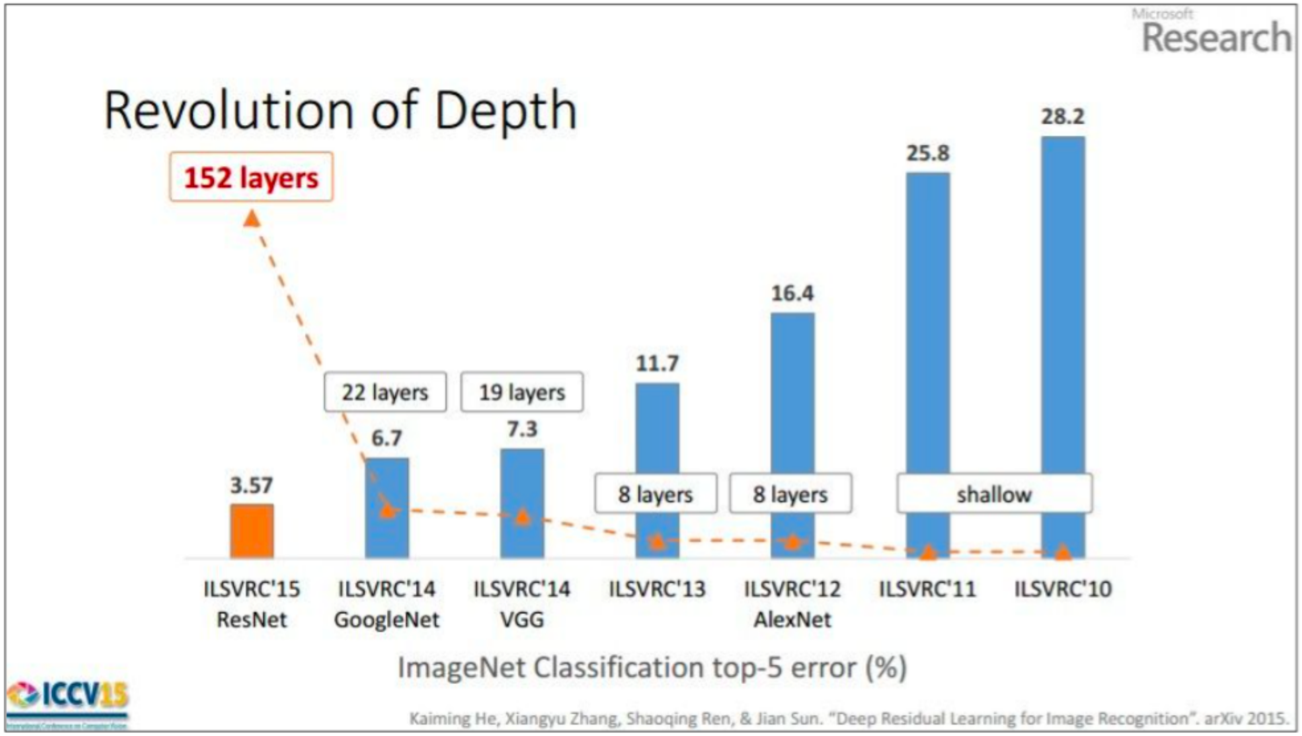
TTIC 31230, Fundamentals of Deep Learning

David McAllester, Winter 2020

Convolutional Neural Networks (CNNs)

Imagenet Classification

1000 kinds of objects.



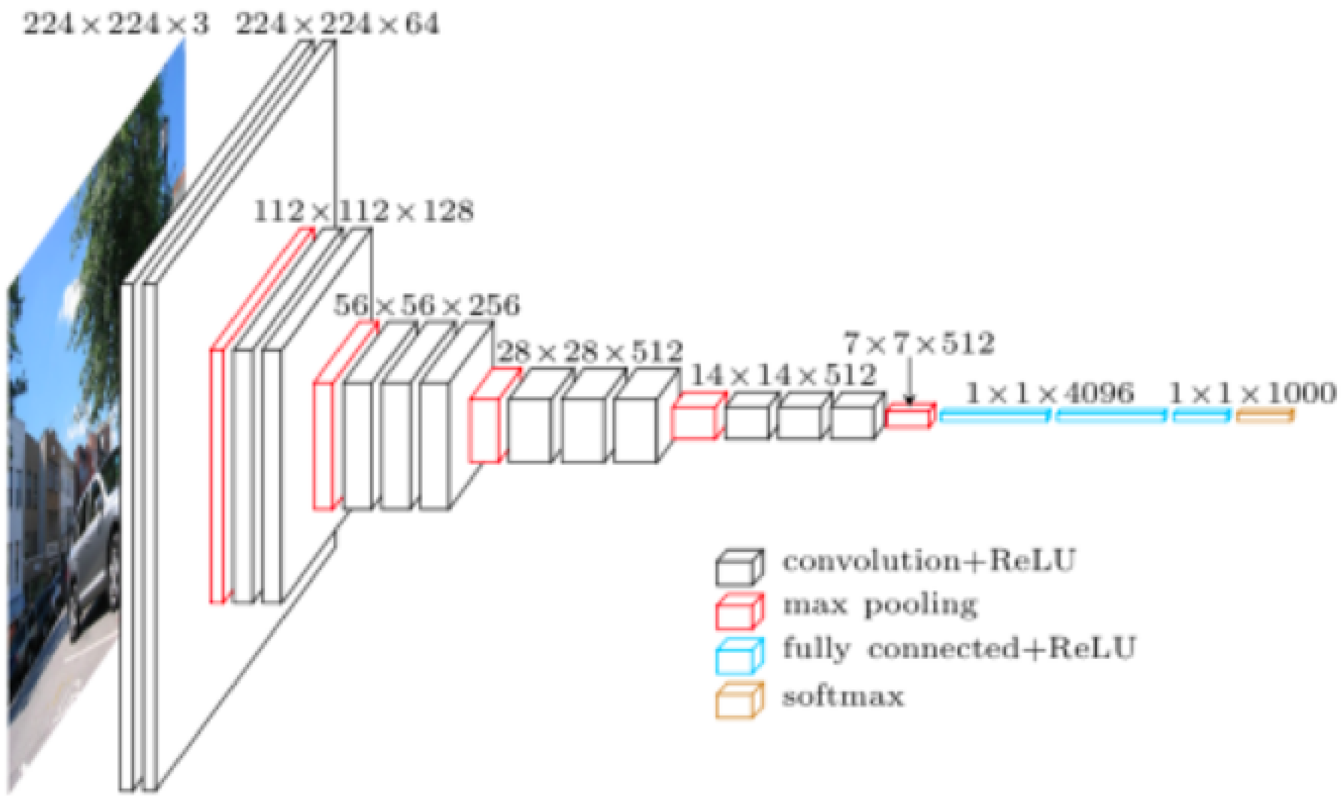
(slide from Kaiming He's recent presentation)

2016 is 3.0%, is 2017 2.25%

SOTA as of January 2020 is 1.3%

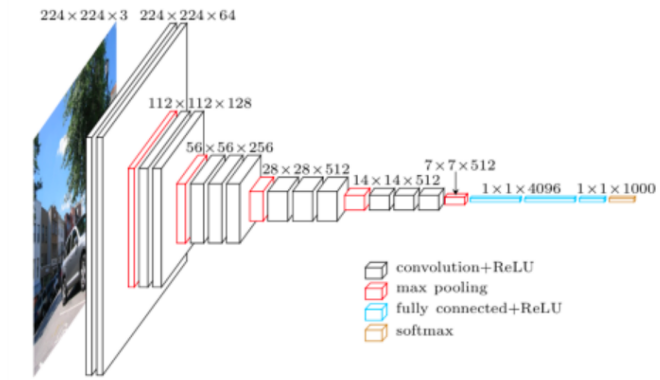
What is a CNN?

VGG, Zisserman, 2014



Davi Frossard

A Convolution Layer

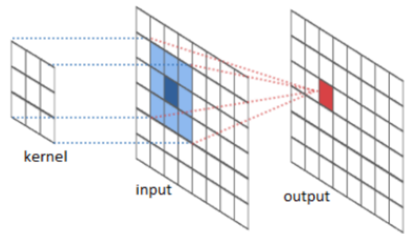


Each box is a tensor $L_\ell[b, x, y, i]$

batches, special index,spacial index and feature index.

For a convolution layer, each $L_{\ell+1}[b, x, y, j]$ is the output of a single linear threshold unit computed from $L_\ell[b, x, y, i]$.

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

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

we are computing the jth feature for the output

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

moving to the neighbor of x and y

also summing over the feature index of the previous layer

Many "Neurons" (Linear Threshold Units)

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

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

2D CNN in PyTorch

each of this is object

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

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

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

bias tensor (out-channels) . Default: None

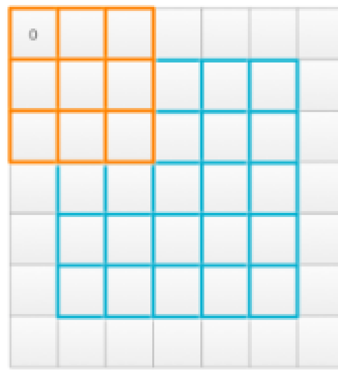
stride Single number or (sH, sW). Default: 1

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
```

the thing we are padding

Padding

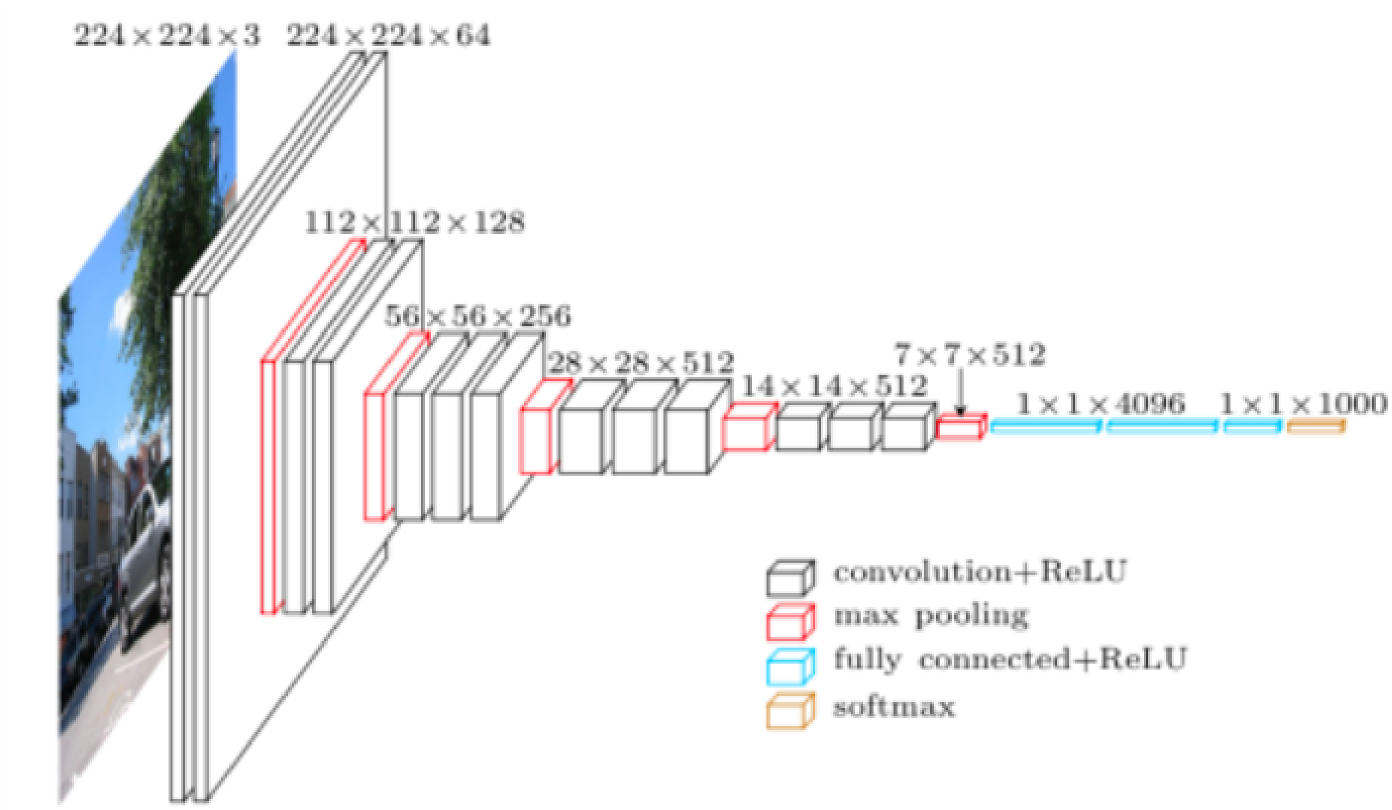
$$L'_\ell = \mathrm{Padd}(L_\ell,\ p)$$

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

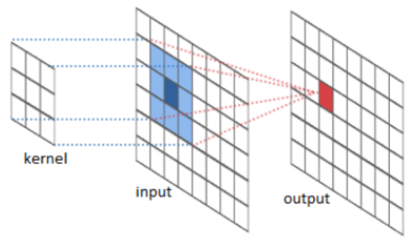
$$\sigma\left(\left(\sum_{\Delta x, \Delta y, i}\ W[\Delta x, \Delta y, i, j]\ L'_\ell[b, x + \Delta x, y + \Delta y, i]\right) - B[j]\right)$$

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



computing the max value over the box

s: stride parameter
typical stride = 2

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

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.

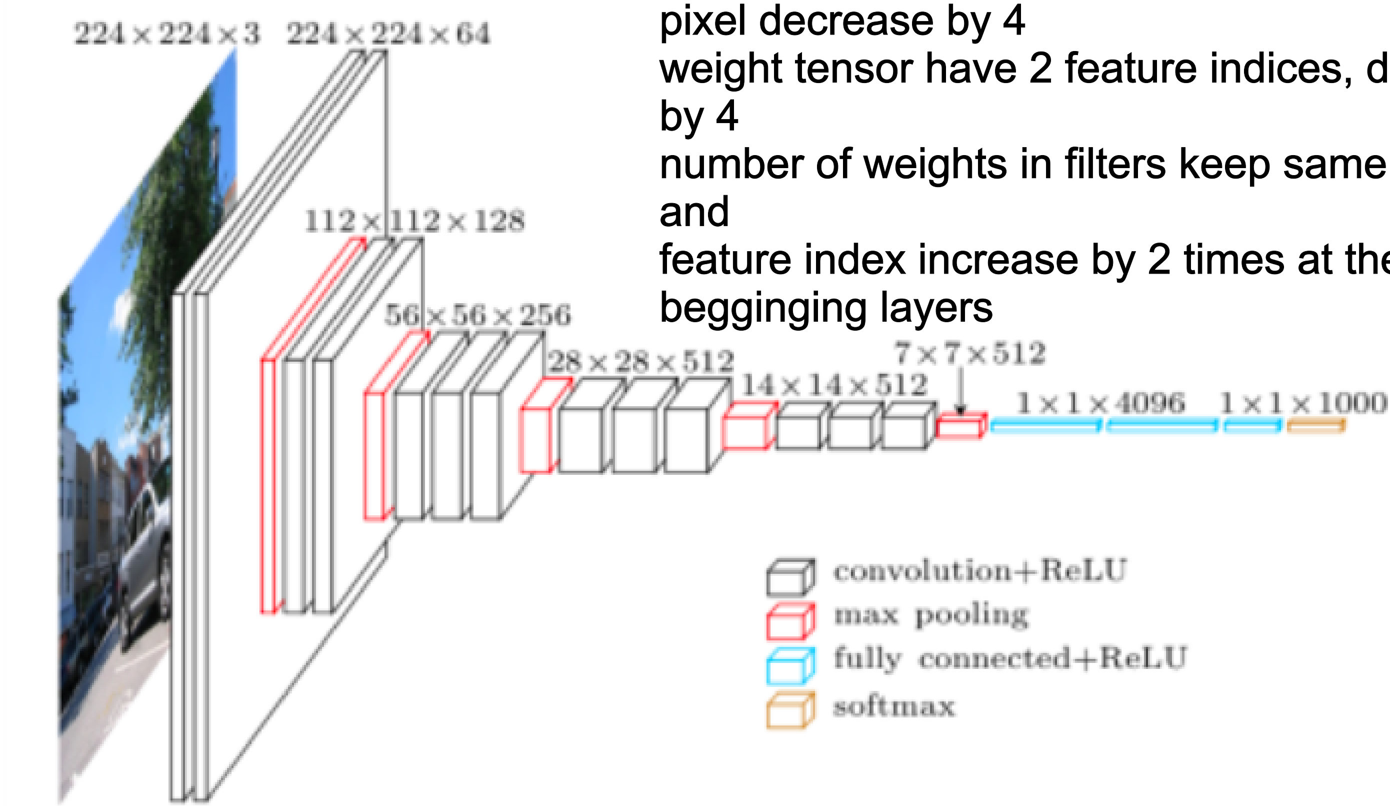
$$L_{\ell+1}[b, x, y, j] =$$

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

For strides greater than 1 the spatial dimention is reduced.

Fully Connected (FC) Layers

every time the special decrease by 2,
pixel decrease by 4
weight tensor have 2 feature indices, decrease by 4
number of weights in filters keep same
and
feature index increase by 2 times at the beggingging 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.

convolution write as a matrix
multiplication

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

6 variable here

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

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

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

build a bigger tensor

Image to Column (Im2C)

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

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

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

group 3 of the same indices

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

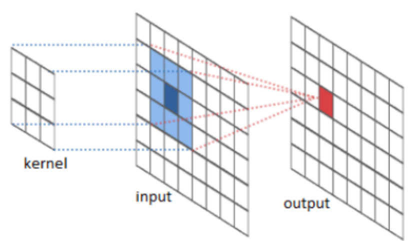
treat as one index

L becomes a matrix,   W is a matrix

Dilation

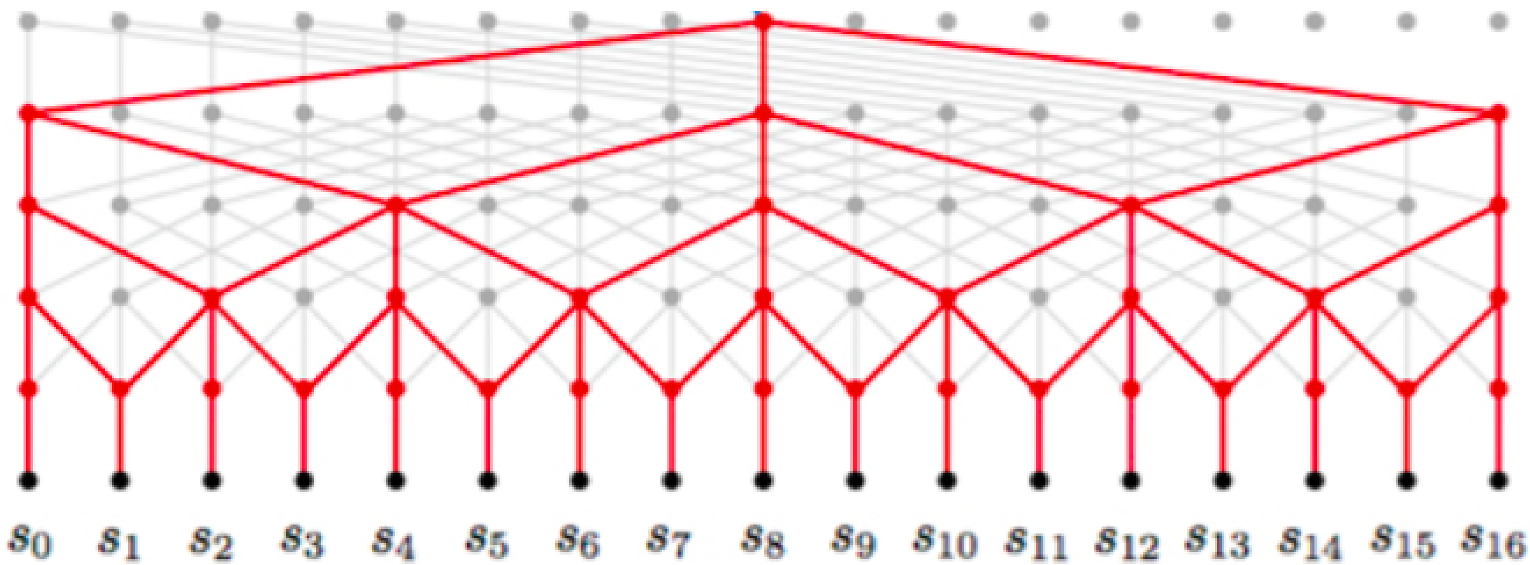
A CNN for image classification typically reduces an $N \times N$ image to a single feature vector.

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



This is called a “fully convolutional” CNN.

Dilation

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

d = dilation parameter

we are spreading out the previous layer

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

https://towardsdatascience.com/review-dilated-convolution-semantic-

Hypercolumns

An alternative to dilation and fully convolutional networks is hypercolumns.

$$L[b,x,y] = L_1[b,x,y];\cdots;L_\ell[b,\lfloor x/W_\ell\rfloor,\lfloor y/H_\ell\rfloor];\cdots;L_{\mathcal{L}}[b]$$

where

$$L_\ell[b,\lfloor x/W_\ell\rfloor,\lfloor y/H_\ell\rfloor]\;;\;L_{\ell+1}[b,\lfloor x/W_{\ell+1}\rfloor,\lfloor y/H_{\ell+1}\rfloor]$$

denotes the concatenation of vectors

$$L_\ell[b,\lfloor x/W_\ell\rfloor,\lfloor y/H_\ell\rfloor]$$

and

$$L_{\ell+1}[b,x/W_{\ell+1},y/H_{\ell+1}].$$

making convolution weight lighter

i = 1000
group = 10

each group 100 features
so the summation if from 1
to 100

Grouping

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

concat of convolutionally computed vectors

$$L^g_{\ell+1}[b,x,y,j]$$

each group is computed by
this

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

For a fixed number of features j in the total output, using G groups reduces the number of weight parameters by a factor of G.

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, in-channels,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.

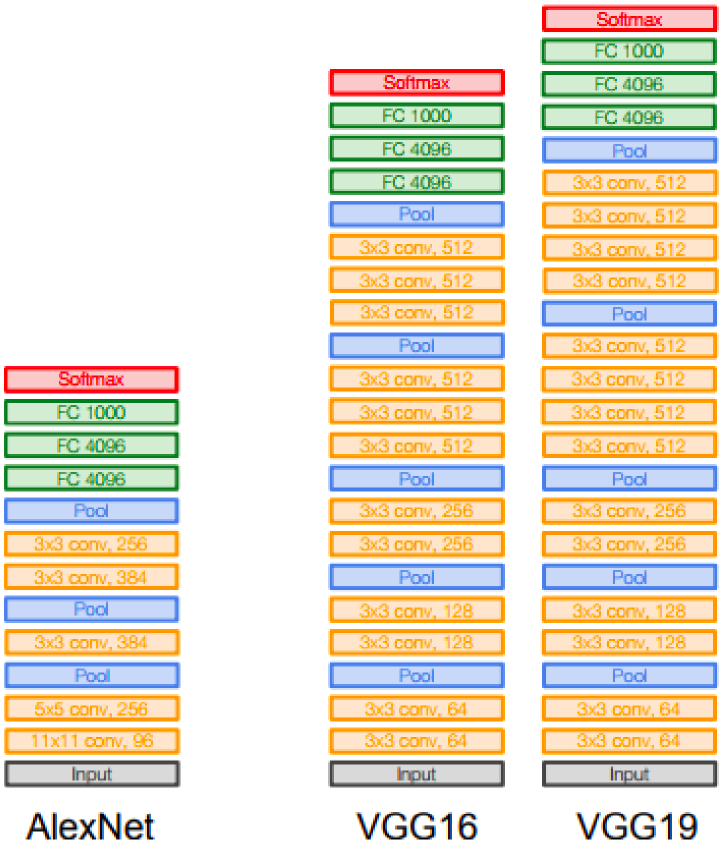
Resnet and resnet-like architectures are now dominant (next lecture).

Alexnet

Given Input$[227, 227, 3]$

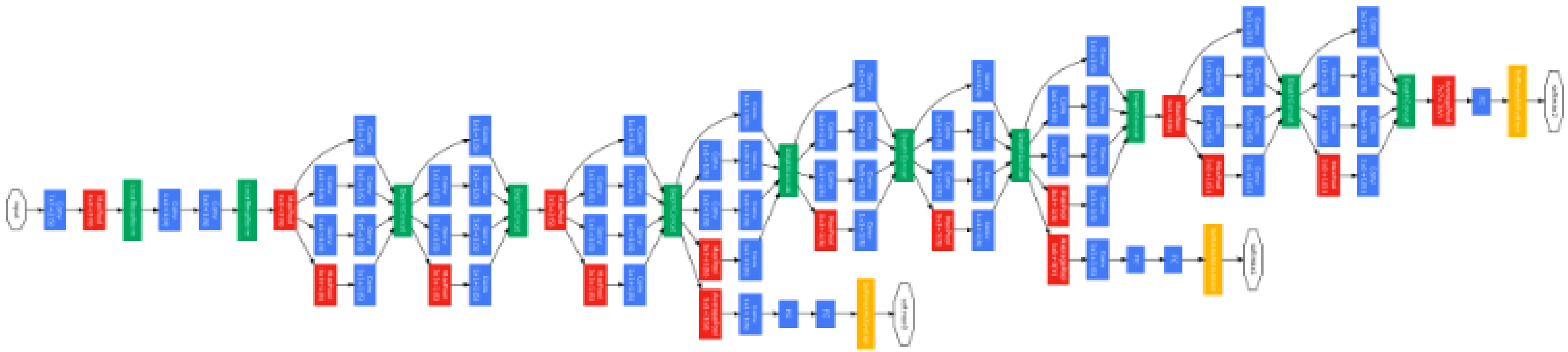
$$
\begin{aligned}
L_1[55 \times 55 \times 96] &= \mathrm{ReLU}(\mathrm{CONV}(\mathrm{Input}, \Phi_1, \text{width } 11, \text{pad } 0, \text{stride } 4)) \\
L_2[27 \times 27 \times 96] &= \mathrm{MaxPool}(L_1, \text{width } 3, \text{stride } 2)) \\
L_3[27 \times 27 \times 256] &= \mathrm{ReLU}(\mathrm{CONV}(L_2,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

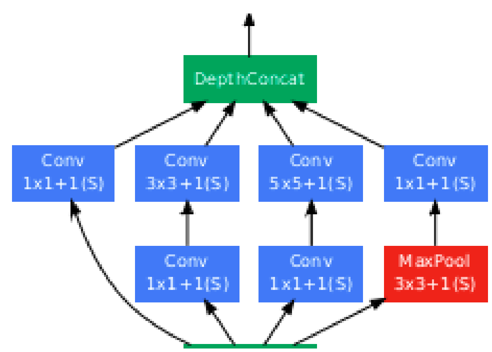
VGG



Stanford CS231

Inception, Google, 2014



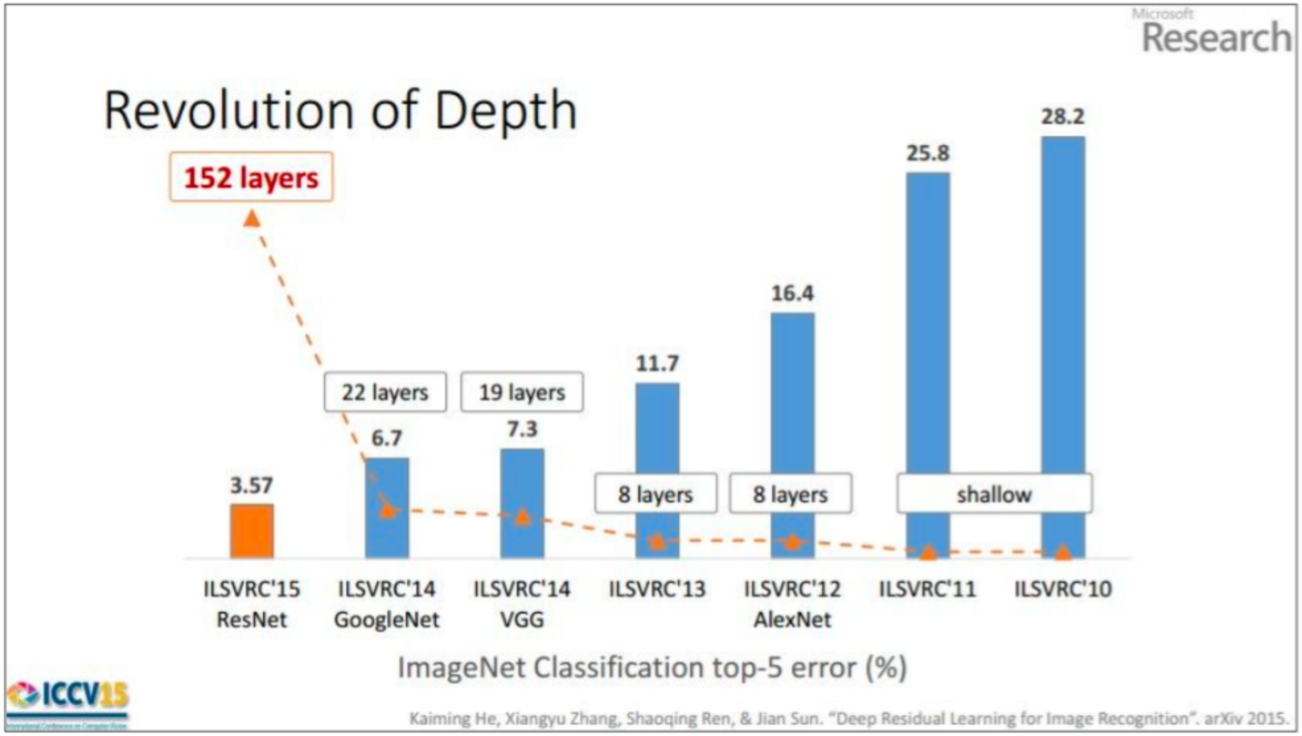


Models for Image Classification in PyTorch

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

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%

END