CS772: Deep Learning for Natural Language Processing (DL-NLP)

Convolutional Neural Networks

Jimut Bahan Pal

C-MInDS, IIT Bombay

23rd September, 2025

Instructor: Prof. Pushpak Bhattacharyya

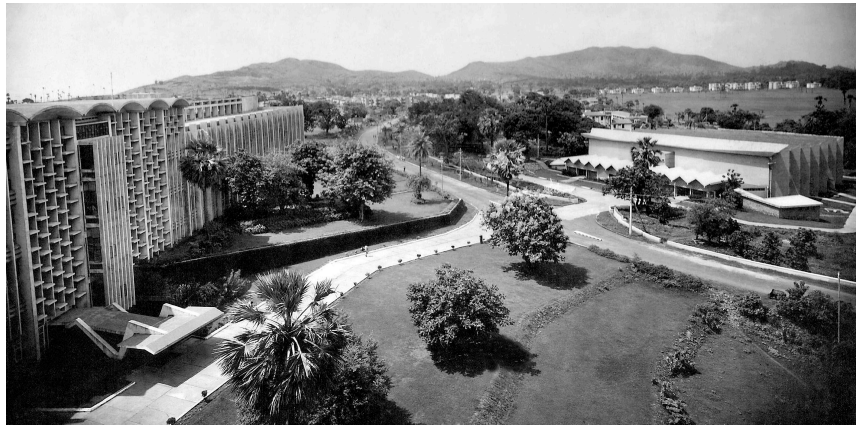
Contents

- Images, matrices, and ambiguity of images
- A bit of history about CNNs
- Convolutional Neural Networks (CNNs)
- Key-aspect of CNNs
- Padding, Pooling, Stride, Receptive field of kernel/filter, different types of convolution
- Example of convolution and parameter calculation
- Parameter and feature calculation of a CNN (using Pytorch code example)
- Features learnt by CNNs
- More CNN architectures – VGGNet, GoogLeNet, ResNet
- Transfer Learning in CNNs
- **Demo – ResUNet++ for Multiclass image segmentation**

Images, Matrices and Ambiguity of images

Images – in Computer Vision

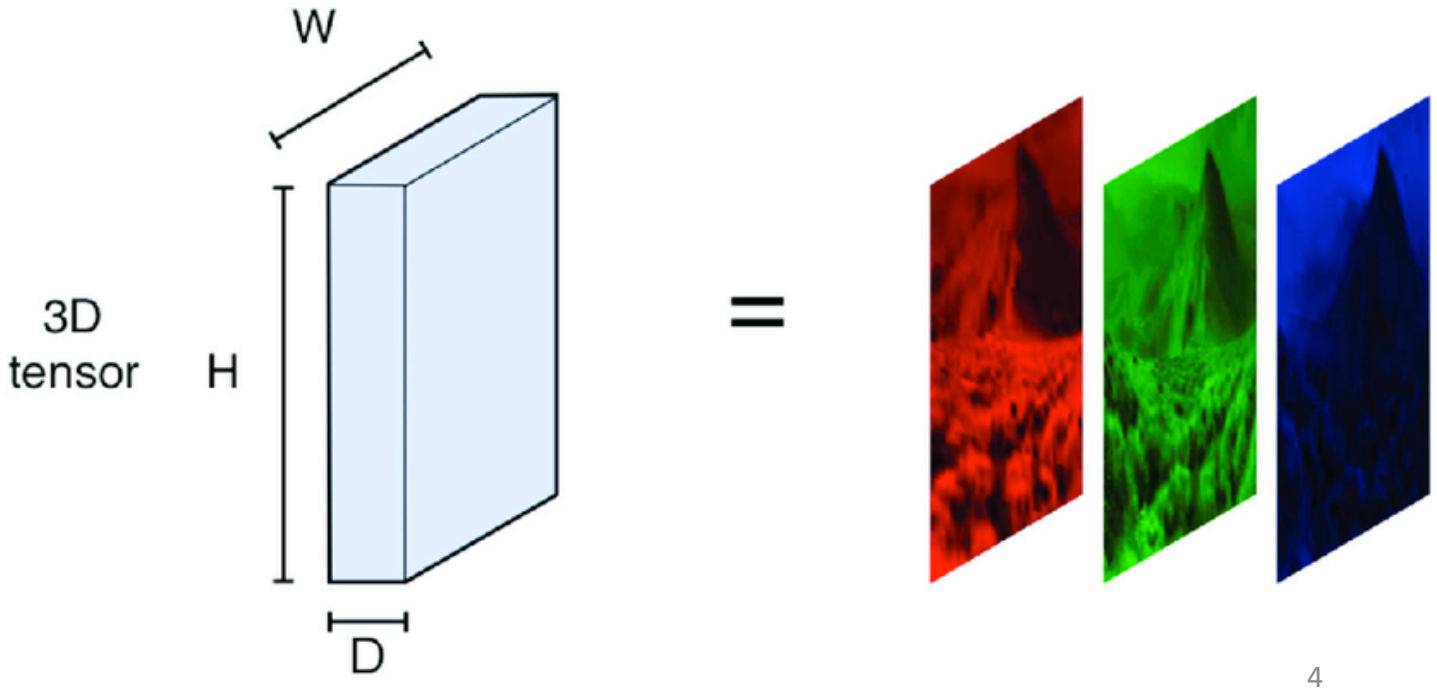
- 2D representation of visual information composed of pixels, each containing a specific color or intensity value
- Digital images are matrices of discrete data (photograph, video frames etc.)
- Natural images are normally 3-channel (RGB color) or 1-channel gray-scale.
- Can be represented by 8-bit unsigned integer, (i.e., 2^8 values); **numpy.uint8**
- Starts from 0 and ends at 255; hence, intensities are between 0-255



View from Main Building, IITB 1958

```
shape of image =  (1409, 2835)
first 10x10 pixels from top left =
 [[174 173 171 170 170 172 173 174 177 178]
 [174 173 172 171 171 172 173 174 178 179]
 [175 174 173 172 172 173 174 174 178 179]
 [176 176 175 175 175 175 175 175 179 180]
 [179 179 179 179 178 177 177 176 180 181]
 [182 183 183 183 182 181 180 179 181 182]
 [185 186 187 187 186 184 182 181 181 182]
 [187 188 189 189 188 186 184 182 182 183]
 [188 188 188 188 188 188 188 188 181 183]
 [187 187 187 188 188 189 189 189 183 184]]
```

10x10 pixels from top left



https://version2024.iitb.ac.in/en/gallery

https://www.researchgate.net/figure/RGB-image-divided-into-three-color-channels_fig2_373653146

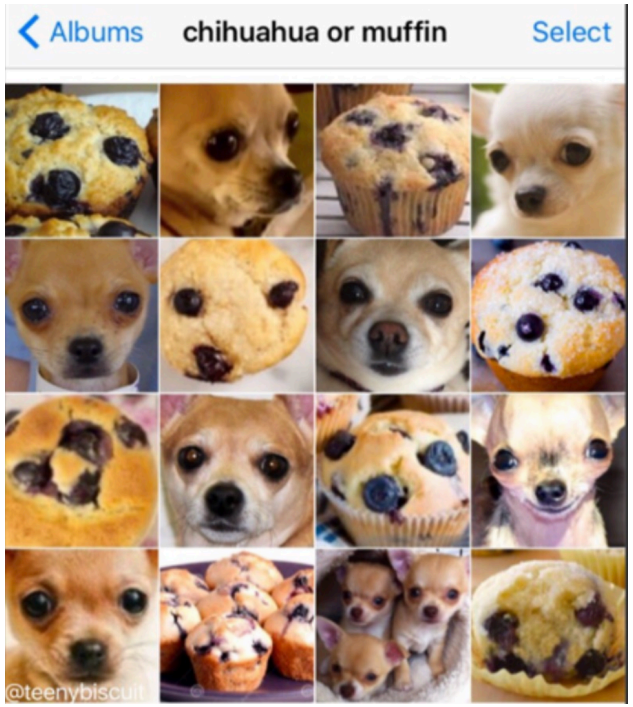
Images – in Computer Vision

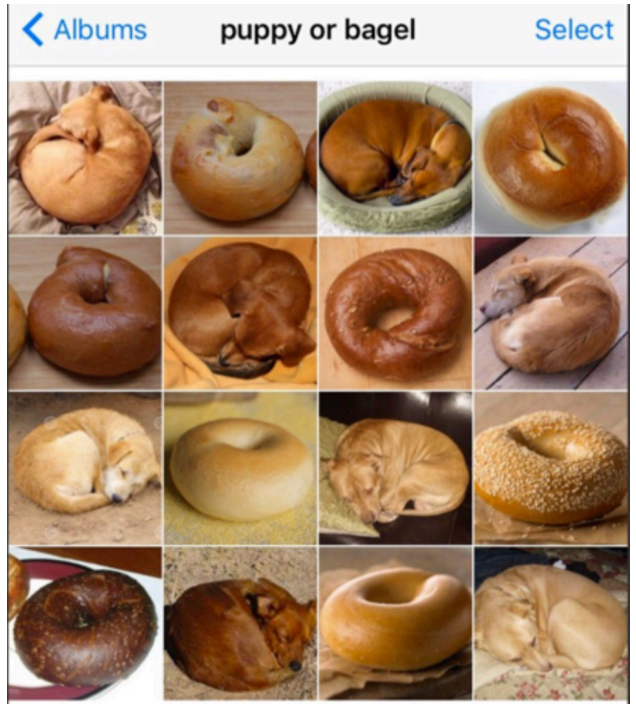
- When dealing with features in Deep Neural Networks (DNNs) and Convolutional Neural Networks (CNNs) we can have ~2^64 (torch.float64); ~2^32 (torch.float32 default) and ~2^16 (torch.float16) values depending on the precision required for computations.
- Images are often normalized between 0.0-1.0 intensities before passing into DNN models to stabilize training (avoiding large values after ReLU activations)

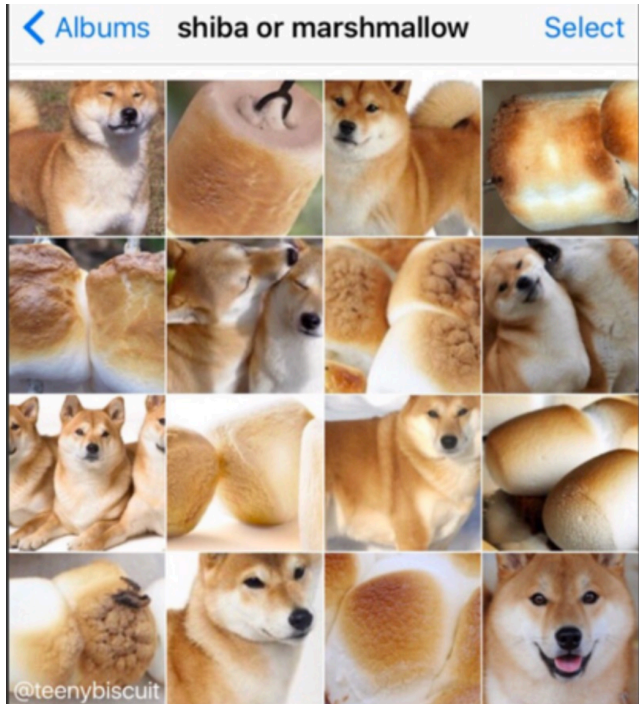
```
shape of image =
 (2088, 4640, 3)
first 5x5 pixels
 from top left =
 [[[224 198 128]
  [226 200 130]
  [230 204 134]
  [226 200 130]
  [225 199 129]]

 [[230 204 134]
  [225 199 129]
  [226 200 130]
  [223 197 127]
  [226 200 130]]

 [[232 207 137]
  [227 202 132]
  [230 205 135]
  [228 203 133]
  [232 206 136]]

 [[224 199 129]
  [223 198 128]
  [227 202 132]
  [227 201 131]
  [228 202 132]]

 [[222 197 127]
  [225 200 130]
  [228 202 132]
  [228 202 132]
  [228 202 132]]]
```

https://stackoverflow.com/questions/62111708/what-is-the-significance-of-normalization-of-data-before-feeding-it-to-a-ml-dl-m

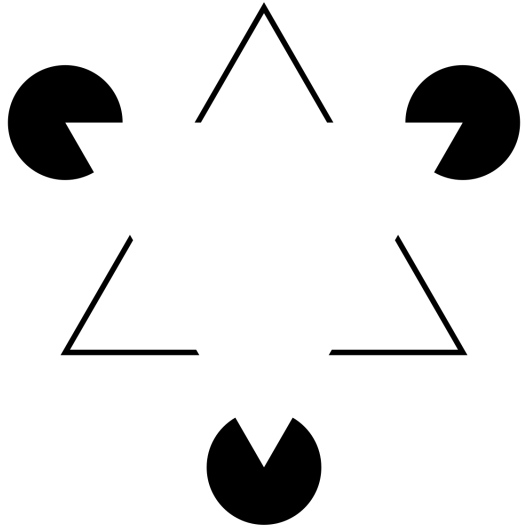
Un-realted: Images can sometimes be ambiguous

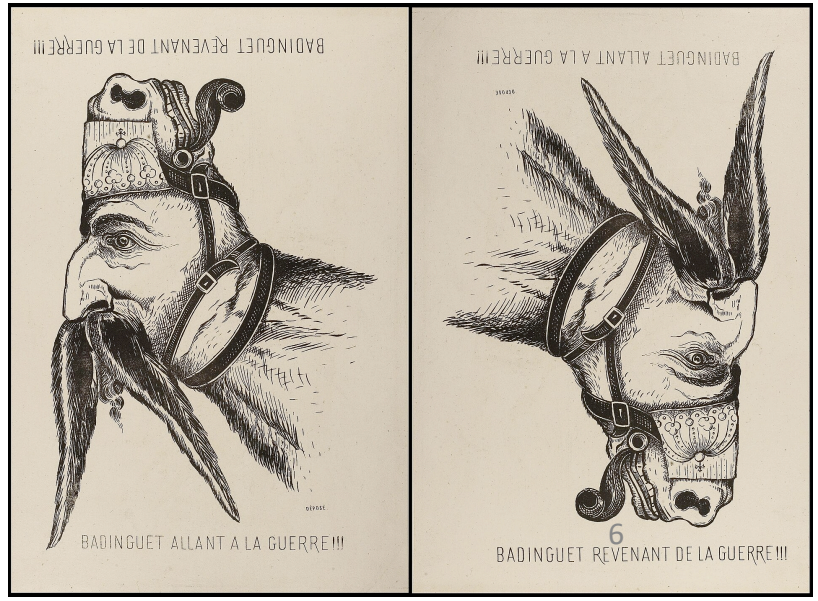






- "Kanizsa Triangle" – these spatially separate fragments give the impression of illusory contours (also known as modal completion) of a triangle.
- French caricature of Napoleon III, 1870, rotated turns into a donkey



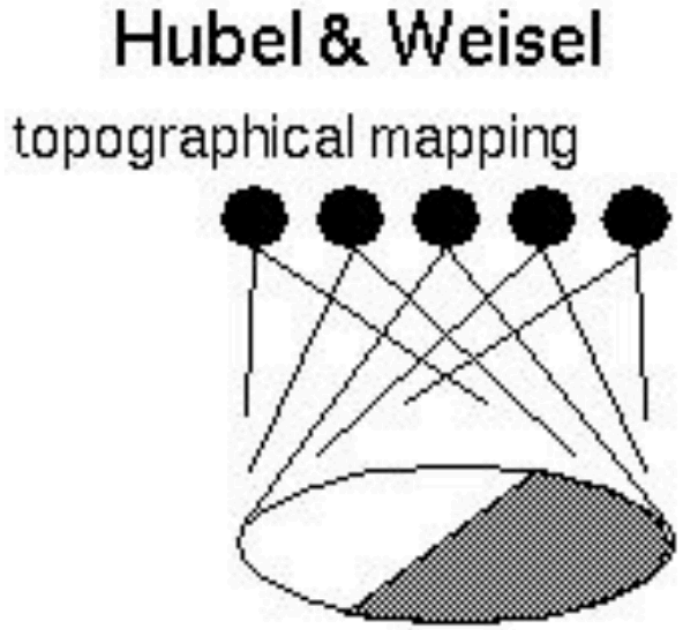


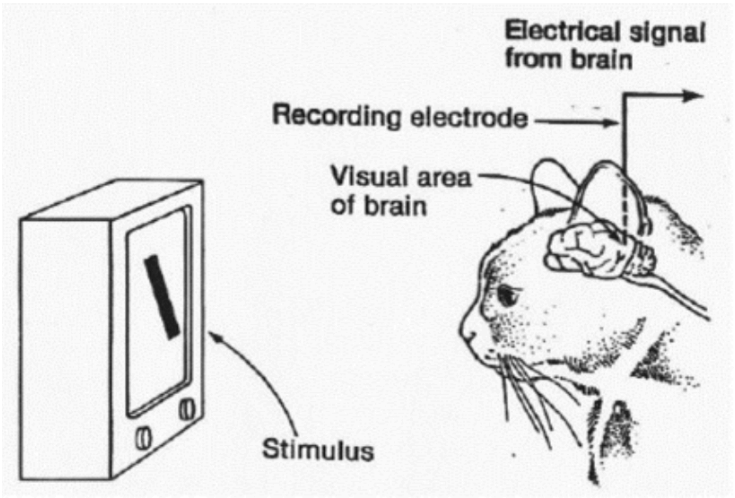
http://16385.courses.cs .cmu.edu https://en.wikipedia.org/wiki/Ambiguous_image

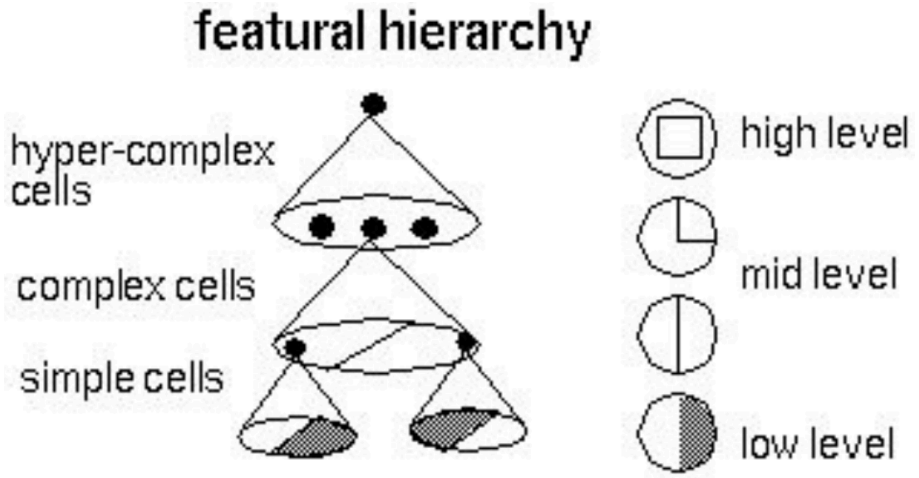
A bit of history about CNNs

A bit of history – Stimulus Response

- Hubel & Wiesel studied the responses of a cat to orientation of light stimulus by placing electrode in it's brain.
- Key Finding: cells are organized as a hierarchy of feature detectors, with higher level features responding to patterns of activation in lower level cells.



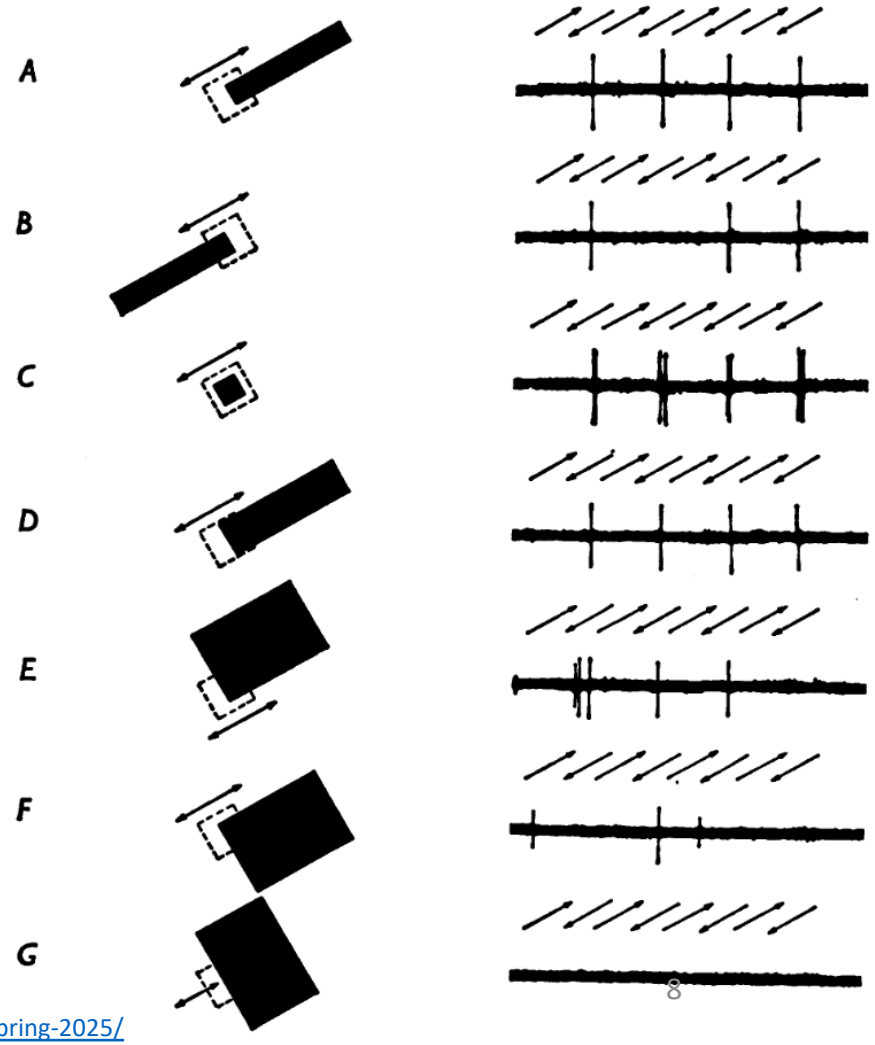




J. Physiol. (1968), **195**, *pp.* 215–243
With 3 *plates and* 14 *text-figures*
Printed in Great Britain

RECEPTIVE FIELDS AND FUNCTIONAL ARCHITECTURE OF MONKEY STRIATE CORTEX

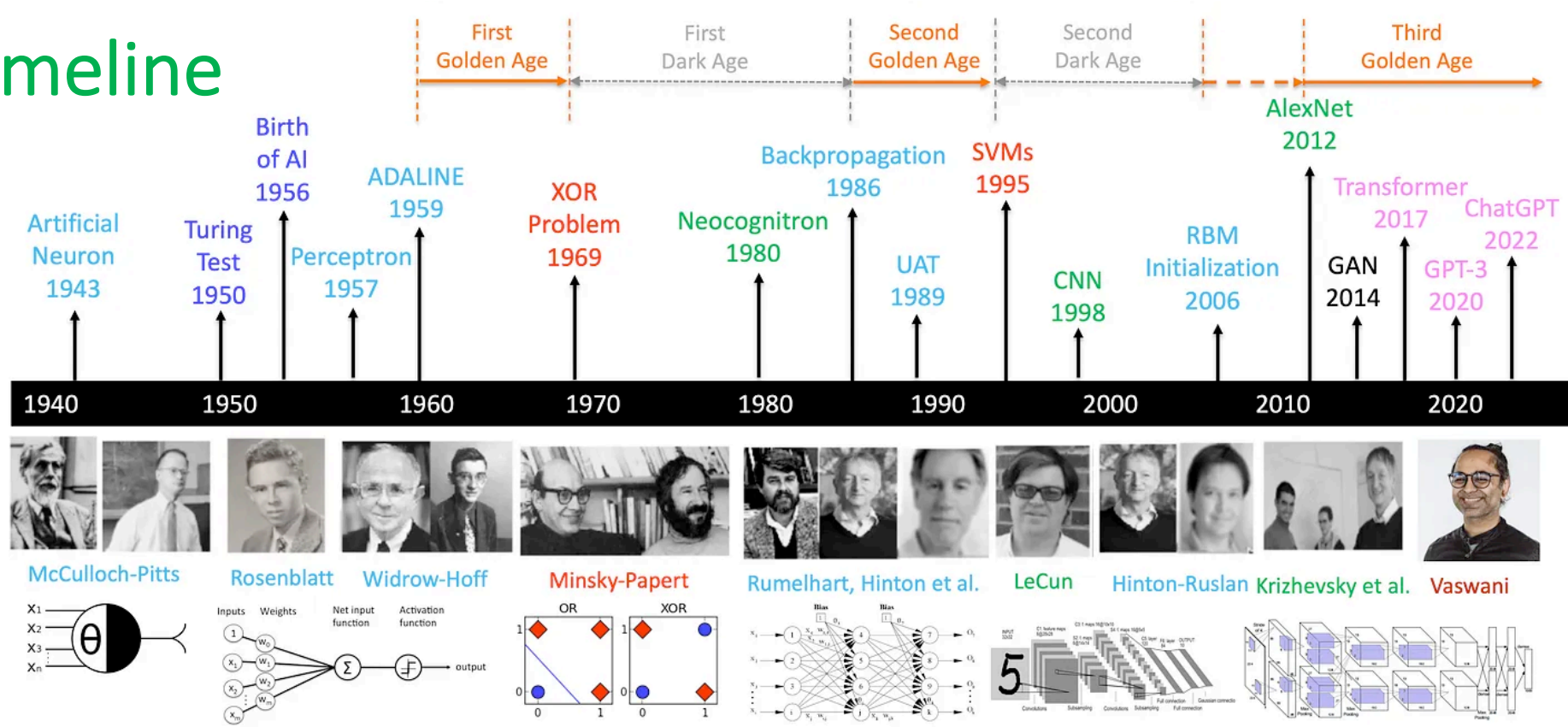
By D. H. HUBEL and T. N. WIESEL
From the Department of Physiology, Harvard Medical School, Boston, Mass., U.S.A.



https://www.cns.nyu.edu/~tony/vns/readings/hubel-wiesel-1968.pdf https://dannagurari.colorado.edu/course/neural-networks-and-deep-learning-spring-2025/

A bit of history - Timeline

- XOR Problem lead to the first AI winter.
- Popularity of SVMs caused the second AI winter.
- AI winters were usually triggered by inflated expectations and disappointing results, which led to funding cuts and fading interest.

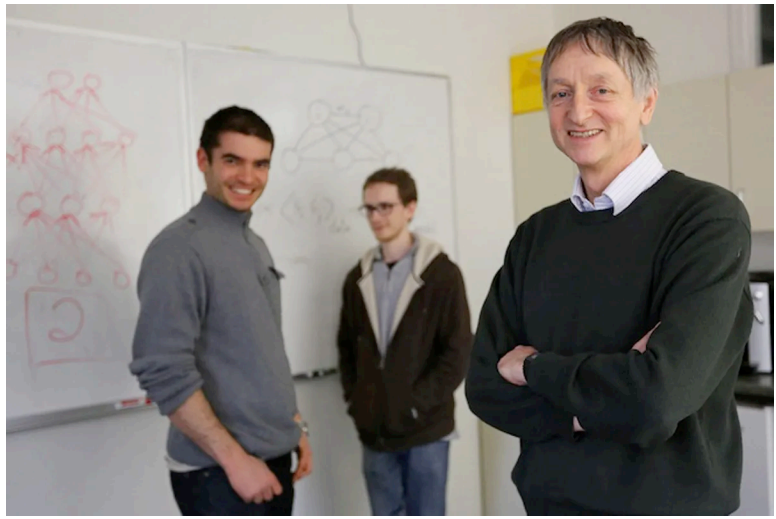


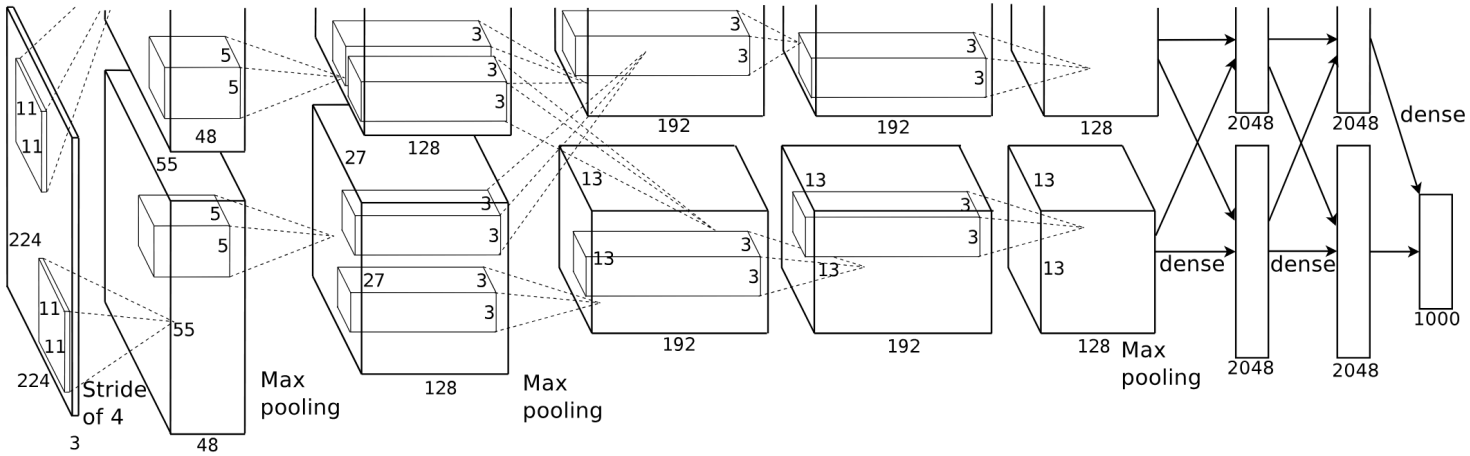
- 2018 – Turing Award in Computer Science: Bengio, Hinton and LeCun ushered in Major Breakthroughs in AI
- 2024 – Nobel Prize in Physics: John Hopfield and Hinton - for foundational discoveries and inventions that enable machine learning with artificial neural networks

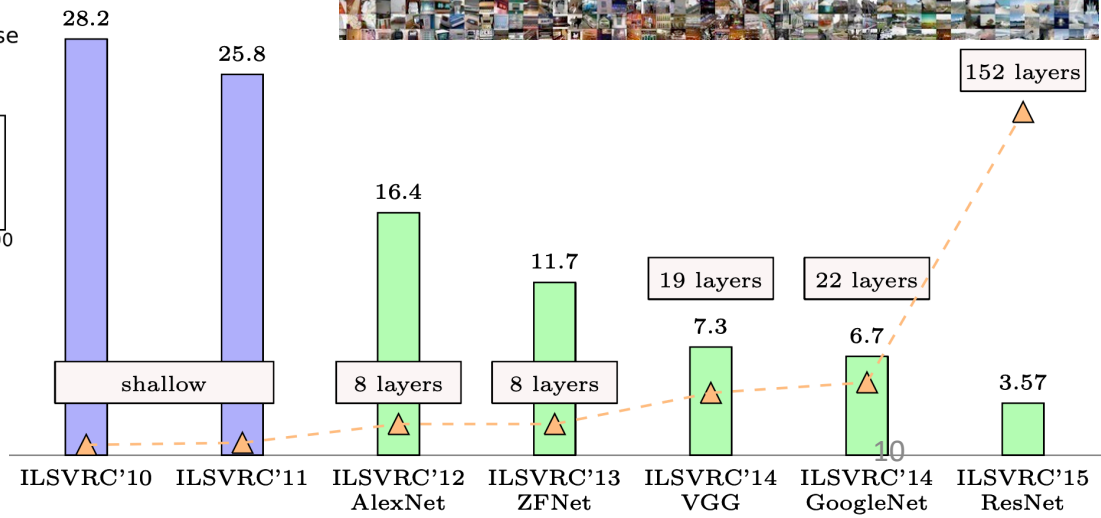
https://medium.com/@lmpo/a-brief-history-of-ai-with-deep-learning-26f7948bc87b
https://awards.acm.org/about/2018-turing
https://www.nobelprize.org/prizes/physics/2024/hinton/facts/

A bit of history – the hype of Deep Learning after AlexNet

- The first DNN architecture which was trained on 2 NVIDIA GTX 580 GPUs, having 3GB VRAM each for 5-6 days; in C++/CUDA.
- Used ReLU activation, Convolutional layers, dropout (p=0.5) as regularization, and pooling layers with parameter ~60 million.
- Won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC'12) with error rate of 15.3%, the second was 26.2%, was the first DNN method to be used in the challenge.
- Had 11x11 filters; all winner methods used DNNs after this.







https://media.wired.com/photos/5c1002bdbbcfae2d7b3dea28/3:2/w_2240,c_limit/hinton1-FINAL.jpg
https://papers.nips.cc/paper_files/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html
https://www.cse.iitm.ac.in/~miteshk/CS6910.html
https://www.image-net.org/

There were resistance to this "Revolution"

- Rejected paper uses CNN as a step in feature extraction for parsing/ segmenting scene
- The first CNN architecture created was **LeNet-5** which was trained on CPU for handwritten character recognition.

Yann LeCun's letter to CVPR organizer about 2012 submission:
(*Paper ratings: "Definitely Reject," "Borderline", "Weakly Reject")*

"… I was very sure that this paper was going to get good reviews because: 1) it has two simple and generally applicable ideas for segmentation ("purity tree" and "optimal cover"); **2) it uses no hand-crafted features (it's all learned all the way through. Incredibly, this was seen as a negative point by the reviewers!); 3) it beats all published results on 3 standard datasets for scene parsing; 4) it's an order of magnitude faster than the competing methods.**

If that is not enough to get good reviews, I just don't know what is.

Yann LeCun's Facebook post on March 28, 2019:

"The injustice of any award is that it has to pick a small number of winners. **But the winners are merely the visible part of an iceberg and wouldn't come to the surface without the much-larger submerged part that supports it…**

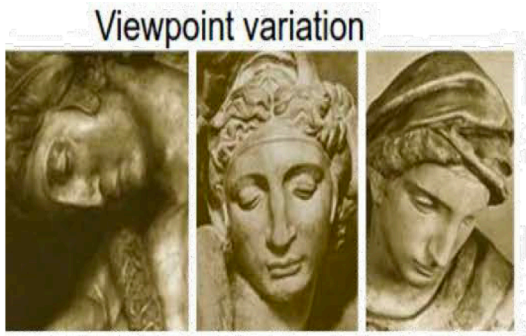
I am very thankful to all my mentors, collaborators, postdocs and students over the years. To a large extent, it is their work that the Turing Award rewards… I have been very fortunate to work with incredibly talented people over the years…

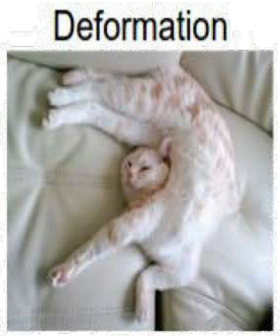
Mentors include Maurice Milgram & Françoise Soulié-Fogelman, my PhD advisors, Geoff Hinton with whom I did my postdoc, Larry Jackel and Rich Howard who hired me at Bell Labs, and Lawrence Rabiner my lab director at AT&T Labs…"

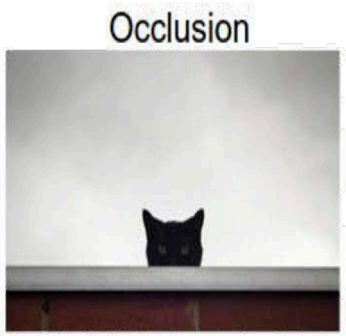
https://www.reddit.com/r/MachineLearning/comments/4lq701/yann_lecuns_letter_to_cvpr_chair_after_bad/ https://arxiv.org/pdf/1202.2160

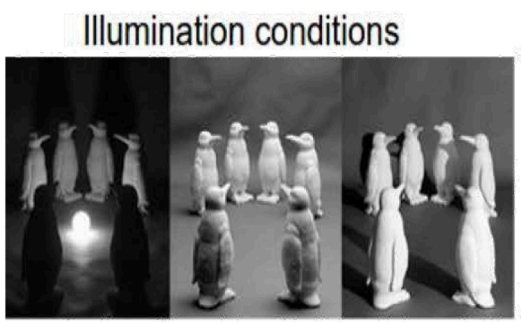
Deep Learning replaces Image Processing tasks using CNNs

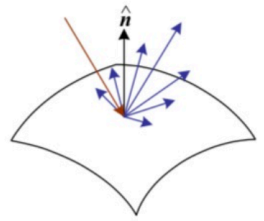
- Many tasks which were done by traditional image processing techniques are being replaced by Deep Learning which uses CNNs.
- Though these tasks face many challenges that are inherent in images.

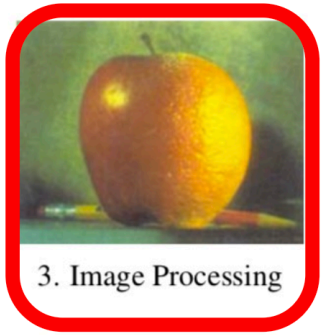








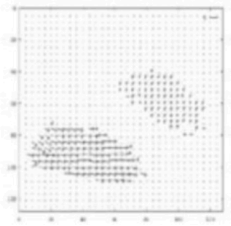

2. Image Formation

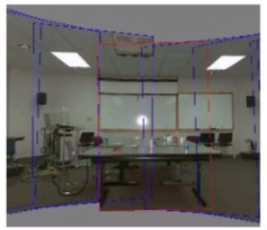

3. Image Processing

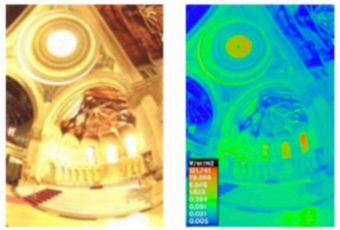

4. Features

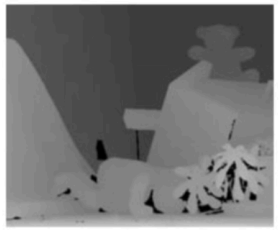

5. Segmentation


6-7. Structure from Motion

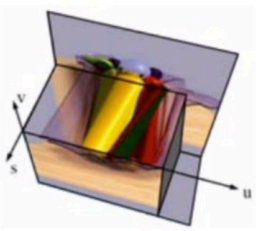

8. Motion


9. Stitching


10. Computational Photography


11. Stereo


12. 3D Shape


13. Image-based Rendering


14. Recognition

http://mit6874.github.io https://dellaert.github.io/21F-x476/

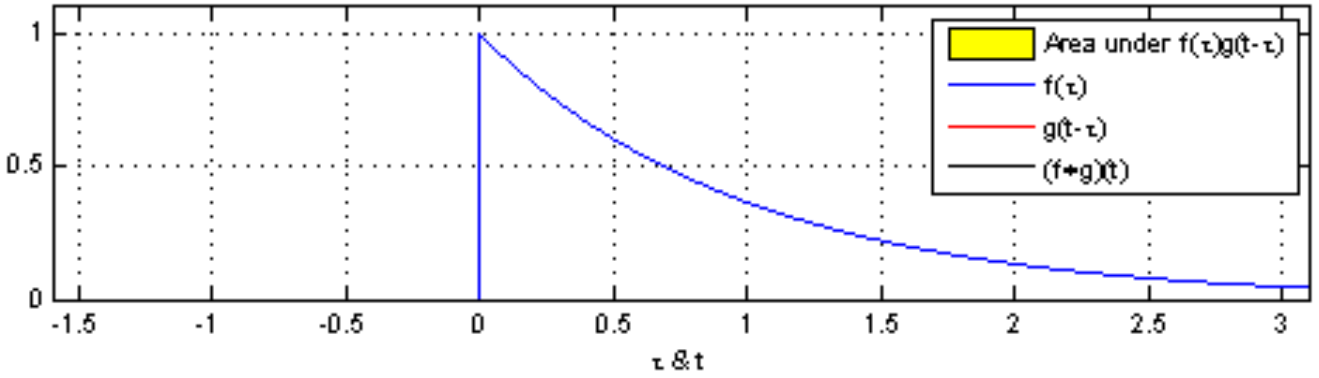
Technical aspects of CNNs

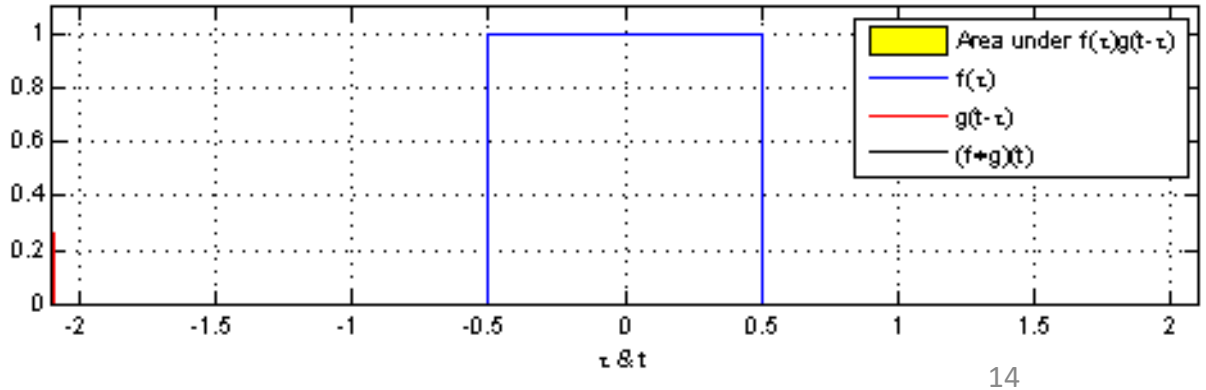
What is convolution?

- Convolution is the area under the curve $f(\tau)$ weighted by $g(t-\tau)$
- In continuous case, the convolution of two functions f and g can be written as

$$(f * g)(t) = \int_{-\infty}^{+\infty} f(\tau) g(t-\tau) d\tau$$

- Terminology:
  - The function f is referred to as the input
  - The function g is referred to as the kernel/filter
  - The output $(f * g)(t)$ is referred to as the feature map
- Two examples of convolution over different function f (the input) are given below:





Pushpak Bhattacharyya, DL-NLP slides, IITB 2023; Dripta Mj, DL-1 slides, RKMVERI 2020; https://en.wikipedia.org/wiki/Convolution

What is Convolutional Neural Networks (CNNs)?

- CNN: Neural Networks using convolution in place of general matrix multiplication in at least one layers, i.e., neural network for grid like topology; the filter component is learnt.
- The name *CNN* indicate that the mathematical operation convolution is used
- In ML, we work on discrete set of data points, for data sampled at regular intervals (say at integer values), the convolution can be defined as

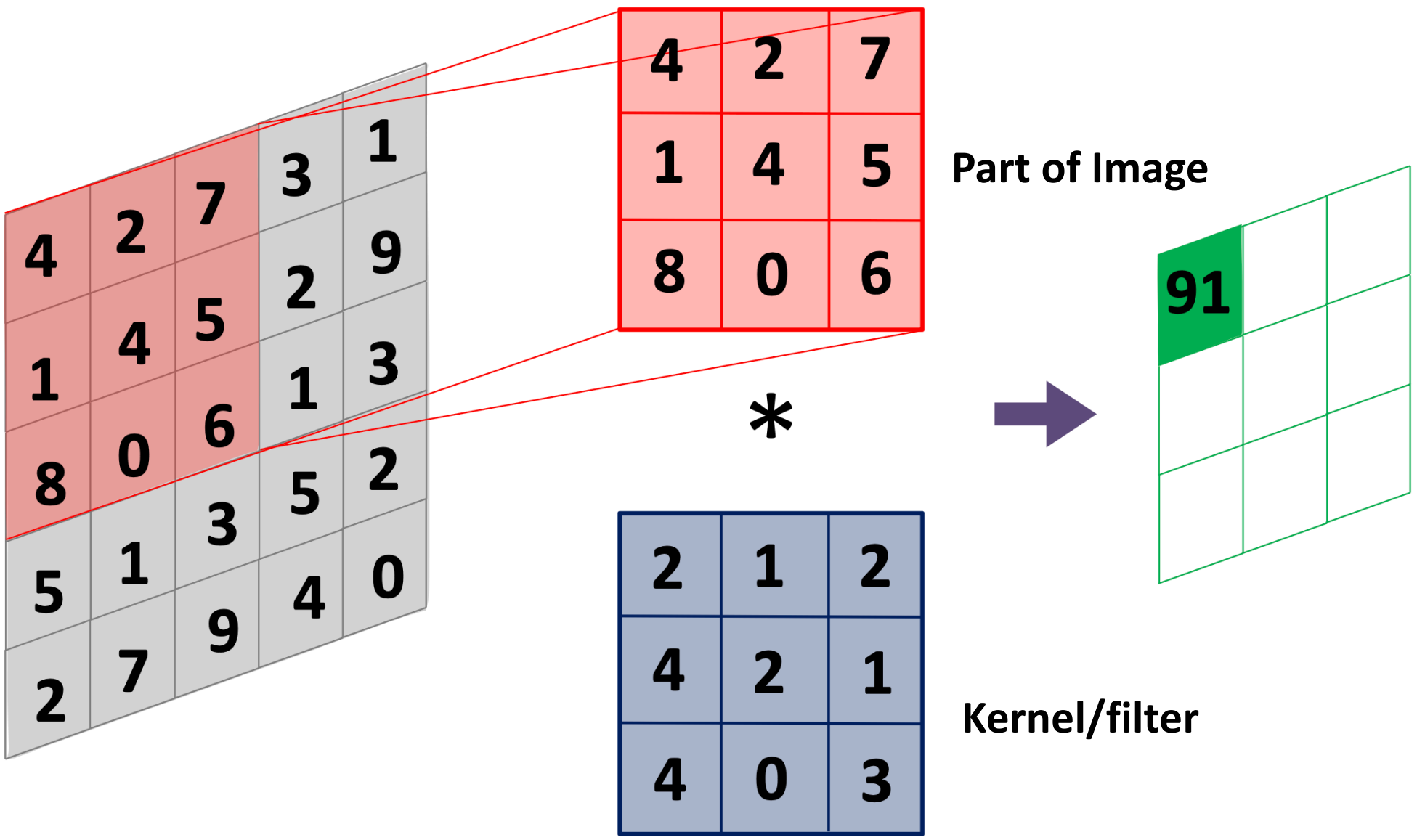
$$(f * g)[n] = \sum_{m=-\infty}^{+\infty} f(m)g(n-m)$$

- In many problems, the input dataset is usually a multidimensional array (tensor), and the kernel is a tensor of parameters that are learnt by the algorithm
- In practice, the inputs and the kernel are finite dimensional, hence their values are taken to be 0 outside a finite set of points; hence can be implemented as a finite summation.
- Convolution can be implemented in more than one dimension simultaneously, for example using a 2D kernel k, using an input image x

$$g(i,j) = (x * k)(i,j) = \sum_{\alpha} \sum_{\beta} \mathbf{x}(i,j)k(i-\alpha, j-\beta)$$

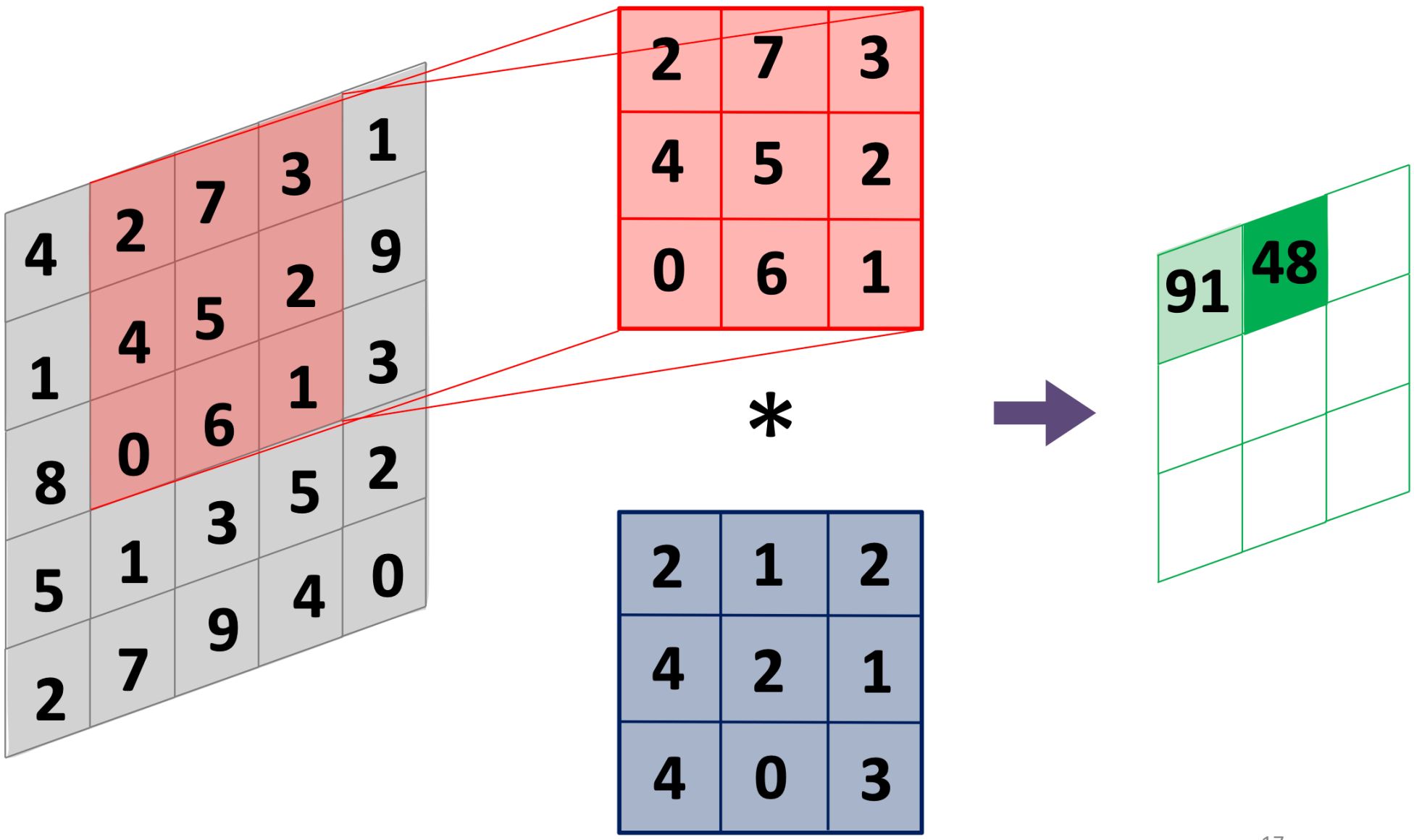
Example of Convolution using a kernel and image in 2D

- Element wise multiplication, just like dot-product of vectors.
- 4*2+2*1+7*2+ 1*4+4*2+5*1+ 8*4+0*0+6*3 = 91



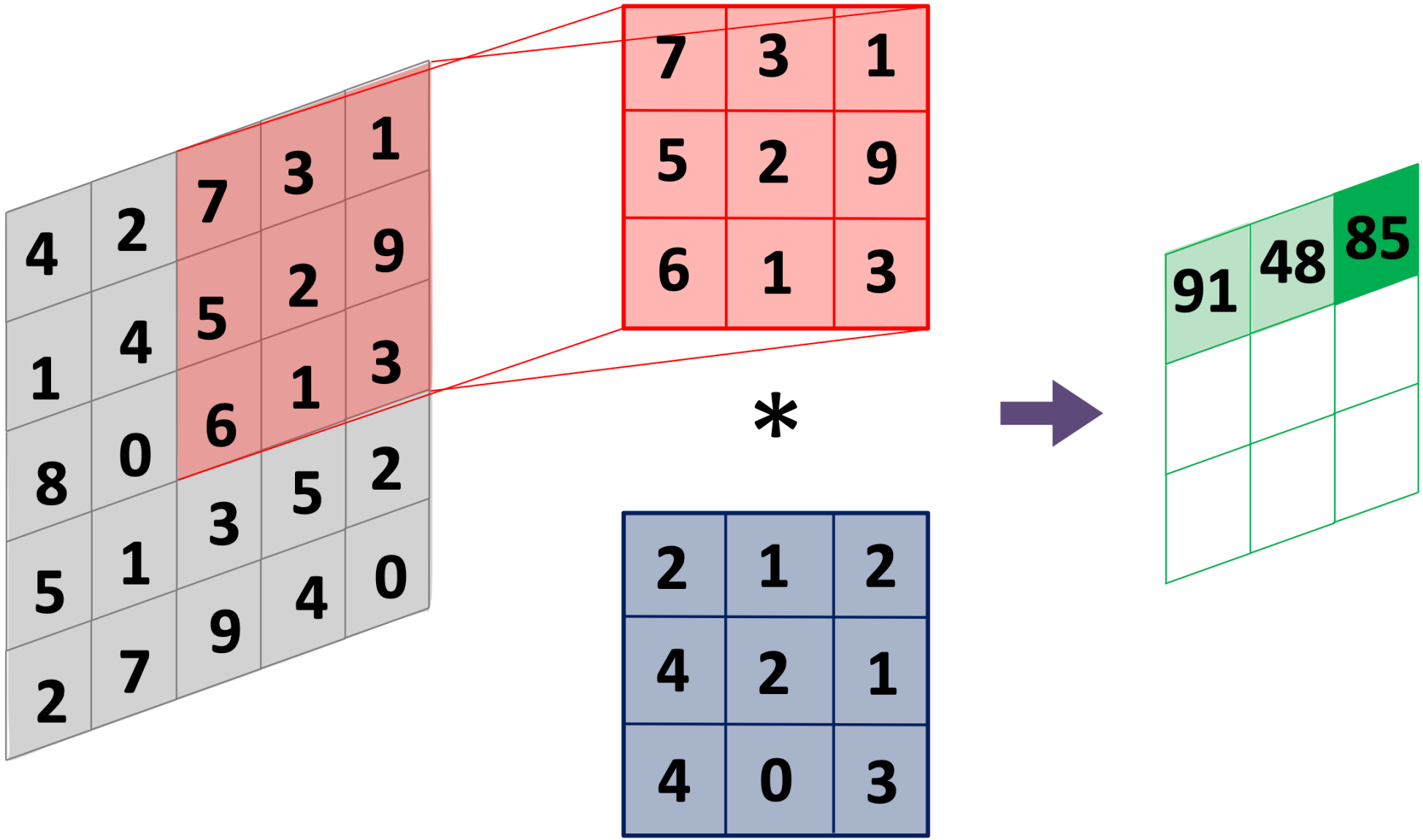
Example of Convolution using a kernel and image in 2D

- Element wise multiplication, just like dot-product of vectors.
- 2*2+7*1+3*2+ 4*4+5*2+2*1+ 0*4+6*0+1*3 = 48



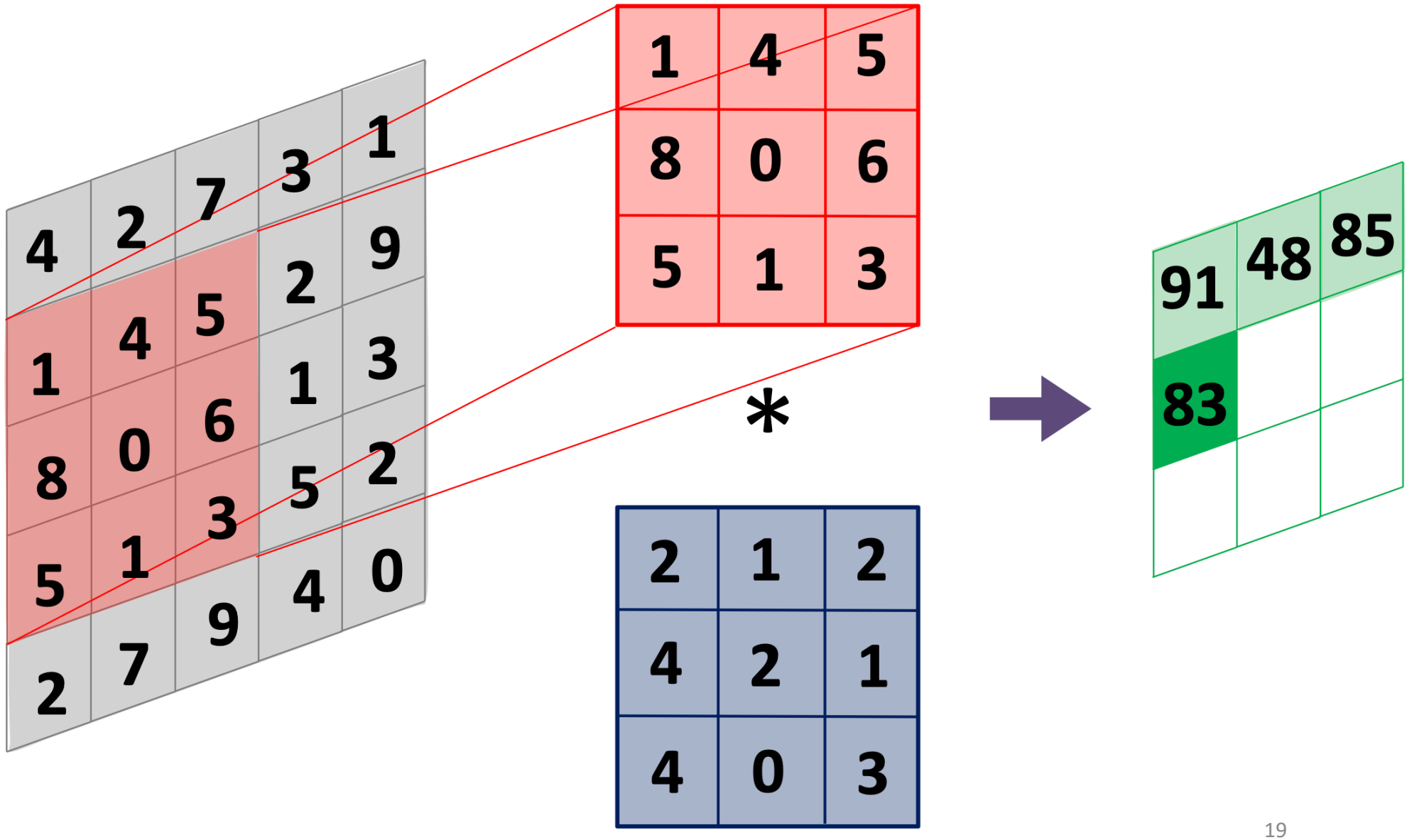
Example of Convolution using a kernel and image in 2D

- Element wise multiplication, just like dot-product of vectors.
- 7*2+3*1+1*2+ 5*4+2*2+9*1+ 6*4+1*0+3*3 = 85



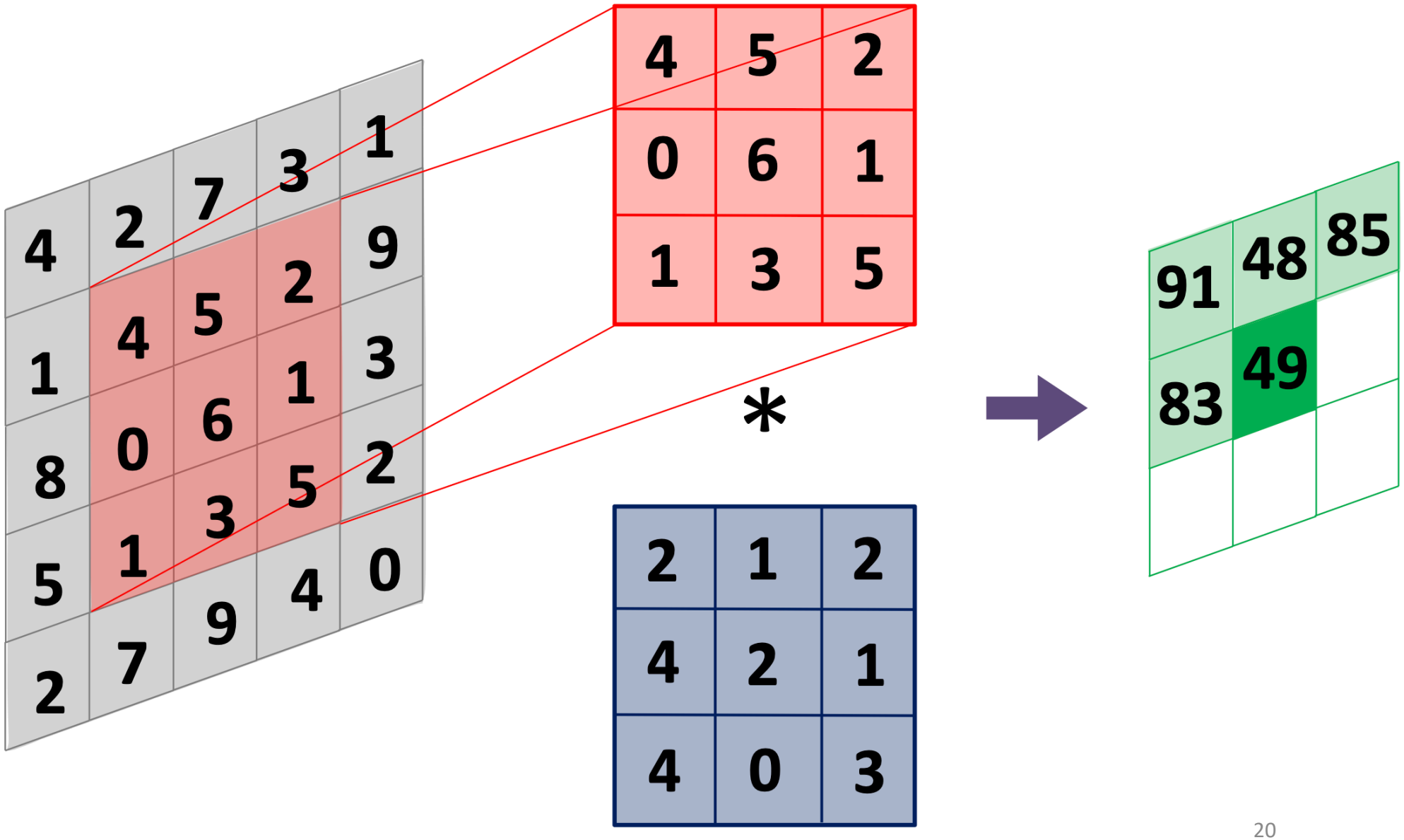
Example of Convolution using a kernel and image in 2D

- Element wise multiplication, just like dot-product of vectors.
- 1*2+4*1+5*2+ 8*4+0*2+6*1+ 5*4+1*0+3*3 = 83



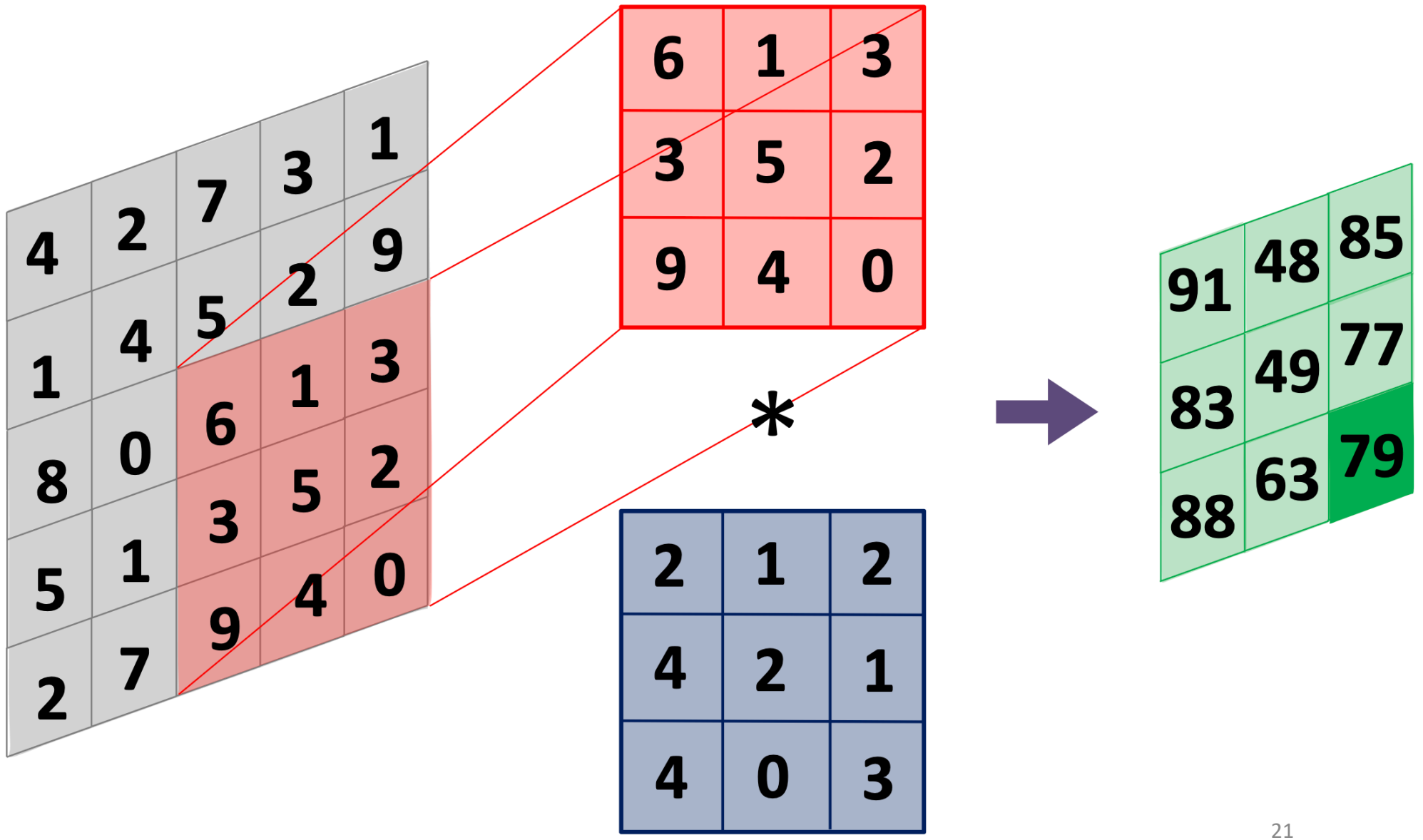
Example of Convolution using a kernel and image in 2D

- Element wise multiplication, just like dot-product of vectors.
- 4*2+5*1+2*2+ 0*4+6*2+1*1+ 1*4+3*0+5*3 = 49
- And so on…

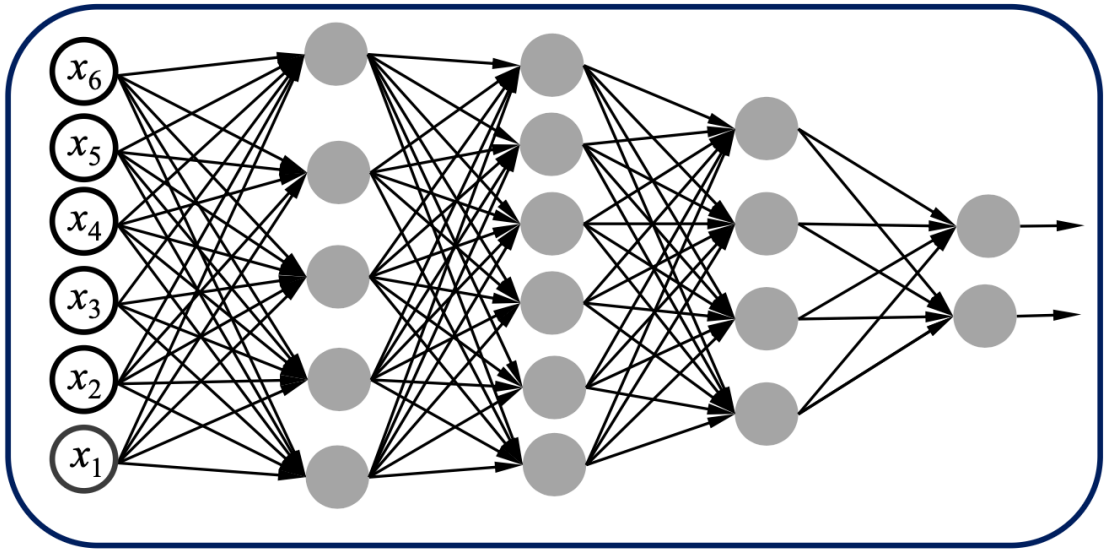


Example of Convolution using a kernel and image in 2D

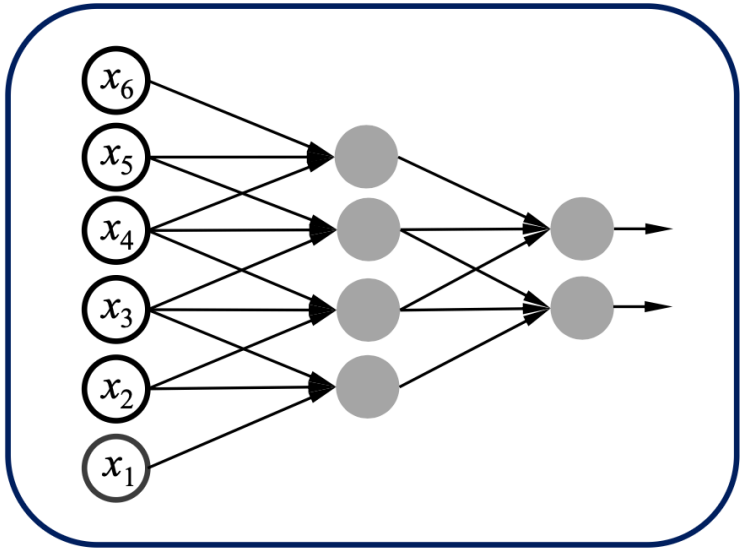
- Element wise multiplication, just like dot-product of vectors.
- 6*2+1*1+3*2+ 3*4+5*2+2*1+ 9*4+4*0+0*3 = 79



Key Aspects of CNNs



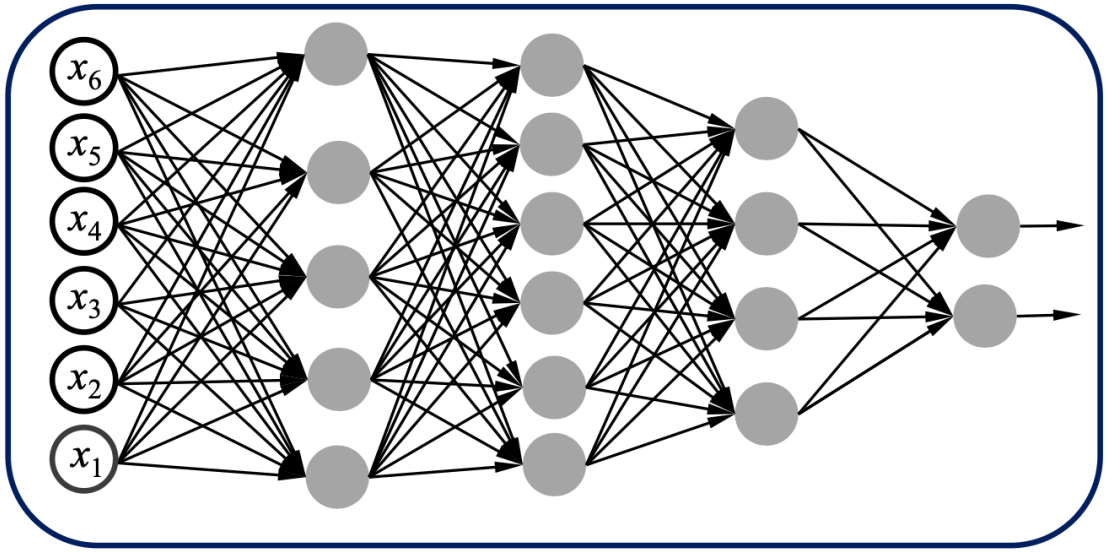
Traditional Neural Networks with Fully connected layers



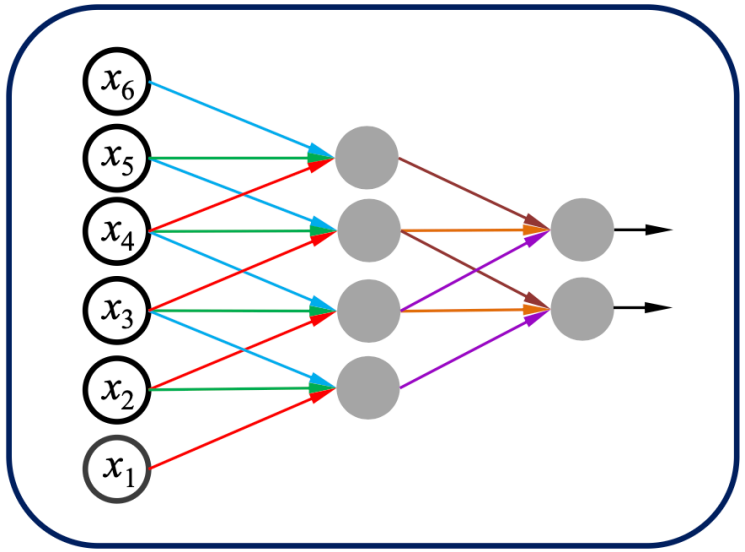
CNNs with kernels

- **Sparse connectivity**
  - Traditional Neural Network layers have a separate parameter for interaction b/w each i/p and o/p unit; In CNN, kernel preserves the 2D spatial structure of input images.
  - In CNN, kernels are used. Which have size smaller than that of the inputs, hence the number of parameters are much less – lesser chance of overfitting
  - Fewer operations are required to compute the outputs – less training time of models

Key Aspects of CNNs



Traditional Neural Networks with Fully connected layers

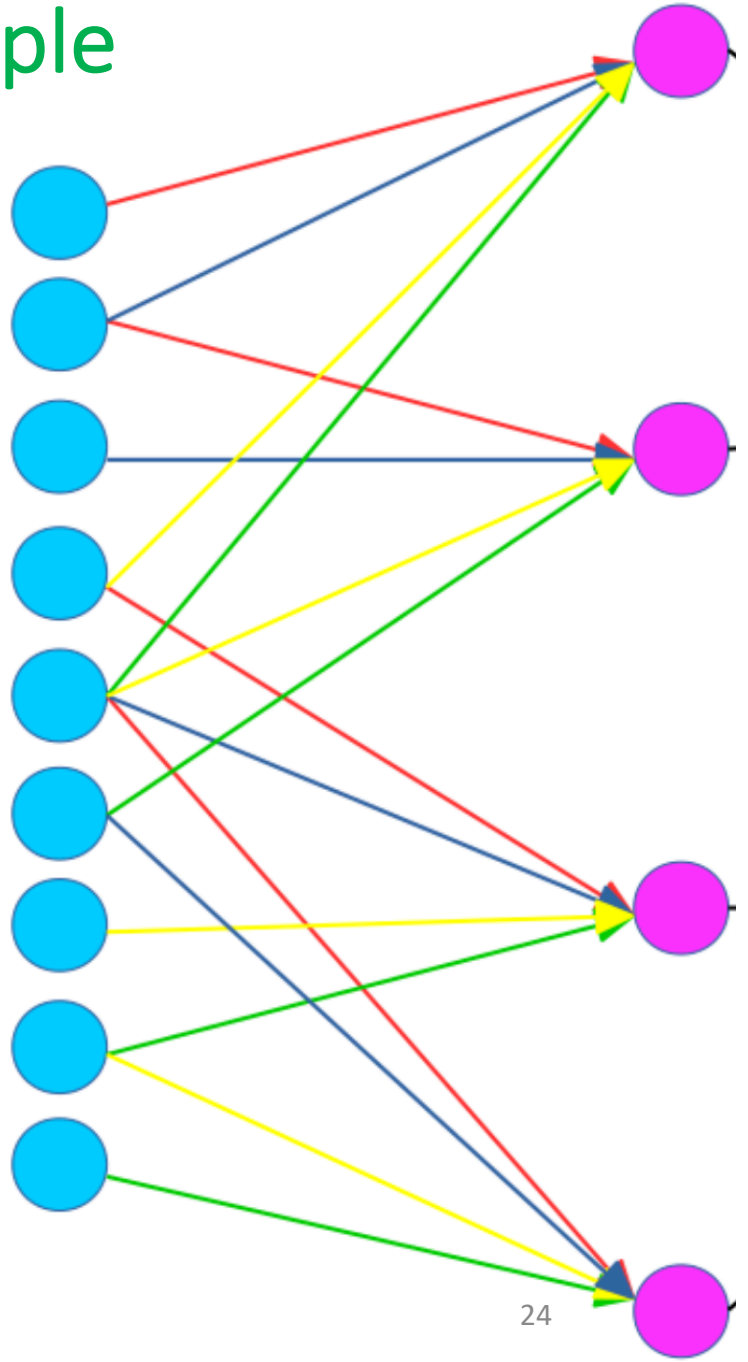


CNNs with kernels

- **Sharing of parameters:**
  - In traditional neural networks, a parameter is used only once while evaluating the o/p of a unit.
  - In CNN, the same set of parameters are used for all the inputs
  - Do not need to learn separate parameters for every input. Only need to learn the same set of parameters for all the inputs

Key Aspects of CNNs – parameters count example

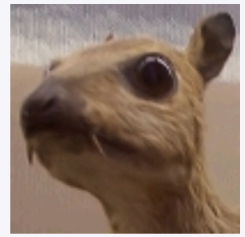
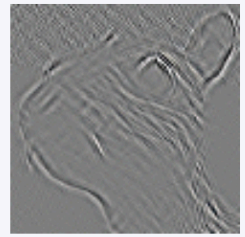
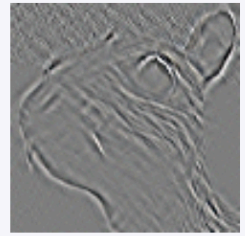
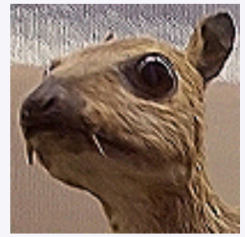
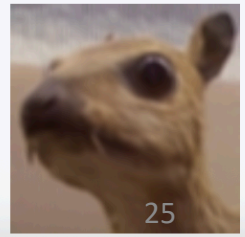
- Let's consider an 1-D example, we have 4 weights, (i.e., red, blue, yellow and green values in the filter)
- If instead we use a fully-connected layer, we would use 9x4 = 36 (9 blue for first layer, 4 magenta for second layer) connections.
- The bias of a particular layer is the number of neurons in that layer, we don't consider input as a layer.
- Hence the weights to be learned for this example is 9x4+4 = 36 weights+4 bias = 40 parameters
- If instead we use kernel/filter using CNN, we have 4 weights+1 bias = 5 parameters, hence the number of parameters is significantly reduced.
- In images, input sizes are high, i.e., (256x256x3 = 196608 pixels, which can lead to million of parameters in a couple of layers using very few fully-connected neurons)



https://www.jefkine.com/general/2016/09/05/backpropagation-in-convolutional-neural-networks/

Key Aspects of CNNs

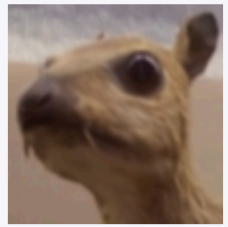
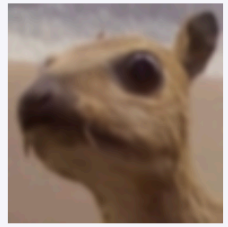
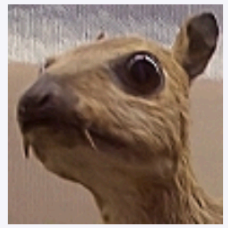
- **Equivariance to translation** – if the location of a certain feature is changed, then the output of the convolution also changes accordingly
- Feature - pattern, characteristic, or attribute extracted from the input data, such as edges, textures, corners, or shapes which helps in pattern recognition.
- Same features occur at multiple locations in the input space
- The o/p of the convolution indicate whether different features occur in the input space

Operation	Kernel ω	Image result g(x,y)
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Ridge or edge detection	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	

 https://en.wikipedia.org/wiki/Kernel_(image_processing)

Key Aspects of CNNs

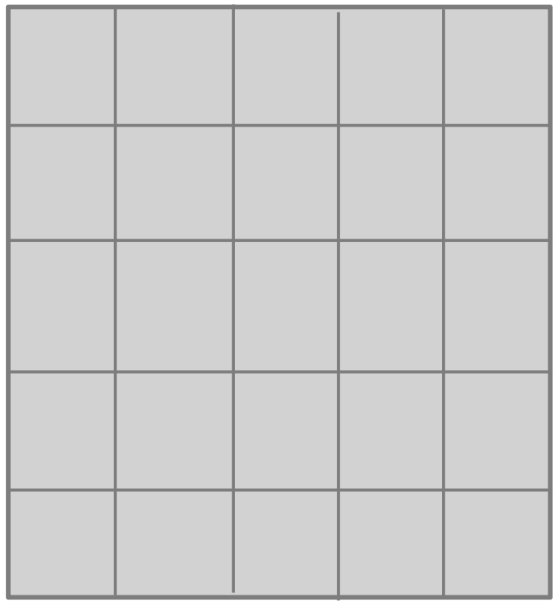
- For example, an edge detecting filter will generate a 2D map of the occurrence of such an edge in the i/p
- Convolution is not equivariant to transformations such as scaling and rotations.
- The higher the layer (i.e., closer the layer to the output layer), the more complex the feature becomes
- Initial layers detects edges, color etc. from raw pixels
- These features are used in the higher layers to detect more complex shapes
- Again these features are then used to detect more higher level features belonging to a class etc.

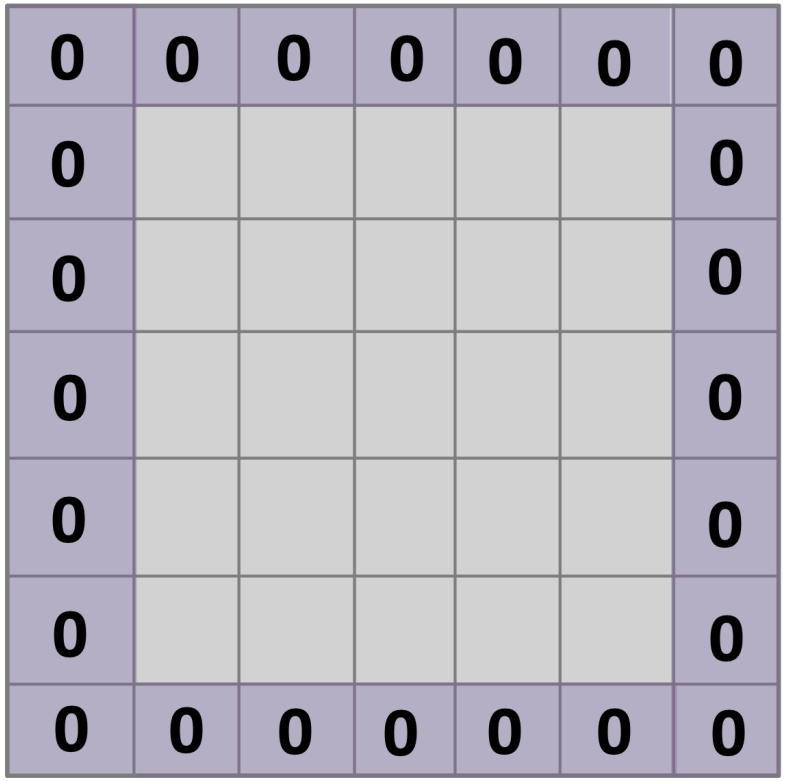
Operation	Kernel ω	Image result g(x,y)
Gaussian blur 3 × 3 (approximation)	$\frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix}$	
Gaussian blur 5 × 5 (approximation)	$\frac{1}{256}\begin{bmatrix}1 & 4 & 6 & 4 & 1\\ 4 & 16 & 24 & 16 & 4\\ 6 & 24 & 36 & 24 & 6\\ 4 & 16 & 24 & 16 & 4\\ 1 & 4 & 6 & 4 & 1\end{bmatrix}$	
Unsharp masking 5 × 5 Based on Gaussian blur with amount as 1 and threshold as 0 (with no image mask)	$\frac{-1}{256}\begin{bmatrix}1 & 4 & 6 & 4 & 1\\ 4 & 16 & 24 & 16 & 4\\ 6 & 24 & -476 & 24 & 6\\ 4 & 16 & 24 & 16 & 4\\ 1 & 4 & 6 & 4 & 1\end{bmatrix}$	

https://en.wikipedia.org/wiki/Kernel_(image_processing)

Padding

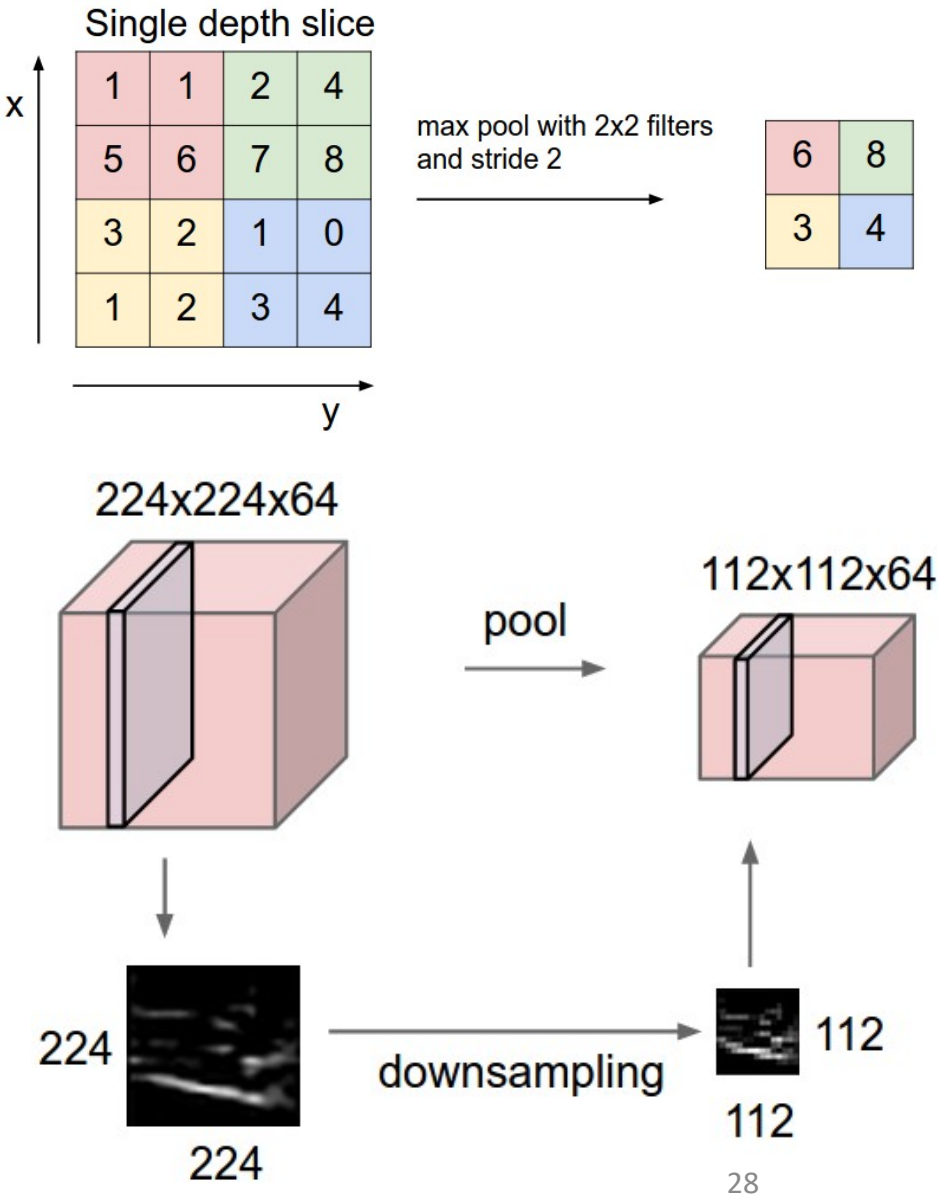
- Why do we need Padding?
- Risk of losing information from the edges of i/p with no padding
- Without padding in DNNs, the inputs to the later layers will be significantly reduced in size
- Used to preserve image/feature sizes
- Can also be used to make the feature volume of the next layer of a desired size/dimension





Pooling

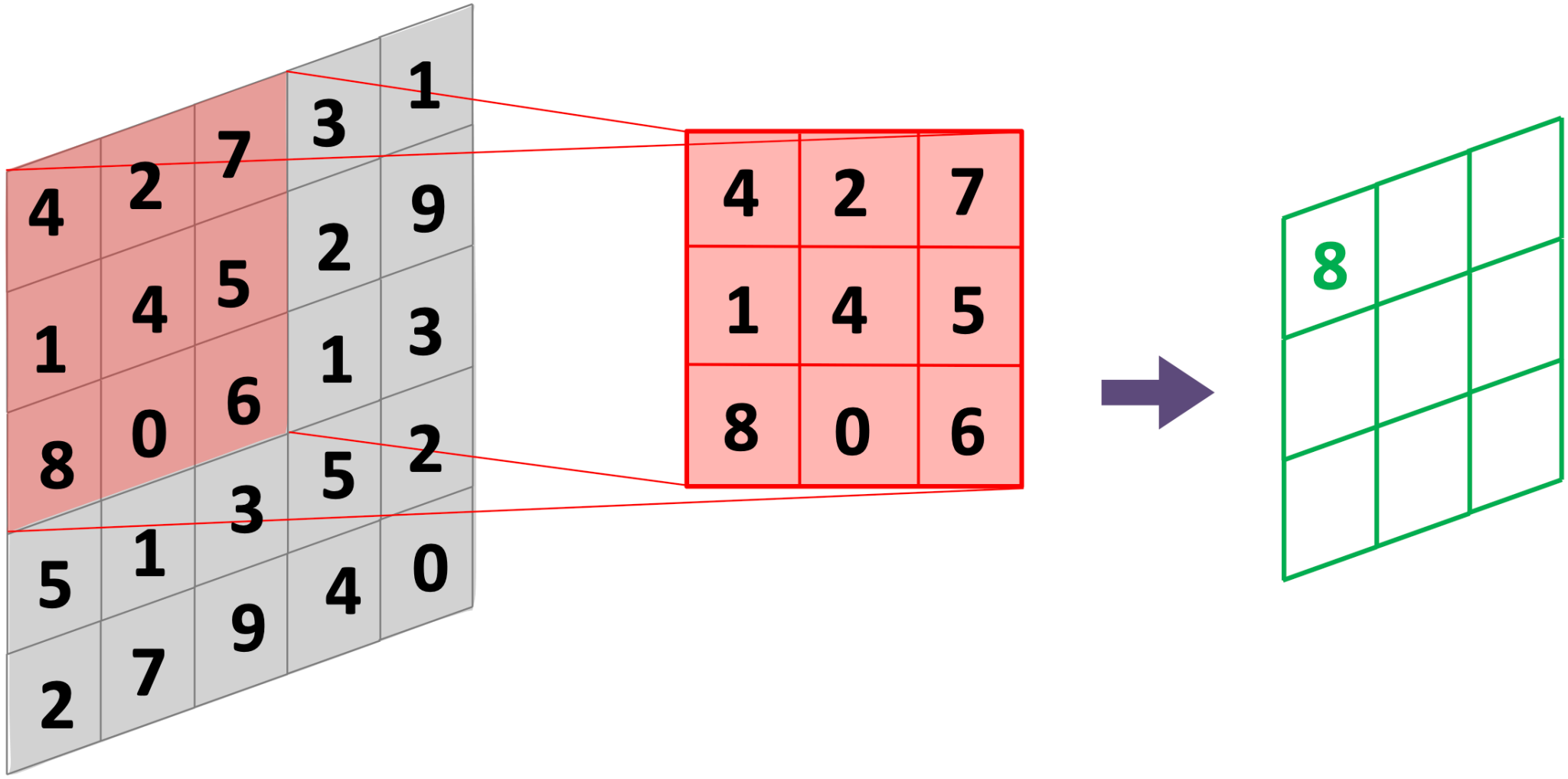
- **Replaces the output with a summary statistic of the nearby outputs** – reduces computational/memory requirements, requires no additional parameters to be learnt
- Motivation: pooling assists in making a representation approximately invariant to small translations of the input
- Pooling is useful as in many cases we are concerned about the presence of some feature rather than their exact location
- Example:
  - Max pooling: Computes the maximum output within a rectangular neighbourhood
  - Average pooling: Average of a rectangular neighbourhood
  - L^2 norm of a rectangular neighbourhood



Dripta Mj, ML-1 slides RKMVERI 2020; https://cs231n.github.io/convolutional-networks/#pool

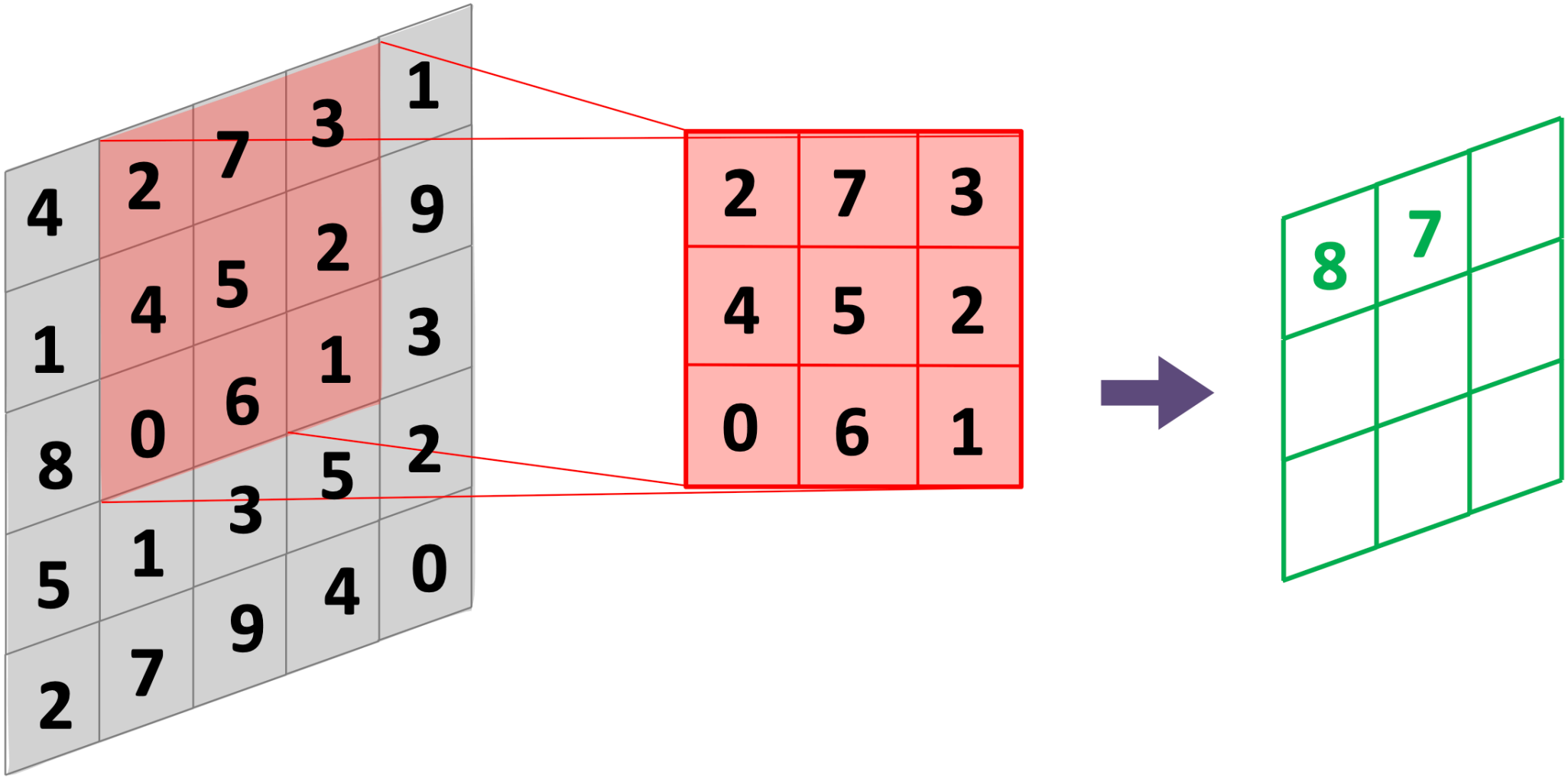
Max Pooling

- Computing the maximum o/p in a 3x3 neighbour hood



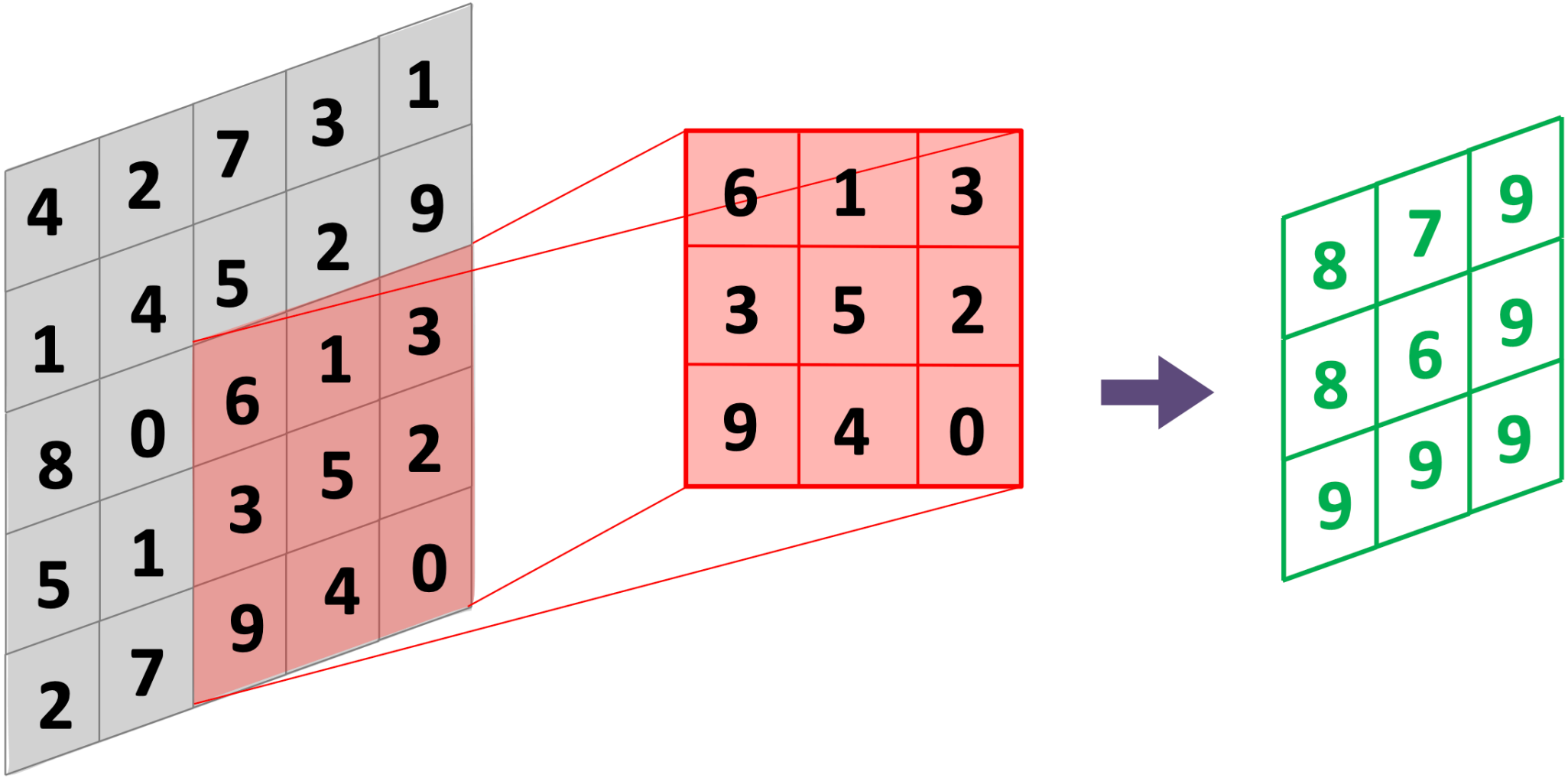
Max Pooling

- Computing the maximum o/p in a 3x3 neighbour hood



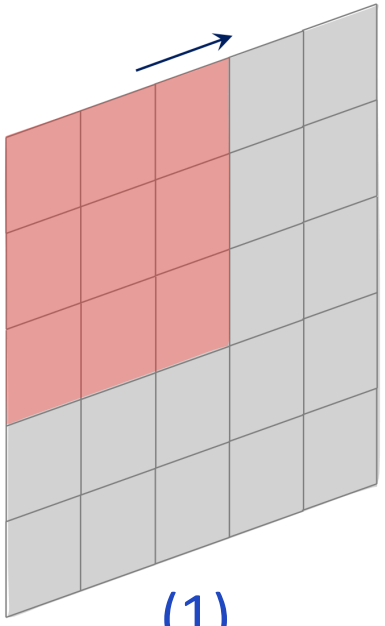
Max Pooling

- Computing the maximum o/p in a 3x3 neighbour hood

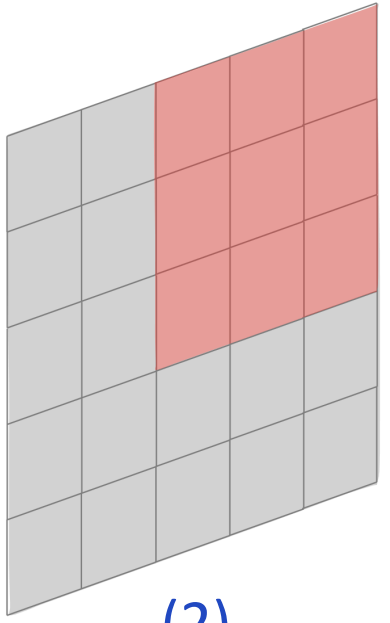


Stride

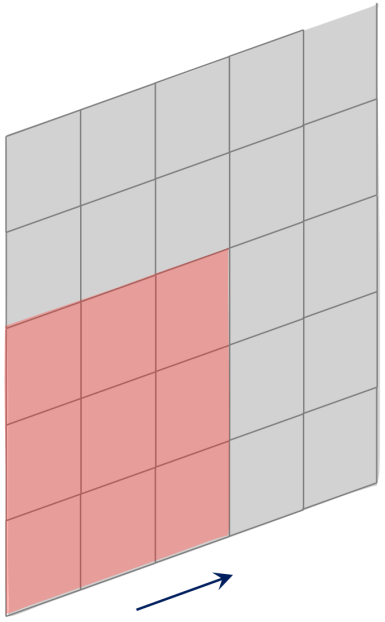
- Some positions of the kernel can be skipped to reduce the computational cost.
- Samples are taken every (say) s grid points in a particular direction
- Here s = 2 is used
- s is referred to as the stride of the downsampled convolution
- The more the stride, the lower the dimension of the feature in the next block; i.e., more compression of features in the feature space.
- It is possible to define a separate stride s for each direction of motion



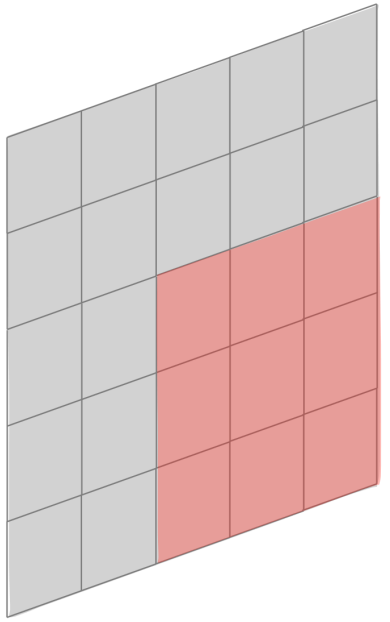
(1)



(2)



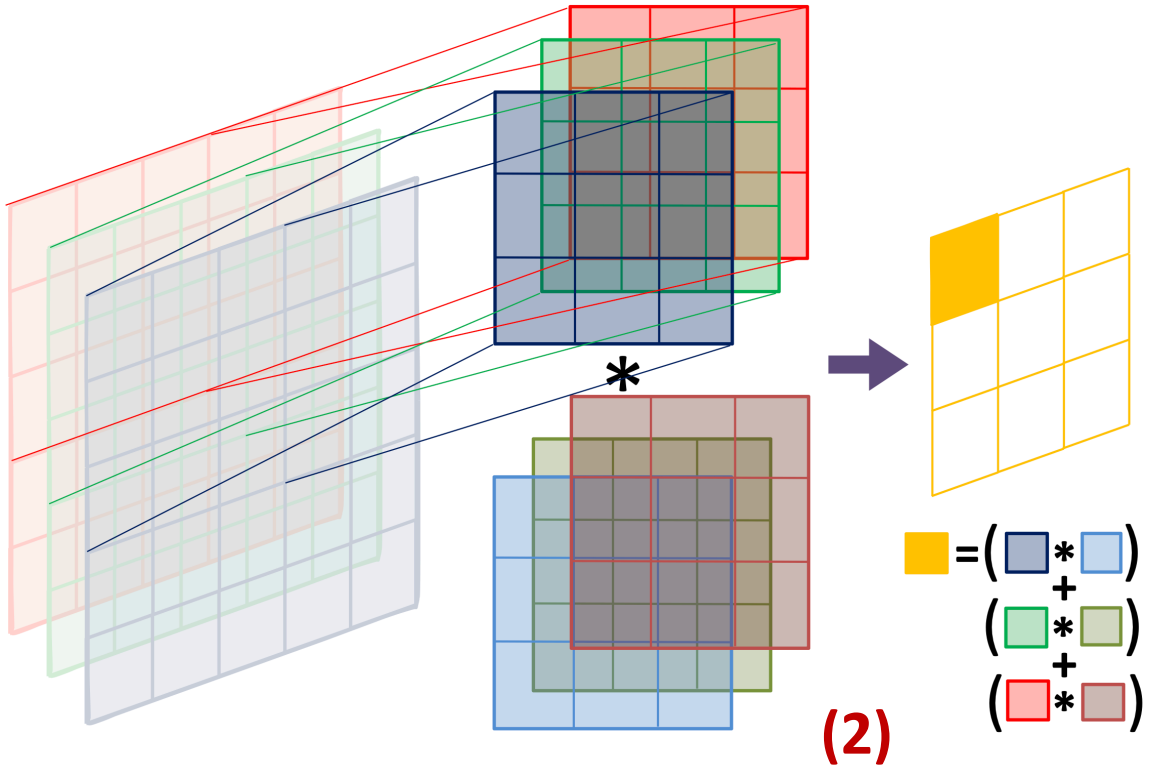
(3)

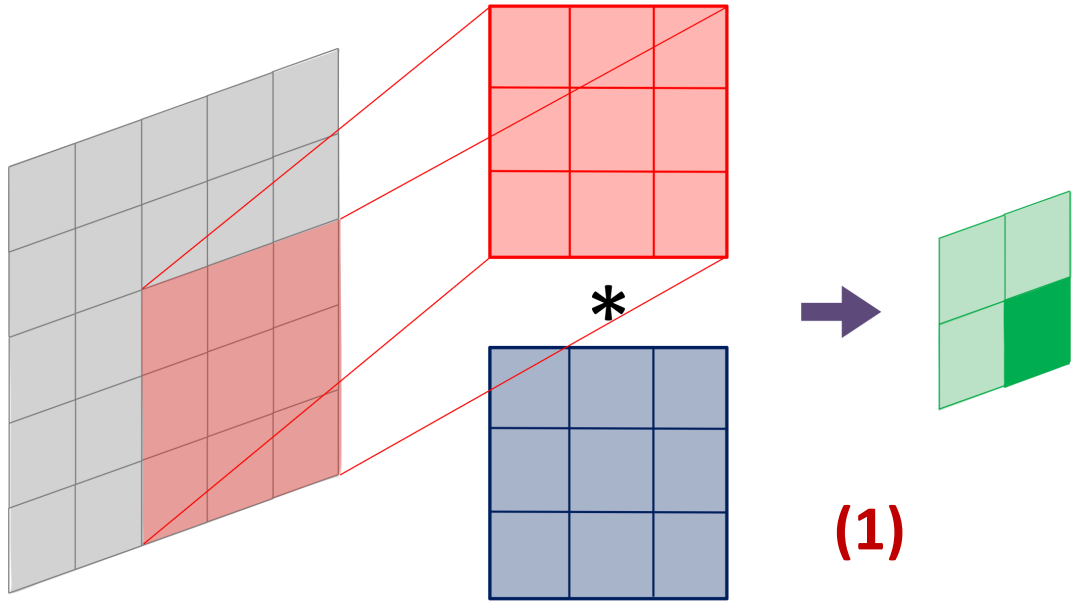


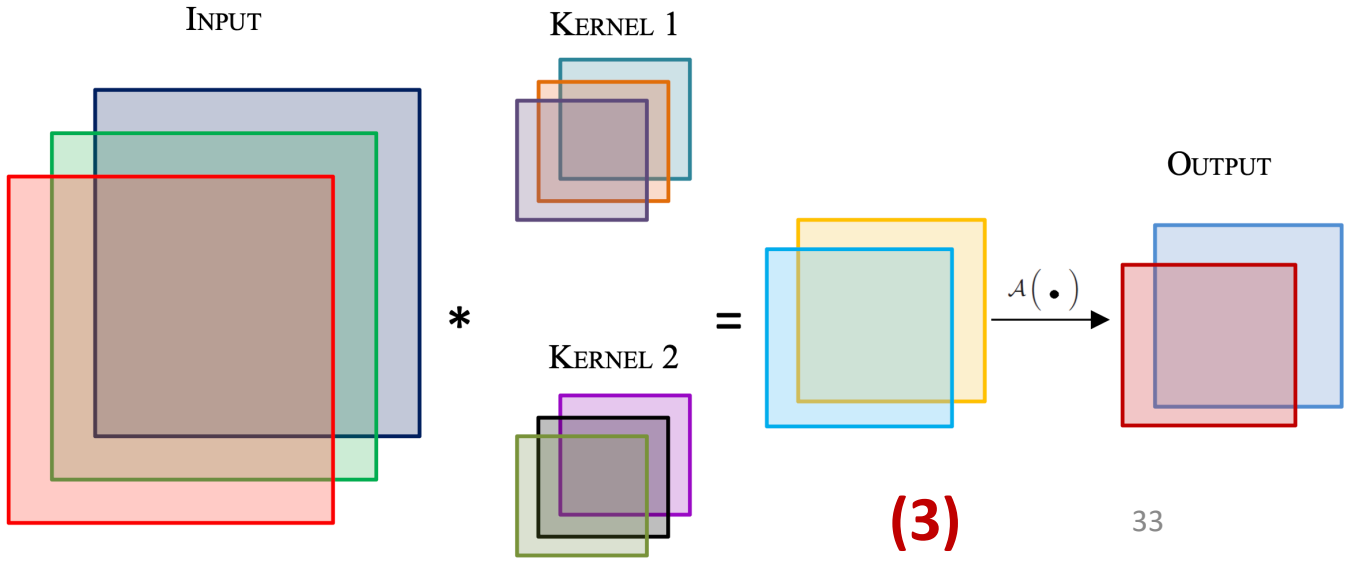
(4)

Stride

- (1) – we can use strides in Convolution operations to reduce the size of feature map.
- (2) – we can have different dimensions for kernels for generating different sizes of feature maps.
- (3) – we can have activations $\mathcal{A}(\cdot)$ like $ReLU(\cdot)$ /$Sigmoid(\cdot)$ etc., to generate output from feature maps

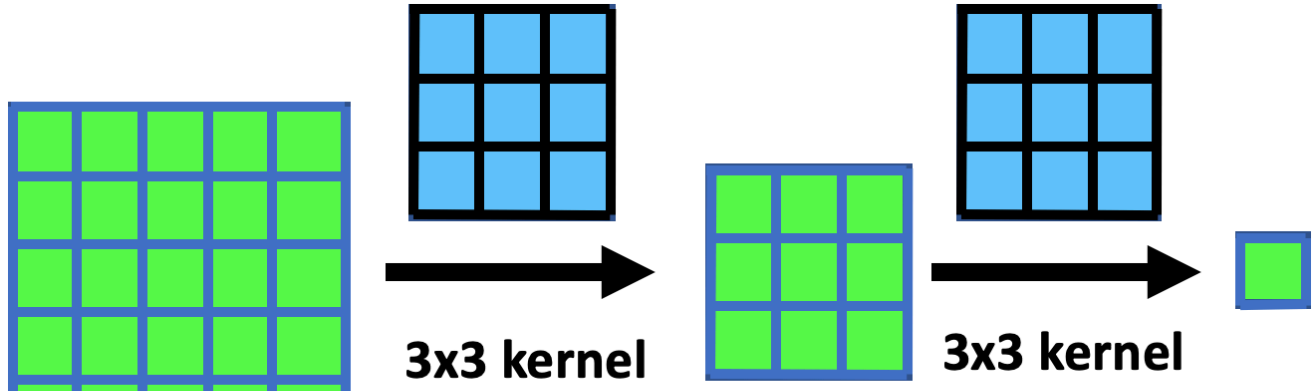


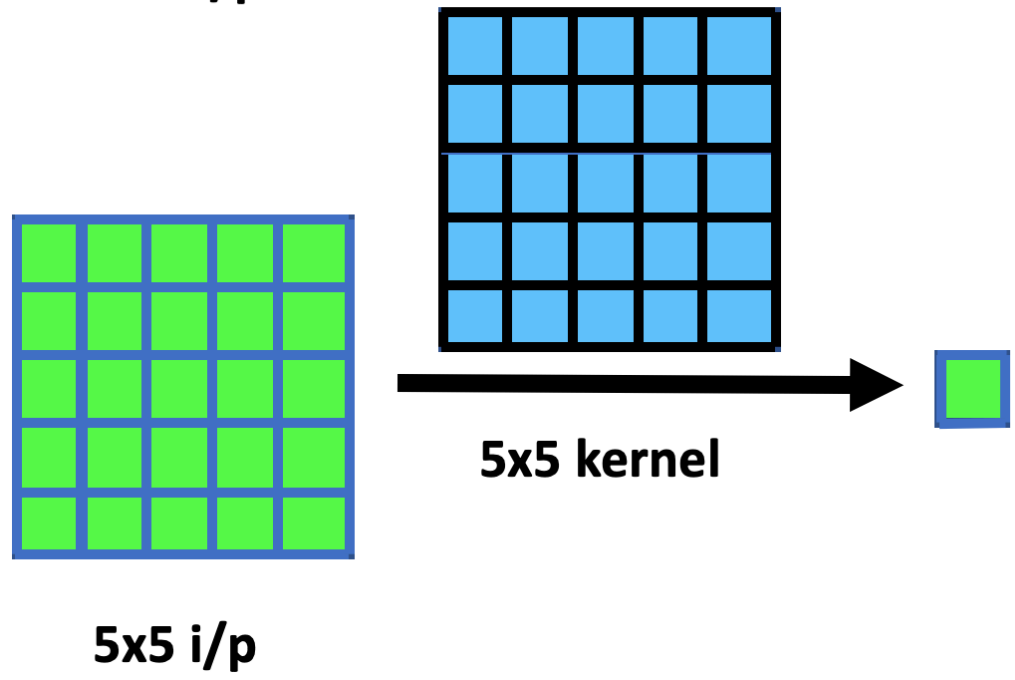




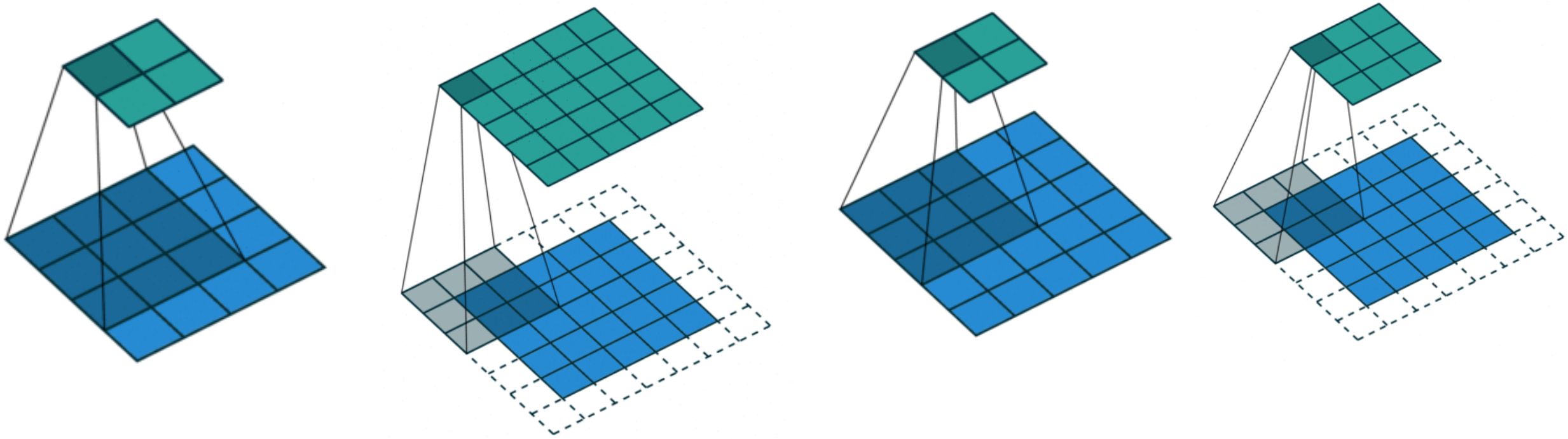
Receptive field of a kernel/filter

- Say, we have an input of 5x5 pixel, which is applied to a kernel of size 3x3, the o/p will be of size 3x3.
- Again, if we apply the same kernel on the result, the output will be of size 1x1.
- The total number of parameters = (3x3+1)*2 = 20
- Similarly, if an input of size 5x5 pixel is convolved with a kernel of size 5x5, we get a output of size 1x1 directly.
- The total number of parameters = (5*5+1) = 26
- Hence, we get the same receptive field by applying two 3x3 kernels in series as applying one 5x5 kernel, which leads to more parameters





Different types of convolution

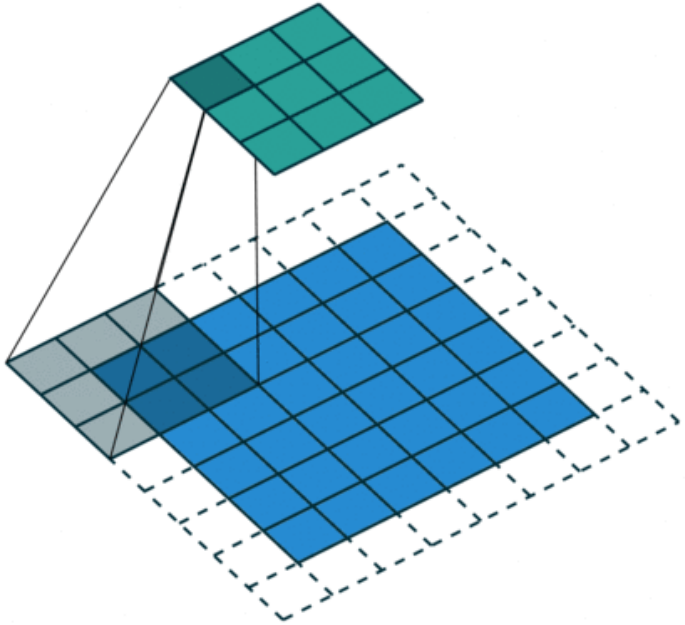


Padding=0, Stride=1 Padding=1, Stride=1 Padding=0, Stride=2 Padding=1, Stride=2

- Here, a kernel size of 3x3 is used to demonstrate the different types of convolution operations
- Blue maps are inputs and cyan maps are outputs

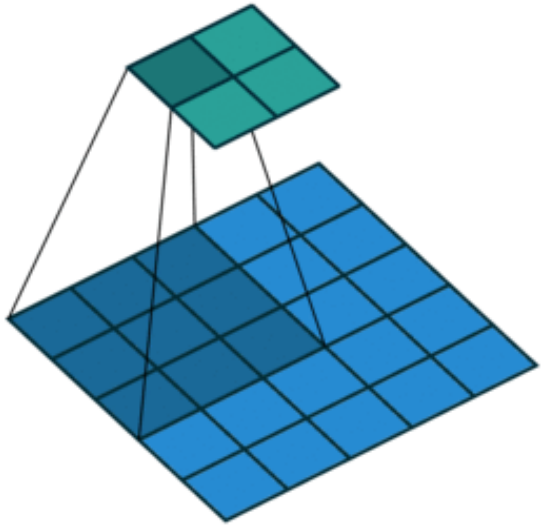
https://github.com/vdumoulin/conv_arithmetic
https://arxiv.org/pdf/1603.07285

Different types of convolution



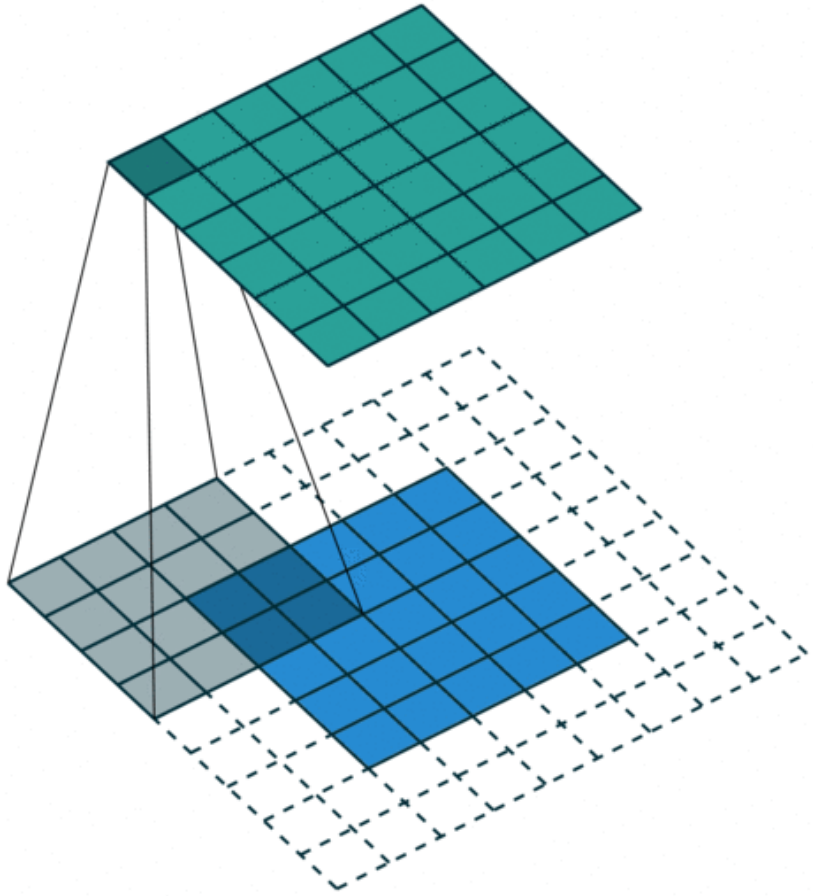
Padding=1, Stride=2,

Kernel size=3x3



Padding=0, Stride=2,

Kernel size=3x3



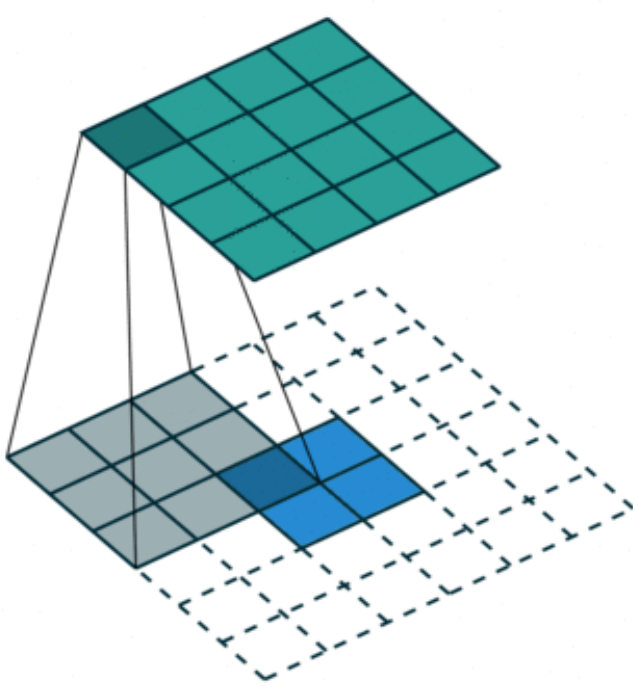
Padding=2, Stride=1,

Kernel size = 4x4

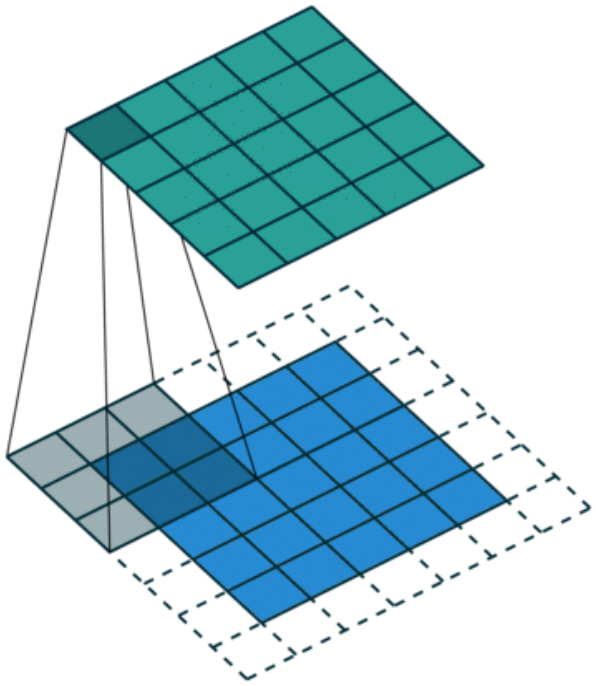
- Blue maps are inputs and cyan maps are outputs

https://github.com/vdumoulin/conv_arithmetic

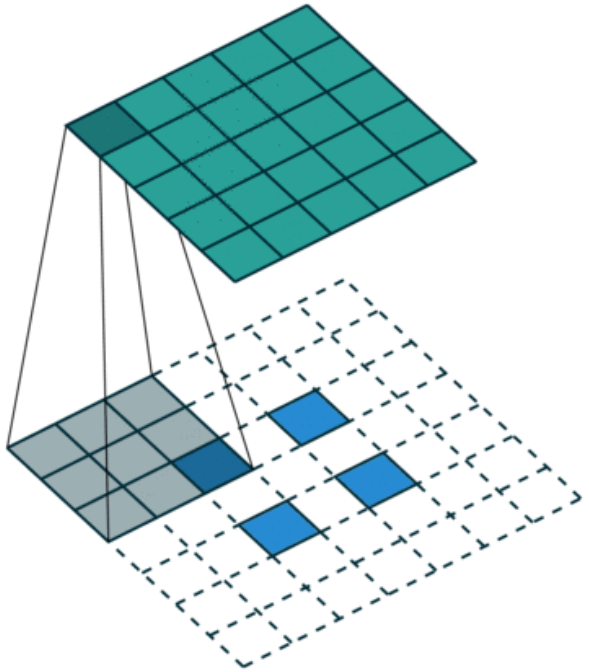
Different types of deconvolution



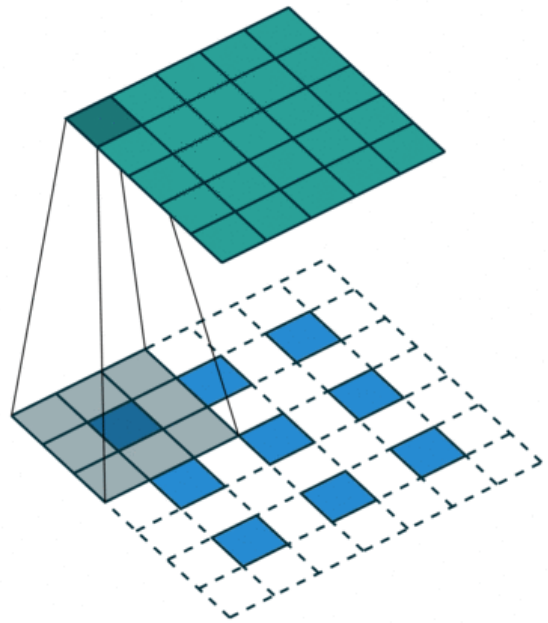
Padding=0, Stride=2



Padding=1, Stride=1



Padding=0, Stride=2



Padding=1, Stride=2

- Deconvolution is also called as transposed convolution
- An operation in CNN that increases the spatial dimensions (height and width) of its input feature maps effectively upsampling the data

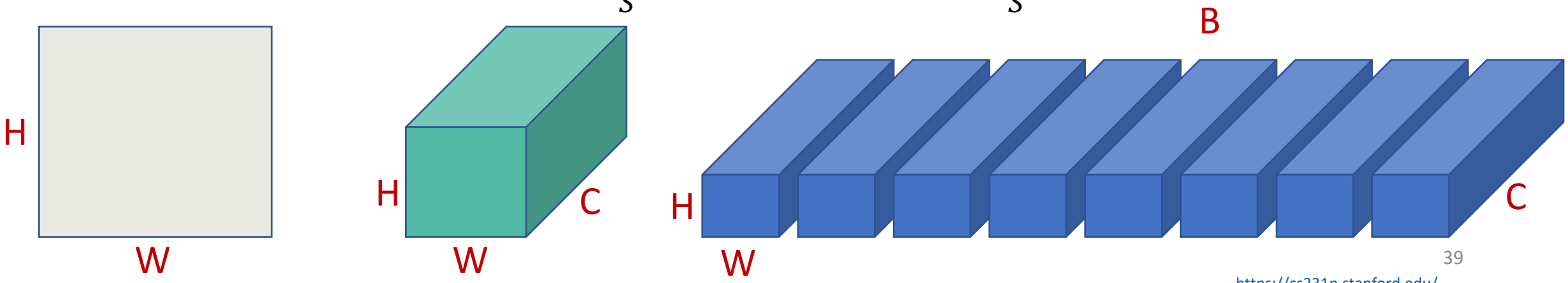
https://github.com/vdumoulin/conv_arithmetic

Parameters and feature dimension calculations in a CNN

Examples of convolutions and parameter calculations

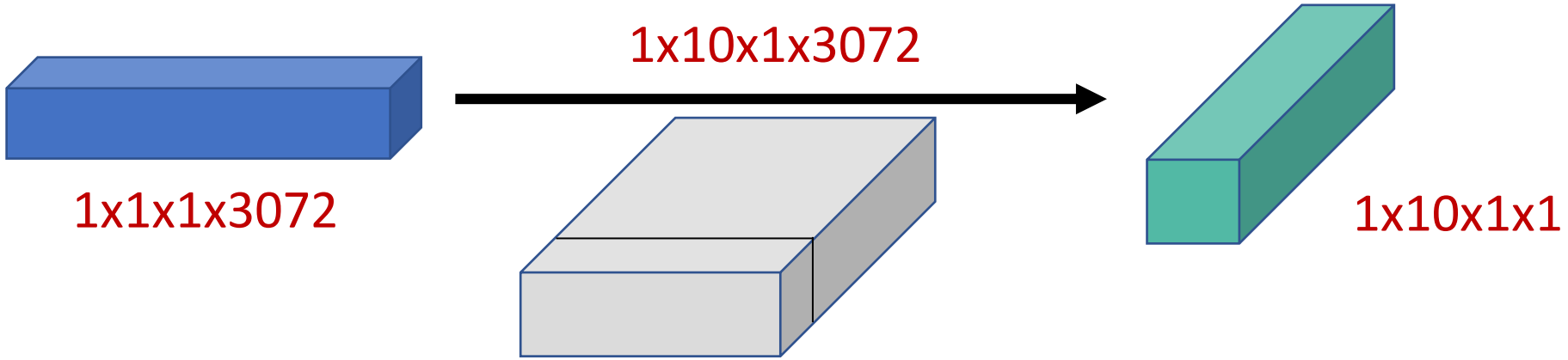
- Here, we will consider the dimensions of feature maps as input and calculate the number of parameters for a particular conv layer
- Let us consider that the features are present in a tensor represented by BxCxHXW where B=batch size; C=number of channels (channel first order); H=height of the tensor; W=width of the tensor
- Formula to calculate the output of the feature map, given input $W_1 \times H_1 \times C$; the Conv layer need 4 hyper-parameters (i) number of filters = K; (ii) the filter size = F, (iii) the stride = S (iv) the zero padding P
- This produces an o/p of $W_2 \times H_2 \times K$ (though I don't recommend memorizing the formula)

$$W_2 = \frac{W_1 - F + 2P}{S} + 1 \text{ and } H_2 = \frac{H_1 - F + 2P}{S} + 1$$



https://cs231n.stanford.edu/

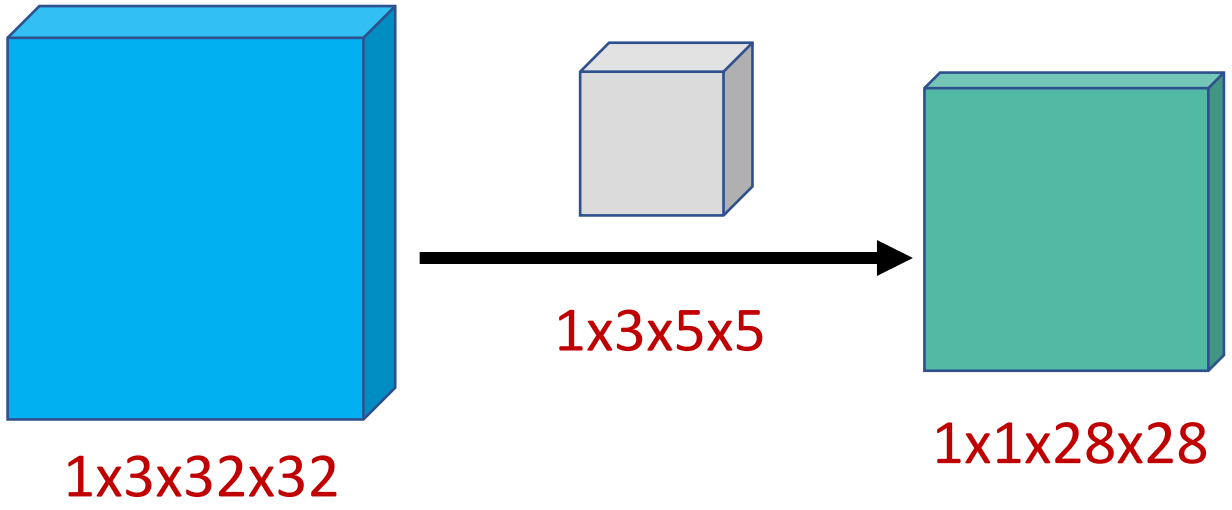
Examples of convolutions and parameter calculations

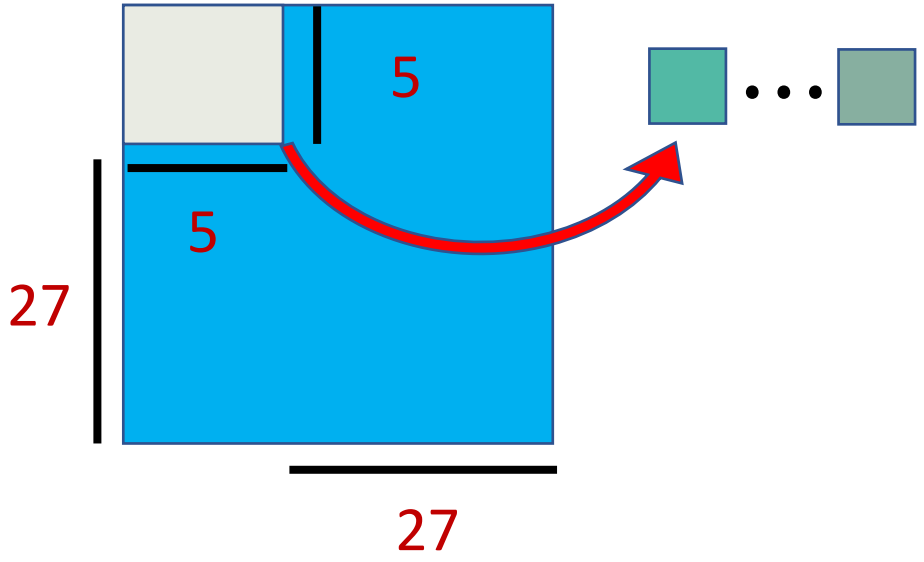


- This is a bit non-standard example, where the kernel is of bigger size than the input feature volume; taken from Stanford lectures.
- Let's consider an input of size 1x1x1x3072 which is applied to a kernel of size 1x10x1x3072
- Here the feature dimensions are measured in BxCxHxW
- Feature of size 1x1x1x3072 is multiplied by each of the channel of 1x10x1x3072 to get the output dimension of 1x10x1x1
- Since there are 10 output channel, bias = 10 (each channel of the output contributes to a bias of 1)
- The number of parameters of the kernel/conv operation = 1x10x1x3072+10 = 30730

https://cs231n.stanford.edu/

Examples of convolutions and parameter calculations

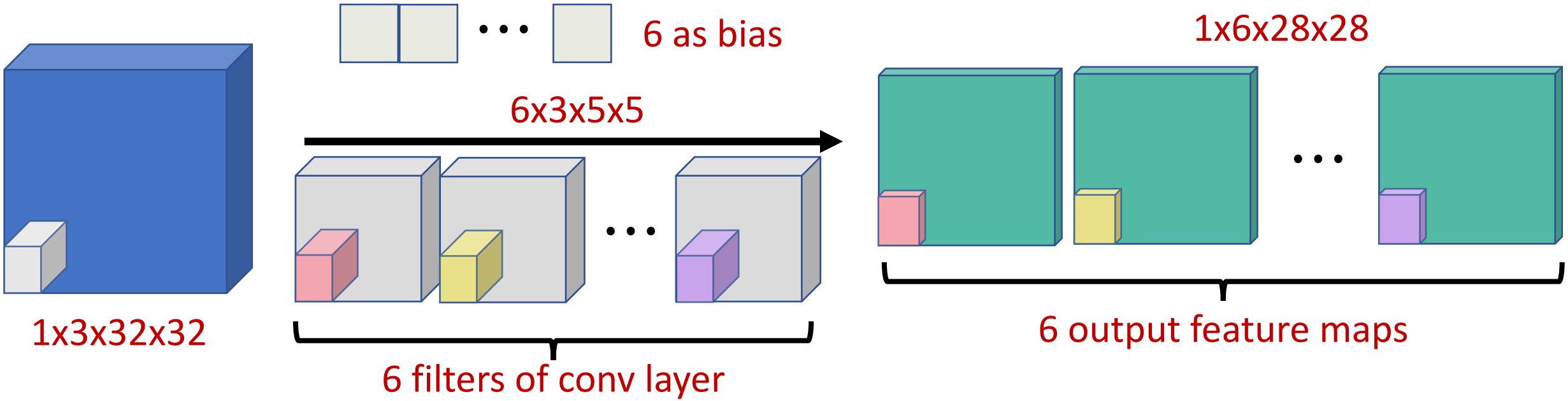




- Here, stride = 1, H=32; W=32; kernel size = 3x5x5
- Hence each of the 3x5x5 block of kernel convolves with the 3x5x5 block of the input volume to produce output feature
- After the convolution is done, the resultant is of size 1x1x28x28
- The convolution kernel has a bias of 1; also the output channel is 1, for which a bias of 1 is contributed to the whole operation
- The number of parameters of the kernel = 3x5x5+1 = 76

https://cs231n.stanford.edu/

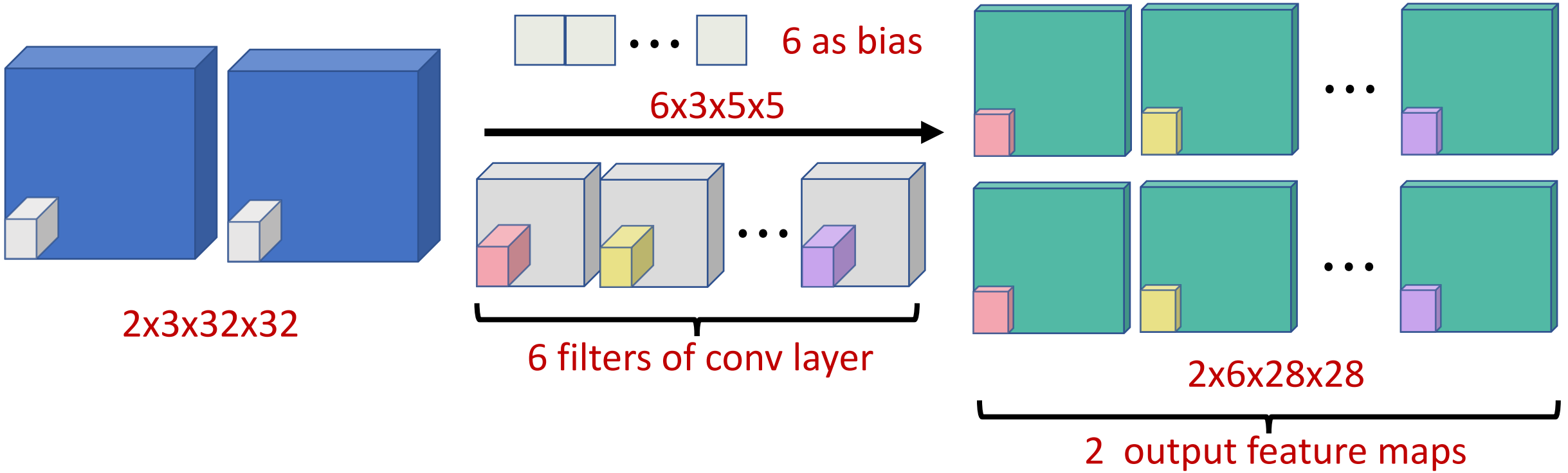
Examples of convolutions and parameter calculations



- Here, stride = 1, H=32; W=32; kernel size = 3x5x5 (channel first kernel) and 6 filters in total
- Hence there are 6 outputs of dimension 28x28; the final output dimension=1x6x28x28
- The total parameters: 6x3x5x5+6= 456
- Since there are 6 filters, there are 6 biases
- The number of filters will give the number of channels of output

https://cs231n.stanford.edu/

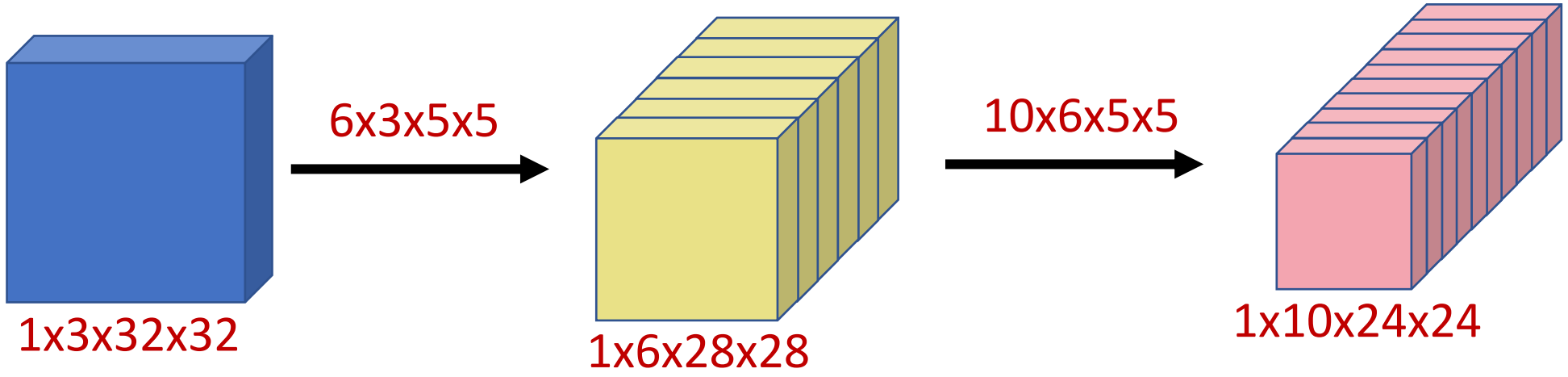
Examples of convolutions and parameter calculations



- Here, stride = 1, H=32; W=32; kernel size = 3x5x5 (channel first kernel) and 6 filters in total
- Hence there are 2 outputs of dimension 6x28x28; the final output dimension=2x6x28x28
- The total parameters: 6x3x5x5+6= 456
- Since there are 6 filters, there are 6 biases
- This example shows when there is a batched input, there is a batched output as well

https://cs231n.stanford.edu/

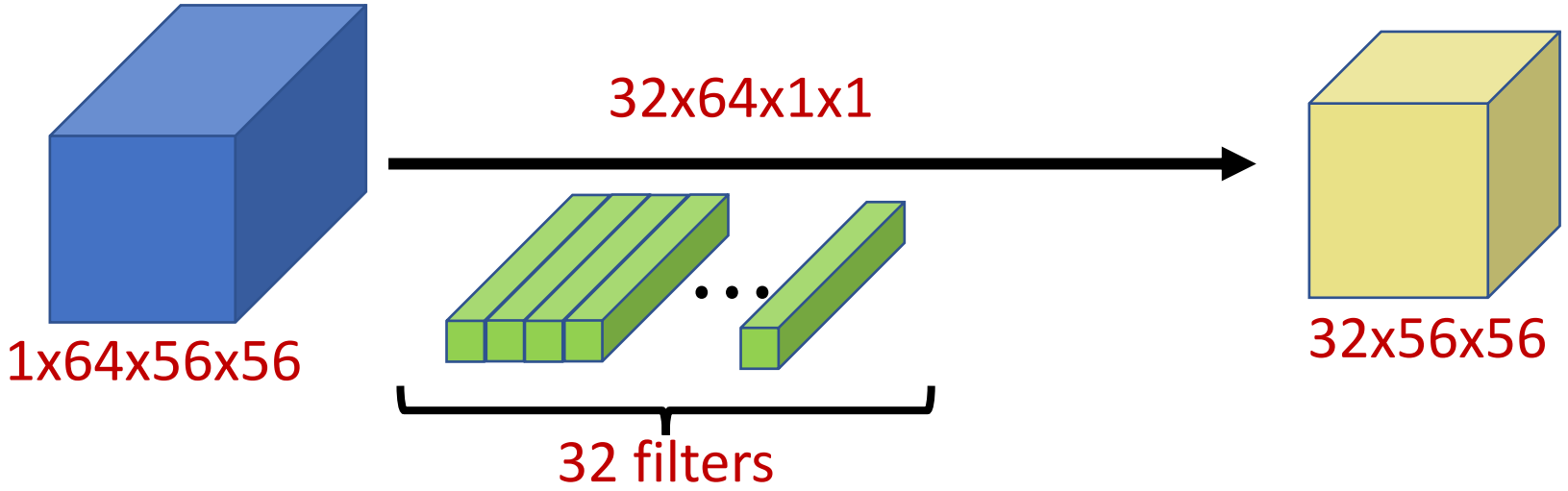
Examples of convolutions and parameter calculations



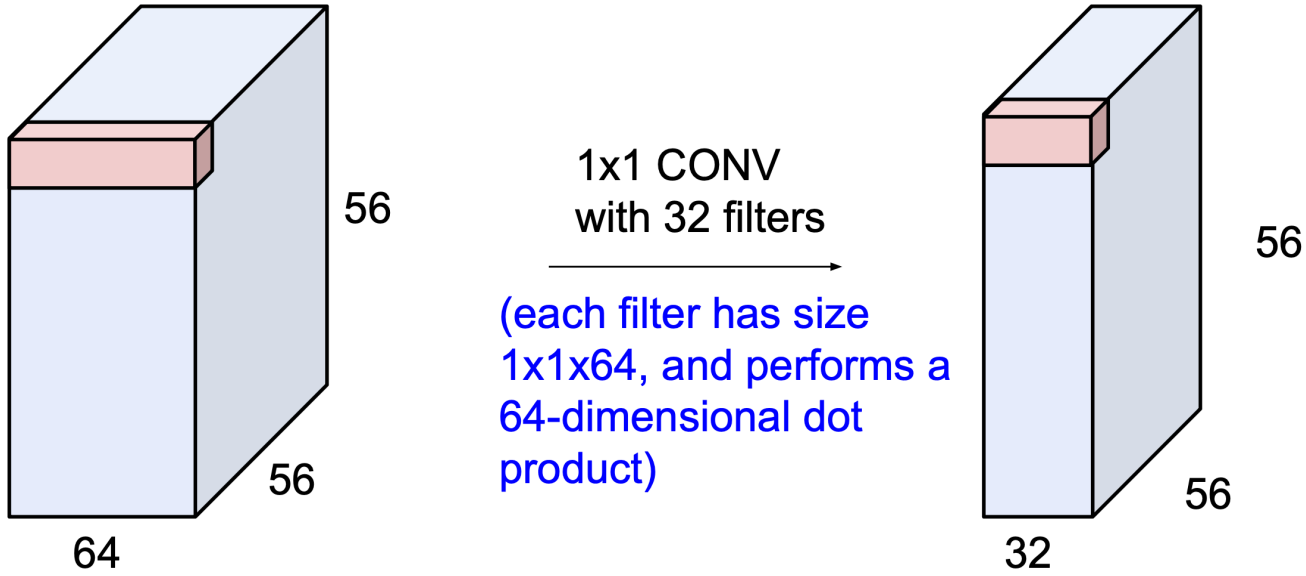
- A sample of convnet is provided, here we see that we apply repeated convolution to make the feature sizes as smaller and thicker (by channels) as possible; 0 padding and stride of 1
- The number of kernels should match the output of the channel of the feature map, hence we use different number of kernels to determine the thickness (of channel) of feature map
- The number of parameters for these two conv operations:
  - 6x3x5x5+6+10x6x5x5+10 = 1966
  - First conv layer has 6 filters hence bias of 6; similarly for 2nd conv layer, it has 10 filters; hence a bias of 10

https://cs231n.stanford.edu/

Examples of convolutions and parameter calculations

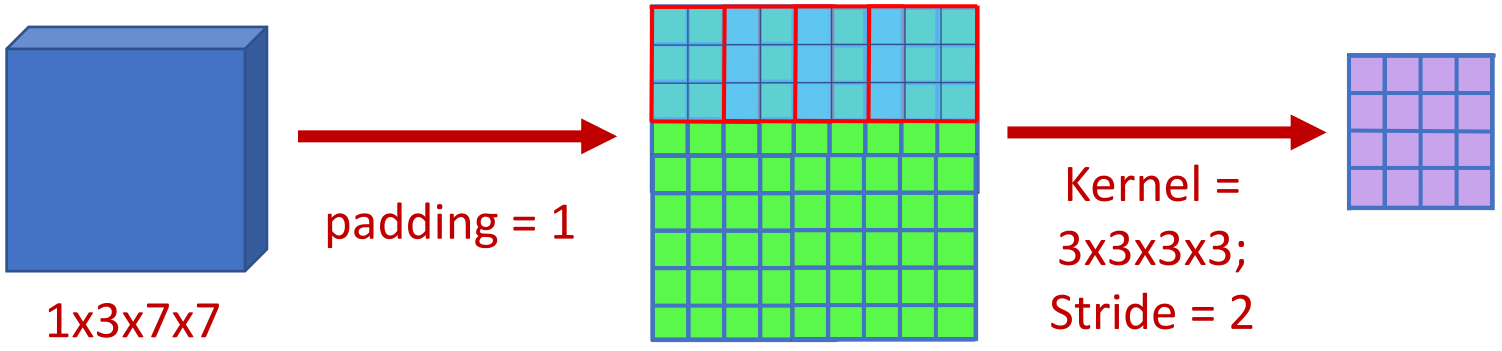


- Here, we apply 1x1 convolution of 64 channel; we apply 32 of them all together
- The number of parameters for these two conv operations:
  - 32x64+32 = 2080
- Convolution of Cx1x1 transforms an input volume of 1xCxHxW to 1xHxW, hence we can apply any number of such convolution to shrink or expand the input volume.



https://cs231n.stanford.edu/

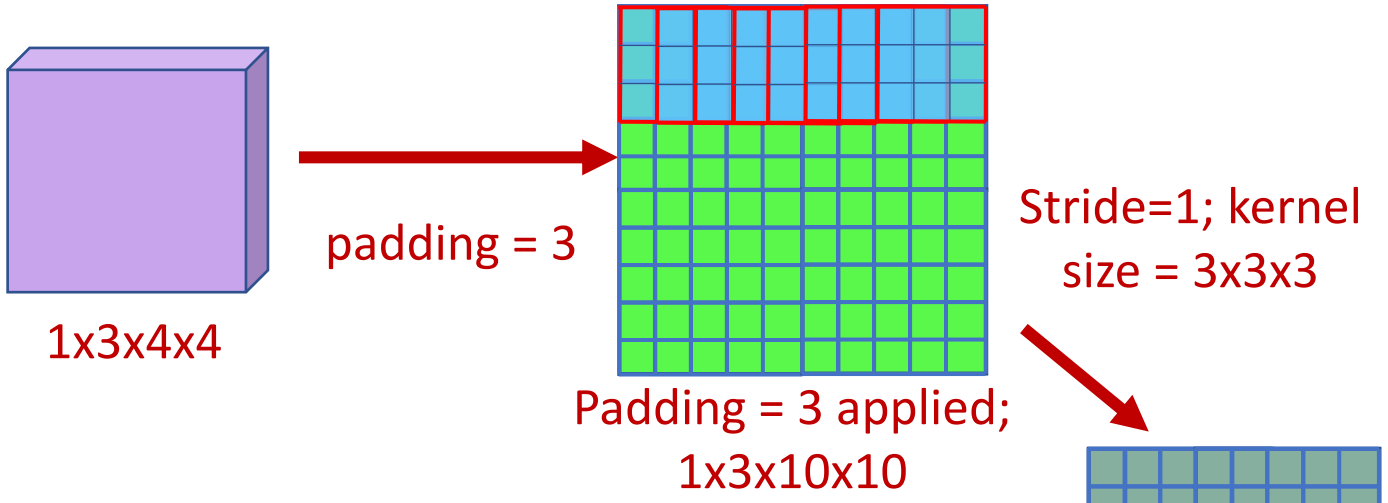
Examples of convolutions and parameter calculations




```
class Example_ffcnn(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(in_channels=3,
                               out_channels=3,kernel_size=3,
                               stride=2, padding=1)
        self.conv2 = nn.Conv2d(in_channels=3,
                               out_channels=3, kernel_size=3,
                               stride=1, padding=3)
        self.bn2d = nn.BatchNorm2d(3)
        self.maxpool = nn.MaxPool2d(2)
        self.flatten = torch.nn.Flatten()
        self.fc1 = nn.Linear(48, 10)
        self.bn1d = nn.BatchNorm1d(10)
        self.out = nn.Linear(10, 1)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = self.bn2d(x)
        x = self.maxpool(x)
        x = self.flatten(x)
        x = self.fc1(x)
        x = self.bn1d(x)
        x = self.out(x)
        x = self.sigmoid(x)
        return x
```

- Let's assume we have an input of size 1x3x7x7; what are the feature volumes and parameters of the whole CNN?
- Since out-channels = 3; hence the output channel of the feature volume=3;
- First we add a padding of 1 (on all side of the image) then we perform convolution using a stride of 2
- Number of params for this layer = 3x3x3x3+3=84

Examples of convolutions and parameter calculations



- Since out-channels = 3; hence the output channel of the feature volume=3;
- First we add a padding of 3 (on all side of the image) then we perform convolution using stride 1
- Number of params for this layer = 3x3x3x3+3=84

```
class Example_ffcnn(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(in_channels=3,
                               out_channels=3,kernel_size=3,
                               stride=2, padding=1)
        self.conv2 = nn.Conv2d(in_channels=3,
                               out_channels=3, kernel_size=3,
                               stride=1, padding=3)
        self.bn2d = nn.BatchNorm2d(3)
        self.maxpool = nn.MaxPool2d(2)
        self.flatten = torch.nn.Flatten()
        self.fc1 = nn.Linear(48, 10)
        self.bn1d = nn.BatchNorm1d(10)
        self.out = nn.Linear(10, 1)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = self.bn2d(x)
        x = self.maxpool(x)
        x = self.flatten(x)
        x = self.fc1(x)
        x = self.bn1d(x)
        x = self.out(x)
        x = self.sigmoid(x)
        return x
```

Examples of convolutions and parameter calculations

- Batchnorm2D independently normalizes, scales, and shifts each channel of the input feature map
- Hence, the number of parameters is the number of channel x 2; here it is 3 for **gamma** and 3 for **beta** as it has 2 parameters per channel.
- The dimension of the feature volume doesn't change in this case.
- Hence the total number of parameters for this layer is 6

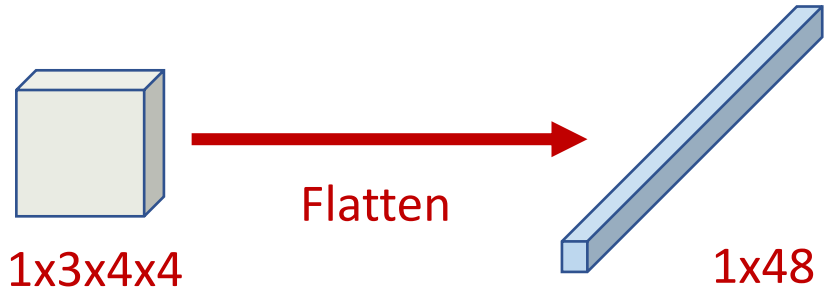
```
class Example_ffcnn(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(in_channels=3,
                               out_channels=3,kernel_size=3,
                               stride=2, padding=1)
        self.conv2 = nn.Conv2d(in_channels=3,
                               out_channels=3, kernel_size=3,
                               stride=1, padding=3)
        self.bn2d = nn.BatchNorm2d(3)
        self.maxpool = nn.MaxPool2d(2)
        self.flatten = torch.nn.Flatten()
        self.fc1 = nn.Linear(48, 10)
        self.bn1d = nn.BatchNorm1d(10)
        self.out = nn.Linear(10, 1)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = self.bn2d(x)
        x = self.maxpool(x)
        x = self.flatten(x)
        x = self.fc1(x)
        x = self.bn1d(x)
        x = self.out(x)
        x = self.sigmoid(x)
        return x
```

Examples of convolutions and parameter calculations

- Maxpool doesn't have any extra parameters
- Maxpool just collects the most prominent features and downsamples the feature block
- Hence the total number of parameters for this layer = 0

```
class Example_ffcnn(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(in_channels=3,
                               out_channels=3,kernel_size=3,
                               stride=2, padding=1)
        self.conv2 = nn.Conv2d(in_channels=3,
                               out_channels=3, kernel_size=3,
                               stride=1, padding=3)
        self.bn2d = nn.BatchNorm2d(3)
        self.maxpool = nn.MaxPool2d(2)
        self.flatten = torch.nn.Flatten()
        self.fc1 = nn.Linear(48, 10)
        self.bn1d = nn.BatchNorm1d(10)
        self.out = nn.Linear(10, 1)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = self.bn2d(x)
        x = self.maxpool(x)
        x = self.flatten(x)
        x = self.fc1(x)
        x = self.bn1d(x)
        x = self.out(x)
        x = self.sigmoid(x)
        return x
```

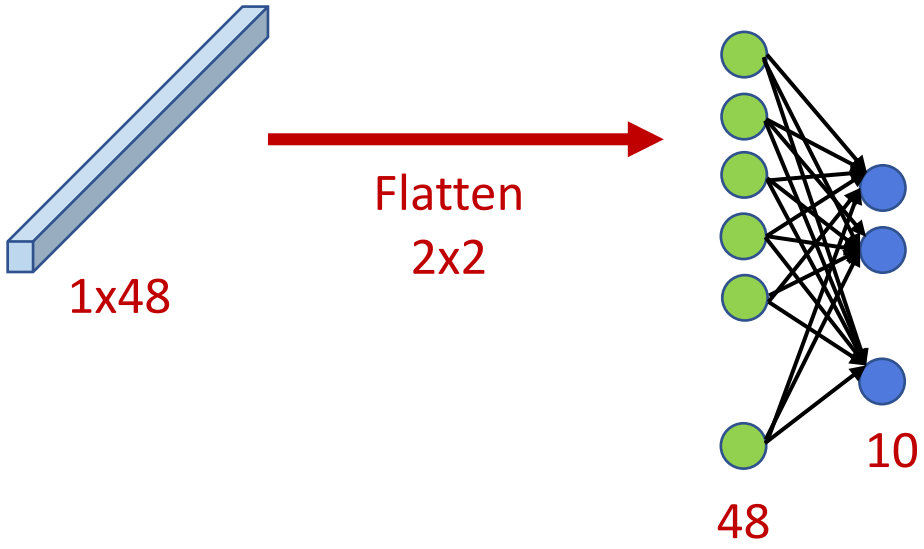
Examples of convolutions and parameter calculations



- Flatten doesn't have any extra parameters
- Flatten just flats the block to make it compatible as an input to a fully connected layer
- Hence the total number of parameters for this layer = 0

```
class Example_ffcnn(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(in_channels=3,
                               out_channels=3,kernel_size=3,
                               stride=2, padding=1)
        self.conv2 = nn.Conv2d(in_channels=3,
                               out_channels=3, kernel_size=3,
                               stride=1, padding=3)
        self.bn2d = nn.BatchNorm2d(3)
        self.maxpool = nn.MaxPool2d(2)
        self.flatten = torch.nn.Flatten()
        self.fc1 = nn.Linear(48, 10)
        self.bn1d = nn.BatchNorm1d(10)
        self.out = nn.Linear(10, 1)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = self.bn2d(x)
        x = self.maxpool(x)
        x = self.flatten(x)
        x = self.fc1(x)
        x = self.bn1d(x)
        x = self.out(x)
        x = self.sigmoid(x)
        return x
```

Examples of convolutions and parameter calculations



- Structure of fully connected layers are just like complete bipartite graph
- The layer to which it is getting connected has number of nodes as the bias
- Hence the number of parameters = 48x10+10=490

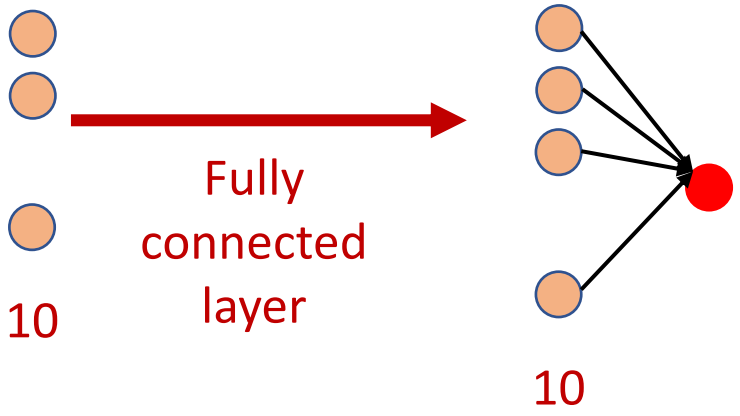
```
class Example_ffcnn(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(in_channels=3,
                               out_channels=3,kernel_size=3,
                               stride=2, padding=1)
        self.conv2 = nn.Conv2d(in_channels=3,
                               out_channels=3, kernel_size=3,
                               stride=1, padding=3)
        self.bn2d = nn.BatchNorm2d(3)
        self.maxpool = nn.MaxPool2d(2)
        self.flatten = torch.nn.Flatten()
        self.fc1 = nn.Linear(48, 10)
        self.bn1d = nn.BatchNorm1d(10)
        self.out = nn.Linear(10, 1)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = self.bn2d(x)
        x = self.maxpool(x)
        x = self.flatten(x)
        x = self.fc1(x)
        x = self.bn1d(x)
        x = self.out(x)
        x = self.sigmoid(x)
        return x
```

Examples of convolutions and parameter calculations

- Batchnorm1D independently normalizes, scales, and shifts each neuron/node of the input feature map
- There are 2 parameters for each of the neurons, hence this layer has 10x2=20 parameters

```
class Example_ffcnn(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(in_channels=3,
                               out_channels=3,kernel_size=3,
                               stride=2, padding=1)
        self.conv2 = nn.Conv2d(in_channels=3,
                               out_channels=3, kernel_size=3,
                               stride=1, padding=3)
        self.bn2d = nn.BatchNorm2d(3)
        self.maxpool = nn.MaxPool2d(2)
        self.flatten = torch.nn.Flatten()
        self.fc1 = nn.Linear(48, 10)
        self.bn1d = nn.BatchNorm1d(10)
        self.out = nn.Linear(10, 1)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = self.bn2d(x)
        x = self.maxpool(x)
        x = self.flatten(x)
        x = self.fc1(x)
        x = self.bn1d(x)
        x = self.out(x)
        x = self.sigmoid(x)
        return x
```

Examples of convolutions and parameter calculations



- Fully connectd layers have 10x1+1 = 11 parameters for this layer

```
class Example_ffcnn(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(in_channels=3,
                               out_channels=3,kernel_size=3,
                               stride=2, padding=1)
        self.conv2 = nn.Conv2d(in_channels=3,
                               out_channels=3, kernel_size=3,
                               stride=1, padding=3)
        self.bn2d = nn.BatchNorm2d(3)
        self.maxpool = nn.MaxPool2d(2)
        self.flatten = torch.nn.Flatten()
        self.fc1 = nn.Linear(48, 10)
        self.bn1d = nn.BatchNorm1d(10)
        self.out = nn.Linear(10, 1)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = self.bn2d(x)
        x = self.maxpool(x)
        x = self.flatten(x)
        x = self.fc1(x)
        x = self.bn1d(x)
        x = self.out(x)
        x = self.sigmoid(x)
        return x
```

Examples of convolutions and parameter calculations

- Sigmoid doesn't have any parameter of it's own, it's an activation function
- Total parameters of this network:

  84+84+6+0+0+490+20+11 = **695**

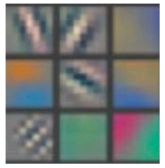
- This network is a very simple neural network, can be used for binary classification of 3-channel 7x7 images

```
class Example_ffcnn(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(in_channels=3,
                               out_channels=3,kernel_size=3,
                               stride=2, padding=1)
        self.conv2 = nn.Conv2d(in_channels=3,
                               out_channels=3, kernel_size=3,
                               stride=1, padding=3)
        self.bn2d = nn.BatchNorm2d(3)
        self.maxpool = nn.MaxPool2d(2)
        self.flatten = torch.nn.Flatten()
        self.fc1 = nn.Linear(48, 10)
        self.bn1d = nn.BatchNorm1d(10)
        self.out = nn.Linear(10, 1)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = self.bn2d(x)
        x = self.maxpool(x)
        x = self.flatten(x)
        x = self.fc1(x)
        x = self.bn1d(x)
        x = self.out(x)
        x = self.sigmoid(x)
        return x
```

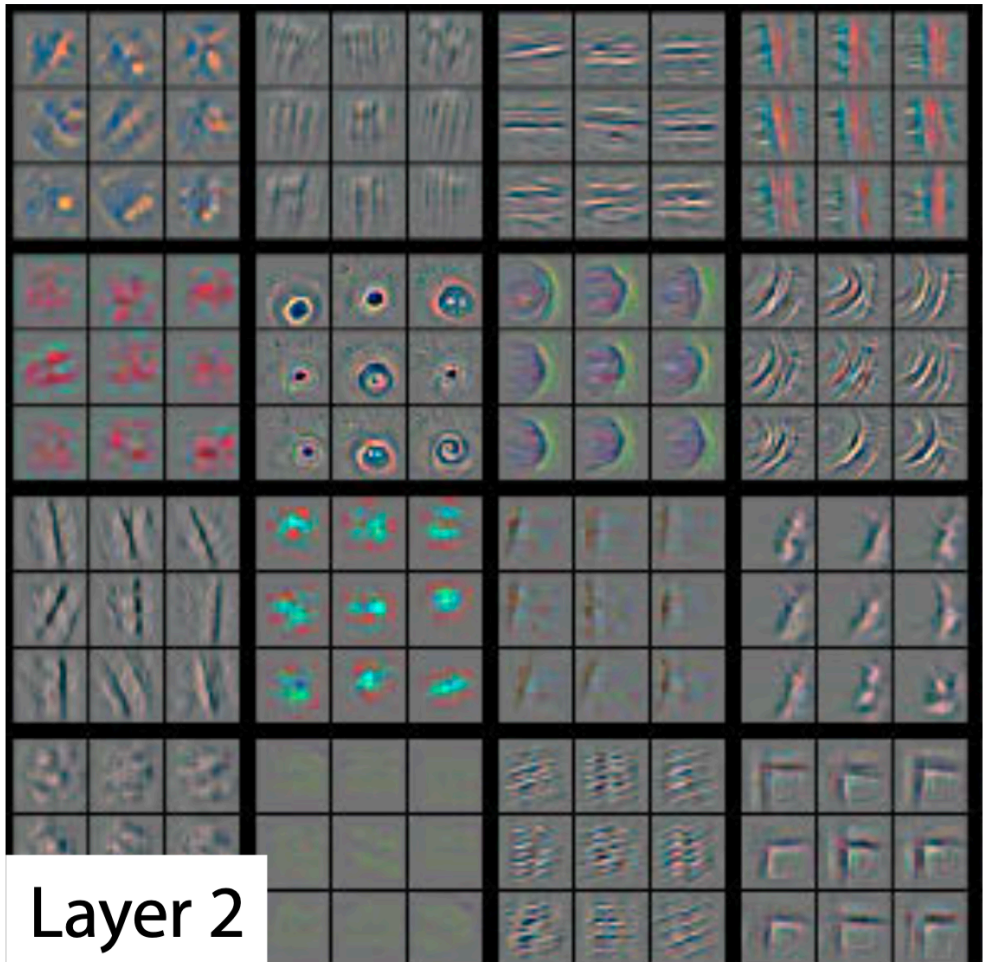
Features learnt by a CNN

Feature learnt by the CNNs

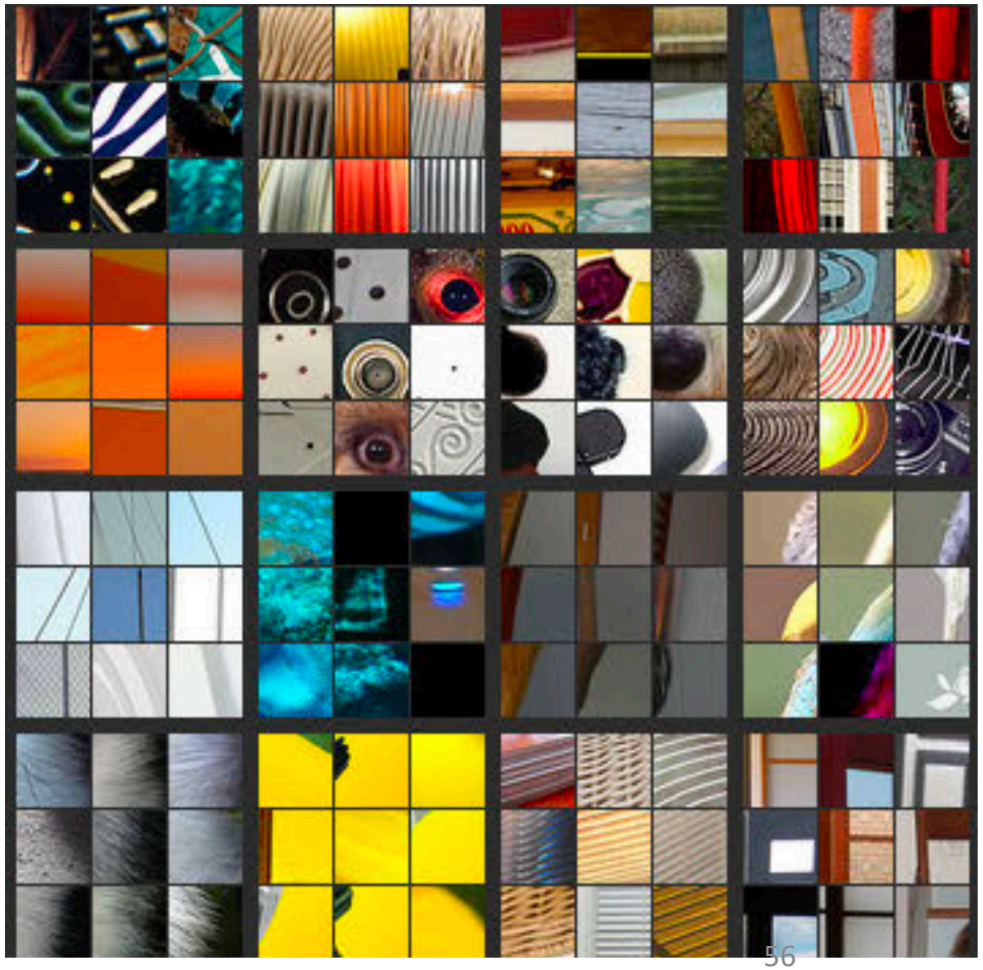
- Visualization obtained by passing an input image through a trained CNN, then selectively projecting activations back into image space using a “Deconvnet” (de-convolutional network)
- Early layers detect edges, colors and low-level features



Layer 1



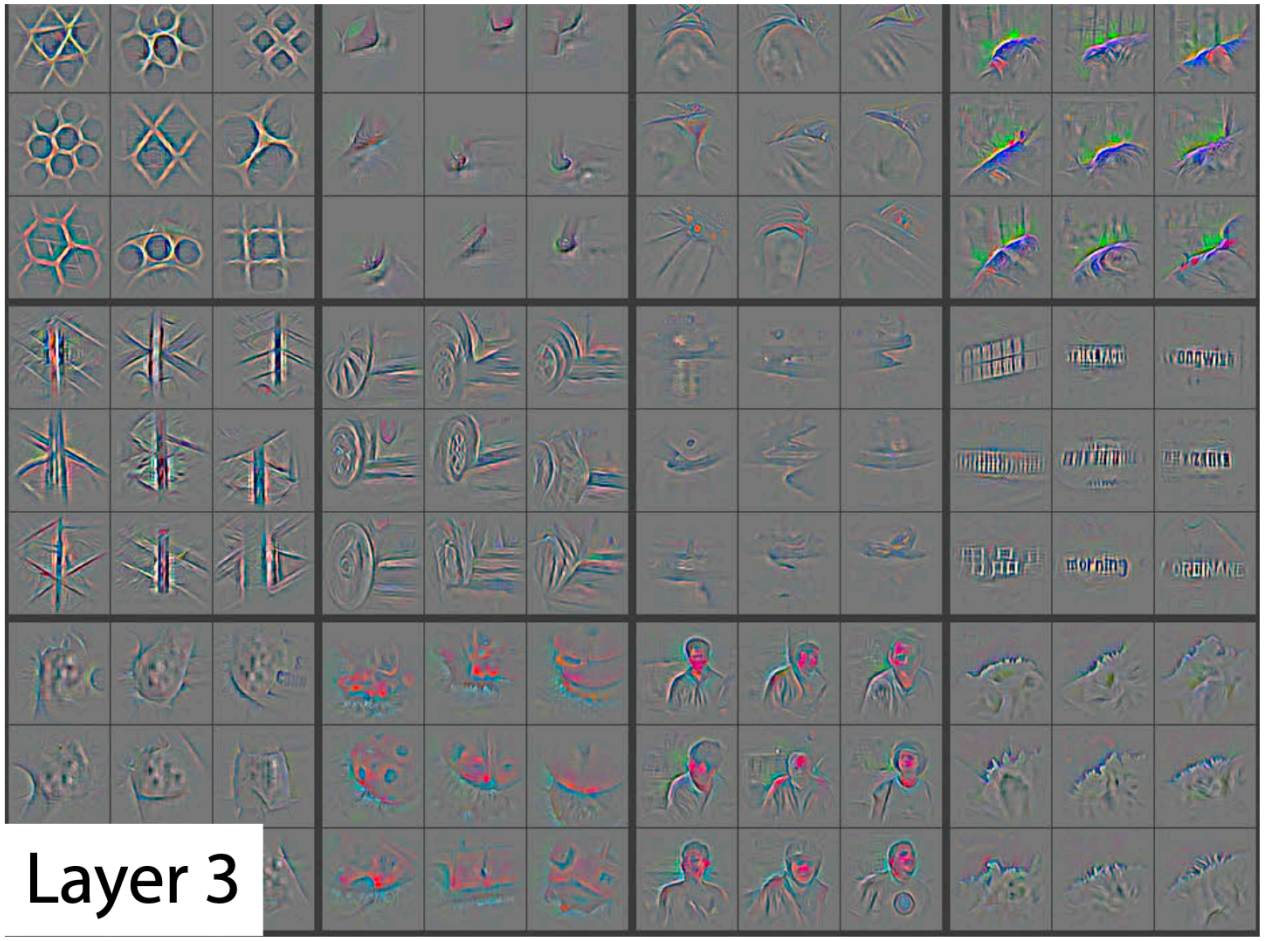
Layer 2



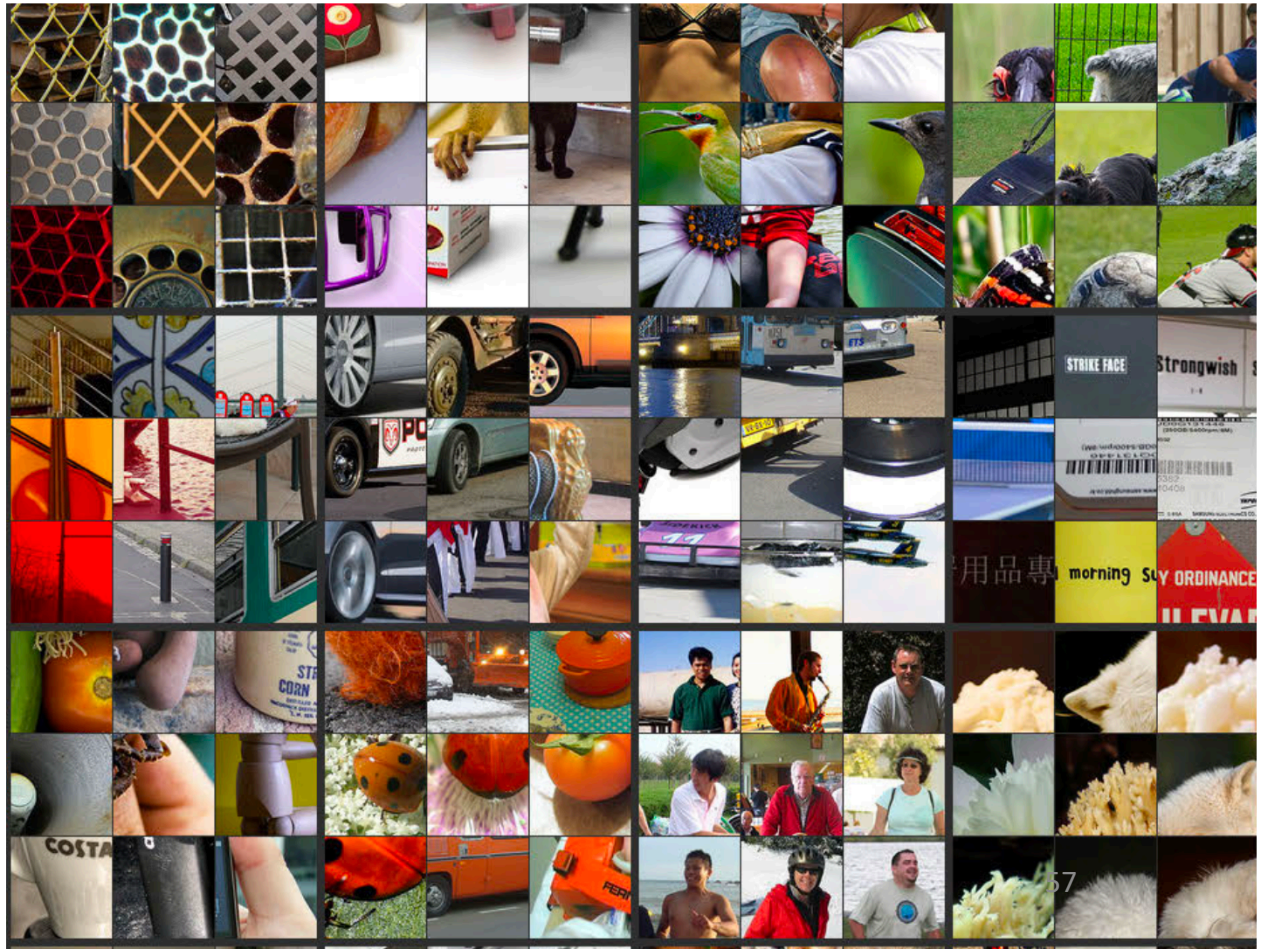
https://arxiv.org/abs/1311.2901

Feature learnt by the CNNs

- During the reverse pass, they used the stored max-pooling indices to put features back in their original locations (un-pooling), then inverted the convolution/ReLU layers to reconstruct which image patterns activated specific neurons
- Mid-level layers captures textures/shapes

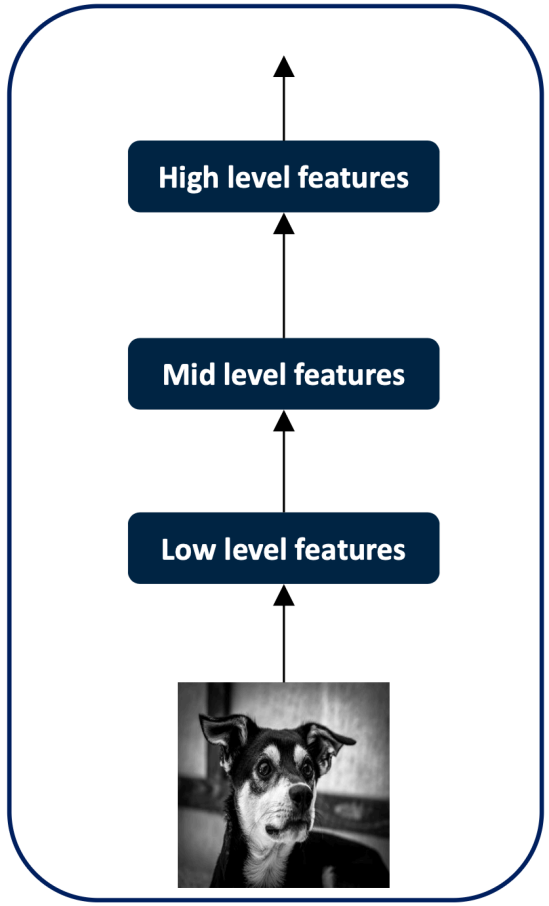


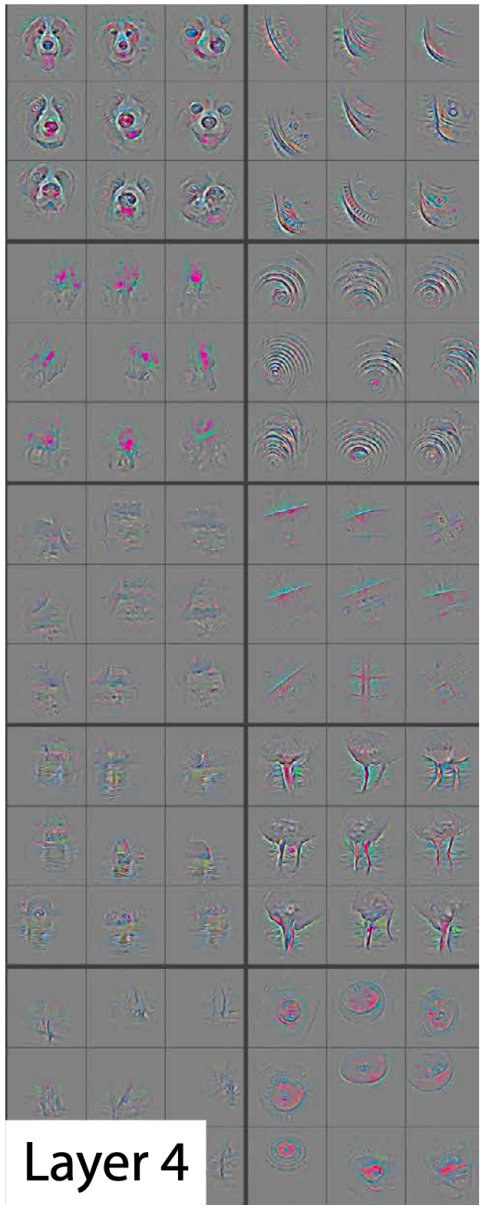
Layer 3

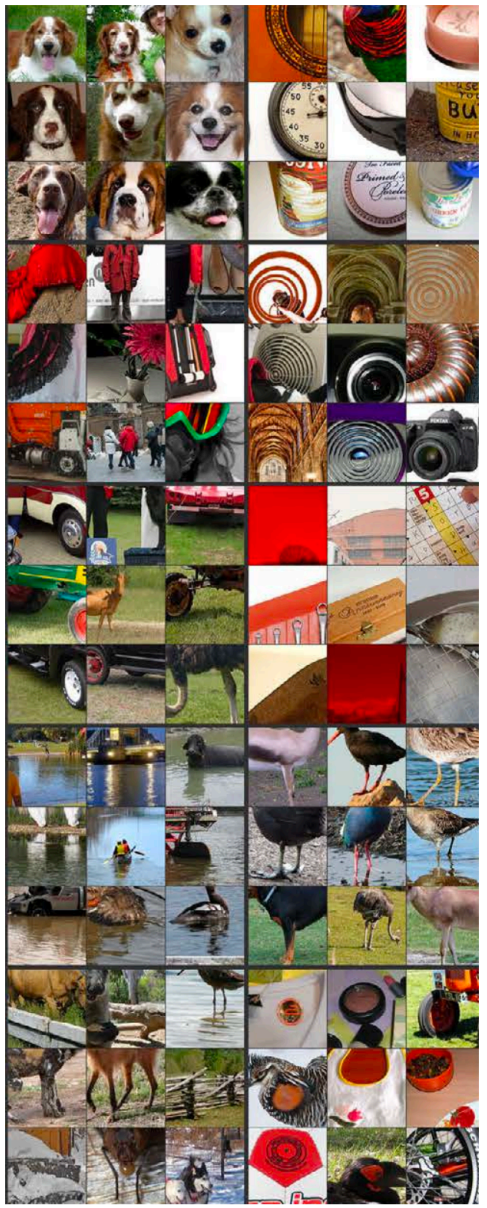


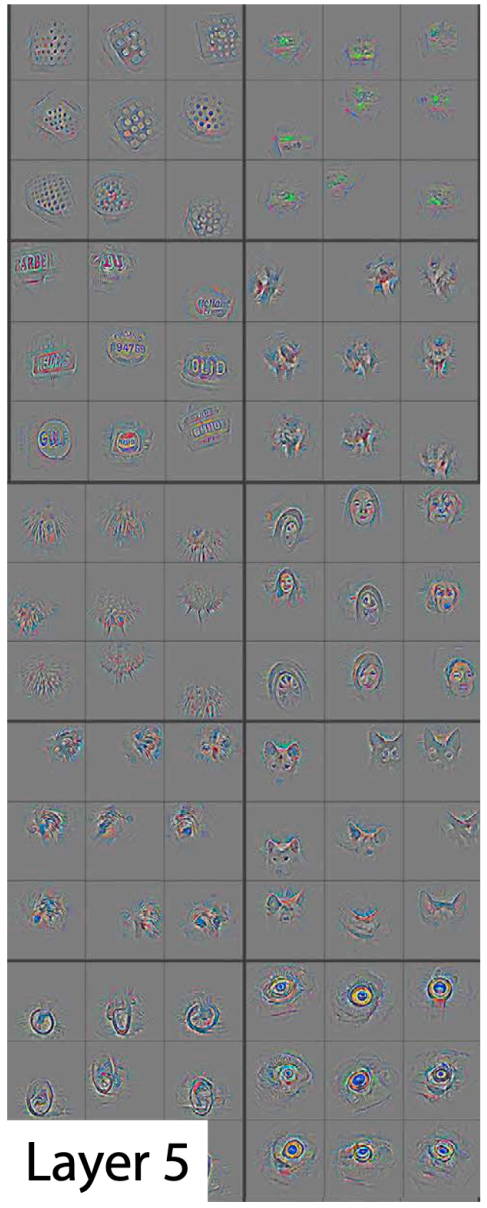
Feature learnt by the CNNs

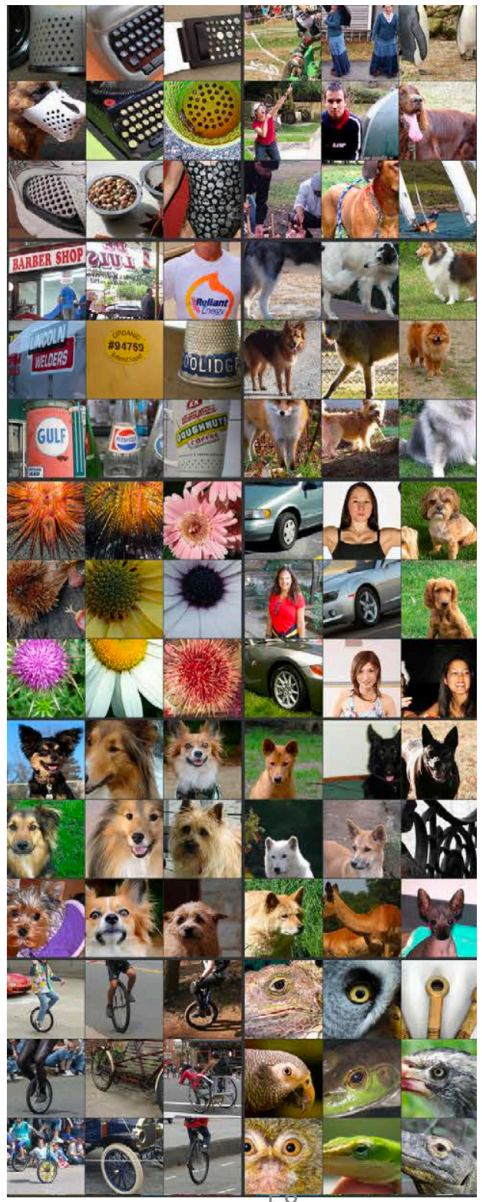
- Higher layers (close to the o/p layer) responds to object parts or entire objects









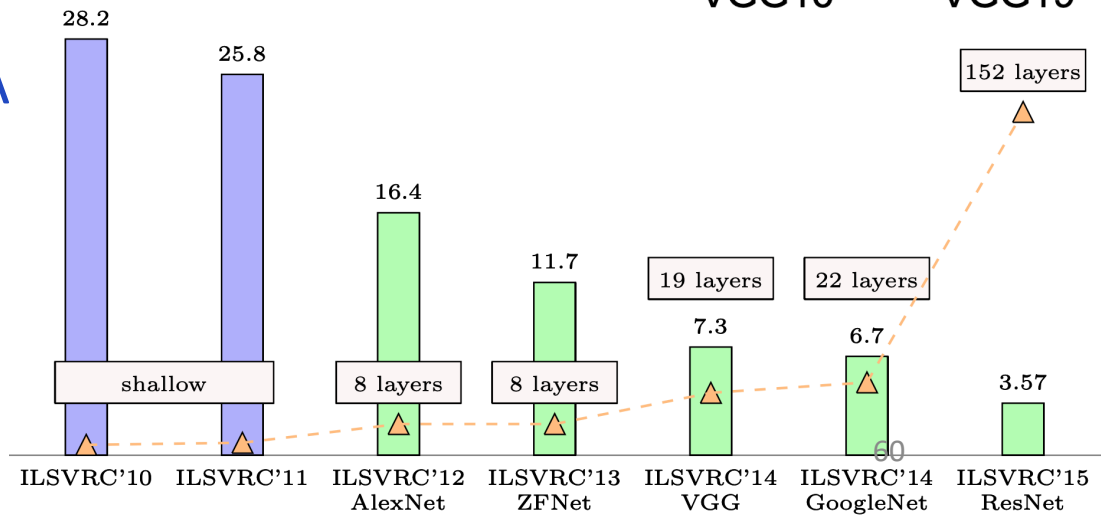


https://arxiv.org/abs/1311.2901

Some more architectures

CNN Architectures - VGGNet

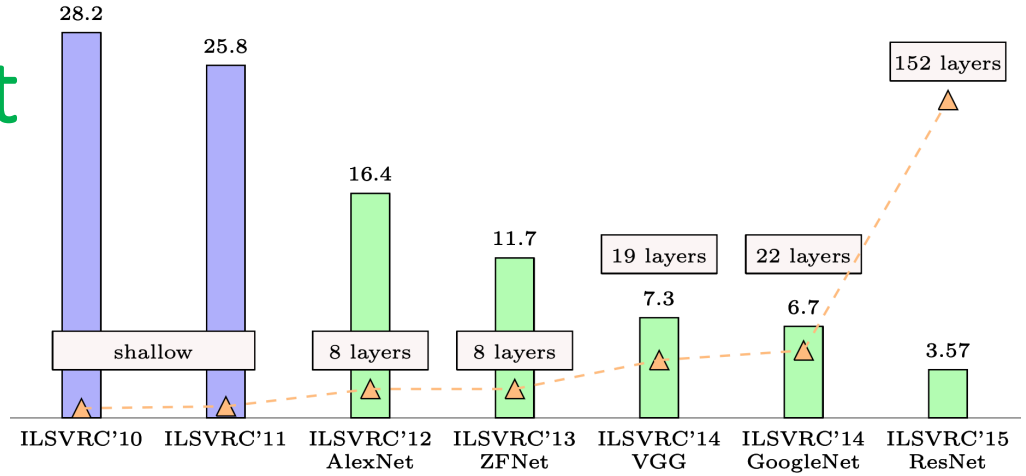
- Significantly deeper (with 16-19 layers) than AlexNet (8 layers)with about 140 million parameters, ~7.3% in top 5-error.
- Used smaller filter's sizes, a stack of three 3x3 conv (stride 1) layers (30 parameters) has the same effective receptive field as one 7x7 conv layer (50 parameters).
- Made deeper networks with lesser parameters, the **more deeper the more non-linearity it captures**.
- Used ensembles, features of deeper layers generalizes well to other tasks, trained on 4 NVIDIA Titan Black GPUs for 2-3 weeks
- VGGNet reinforced the notion that CNN have to have a deep network of layers in order for the hierarchical representation of visual data to work

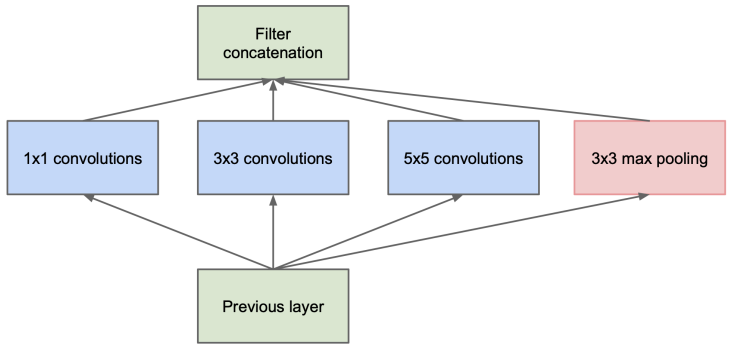


https://mit6874.github.io/ https://cs231n.stanford.edu/ https://arxiv.org/abs/1409.1556

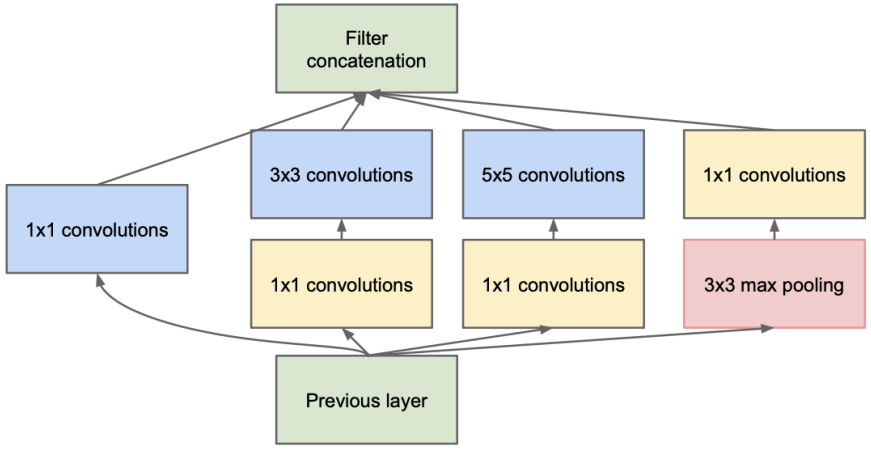
CNN Architectures - GoogLeNet

- Had no fully connected layers, ~5 million parameters, ~6.7% top 5 error; 22 layers.
- Used Inception modules – good network topology (network within a network) and then stack these modules on top of each other
- Apply parallel filter operations on input from previous layer, multiple receptive field sizes for conv (5x5, 3x3, 1x1), pooling operations (3x3) and concatenate together channel-wise.
- Used "bottleneck layers" that use 1x1 convolutions to reduce feature channel size.
- Uses Average pool to 1x1 and then a final Softmax layer for the final classification output

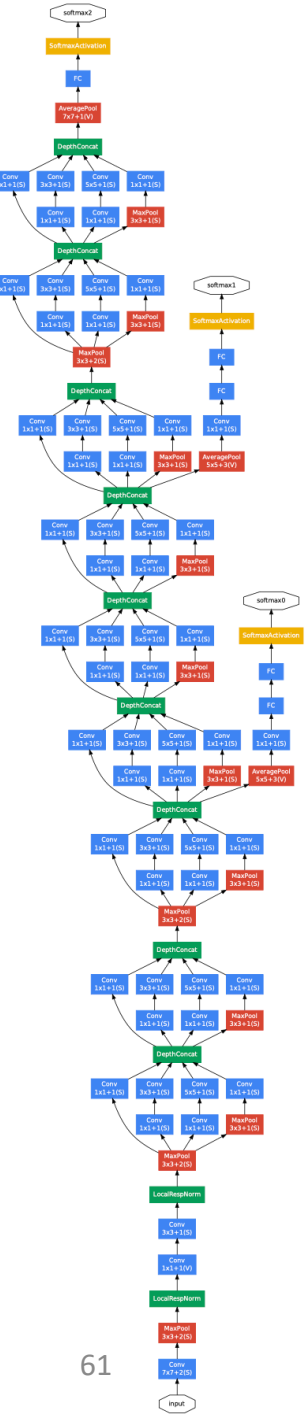




(a) Inception module, naïve version



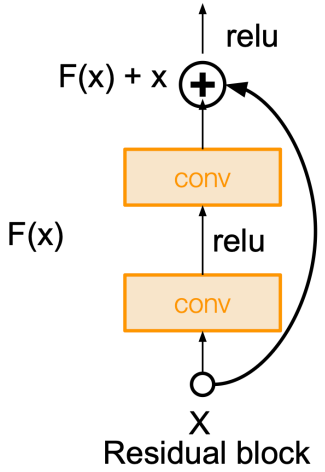
(b) Inception module with dimension reductions

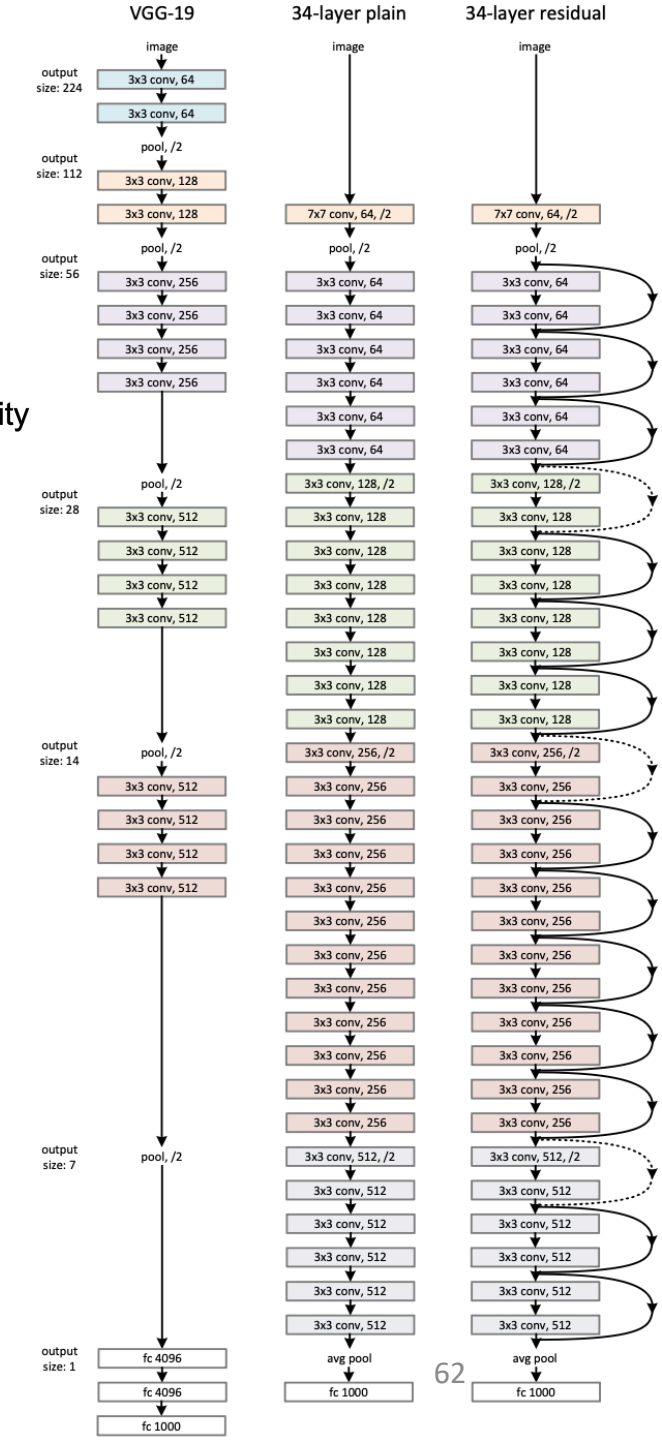


https://mit6874.github.io/ https://cs231n.stanford.edu/ https://arxiv.org/pdf/1409.4842

CNN Architectures - ResNet

- Surpasses human level (5.1% error) giving a top 5 error of 3.6%
- 152-layer model, winner of classification and detection competitions in ILSVRC'15 and COCO'15
- Showed that if we keep on stacking layers (deeper models), we might under-perform; not by overfitting! rather harder to optimize

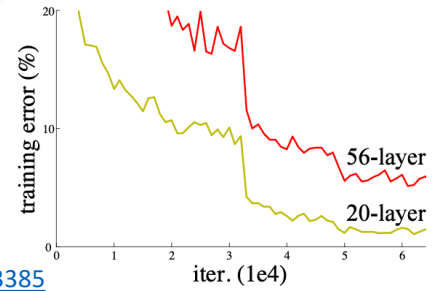


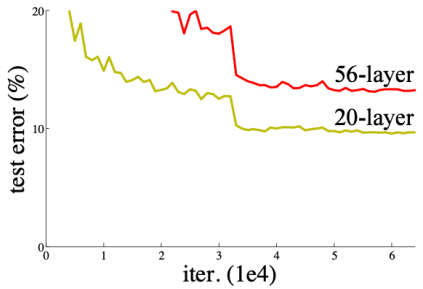


- Problem of vanishing and exploding gradients in training DNNs.
- Deeper model should learn at **least as good as the shallow layer model** – copying the learned layers from shallower model and setting additional layers to identity mapping, i.e., H(x) = F(x)+x = x, if F(x)=0; this is still one of the best architectures we have till date!

MSRA @ ILSVRC & COCO 2015 Competitions

- **1st places in all five main tracks**
  - ImageNet Classification: "*Ultra-deep*" (quote Yann) 152-layer nets
  - ImageNet Detection: 16% better than 2nd
  - ImageNet Localization: 27% better than 2nd
  - COCO Detection: 11% better than 2nd
  - COCO Segmentation: 12% better than 2nd





https://mit6874.github.io/ https://cs231n.stanford.edu/ https://arxiv.org/pdf/1512.03385

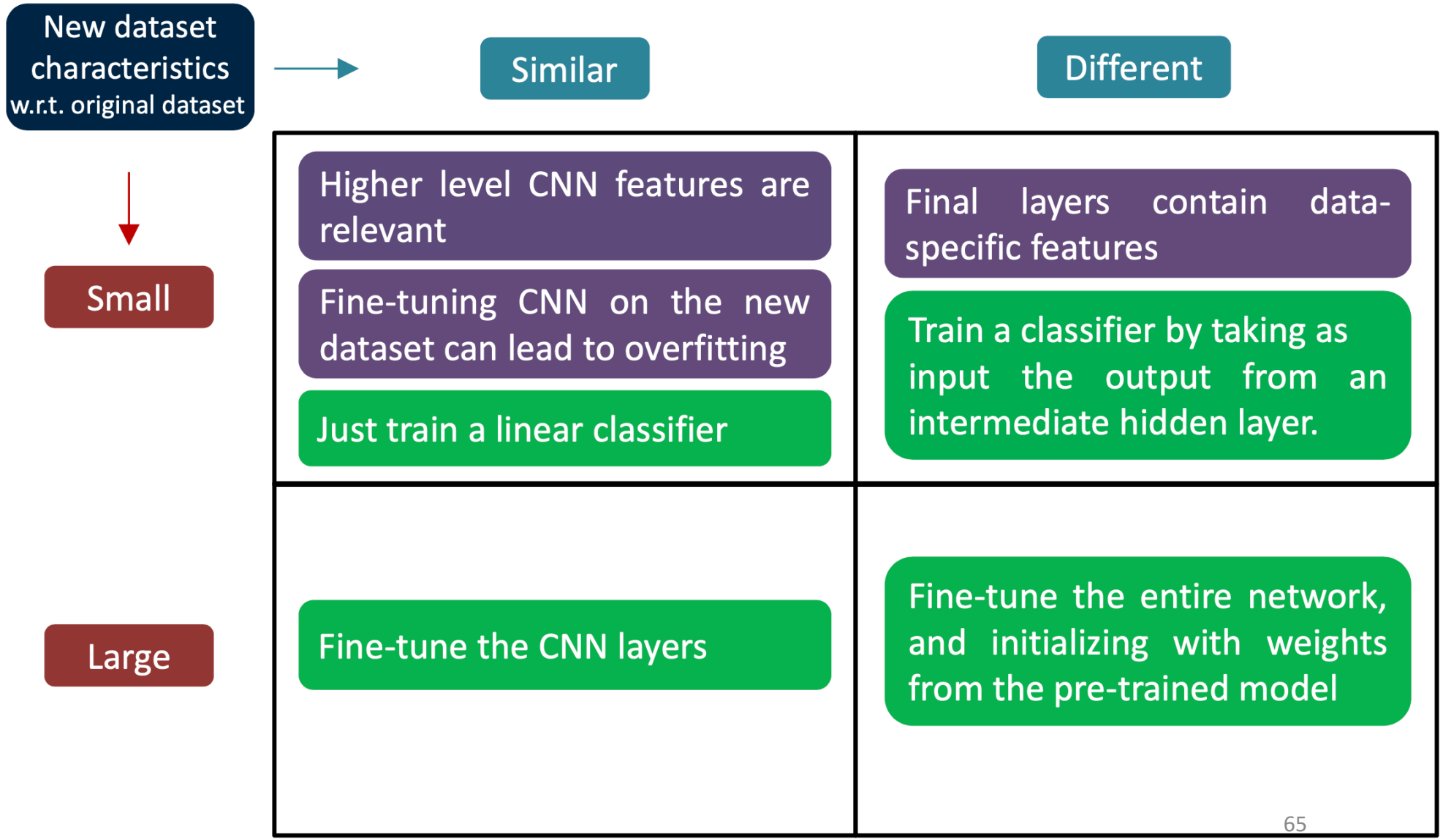
Transfer Learning in CNNs

Transfer Learning in CNNs

- Training a CNN model from scratch is difficult:
  - Need a large dataset to train the model
  - Well-known architectures takes weeks to train using multiple GPUs
- Inspiration:
  - Low-level features such as edge, corner, color-blob detectors are generic
  - High-level features become more specific to the classes in the original trained dataset
- Extraction of features
  - Remove the last Fully-connected layer, but treat all the other layers as fixed
  - On passing a new dataset yields a features of fixed dimension for each example
  - Use the resultant features to train a new classifier
- Fine-tuning CNN model
  - Fine-tune the weights of the pre-trained model using back-propagation
  - The whole network or some higher layers (close to the o/p) can be fine-tuned

Transfer Learning in CNNs

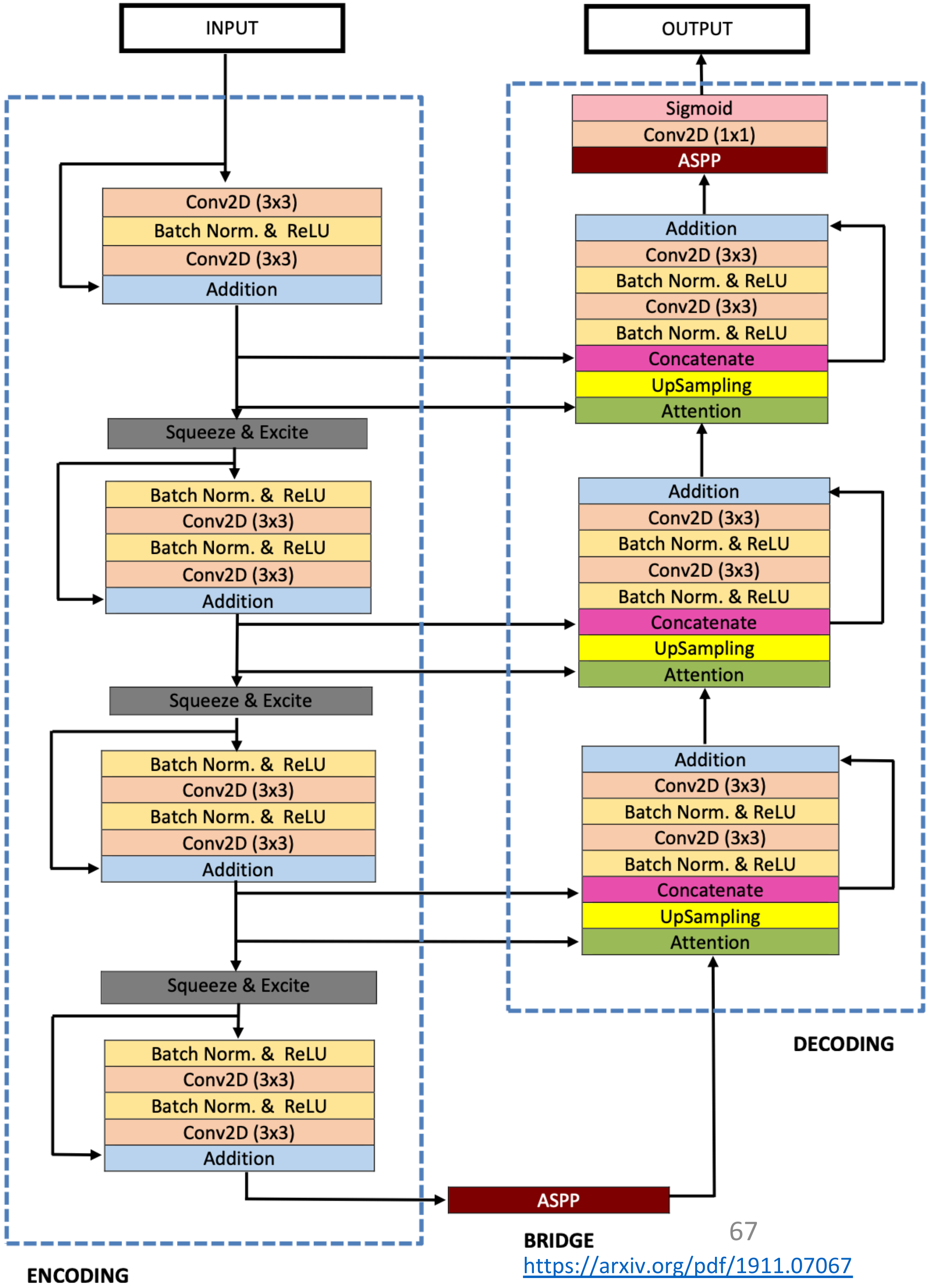
- These scenarios can be applied to transfer learning in Transformers as well – which are known for training on large datasets
- Models pre-trained on large datasets can be fine-tuned on different down-stream tasks



Demo – Multiclass Image segmentation: segmenting wild plants

Demo – ResUNet++ for Multiclass Image Segmentation

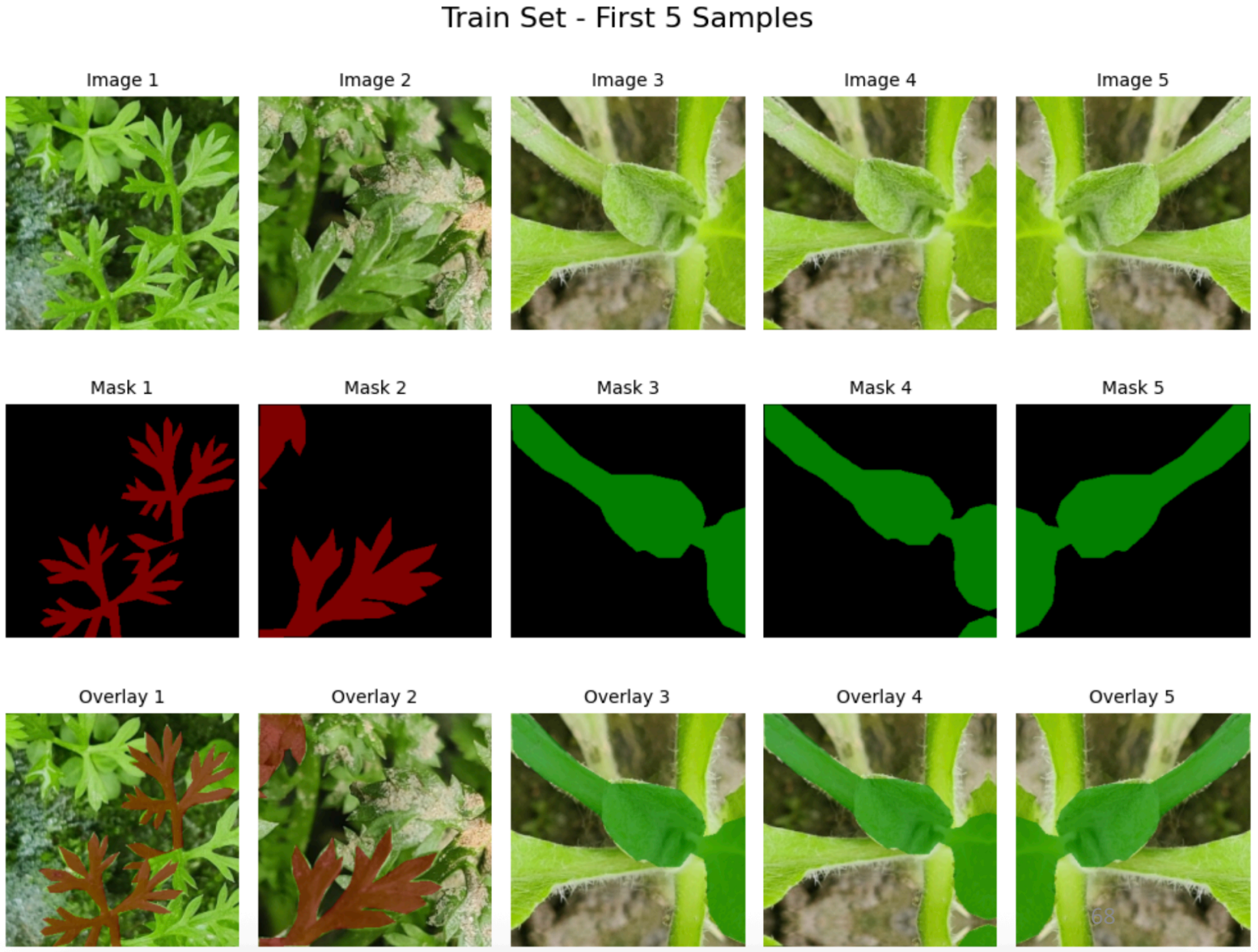
- **Improved ResUNet** architecture for colonoscopy segmentation.
- **Residual unit** makes deep network easy to train, and skip connection helps to propagate the information without degradation.
- Used **Squeeze and Excitation block** - each channel is squeezed by using **Global Avg. pooling** for generating channel-wise stats, the second step is **excitation for active calibration**.
- **Atrous Spatial Pyramid Pooling (ASPP)** - captures multi-scale information precisely by allowing controlling field of view, and resampling features at multiple scales.
- Uses attention mechanism; very lightweight network, has around 4.06 M parameters



https://arxiv.org/pdf/1911.07067

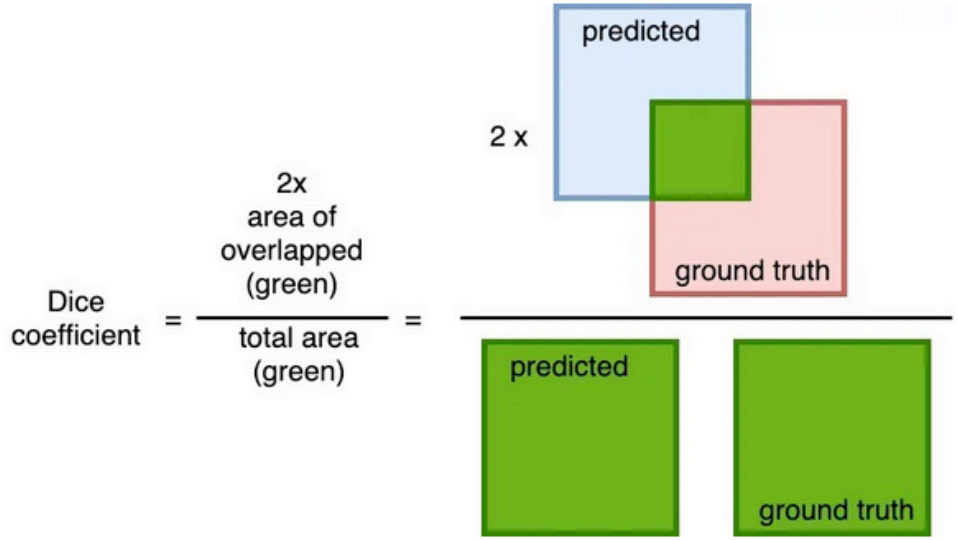
Demo – ResUNet++ for Multiclass Image Segmentation

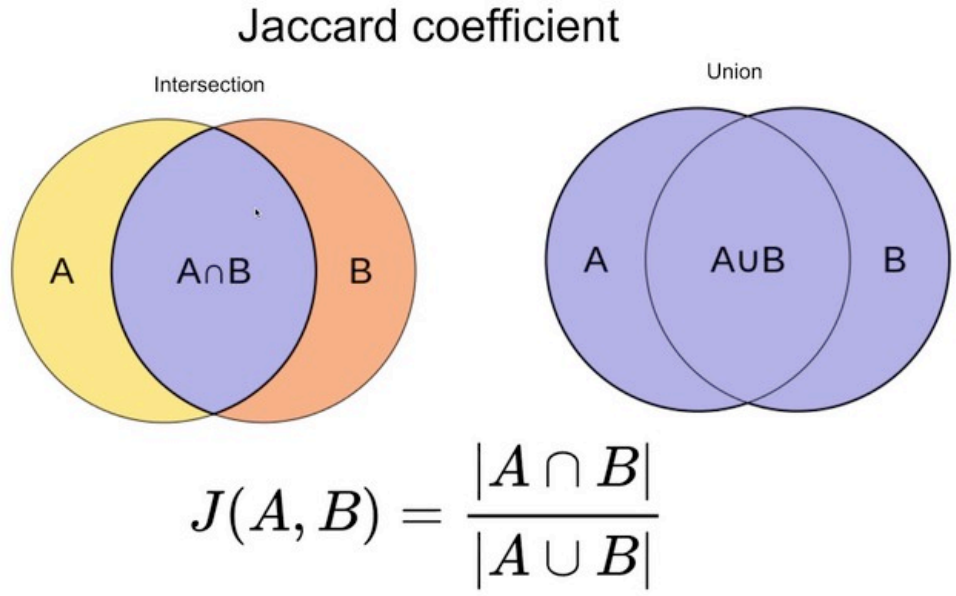
- There are two classes of wild plants, we need to divide the pixel into either class-0 (background, black), class-1 (wild plant, red) or class-2 (wild plant, green)
- We download the dataset, divide into 60-10-30 for train, validation and test split
- Use Softmax as output in the last layer of the model, input images are 3-channel of size 256x256
- We use pytorch and google colab for training.



Demo – ResUNet++ for Multiclass Image Segmentation

- We use Dice coefficient and Intersection over Union (jaccard) as metrics
- These are set operations, the classes 0,1,2; which can be written as one hot vectors over the pixel as [1, 0, 0], [0, 1, 0], and [0, 0, 1], hence, the mask output after softmax will be 3x256x256, where each of the pixel will have a probability values, we do an argmax to compute the scores/metrics as epochs progresses
- We have a train, validation and test function
- The train function trains the model, along with dumping the current statistics of the metrics for each epoch
- Validation dumps the statistics for each epoch on validation dataset
- Test epochs dumps the statistics and sample predictions for the test dataset





https://vipanchikatthula.github.io/project/jaccard-cosine-similarity/ https://i.sstatic.net/OsH4y.png

End of Lecture!