CONVOLUTIONAL NEURAL NETWORKS

\*Computer Vision

Computer vision, powered by deep learning and Convolutional Networks (ConvNets), enables machines to interpret and process visual data for applications such as

self-driving cars, facial recognition, and artistic creation

Problems-

1.image classification

2.neural style transfer

3.object detection

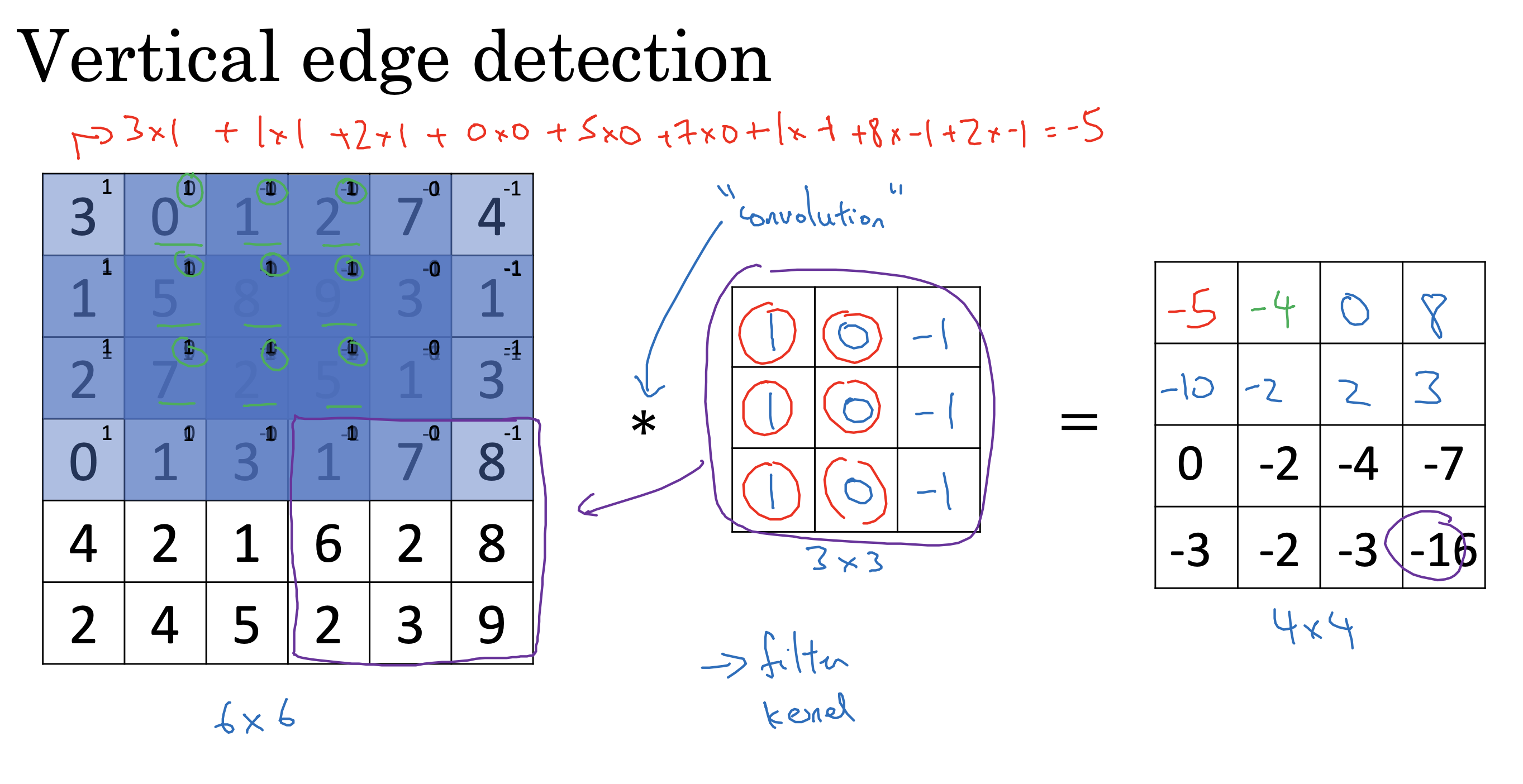
Problem with deep learning - when we work on large pixels/images(millions of input feature) , it is difficult to get enough data to prevent overfitting and requirement of large computational memory is infeasible ,

**\*CNN-Edge detection**

How to detect edges in the image?

1.Vertical edges

2.horizontal edges



The above image is the example from Andrew’s course,

6x6 grayscale image — input

* — convolution operator

3x3 matrix — filter

4x4— output image

General dimension - nxn \* fxf = (n-f+1)\*(n-f+1)

Where nxn = dimension of input image

fxf= filter dimension

**\*Types of filters**

**1.vertical filter**

|  |  |  |
| --- | --- | --- |
| 1 | 0 | -1 |
| 1 | 0 | -1 |
| 1 | 0 | -1 |

**2.horizontal filter**

|  |  |  |
| --- | --- | --- |
| 1 | 1 | 1 |
| 0 | 0 | 0 |
| -1 | -1 | -1 |

**3.sobel filter**

|  |  |  |
| --- | --- | --- |
| 1 | 0 | -1 |
| 2 | 0 | -2 |
| 1 | 0 | -1 |

**4.scharr filter**

|  |  |  |
| --- | --- | --- |
| 3 | 0 | -3 |
| 10 | 0 | -10 |
| 3 | 0 | -3 |

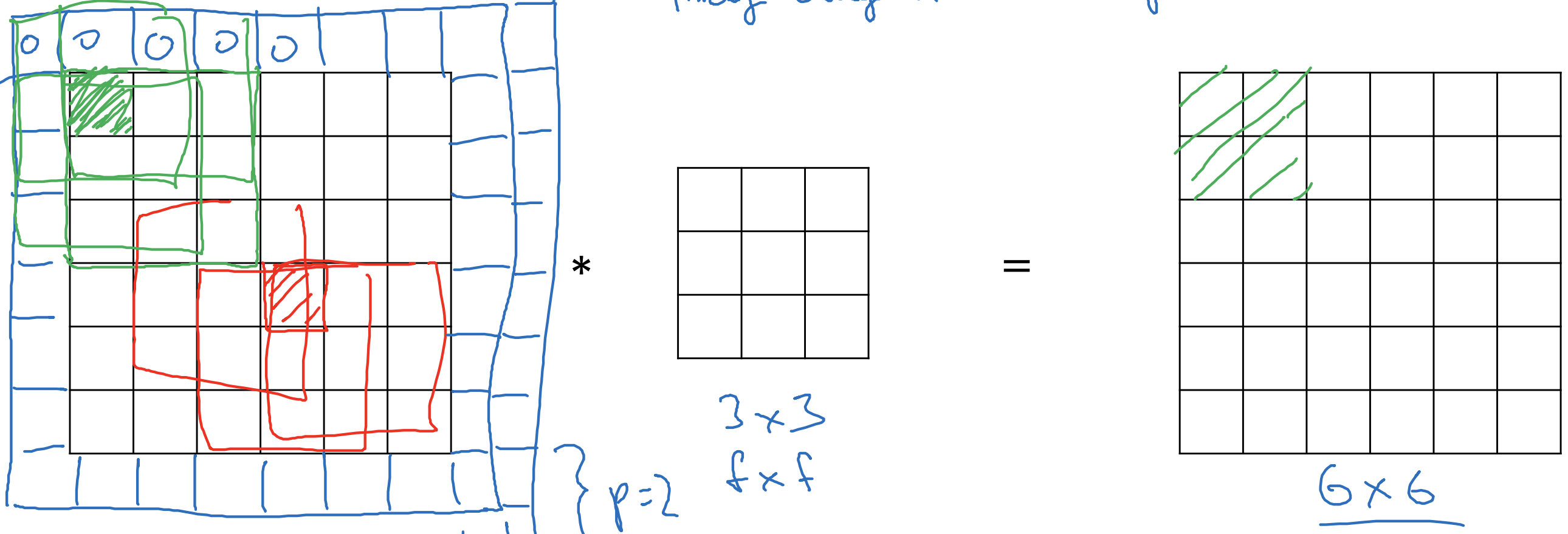
**\*Padding**

Disadvantage of using the edge detection method using only filters -

1.shrinkage in image everytime we apply convolution

2.throwing away a lot of information

So we pad the image to solve above problems.

Lets discuss this with following example-

We added a pixel on each edge of 6x6 input image

Now convolve new 8x8 image with 3x3 filter to get output as 6x6

p=padding amount

p=1 which means we are padding border with extra 1 pixel

General output = (n+2p-f+1)x(n+2p-f+1)

**\*Valid and same convolutions**

**1.valid** - no padding , dimensional output —-> nxn \* fxf = (n-f+1)x(n-f+1)

**2.same-** pad to get the same size output image ,

Dimensional output —->nxn \* fxf = (n+2p-f+1)x(n+2p-f+1)

f is generally odd (computer vision convention)

**\*Strided Convolution**

To shift the filter matrix by any other unit in both direction on inout image matrix

In previous examples stride =1

Dimensional output—-> nxn \* fxf = (n+2p-f/s +1)x(n+2p-f/s +1)

**\*Convolution over volume**

Ex- convolution on RGB images

**Applying 3D Filters**

RGB images are represented as height x width x channels (e.g., 6x6x3).

Use 3D filters (e.g., 3x3x3) that match the number of image channels.

**Convolution Operation**

Placement: Place the 3x3x3 filter at the top-left corner of the 6x6x3 image.

Multiplication: Multiply the filter's 27 parameters with the corresponding image pixels.

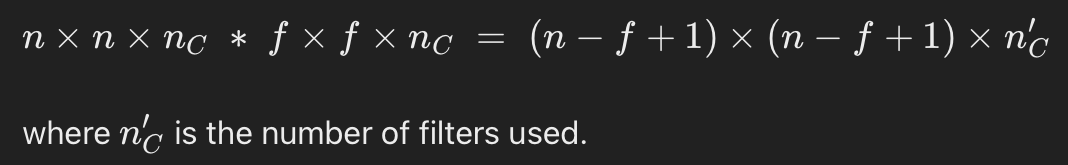
Summation: Sum the results to get a single value for the output matrix.

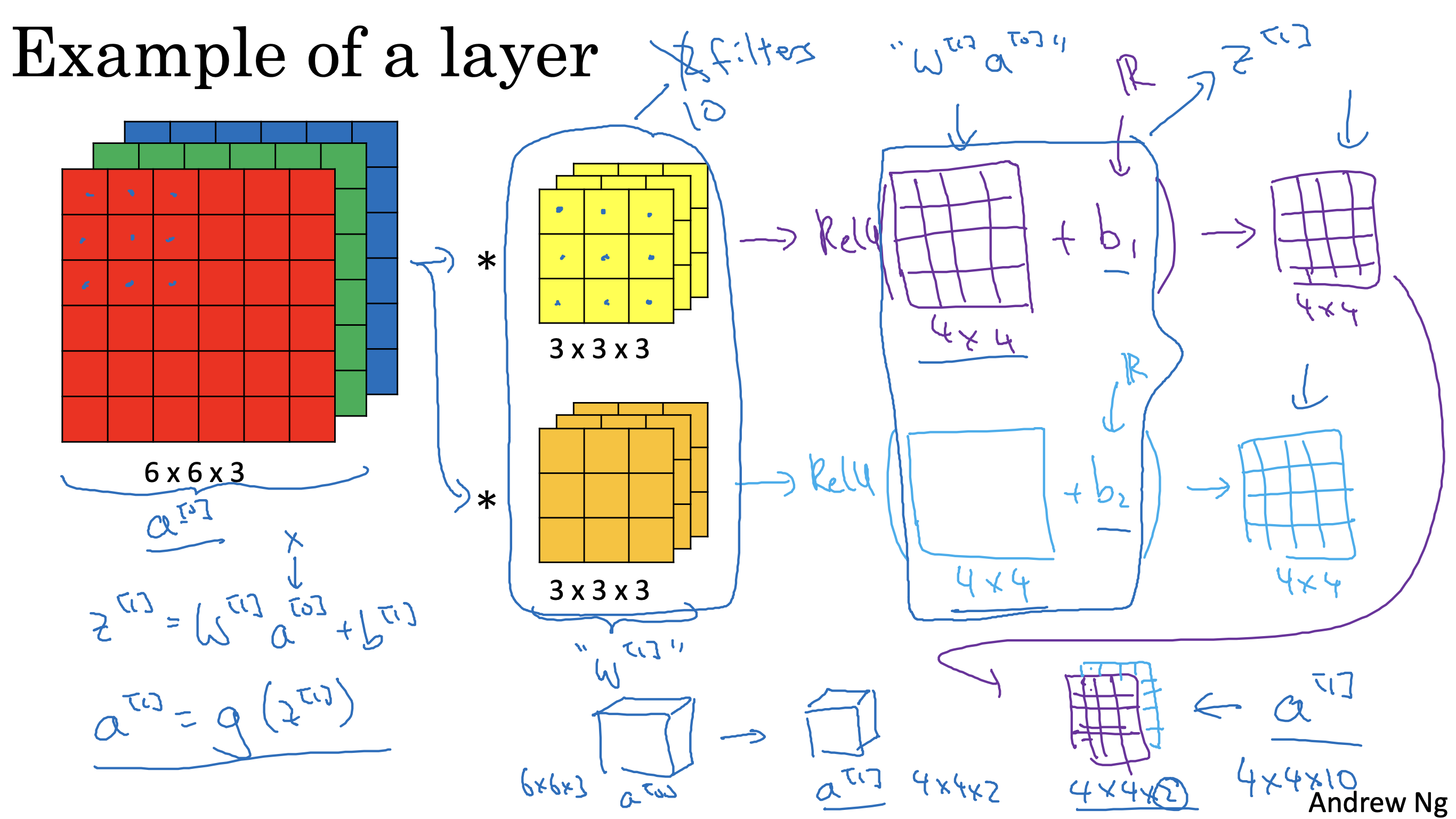
Sliding: Slide the filter across the image to compute the entire 4x4 output.

**Multi-Filter Convolutions**

Apply multiple filters to detect various features.

Ex: Two filters produce a 4x4x2 output by stacking two 4x4 results.

**Dimensional output**

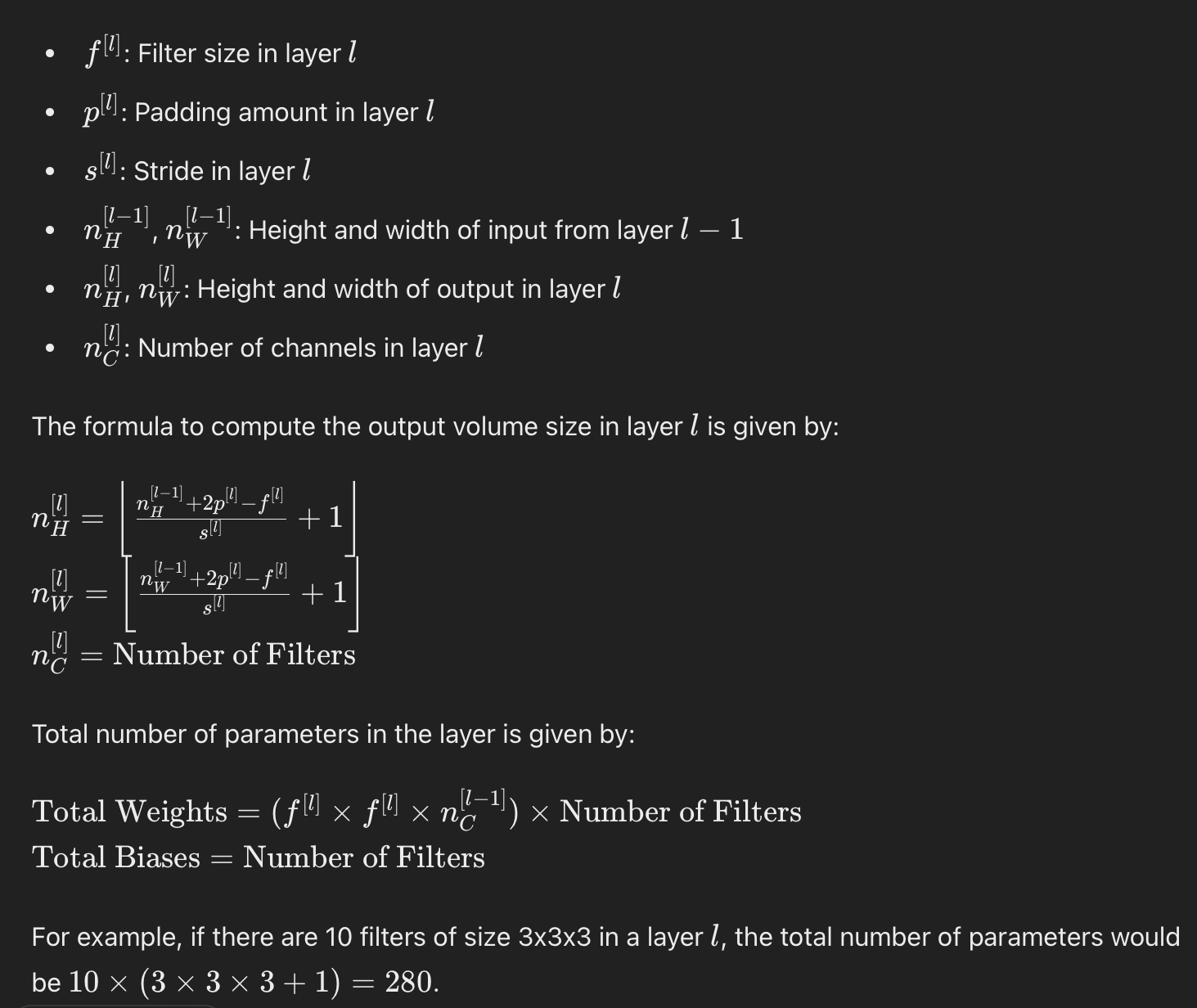
**\*Building a layer in CNN**we’ll use the example from Ng’s lecture

Convolve a 3D volume with multiple filters to generate different outputs. For instance, two filters yield two 4x4 outputs.

Bias and Non-Linearity: Each output includes bias addition and a non-linear activation (e.g., ReLU), resulting in a 4x4 output per filter.

Multi-Filter Outputs: Stack outputs from various filters to form a 4x4x2 volume, representing a single layer in a convolutional neural network.

Parameter Calculation: Each filter contains 27 parameters (3x3x3) plus bias, totaling 28 parameters. With ten filters, the layer comprises 280 parameters, irrespective of the input image size.

**Notation overview**

**\*Convolutional Network**

**\*types of layers-**

1.convolution(conv)

2.pooling(pool)

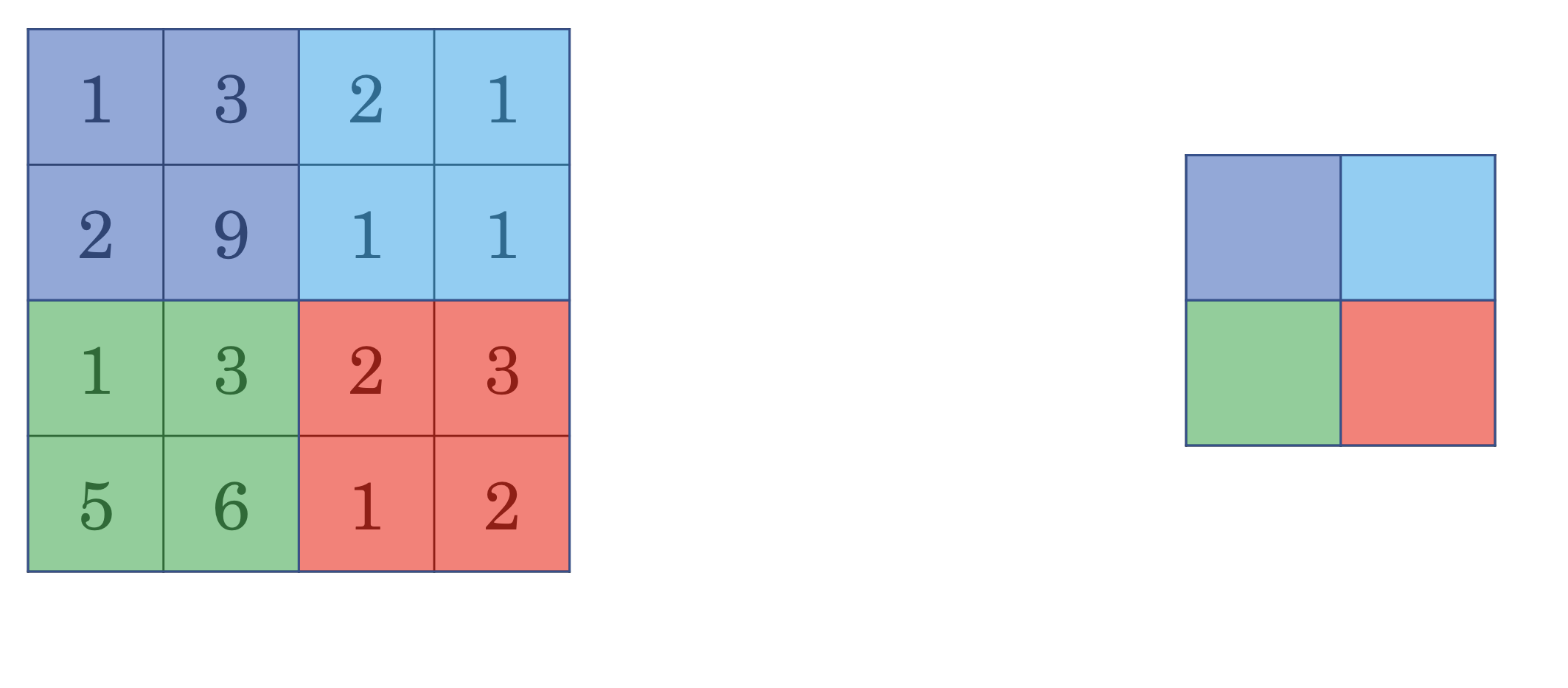
3.fully connected(fc)

**\*Pooling layer**

**\*Max pooling:**

Using pooling layers reduces the size of representation and make some features more robust.

For ex,



In the above example —-> filter f=2 ,stride=2 ,p=0 is used

2x2 output is generated by selecting the maximum of the input feature value from the respective 4 grids in input.(9 2

6 3)

There is no underlying reason to use max pooling in Convnets. In experiments it was found it works well.

**Property:** it has set of hyperparameters but no parameters to learn, So there is nothing for gradient descent to learn one we fit f and s .

It’s fixed omputation and gradient descent doesn’t change anything.

**\*Average pooling**

Instead of maxes in each filter you take average .

It is used to collapse your representation.

**\*NN**

Ex-

Input —>Conv1—>pool1—>Conv2—>pool2—>fc3—>fc4—>fc5—>softmax

**\*Why Convolutions?**

Conv net has smaller no. of parameters and allow-

1.parameter sharing

2.sparsity of connections

**\*CASE STUDIES**

Filter size =f , stride = s , padding=p

**\*LeNet-5**

Purpose: Designed for handwritten digit recognition.

Input: 32x32x1 grayscale image.

Layers:

Conv1 : 6 filters,f=5, s=1, no p. Output: 28x28x6.

Avg pool1: f=2, s=2. Output: 14x14x6.

Conv 2: 16 filters,f=5, s=1, no p. Output: 10x10x16.

Avg pool2: f=2, s=2. Output: 5x5x16.

FC1: 400 units connected to 120 neurons.

FC2: 84 neurons.

Output: 10 neurons (one for each digit 0-9).

Parameters: About 60,000 parameters.

**\*AlexNet**

Purpose: Designed for large-scale image classification.

Input: 227x227x3 RGB image.

Layers:

Conv1: 96 filters,f=11, s=4. Output: 55x55x96.

Max Pool1: f=3, s=2. Output: 27x27x96.

Conv2: 256 filters,f=5, s=1, same p. Output: 27x27x256.

Max Pool2: f=3, s=2. Output: 13x13x256.

Conv3: 384 filters,f=3, s=1, same p. Output: 13x13x384.

Conv4: 384 filters,f=3, s=1, same p. Output: 13x13x384.

Conv5: 256 filters,f=3, s=1, same p. Output: 13x13x256.

Max Pool3: f=3, s=2. Output: 6x6x256.

FC1: 4096 units.

FC2: 4096 units.

Output: 1000-way softmax.

Parameters: About 60 million parameters.

**\*VGGNet (VGG-16)**

Purpose: Simplified network design with uniform architecture.

Input: 224x224x3 RGB image.

Layers:

Conv layers: Series of filters with f=3, s=1, and same p.

Pooling Layers: 2x2 filters with s=2, reducing height and width by half.

Conv Layers:

I block: 1 layers with 64 filters.

2 block: 2 layers with 128 filters.

3 block: 3 layers with 256 filters.

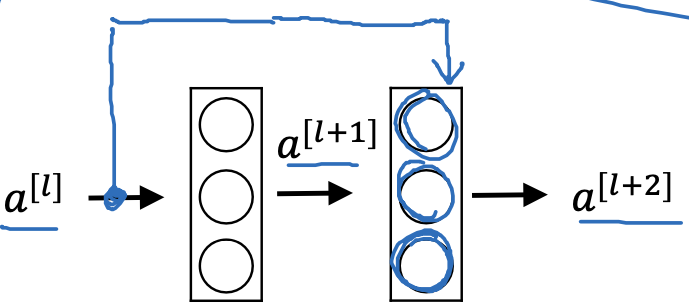
4 block: 3 layers with 512 filters.

5 block: 3 layers with 512 filters.

Fully Connected Layers: 2layers with 4096 units each.

Output Layer: 1000-way softmax.

Parameters: About 138 million parameters.

**\*Residual block**

Information in above flows from to

Map for above ex,

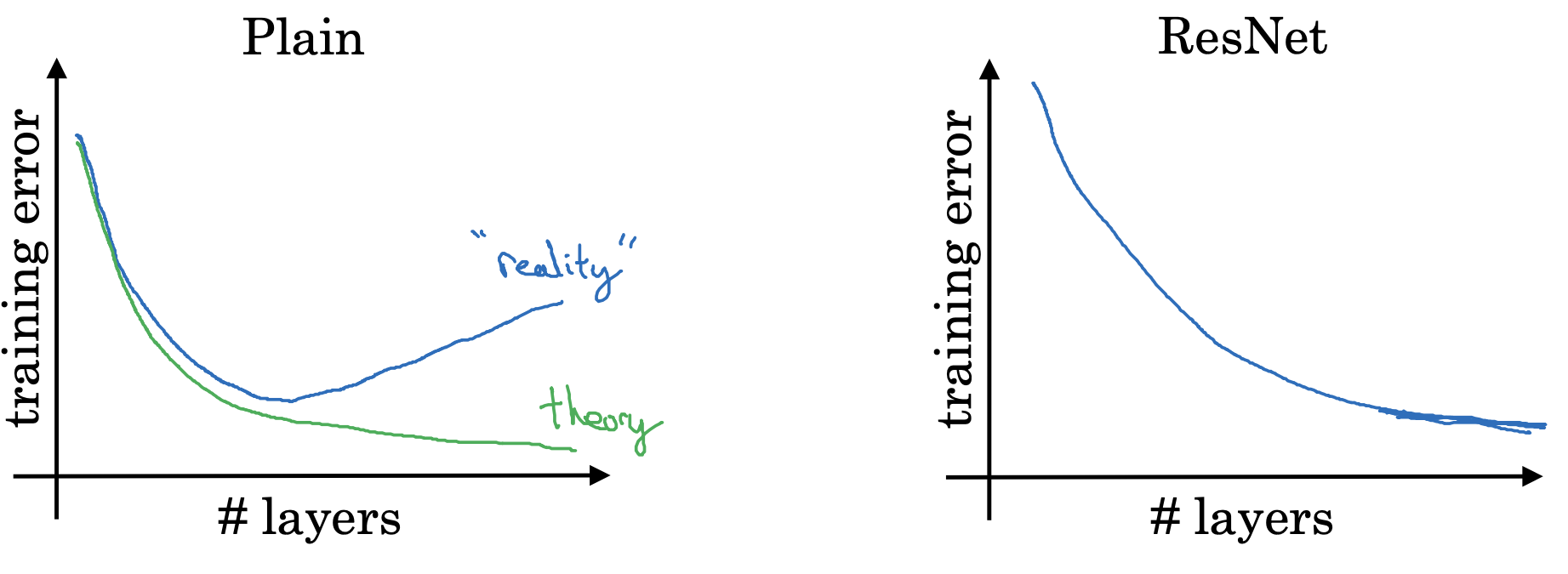
—>linear—>ReLu(—>linear—>ReLu—>

In residual layer,

fast forward , is copied into much further in NN.

follows the shorcut/skip connection to go deeper in the network

Now:



Resent solves the problem of vanishing and exploding gradient and allows to train much deeper in the network without loss in performance.

In plain:optimization algorithm face hard time as we go deeper in the network.

**\*1x1 convolution**

One-by-one convolutions are used to perform non-trivial operations on input volumes, particularly for reducing the number of channels or adding nonlinearity to the network.

**\*Inception Network**

Utilizes multiple filter sizes (1x1, 3x3, 5x5) and pooling layers simultaneously.

Combines outputs from different paths to improve network performance.

Inception Layer Implementation:

Allows for diverse filter sizes and layer types within a single layer.

Concatenates outputs from different paths to create the final layer output.

Computational Cost Challenge:

Original Inception layer design can be computationally expensive due to multiple operations.

Bottleneck Layer Optimization:

Introduces a bottleneck layer with 1x1 convolutions to reduce computational cost.

Shrinks the volume size before applying larger filters, maintaining performance while saving computation.

**\*Inception Network Architecture:**

Consists of repeated Inception blocks in different positions within the network.

Involves additional side-branches for regularization and preventing overfitting.

Collaborative effort by Google researchers, leading to variations like Inception v2, v3, v4.

**\*MobileNets:**

Designed for low-compute environments like mobile devices.

Aims to reduce computational costs while retaining performance.

**Depthwise Separable Convolution:**

Breaks convolution into two steps: depthwise convolution and pointwise convolution.

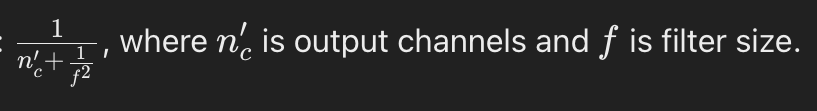
Depthwise convolution operates on individual channels, reducing redundancy.

**Pointwise convolution** combines depthwise outputs efficiently.

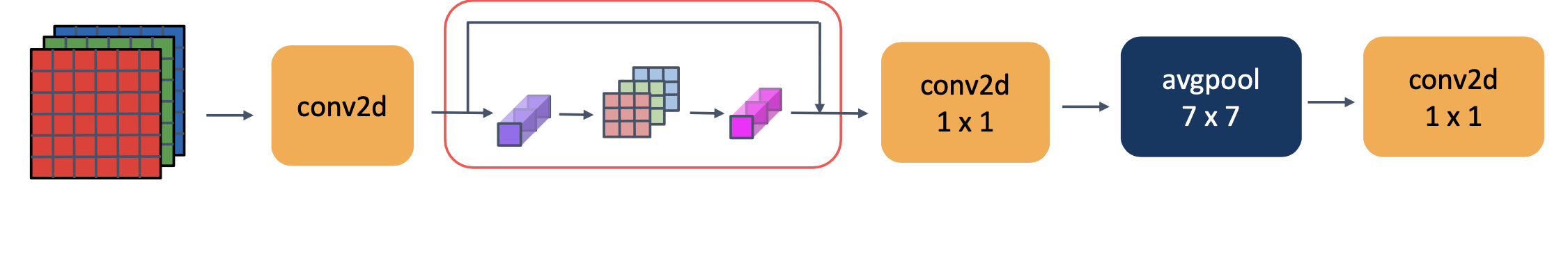
**Computational Cost Comparison:**

Normal convolution requires extensive computations based on filter size and channels.

Depthwise separable convolution significantly reduces computations, often by 60-70%.

Cost Reduction Formula:

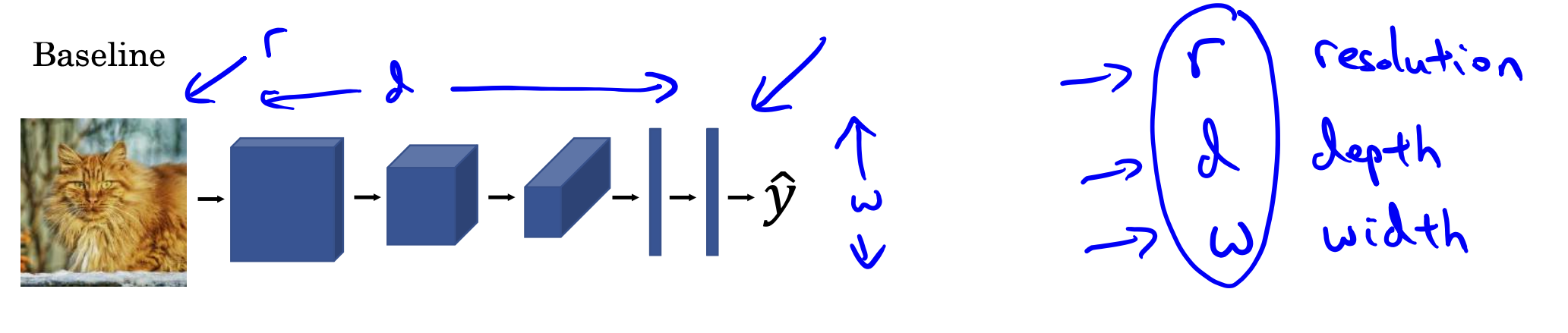
Offers substantial computational savings, crucial for mobile and edge devices.

Below is mobile Net architecture(V2)

**\*Efficient Net**

Purpose: EfficientNet adjusts resolution (r), depth (d), and width (w) of neural networks for optimal performance within a specified computational budget.

Scaling Factors: It determines the best trade-off between r, d, and w based on device constraints, ensuring efficient resource utilization.

Compound Scaling: Allows simultaneous adjustments to resolution, depth, and width, optimizing performance under resource limitations.

**\*Transfer learning**

Transfer learning involves leveraging pre-trained neural network weights for initializing new networks on different tasks. This practice is prevalent in computer vision, utilizing datasets like ImageNet for extensive pre-training. Modifications, such as custom softmax layers, adapt pre-trained architectures to specific tasks like cat detection. Freezing early layers preserves pre-trained knowledge, while selective training optimizes computation. This approach accelerates model development, especially with smaller datasets, by tapping into pre-existing expertise and resources.

**\*Data augmentation**

Data augmentation is vital in computer vision, diversifying training data to improve model performance. Techniques like mirroring, random cropping, and color shifting enhance dataset variability. Parallel processing threads handle data loading, augmentation, and training, optimizing model learning. Advanced methods like PCA Color Augmentation further boost model robustness against color variations.