Convolutional Neural Network

MLRS 2019

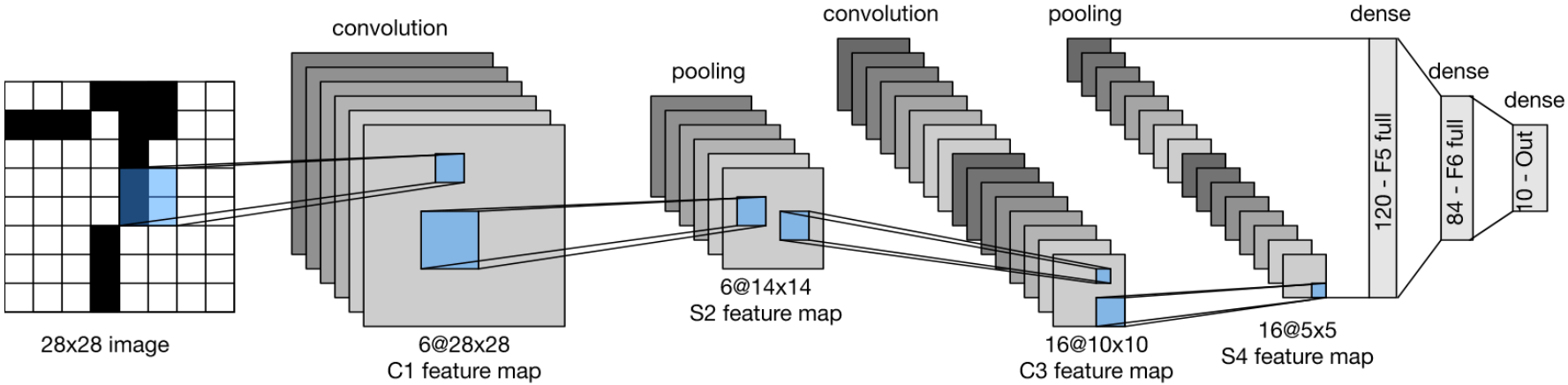
Rachel Hu

Amazon AI

Outline

- Convolutions
- Pooling, Padding, Stride and Multi-layer
- **Convolutional Neural Networks (LeNet)**
- **Deep ConvNets (AlexNet)**
- **Networks using Blocks (VGG)**
- **Networks in Networks (NiN)**
- **Residual Neural Networks (ResNet)**
- **Other Ideas**
- **Fine Tuning**

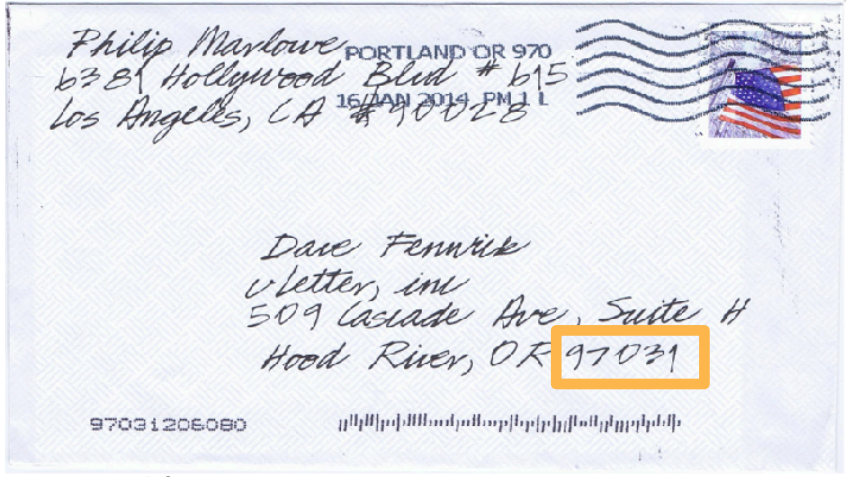
LeNet

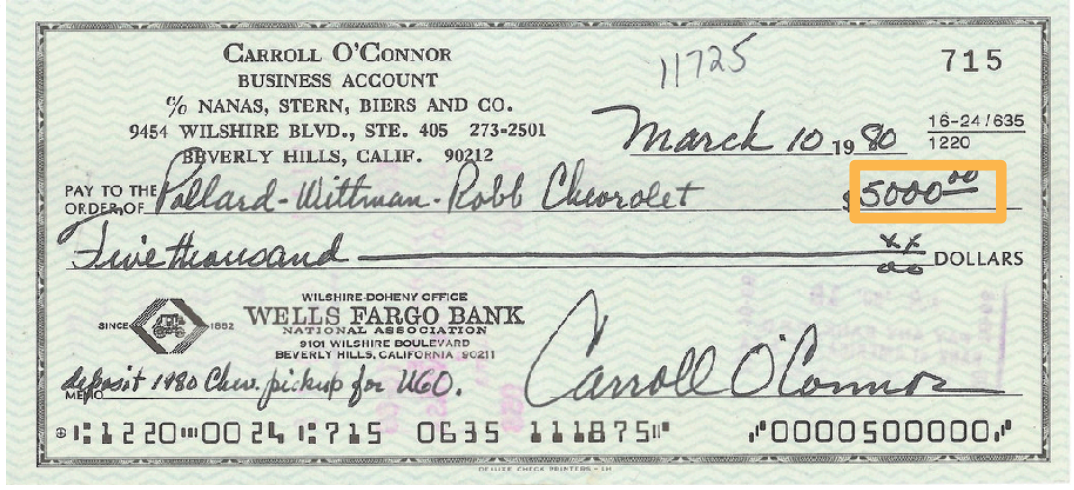


http://yann.lecun.com/exdb/lenet/

Why Handwritten Digit Recognition in 1990s?

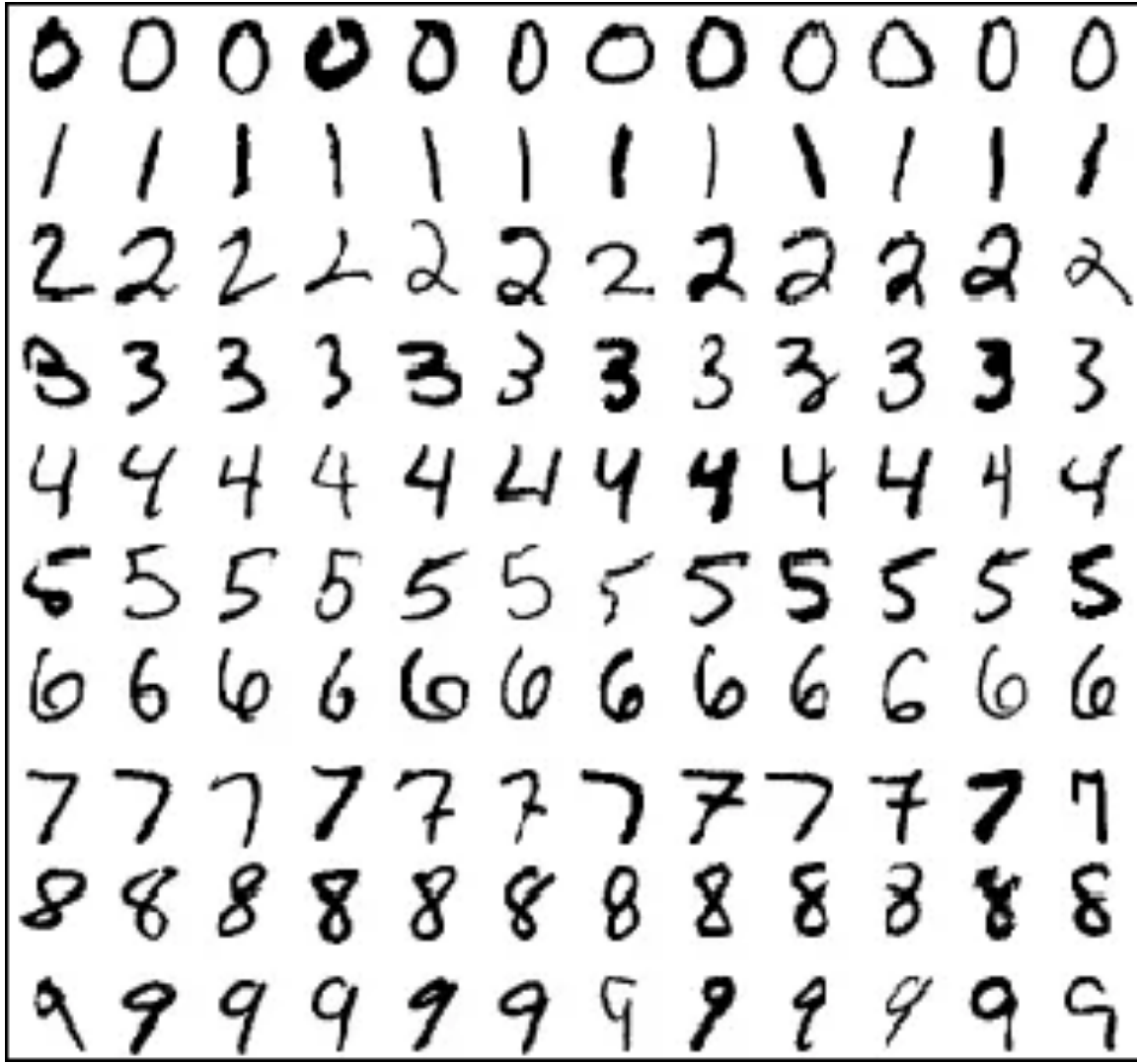
AT&T had a project in order to recognize handwritten characters for postal codes on letters and also recognize the dollar amounts on checks.





MNIST

- 50,000 training data
- 10,000 test data
- 10 classes
- Grayscale
- 28 x 28 pixels
- Centered and scaled



Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, 1998 Gradient-based learning applied to document recognition

aws

Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, 1998 Gradient-based learning applied to document recognition

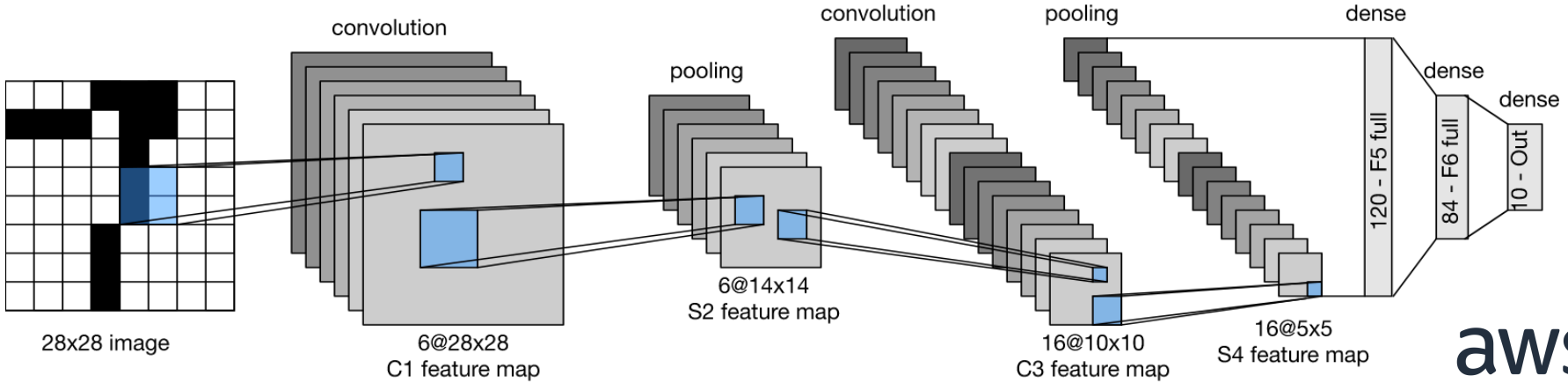
aws

LeNet

LeNet consists of two parts:

Part I. Convolution Block

- Convolution layer - To recognize the spatial patterns
  - 5×5 kernel
  - sigmoid activation function
- Average pooling layer - To reduce the dimensionality

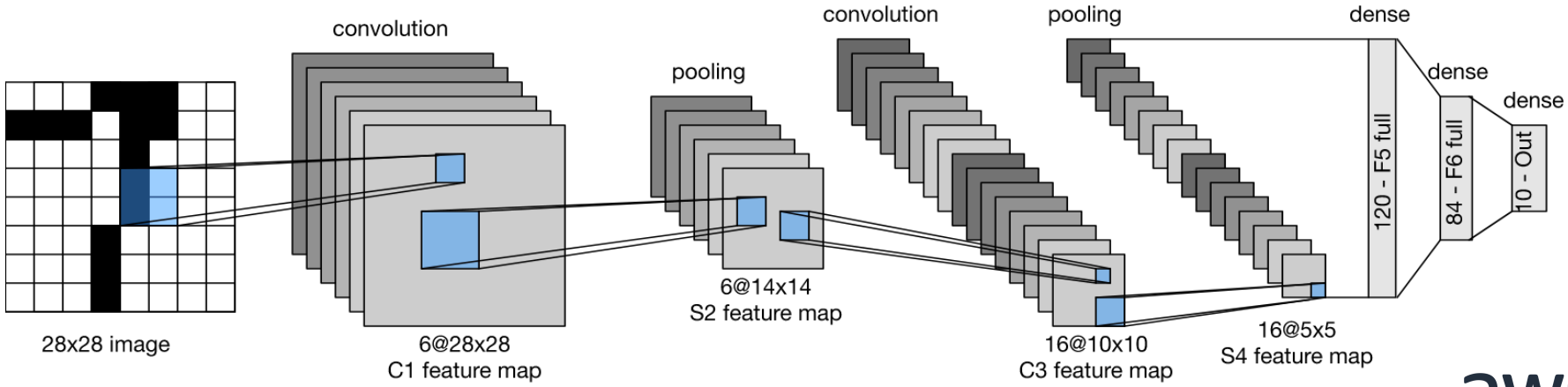


LeNet

LeNet consists of two parts:

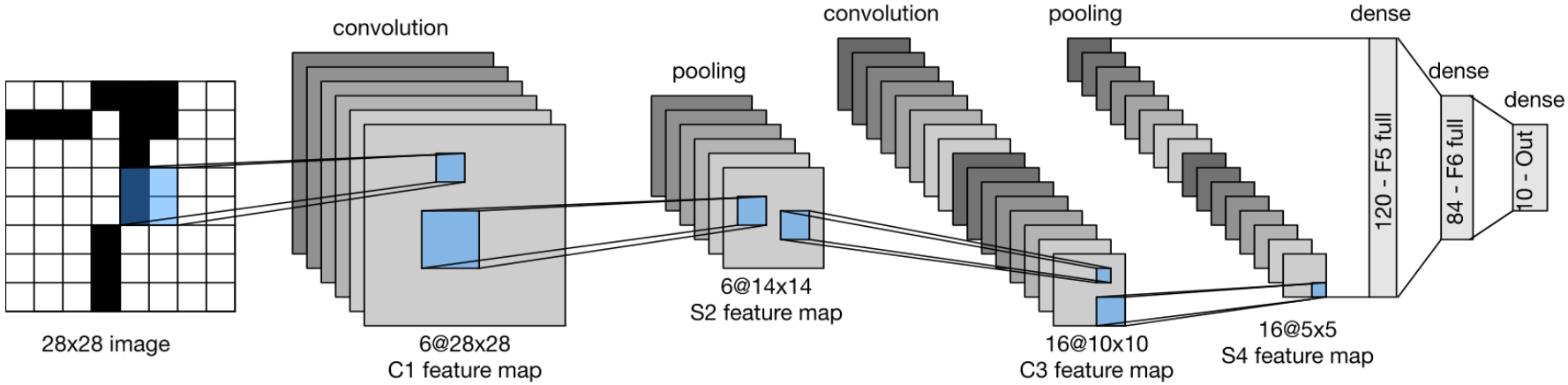
Part II. Fully-connected layers Block

- 3 fully-connected layers
  - with 120, 84, and 10 outputs, respectively



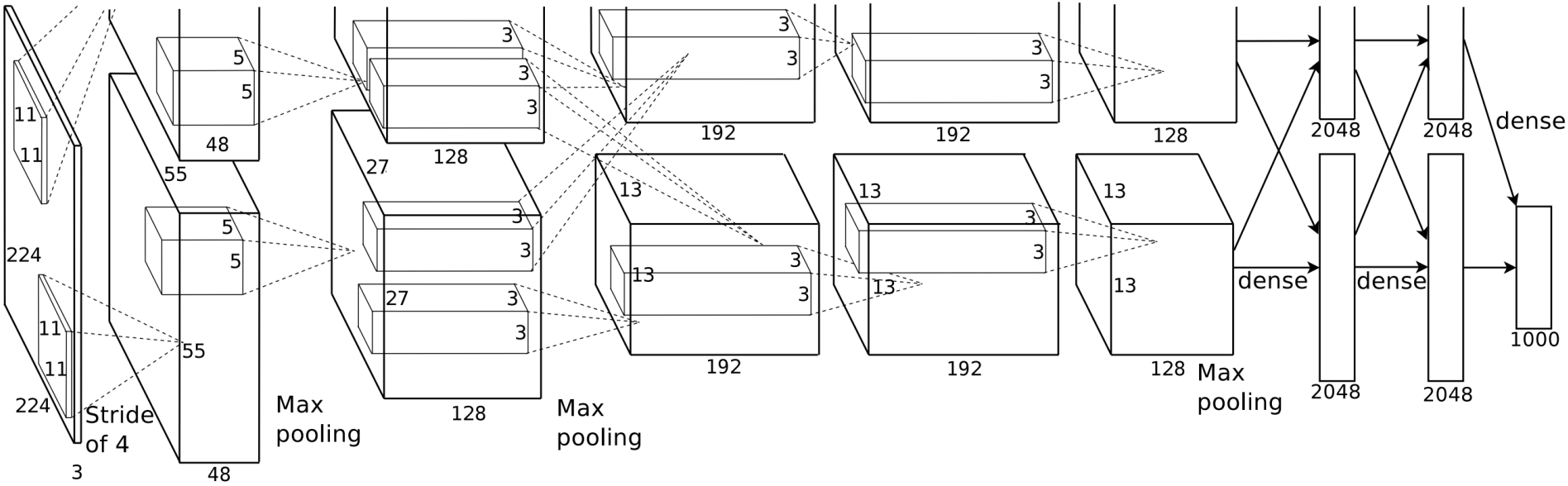
LeNet

Expensive if we have many outputs

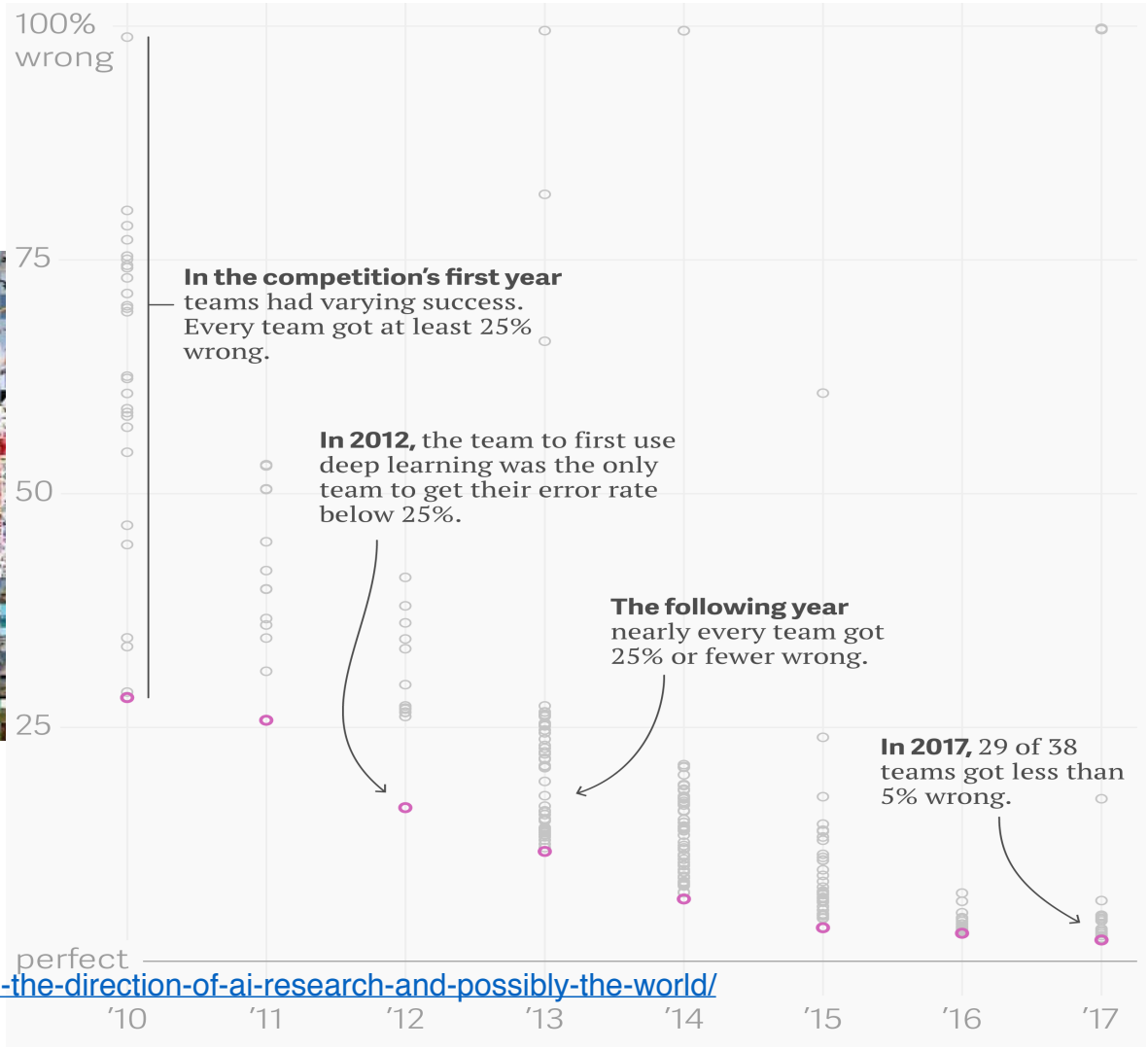


LeNet Notebook

AlexNet

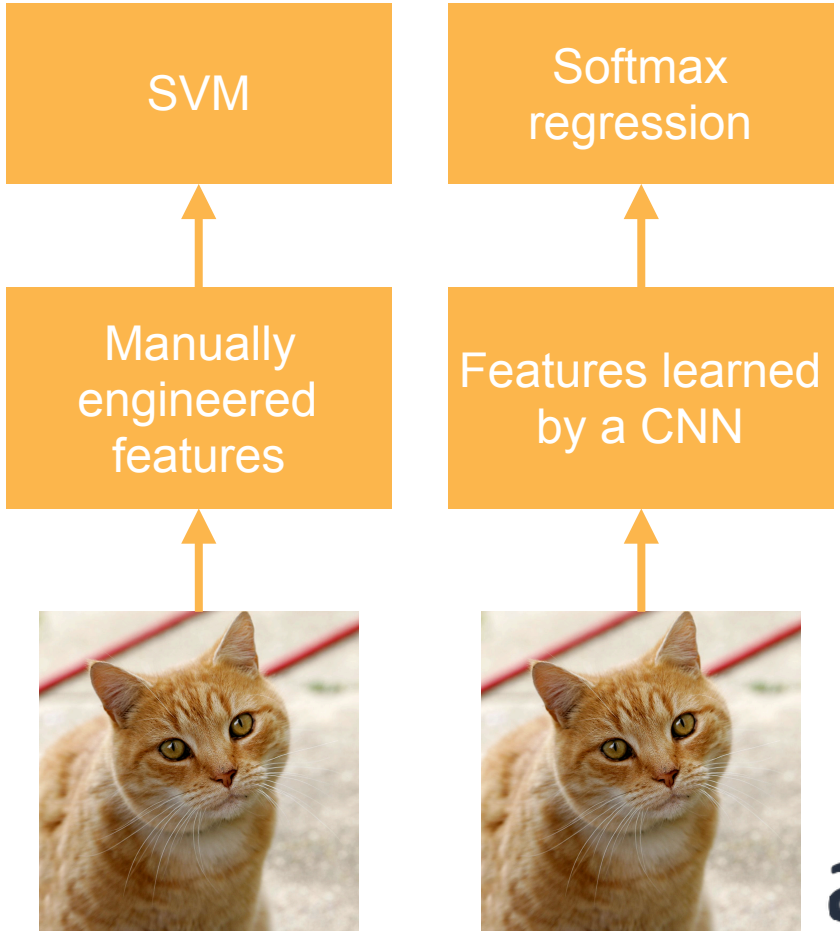

224
11
11
224
3
Stride
of 4
55
48
5
5
48
55
5
5
Max
pooling
27
128
3
3
3
128
27
3
3
3
Max
pooling
13
192
3
3
13
3
3
13
192
3
3
13
192
3
3
13
128
Max
pooling
dense
dense
2048
2048
2048
2048
dense
1000

AlexNet



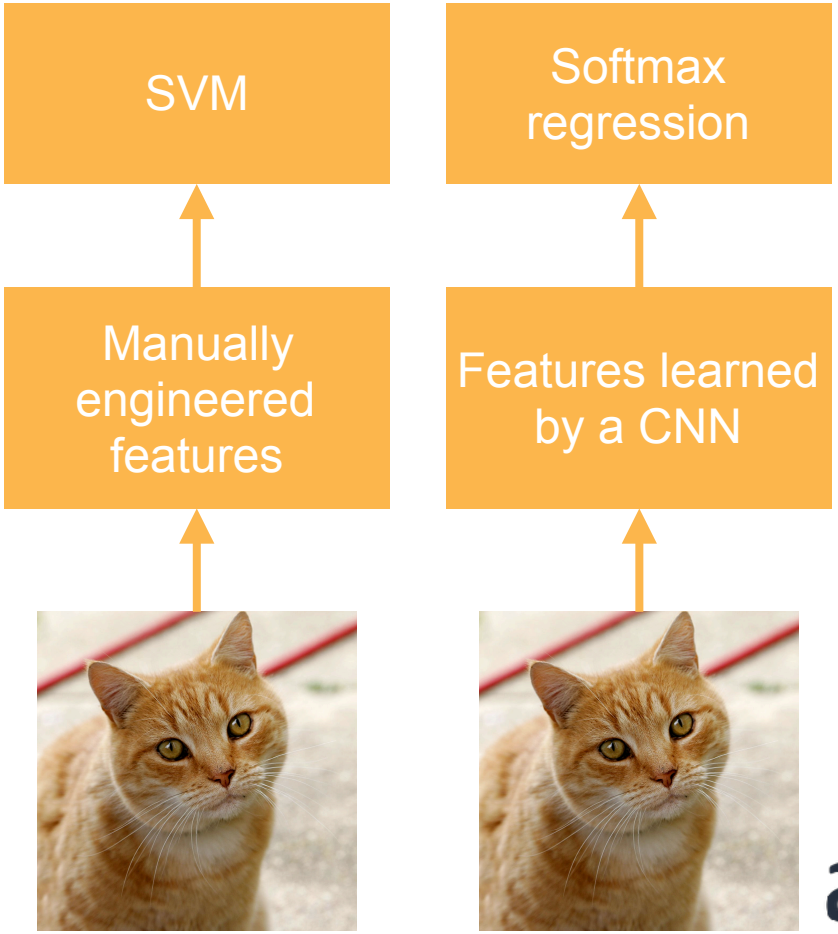
Yanofsky, Quartz
https://qz.com/1034972/the-data-that-changed-the-direction-of-ai-research-and-possibly-the-world/

AlexNet

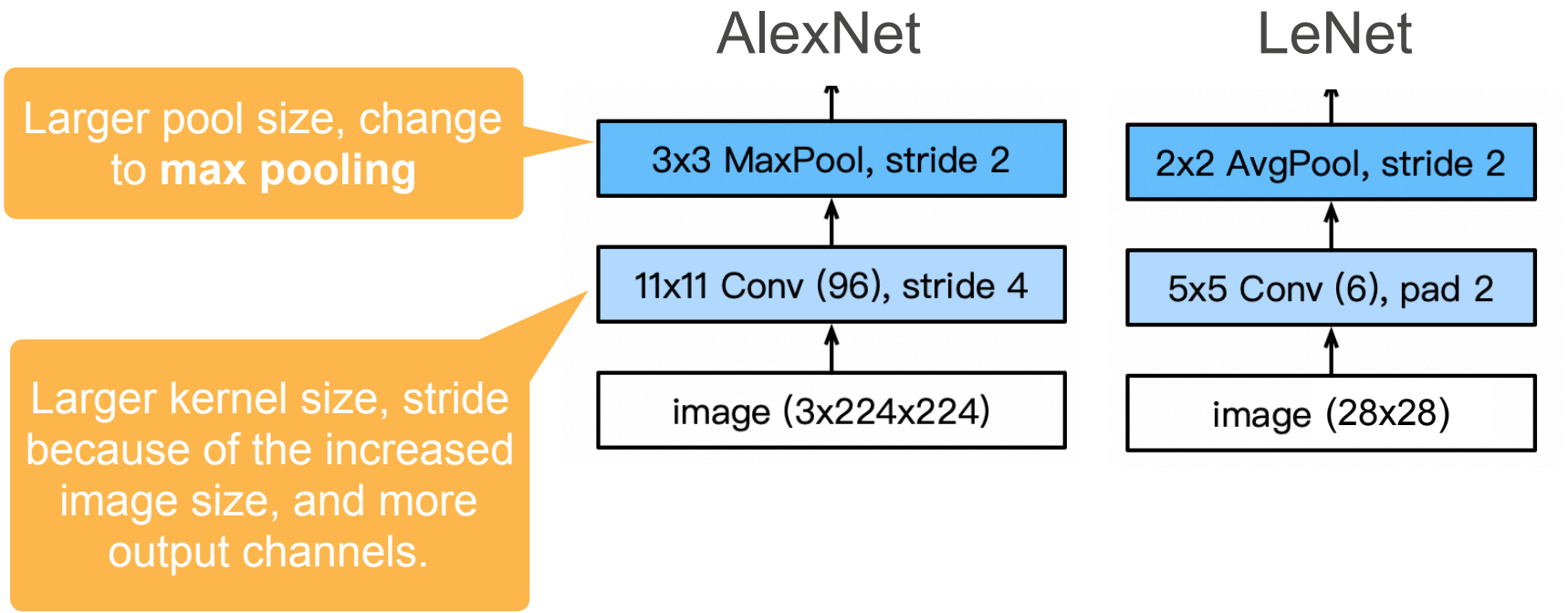

SVM
Softmax regression
Manually engineered features
Features learned by a CNN

AlexNet

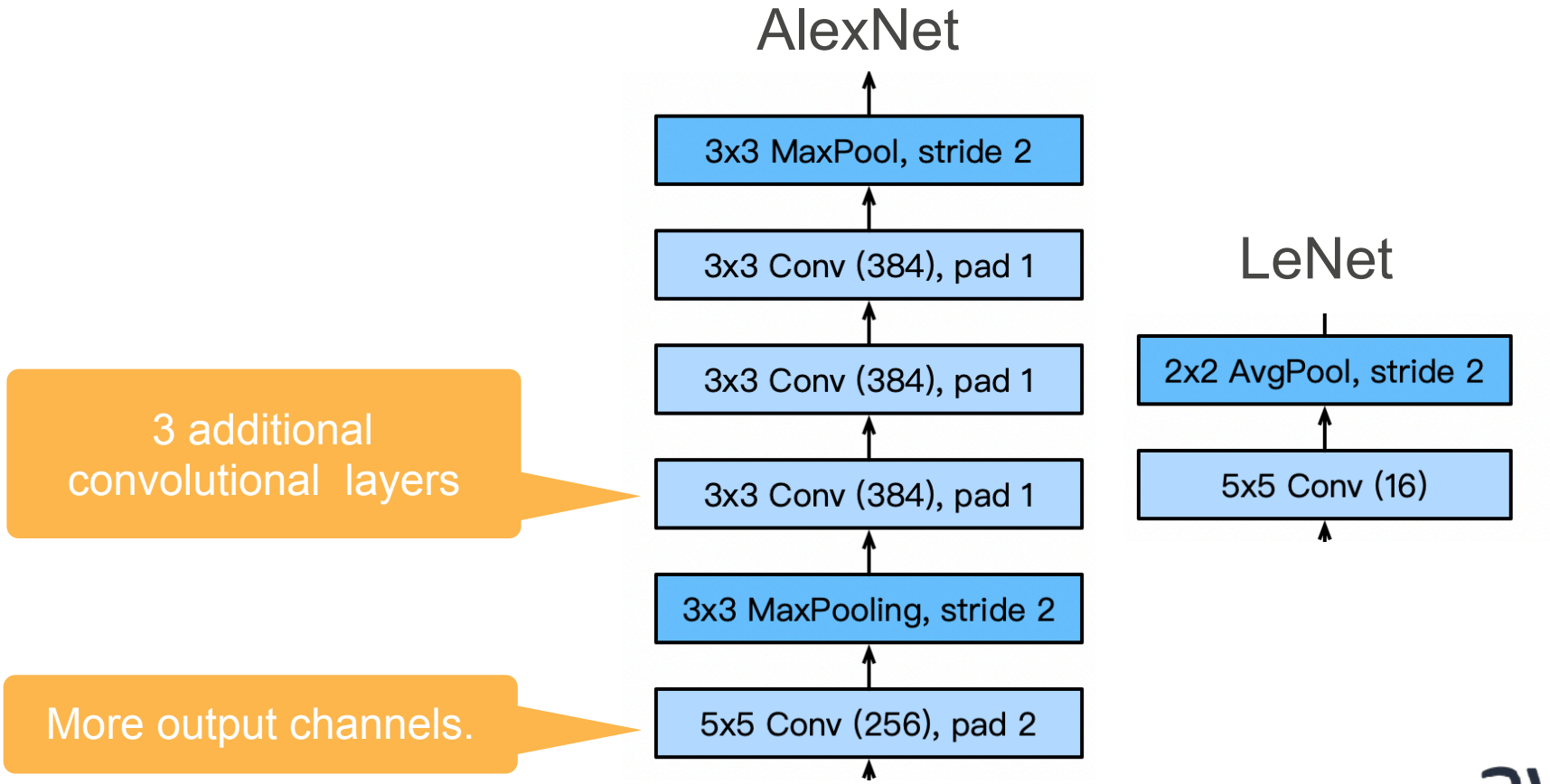
- Deeper and bigger LeNet
- Key modifications
  - Dropout (regularization)
  - ReLu (training)
  - MaxPooling
- Paradigm shift for computer vision



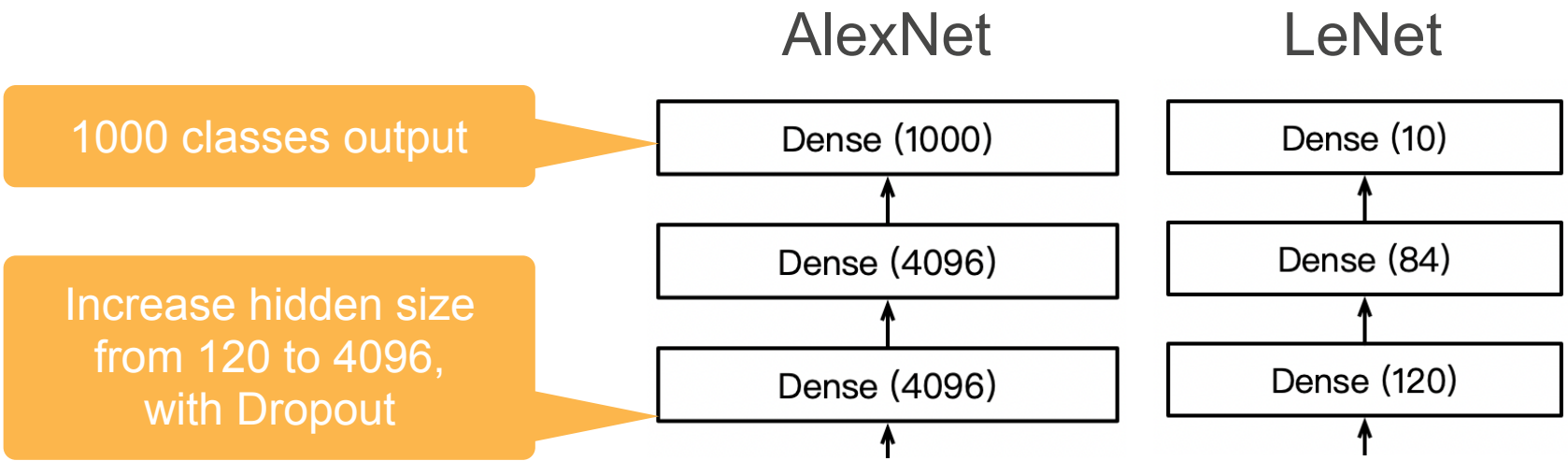
AlexNet Architecture



AlexNet Architecture



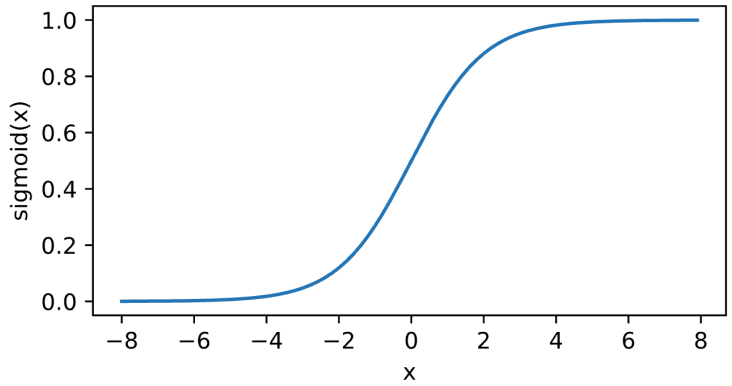
AlexNet Architecture



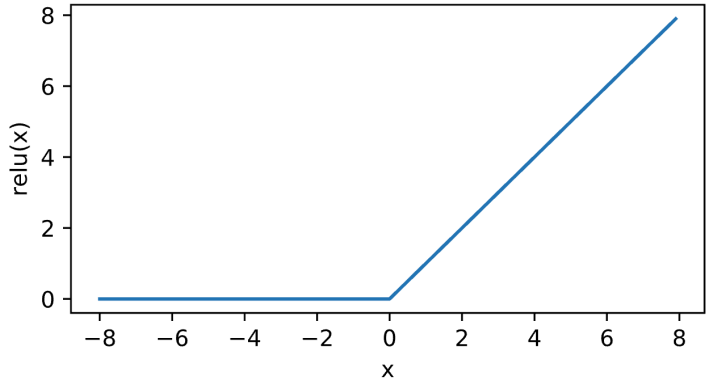
More Tricks

1. Change activation function from sigmoid to **ReLu** (no more vanishing gradient)

$$\text{sigmoid}(x) = \frac{1}{1 + \exp(-x)}$$



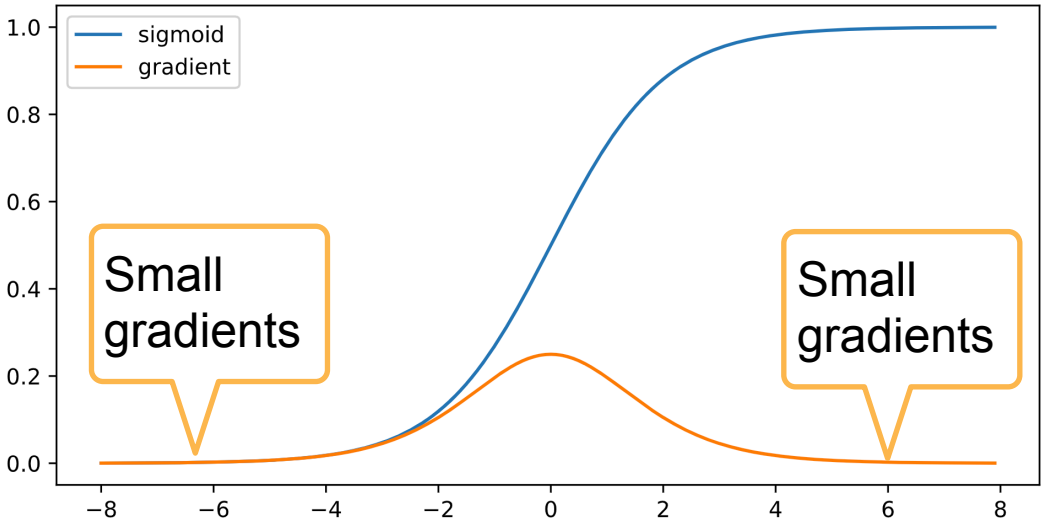
$$\text{ReLU}(x) = \max(x, 0)$$



Gradient Vanishing

- Use sigmoid as the activation function

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



Gradient Vanishing

- Use sigmoid as the activation function

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

- Elements $\prod_{i=t}^{d-1} \frac{\partial \mathbf{h}^{i+1}}{\partial \mathbf{h}^i} = \prod_{i=t}^{d-1} \text{diag}\left(\sigma'(\mathbf{W}^i \mathbf{h}^{i-1})\right)(W^i)^T$ are products of *d-t* small values

$$0.8^{100} \approx 2 \times 10^{-10}$$

Gradient Vanishing Issues

- Gradients with value 0
  - Severe with 16-bit floating points (Range: 6e-5 - 6e4)
- No progress in training
  - No matter how to choose learning rate
- Severe with bottom layers
  - Only top layers are well trained
  - No benefit to make networks deeper

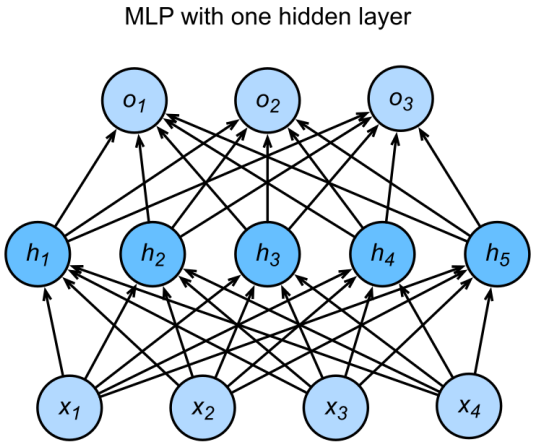
More Tricks

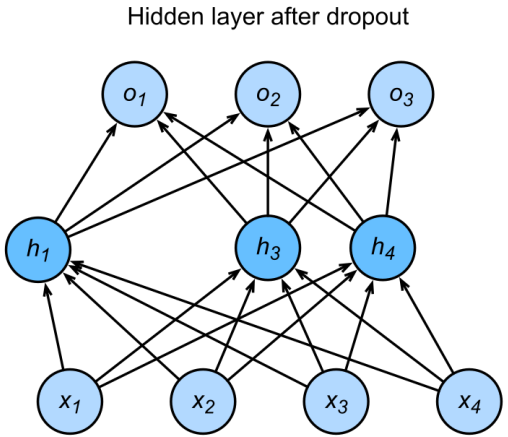
1. Change activation function from sigmoid to **ReLu** (no more vanishing gradient)
2. Add a **dropout** layer after two hidden dense layers (better robustness / regularization)

Dropout

- Often apply dropout on the output of hidden fully-connected layers

$$\mathbf{h} = \sigma(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)$$
$$\mathbf{h}' = \text{dropout}(\mathbf{h})$$
$$\mathbf{o} = \mathbf{W}_2\mathbf{h}' + \mathbf{b}_2$$
$$\mathbf{y} = \text{softmax}(o)$$





Dropout - Inference

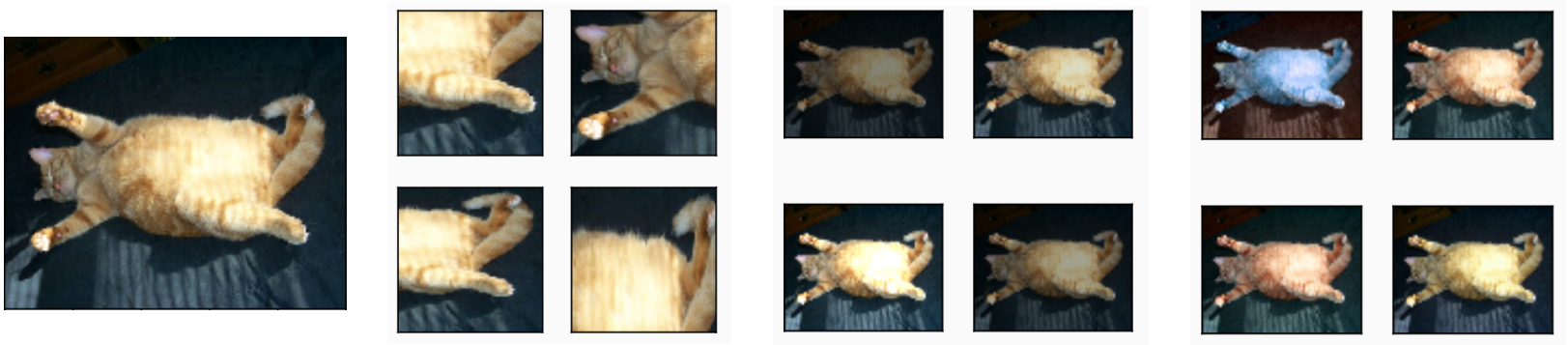
- Regularization is only used in training
- The dropout layer for inference is

$$\mathbf{h}' = \text{dropout}(\mathbf{h})$$

  - Guarantee deterministic results

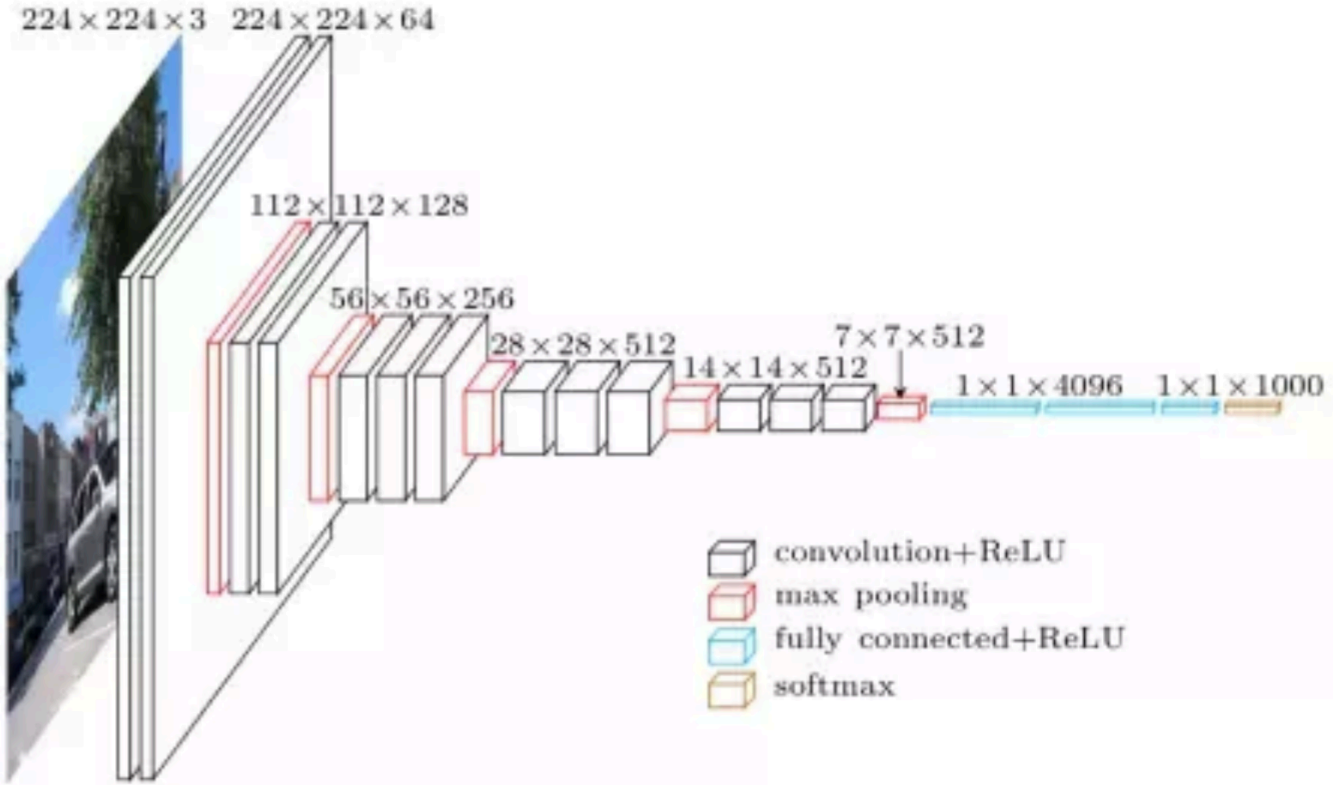
More Tricks

1. Change activation function from sigmoid to **ReLu** (no more vanishing gradient)
2. Add a **dropout** layer after two hidden dense layers (better robustness / regularization)
3. Data augmentation



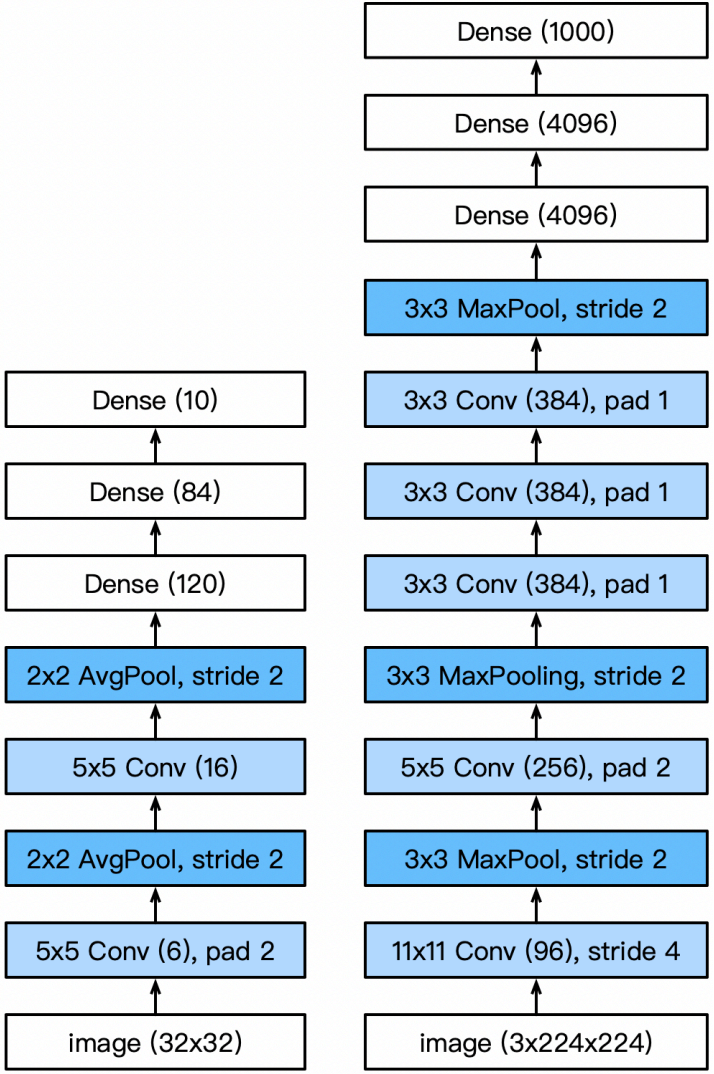
AlexNet Notebook

VGG


224 × 224 × 3
224 × 224 × 64
112 × 112 × 128
56 × 56 × 256
28 × 28 × 512
14 × 14 × 512
7 × 7 × 512
1 × 1 × 4096
1 × 1 × 1000
convolution+ReLU
max pooling
fully connected+ReLU
softmax

VGG

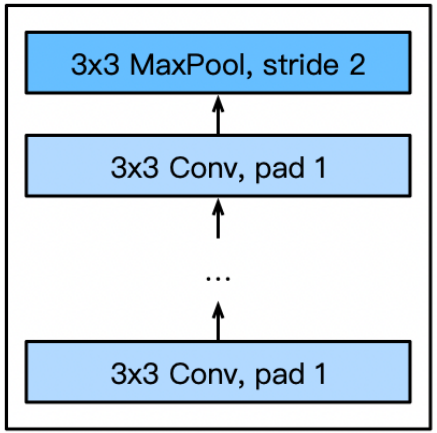
- AlexNet is deeper and bigger than LeNet to get performance
- Go even bigger & deeper?
- Options
  - More dense layers (too expensive)
  - **More** convolutions
  - Group into **blocks**



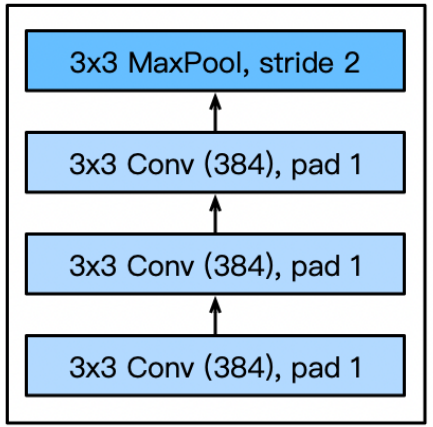
VGG Blocks

- Deeper vs. wider?
  - 5x5 convolutions
  - 3x3 convolutions (more)
  - **Deep & narrow better**
- VGG block
  - *3x3* convolutions (pad 1) **(n layers, m channels)**
  - 2x2 max-pooling (stride 2)

VGG block

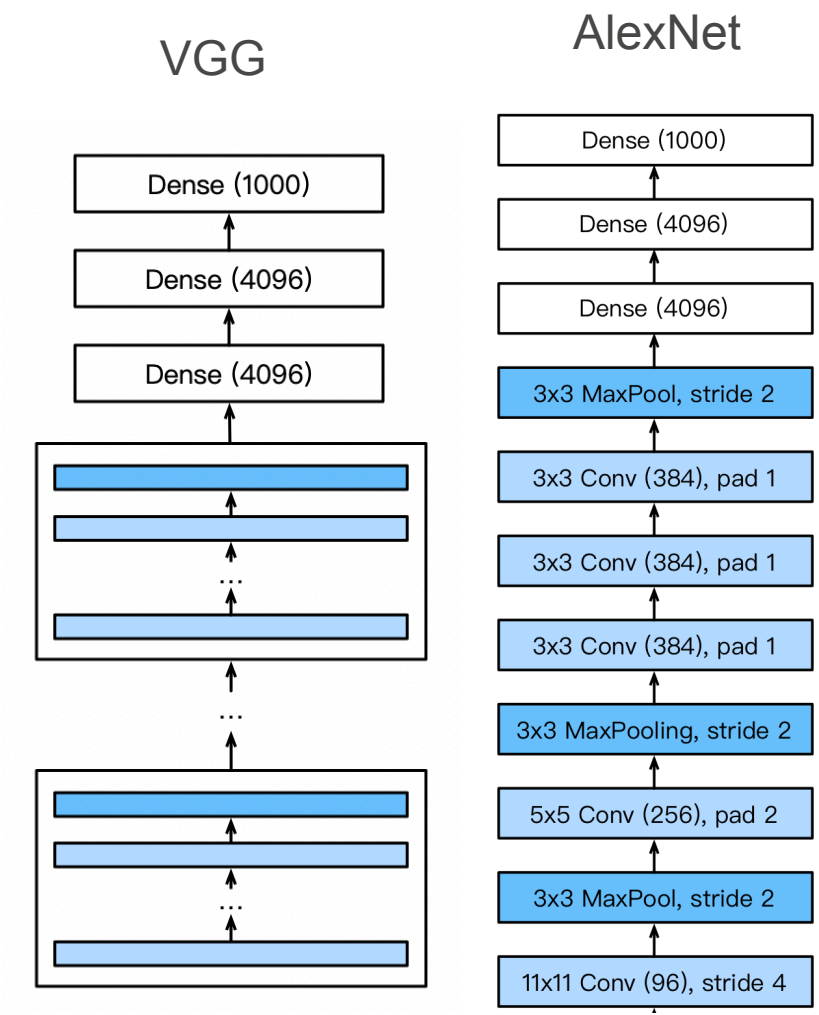


Part of AlexNet



VGG Architecture

- Multiple VGG blocks followed by dense layers
- Vary the repeating number to get different architectures, such as VGG-16, VGG-19, …

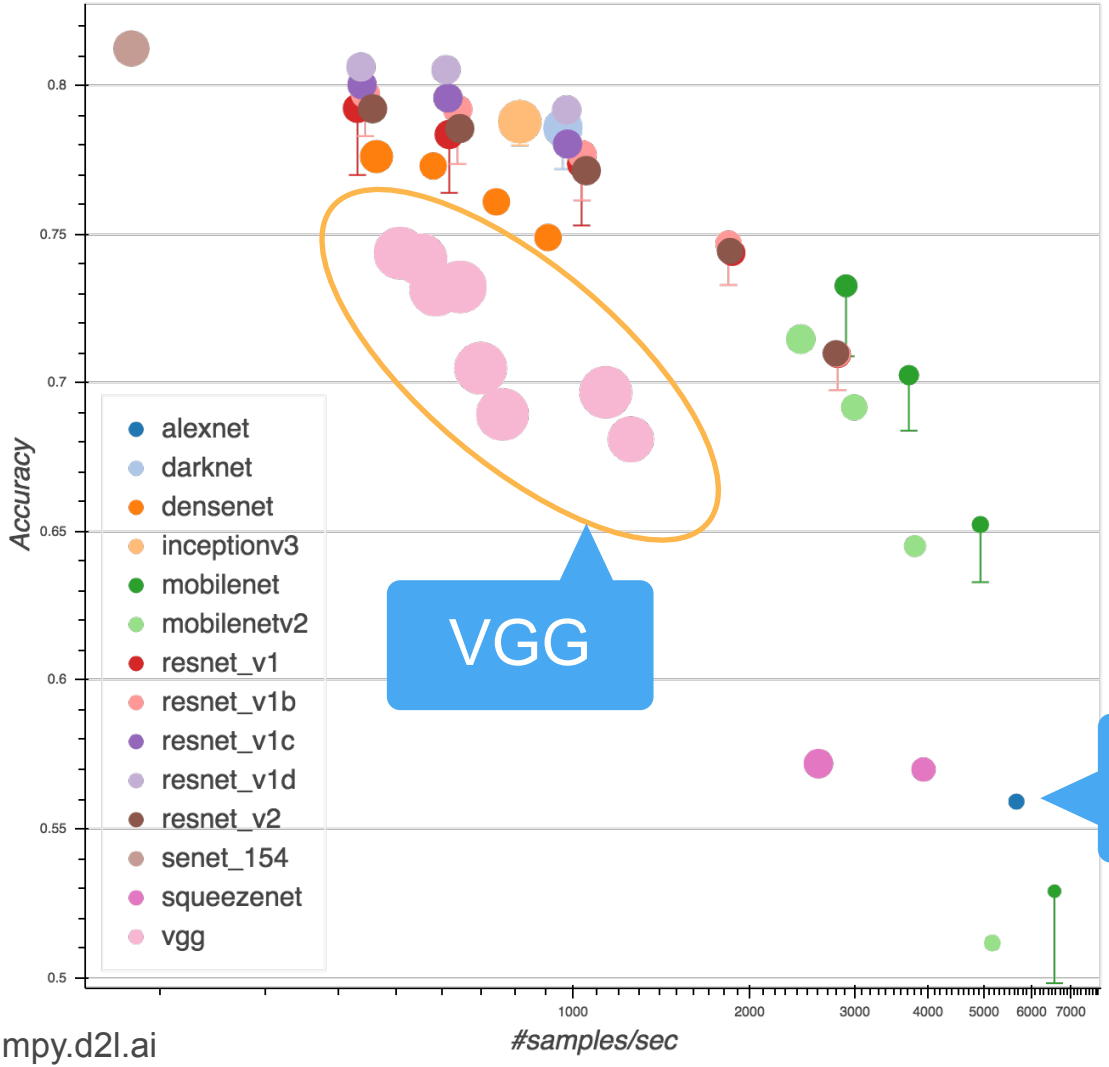


Progress

- LeNet (1995)
  - 2 convolution + pooling layers
  - 2 hidden dense layers
- AlexNet
  - Bigger and deeper LeNet
  - ReLu, Dropout, preprocessing
- VGG
  - Bigger and deeper AlexNet (repeated VGG blocks)

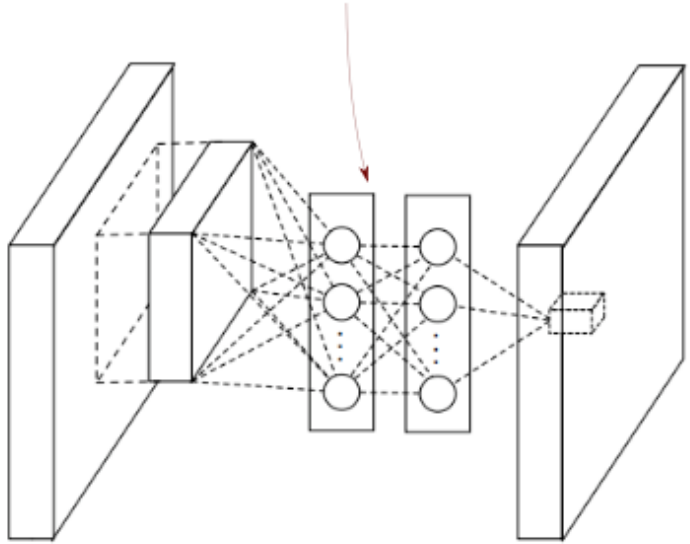
GluonCV Model Zoo

gluon-cv.mxnet.io/model_zoo/classification.html

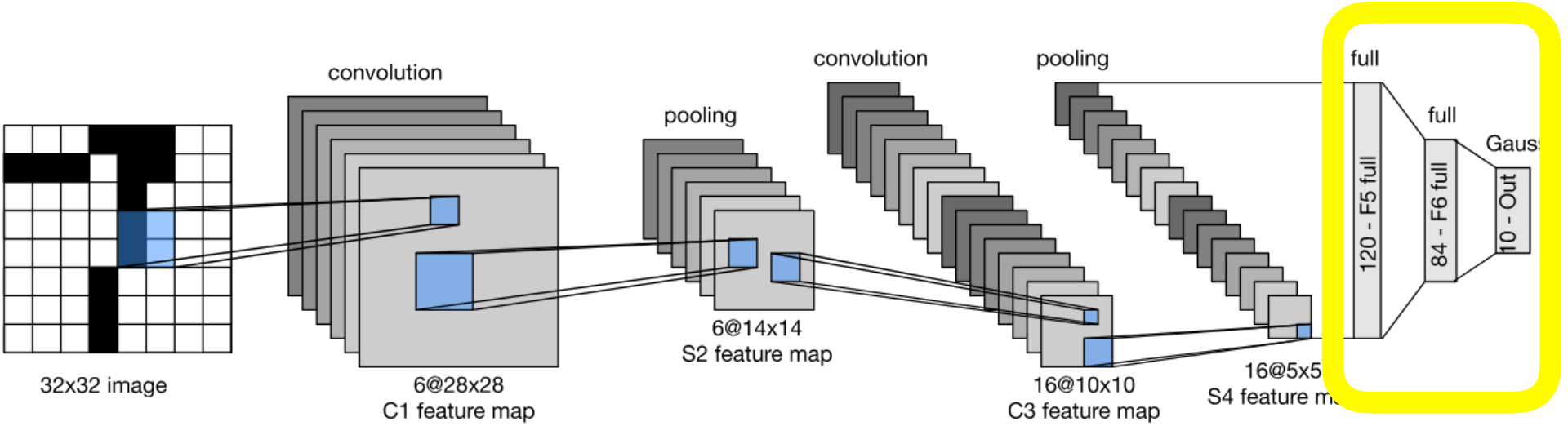


Network in Network (NiN)

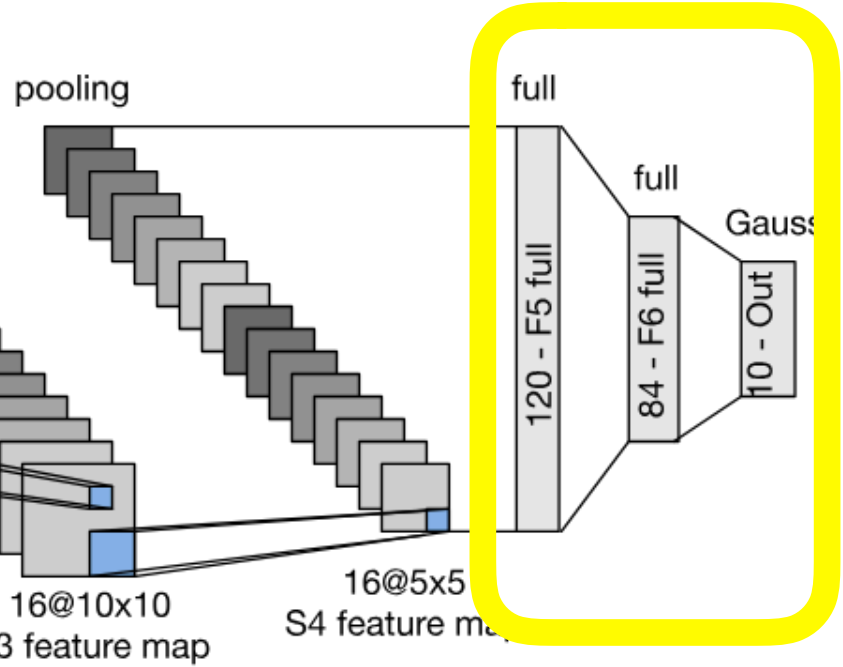
Non linear mapping introduced by mlpconv layer consisting of multiple fully connected layers with non linear activation function.



The Curse of the Last Layer(s)



The Last Layer(s)



- Convolution layers need relatively few parameters

$$c_i \times c_o \times k^2$$

- Last layer needs many parameters for n classes

$$c \times m_w \times m_h \times n$$

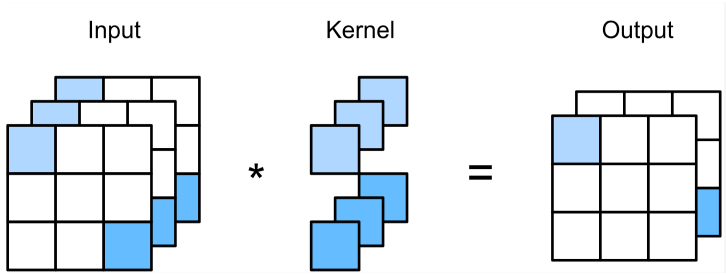
- LeNet 16x5x5x120 = 48k
- AlexNet 256x5x5x4096 = 26M
- VGG 512x7x7x4096 = 102M

Breaking the Curse of the Last Layer

- Key Idea
  - **Get rid of the fully connected last layer(s)**
  - Convolutions and pooling reduce resolution (e.g. stride of 2 reduces resolution 4x)

Breaking the Curse of the Last Layer

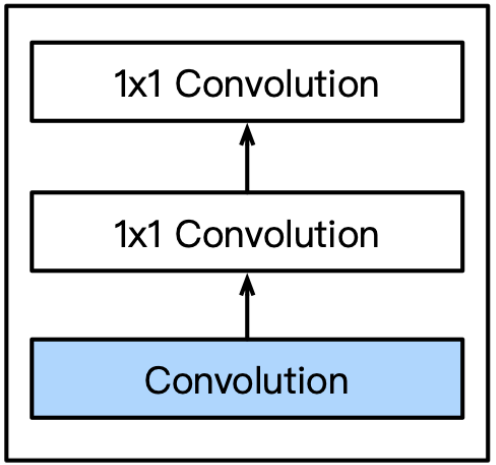
- Key Idea
  - **Get rid of the fully connected last layer(s)**
  - Convolutions and pooling reduce resolution (e.g. stride of 2 reduces resolution 4x)
- Implementation details
  - Reduce resolution progressively
  - Increase number of channels
  - Use 1x1 convolutions (they only act per pixel)
  - Global average pooling in the end



aws

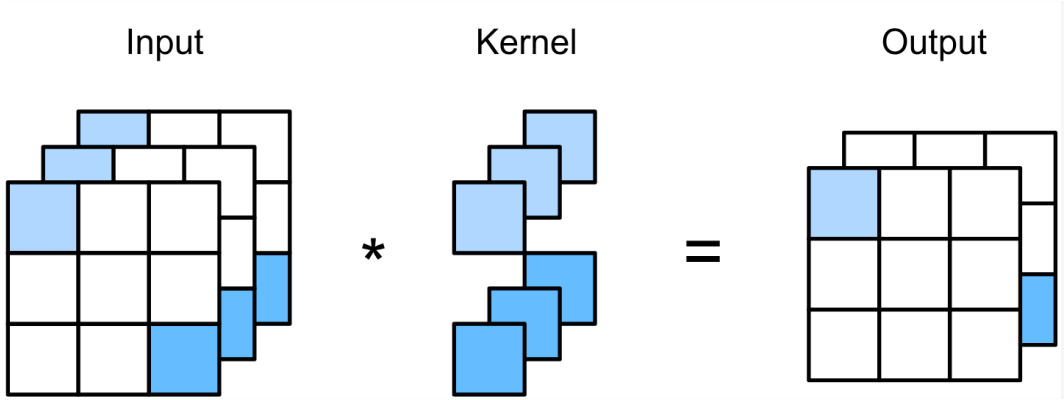
NiN Block

- A convolutional layer
  - kernel size, stride, and padding are hyper-parameters
- Following by two 1x1 convolutions
  - 1 stride and no padding, share the same output channels as first layer
  - Act as dense layers



1 x 1 Convolutional Layer

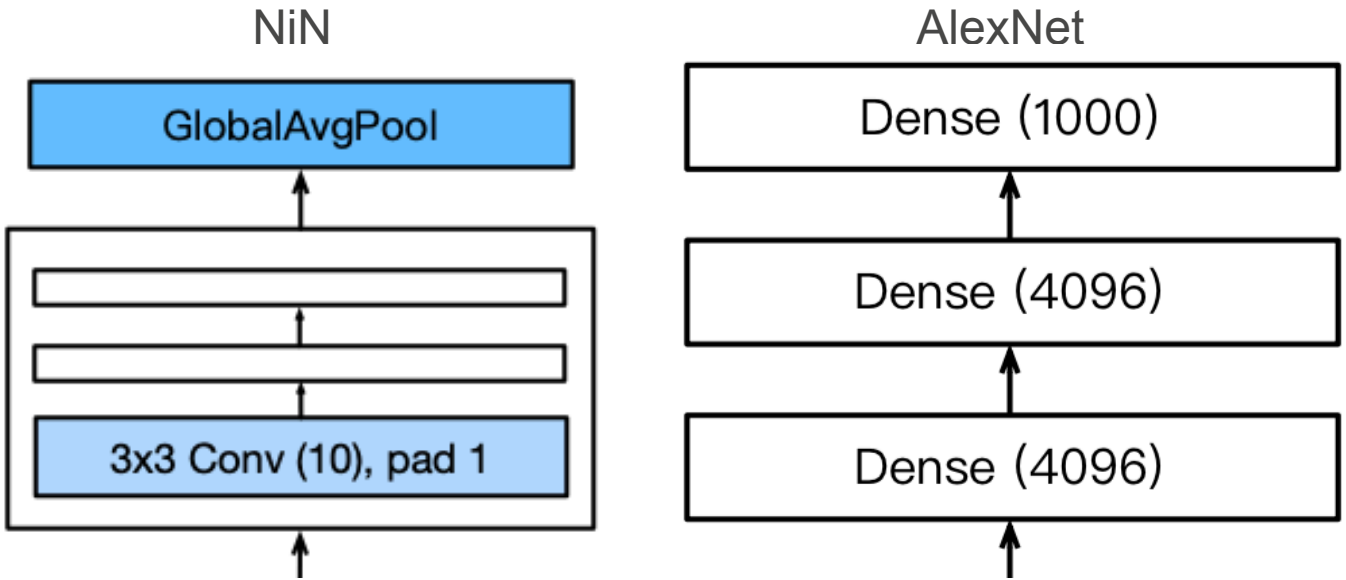
$k_h = k_w = 1$ is a popular choice. It doesn't recognize spatial patterns, but fuse channel dimensions.



Equal to a dense layer with $n_h n_w \times c_i$ input and $c_o \times c_i$ weight.

NiN Last Layers

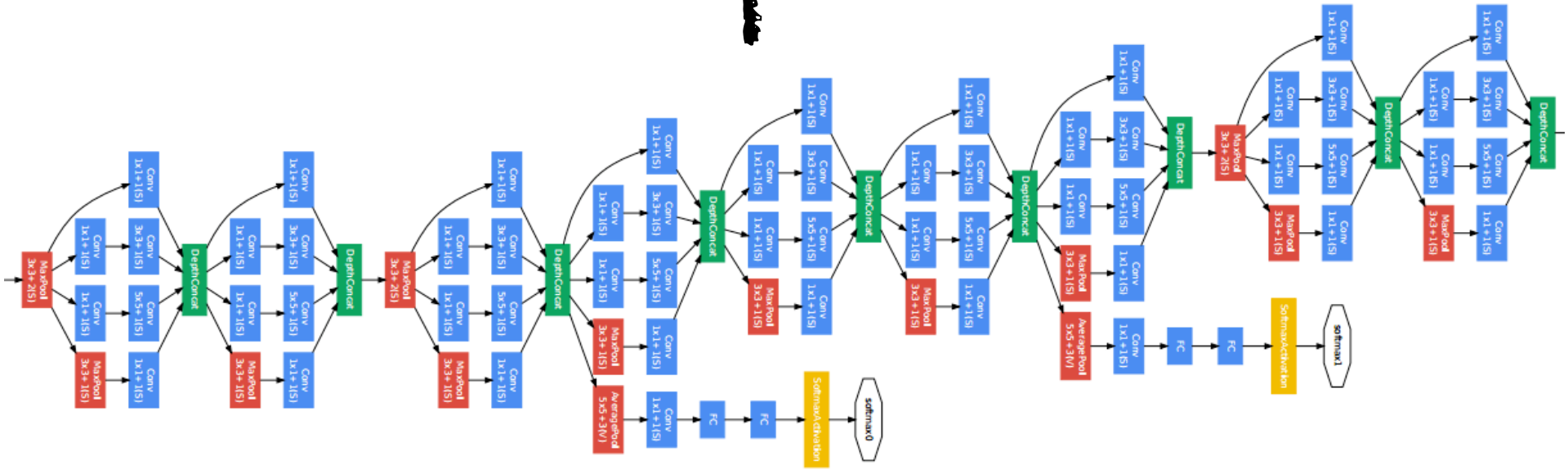
- Replaced AlexNet's dense layers with a NiN block
- Global average pooling layer to combine outputs



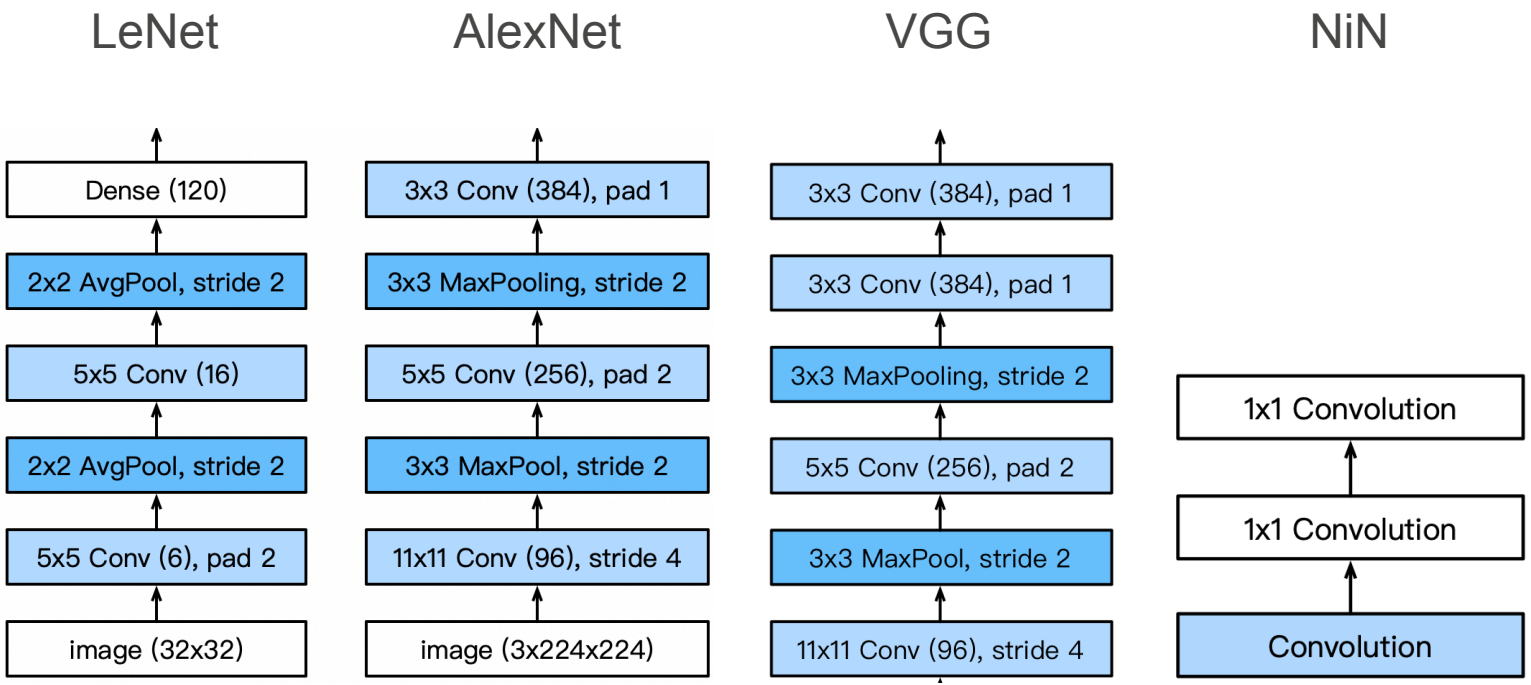
Summary

- **LeNet** (the first convolutional neural network)
- **AlexNet**
  - More of everything
  - ReLu, Dropout, MaxPooling
- **VGG**
  - Even more of everything (narrower and deeper)
  - Repeated blocks
- **NiN**
  - 1x1 convolutions + global pooling instead of dense

Inception


MaxPool 3x3+ 2(S)
Conv 1x1+ 1(S)
Conv 3x3+ 1(S)
Conv 5x5+ 1(S)
DepthConcat
AveragePool 5x5+ 3(V)
FC
SoftmaxActivation
softmax0
softmax1

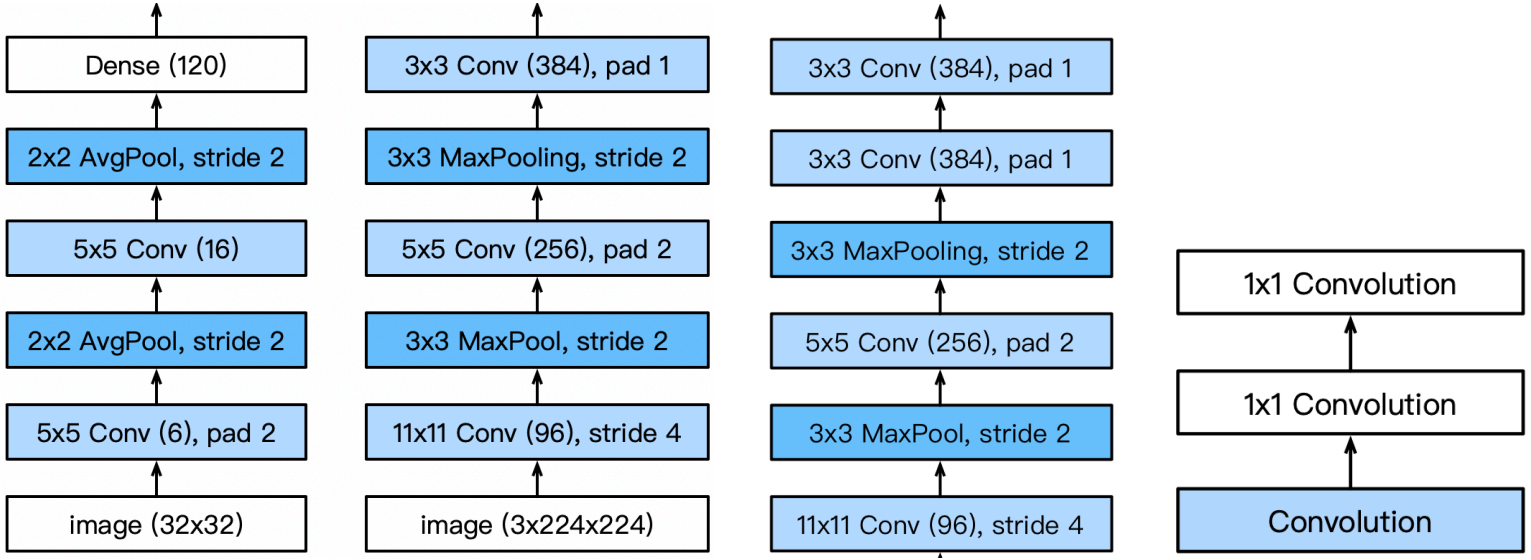
Picking the best convolution …



Picking the best convolution …

1x1 | 3x3 | 5x5 | Max pooling | Multiple 1x1

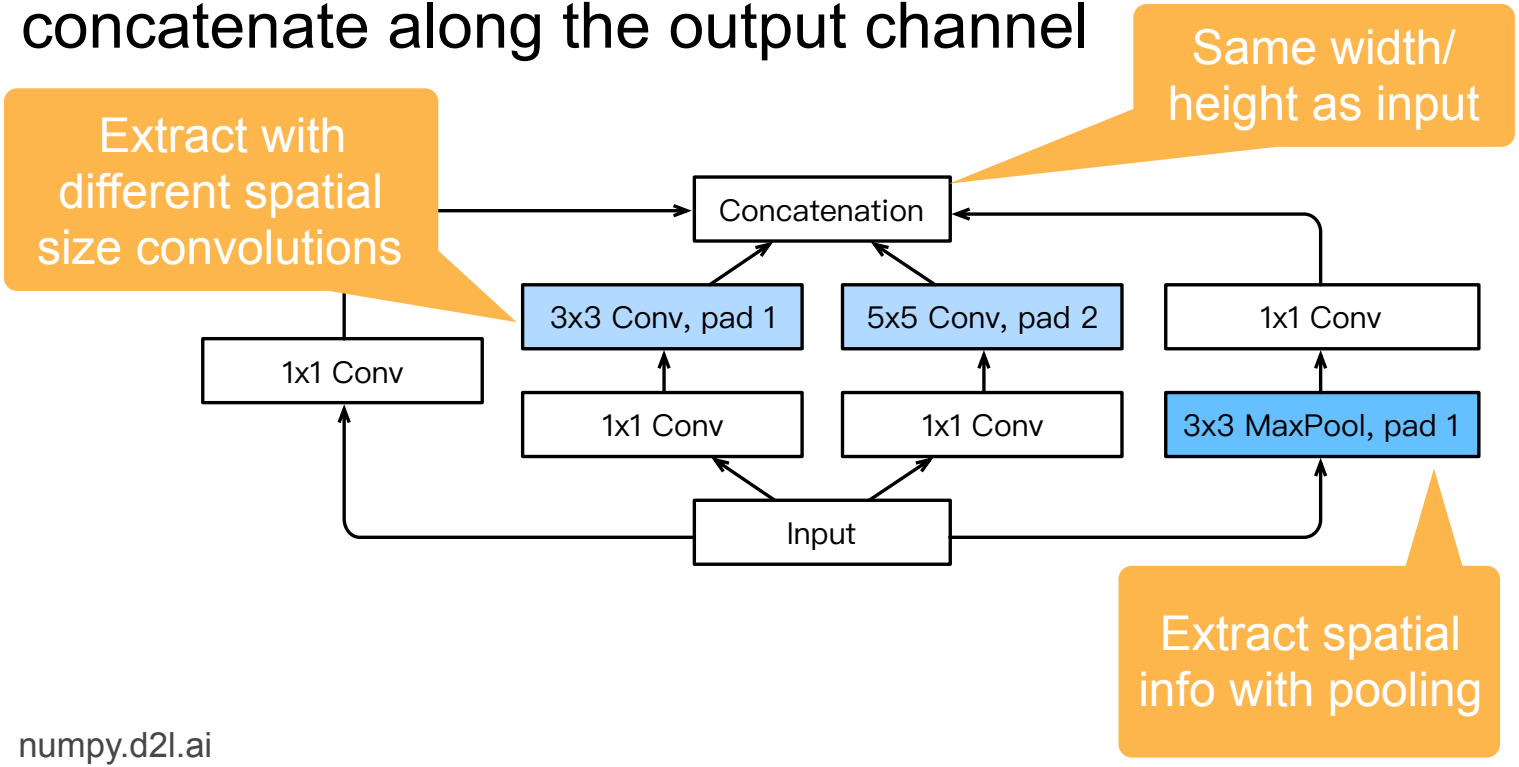
LeNet | AlexNet | VGG | NiN



Why choose? Just pick them all.

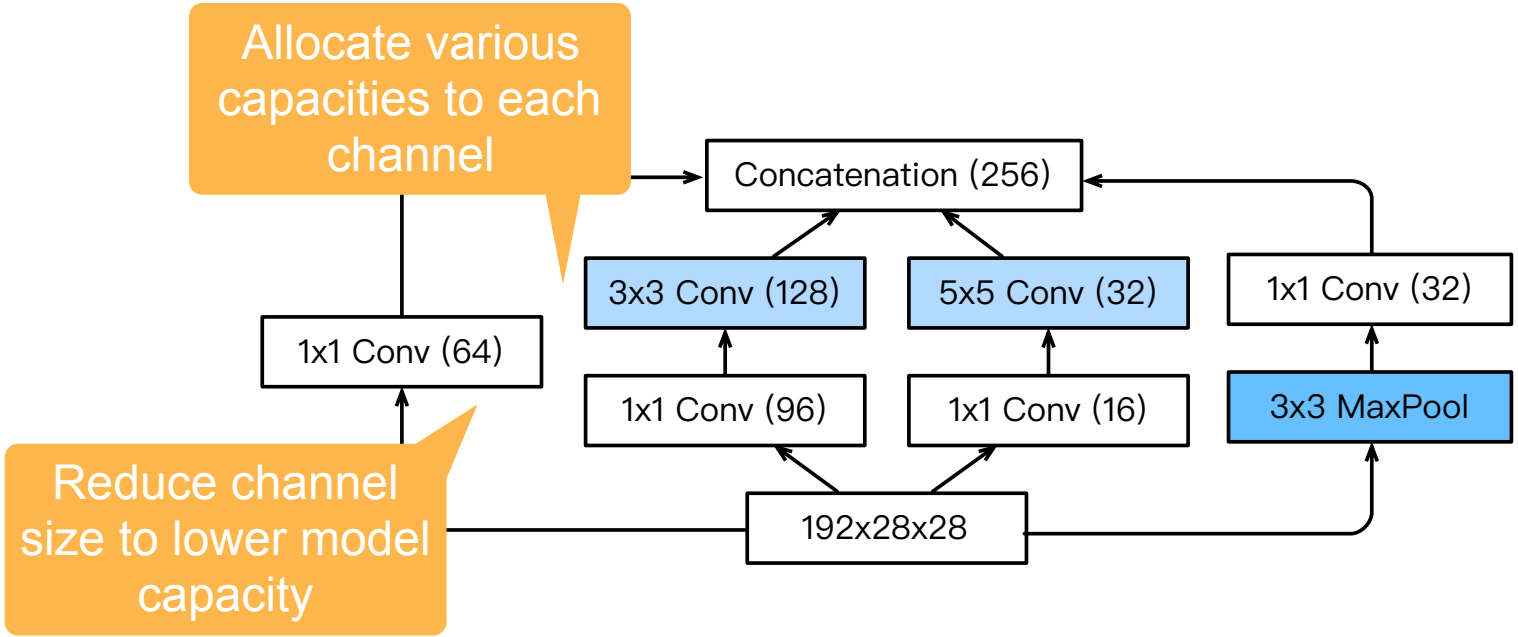
Inception Blocks

4 paths extract information from different aspects, then concatenate along the output channel

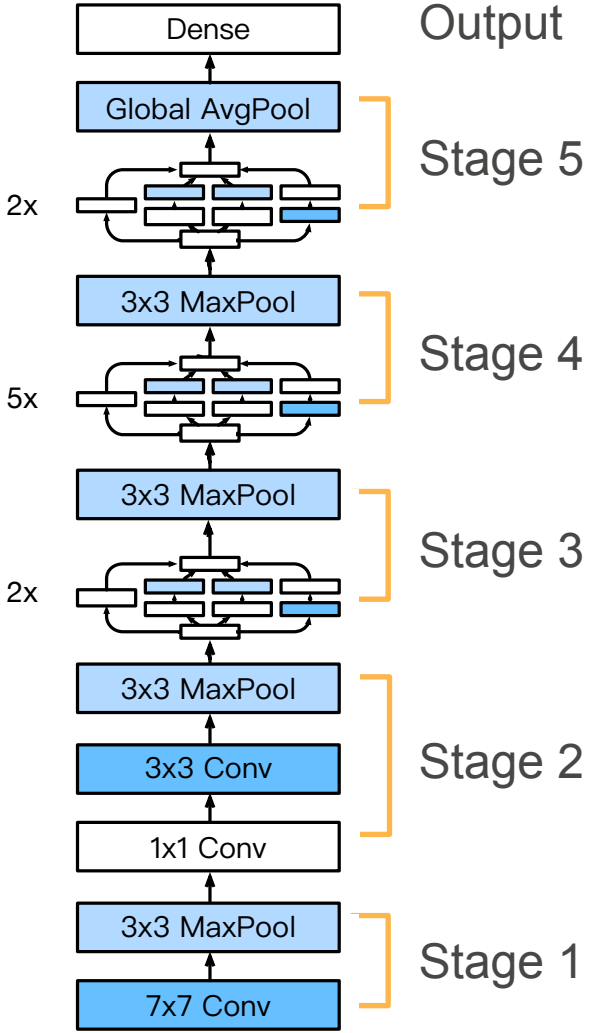


Inception Blocks

The first inception block with channel sizes specified

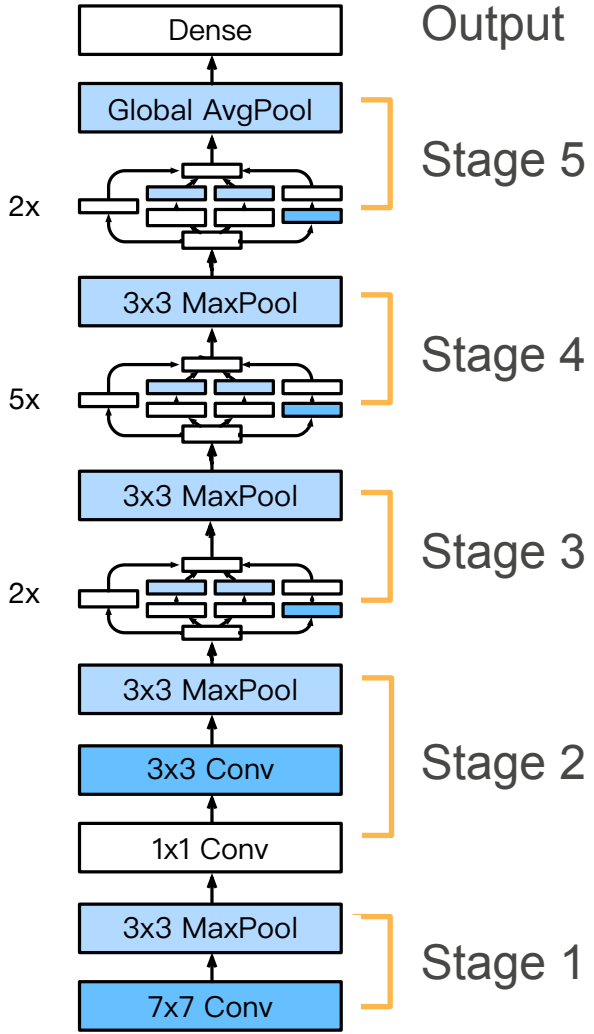


GoogLeNet


Dense
Output
Global AvgPool
Stage 5
2x
3x3 MaxPool
Stage 4
5x
3x3 MaxPool
Stage 3
2x
3x3 MaxPool
3x3 Conv
Stage 2
1x1 Conv
3x3 MaxPool
Stage 1
7x7 Conv

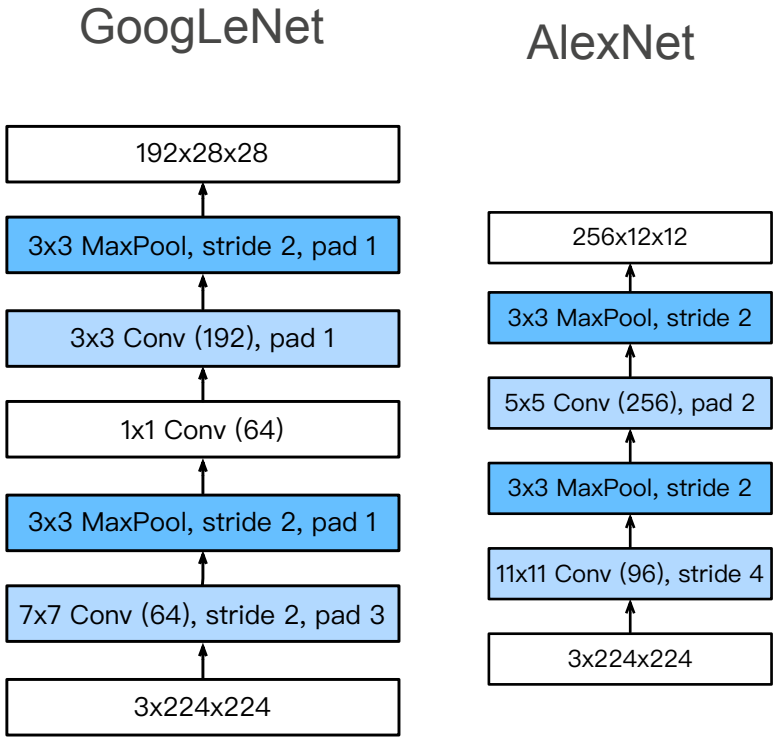
GoogLeNet

- 5 stages with 9 inception blocks
- max pooling after each stage
- global average pooling in the end
- first part is identical to AlexNet and LeNet

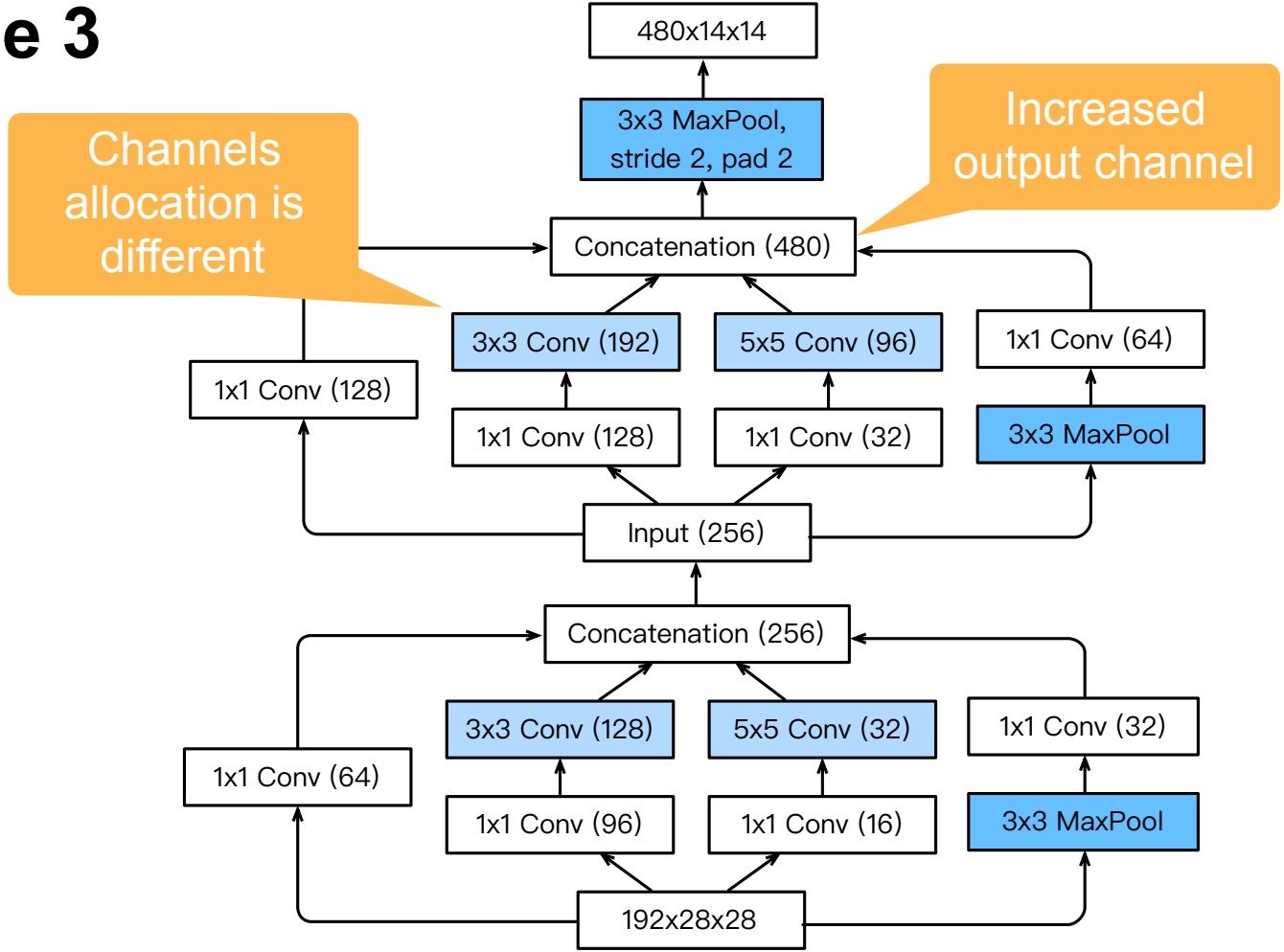


Stage 1 & 2

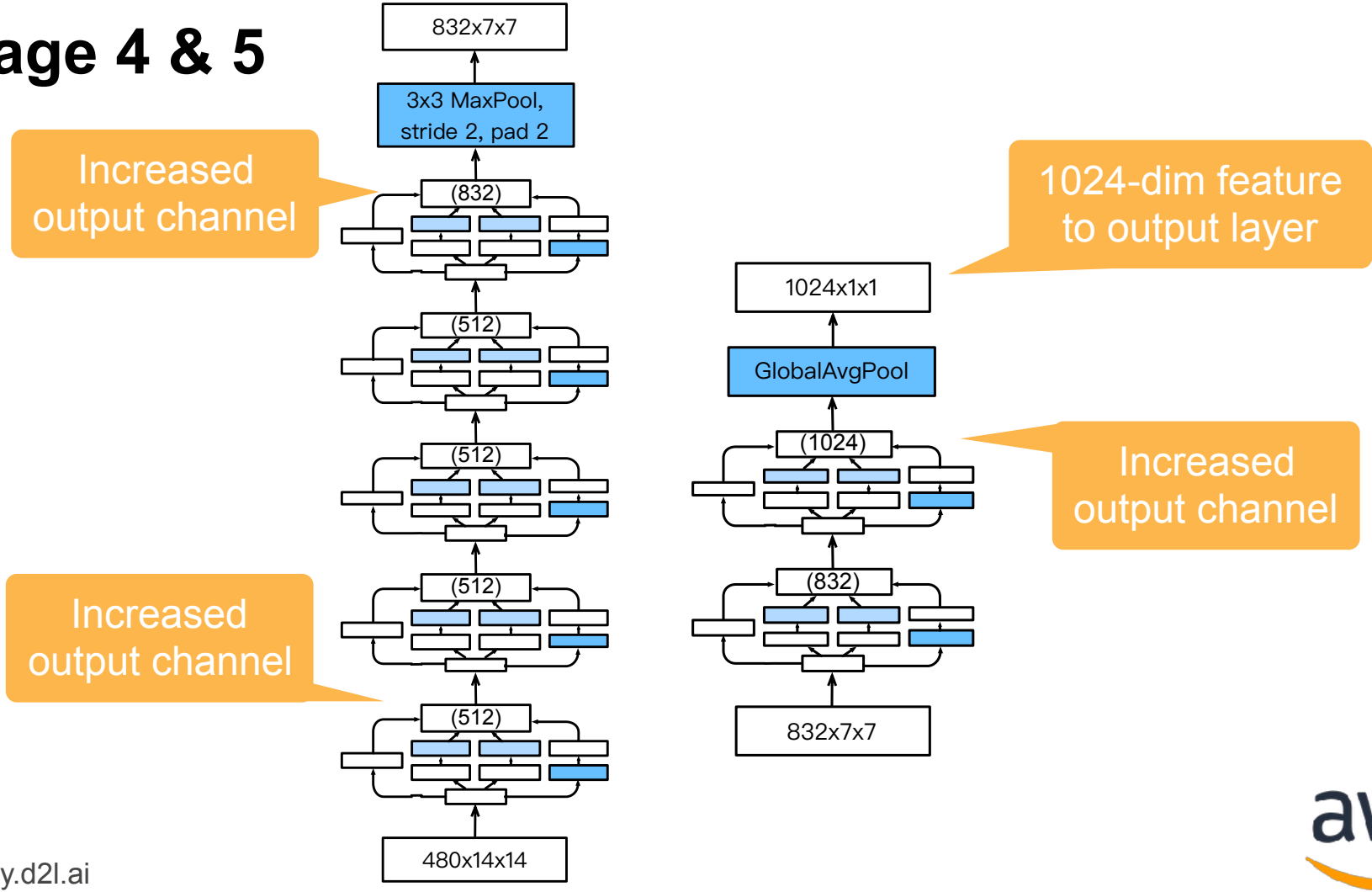
- Smaller kernel size and output channels due to more layers

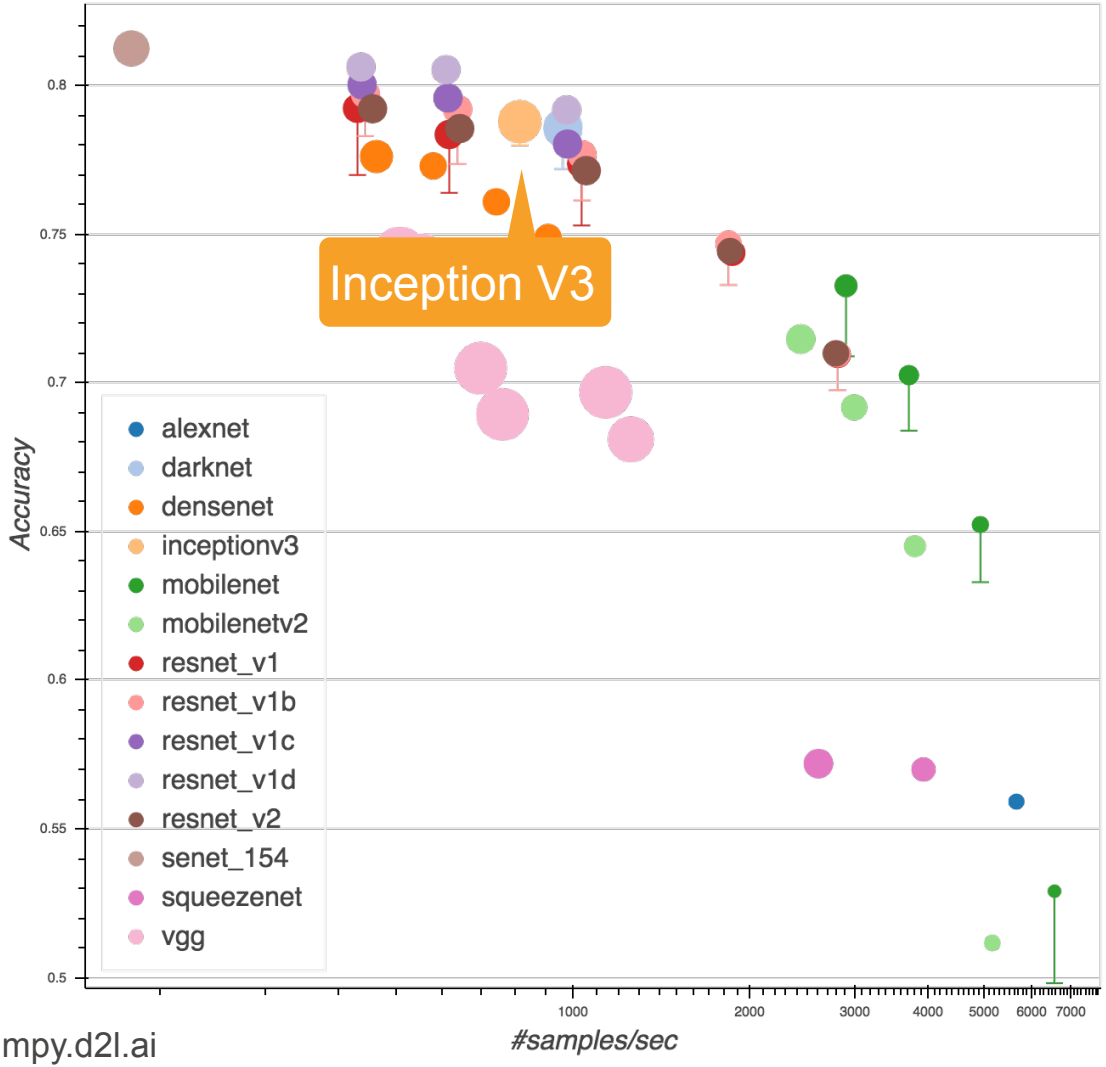


Stage 3


480x14x14
3x3 MaxPool, stride 2, pad 2
Increased output channel
Channels allocation is different
Concatenation (480)
3x3 Conv (192)
5x5 Conv (96)
1x1 Conv (64)
1x1 Conv (128)
1x1 Conv (128)
1x1 Conv (32)
3x3 MaxPool
Input (256)
Concatenation (256)
3x3 Conv (128)
5x5 Conv (32)
1x1 Conv (32)
1x1 Conv (64)
1x1 Conv (96)
1x1 Conv (16)
3x3 MaxPool
192x28x28

Stage 4 & 5


832x7x7
3x3 MaxPool, stride 2, pad 2
Increased output channel
(832)
(512)
(512)
(512)
(512)
480x14x14
1024-dim feature to output layer
1024x1x1
GlobalAvgPool
(1024)
Increased output channel
(832)
832x7x7



GluonCV Model Zoo

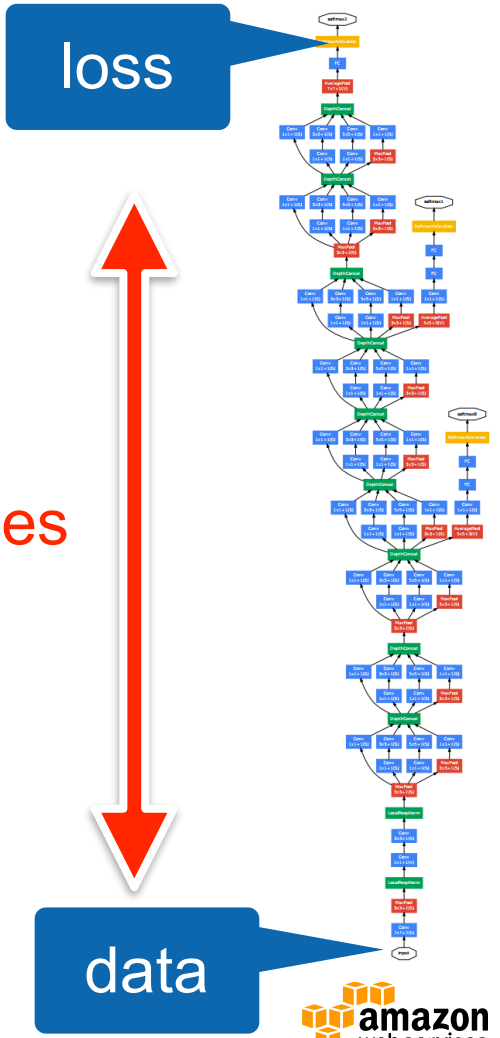
https://gluon-cv.mxnet.io/model_zoo/classification.html

aws


SMALL BATCH NORM
• PACIFICA, CALIFORNIA •
Batch # 22716-06
Hand Made All Natural
BLACKBERRY JAM

Batch Normalization

- Loss occurs at last layer
  - Last layers learn quickly
- Data is inserted at bottom layer
  - Bottom layers change - **everything** changes
  - Last layers need to relearn many times
  - Slow convergence
- This is like covariate shift
  Can we avoid changing last layers while learning first layers?



Batch Normalization

- Can we avoid changing last layers while learning first layers?
- Fix mean and variance

$$\mu_B = \frac{1}{|B|}\sum_{i \in B} x_i \text{ and } \sigma_B^2 = \frac{1}{|B|}\sum_{i \in B}(x_i - \mu_B)^2 + \epsilon$$

and adjust it separately

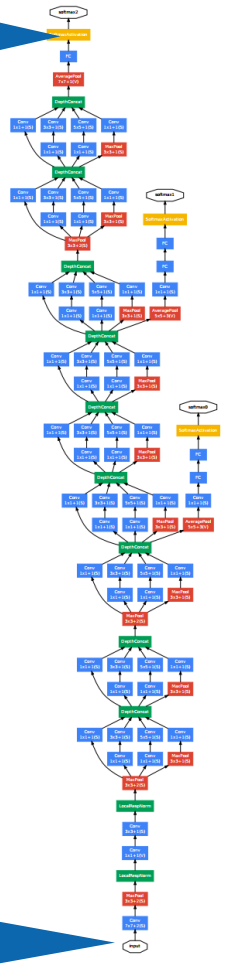
$$x_{i+1} = \gamma \frac{x_i - \mu_B}{\sigma_B} + \beta$$

mean

variance

loss

data



What Batch Norms really do

- Doesn't really reduce covariate shift (Lipton et al., 2018)
- Regularization by noise injection

$$x_{i+1} = \gamma \frac{x_i - \hat{\mu}_B}{\hat{\sigma}_B} + \beta$$

Empirical mean

Empirical std.

  - Random shift per minibatch
  - Random scale per minibatch
- No need to mix with dropout (both are capacity control)
- Ideal minibatch size of 64 to 256

What Batch Norms really do

- Doesn't really reduce covariate shift (Lipton et al., 2018)
- Regularization by noise injection

$$x_{i+1} = \gamma \frac{x_i - \hat{\mu}_B}{\hat{\sigma}_B} + \beta$$

Random offset

Random scale

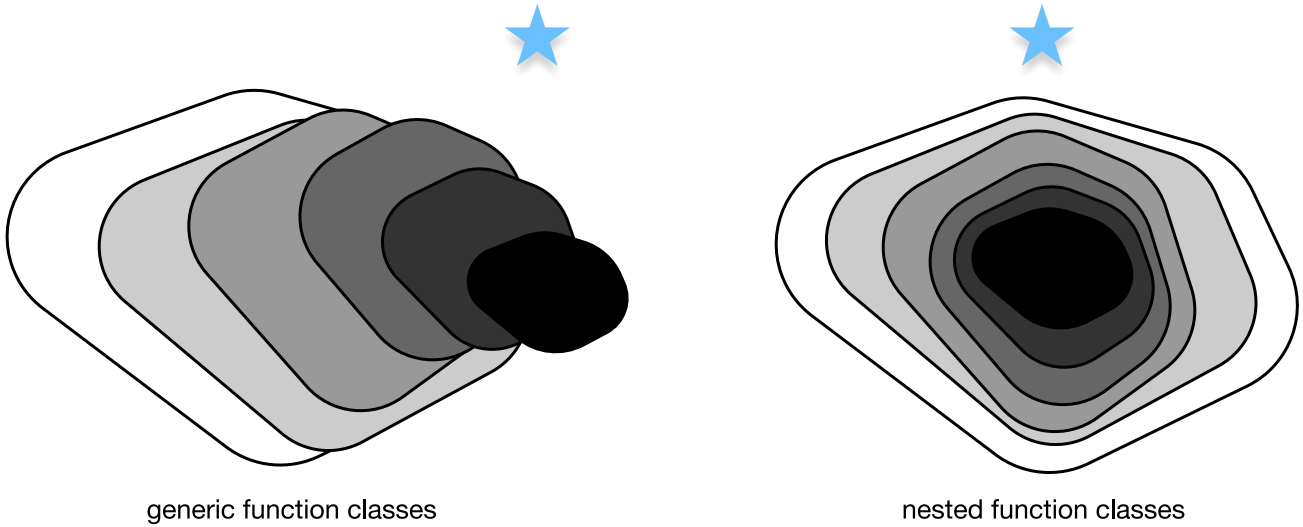
  - Random shift per minibatch
  - Random scale per minibatch
- No need to mix with dropout (both are capacity control)
- Ideal minibatch size of 64 to 256

Details

`gluon.nn.BatchNorm(…)`

- **Dense Layer**
  One normalization for all
- **Convolution**
  One normalization per channel
- Compute **new mean and variance** for every minibatch
  - Effectively acts as regularization
  - Optimal minibatch size is ~128
    (watch out for parallel training with many machines)

Residual Networks

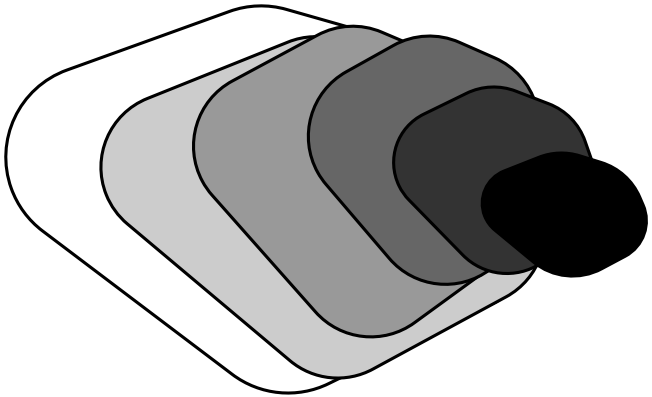


generic function classes

nested function classes

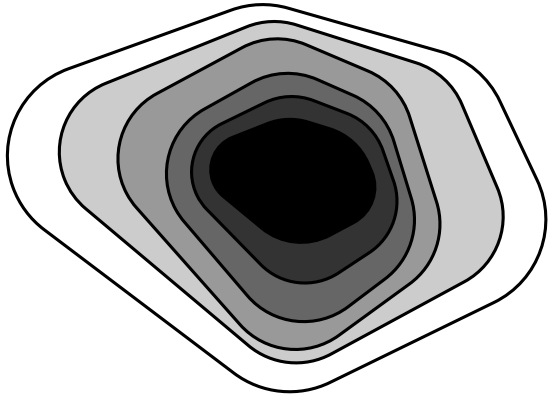
Does adding layers improve accuracy?

$$f^*_{\mathcal{F}} := \underset{f}{\operatorname{argmin}}\, L(X, Y, f) \text{ subject to } f \in \mathcal{F}$$



generic function classes

?



nested function classes

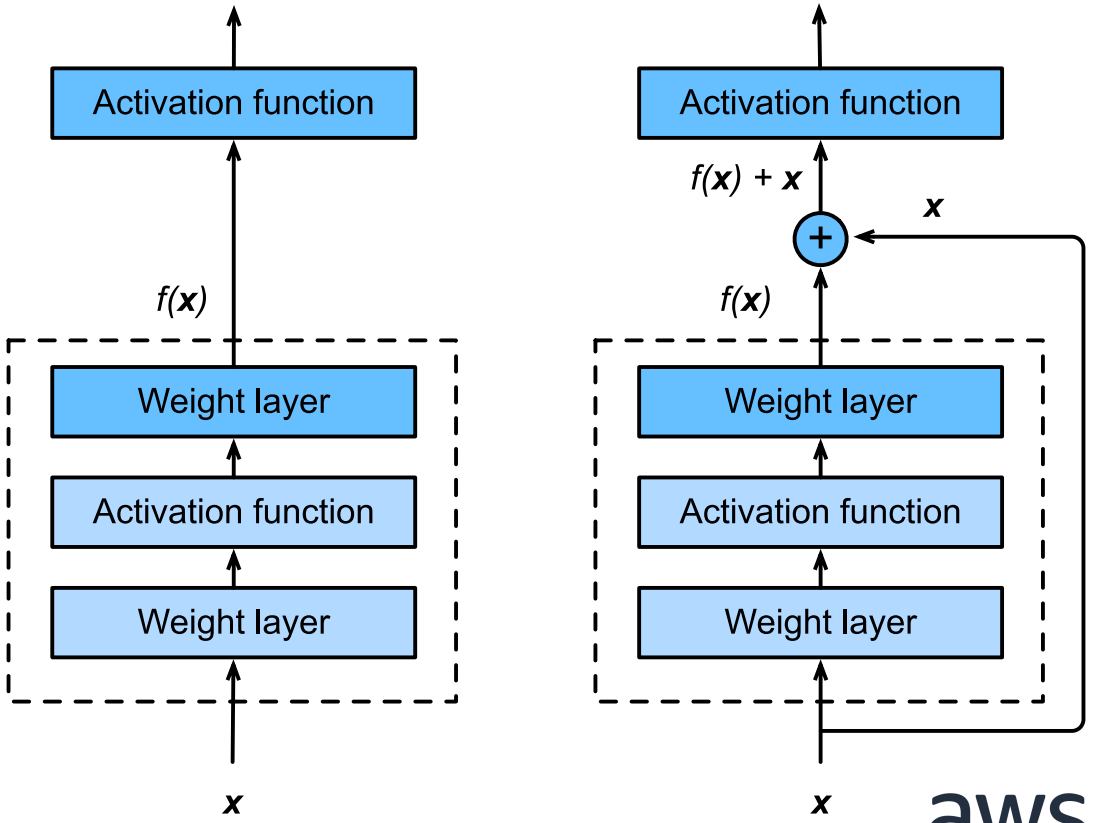
✔

aws

Residual Networks

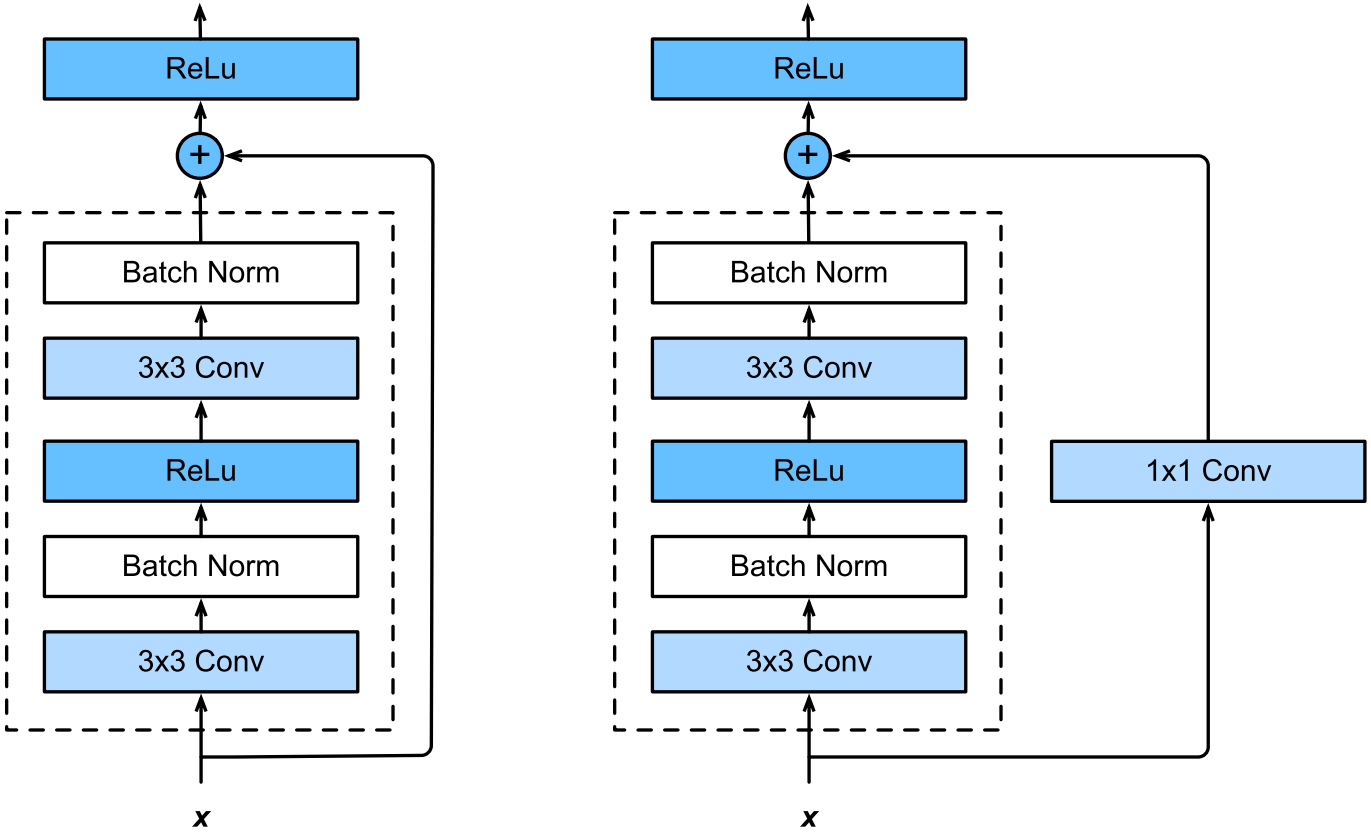
- Adding a layer **changes** function class
- We want to **add to** the function class
- 'Taylor expansion' style parametrization

$$f(x) = x + g(x)$$

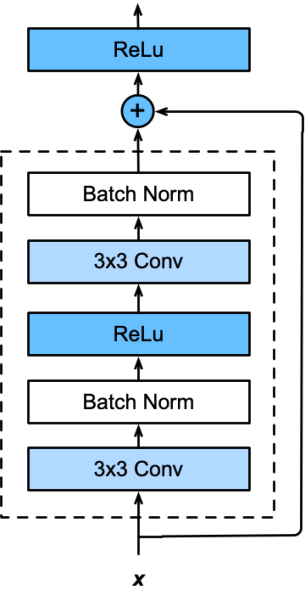


He et al., 2015

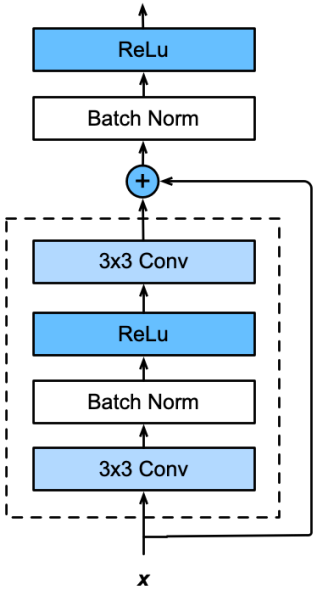
ResNet Block in detail



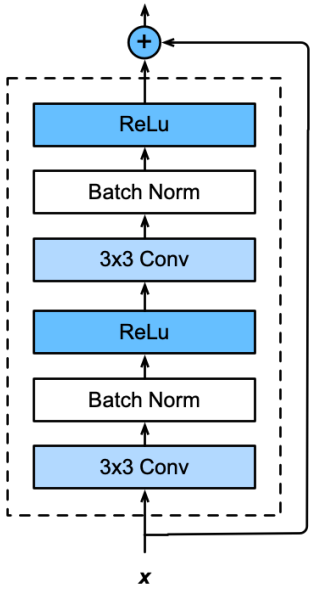
The many flavors of ResNet blocks



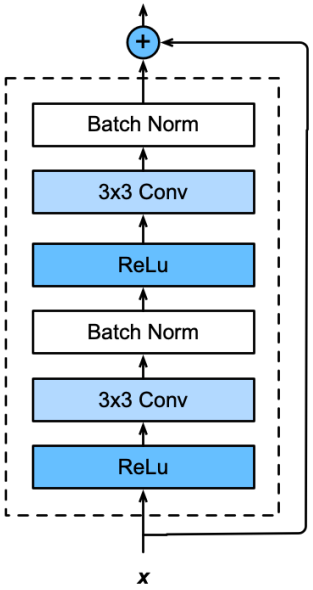
original



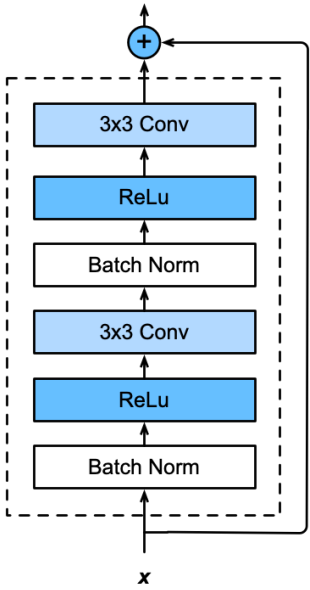
Batch Norm after addition



ReLu last



Batch Norm last



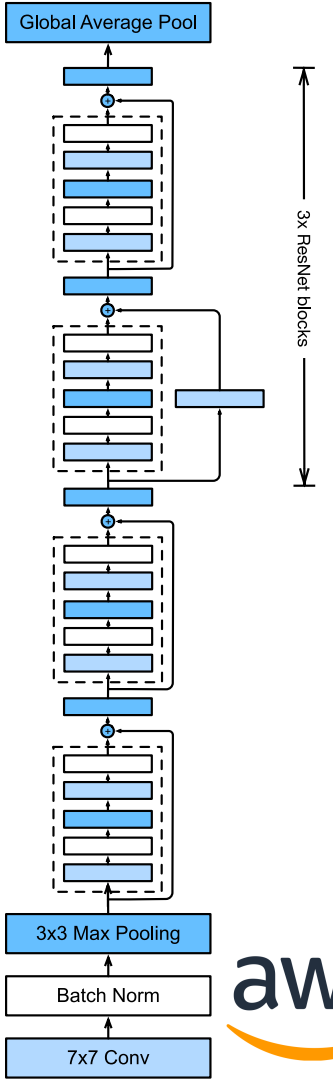
Pre-activation

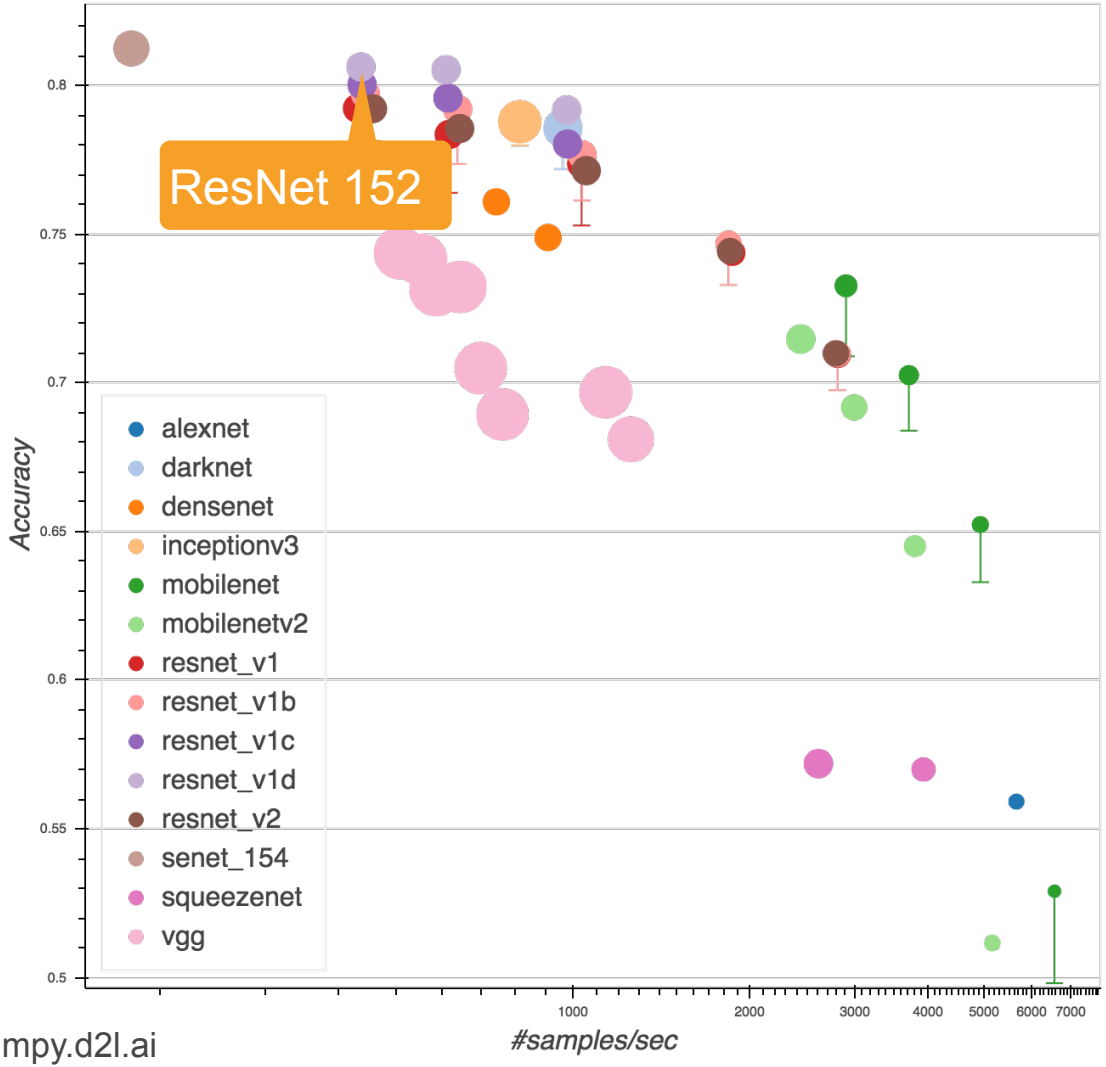
Try every permutation

Putting it all together

- Same block structure as e.g. VGG or GoogleNet
- Residual connection to add to expressiveness
- Pooling/stride for dimensionality reduction
- Batch Normalization for capacity control

… train it at scale …





GluonCV Model Zoo

https://gluon-cv.mxnet.io/model_zoo/classification.html

aws

Jupyter Notebook

More Ideas

DenseNet (Huang et al., 2016)

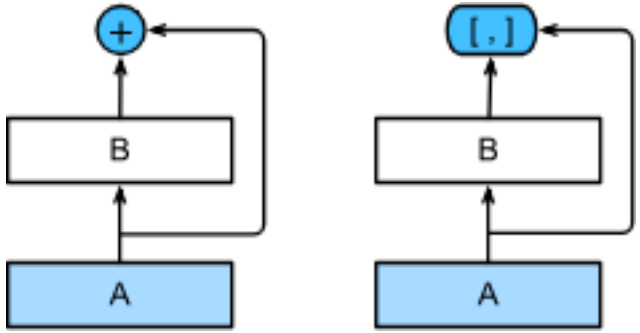
- ResNet combines x and f(x)
- DenseNet uses higher order 'Taylor series' expansion

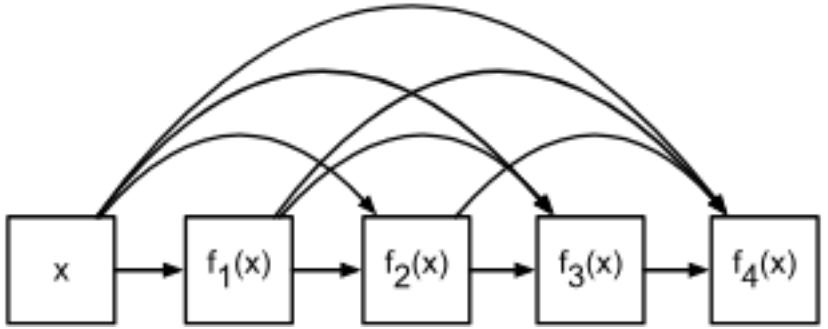
$$x_{i+1} = [x_i, f_i(x_i)]$$

$$x_1 = x$$

$$x_2 = [x, f_1(x)]$$

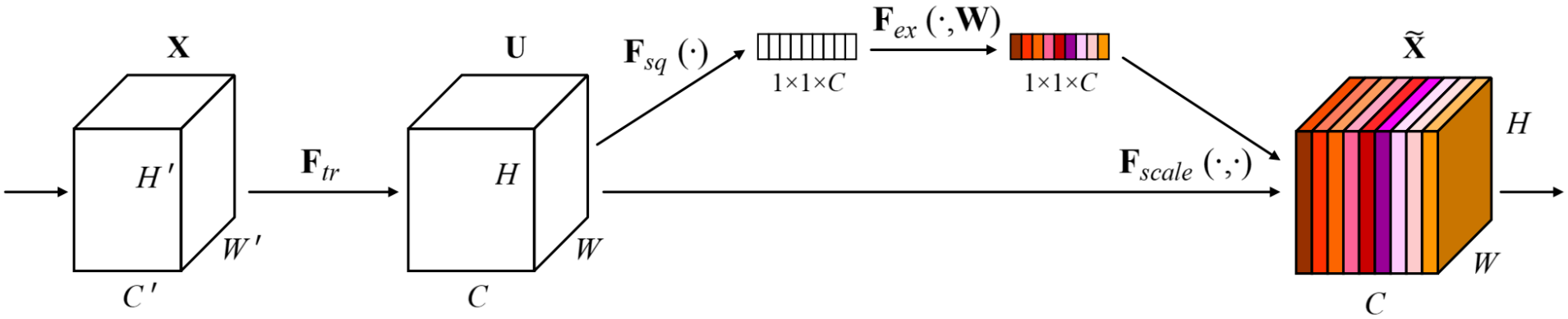
$$x_2 = [x, f_1(x), f_2([x, f_1(x)])]$$





aws

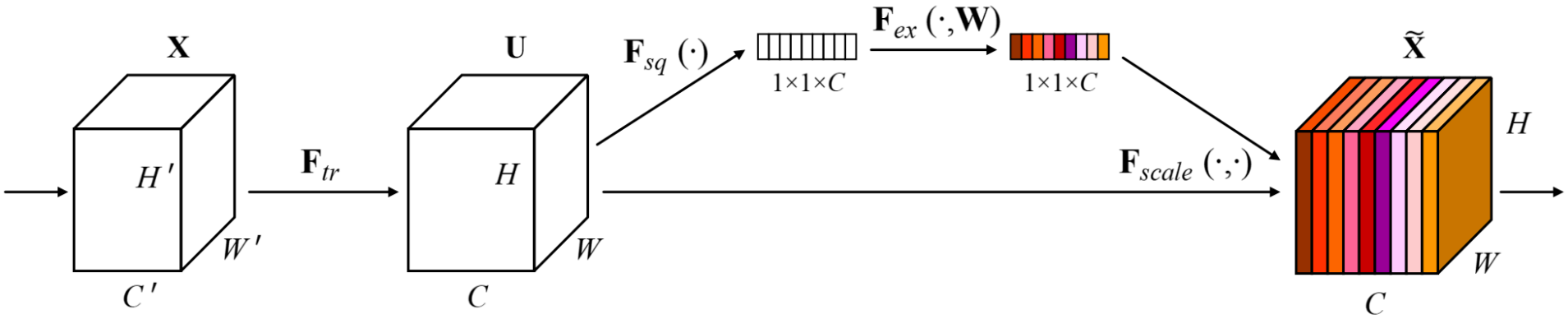
Squeeze-Excite Net (Hu et al., 2017)



For the standard neural net, the network weights each of its channels **equally** when creating the output feature maps.

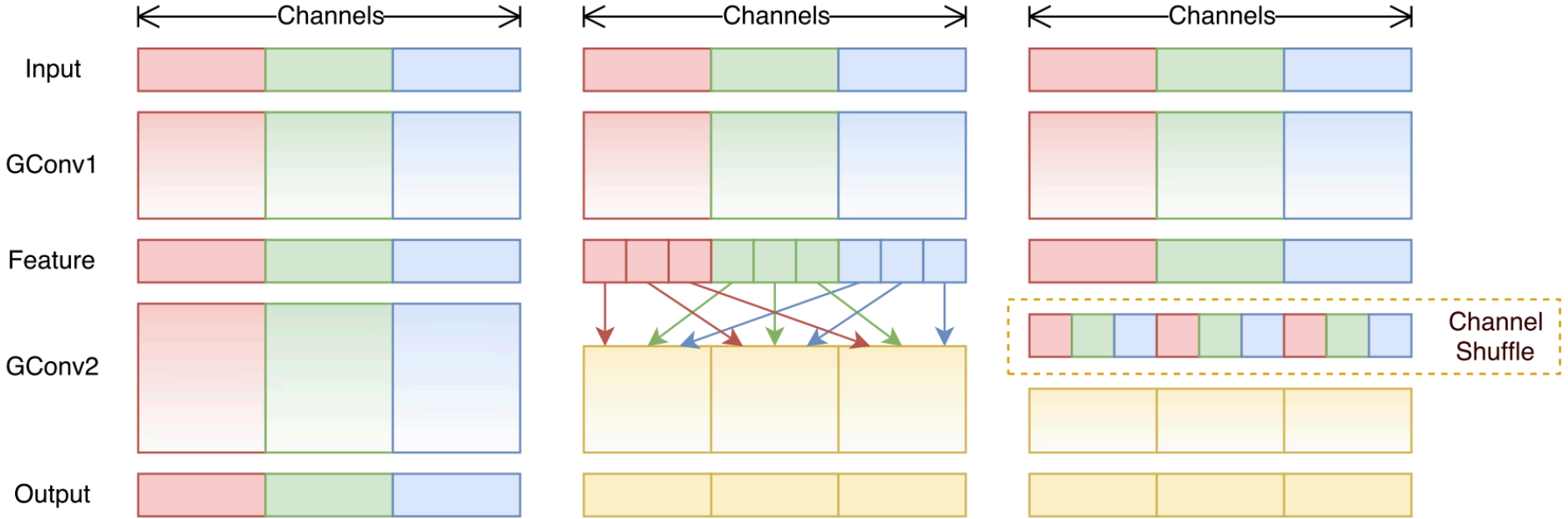
SENets are all about changing this by **adding a content aware mechanism** to weight each channel adaptively.

Squeeze-Excite Net (Hu et al., 2017)



- Learn global weighting function per channel
- Allows for fast information transfer between pixels in different locations of the image

ShuffleNet (Zhang et al., 2018)



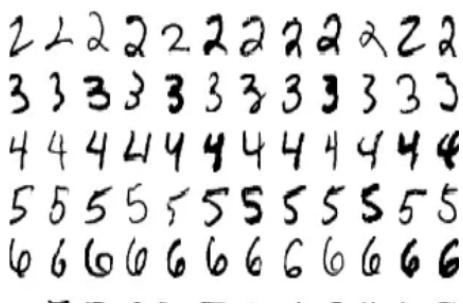
- ResNeXt breaks convolution into channels
- ShuffleNet mixes by grouping (very efficient for mobile)

aws


Fine Tuning

Labelling a Dataset is Expensive

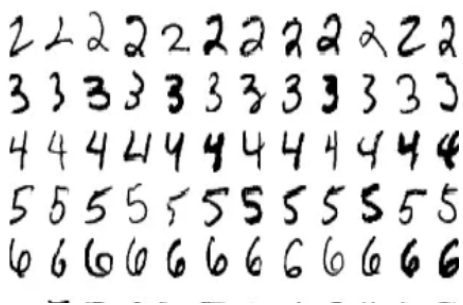
# examples	1.2M	50K	60K
# classes	1,000	100	10



My dataset

Labelling a Dataset is Expensive

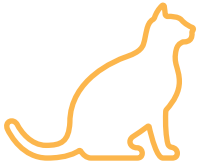
# examples	1.2M	50K	60K
# classes	1,000	100	10



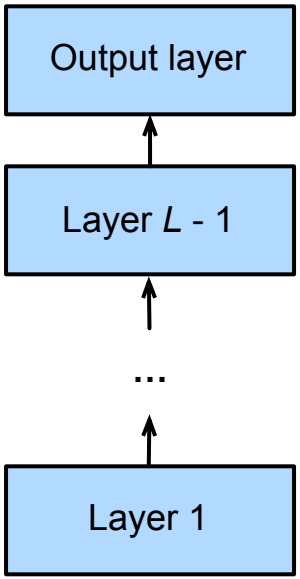
Can we reuse this?

My dataset

aws



Network Structure



Softmax classifier

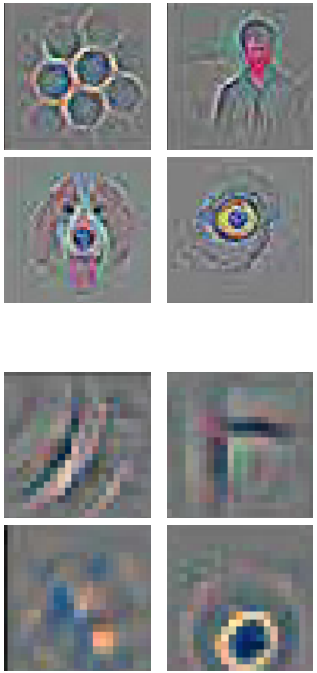
Feature extractor

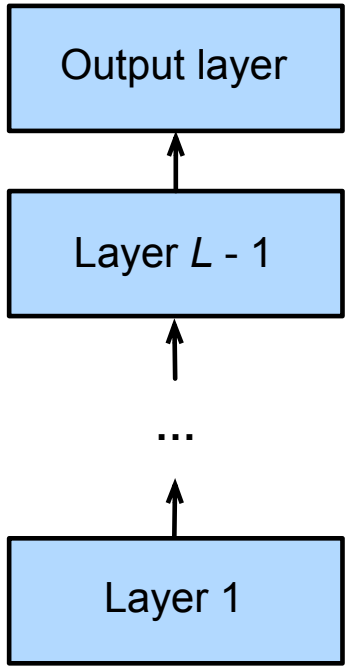
Two components in deep network

- **Feature extractor to map raw pixels into linearly separable features.**
- Linear classifier for decisions

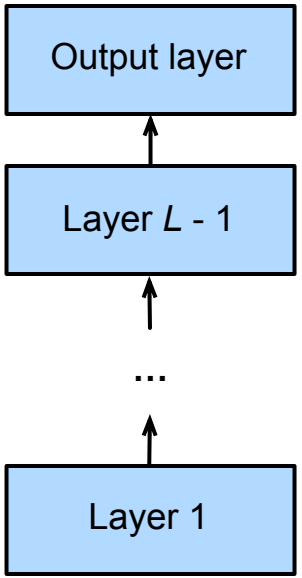
Fix Lower Layers

- Neural networks learn hierarchical feature representations
  - Low-level features are universal
  - High-level features are more related to objects in the dataset
- **Fix the bottom layer parameters during fine tuning**
  (useful for regularization)





Fine Tuning



Don’t use last layer since classification problem is different

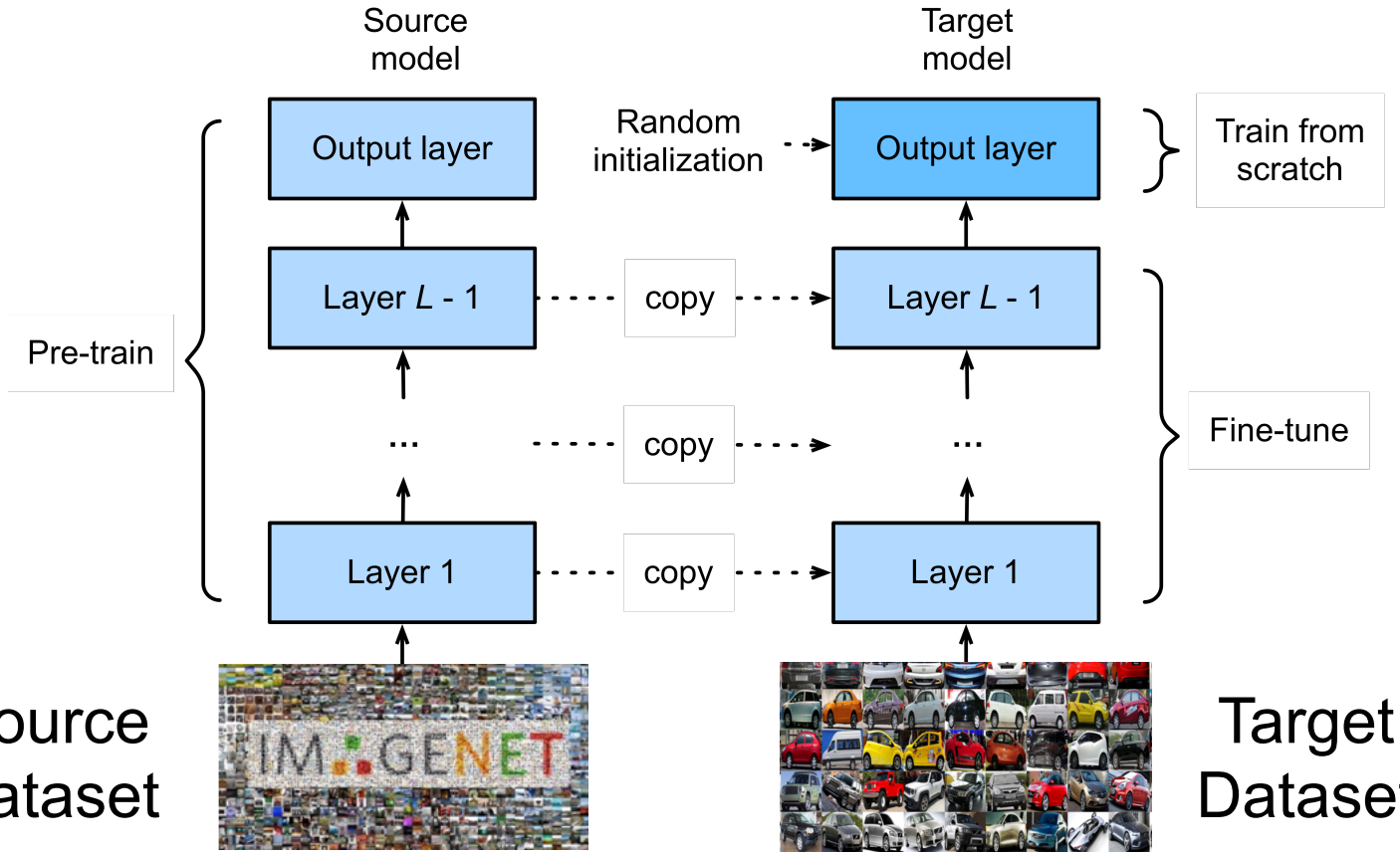
Likely good feature extractor for target

Source Dataset

Target Dataset

aws

Fine Turning - Weight Initialization



Fine-tuning Training Recipe

- Train on the target dataset as normal but with strong regularization
  - Small learning rate
  - Fewer epochs
- If source dataset is more complex than the target dataset, fine-tuning can lead to better models (source model is a good prior)

Fine-tuning Notebook