# Convolutional Neural Networks

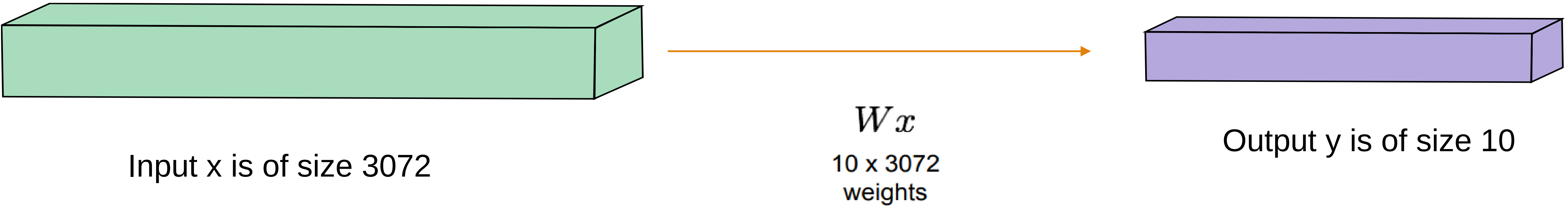
MUSTAFA HAJIJ

## Purpose

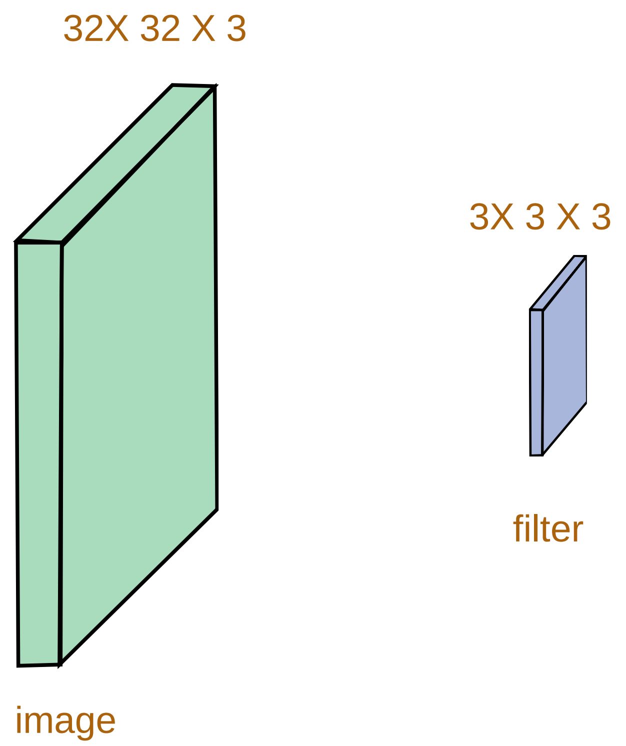
- Basics of convolution layer
- Convolutional neural networks
- Representation learning for CNNs
- End-to-end training
- Pooling
- Inductive bias
- Manifold hypothesis
- Study cases of CNNs
- Transfer learning

## Convolution

Recall first the fully connected layer on image. Say that we have 32X32X3 size image. We flatten this image to a vector 3072.



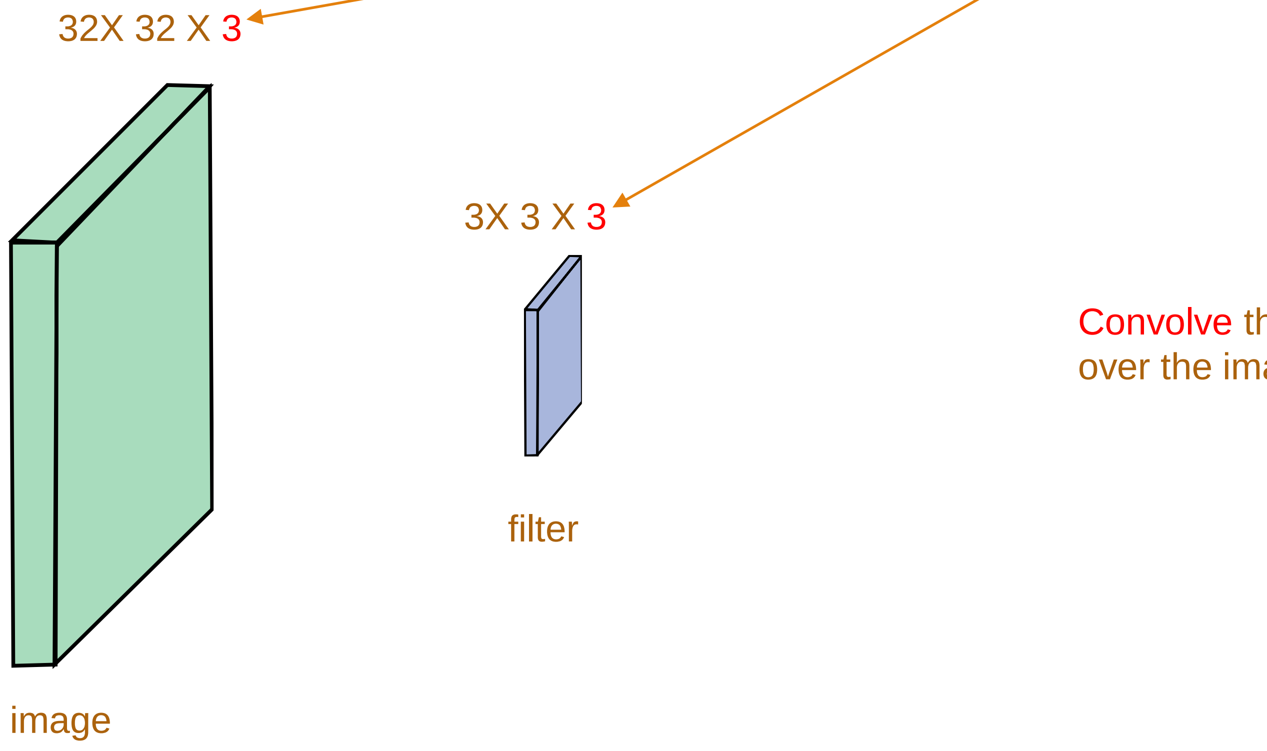
## Convolution



Convolve the filter with the image means sliding the filter, over the image spatially and compute the dot product

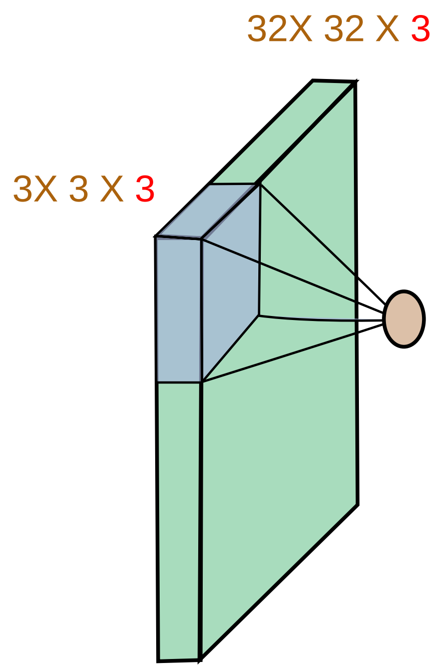
## Convolution

Filters always extends over the full depth of the convolved image



Convolve the filter with the image means sliding the filter, over the image spatially and compute the dot product

## Convolution

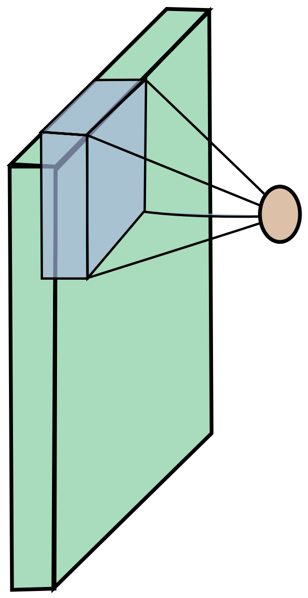


Perform dot product between the
filter and the image (3X3X3 products) + bias

We can write this operation as $w^T x + b$

## Convolution

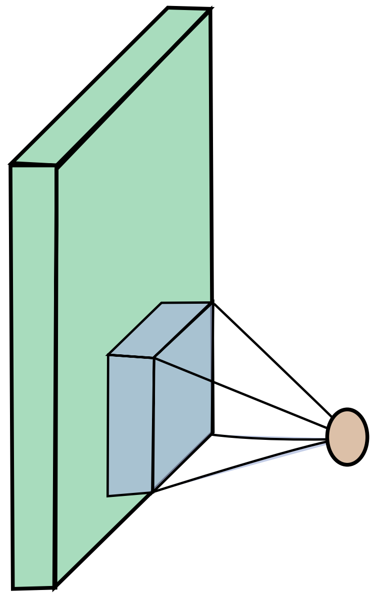
32X 32 X 3



Shift and perform the dot product again

## Convolution

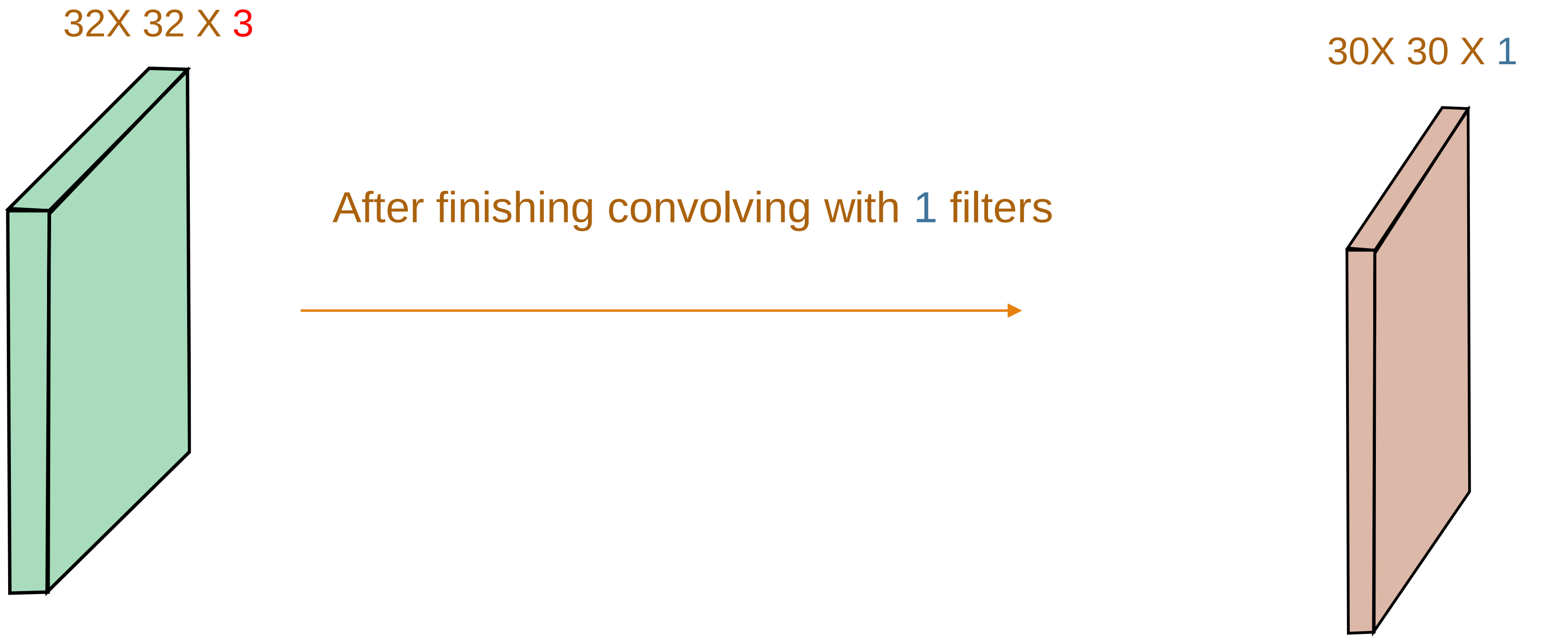
32X 32 X 3



Keep sliding spatially and compute the dot product till
You get to the other bottom right corner

## Convolution

Activation map



Question : Why the volume is of size 30X30X1 ?

## Convolution

Input tensor

filter

Activation map

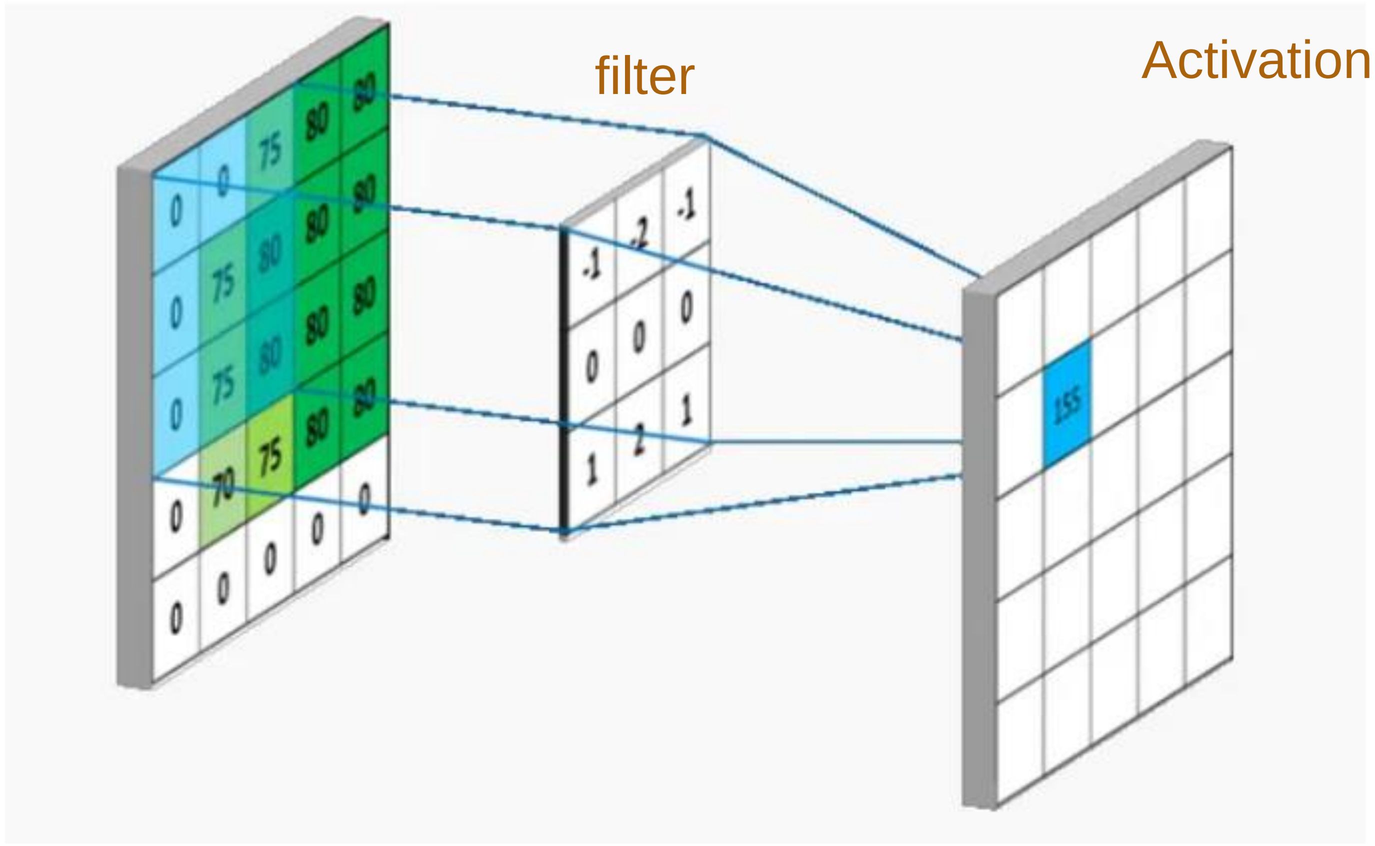
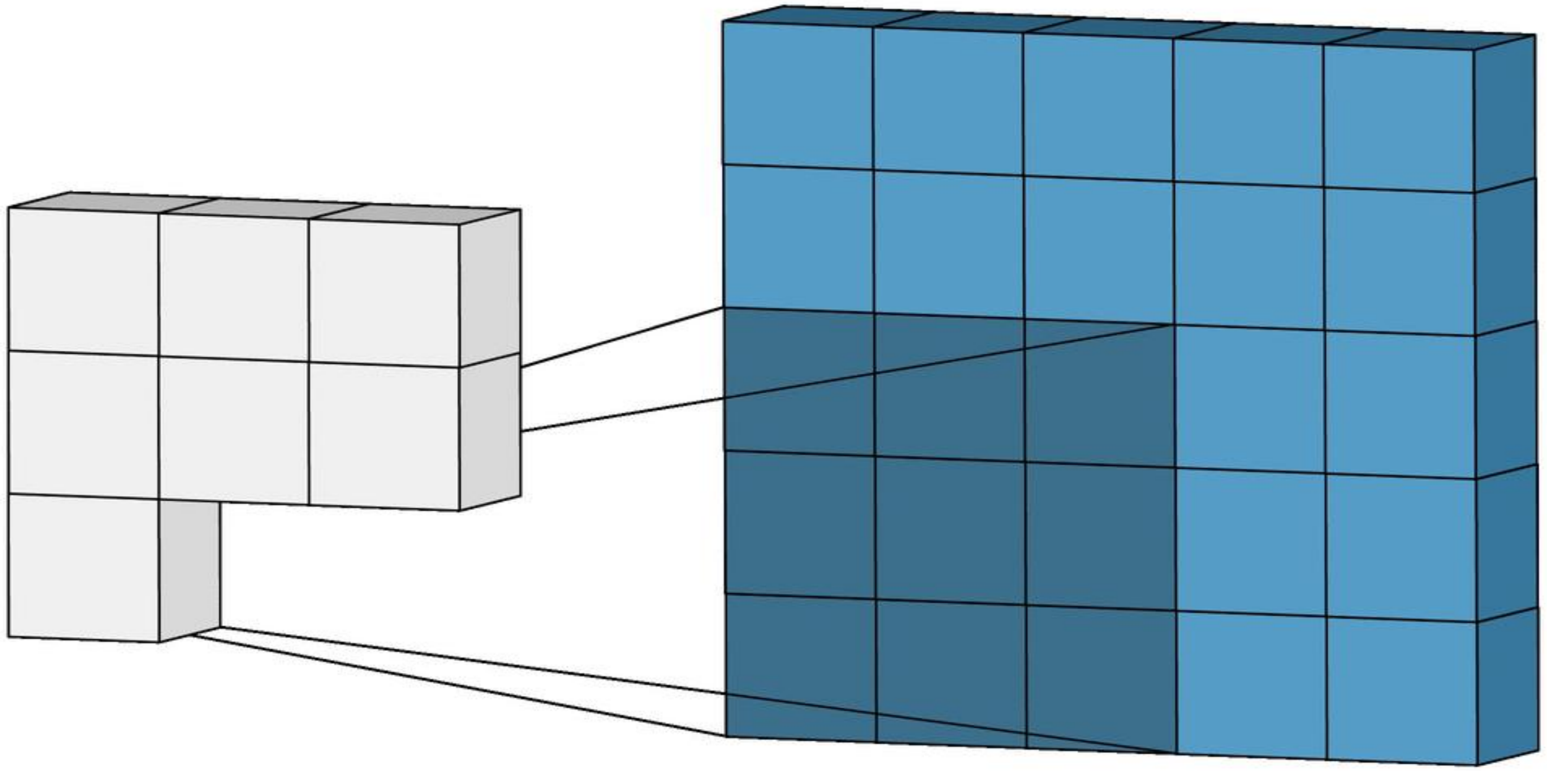


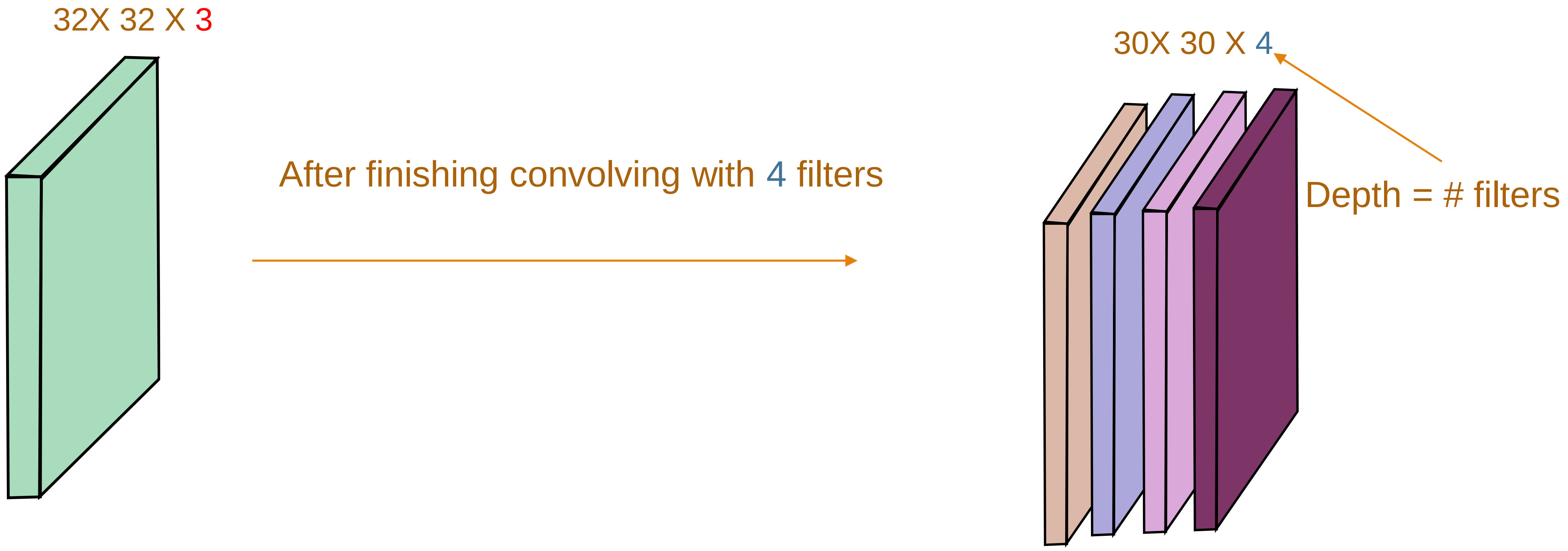
Image source

## Convolution

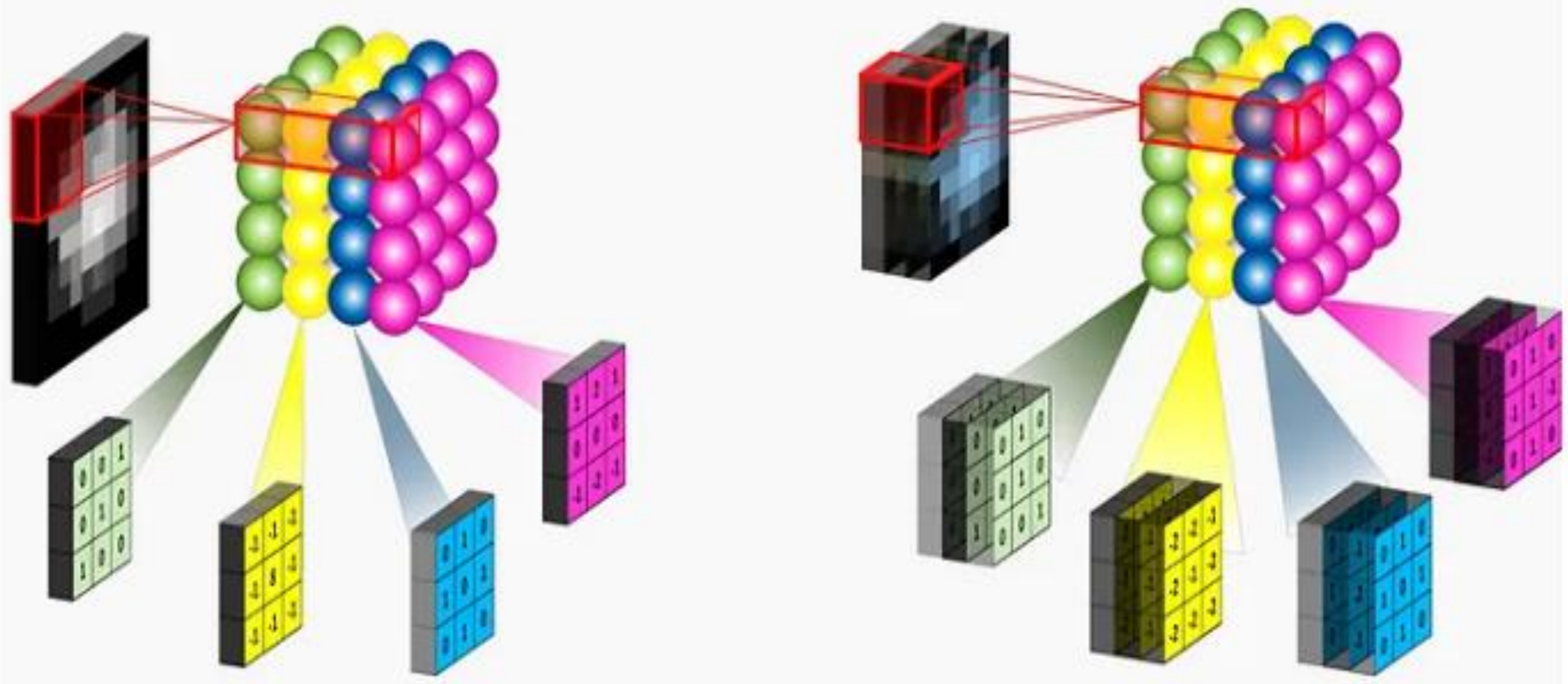


For a vis of the conv operation check the link to view the gif animation

## Convolution



## Convolution



Convolutional layer with four 3x3 filters on a black and white image (just one channel)

Convolutional layer with four 3x3 filters on an RGB image. As you can see, the filters are now cubes, and they are applied on the full depth of the image..

Image source

## Convolution

You can think about the conv operation as a function that maps a 3d volume tensor to a 3d tensor volume.

This operator maps a slice in the input tensor with the size of the filter size to a slice of the output tensor as illustrated in the figure.

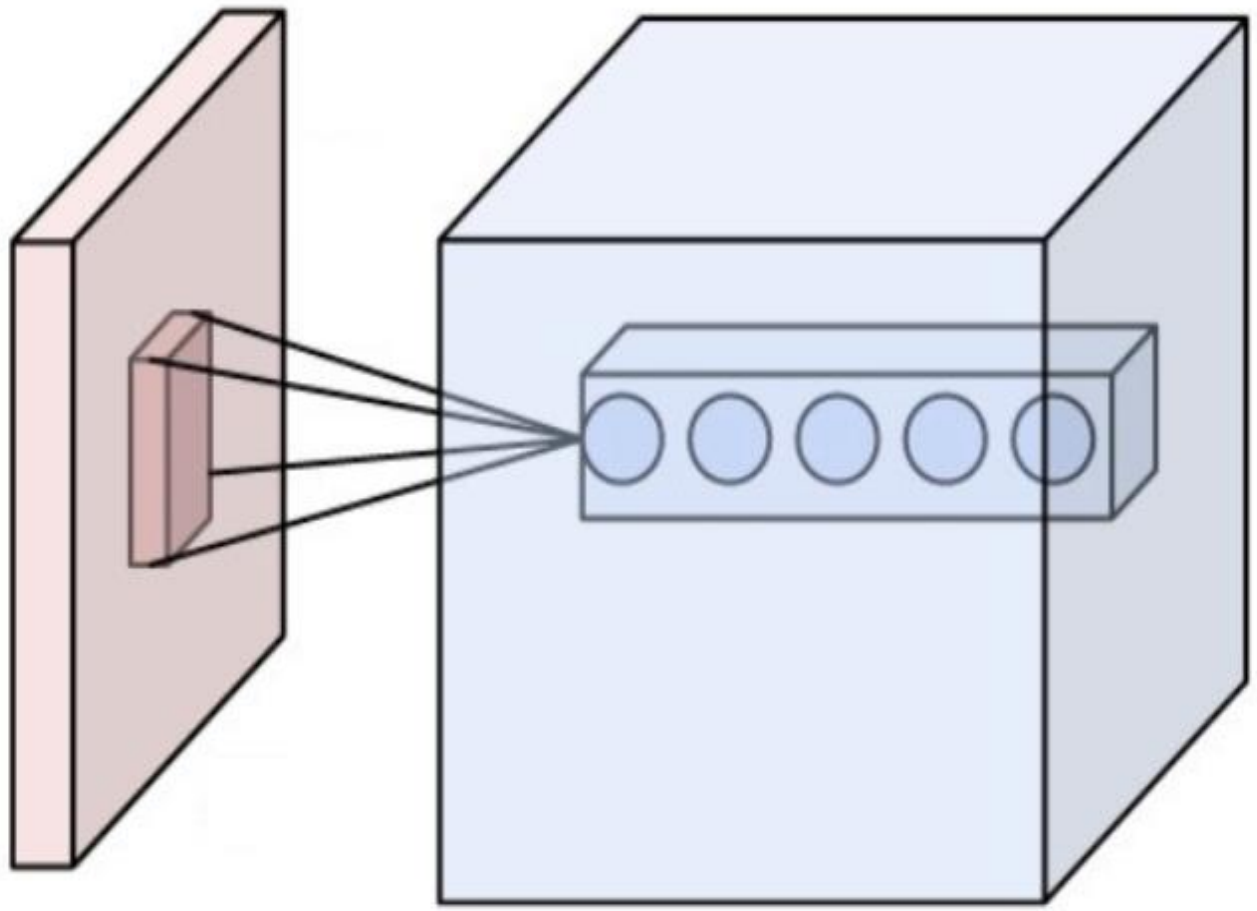
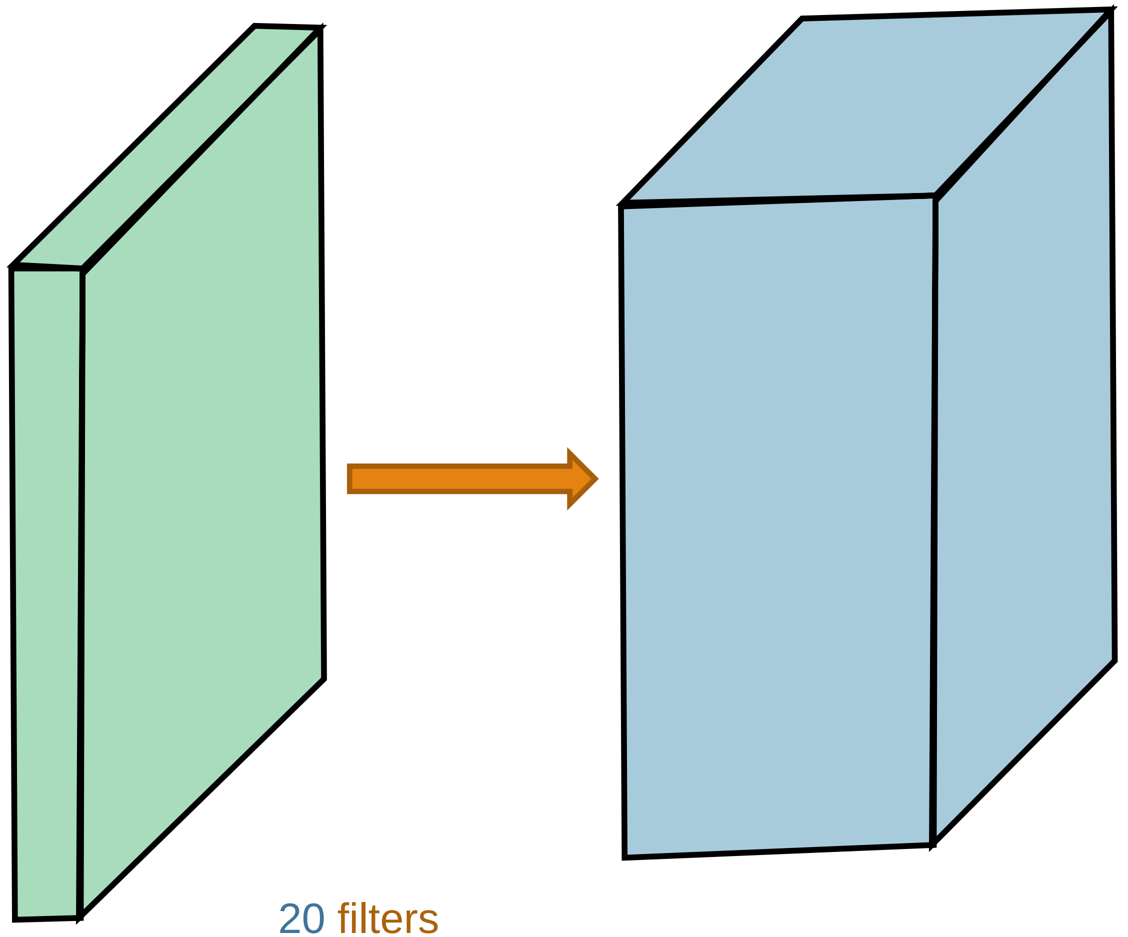


Image source : Convolutional neural network - Wikipedia

## Convolution



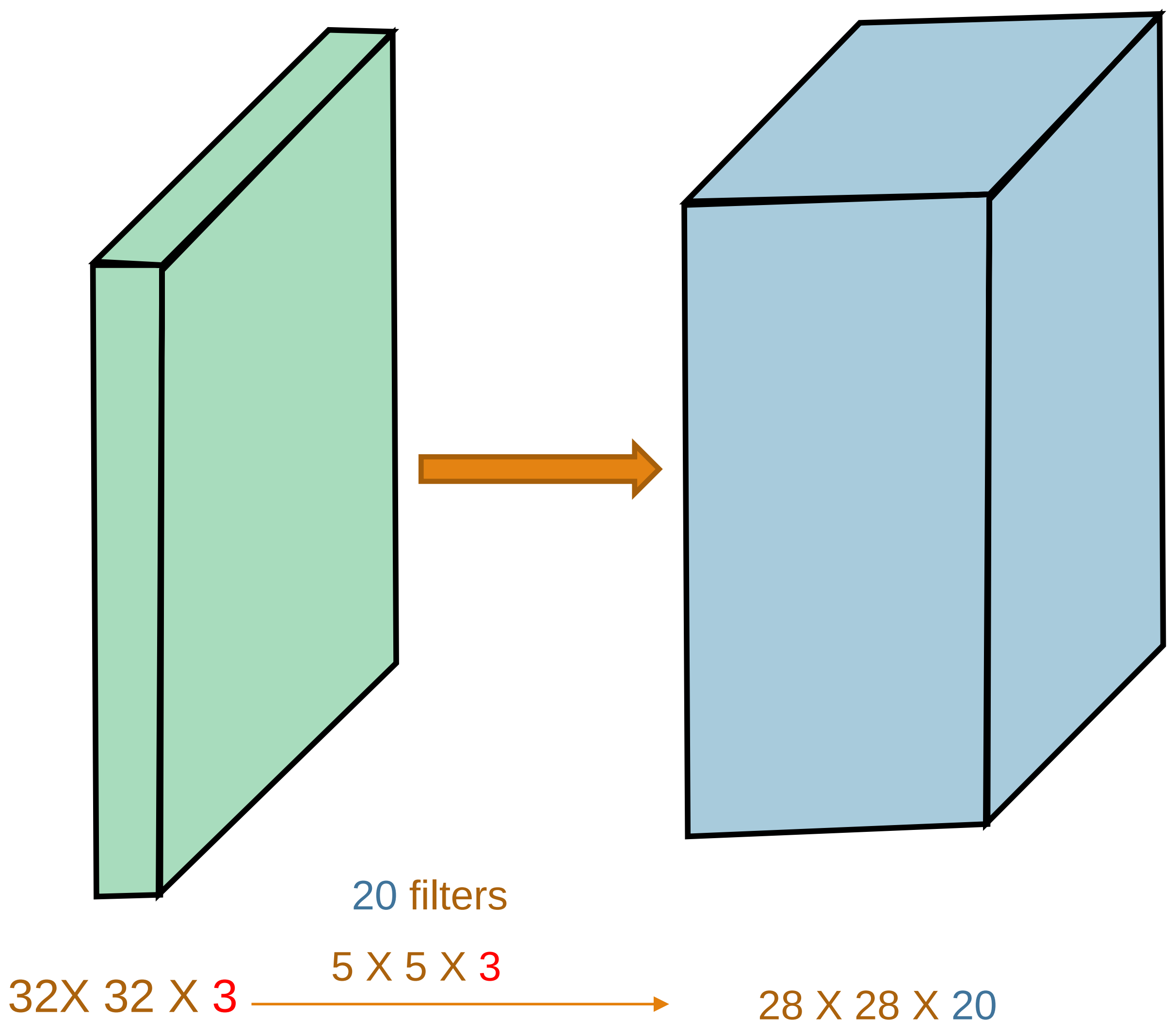
20 filters

5 X 5 X 3

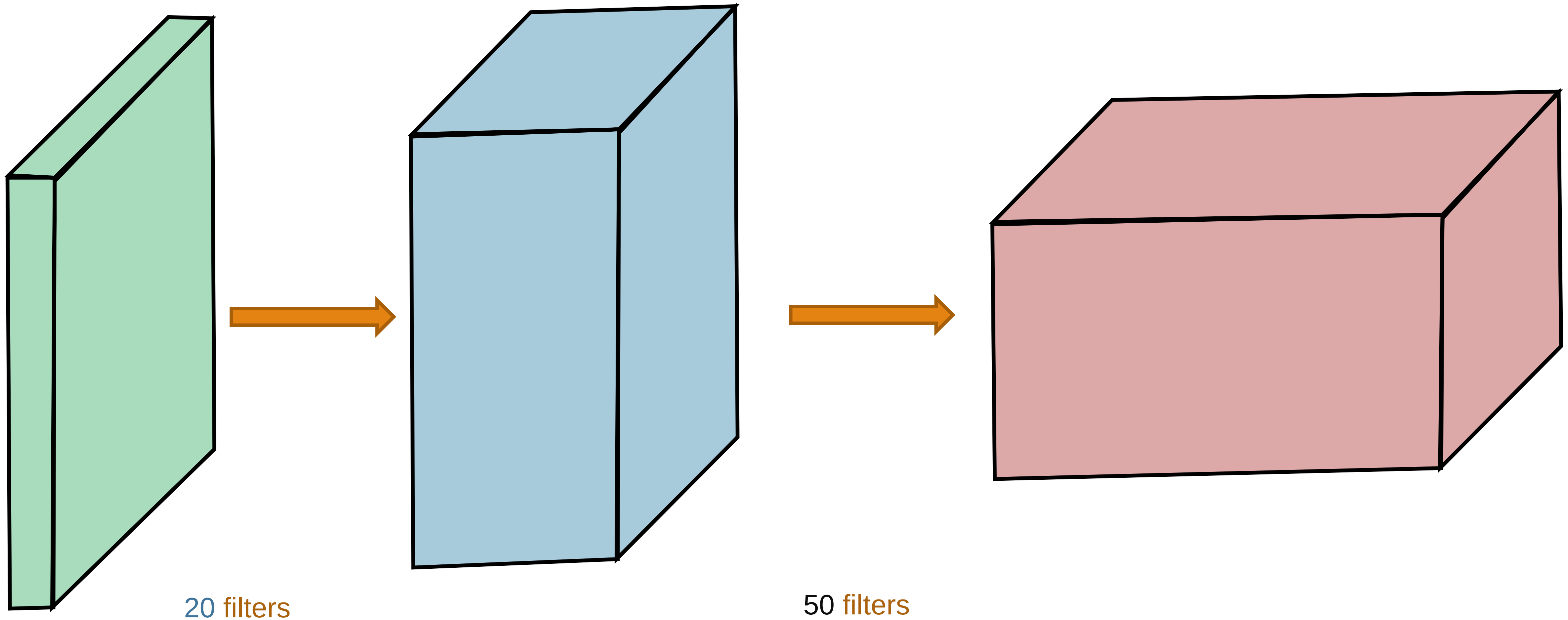
32X 32 X 3 $\longrightarrow$ ?

What is the size ?

## Convolution



## Convolution



20 filters

5 X 5 X 3

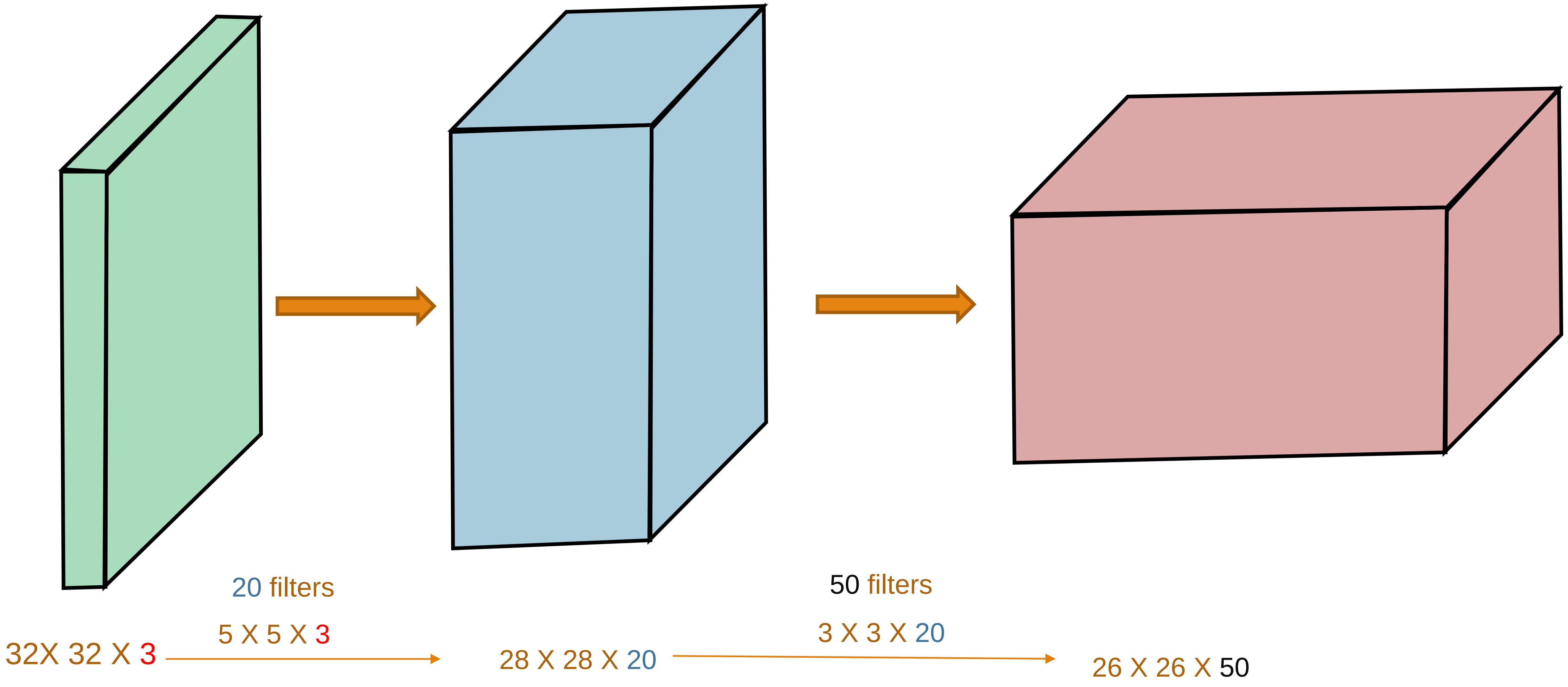
50 filters

3 X 3 X 20

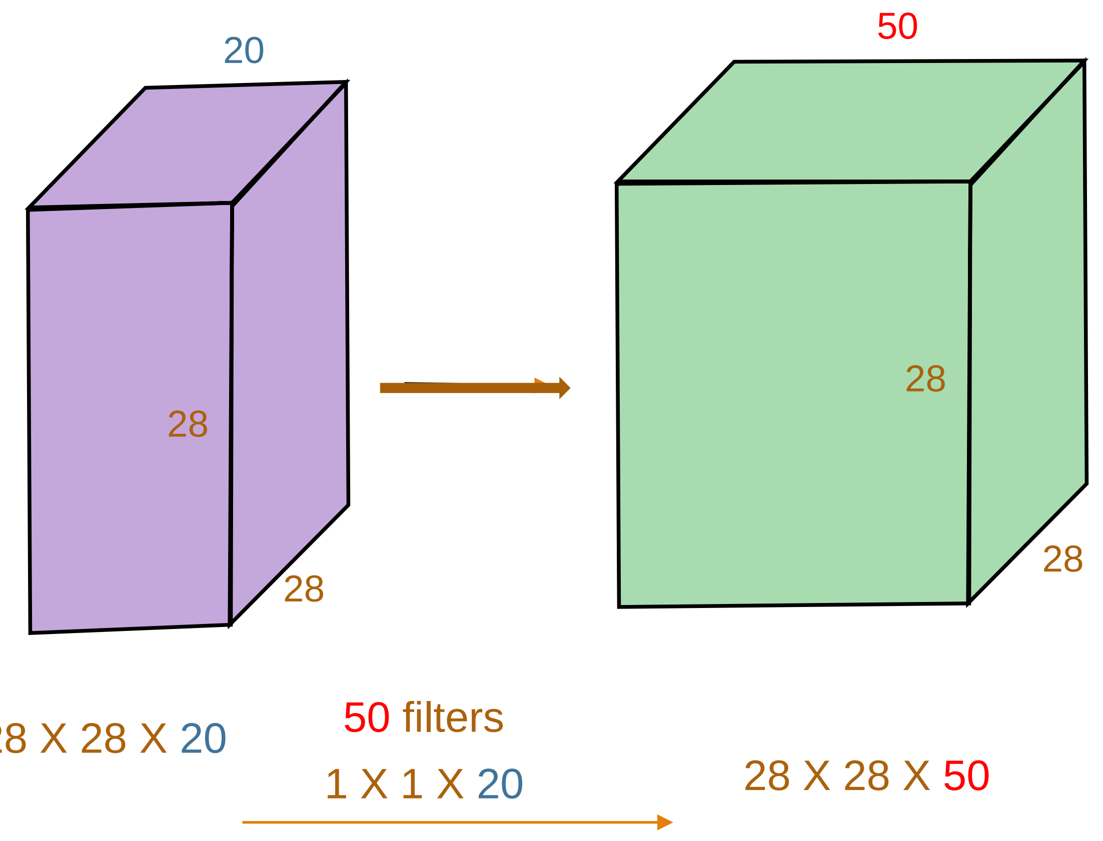
$$32 \times 32 \times 3 \xrightarrow{} 28 \times 28 \times 20 \xrightarrow{} ?$$

What is the size ?

# Convolution



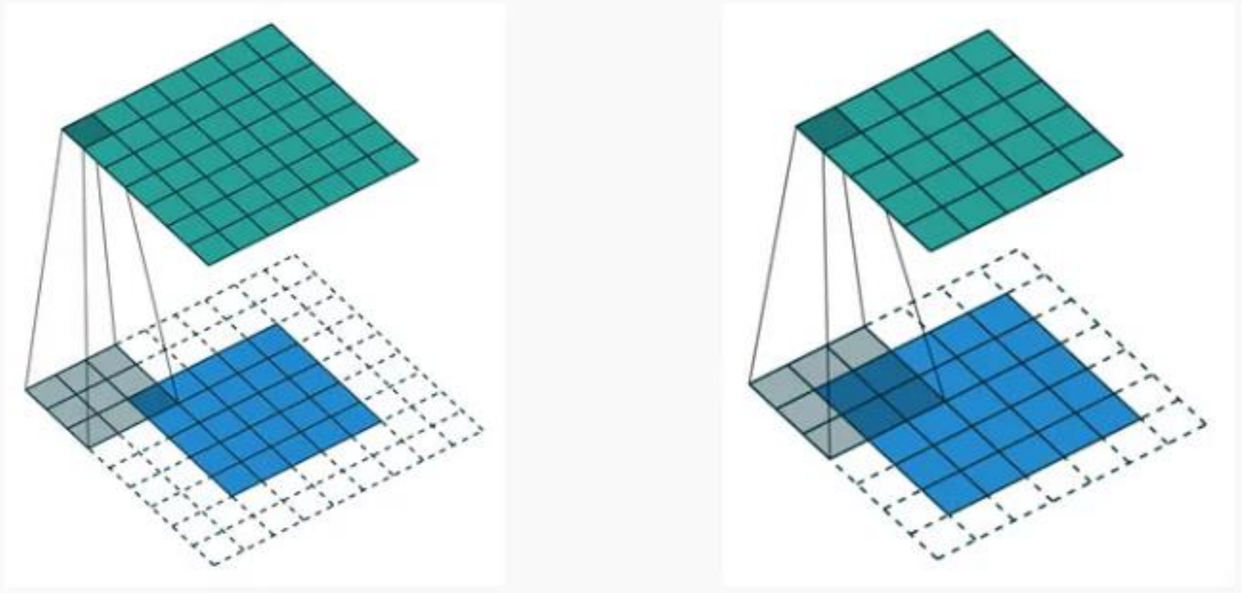
## Special cases 1X1 filters



You can think about these 1X1 filters as changing the volume depth but keeping the width and height unchanged

## Padding and stride

Padding is s technique used to prevent the image height and width from shrinking as we apply convolution operations. It works by “padding zeros to the boundary of the image”



Full padding. Introduces zeros such that all pixels are visited the same amount of times by the filter. Increases size of output.

Same padding. Ensures that the output has the same size as the input.

Padding Image source

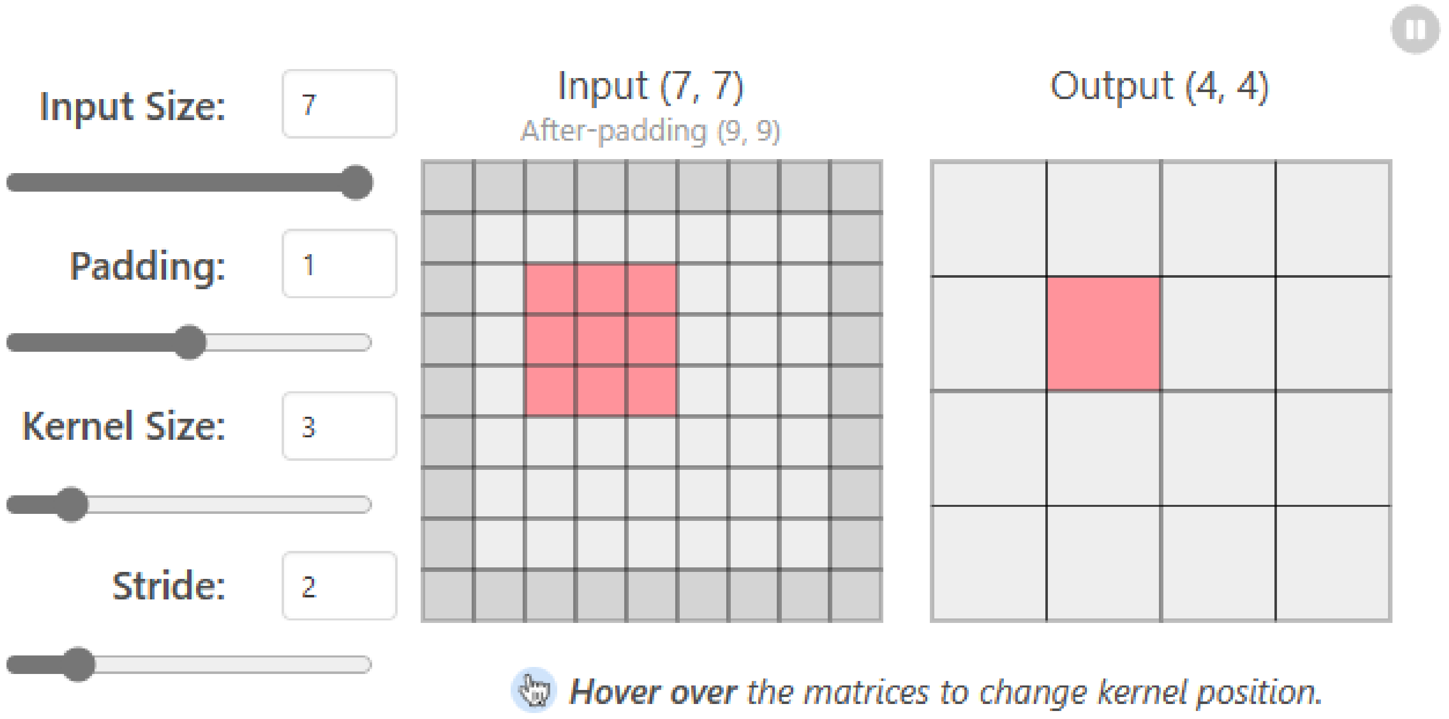
## Padding and stride

Padding is s technique used to prevent the image height and width from shrinking as we apply convolution operations. It works by “padding zeros to the boundary of the image”

Stride determines how exactly do we walk when over the image as we convolve the filter With the image. Let's explore these notions interactively!

## Convolution, padding and other concepts interactive tour

Lets explore this interactively!


Input Size: 7

Padding: 1

Kernel Size: 3

Stride: 2

Input (7, 7)
After-padding (9, 9)

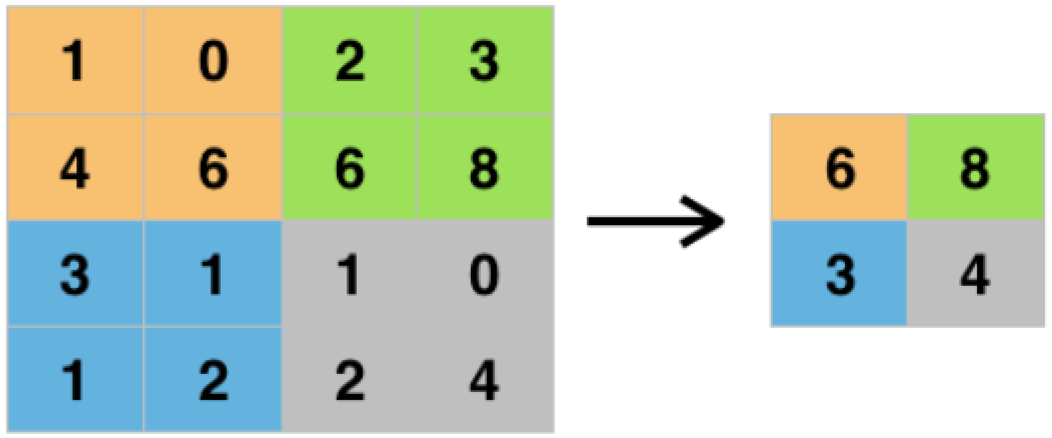
Output (4, 4)

***Hover over*** *the matrices to change kernel position.*

[CNN Explainer (poloclub.github.io)](https://poloclub.github.io)

## Pooling

Pooling in deep learning is a technique that reduces the dimensionality of data by summarizing or **downsampling** it, aiding in capturing important features while reducing computational complexity.



Max pooling

Image source : Convolutional neural network - Wikipedia

## Before we give an example : ImageNet

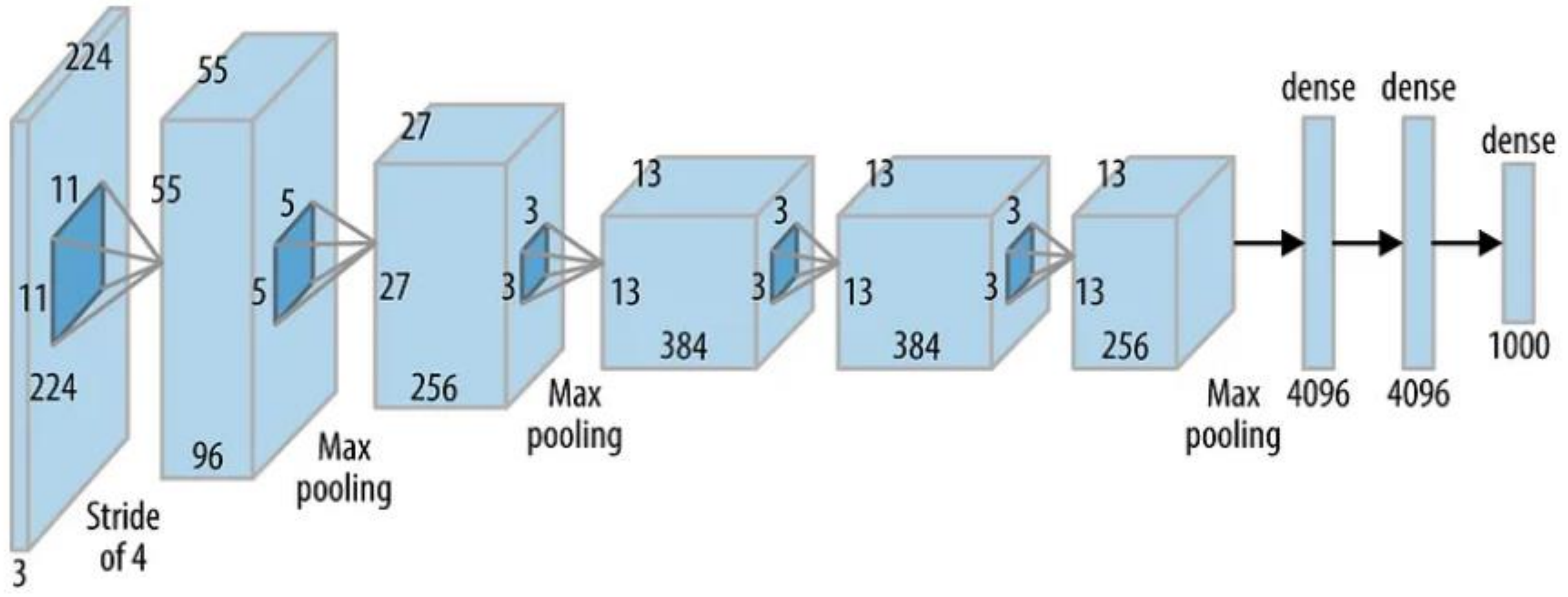
ImageNet is a vast dataset of labeled images used for visual object recognition research, comprising millions of images across a 1000 categories. [ImageNet (image-net.org)](https://image-net.org)

Samples from the ImageNet dataset

## Example : AlexNet

AlexNet, a deep convolutional neural network architecture, transformed image classification by winning the 2012 ImageNet Large Scale Visual Recognition Challenge, marking a pivotal advancement in deep learning.

The network is essentially a conv network followed by two dense network, and finally the 1,000 classes.

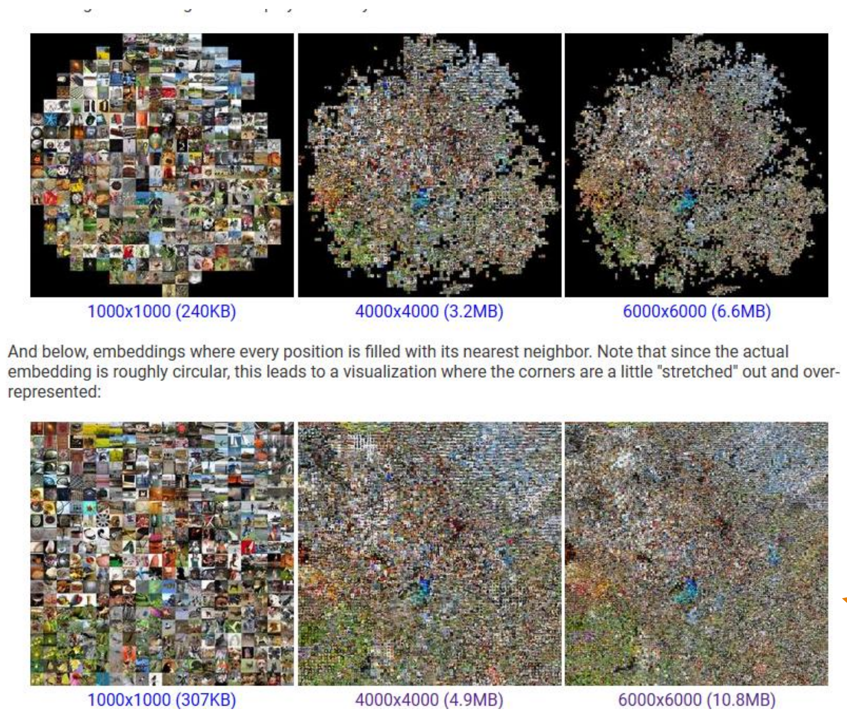


Alexnet Block Diagram (source:oreilly.com)

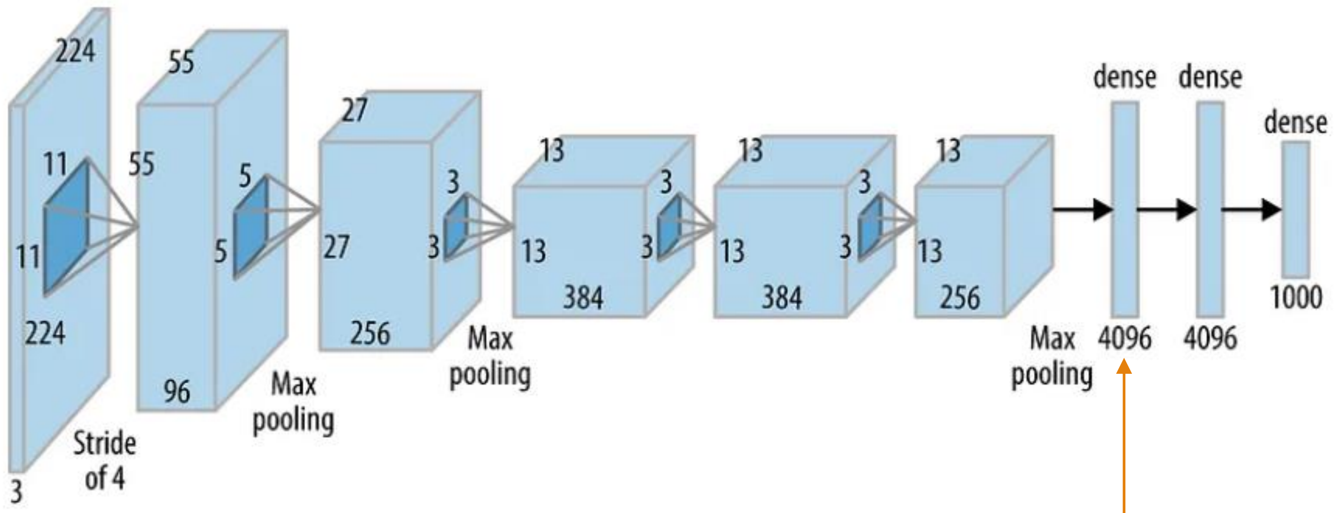
## AlexNet : intuition

Other CNN architectures appeared later that achieve perfect score on the ImageNet benchmark. We explore these later.

How exactly these network do what they do?



And below, embeddings where every position is filled with its nearest neighbor. Note that since the actual embedding is roughly circular, this leads to a visualization where the corners are a little "stretched" out and over-represented:

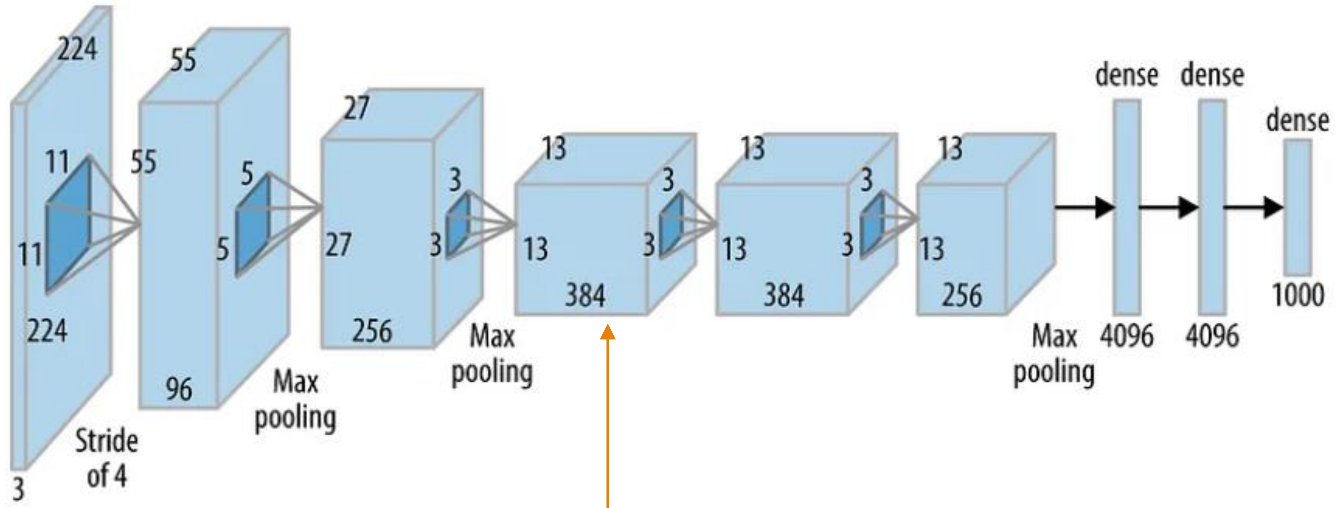


Alexnet Block Diagram (source:oreilly.com)

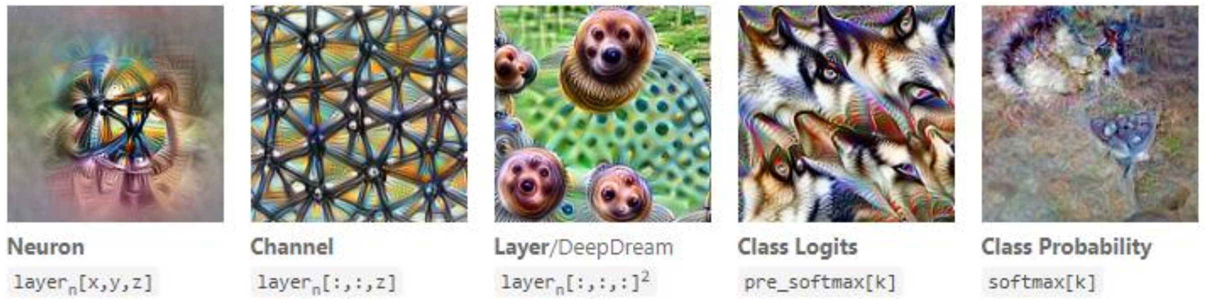
Let us explore the embeddings of the ImageNet at this layer of the AlexNet

[t-SNE visualization of CNN codes (stanford.edu)]

## Maximally activated patches



Alexnet Block Diagram (source:oreilly.com)



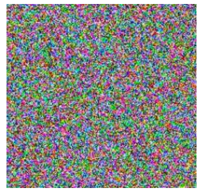
Alternatively, we can synthesize these images by optimization : find an input x that maximize certain (neuron, channel, layer) in the network.

[Feature Visualization (distill.pub)]

## Optimization of a CNN input to maximally activate an neuron

Here's a step-by-step algorithm for visualizing the function of a specific unit in a neural network by synthesizing inputs that cause that unit to have high activation:

1. Initialize a random image: Start with a random image where each pixel's color is randomly chosen. This random image serves as the initial input.



2. Forward pass: Perform a forward pass using the random image x as input to the neural network. This computes the activation $a_i(x)$ caused by the input x at the desired neuron i somewhere in the middle of the network.

3. Backward pass (Backpropagation):
   - Compute gradient: Perform backpropagation to compute the gradient of the activation $a_i(x)$ with respect to earlier activations in the network.

4. Update the image:
   - Adjust pixel colors: Use the gradient obtained in the backward pass to determine how to change the color of each pixel in the image to increase the activation of the neuron i.
   - Scaling factor: Add a small fraction alpha of the gradient to the image. The fraction alpha controls the magnitude of the change applied to the image.

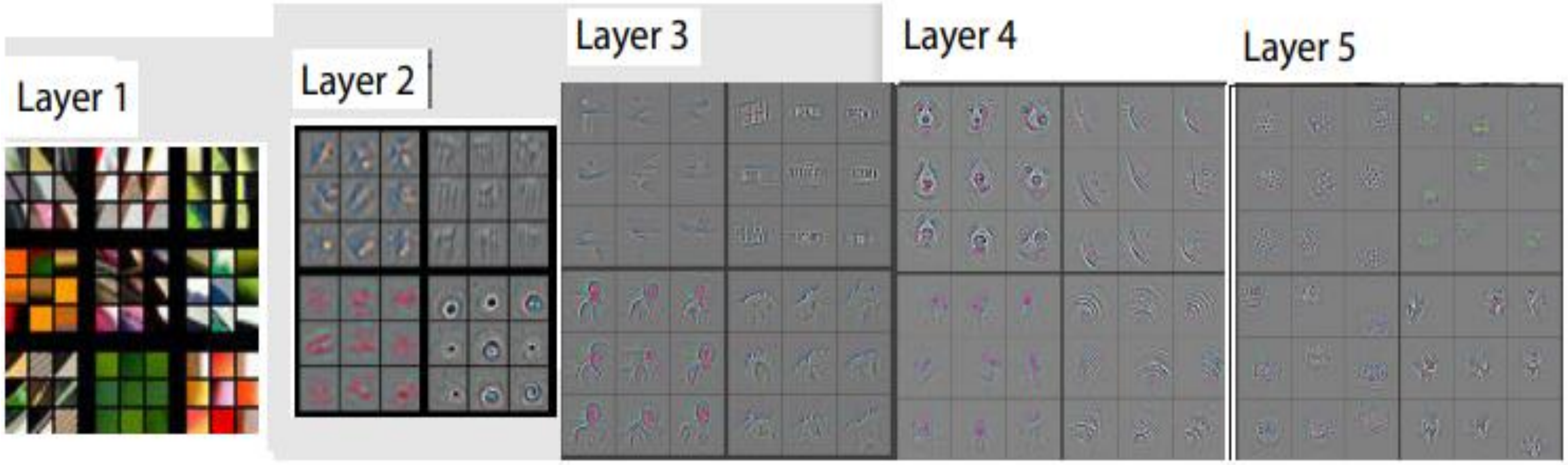
$$x \leftarrow x + \alpha \cdot \partial a_i(x) / \partial x$$

5. Repeat: Iterate this process by using the updated image as the input and repeating steps 2 to 4. This iterative process continues until convergence or until a desired number of iterations is reached.

[Feature Visualization (distill.pub)]

## Convolution

### What do the filters learn ?



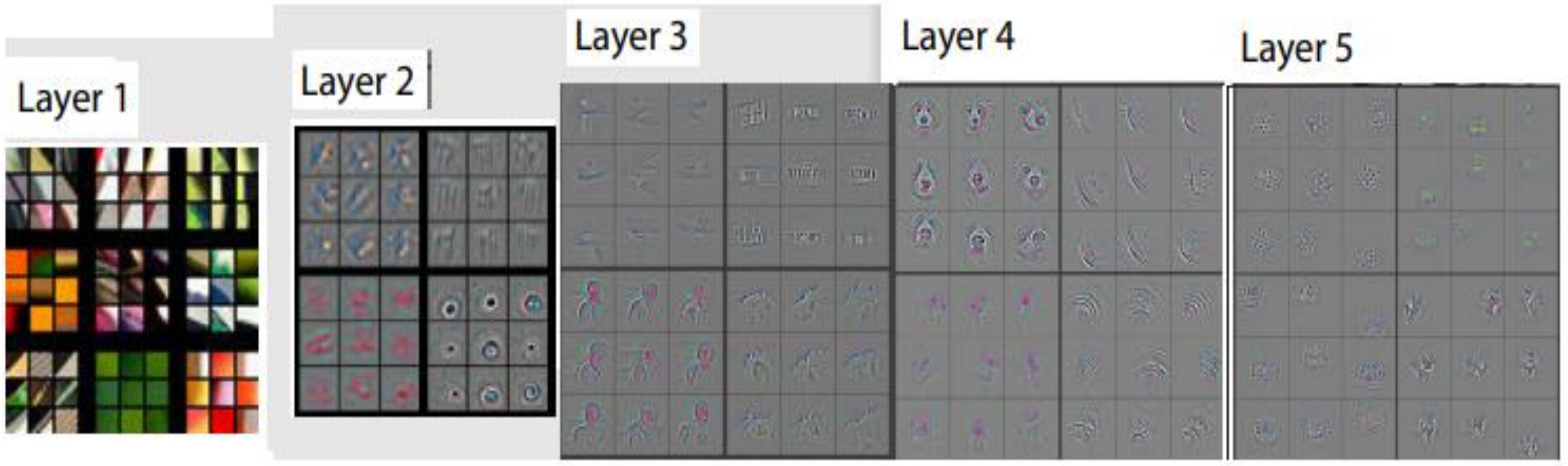
The learned representations refer to the hierarchical and abstract features that the network extracts from the input data during the training process. These representations are obtained by applying convolutional filters and pooling operations throughout the network's layers.

The learned representations in a CNN capture increasingly complex and abstract features from the input data. The earlier layers tend to learn low-level features, such as edges and textures, while deeper layers learn more high-level features that are specific to the task the network was trained on.

Source : Visualizing and Understanding Convolutional Networks (arxiv.org)

## End-to-end training

These feature are learned automatically! We never asked the network to attend to cloth texture or text, etc. The network learns all these features in an end-to-end fashion!

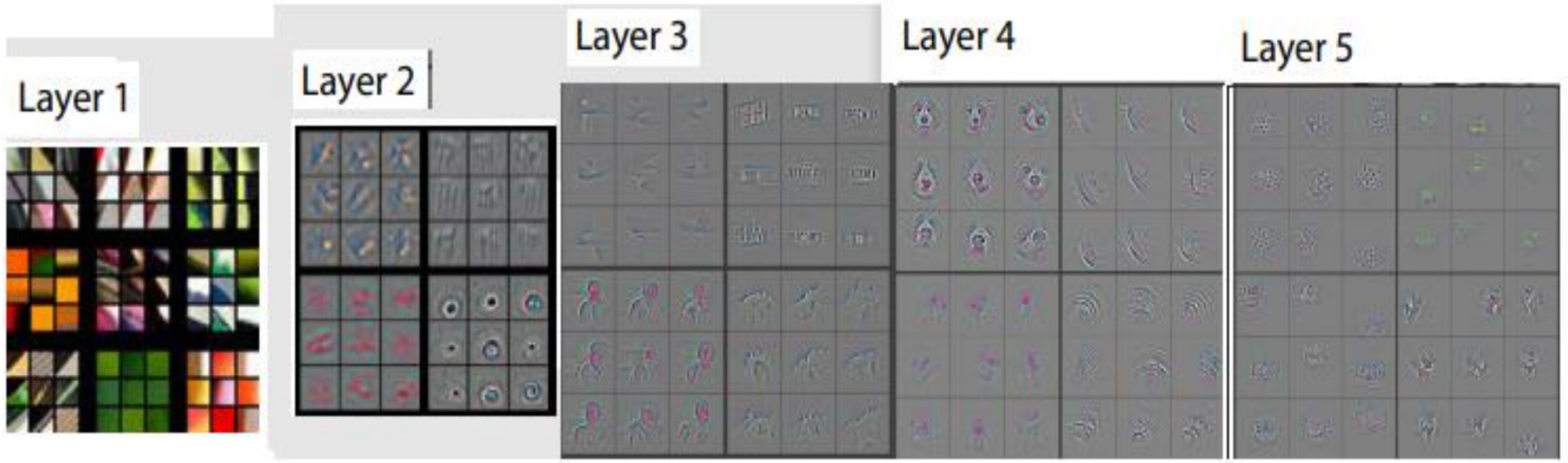


- End-to-end training refers to a machine learning approach where a model learns to perform a task directly from raw input data to desired output, without relying on manual feature engineering or intermediate steps.

- In this process, the model learns to extract relevant features and make predictions simultaneously, effectively learning the entire process "end to end."
- This approach simplifies the development pipeline, as it eliminates the need for designing complex feature extraction algorithms and allows the model to automatically learn the most relevant patterns and representations from the data, leading to potentially more accurate and efficient results.

## End-to-end training

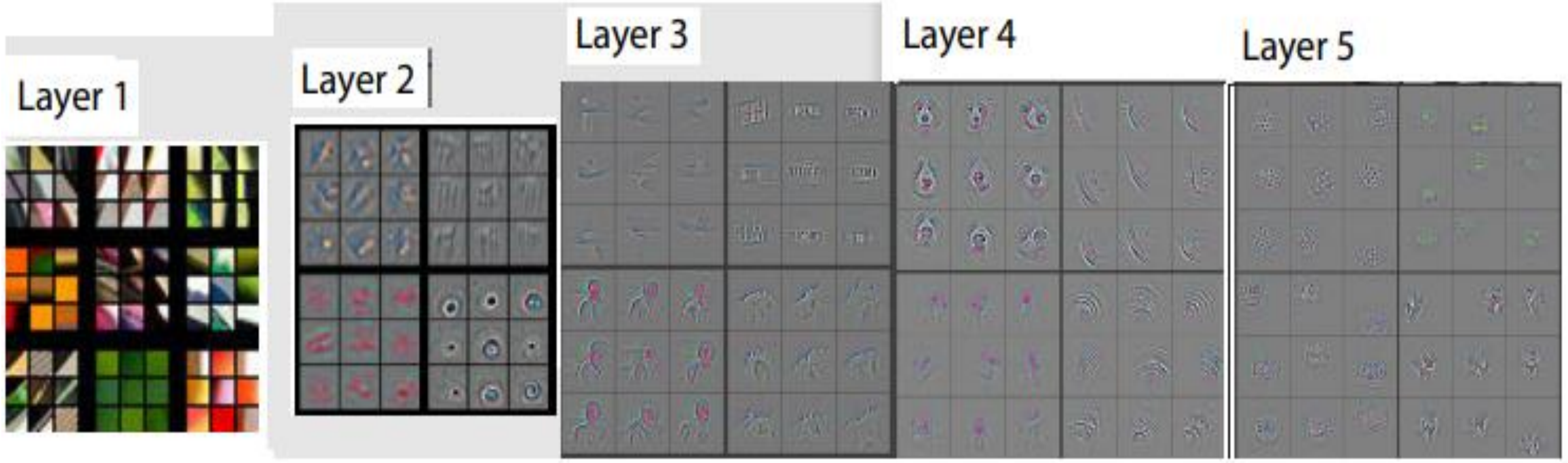
These feature are learned automatically! We never asked the network to attend to cloth texture or text, etc. The network learns all these features in an end-to-end fashion!



- In the context of CNN it refers to training the network to directly map input images to desired outputs without explicitly designing and incorporating handcrafted features.

- In traditional approaches, engineers would manually extract features from images, such as edges or textures, and then feed these features into a separate classifier. However, with end-to-end training using CNNs, the network learns to automatically extract relevant features from raw pixel values and make predictions in a single step.
- The CNN can learn to extract hierarchical representations of the input data, enabling more efficient and accurate performance on a wide range of computer vision tasks.

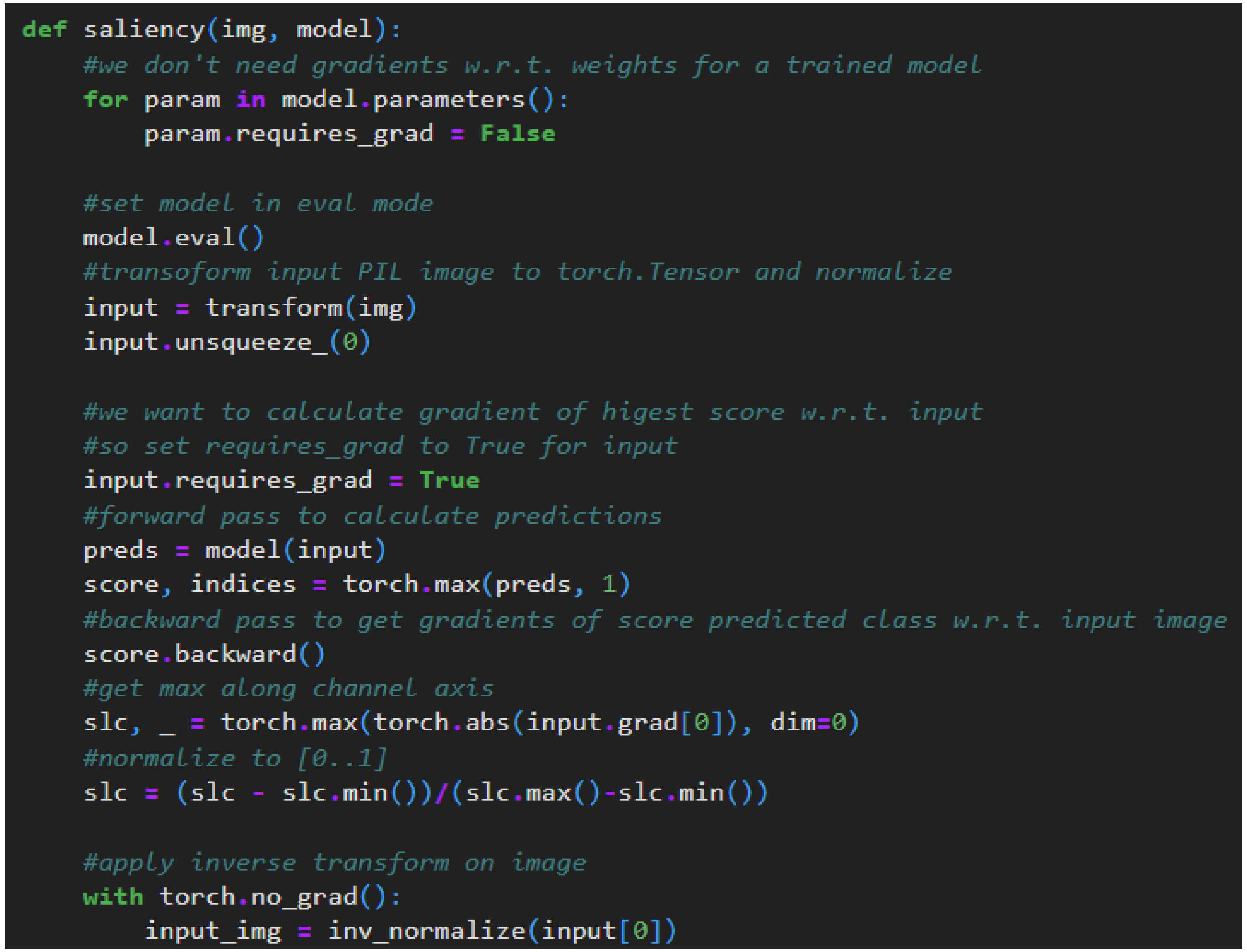
## Learned features in the first few layers



In the initial layers of a CNN, low-level features such as edges, textures, and simple shapes are learned. These features are fundamental building blocks for recognizing more complex patterns and objects in images. Since these low-level features are common across various types of images, CNNs trained on different datasets tend to learn similar representations in their early layers.

Furthermore, the convolutional layers of CNNs are designed to be translationally invariant (why?), meaning they can detect the same patterns regardless of their location in the image. This property allows CNNs to generalize well across different datasets and learn similar low-level features.

## Differentiate the input with respect to the output : Saliency maps

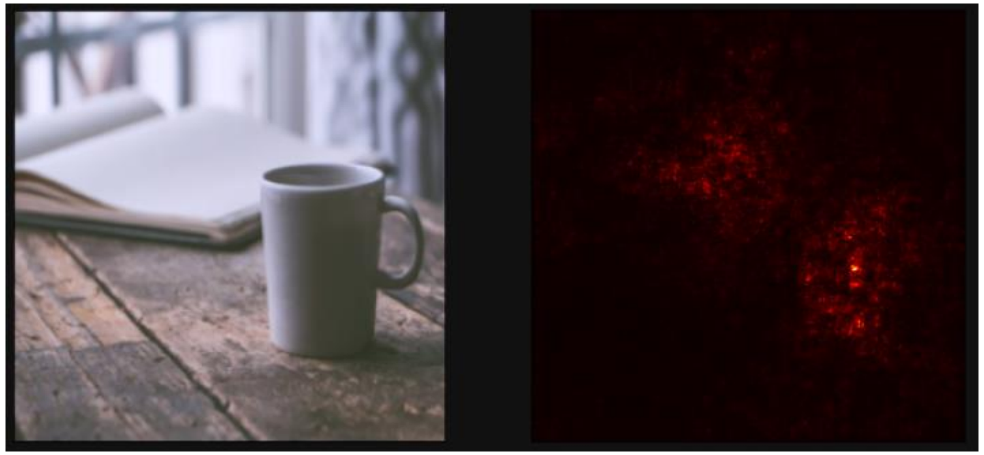


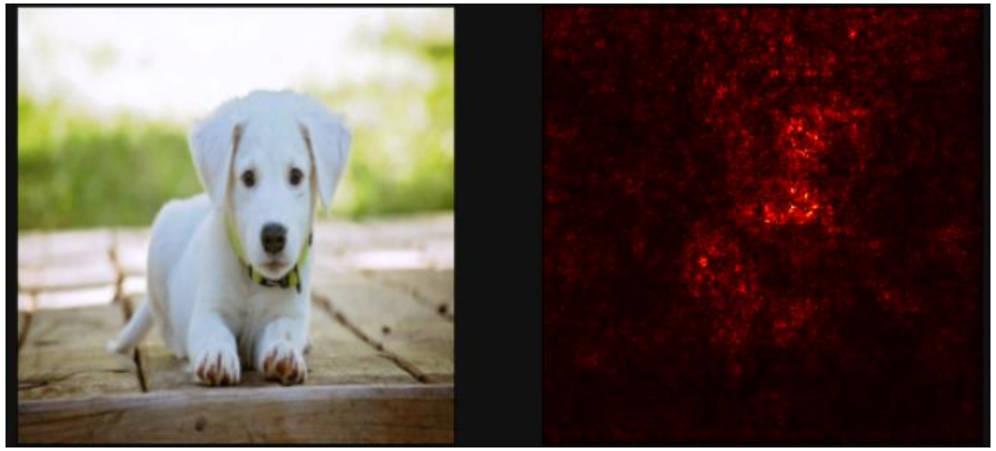
```python
def saliency(img, model):
    #we don't need gradients w.r.t. weights for a trained model
    for param in model.parameters():
        param.requires_grad = False

    #set model in eval mode
    model.eval()
    #transoform input PIL image to torch.Tensor and normalize
    input = transform(img)
    input.unsqueeze_(0)

    #we want to calculate gradient of higest score w.r.t. input
    #so set requires_grad to True for input
    input.requires_grad = True
    #forward pass to calculate predictions
    preds = model(input)
    score, indices = torch.max(preds, 1)
    #backward pass to get gradients of score predicted class w.r.t. input image
    score.backward()
    #get max along channel axis
    slc, _ = torch.max(torch.abs(input.grad[0]), dim=0)
    #normalize to [0..1]
    slc = (slc - slc.min())/(slc.max()-slc.min())

    #apply inverse transform on image
    with torch.no_grad():
        input_img = inv_normalize(input[0])
```





pytorch-saliency-maps/Saliency_maps_in_pytorch.ipynb at master · sunnynevarekar/pytorch-saliency-maps (github.com)

## Why CNNs work really well in practice?

## Inductive bias

- Inductive bias is the set of assumptions (or biases) that a machine learning algorithm makes to generalize from limited training data and make predictions about unseen examples.

- An example of inductive bias is a machine learning algorithm assuming that simpler explanations or hypotheses are more likely to be correct than complex ones.

## Inductive bias in CNNs

- In the context of Convolutional Neural Networks (CNNs), the inductive bias refers to the assumptions and design choices made in the network architecture that enable it to effectively learn and extract features from visual data.

## Inductive bias in CNNs

- In the context of Convolutional Neural Networks (CNNs), the inductive bias refers to the assumptions and design choices made in the network architecture that enable it to effectively learn and extract features from visual data.

- For example, CNNs have a built-in inductive bias that assumes spatial locality and translation invariance. This means that the network assumes that nearby pixels in an image are more likely to be related and that the learned features should be applicable regardless of their position in the image. By incorporating convolutional layers and pooling operations, CNNs can exploit these assumptions and efficiently capture local patterns and hierarchical representations in images.

## Inductive bias in CNNs

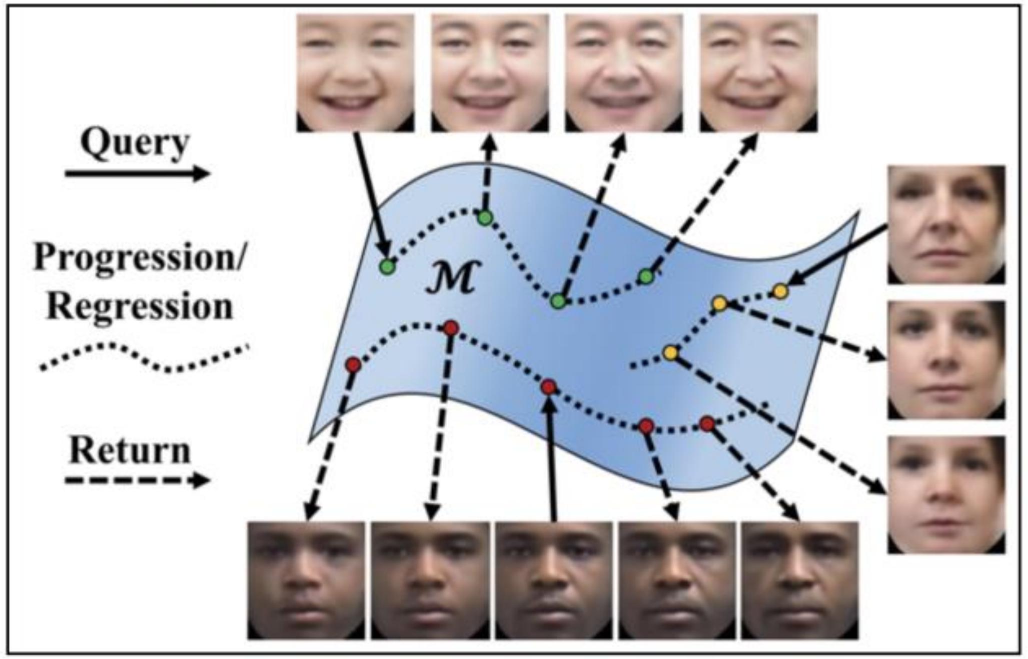
- In the context of Convolutional Neural Networks (CNNs), the inductive bias refers to the assumptions and design choices made in the network architecture that enable it to effectively learn and extract features from visual data.

- For example, CNNs have a built-in inductive bias that assumes spatial locality and translation invariance. This means that the network assumes that nearby pixels in an image are more likely to be related and that the learned features should be applicable regardless of their position in the image. By incorporating convolutional layers and pooling operations, CNNs can exploit these assumptions and efficiently capture local patterns and hierarchical representations in images.

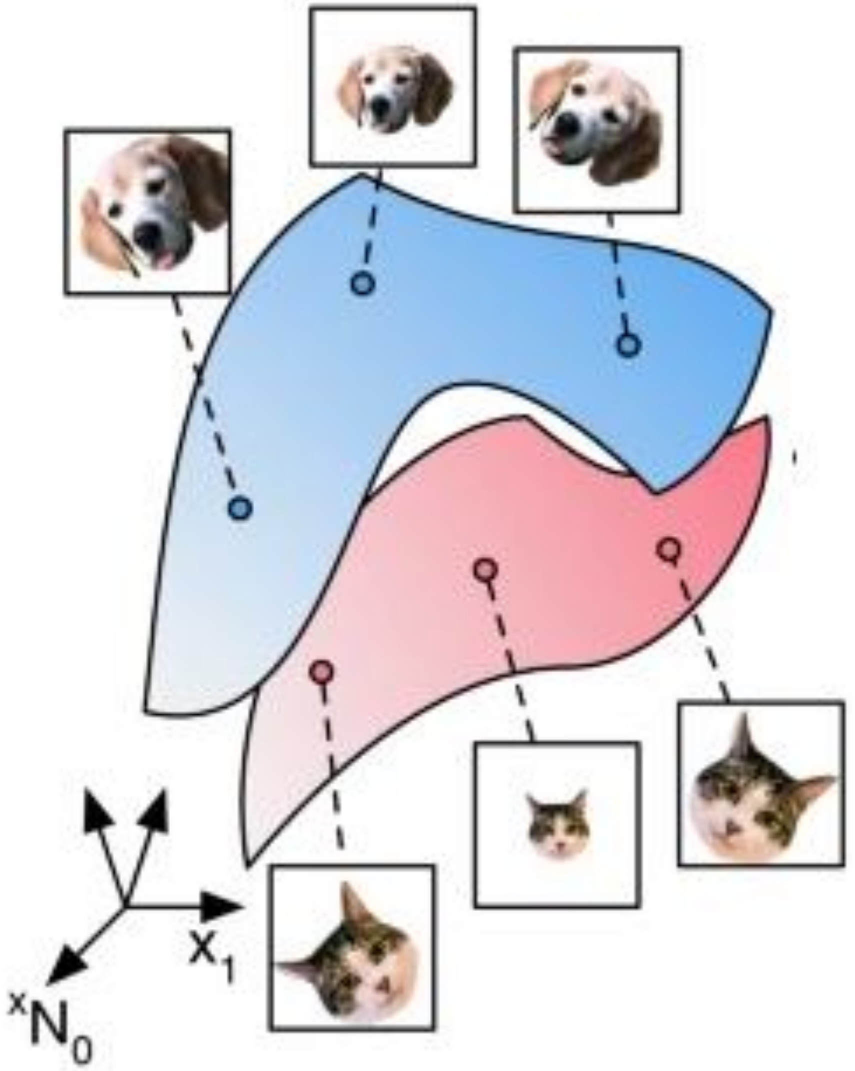
- The inductive bias of CNNs helps them excel in tasks such as image classification, object detection, and image segmentation, where spatial relationships and local patterns play a crucial role.

# The Manifold Hypothesis

- The manifold hypothesis suggests that real-world data, even in high-dimensional spaces (number of pixels), often concentrates around a lower-dimensional manifold embedded within that space. NNs have the ability to learn and model complex mappings from the input space to the output space, which includes capturing the intricate relationships and patterns present in the data.



14.3 Learning Manifolds (buffalo.edu)

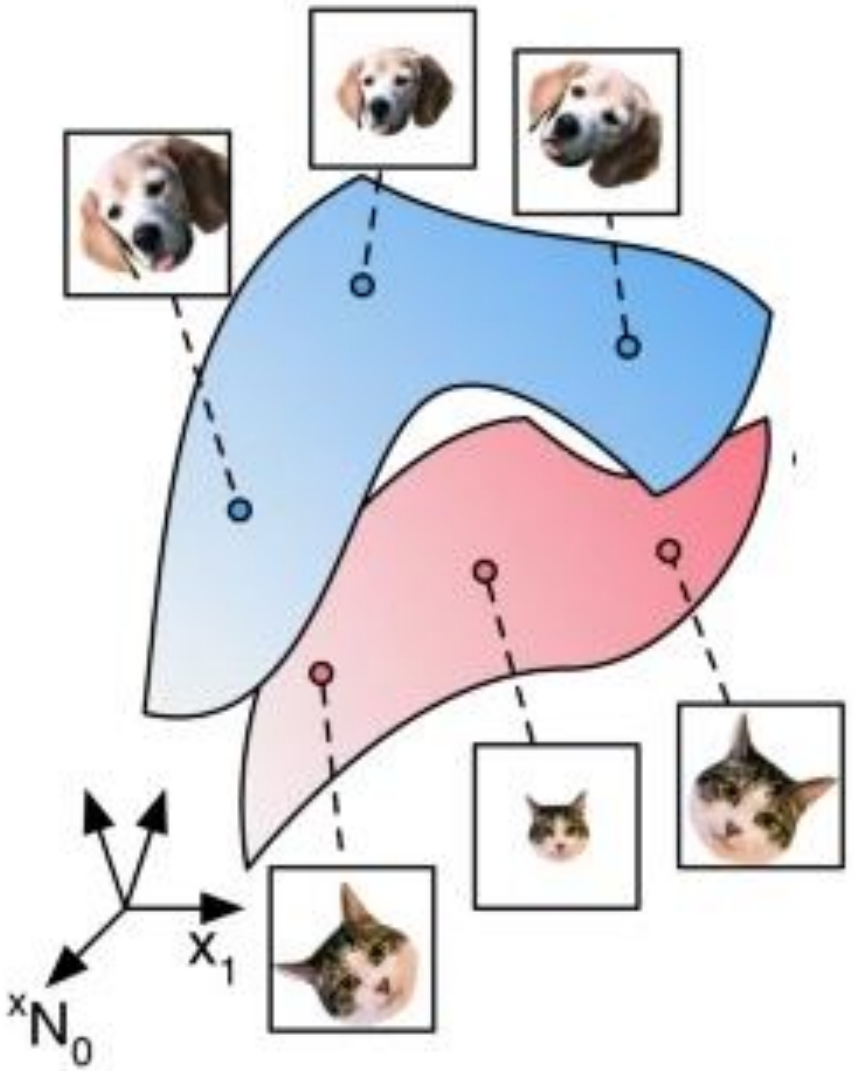


Separability and geometry of object manifolds in deep neural networks | Nature Communications

## The Manifold Hypothesis

- The manifold hypothesis suggests that real-world data, even in high-dimensional spaces (number of pixels), often concentrates around a lower-dimensional manifold embedded within that space. NNs have the ability to learn and model complex mappings from the input space to the output space, which includes capturing the intricate relationships and patterns present in the data.

- Neural Networks work effectively because they are capable of learning and approximating the underlying structure of high-dimensional data that lies on or near a lower-dimensional manifold.

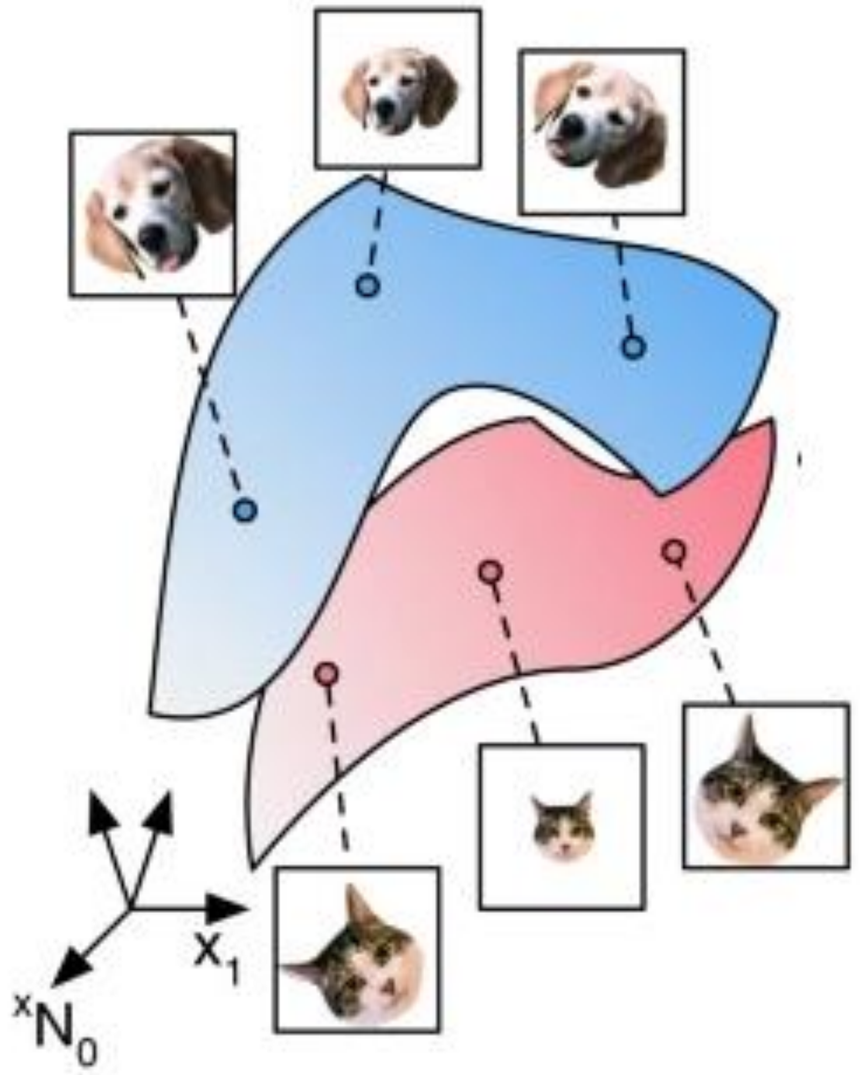


Separability and geometry of object manifolds in deep neural networks | Nature Communications

# The Manifold Hypothesis

- Through backpropagation, NNs adjust their weights and biases during training to minimize the discrepancy between the predicted outputs and the ground truth labels.

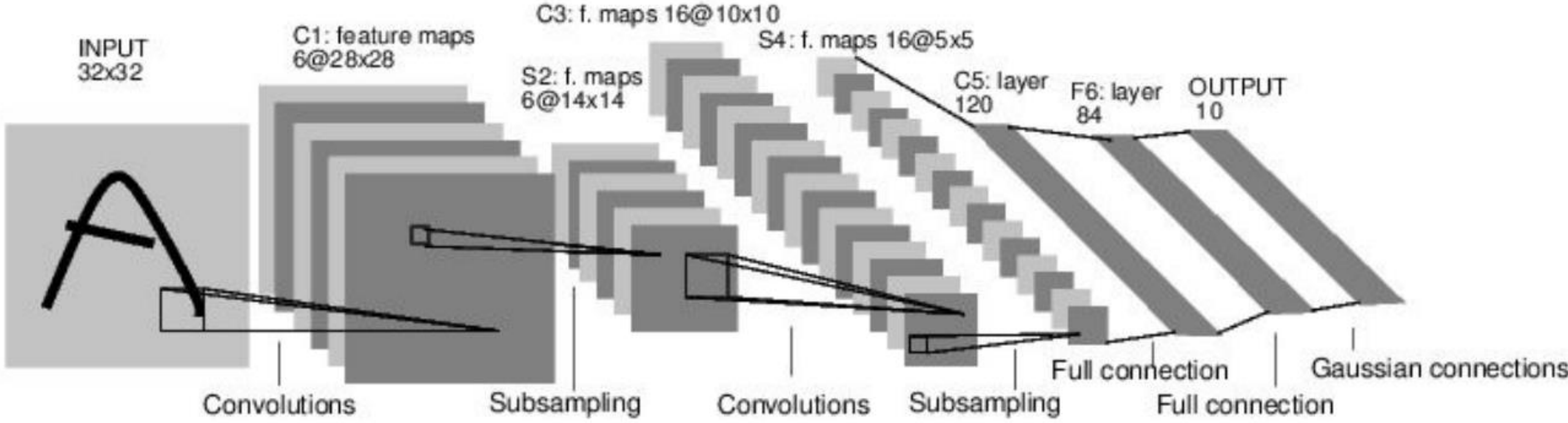
- This optimization process helps the network align its learned representations with the underlying manifold, enabling it to generalize well and make accurate predictions on unseen examples.



[Separability and geometry of object manifolds in deep neural networks | Nature Communications]

## Case study : LeNet

[LeCun et al., 1998]

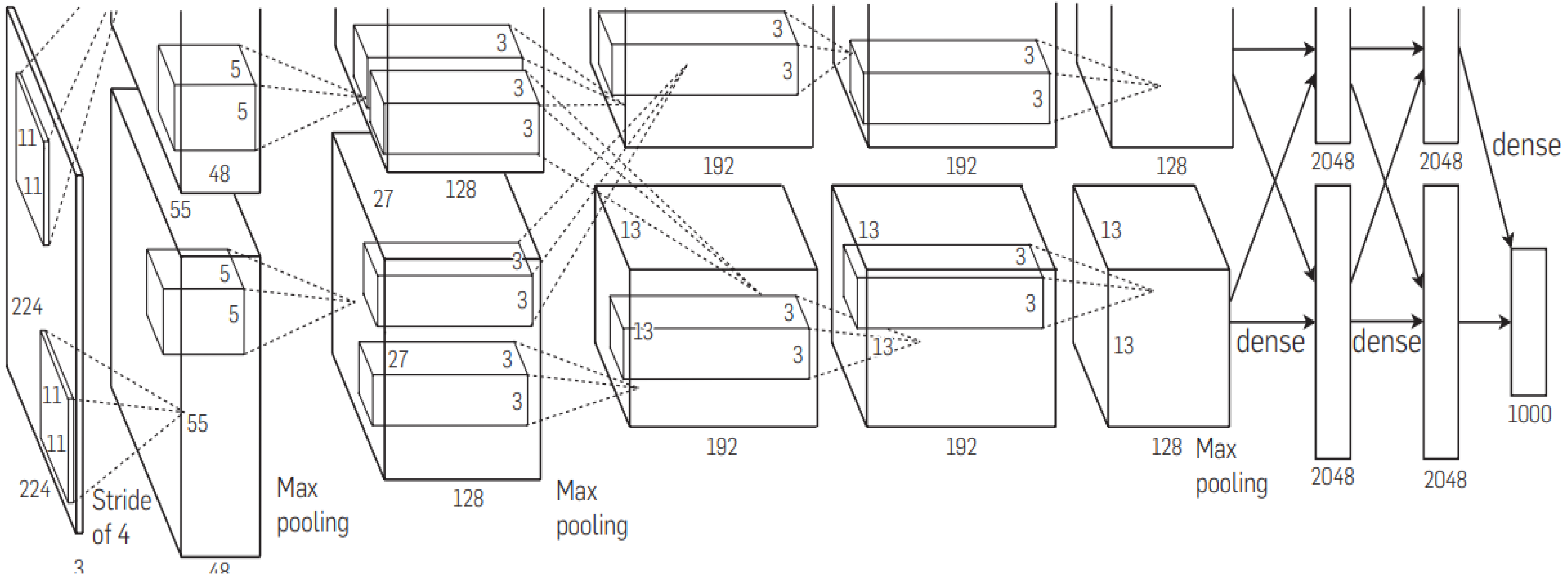


Conv filters were 5x5, applied at stride 1 Subsampling (Pooling) layers were 2x2 applied at stride

architecture is [CONV-POOL-CONV-POOL-CONV-FC] : almost standard nowdays in CNN

## Case study : AlexNet

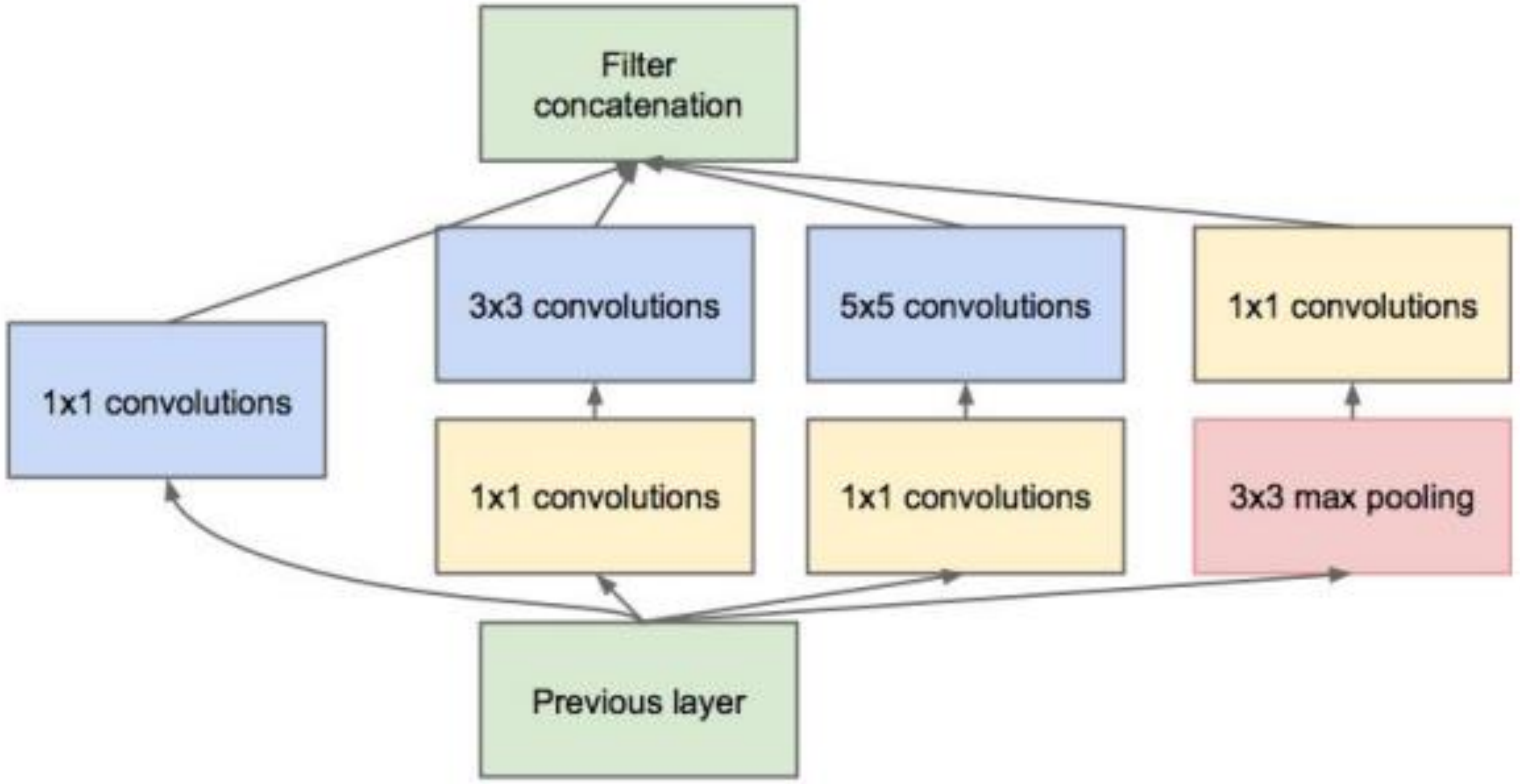
The new arrival of modern DL :



Input: 227x227x3
After CONV1: 55x55x96
After POOL1: 27x27x96

[ImageNet classification with deep convolutional neural networks (acm.org)]

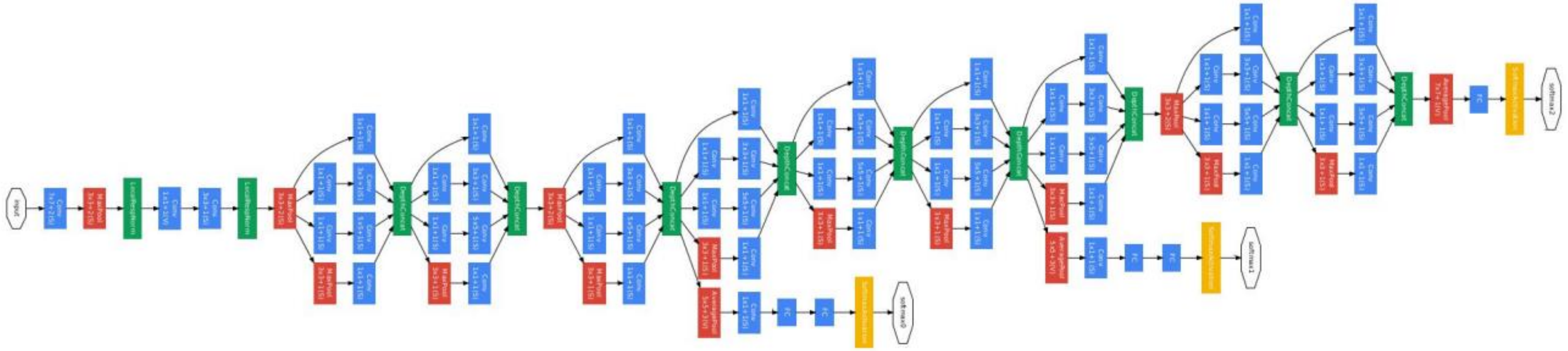
## Case study : GoogleNet



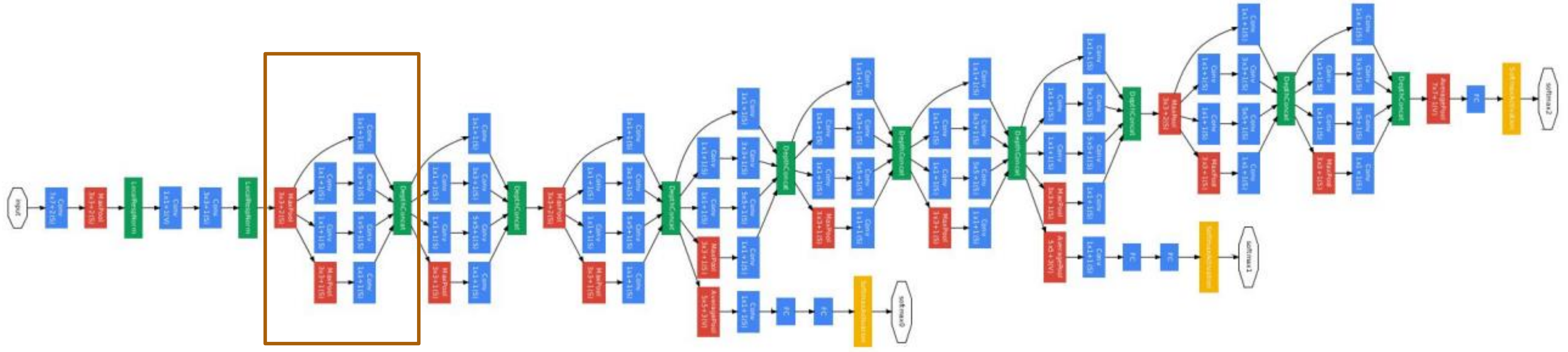
Inception module

Main idea: apply filters with different size on the input image and then concatenate all of them together

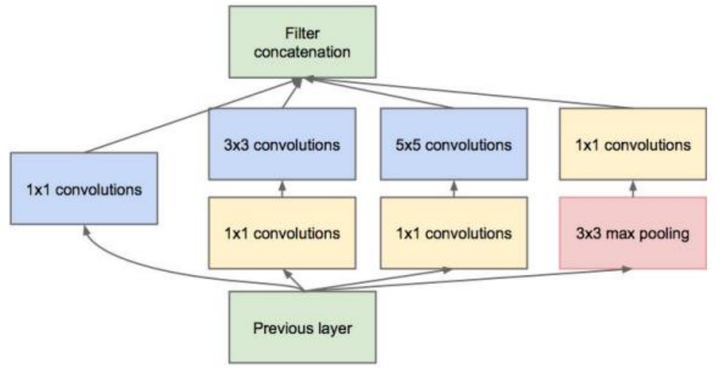
## Case study : GoogleNet



## Case study : GoogleNet



Although it looks complicated : it repeats the inception module multiple times



## The Vanishing Gradient Problem

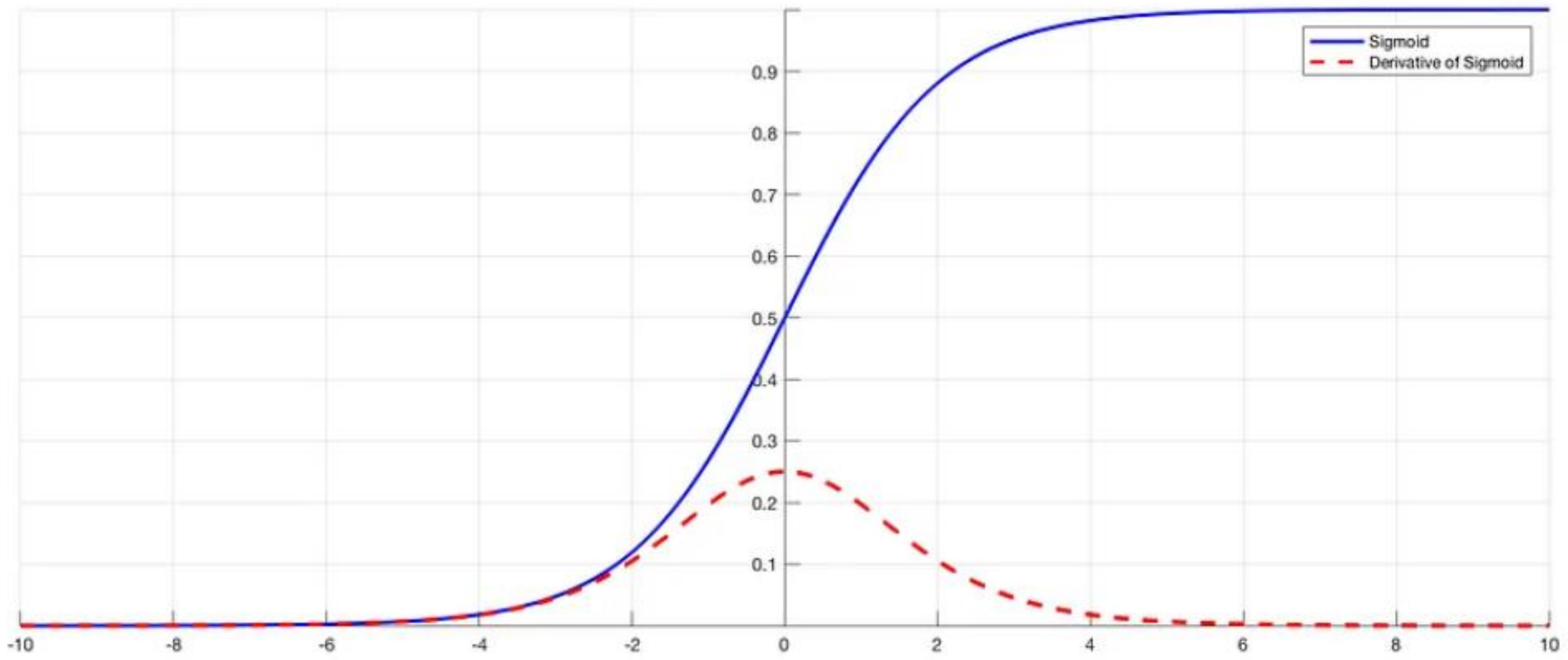


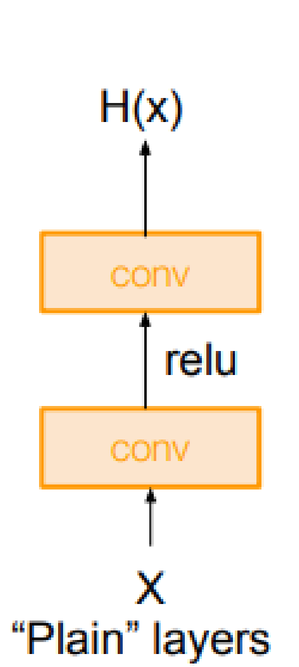
Image 1: The sigmoid function and its derivative // Source

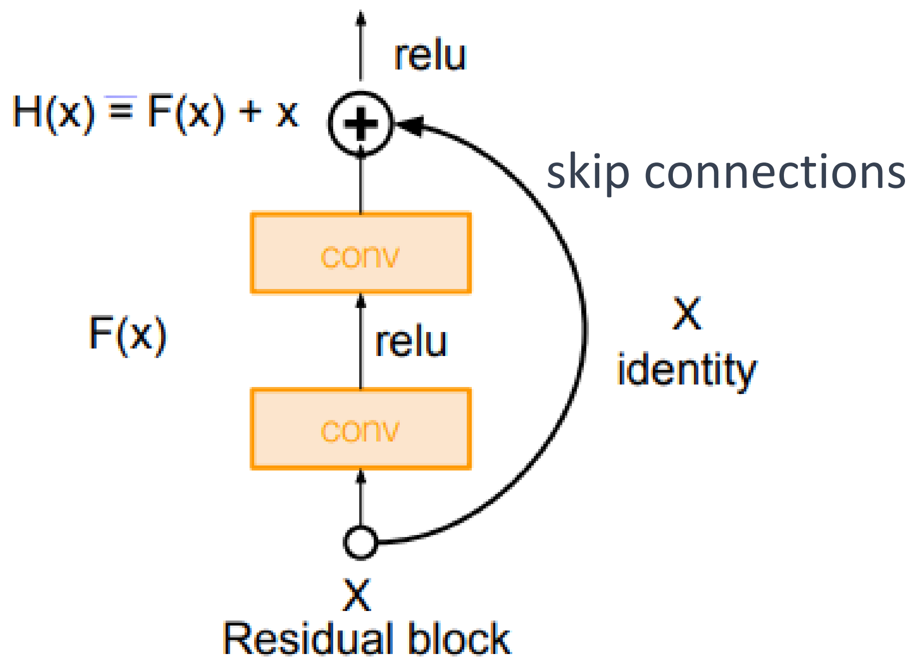
Activation functions such as the sigmoid function compress a wide input range into a narrow output range, causing a small change in output for a large input change. As a result, the derivative becomes small.

## The Vanishing Gradient Problem

- Gradients of neural networks are found using backpropagation.
- Simply put, backpropagation finds the derivatives of the network by moving layer by layer from the final layer to the initial one.
- By the chain rule, the derivatives of each layer are multiplied down the network (from the final layer to the initial) to compute the derivatives of the initial layers.
- However, when *n* hidden layers use an activation like the sigmoid function, *n* small derivatives are multiplied together. Thus, the gradient decreases exponentially as we propagate down to the initial layers.
- A small gradient means that the weights and biases of the initial layers will not be updated effectively with each training session. Since these initial layers are often crucial to recognizing the core elements of the input data, it can lead to overall inaccuracy of the whole network.

## Case study : RezNet

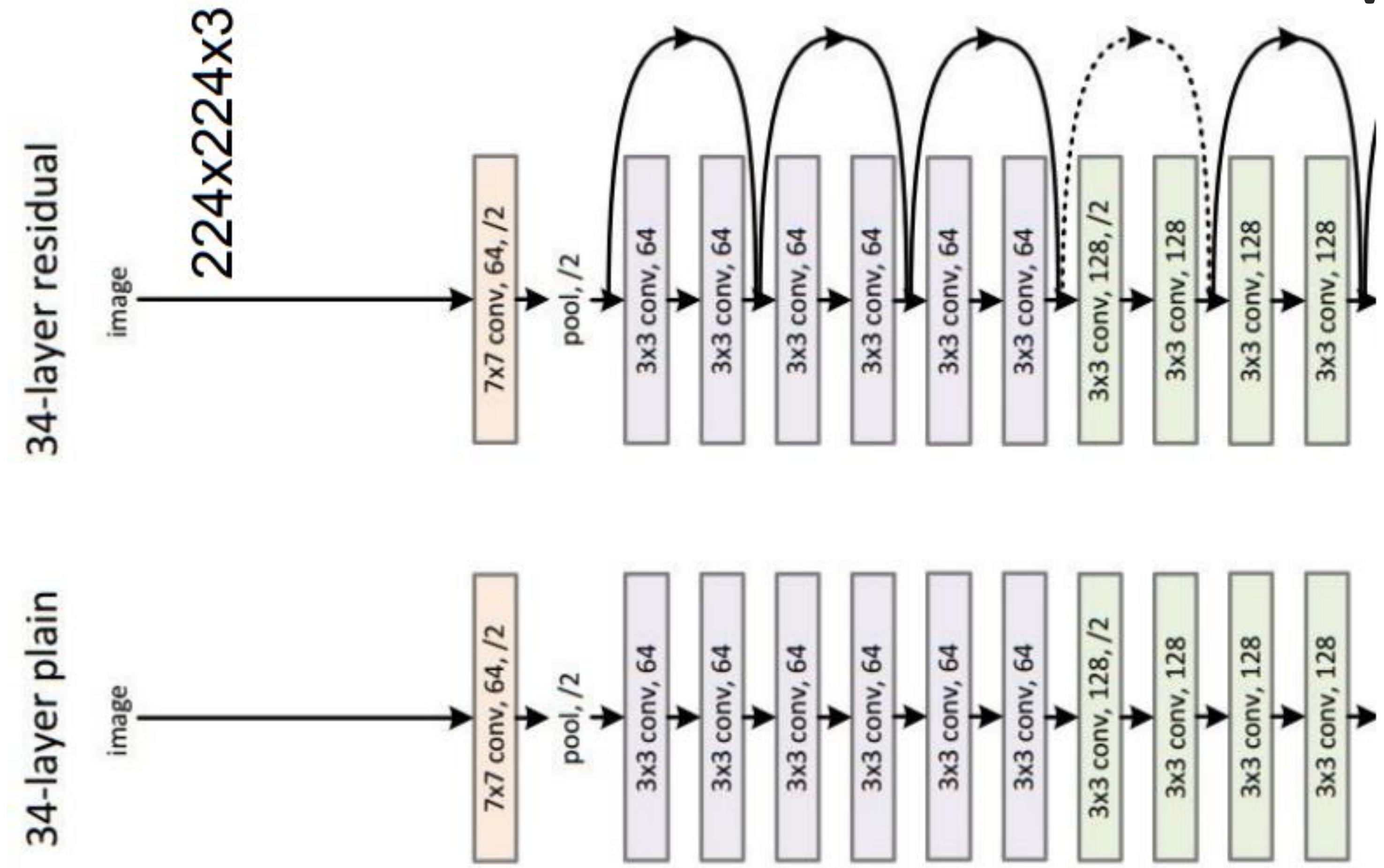




The residual connection directly adds the value at the beginning of the block, **x**, to the end of the block (F(x)+x). This residual connection doesn't go through activation functions that "squashes" the derivatives, resulting in a higher overall derivative of the block.

## Case study : ResNet

ResNets allow training very deep neural networks



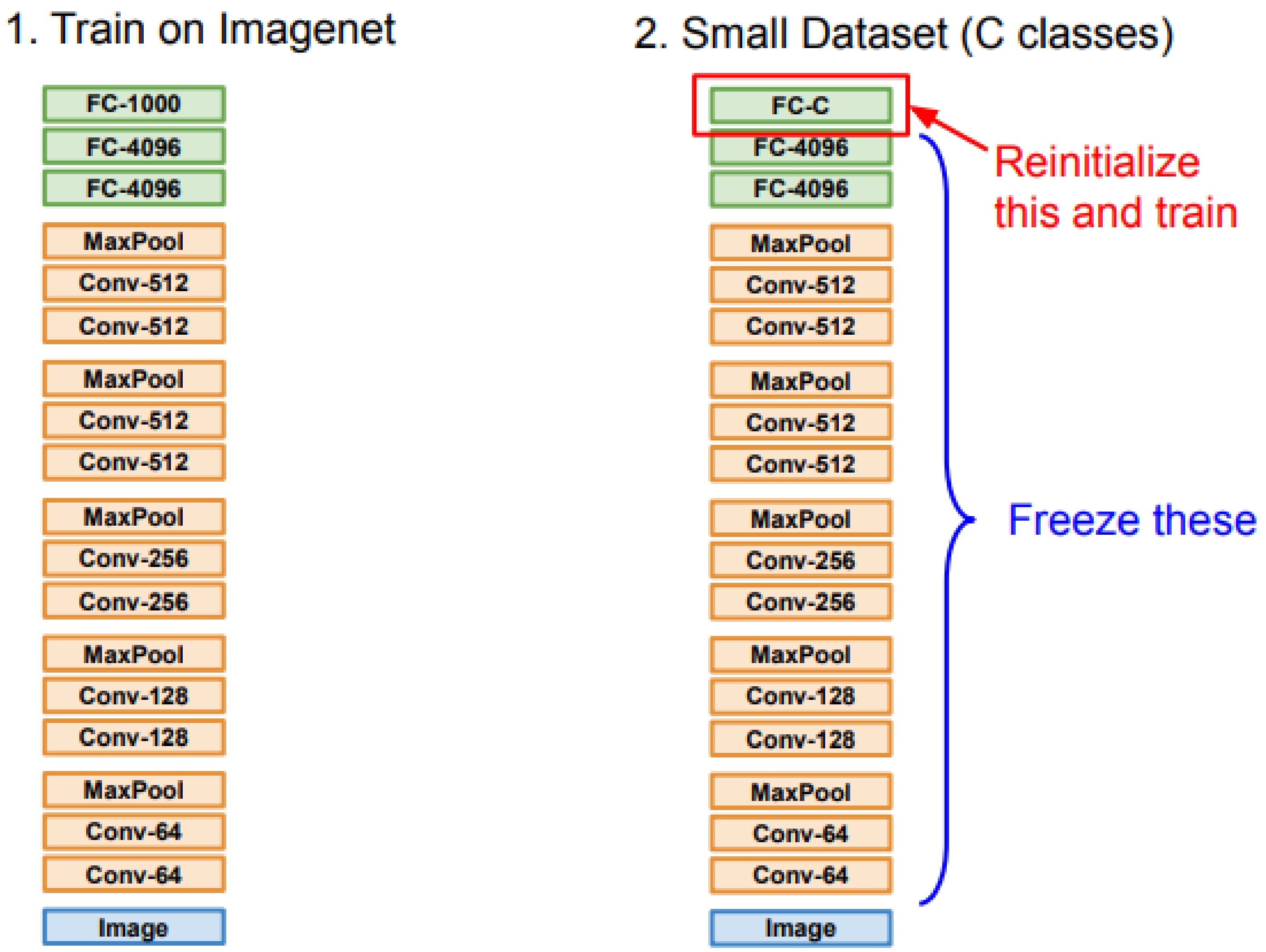
## Transfer Learning

•Transfer learning is a technique in machine learning where knowledge gained from training on one task is leveraged to improve performance on a different but related task.

•It involves using pre-trained models, which are neural networks trained on large-scale datasets, as a starting point for a new task.

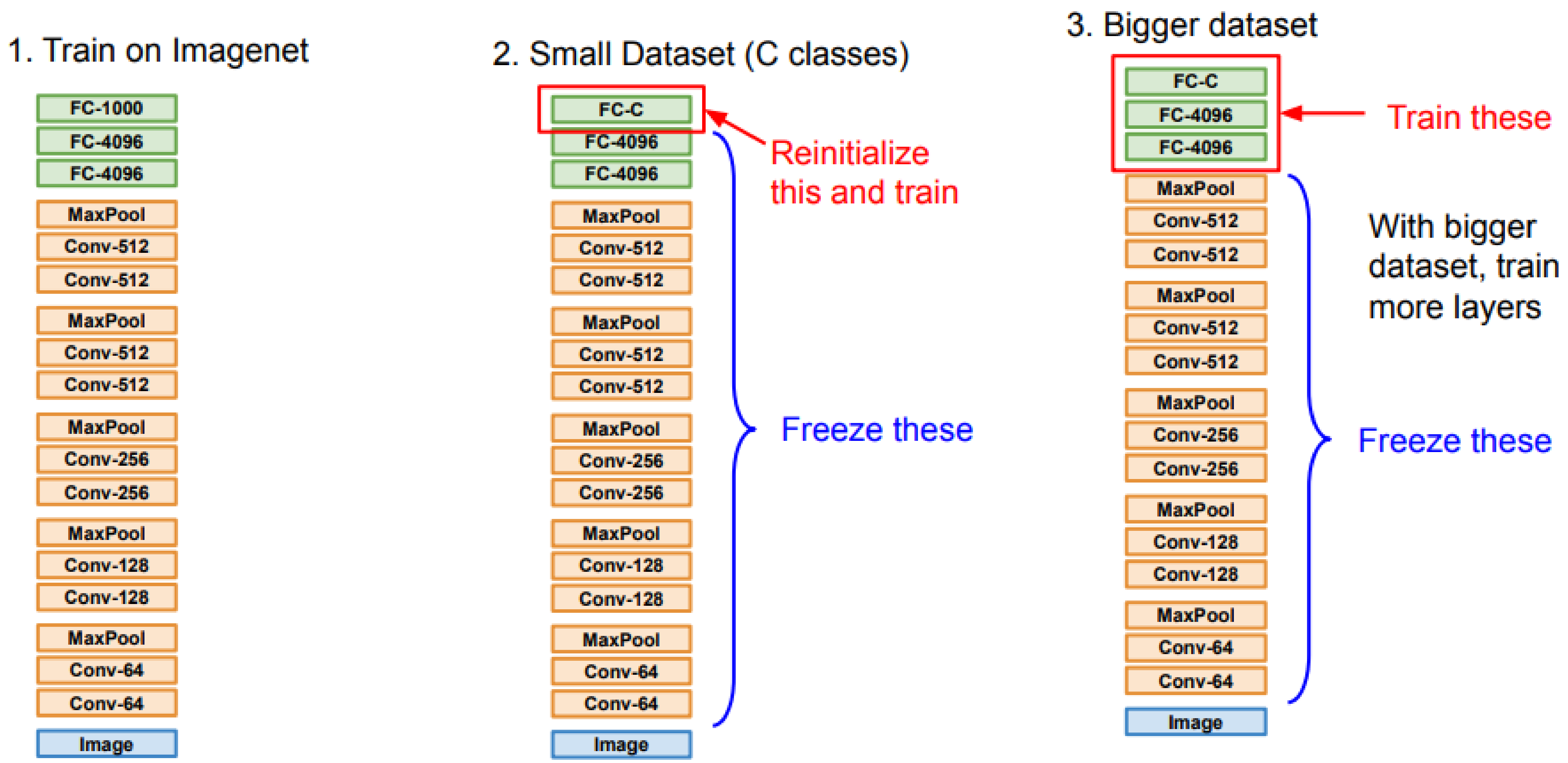
•By utilizing the learned representations and knowledge from the pre-trained model, transfer learning allows for faster convergence, better generalization, and improved performance, especially when the new task has limited data.

## Transfer Learning



Donahue et al, “DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition”, ICML 2014 Razavian et al, “CNN Features Off-the-Shelf: An Astounding Baseline for Recognition”, CVPR Workshops 2014

## Transfer Learning



Donahue et al, “DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition”, ICML 2014 Razavian et al, “CNN Features Off-the-Shelf: An Astounding Baseline for Recognition”, CVPR Workshops 2014

## Refs

The Vanishing Gradient Problem. The Problem, Its Causes, Its… | by Chi-Feng Wang | Towards Data Science

Lecture 5.pptx (stanford.edu)

Lecture 6.pptx (stanford.edu)

winter1516_lecture7.pdf (stanford.edu)

14.3 Learning Manifolds (buffalo.edu)

Maximum Likelihood Estimation - how neural networks learn | Chan`s Jupyter (goodboychan.github.io)

Convolutional neural network - Wikipedia

Separability and geometry of object manifolds in deep neural networks | Nature Communications