TTIC 31230, Fundamentals of Deep Learning

David McAllester, Autumn 2023

Eintein Notation

and Convolutional Neural Networks (CNNs)

Einstein Notation

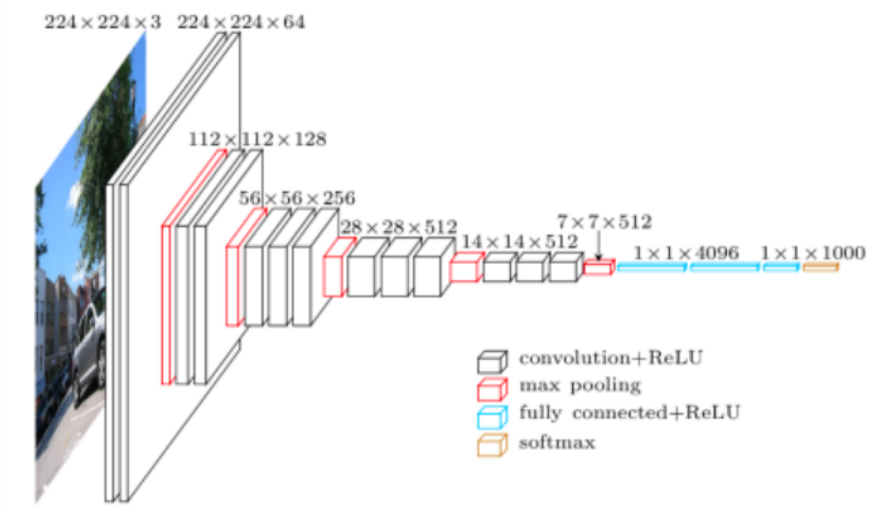
For the representation of general relativity, Einstein introduced the convention of explicitly writing all indeces of tensors where repeated indeces in a product of tensors are implicitly summed.

Writing indeces explicitly improves the clarity of the notation at the expense of not being in correspondence with framework notation. Most frameworks hide indeces.

This course will focus on conceptual understanding rather than framework implementations. For conceptual understanding Einstein notation seems preferable.

Advantages of Einstein Notation

The indeces of tensors generally have types such as "batch index", "x coordinate", "y coordinate" and "neuron index".



A layer in a CNN has "shape" $L[b, x, y, n]$ where b is a batch intex, n is a neuron index, and x and y specify a spacial locotation.

Advantages of Einstein Notation

A layer in a CNN has shape $L[b, x, y, n]$.

Writing the indeces explicitily typically makes the meaning of indeces clear and avoids having to remember the somewhat arbitrary order of the indeces (the order matters for efficiency of memory access — discussed later).

Einstein Notation

We will use a modified form of Einstein notation where captial letters are used to denote slices of a tensor. For example:

- $L[b, x, y, n]$ denotes a single (scalar) number.
- $L[b, x, y, N]$ denotes a vector of neuron values.
- $L[b, X, Y, N]$ denotes the entire layer of the bth batch element.

Einstein Notation

Frameworks (and NumPy) use C-order (row-major) in laying out a tensor in memory.

The vector $L[b, x, y, N]$ denotes a contiguous block of memory.

The bth batch layer $L[b, X, Y, N]$ also denotes a (larger) contiguous block of memory.

Vector and matrix operations on contiguous memory better utilize the memory hierarchy (caching).

Einstein Notation

Following Einstein I will use repeated capital letters in a product of tensors to denote summation over those letters.

$$\begin{aligned} y = Wx \quad &\equiv \quad y[i] = \sum_j W[i,j]x[j] \\ &\equiv \quad y[i] = W[i,J]x[J] \end{aligned}$$

$$\begin{aligned} y = x^\top W \quad &\equiv \quad y[j] = \sum_i W[i,j]x[i] \\ &\equiv \quad y[j] = W[I,j]x[I] \end{aligned}$$

Einstein Notation

For vectors x and y and matrices A, B, and C we have

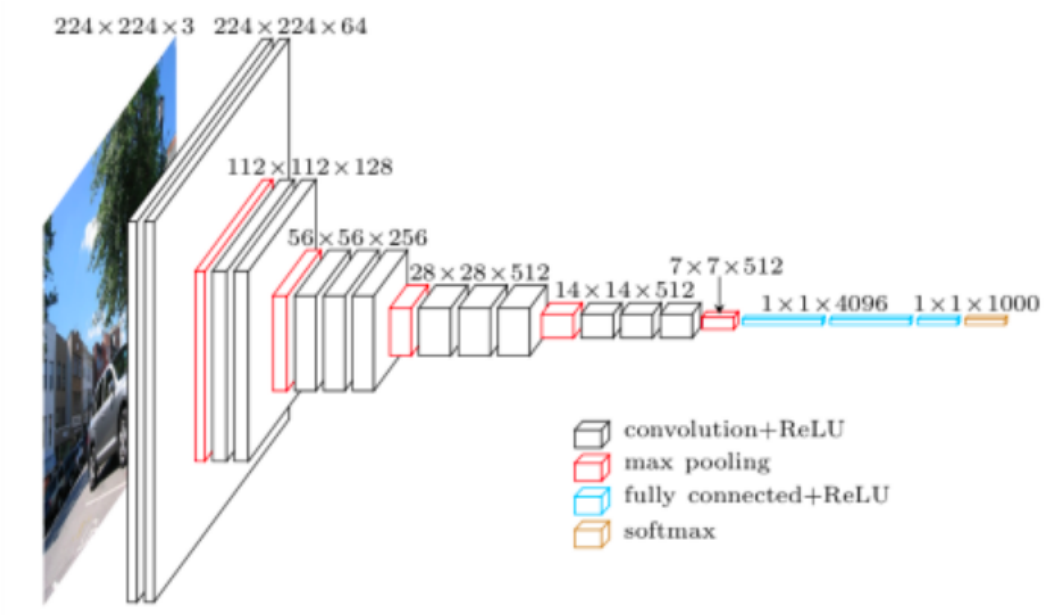
$$y = Ax \quad \equiv \quad y[i] = A[i, J]x[J]$$

$$y = x^\top A \quad \equiv \quad y[j] = A[I, j]x[I]$$

$$A = B^\top C \quad \equiv \quad A[i, j] = B[K, i]C[K, j]$$

In Einstein notation we never use transpose.

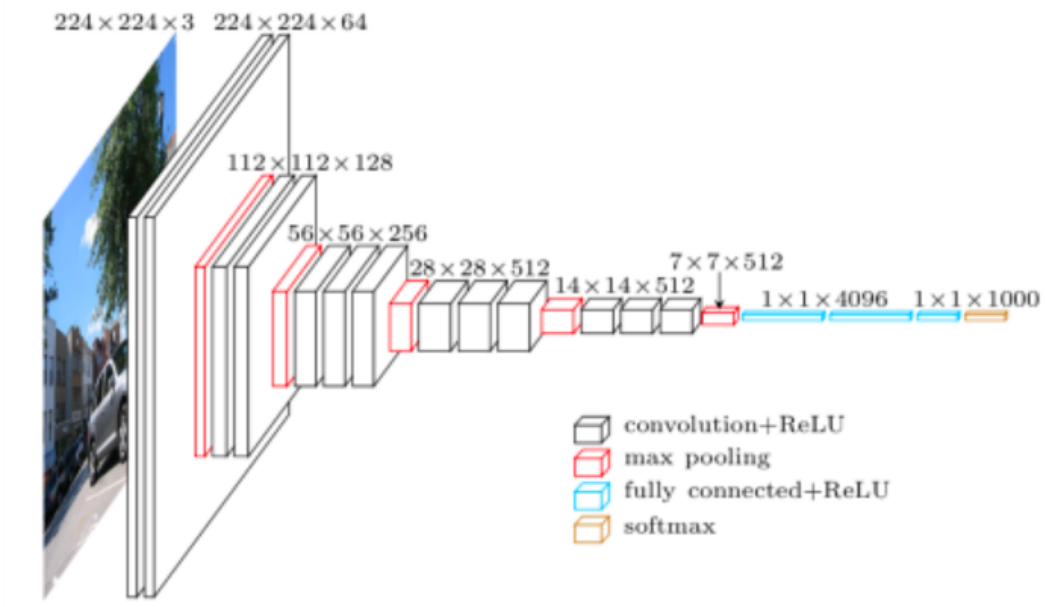
Convolution



A layer is a tensor shape $L(b, x, y, n)$.

I will refer to the index n as a "neuron". The index n is also called a "feature" or "channel".

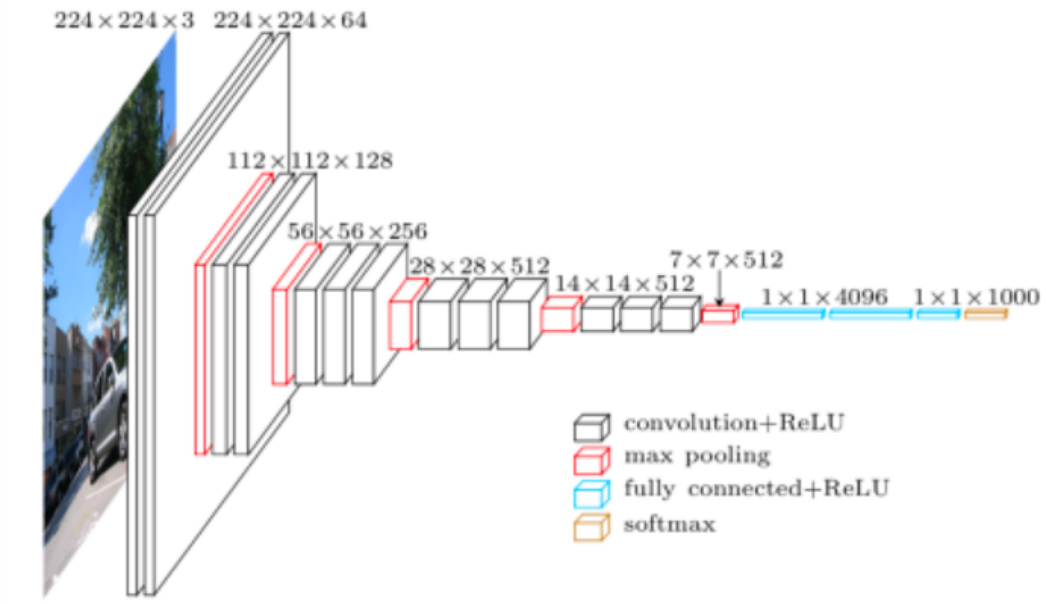
Convolution



The input image (for each batch element) has three "neurons" (Red, Green, Blue).

In VGG (above) the input image is $224 \times 224 \times 3$ (for each batch element). The next two layers are $224 \times 224 \times 64$.

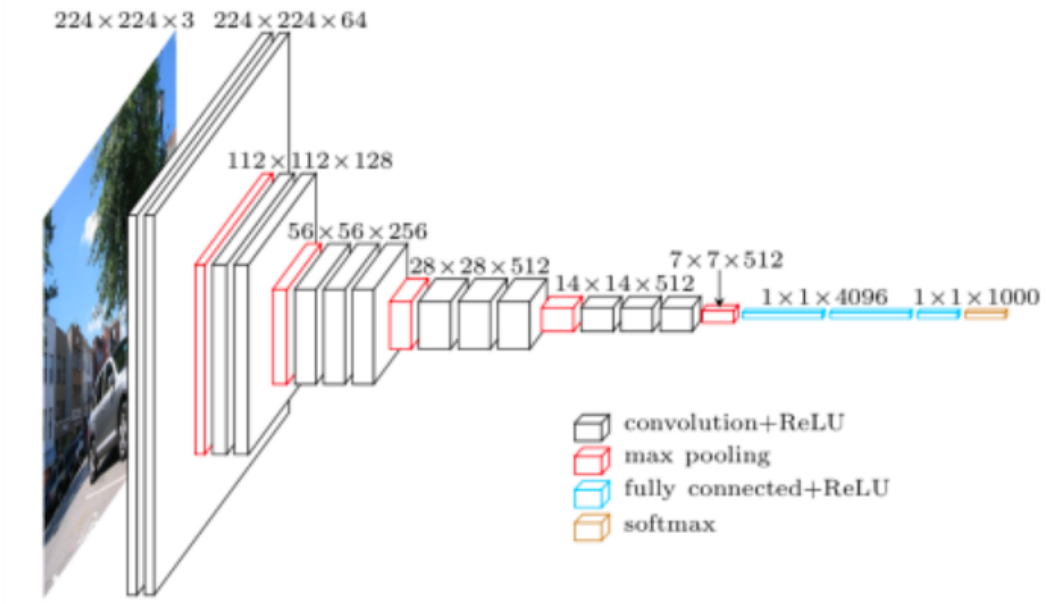
Convolution



A given neuron (index) n occurs in every batch element and every position x, y in the tensor $L(b, x, y, n)$.

Each occurance of that neuron uses the same computation rule for compurting its response.

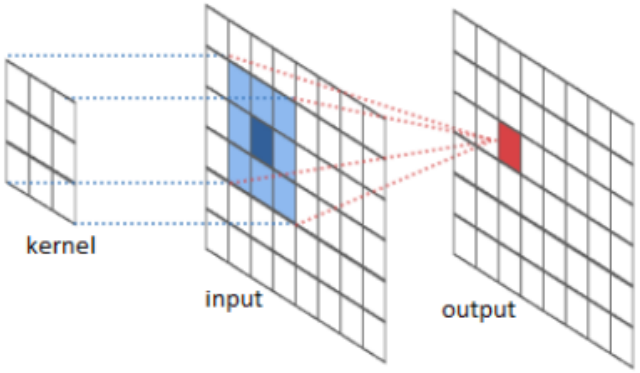
Convolution



A given neuron (index) n occurs in every batch element and every position x, y in the tensor $L(b, x, y, n)$.

The "receptive field" of an occurance of neuron n at position x, y is a region in the previous layer centered at x and y.

A Convolution Layer



$K_{\ell+1}[n_{\text{out}}, \Delta x, \Delta y, n_{\text{in}}]$ $L_\ell[b, x, y, n_{\text{in}}]$ $L_{\ell+1}[b, x, y, n_{\text{out}}]$

$$L_{\ell+1}[b, x, y, n_{\text{out}}]$$

$$= \sigma\left(K_{\ell+1}[n_{\text{out}}, \Delta X, \Delta Y, N_{\text{in}}]\; L_\ell[b, x + \Delta X, y + \Delta Y, N_{\text{in}}] - B[n_{\text{out}}]\right)$$

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

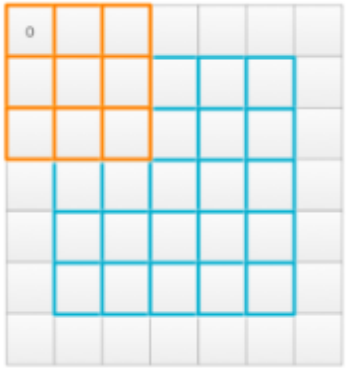
stride – Single number or (sH, sW). Default: 1

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

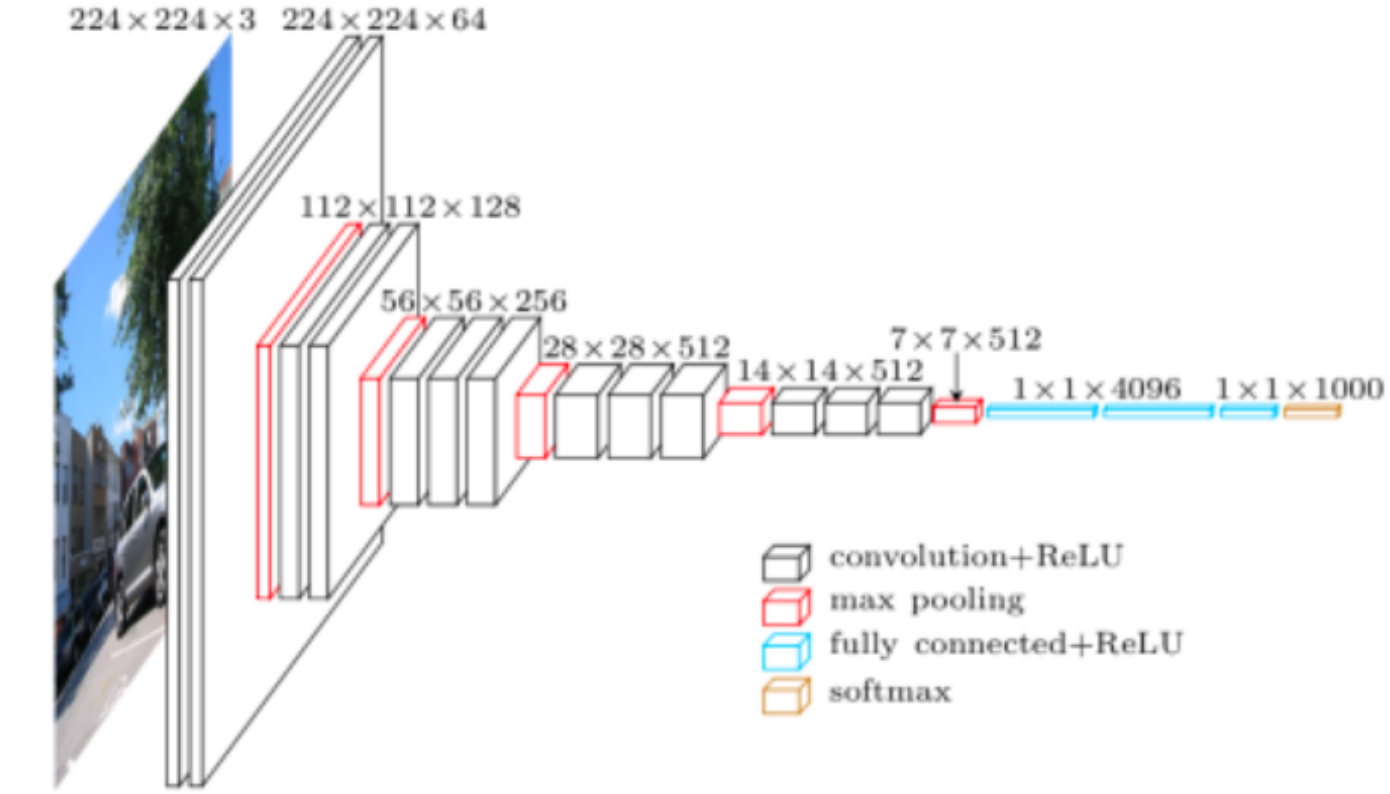
$$L'_\ell = \text{Padd}(L_\ell,\ p)$$

$$L_{\ell+1}[b, x, y, n_{\text{out}}]$$

$$= \sigma\left(K_{\ell+1}[n_{\text{out}}, \Delta X, \Delta Y, N_{\text{in}}]\ L_\ell[b, x + \Delta X, y + \Delta Y, N_{\text{in}}] - B[n_{\text{out}}]\right)$$

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

Reducing Spatial Dimention



Reducing Spatial Dimensions: Strided Convolution

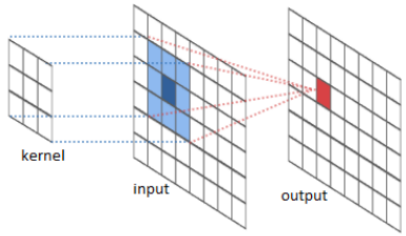
We can move the filter by a "stride" s for each spatial step.

$$L_{\ell+1}[b, x, y, n_{\text{out}}]$$

$$= \sigma\left(K_{\ell+1}[n_{\text{out}}, \Delta X, \Delta Y, N_{\text{in}}]\; L_{\ell}[b, s * x + \Delta X, s * y + \Delta Y, N_{\text{in}}] - B[n_{\text{out}}]\right)$$

For strides greater than 1 the spatial dimention is reduced.

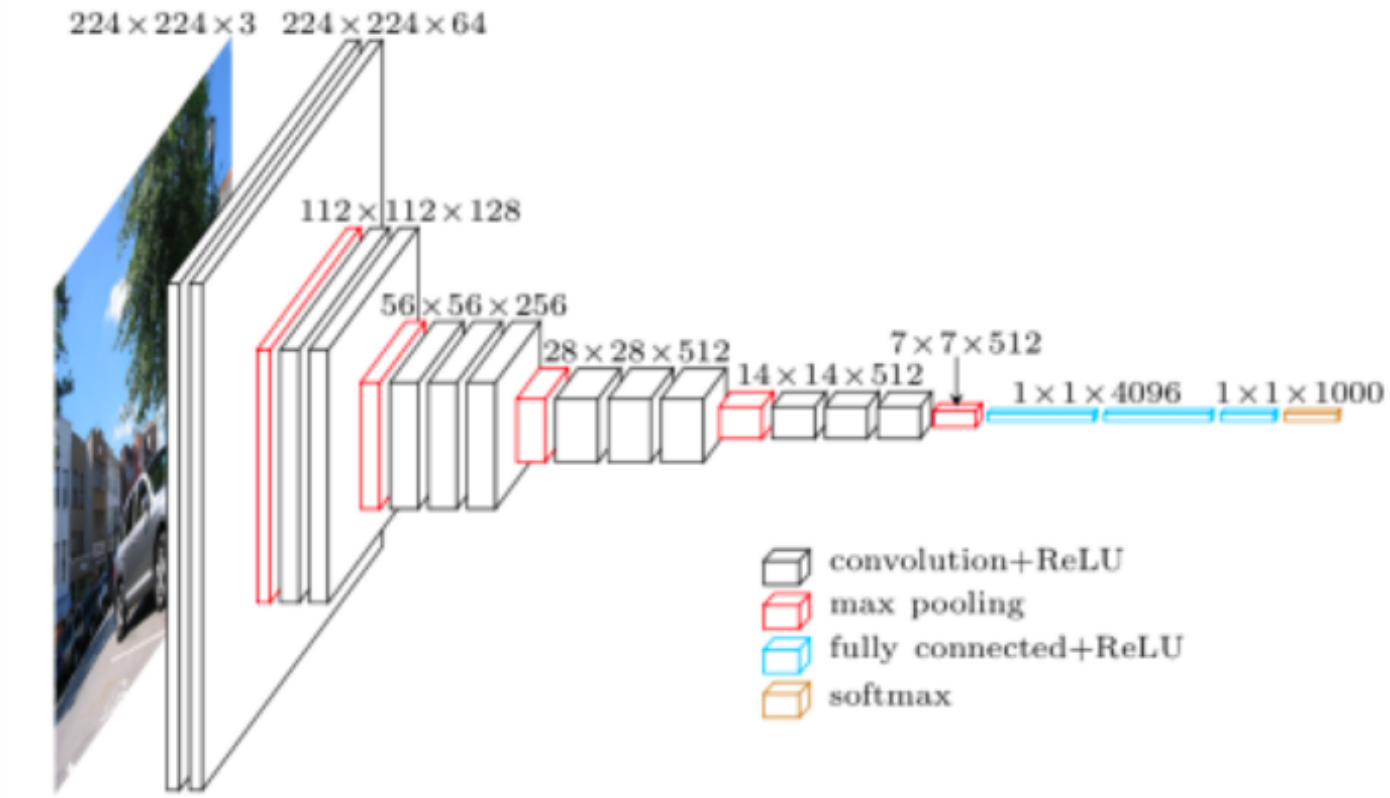
Reducing Spatial Dimensions: Max Pooling



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

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

Fully Connected (FC) Layers



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

Dilation and Grouping

Dilation is used for "hypercolumns" where higher layers have the same spatial dimension as the input but each spatial location in a higher layer is a whole-image representation of a region of the input image.

Grouping reduces the computation by limiting the inputs to a neuron to be values in the same "neuron group" as the input.

Dilation and grouping are rarely used today.

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 has disappeared.

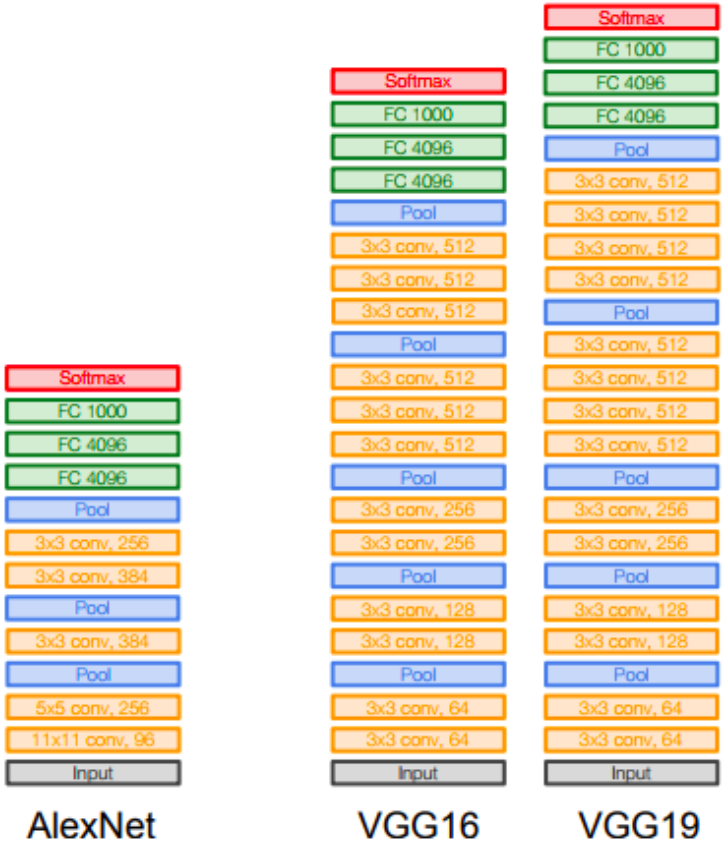
ResNet and resnet-like architectures are now dominant.

Alexnet, 2012

Given Input$[227, 227, 3]$

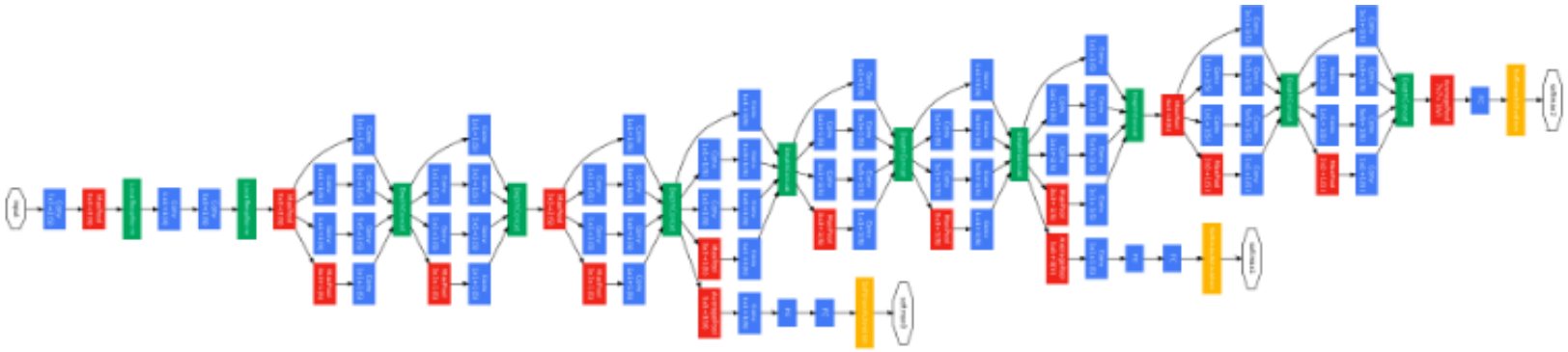
$$
\begin{aligned}
L_1[55 \times 55 \times 96] &= \text{ReLU}(\text{CONV}(\text{Input}, \Phi_1, \text{width } 11, \text{pad } 0, \text{stride } 4)) \\
L_2[27 \times 27 \times 96] &= \text{MaxPool}(L_1, \text{width } 3, \text{stride } 2)) \\
L_3[27 \times 27 \times 256] &= \text{ReLU}(\text{CONV}(L_2,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

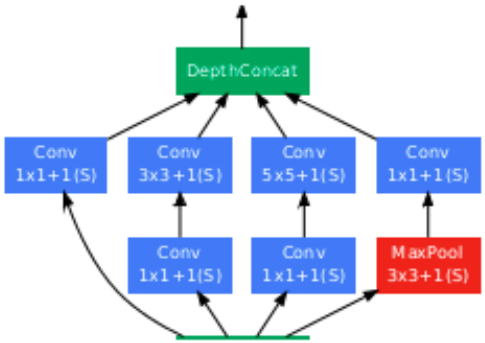
VGG, 2014



Stanford CS231

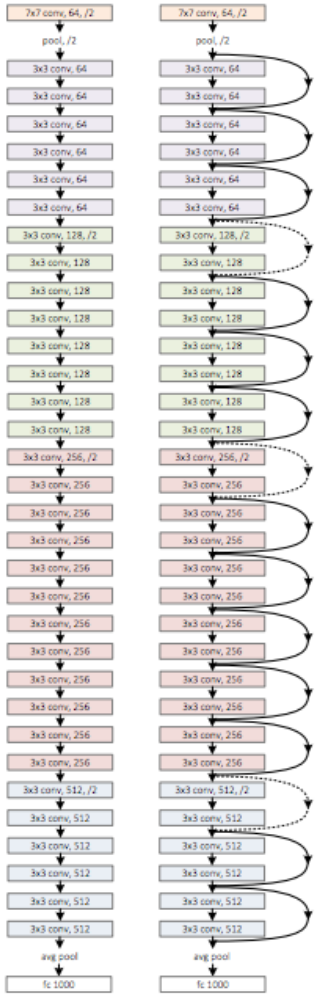
Inception, Google, 2014





ResNet, 2015

plain net



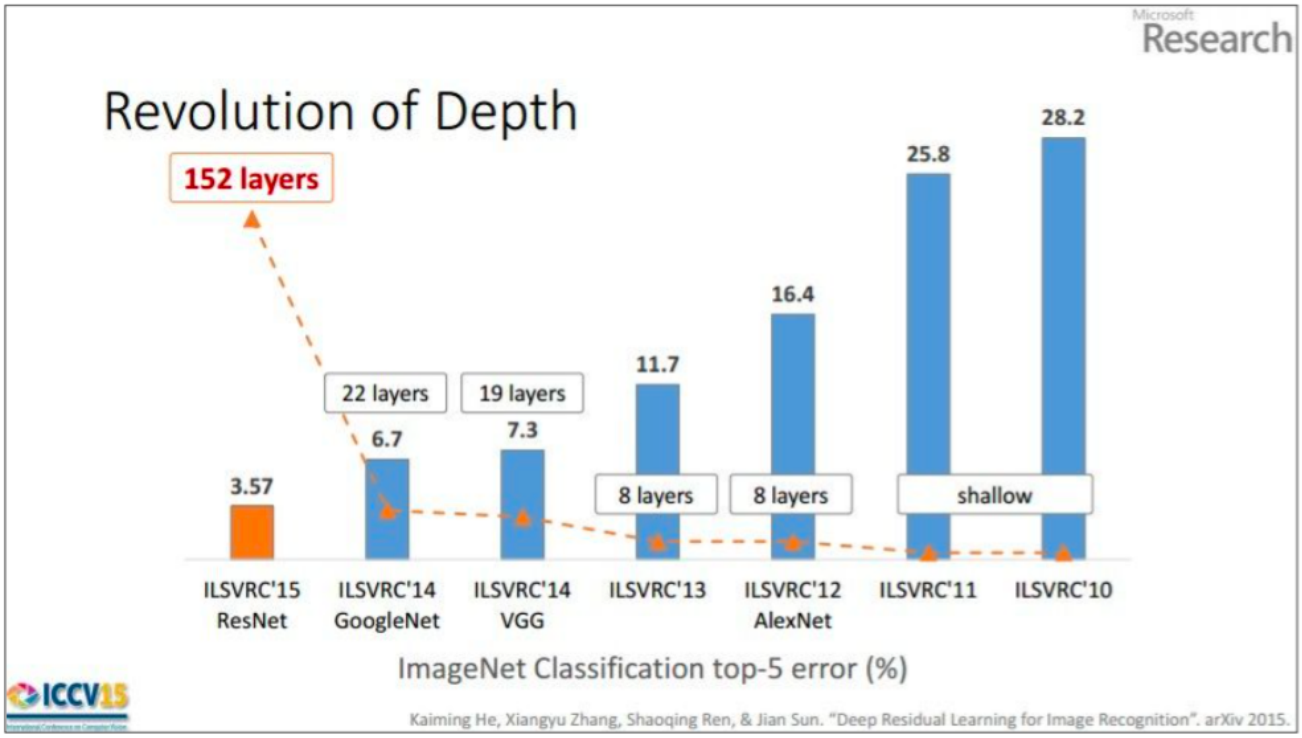
ResNet

er)

[Kaiming He]

CNN Imagenet Classification

1000 kinds of objects.



(slide from Kaiming He's recent presentation)

2016 was 3.0%, 2017 was 2.25%, 2020 was 1.3%

END