

Lecture 11:

Training Neural Networks

(Part 2)

Reminder: A3

- Due Monday, October 14
- Remember to run the validation script!

Reminder: Midterm

- Monday, October 21 (two weeks from today!)
- Location: Chrysler 220 (NOT HERE!)

- Format:

- True / False, Multiple choice, short answer
- Emphasize concepts – you don't need to memorize AlexNet!
- Closed-book
- You can bring 1 page "cheat sheet" of handwritten notes
(standard 8.5" x 11" paper)

- Alternate exam times: Fill out this form: <https://forms.gle/uiMpHdg9752p27bd7>
 - Conflict with EECS 551
 - SSD accommodations
 - Conference travel for Michigan

Reminder: No class on Monday 10/14!
(Fall Study Break)

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

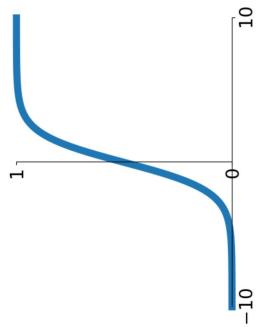
Last Time

Today

Last Time: Activation Functions

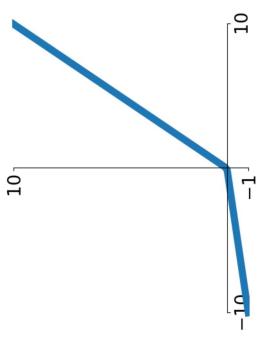
Sigmoid

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



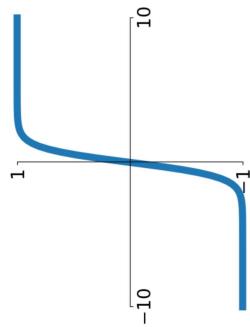
Leaky ReLU

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



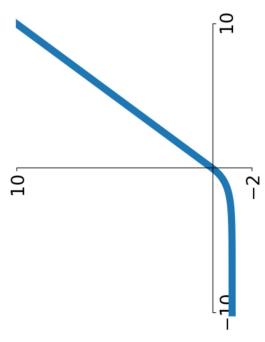
tanh

$$\tanh(x)$$



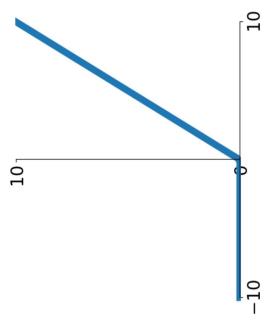
Maxout

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

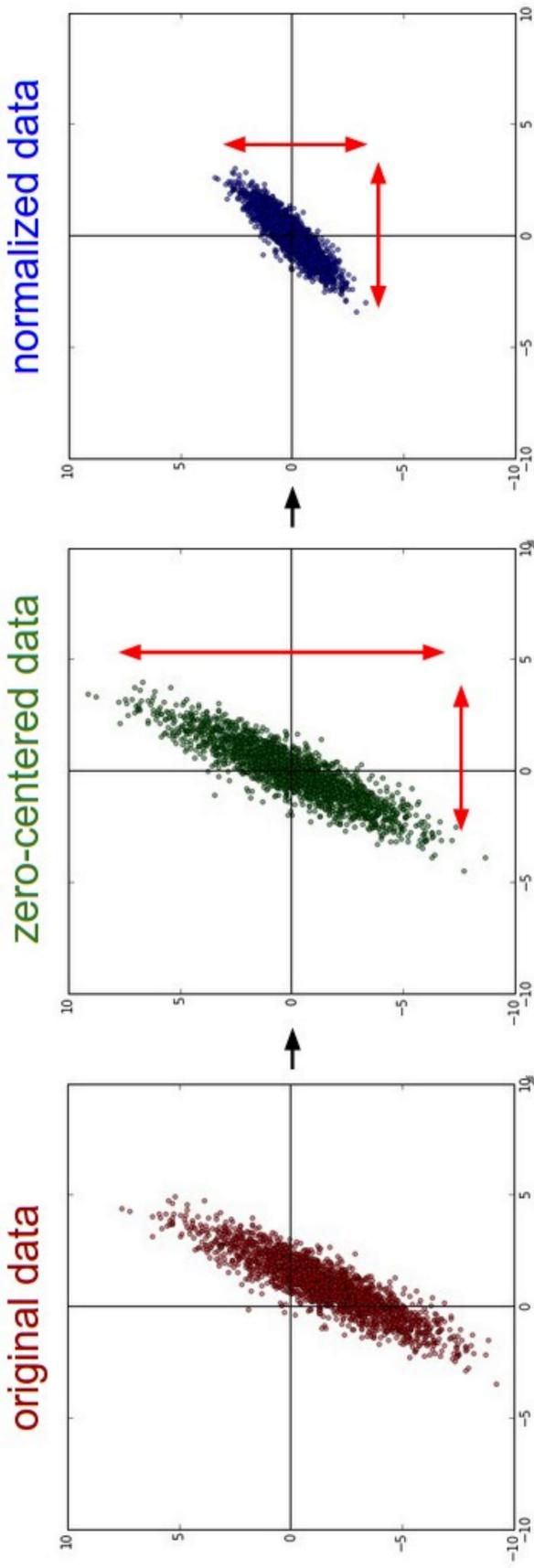


ReLU

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

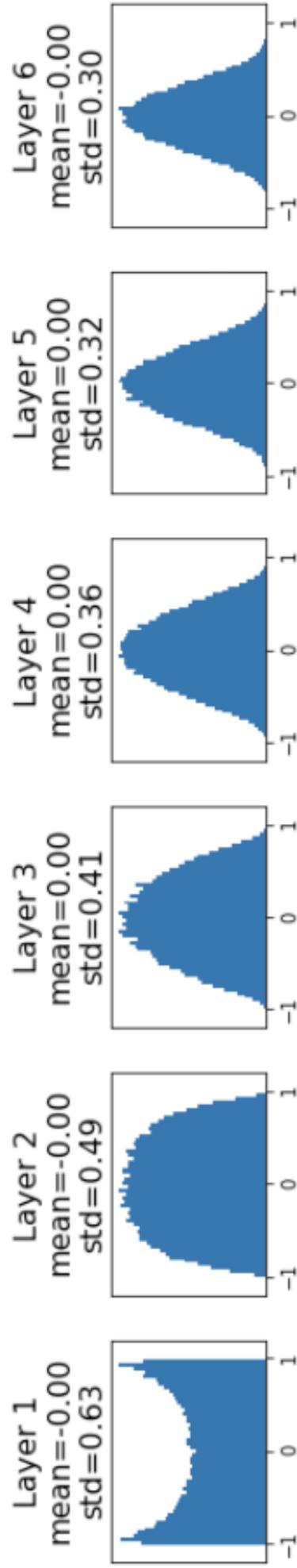


Last Time: Data Preprocessing



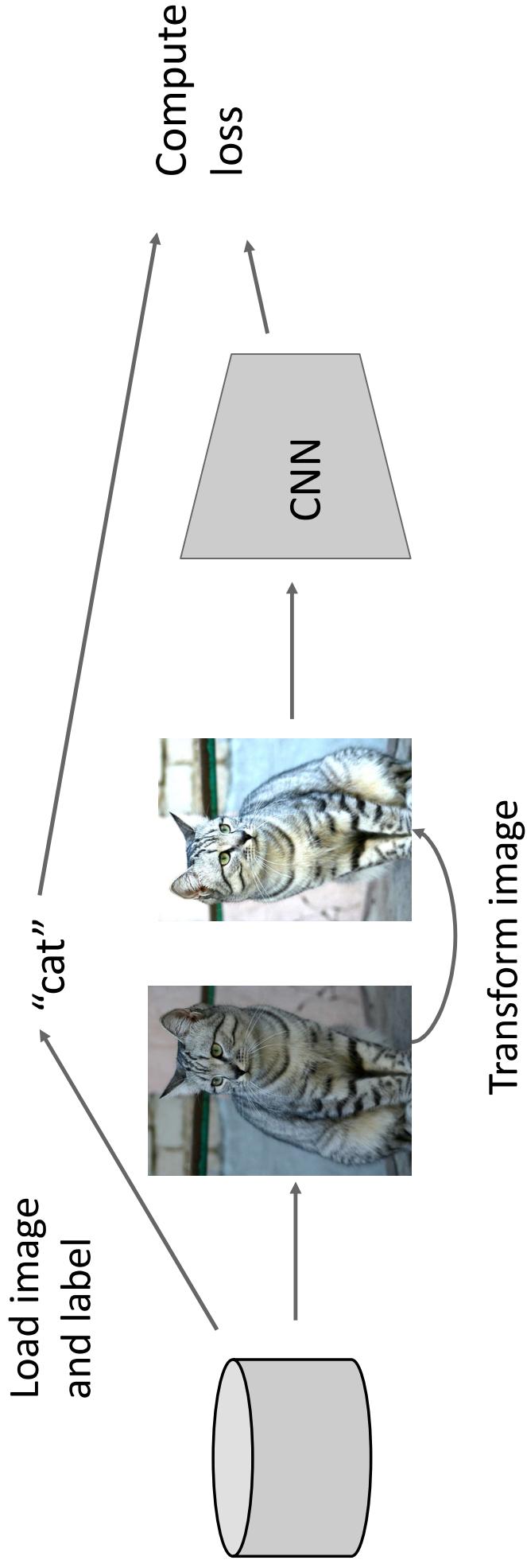
Last Time: Weight Initialization

```
dims = [ 4096 ] * 7      "Xavier" initialization:  
hs = []  
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)
```



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

Last Time: Data Augmentation



Last Time: Regularization

Training: Add randomness

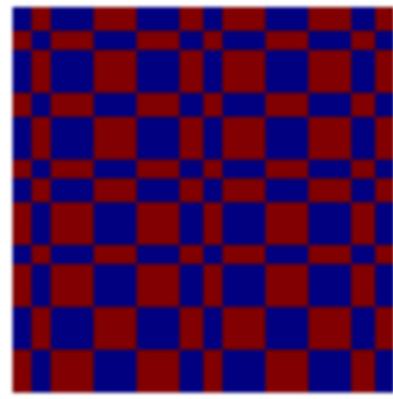
Testing: Marginalize out randomness

Examples:

Batch Normalization

Data Augmentation

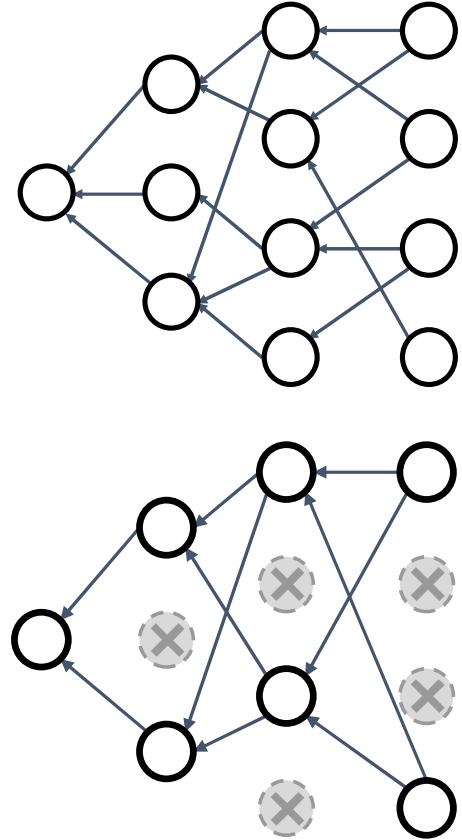
Fractional pooling



Cutout

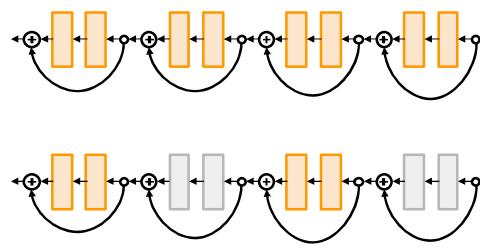


Dropout



DropConnect

Stochastic Depth



Mixup



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

Today

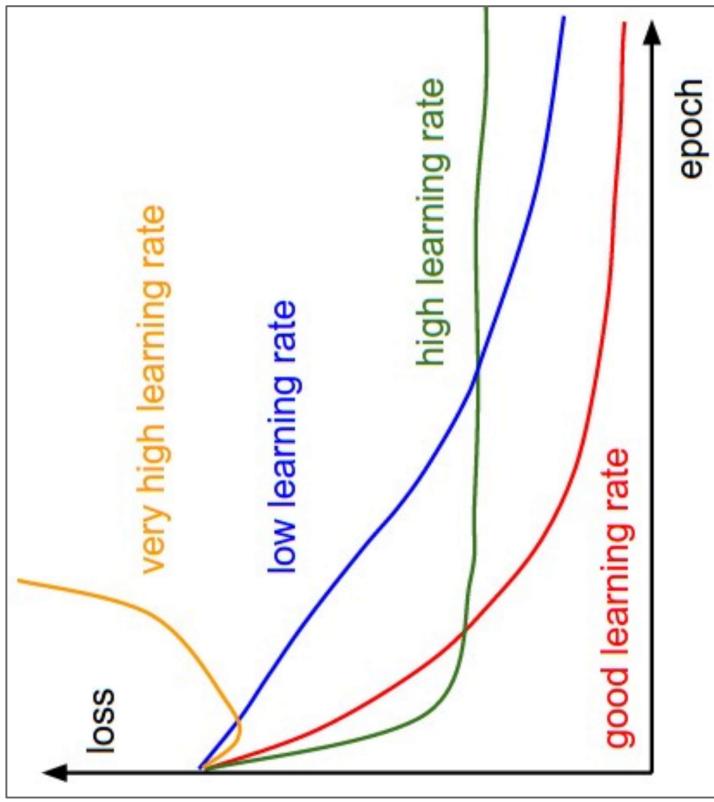
Learning Rate Schedules

Justin Johnson

Lecture 11 - 12

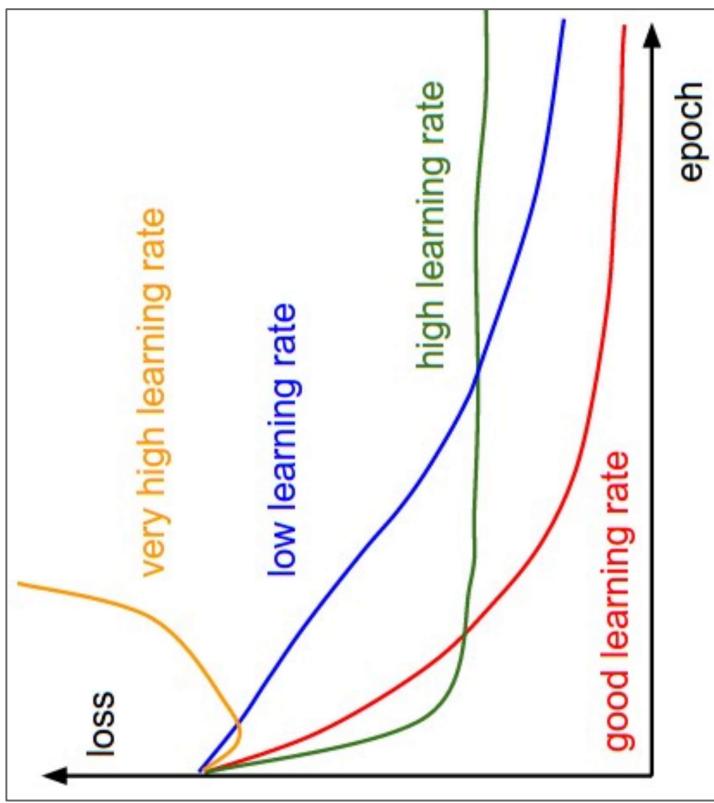
October 9, 2019

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.

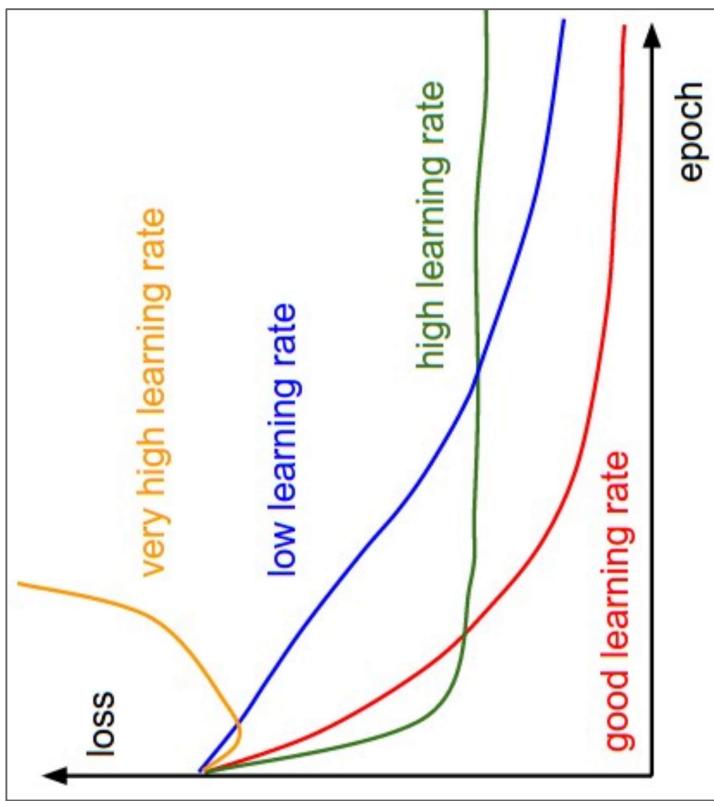
Q: Which one of these learning rates
is best to use?



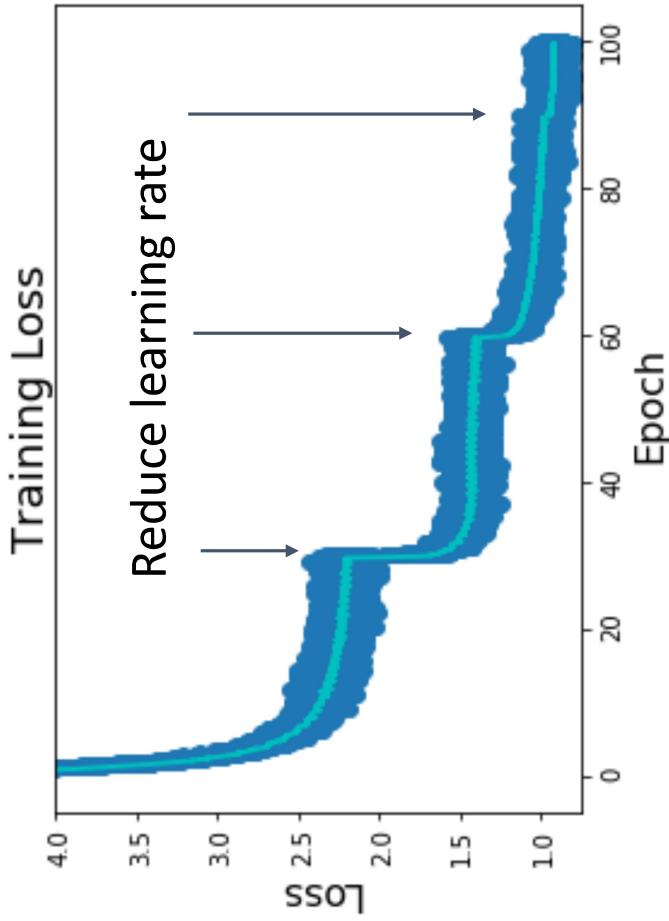
SGD, SGD+Momentum, Adagrad, RMSProp, Adam
all have **learning rate** as a hyperparameter.

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

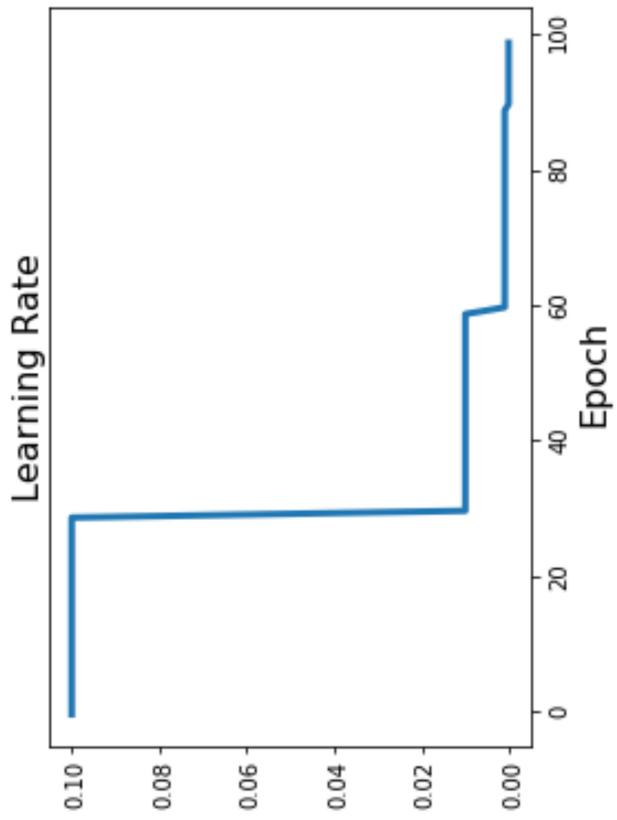
A: All of them! Start with large
learning rate and decay over time



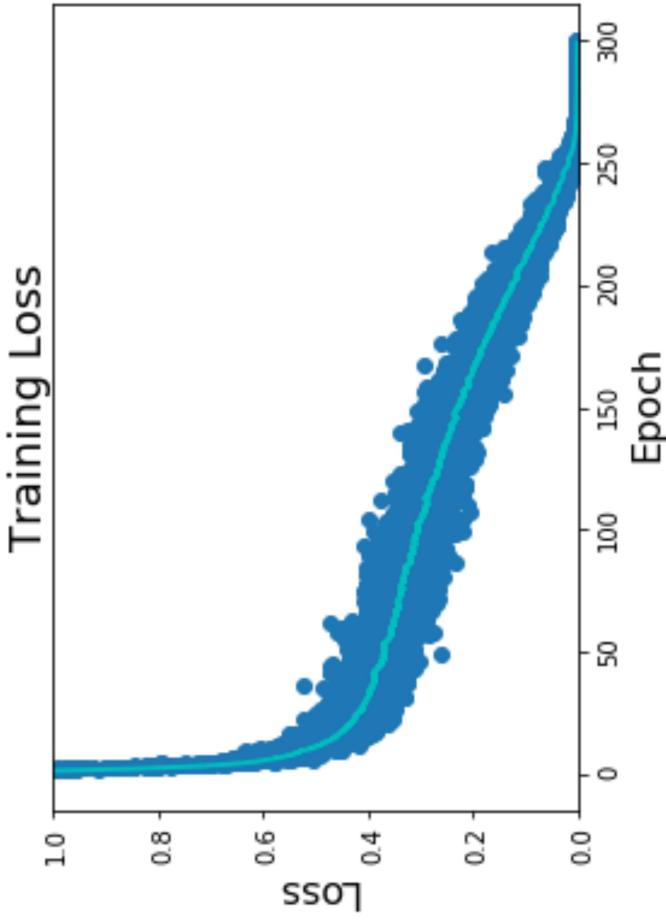
Learning Rate Decay: Step



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

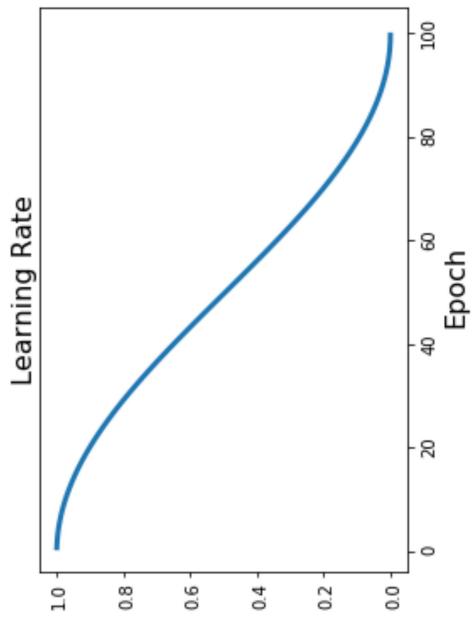


Learning Rate Decay: Cosine



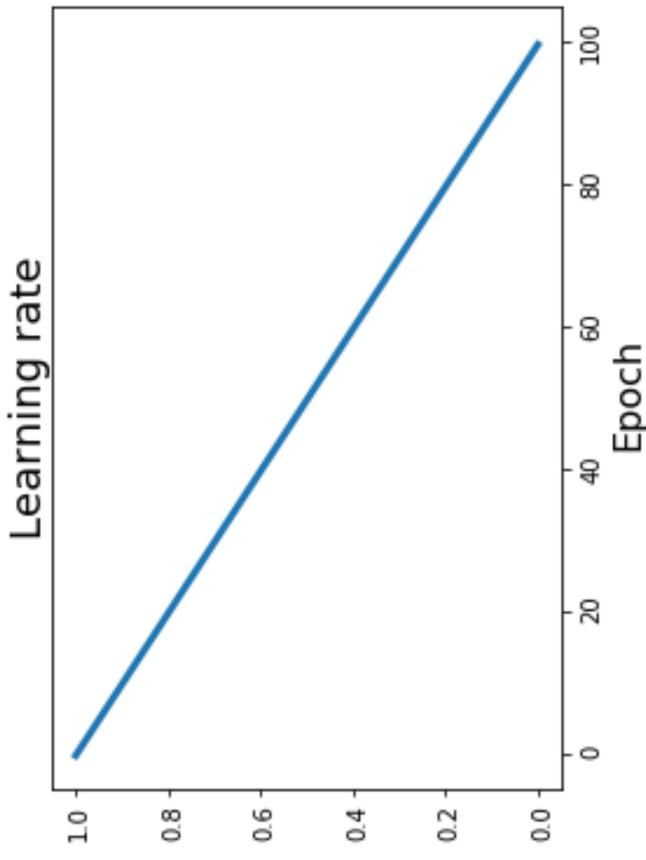
Step: Reduce learning rate at a few fixed points.
E.g. for ResNets, multiply LR by 0.1 after epochs
30, 60, and 90.

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



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



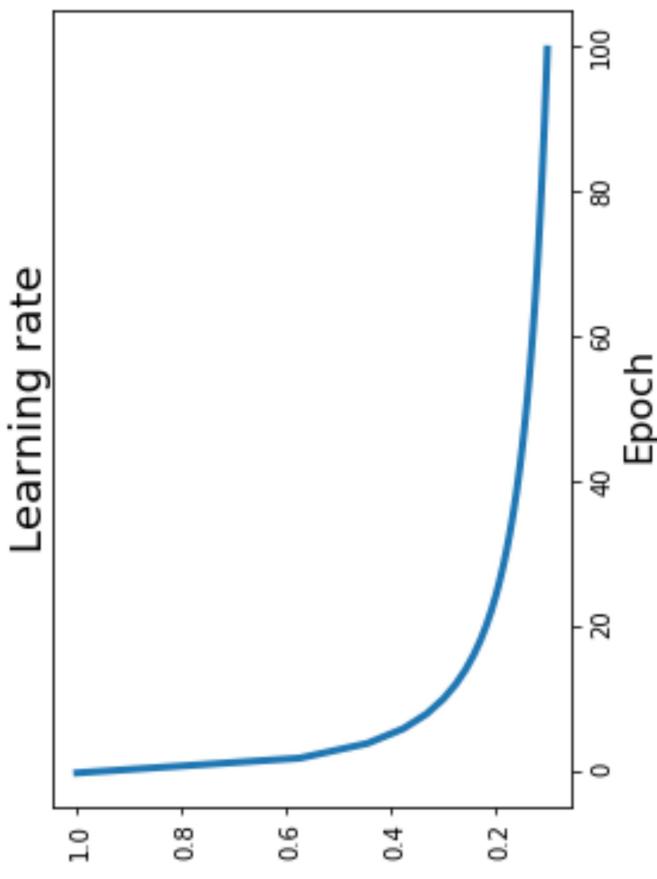
Step: Reduce learning rate at a few fixed points.
E.g. for ResNets, multiply LR by 0.1 after epochs 30, 60, and 90.

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

$$\text{Linear: } \alpha_t = \alpha_0(1 - t/T)$$

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



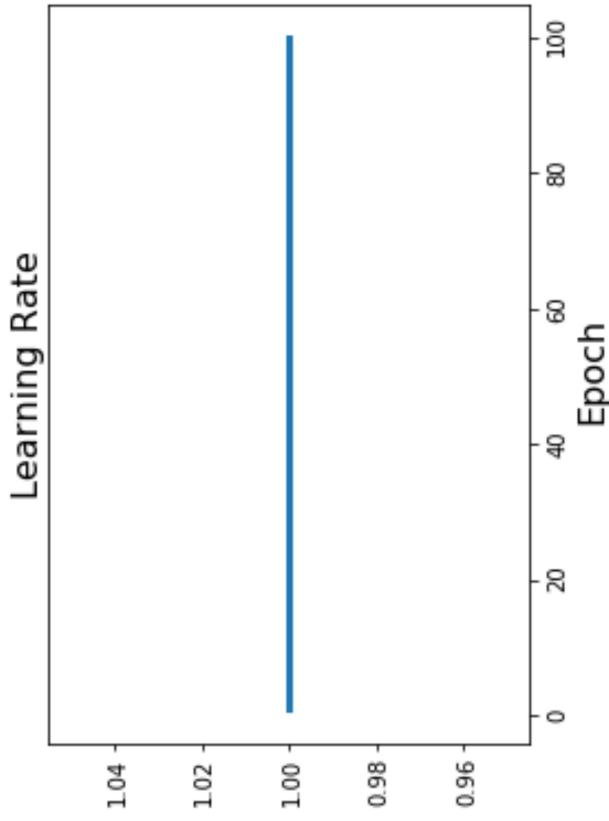
Step: Reduce learning rate at a few fixed points.
E.g. for ResNets, multiply LR by 0.1 after epochs 30, 60, and 90.

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

$$\text{Linear: } \alpha_t = \alpha_0(1 - t/T)$$

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

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.

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

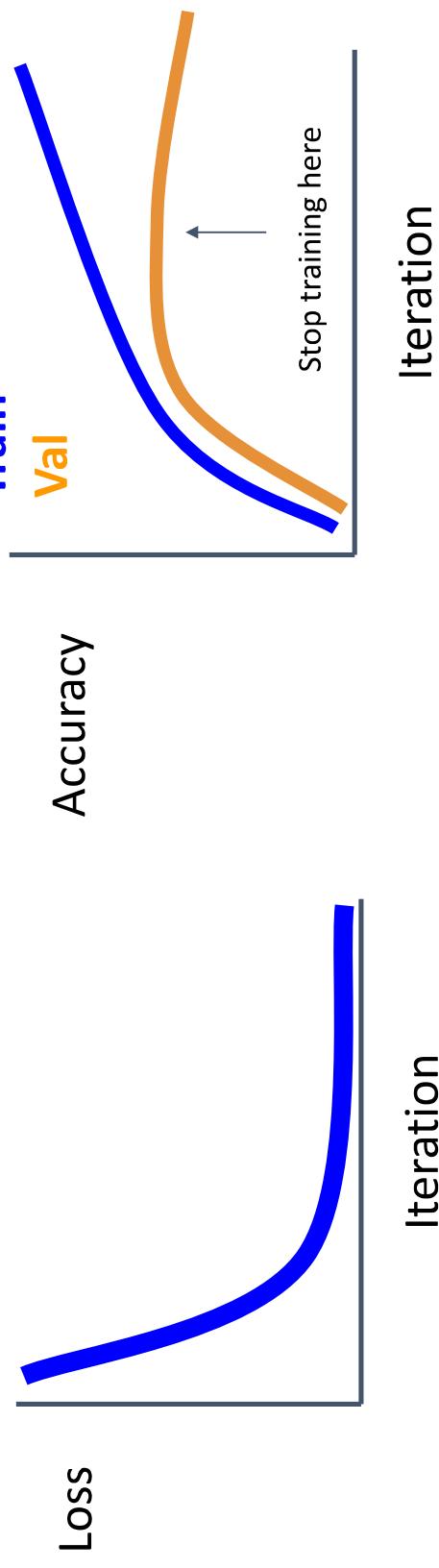
$$\text{Linear: } \alpha_t = \alpha_0(1 - t/T)$$

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

$$\text{Constant: } \alpha_t = \alpha_0$$

Brock et al, "Large Scale GAN Training for High Fidelity Natural Image Synthesis", ICLR 2019
Donahue and Simonyan, "Large Scale Adversarial Representation Learning", NeurIPS 2019

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

Justin Johnson

Lecture 11 - 22

October 9, 2019

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:

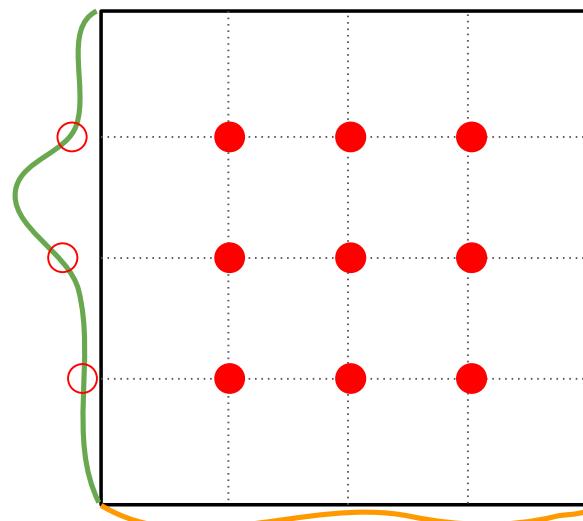
Weight decay: log-uniform on $[1 \times 10^{-4}, 1 \times 10^{-1}]$

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

Run many different trials

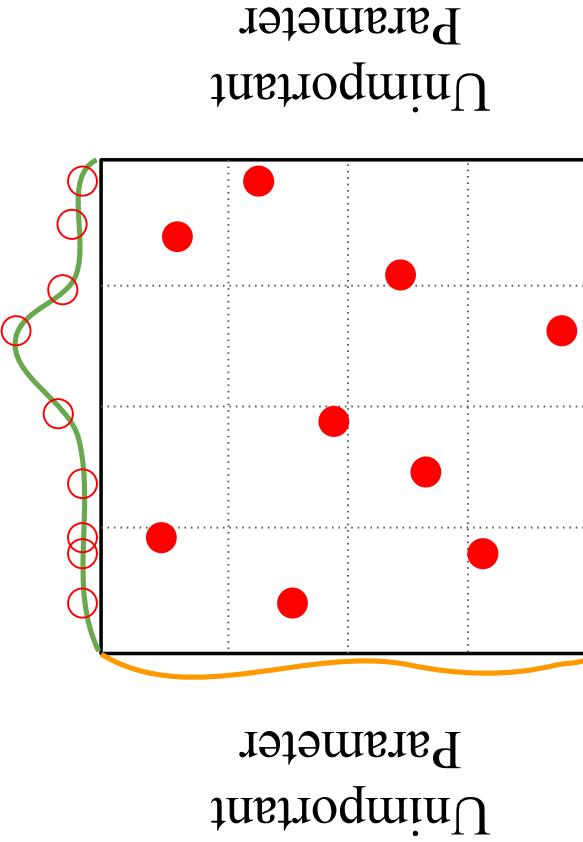
Hyperparameters: Random vs Grid Search

Grid Layout



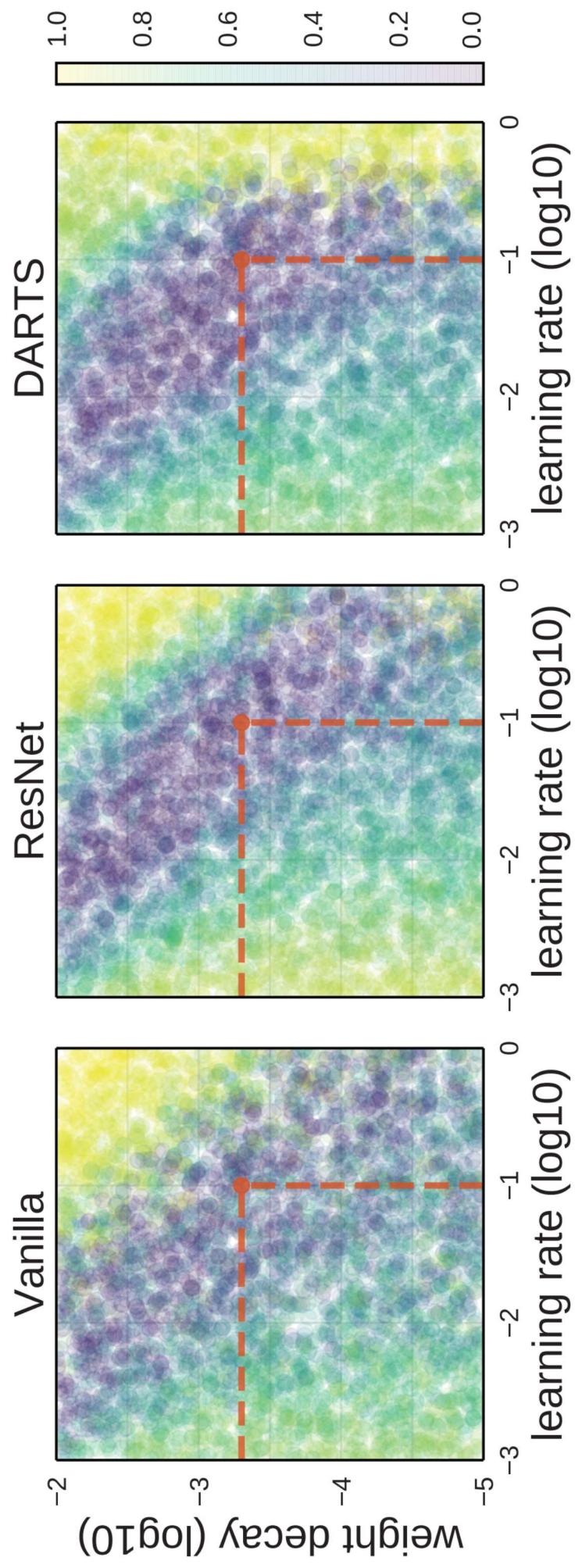
Important
Parameter

Random Layout



Important
Parameter

Choosing Hyperparameters: Random Search



Radosavovic et al., "On Network Design Spaces for Visual Recognition", ICCV 2019

Justin Johnson

Lecture 11 - 26

October 9, 2019

Choosing Hyperparameters

(without tons of GPUs)

Justin Johnson

Lecture 11 - 27

October 9, 2019

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

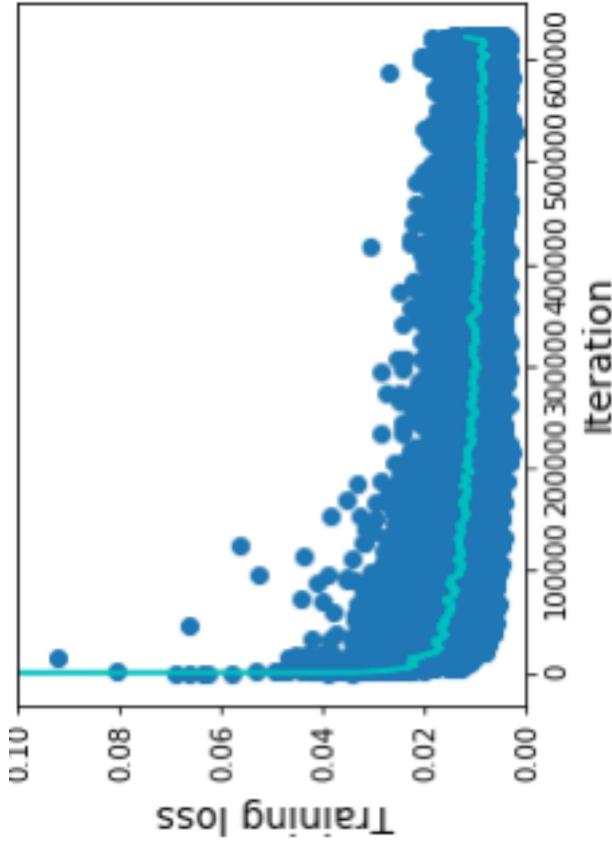
Step 5: Refine grid, train longer

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

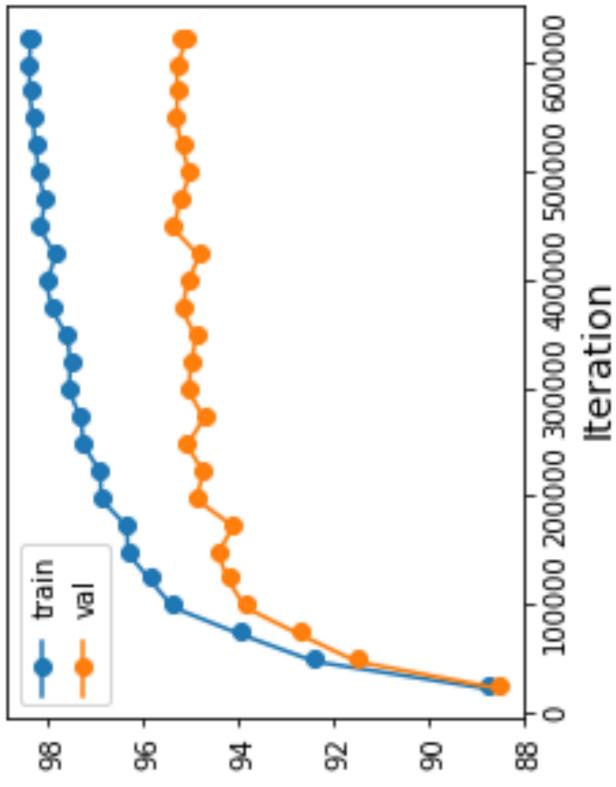
Choosing Hyperparameters

- 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!



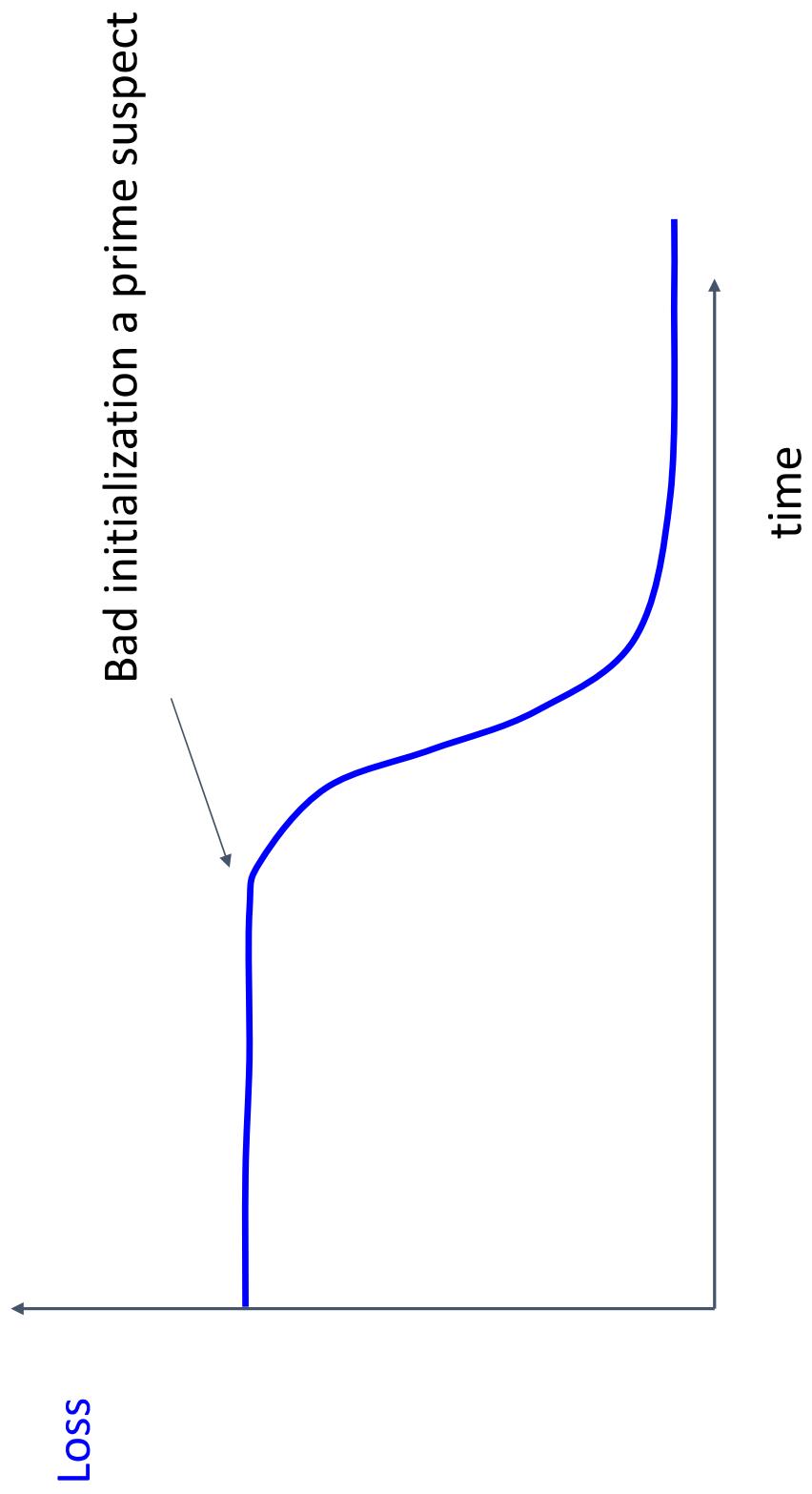
Losses may be noisy, use a scatter plot and also plot moving average to see trends better

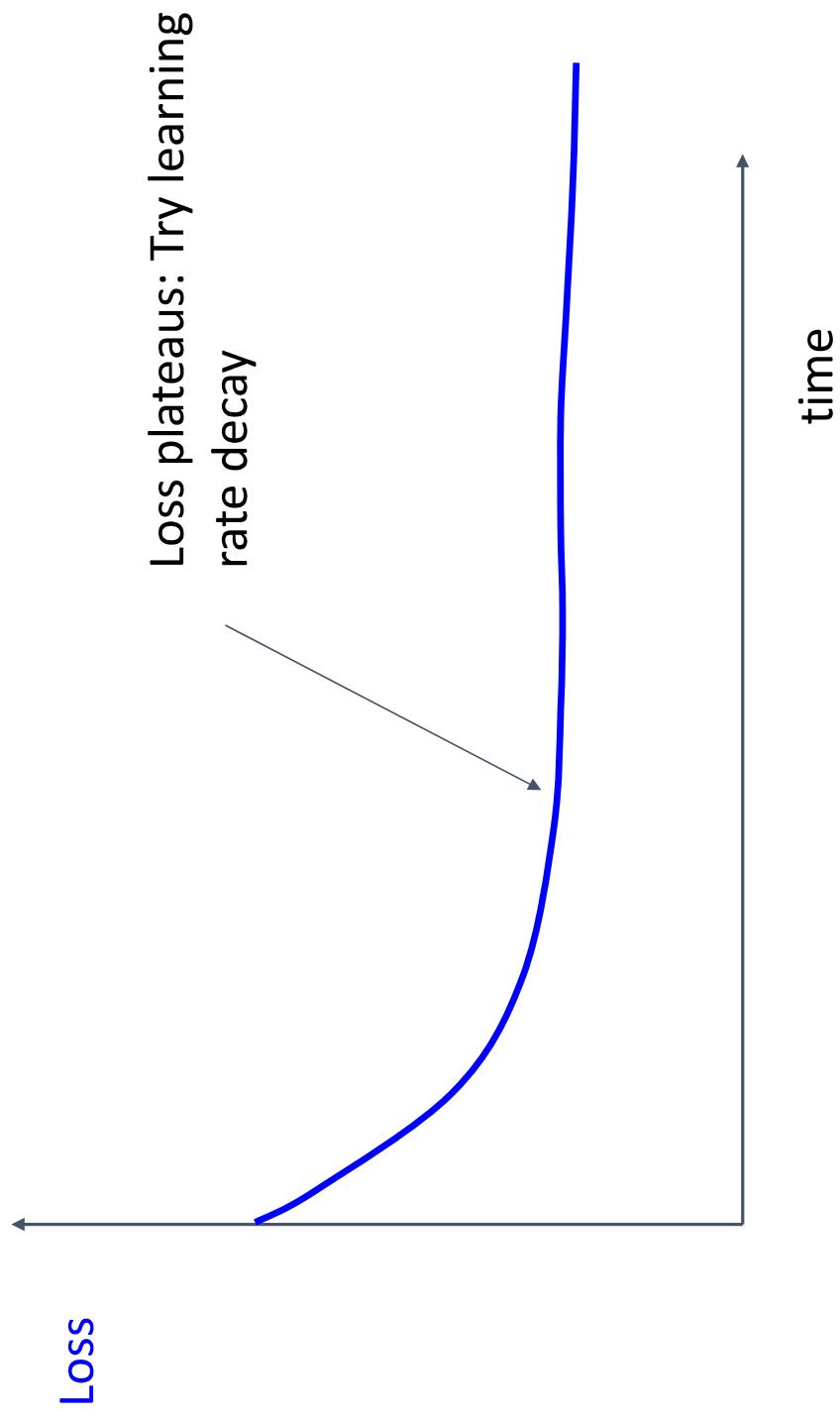


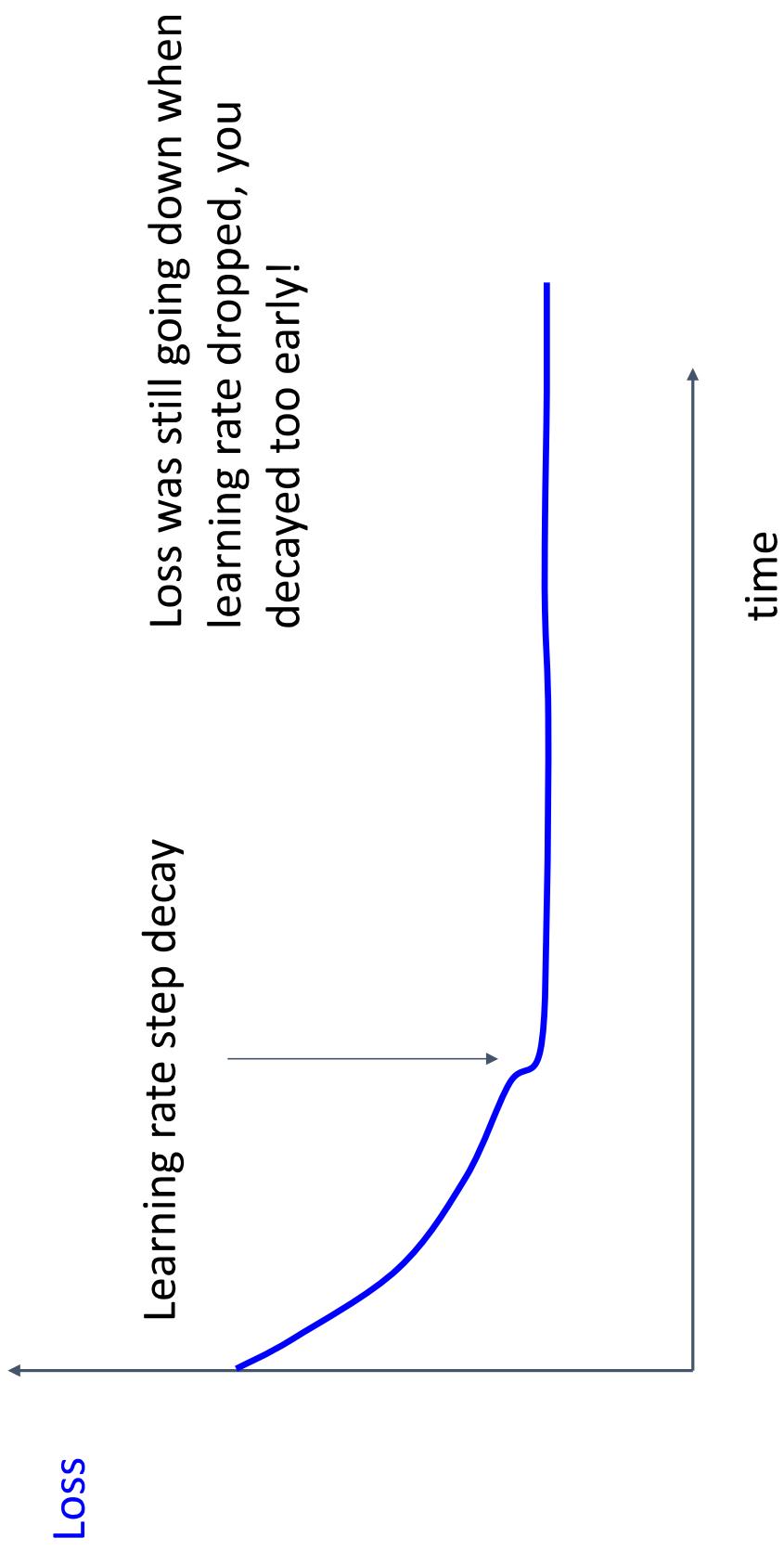
October 9, 2019

Lecture 11 - 35

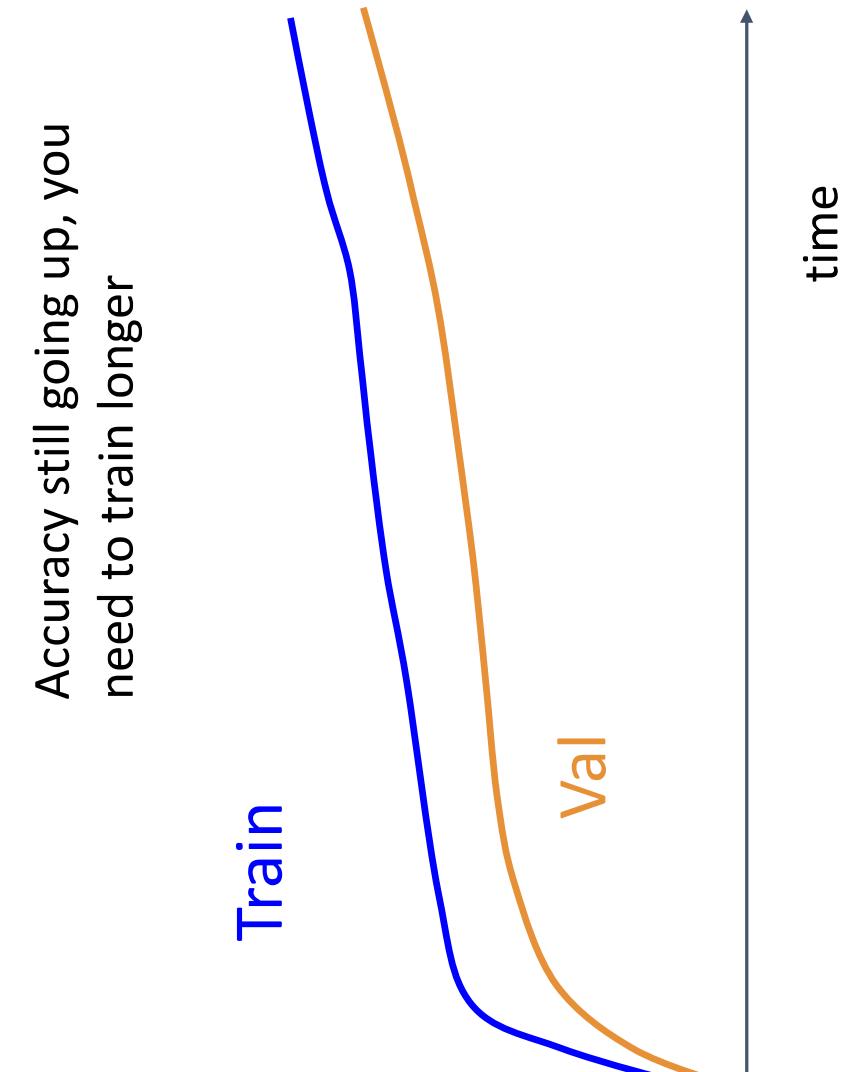
Justin Johnson

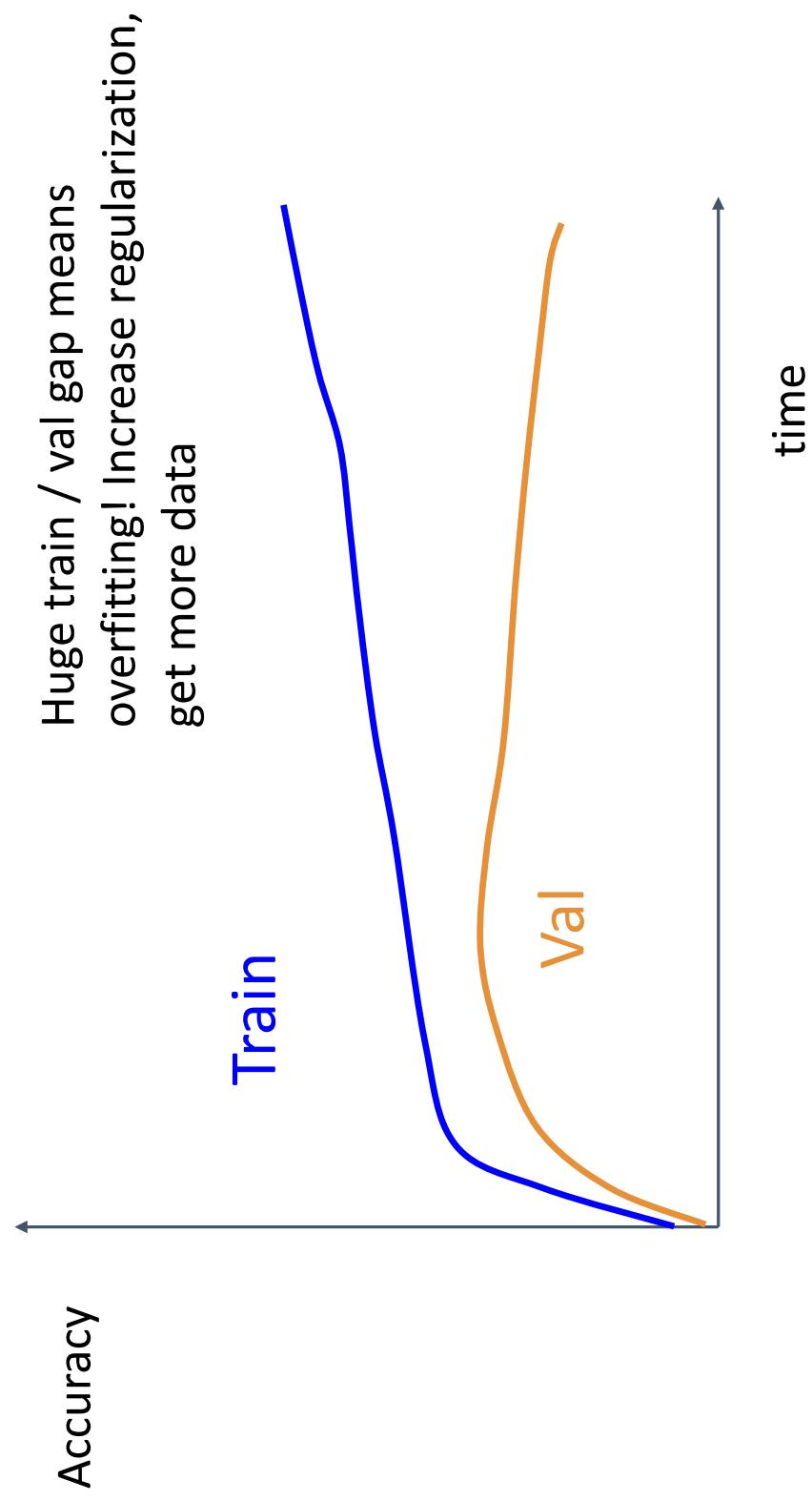


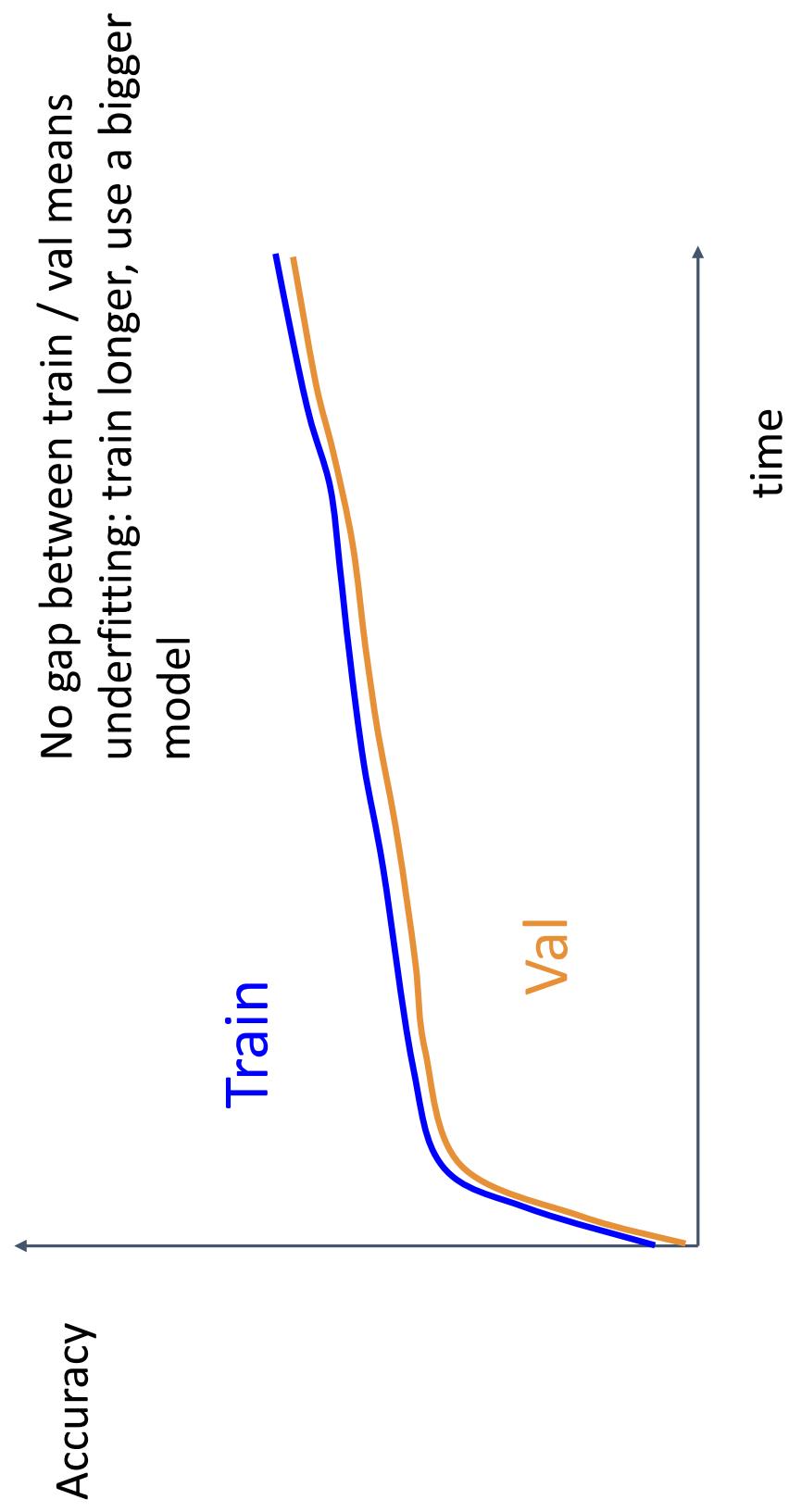




Accuracy still going up, you
need to train longer





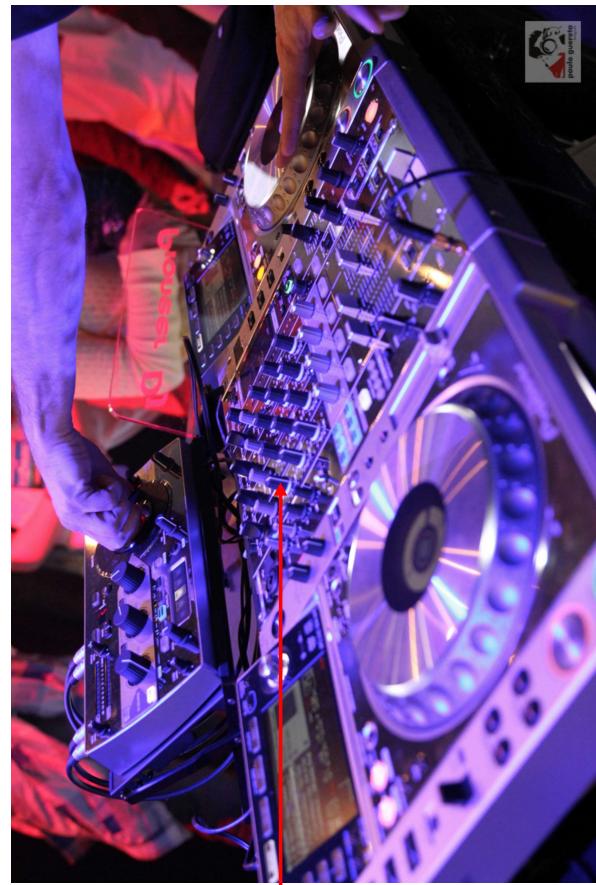


Choosing Hyperparameters

- 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

This image by Paolo Guereta is licensed under CC-BY 2.0

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

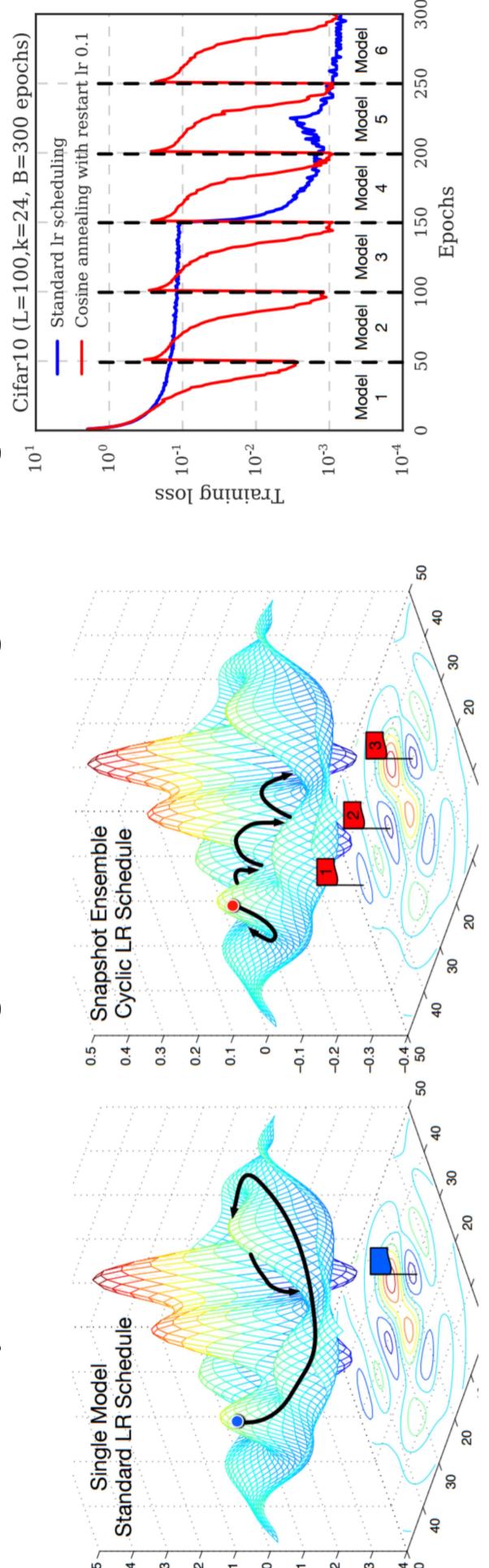
Model Ensembles

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!



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.

Cyclic learning rate schedules can make this work even better!

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

Justin Johnson

Lecture 11 - 49

October 9, 2019

Transfer Learning

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

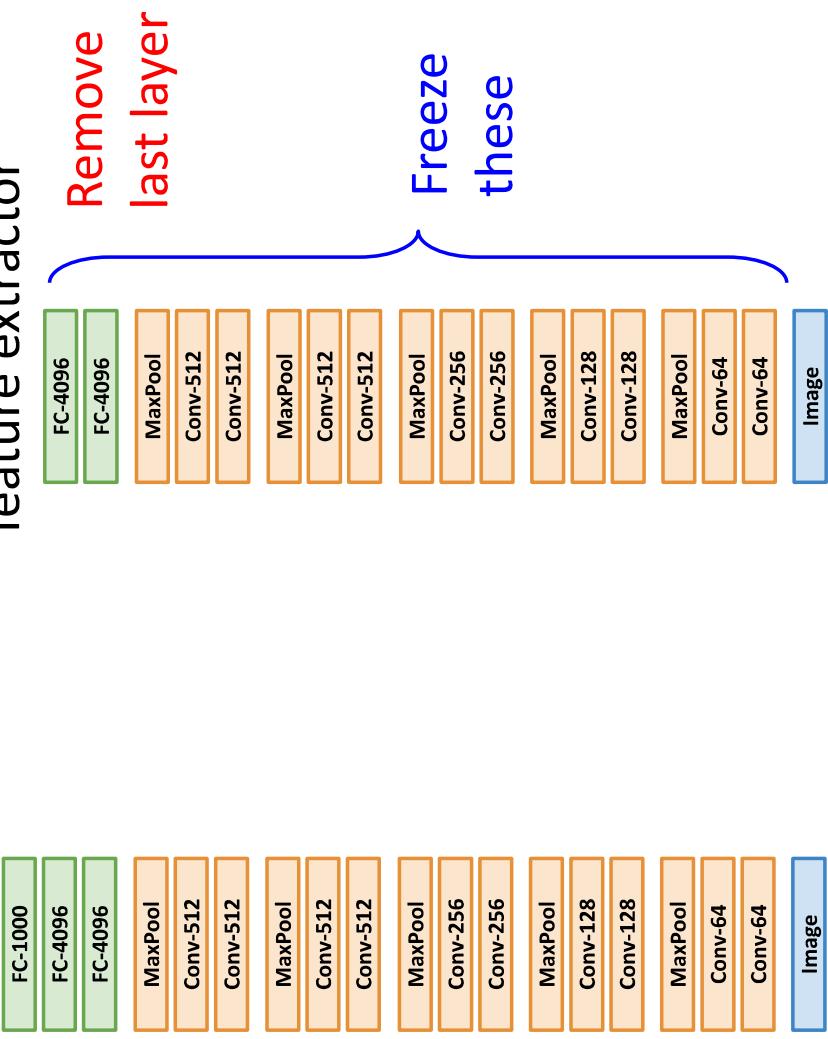
Transfer Learning

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

Transfer Learning with CNNs

1. Train on Imagenet

2. Use CNN as a
feature extractor

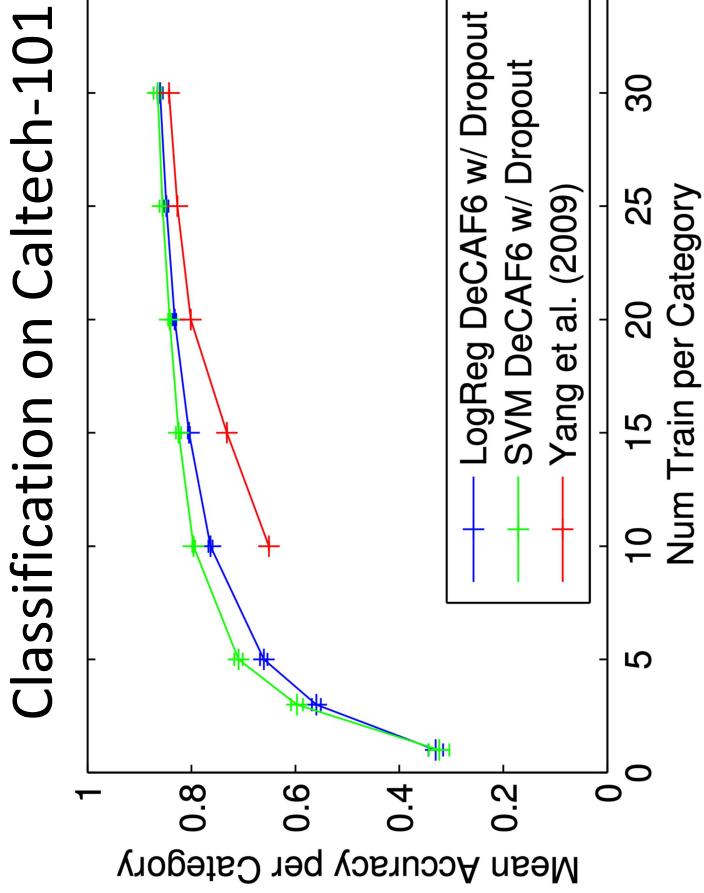
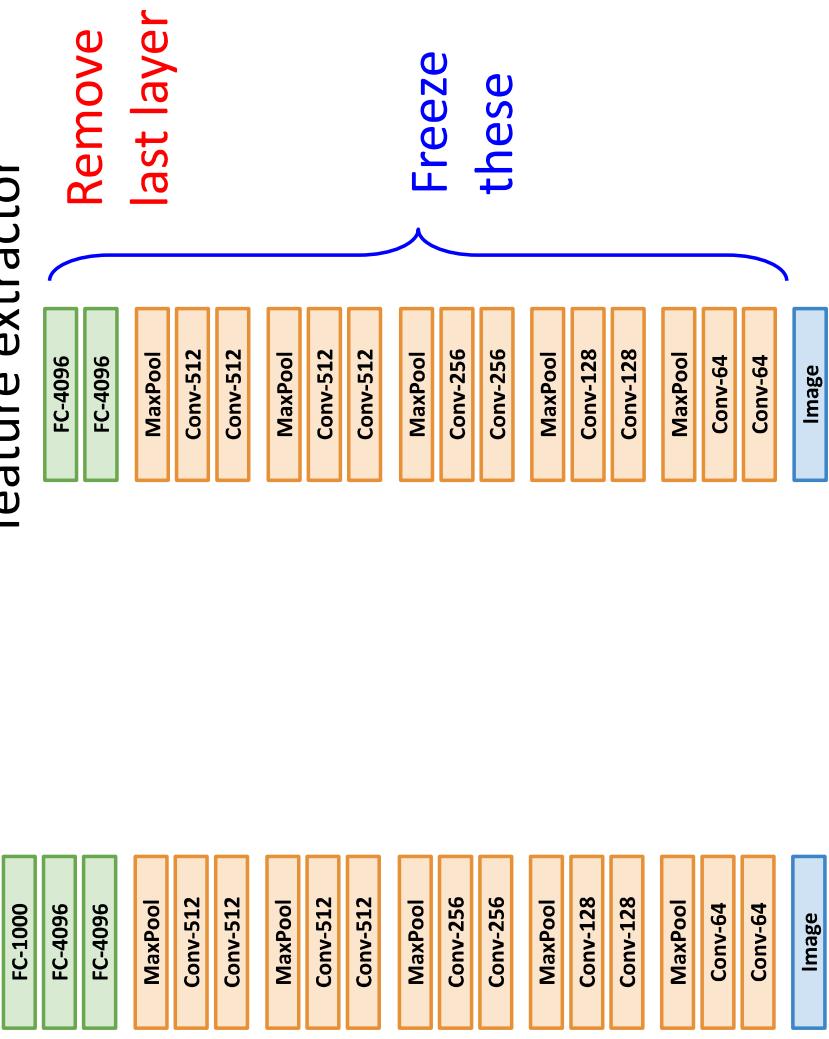


Donahue et al., "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014

Transfer Learning with CNNs

1. Train on Imagenet

2. Use CNN as a
feature extractor

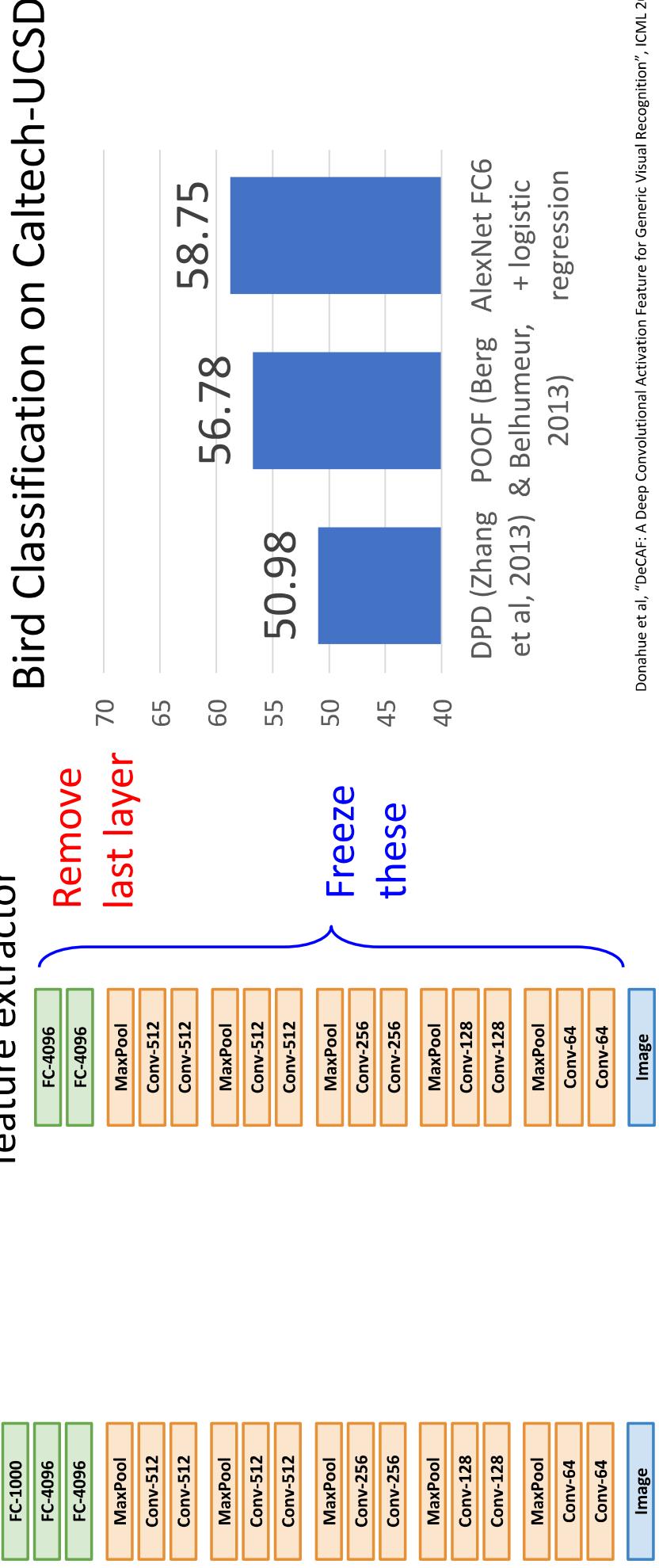


Donahue et al., "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014

Transfer Learning with CNNs

1. Train on Imagenet

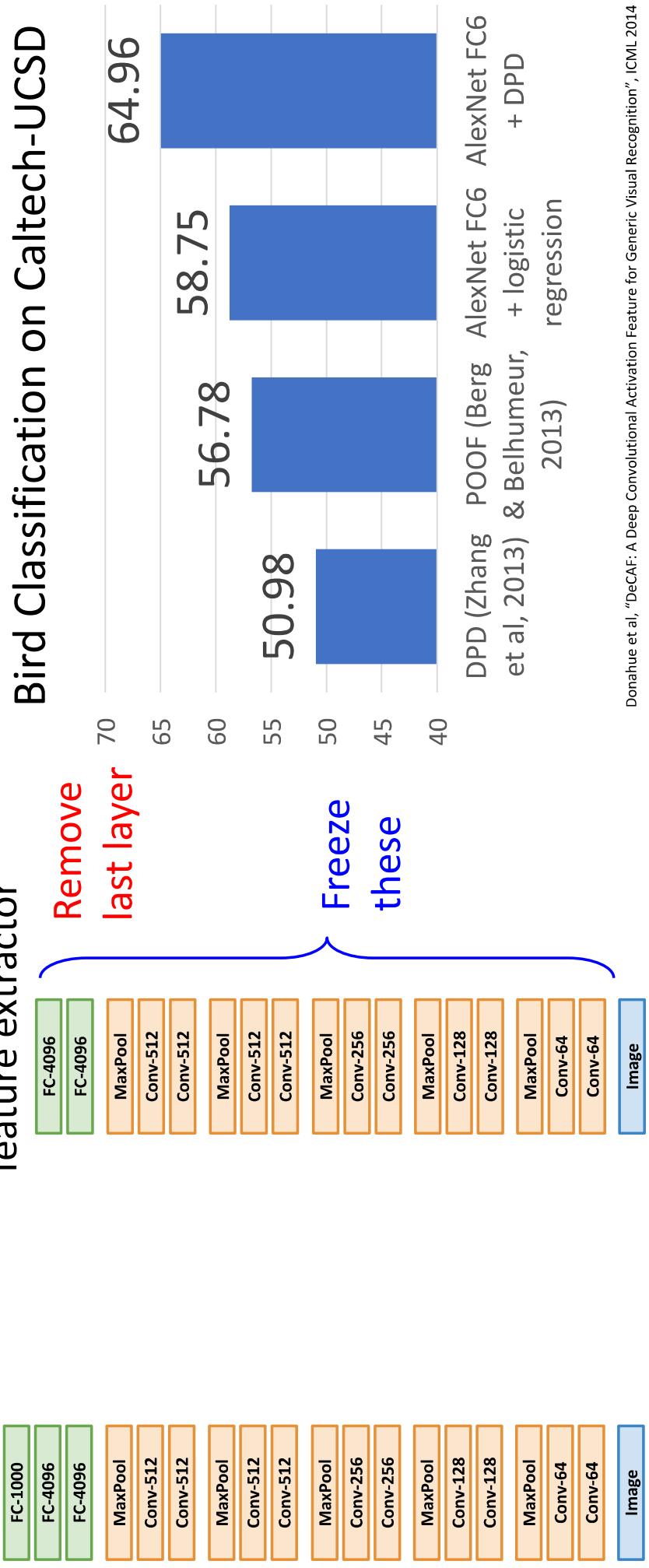
2. Use CNN as a
feature extractor



Transfer Learning with CNNs

1. Train on Imagenet

2. Use CNN as a feature extractor

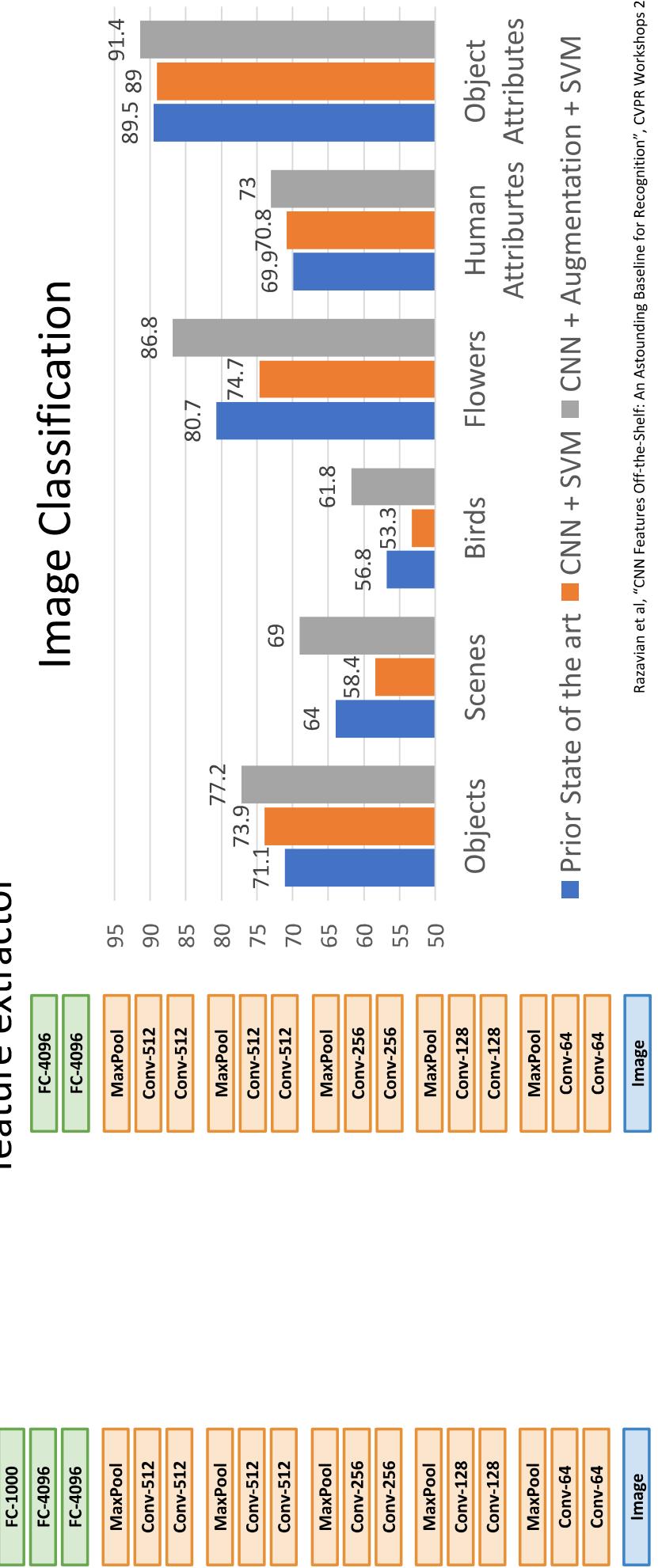


Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014

Transfer Learning with CNNs

1. Train on Imagenet

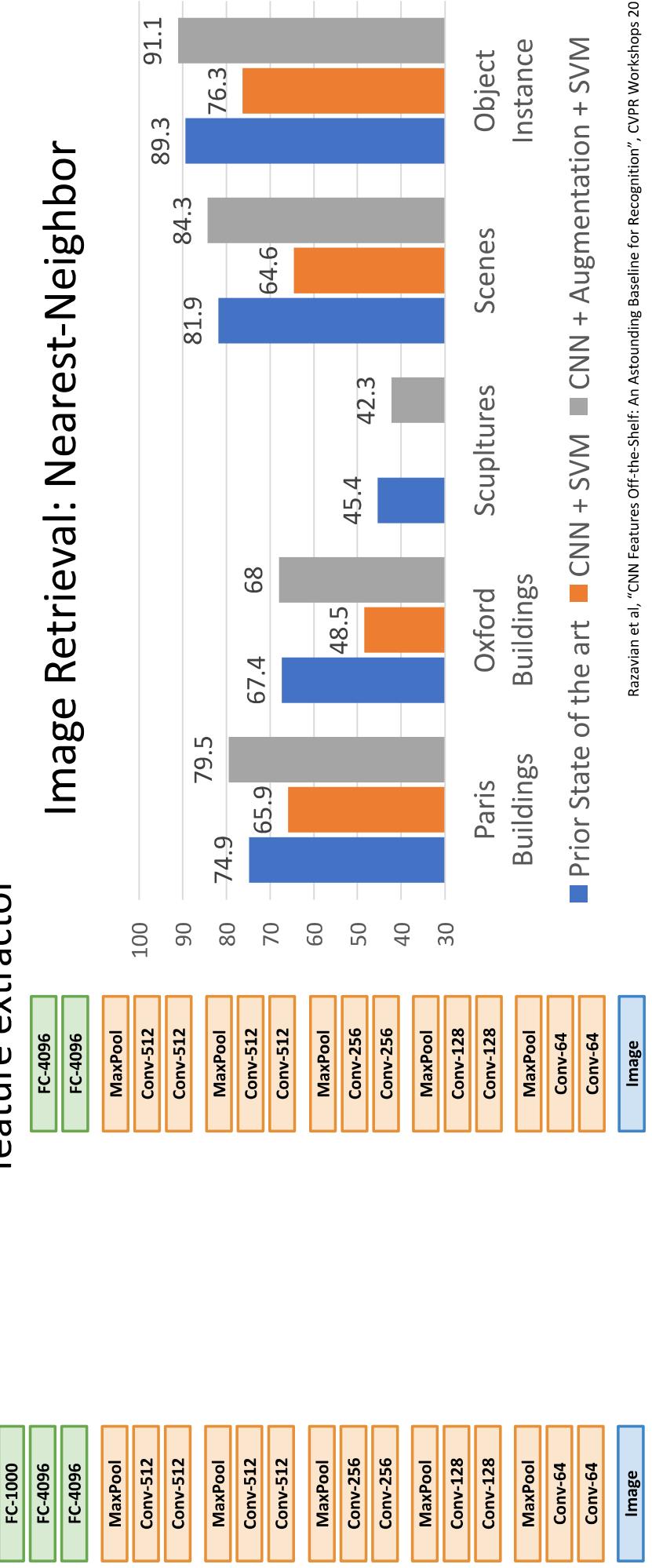
2. Use CNN as a
feature extractor



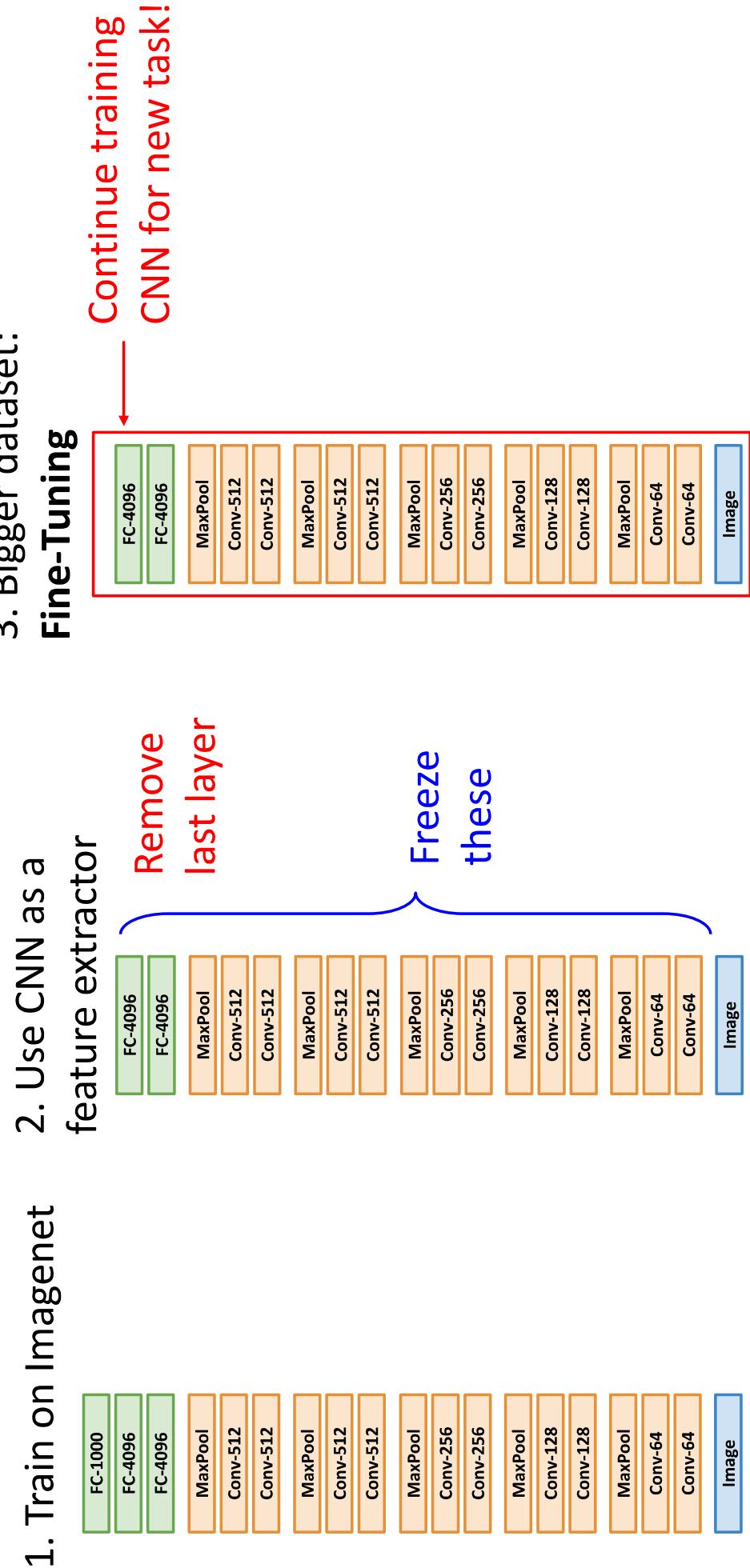
Transfer Learning with CNNs

1. Train on Imagenet

2. Use CNN as a
feature extractor



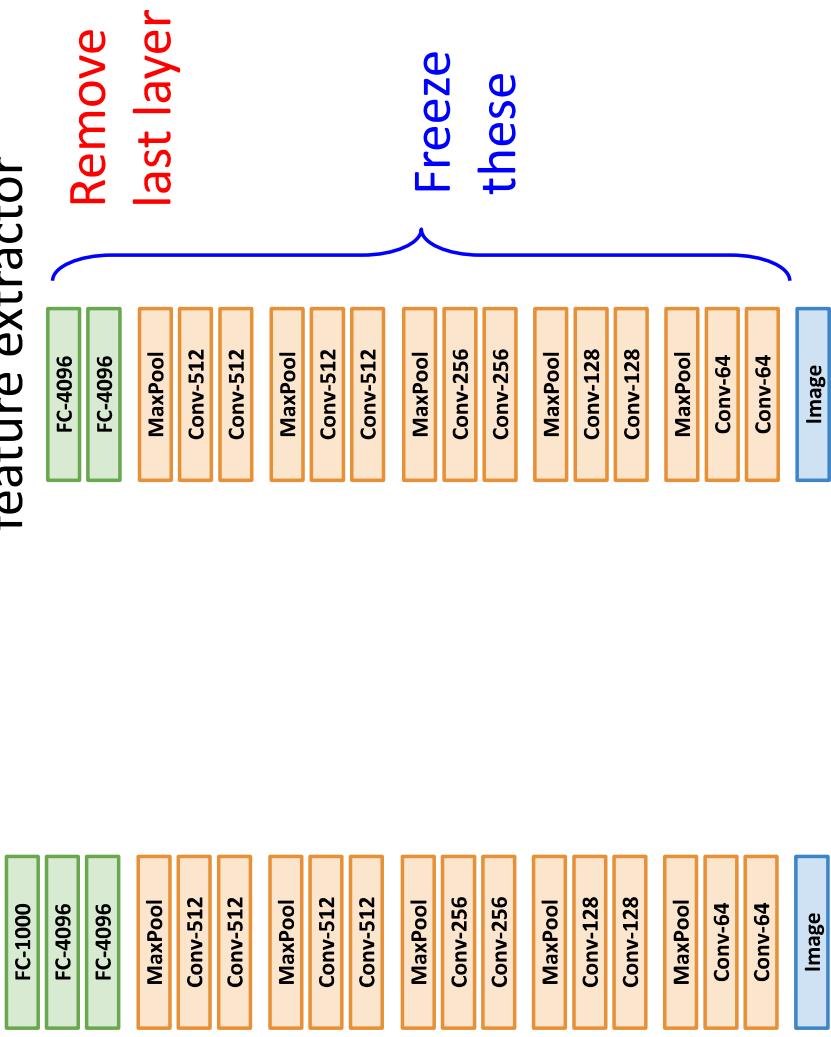
Transfer Learning with CNNs



Transfer Learning with CNNs

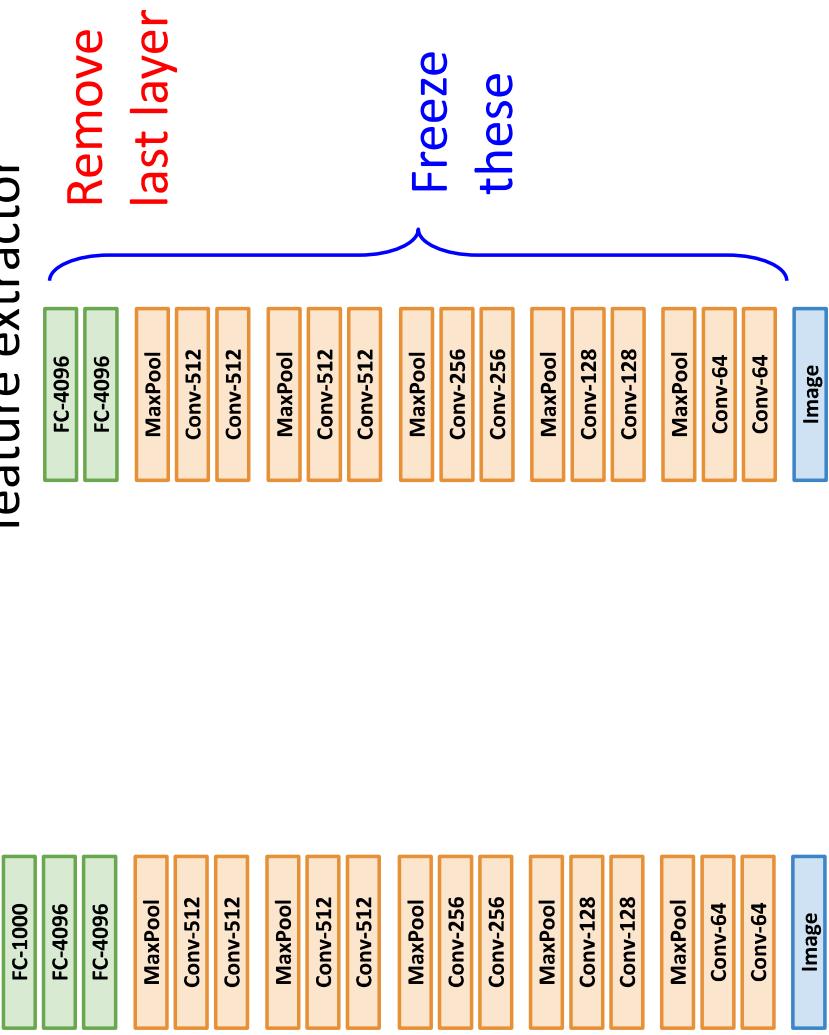
1. Train on Imagenet

3. Bigger dataset:
2. Use CNN as a
feature extractor

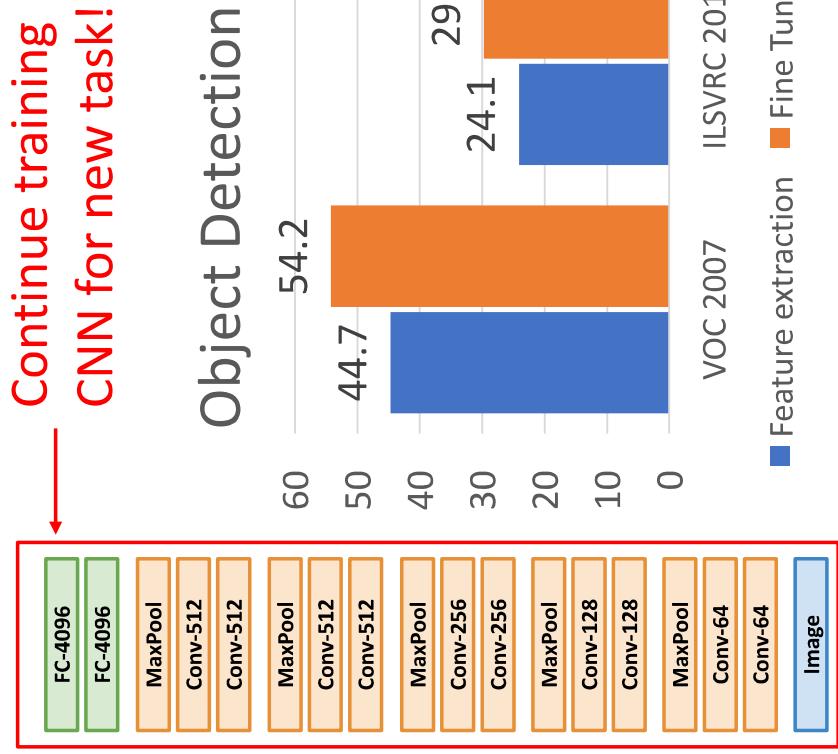


Transfer Learning with CNNs

1. Train on Imagenet 2. Use CNN as a feature extractor



3. Bigger dataset:
Fine-Tuning



Transfer Learning with CNNs: Architecture Matters!

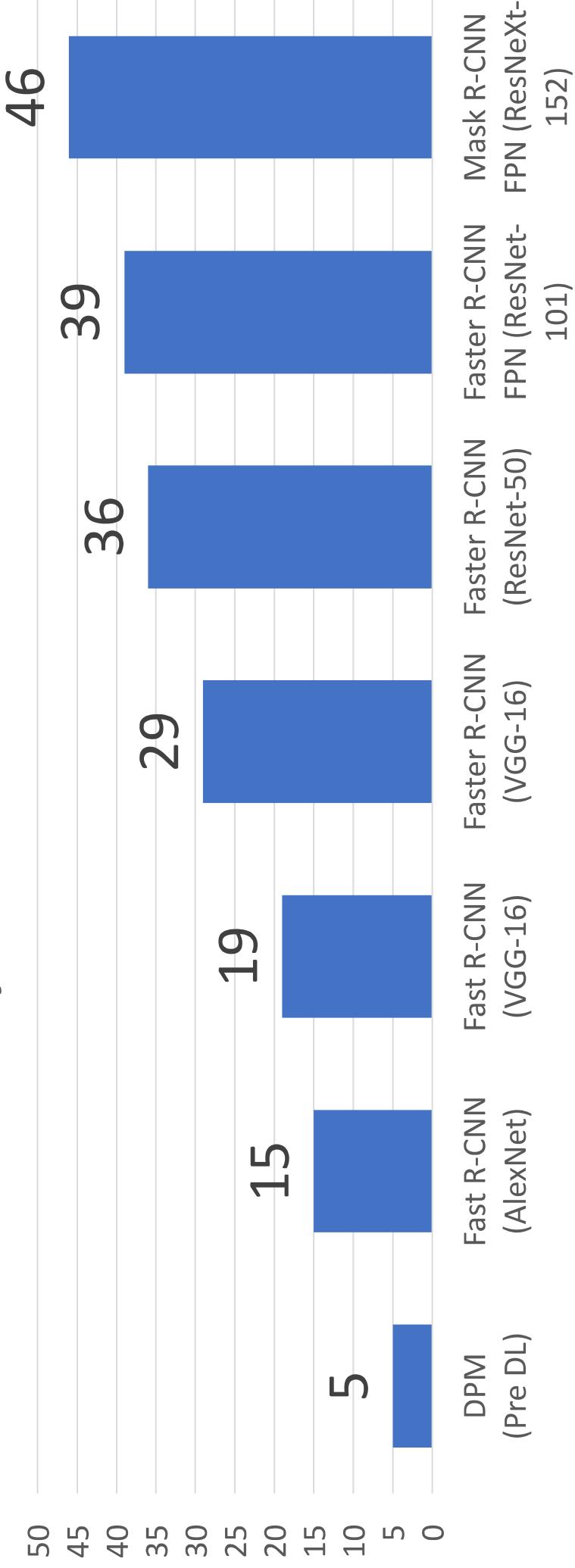
ImageNet Classification Challenge



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

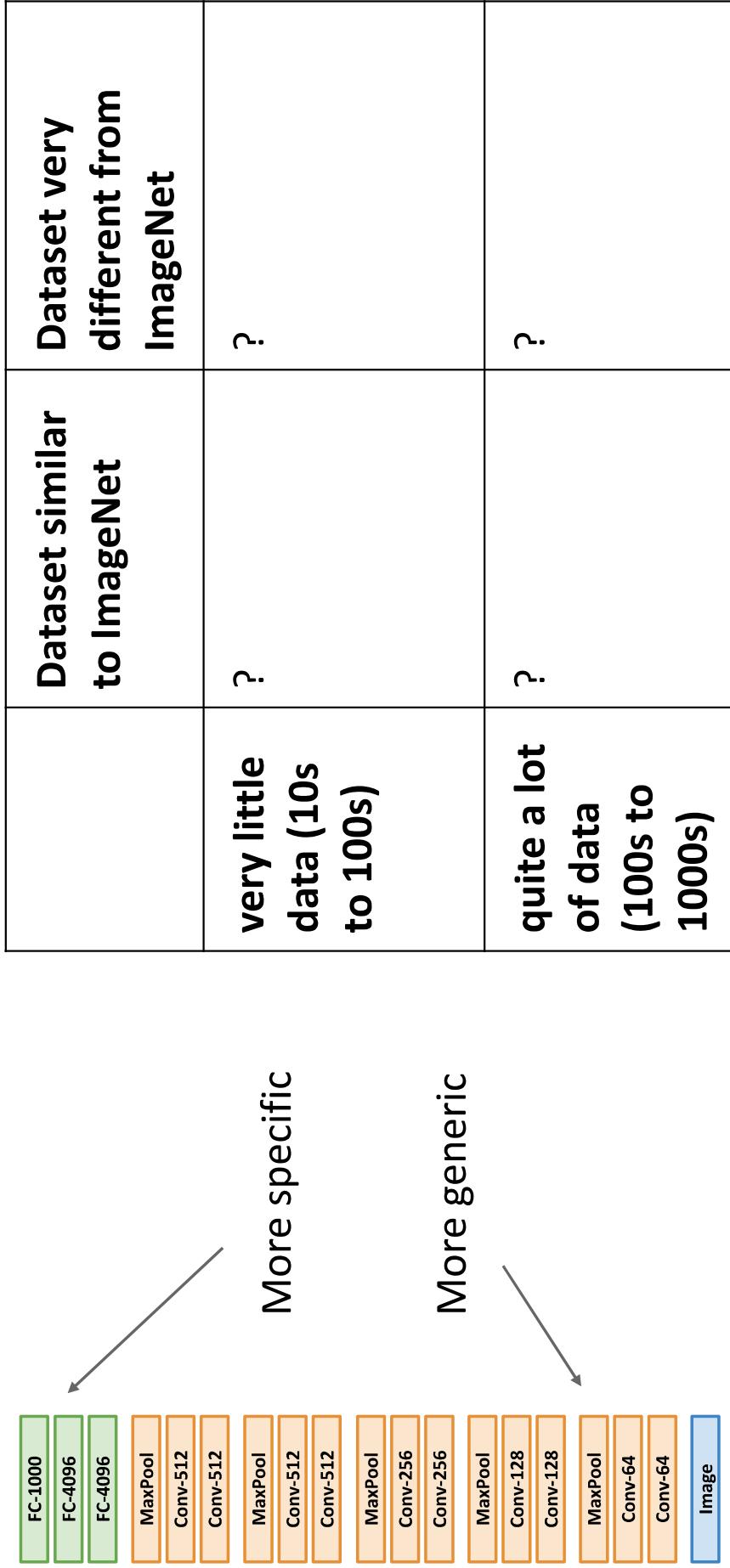
Transfer Learning with CNNs: Architecture Matters!

Object Detection on COCO

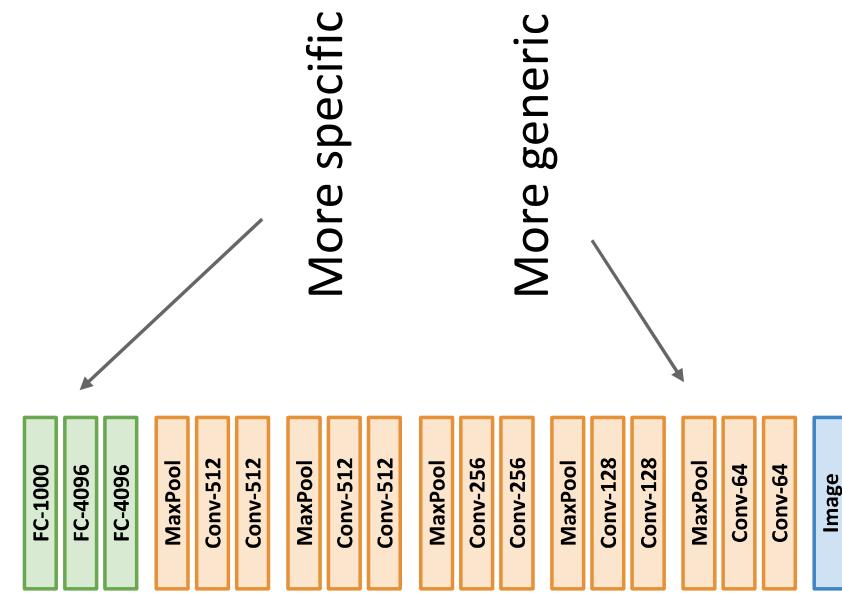


Ross Girshick, "The Generalized R-CNN Framework for Object Detection", ICCV 2017 Tutorial on Instance-Level Visual Recognition

Transfer Learning with CNNs

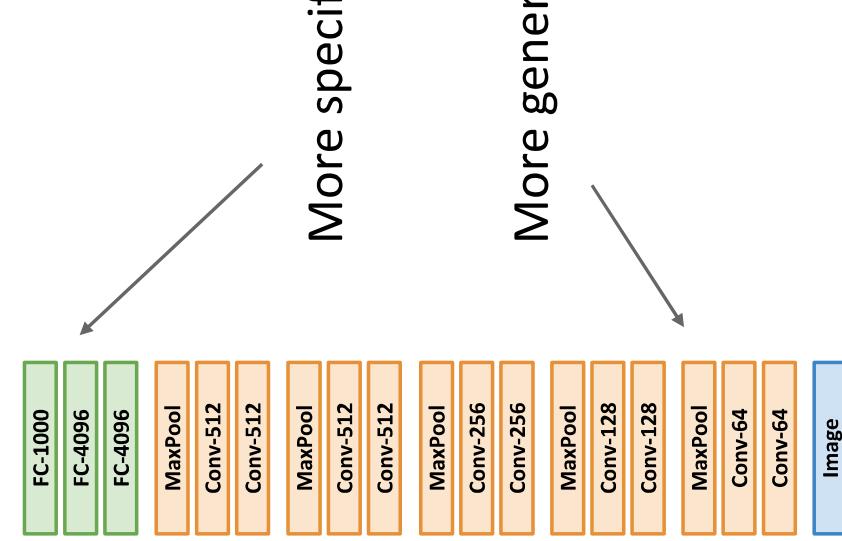


Transfer Learning with CNNs



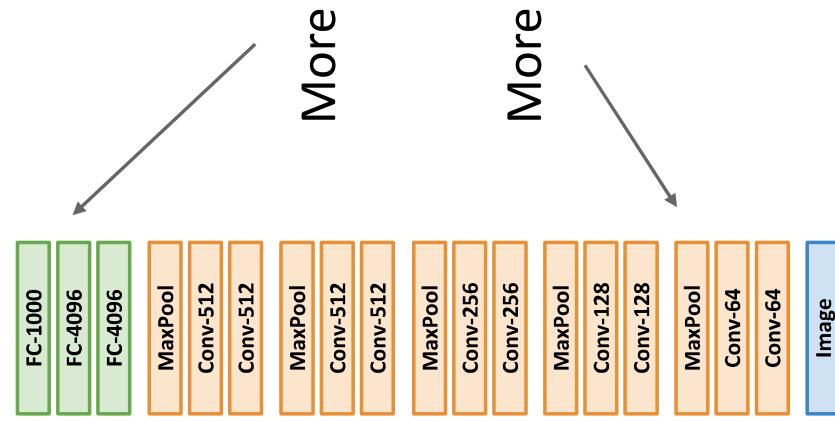
Dataset similar to ImageNet	Dataset very different from ImageNet
very little data (10s to 100s)	Use Linear Classifier on top layer
quite a lot of data (100s to 1000s)	Finetune a few layers

Transfer Learning with CNNs



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 larger number of layers

Transfer Learning with CNNs

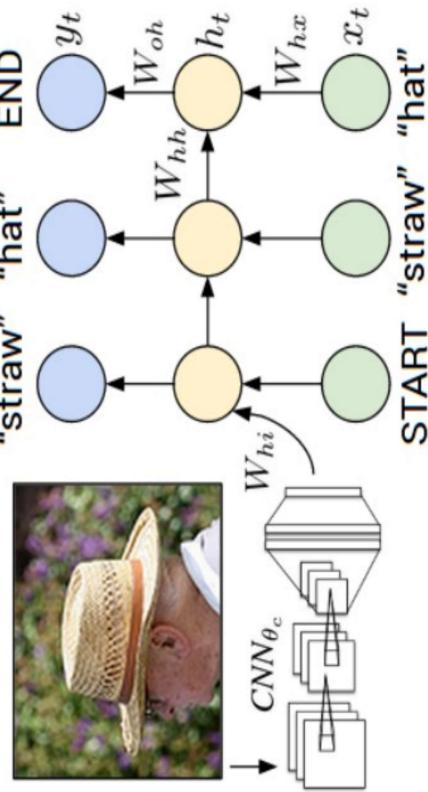
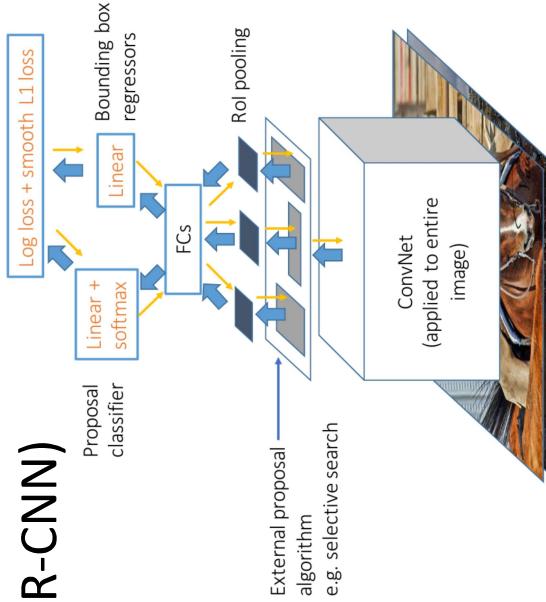


	Dataset similar to ImageNet	Dataset very different from ImageNet
very little data (10s to 100s)	Use Linear Classifier on top layer	You're in trouble... Try linear classifier from different stages
quite a lot of data (100s to 1000s)	Finetune a few layers	Finetune a larger number of layers

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.

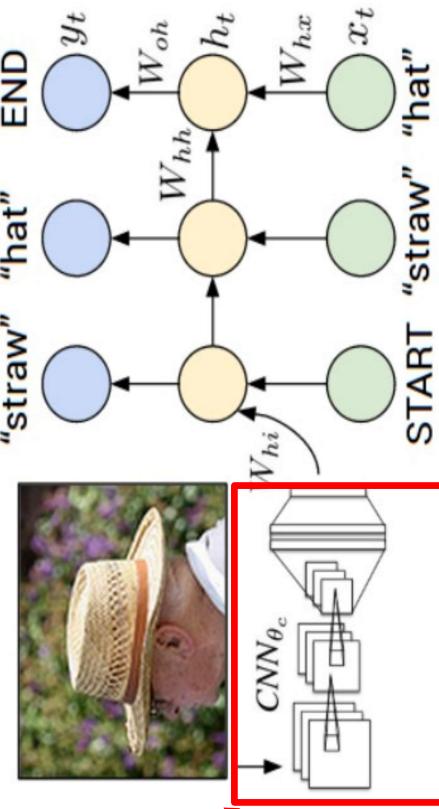
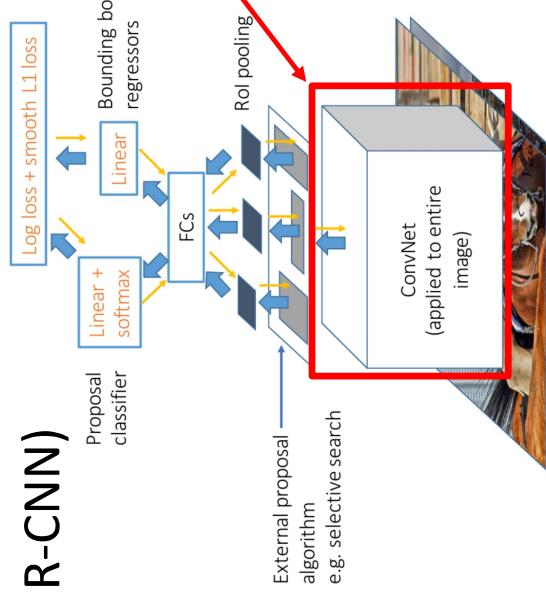
Karpathy and Fei-Fei, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015

Transfer learning is pervasive!
It's the norm, not the exception

Object

Detection (Fast R-CNN)

CNN pretrained
on ImageNet



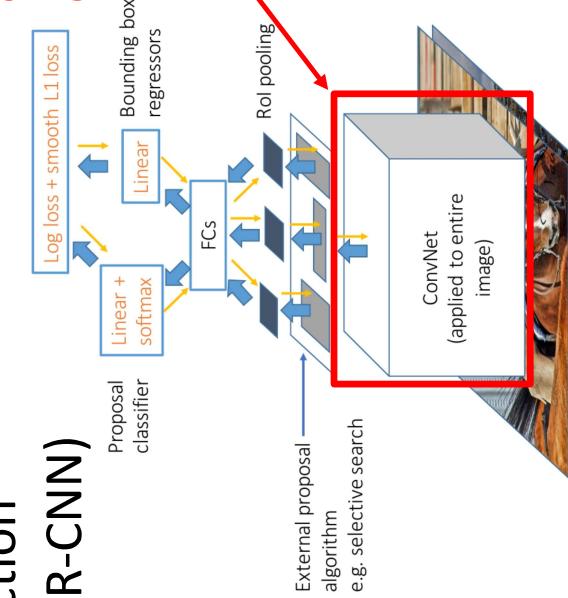
Girshick, "Fast R-CNN", ICCV 2015
Figure copyright Ross Girshick, 2015. Reproduced with permission.

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

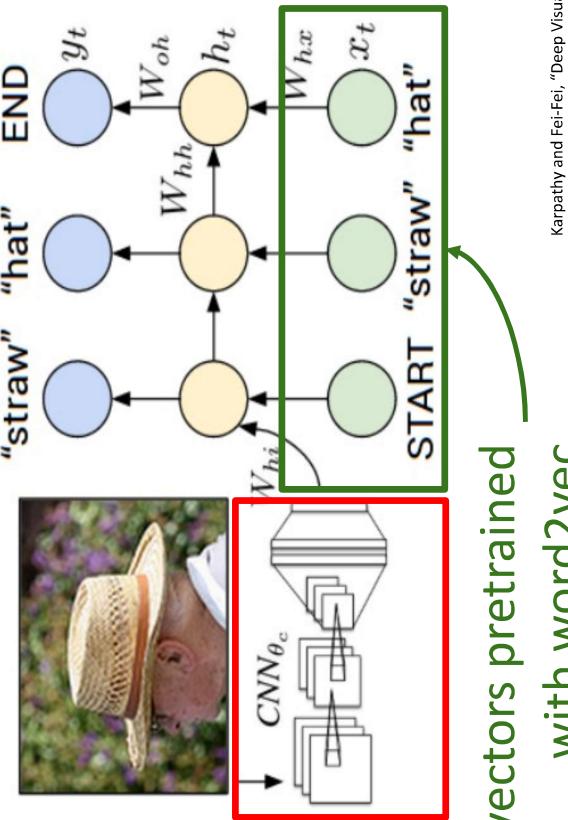
Transfer learning is pervasive!
It's the norm, not the exception

Object

Detection (Fast R-CNN)



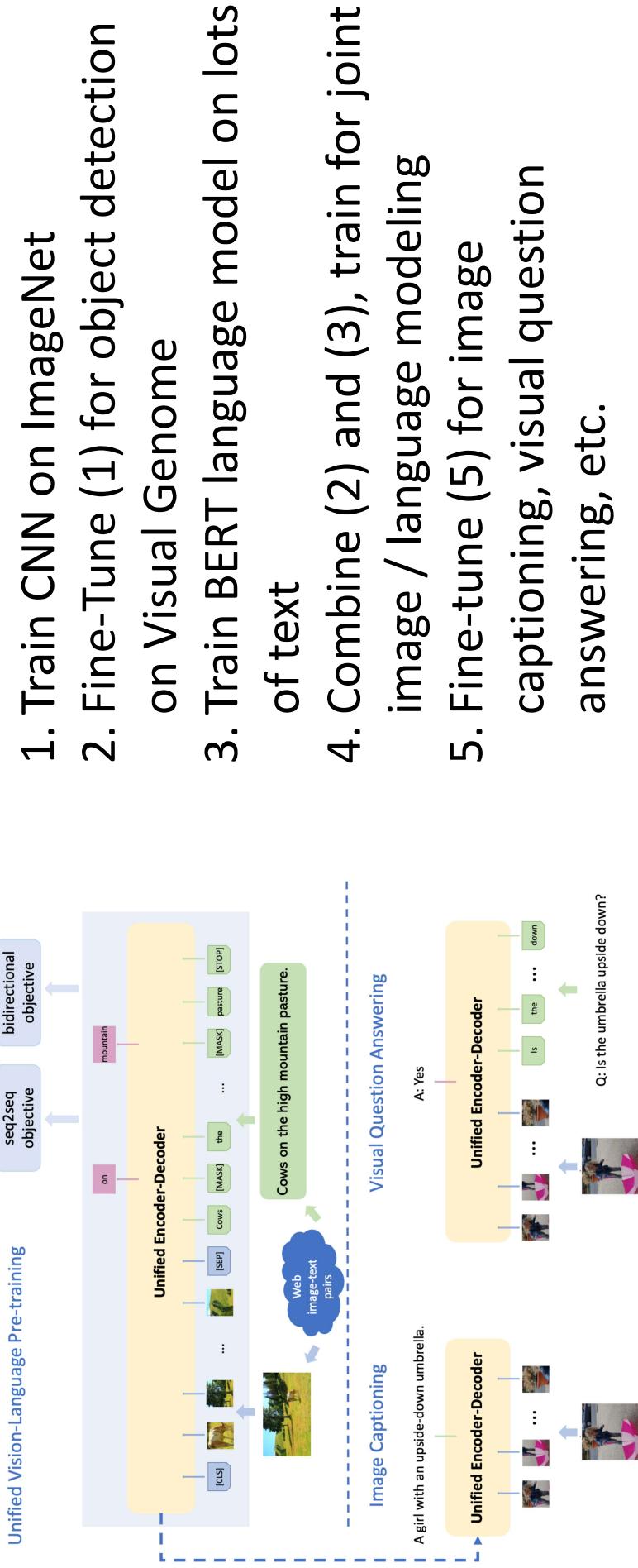
CNN pretrained
on ImageNet



Word vectors pretrained
with word2vec

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

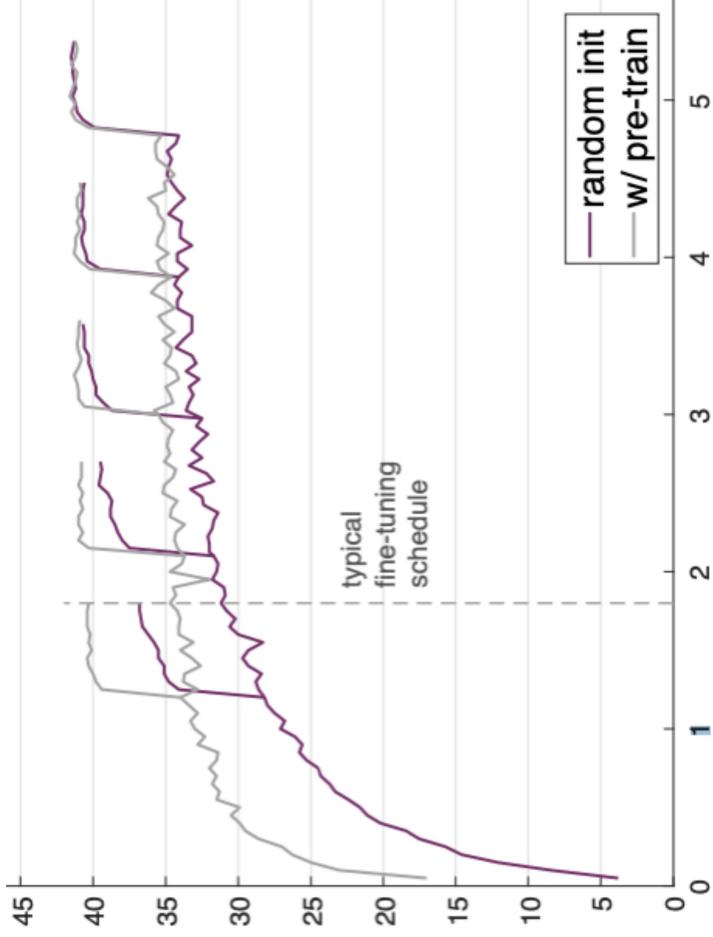


Zhou et al., "Unified Vision-Language Pre-Training for Image Captioning and VQA", arXiv 2019

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

Justin Johnson

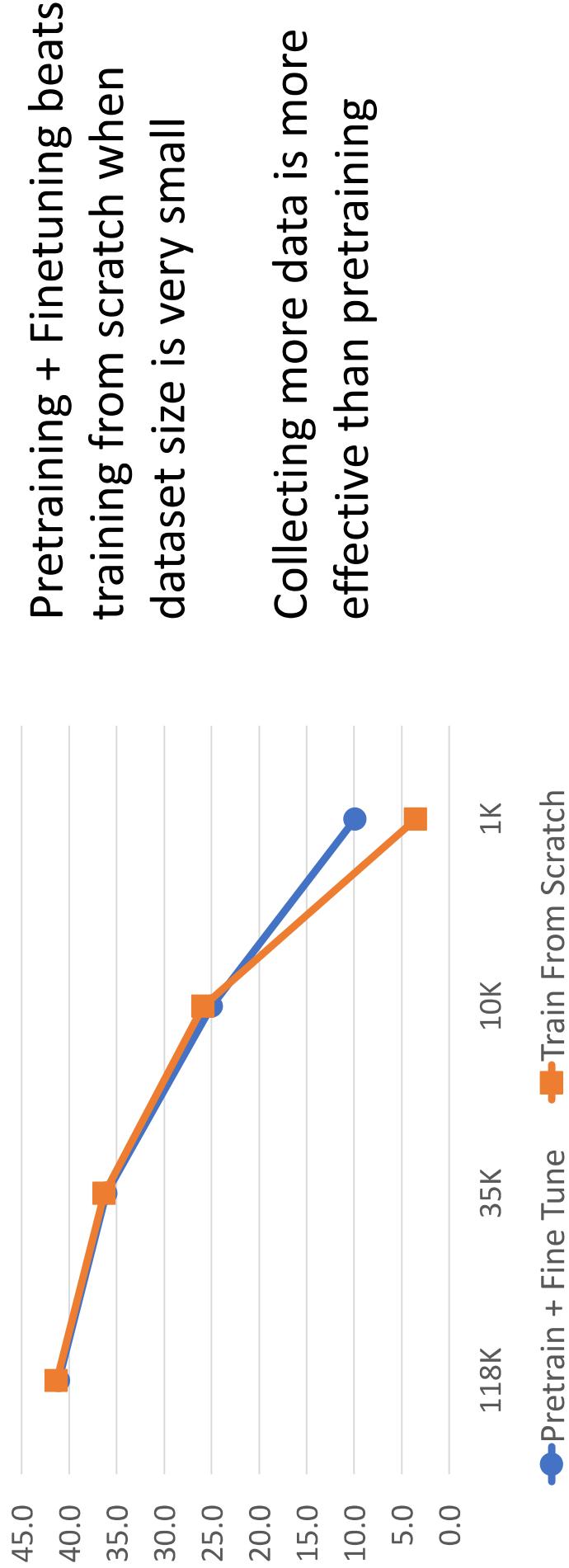
Lecture 11 - 71

October 9, 2019

Transfer learning is pervasive!

Some very recent results have questioned it

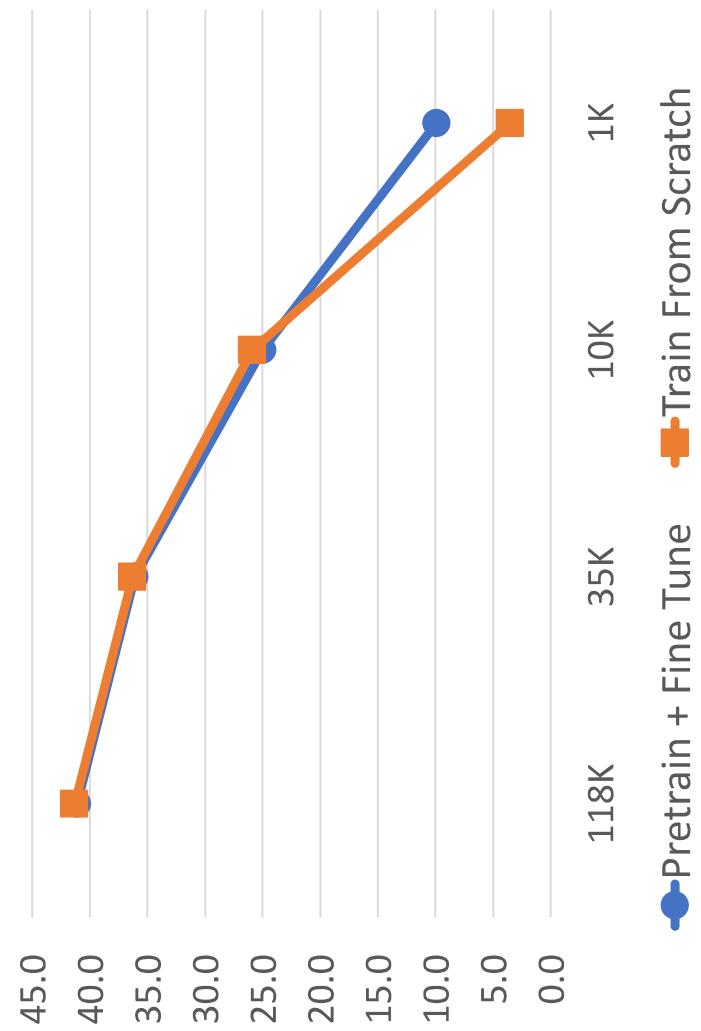
cOCO object detection



Transfer learning is pervasive!

Some very recent results have questioned it

cOCO object detection



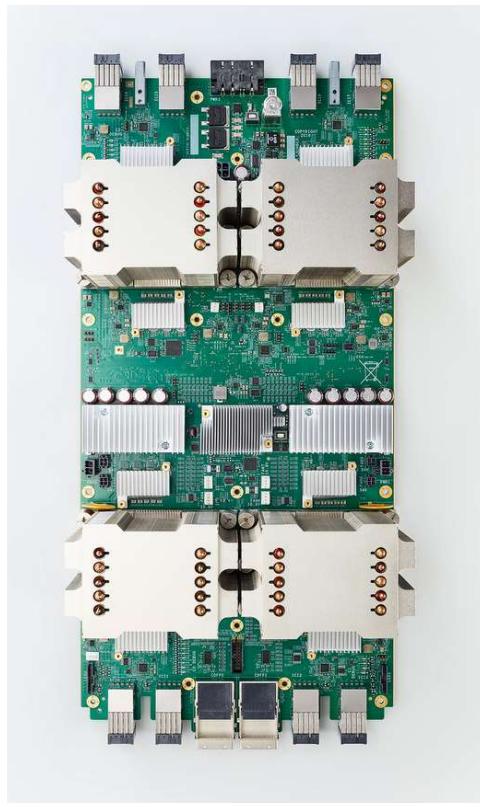
My current view on transfer learning:

- 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

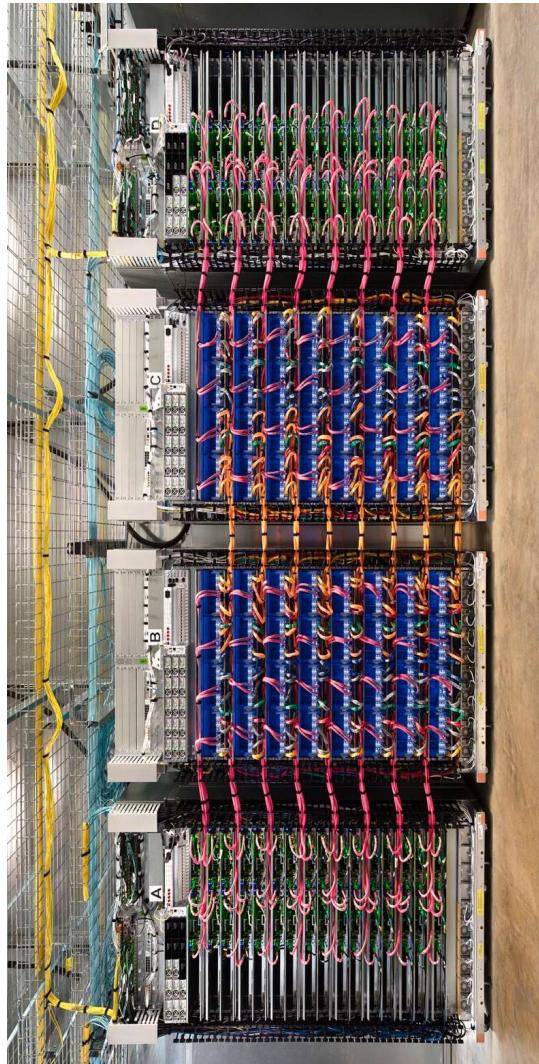
Distributed Training

Beyond individual devices



Cloud TPU v2

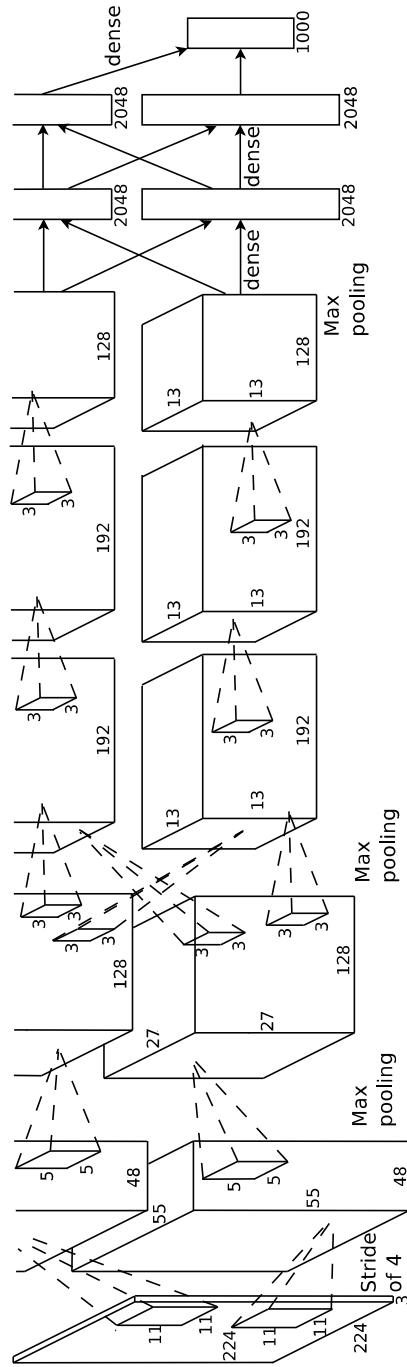
180 TFLOPs
64 GB HBM memory
\$4.50 / hour
(free on Colab!)



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

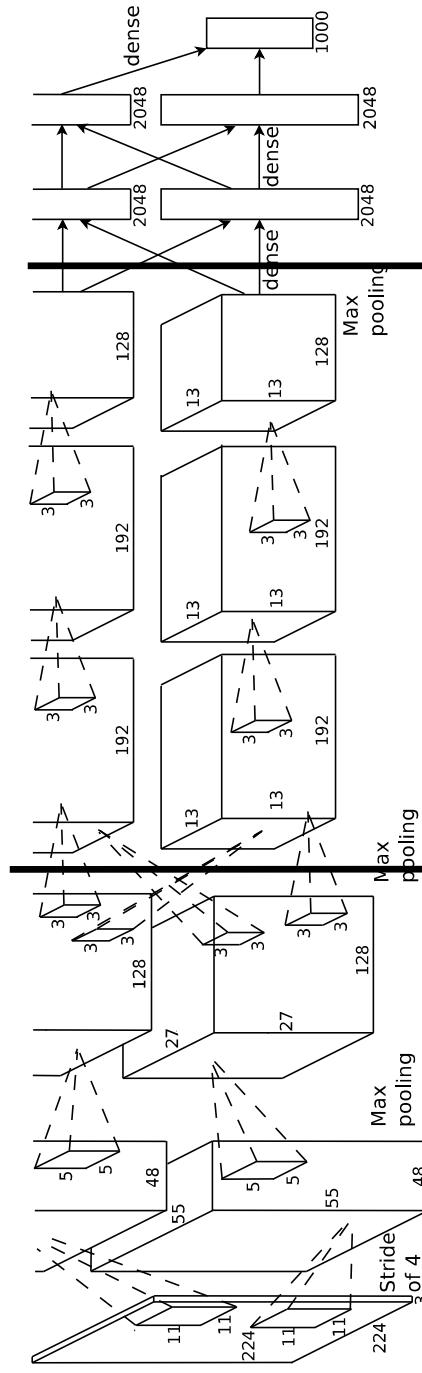
Justin Johnson

Lecture 11 - 76

October 9, 2019

Model Parallelism: Split Model Across GPUs

Idea #1: Run different layers on different GPUs



This image is in the public domain

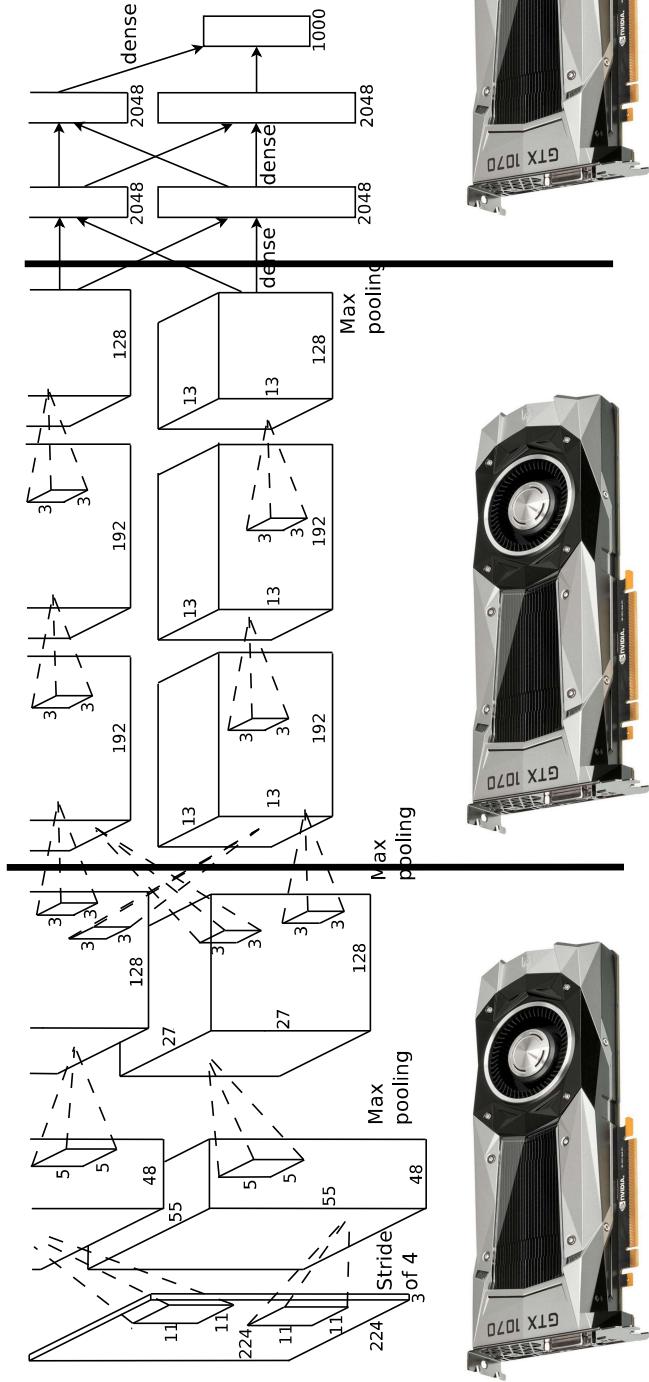
Justin Johnson

Lecture 11 - 77

October 9, 2019

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

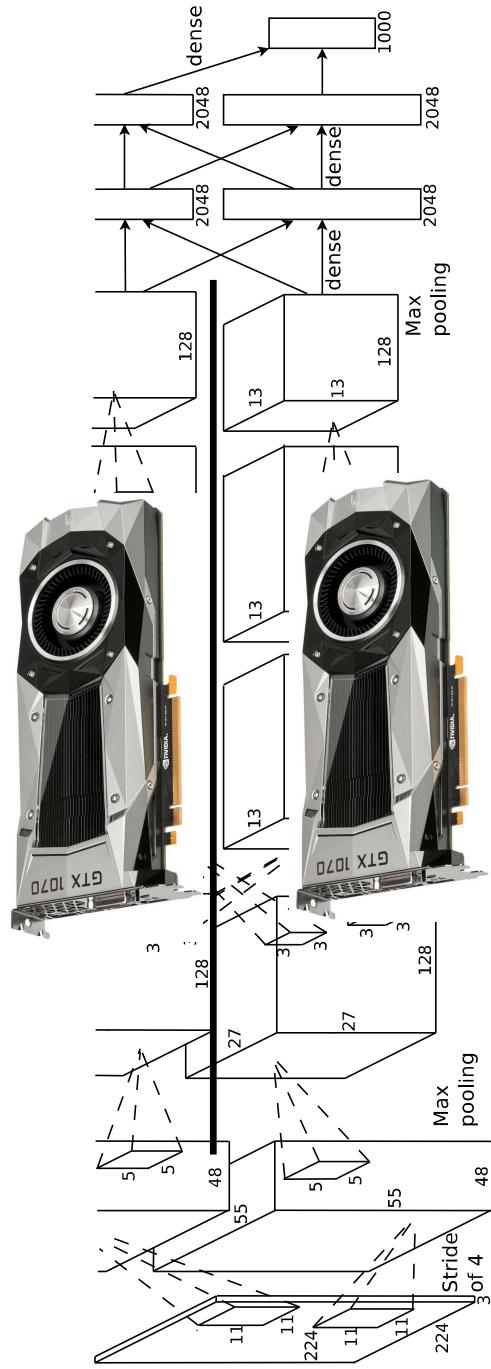
Justin Johnson

Lecture 11 - 78

October 9, 2019

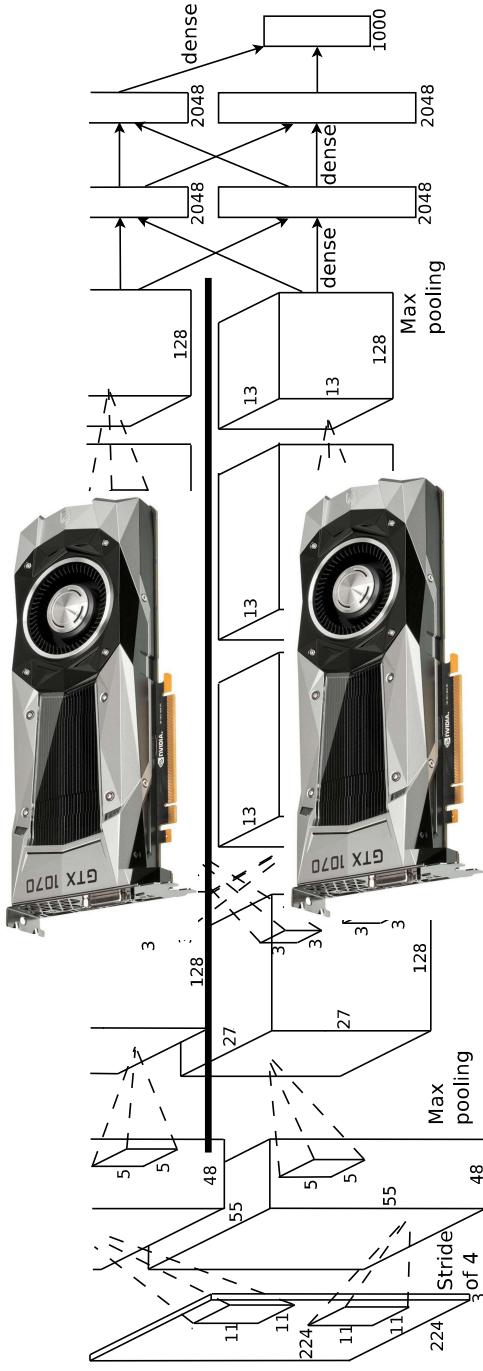
Model Parallelism: Split Model Across GPUs

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

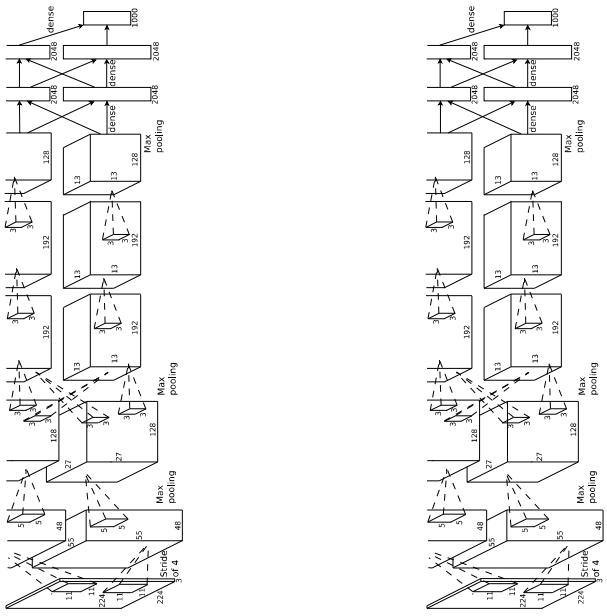


Model Parallelism: Split Model Across GPUs

Idea #2: Run parallel branches of model on different GPUs
Problem: Synchronizing across GPUs is expensive;
Need to communicate activations and grad activations



Data Parallelism: Copy Model on each GPU, split data



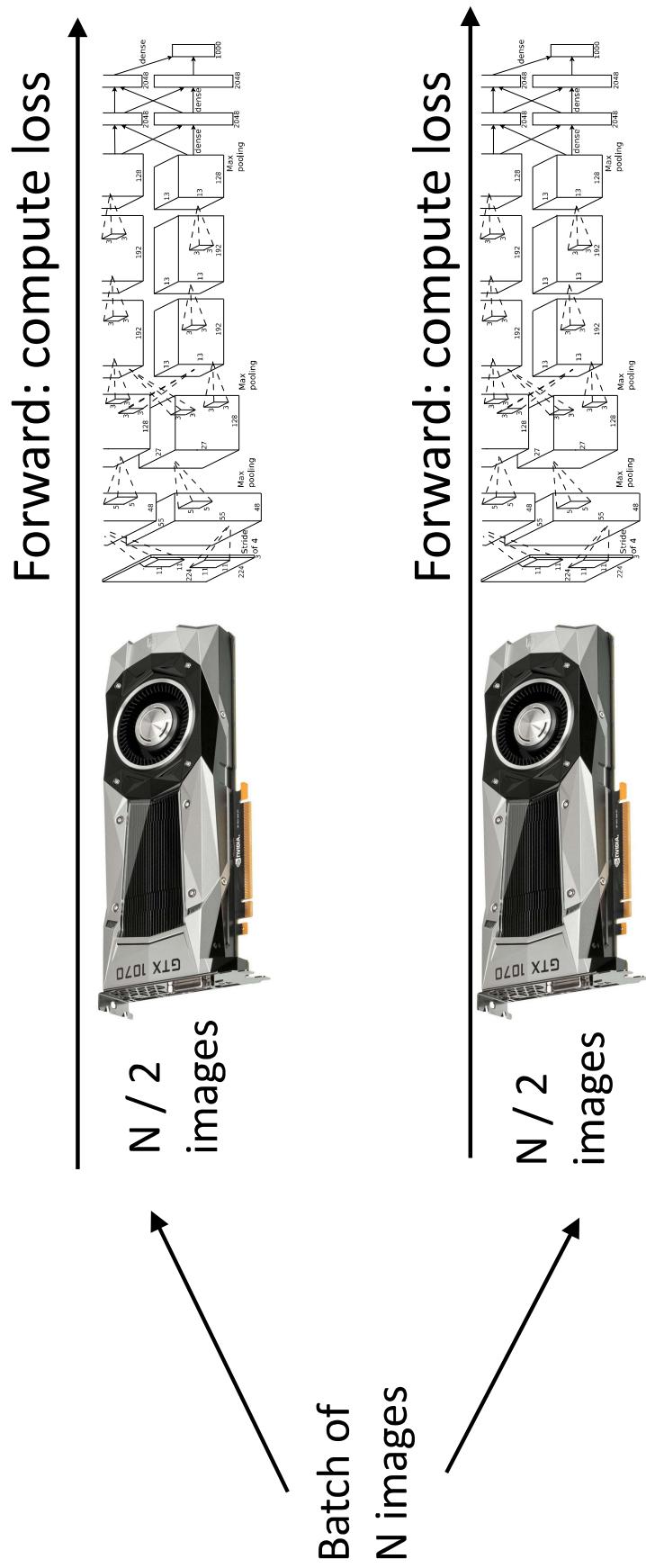
$N / 2$
images

Batch of
 N images

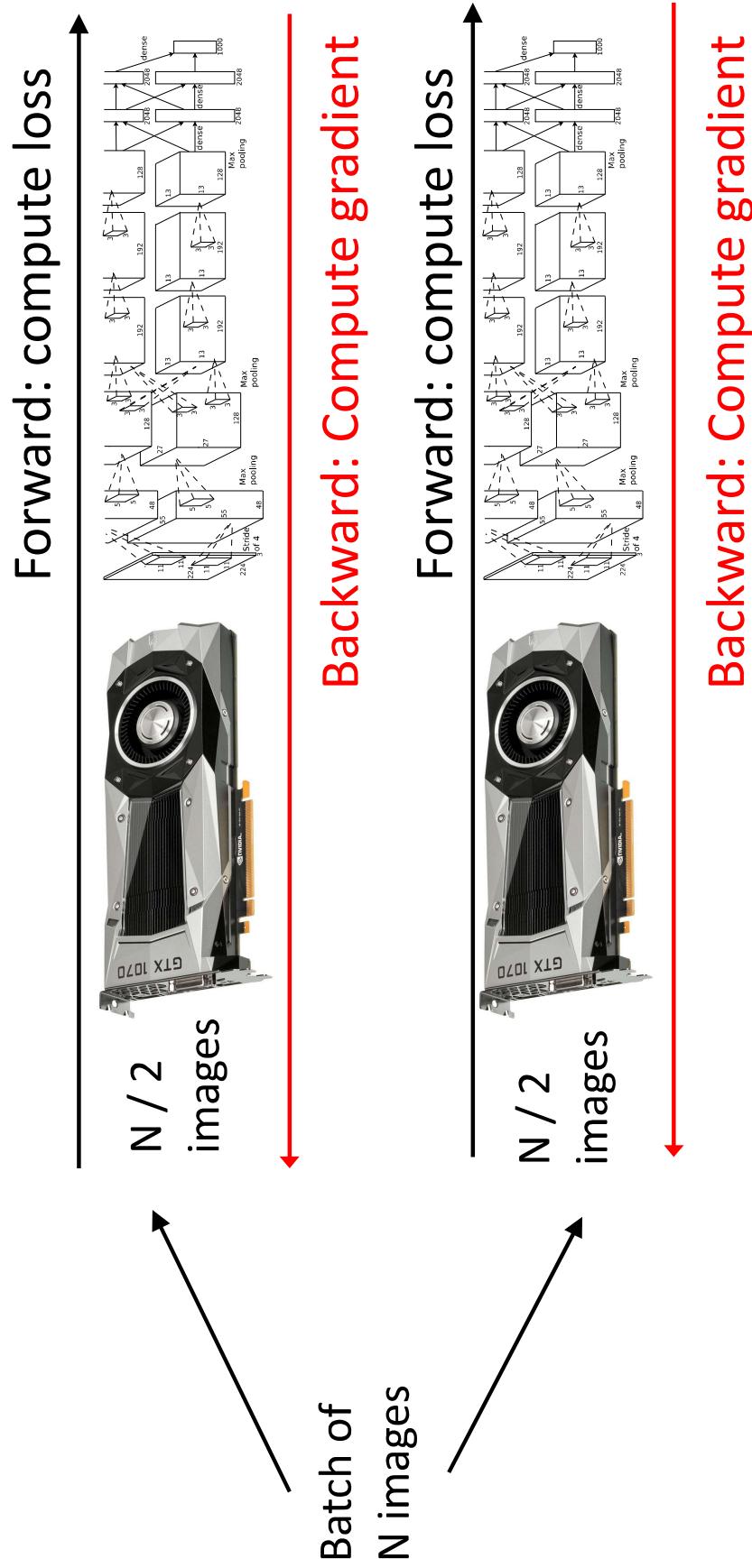


$N / 2$
images

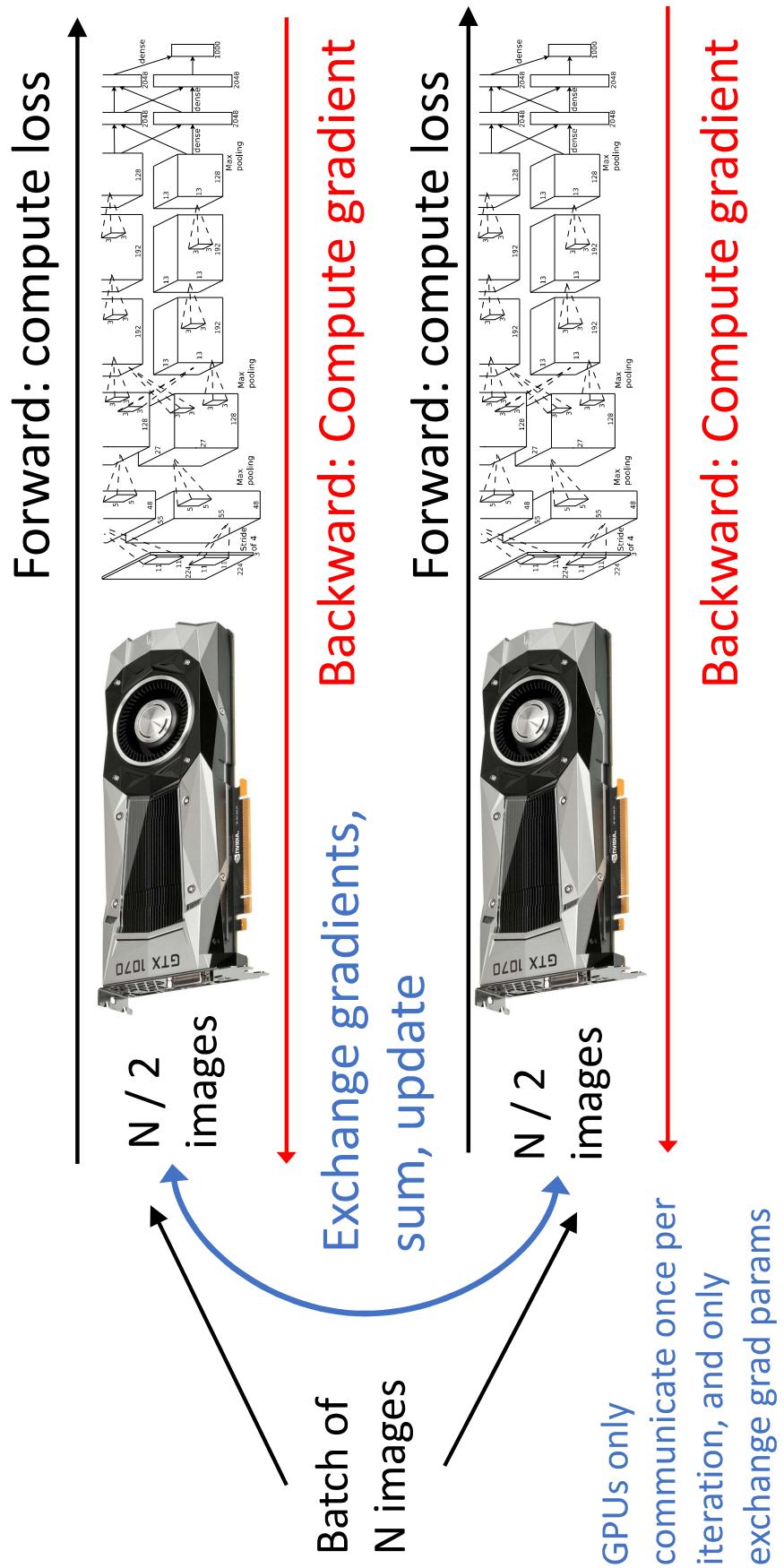
Data Parallelism: Copy Model on each GPU, split data



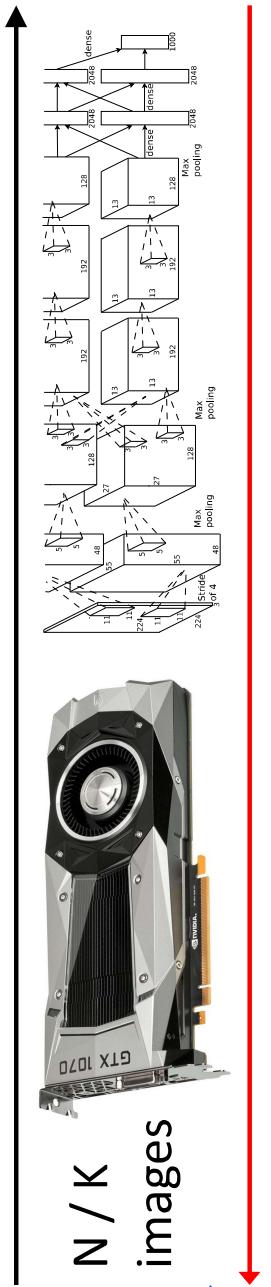
Data Parallelism: Copy Model on each GPU, split data



Data Parallelism: Copy Model on each GPU, split data



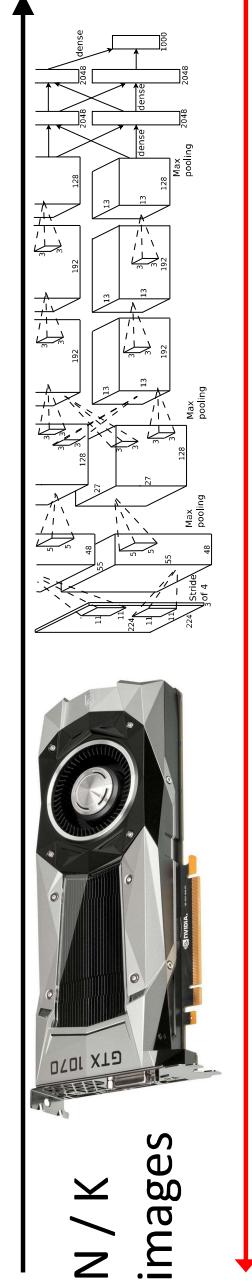
Data Parallelism: Copy Model on each GPU, split data



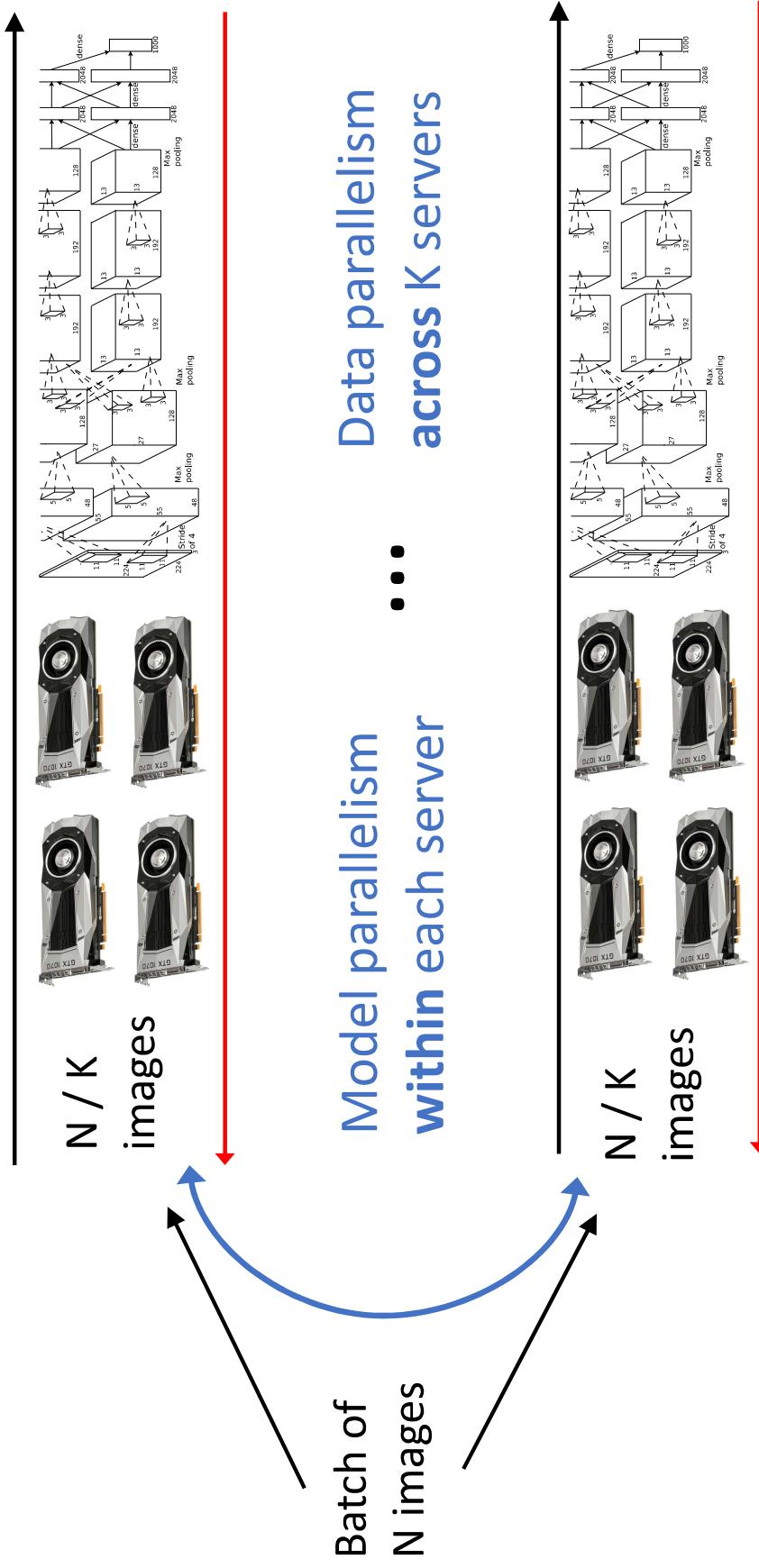
Scale up to K GPUs
(within and across
servers)

Main mechanism for
distributing across
many GPUs today

• • •



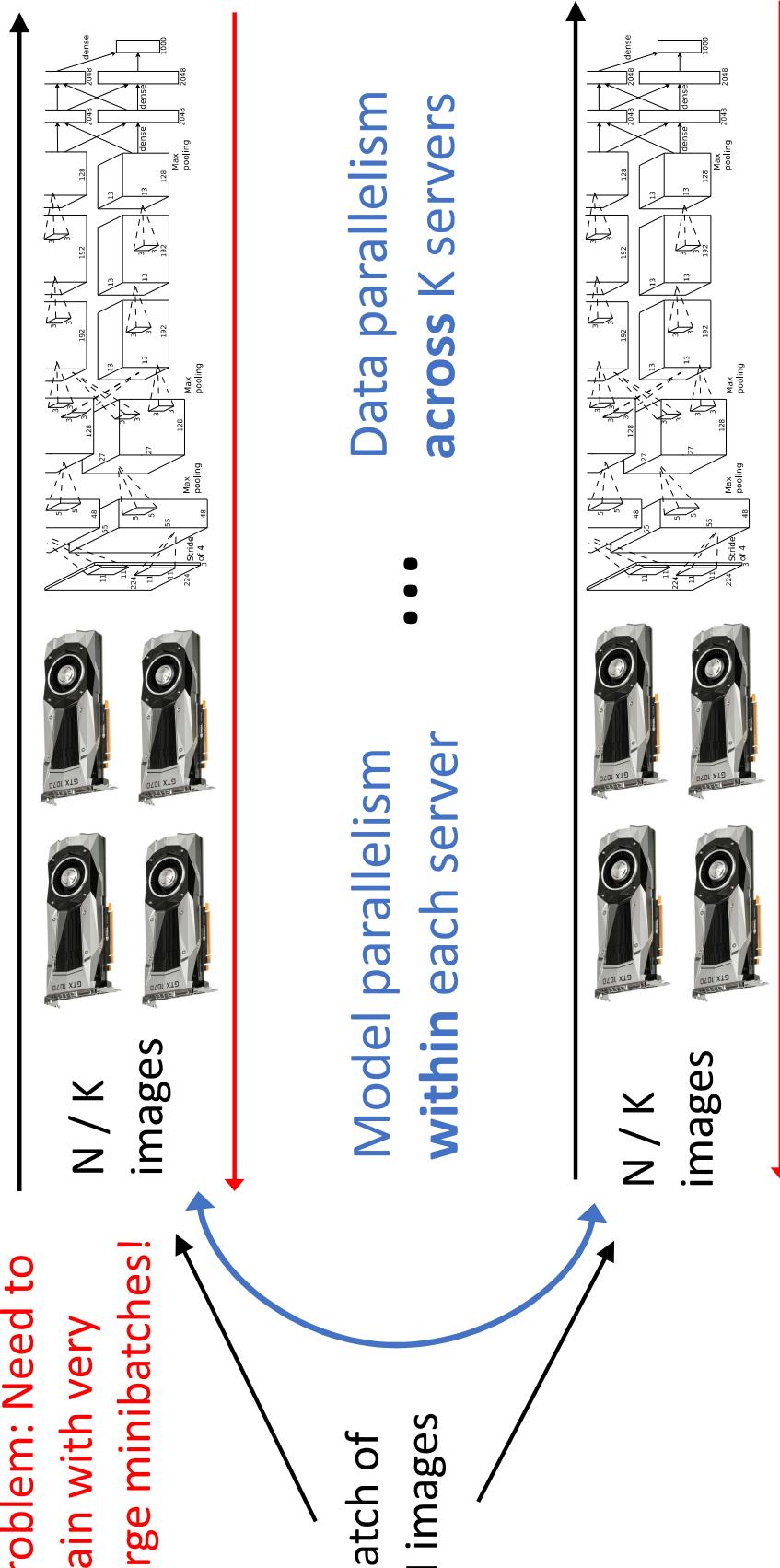
Mixed Model + Data Parallelism



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

Mixed Model + Data Parallelism

Problem: Need to train with very large minibatches!



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

Large-Batch Training

Justin Johnson

Lecture 11 - 88

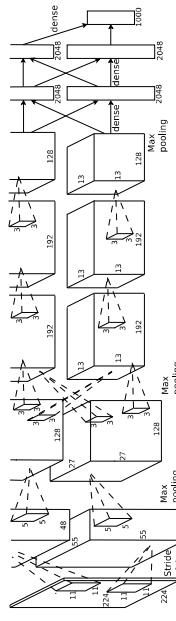
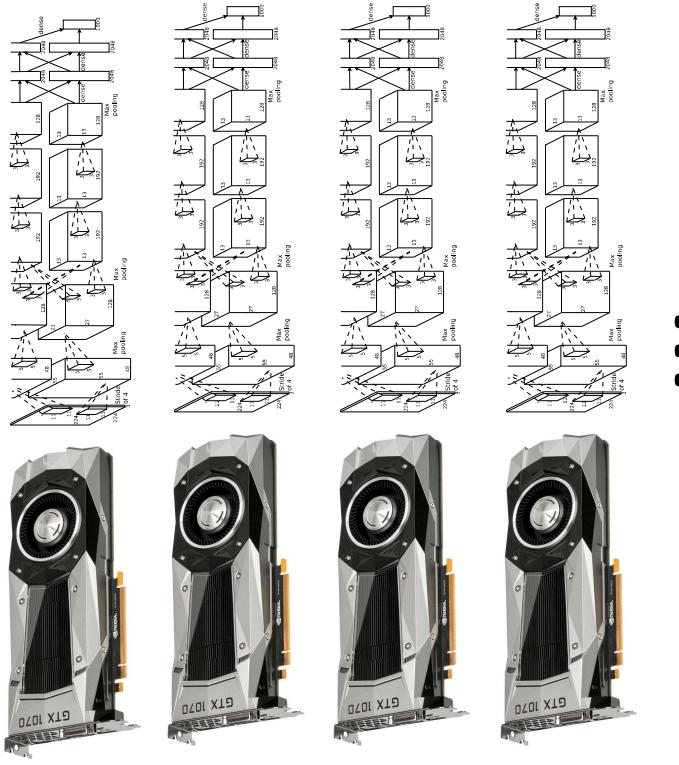
October 9, 2019

Large-Batch Training

Suppose we can train a good model with one GPU



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

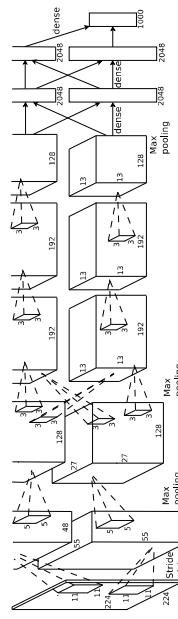
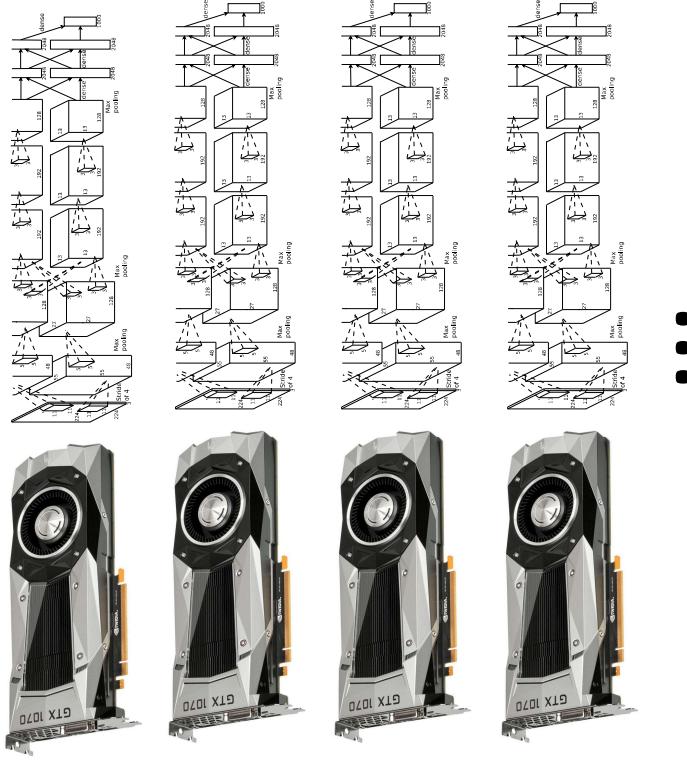


Large-Batch Training

Suppose we can train a good model with one GPU



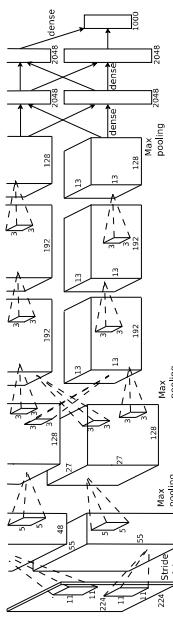
How to scale up to data-parallel training on K GPUs?



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

Large-Batch Training: Scale Learning Rates

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

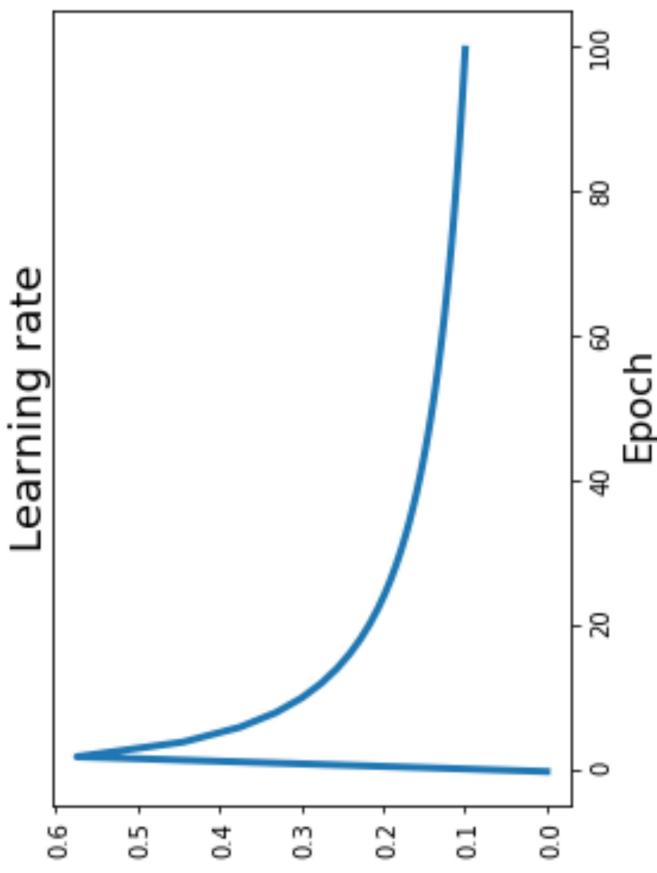


K-GPU model:
batch size KN, learning rate $K\alpha$



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**

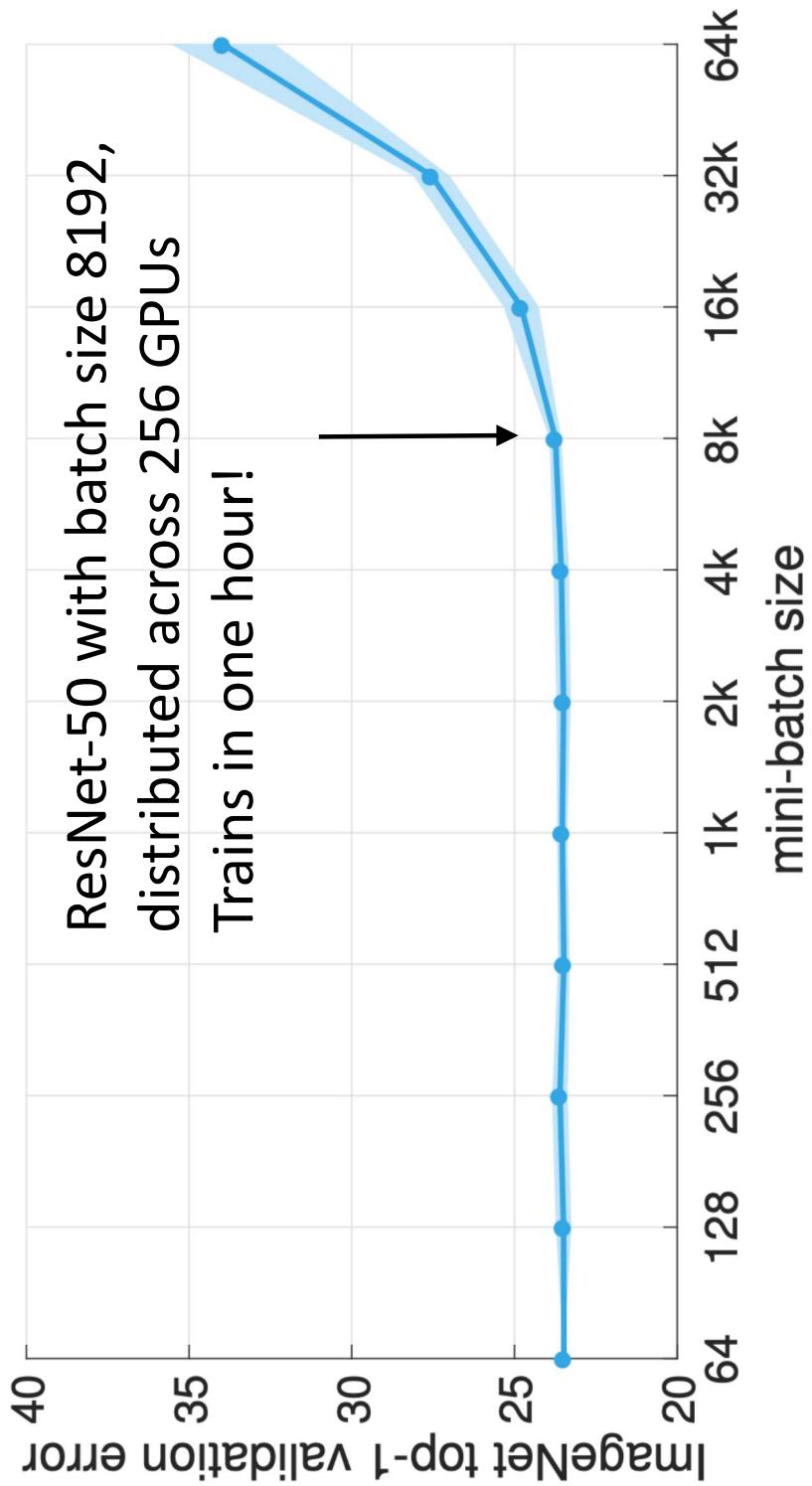
Goyal et al, “Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour”, arXiv 2017

Justin Johnson

Lecture 11 - 93

October 9, 2019

Large-Batch Training: ImageNet in One Hour!



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

Justin Johnson

Lecture 11 - 94

October 9, 2019

Large-Batch Training

Goyal et al, “Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour”, 2017
Batch size: 8192; 256 P100 GPUs; 1 hour

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

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

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

Recap

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

Last Time

Today

Next Time:
Recurrent Neural Networks