Deep Learning for Computer Vision

Ahmed Hosny Abdel-Gawad

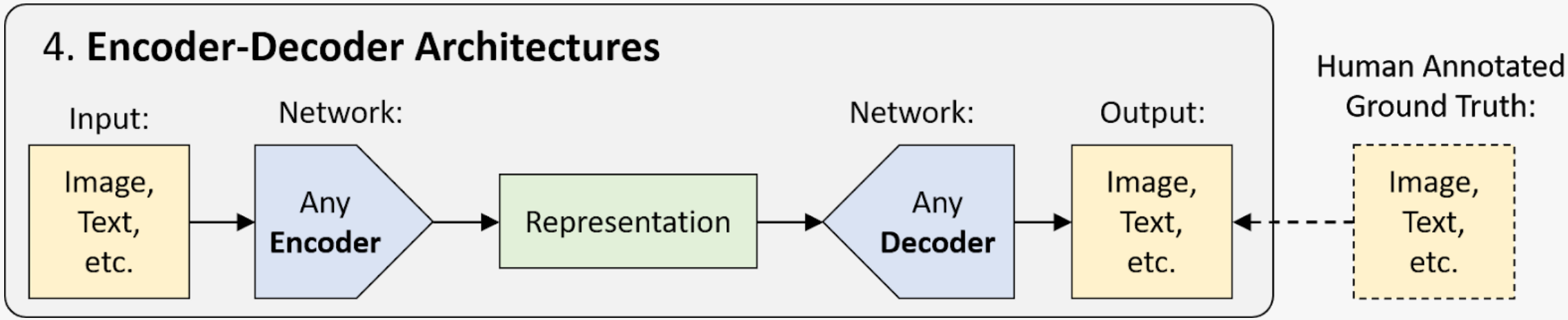
Senior AI/CV Engineer

Deep Learning:
ConvNets

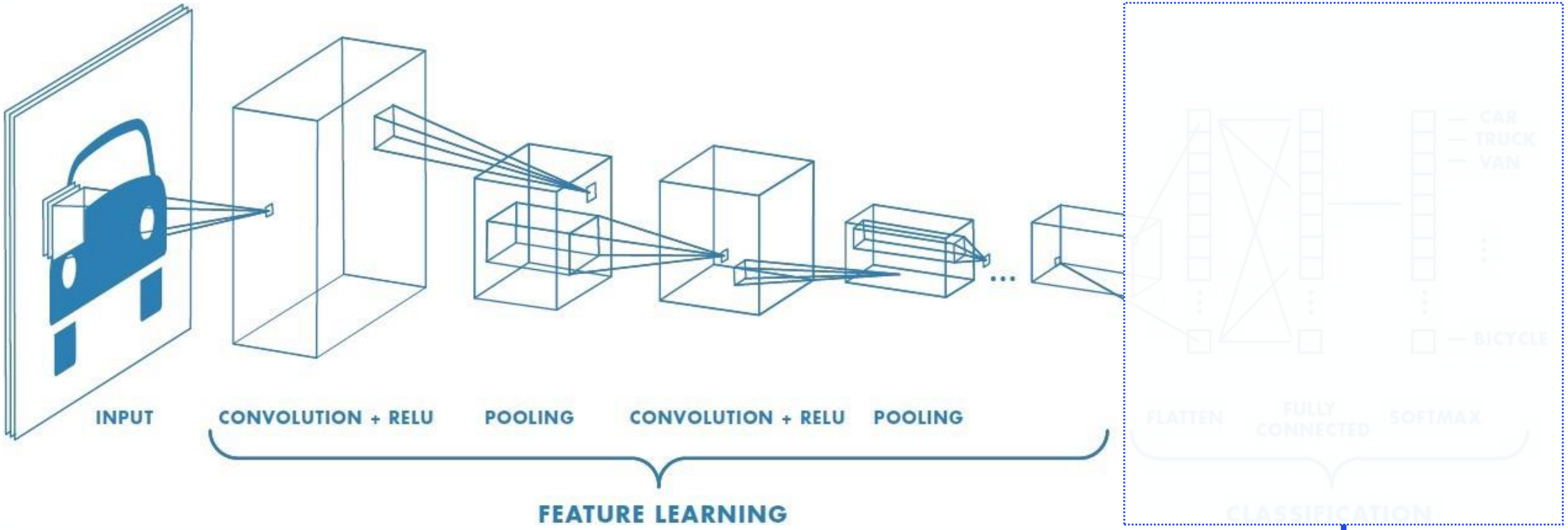
Encoder-Decoder Architectures (Pattern)

At a high-level, neural networks are either encoders, decoders, or a combination of both:

- **Encoders** find patterns in raw data to form compact, useful representations.
- **Decoders** generate high-resolution data from those representations. The generated data is either new examples or descriptive knowledge.



An Architecture for Many Applications



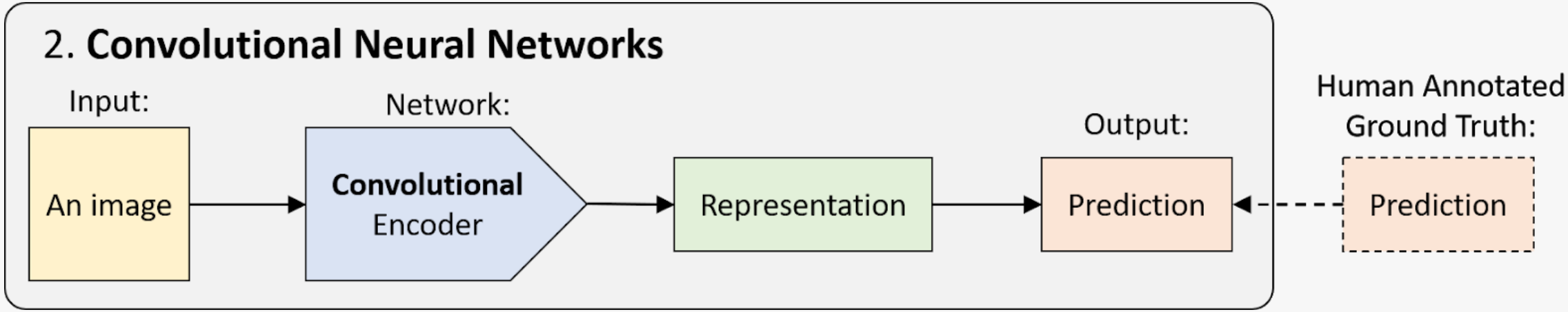
Classification

Object Detection

Segmentation

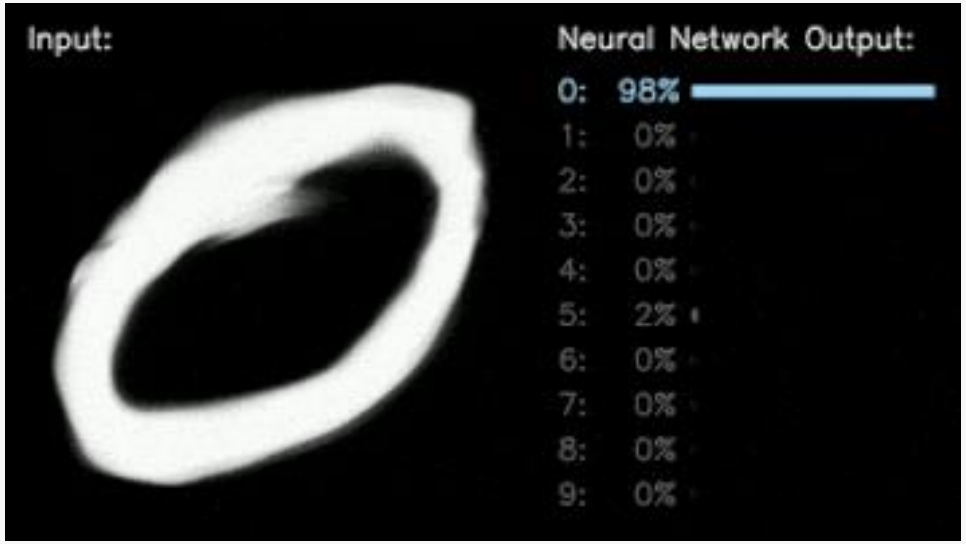
...

Convolutional Neural Networks (CNNs)



Instead of using only densely-connected layers, they use convolutional layers (**convolutional encoder**).

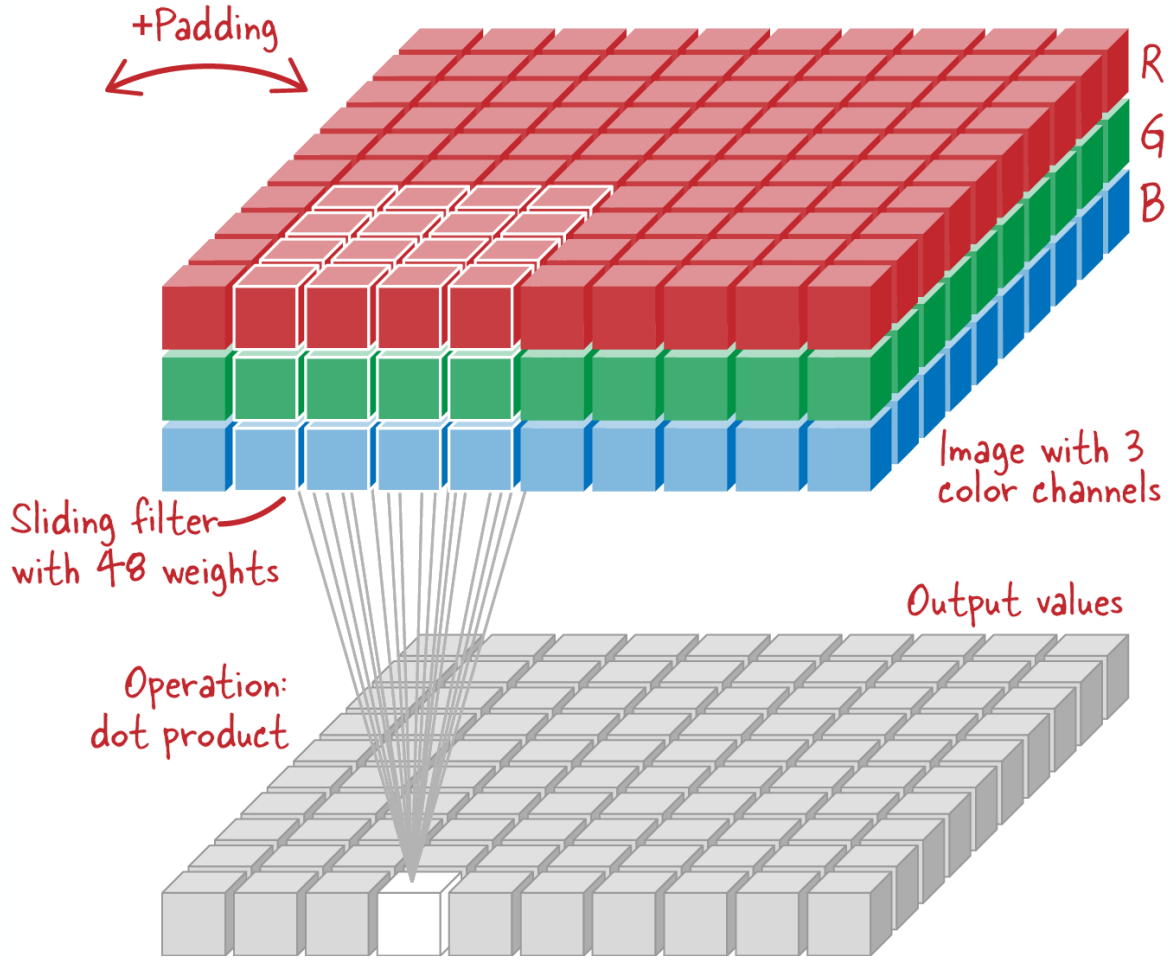
These networks are used for image classification, object detection, video action recognition, and any data that has some spatial invariance in its structure (e.g., speech audio).



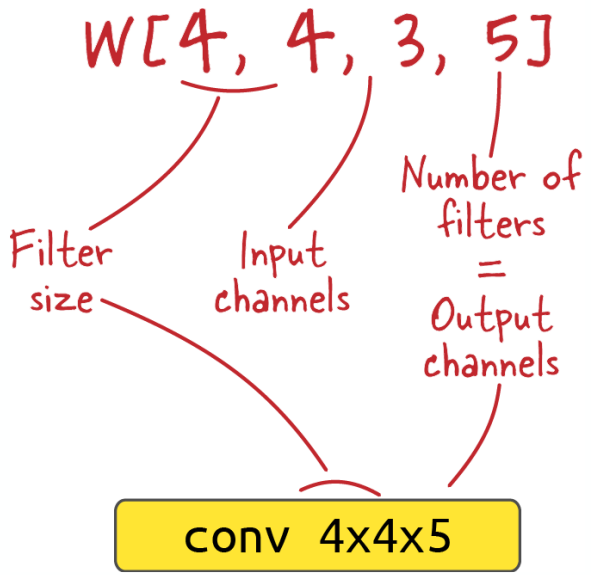
Convolutional Neural Networks (CNNs)

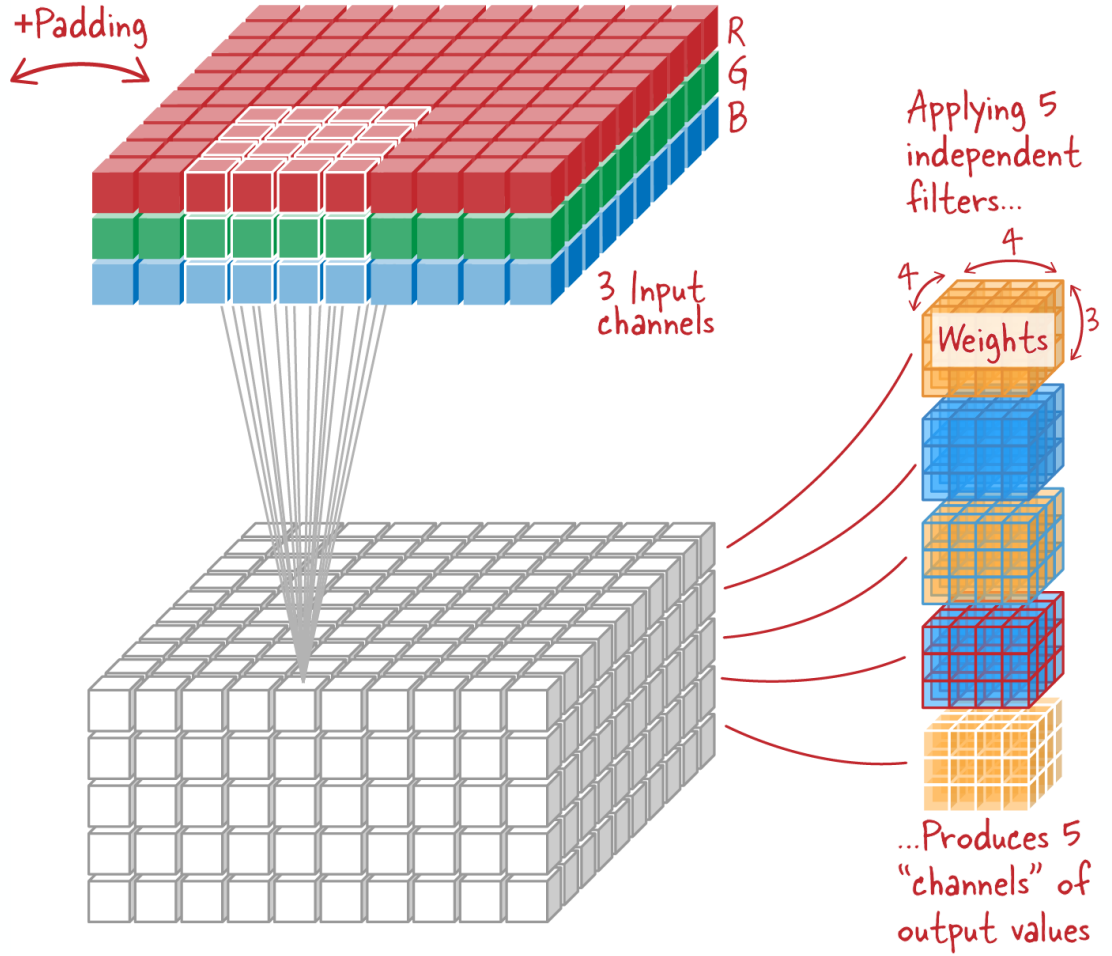
- Convolutional layers were designed specifically for images.
- They operate in **two dimensions** and can capture shape information.
- They work by sliding a small window, called a **convolutional filter**, across the image in both directions.

For color images (RGB), the filter will have $4 \times 4 \times 3 = 48$ learnable weights in total.



CNNs: Weight Calculation

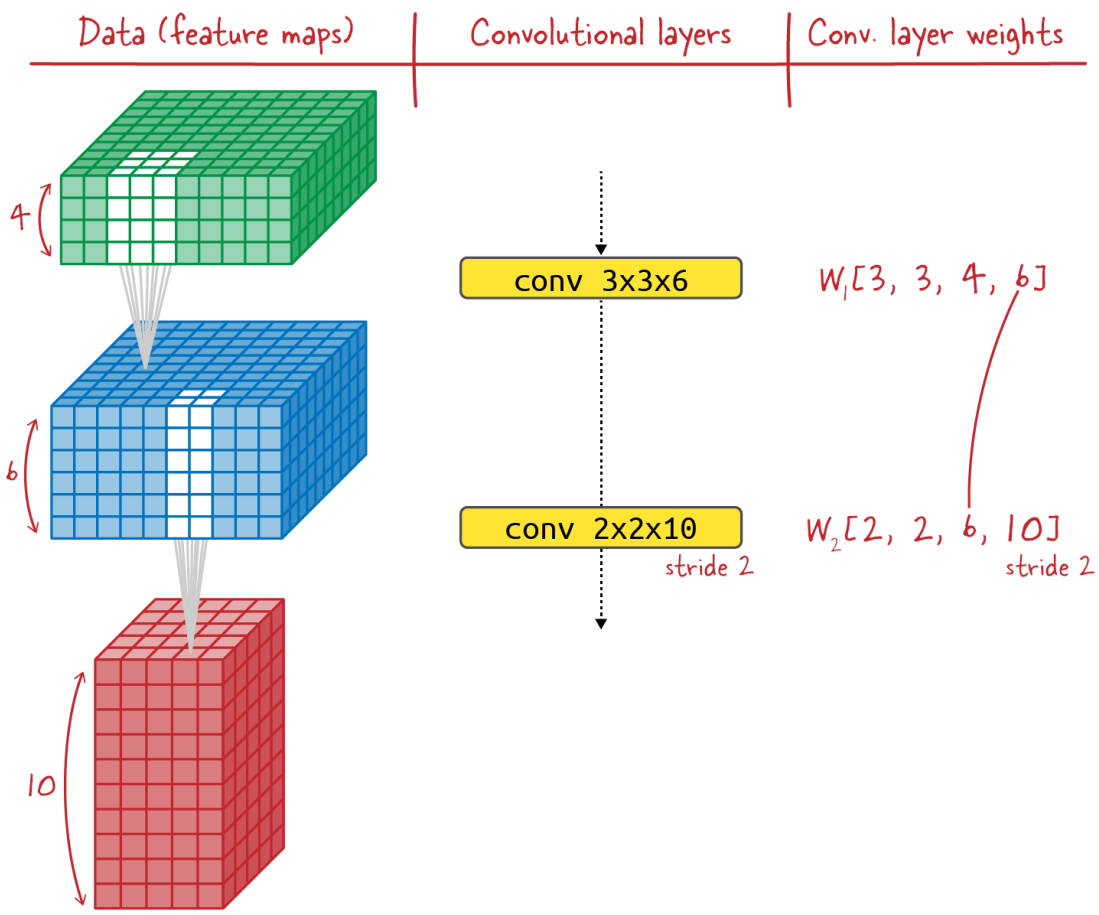




With **5** filters applied, the total number of learnable weights in this convolutional layer is $\mathbf{4 \times 4 \times 3 \times 5 = 240}$.

CNNs: Stacking Convolutional Layers

- Starting from the top, the first layer is a $3x3$ filter applied to an input with 4 channels of data.
- The filter is applied to the input 6 times, each time with **different filter weights**, resulting in 6 channels of output values.
- This is fed into a second convolutional layer using $2x2$ filters.
- Notice that the second convolutional layer uses a **stride of 2** (every other pixel) when applying its filters to obtain fewer output values (in the horizontal plane).

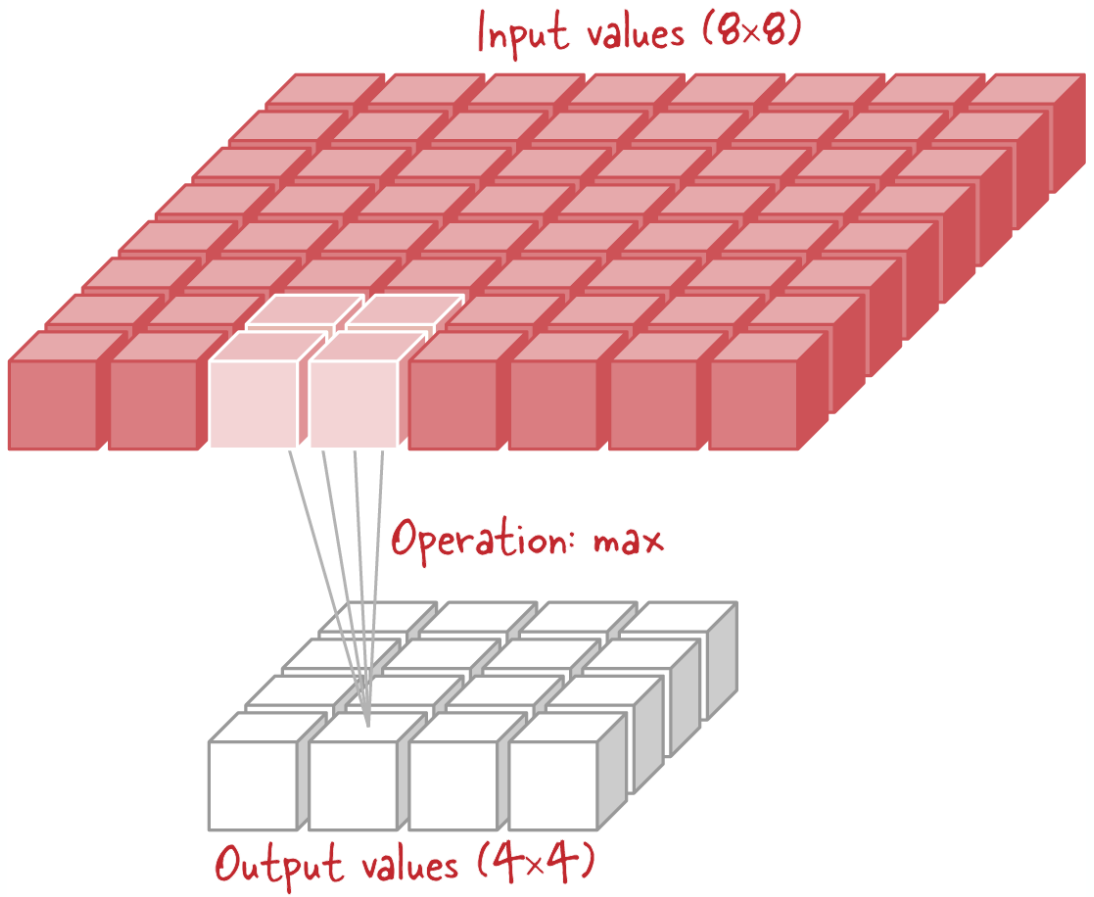


CNNs: Pooling Layers

The number of filters applied in each convolutional layer determines the number of channels in the output. **But how can we control the amount of data in each channel?**

The most commonly used downsampling operation is $2x2$ **max pooling**.

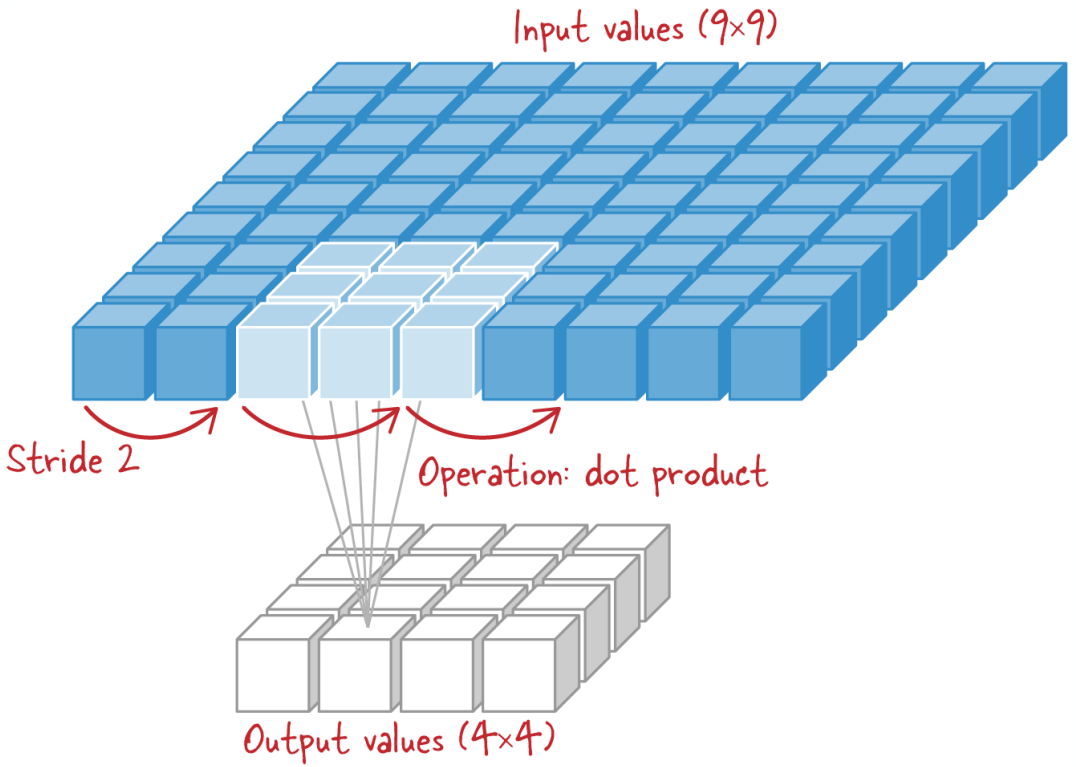
The most commonly used downsampling operation is $2x2$ **max pooling**.



CNNs: Downsampling with Convolutional Layers

A second option for downsampling channel information is to apply convolutions with a stride of $2\ or\ 3$ instead of 1.

The convolutional filters then slide over the input image by steps of $2\ or\ 3$ pixels in each direction.



ConvNets Sizes Calculation

CNNs: Sizing

We can use the following equations to calculate the exact size of the convolution output for an input with the size of ($width = \boldsymbol{W}, height = \boldsymbol{H}$), a Filter with the size of ($width = \boldsymbol{F_w}, height = \boldsymbol{F_h}$), strides of $\boldsymbol{S_w}$ and $\boldsymbol{S_w}$ in both directions, and Padding $\boldsymbol{P}$.

$$output\ width\ = \frac{W - F_w + 2P}{S_w} + 1$$

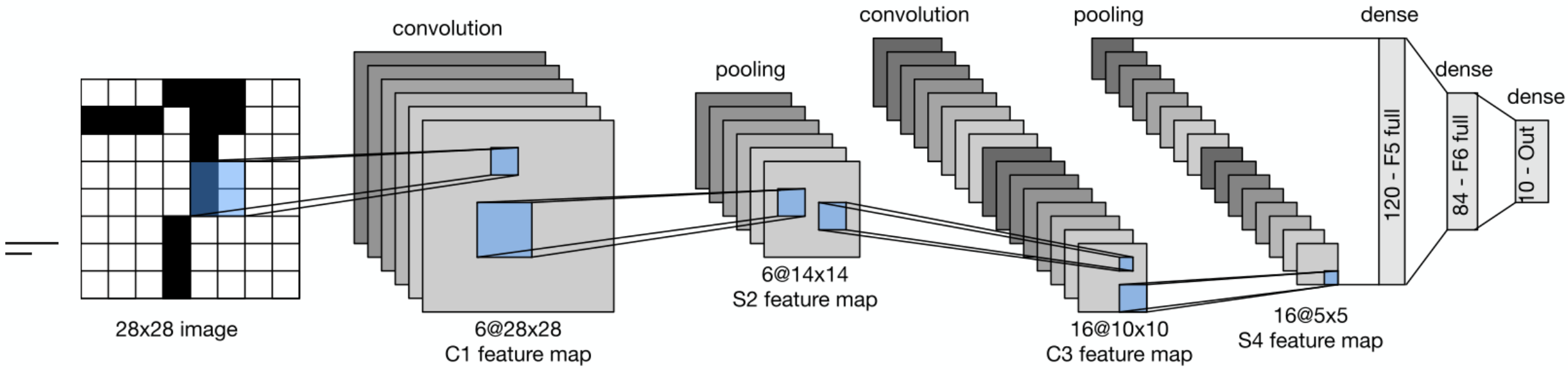
$$output\ height\ = \frac{H - F_h + 2P}{S_h} + 1$$

link...

ConvNets Meta Architectures


LeNet

Vanilla ConvNet (LeNet)

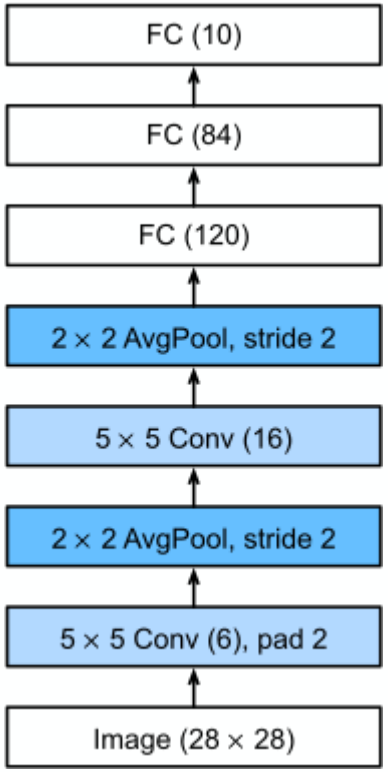


Instead of At a high level, LeNet (LeNet-5) consists of two parts:

1. A convolutional encoder consisting of **two** convolutional layers
2. A dense block (decoder) consisting of **three** fully connected layers

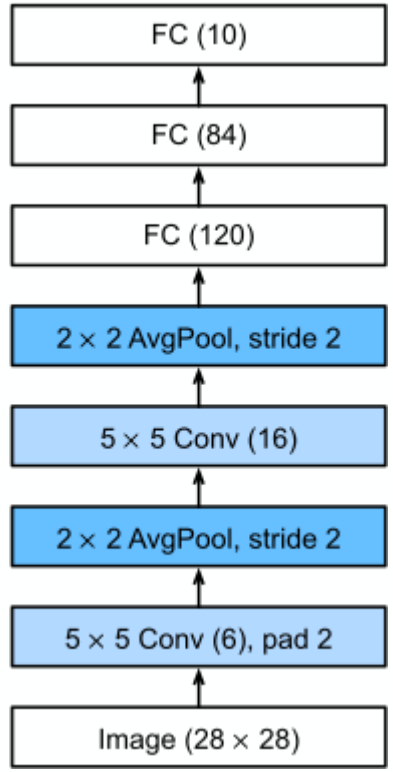
Vanilla ConvNet (LeNet)

- The **input** is images of size 28×28
- **C1** is the first convolutional layer with 6 convolution kernels of size 5×5.
- **S2** is the pooling layer that outputs 6 channels of 14×14 images. The pooling window size, in this case, is a square matrix of size 2×2.
- **C3** is a convolutional layer with 16 convolution kernels of size 5×5. Hence, the output of this layer is 16 feature images of size 10×10.
- **S4** is a pooling layer with a pooling window of size 2×2. Hence, the dimension of images through this layer is halved, it outputs 16 feature images of size 5×5.



Vanilla ConvNet (LeNet)

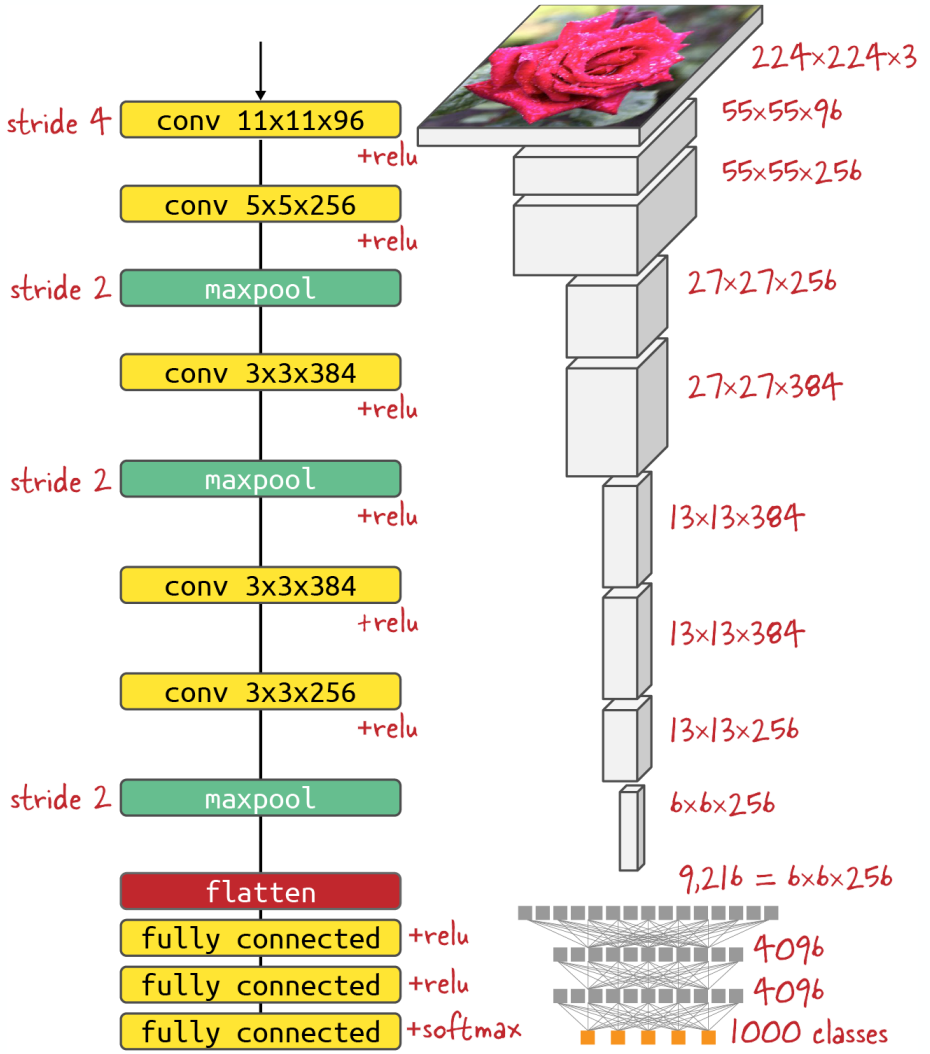
- **F5** is a fully connected layer with 120 neurons which are all connected to the flattened output of C4.
- **F6** is a fully connected layer with 84 neurons which are all connected to the output of F5.
- The **output** layer consists of 10 neurons corresponding to the number of classes (numbers from 0 to 9).




AlexNet

Deep ConvNet (AlexNet)

- In AlexNet's first layer, the convolution window shape is **11x11**, It is because of input size is large, so we need to use a large kernel to capture the object.
- The convolution window shape in the second layer is reduced to **5x5**, followed by **3x3**.
- In addition, after the first, second, and fifth convolutional layers, the network adds max-pooling layers with a window shape of and a stride of 2.
- Moreover, AlexNet has ten times more convolution channels than LeNet.



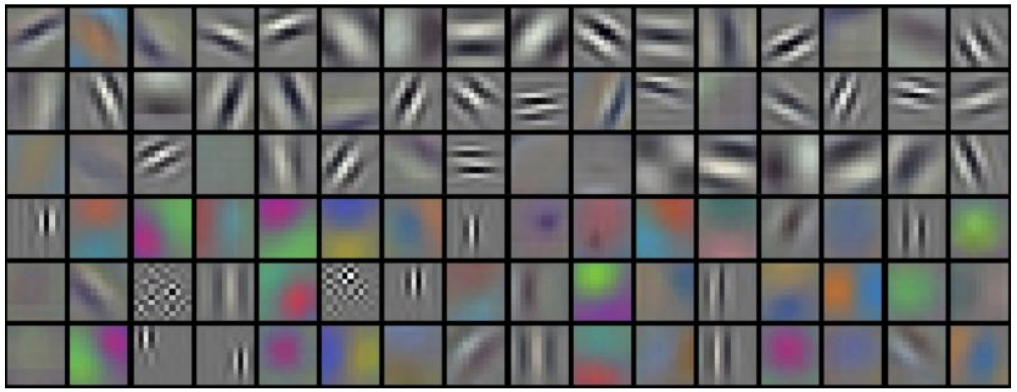
Deep ConvNet (AlexNet)

- AlexNet controls the model complexity of the fully connected layer by dropout with ratio of 50%.
- To augment the data even further, the training loop of AlexNet added a great deal of image augmentation, such as flipping, clipping, and color changes.

Model	Parameters (excl. classification head)	ImageNet accuracy
AlexNet	3.7M	60%

Deep ConvNet (AlexNet)

Notice: AlexNet starts with a very large $11x11$ convolutional filter. This is costly in terms of learnable weights. However, one advantage of the large $11x11$ filters is that their learned weights can be visualized as $\boldsymbol{11x11 - pixel}$ images.



- The network learned to detect ***vertical, horizontal and slanted lines*** of various orientations.
- Two filters exhibit a checkerboard pattern, which probably reacts to grainy textures in the image. You can also see detectors for single colors or pairs of adjacent colors.

- All these are **basic features** that subsequent convolutional layers will assemble into semantically more significant constructs.

The Quest for Depth

After AlexNet, researchers started increasing the depths of their convolutional networks. They found that adding more layers resulted in better classification accuracy. Several explanations have been offered for this:

1. The expressivity argument

A single layer is a linear function. It cannot approximate complex nonlinear functions, whatever its number of parameters. Each layer is, however, activated with a nonlinear activation function such as sigmoid or ReLU. Stacking multiple layers results in multiple successive nonlinearities and a better chance of being able to approximate the desired highly complex functionality.

2. The generalization argument

Adding parameters to a single layer increases the “memory” of the neural network and allows it to learn more complex things. However, it will tend to learn them by memorizing input examples. This does not generalize well. On the other hand, stacking many layers forces the network to break down its input semantically into a hierarchical structure of features.

The Quest for Depth

After AlexNet, researchers started increasing the depths of their convolutional networks. They found that adding more layers resulted in better classification accuracy. Several explanations have been offered for this:

3. The perceptive field argument

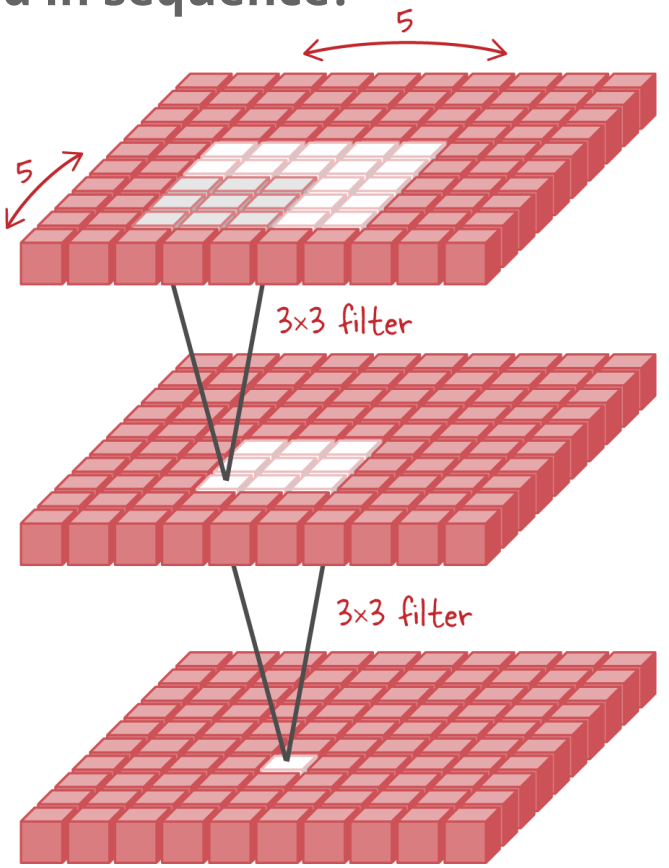
If a cat's head covers a significant portion of an image—say, a $128x128$-pixel region—a single-layer convolutional network would need $128x128$ filters to be able to capture it, which would be prohibitively expensive in term of learnable weights. Stacked layers, on the other hand, can use small 3x3 or $5x5$ filters and still be able to "see" any $128x128 - pixel$ region if they are sufficiently deep in the convolutional stack.

The Quest for Depth: Filter Factorization

In order to design deeper convolutional networks without growing the parameter count uncontrollably, researchers also started designing cheaper convolutional layers.

Which one is better: a 5x5 convolutional filter or two 3x3 filters applied in sequence?

Both have a receptive area of $5x5$. The difference is that **two 3x3** filters applied in sequence have a total of $\mathbf{2 \times 3 \times 3 = 18}$ learnable parameters, whereas a **single 5x5** filter has $\mathbf{5 \times 5 = 25}$ learnable weights. So, **two 3x3** filters are **cheaper**.



Another advantage is that a pair of $\mathbf{3x3}$ convolutional layers will involve two applications of the activation function, since each convolutional layer is followed by an activation.

A single $\mathbf{5x5}$ layer has a single activation. The activation function is the only nonlinear part of a neural network and it is probable that the composition of nonlinearities in sequence will be able to express more complex nonlinear representations of the inputs.

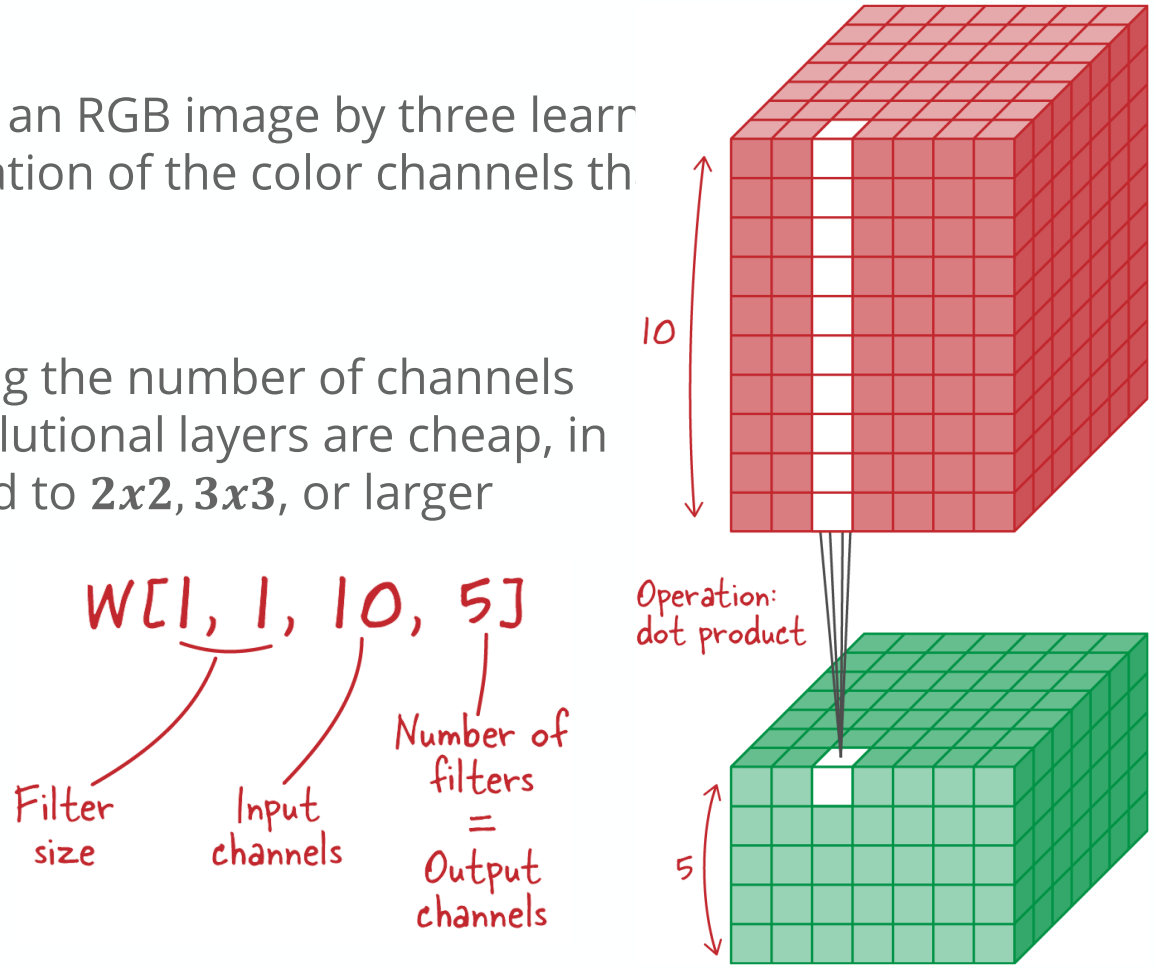
The Quest for Depth: 1x1 Convolutions

Sliding a single-pixel filter across an image sounds silly. It's multiplying the image by a constant. However, on multichannel inputs, with a different weight for each channel, it actually makes sense.

For example, multiplying the three color channels of an RGB image by three learr and then adding them up produces a linear combination of the color channels th be useful.

A $\mathbf{1x1}$ convolutional layer is a useful tool for adjusting the number of channels of the data. The second advantage is that $\mathbf{1x1}$ convolutional layers are cheap, in terms of number of learnable parameters, compared to $\mathbf{2x2}, \mathbf{3x3}$, or larger layers.

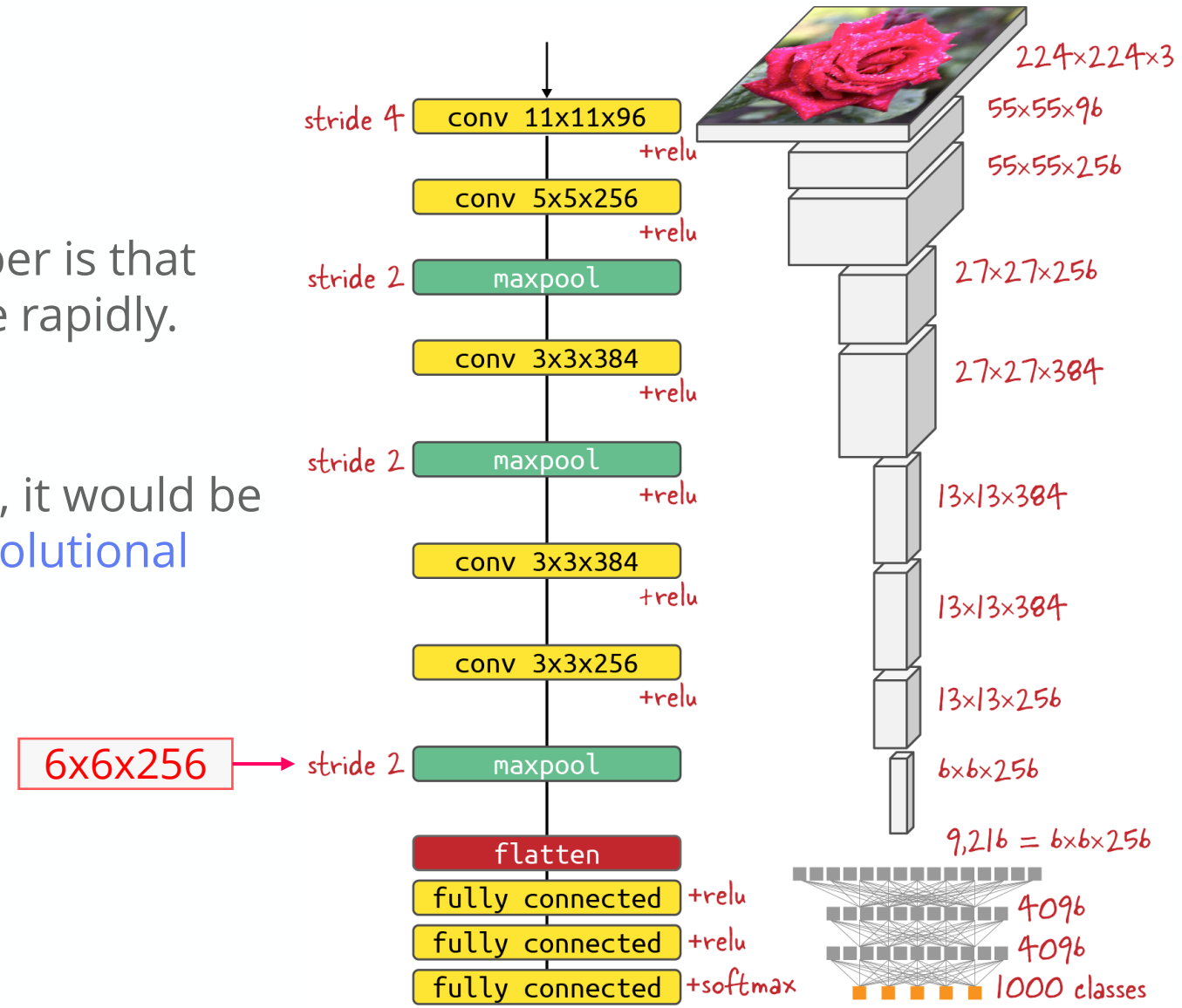
The number of learnable weights is $\mathbf{1} \times \mathbf{1} \times \mathbf{10} \times \mathbf{5} = \mathbf{50}$. A 3×3 layer with the same number of input and output channels would require $\mathbf{3} \times \mathbf{3} \times \mathbf{10} \times \mathbf{5} = \mathbf{450}$ weights.



The Quest for Depth

One of the problems with going deeper is that the spatial resolution decreases quite rapidly.

For instance, in the case of ImageNet, it would be impossible to have more than 8 convolutional layers in this way.

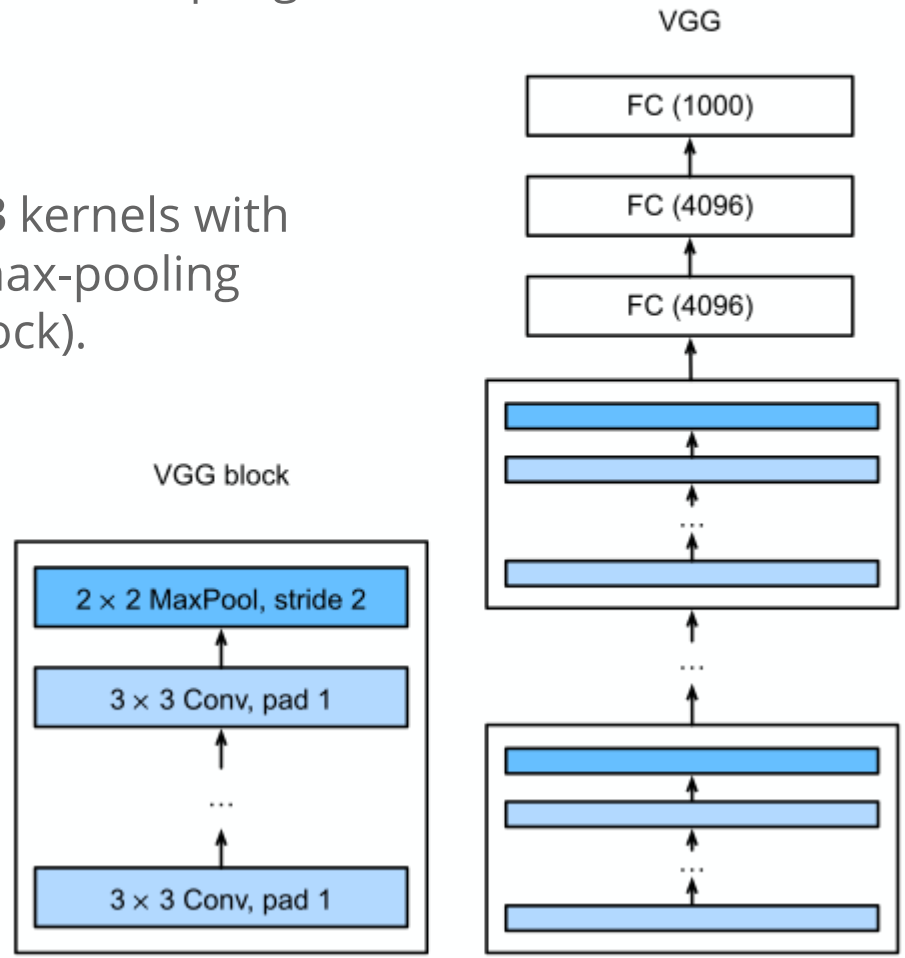


VGGNet

VGG Block

The key idea was to use multiple convolutions in between down-sampling via max-pooling in the form of a block.

VGG block consists of a sequence of convolutions with **3×3** kernels with padding of **1** (keeping height and width) followed by a **2×2** max-pooling layer with stride of **2** (halving height and width after each block).

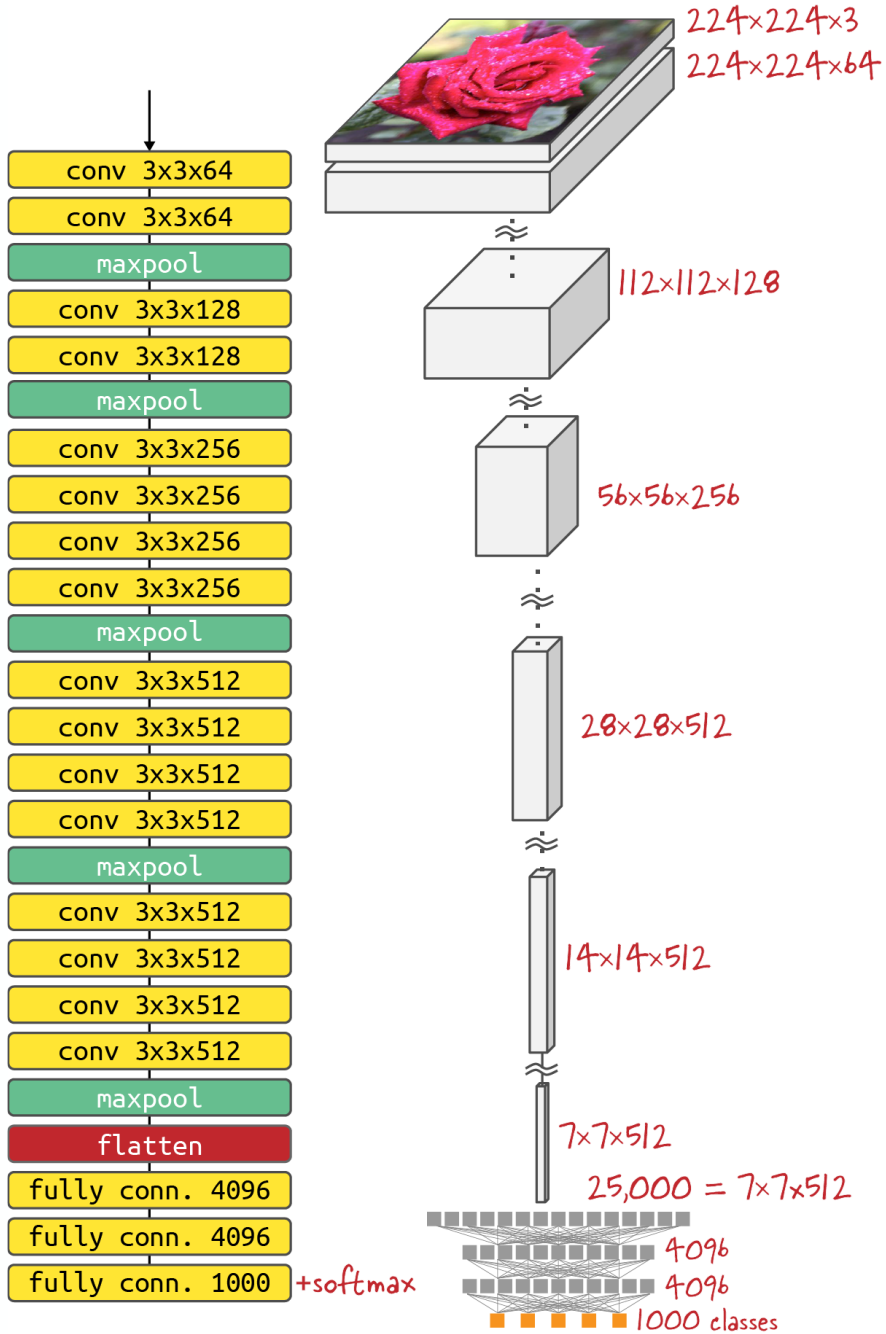


Networks Using Blocks (VGG)

VGG19 is composed of 16 convolutional layers, 5 max-pooling layers, and 3 fully connected layers.

Therefore, the number of layers having tunable parameters is 19 (16 convolutional layers and 3 fully connected layers).

Model	Parameters (excl. classification head)	ImageNet accuracy
VGG19	20M	71%



VGG Networks

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 2: **Number of parameters** (in millions).

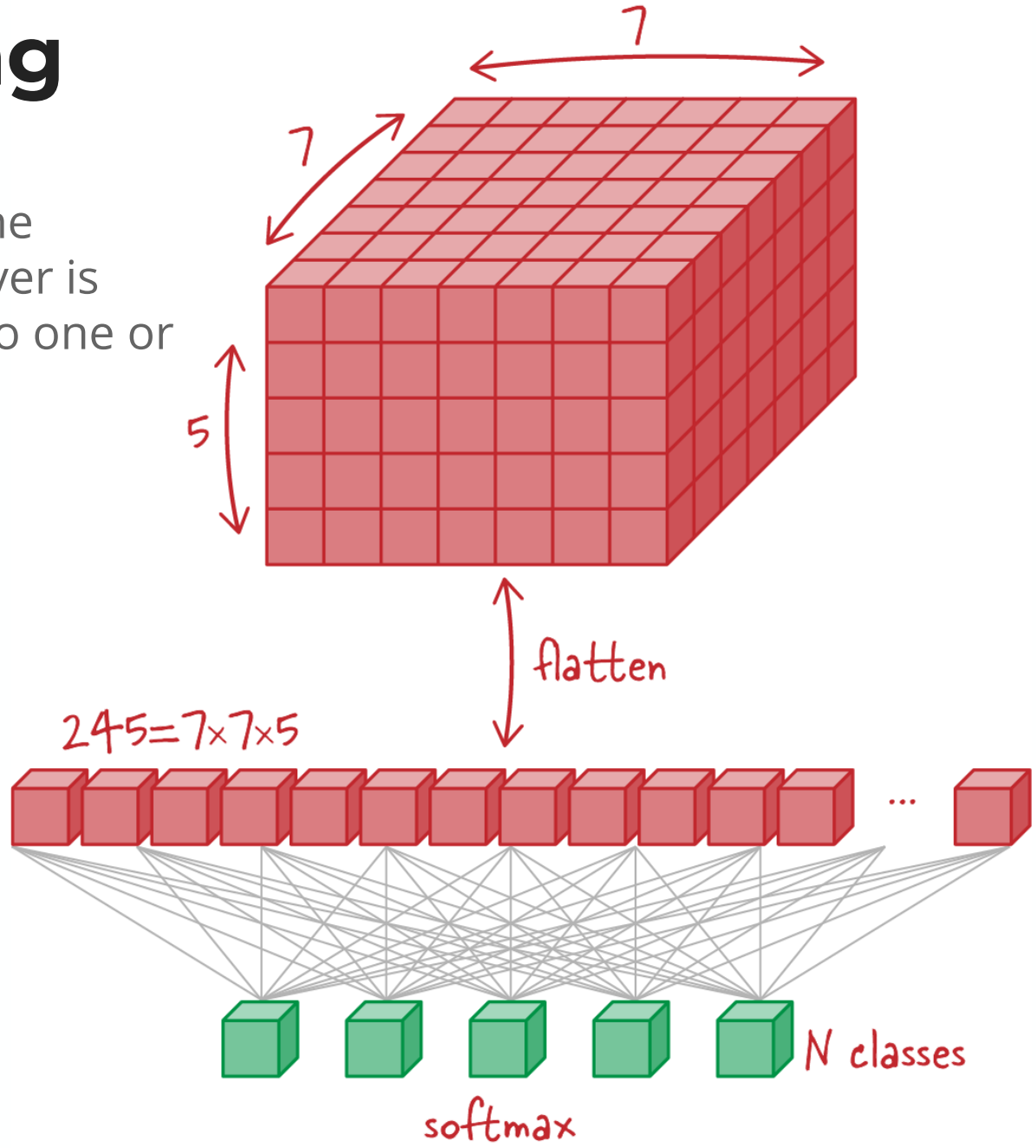
Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

Global Average Pooling

In both the AlexNet and VGG19 architectures, the feature map output by the last convolutional layer is turned into a vector (flattened) and then fed into one or more fully connected layers.

The goal is to end on a softmax-activated fully connected layer with exactly as many neurons as classes in the classification problem at hand.

This fully connected layer has ***input * outputs*** weights, which tends to be a lot.

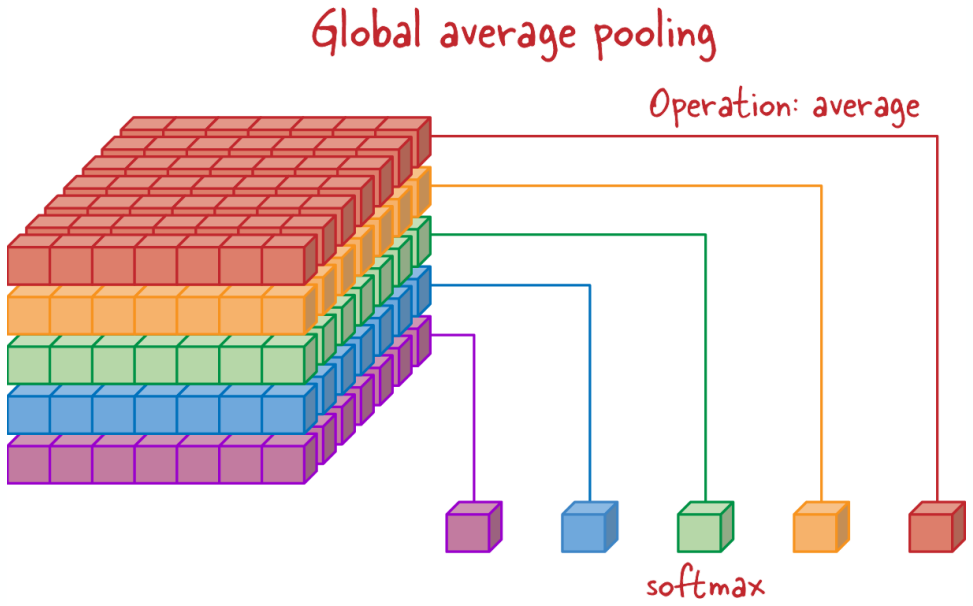


Global Average Pooling

If the only goal is to obtain N values to feed an $N-way$ softmax function, there is an easy way to achieve that: adjust the convolutional stack so that it ends on a final feature map with exactly N channels and simply average the values in each channel.

This is known as global average pooling. Global average pooling involves no learnable weights, so from this perspective it's cheap.

Note: Averaging removes a lot of the positional information present in the channels. That might or might not be a good thing depending on the application.



Modular Architectures

A straight succession of convolutional and pooling layers is enough to build a basic convolutional neural network.

However, to further increase prediction accuracy, researchers designed more complex building blocks, or modules, often given arcane names such as ***"Inception modules", "residual blocks",*** or ***"inverted residual bottlenecks"*** and then assembled them into complete convolutional architectures.

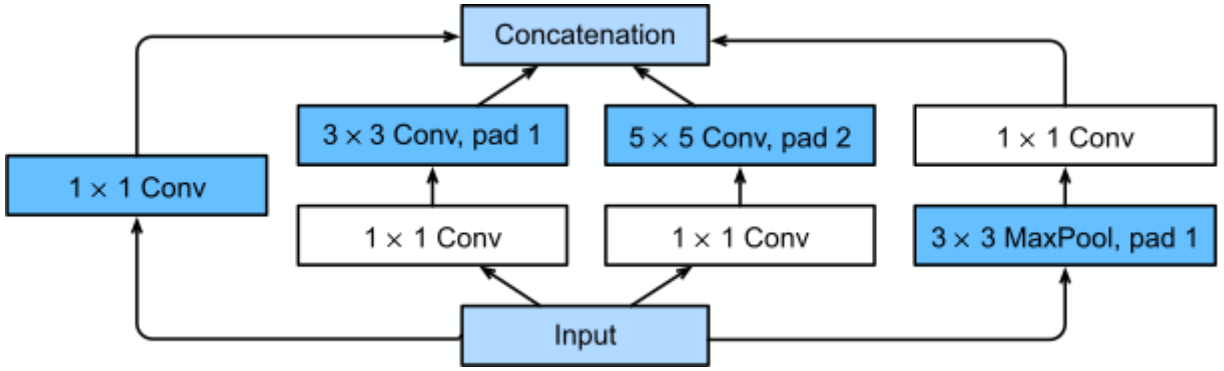
Having higher-level building blocks also made it easier to create automated algorithms to search for better architectures, as we will see in the section on neural architecture search.


InceptionNet

Inception Blocks

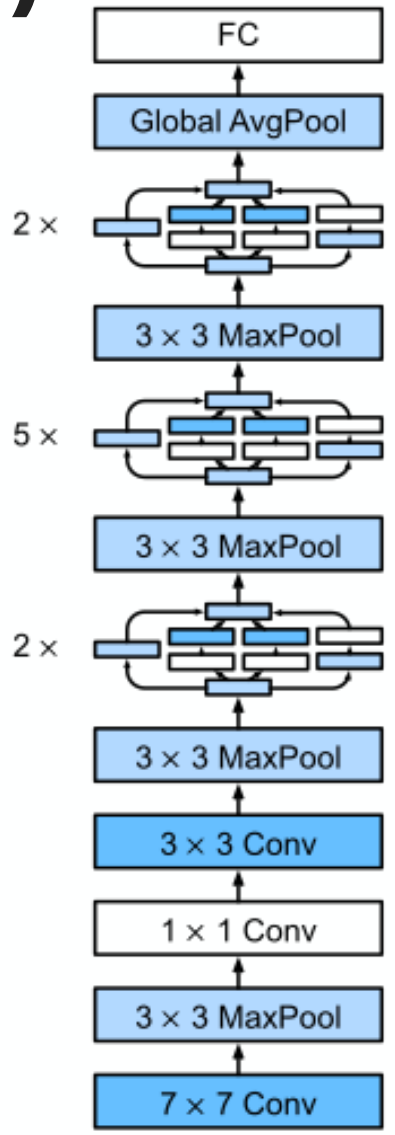
Inception block consists of four parallel paths at which convolution layers with different kernel sizes:

- The first path uses a convolutional layer with a window size of 1 × 1.
- In the second and the third paths, a convolutional layer of size 1 × 1 is used before applying two expensive 3 × 3 and 5 × 5 convolutions. The 1×1 convolution helps to reduce the number of filter channels, thus reducing the model complexity.
- The fourth path uses a max-pooling layer to reduce the resolution of the input, and it is followed by a 1 × 1 convolutional layer to reduce the dimension.

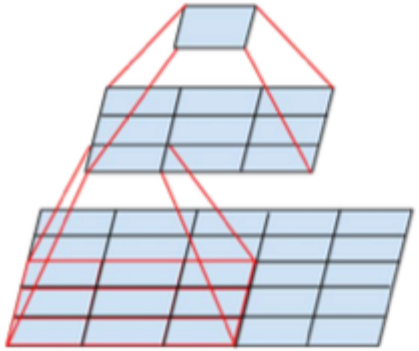


Multi-Branch Networks (GoogLeNet)

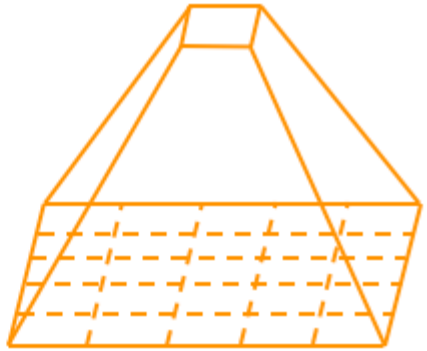
- The input size image is 224 × 224.
- There are nine Inception blocks in this network.
- There are four max-pooling layers outside the Inception blocks, in which two layers are located between blocks 3–4 and block 7–8. These max-pooling layers help to reduce the size of the input data, thus reduce the model complexity as well as the computational cost.
- This network uses the idea of using an average pooling layer, which helps to improve the model performance and reduce overfitting.
- A dropout layer (with 40%) is utilized before the linear layer. This is also an efficient regularization method to reduce the overfitting phenomena.
- The output layer uses the softmax activation function to give 1000 outputs which are corresponding to the number of categories in the ImageNet dataset.



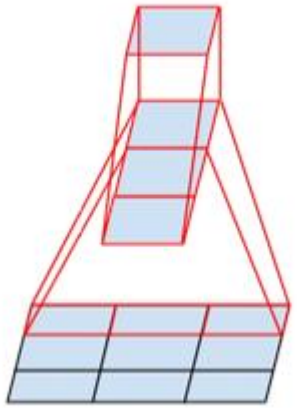
Modified GoogLeNet



two successive
3x3 convolutions



5x5 convolution



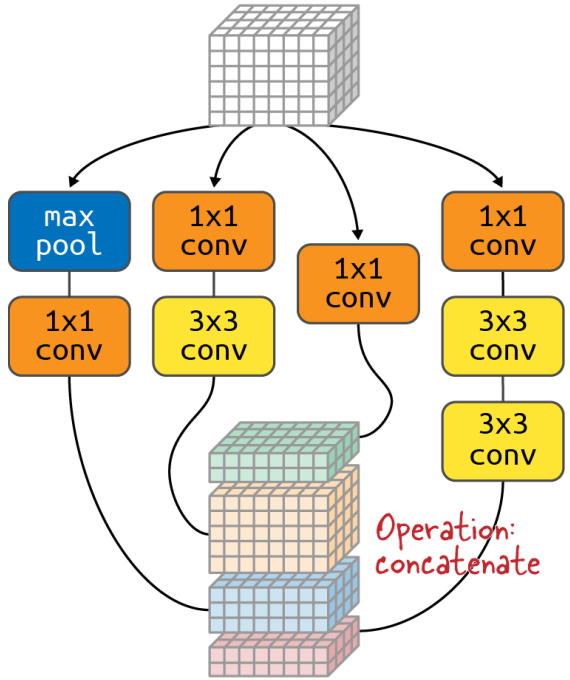
For instance, the successive application of **two 3×3** convolutions touches the **same** pixels as a single **5×5** convolution does.

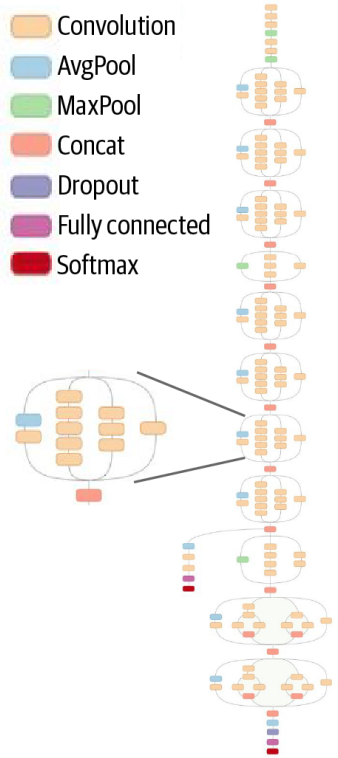
It was also shown that **3x3** convolutions could be further deconstructed into successive **3x1** and **1x3** convolutions.

Modified GoogLeNet

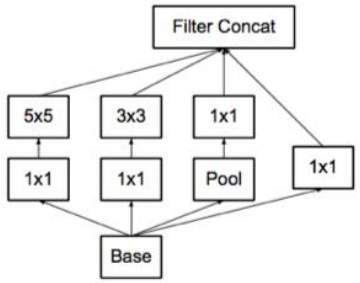
When lining up the convolutional and pooling layers in a neural network, the designer has multiple choices, and the best one is not obvious. Instead of relying on guesswork and experimentation, why not build multiple options into the network itself and let it learn which one is the best?

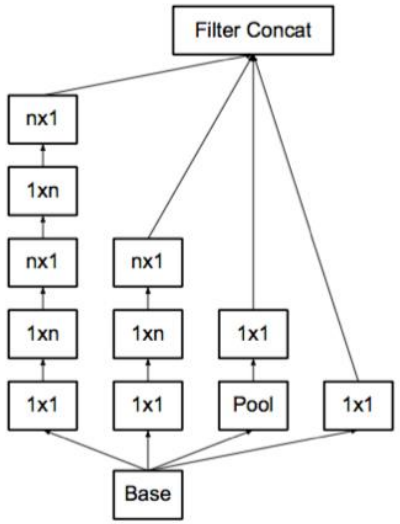
Model	Parameters (excl. classification head)	ImageNet accuracy
InceptionV3	22M	78%

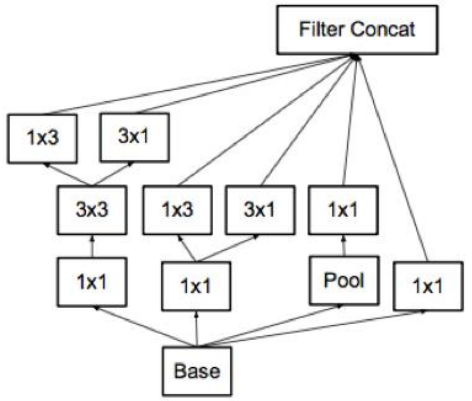


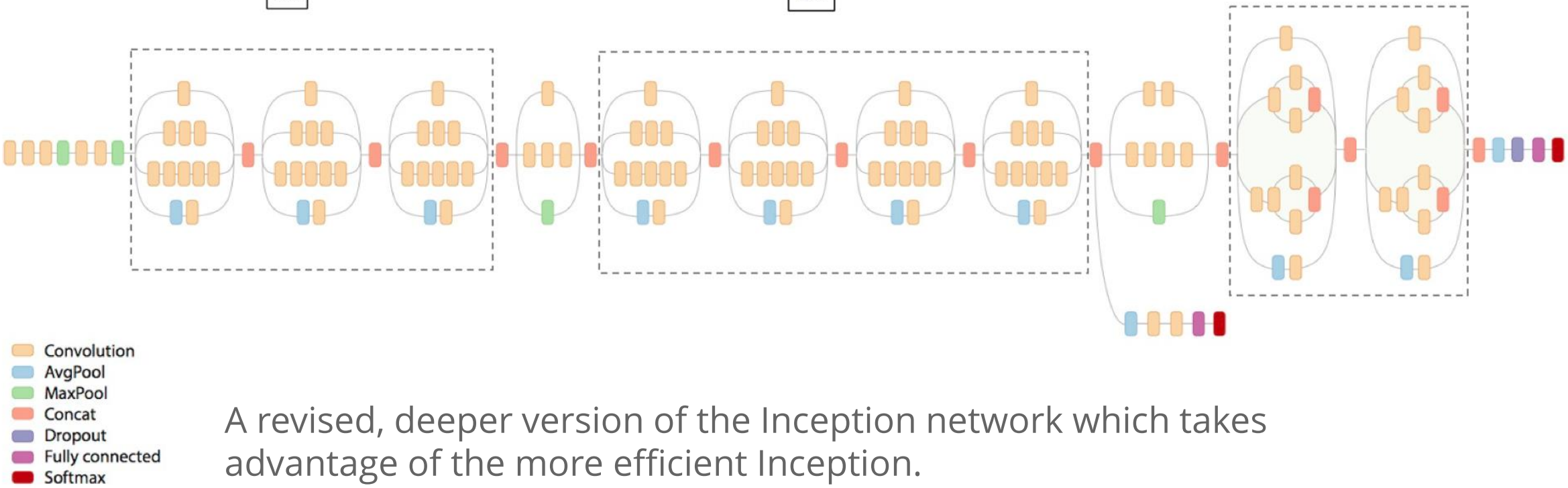


Deeper GoogleNet



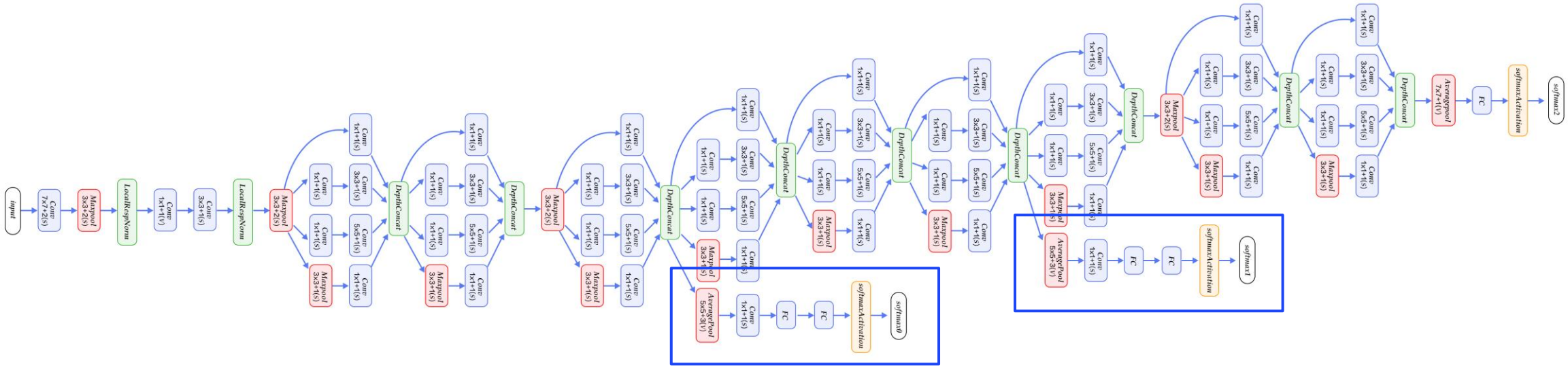






A revised, deeper version of the Inception network which takes advantage of the more efficient Inception.

GoogLeNet (Auxiliary Loss)



link...

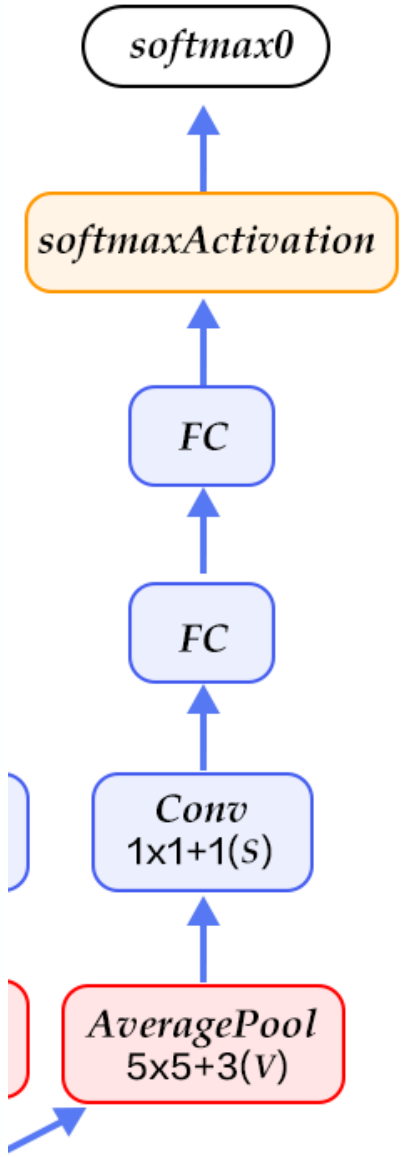
GoogLeNet (Auxiliary Loss)

In order to improve overall network performance, two auxiliary outputs are added throughout the network.

It was later discovered that the earliest auxiliary output had no discernible effect on the final quality of the network.

The addition of auxiliary outputs primarily benefited the end performance of the model, converging at a slightly better value than the same network architecture without an auxiliary branch.

It is believed the addition of auxiliary outputs had a regularizing effect on the network.




SqueezeNet

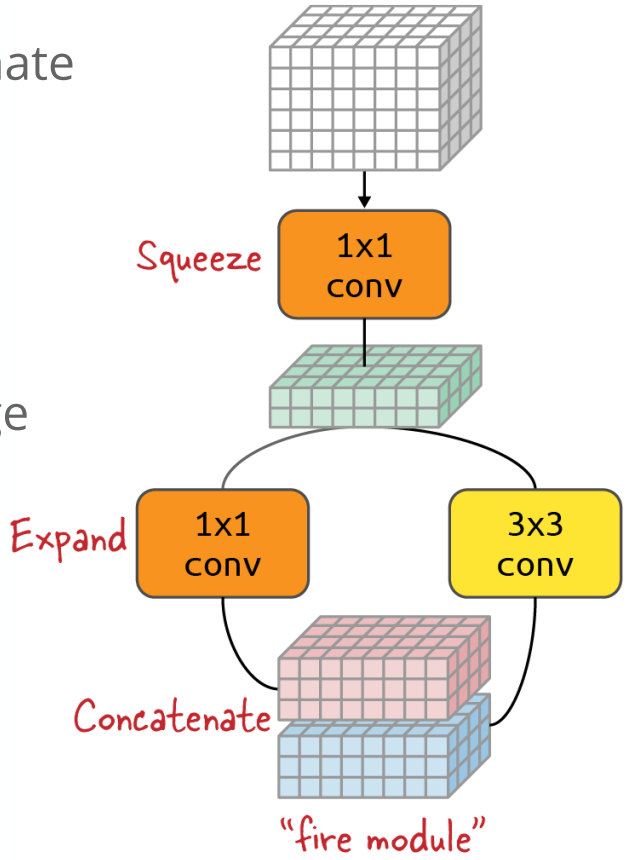
SqueezeNet

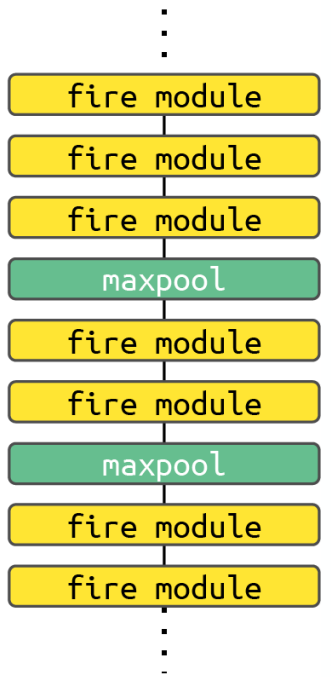
The idea of modules was simplified by the SqueezeNet architecture, which kept the basic principle of offering multiple paths for the network to choose from but streamlined the modules themselves into their simplest expression. The SqueezeNet paper calls them ***"fire modules"***.

The modules used in the SqueezeNet architecture alternate a contraction stage, where the number of channels is reduced by a $\mathbf{1x1}$ convolution, with an expansion stage where the number of channels is increased again.

To save on weight count, SqueezeNet uses global average pooling for the last layer.

Also, two out of the three convolutional layers in each module are 1x1 convolutions, which saves on learnable weights.

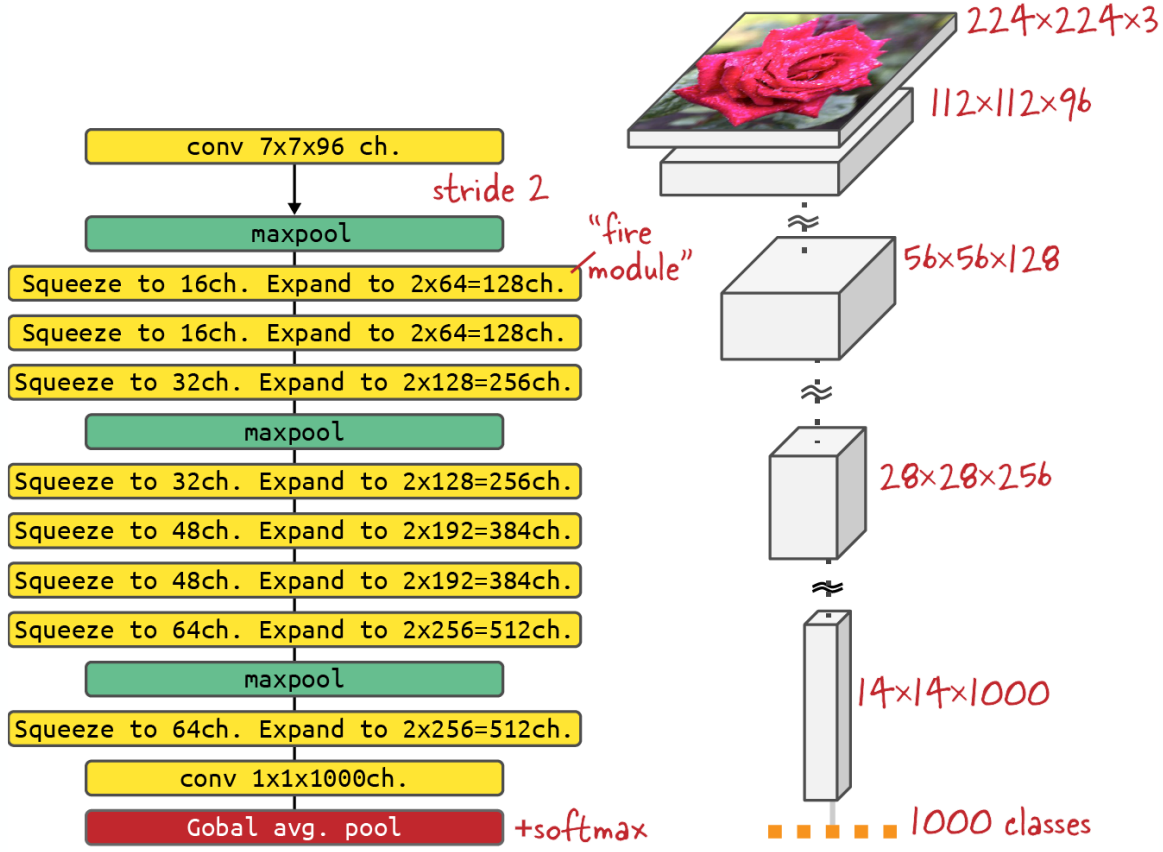




SqueezeNet

The idea of modules was simplified by the SqueezeNet architecture, which kept the basic principle of offering multiple paths for the network to choose from but streamlined the modules themselves into their simplest expression. The SqueezeNet paper calls them ***"fire modules"***.

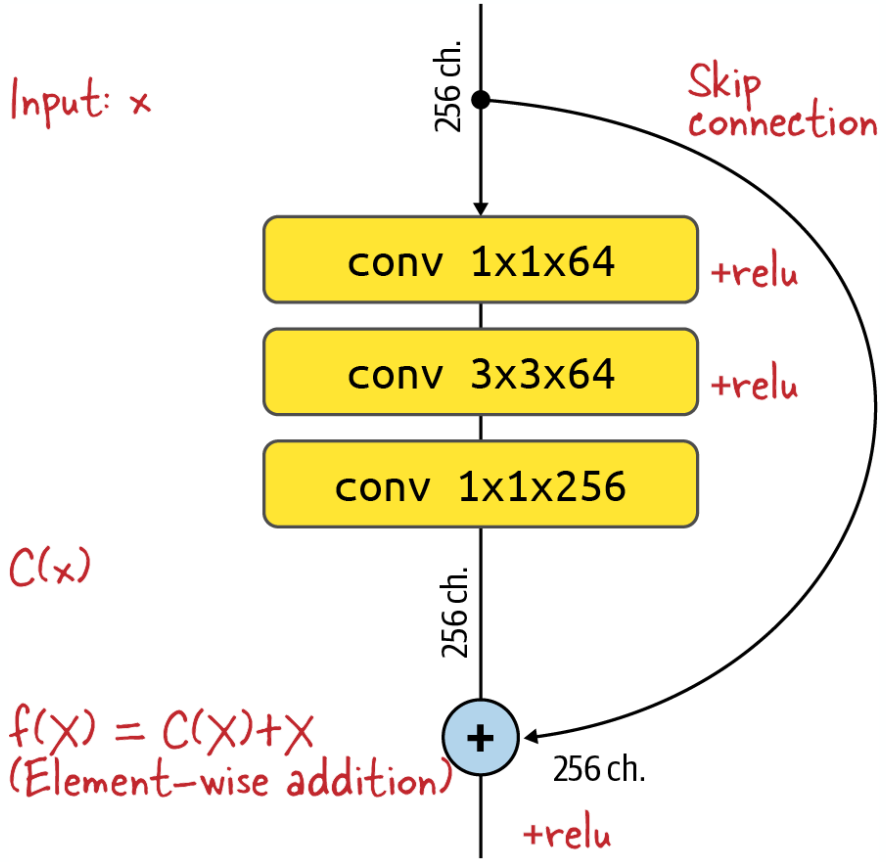
Model	Parameters (excl. classification head)	ImageNet accuracy
SqueezeNet, 24 layers	2.7M	-
SqueezeNet, 18 layers	1.2M	56%



ResNet

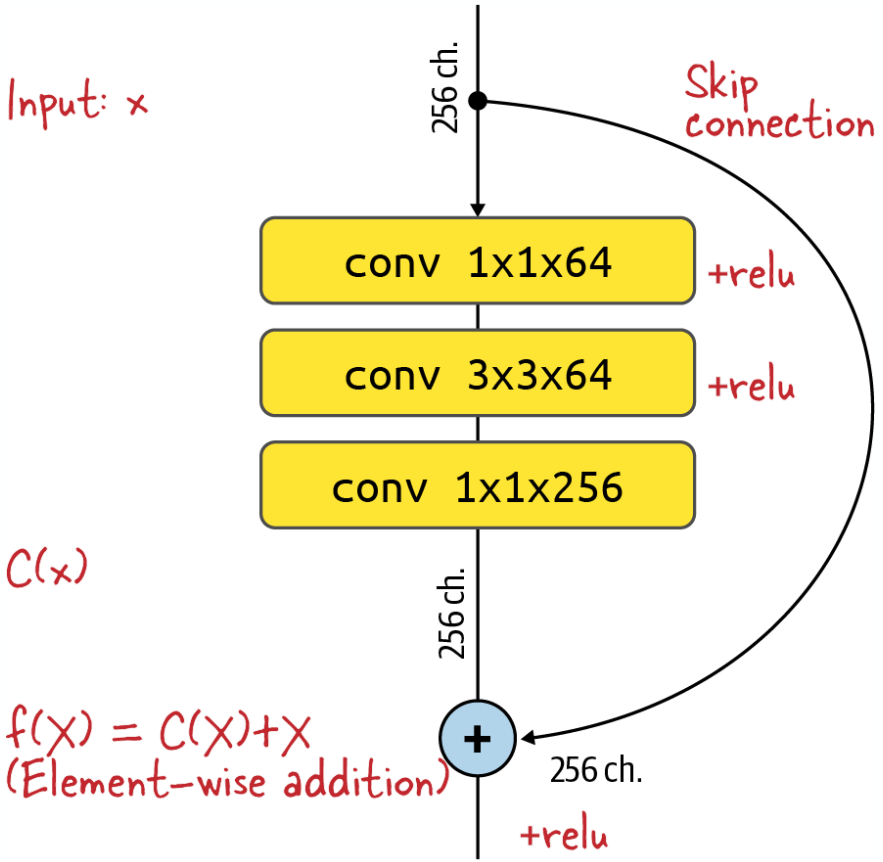
Residual Blocks

- During training, a neural network sees what error (or loss) it is making and tries to minimize this error by adjusting its internal weights. It is guided in this by the first derivative (or gradient) of the error.
- Unfortunately, with many layers, the gradients tend to be spread too thin across all layers and the network converges slowly or not at all.
- This is a common challenge with very deep neural networks—they tend to converge badly because of vanishing or exploding gradient problems.



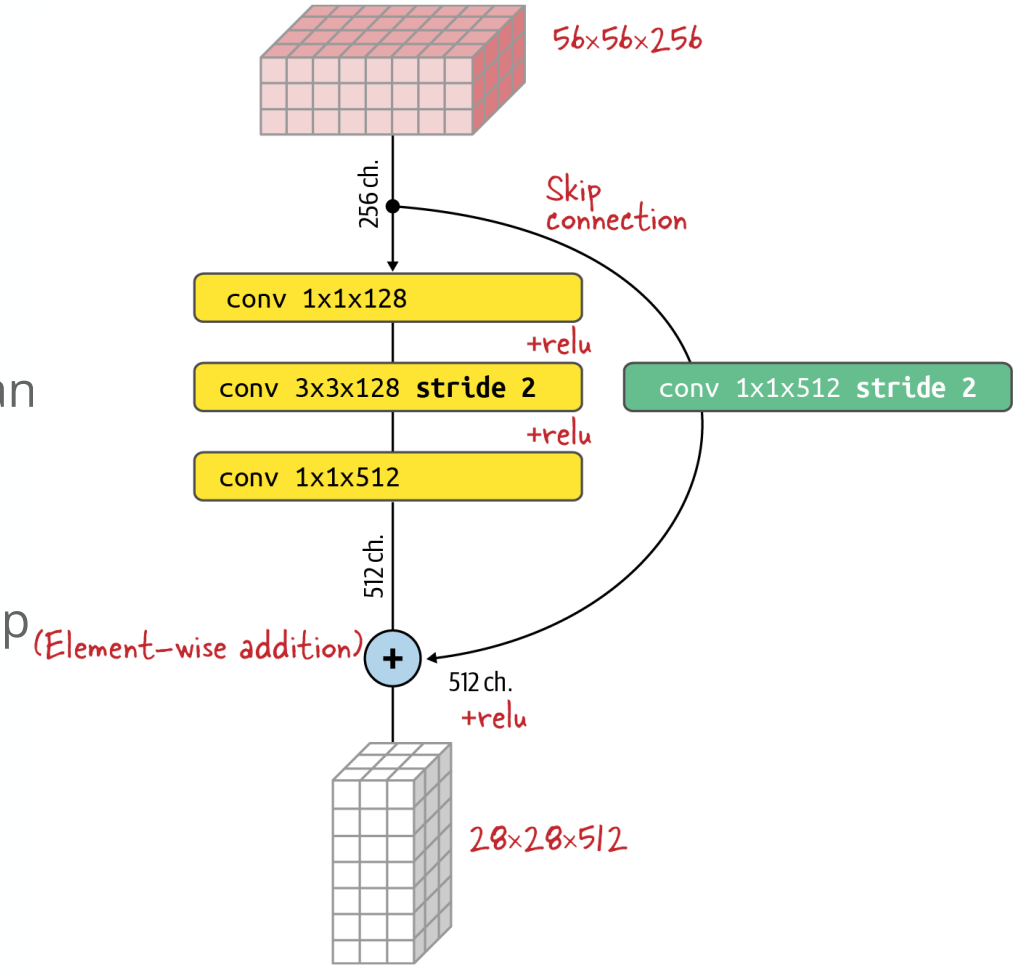
Residual Blocks

- ResNet tried to remedy this by adding ***skip connections*** alongside its convolutional layers.
- Skip connections convey the signal as is, then recombine it with the data that has been transformed by one or more convolutional layers.
- The combining operation is a simple element-by-element addition.
- An obvious limitation is that the element-wise addition can only work if the dimensions of the data remain unchanged.
- The sequence of layers that is straddled by a skip connection (called ***a residual block***) must preserve the height, the width, and the number of channels of the data.



Residual Blocks

- When size adjustments are needed, a different kind of residual block is used.
- Different numbers of channels can be matched in the skip connection by using a 1x1 convolution instead of an identity.
- Height and width adjustments are obtained by using a stride of 2 both in the convolutional path and in the skip connection.

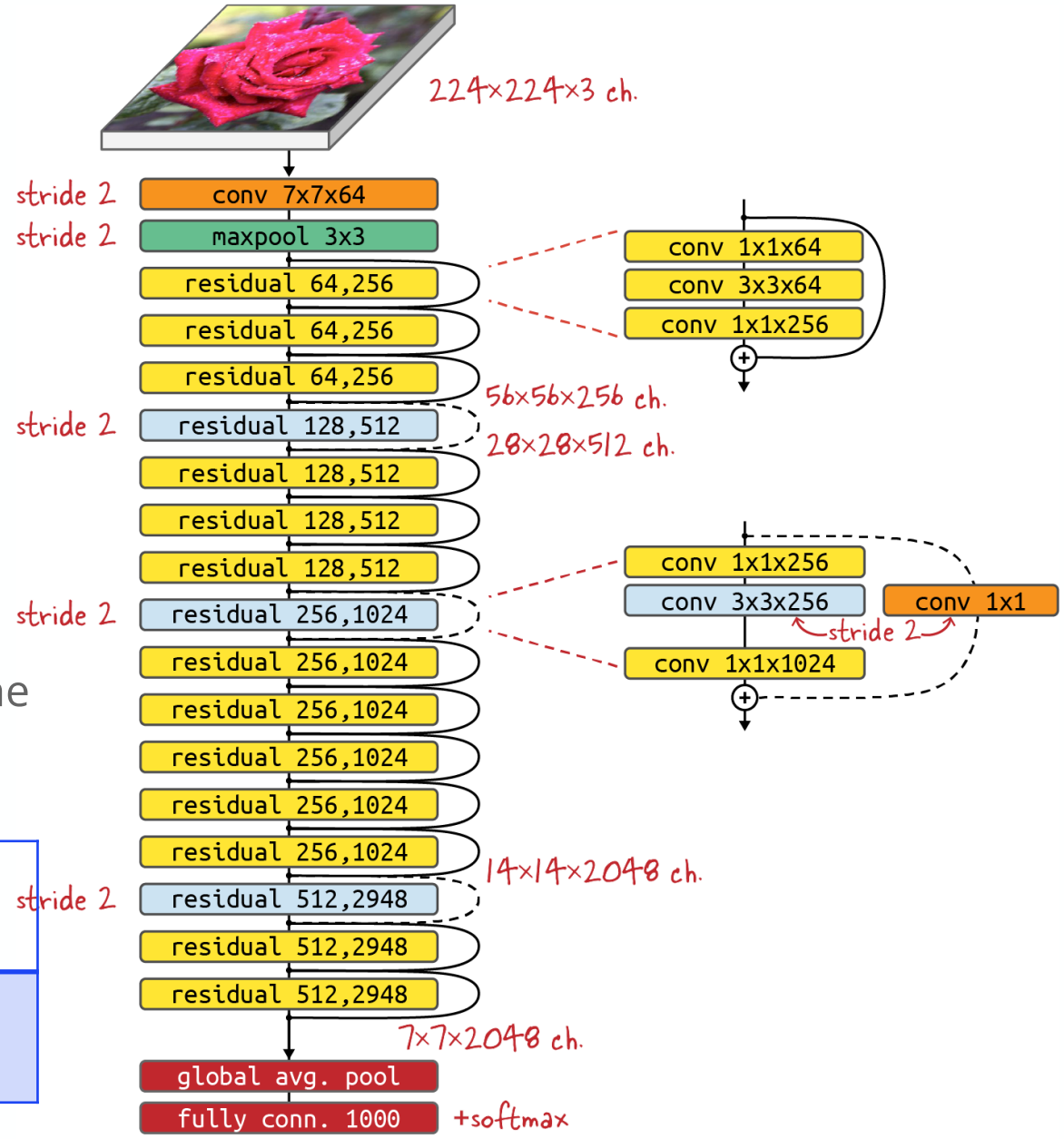


ResNet Model

- The first two layers of ResNet are the same as those of the GoogLeNet we described before: the **7×7** convolutional layer with 64 output channels and a stride of 2 is followed by the **3×3** max-pooling layer with a stride of 2.
- The difference is the **batch normalization** layer added after each convolutional layer in ResNet.

- Till SqueezeNet we can build **24 layers** only, with the skip connection we can build **150 layers**!

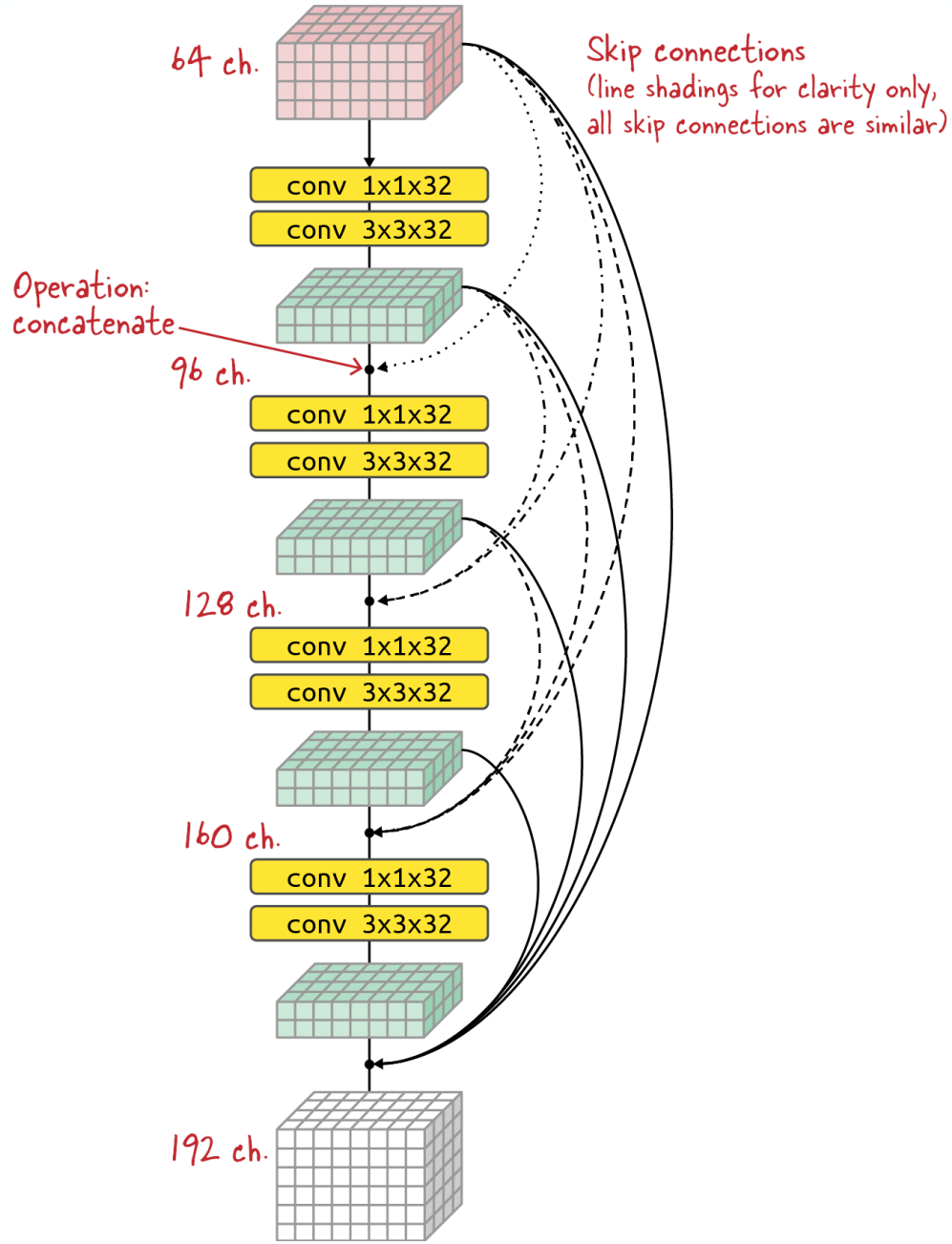
Model	Parameters (excl. classification head)	ImageNet accuracy
ResNet50	23M	75%



DenseNet

Dense Blocks

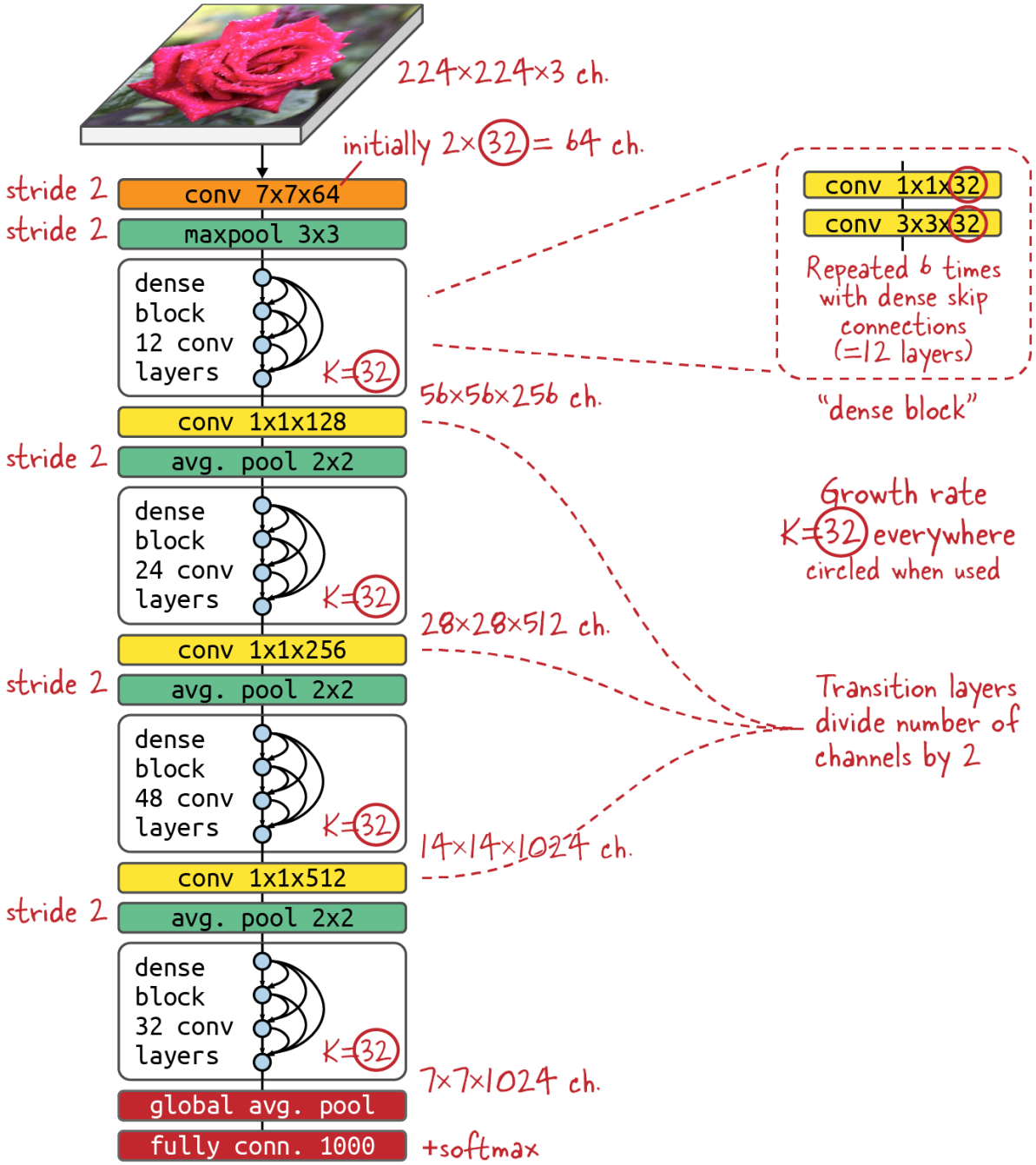
- Dense blocks are the basic building blocks of DenseNet.
- In a dense block, convolutions are grouped in pairs, with each pair of convolutions receiving as its input the output of all previous convolution pairs.
- In the dense block, data is combined by concatenating it ***channel-wise***.
- Channel-wise concatenation only works if the height and width dimensions of the data are the same, so convolutions in a dense block are all of stride 1 and do not change these dimensions.
- Pooling layers will have to be inserted between dense blocks.



DenseNet121 Model

- Dense blocks and pooling layers are interleaved to create a full DenseNet network.
- The use of shallow convolutional layers (K=32, for example) is a characteristic feature of DenseNet.
- DenseNet can afford to use shallow convolutions because each convolutional layer sees all previously computed features.

Model	Parameters (excl. classification head)	ImageNet accuracy
DenseNet201	18M	77%
DenseNet121	7M	75%



Thank You!