GEORG-AUGUST-UNIVERSITÄT
GÖTTINGEN

Deep Learning

Lecture 6: Convolutional neural networks (CNN)

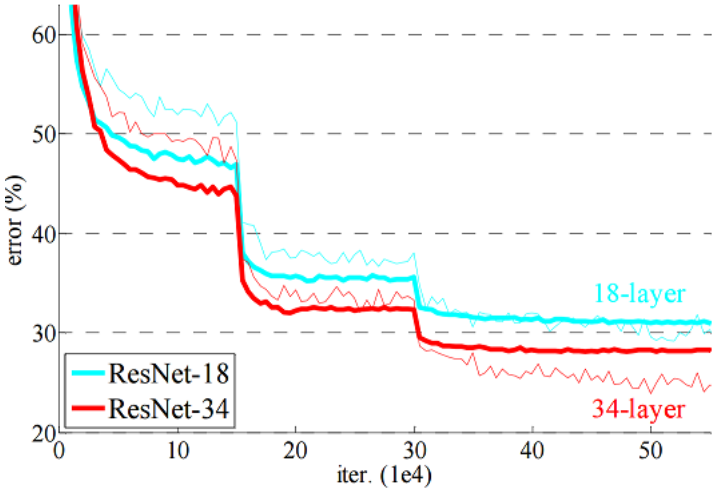
Alexander Ecker
Institut für Informatik, Uni Göttingen

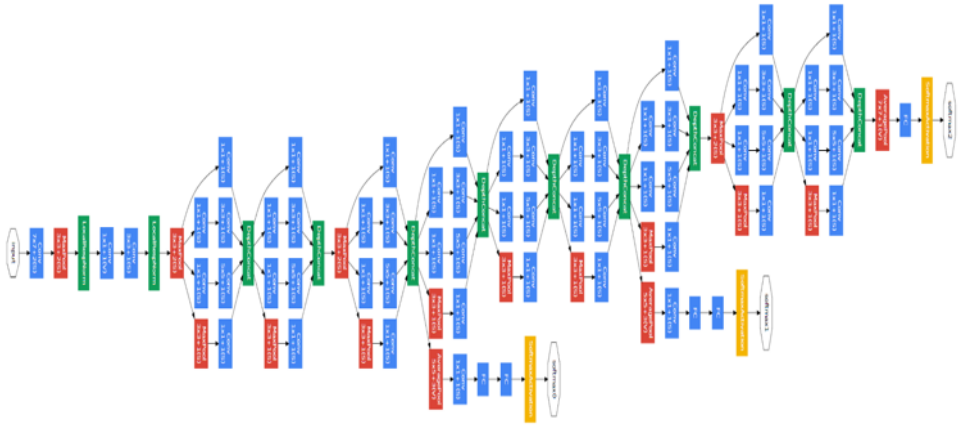
alxecker

https://alexanderecker.wordpress.com

– Credit: some of the slides based on Fei Fei Li, Justin Johnson and Serena Yeung's slides –

Agenda for today

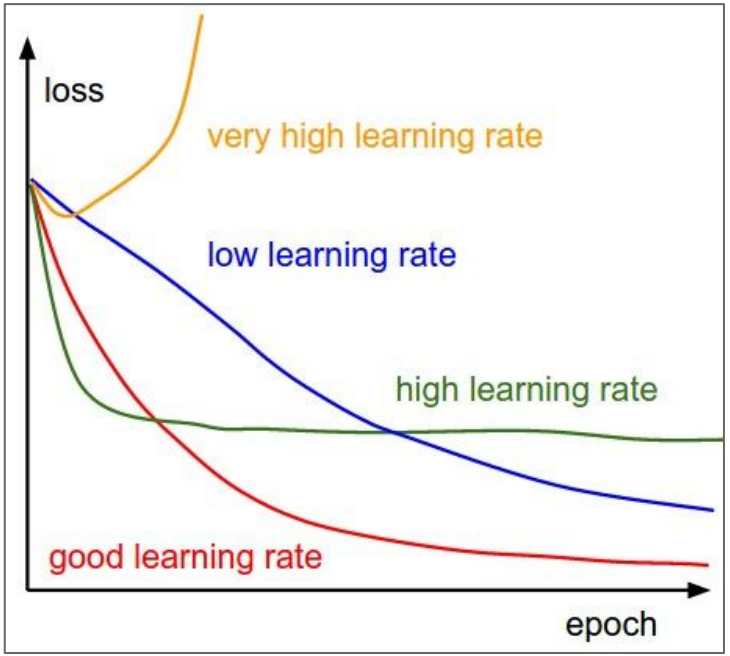




1. Babysitting neural network training
2. Modern CNN architectures

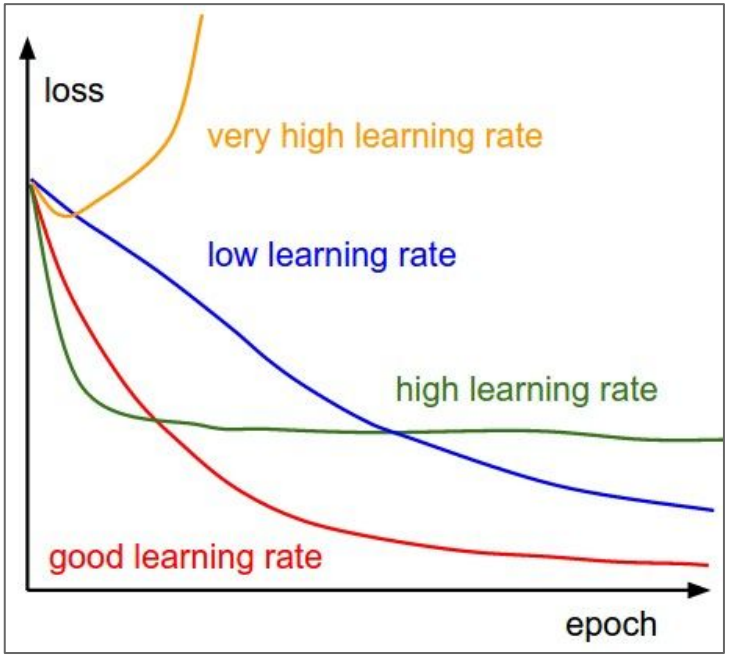
Lecture 8:
Training Neural Networks, Part 2

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.

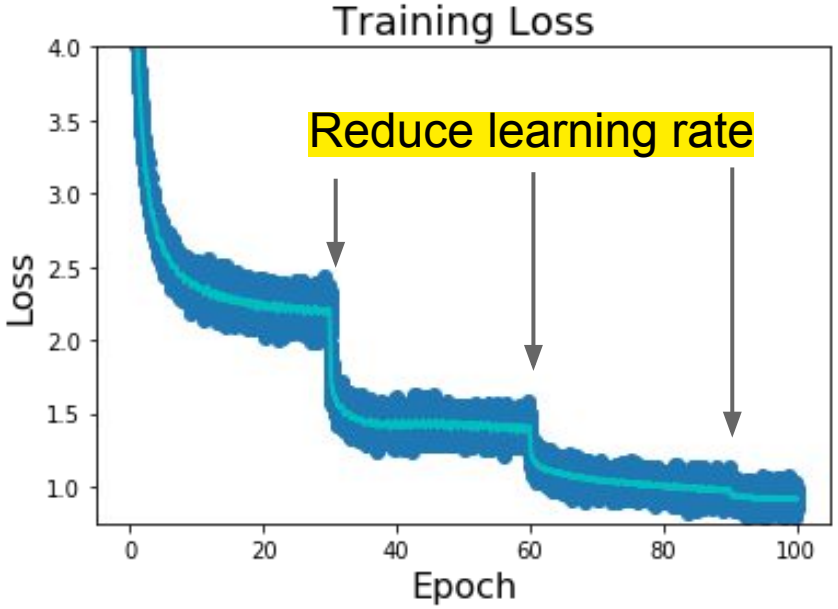


Q: Which one of these learning rates is best to use?

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

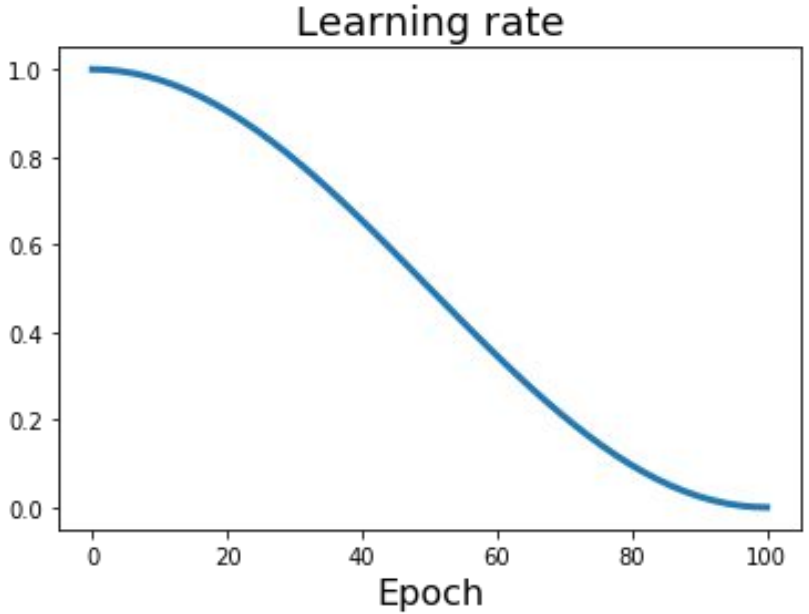
Do not use one fixed learning rate, change it all the time

Learning Rate Decay



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



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

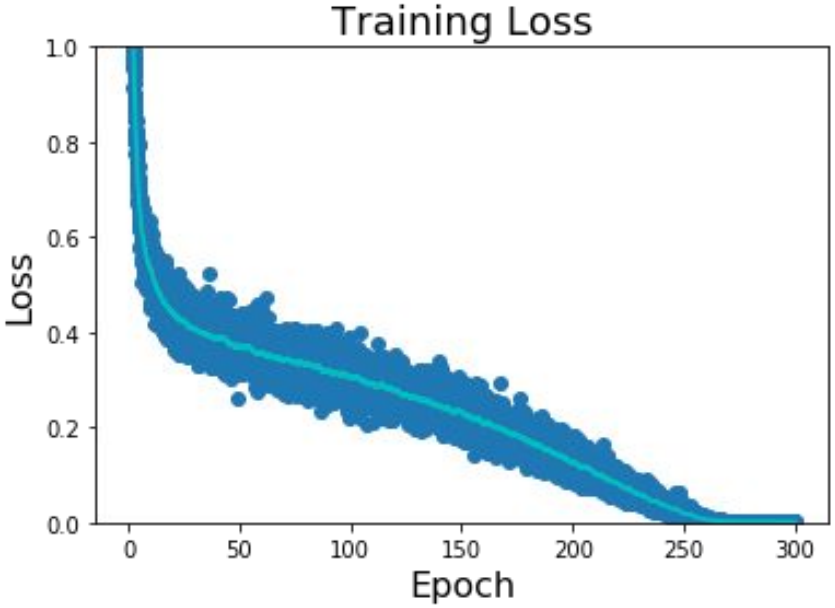
Cosine: $\alpha_t = \frac{1}{2}\alpha_0 \left(1 + \cos(t\pi/T)\right)$

the cosine Learning Rate is very popular (other idea than before)
- start with a fix LR = 1
- decrease the LR by the Cosine over time
- two hyperparameters: LR, T
- difficulty: set T at the beginning
-> see the following pages asw ell

α_0 : Initial learning rate
α_t : Learning rate at epoch t
T : Total number of epochs

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”, arXiv 2018
Child at al, “Generating Long Sequences with Sparse Transformers”, arXiv 2019

Learning Rate Decay



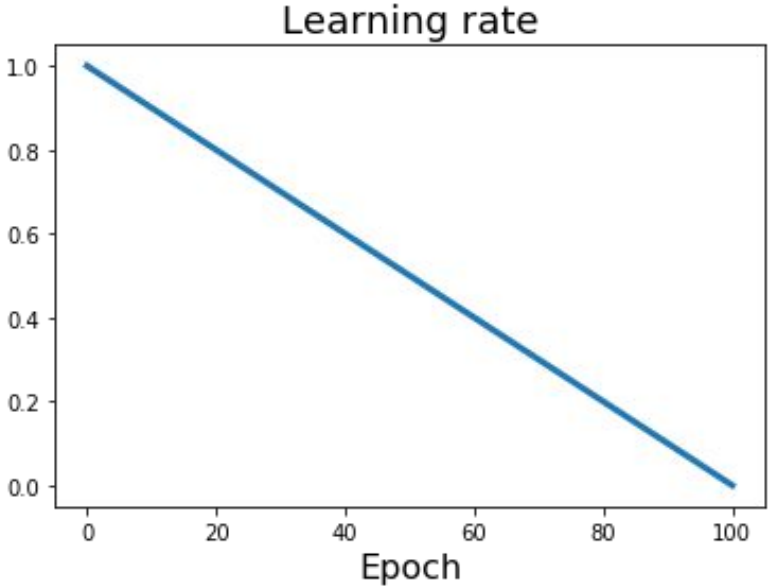
Step: Reduce learning rate at a few fixed points. E.g. for ResNets, multiply LR by 0.1 after epochs 30, 60, and 90.

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

α_0 : Initial learning rate
α_t : Learning rate at epoch t
T : Total number of epochs

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", arXiv 2018
Child at al, "Generating Long Sequences with Sparse Transformers", arXiv 2019

Learning Rate Decay



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

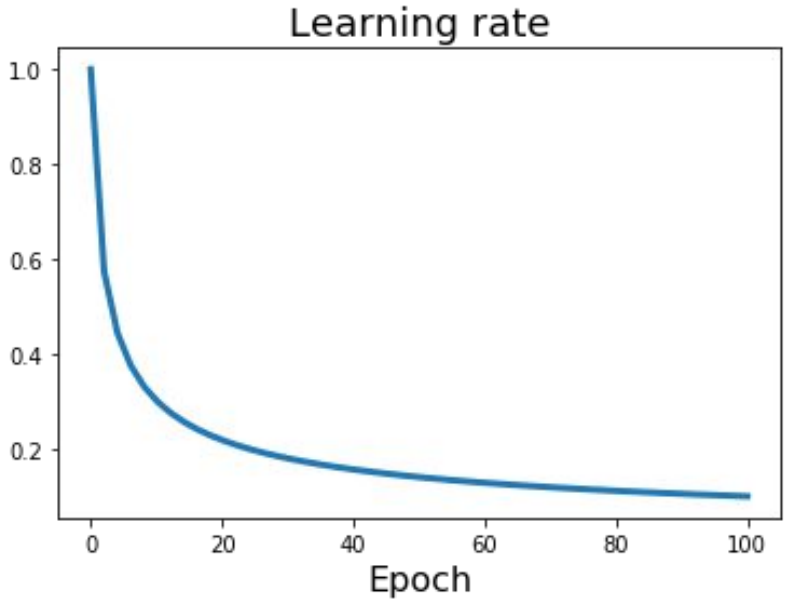
Cosine: $\alpha_t = \frac{1}{2}\alpha_0 \left(1 + \cos(t\pi/T)\right)$

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

α_0 : Initial learning rate
α_t : Learning rate at epoch t
T : Total number of epochs

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

Learning Rate Decay



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

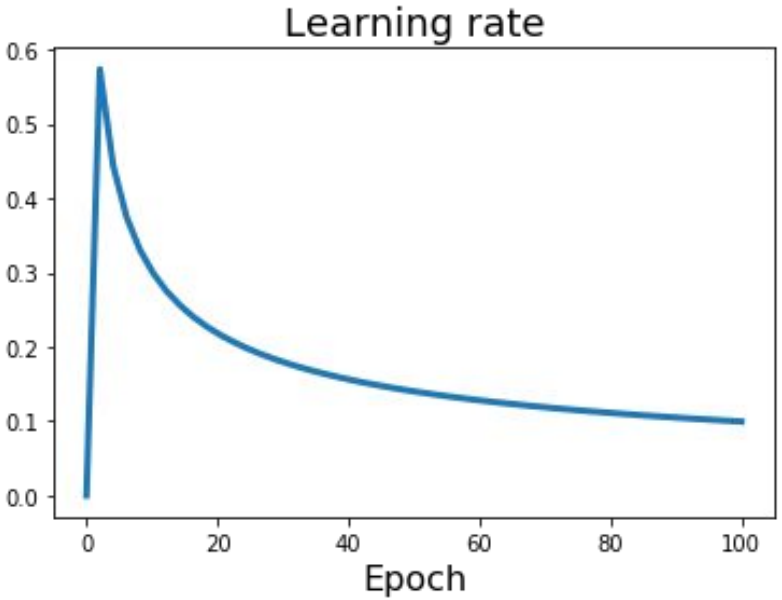
Linear: $\alpha_t = \alpha_0(1 - t/T)$

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

α_0 : Initial learning rate
α_t : Learning rate at epoch t
T : Total number of epochs

Vaswani et al, “Attention is all you need”, NIPS 2017

Learning Rate Decay: Linear Warmup



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

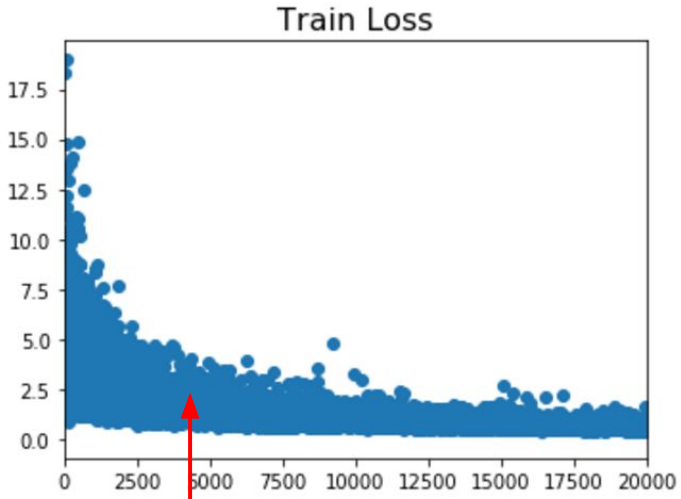
Empirical rule of thumb: If you increase the batch size by N, also scale the initial learning rate by N

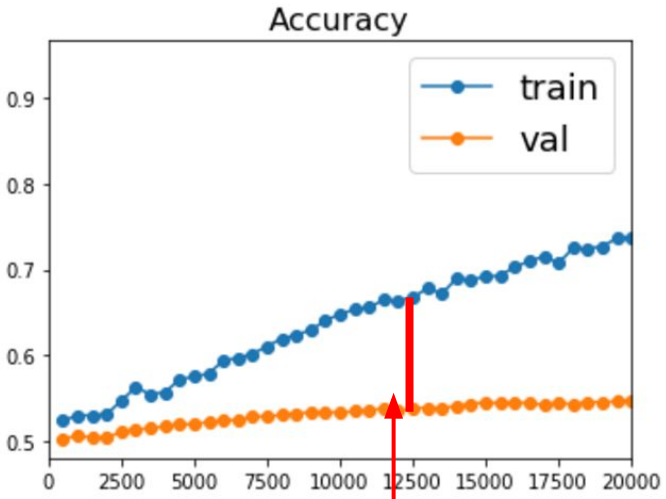
Goyal et al, “Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour”, arXiv 2017

In practice:

- **Adam** is a good default choice in many cases; it often works ok even with constant learning rate
- **SGD+Momentum** can outperform Adam but may require more tuning of LR and schedule
  - Try cosine schedule, very few hyperparameters!

Beyond Training Error

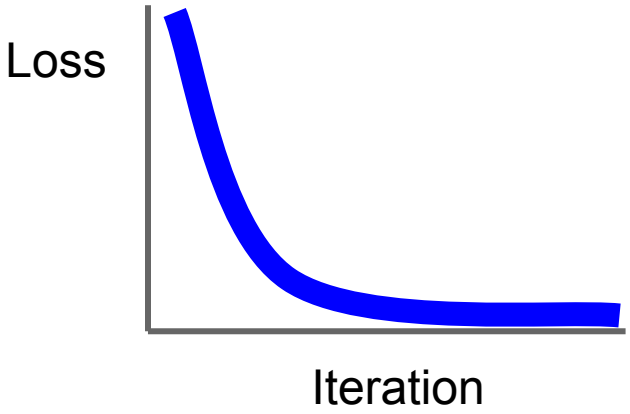


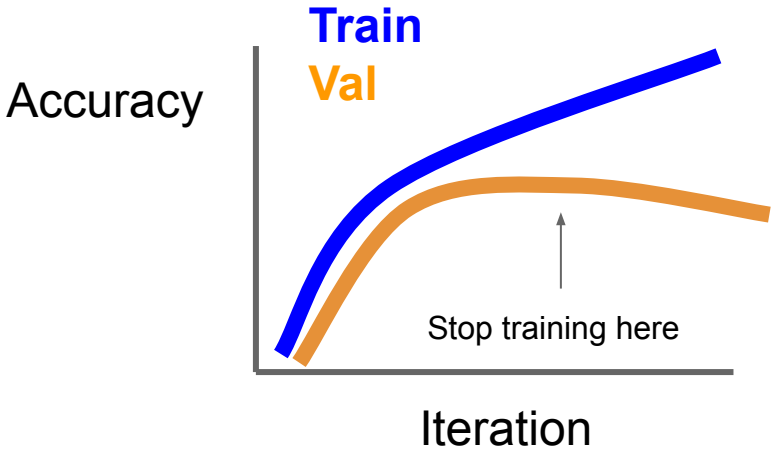


Better optimization algorithms help reduce training loss

But we really care about error on new data - how to reduce the gap?

Early Stopping: Always do this





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

Model Ensembles

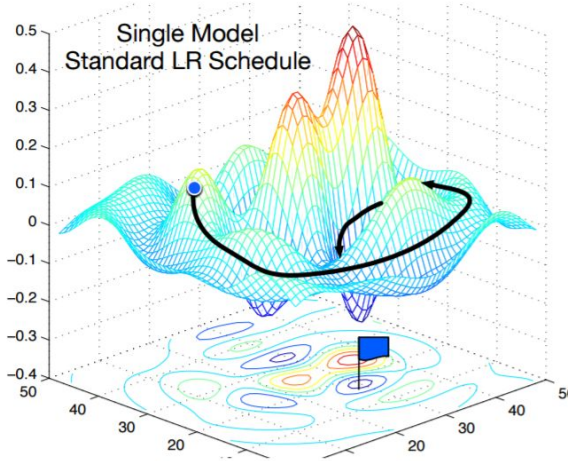
1. Train multiple independent models
2. At test time average their results

   (Take average of predicted probability distributions, then choose argmax)

Enjoy 2% extra performance

Model Ensembles: Tips and Tricks

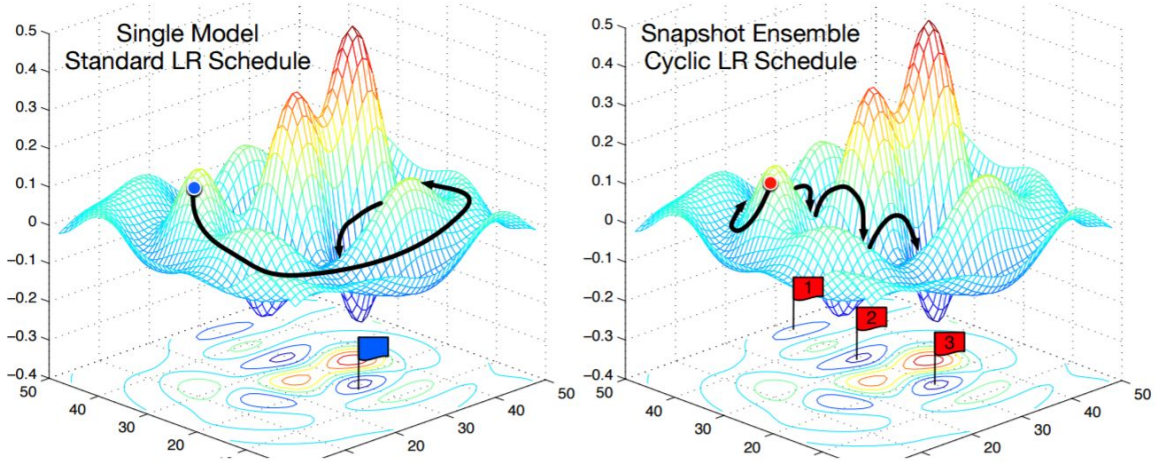
Instead of training independent models, use multiple snapshots of a single model during training!

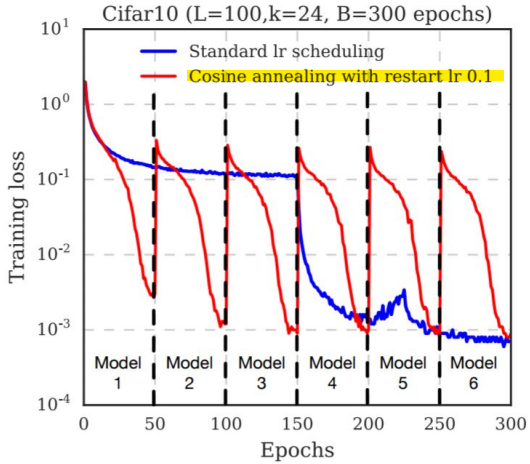


Loshchilov and Hutter, “SGDR: Stochastic gradient descent with restarts”, arXiv 2016
Huang et al, “Snapshot ensembles: train 1, get M for free”, ICLR 2017
Figures copyright Yixuan Li and Geoff Pleiss, 2017. Reproduced with permission.

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

Model Ensembles: Tips and Tricks

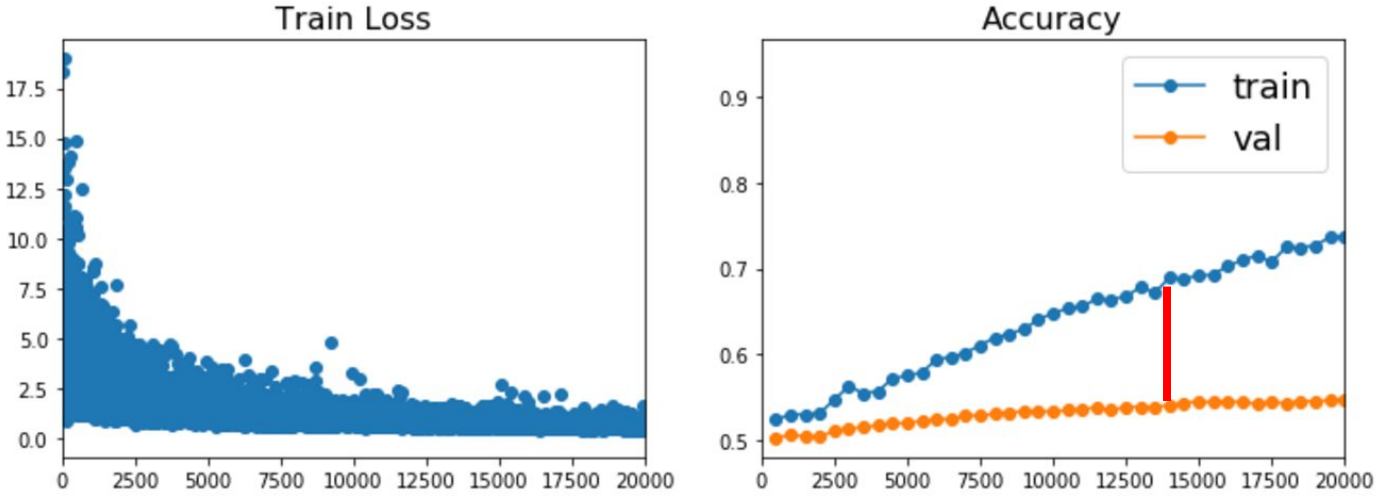
Instead of using actual parameter vector, keep a moving average of the parameter vector and use that at test time (Polyak averaging)

average about the model parameters
-> running average

```
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.

How to improve single-model performance?



Regularization

nichts zu der Folie gesagt, übersprungen

Regularization: Add term to loss

$$L = \frac{1}{N}\sum_{i=1}^{N}\sum_{j\neq y_i}\max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1) + \boxed{\lambda R(W)}$$

In common use:

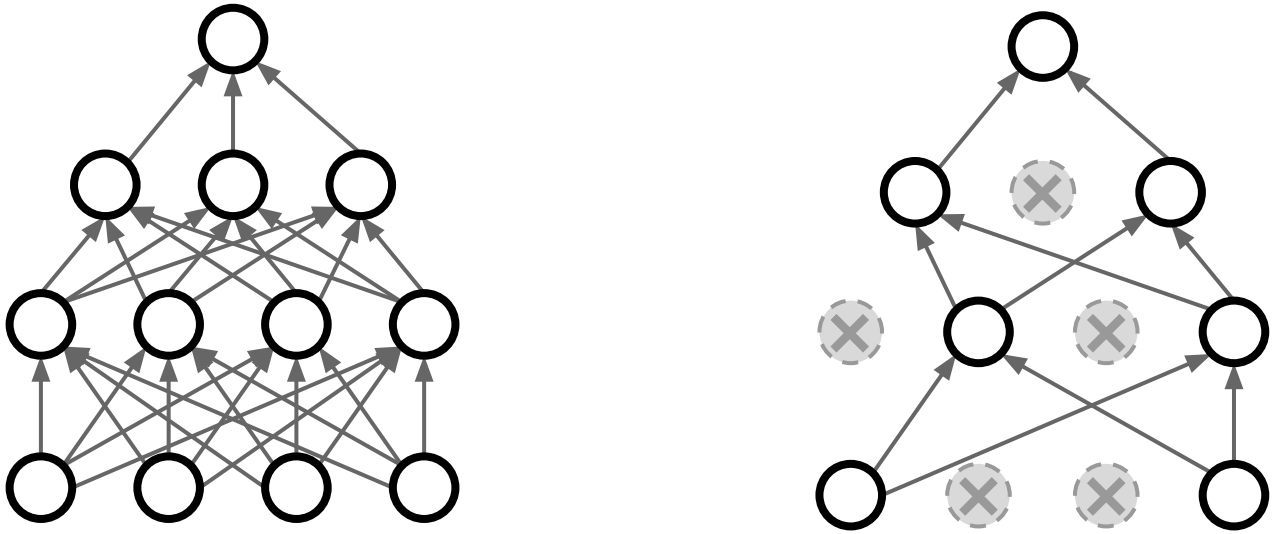
L2 regularization $R(W) = \sum_k \sum_l W_{k,l}^2$ (Weight decay)

L1 regularization $R(W) = \sum_k \sum_l |W_{k,l}|$

Elastic net (L1 + L2) $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$

Regularization: Dropout

In each forward pass, randomly set some neurons to zero
Probability of dropping is a hyperparameter; 0.5 is common



Srivastava et al, “Dropout: A simple way to prevent neural networks from overfitting”, JMLR 2014

Regularization: A common pattern

Training: Add some kind of randomness

$$y = f_W(x, z)$$

Testing: Average out randomness (sometimes approximate)

$$y = f(x) = E_z\big[f(x, z)\big] = \int p(z) f(x, z) dz$$

Regularization: A common pattern

Training: Add some kind of randomness

$$y = f_W(x, z)$$

Testing: Average out randomness (sometimes approximate)

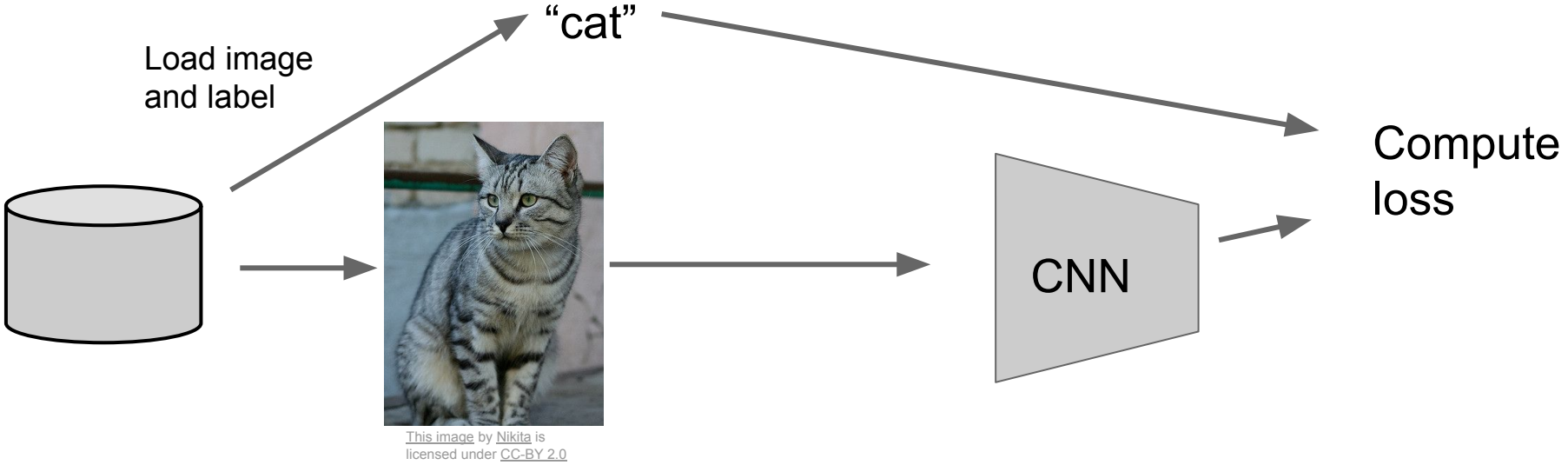
$$y = f(x) = E_z\big[f(x, z)\big] = \int p(z) f(x, z) dz$$

Example: Batch Normalization

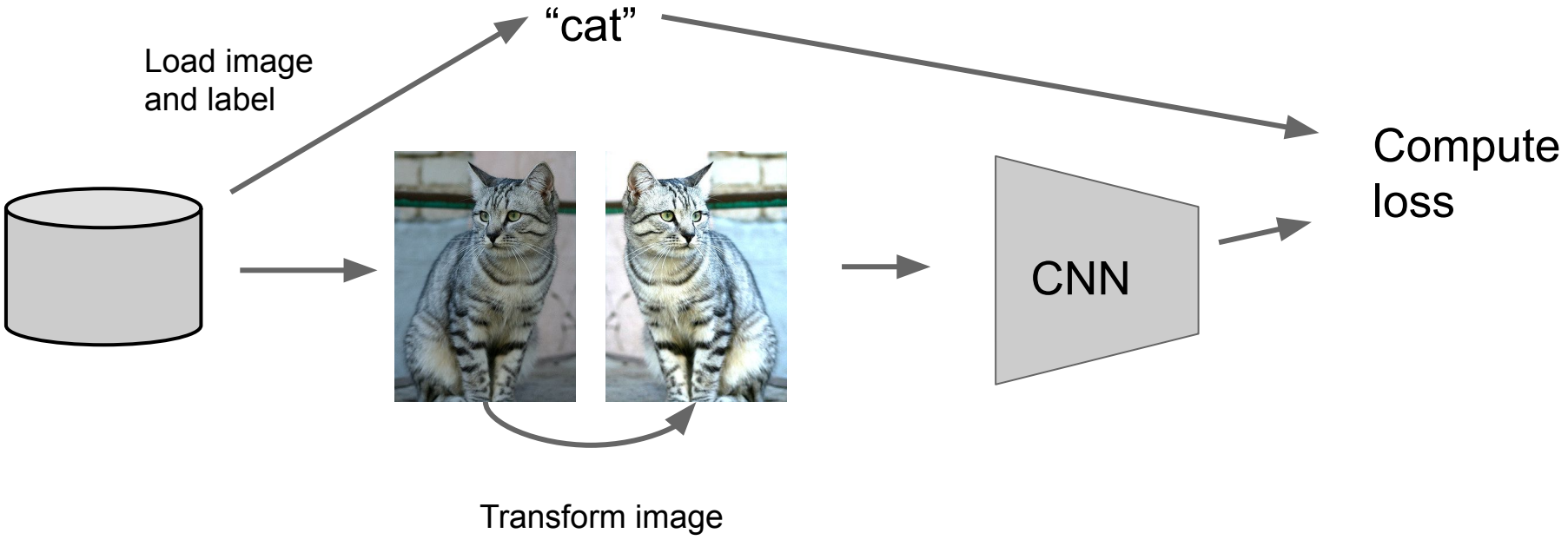
Training: Normalize using stats from random minibatches

Testing: Use fixed stats to normalize

Regularization: Data Augmentation



Regularization: Data Augmentation



Data Augmentation
Horizontal Flips

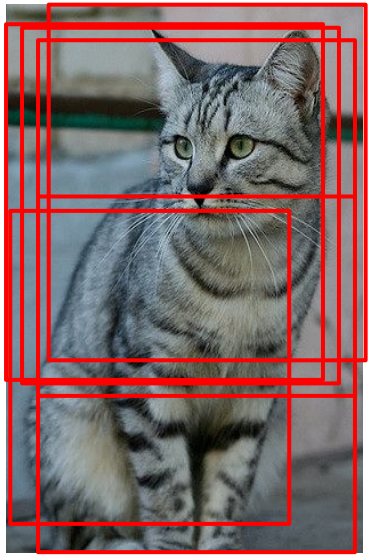
Data Augmentation

Random crops and scales

Training: sample random crops / scales

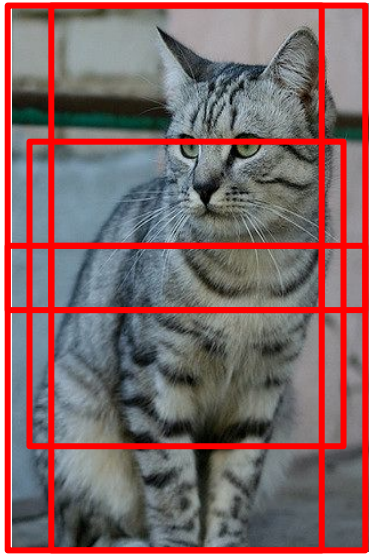
ResNet:

1. Pick random L in range [256, 480]
2. Resize training image, short side = L
3. Sample random 224 x 224 patch



Data Augmentation

Random crops and scales



Training: sample random crops / scales

ResNet:

1. Pick random L in range [256, 480]
2. Resize training image, short side = L
3. Sample random 224 x 224 patch

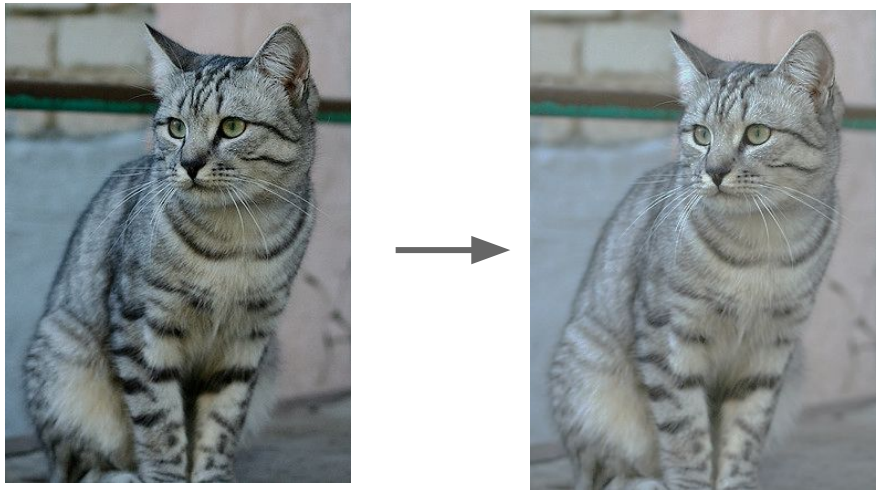
Testing: average a fixed set of crops

ResNet:

1. Resize image at 5 scales: {224, 256, 384, 480, 640}
2. For each size, use 10 224 x 224 crops: 4 corners + center, + flips

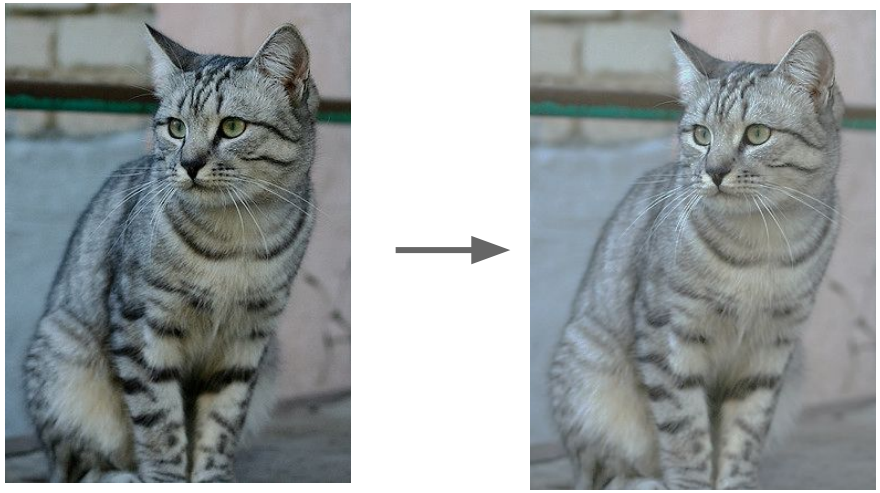
Data Augmentation
Color Jitter

Simple: Randomize
contrast and brightness



Data Augmentation
Color Jitter

Simple: Randomize contrast and brightness



More Complex:

1. Apply PCA to all [R, G, B] pixels in training set
2. Sample a “color offset” along principal component directions
3. Add offset to all pixels of a training image

(As seen in *[Krizhevsky et al. 2012],* ResNet, etc)

Data Augmentation

Get creative for your problem!

Random mix/combinations of :

- translation
- rotation
- stretching
- shearing,
- lens distortions, … (go crazy)

Regularization: A common pattern

Training: Add random noise

Testing: Marginalize over the noise

Examples:

Dropout

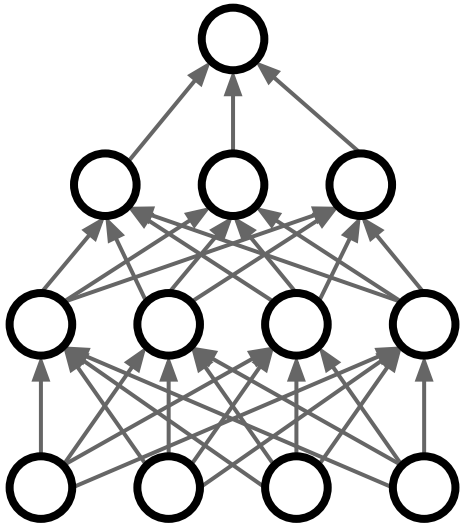
Batch Normalization

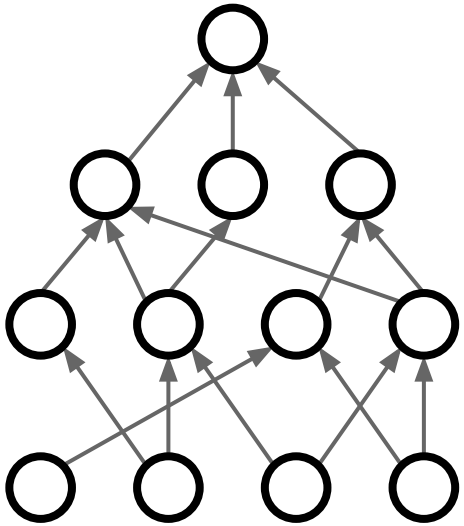
Data Augmentation

Regularization: DropConnect

Training: Drop connections between neurons (set weights to 0)
Testing: Use all the connections

Examples:
Dropout
Batch Normalization
Data Augmentation
DropConnect





Wan et al, “Regularization of Neural Networks using DropConnect”, ICML 2013

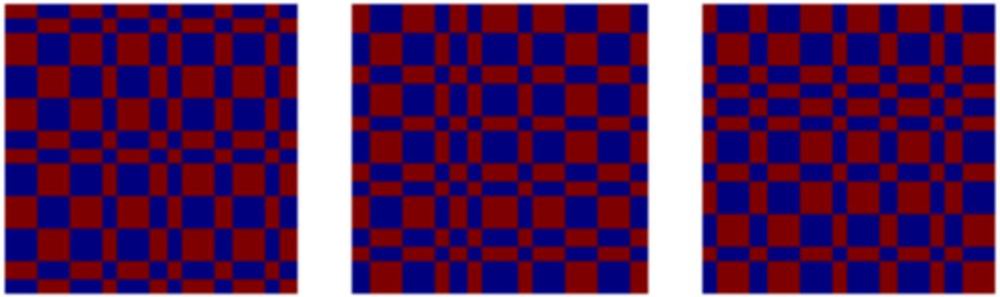
Regularization: Fractional Pooling

Training: Use randomized pooling regions pooling regions are identical, but do not have the same shape

Testing: Average predictions from several regions

Examples:

Dropout
Batch Normalization
Data Augmentation
DropConnect
Fractional Max Pooling



Graham, “Fractional Max Pooling”, arXiv 2014

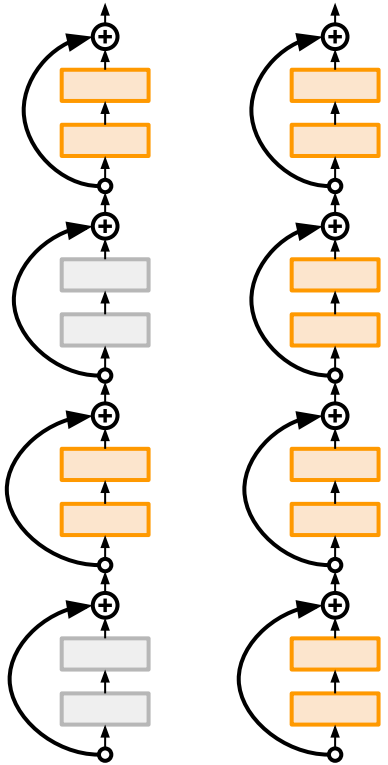
Regularization: Stochastic Depth

Training: Skip some layers in the network

Testing: Use all the layer

Examples:

Dropout
Batch Normalization
Data Augmentation
DropConnect
Fractional Max Pooling
Stochastic Depth



Huang et al, “Deep Networks with Stochastic Depth”, ECCV 2016

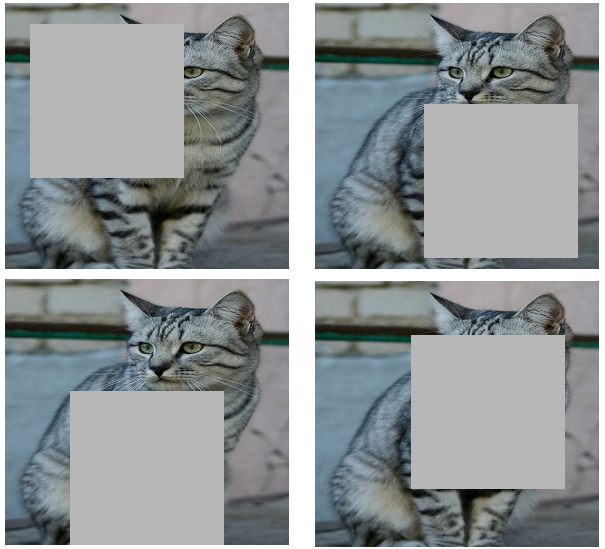
Regularization: Cutout

Training: Set random image regions to zero

Testing: Use full image

Examples:

Dropout
Batch Normalization
Data Augmentation
DropConnect
Fractional Max Pooling
Stochastic Depth
Cutout



Works very well for small datasets like CIFAR, less common for large datasets like ImageNet

DeVries and Taylor, “Improved Regularization of Convolutional Neural Networks with Cutout”, arXiv 2017

Regularization: Mixup

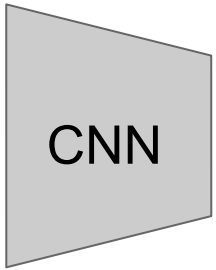
Training: Train on random blends of images

Testing: Use original images

Examples:

Dropout
Batch Normalization
Data Augmentation
DropConnect
Fractional Max Pooling
Stochastic Depth
Cutout
Mixup

Randomly blend the pixels of pairs of training images, e.g. 40% cat, 60% dog



Target label:
cat: 0.4
dog: 0.6

Zhang et al, "*mixup*: Beyond Empirical Risk Minimization", ICLR 2018

Regularization

Training: Add random noise

Testing: Marginalize over the noise

Examples:

Dropout

Batch Normalization

Data Augmentation

DropConnect

Fractional Max Pooling

Stochastic Depth

Cutout

Mixup

- Consider dropout for large fully-connected layers
- Batch normalization and data augmentation almost always a good idea
- Try cutout and mixup especially for small classification datasets

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

Compute the loss, aber initalisation the model, without training the model
- compute it again after some training rounds
- if the loss is not improving, the model is not learning anything
-> think about the model again, does it make sanse?

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

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

use logarithmic steps

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

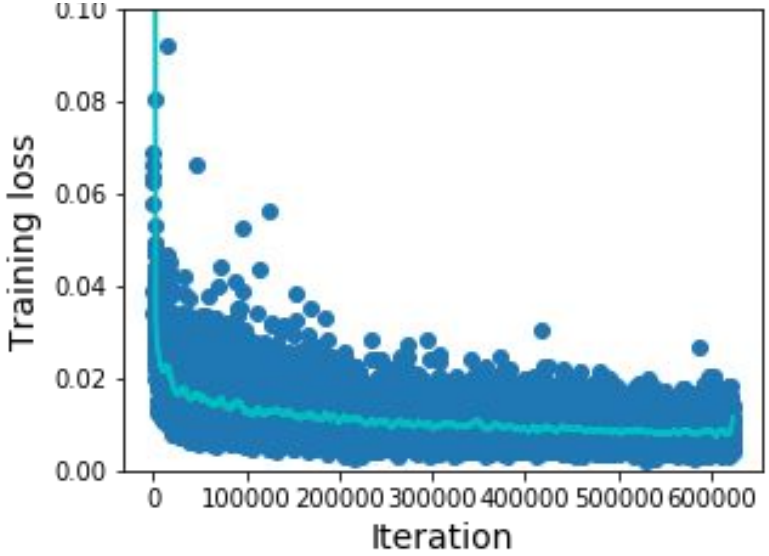
Pick best models from Step 4, train them for longer (~10-20 epochs) without learning rate decay

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 loss curves

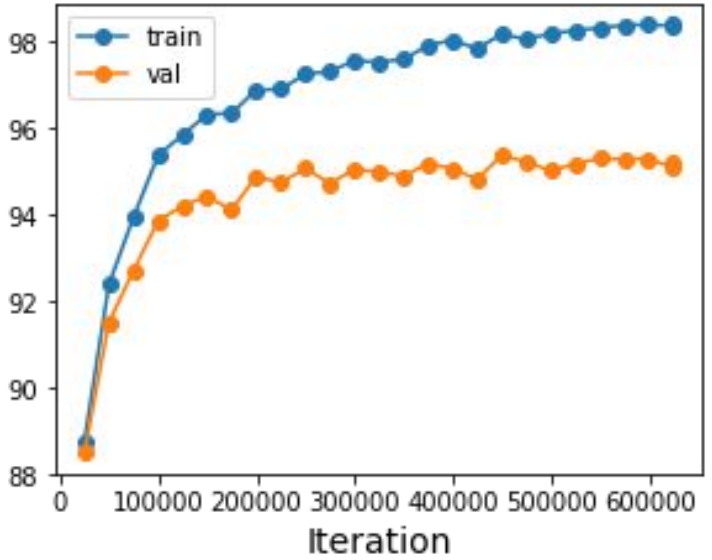
Look at learning curves!

Training Loss

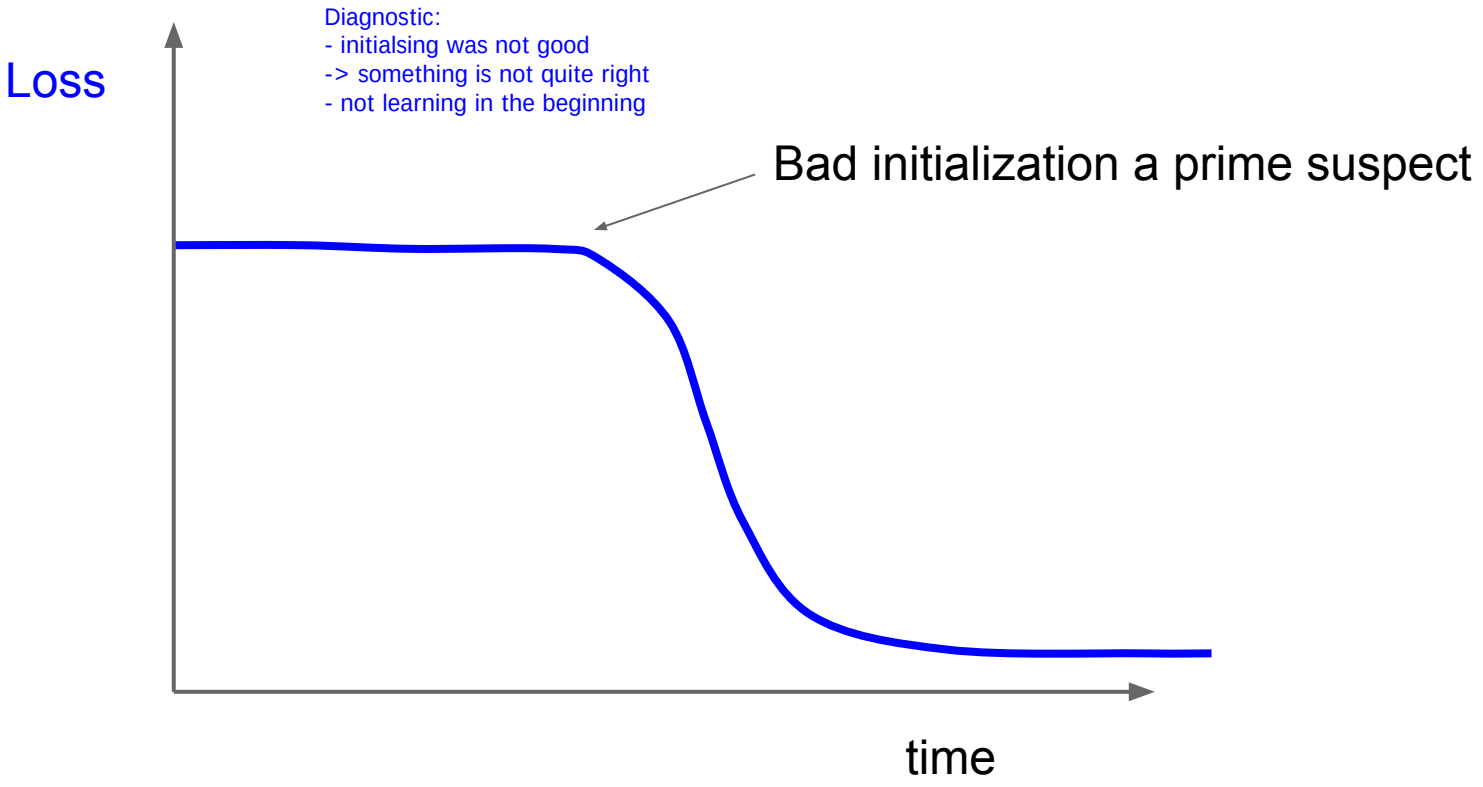


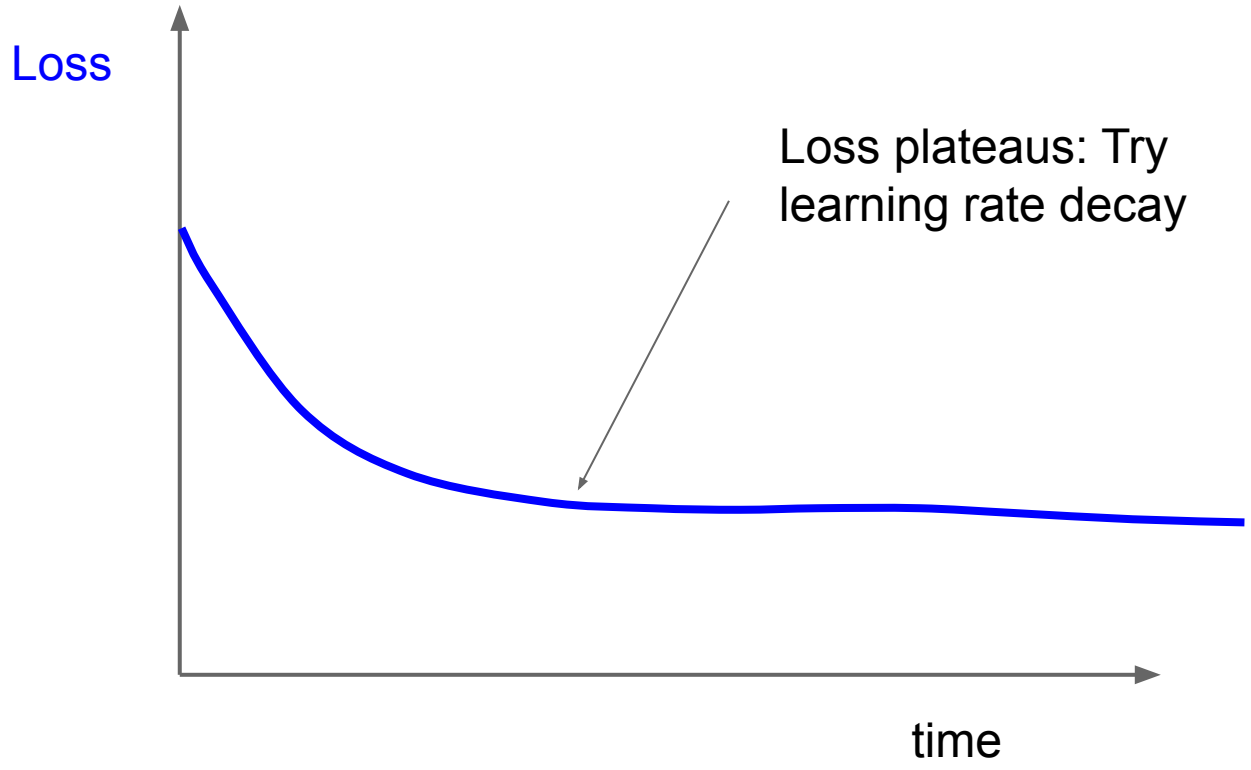
Losses may be noisy, use a scatter plot and also plot moving average to see trends better

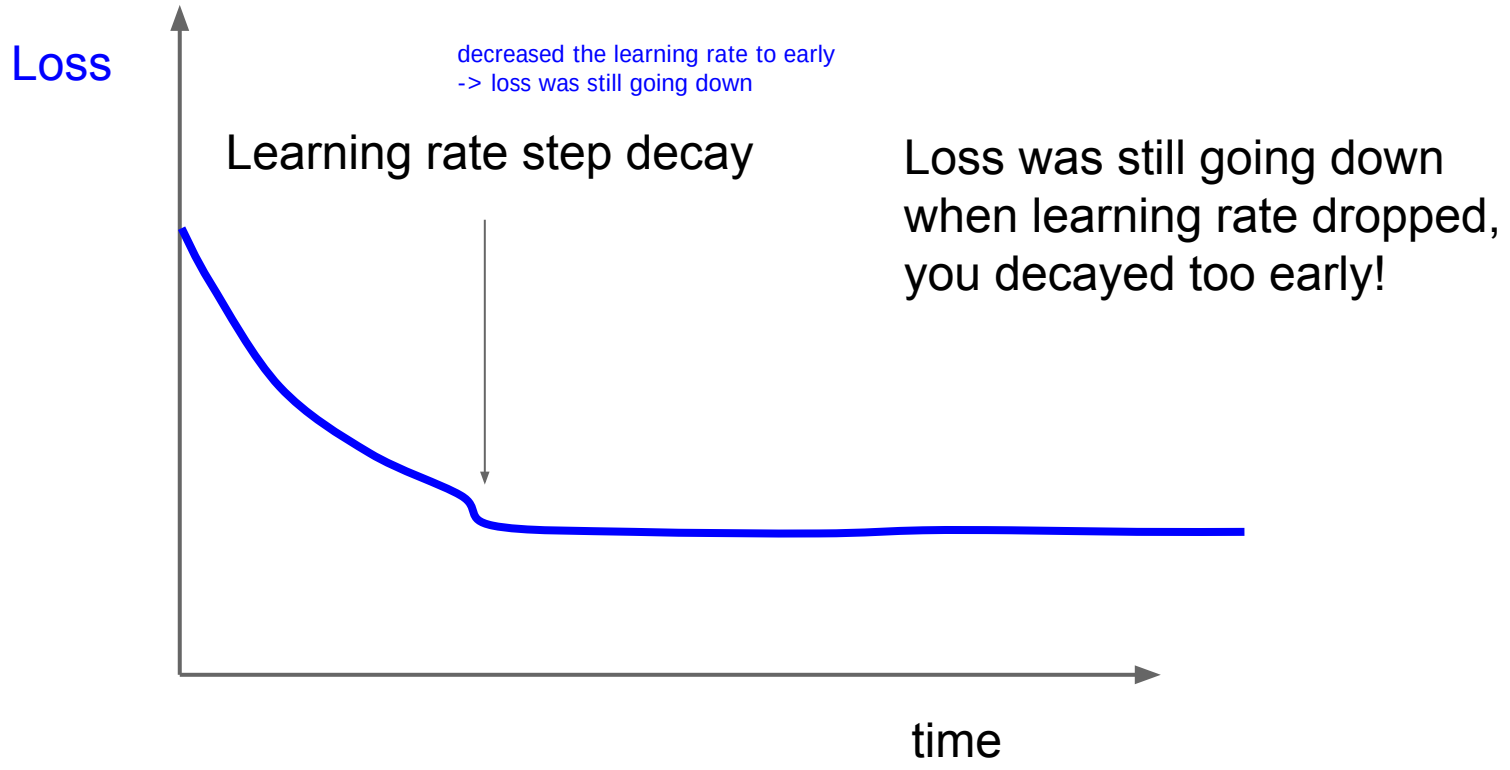
Train / Val Accuracy

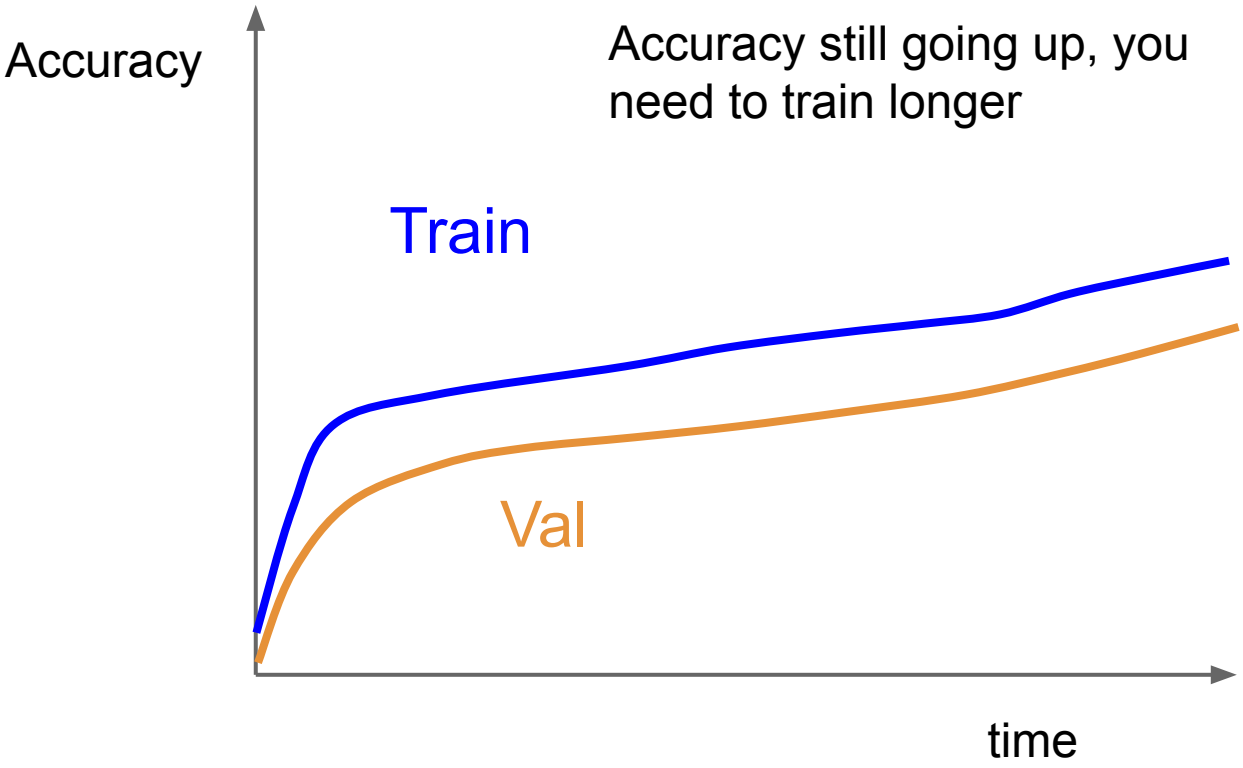


the accurancy of the train and val set, deviate, but both increase
-> slowly overfitting

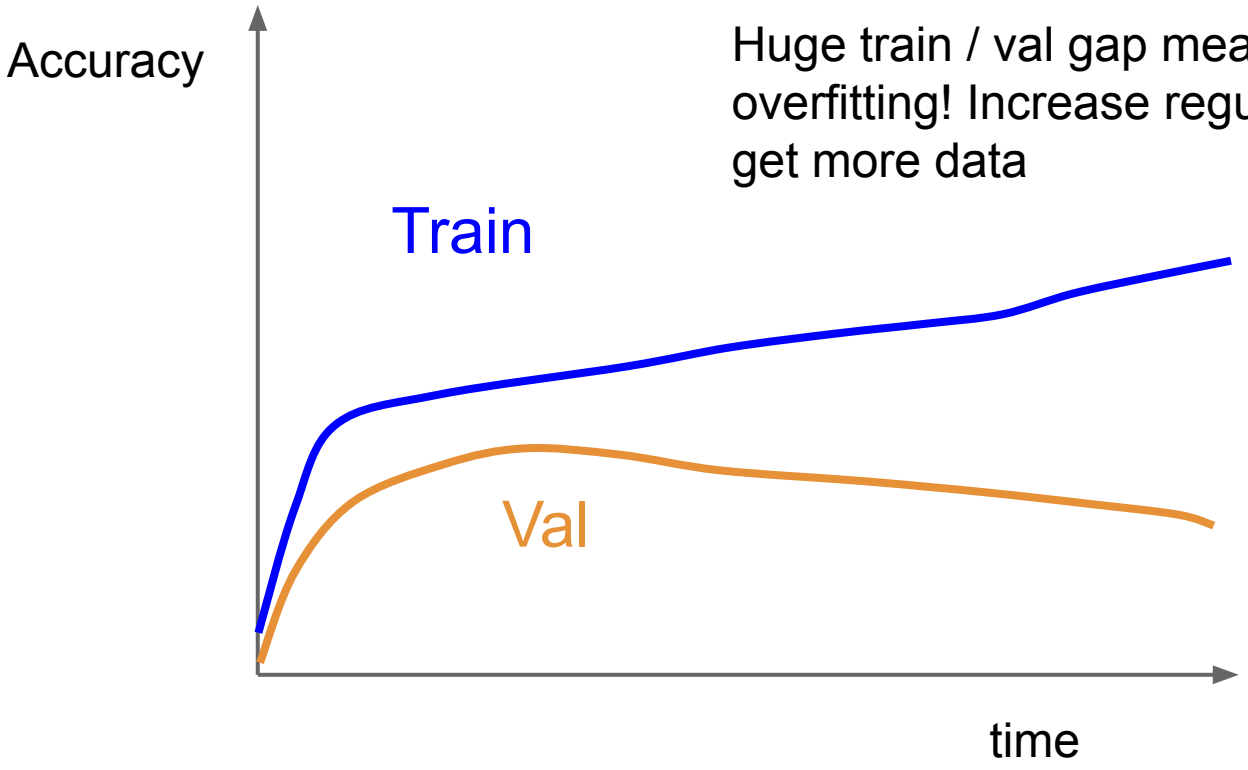

Loss
Diagnostic:
- initialsing was not good
-> something is not quite right
- not learning in the beginning
Bad initialization a prime suspect
time


Loss
Loss plateaus: Try
learning rate decay
time


Loss
decreased the learning rate to early
-> loss was still going down
Learning rate step decay
Loss was still going down
when learning rate dropped,
you decayed too early!
time

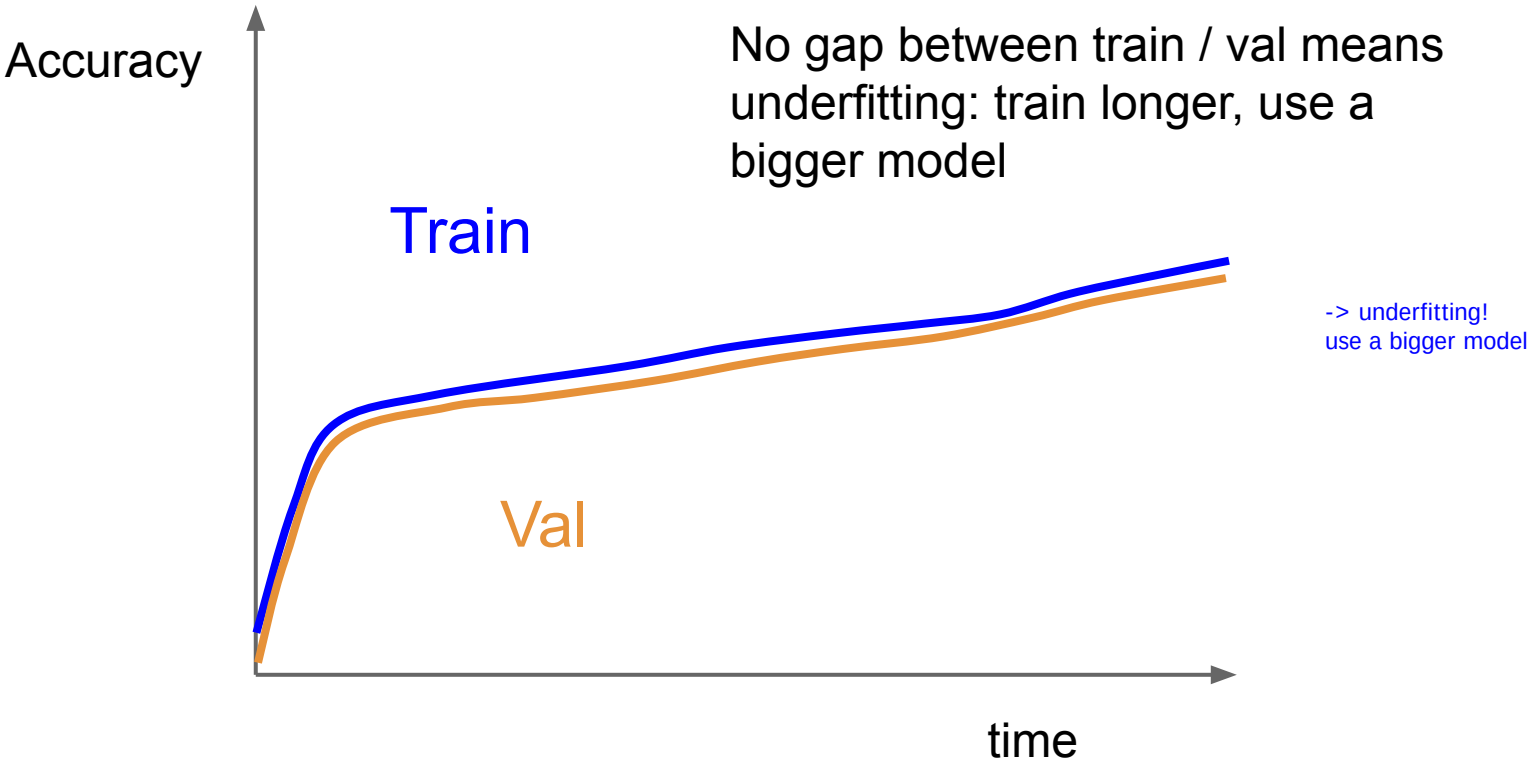


also an indicator, that the learning rate is to small



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

training dataset is to small
-> overfitting


Accuracy
No gap between train / val means underfitting: train longer, use a bigger model
Train
Val
-> underfitting!
use a bigger model
time

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

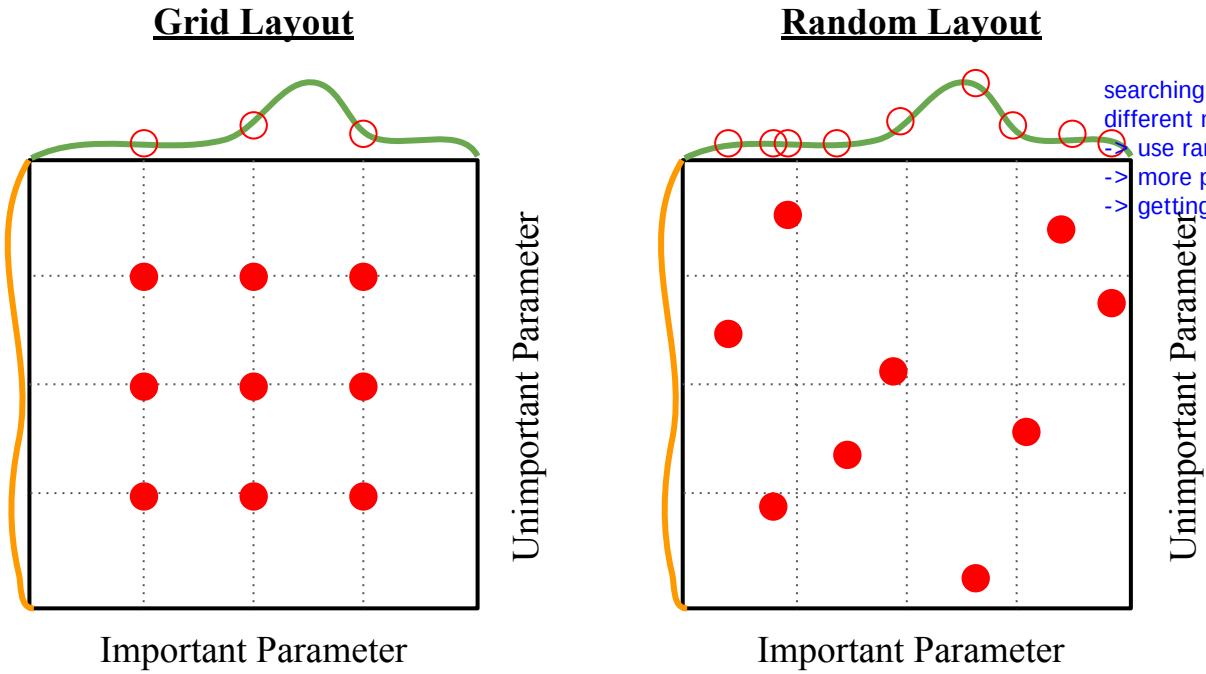
Hyperparameters to play with:

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

neural networks practitioner
music = loss function

Random Search vs. Grid Search

Random Search for Hyper-Parameter Optimization
Bergstra and Bengio, 2012



searching for hyperparameters and running different models at the same time
-> use random
-> more precised
-> getting different samples for free

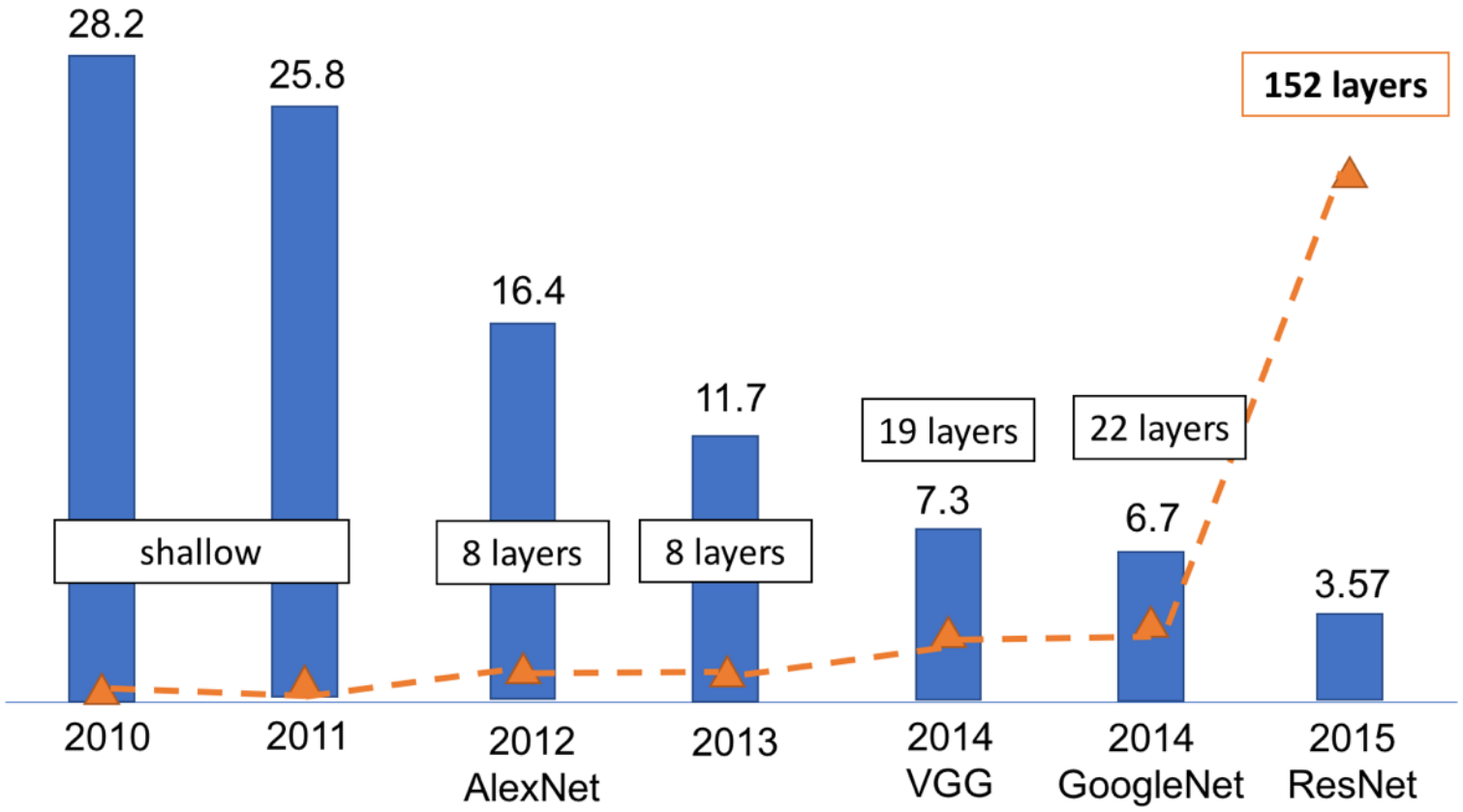
Illustration of Bergstra et al., 2012 by Shayne Longpre, copyright CS231n 2017

Summary

- Improve your training error:
  - Optimizers
  - Learning rate schedules
- Improve your test error:
  - Regularization
  - Choosing Hyperparameters

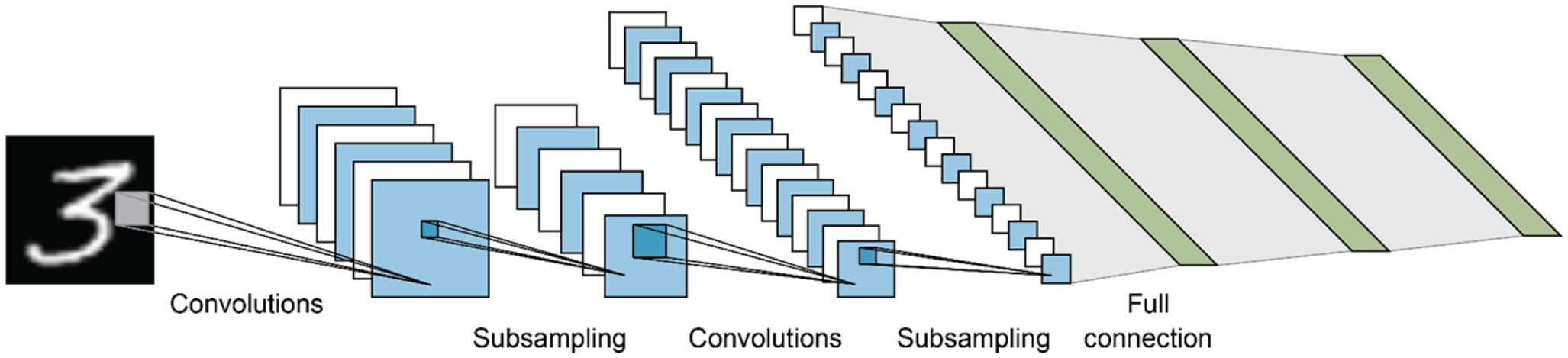
CNN architectures

ImageNet performance over time



Source: https://subscription.packtpub.com/book/big_data_and_business_intelligence/9781785880360/1/ch01lvl1sec12/why-deep-learning

Convolutional neural networks (CNN)

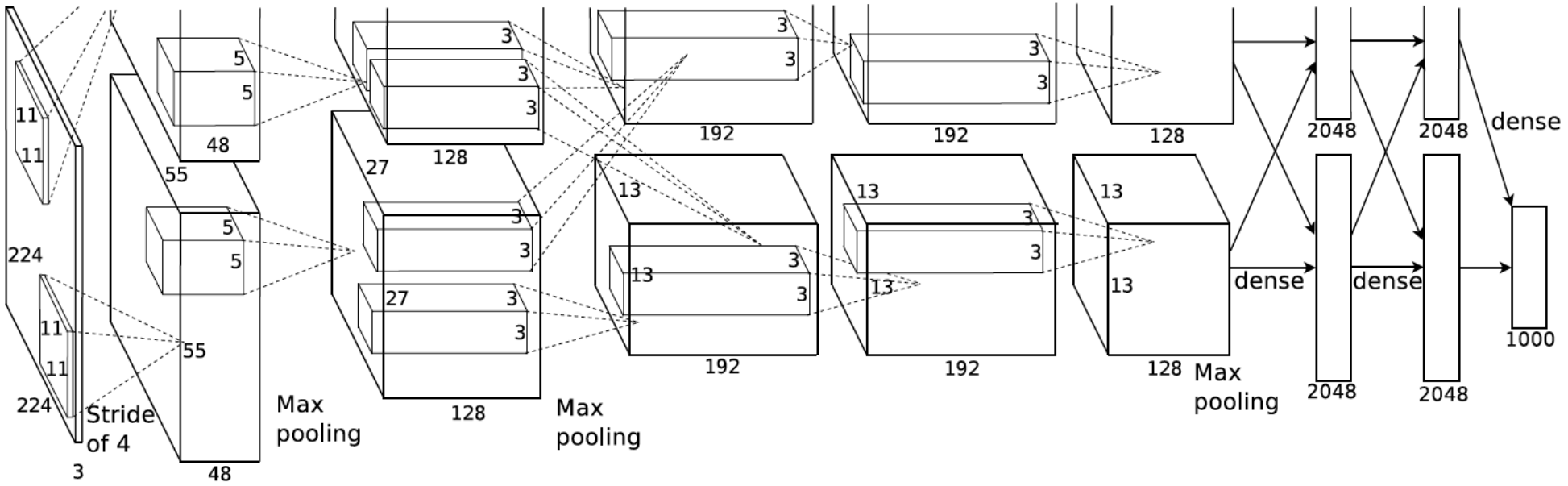


Denker et al. 1989, LeCun et al. 1989, 1998

Image source: https://livebook.manning.com/book/grokking-deep-learning-for-computer-vision/chapter-5/v-3/13

AlexNet

```
torchvision.models.alexnet()
```

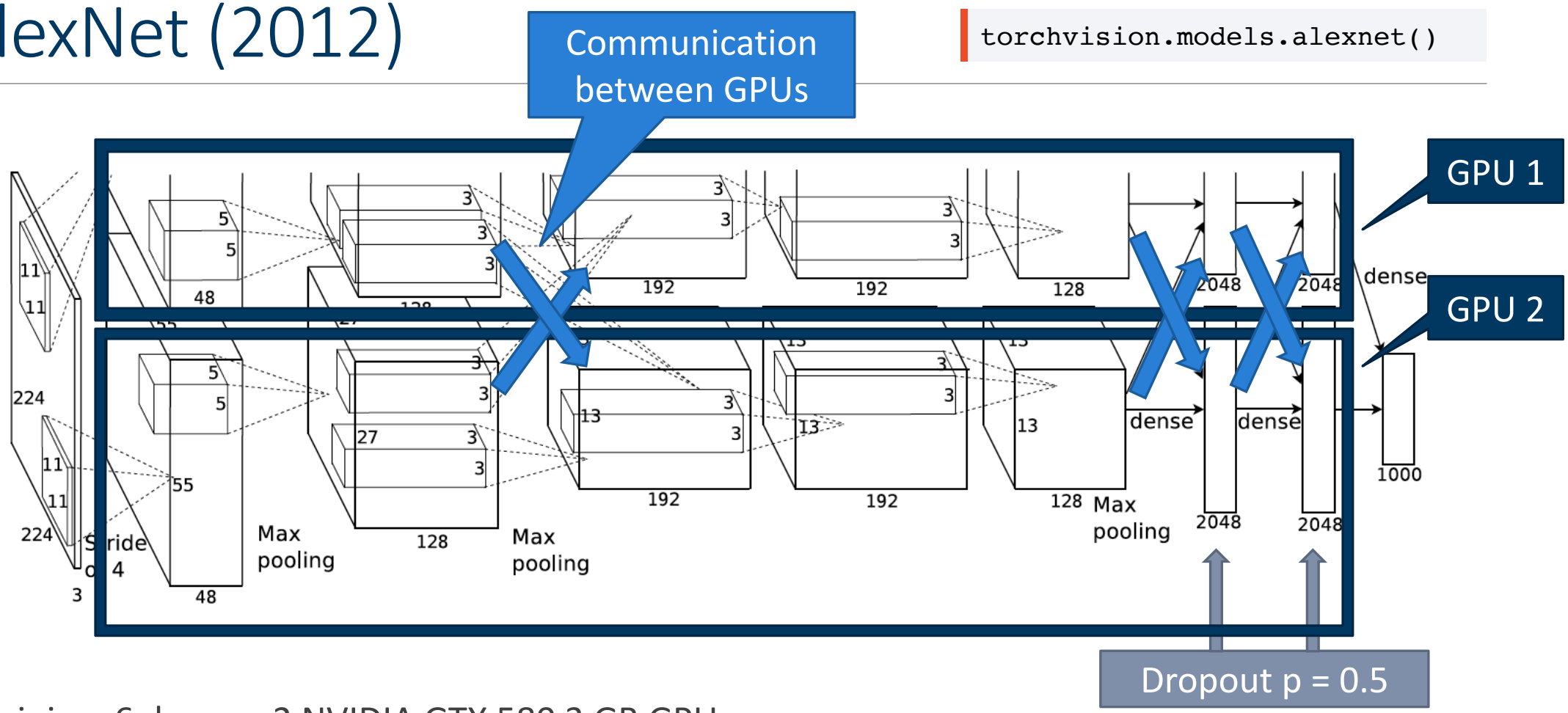


Training: 6 days on 2 NVIDIA GTX 580 3 GB GPUs

Krizhevsky, Sutskever, Hinton 2012

AlexNet (2012)

```
torchvision.models.alexnet()
```

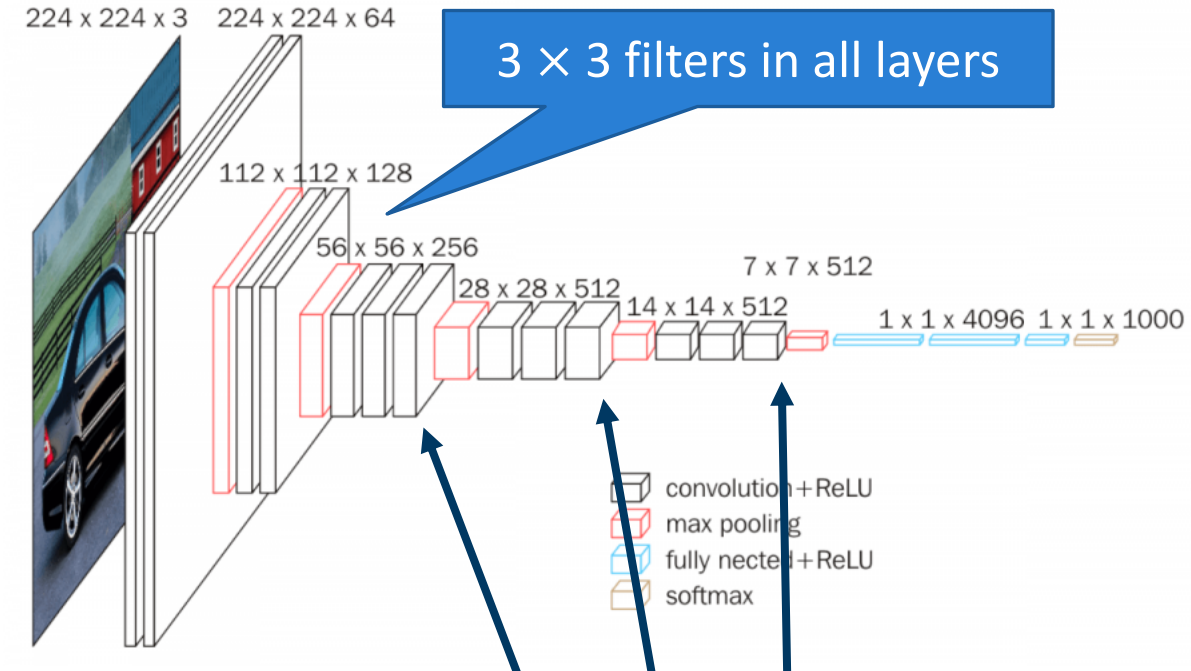


Training: 6 days on 2 NVIDIA GTX 580 3 GB GPUs
8 layers / 60 million parameters

Krizhevsky, Sutskever, Hinton 2012

VGG (2014)

```
torchvision.models.vgg16()
torchvision.models.vgg19()
torchvision.models.vgg19_bn()
…
```



VGG 16

VGG 19:
3 additional layers

Training:

3 weeks on 4 NVIDIA Titan Black 6 GB GPUs

First train smaller configurations, then inject layers in between:
11 → 13 → 16 → 19 layers

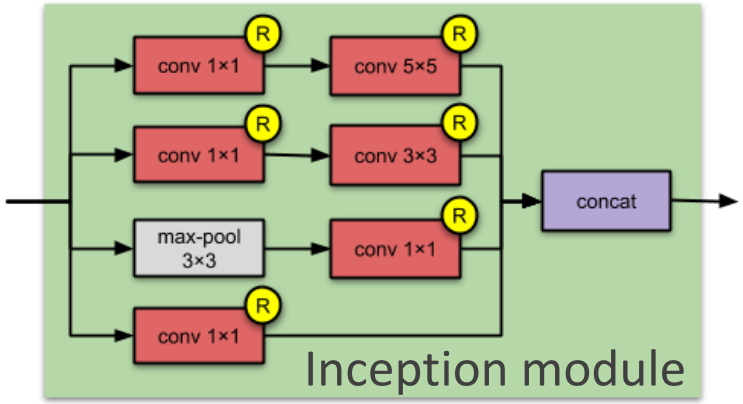
138 million parameters / 500 MB

takes probably a lot of time to train/ optimize etc. all of them

Simonyan & Zisserman, *ICLR* 2015

GoogLeNet/Inception-v1 (2014)

`torchvision.models.googlenet()`

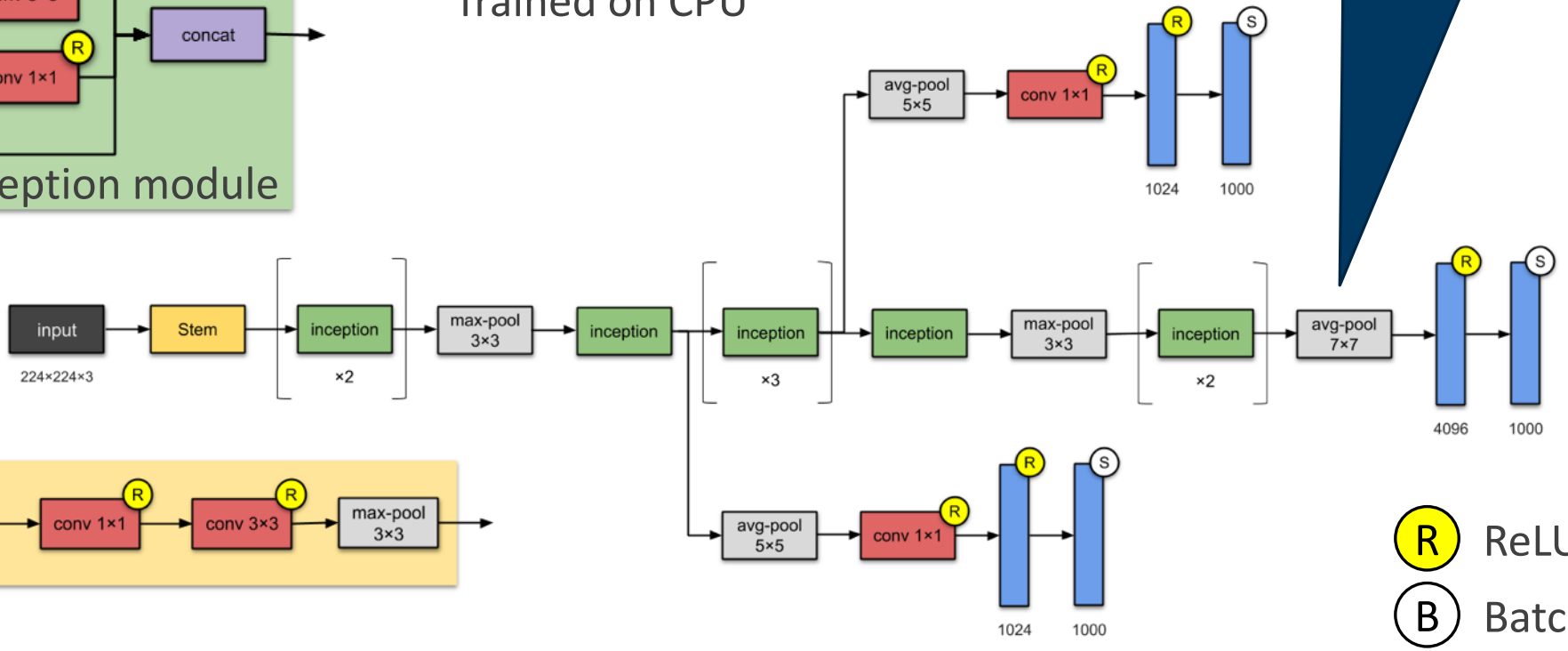


22 layers

5 million parameters

Trained on CPU

Replaced fully connected layer by global avg pool



R ReLU

B Batch norm

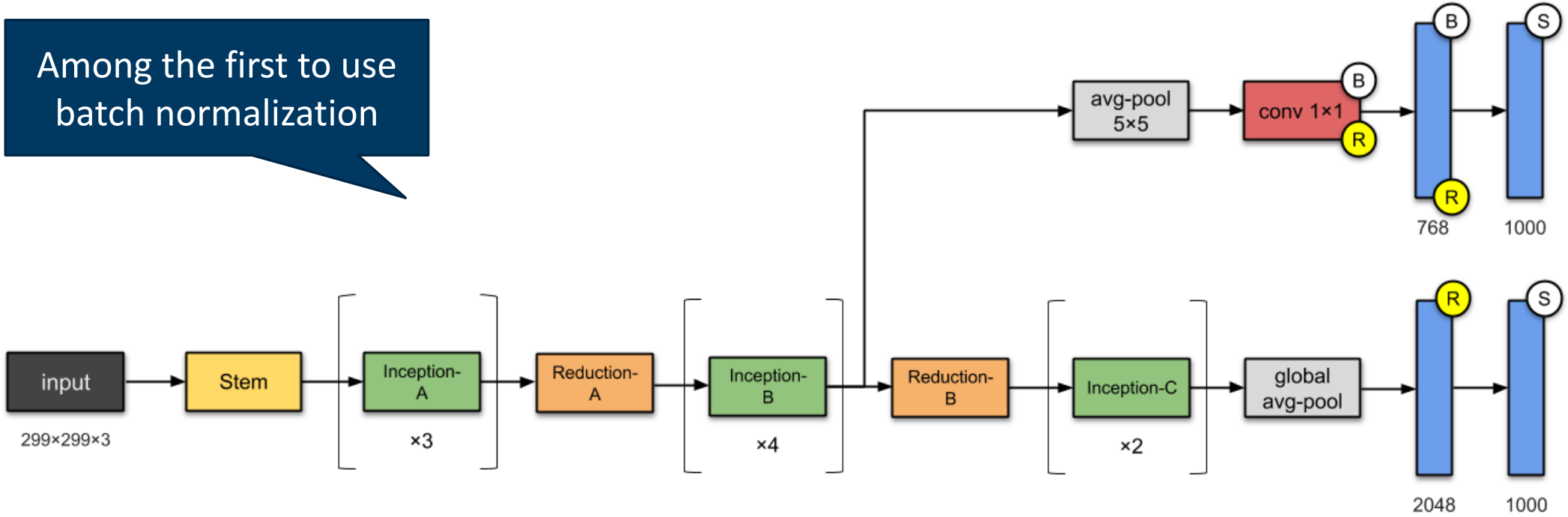
Source: https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d#e276

Szegedy et al., *CVPR* 2015

Inception-v3 (2015)

```
torchvision.models.inception_v3()
```

Among the first to use batch normalization



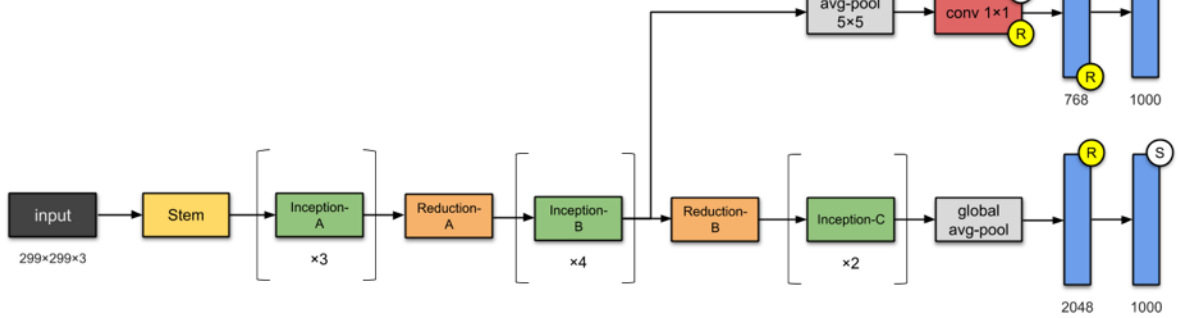
Trained on 50 NVIDIA K40 12 GB GPUs

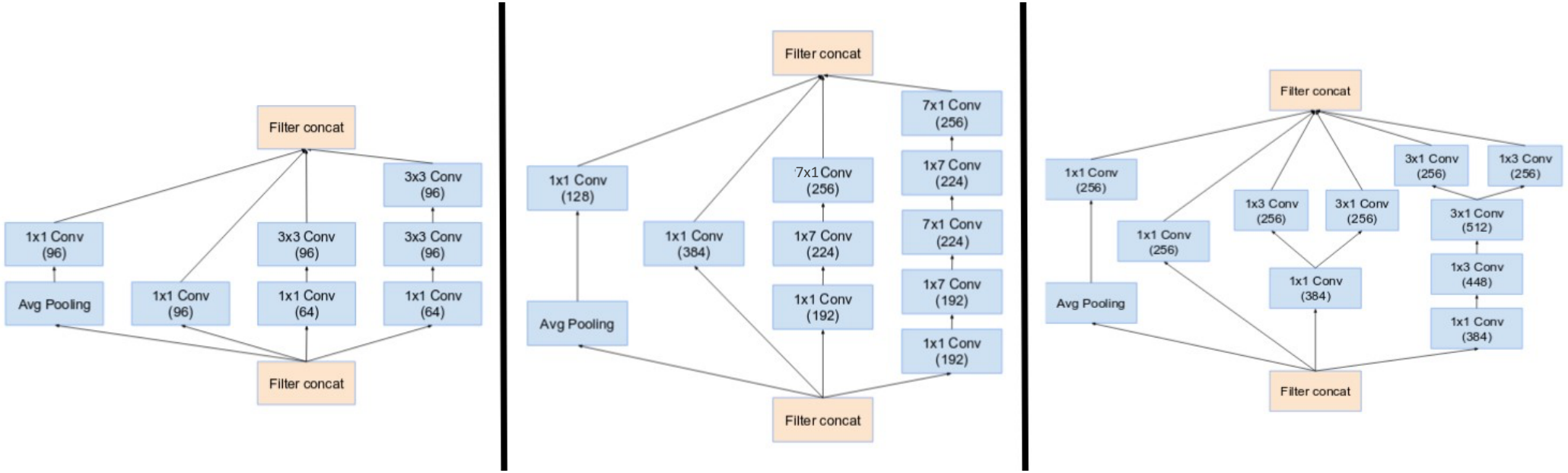
24 million parameters

Source: https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d#e276

Szegedy et al., *CVPR* 2016

Inception-v3 (2015)

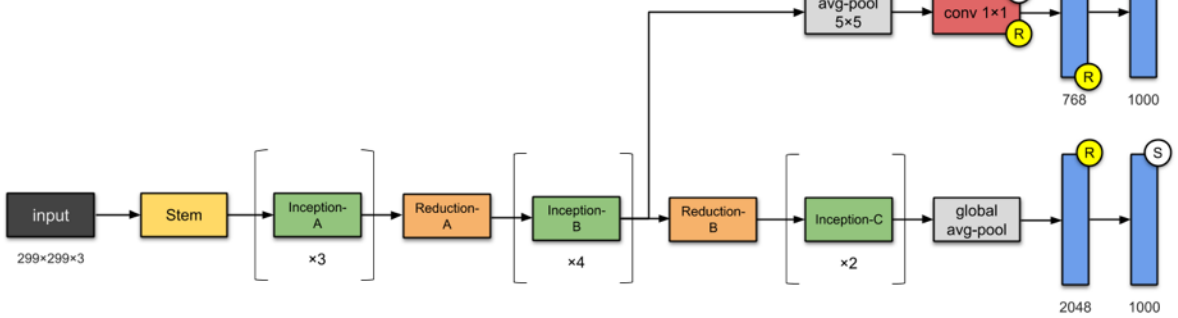


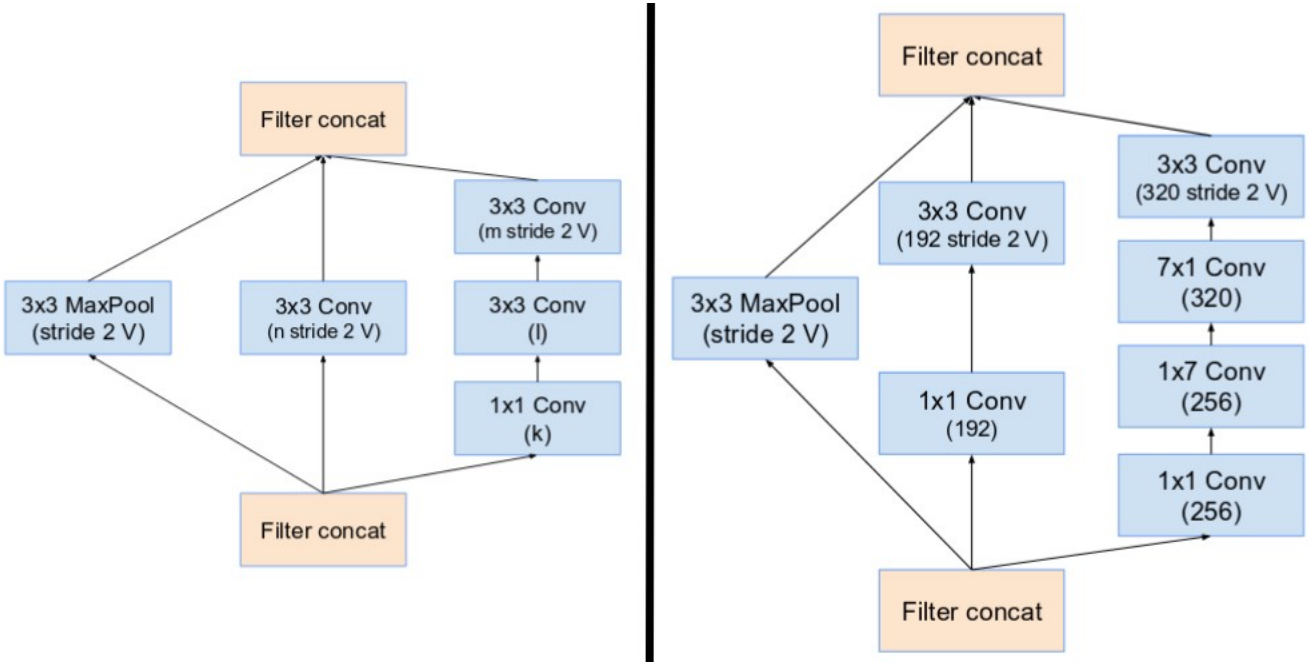


Inception A | Inception B | Inception C

Szegedy et al., *CVPR* 2016

Inception-v3 (2015)





Reduction A

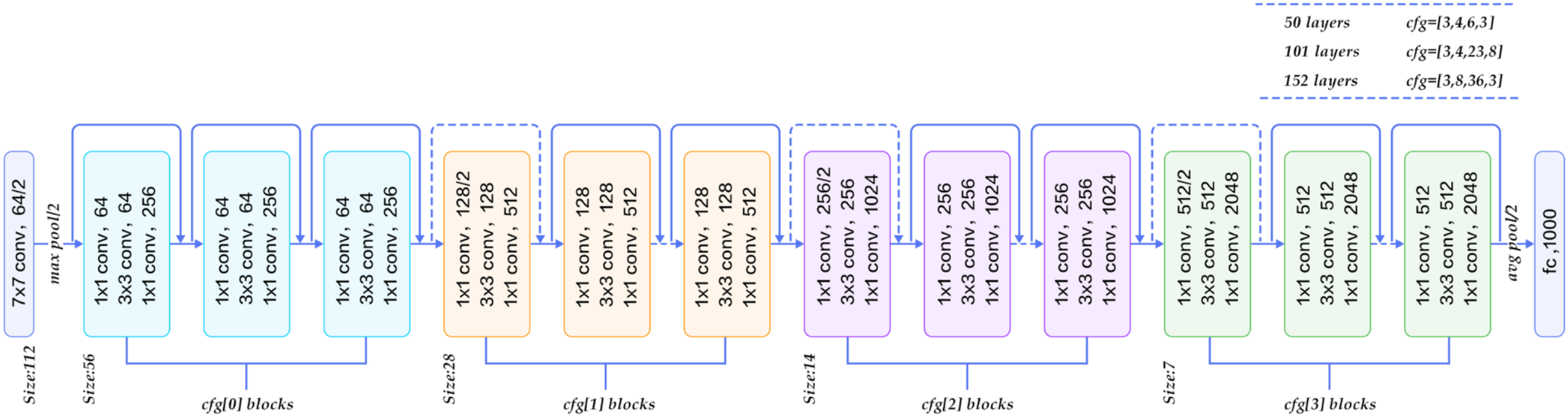
Reduction B

Szegedy et al., *CVPR* 2016


WE NEED TO GO
DEEPER
troll-face.ru

ResNet (2015)

```
torchvision.models.resnet18()
…
torchvision.models.resnet152()
```

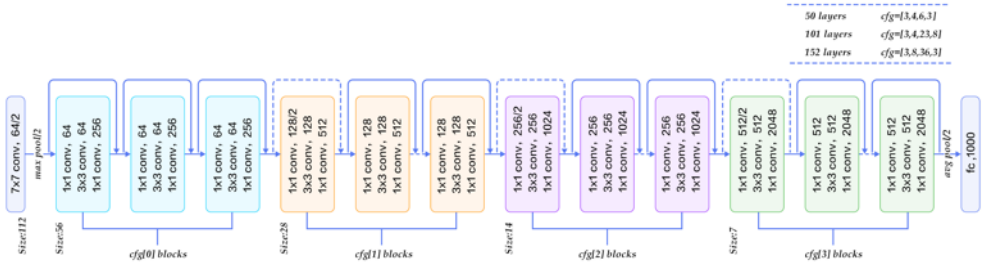


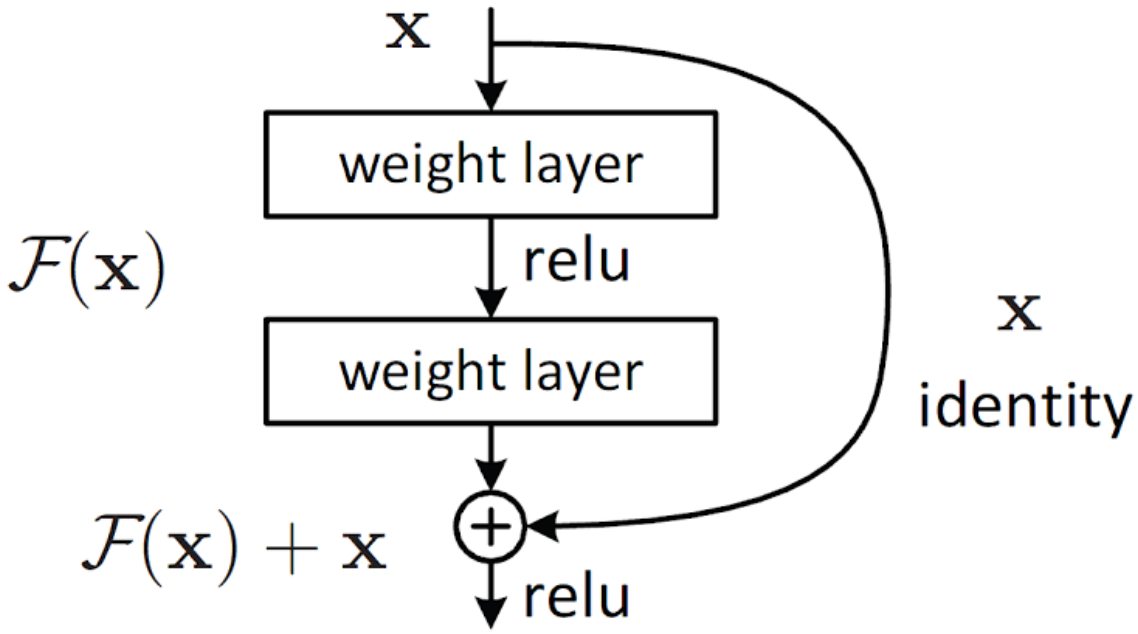
ResNet-50 (-18, -34, -101, -152 also exist)

26 million parameters

He et al., *CVPR* 2016

Residual blocks (“skip connections”)



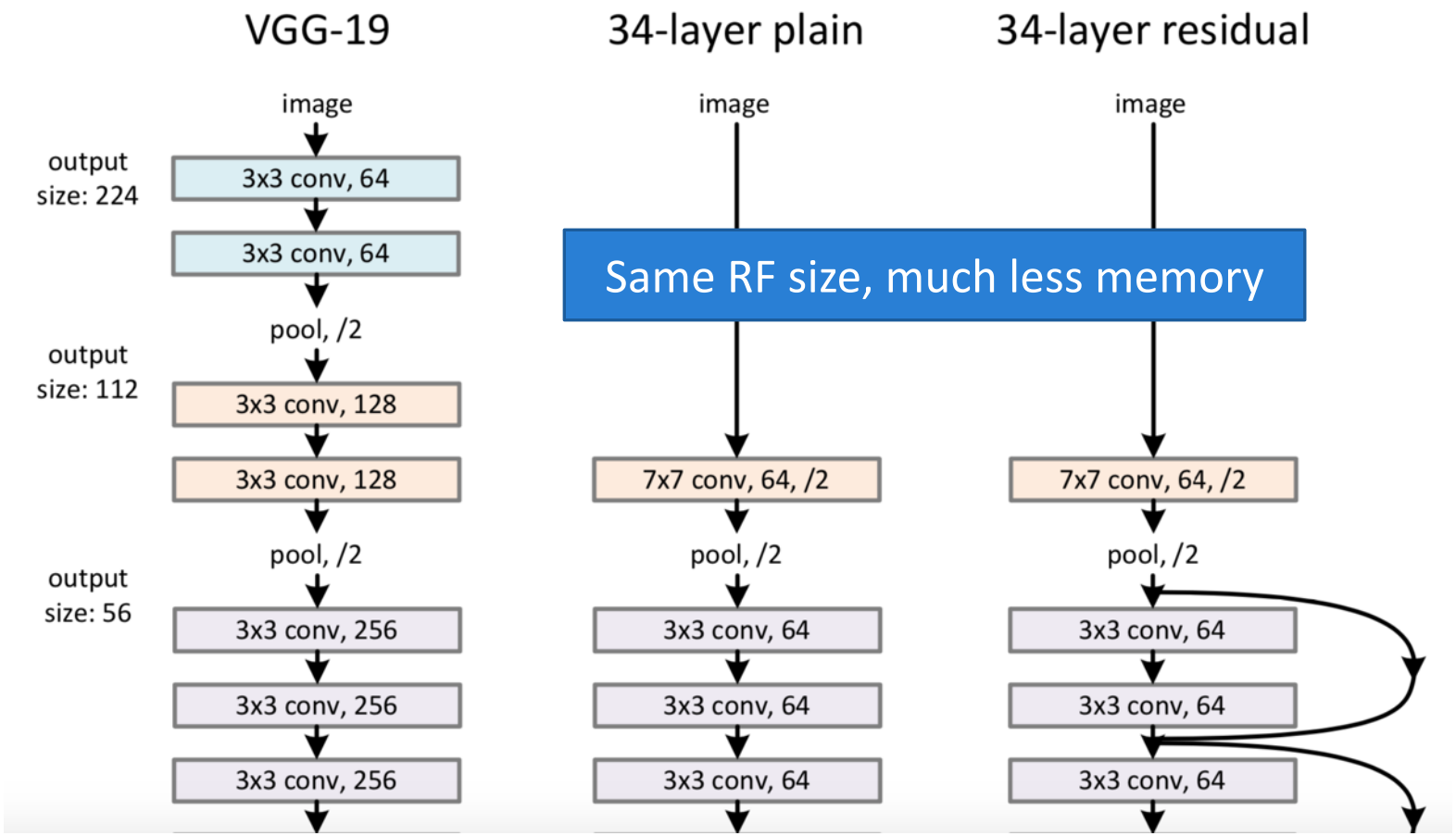


Better gradient flow because of skip connections

→ At initialization, the network computes approximately the identity function

He et al., *CVPR* 2016

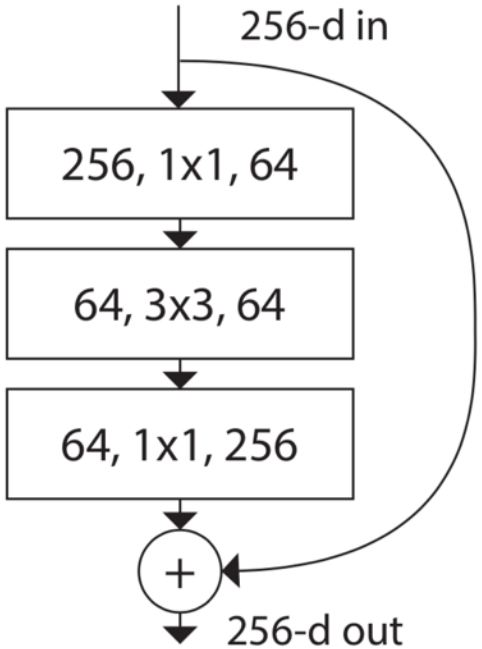
ResNet



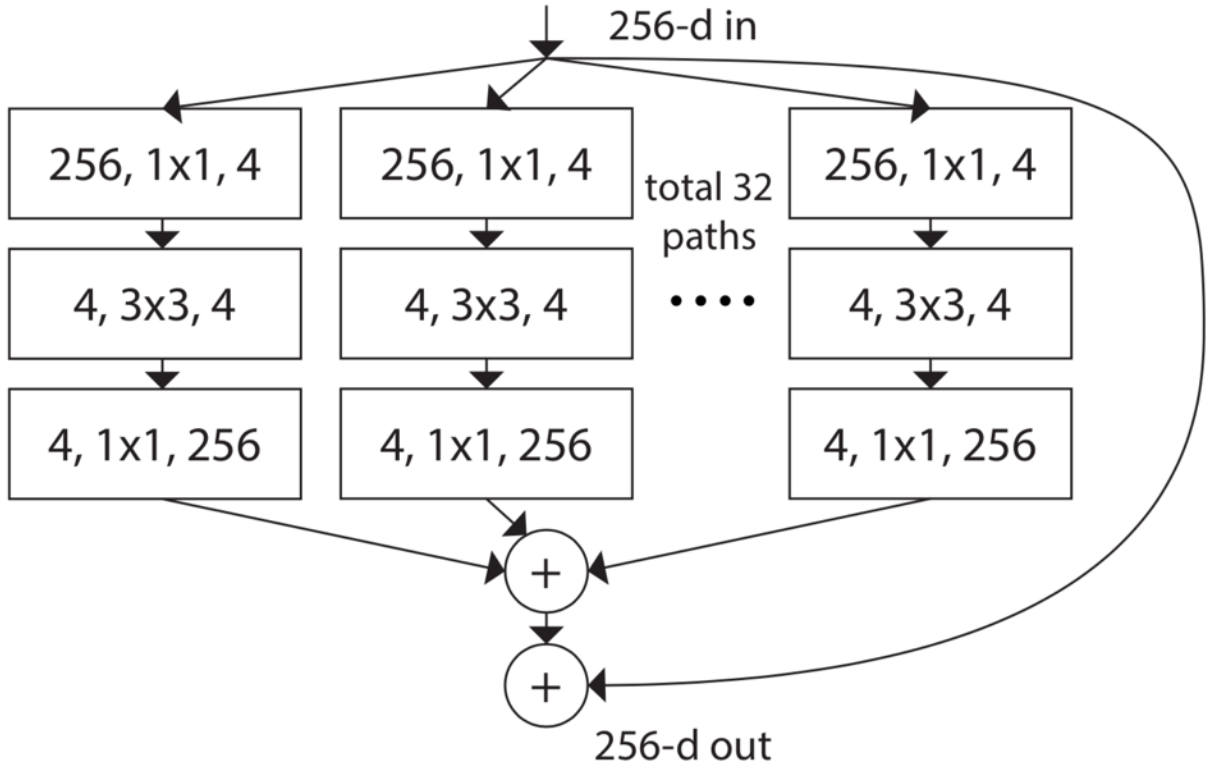
He et al., *CVPR* 2016

ResNeXt (2017)

```
torchvision.models.resnext50_32x4d()
```



ResNet

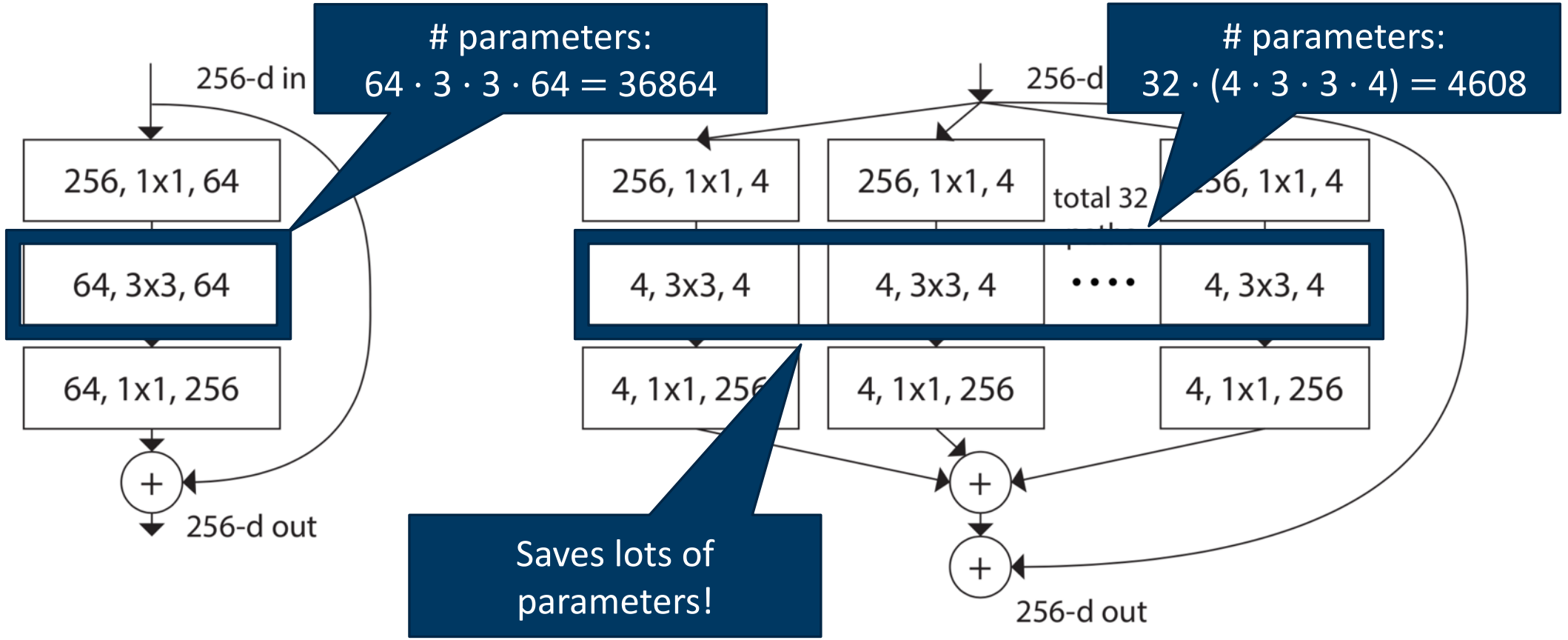


ResNeXt

Xie et al., *CVPR* 2016

ResNeXt (2017)

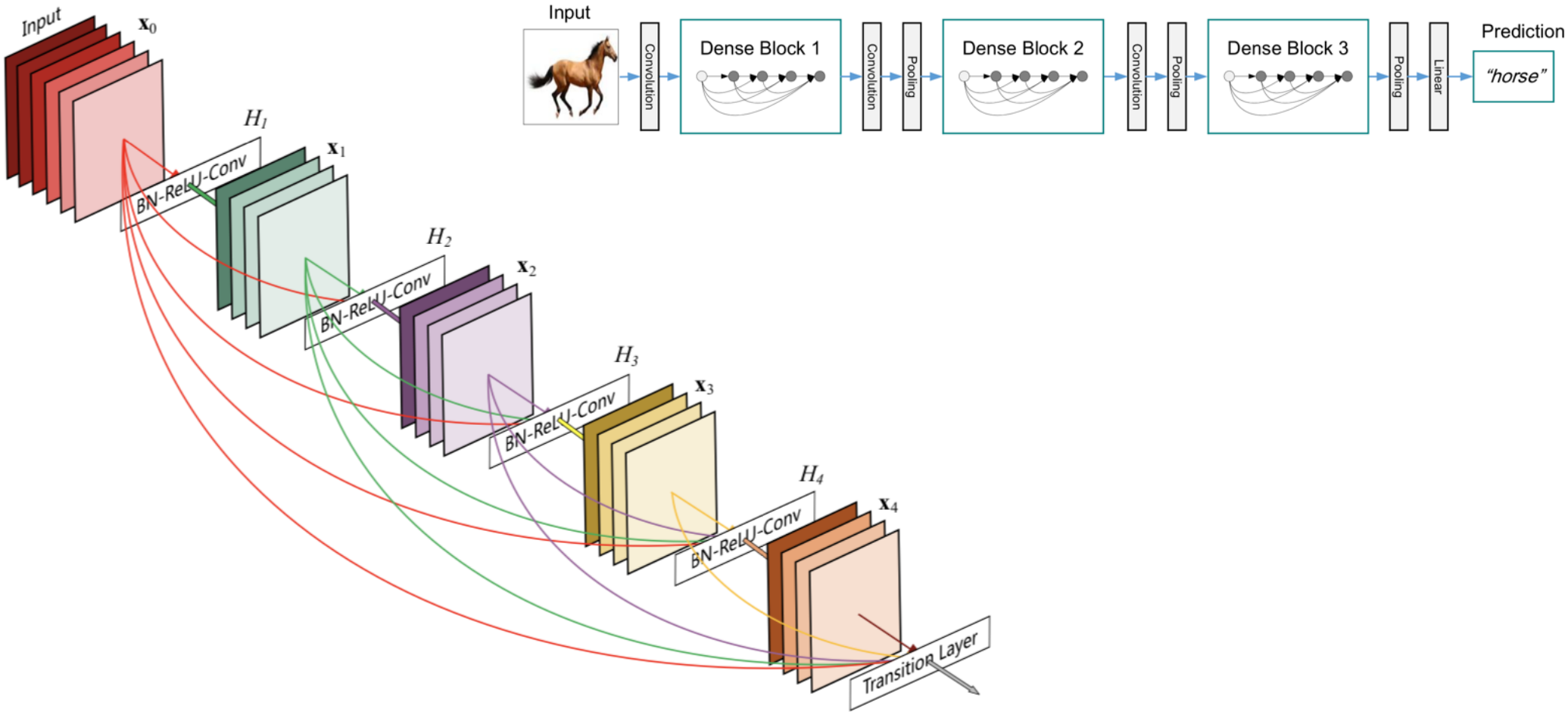
```
torchvision.models.resnext50_32x4d()
```



Xie et al., *CVPR* 2016

DenseNet

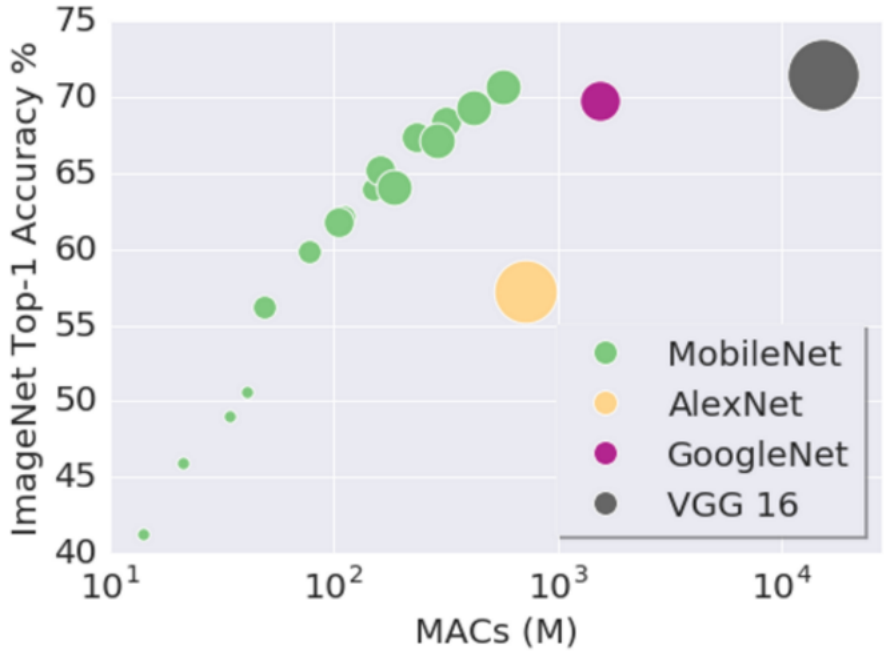
```
torchvision.models.densenet161()
```

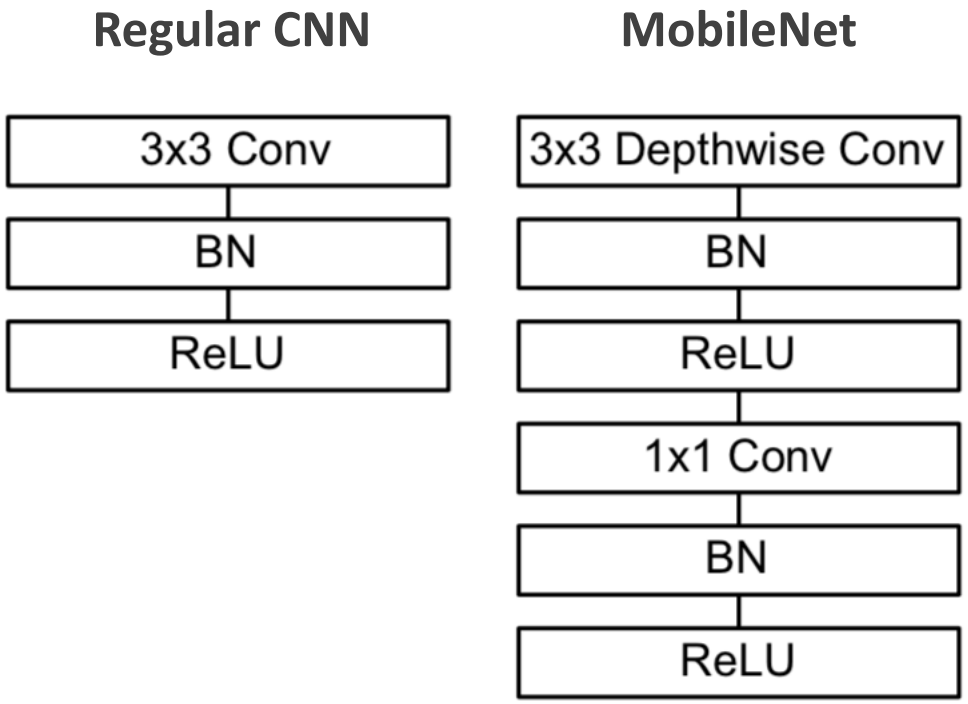


Huang et al., *CVPR* 2017

MobileNet (2017)

```
torchvision.models.mobilenet_v2()
```





Runs at > 100 frames/sec on an iPhone 7
at ~70% top-1 / 90% top-5 accuracy on ImageNet

Howard et al., *CVPR* 2018

dazu noch weitere Informationen suchen!

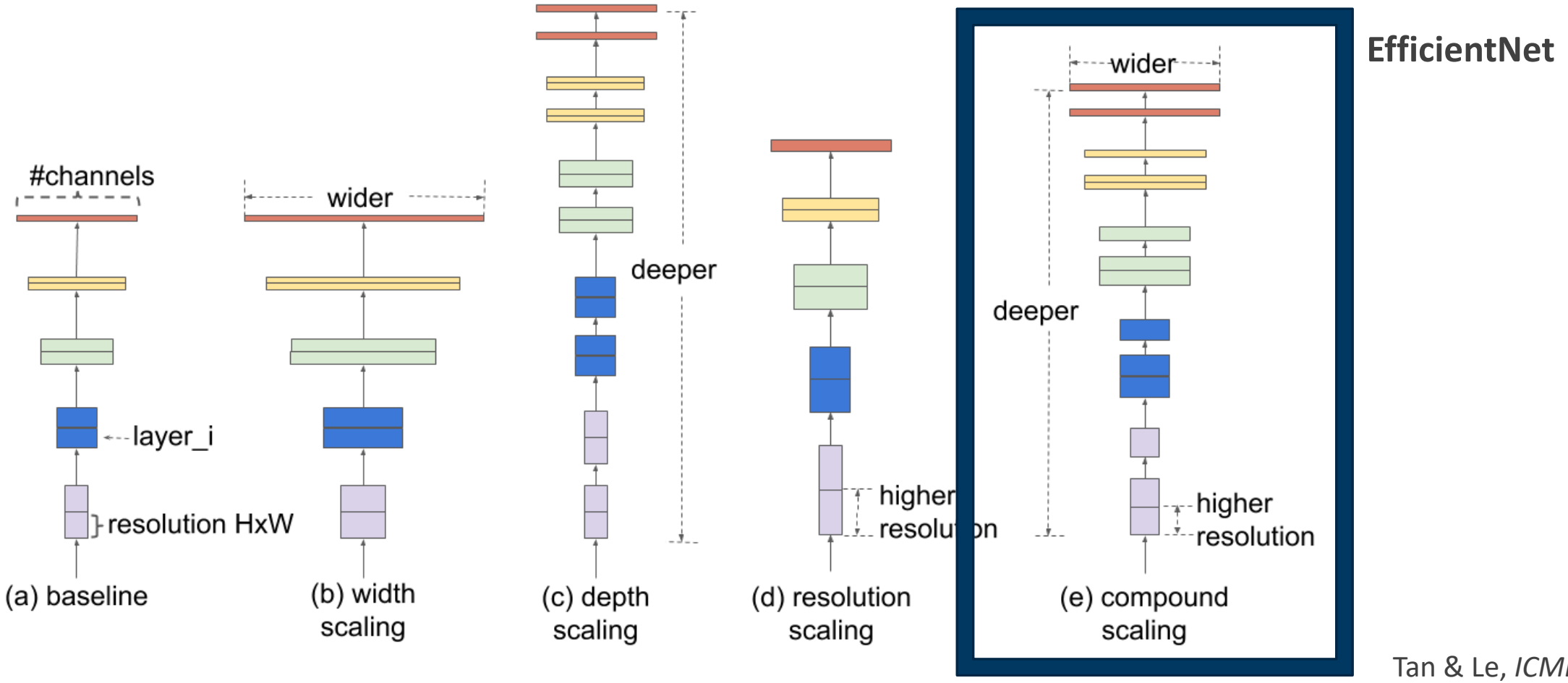
Depthwise separable convolution

Input features	Filters	Feature maps	Filters	Output features
$32 \times 32 \times 4$	$4 \times 3 \times 3 \times 1$	$32 \times 32 \times 4$	$4 \times 1 \times 1 \times 2$	$32 \times 32 \times 2$

Depthwise separable: $4 \cdot 3 \cdot 3 + 4 \cdot 2 = 44$ parameters $O(MK^2 + MN)$ parameters

Regular convolution: $4 \cdot 3 \cdot 3 \cdot 2 = 72$ parameters $O(MK^2N)$ parameters

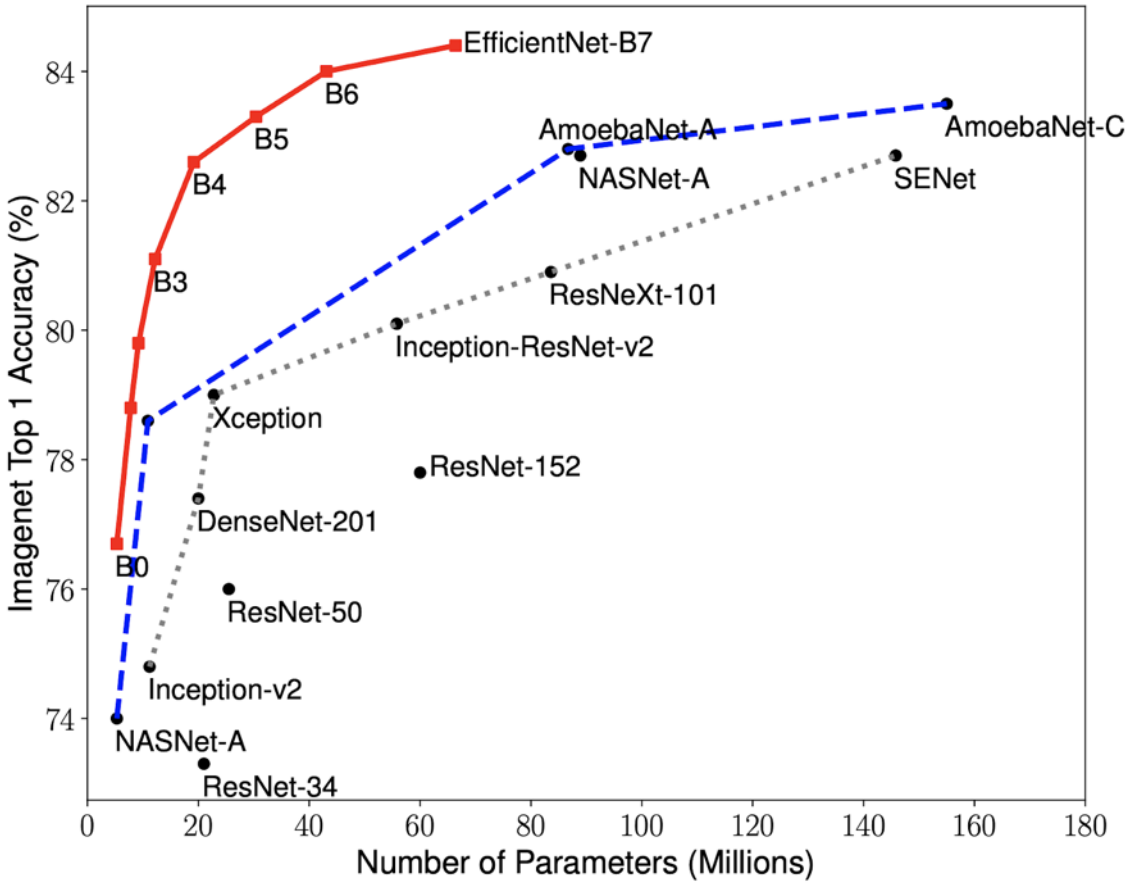
EfficientNet (2019)



Tan & Le, *ICML* 2019

https://ai.googleblog.com/2019/05/efficientnet-improving-accuracy-and.html

EfficientNet (2019)



Tan & Le, *ICML* 2019

https://ai.googleblog.com/2019/05/efficientnet-improving-accuracy-and.html

Questions!