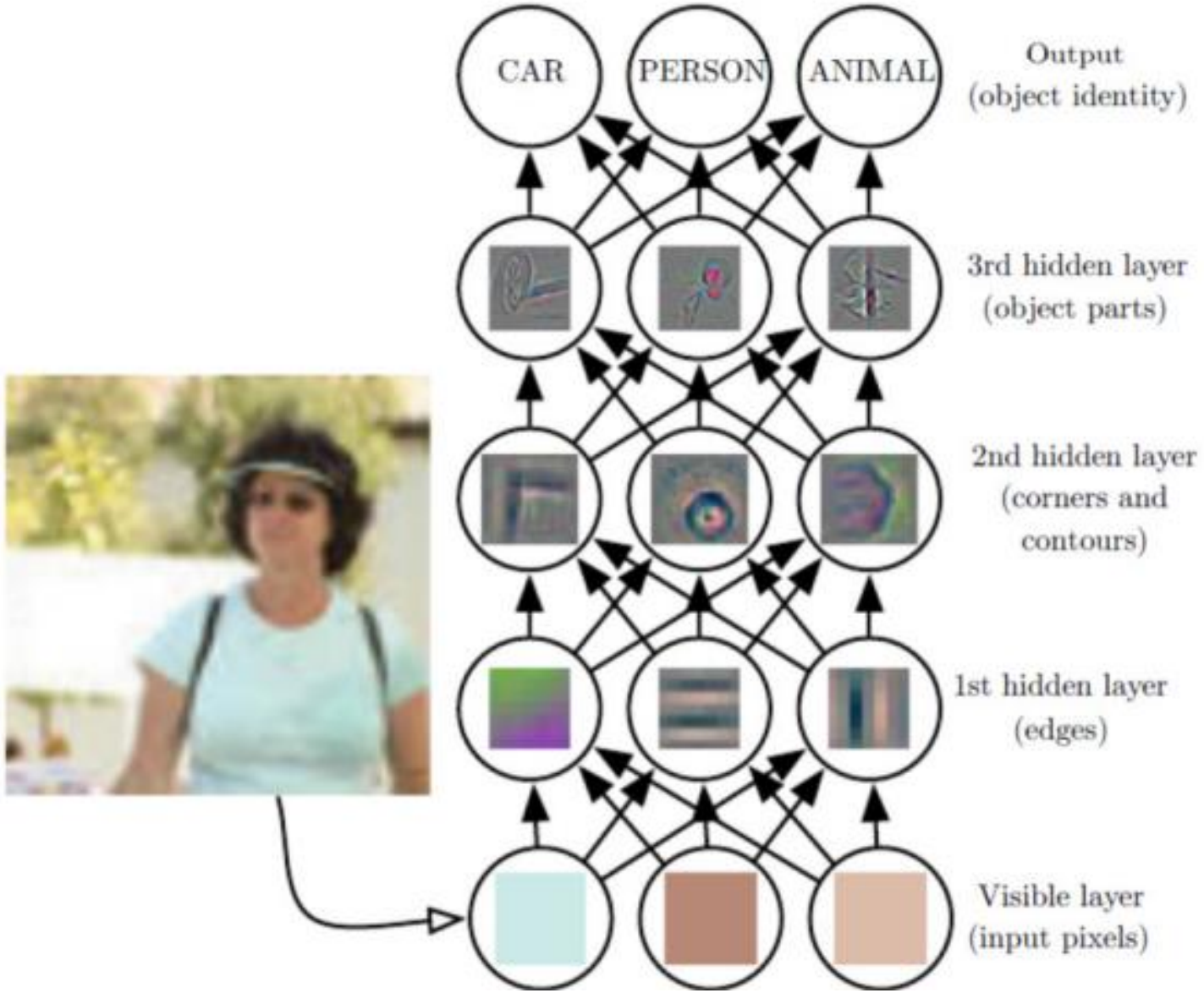
CSL446
Neural Network and Deep Learning
Module 2

Dr. Snehal B Shinde

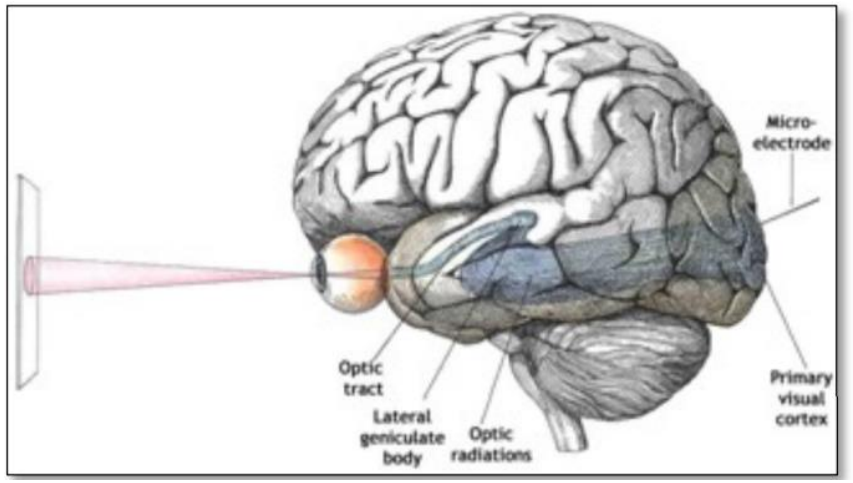
Computer Science and Engineering

Indian Institute of Information Technology, Nagpur.

Human Vision - many layers of abstraction - Deep learning

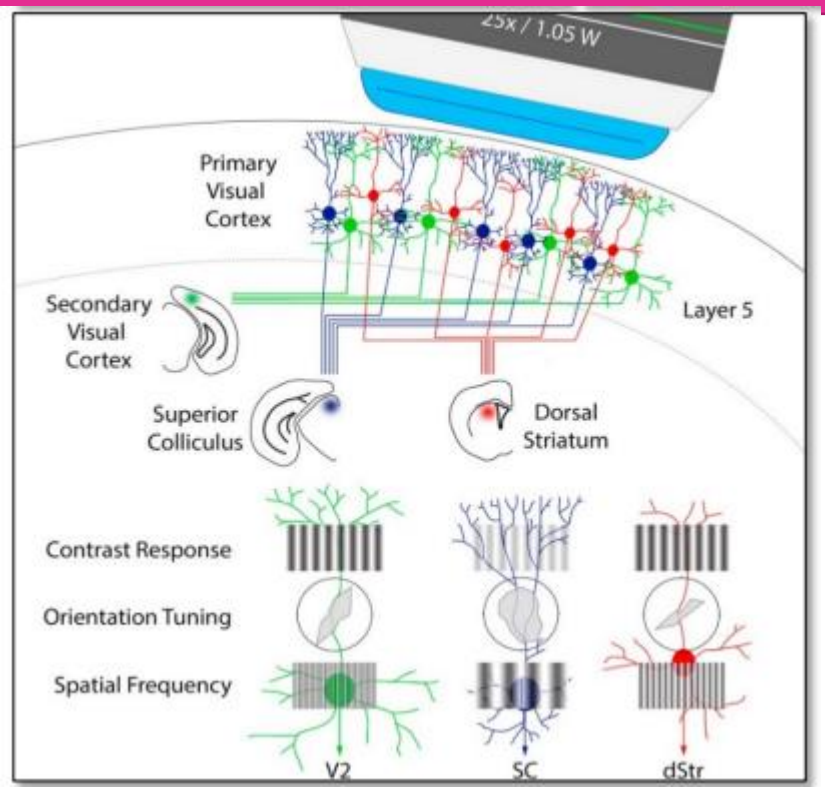


CNN inspiration in the 50s/60s: human/animal visual cortex



Visual Pathway: Shows how visual signals travel from the eyes through the optic nerves to different areas of the brain, primarily the **primary visual cortex** (V1).

Hubel and Wiesel's Research (1968) conducted foundational research using cats and monkeys, uncovering critical aspects of the visual cortex:



Receptive Fields: Neurons in the visual cortex respond to specific regions of the visual field, enabling localized computations.

Simple Cells: Detect basic visual features like edges and orientations.

Complex Cells: Integrate inputs from simple cells to recognize patterns regardless of position (position invariance).

CNN inspiration in the 50s/60s: human/animal visual cortex

Hierarchical Structure of Vision

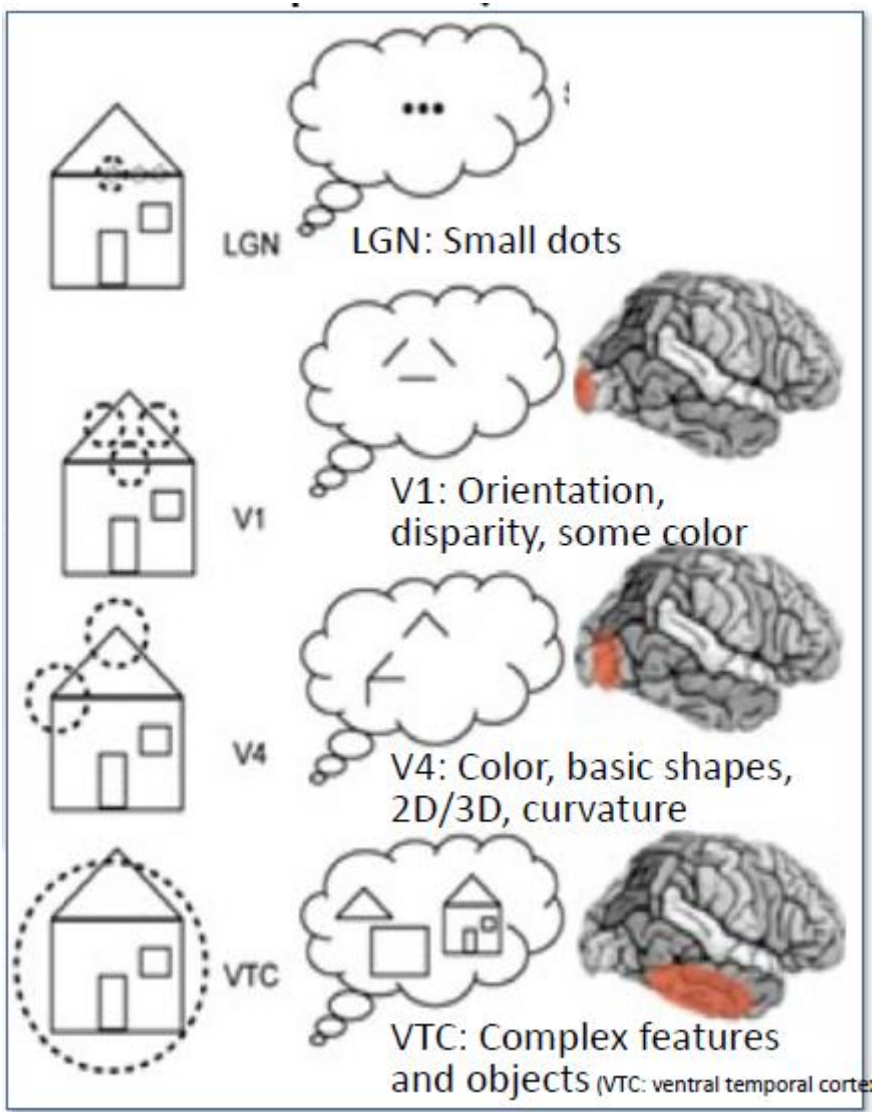
1. **Low-Level Processing (Pixel-Level) (detect edges):**
   - The first layers of processing detect simple visual components like edges and orientations (contrast, slant, etc.).

2. **Mid-Level Processing (shapes or textures):**
   - Shapes and primitives are recognized as inputs are combined.

3. **High-Level Processing (recognize complex objects):**
   - These primitives are abstracted into larger patterns, enabling recognition of objects and scenes.

86 billion neurons, each connects to 10k neurons, 1 quadrillion (1012) connections

Abstraction layers: edges, bars, dir., shapes, objects, scenes



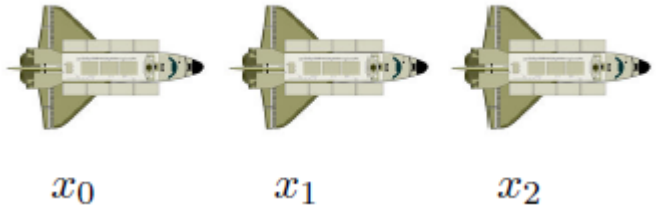
- **LGN (Lateral Geniculate Nucleus):**
  - The first stage of visual processing in the brain after the retina.
  - Detects **small dots** and basic luminance differences (light/dark contrasts are processed).

- **V1 (Primary Visual Cortex):**
  - Processes **orientation**, **disparity**, and **some color information**.
  - Detects lines, bars, and edges at specific orientations.

- **V4 (Higher Visual Cortex):**
  - Processes **color**, **basic shapes**, **2D/3D curvature**, and form.
  - Integrates information from earlier layers to represent more complex patterns.

- **VTC (Ventral Temporal Cortex):**
  - Processes **complex features and objects**.
  - Recognizes objects, scenes, and more abstract concepts.
  - This stage corresponds to the deeper layers in CNNs, which detect object-level features.

The convolution operation



x_0 x_1 x_2

$$s_t = \sum_{a=0}^{6} x_{t-a} w_{-a}$$

	w_{-6}	w_{-5}	w_{-4}	w_{-3}	w_{-2}	w_{-1}	w_0
W	0.01	0.01	0.02	0.02	0.04	0.4	0.5

X	1.00	1.10	1.20	1.40	1.70	1.80	1.90	2.10	2.20	2.40	2.50	2.70

S					1.80		

	w_{-6}	w_{-5}	w_{-4}	w_{-3}	w_{-2}	w_{-1}	w_0
W	0.01	0.01	0.02	0.02	0.04	0.4	0.5

X	1.00	1.10	1.20	1.40	1.70	1.80	1.90	2.10	2.20	2.40	2.50	2.70

S					1.80	1.96	2.11	2.16	2.28	2.42

- Suppose we are tracking the position of an aeroplane using a laser sensor at discrete time intervals.
- Now suppose our sensor is noisy. To obtain a less noisy estimate we would like to average several measurements.
- More recent measurements are more important so we would like to take a weighted average.
- In practice, we would only sum over a small window. The weight array (w) is known as the filter.
- We just slide the filter over the input and compute the value of St based on a window around Xt.
- Here the input (and the kernel) is one dimensional

Can we use a convolutional operation on a 2D input also?

The convolution operation (connect patches of input to neurons in hiddenlayer.)

We can think of images as 2D inputs

We would now like to use a 2D filter (m *n).

First let us see what the 2D formula looks like This formula looks at all the preceding neighbours (i –a, j -b)

$$S_{ij} = (I * K)_{ij} = \sum_{a=0}^{m-1} \sum_{b=0}^{n-1} I_{i-a,j-b} K_{a,b} I_{i+a,j+b} K_{a,b}$$

In practice, we use the following formula which looks at the succeeding neighbours

Input

a	b	c	d
e	f	g	h
i	j	k	ℓ

Kernel

w	x
y	z

Output

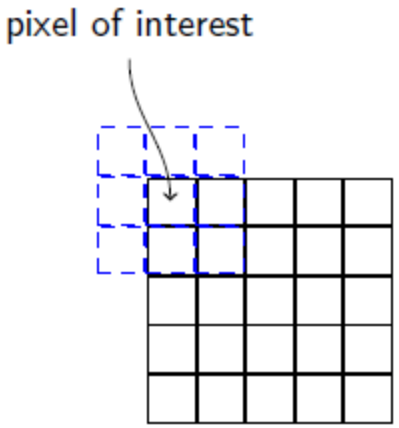
aw+bx+ey+fz	bw+cx+fy+gz	cw+dx+gy+hz
ew+fx+iy+jz	fw+gx+jy+kz	gw+hx+ky+ℓz

The convolution operation

$$S_{ij} = (I * K)_{ij} = \sum_{a=\lfloor -\frac{m}{2} \rfloor}^{\lfloor \frac{m}{2} \rfloor} \sum_{b=\lfloor -\frac{n}{2} \rfloor}^{\lfloor \frac{n}{2} \rfloor} I_{i-a,j-b} K_{\frac{m}{2}+a,\frac{n}{2}+b}$$

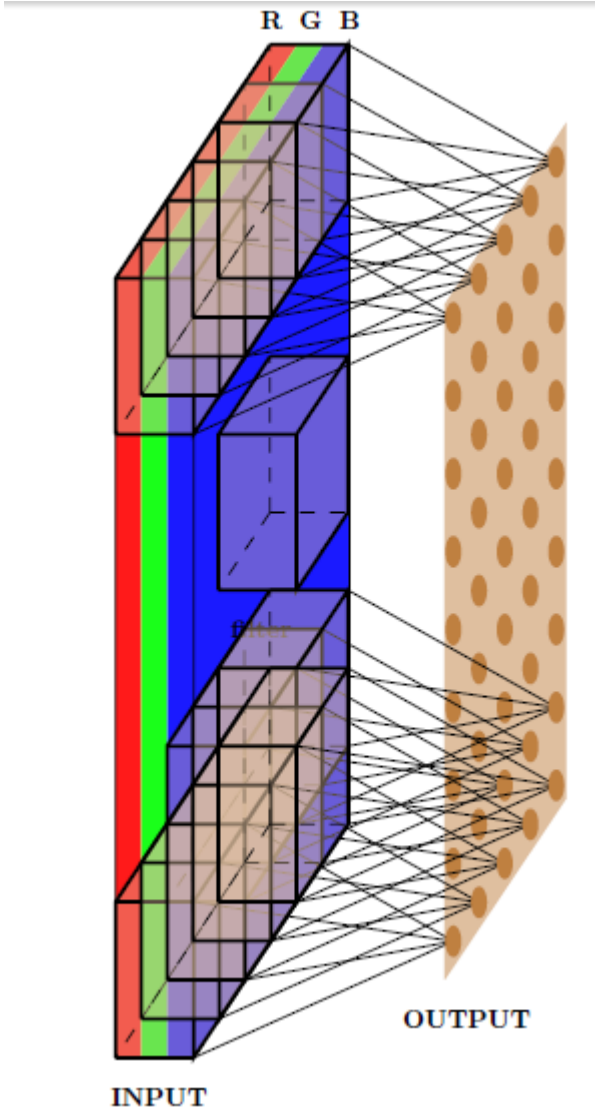
In other words we will assume that the kernel is centered on the pixel of interest.

So we will be looking at both preceeding and succeeding neighbors



Neuron connected to region of input. Only “sees” these values.

The convolution operation



- In the 1D case, we slide a one dimensional filter over a one dimensional input
- In the 2D case, we slide a two dimensional filter over a two dimensional output
- What would happen in the 3D case?
- It will be 3D and we will refer to it as a volume.
- In effect, we are doing a 2D convolution operation on a 3D input (because the filter moves along the height and the width but not along the depth).
- As a result the output will be 2D (only width and height, no depth).
- Once again we can apply multiple filters to get multiple feature maps

some examples of 2D convolutions applied to images

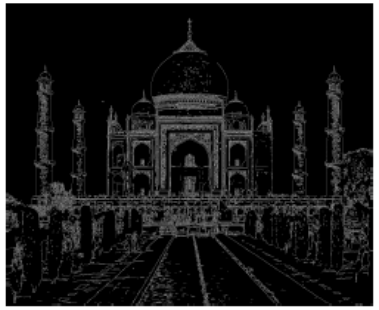
$$* \begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix} =$$

blurs the image

$$* \begin{matrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{matrix} =$$

sharpens the image

$$* \begin{matrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{matrix} =$$

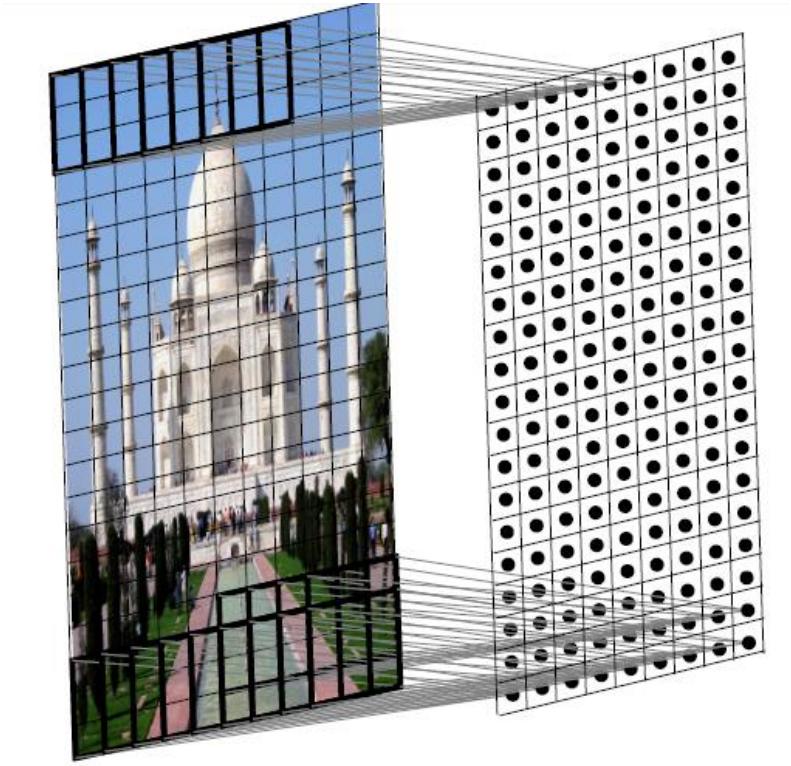


detects the edges

Simple Kernels / Filters

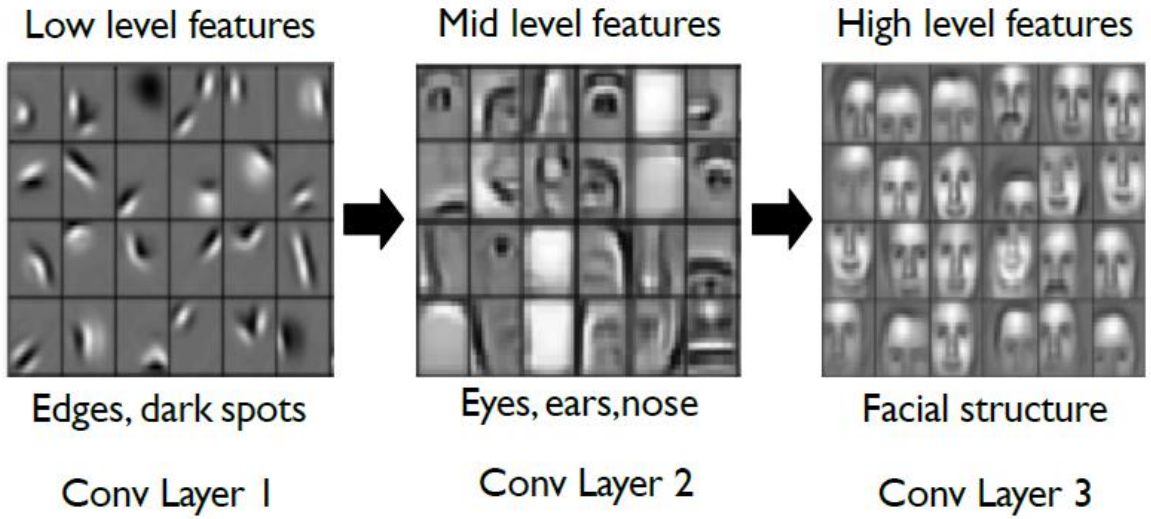
Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

The convolution operation

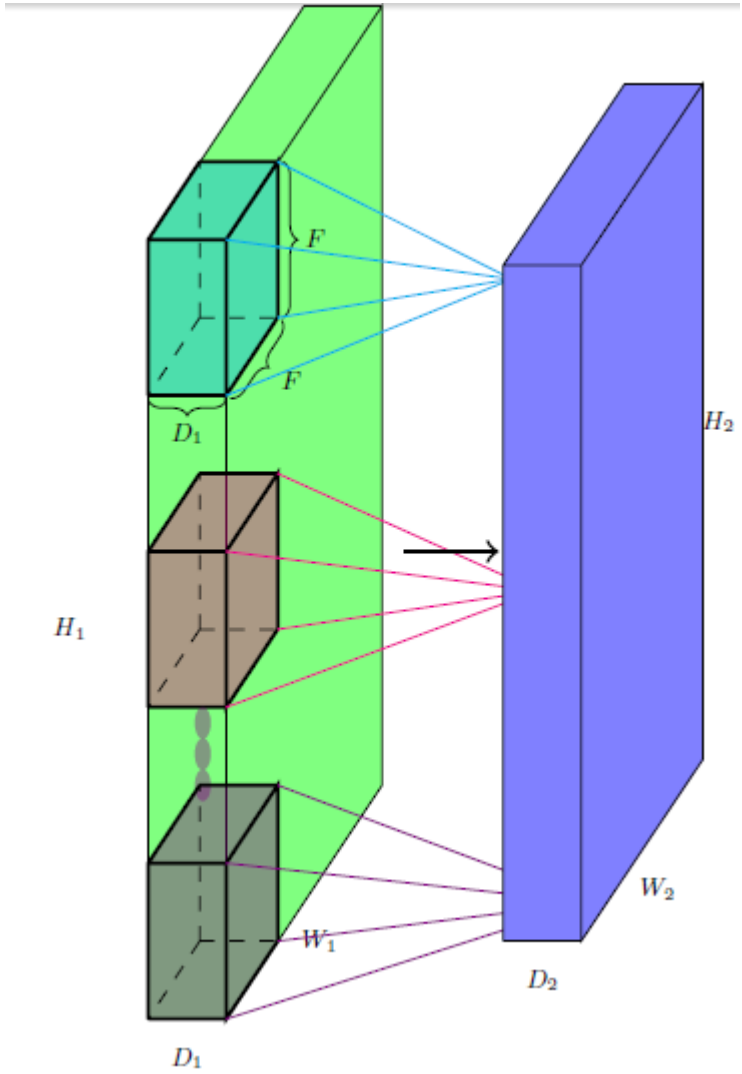


- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output
- The resulting output is called a feature map.
- We can use multiple filters to get multiple feature maps.

Representation Learning in Deep CNNs

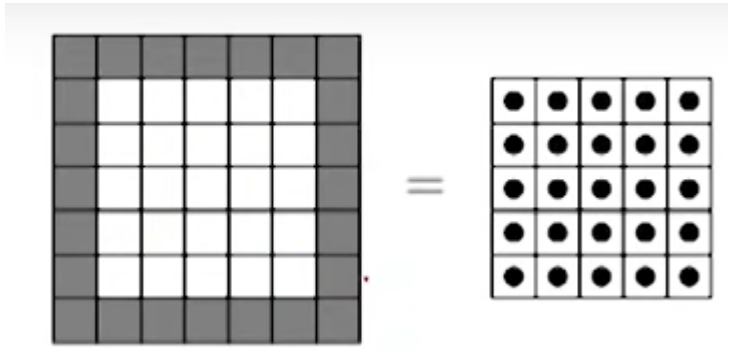


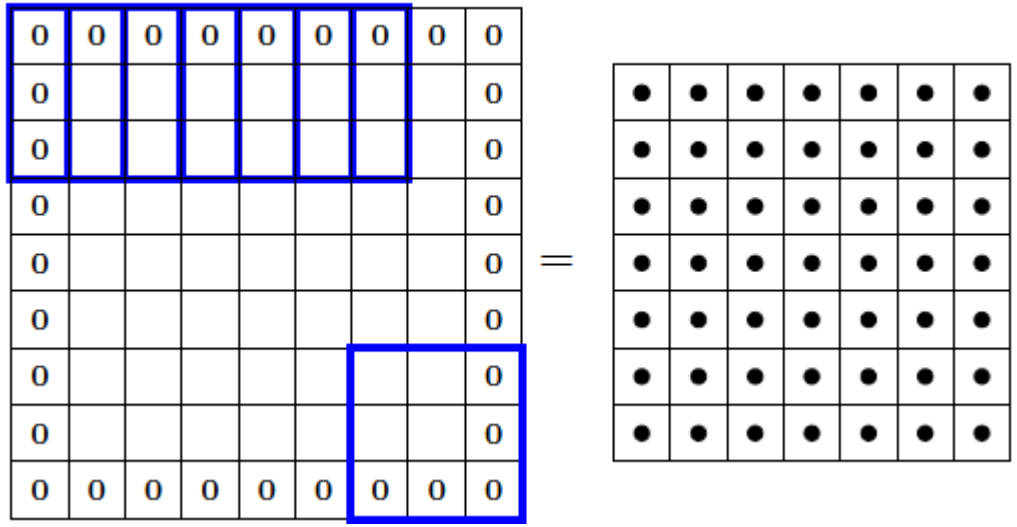
The convolution operation



- Width (W1), Height (H1) and Depth (D1) of the original input
- The number of filters K. The spatial extent (F) of each filter (the depth of each filter is same as the depth of each input)
- The output is W2 *H2 *D2 (we will soon see a formula for computing W2, H2 and D2)
- Notice that we can't place the kernel at the corners as it will cross the input boundary
- This results in an output which is of smaller dimensions than the input.
- As the size of the kernel increases, this becomes true for even more pixels

Relation between input, filter size, and output size

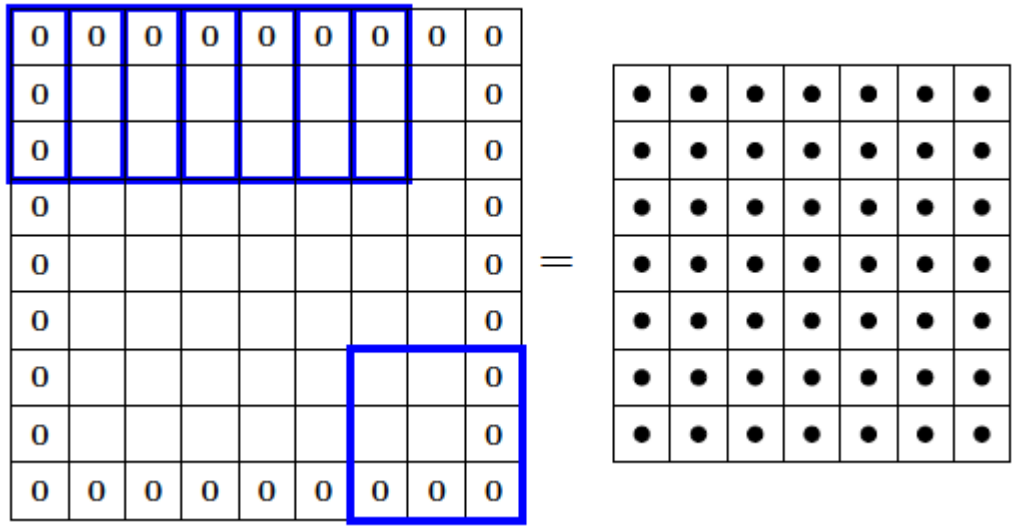




- Width (W1), Height (H1) and Depth (D1) of the original input
- The number of filters K. The spatial extent (F) of each filter (the depth of each filter is same as the depth of each input)
- The output is W2 *H2 *D2 (we will soon see a formula for computing W2, H2 and D2)
- Notice that we can't place the kernel at the corners as it will cross the input boundary
- This results in an output which is of smaller dimensions than the input.
- As the size of the kernel increases, this becomes true for even more pixels

In general, $W_2 = W_1 - F + 1$, $H_2 = H_1 - F + 1$

What if we want the output to be of same size as the input?



- We can use something known as padding
- Pad the inputs with appropriate number of 0 inputs so that you can now apply the kernel at the corners
- Let us use pad P = 1 with a 3 * 3 kernel
- This means we will add one row and one column of 0 inputs at the top, bottom, left and right

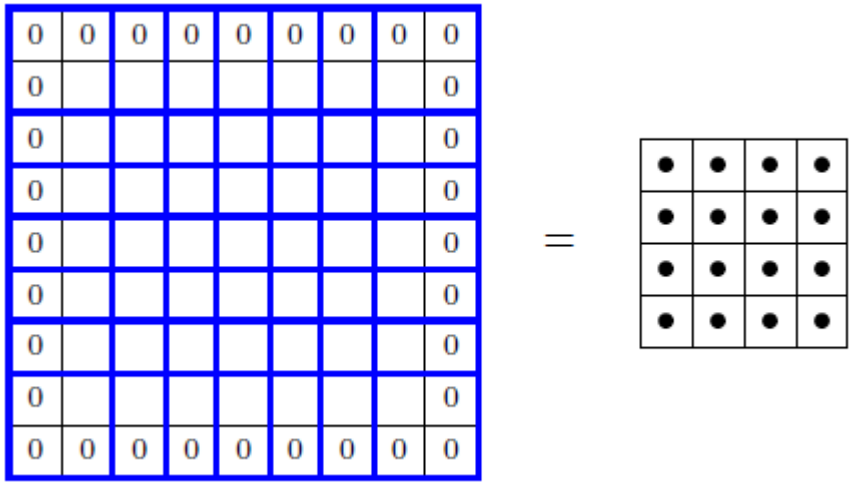
We now have,

$$W_2 = W_1 - F + 2P + 1$$

$$H_2 = H_1 - F + 2P + 1$$

We will refine this formula further

What does the stride S do?

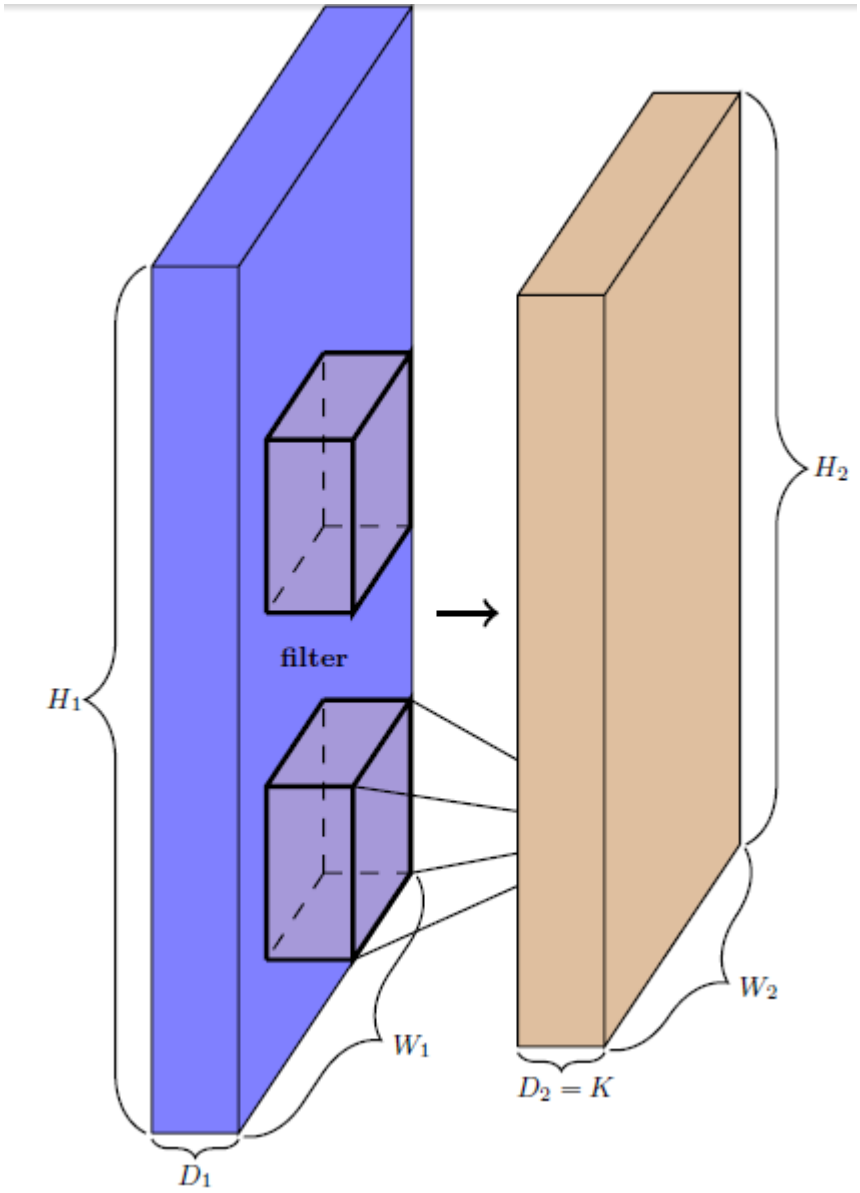


$$W_2 = \frac{W_1 - F + 2P}{S} + 1$$

$$H_2 = \frac{H_1 - F + 2P}{S} + 1$$

- It defines the intervals at which the filter is applied (here S = 2)
- Here, we are essentially skipping every 2nd pixel which will again result in an output which is of smaller dimensions

What does the stride S do?



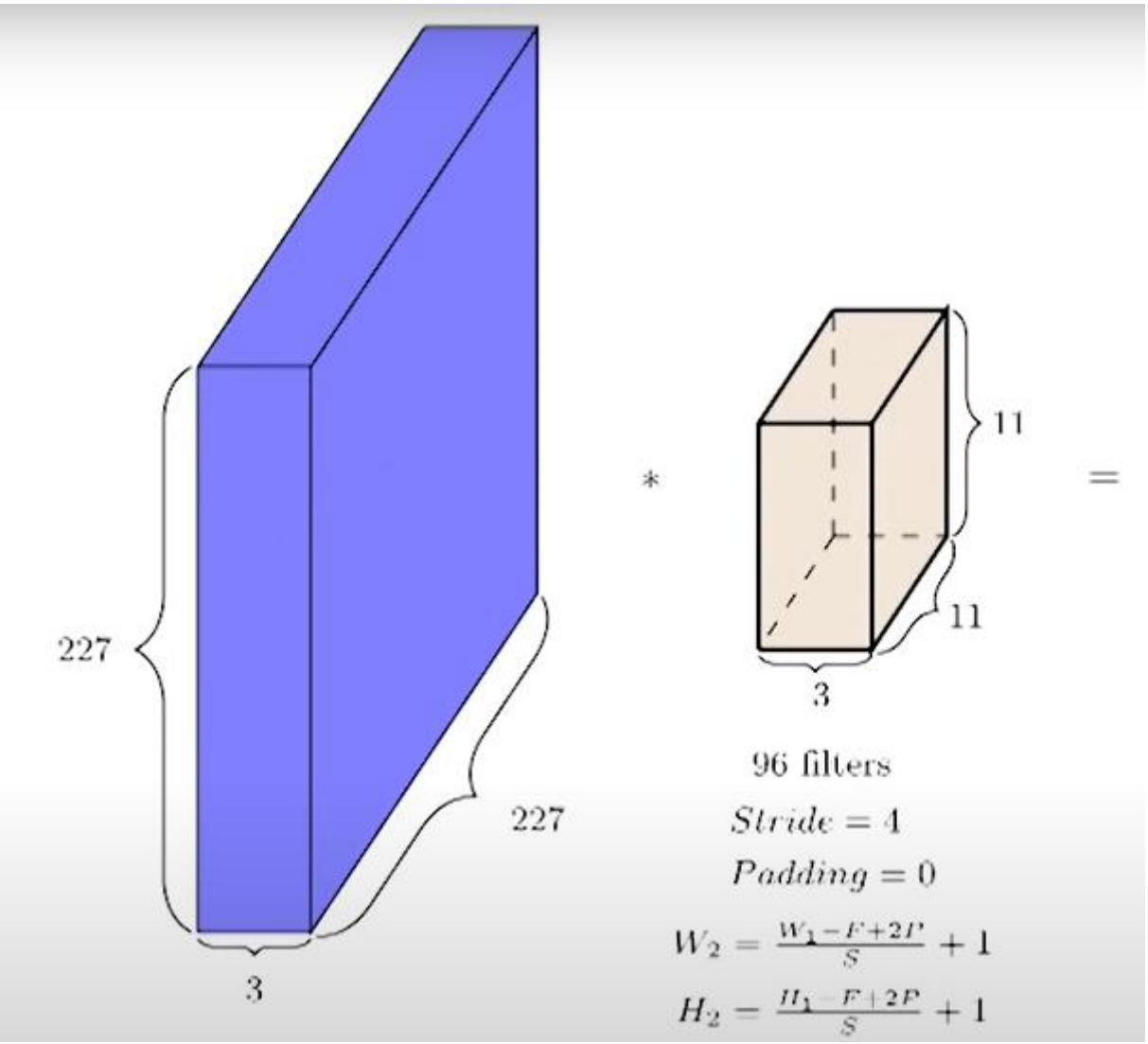
- Each filter gives us one 2D output.
- K filters will give us K such 2D outputs
- We can think of the resulting output as K * W2 * H2 volume
- Thus D2 = K

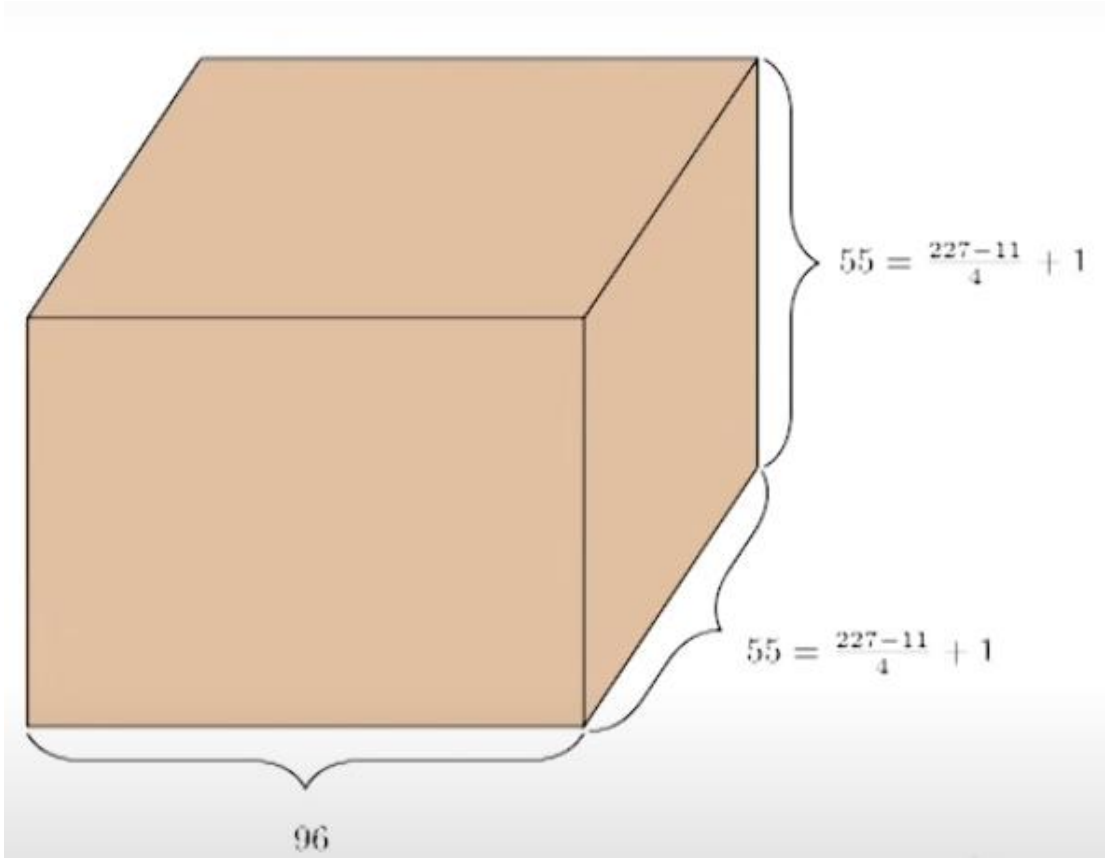
$$W_2 = \frac{W_1 - F + 2P}{S} + 1$$

$$H_2 = \frac{H_1 - F + 2P}{S} + 1$$

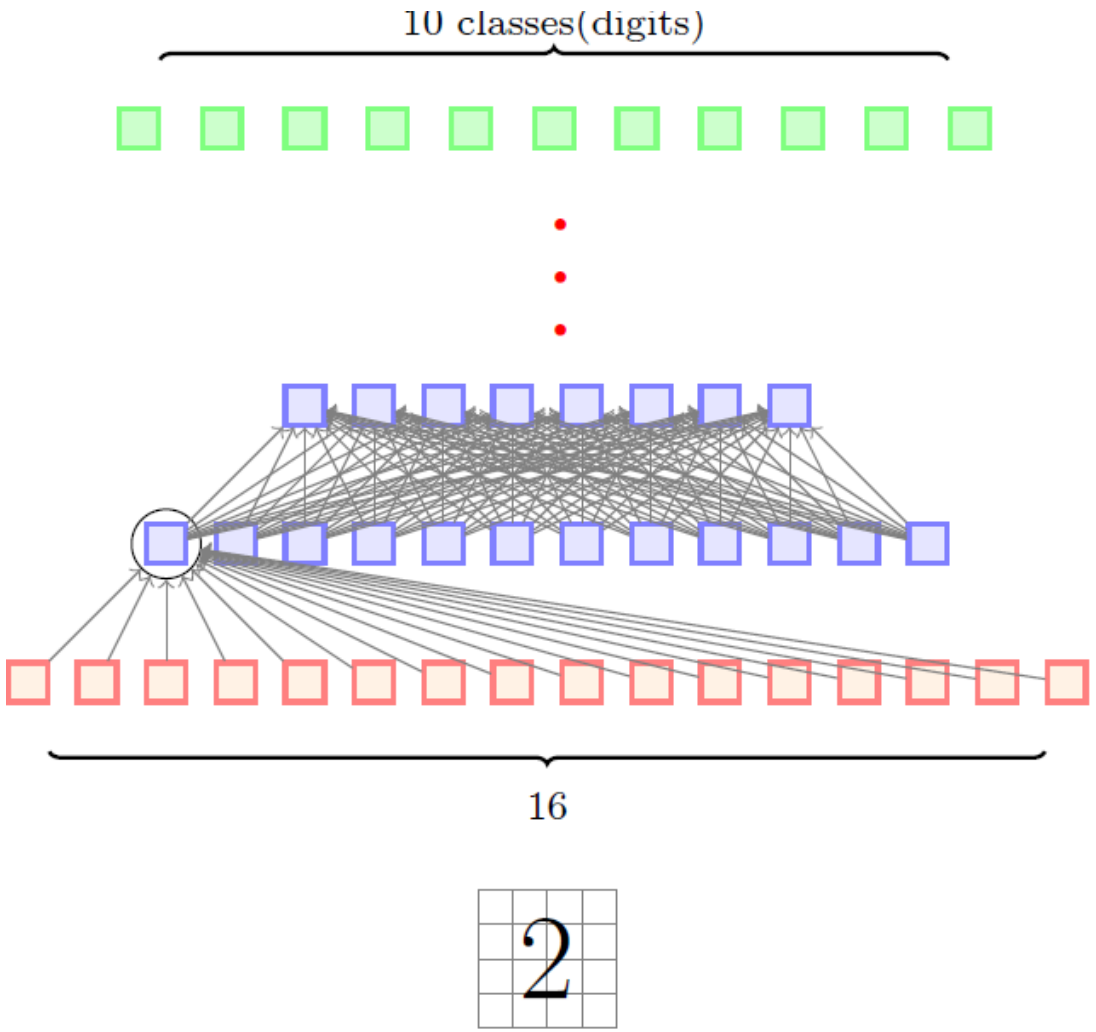
$$D_2 = K$$

What does the stride S do?





Convolutional Neural Networks

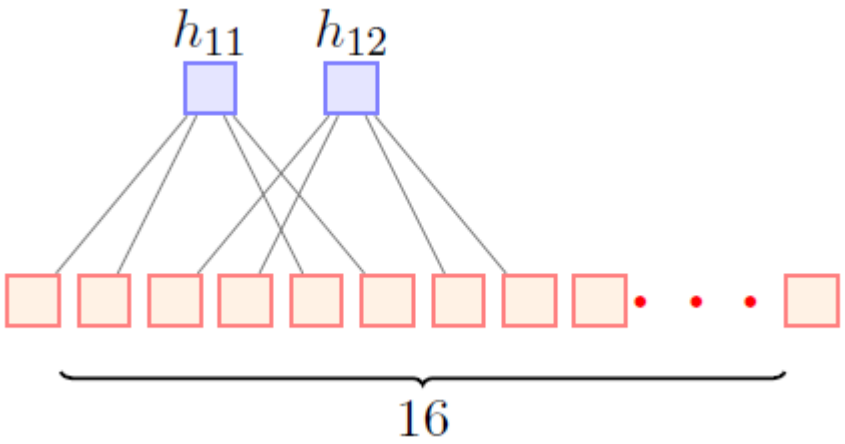


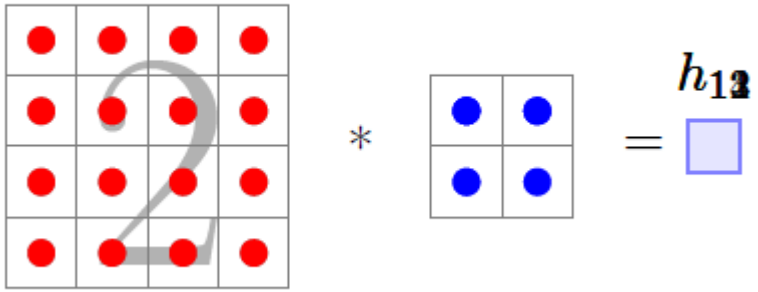
This is what a regular feed-forward neural network will look like

For example all the 16 input neurons are contributing to the computation of h11

Contrast this to what happens in the case of convolution

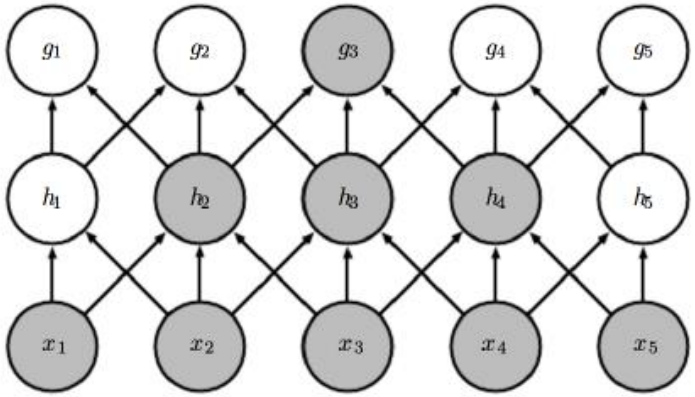
Convolutional Neural Networks

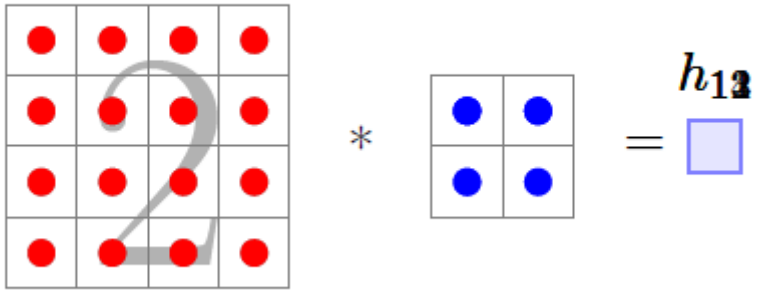


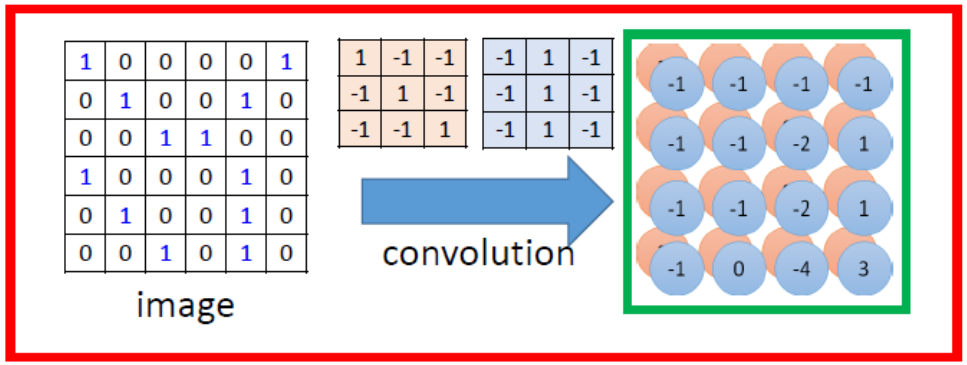


- Only a few local neurons participate in the computation of h11
- For example, only pixels 1, 2, 5, 6 contribute to h11.
- The connections are much **sparser**
- We are taking advantage of the structure of the image (interactions between neighboring pixels are more interesting)
- This sparse connectivity reduces the number of parameters in the model

Convolutional Neural Networks

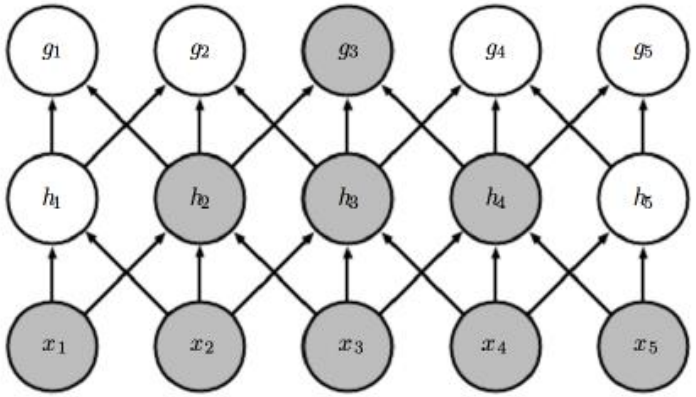


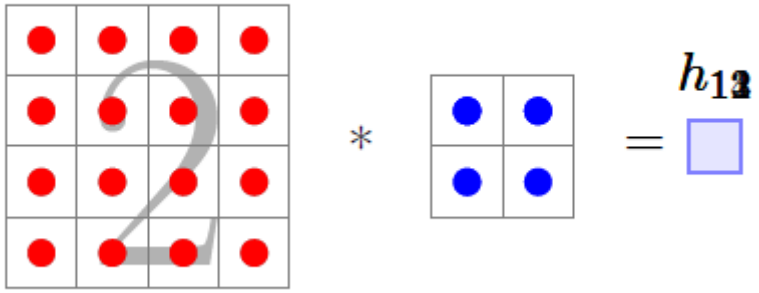


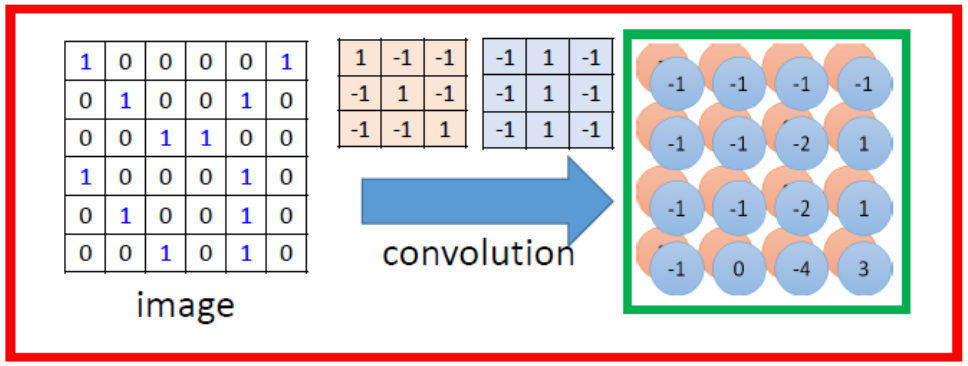


- Aren't we losing information (by losing interactions between some input pixels)
- The two highlighted neurons (x1 & x5) do not interact in layer 1 . But they indirectly contribute to the computation of g3 and hence interact indirectly.
- Another characteristic of CNNs is **weight sharing**.
- Shouldn't we be applying the same kernel at all the portions of the image? No, we can have many such kernels but the kernels will be shared by all locations in the image

Properties with convolution

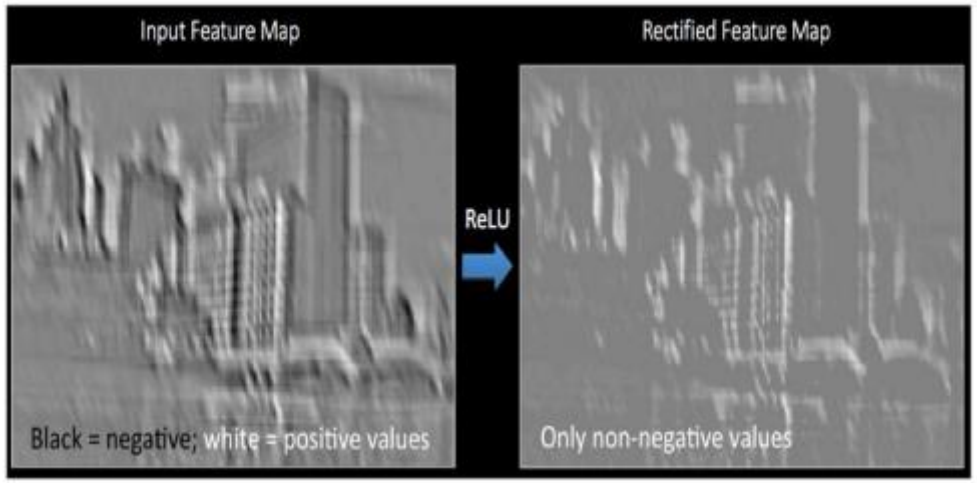






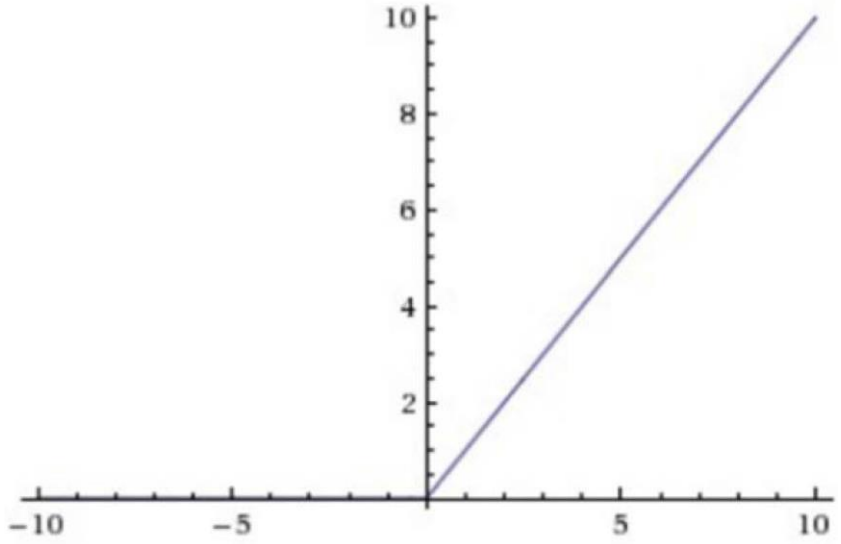
1. Apply a set of weights –a filter –to extract local features
2. Use multiple filters to extract different features
3. Spatially share parameters of each filter

Introducing Non-Linearity



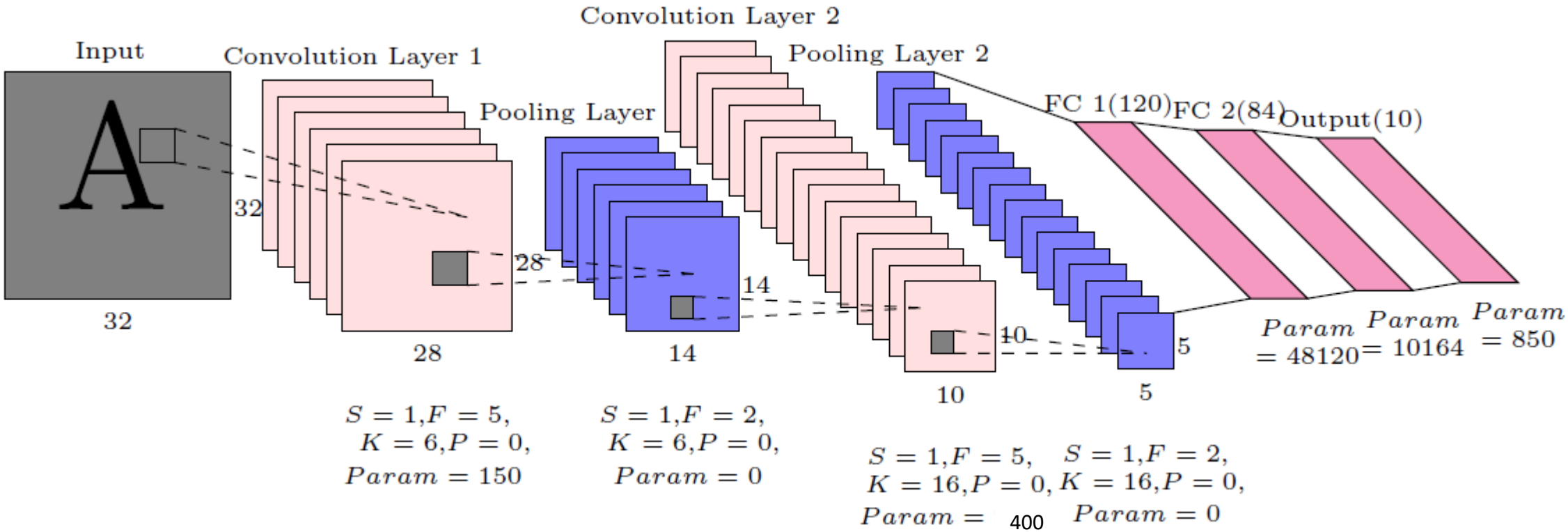
- Apply after every convolution operation (i.e., after convolutional layers)

- ReLU: pixel-by-pixel operation that replaces all negative values by zero.
- Non-linear operation

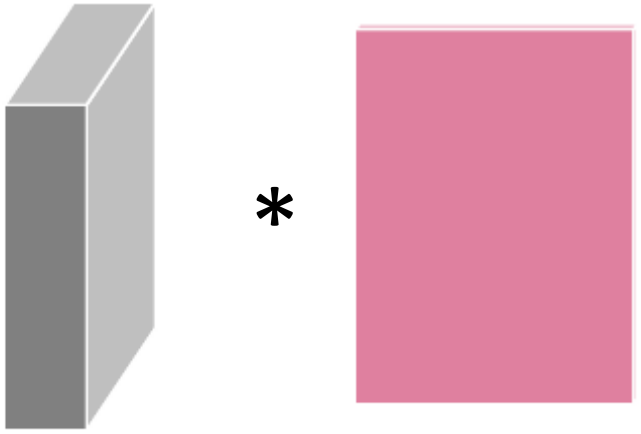


Full convolutional neural network

What does a pooling layer do?



What does a pooling layer do?



*

=

1	4	2	1
5	8	3	4
7	6	4	5
1	3	1	2

maxpool

2x2 filters (stride 2)

8	4
7	5

1	4	2	1
5	8	3	4
7	6	4	5
1	3	1	2

maxpool

2x2 filters (stride 1)

8	8	4
8	8	5
7	6	5

1)Reduced dimensionality
2)Spatial invariance

Instead of max pooling we can also do average pooling

tf.keras.layers.MaxPool2D(pool_size=(2,2),strides=2

What does a pooling layer do?

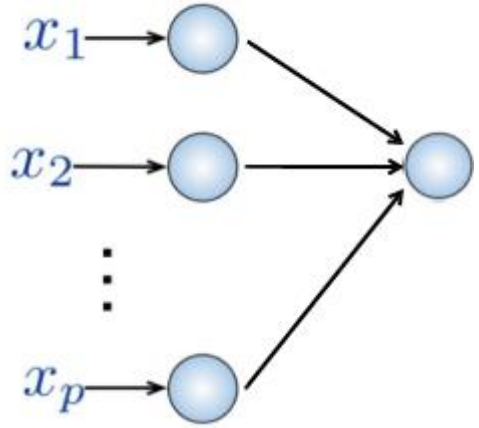
```
import torch.nn as nn
import pandas as pd
import torch
from torch.optim import Adam
from torch.utils.data import Dataset, DataLoader
```

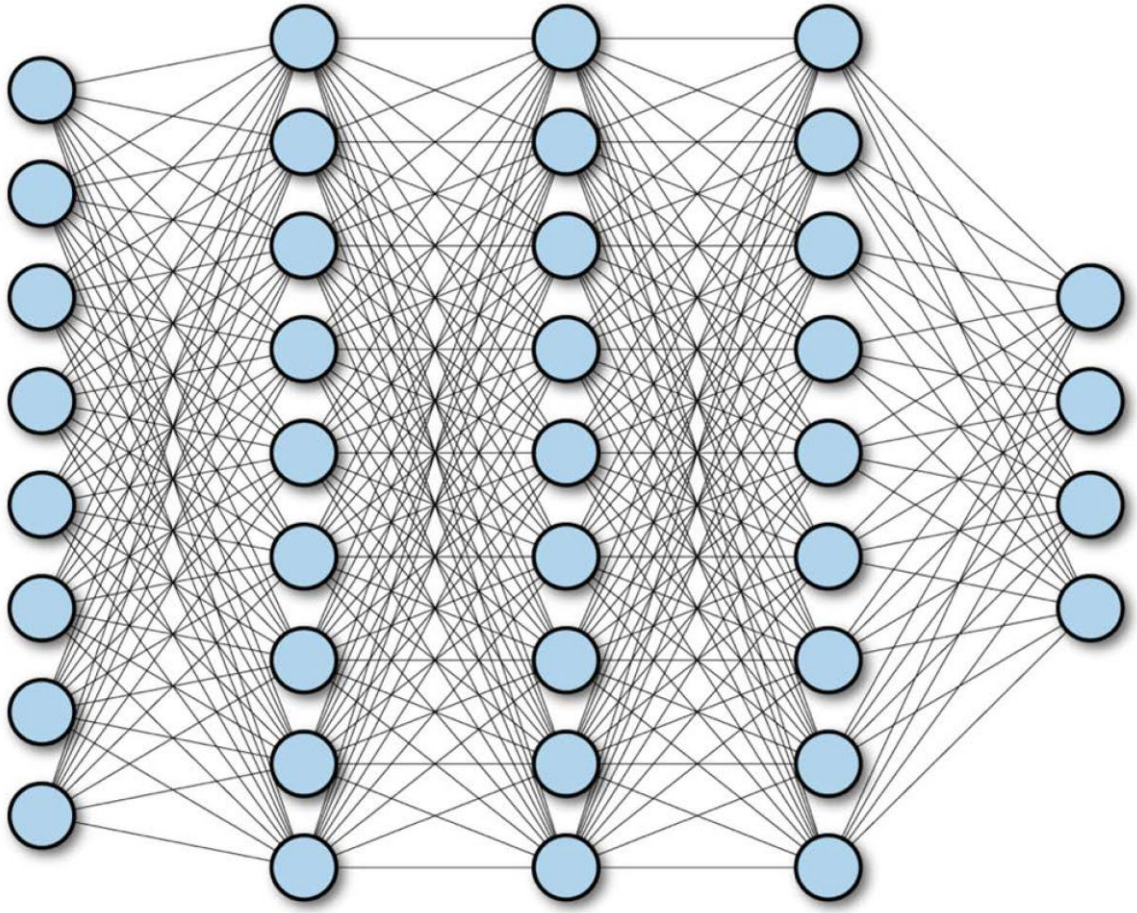
```
n_classes = 24
kernel_size = 3
flattened_img_size = 75 * 3 * 3

model = nn.Sequential(
    # First convolution
    nn.Conv2d(IMG_CHS, 25, kernel_size, stride=1, padding=1),  # 25 x 28 x 28
    nn.BatchNorm2d(25),
    nn.ReLU(),
    nn.MaxPool2d(2, stride=2),  # 25 x 14 x 14
    # Second convolution
    nn.Conv2d(25, 50, kernel_size, stride=1, padding=1),  # 50 x 14 x 14
    nn.BatchNorm2d(50),
    nn.ReLU(),
    nn.Dropout(.2),
    nn.MaxPool2d(2, stride=2),  # 50 x 7 x 7
    # Third convolution
    nn.Conv2d(50, 75, kernel_size, stride=1, padding=1),  # 75 x 7 x 7
    nn.BatchNorm2d(75),
    nn.ReLU(),
    nn.MaxPool2d(2, stride=2),  # 75 x 3 x 3
    # Flatten to Dense
    nn.Flatten(),
    nn.Linear(flattened_img_size, 512),
    nn.Dropout(.3),
    nn.ReLU(),
    nn.Linear(512, n_classes)
)
```

Flatten

Fully Connected:
- Each neuron in hidden layer connected to all neurons in input layer
- No spatial information
- Many, many parameters

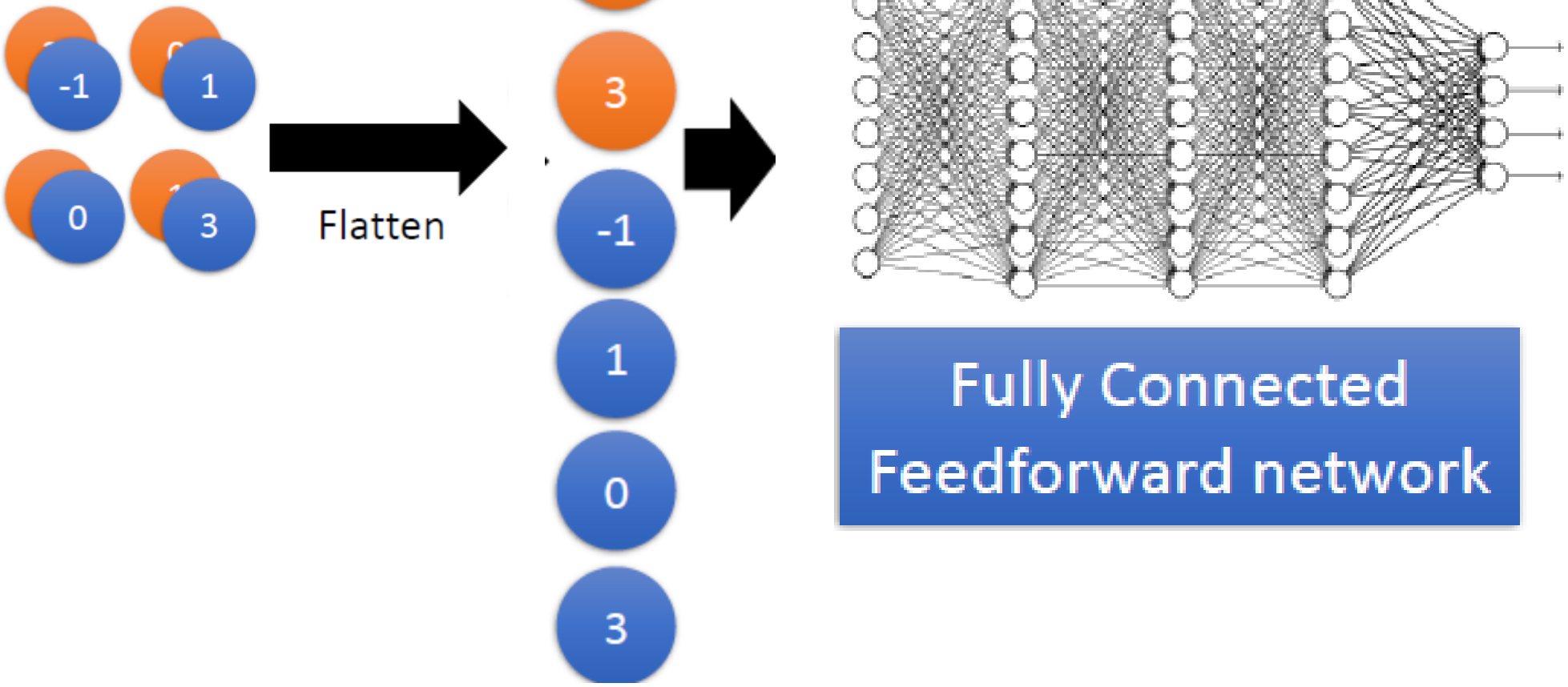
Input:
2Dimage
Vector of pixel values



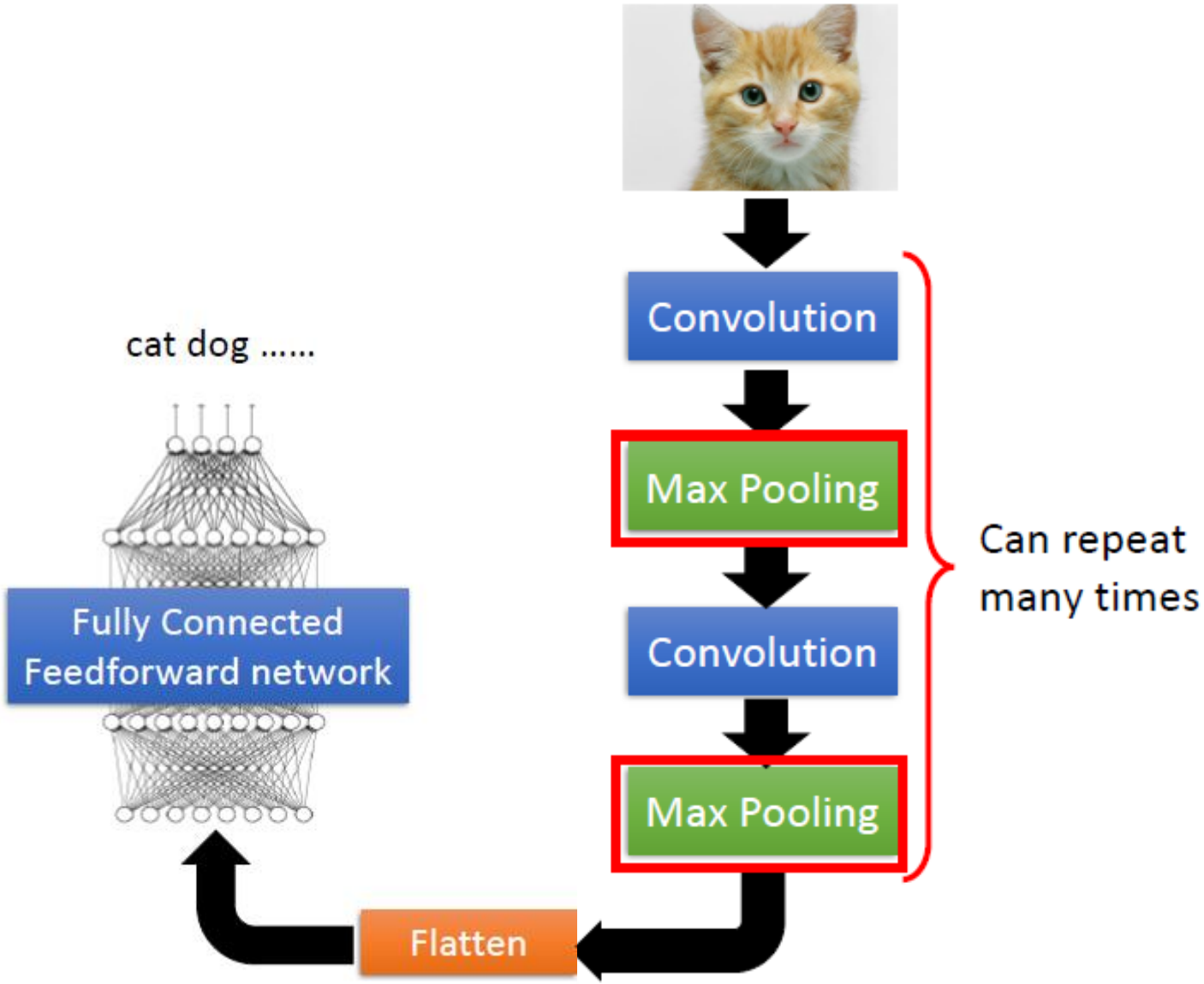


Flatten

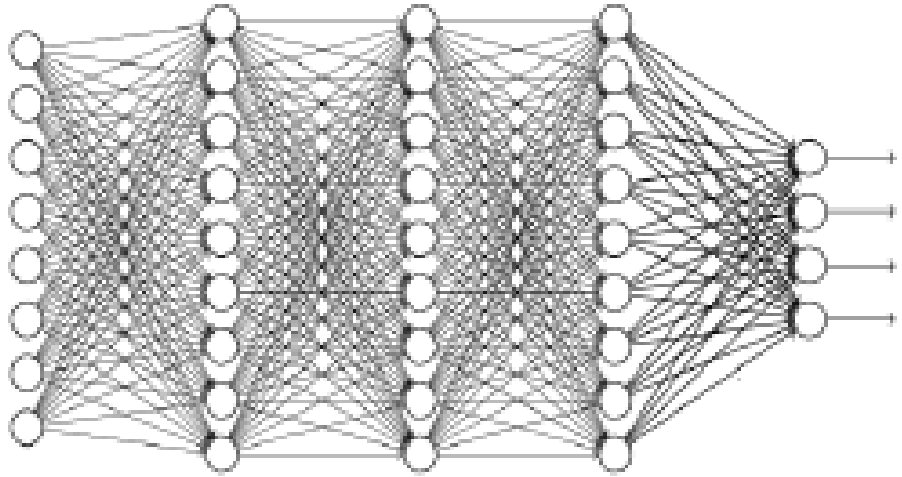
The aim of the Fully connected layer is to use the high-level feature of the input image produced by convolutional and pooling layers for classifying the input image into various classes based on the training dataset.



The whole CNN

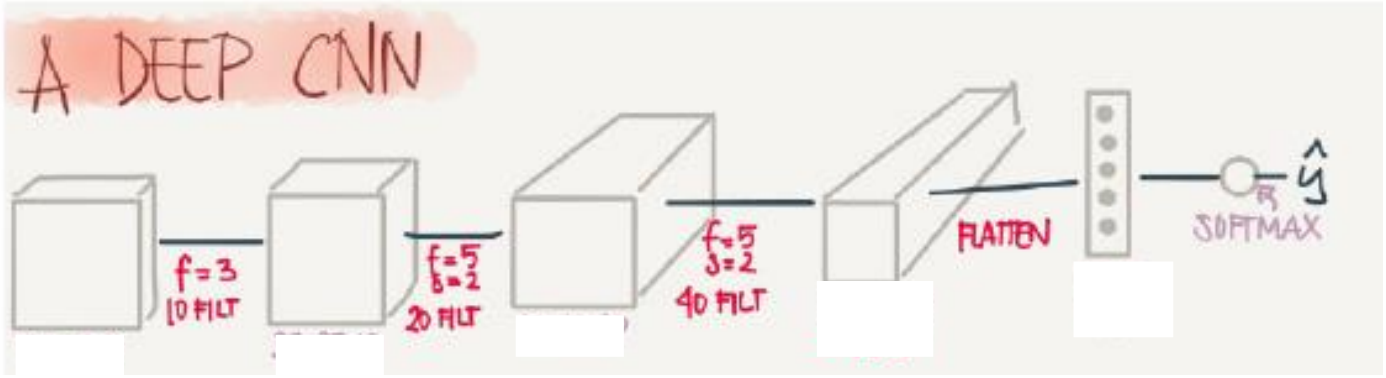


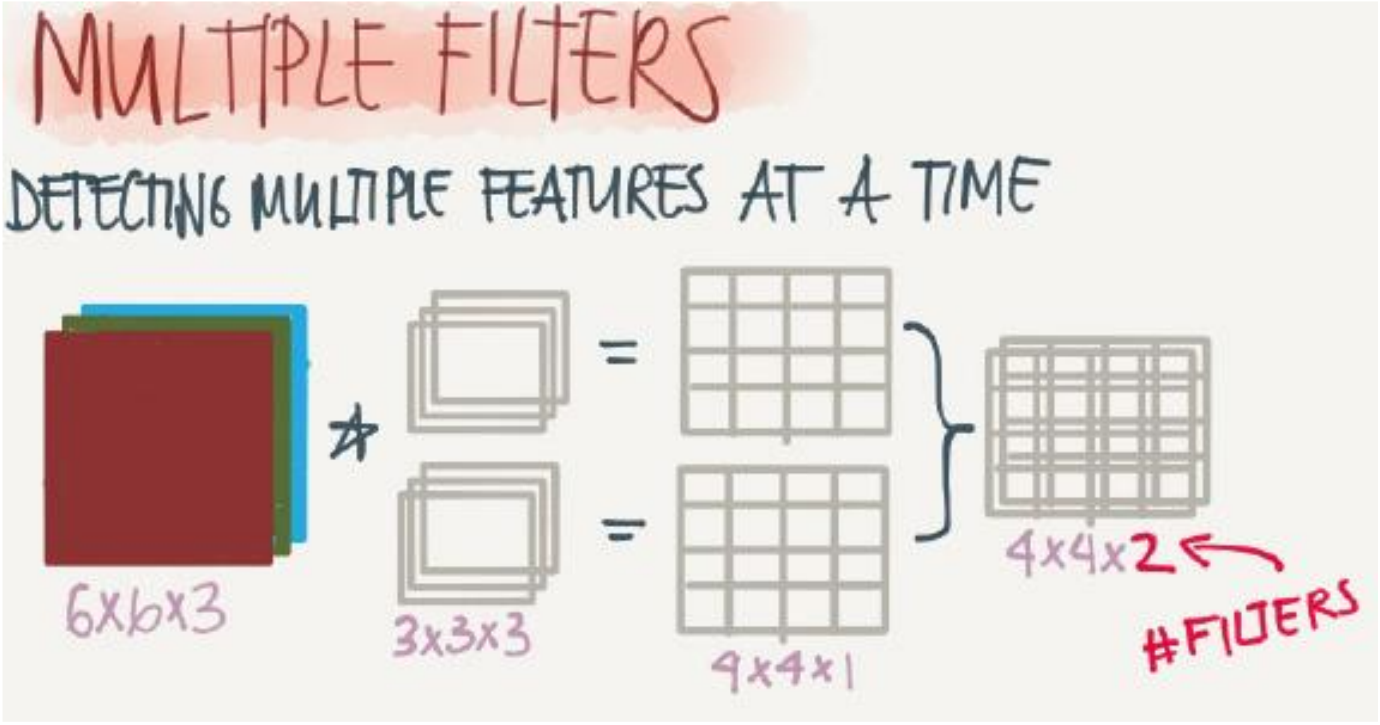
Flatten



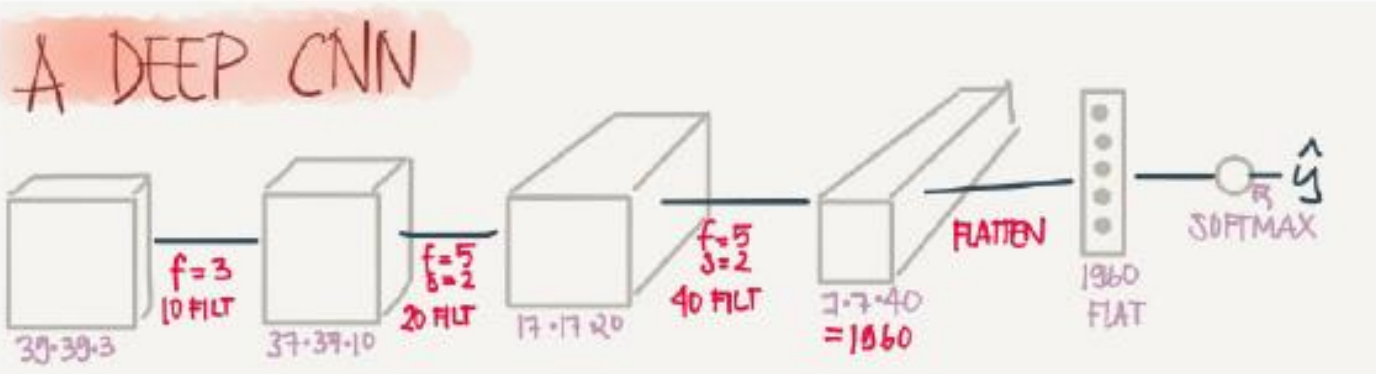
Fully Connected Feedforward network

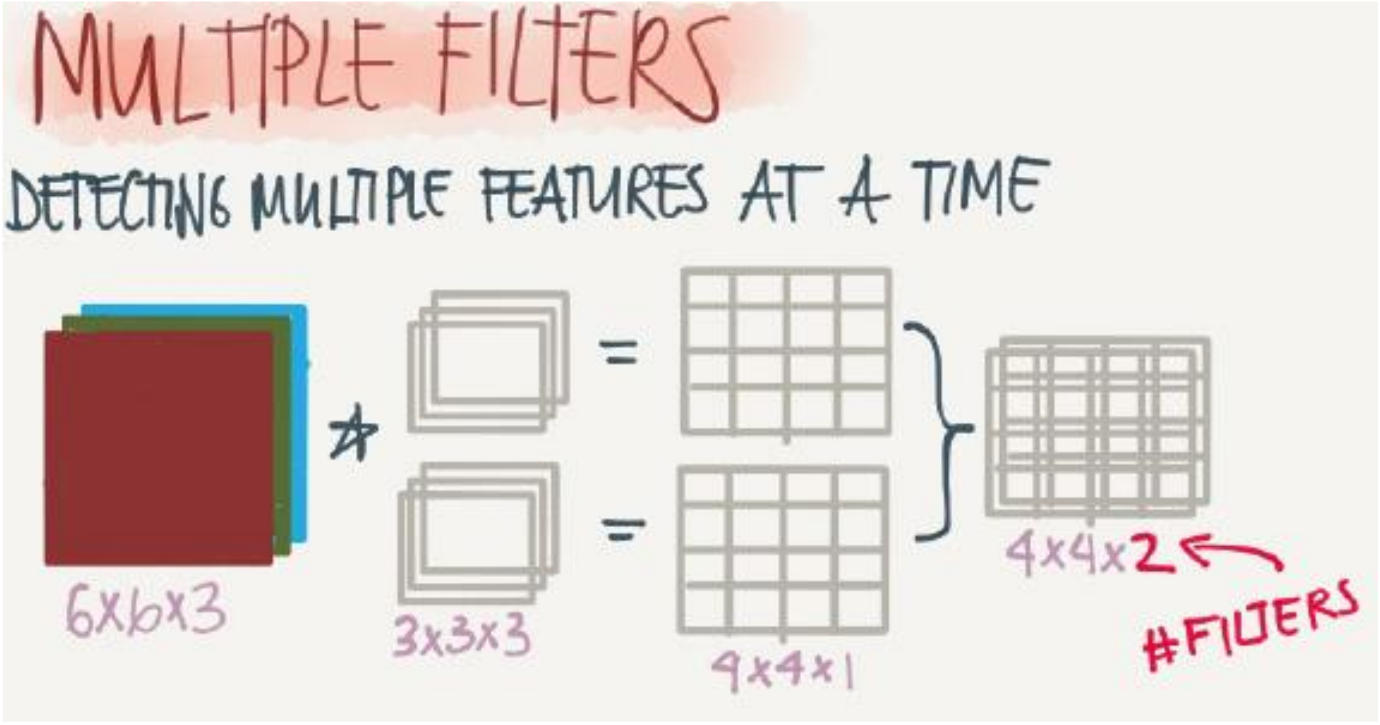
Flatten


A DEEP CNN
f=3
10 FILT
f=5
S=2
20 FILT
f=5
S=2
40 FILT
FLATTEN
SOFTMAX
ŷ

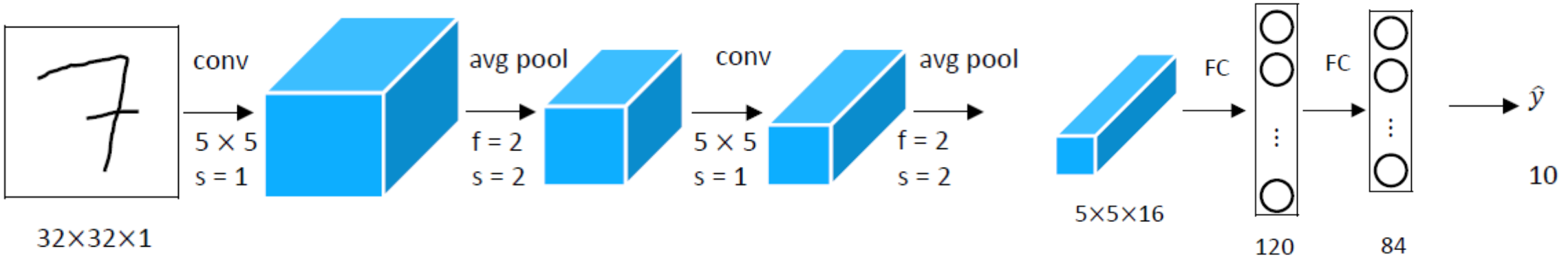

MULTIPLE FILTERS
DETECTING MULTIPLE FEATURES AT A TIME
6x6x3
3x3x3
4x4x2
#FILTERS
4x4x1

Flatten

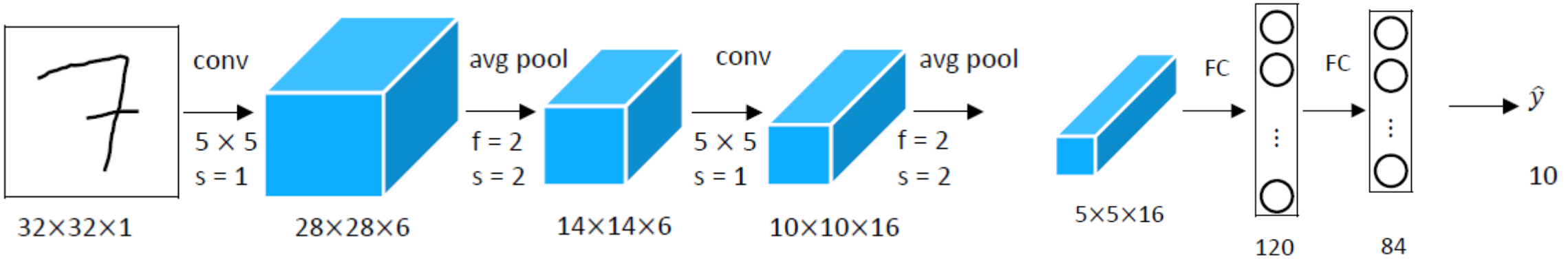

A DEEP CNN
f=3
10 FILT
f=5
S=2
20 FILT
f=5
S=2
40 FILT
FLATTEN
1960
FLAT
SOFTMAX
39·39·3
37·37·10
17·17·20
3·7·40
=1960


MULTIPLE FILTERS
DETECTING MULTIPLE FEATURES AT A TIME
6x6x3
3x3x3
9x4x1
4x4x2
#FILTERS

LeNet-5 for handwritten character recognition

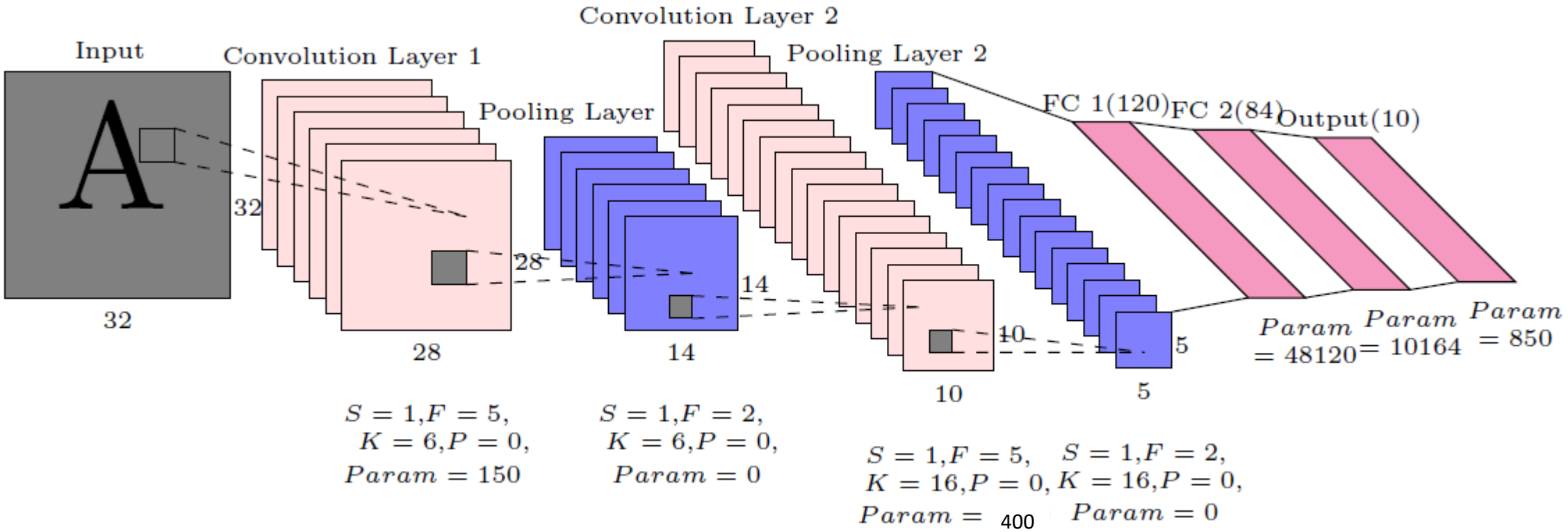


LeNet-5 for handwritten character recognition



LeNet-5 for handwritten character recognition

Gradient Based Learning Applied To Document Recognition -Y. Lecun,L. Bottou,Y. Bengio,P. Haffner; 1998
Helped establish how we use CNNs today
Replaced manual feature extraction



Backpropagation of convolution

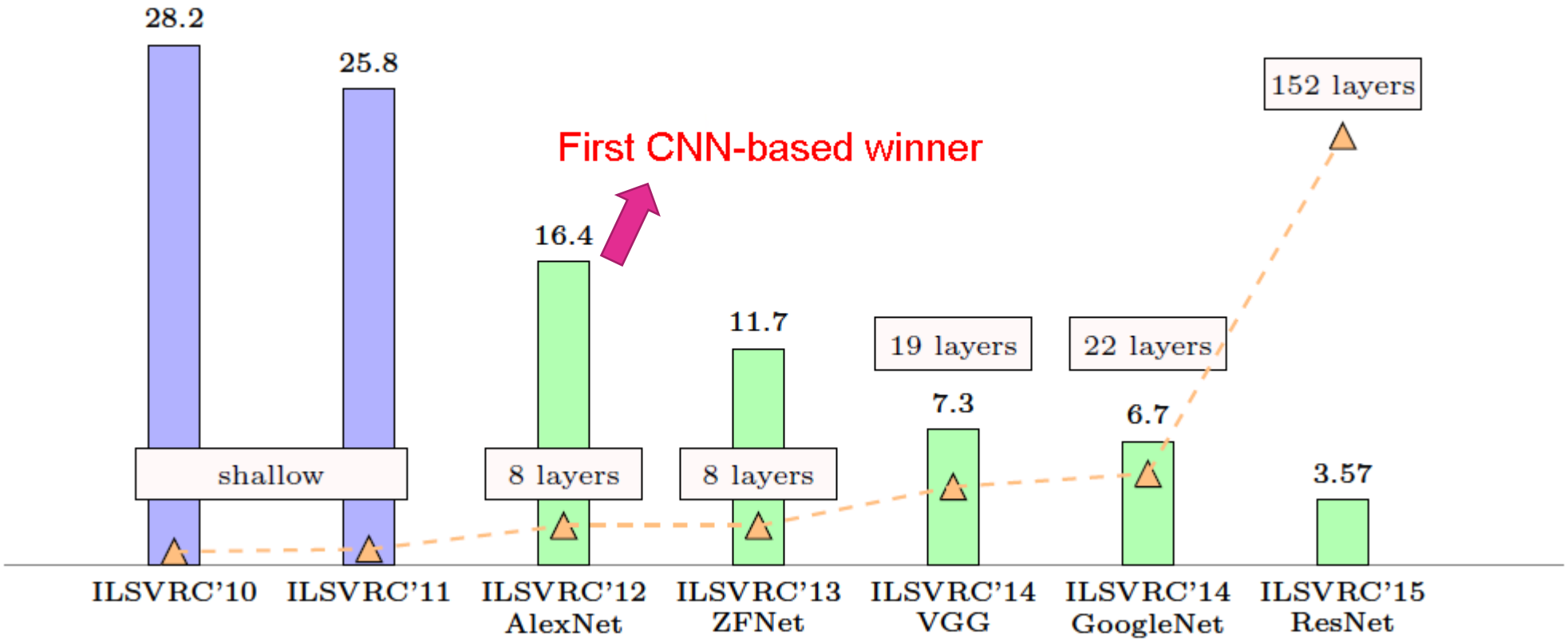
$$\begin{bmatrix} O_{11} & O_{12} \\ O_{21} & O_{22} \end{bmatrix} = \text{Convolution}\left(\begin{bmatrix} X_{11} & X_{12} & X_{13} \\ X_{21} & X_{22} & X_{23} \\ X_{31} & X_{32} & X_{33} \end{bmatrix}, \begin{bmatrix} F_{11} & F_{12} \\ F_{21} & F_{22} \end{bmatrix} \right)$$

$$\begin{bmatrix} \partial E/\partial F_{11} & \partial E/\partial F_{12} \\ \partial E/\partial F_{21} & \partial E/\partial F_{22} \end{bmatrix} = \text{Convolution}\left(\begin{bmatrix} X_{11} & X_{12} & X_{13} \\ X_{21} & X_{22} & X_{23} \\ X_{31} & X_{32} & X_{33} \end{bmatrix}, \begin{bmatrix} \partial E/\partial O_{11} & \partial E/\partial O_{12} \\ \partial E/\partial O_{21} & \partial E/\partial O_{22} \end{bmatrix} \right)$$

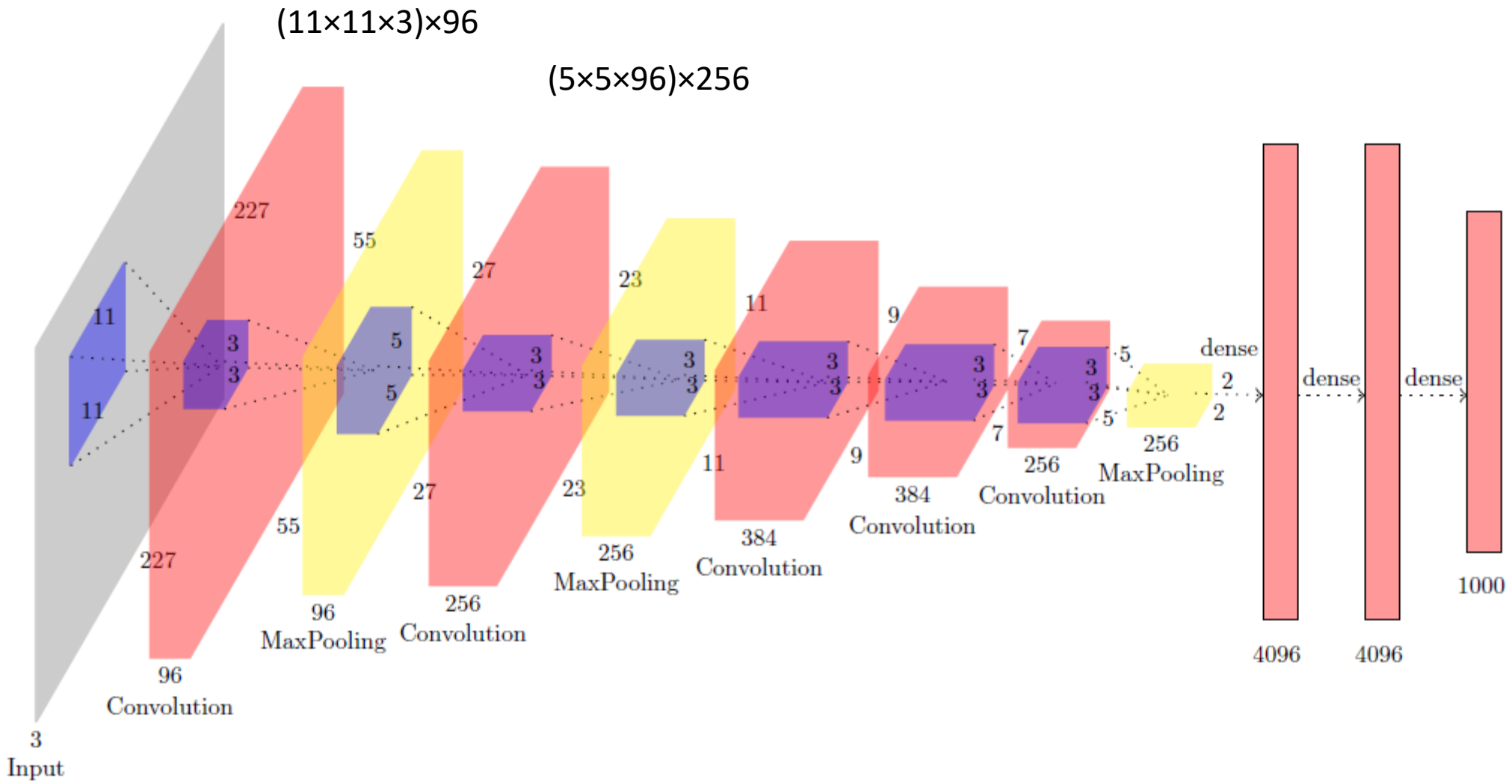
ImageNet Success Stories(roadmap for rest of the talk)

ImageNet Large Scale Visual Recognition Challenge (ILSVRC). The annual “Olympics” of computer vision.

Teams from across the world compete to see who has the best computer vision model for tasks such as classification, localization, detection, and more.



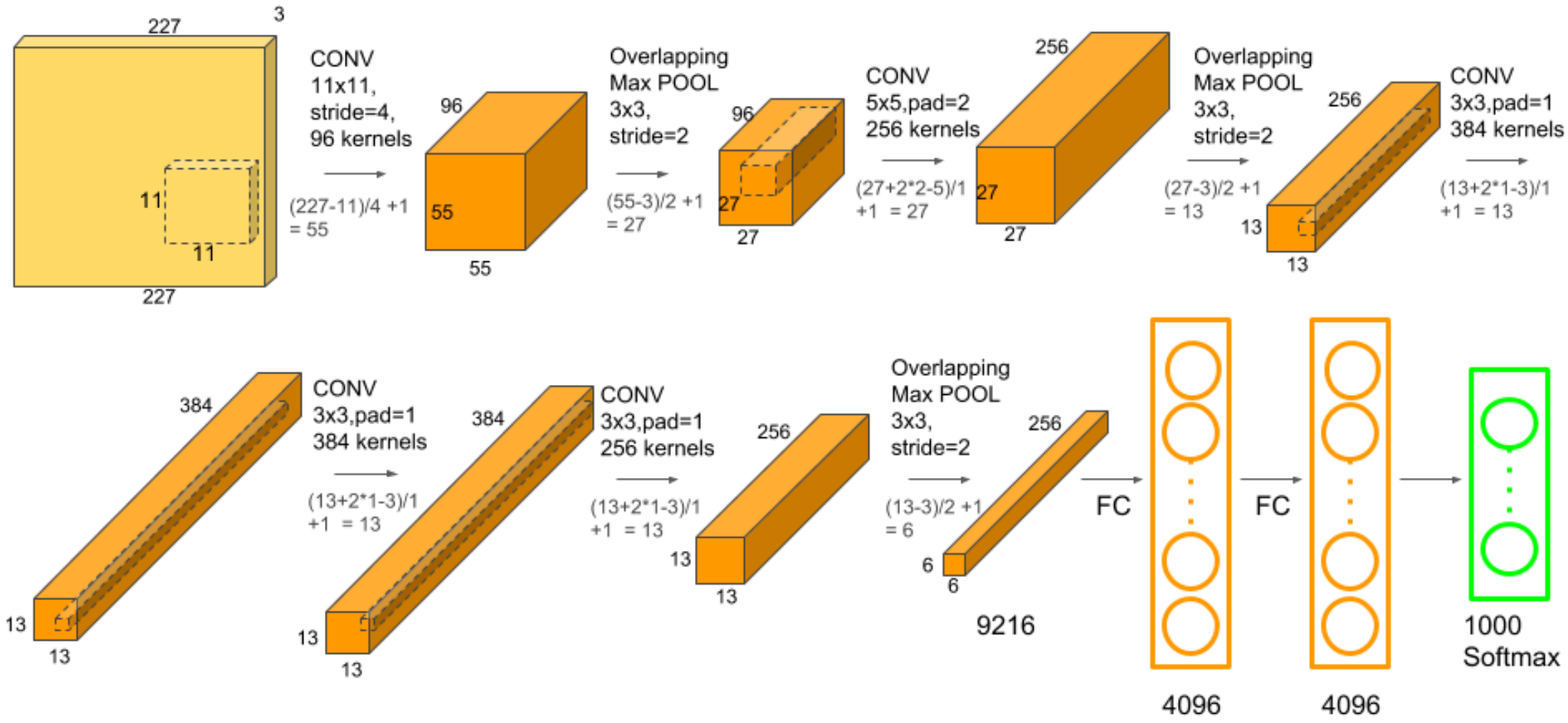
AlexNet


(11×11×3)×96
(5×5×96)×256
3
Input
227
227
11
11
96
Convolution
55
55
3
3
96
MaxPooling
27
27
5
5
256
Convolution
23
23
3
3
256
MaxPooling
11
11
3
3
384
Convolution
9
9
3
3
384
Convolution
7
7
3
3
256
Convolution
5
5
3
3
256
MaxPooling
dense
2
2
4096
dense
4096
dense
1000

AlexNet

1. **Input:** Color images of size 227x227x3.
2. Conv-1: The first convolutional layer consists of 96 kernels of size 11×11 applied with a stride of 4 and padding of 0.
3. MaxPool-1: The maxpool layer following Conv-1 consists of pooling size of 3×3 and stride 2.
4. Conv-2: The second conv layer consists of 256 kernels of size 5×5 applied with a stride of 1 and padding of 2.
5. MaxPool-2: The maxpool layer following Conv-2 consists of pooling size of 3×3 and a stride of 2.
6. Conv-3: The third conv layer consists of 384 kernels of size 3×3 applied with a stride of 1 and padding of 1.
7. Conv-4: The fourth conv layer has the same structure as the third conv layer. It consists of 384 kernels of size 3×3 applied with a stride of 1 and padding of 1.
8. Conv-5: The fifth conv layer consists of 256 kernels of size 3×3 applied with a stride of 1 and padding of 1.
9. MaxPool-3: The maxpool layer following Conv-5 consists of pooling size of 3×3 and a stride of 2.
10. FC-1: The first fully connected layer has 4096 neurons.
11. FC-2: The second fully connected layer has 4096 neurons.
12. FC-3: The third fully connected layer has 1000 neurons.

AlexNet


227
3
227
11
11
CONV
11x11,
stride=4,
96 kernels
(227-11)/4 +1
= 55
96
55
55
Overlapping
Max POOL
3x3,
stride=2
(55-3)/2 +1
= 27
96
27
27
CONV
5x5,pad=2
256 kernels
(27+2*2-5)/1
+1 = 27
256
27
27
Overlapping
Max POOL
3x3,
stride=2
(27-3)/2 +1
= 13
256
13
13
CONV
3x3,pad=1
384 kernels
(13+2*1-3)/1
+1 = 13
384
13
13
CONV
3x3,pad=1
384 kernels
(13+2*1-3)/1
+1 = 13
384
13
13
CONV
3x3,pad=1
256 kernels
(13+2*1-3)/1
+1 = 13
256
13
13
Overlapping
Max POOL
3x3,
stride=2
(13-3)/2 +1
= 6
256
6
6
9216
FC
4096
FC
4096
1000
Softmax

Calculate parameters in Maxpooling Layer

There are no parameters associated with a MaxPool layer.

The pool size, stride, and padding are hyperparameters.

Calculate parameters in Convolution Layer

W_c = Number of weights of the Conv Layer.
B_c = Number of biases of the Conv Layer.
P_c = Number of parameters of the Conv Layer.
K = Size (width) of kernels used in the Conv Layer.
N = Number of kernels.
C = Number of channels of the input image.

$$W_c = K^2 \times C \times N$$
$$B_c = N$$
$$P_c = W_c + B_c$$

$$W_c = 11^2 \times 3 \times 96 = 34,848$$
$$B_c = 96$$
$$P_c = 34,848 + 96 = 34,944$$

number of parameters for Conv-2, Conv-3, Conv-4, Conv-5 are 614656 , 885120, 1327488 and 884992 respectively.

Number of Parameters of a Fully Connected (FC) Layer

There are two kinds of fully connected layers in a CNN. The first FC layer is connected to the last Conv Layer, while later FC layers are connected to other FC layers. Let's consider each case separately.

W_{cf} = Number of weights of a FC Layer which is connected to a Conv Layer.
B_{cf} = Number of biases of a FC Layer which is connected to a Conv Layer.
O = Size (width) of the output image of the previous Conv Layer.
N = Number of kernels in the previous Conv Layer.
F = Number of neurons in the FC Layer.

$$W_{cf} = O^2 \times N \times F$$
$$B_{cf} = F$$
$$P_{cf} = W_{cf} + B_{cf}$$

The first fully connected layer of AlexNet is connected to a Conv Layer.
For this layer 0=6, N=256, F=4096

$$W_{cf} = 6^2 \times 256 \times 4096 = 37,748,736$$
$$B_{cf} = 4096$$
$$P_{cf} = W_{cf} + B_{cf} = 37,752,832$$

Number of Parameters of a Fully Connected (FC) Layer

Case 2: Number of Parameters of a Fully Connected (FC) Layer connected to a FC Layer

W_{ff} = Number of weights of a FC Layer which is connected to an FC Layer.
B_{ff} = Number of biases of a FC Layer which is connected to an FC Layer.
P_{ff} = Number of parameters of a FC Layer which is connected to an FC Layer.
F = Number of neurons in the FC Layer.
F_{-1} = Number of neurons in the previous FC Layer.

$$W_{ff} = F_{-1} \times F$$
$$B_{ff} = F$$
$$P_{ff} = W_{ff} + B_{ff}$$

Example: The last fully connected layer of AlexNet is connected to an FC Layer. For this layer, $F_{-1} = 4096$ and $F = 1000$. Therefore,

$$W_{ff} = 4096 \times 1000 = 4,096,000$$
$$B_{ff} = 1,000$$
$$P_{ff} = W_{ff} + B_{ff} = 4,097,000$$

Layer Name	Tensor Size	Weights	Biases	Parameters
Input Image	227x227x3	0	0	0
Conv-1	55x55x96	34,848	96	34,944
MaxPool-1	27x27x96	0	0	0
Conv-2	27x27x256	614,400	256	614,656
MaxPool-2	13x13x256	0	0	0
Conv-3	13x13x384	884,736	384	885,120
Conv-4	13x13x384	1,327,104	384	1,327,488
Conv-5	13x13x256	884,736	256	884,992
MaxPool-3	6x6x256	0	0	0
FC-1	4096×1	37,748,736	4,096	37,752,832
FC-2	4096×1	16,777,216	4,096	16,781,312
FC-3	1000×1	4,096,000	1,000	4,097,000
Output	1000×1	0	0	0
Total				**62,378,344**

Number of Parameters and Tensor Sizes in AlexNet

Layer Name	Tensor Size	Weights	Biases	Parameters
Input Image	227x227x3	0	0	0
Conv-1	55x55x96	34,848	96	34,944
MaxPool-1	27x27x96	0	0	0
Conv-2	27x27x256	614,400	256	614,656
MaxPool-2	13x13x256	0	0	0
Conv-3	13x13x384	884,736	384	885,120
Conv-4	13x13x384	1,327,104	384	1,327,488
Conv-5	13x13x256	884,736	256	884,992
MaxPool-3	6x6x256	0	0	0
FC-1	4096×1	37,748,736	4,096	37,752,832
FC-2	4096×1	16,777,216	4,096	16,781,312
FC-3	1000×1	4,096,000	1,000	4,097,000
Output	1000×1	0	0	0
Total				**62,378,344**

AlexNet

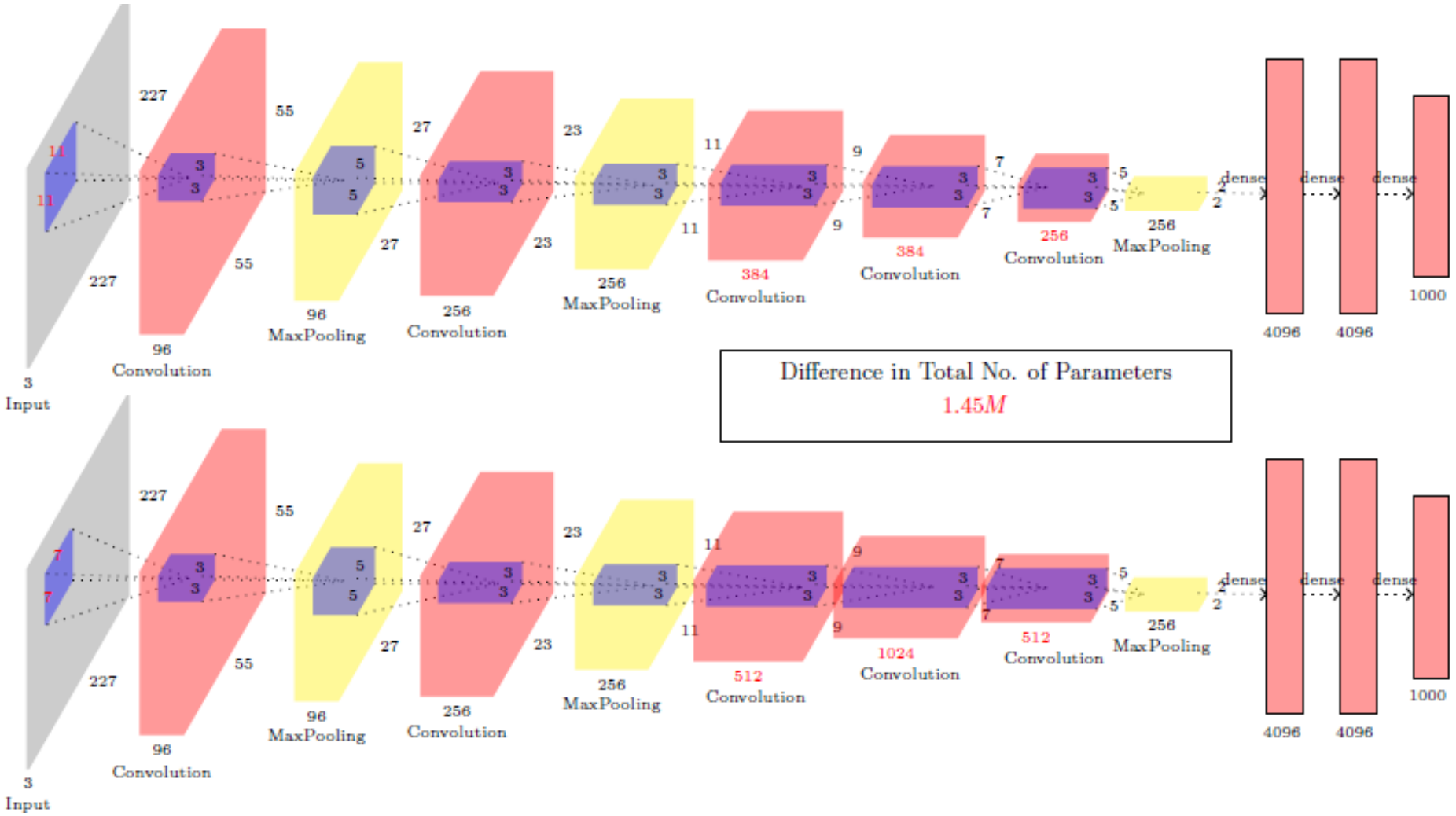
- AlexNet consists of 5 Convolutional Layers and 3 Fully Connected Layers.

- The input to AlexNet is an RGB image of size 256×256.

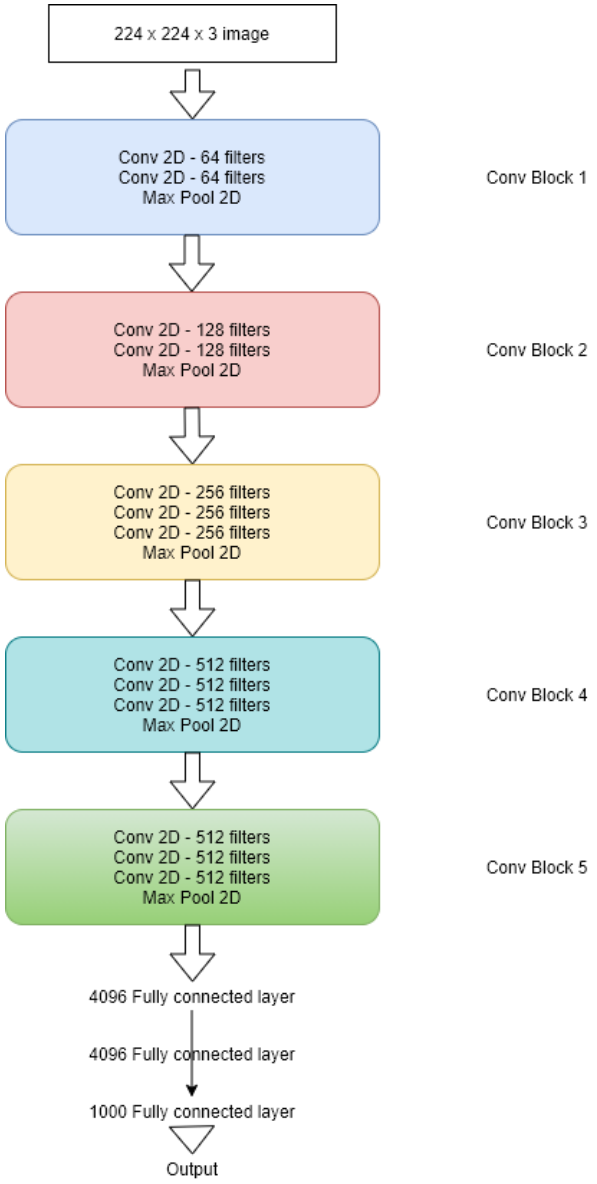
- Random crops of size 227×227 were generated from inside the 256×256 images to feed the first layer of AlexNet. It has 60 million parameters and 650,000 neurons and took five to six days to train on two GTX 580 3GB GPUs.

[**Krizhevskyet al.,2012**]
Details/Retrospectives:
- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- 7 CNN ensemble

AlexNet Vs ZFNet



VGG16 Architecture



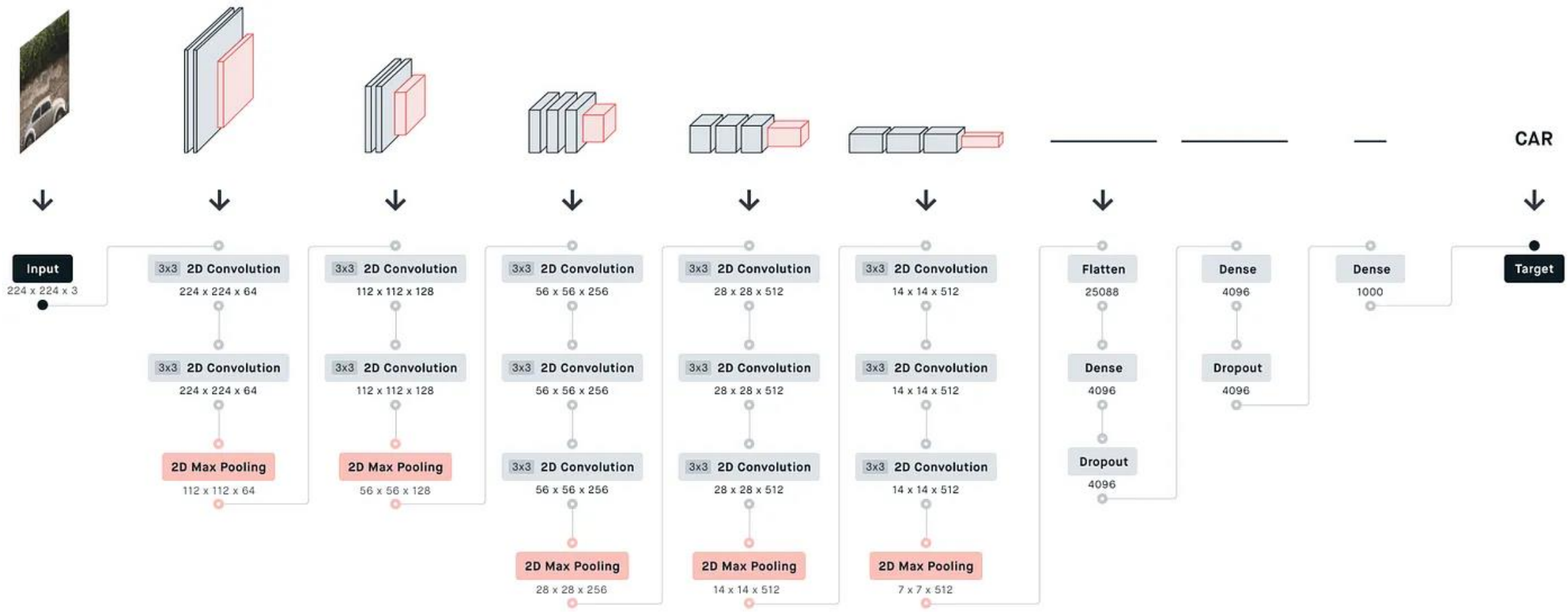
- **Layers**: 13 convolutional + 3 fully connected = 16 weight layers.
- **Input**: 224×224×32
- **Blocks**:

1. **Block 1**: 2 Conv (64 filters) → MaxPooling → 112×112×64.
2. **Block 2**: 2 Conv (128 filters) → MaxPooling → 56×56×128
3. **Block 3**: 3 Conv (256 filters) → MaxPooling → 28×28×256
4. **Block 4**: 3 Conv (512 filters) → MaxPooling → 14×14×512
5. **Block 5**: 3 Conv (512 filters) → MaxPooling → 7×7×512

- **Fully Connected**: Flatten → FC1 (4096) → FC2 (4096) → FC3 (1000 for ImageNet classes).

- Kernel size is 3 *3 throughout, stride of 1 and using the ‘same’ padding
- Total parameters in non FC layers = 16M
- Total Parameters in FC layers = (512 *7* 7* 4096) + (4096*4096) + (4096*1024) = 122M
- Most parameters are in the first FC layer (102M)

VGG16 Architecture



VGG19 Architecture

VGG19 Architecture

- **Layers**: 16 convolutional + 3 fully connected = 19 weight layers.
- **Input and Blocks**: Same as VGG16, except:
  - **Block 3**: 4 Conv (256 filters).
  - **Block 4**: 4 Conv (512 filters).
  - **Block 5**: 4 Conv (512 filters).

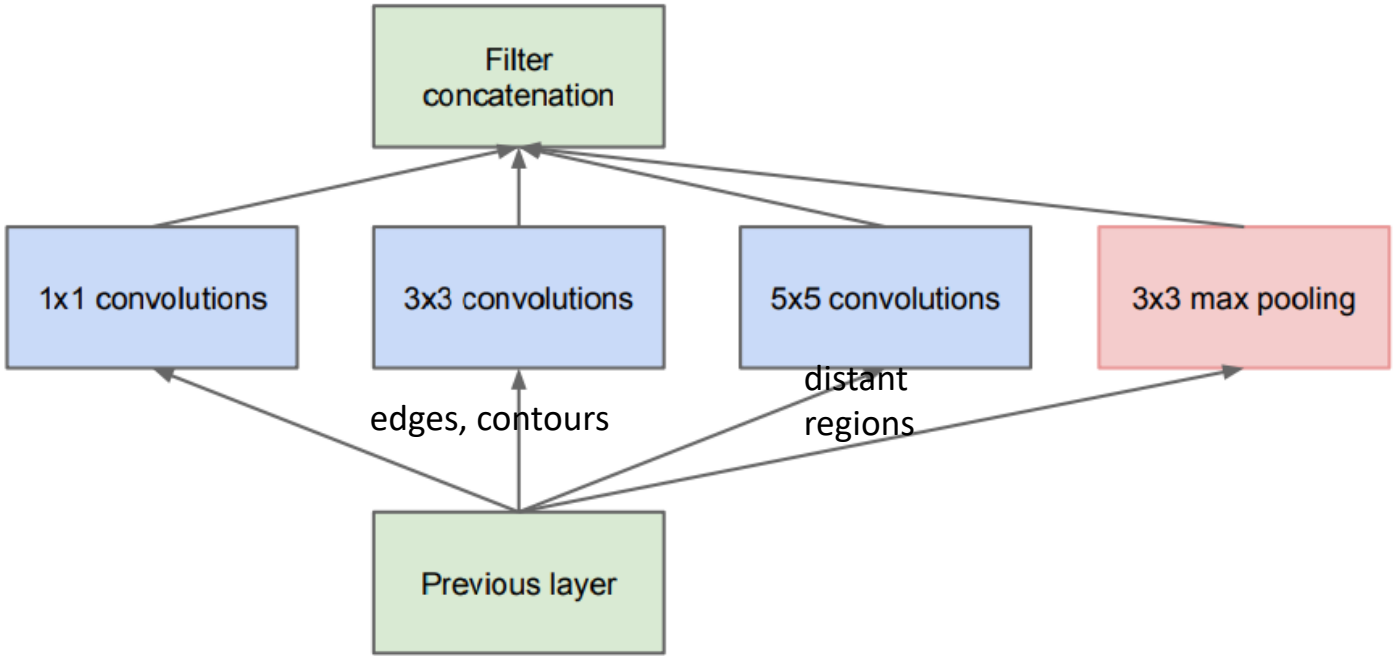
From left: A dog occupying most of the image, a dog occupying a part of it, and a dog occupying very little

- **Salient parts** in the image can have extremely **large variation** in size.
- The area occupied by the dog is different in each image. Because of this huge variation in the location of the information, choosing the **right kernel size** for the convolution operation becomes tough.
- A **larger kernel** is preferred for information that is distributed more **globally**, and a **smaller kernel** is preferred for information that is distributed more **locally.**
- **Very deep networks** are prone to **overfitting**. It also hard to pass gradient updates through the entire network.
- Naively stacking large convolution operations is **computationally expensive**.

The Solution:

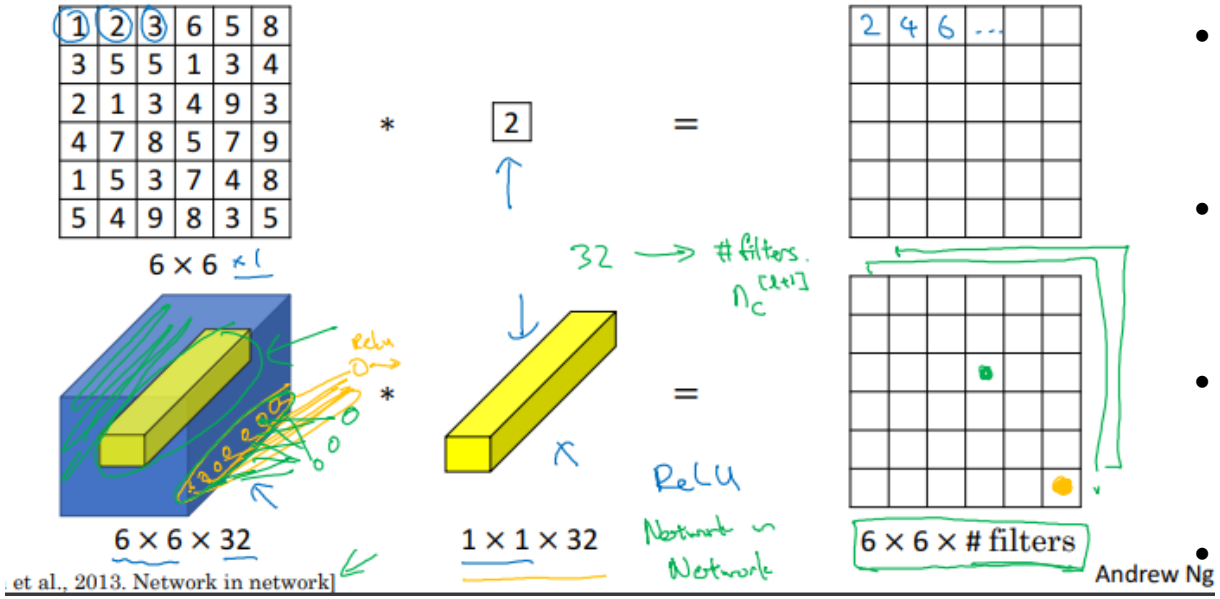
Why not have filters with **multiple sizes** operate on the **same level**? The network essentially would get a bit "**wider**" rather than "deeper". The authors designed the inception module to reflect the same.

The below image is the "naive" inception module. It performs **convolution** on an input, with **3 different sizes of filters** (1x1, 3x3, 5x5). Additionally, **max pooling** is also performed. The outputs are **concatenated** and sent to the next inception module.



(a) Inception module, naïve version

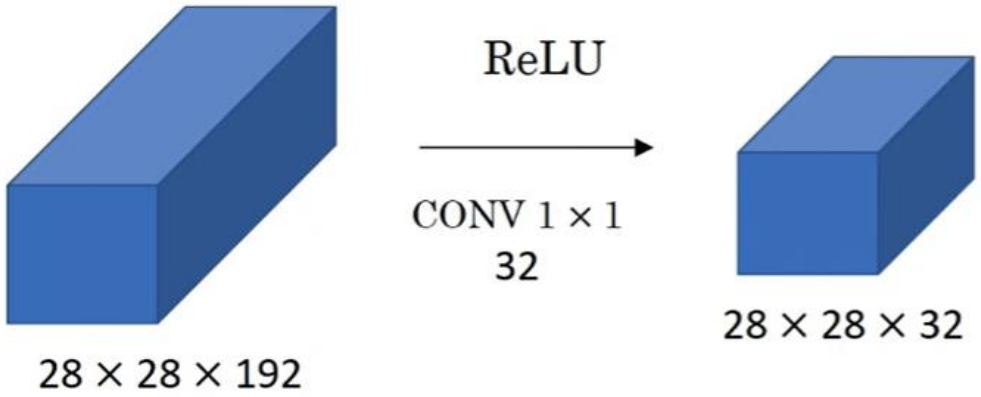
Network in Network and 1×1 convolutions



The input is a **6×6×C** feature map (C represents the number of input channels or depth).

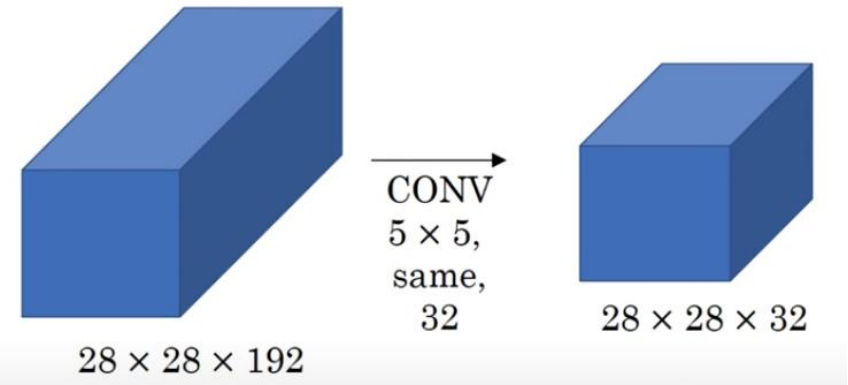
- A 1×1 convolution uses a filter (or kernel) that is 1×1 in spatial dimensions.
- Despite its small size, it is applied across all the channels (depth) of the input feature map.
- It performs a weighted sum (a dot product) across the depth dimension (channels) for each spatial location.
- You can reduce or increase the number of channels (depth) of your feature map using 1×1 convolutions.
- This is controlled by the number of filters
- For example, using 32 filters converts the depth of the feature map to 32.

Network in Network and 1×1 convolutions



- Each **1×1 filter** processes all 192 channels at a single spatial location and computes a weighted sum across these channels.
- This results in a single value per filter for that spatial location.
- With 32 filters, the depth of the output feature map becomes 32.
- The resulting feature map has dimensions **28×28×32.**
- The depth is reduced from **192 to 32**, which: Decreases the number of parameters.
- Reduces computational cost.

Network in Network and 1×1 convolutions



The total number of computations required is calculated as:

1.Filter Size:

Each filter has **5×5×192 = 4800** weights.

2.Number of Filters:

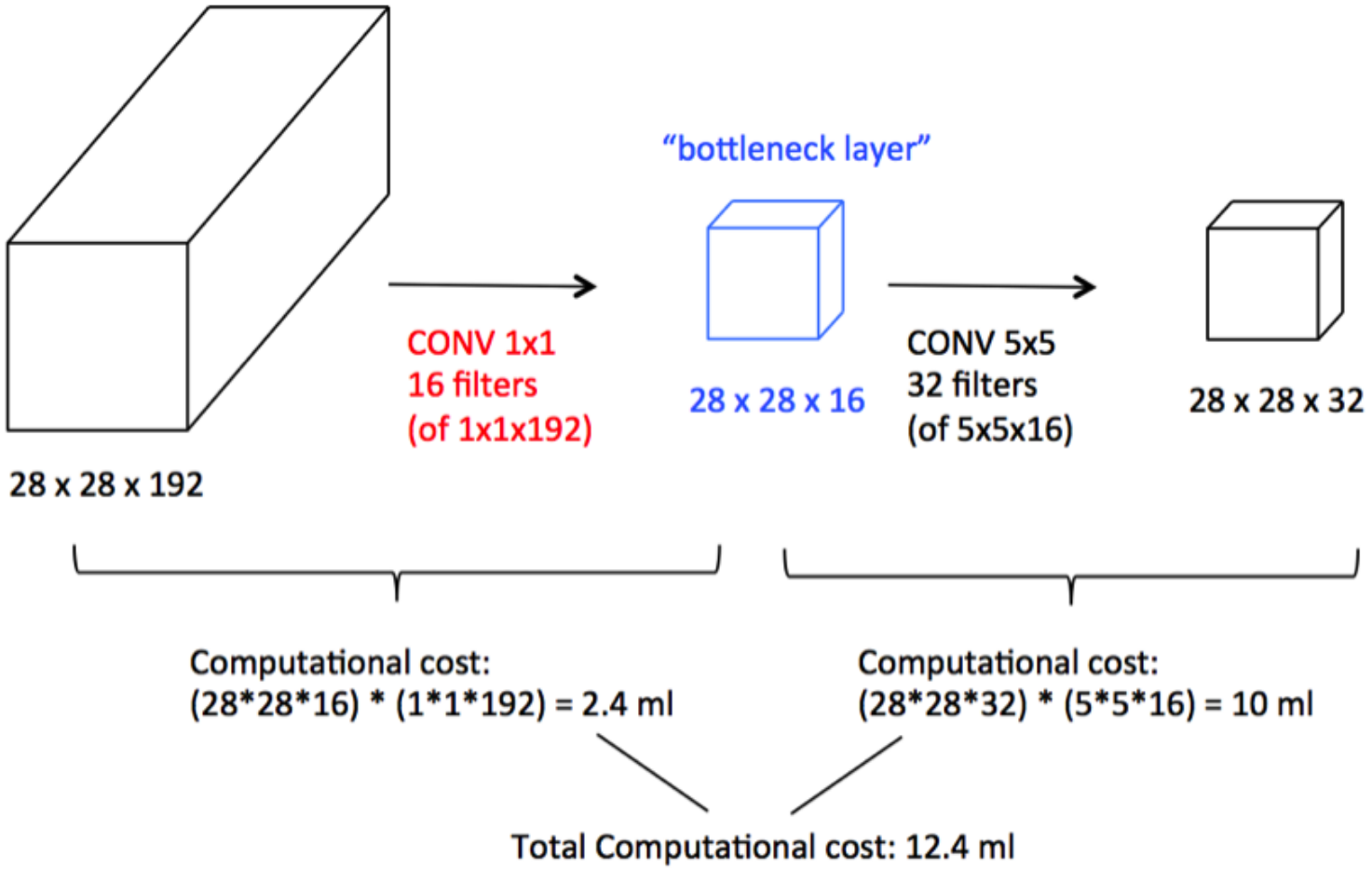
32 filters, so the total weights across all filters = **4800 × 32 = 153,600**.

3.Feature Map Dimensions:

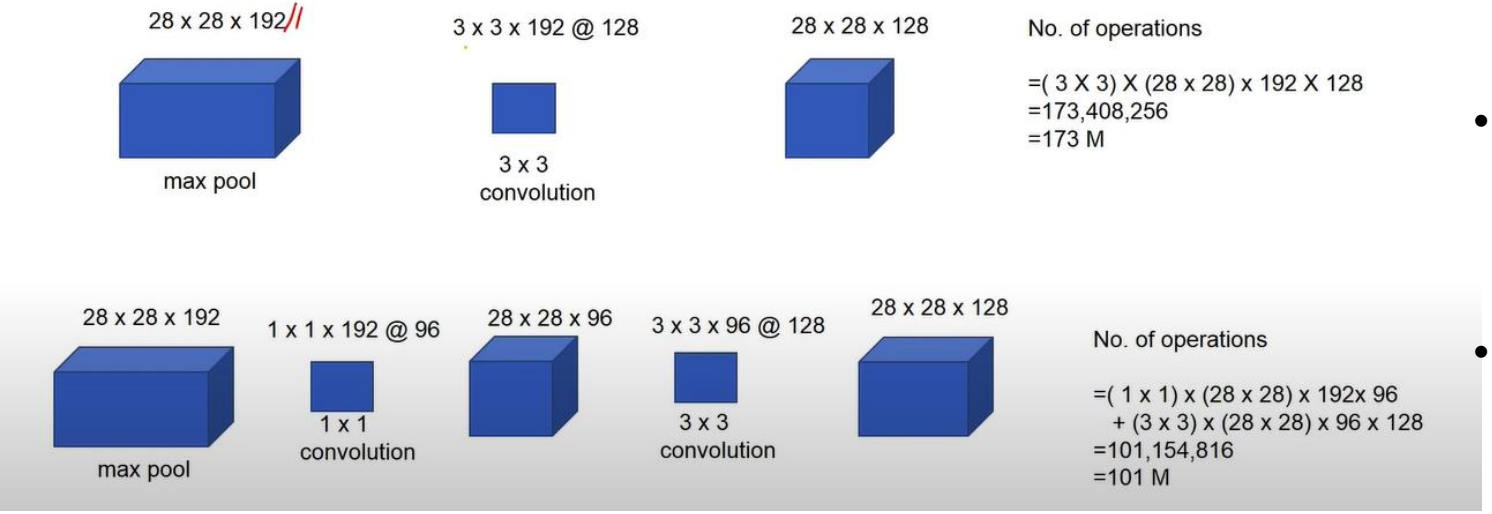
Each filter is applied to every position in the **28×28** spatial grid:

1. Total computations = **28×28×32×4800 = 120 million** operations.

Network in Network and 1×1 convolutions



Inception Module in Inception v1 (GoogLeNet)



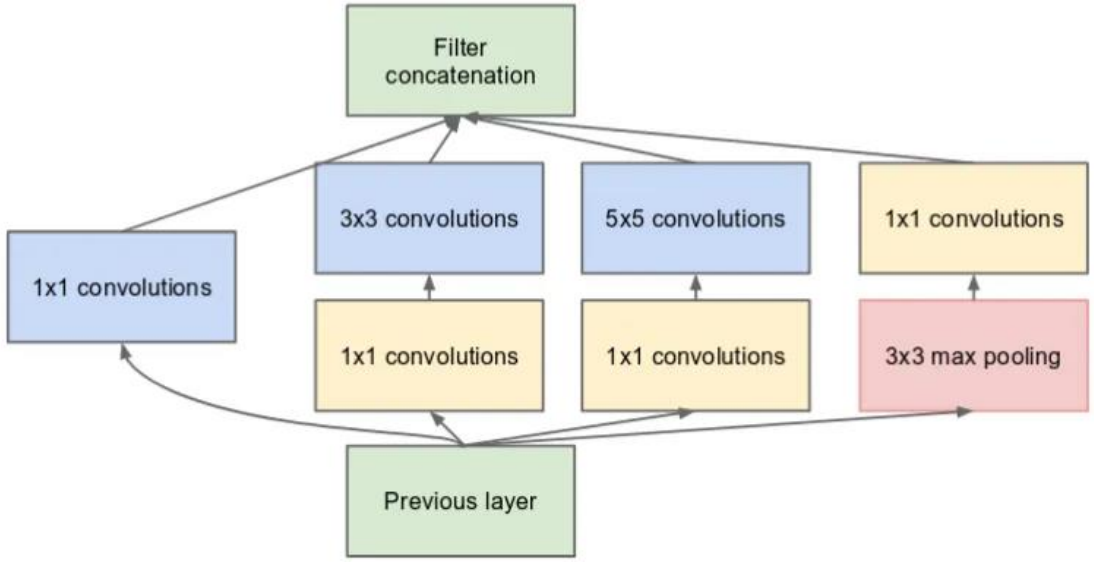
it is a pretty deep classifier. As with any very deep network, it is subject to the vanishing gradient problem.

- Maps an input pixel with all it's channels to an output pixel, not looking at anything around itself.
- It is often used to reduce the number of depth channels, since it is often very slow to multiply volumes with extremely large depths.
- The 1×1 filter can be used to create a linear projection of a stack of feature maps.
- The projection created by a 1×1 can act like channel-wise pooling and be used for dimensionality reduction.
- The projection created by a 1×1 can also be used directly or be used to increase the number of feature maps in a model.

1x1 Convolution- Channel-wise Pooling

- The 1×1 filter can be used to create a linear projection of a stack of feature maps.
- The projection created by a 1×1 can act like channel-wise pooling and be used for dimensionality reduction.
- The projection created by a 1×1 can also be used directly or be used to increase the number of feature maps in a model.

Network in Network and 1×1 convolutions

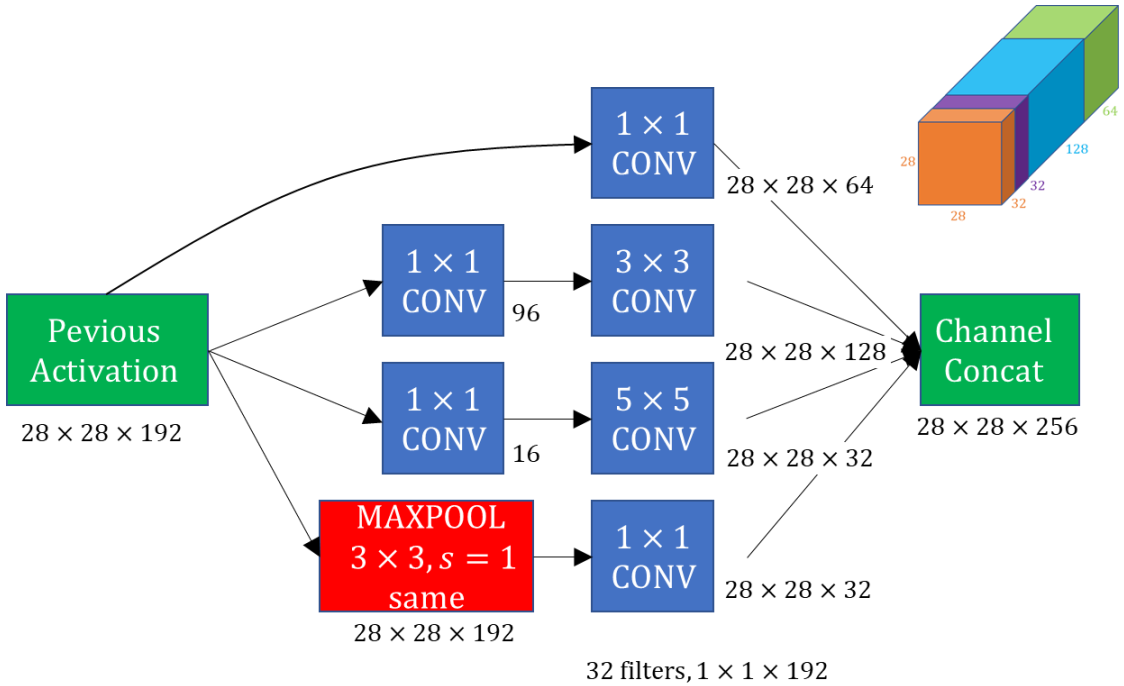


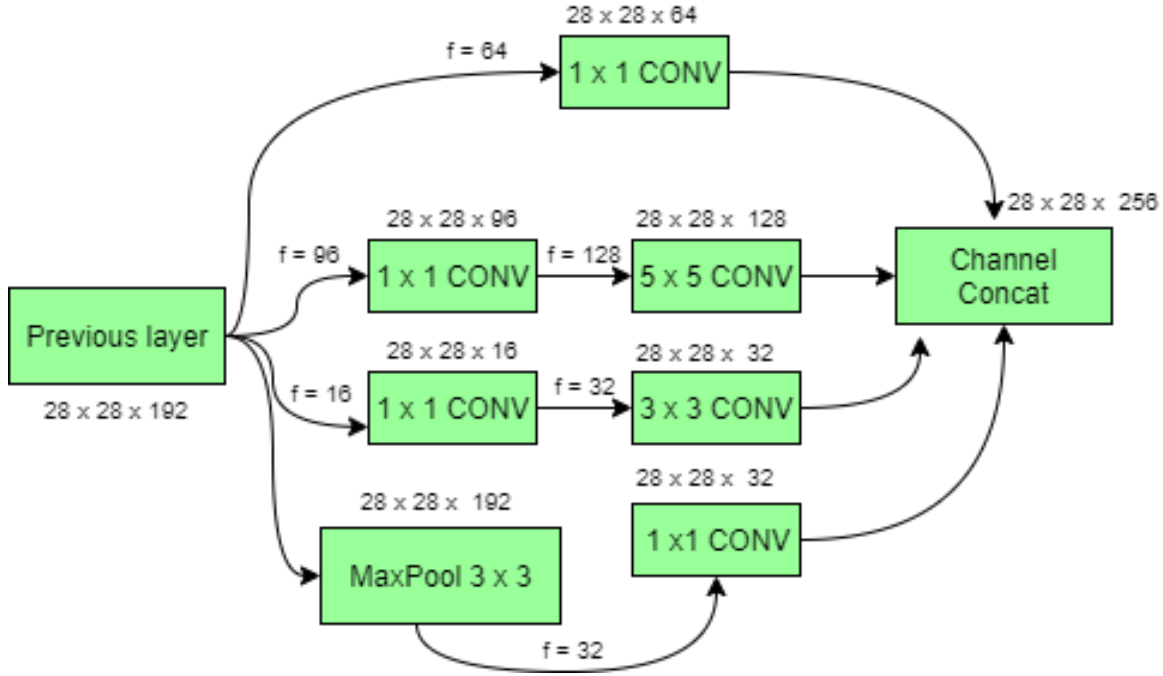
(b) Inception module with dimension reductions

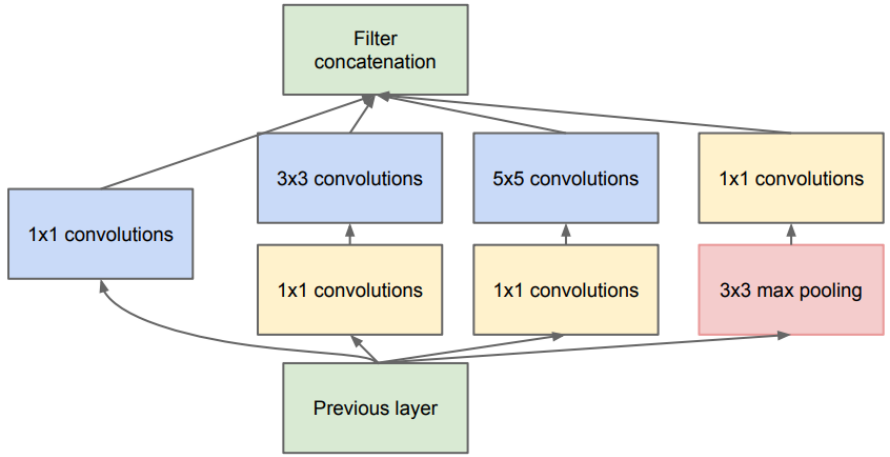
The authors limit the number of input channels by adding an extra 1x1 convolution before the 3x3 and 5x5 convolutions.

Using the dimension reduced inception module, a neural network architecture was built. This was popularly known as GoogLeNet (Inception v1).

Inception Networks

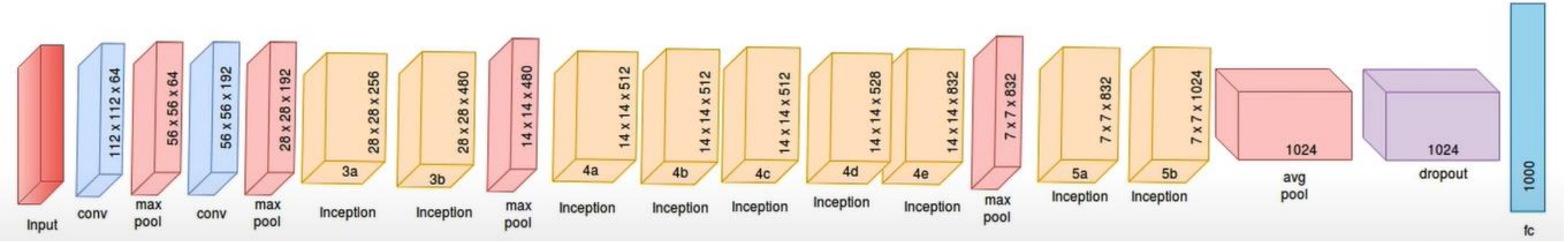




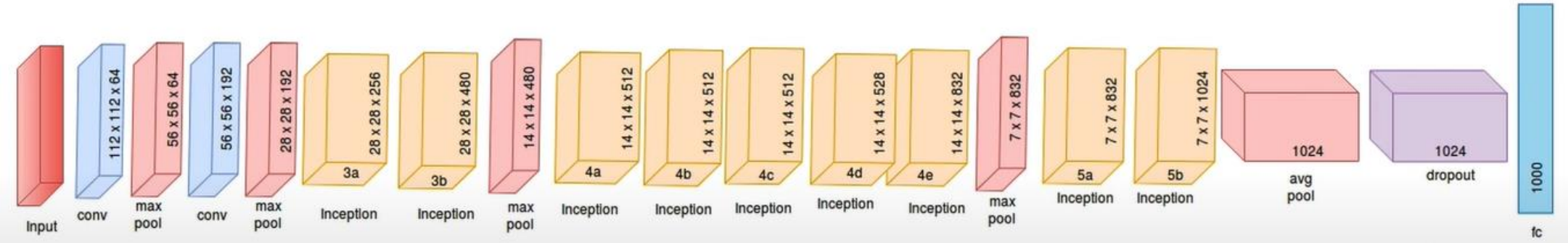


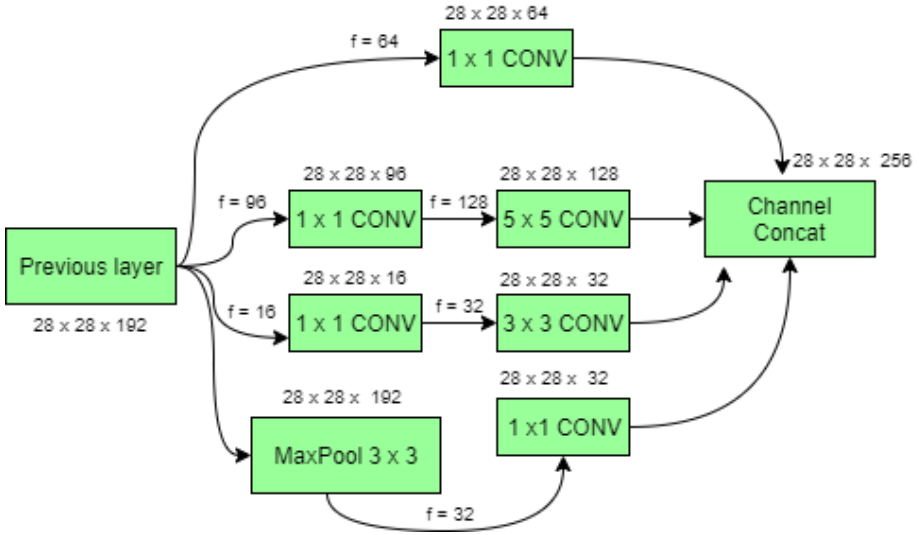
Inception Networks (V1)

- ImageNet is a dataset consisting of millions hand-labeled images categorized in 1000 classes.
- Inception network was once considered a state-of-the-art deep learning architecture for solving image recognition and detection problems.
- better deep learning models- bigger model, either in terms of deepness, i.e., number of layers, or the number of neurons in each layer.
- May Overfit in case of smaller datasets.
- Kernel size matters.
- GoogLeNet has 9 such inception modules stacked linearly.



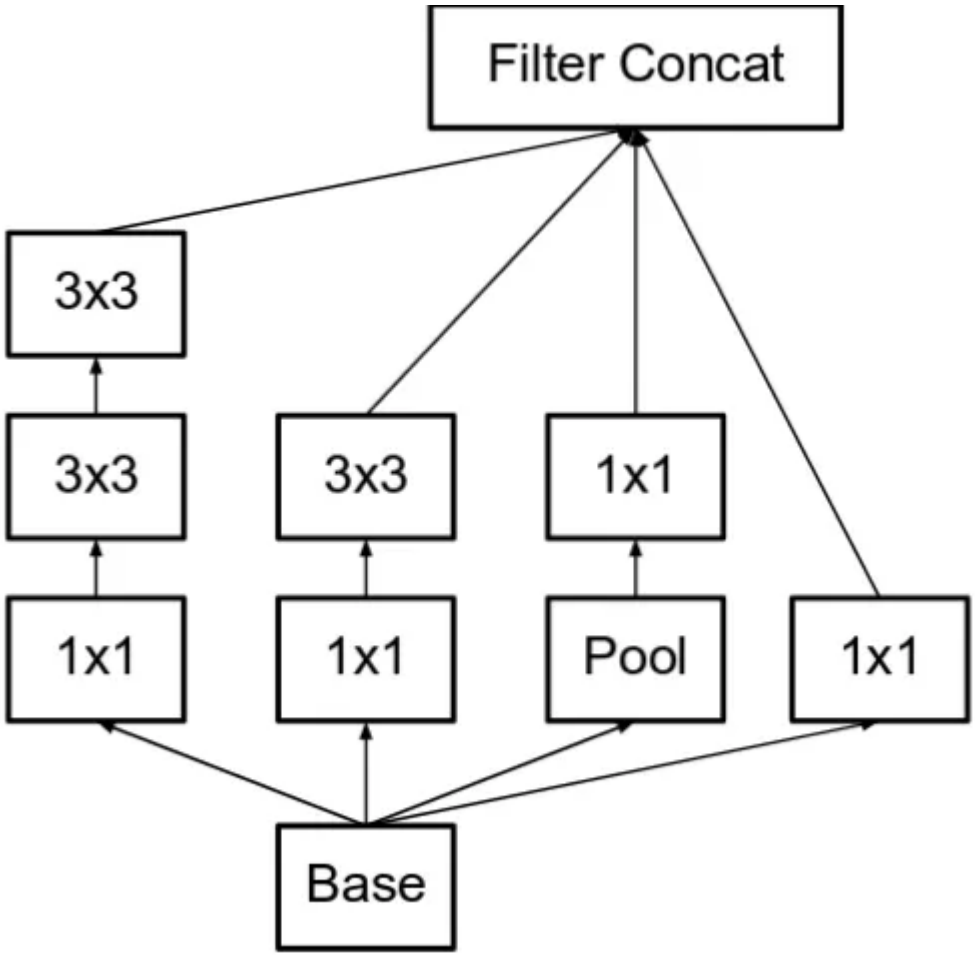
Inception Networks (V1)





Type	Max pool	Inception 3a
Patch size/stride	3 X 3/2	
Output size	28 X 28 X 192	28 X 28 X 256
Depth	0	2
# 1 X 1		64
# 3 X 3 reduce		96
# 3 X 3		128
# 5 X 5 reduce		16
# 5 X 5		32
Pool projection		32
Parameters		159K
Operations		128M

Inception V2

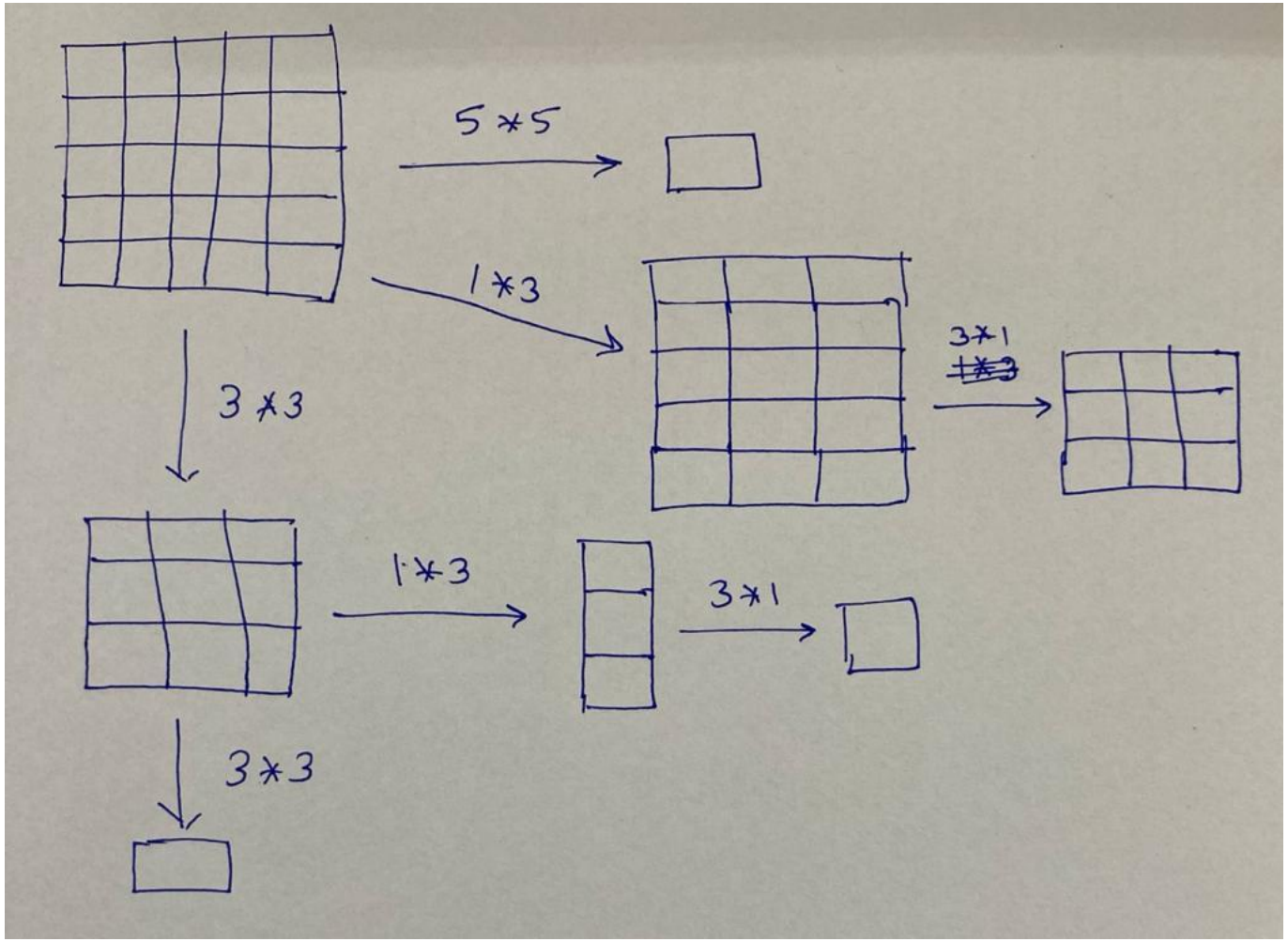


neural networks perform better when convolutions didn't alter the dimensions of the input drastically. Reducing the dimensions too much may cause loss of information, known as a "representational bottleneck"

Using smart factorization methods, convolutions can be made more efficient in terms of computational complexity. (**Factorize 5x5** convolution **to two 3x3** convolution operations).

a 5x5 convolution is 2.78 times more expensive than a 3x3 convolution. So stacking two 3x3 convolutions infact leads to a boost in performance. This is illustrated in the below image.

Inception V2



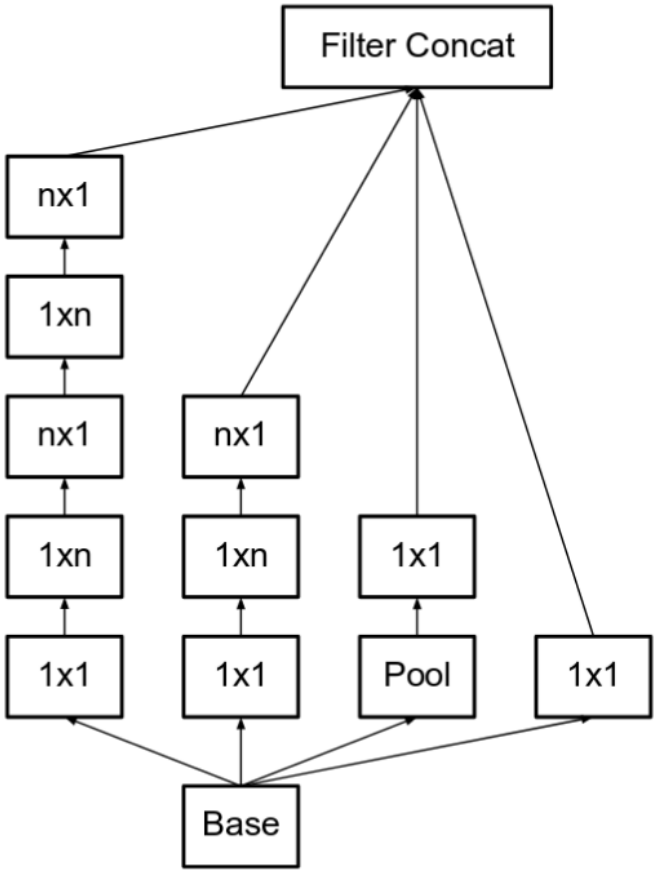
Moreover, they factorize convolutions of filter size nxn to a combination of 1xn and nx1 convolutions.

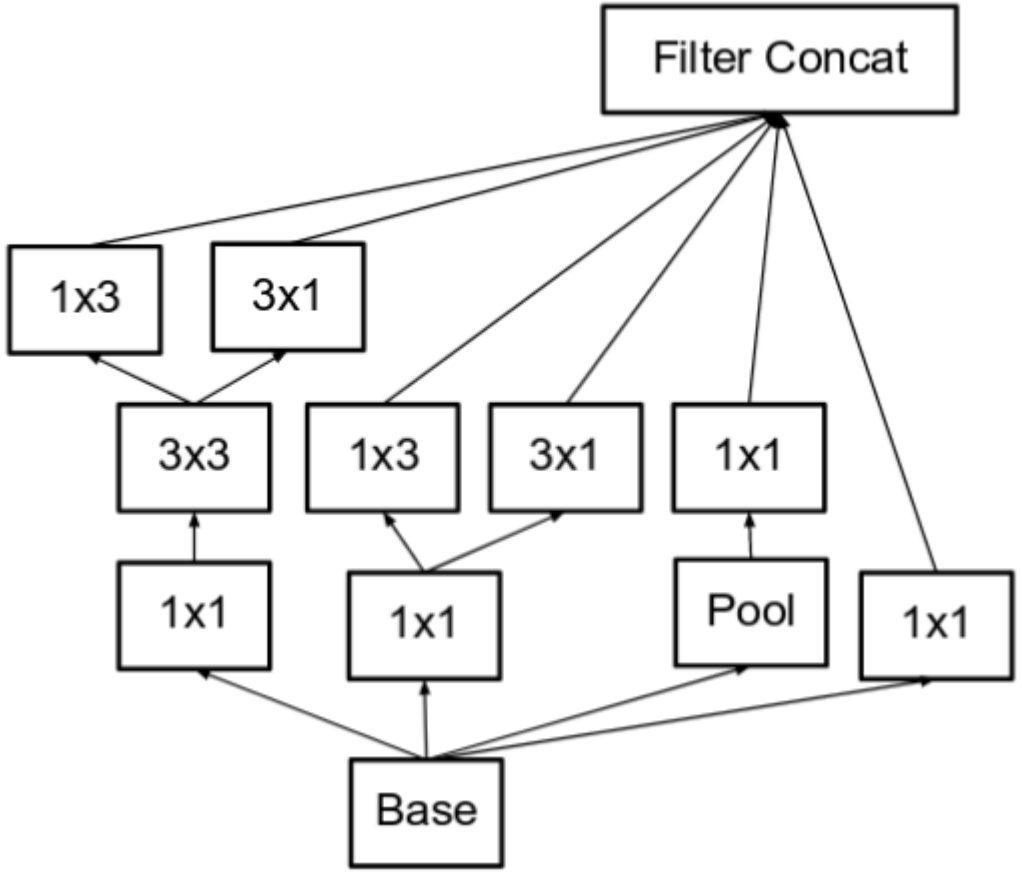
For example, a 3x3 convolution is equivalent to first performing a 1x3 convolution, and then performing a 3x1 convolution on its output.

They found this method to be 33% more cheaper than the single 3x3 convolution. This is illustrated in the below image.

Inception V2

- The **filter banks** in the module were **expanded** (made wider instead of deeper) to remove the representational bottleneck.





Inception V3

- RMSProp Optimizer.
- Factorized 7x7 convolutions.
- BatchNorm in the Auxillary Classifiers.
- Label Smoothing (A type of regularizing component added to the loss formula that prevents the network from becoming too confident about a class. Prevents over fitting).

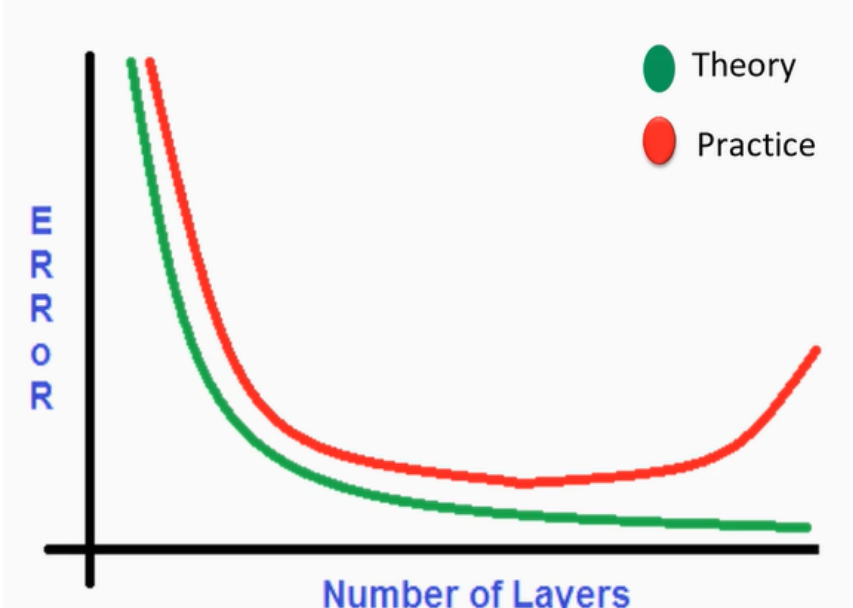
Inception V4

- Make the modules more **uniform**.

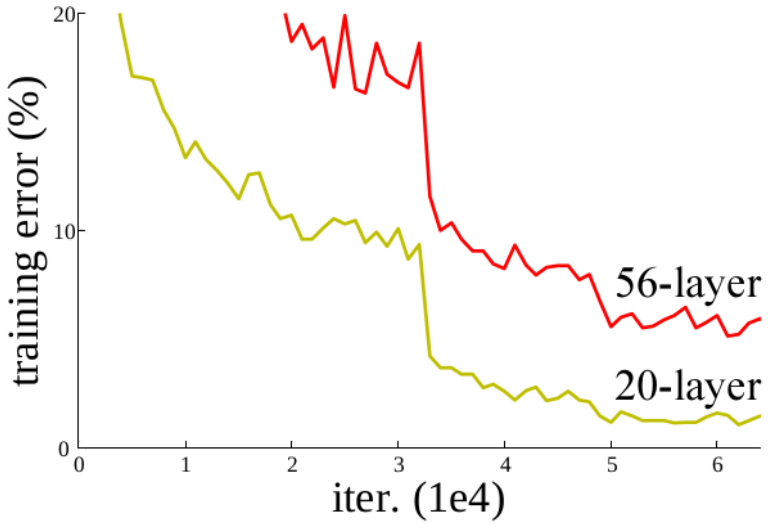
- Inception v4 introduced specialized “**Reduction Blocks**” which are used to change the width and height of the grid.

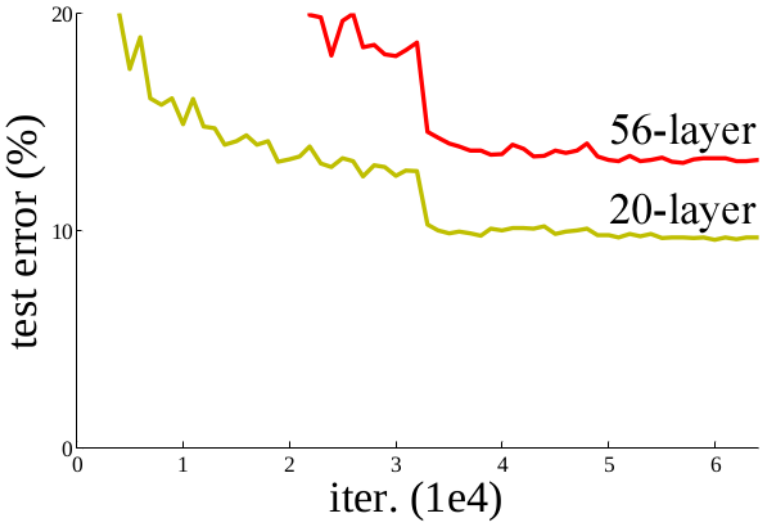
Vanishing Gradient Problem

- Learning simplex vs. Complex function in a deep networks
- Model Complexity vs. Model Accuracy- A trade off
- Vanishing gradient: when the network is too deep, the gradients from where the loss function is calculated easily shrink to zero after several applications of the chain rule. Not caused by overfitting.
- Curse of Dimensionality



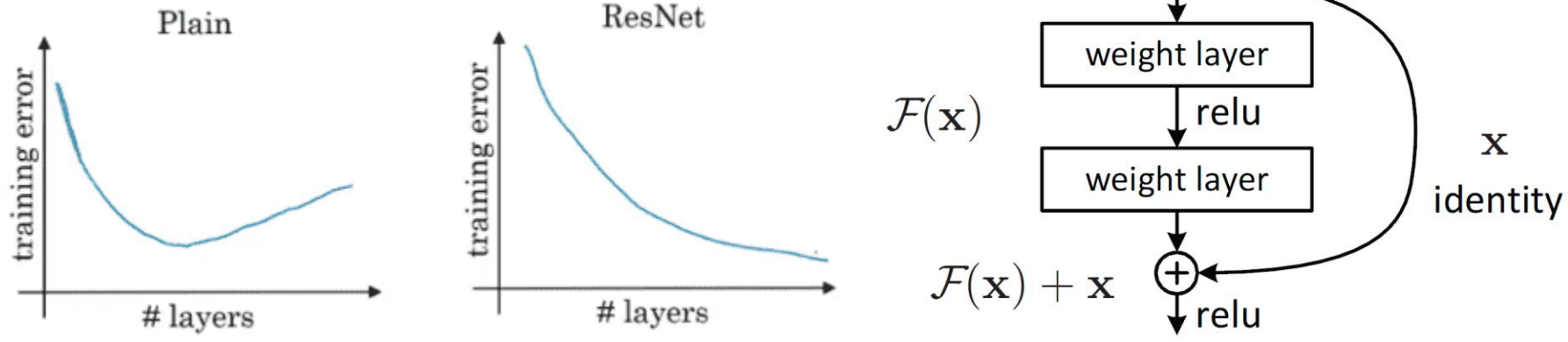
Training Testing Error





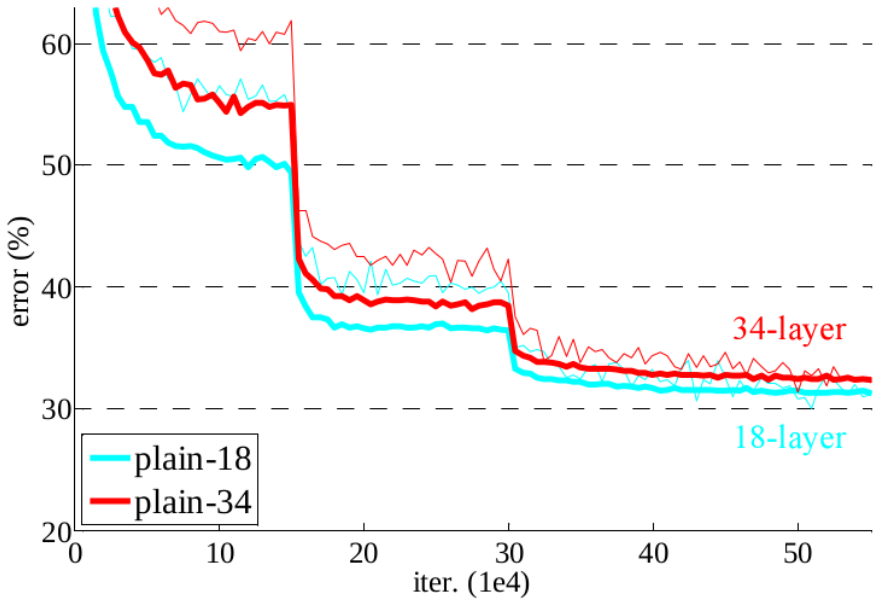
Residual Networks (ResNets)

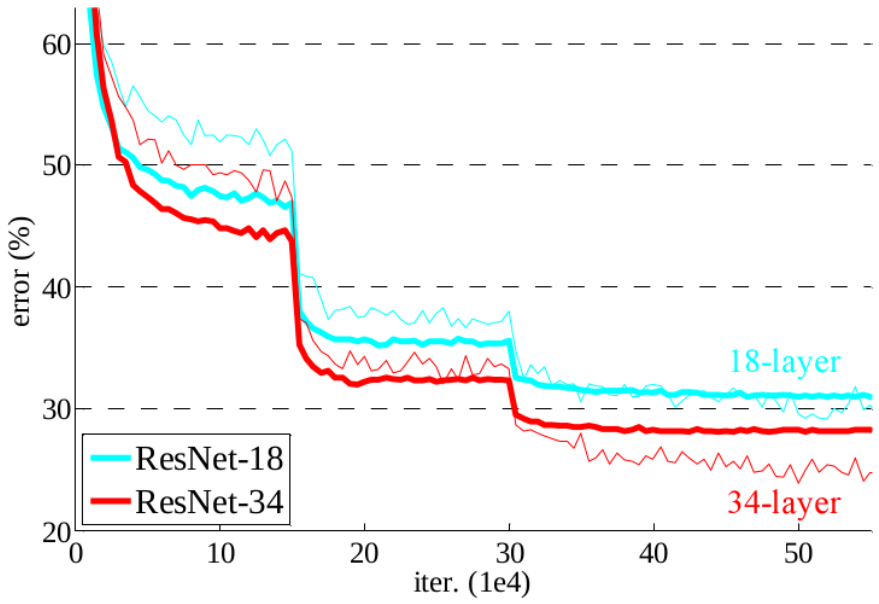
- Shortcut connections are those skipping one or more layers.
- Identity shortcut connections add neither extra parameter nor computational complexity.
- The entire network can still be trained end-to-end by SGD with backpropagation.
- Our extremely deep residual nets are easy to optimize, but the counterpart “plain” nets (that simply stack layers) exhibit higher training error when the depth increases.
- Our deep residual nets can easily enjoy accuracy gains from greatly increased depth, producing results substantially better than previous networks.



Training Testing Error

- Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer “plain” networks.

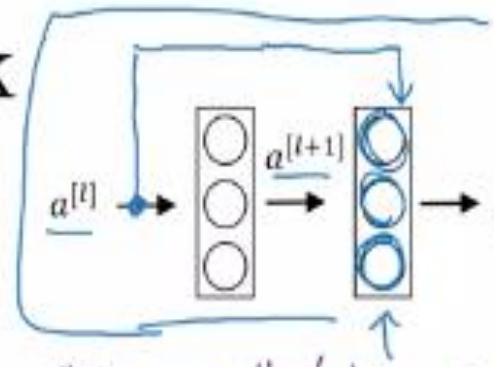




ResNet

- Initially applied to the image recognition task but the framework can also be used for non computer vision tasks also to achieve better accuracy.
- Is getting better model performance as easy as stacking more layers?
- Gradients can flow directly through the skip connections backwards from later layers to initial filters.

Residual block



"short cut" / skip connection

$a^{[l]}$ → Linear → ReLU $a^{[l+1]}$ → Linear → ReLU → $a^{[l+2]}$

"main path"

$$z^{[l+1]} = W^{[l+1]} a^{[l]} + b^{[l+1]} \qquad a^{[l+1]} = g(z^{[l+1]}) \qquad z^{[l+2]} = W^{[l+2]} a^{[l+1]} + b^{[l+2]} \qquad a^{[l+2]} = g(z^{[l+2]})$$

$$a^{[l+2]} = g(z^{[l+2]} + a^{[l]})$$

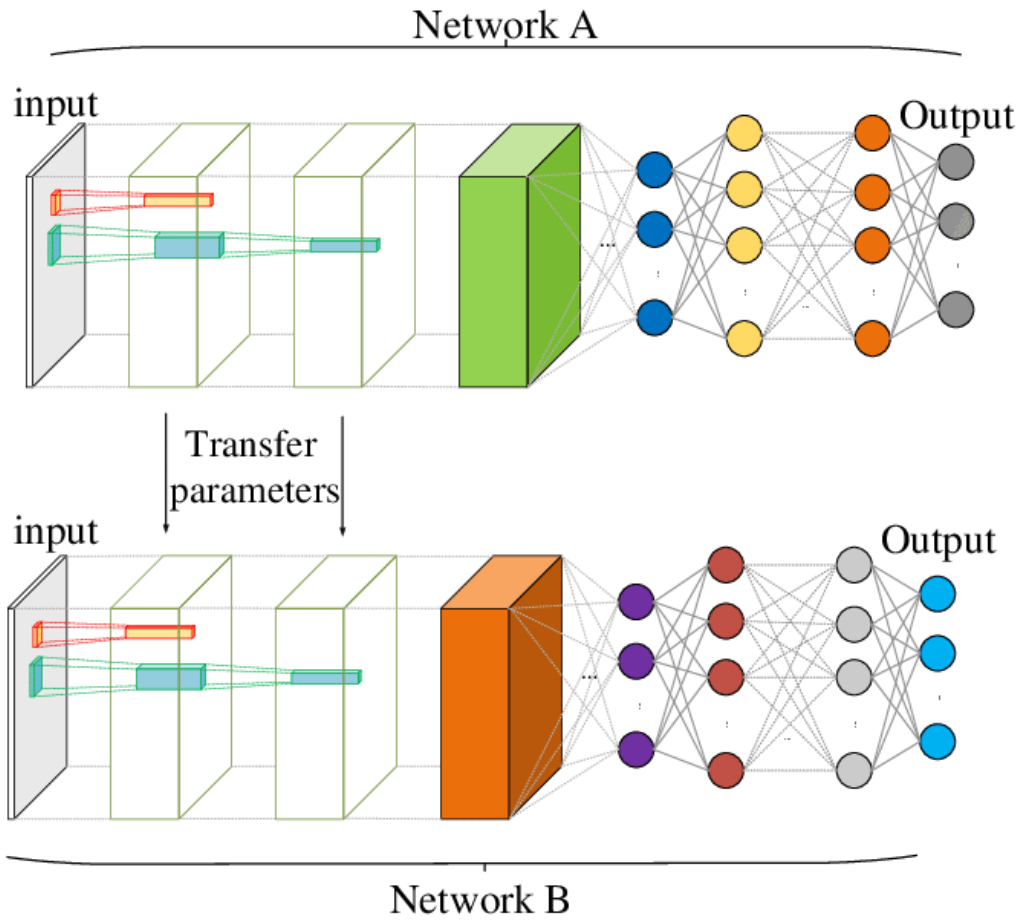
[He et al., 2015. Deep residual networks for image recognition]

Andrew Ng

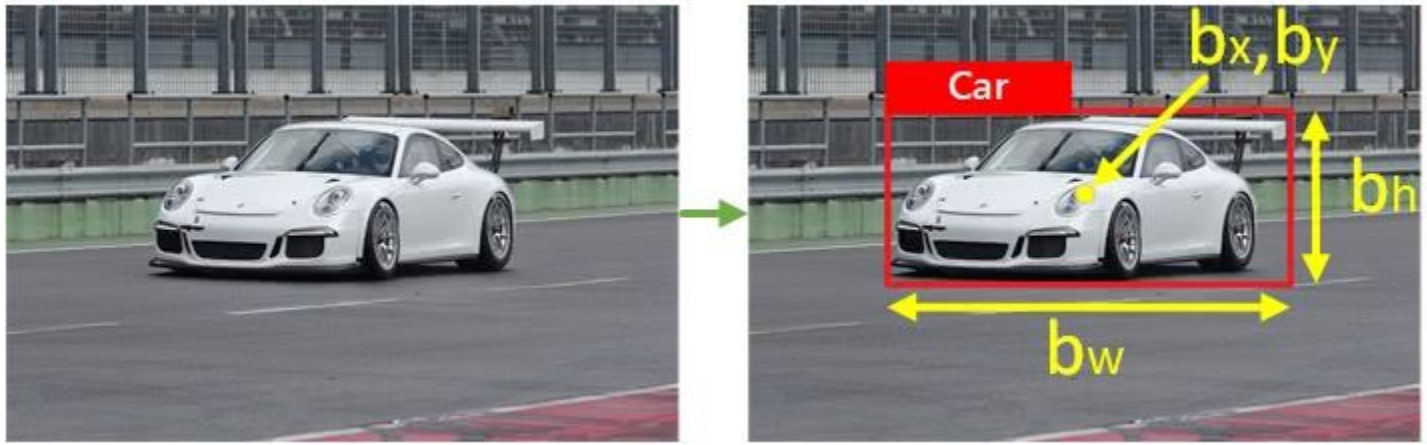
Some Tips on Starting using CNNs

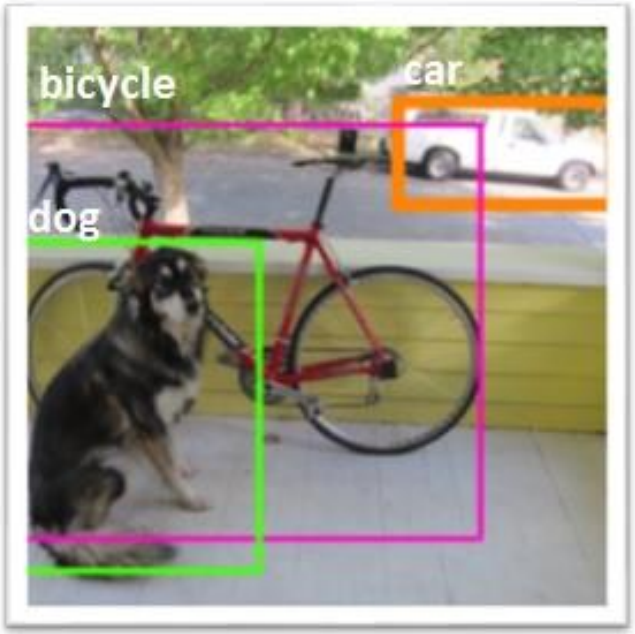
- Use open source repository.
- Use transfer learning to fine tune your datasets.
- Data Augmentation Techniques.

Transfer Learning

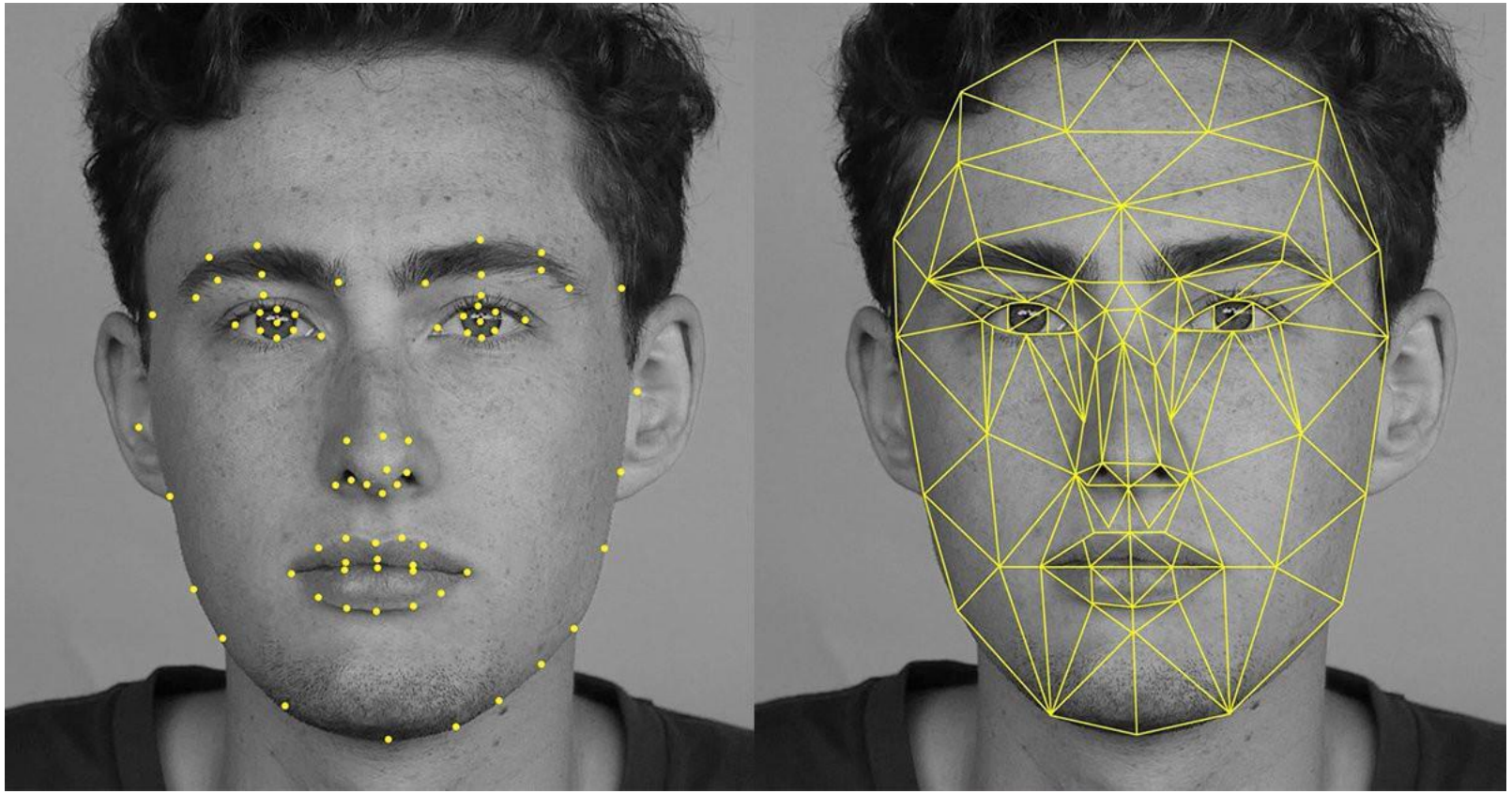


Object Localization





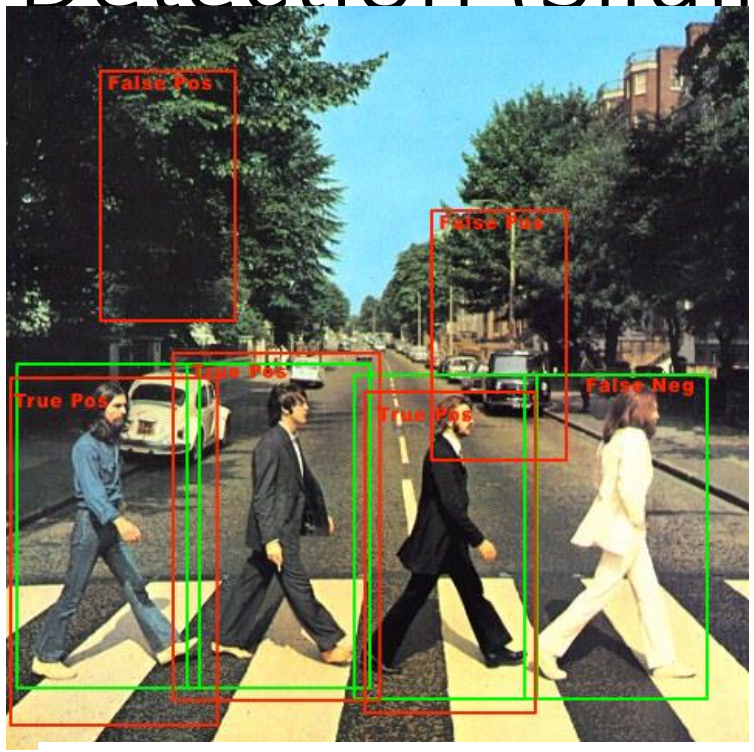
Landmark Detection

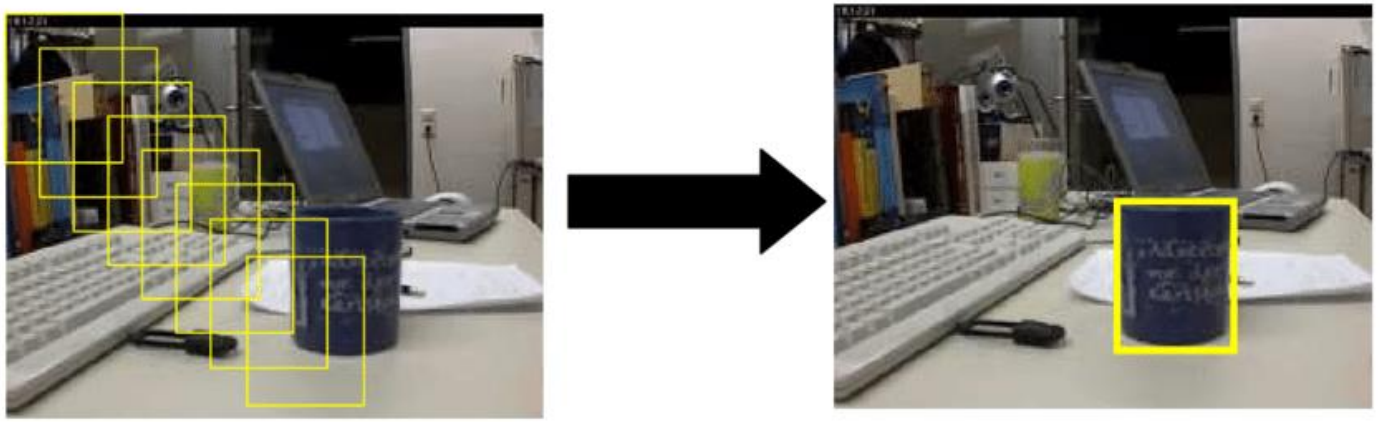


Neural Style Transfer

- Neural Style Transfer refers to a class of software algorithms that manipulate digital images, or videos, to adopt the appearance or visual style of another image.

Object Detection (Sliding Window)





Loss/Error Convergence

- The general methodology to build a Neural Network is to:

1. Define the neural network structure (# of input units, # of hidden units, etc.)
2. Initialize the model's parameters
3. Loop:
   - Implement forward propagation
   - Compute loss
   - Implement backward propagation to get the gradients
   - Update parameters (gradient descent)

Hyperparameters Tuning

- Learning Rate & Learning rate decay
- Mini batch Size
- No. of Layers
- No. of neurons in each layers
- Optimizations tech. parameters

Many similarities with the brain

Property	Human Visual System Property	Deep Learning CNN Building Block
Locality	Low-level neurons respond to local patches (receptive field)	Local computation of convolutional filters (not a fully-connected network)
Filters	Specialized neurons carry out low-level detection operation	Low-level filters carry out the same operation throughout the input
Layers/Abstraction	Layers of neurons learn increasingly abstract 'concepts'	Layers of hidden units, abstract concepts learned from simpler parts/building blocks
Threshold	Neurons fire after cross activation threshold → non-linearity	Activation functions introduce non-linearities → expand universe of functions
Pooling	Higher-level neurons invariant to exact position, sum/max of prev.	Max/Avg pooling layers: positional invariance reduced # parameters, speed up compute
Multimodal	Different neurons extract different features of image	Multiple filters applied simultaneously, each captures different aspects of original image
Saturation	Neurons 'tired' after activation, signal quiets down	Limiting weight of individual hidden units, dropout learning, regularization
Feed-forward edges	Neurons with long connections from lower levels to higher ones	Residual networks (ResNets) feed lower-level signal, avoid vanishing gradients