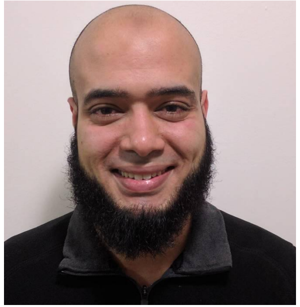
Machine Learning

Convolutional Layer

Mostafa S. Ibrahim
Teaching, Training and Coaching for more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / MSc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)

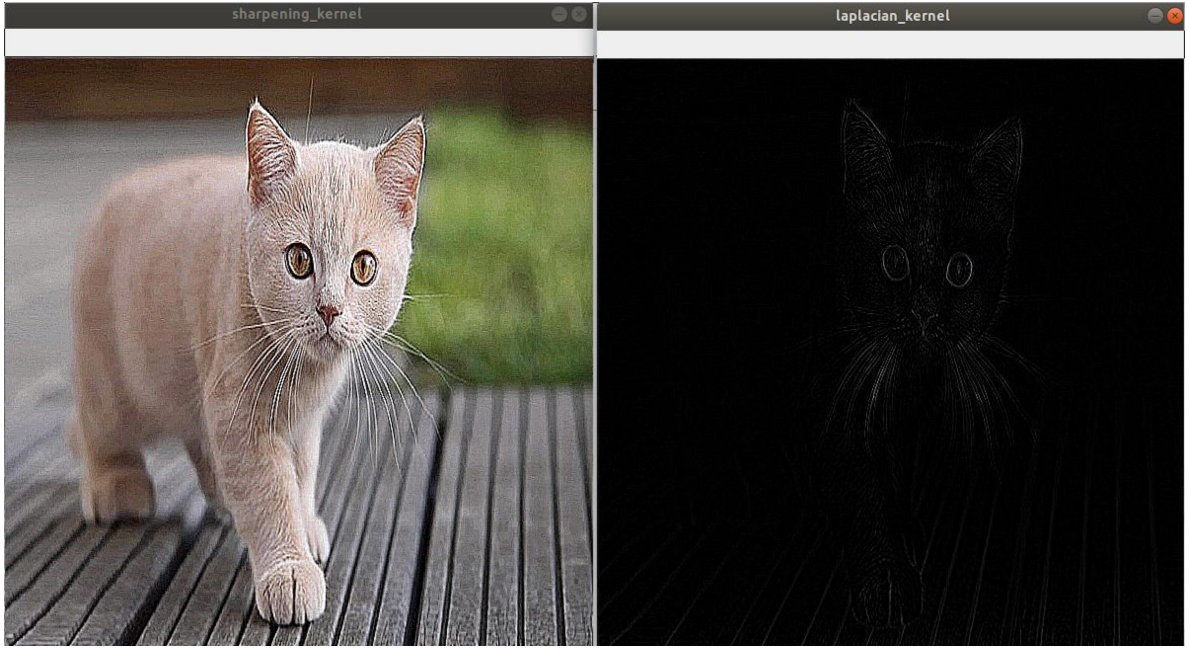


A classification task

- Let's pretend our input is a gray image of 224x224
- We don't want to **extract the features** by ourselves!

Predefined Filters

- There already several predefined filters that extract specific features!
- But this can't help in finding **relevant and complex** features to our **goal**!




```
# Laplacian kernel
laplacian_kernel =
        [0, 1, 0],
        [1, -4, 1],
        [0, 1, 0]
], dtype=np.float3
apply_custom_filte

sharpening_kernel
        [-1, -1, -1],
        [-1, 9, -1],
        [-1, -1, -1]
], dtype=np.float3
apply_custom_filte
```

Sequence of Predefined Filters

- What if we defined series of filters and aggregated all results?
    - Blur ⇒ Blur ⇒ Blur
    - Blur ⇒ Blur ⇒ Laplace
    - Blur ⇒ Laplace
    - Laplace ⇒ Blur ⇒ Blur
- Cool idea to generate more features! But then what?
    - Still limited
    - Doesn’t align with our goal (loss)

Deep Learning Key

- What if instead of defining a **fixed** 3x3 filter, if we train the network to **learn a filter** that is suitable for **our loss**!
    - Assume the **input image X** (6x6) is convolved using **valid padding** and stride = 2

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30
31	32	33	34	35	36

w1	w2	w3
w4	w5	w6
w7	w8	w9

o1	o2
o3	o4

Max pool

o

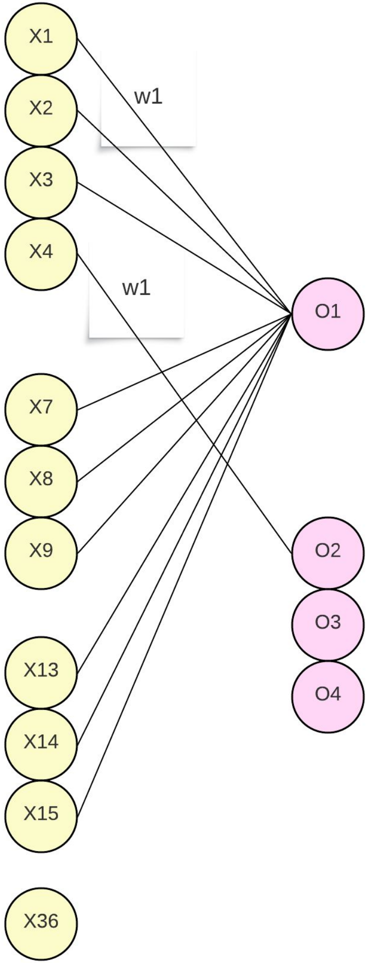
Sigmoid + Log loss

o1 = x1 * w1 + x2 * w2 + x3 * w3 +
x7 * w4 + x5 * w2 + x6 * w3 +
x13 * w7 + x14 * w8 + x15 * w9 +

Eventually just a **computational graph**!
However, each weight will appear in different output nodes

Shared Weights

- If you tried to sketch the network like in NN, you will notice we need draw the same weight multiple times
- This is because the weight is **shared** in every single convolution evaluation
- From NN to DNN
    - DNN like NN can just be represented as **computational graph**
    - However, it is not regular/dense as DNN
        - A single weight is not attached to every input
        - A single weight can be shared with many inputs
        - The network is so deep
    - This **depth and complexity** are what differentiate DNNs from NNs



Parameters in One Filter

- To complement the picture, remember our normal NN has activation(WX+b)
  - So our o1 node is actually
  - o1_net = x1*w1+ x2*w2+x3*w3+x7*w4+x5*w2+x6*w3+x13*w7+x14*w8+x15*w9 **+ bias**
  - o1_out = activation(o1_out)
    - The activation for intermediate layers is typically RELU-based
- How many parameters do we have?
  - For a 2D input (WxH), we have KxK + 1

Multi-dimensional input

- For a single 2D input, we have a single filter KxK
  - What if we have multiple input layers, such as 3 in RGB images?
  - In theory, we can use the same filter for each 2D input
  - However, in practice it is common to have **one filter <u>for every</u>** 2D input
  - But keep in mind, after we aggregate all, we add a single bias
- Parameters for multi-dimensional input
  - Let input be C1 x W x H
  - Apply kernel of K x K
  - Then we need **C1 x K x K +1** parameters
    - NOT: C1 x (K x K +1): one shared bias only
  - Observe: both **filter inference and parameters** count doesn't depend on the resolution (WxH)
- In DNN, we express all of the kernels **as a single 3D filter** of size: C1 x k x K
  -

Multi-dimensional input

1	2	3	4
7	8	9	8
8	14	4	16
19	5	3	22

3	1
2	5

Kernel 1: 2x2

5	1	2	3
2	6	1	4
1	2	13	7
5	15	1	6

1	4
3	5

Kernel 2: 2x2

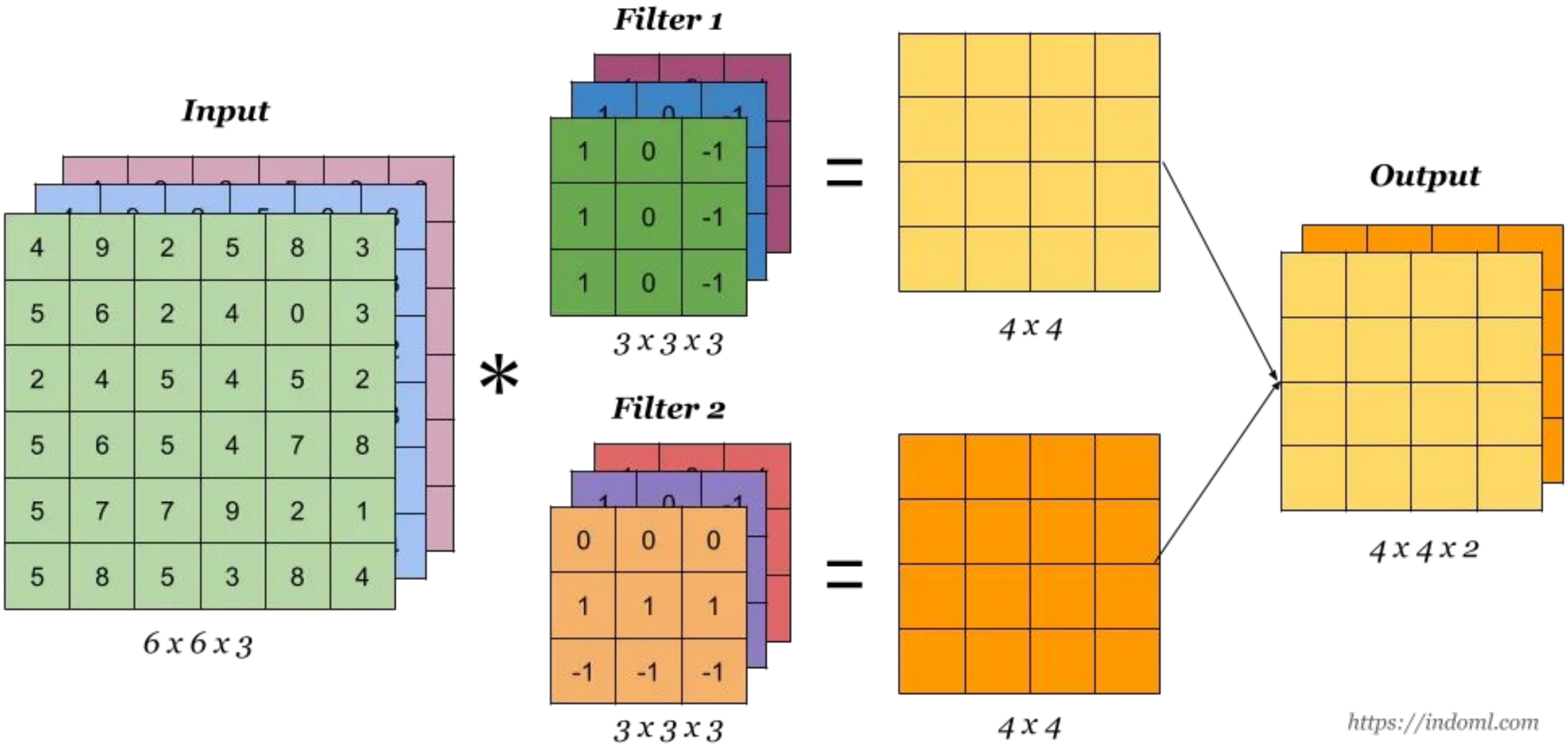
10

Shared Bias

1x3+2x1+ 7x2+8x5+ 5x1+1x4+ 2x3+6x5+ 10	
	Under stride = 2

Multiple Filters

- Back then, we can extract multiple fixed filters (blue, gaussian, laplacian, etc)
- What if we want to learn **multiple filters** for a **multidimensional** input?
- Then for each filter to learn, repeat the process **independently**
- Parameters for multiple filters for multi-dimensional input
    - Let input be C1 x W x H
    - Apply kernel of K x K
    - Assume we want to learn C2 filters (independent)
        - Then generated output is: C2 x W x H
    - Then we need **C2 x** (C1 x K x K +1) parameters
- Feature map
    - We call the input and output as feature **maps**
        - A 3D tensor: C x W x H
    - The **RGB** image is the **first** input feature map (3 x W x H)



- Input C1=3 feature maps and output C2=2 feature maps with stride jump
    - 2 3D filters each C1 x K x K (where K=3)

Img [src]

Number of Operations

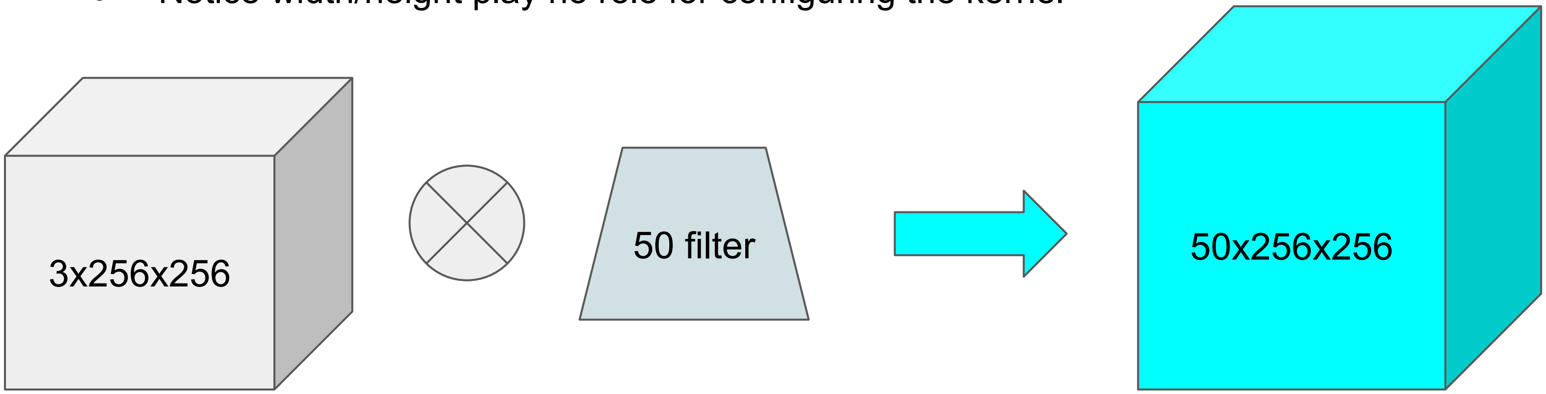
- How many operations to a apply kernel KxK for once?
  - We multiply K*K numbers from the kernel with the input
  - Then we **add** them together (observe first value is not added) for a total of K*K - 1
    - Adding 1+2+3+4+5+6 requires 5 additions not 6
  - So in total 2*K*K-1 (2 * multiplications - 1)
- Applying K*K kernel over C1 input maps
  - 2 * multiplications - 1 ⇒ 2*K*K*C1 - 1
- Generating C2 independent feature maps
  - C2 x [2*K*K*C1 - 1]
- Applying that for every pixel location on WxH (same padding)
  - W x H x C2 x [2*K*K*C1 - 1]

Convolutional Layer

- A normal NN layer just receives input of D1 features and map to D2 features
  - So we learn one matrix of weights D1 x D2
- A convolutional layer **transforms** an input feature map C1xWxH to output feature map C2xWxH through convolution operation
  - So C1 2D inputs into C2 2D outputs
  - All operations are just addition and multiplication
- We can treat its *[C2 x (C1 x K x K +1)]* parameters as weights and use it as a building block inside our neural network
- As a result, the network will learn C2 filters
  - Each DNN filter is a set of kernels (3D: C1x**KxK**) to process the C1x**WxH** input

Convolutional Layer

- Now, start to abstract this operation and its input and output
  - Input feature map: 3x256x256 (e.g. image input)
  - Convolution layer: 50 filters each filter is 3x3x3
  - Output feature map: 50x256x256
- In Pytorch: nn.Conv2d(3, 50, kernel_size=3, padding='same')
  - It means we convert 3 channel inputs to 50 channel inputs using kernel=3
    - Default padding: valid (0)
  - Notice width/height play no role for configuring the kernel



"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."