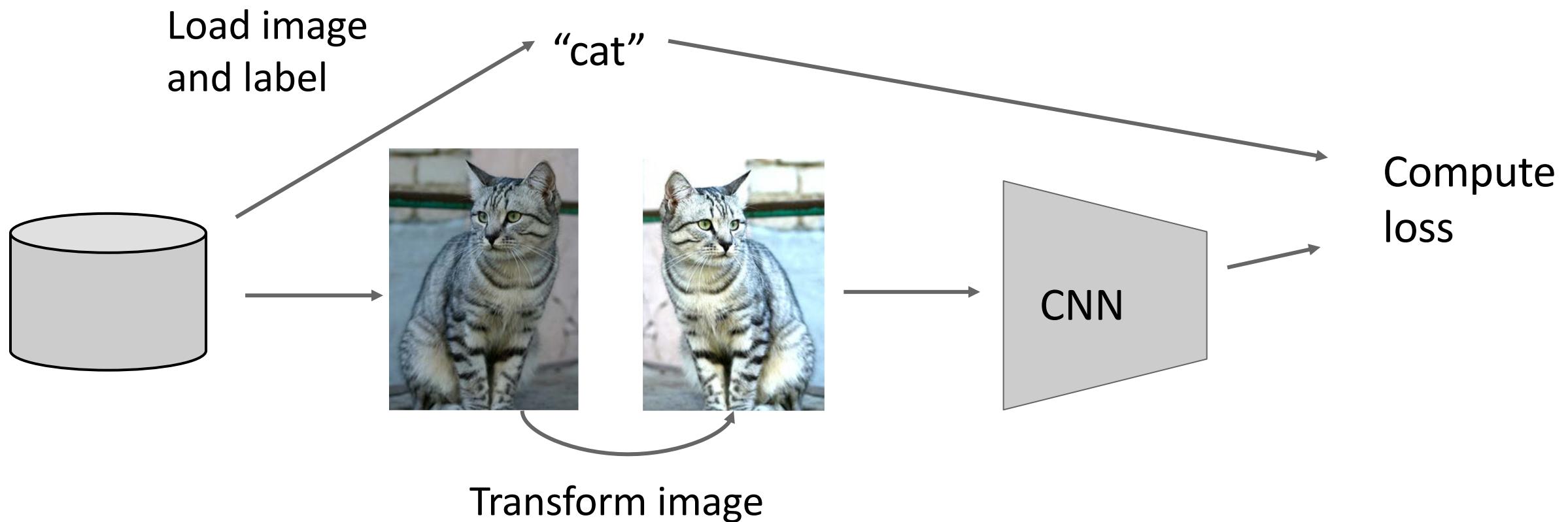
```
dims = [4096] * 7          "Xavier" initialization:  
hs = []                      std = 1/sqrt(Din)  
x = np.random.randn(16, dims[0])  
for Din, Dout in zip(dims[:-1], dims[1:]):  
    W = np.random.randn(Din, Dout) / np.sqrt(Din)  
    x = np.tanh(x.dot(W))  
    hs.append(x)
```

"Just right": Activations are nicely scaled for all layers!



Glorot and Bengio, "Understanding the difficulty of training deep feedforward neural networks", AISTAT 2010

Last Time: Data Augmentation



Last Time: Regularization

Training: Add randomness

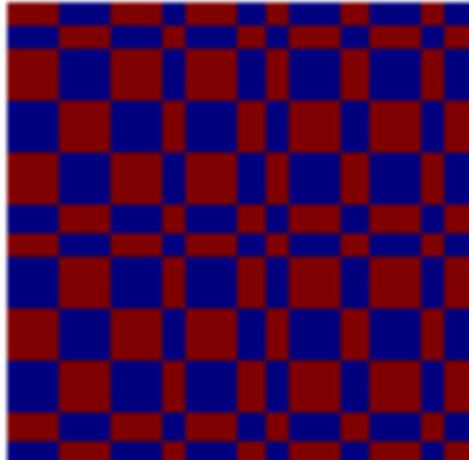
Testing: Marginalize out randomness

Examples:

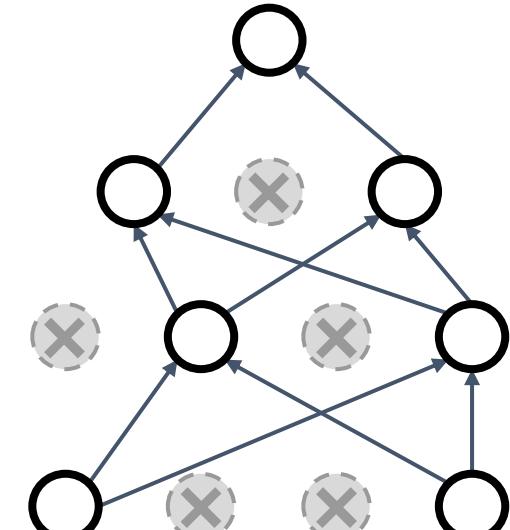
Batch Normalization

Data Augmentation

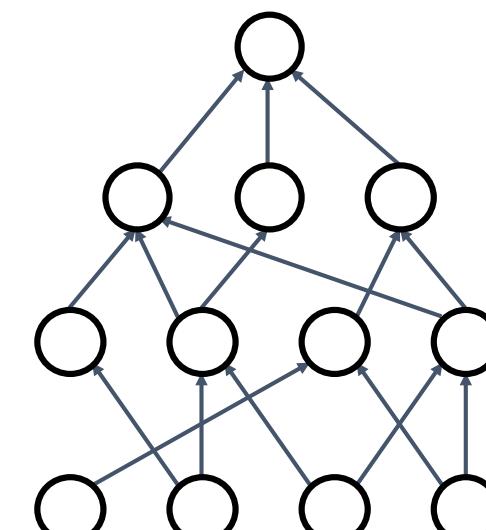
Fractional pooling



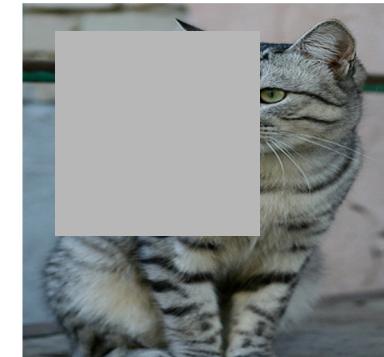
Dropout



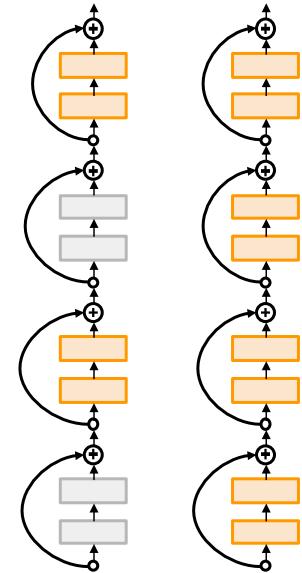
DropConnect



Cutout



Stochastic Depth



Mixup



Overview

1. One time setup

Activation functions, data preprocessing, weight initialization, regularization

2. Training dynamics

Learning rate schedules;
hyperparameter optimization

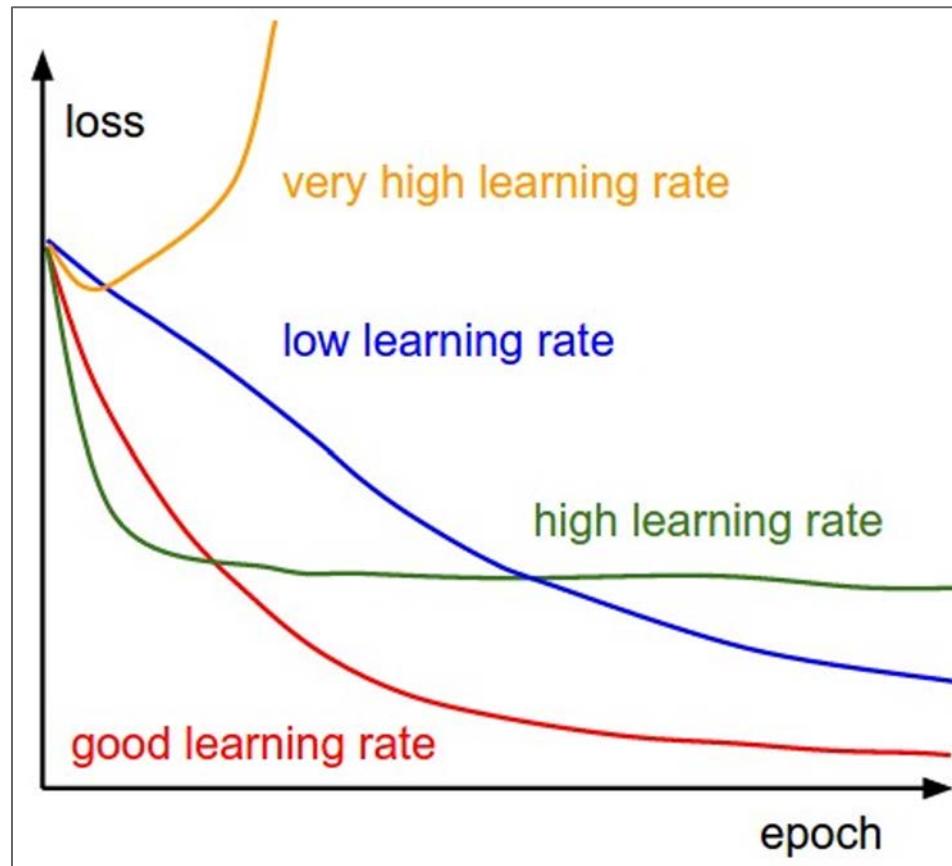
Today

3. After training

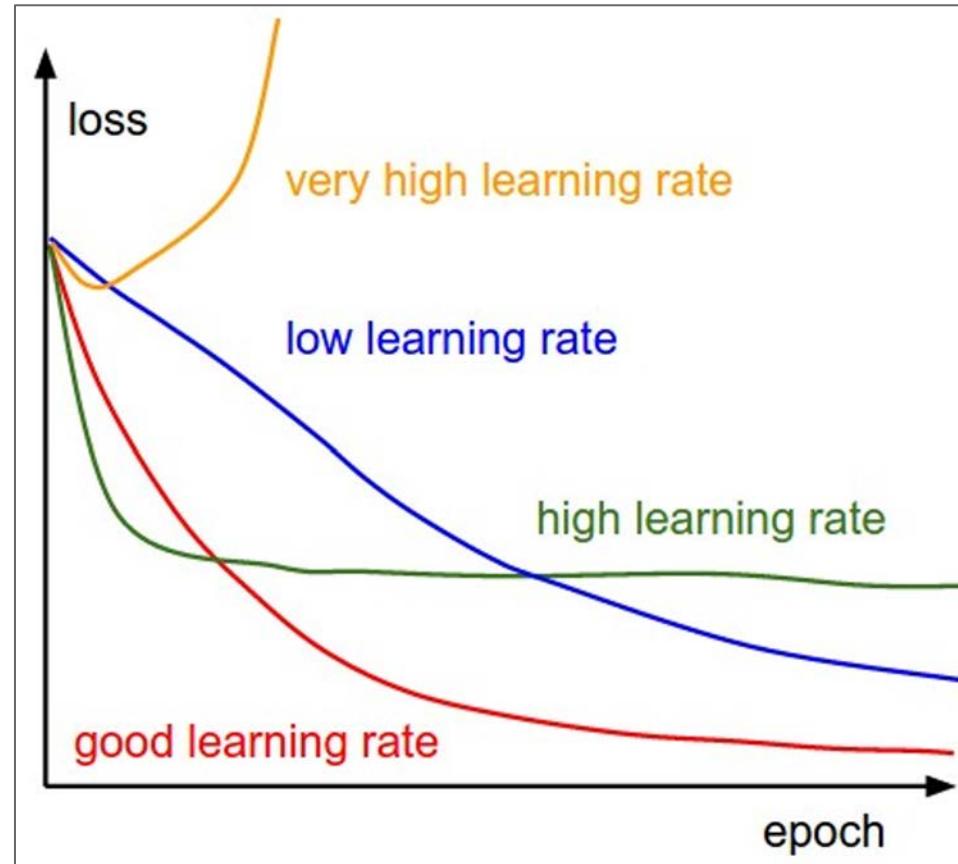
Model ensembles, transfer learning,
large-batch training

Learning Rate Schedules

SGD, SGD+Momentum, Adagrad, RMSProp, Adam
all have **learning rate** as a hyperparameter.

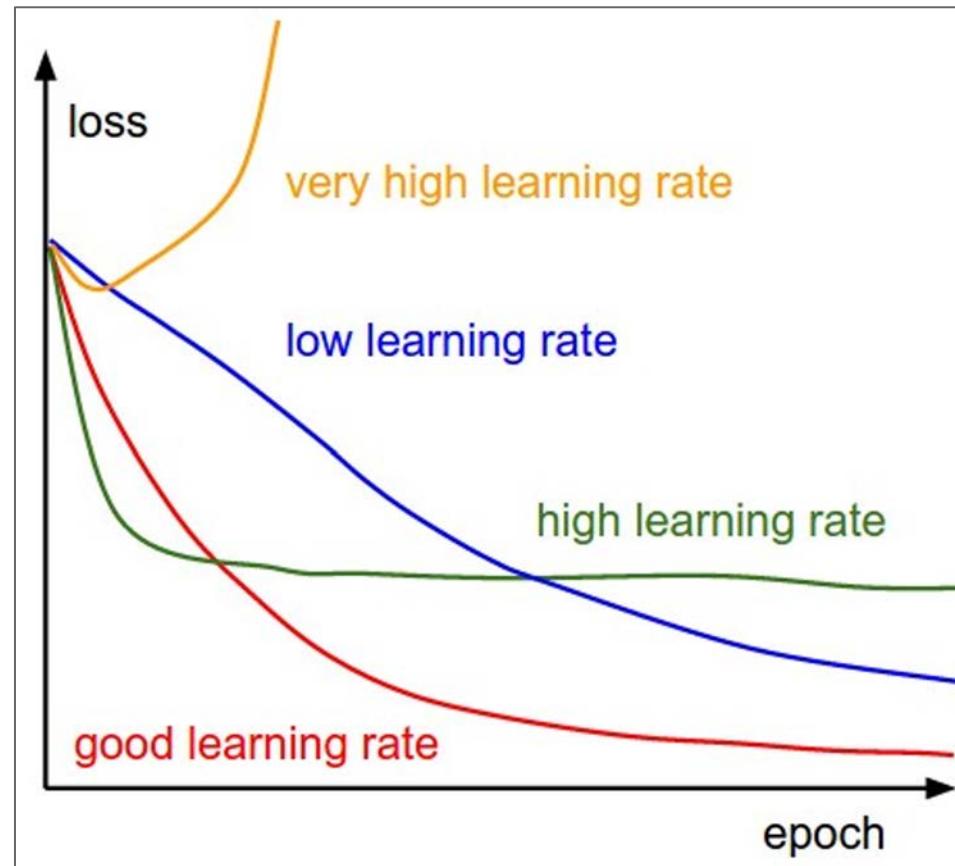


SGD, SGD+Momentum, Adagrad, RMSProp, Adam
all have **learning rate** as a hyperparameter.



Q: Which one of these learning rates is best to use?

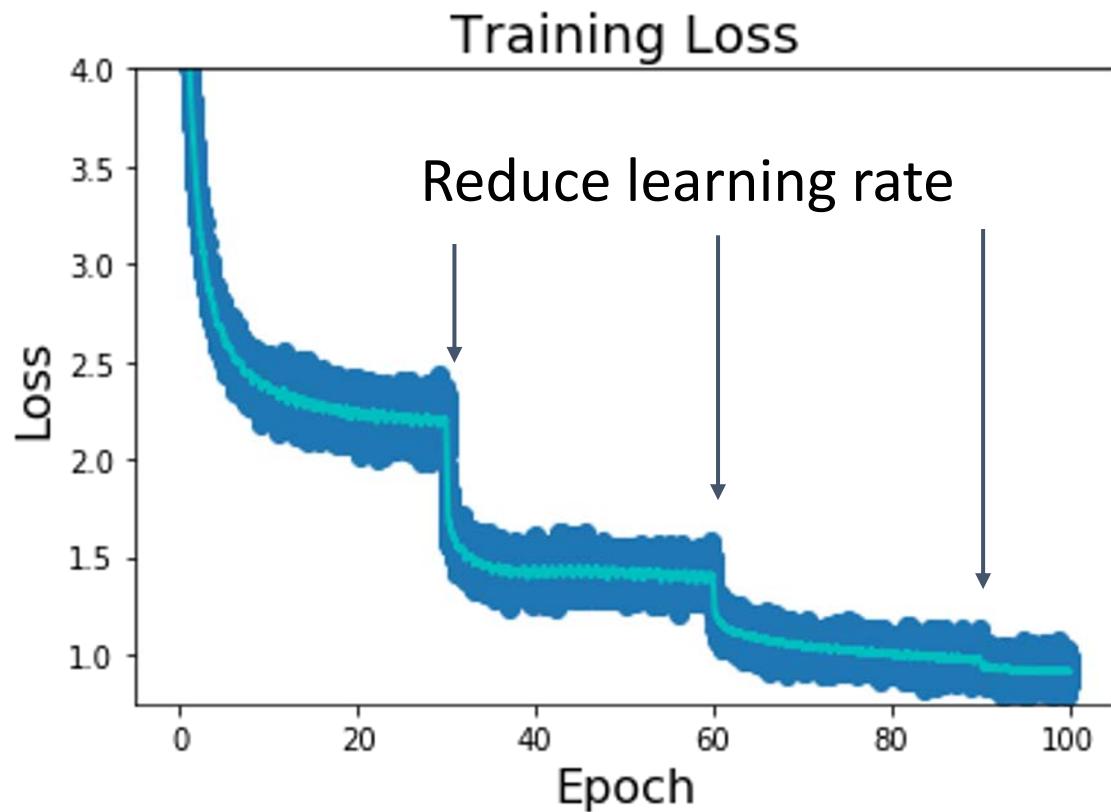
SGD, SGD+Momentum, Adagrad, RMSProp, Adam
all have **learning rate** as a hyperparameter.



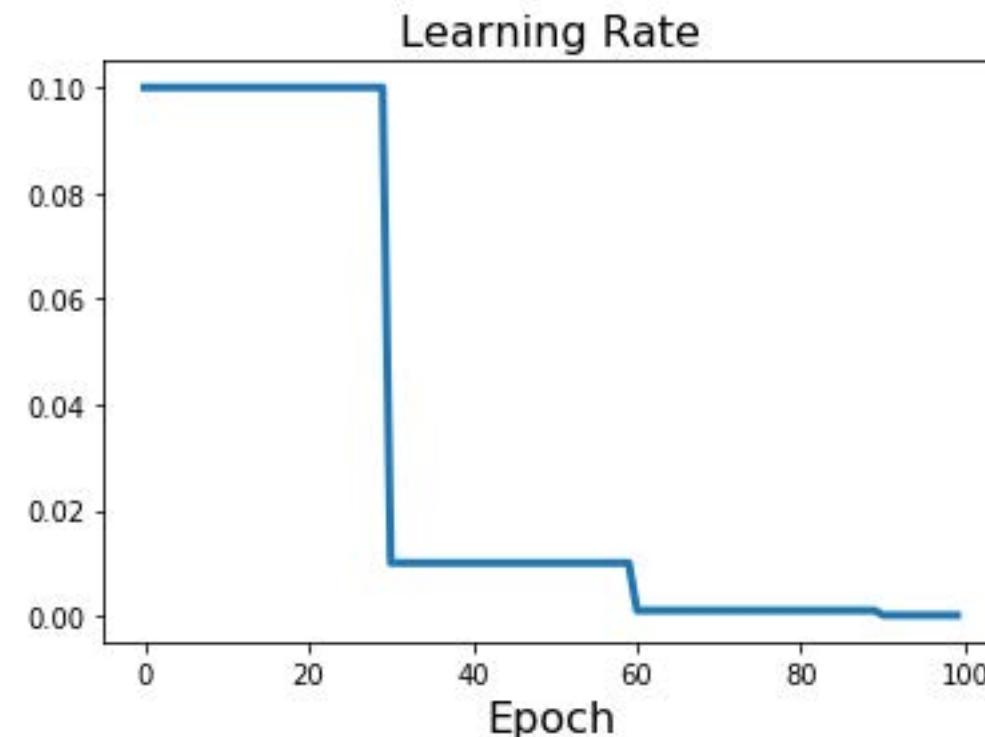
Q: Which one of these learning rates is best to use?

A: All of them! Start with large learning rate and decay over time

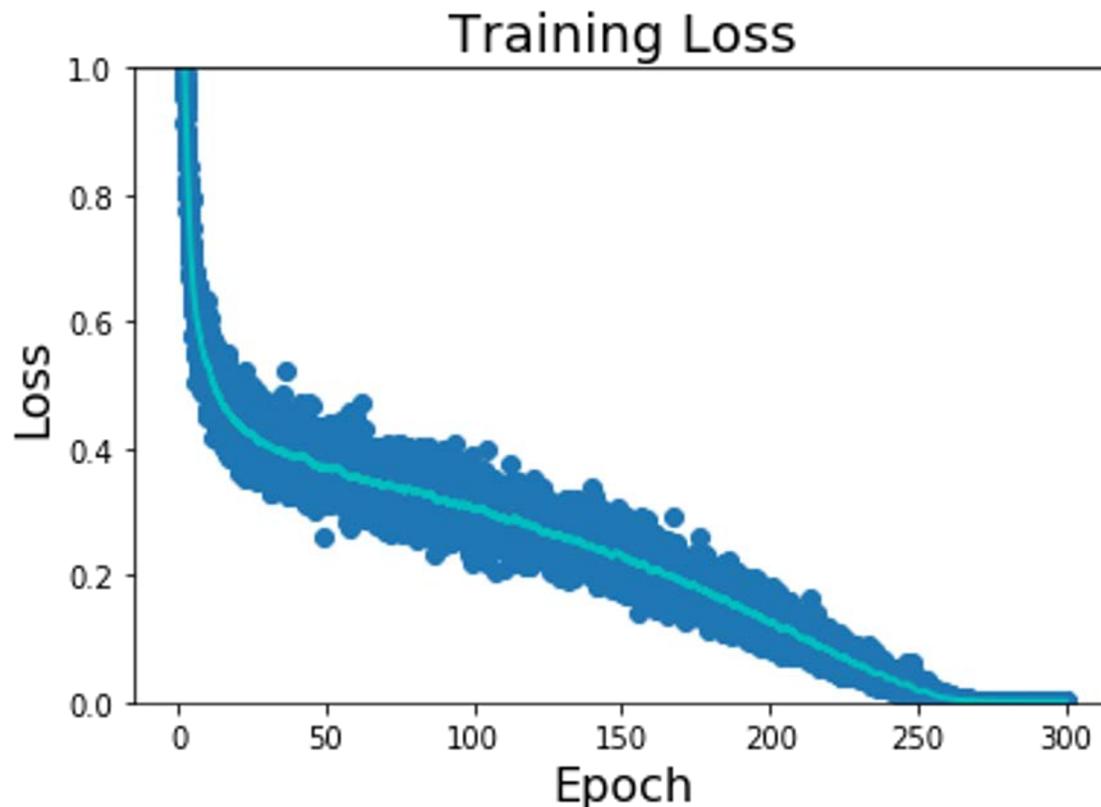
Learning Rate Decay: Step



Step: Reduce learning rate at a few fixed points.
E.g. for ResNets, multiply LR by 0.1 after epochs 30, 60, and 90.



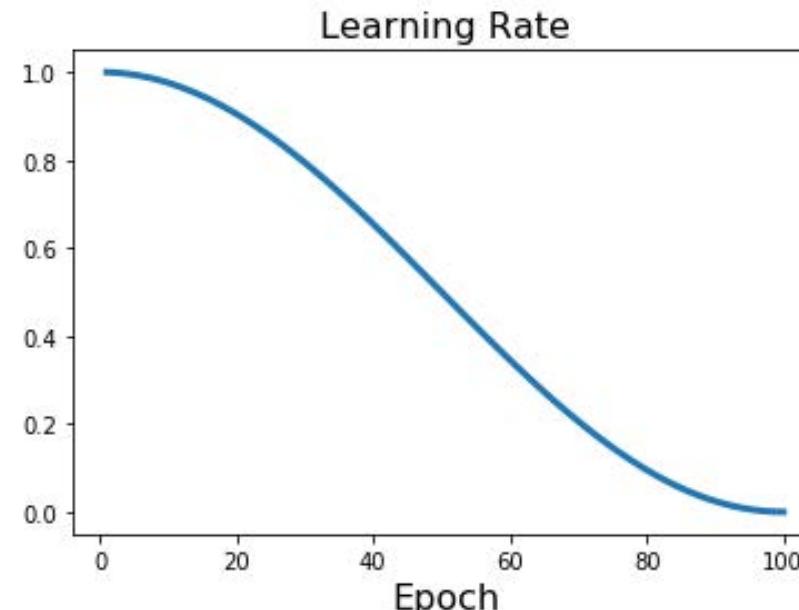
Learning Rate Decay: Cosine



Step: Reduce learning rate at a few fixed points.
E.g. for ResNets, multiply LR by 0.1 after epochs
30, 60, and 90.

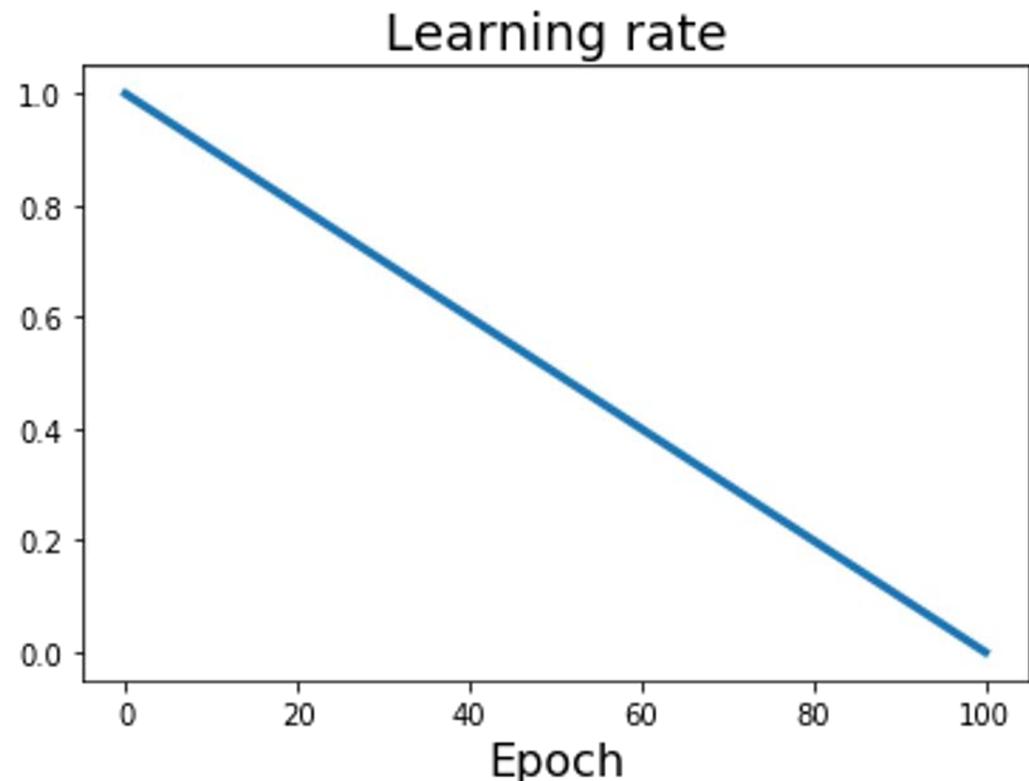
Cosine:

$$\alpha_t = \frac{1}{2} \alpha_0 \left(1 + \cos \left(\frac{t\pi}{T} \right) \right)$$



- Loshchilov and Hutter, "SGDR: Stochastic Gradient Descent with Warm Restarts", ICLR 2017
- Radford et al, "Improving Language Understanding by Generative Pre-Training", 2018
- Feichtenhofer et al, "SlowFast Networks for Video Recognition", ICCV 2019
- Radosavovic et al, "On Network Design Spaces for Visual Recognition", ICCV 2019
- Child et al, "Generating Long Sequences with Sparse Transformers", arXiv 2019

Learning Rate Decay: Linear



Step: Reduce learning rate at a few fixed points.
E.g. for ResNets, multiply LR by 0.1 after epochs 30, 60, and 90.

Cosine: $\alpha_t = \frac{1}{2} \alpha_0 \left(1 + \cos \left(\frac{t\pi}{T} \right) \right)$

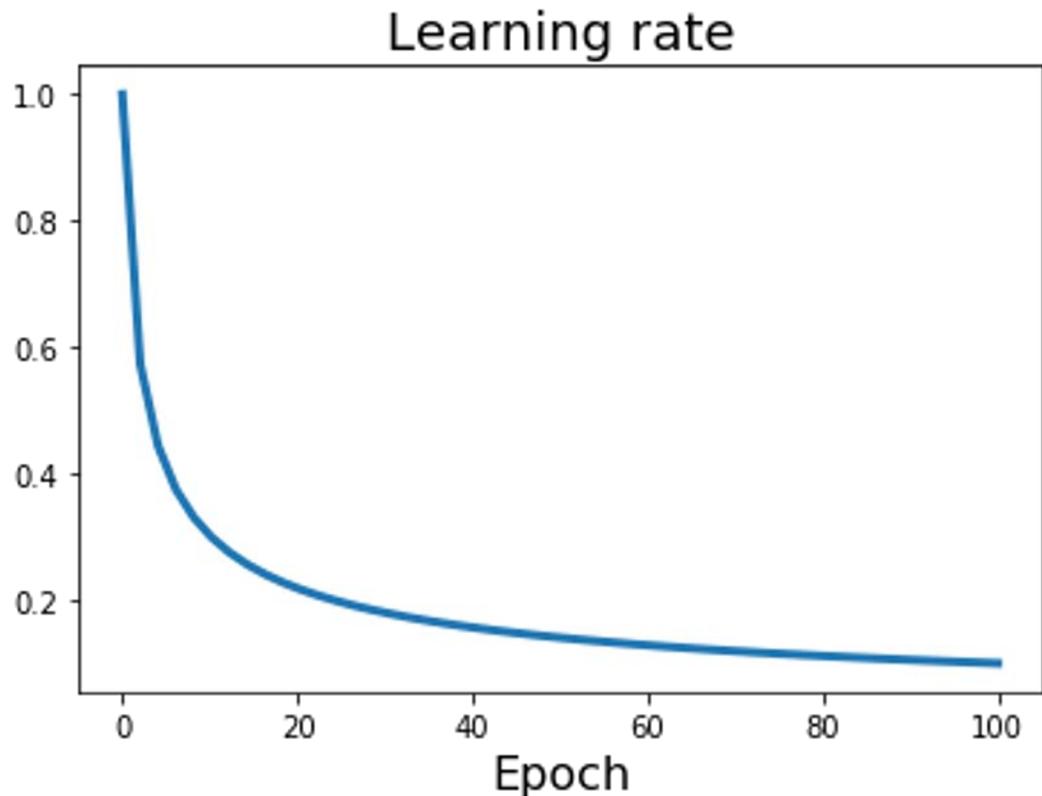
Linear: $\alpha_t = \alpha_0 \left(1 - \frac{t}{T} \right)$

Devlin et al, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", NAACL 2018

Liu et al, "RoBERTa: A Robustly Optimized BERT Pretraining Approach", 2019

Yang et al, "XLNet: Generalized Autoregressive Pretraining for Language Understanding", NeurIPS 2019

Learning Rate Decay: Inverse Sqrt



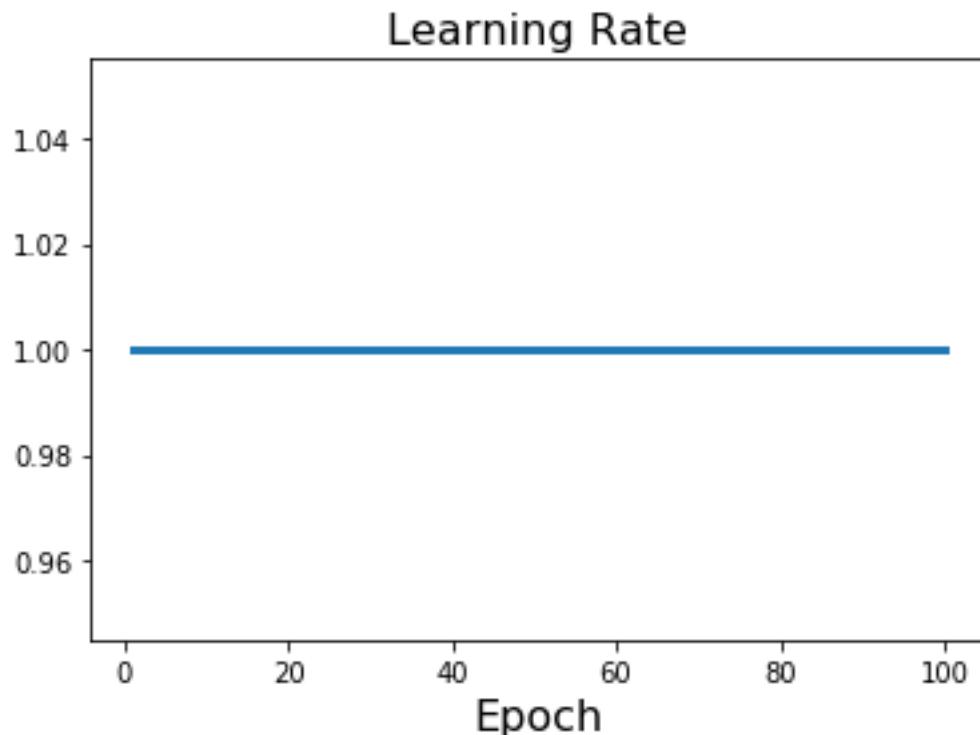
Step: Reduce learning rate at a few fixed points.
E.g. for ResNets, multiply LR by 0.1 after epochs 30, 60, and 90.

Cosine: $\alpha_t = \frac{1}{2} \alpha_0 \left(1 + \cos \left(\frac{t\pi}{T} \right) \right)$

Linear: $\alpha_t = \alpha_0 \left(1 - \frac{t}{T} \right)$

Inverse sqrt: $\alpha_t = \alpha_0 / \sqrt{t}$

Learning Rate Decay: Constant!



Step: Reduce learning rate at a few fixed points.
E.g. for ResNets, multiply LR by 0.1 after epochs 30, 60, and 90.

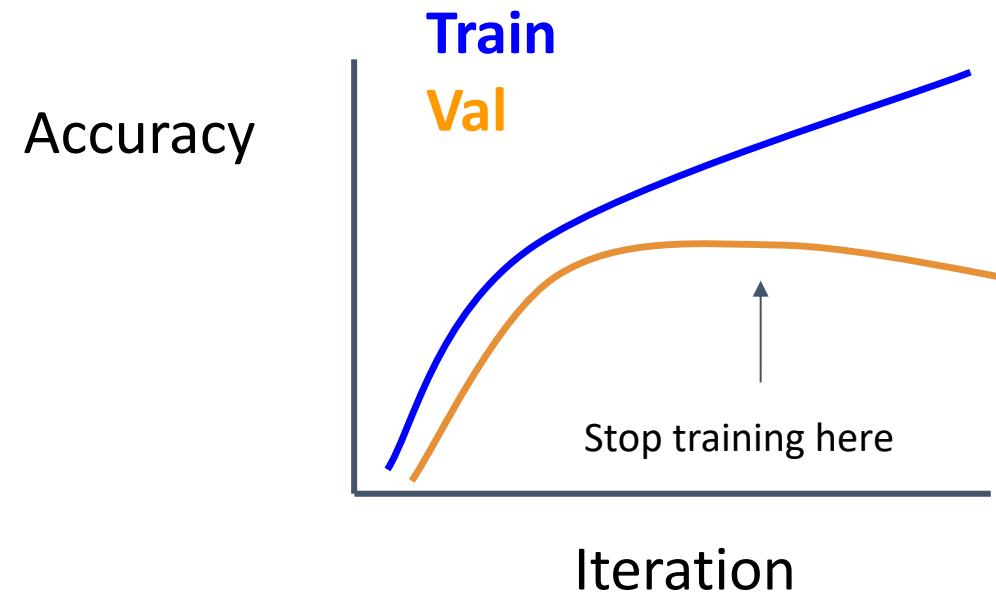
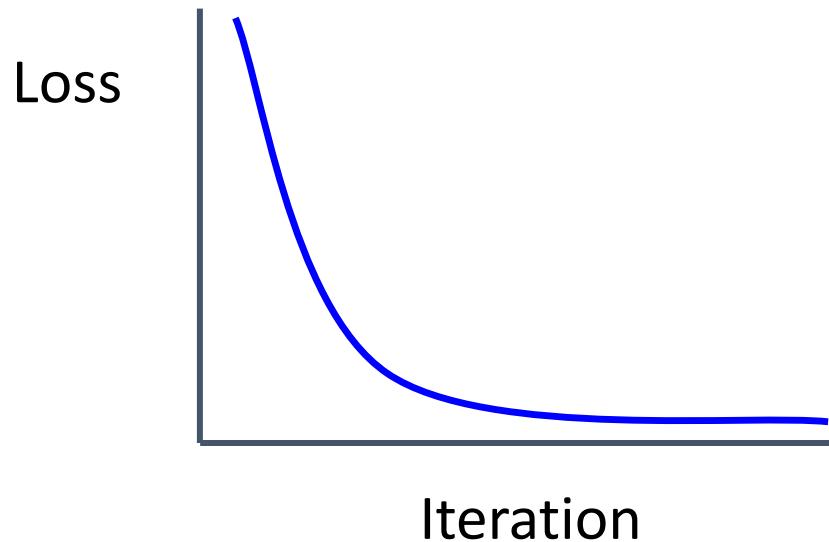
Cosine: $\alpha_t = \frac{1}{2} \alpha_0 \left(1 + \cos \left(\frac{t\pi}{T} \right) \right)$

Linear: $\alpha_t = \alpha_0 \left(1 - \frac{t}{T} \right)$

Inverse sqrt: $\alpha_t = \alpha_0 / \sqrt{t}$

Constant: $\alpha_t = \alpha_0$

How long to train? Early Stopping



Stop training the model when accuracy on the validation set decreases
Or train for a long time, but always keep track of the model snapshot that worked best on val. **Always a good idea to do this!**

Choosing Hyperparameters

Choosing Hyperparameters: Grid Search

Choose several values for each hyperparameter
(Often space choices log-linearly)

Example:

Weight decay: $[1 \times 10^{-4}, 1 \times 10^{-3}, 1 \times 10^{-2}, 1 \times 10^{-1}]$

Learning rate: $[1 \times 10^{-4}, 1 \times 10^{-3}, 1 \times 10^{-2}, 1 \times 10^{-1}]$

Evaluate all possible choices on this
hyperparameter grid

Choosing Hyperparameters: Random Search

Choose several values for each hyperparameter
(Often space choices log-linearly)

Example:

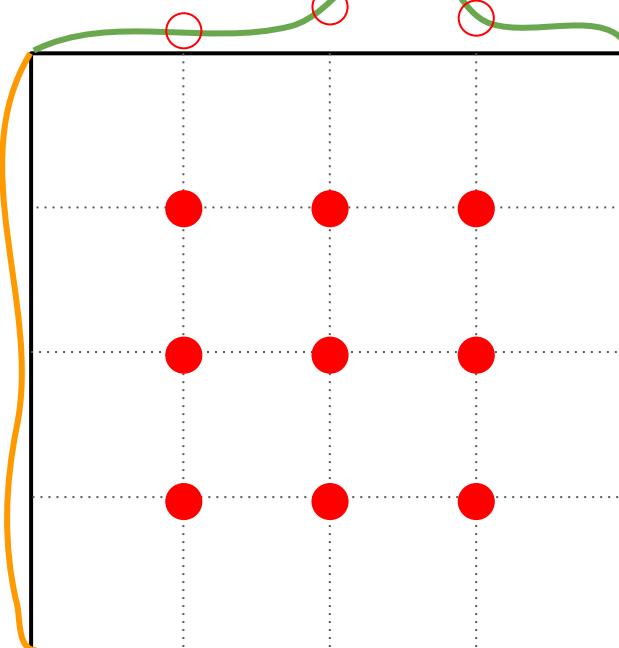
Weight decay: log-uniform on $[1 \times 10^{-4}, 1 \times 10^{-1}]$

Learning rate: log-uniform on $[1 \times 10^{-4}, 1 \times 10^{-1}]$

Run many different trials

Hyperparameters: Random vs Grid Search

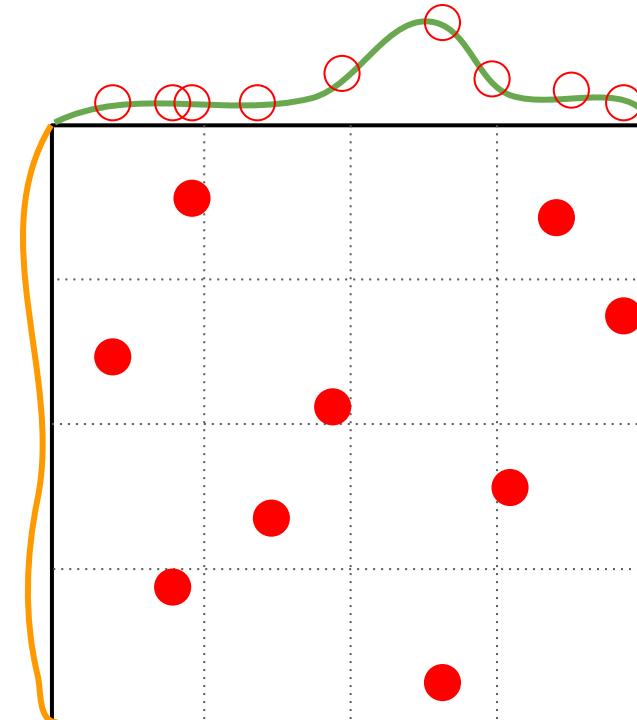
Grid Layout



Important
Parameter

Unimportant
Parameter

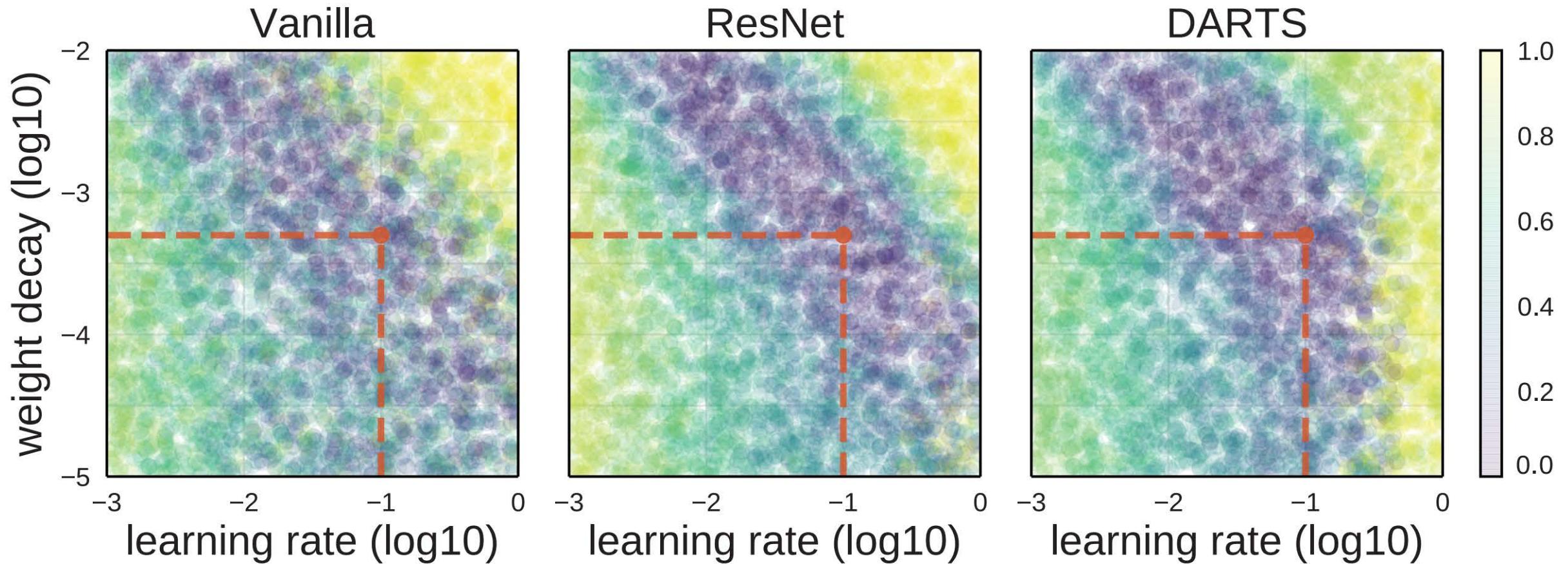
Random Layout



Important
Parameter

Unimportant
Parameter

Choosing Hyperparameters: Random Search



Choosing Hyperparameters

(without tons of GPUs)

Choosing Hyperparameters

Step 1: Check initial loss

Turn off weight decay, sanity check loss at initialization

e.g. $\log(C)$ for softmax with C classes

Choosing Hyperparameters

Step 1: Check initial loss

Step 2: Overfit a small sample

Try to train to 100% training accuracy on a small sample of training data (~5-10 minibatches); fiddle with architecture, learning rate, weight initialization. Turn off regularization.

Loss not going down? LR too low, bad initialization

Loss explodes to Inf or NaN? LR too high, bad initialization

Choosing Hyperparameters

Step 1: Check initial loss

Step 2: Overfit a small sample

Step 3: Find LR that makes loss go down

Use the architecture from the previous step, use all training data, turn on small weight decay, find a learning rate that makes the loss drop significantly within ~100 iterations

Good learning rates to try: 1e-1, 1e-2, 1e-3, 1e-4

Choosing Hyperparameters

Step 1: Check initial loss

Step 2: Overfit a small sample

Step 3: Find LR that makes loss go down

Step 4: Coarse grid, train for ~1-5 epochs

Choose a few values of learning rate and weight decay around what worked from Step 3, train a few models for ~1-5 epochs.

Good weight decay to try: 1e-4, 1e-5, 0

Choosing Hyperparameters

Step 1: Check initial loss

Step 2: Overfit a small sample

Step 3: Find LR that makes loss go down

Step 4: Coarse grid, train for ~1-5 epochs

Step 5: Refine grid, train longer

Pick best models from Step 4, train them for longer (~10-20 epochs) without learning rate decay

Choosing Hyperparameters

Step 1: Check initial loss

Step 2: Overfit a small sample

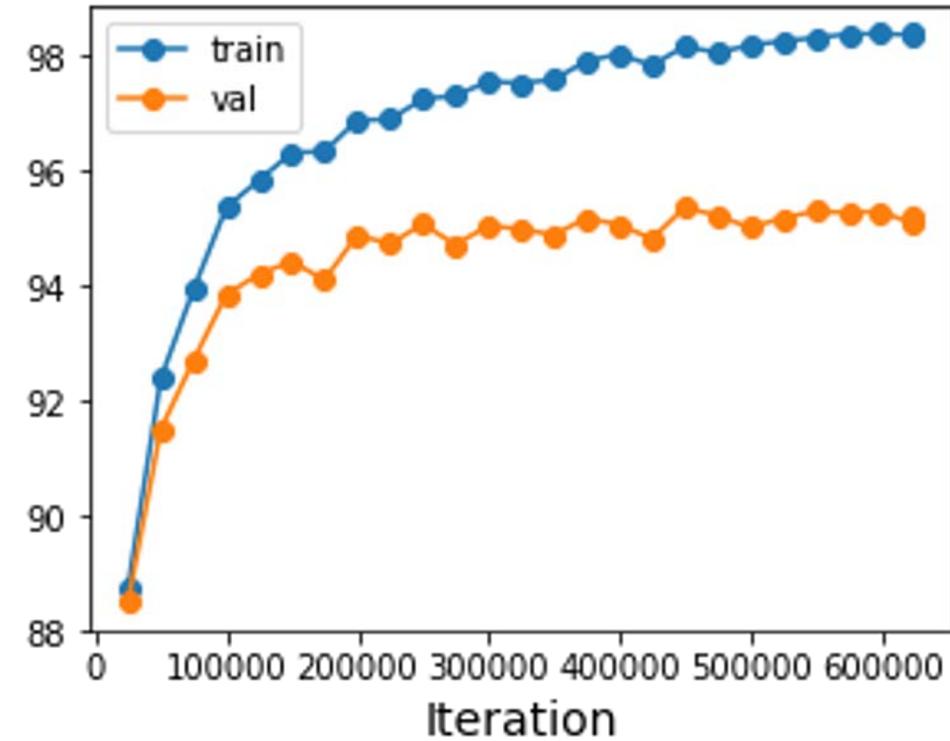
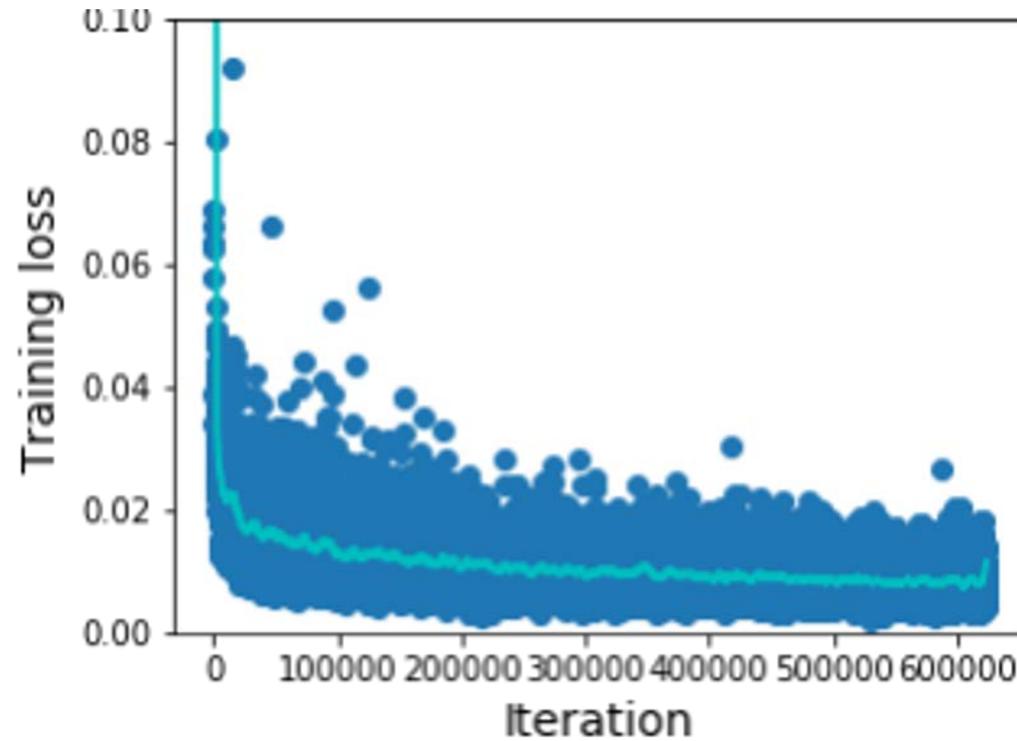
Step 3: Find LR that makes loss go down

Step 4: Coarse grid, train for ~1-5 epochs

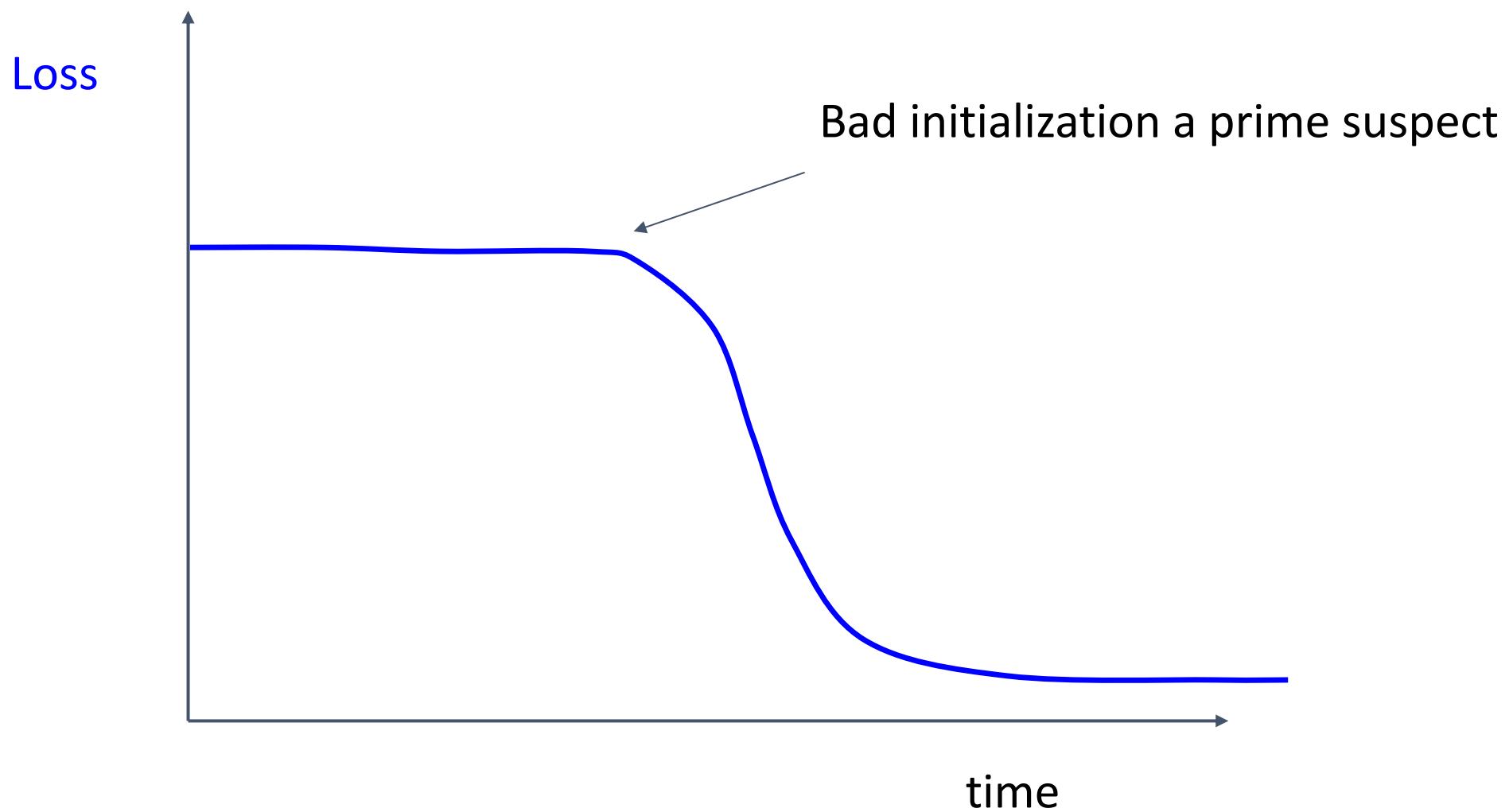
Step 5: Refine grid, train longer

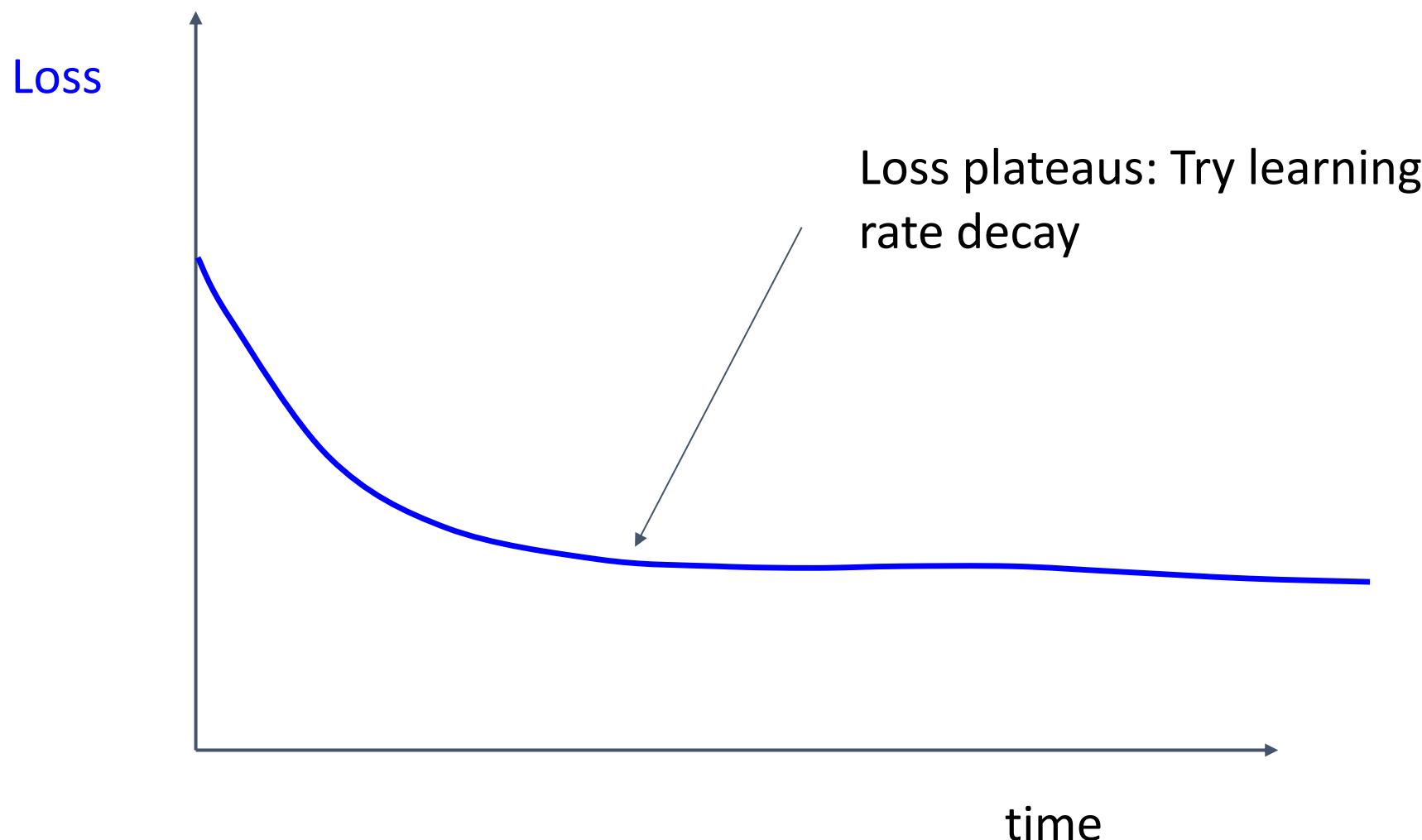
Step 6: Look at learning curves

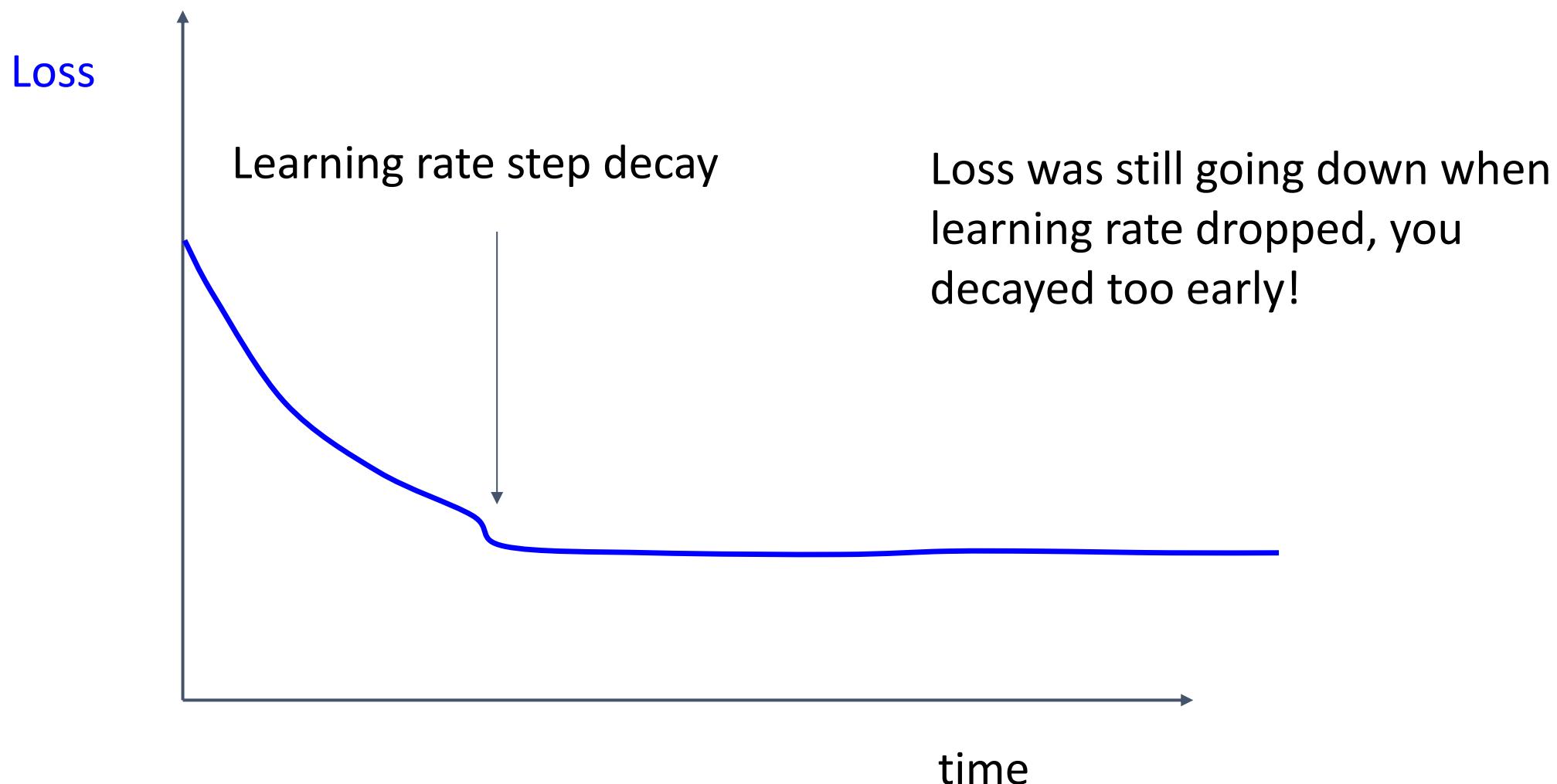
Look at Learning Curves!

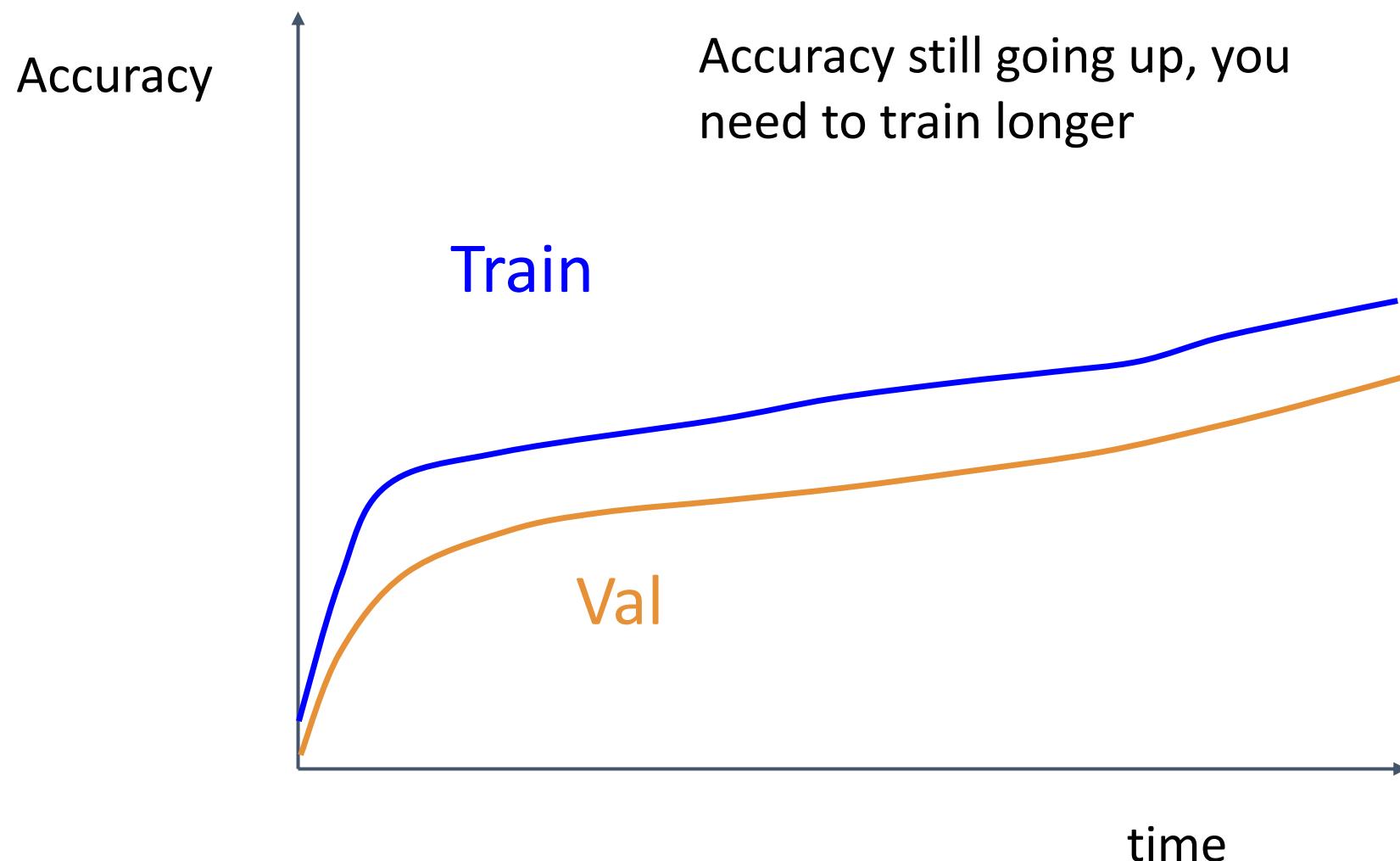


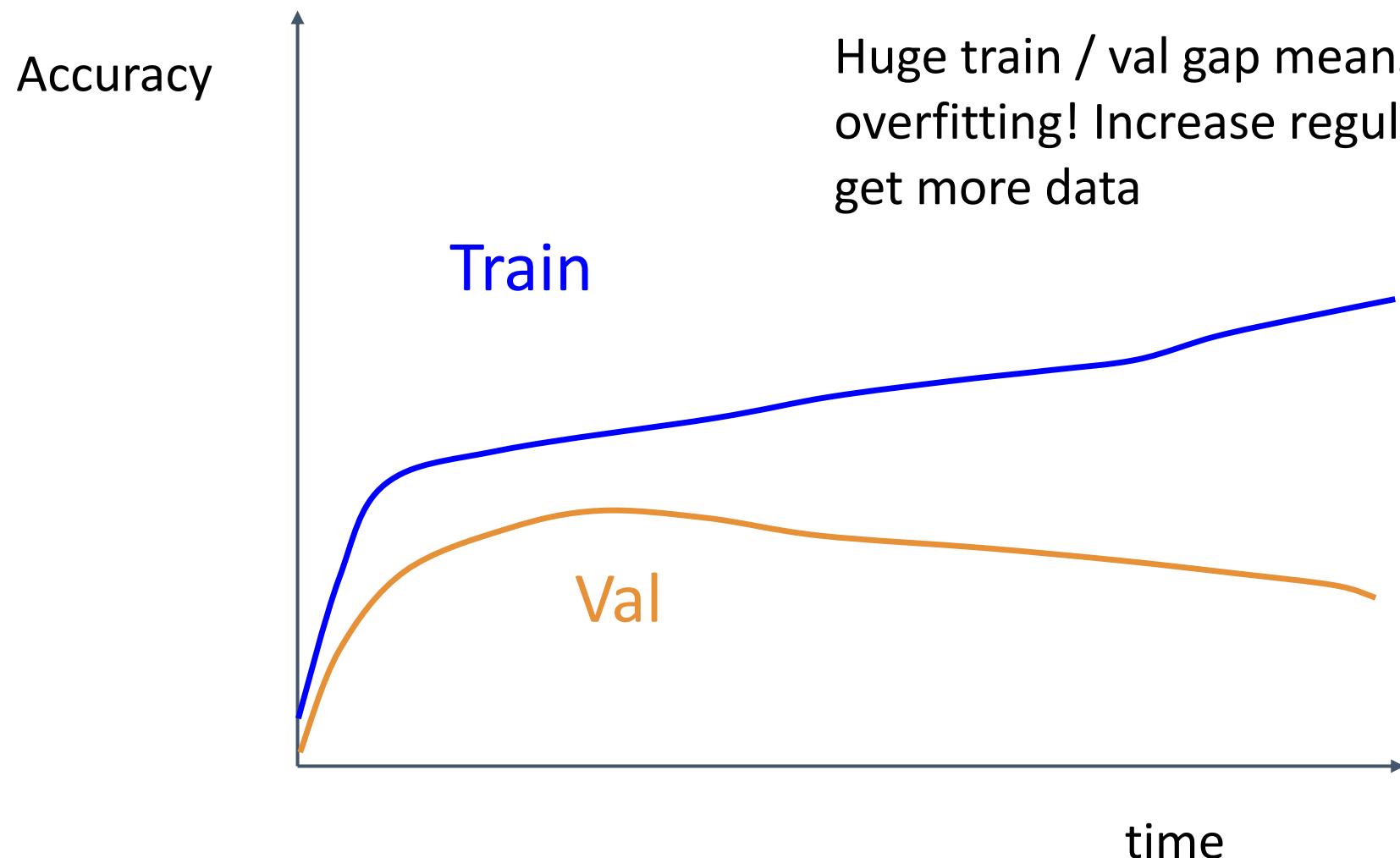
Losses may be noisy, use a scatter plot and also plot moving average to see trends better



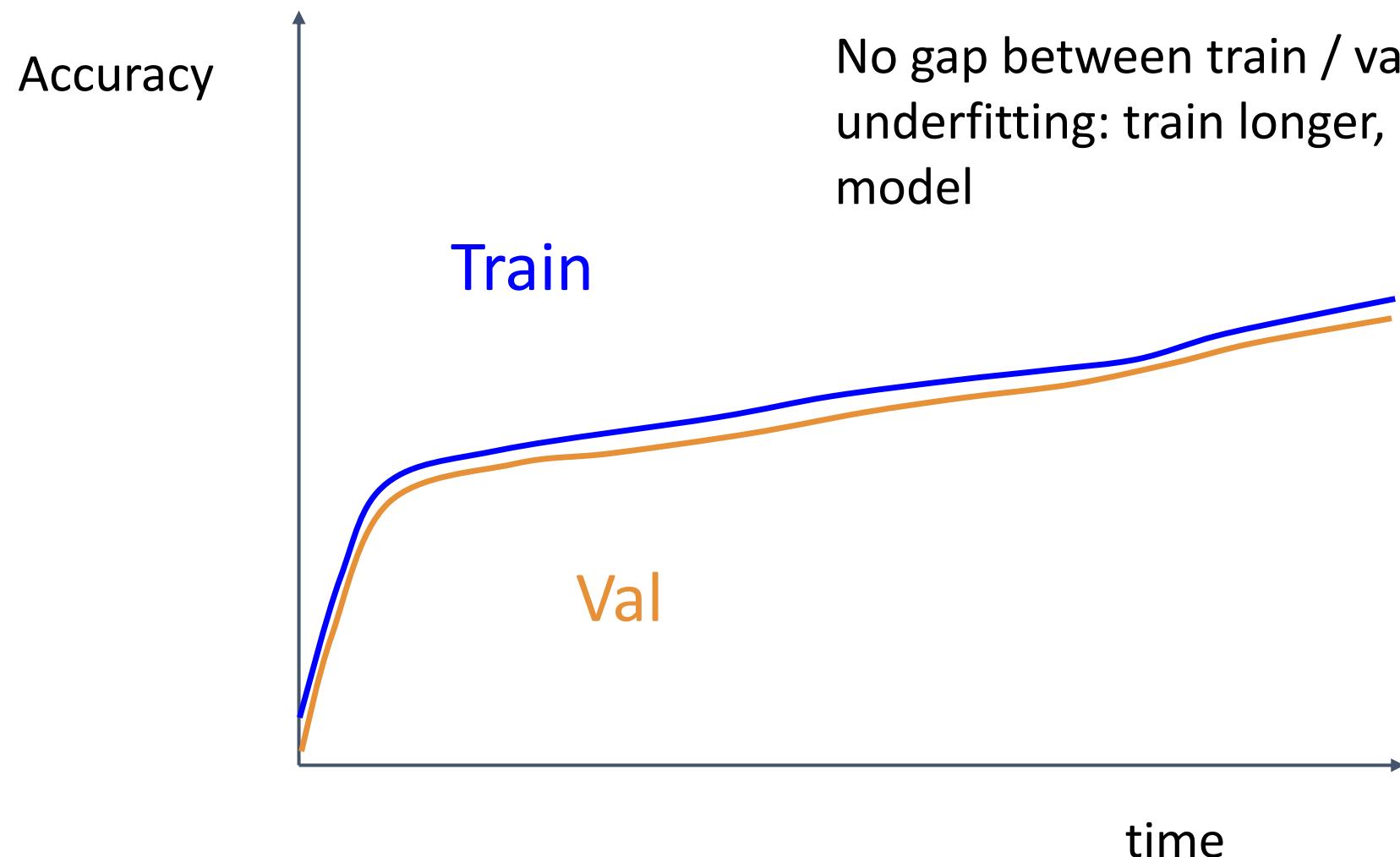








Huge train / val gap means
overfitting! Increase regularization,
get more data



No gap between train / val means underfitting: train longer, use a bigger model

Choosing Hyperparameters

Step 1: Check initial loss

Step 2: Overfit a small sample

Step 3: Find LR that makes loss go down

Step 4: Coarse grid, train for ~1-5 epochs

Step 5: Refine grid, train longer

Step 6: Look at loss curves

Step 7: GOTO step 5

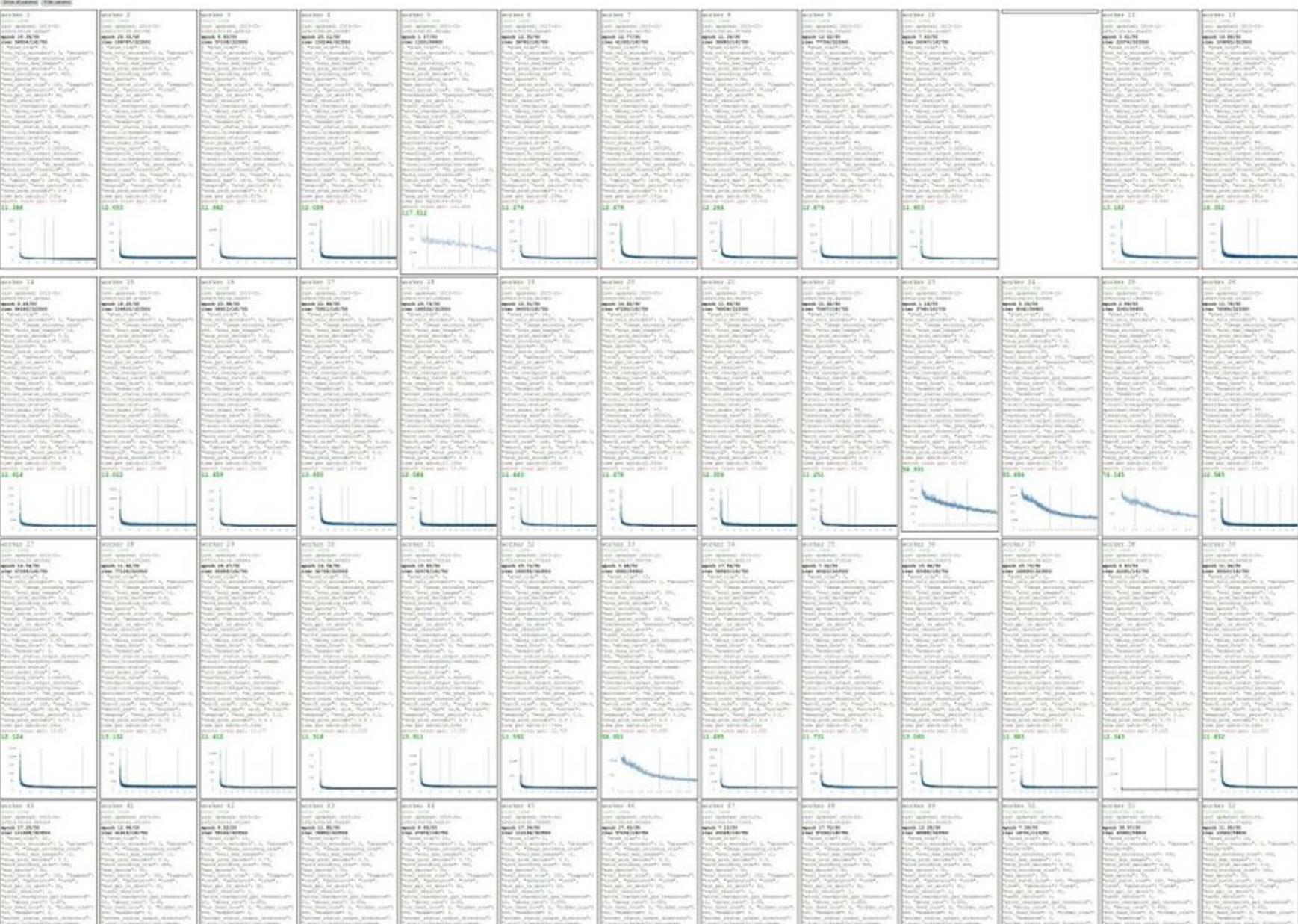
Hyperparameters to play with:

- network architecture
- learning rate, its decay schedule, update type
- regularization (L2/Dropout strength)

neural networks practitioner
music = loss function



Cross-validation “command center”



Track ratio of weight update / weight magnitude

```
# assume parameter vector W and its gradient vector dW
param_scale = np.linalg.norm(W.ravel())
update = -learning_rate*dW # simple SGD update
update_scale = np.linalg.norm(update.ravel())
W += update # the actual update
print update_scale / param_scale # want ~1e-3
```

ratio between the updates and values: $\sim 0.0002 / 0.02 = 0.01$ (about okay)
want this to be somewhere around 0.001 or so

Overview

1. One time setup

Activation functions, data preprocessing, weight initialization, regularization

2. Training dynamics

Learning rate schedules;
hyperparameter optimization

3. After training

Model ensembles, transfer learning,
large-batch training

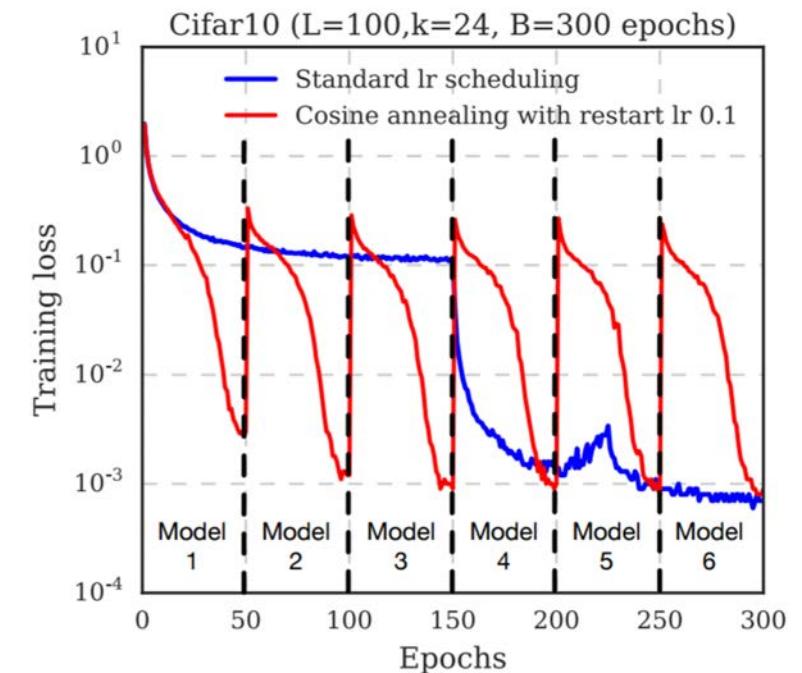
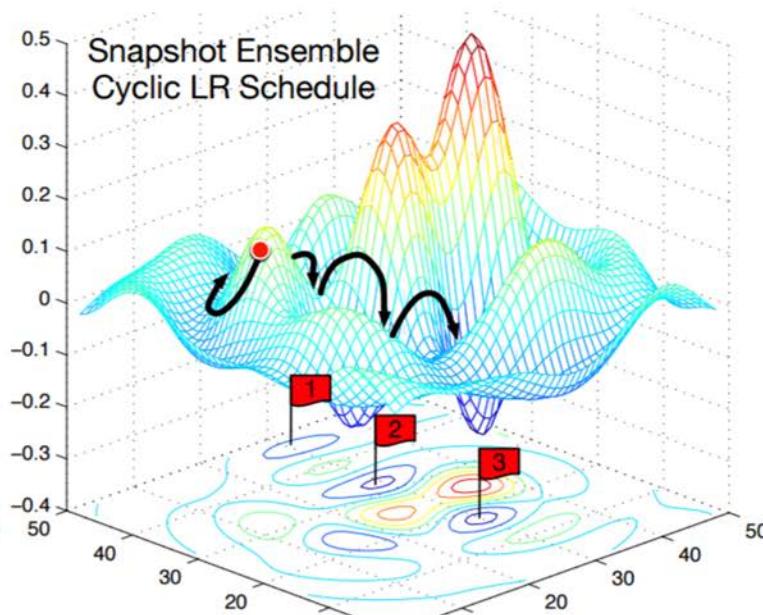
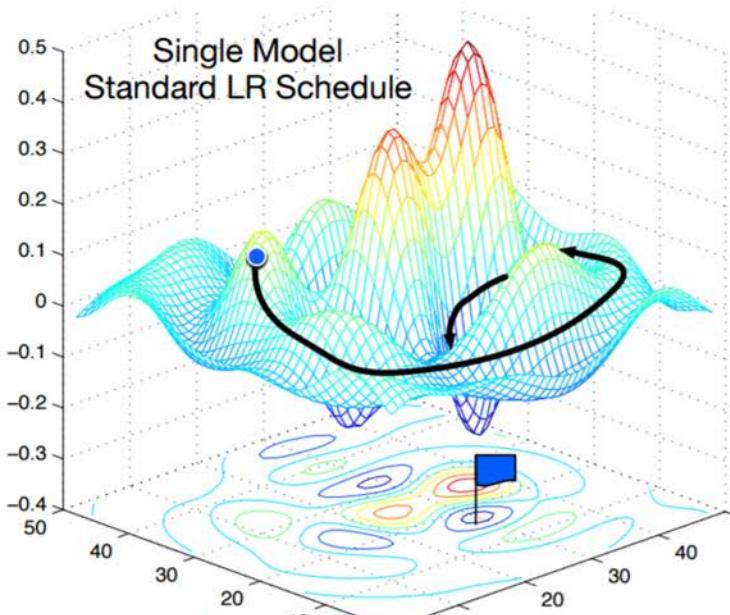
Model Ensembles

1. Train multiple independent models
2. At test time average their results
(Take average of predicted probability distributions, then choose argmax)

Enjoy 2% extra performance

Model Ensembles: Tips and Tricks

Instead of training independent models, use multiple snapshots of a single model during training!



Cyclic learning rate schedules can make this work even better!

Loshchilov and Hutter, "SGDR: Stochastic gradient descent with restarts", arXiv 2016

Huang et al, "Snapshot ensembles: train 1, get M for free", ICLR 2017

Figures copyright Yixuan Li and Geoff Pleiss, 2017. Reproduced with permission.

Model Ensembles: Tips and Tricks

Instead of using actual parameter vector, keep a moving average of the parameter vector and use that at test time (Polyak averaging)

```
while True:  
    data_batch = dataset.sample_data_batch()  
    loss = network.forward(data_batch)  
    dx = network.backward()  
    x += - learning_rate * dx  
    x_test = 0.995*x_test + 0.005*x # use for test set
```

Polyak and Juditsky, "Acceleration of stochastic approximation by averaging", SIAM Journal on Control and Optimization, 1992.

Karras et al, "Progressive Growing of GANs for Improved Quality, Stability, and Variation", ICLR 2018

Brock et al, "Large Scale GAN Training for High Fidelity Natural Image Synthesis", ICLR 2019

Transfer Learning

Transfer Learning

“You need a lot of data if you
want to train/use CNNs”

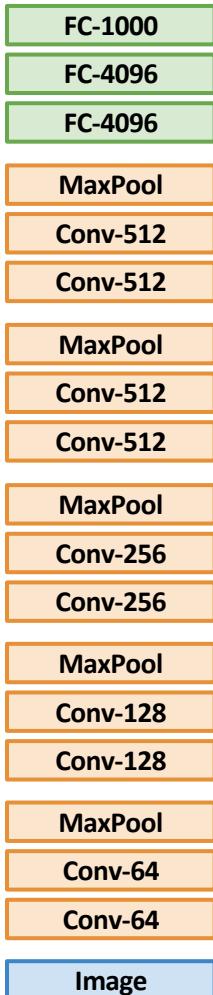
Transfer Learning

“You need a lot of data if you
want to train/use CNNs”

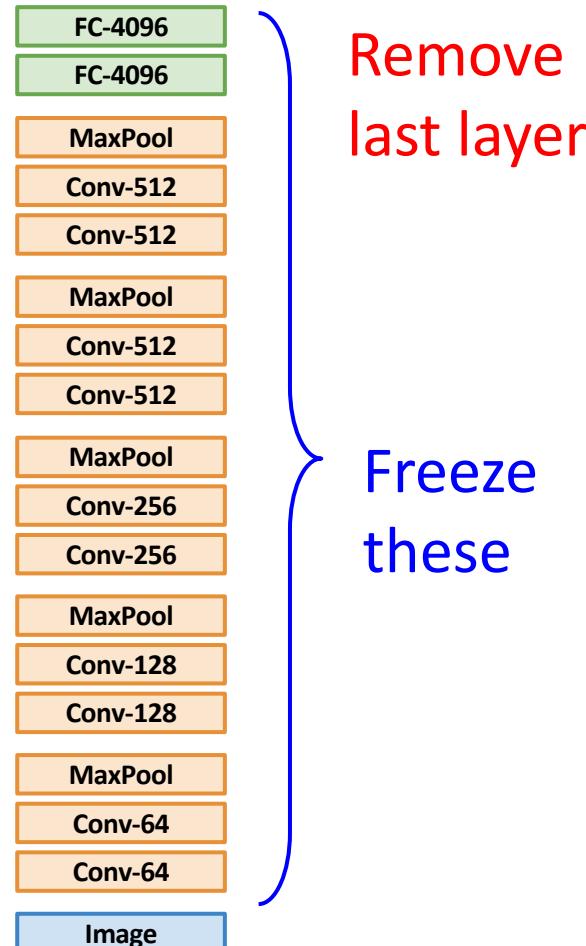
BUSTED

Transfer Learning with CNNs

1. Train on Imagenet



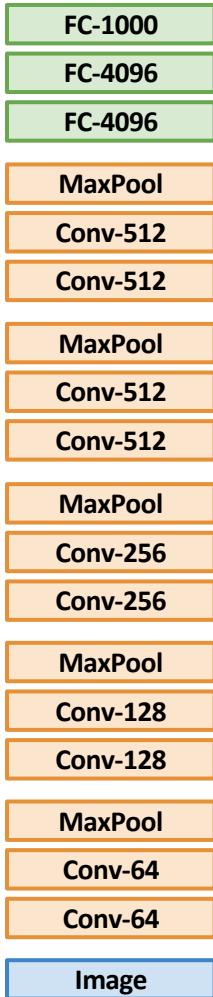
2. Use CNN as a feature extractor



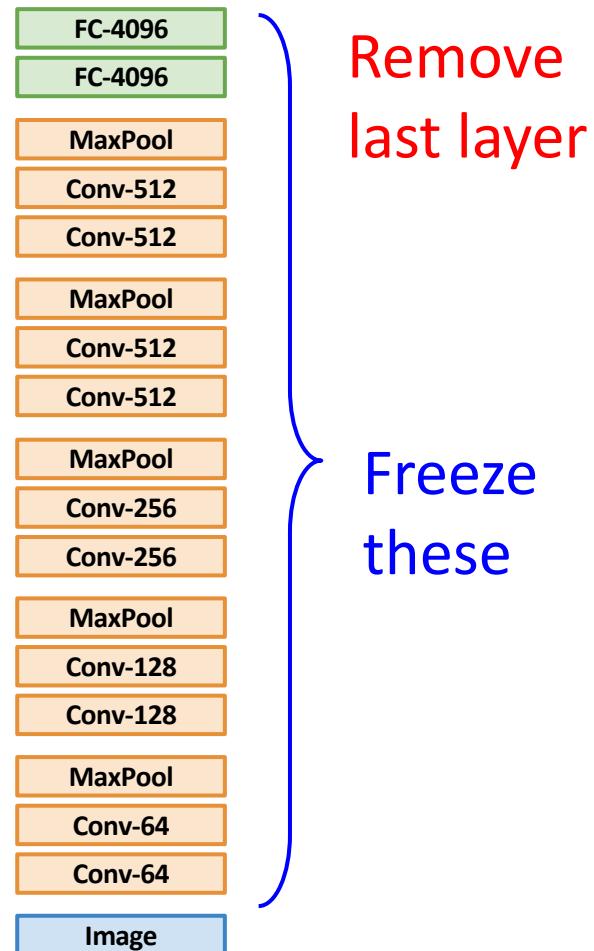
Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014

Transfer Learning with CNNs

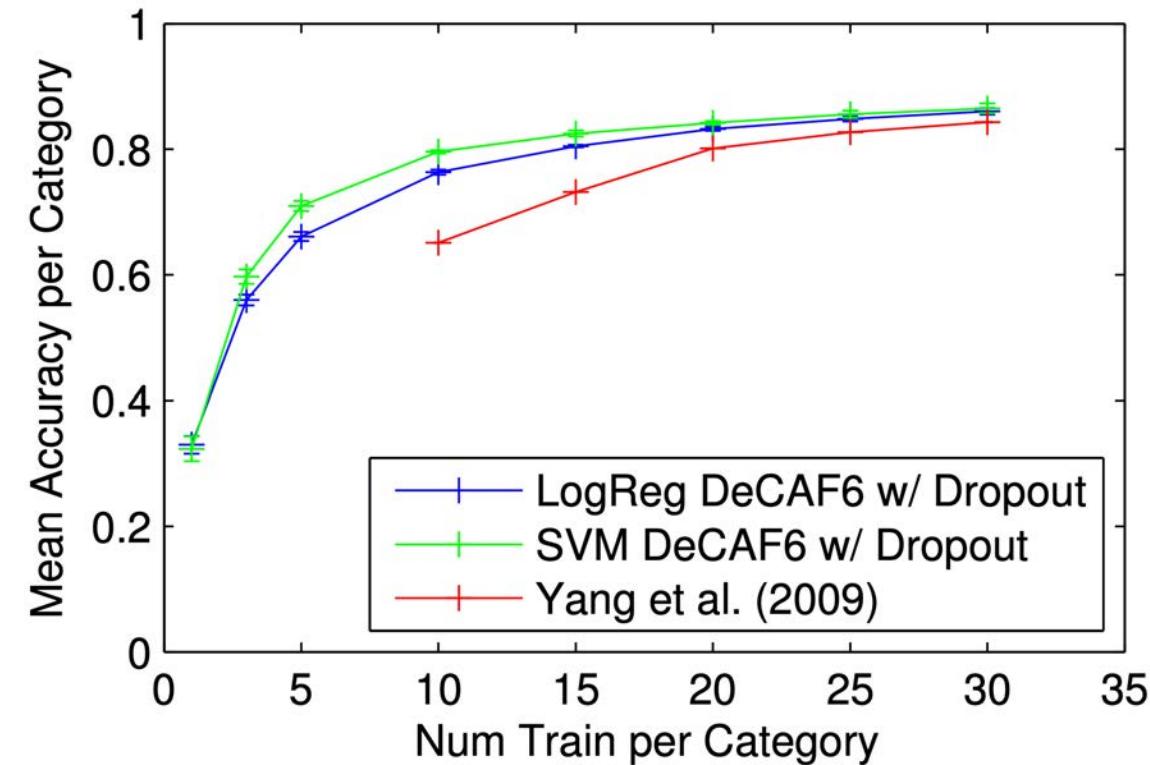
1. Train on Imagenet



2. Use CNN as a feature extractor



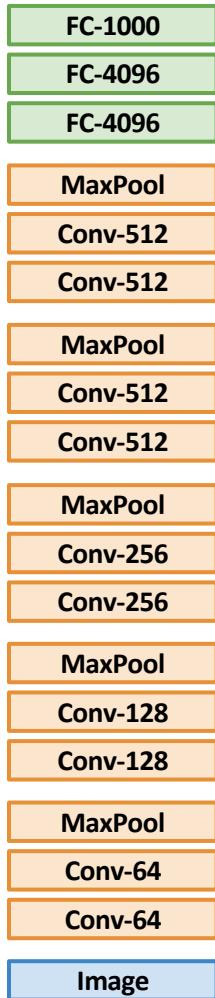
Classification on Caltech-101



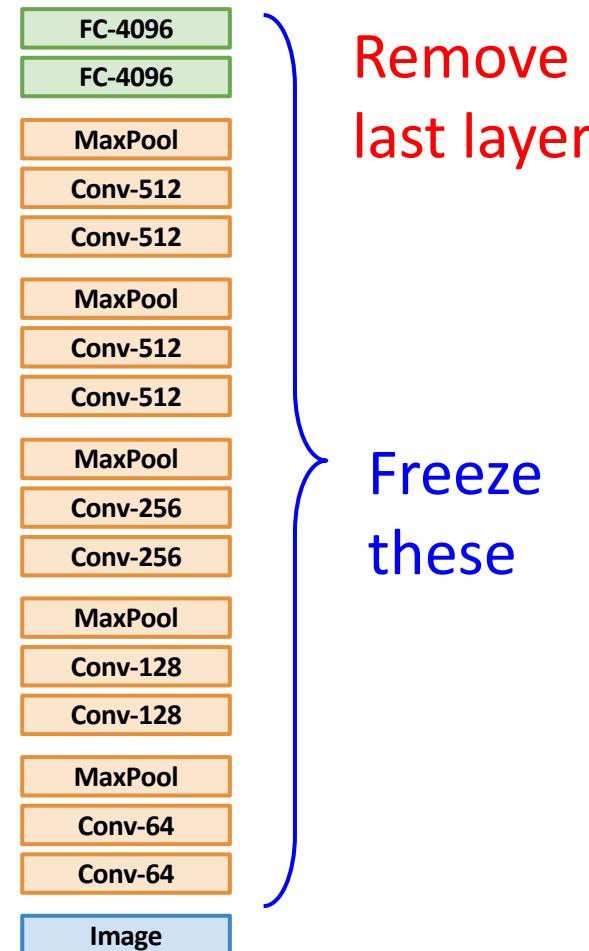
Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014

Transfer Learning with CNNs

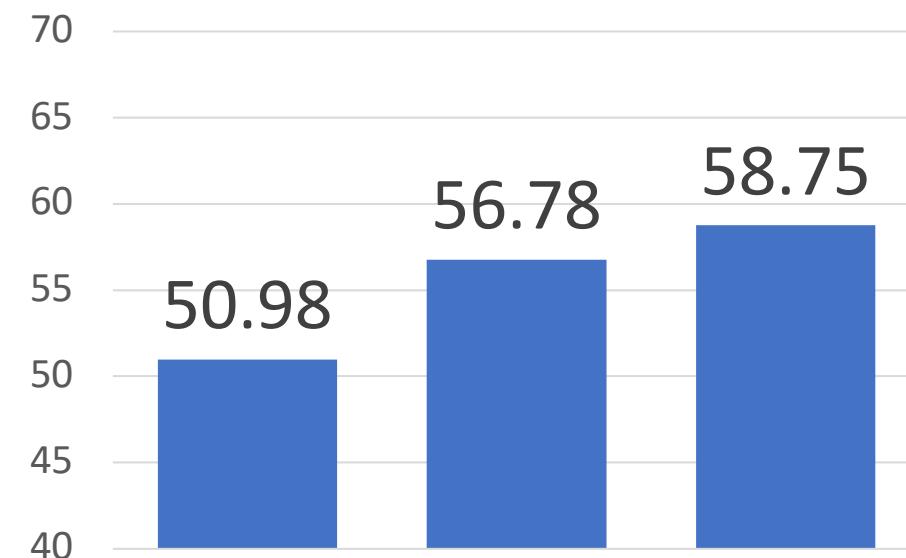
1. Train on Imagenet



2. Use CNN as a feature extractor



Bird Classification on Caltech-UCSD

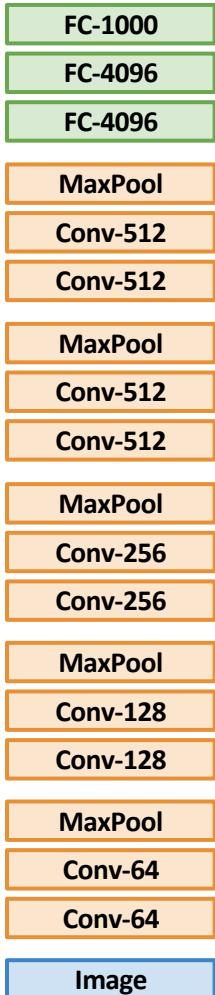


DPD (Zhang et al, 2013)
POOF (Berg & Belhumeur, 2013)
AlexNet FC6 + logistic regression

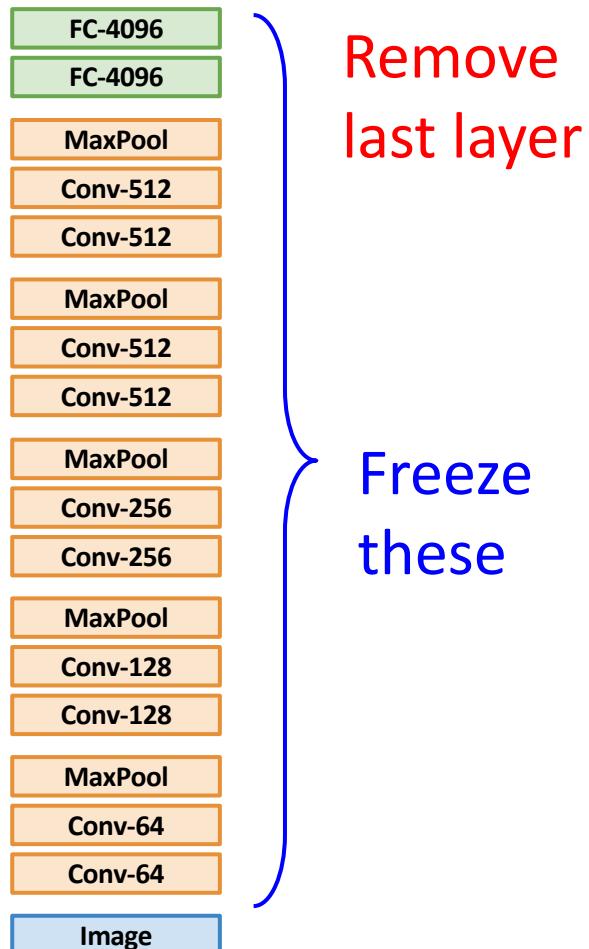
Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014

Transfer Learning with CNNs

1. Train on Imagenet



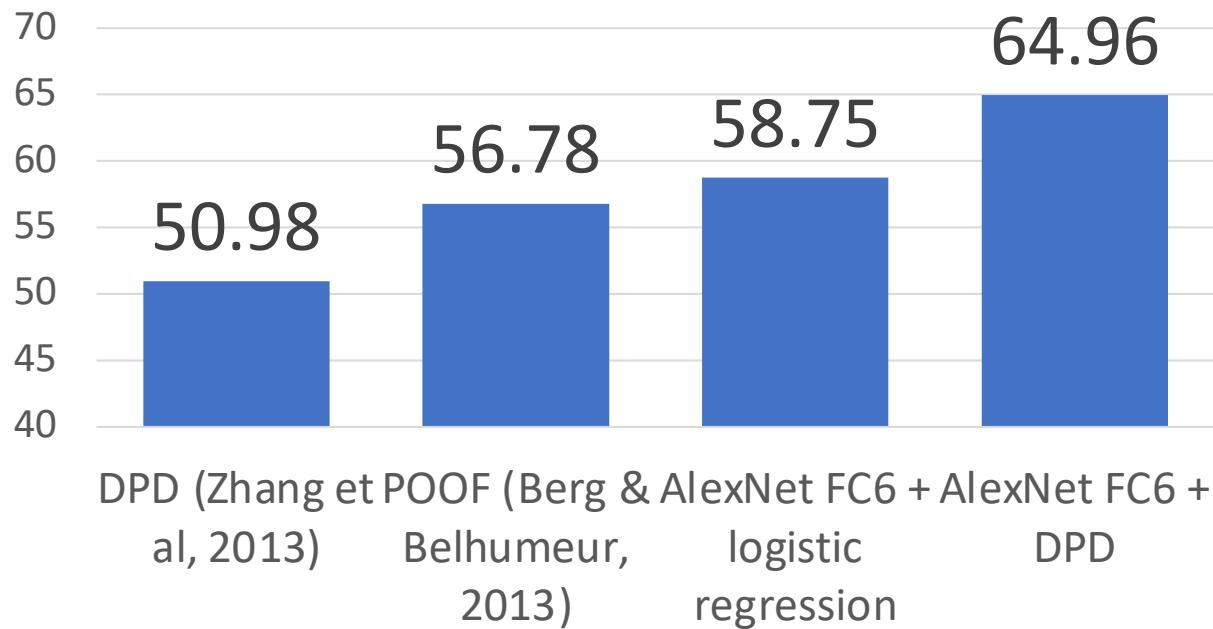
2. Use CNN as a feature extractor



Remove
last layer

Freeze
these

Bird Classification on Caltech-UCSD



Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014

Transfer Learning with CNNs

1. Train on Imagenet



2. Use CNN as a feature extractor

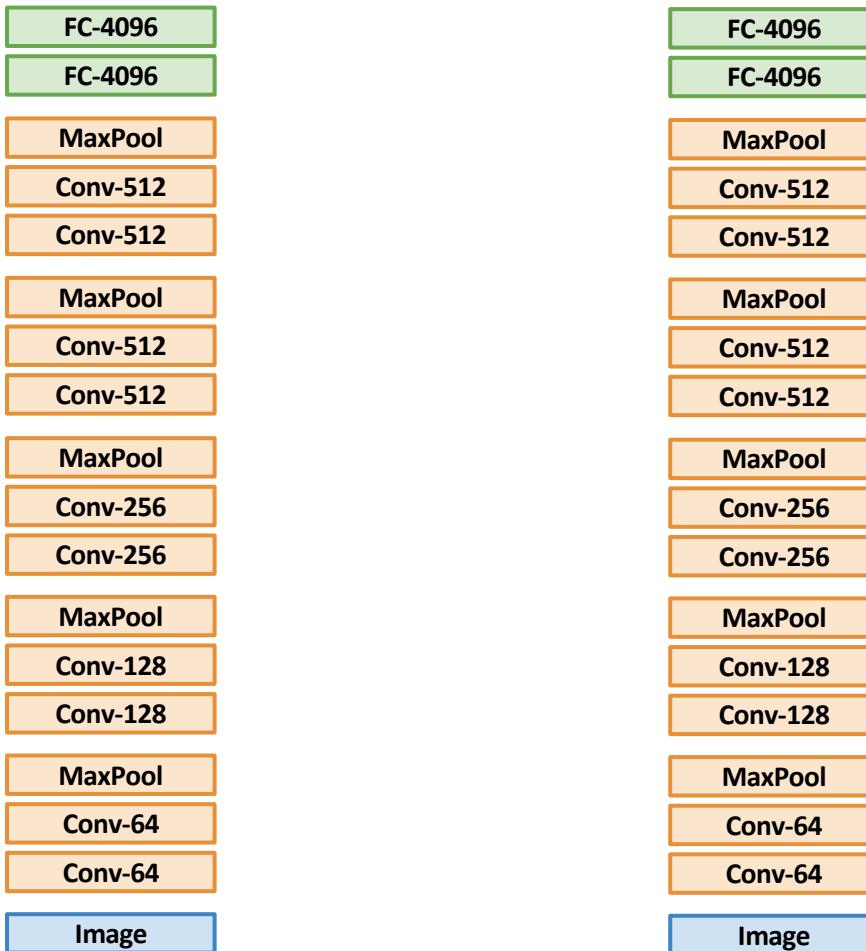
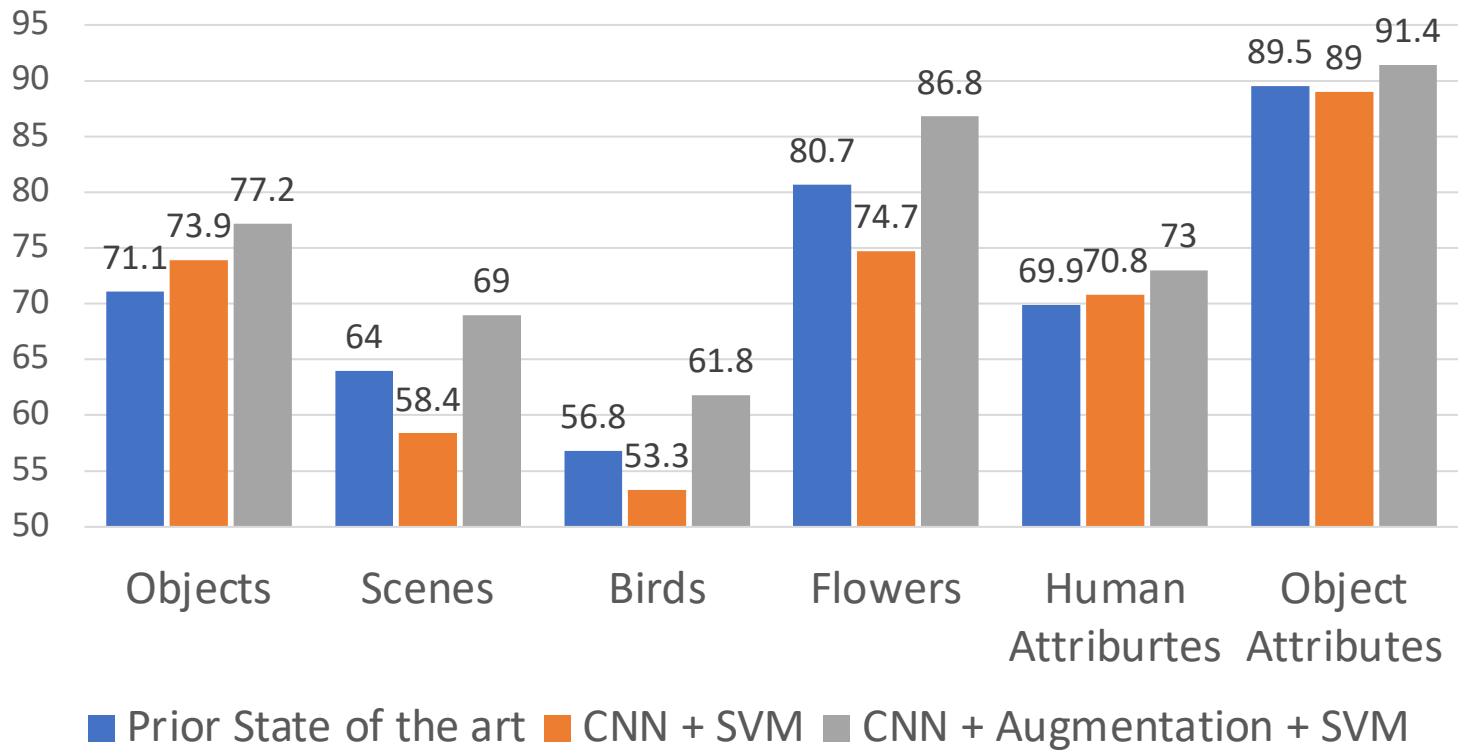


Image Classification



■ Prior State of the art ■ CNN + SVM ■ CNN + Augmentation + SVM

Razavian et al, "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition", CVPR Workshops 2014

Transfer Learning with CNNs

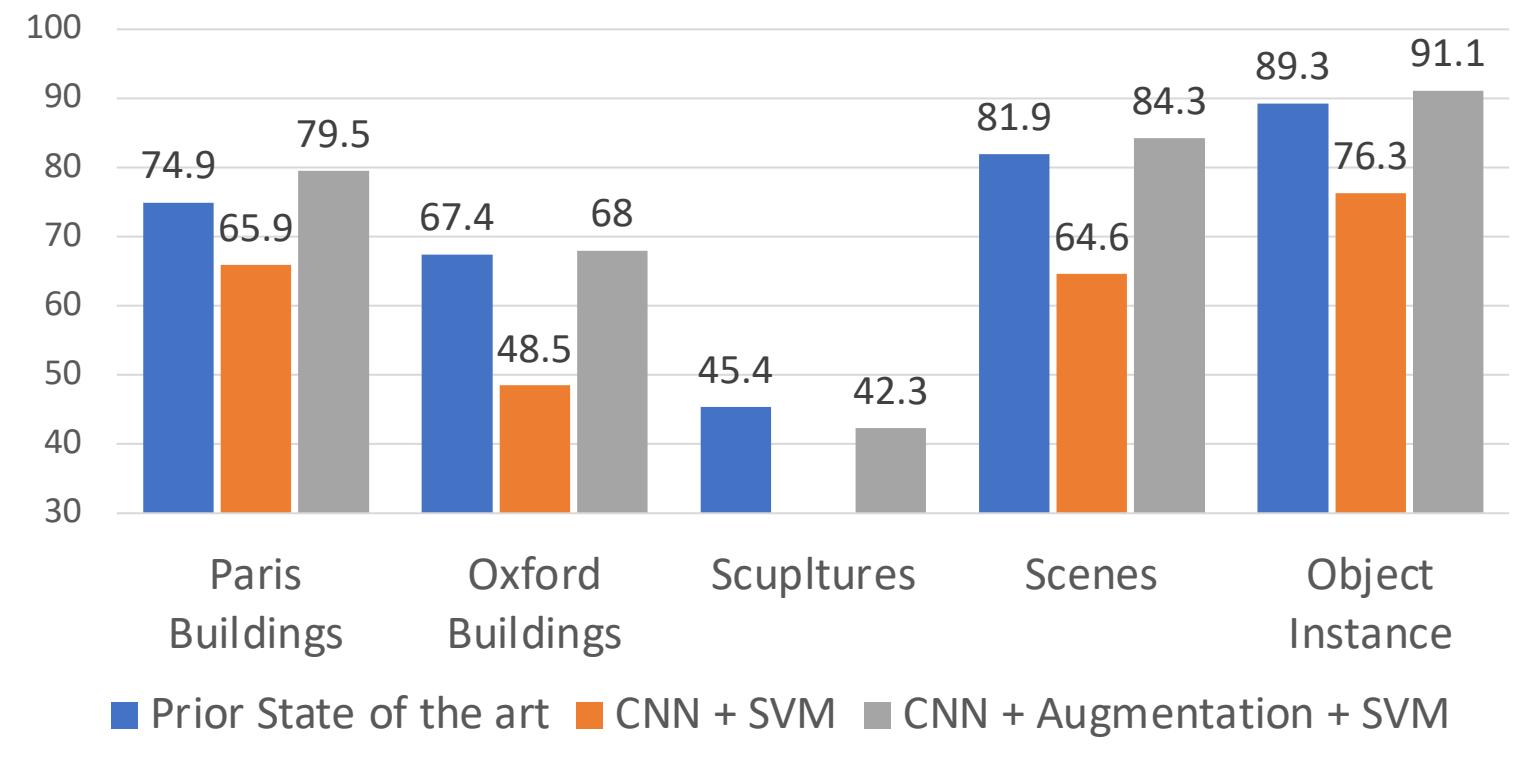
1. Train on Imagenet



2. Use CNN as a feature extractor



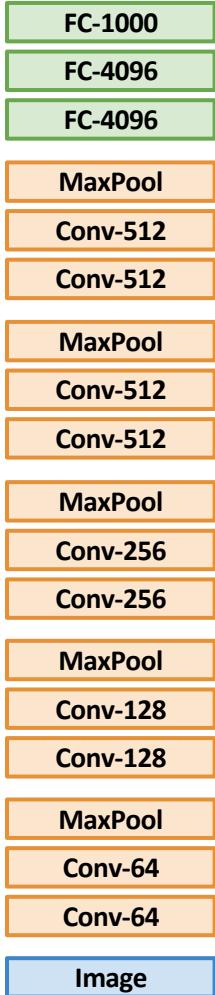
Image Retrieval: Nearest-Neighbor



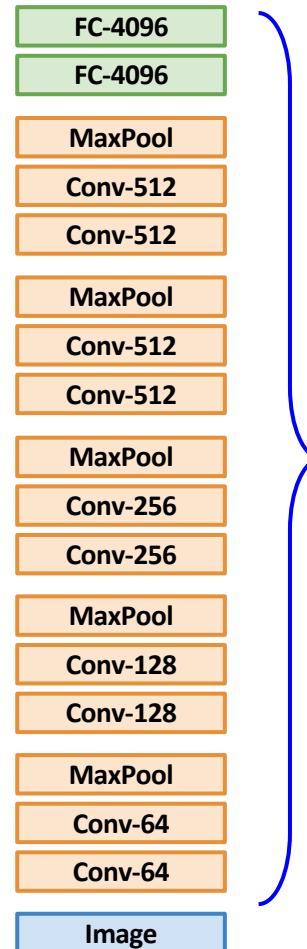
Razavian et al, "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition", CVPR Workshops 2014

Transfer Learning with CNNs

1. Train on Imagenet



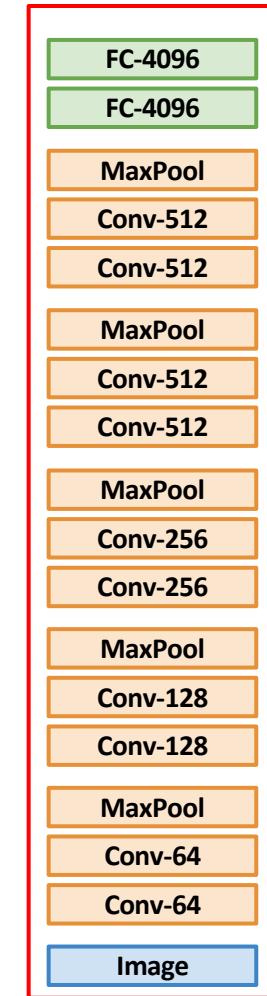
2. Use CNN as a feature extractor



Remove
last layer

Freeze
these

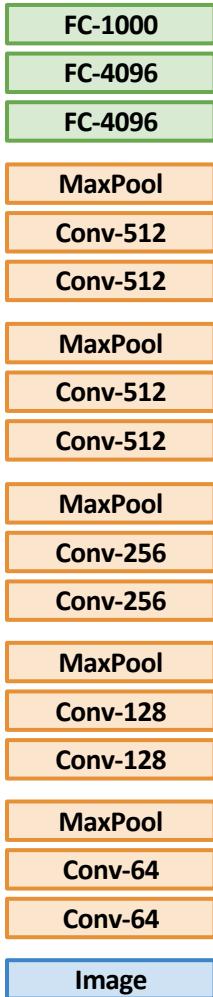
3. Bigger dataset:
Fine-Tuning



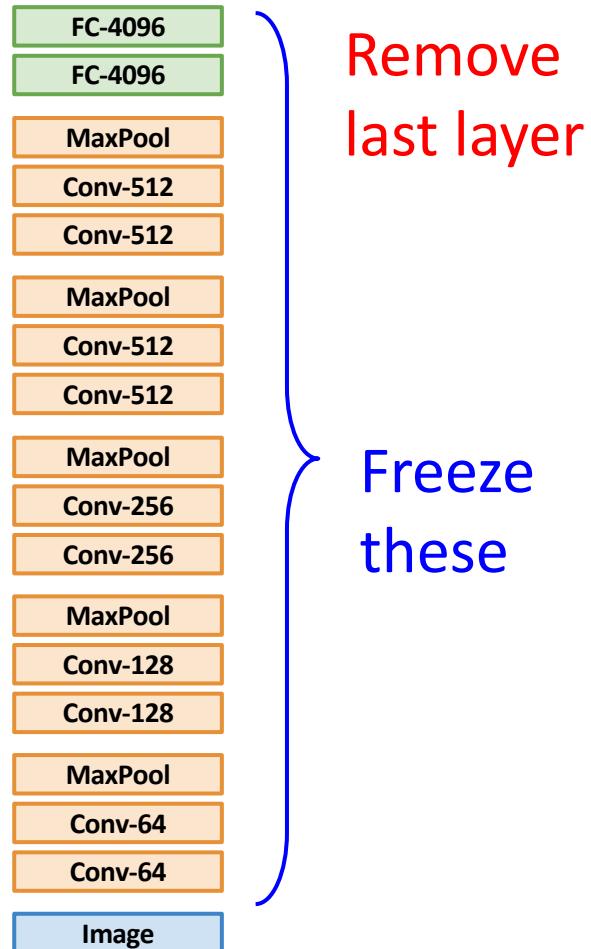
Continue training
CNN for new task!

Transfer Learning with CNNs

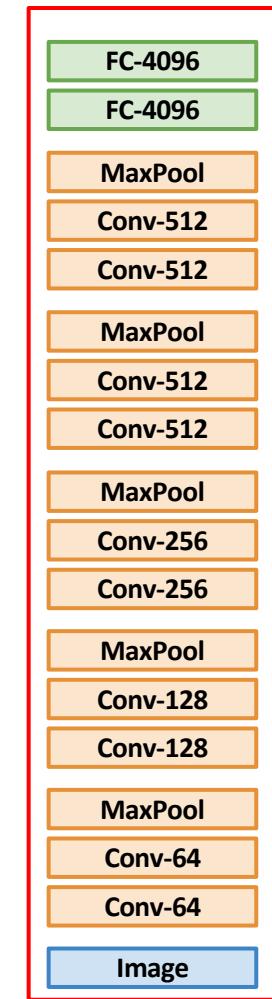
1. Train on Imagenet



2. Use CNN as a feature extractor



3. Bigger dataset: Fine-Tuning



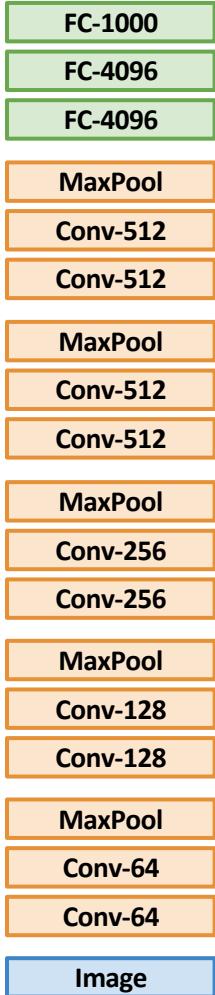
Continue training
CNN for new task!

Some tricks:

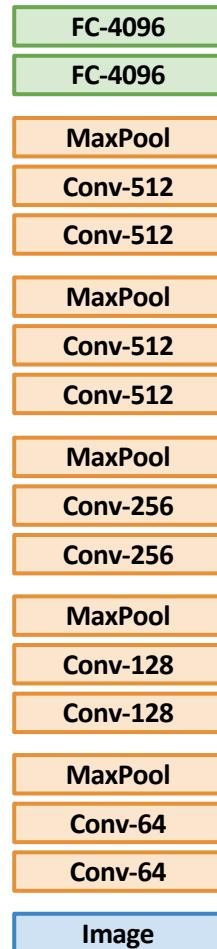
- Train with feature extraction first before fine-tuning
- Lower the learning rate: use ~1/10 of LR used in original training
- Sometimes freeze lower layers to save computation

Transfer Learning with CNNs

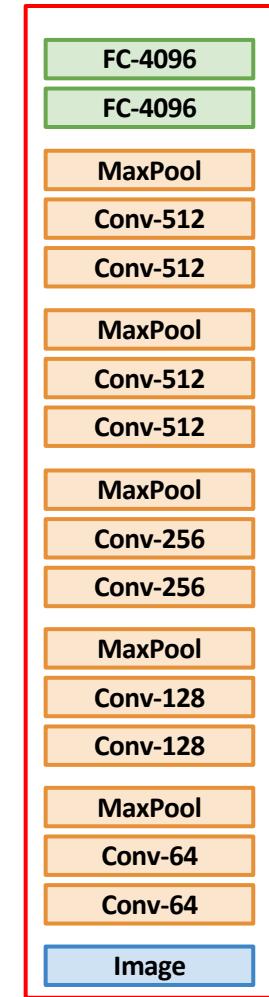
1. Train on Imagenet



2. Use CNN as a feature extractor

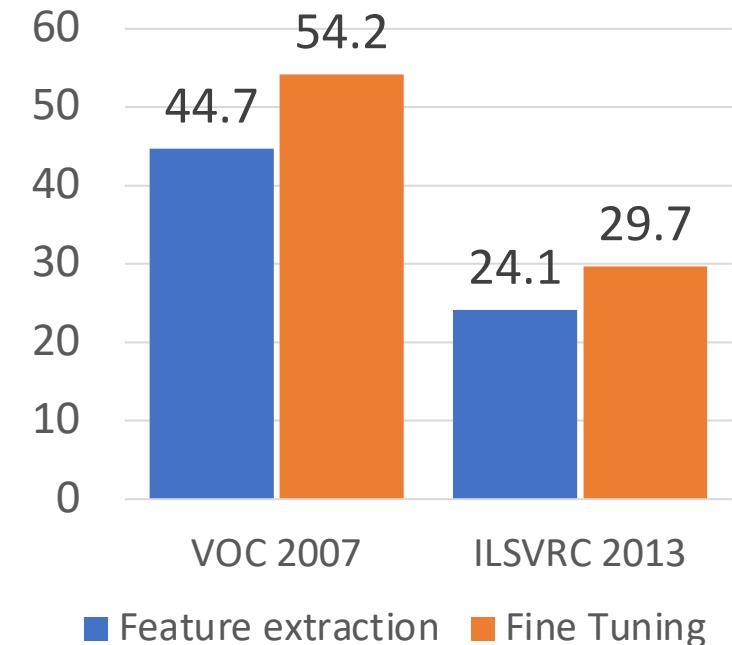


3. Bigger dataset:
Fine-Tuning



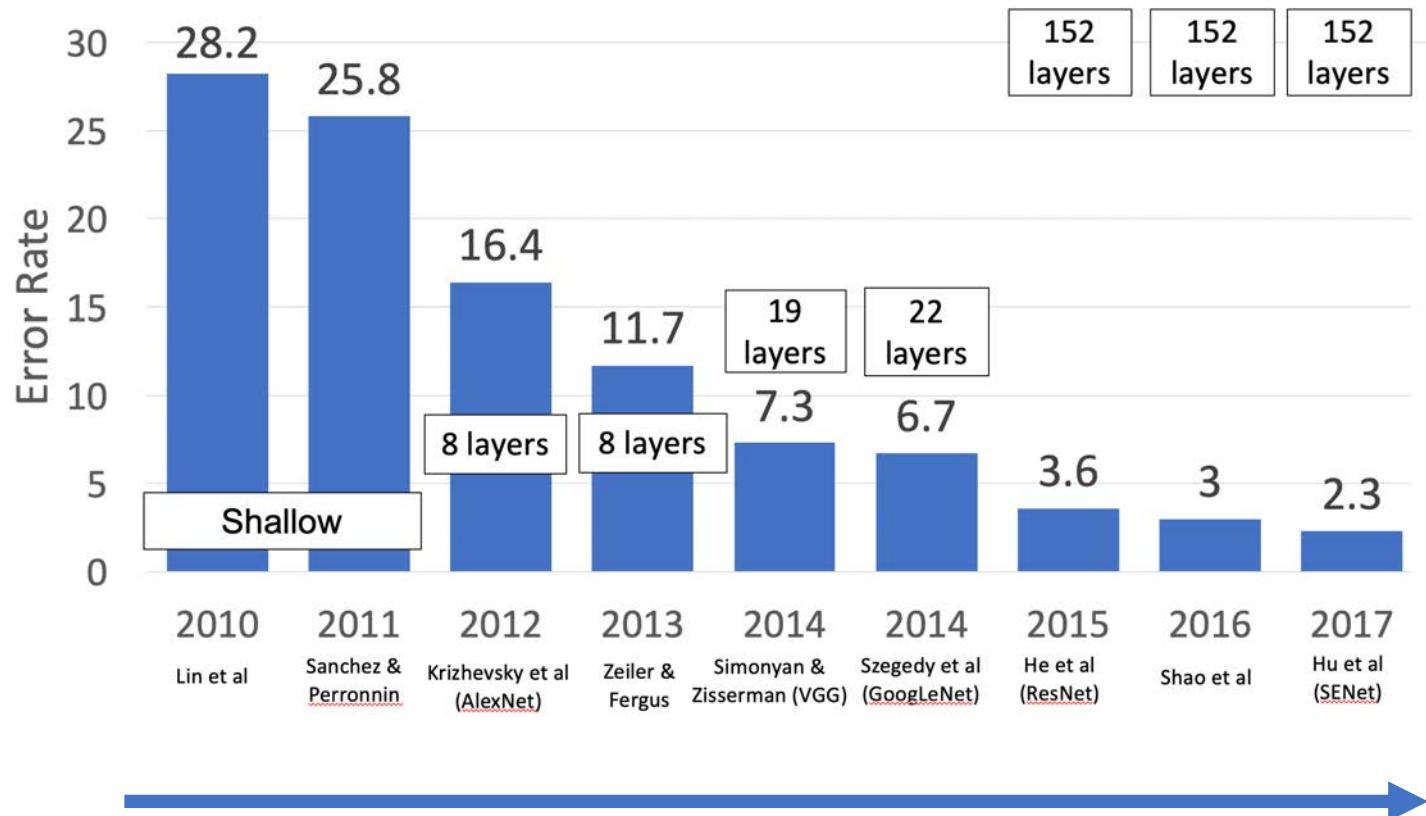
Continue training
CNN for new task!

Object Detection



Transfer Learning with CNNs: Architecture Matters!

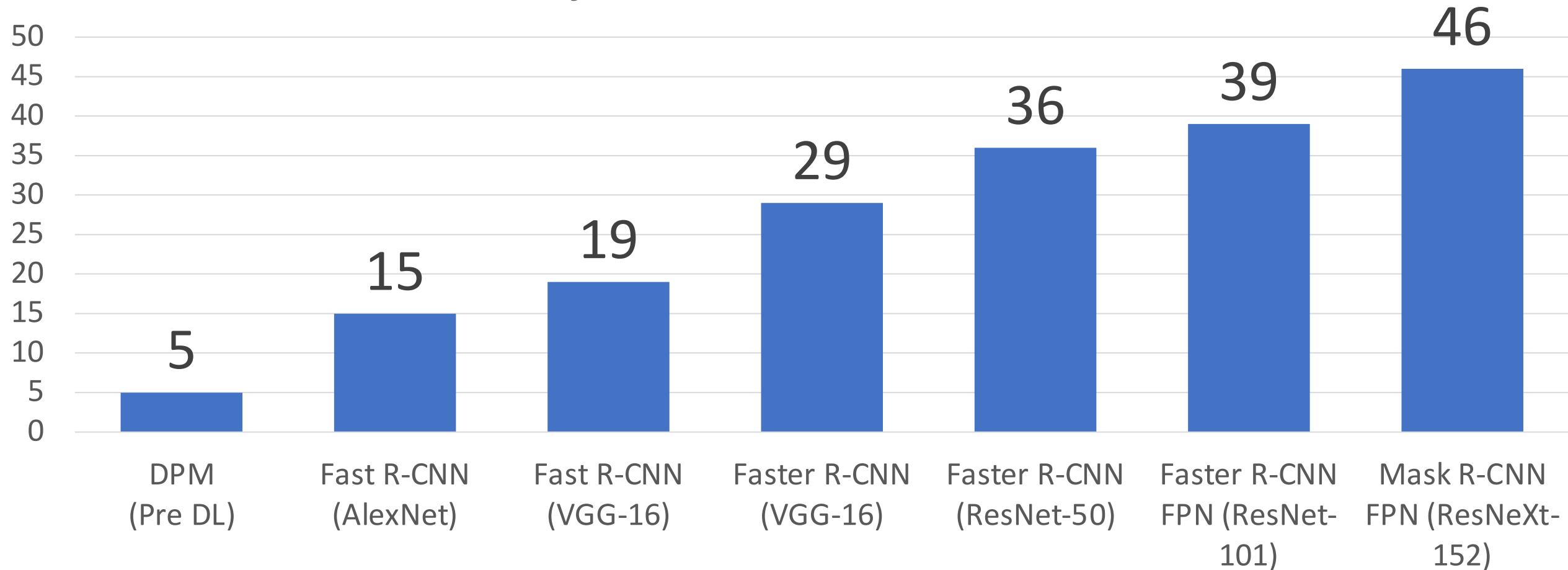
ImageNet Classification Challenge



Improvements in CNN architectures lead to improvements in many downstream tasks thanks to transfer learning!

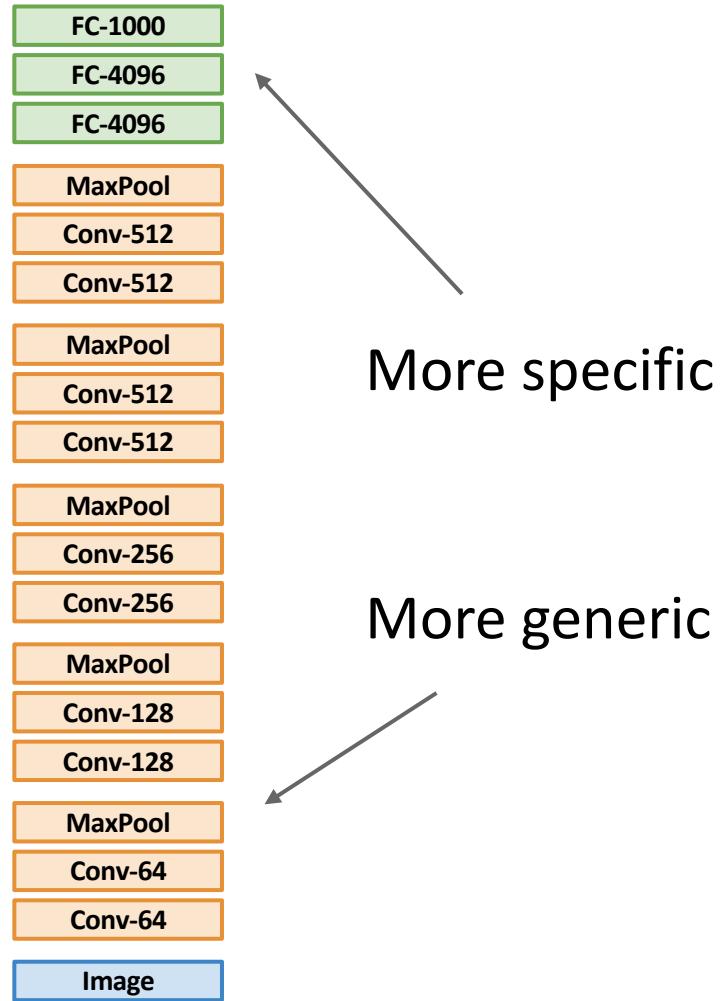
Transfer Learning with CNNs: Architecture Matters!

Object Detection on COCO



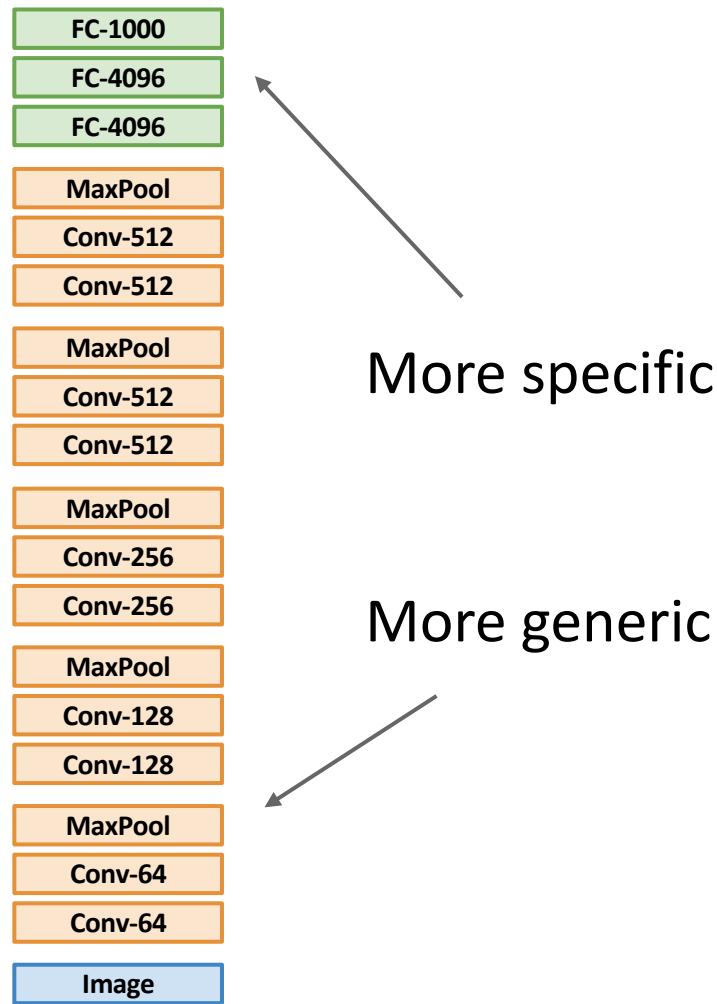
Ross Girshick, "The Generalized R-CNN Framework for Object Detection", ICCV 2017 Tutorial on Instance-Level Visual Recognition

Transfer Learning with CNNs



	Dataset similar to ImageNet	Dataset very different from ImageNet
very little data (10s to 100s)	?	?
quite a lot of data (100s to 1000s)	?	?

Transfer Learning with CNNs



	Dataset similar to ImageNet	Dataset very different from ImageNet
very little data (10s to 100s)	Use Linear Classifier on top layer	?
quite a lot of data (100s to 1000s)	Finetune a few layers	?

Transfer Learning with CNNs



More specific

More generic

	Dataset similar to ImageNet	Dataset very different from ImageNet
very little data (10s to 100s)	Use Linear Classifier on top layer	?
quite a lot of data (100s to 1000s)	Finetune a few layers	Finetune a larger number of layers

Transfer Learning with CNNs



More specific

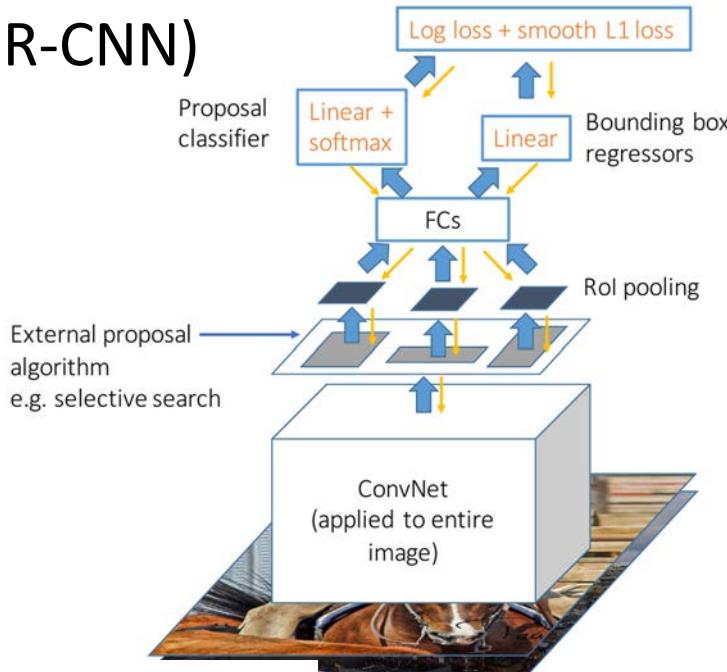
More generic

	Dataset similar to ImageNet	Dataset very different from ImageNet
very little data (10s to 100s)	Use Linear Classifier on top layer	You're in trouble... Try linear classifier from different stages
quite a lot of data (100s to 1000s)	Finetune a few layers	Finetune a larger number of layers

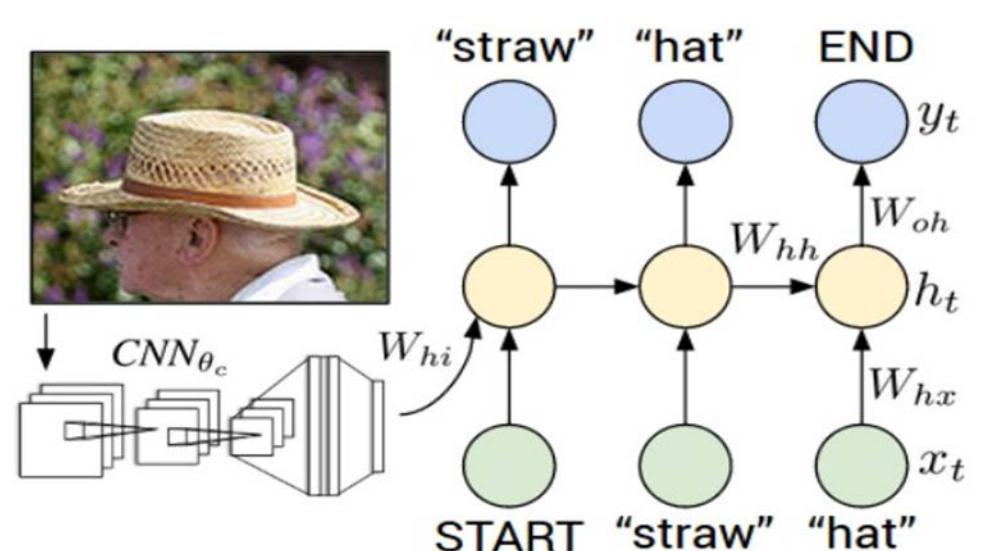
Transfer learning is pervasive!

It's the norm, not the exception

Object Detection (Fast R-CNN)



Girshick, "Fast R-CNN", ICCV 2015
Figure copyright Ross Girshick, 2015. Reproduced with permission.

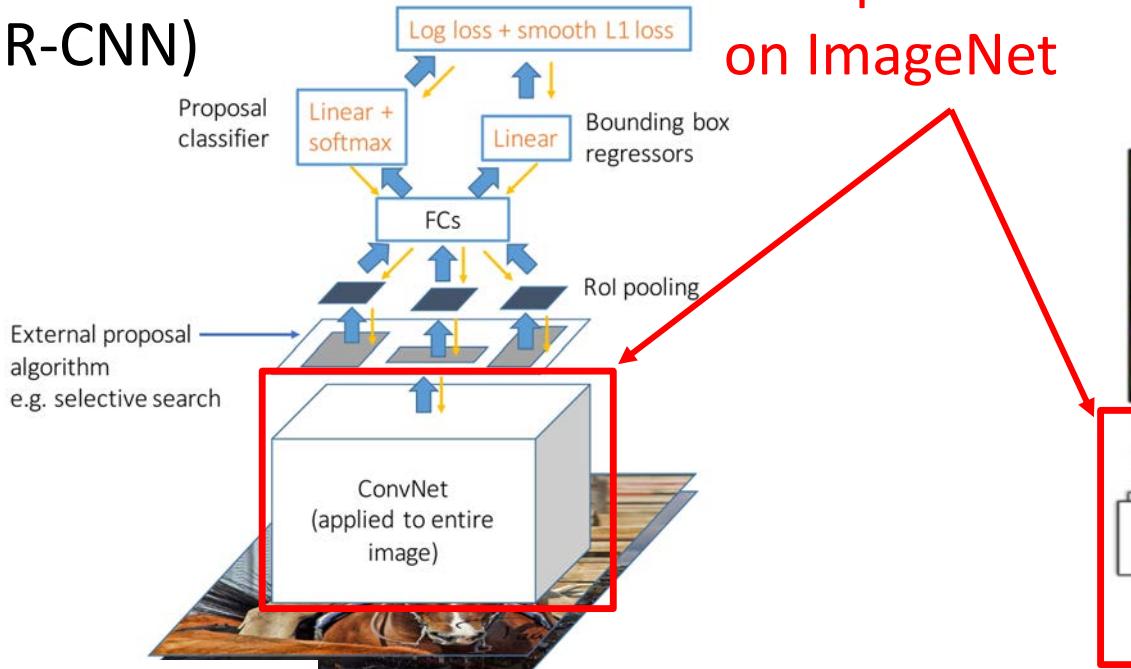


Karpathy and Fei-Fei, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015

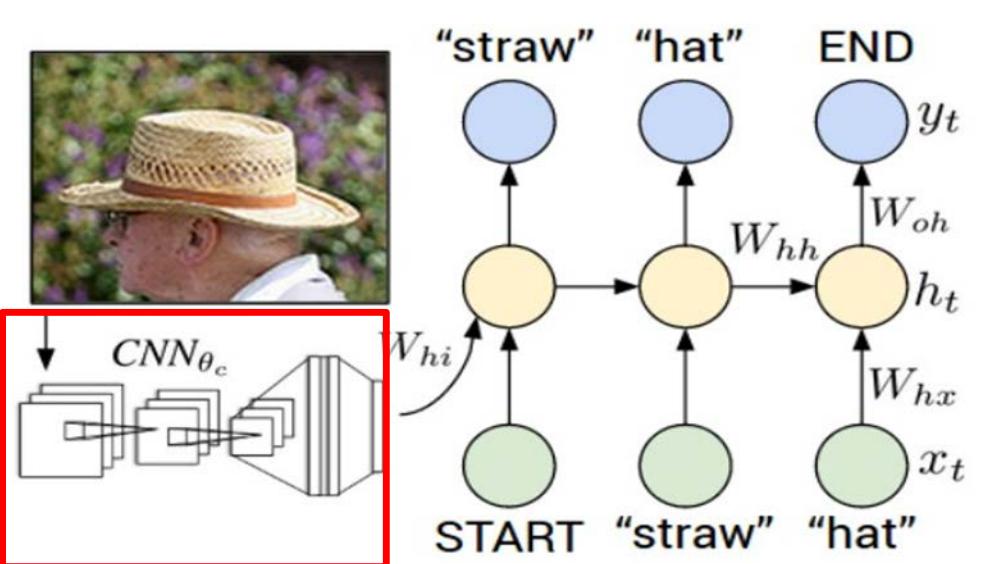
Transfer learning is pervasive!

It's the norm, not the exception

Object Detection (Fast R-CNN)



CNN pretrained
on ImageNet



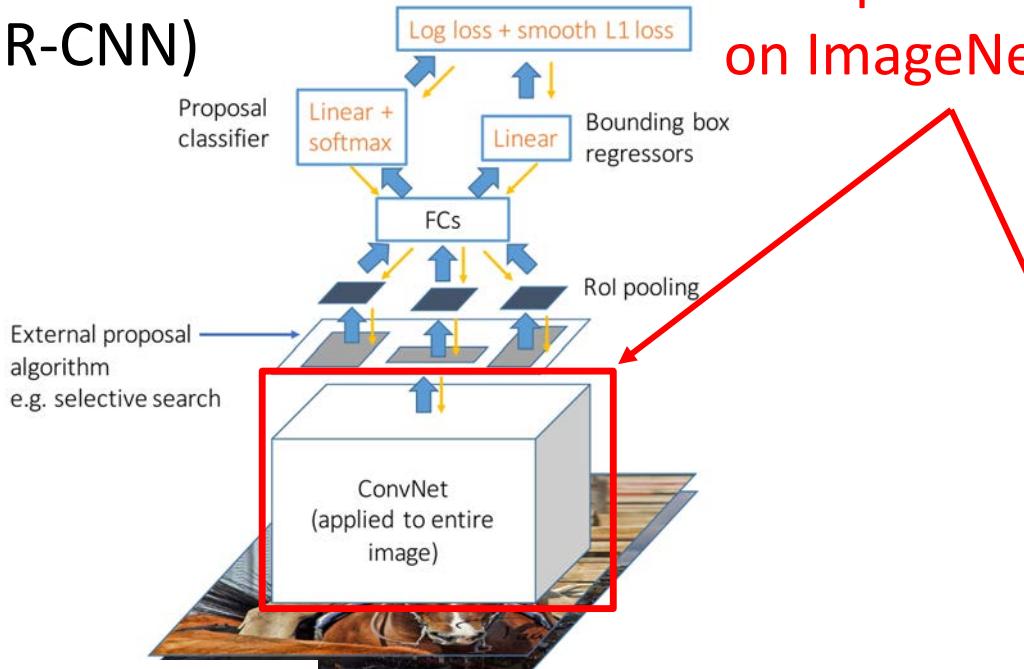
Girshick, "Fast R-CNN", ICCV 2015
Figure copyright Ross Girshick, 2015. Reproduced with permission.

Karpathy and Fei-Fei, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015

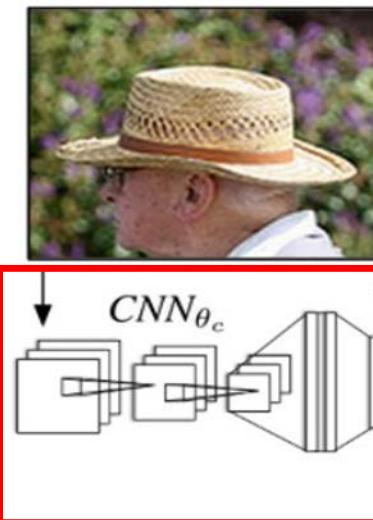
Transfer learning is pervasive!

It's the norm, not the exception

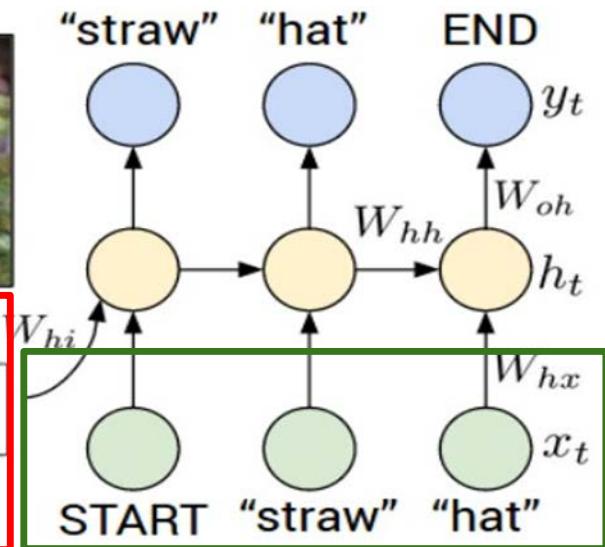
Object
Detection
(Fast R-CNN)



CNN pretrained
on ImageNet



Word vectors pretrained
with word2vec

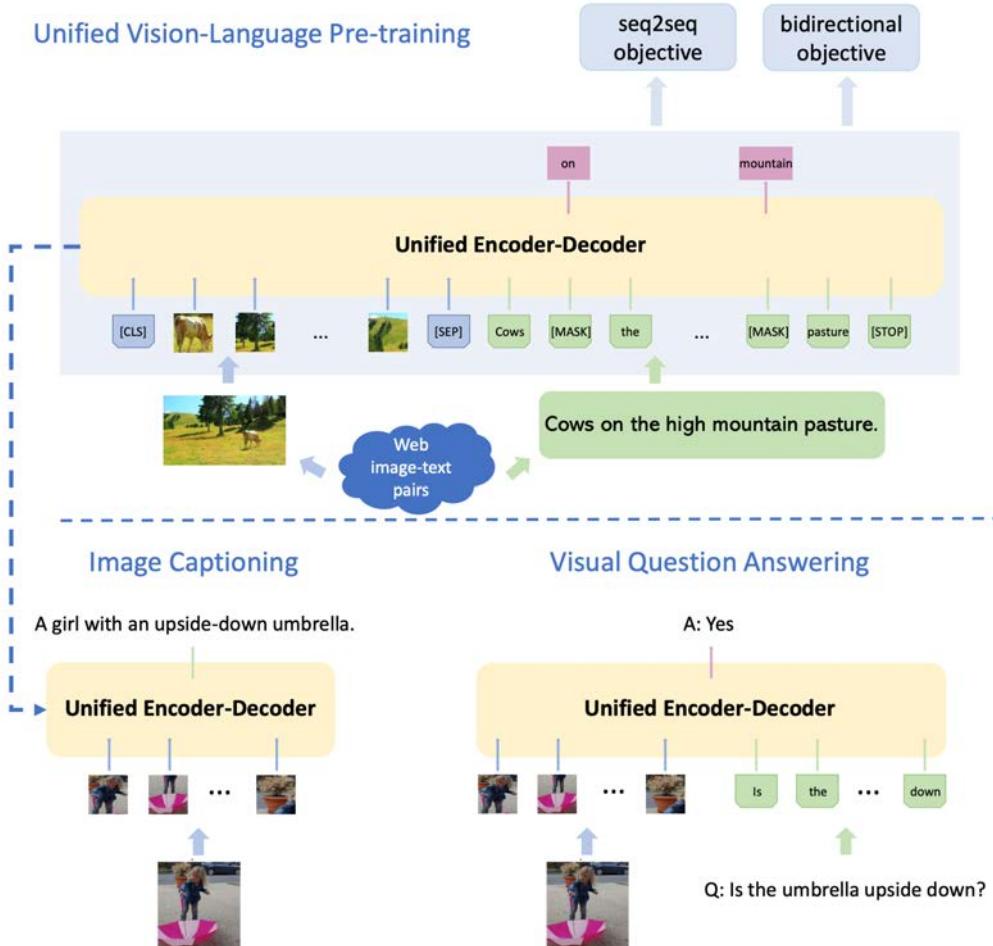


Girshick, "Fast R-CNN", ICCV 2015
Figure copyright Ross Girshick, 2015. Reproduced with permission.

Karpathy and Fei-Fei, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015

Transfer learning is pervasive!

It's the norm, not the exception



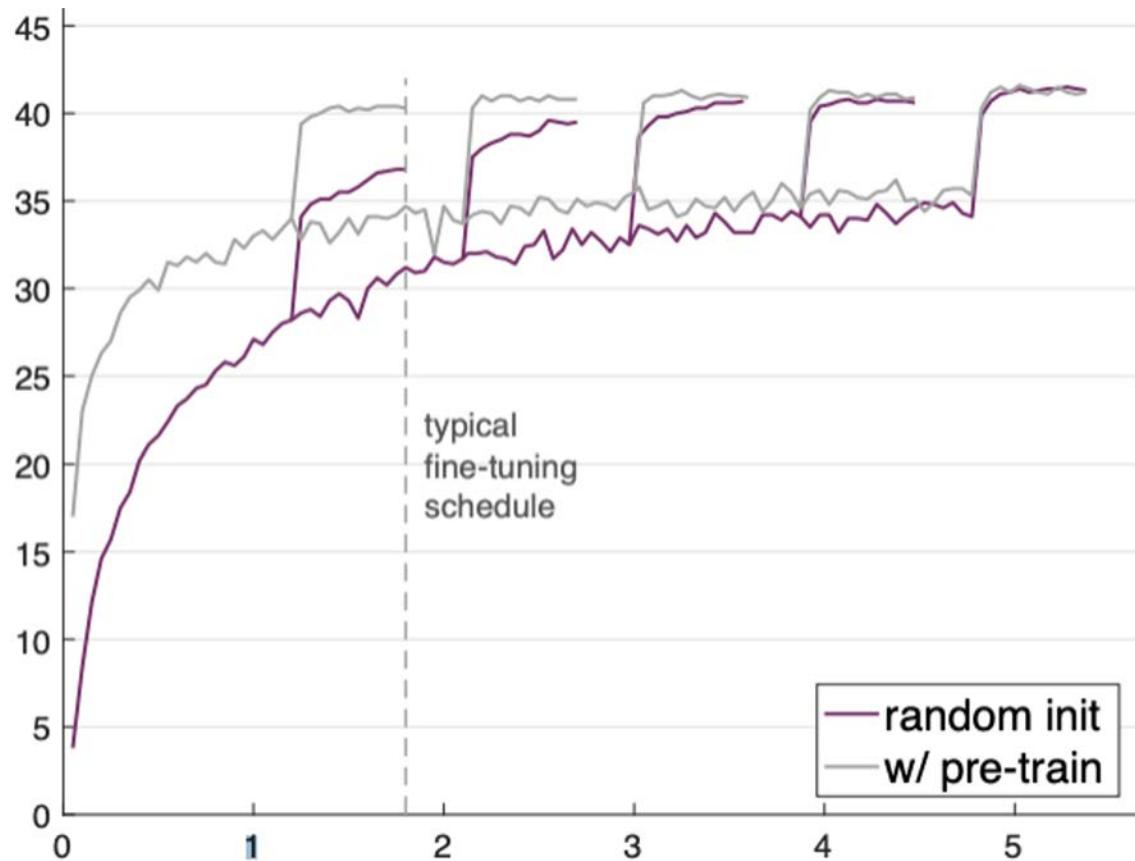
1. Train CNN on ImageNet
2. Fine-Tune (1) for object detection on Visual Genome
3. Train BERT language model on lots of text
4. Combine (2) and (3), train for joint image / language modeling
5. Fine-tune (5) for image captioning, visual question answering, etc.

Zhou et al, "Unified Vision-Language Pre-Training for Image Captioning and VQA", arXiv 2019

Transfer learning is pervasive!

Some very recent results have questioned it

COCO object detection



Training from scratch can
work as well as pretraining
on ImageNet!

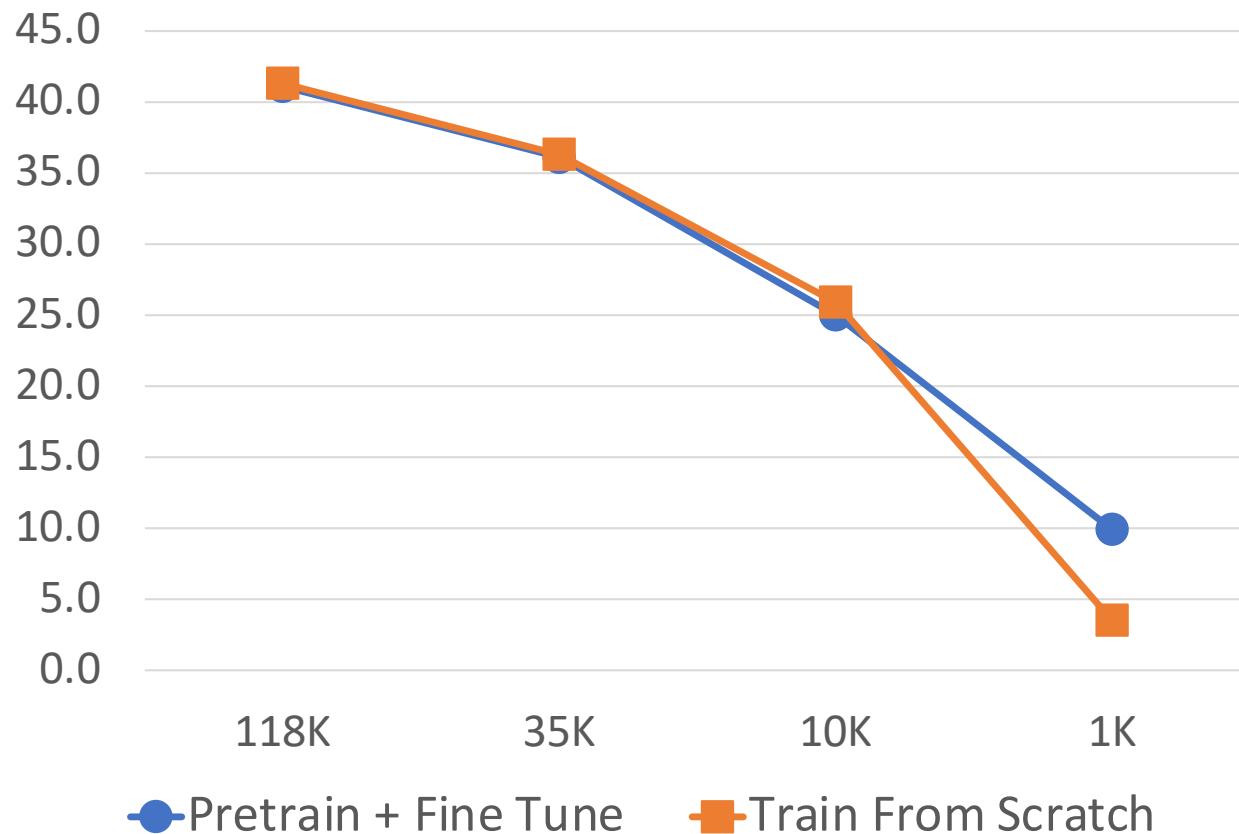
... If you train for 3x as long

He et al, "Rethinking ImageNet Pre-Training", ICCV 2019

Transfer learning is pervasive!

Some very recent results have questioned it

COCO object detection



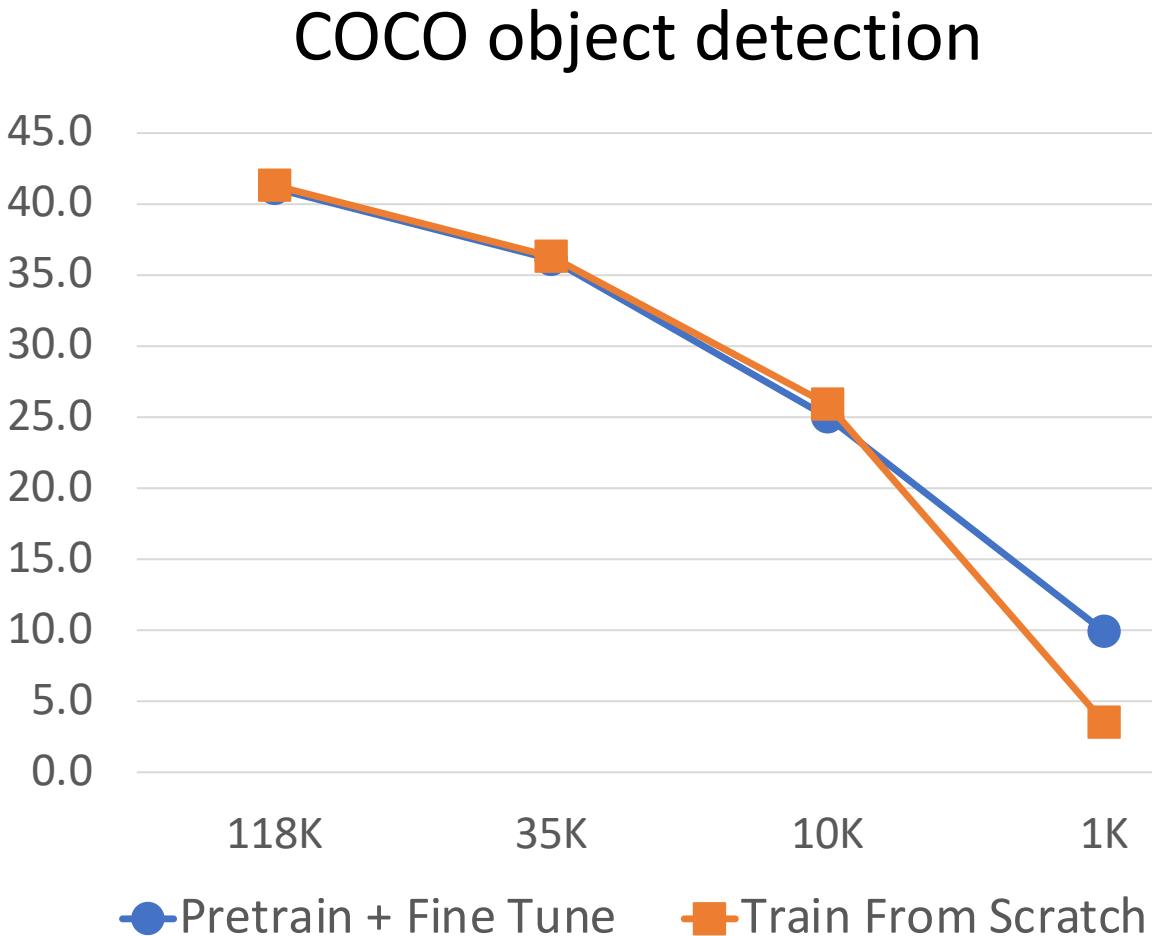
Pretraining + Finetuning beats training from scratch when dataset size is very small

Collecting more data is more effective than pretraining

He et al, "Rethinking ImageNet Pre-Training", ICCV 2019

Transfer learning is pervasive!

Some very recent results have questioned it



My current view on transfer learning:

- Pretrain+finetune makes your training faster, so practically very useful
- Training from scratch works well once you have enough data
- Lots of work left to be done

He et al, "Rethinking ImageNet Pre-Training", ICCV 2019

Distributed Training

Beyond individual devices



Cloud TPU v2

180 TFLOPs

64 GB HBM memory

\$4.50 / hour

(free on Colab!)



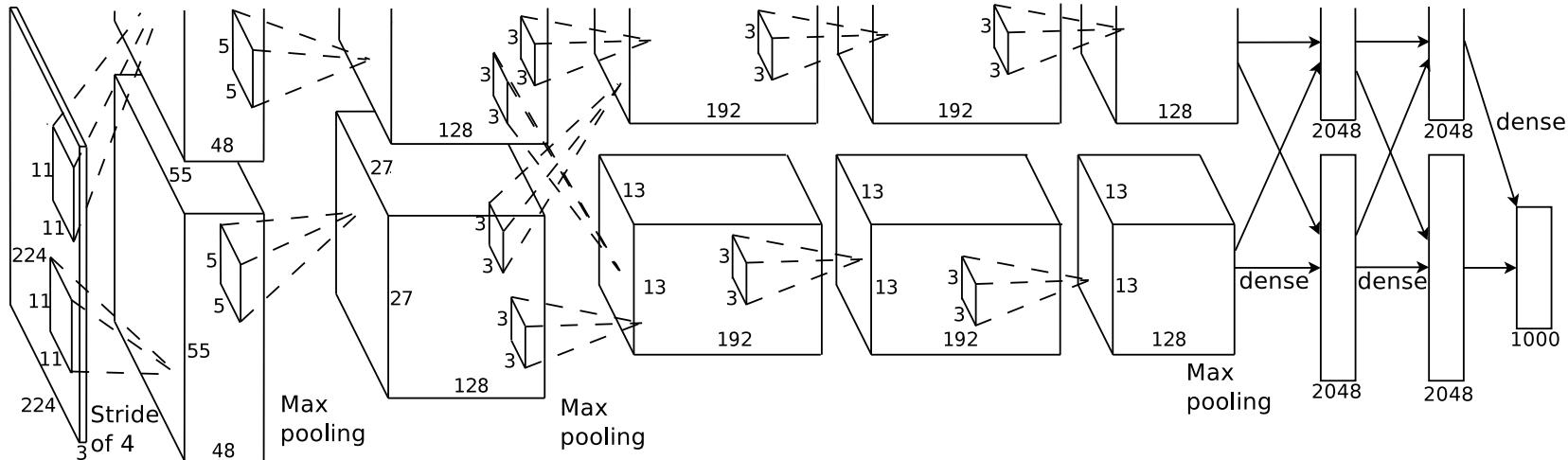
Cloud TPU v2 Pod

64 TPU-v2

11.5 PFLOPs

\$384 / hour

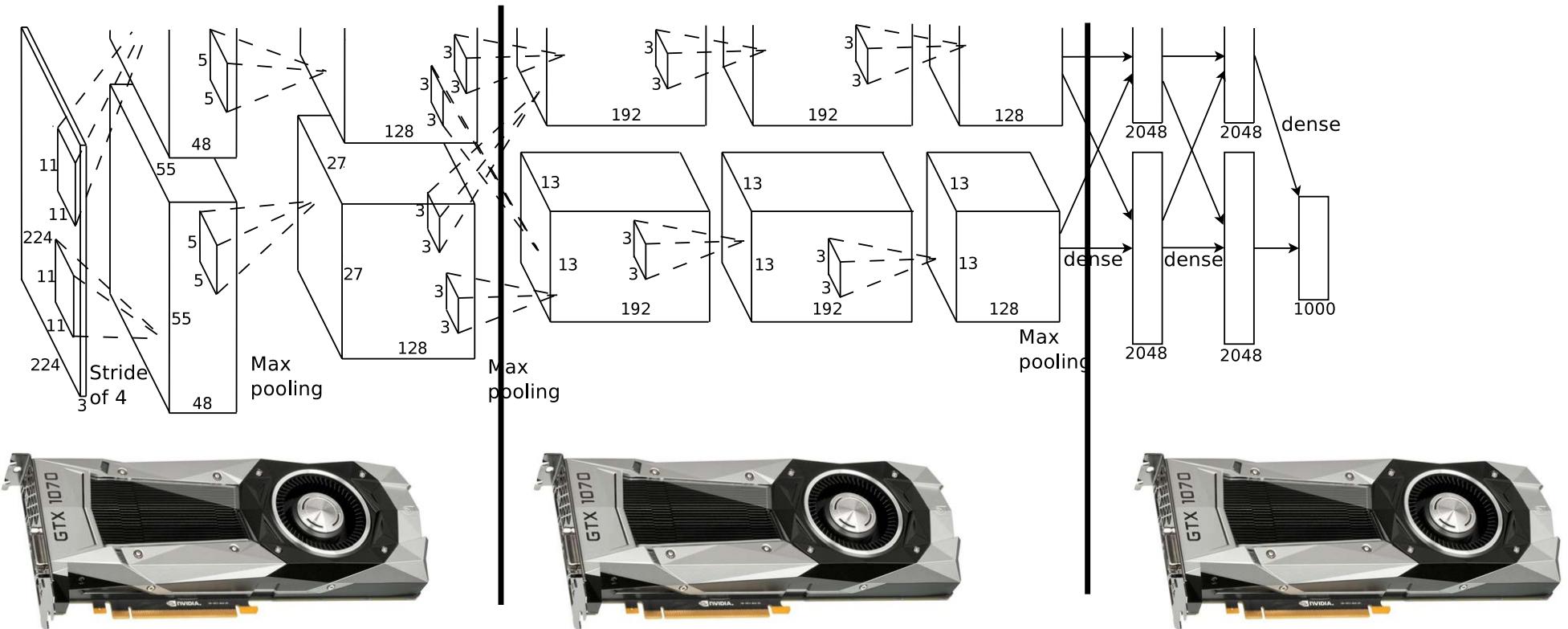
Model Parallelism: Split Model Across GPUs



[This image](#) is in the public domain

Model Parallelism: Split Model Across GPUs

Idea #1: Run different layers on different GPUs

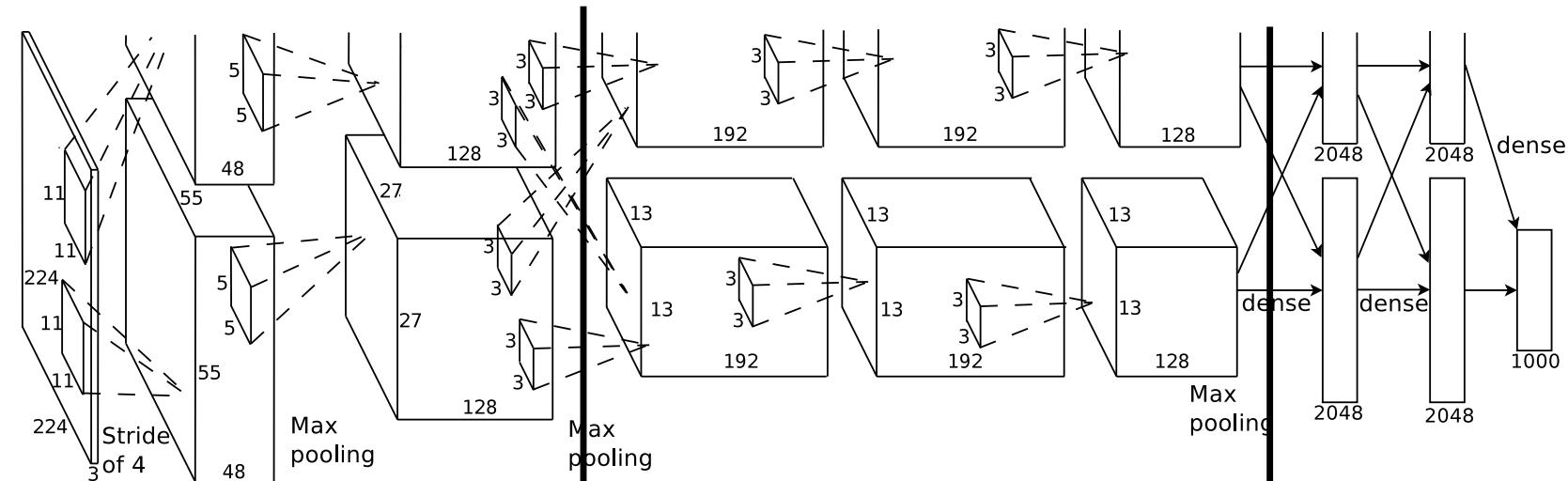


This image is in the public domain

Model Parallelism: Split Model Across GPUs

Idea #1: Run different layers on different GPUs

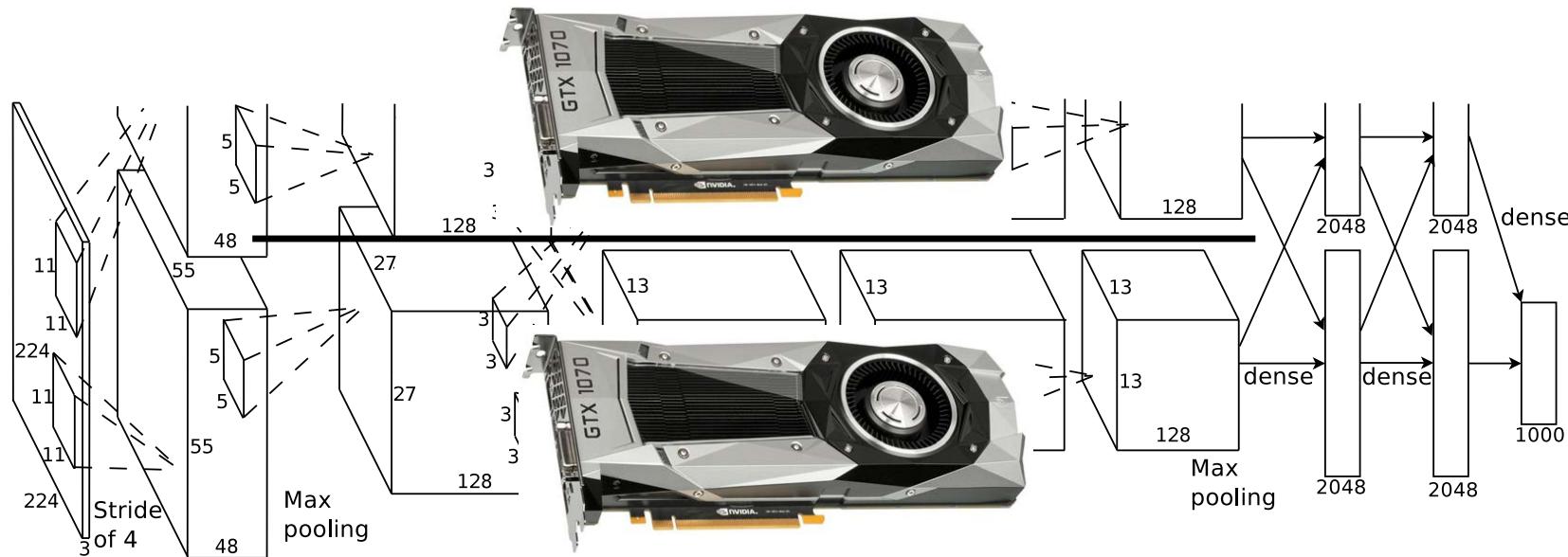
Problem: GPUs spend lots of time waiting



This image is in the public domain

Model Parallelism: Split Model Across GPUs

Idea #2: Run parallel branches of model on different GPUs



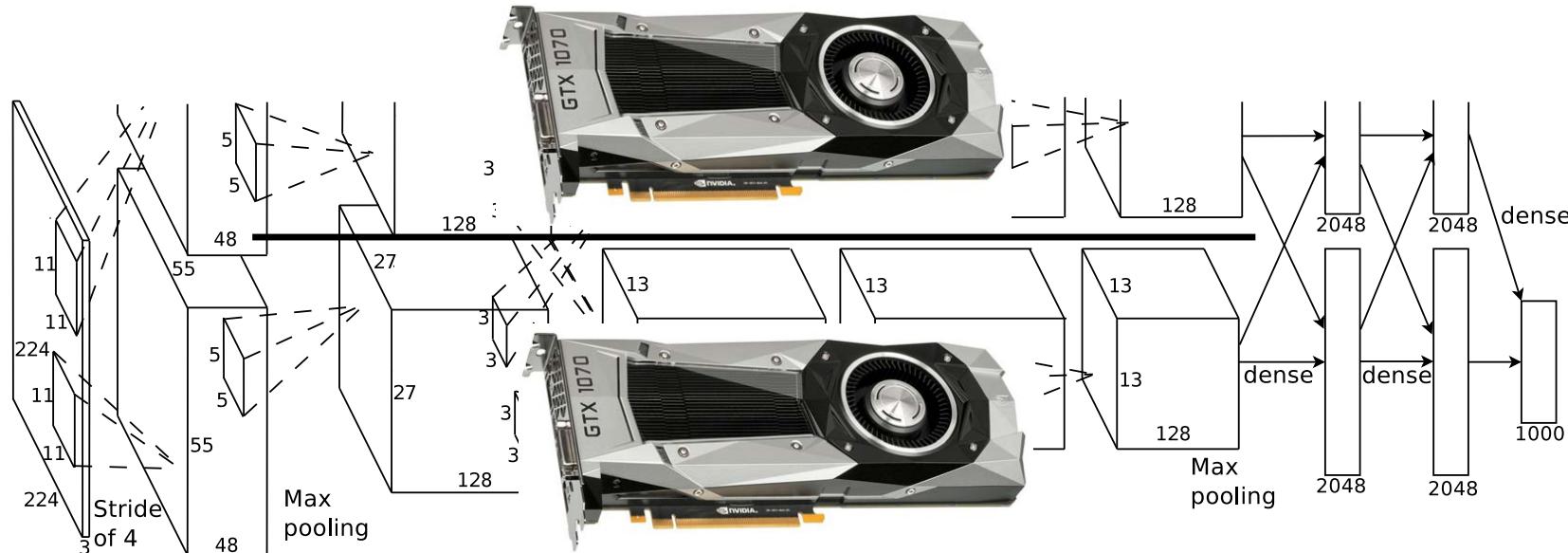
[This image](#) is in the public domain

Model Parallelism: Split Model Across GPUs

Idea #2: Run parallel branches of model on different GPUs

Problem: Synchronizing across GPUs is expensive;

Need to communicate **activations** and **grad activations**



This image is in the public domain

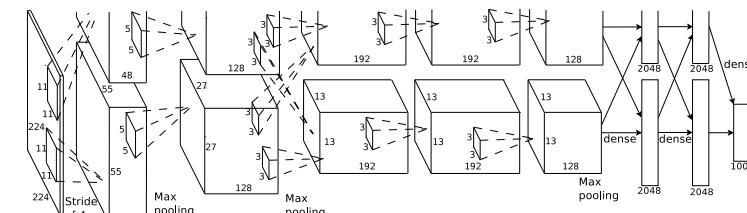
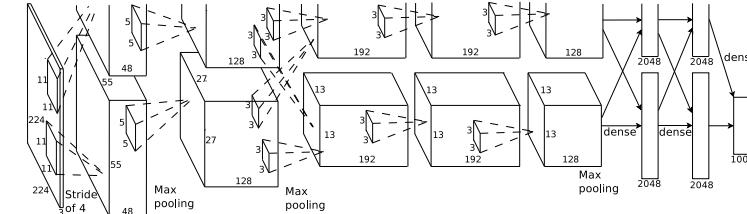
Data Parallelism: Copy Model on each GPU, split data

Batch of
N images

$N / 2$
images

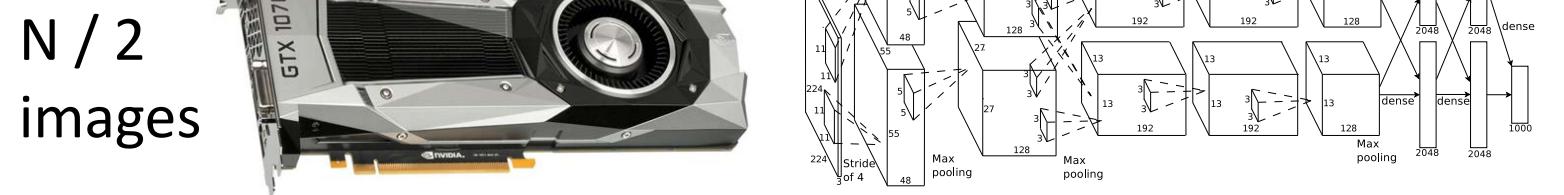
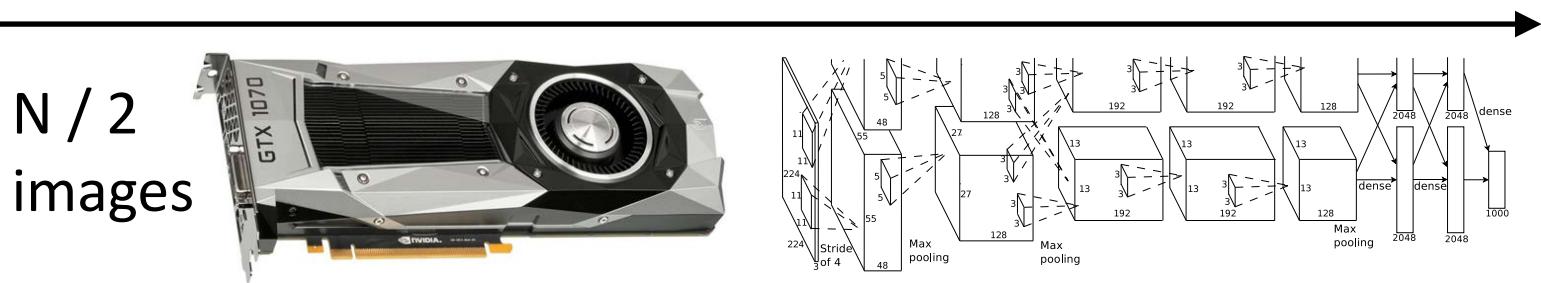


$N / 2$
images

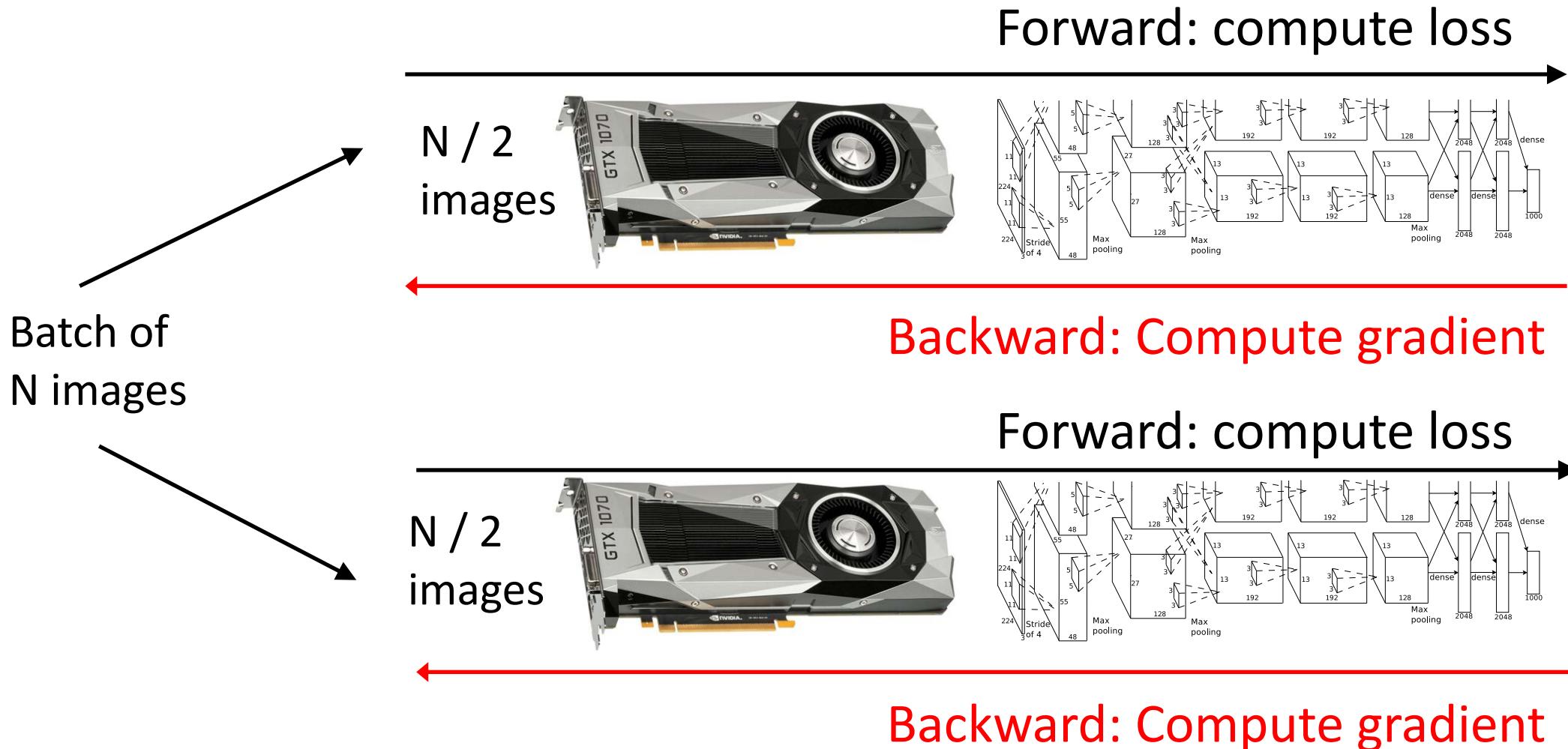


Data Parallelism: Copy Model on each GPU, split data

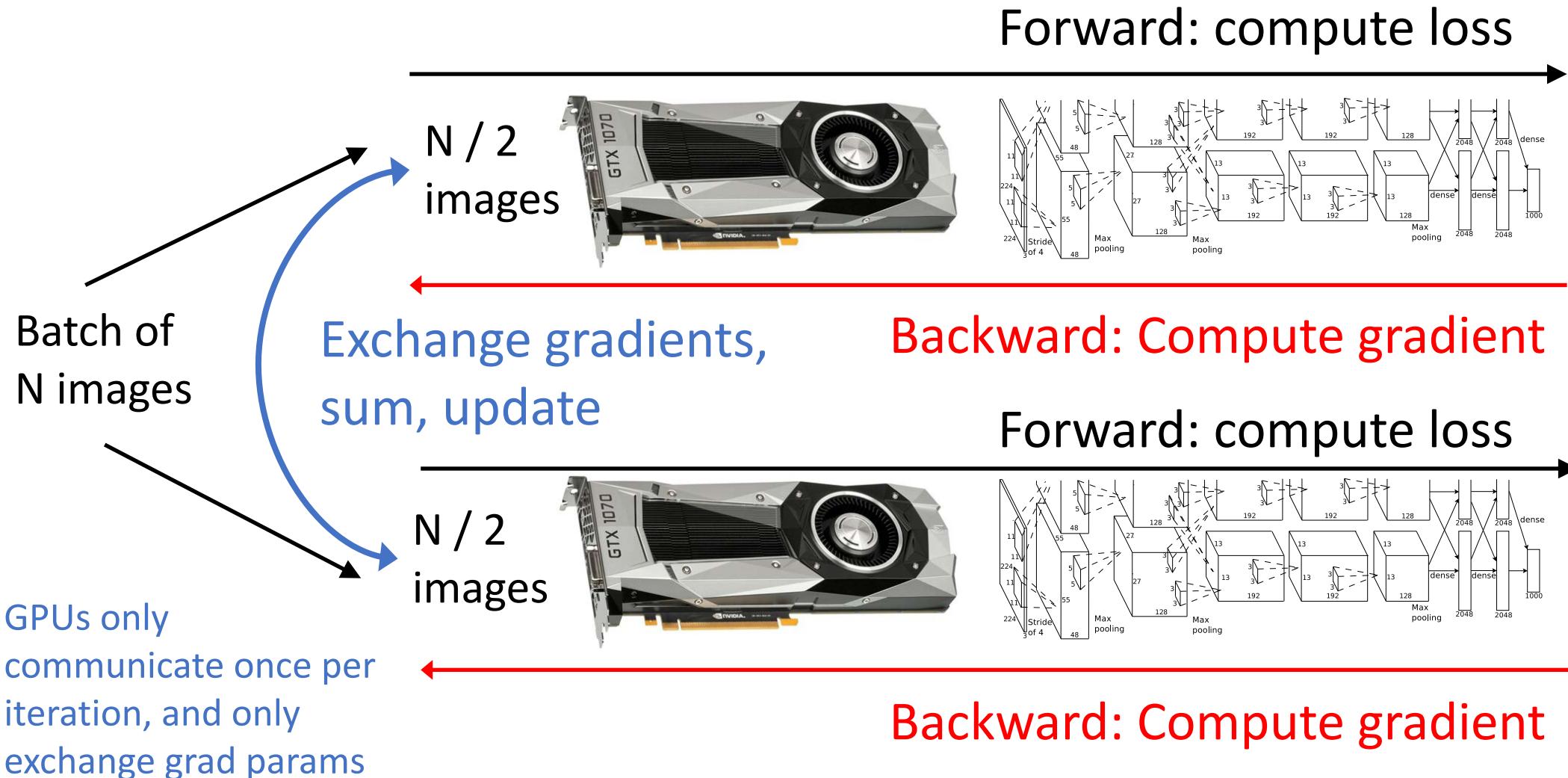
Batch of
N images



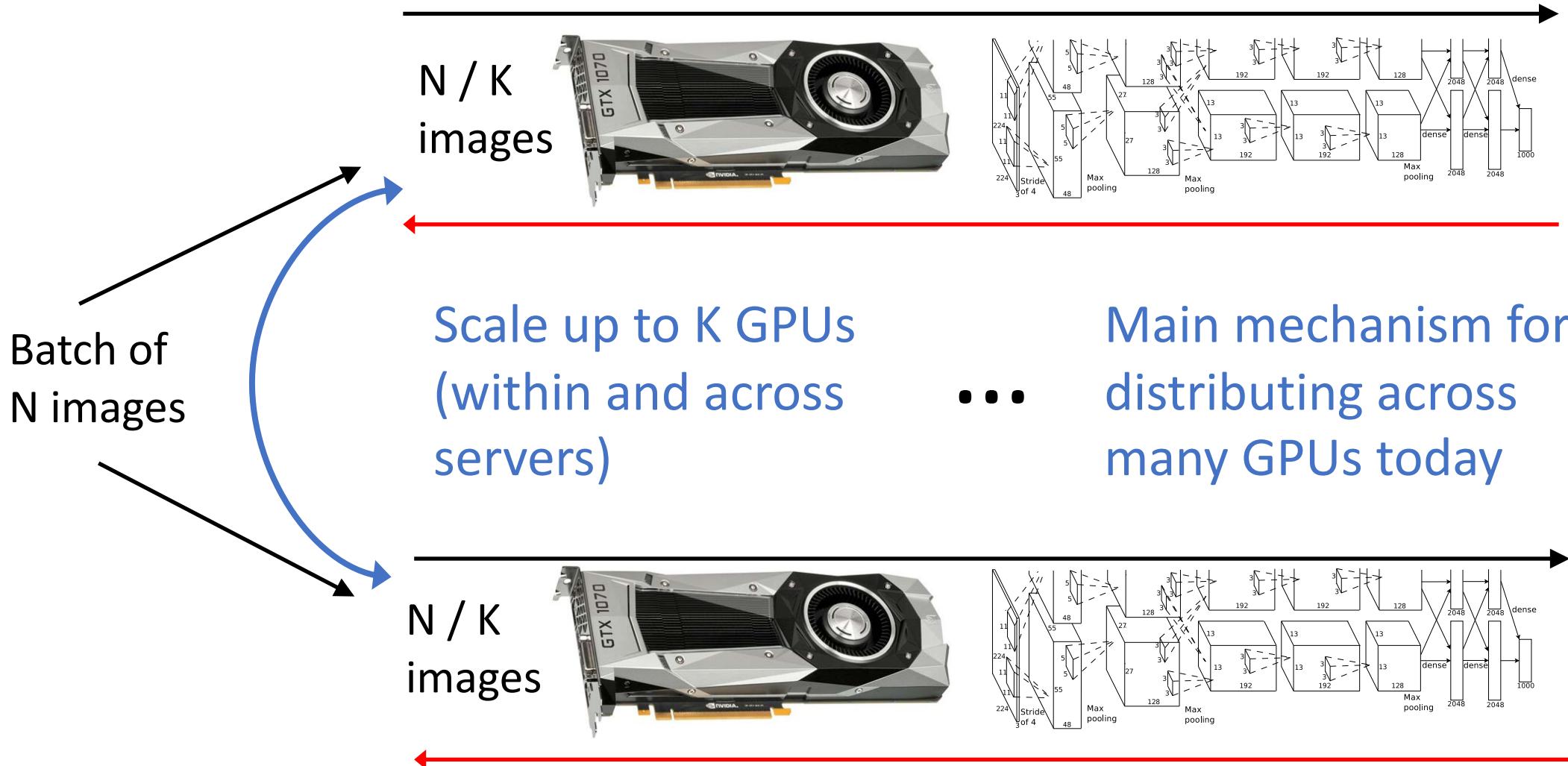
Data Parallelism: Copy Model on each GPU, split data



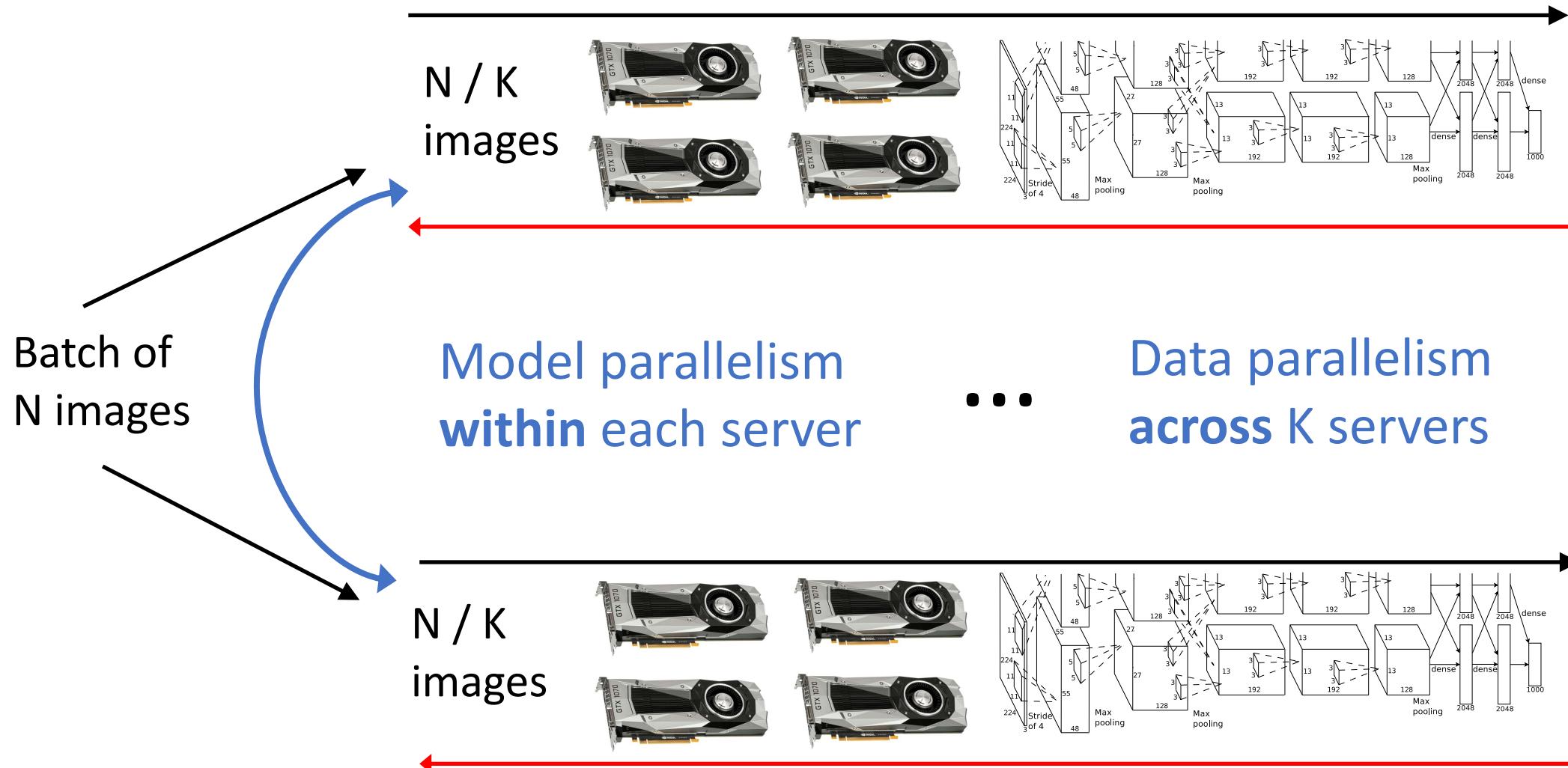
Data Parallelism: Copy Model on each GPU, split data



Data Parallelism: Copy Model on each GPU, split data



Mixed Model + Data Parallelism



Example: <https://devblogs.nvidia.com/training-bert-with-gpus/>

Mixed Model + Data Parallelism

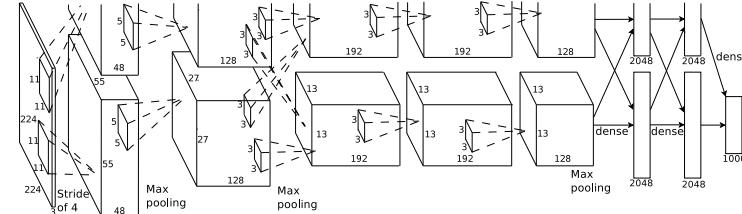
Problem: Need to train with very large minibatches!

Batch of N images

N / K images

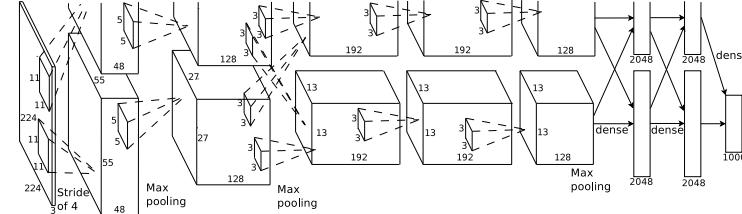


Model parallelism
within each server



Data parallelism
across K servers

N / K images

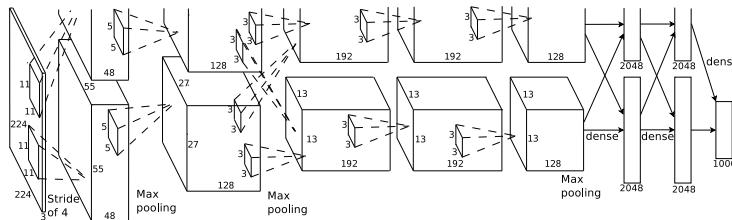


Example: <https://devblogs.nvidia.com/training-bert-with-gpus/>

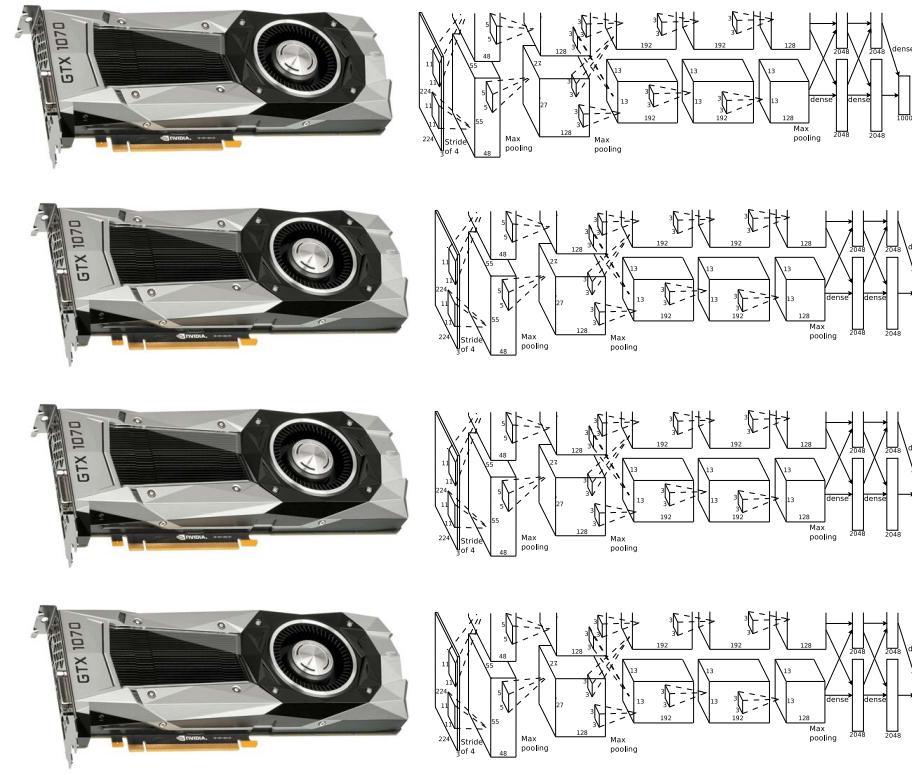
Large-Batch Training

Large-Batch Training

Suppose we can train a good model with one GPU

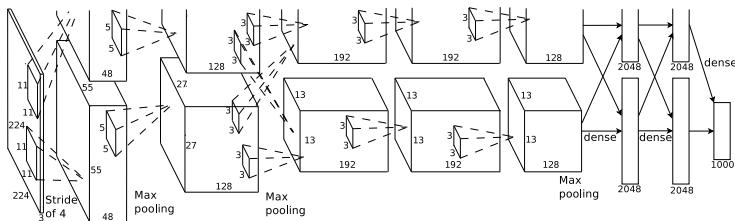


How to scale up to data-parallel training on K GPUs?



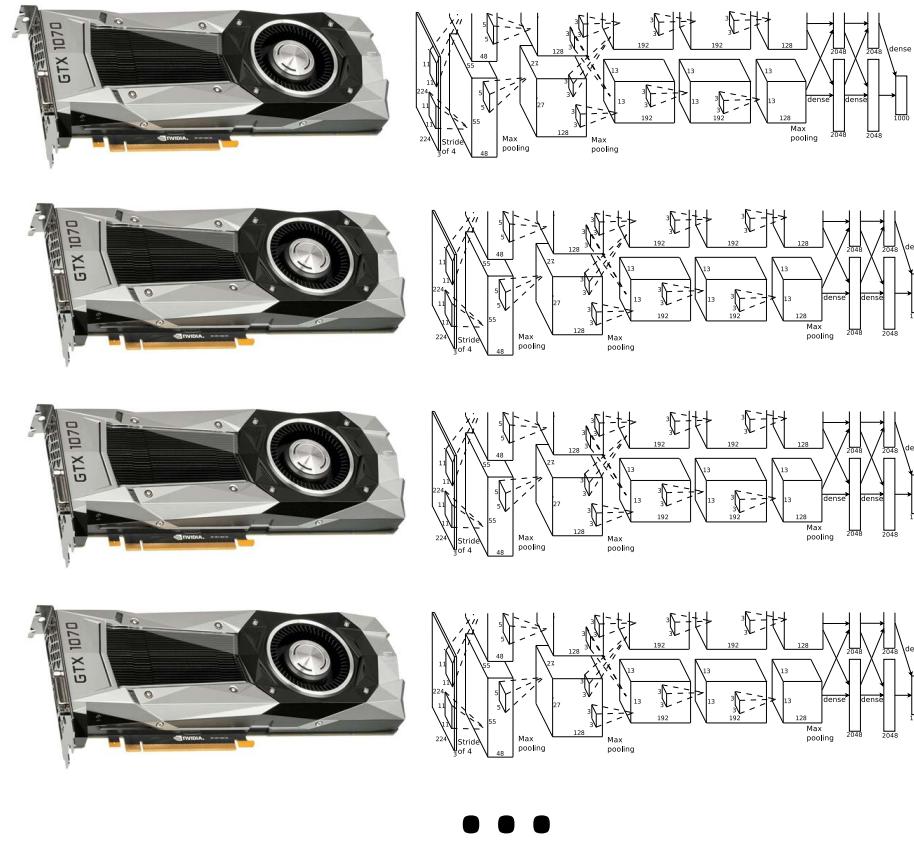
Large-Batch Training

Suppose we can train a good model with one GPU



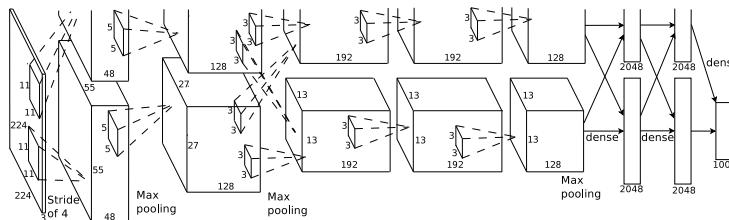
Goal: Train for same number of epochs, but use larger minibatches.
We want model to train K times faster!

How to scale up to data-parallel training on K GPUs?

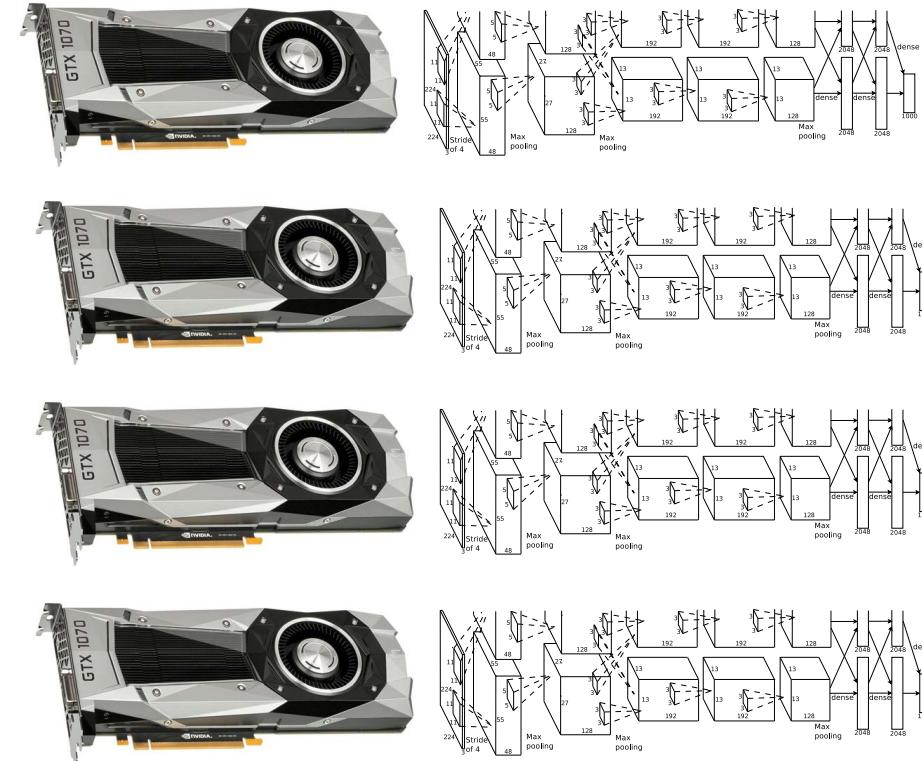


Large-Batch Training: Scale Learning Rates

Single-GPU model:
batch size N , learning rate α

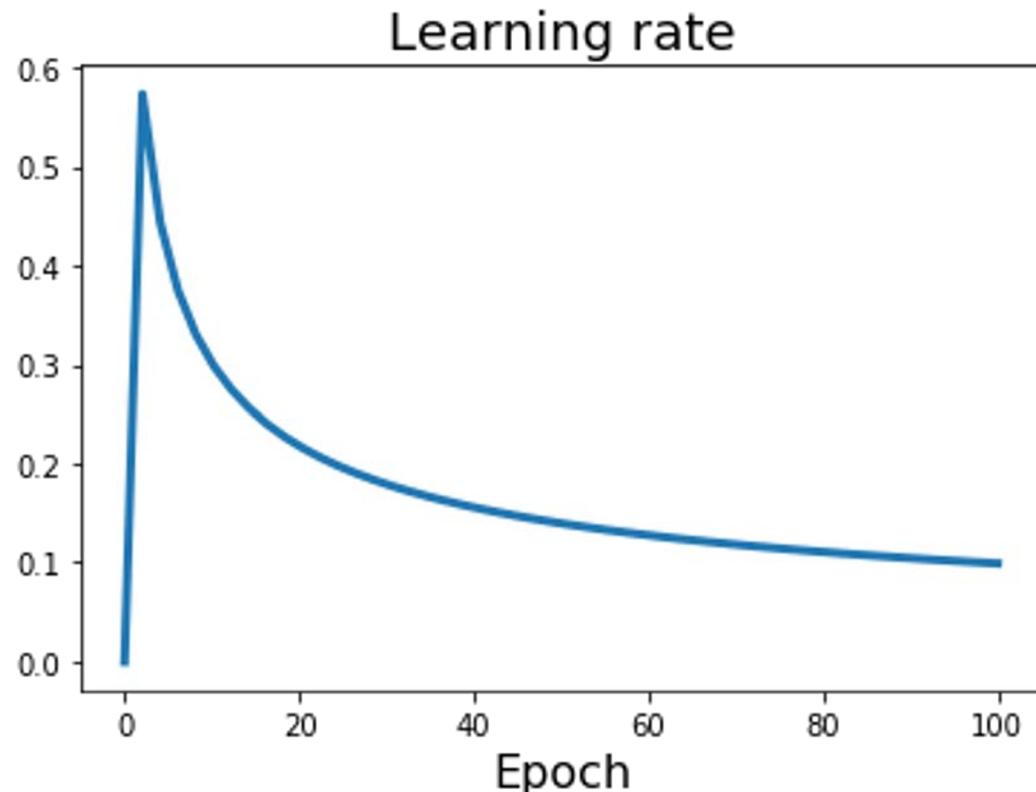


K-GPU model:
batch size KN , learning rate $K\alpha$



Alex Krizhevsky, "One weird trick for parallelizing convolutional neural networks", arXiv 2014
Goyal et al, "Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour", arXiv 2017

Large-Batch Training: Learning Rate Warmup



High initial learning rates can make loss explode; linearly **increasing** learning rate from 0 over the first ~5000 iterations can prevent this

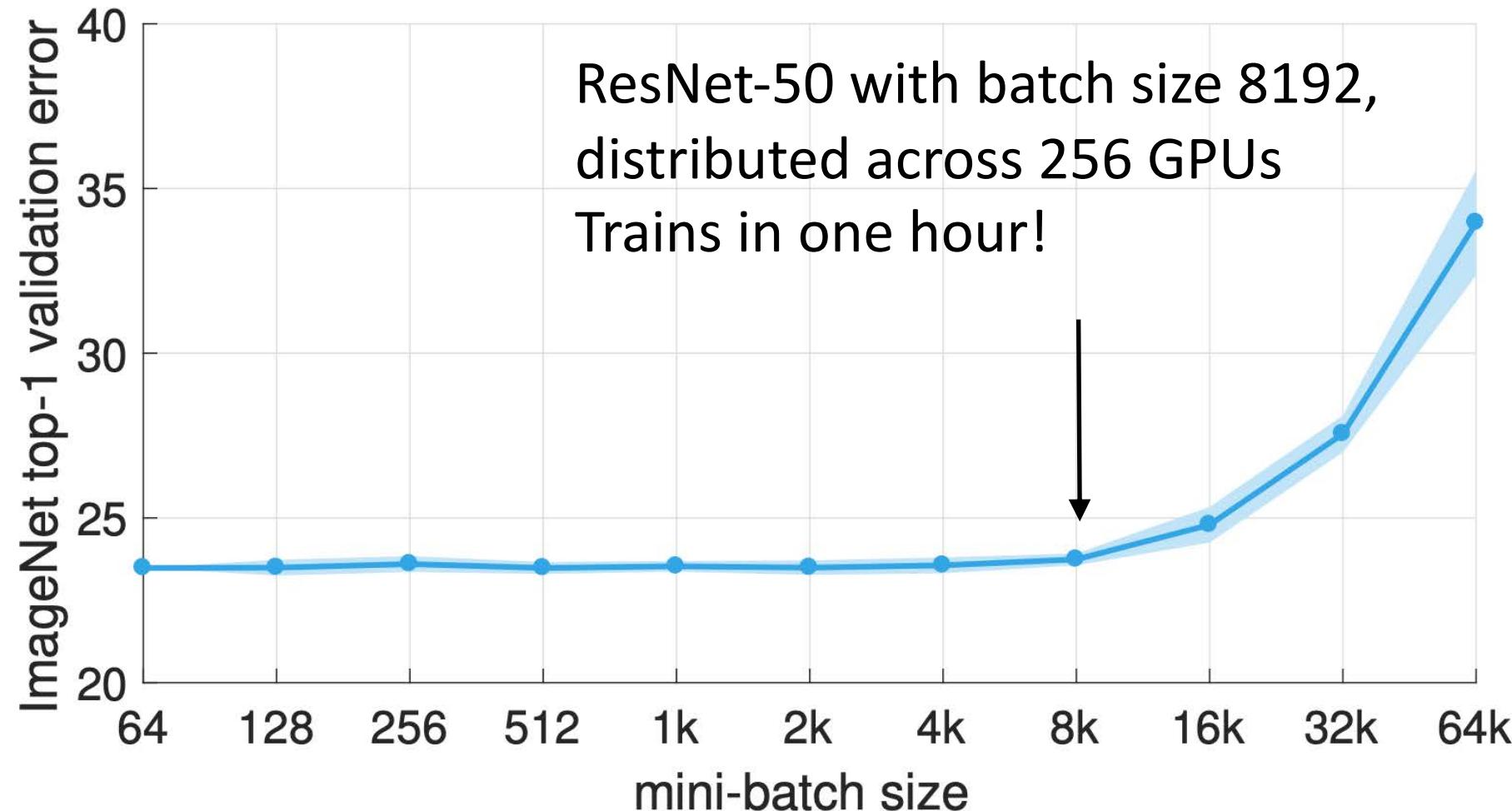
Large-Batch Training: Other Concerns

Be careful with **weight decay** and **momentum**, and **data shuffling**

For Batch Normalization, only normalize **within a GPU**

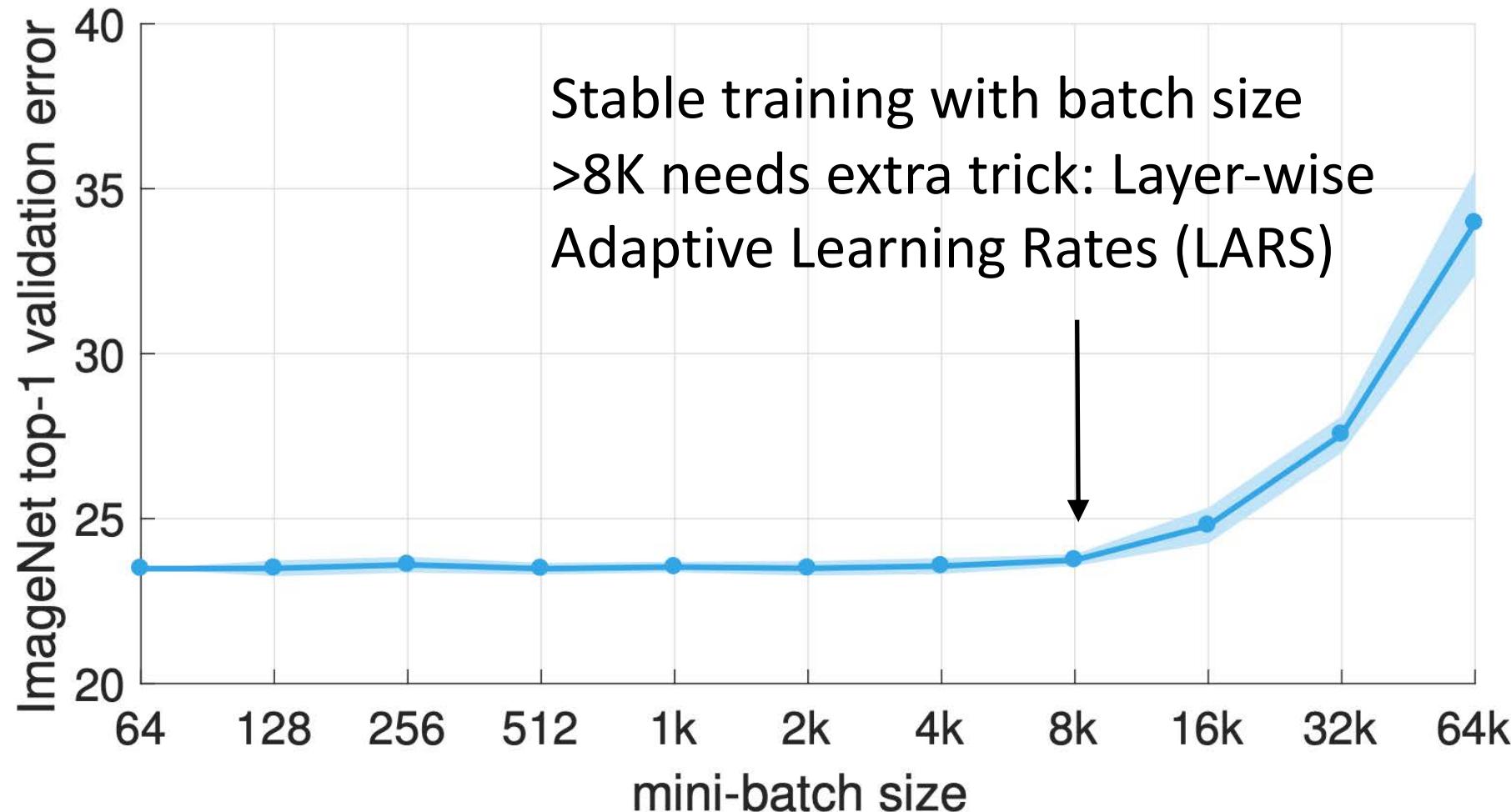
Goyal et al, “Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour”, arXiv 2017

Large-Batch Training: ImageNet in One Hour!



Goyal et al, "Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour", arXiv 2017

Large-Batch Training: ImageNet in One Hour!



Goyal et al, "Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour", arXiv 2017

You et al, "Large Batch Training of Convolutional Networks", arXiv 2017

Large-Batch Training

Goyal et al, “Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour”, 2017

Batch size: 8192; 256 P100 GPUs; 1 hour

Codreanu et al, “Achieving deep learning training in less than 40 minutes on imagenet-1k”, 2017

Batch size: 12288; 768 Knight’s Landing devices; 39 minutes

You et al, “ImageNet training in minutes”, 2017

Batch size: 16000; 1600 Xeon CPUs; 31 minutes

Akiba et al, “Extremely Large Minibatch SGD: Training ResNet-50 on ImageNet in 15 Minutes”, 2017

Batch size: 32768; 1024 P100 GPUs; 15 minutes

MLPerf v0.7 (NVIDIA): 1840 A100 GPUs; **46 seconds**

MLPerf v0.7 (Google): TPU-v3-8192; Batch size 65536; **27 seconds!**



MLPerf

<https://mlperf.org/>

Recap

1. One time setup

Activation functions, data preprocessing, weight initialization, regularization

Last Time

2. Training dynamics

Learning rate schedules;
hyperparameter optimization

Today

3. After training

Model ensembles, transfer learning,
large-batch training

Next Time: Recurrent Neural Networks