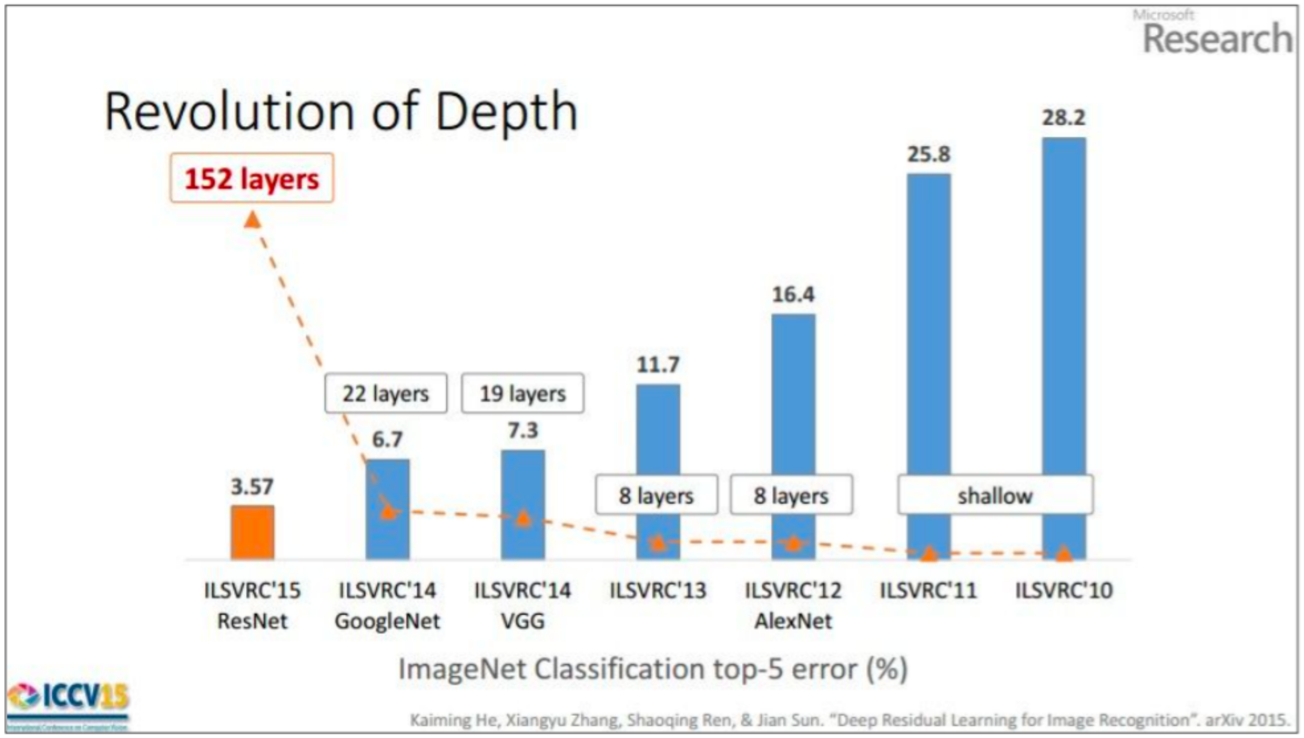
TTIC 31230, Fundamentals of Deep Learning

David McAllester, Winter 2020

Convolutional Neural Networks (CNNs)

Imagenet Classification

1000 kinds of objects.



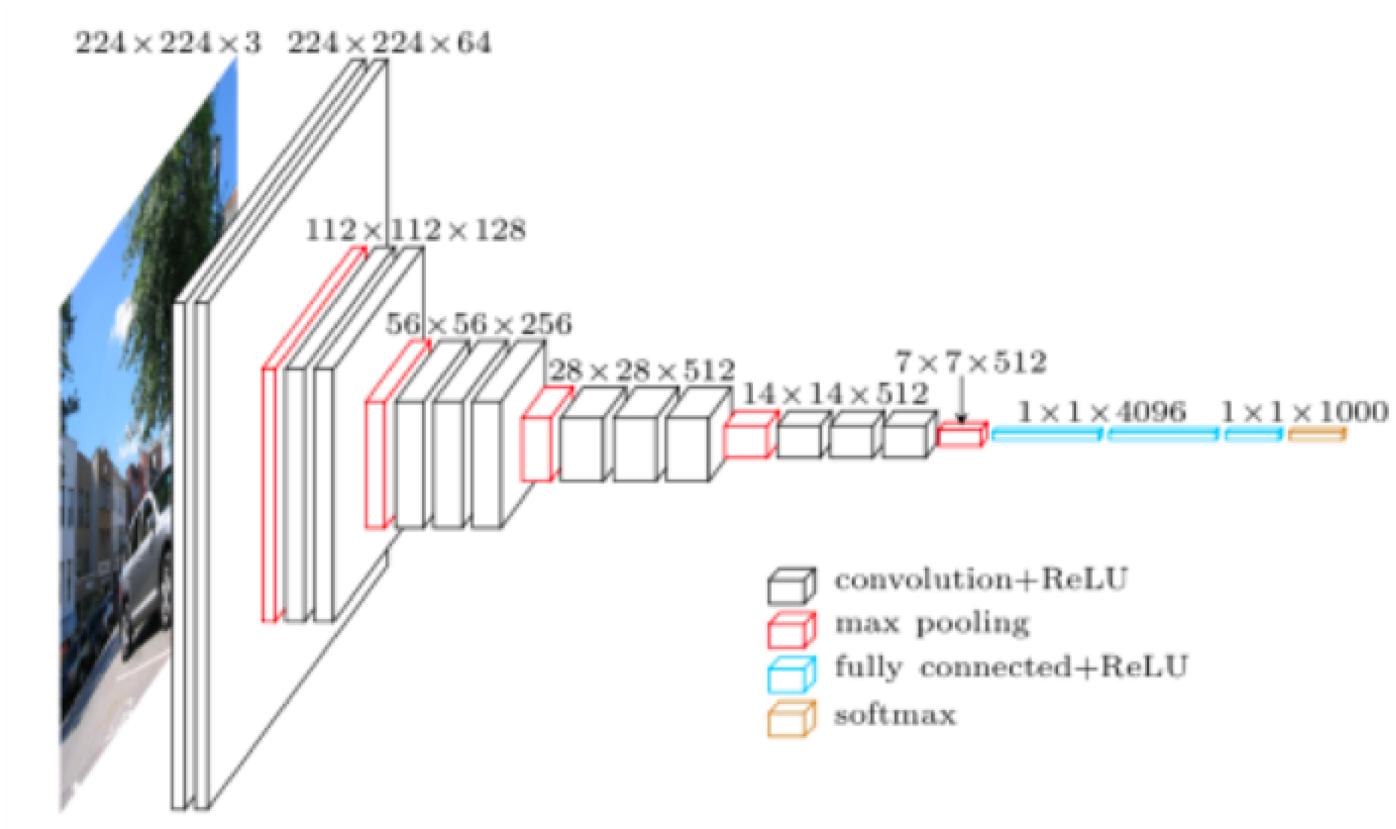
(slide from Kaiming He's recent presentation)

2016 is 3.0%, is 2017 2.25%

SOTA as of January 2020 is 1.3%

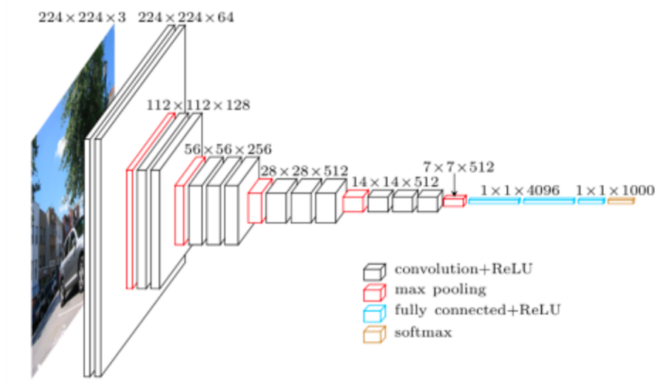
What is a CNN?

VGG, Zisserman, 2014



Davi Frossard

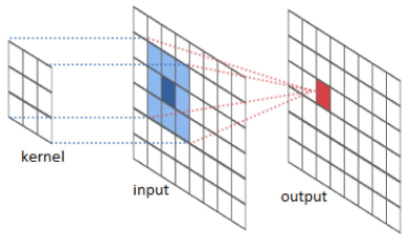
A Convolution Layer



Each box is a tensor $L_\ell[b, x, y, i]$

Each value $L_\ell[b, x, y, j]$ (for $\ell > 0$) is the output of a single linear threshold unit.

A Convolution Layer



$$W[\Delta x, \Delta y, i, j] \qquad L_\ell[b, x, y, i] \qquad L_{\ell+1}[b, x, y, j]$$

River Trail Documentation

$$L_{\ell+1}[b, x, y, j]$$

$$= \sigma\left(\left(\sum_{\Delta x, \Delta y, i} W[\Delta x, \Delta y, i, j]\; L_\ell[b, x + \Delta x, y + \Delta y, i]\right) - B[j]\right)$$

Many “Neurons” (Linear Threshold Units)

Each $L_{\ell+1}[b, x, y, j]$ is the output of a single linear threshold unit.

$$L_{\ell+1}[b, x, y, j]$$

$$= \sigma\left(\left(\sum_{\Delta x, \Delta y, i} W[\Delta x, \Delta y, i, j]\; L_\ell[b, x + \Delta x, y + \Delta y, i]\right) - B[j]\right)$$

2D CNN in PyTorch

conv2d(**input, weight, bias, stride, padding, dilation, groups**)

input tensor (minibatch,in-channels,iH,iW)

weight filters (out-channels, in-channels/groups,kH,kW)

bias tensor (out-channels) . Default: None

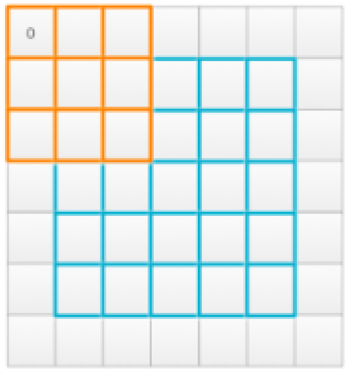
stride Single number or (sH, sW). Default: 1

padding Single number or (padH, padW). Default: 0

dilation Single number or (dH, dW). Default: 1

groups split input into groups. Default: 1

Padding



Jonathan Hui

If we pad the input with zeros then the input and output can have the same spatial dimensions.

Zero Padding in NumPy

In NumPy we can add a zero padding of width p to an image as follows:

```
padded = np.zeros(W + 2*p,  H + 2*p)

padded[p:W+p, p:H+p] = x
```

Padding

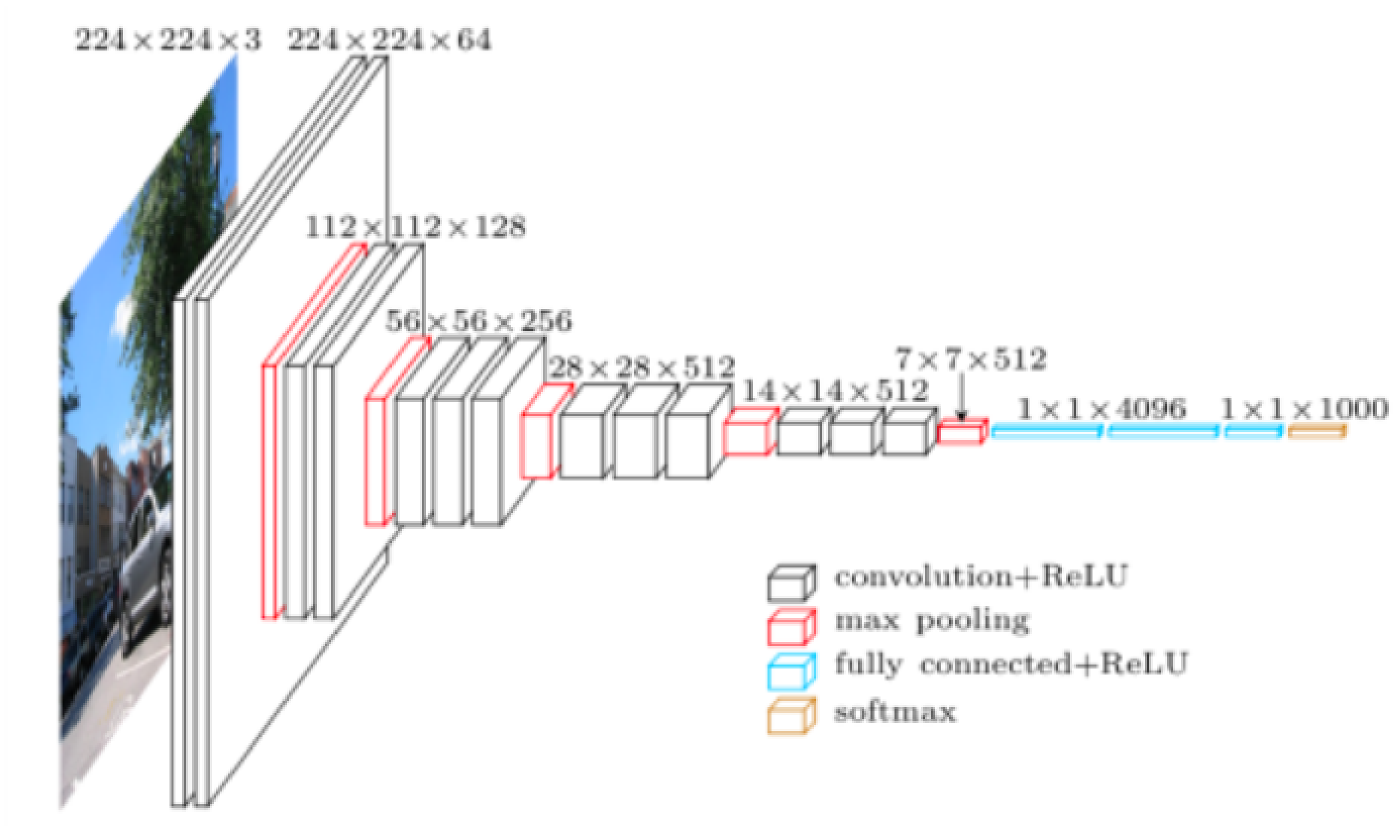
$$L'_\ell = \mathrm{Padd}(L_\ell,\ p)$$

$$L_{\ell+1}[b,x,y,j] =$$

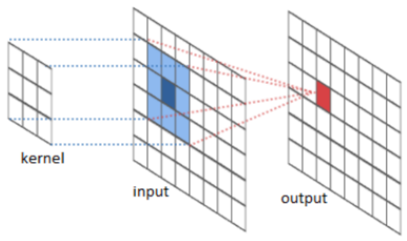
$$\sigma\left(\left(\sum_{\Delta x,\Delta y,i}\ W[\Delta x,\Delta y,i,j]\ L'_\ell[b,x+\Delta x,y+\Delta y,i]\right) - B[j]\right)$$

If the input is padded but the output is not padded then Δx and Δy are non-negative.

Reducing Spatial Dimention



Reducing Spatial Dimensions: Max Pooling



$$L_{\ell+1}[b,x,y,i] = \max_{\Delta x,\Delta y}\; L_\ell[b,\, s*x+\Delta x,\; s*y+\Delta y,\; i]$$

This is typically done with a stride greater than one so that the image dimension is reduced.

Reducing Spatial Dimensions: Strided Convolution

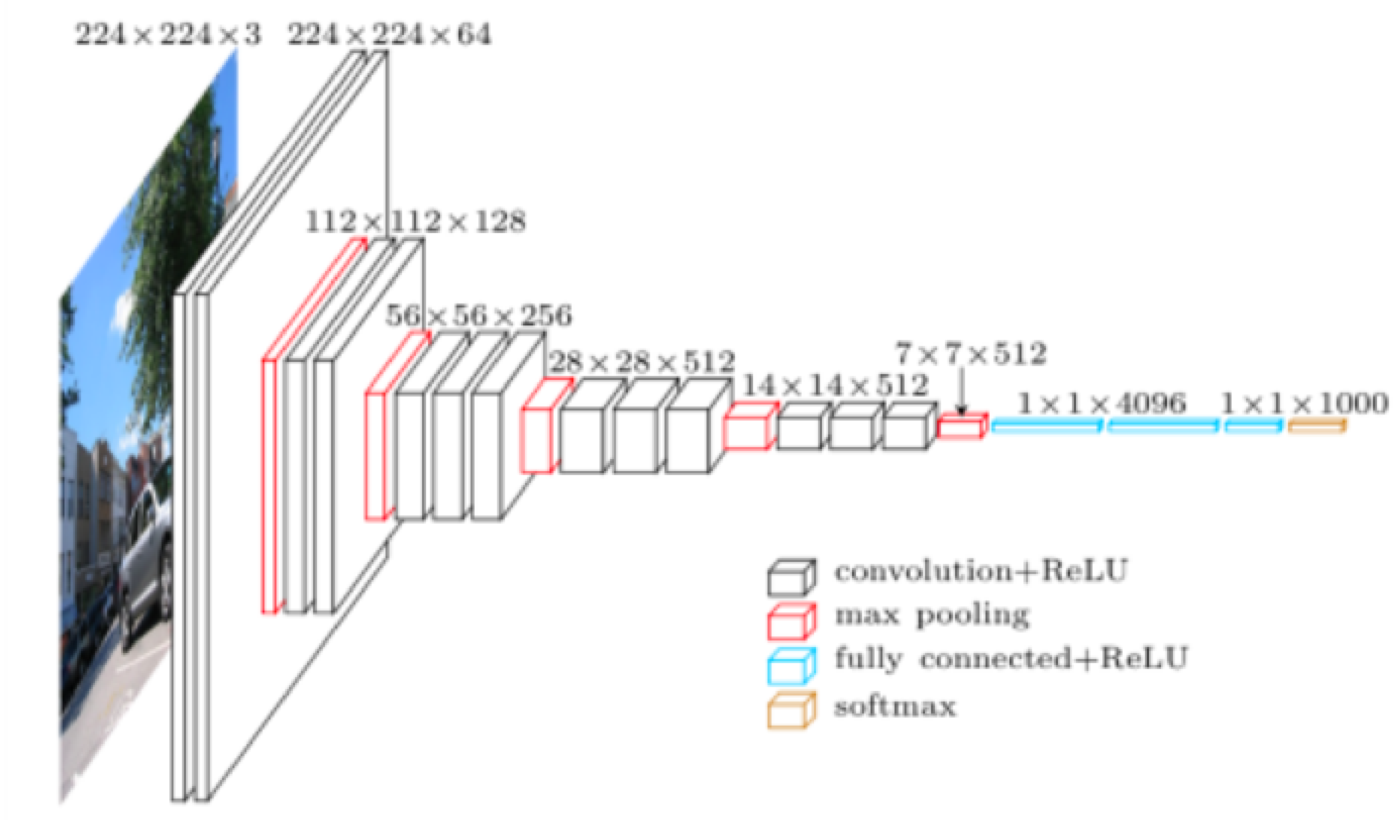
We can move the filter by a “stride” s for each spatial step.

$$L_{\ell+1}[b, x, y, j] =$$

$$\sigma\left(\left(\sum_{\Delta x, \Delta y, i} W[\Delta x, \Delta y, i, j] L_\ell[b, s * x + \Delta x, s * y + \Delta y, i]\right) - B[j]\right)$$

For strides greater than 1 the spatial dimention is reduced.

Fully Connected (FC) Layers



Fully Connected (FC) Layers

We reshape $L_\ell[b,x,y,i]$ to $L_\ell[b,i']$ and then

$$L_{\ell+1}[b,j] = \sigma\left(\left(\sum_{i'} W[j,i']\; L_\ell[b,i']\right) - B[j]\right)$$

Image to Column (Im2C)

Reduce convolution to matrix multiplication

more space but faster.

$$\tilde{L}_{\ell+1}[b,x,y,j]$$

$$= \left(\sum_{\Delta x,\Delta y,i} W[\Delta x,\Delta y,i,j] * L_\ell[b,x+\Delta x,\ y+\Delta y,\ i]\right) + B[j]$$

We make a bigger tensor $\tilde{L}$ with two additional indeces.

$$\tilde{L}_\ell[b,x,y,\Delta x,\Delta y,i] = L_\ell[b,x+\Delta x,y+\Delta y,i]$$

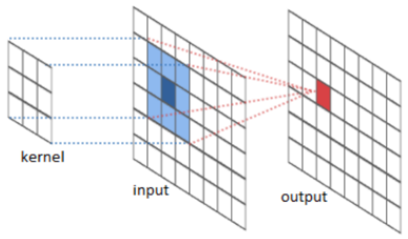
Image to Column (Im2C)

$$
\begin{aligned}
&\tilde{L}_{\ell+1}[b,x,y,j] \\
&= \left(\sum_{\Delta x,\Delta y,i} W[\Delta x,\Delta y,i,j] * L_\ell[b,x+\Delta x,\ y+\Delta y,\ i]\right) + B[j] \\
&= \left(\sum_{\Delta x,\Delta y,i} \tilde{L}_\ell[b,x,y,\Delta x,\Delta y,i] * W[\Delta x,\Delta y,i,j]\right) + B[j] \\
&= \left(\sum_{(\Delta x,\Delta y,i)} \tilde{L}_\ell[(b,x,y),(\Delta x,\Delta y,i)] * W[(\Delta x,\Delta y,i),j]\right) + B[j]
\end{aligned}
$$

Dilation

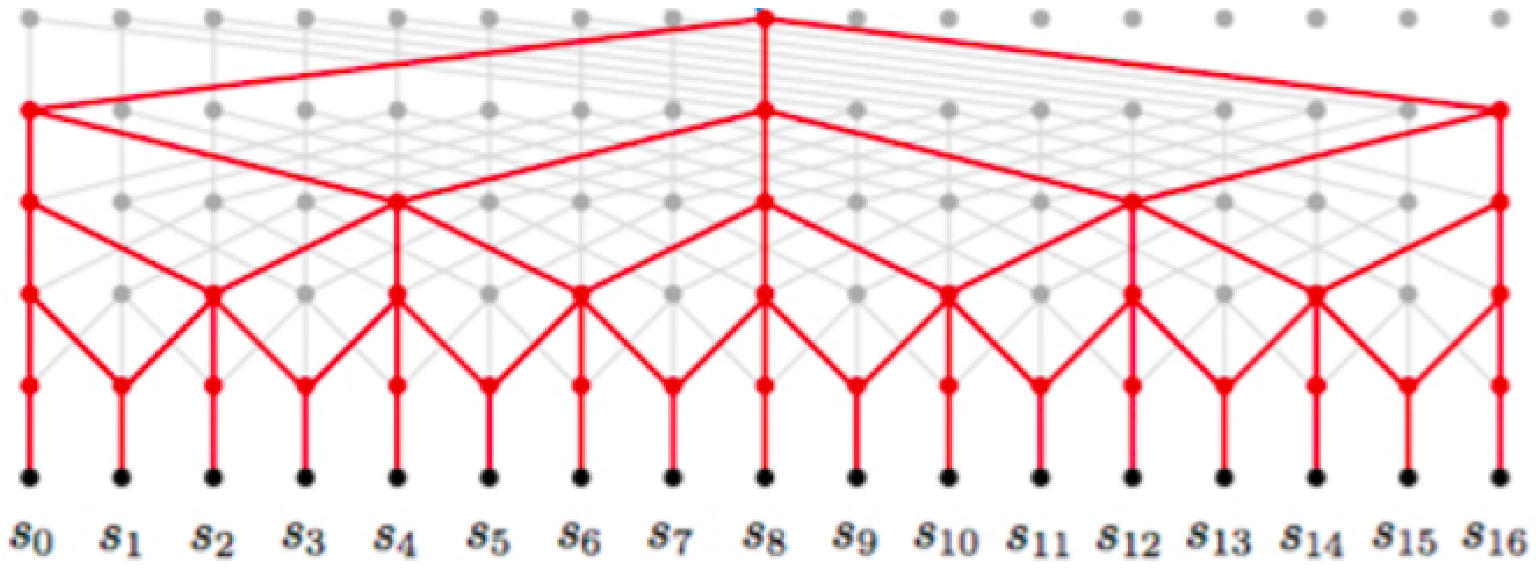
A CNN for image classification typically reduces an $N \times N$ image to a single feature vector.

Dilation is a trick for treating the whole CNN as a "filter" that can be passed over an $M \times M$ image with $M > N$.



An output tensor with full spatial dimension can be useful in, for example, image segmentation.

Dilation


s_0 s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 s_9 s_{10} s_{11} s_{12} s_{13} s_{14} s_{15} s_{16}

This is called a “fully convolutional” CNN.

Dilation

To implement a fully convolutional CNN we can "dilate" the filters by a dilation parameter d.

$$\tilde{L}_{\ell+1}[b,x,y,j] = W[\Delta x,\Delta y,i,j]L_\ell[b,x+d*\Delta x,y+d*\Delta y,i]+B[j]$$

Hypercolumns

An alternative to dilation and fully convolutional networks is hypercolumns.

To define hypercolumns we first introduce some notation.

I will use capital letters for the index dimension.

A capital letter used as an index leaves the value unspecified.

For example $L[b, x, y, J]$ denotes a vector of dimension J for the given indeces b, x and y.

Vector Concatenation

We will write

$$L[b, x, y, J_1 + J_2] = L_1[b, x, y, J_1]\ ; L(b, x, y, J_2]$$

To mean that the vector $L(b, x, y, J_1 + J_2]$ is the concatenation of the vectors $L_1[b, x, y, J_1]$ and $L_2[b, x, y, J_2]$.

Hypercolumns

$$L\left[b, x, y, \sum_{\ell} J_\ell\right]$$

$$\begin{aligned}
= \; & L_0\,[b, x, y, J_0] \\
& \vdots \\
& ;\, L_\ell \left[b, \lfloor x \left(\frac{X_\ell}{X_1}\right) \rfloor, \lfloor y \left(\frac{Y_\ell}{Y_0}\right) \rfloor, J_\ell\right] \\
& \vdots \\
& ;\, L_{\mathcal{L}-1}[b, J_{\mathcal{L}-1}]
\end{aligned}$$

Grouping

The input features and the output features are each divided into G groups.

$$L_{\ell+1}[b,x,y,J] = L^0_{\ell+1}[b,x,y,J/G];\cdots;L^{G-1}_{\ell+1}[b,x,y,J/G]$$

Each tensor $L^g_{\ell+1}[b,x,y,J/G]$ is computed by convolution with a filter

$$W^g[b,x,y,I/G,J/G].$$

This uses a factor of G fewer weights.

2D CNN in PyTorch

conv2d(**input, weight, bias, stride, padding, dilation, groups**)

input tensor (minibatch,in-channels,iH,iW)

weight filters (out-channels, in-channels/groups,kH,kW)

bias tensor (out-channels) . Default: None

stride Single number or (sH, sW). Default: 1

padding Single number or (padH, padW). Default: 0

dilation Single number or (dH, dW). Default: 1

groups split input into groups. Default: 1

Modern Trends

Modern Convolutions use 3X3 filters. This is faster and has fewer parameters. Expressive power is preserved by increasing depth with many stride 1 layers.

Max pooling and dilation seem to have disappeared.

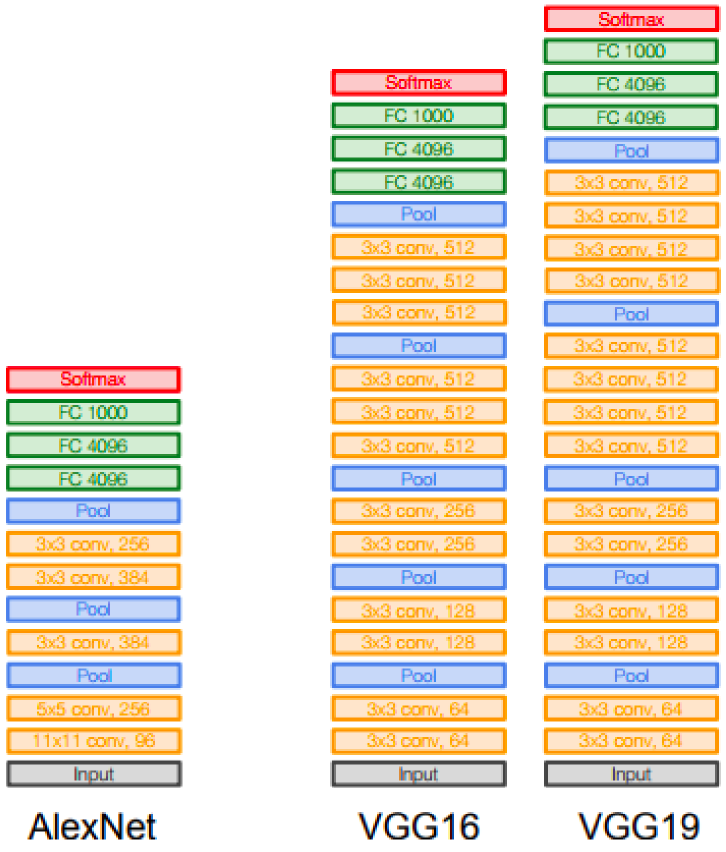
ResNet and resnet-like architectures are now dominant.

Alexnet

Given Input$[227, 227, 3]$

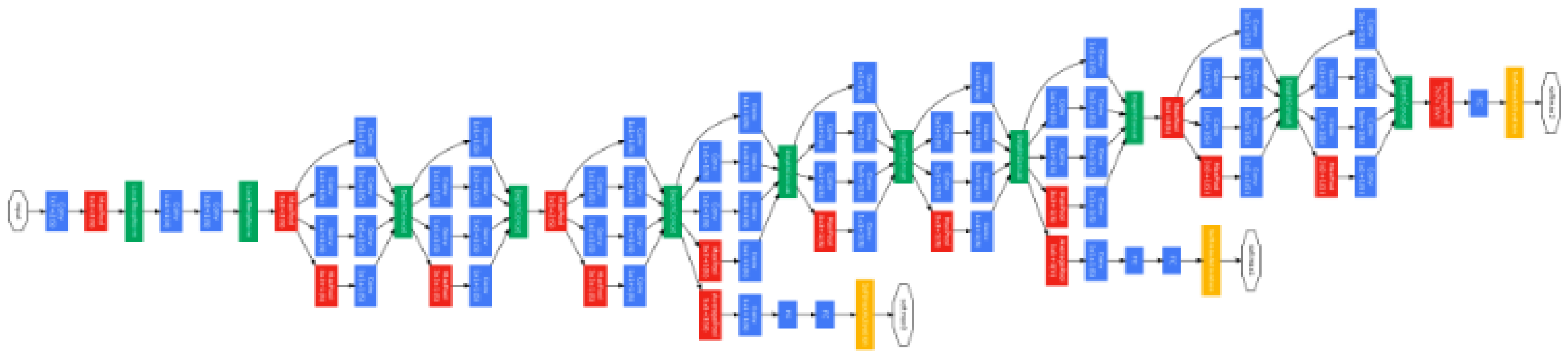
$$\begin{aligned}
L_1[55 \times 55 \times 96] &= \mathrm{ReLU}(\mathrm{CONV}(\mathrm{Input}, \Phi_1, \text{width } 11, \text{pad } 0, \text{stride } 4)) \\
L_2[27 \times 27 \times 96] &= \mathrm{MaxPool}(L_1, \text{width } 3, \text{stride } 2)) \\
L_3[27 \times 27 \times 256] &= \mathrm{ReLU}(\mathrm{CONV}(L_2, \Phi_3, \text{width } 5, \text{pad } 2, \text{stride } 1)) \\
L_4[13 \times 13 \times 256] &= \mathrm{MaxPool}(L_3, \text{width } 3, \text{stride } 2)) \\
L_5[13 \times 13 \times 384] &= \mathrm{ReLU}(\mathrm{CONV}(L_4, \Phi_5, \text{width } 3, \text{pad } 1, \text{stride } 1)) \\
L_6[13 \times 13 \times 384] &= \mathrm{ReLU}(\mathrm{CONV}(L_5, \Phi_6, \text{width } 3, \text{pad } 1, \text{stride } 1)) \\
L_7[13 \times 13 \times 256] &= \mathrm{ReLU}(\mathrm{CONV}(L_6, \Phi_7, \text{width } 3, \text{pad } 1, \text{stride } 1)) \\
L_8[6 \times 6 \times 256] &= \mathrm{MaxPool}(L_7, \text{width } 3, \text{stride } 2)) \\
L_9[4096] &= \mathrm{ReLU}(\mathrm{FC}(L_8, \Phi_9)) \\
L_{10}[4096] &= \mathrm{ReLU}(\mathrm{FC}(L_9, \Phi_{10})) \\
s[1000] &= \mathrm{ReLU}(\mathrm{FC}(L_{10}, \Phi_s)) \ \text{ class scores}
\end{aligned}$$

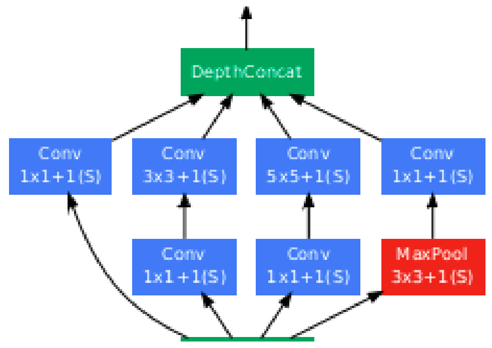
VGG



Stanford CS231

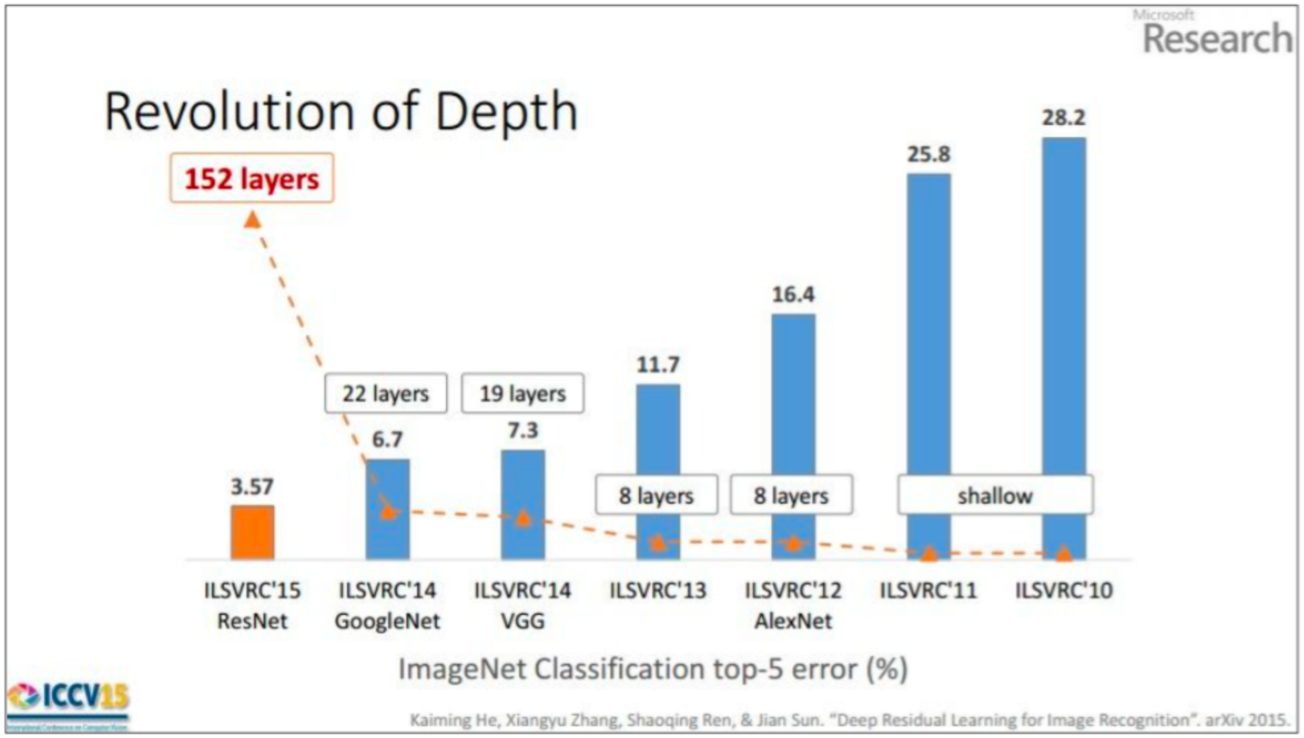
Inception, Google, 2014





Imagenet Classification

1000 kinds of objects.



(slide from Kaiming He's recent presentation)

2016 is 3.0%, is 2017 2.25%

SOTA as of January 2020 is 1.3%

Trainability

The Expressive Power of DNNs

Linear Threshold units generalize logical operations.

Consider Boolean Values P, Q — numbers that are either close to 0 or close to 1.

$$P \wedge Q \approx \sigma(100 * P + 100 * Q - 150)$$

$$P \vee Q \approx \sigma(100 * P + 100 * Q - 50)$$

$$\neg P \approx \sigma(100 * (1 - P) - 50)$$

DNNs generalize digital circuits.

DNNs are Expressive and Trainable

Universality Assumption: DNNs are universally expressive (can model any function) and trainable (the desired function can be found by SGD).

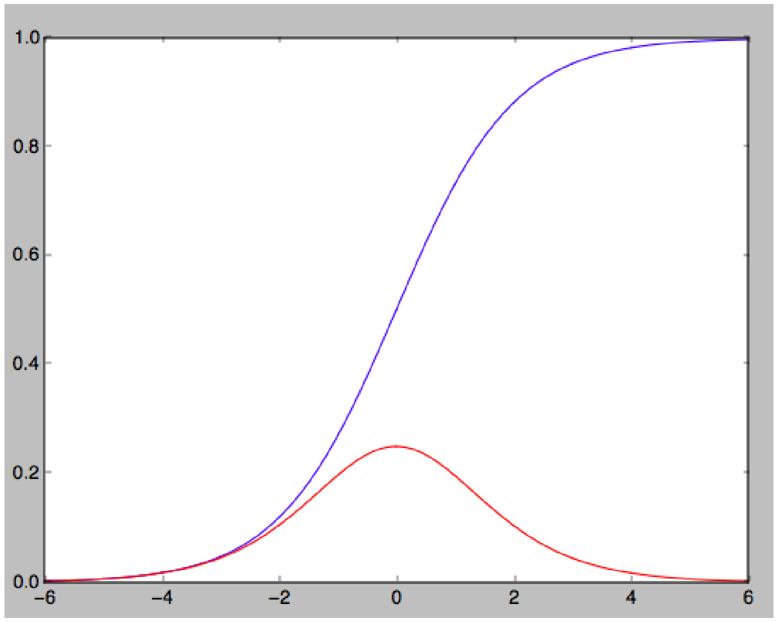
Universal trainability is clearly false but can still guide architecture design.

Equally expressive architectures differ in trainability. This lecture is on trainability.

Vanishing and Exploding Gradients

Activation Function Saturation

Consider the sigmoid activation function $1/(1+e^{-x})$.

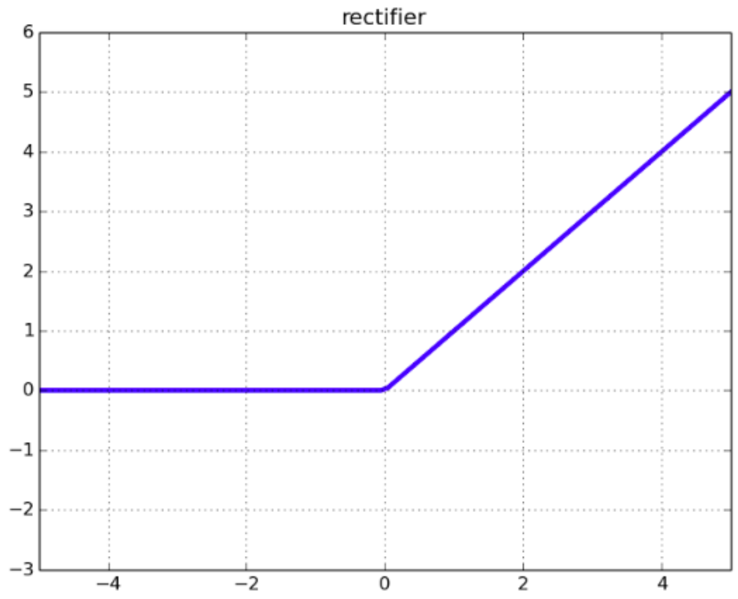


The gradient of this function is quite small for $|x| > 4$.

In deep networks backpropagation can go through many sigmoids and the gradient can "vanish"

Activation Function Saturation

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



The Relu does not saturate at positive inputs (good) but is completely saturated at negative inputs (bad).
Alternate variations of Relu still have small gradients at negative inputs.

Repeated Multiplication by Network Weights

Consider a deep CNN.

$$L_{i+1} = \mathrm{Relu}(\mathrm{Conv}(\Phi_i, L_i))$$

For i large, L_i has been multiplied by many weights.

If the weights are small then the neuron values, and hence the weight gradients, decrease exponentially with depth. **Vanishing Gradients.**

If the weights are large, and the activation functions do not saturate, then the neuron values, and hence the weight gradients, increase exponentially with depth. **Exploding Gradients.**

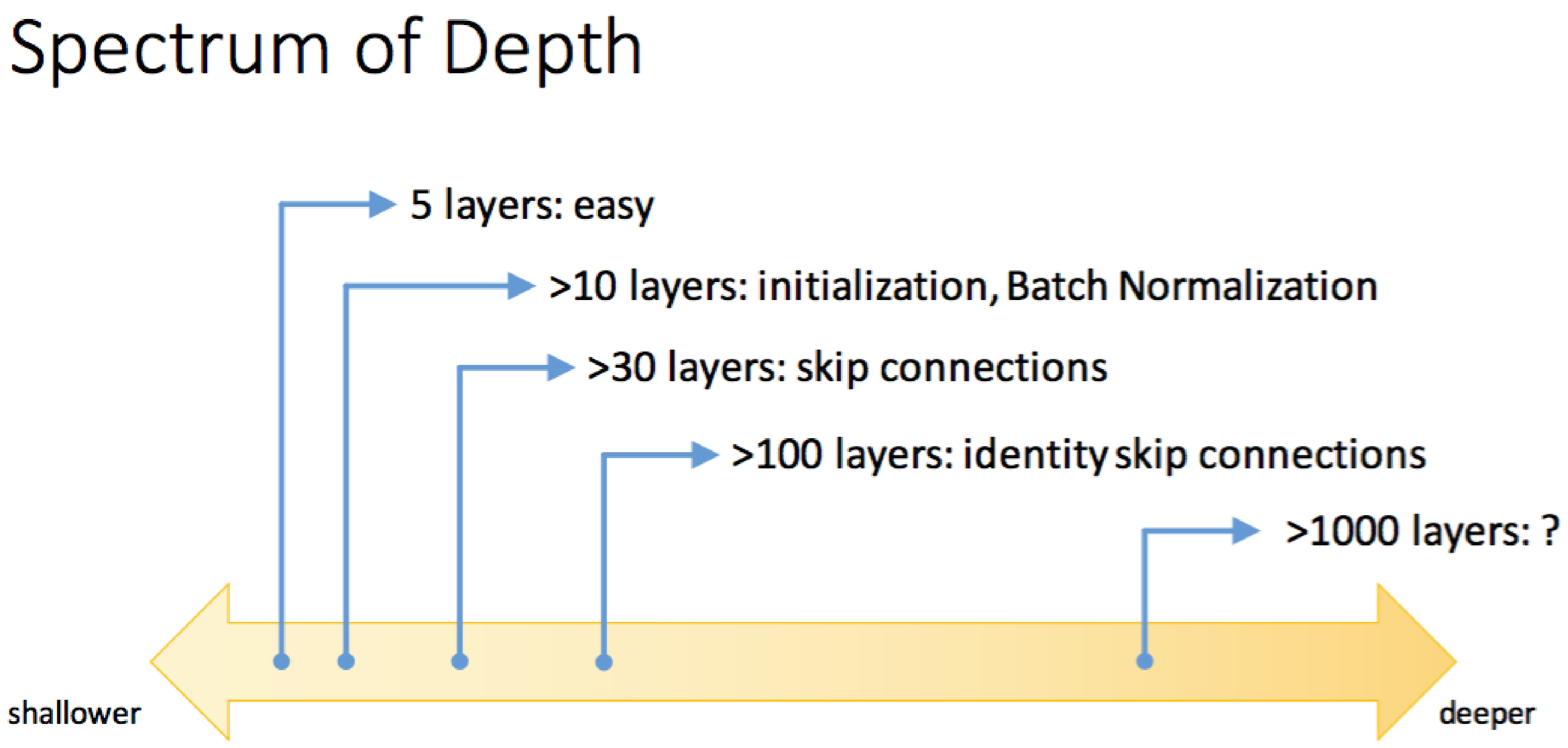
Methods for Maintaining Gradients

Initialization

Batch Normalization

ResNet

Methods for Maintaining Gradients


Spectrum of Depth
5 layers: easy
>10 layers: initialization, Batch Normalization
>30 layers: skip connections
>100 layers: identity skip connections
>1000 layers: ?
shallower
deeper

Kaiming He

Initialization

Xavier Initialization

Initialize a weight matrix (or tensor) to preserve zero-mean unit variance distributions.

If we assume x_i has zero mean and unit variance then we want

$$y_j = \sum_{j=0}^{N-1} x_i w_{i,j}$$

to have zero mean and unit variance.

Xavier initialization randomly draws $w_{i,j}$ from a uniform distribution on the interval $\left(-\sqrt{\frac{3}{N}},\ \sqrt{\frac{3}{N}}\right)$.

Assuming independence this gives zero mean and unit variance for y_j.

Batch Normalization

Normalization

Given a tensor $x[b,j]$ we define $\tilde{x}[b,j]$ as follows.

$$\hat{\mu}[j] = \frac{1}{B}\sum_b \; x[b,j]$$

$$\hat{\sigma}[j] = \sqrt{\frac{1}{B-1}\sum_b (x[b,j]-\hat{\mu}[j])^2}$$

$$\tilde{x}[b,j] = \frac{x[b,j]-\hat{\mu}[j]}{\hat{\sigma}[j]}$$

At test time a single fixed estimate of $\mu[j]$ and $\sigma[j]$ is used.

Spatial Batch Normalization

For CNNs we convert a tensor $x[b,x,y,j]$ to $\tilde{x}[b,x,y,j]$ as follows.

$$\hat{\mu}[j] = \frac{1}{BXY}\sum_{b,x,y} x[b,x,y,j]$$

$$\hat{\sigma}[j] = \sqrt{\frac{1}{BXY-1}\sum_{b,x,y}(x[b,x,y,j]-\hat{\mu}[j])^2}$$

$$\tilde{x}[b,x,y,j] = \frac{x[b,x,y,j]-\hat{\mu}[j]}{\hat{\sigma}[j]}$$

Adding an Affine Transformation

$$\breve{x}[b,x,y,j] = \gamma[j]\tilde{x}[b,x,y,j] + \beta[j]$$

Here $\gamma[j]$ and $\beta[j]$ are parameters of the batch normalization.

This allows the batch normlization to learn an arbitrary affine transformation (offset and scaling).

It can even undo the normaliztion.

Batch Normalization

Batch Normalization appears to be generally useful in CNNs but is not always used.

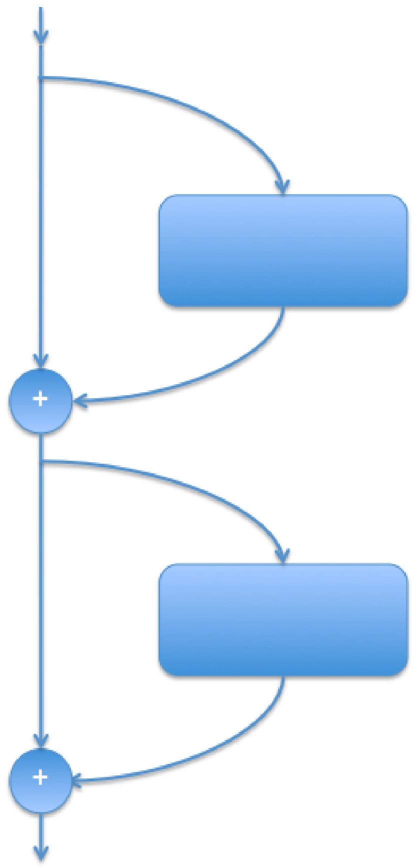
Not so successful in RNNs.

It is typically used just prior to a nonlinear activation function.

It is intuitively justified in terms of “internal covariate shift”: as the inputs to a layer change the zero mean unit variance property underlying Xavier initialization are maintained.

ResNet

Deep Residual Networks (ResNets) by Kaiming He 2015



A "skip connection" is adjusted by a "residual correction"

The skip connections connects input to output directly and hence preserves gradients.

ResNets were introduced in late 2015 (Kaiming He et al.) and revolutionized computer vision.

Simple Residual Skip Connections in CNNs (stride 1)

$$R_{\ell+1}[B,X,Y,J] \;=\; \mathrm{Conv}(W_{\ell+1}[X,Y,J,J], B_{\ell+1}[J], L_{\ell}[B,X,Y,J])$$

for b,x,y,j

$$L_{\ell+1}[b,x,y,j] \;=\; L_{\ell}[b,x,y,j] + R_{\ell+1}[b,x,y,j]$$

I will use capital letter indices to denote entire tensors and lower case letters for particular indeces.

Simple Residual Skip Connections in CNNs (stride 1)

$$R_{\ell+1}[B,X,Y,J] = \mathrm{Conv}(W_{\ell+1}[X,Y,J,J], B_{\ell+1}[J], L_\ell[B,X,Y,J])$$

```
for $b, x, y, j$
    $L_{\ell+1}[b,x,y,j] = L_\ell[b,x,y,j] + R_{\ell+1}[b,x,y,j]$
```

Note that in the above equations $L_\ell[B,X,Y,J]$ and $R_{\ell+1}[B,X,Y,J]$ are the same shape.

In the actual ResNet $R_{\ell+1}$ is computed by two or three convolution layers.

Handling Spacial Reduction

Consider $L_\ell[B, X_\ell, Y_\ell, J_\ell]$ and $R_{\ell+1}[B, X_{\ell+1}, Y_{\ell+1}, J_{\ell+1}]$

$$\begin{aligned} X_{\ell+1} &= X_\ell/s \\ Y_{\ell+1} &= Y_\ell/s \\ J_{\ell+1} &\geq J_\ell \end{aligned}$$

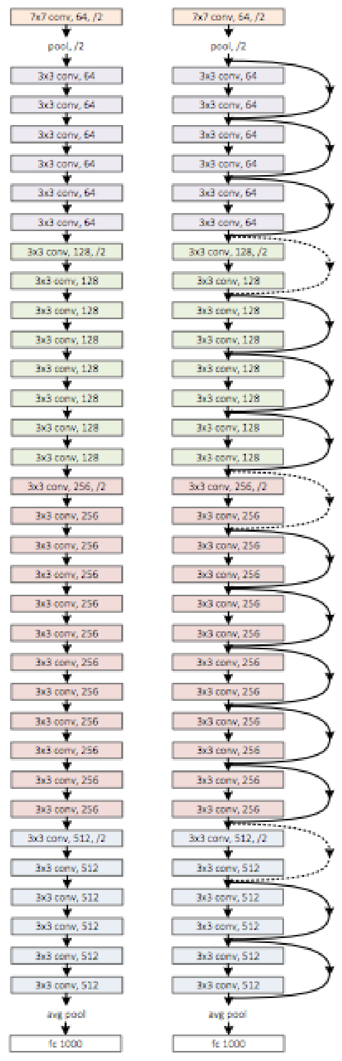
In this case we construct $\tilde{L}_\ell[B, X_{\ell+1}, Y_{\ell+1}, J_{\ell+1}]$

$$\text{for } b, x, y, j \quad \tilde{L}_\ell[b, x, y, j] = \begin{cases} L_\ell[b, s*x, s*y, j] & \text{for } j < J_\ell \\ 0 & \text{otherwise} \end{cases}$$

$$\begin{aligned} L_{\ell+1}[B, X_{\ell+1}, Y_{\ell+1}, J_{\ell+1}] = {} & \tilde{L}_\ell[B, X_{\ell+1}, Y_{\ell+1}, J_{\ell+1}] \\ & + R_{\ell+1}[B, X_{\ell+1}, Y_{\ell+1}, J_{\ell+1}] \end{aligned}$$

ResNet32

plain net



ResNet

er)

[Kaiming He]

Deeper Versions use Bottleneck Residual Paths

We reduce the number of features to $K < J$ before doing the convolution.

$$\begin{aligned}
U[B,X,Y,K] &= \mathrm{Conv}'(\Phi^A_{\ell+1}[1,1,J,K], L_\ell[B,X,Y,J]) \\
V[B,X,Y,K] &= \mathrm{Conv}'(\Phi^B_{\ell+1}[3,3,K,K], U[B,X,Y,K]) \\
R[B,X,Y,J] &= \mathrm{Conv}'(\Phi^R_{\ell+1}[1,1,K,J], V[B,X,Y,K]) \\
L_{\ell+1} &= L_\ell + R
\end{aligned}$$

Here CONV$'$ may include batch normalization and/or an activation function.

A General Residual Connection

$$y = \tilde{x} + R(x)$$

Where $\tilde{x}$ is either x or a version of x adjusted to match the shape of $R(x)$.

Models for Image Classification in PyTorch

- AlexNet
- VGG
- ResNet
- SqueezeNet
- DenseNet
- Inception v3
- GoogLeNet
- ShuffleNet v2
- MobileNet v2
- ResNeXt
- Wide ResNet
- MNASNet

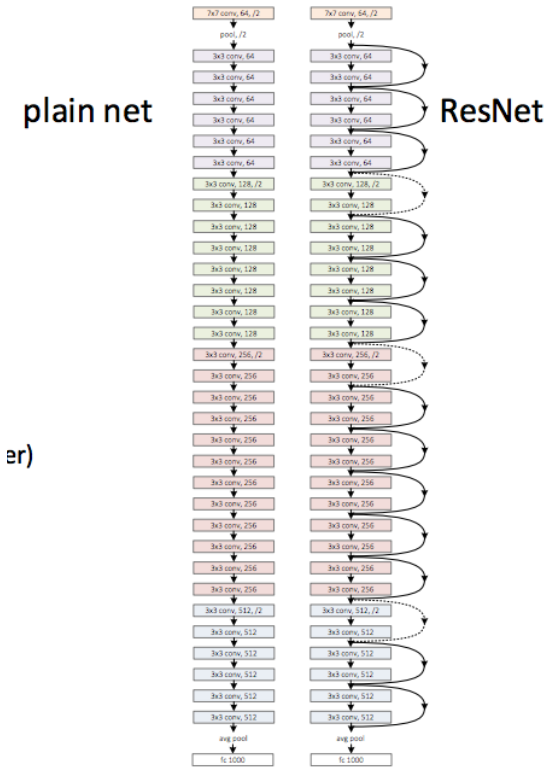
DenseNet

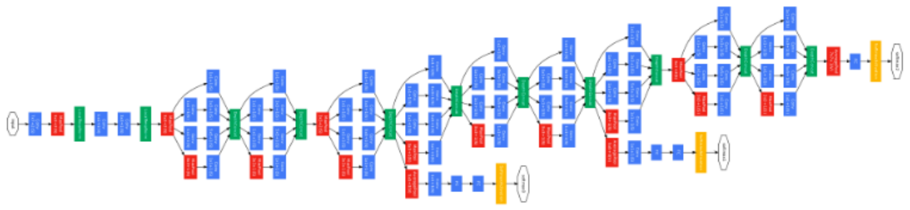
We compute a residual $R[b, x, y, J_R]$ and then simply concatenate the residual onto the previous layer.

$$\text{for } b, x, y \;\; L_{\ell+1}[b, x, y, J_\ell + J_R] = L_\ell[b, x, y, J_\ell]; R[b, x, y, J_R]$$

The number J_R of new features can be can be relatively small.

ResNet Simplicity





[Kaiming He]

ResNet Power

ResNet gives powerful image classification.

ResNet is used in folding proteins.

ResNet is the netowwork used in AlphaZero for Go, Chess and Shogi.

This supports Hinton's claim that general purpose "prior-free" learning is possible.

END