

Deep Learning & Neural Networks

Dr. Shahira Ahmed

DS710

Lecture 9



Deep Learning & Neural Networks DS710

Lecture 9
Convolution Neural Network
CNN

Dr. Shahira Ahmed



LEARNING OUTCOMES

- ◉ Understand the basics of Convolutional Neural Networks (CNN) and their applications in computer vision.
- ◉ Key components of CNN architecture
- ◉ Gain insights into state-of-the-art CNN architectures and recent advancements.



LECTURE OUTLINE

- ◉ Definition and importance of CNN in computer vision
- ◉ Basic Concepts of CNN
- ◉ CNN Architecture
- ◉ Introduction to popular CNN architectures (LeNet-5, AlexNet, VGG)

- CNNs are a class of **deep neural networks** specifically designed for processing and **analyzing visual data**.
- They are widely used in **computer vision tasks** such as image classification, object detection, and image segmentation.
- CNNs excel at capturing **hierarchical patterns and spatial relationships in images**, making them **highly effective** in understanding visual information.

Computer Vision Problems

Describes without errors



A person riding a motorcycle on a dirt road.

Describes with minor errors



Two dogs play in the grass.

Somewhat related to the image



A skateboarder does a trick on a ramp.

Image
Captioning



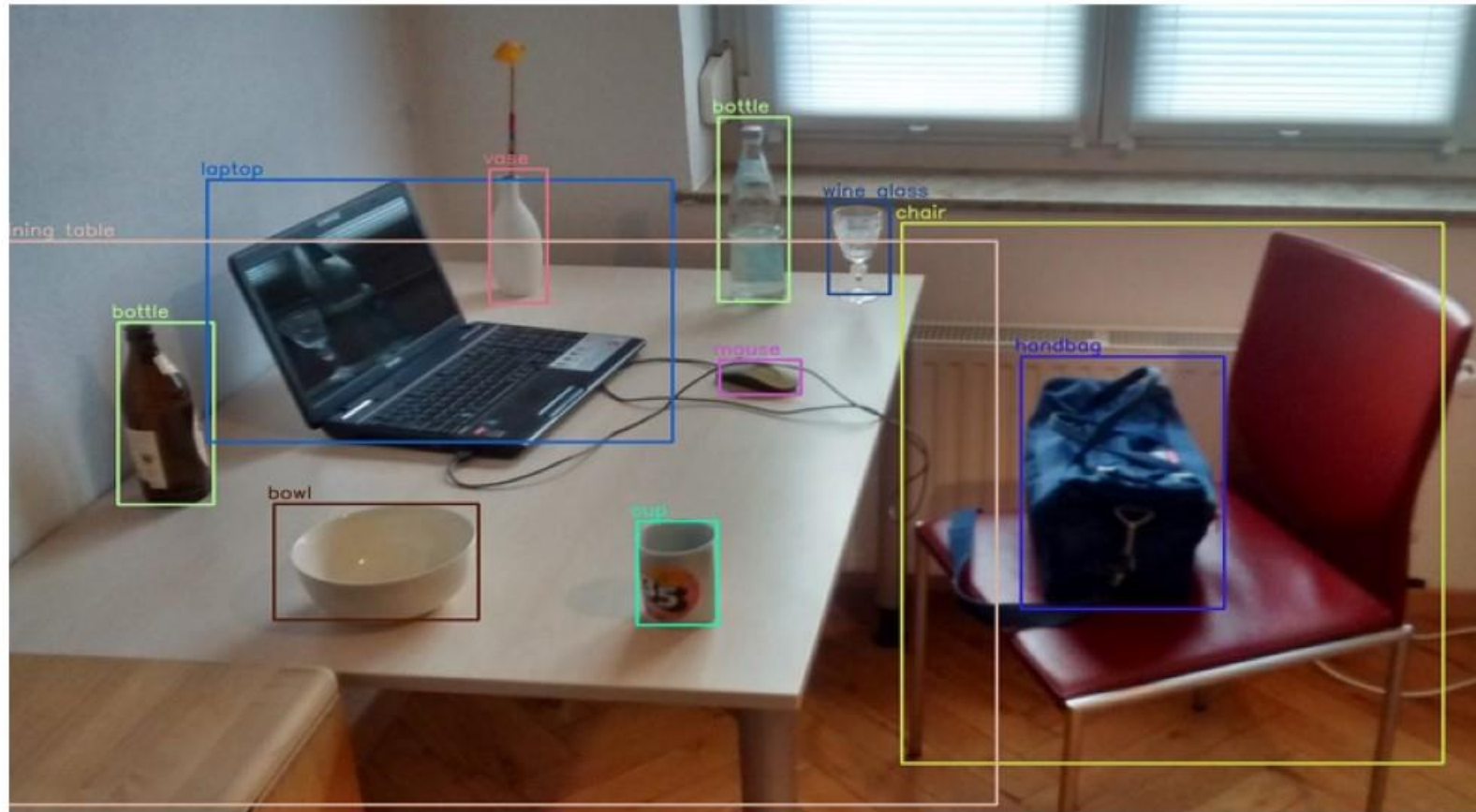
A group of young people playing a game of frisbee.



Two hockey players are fighting over the puck.



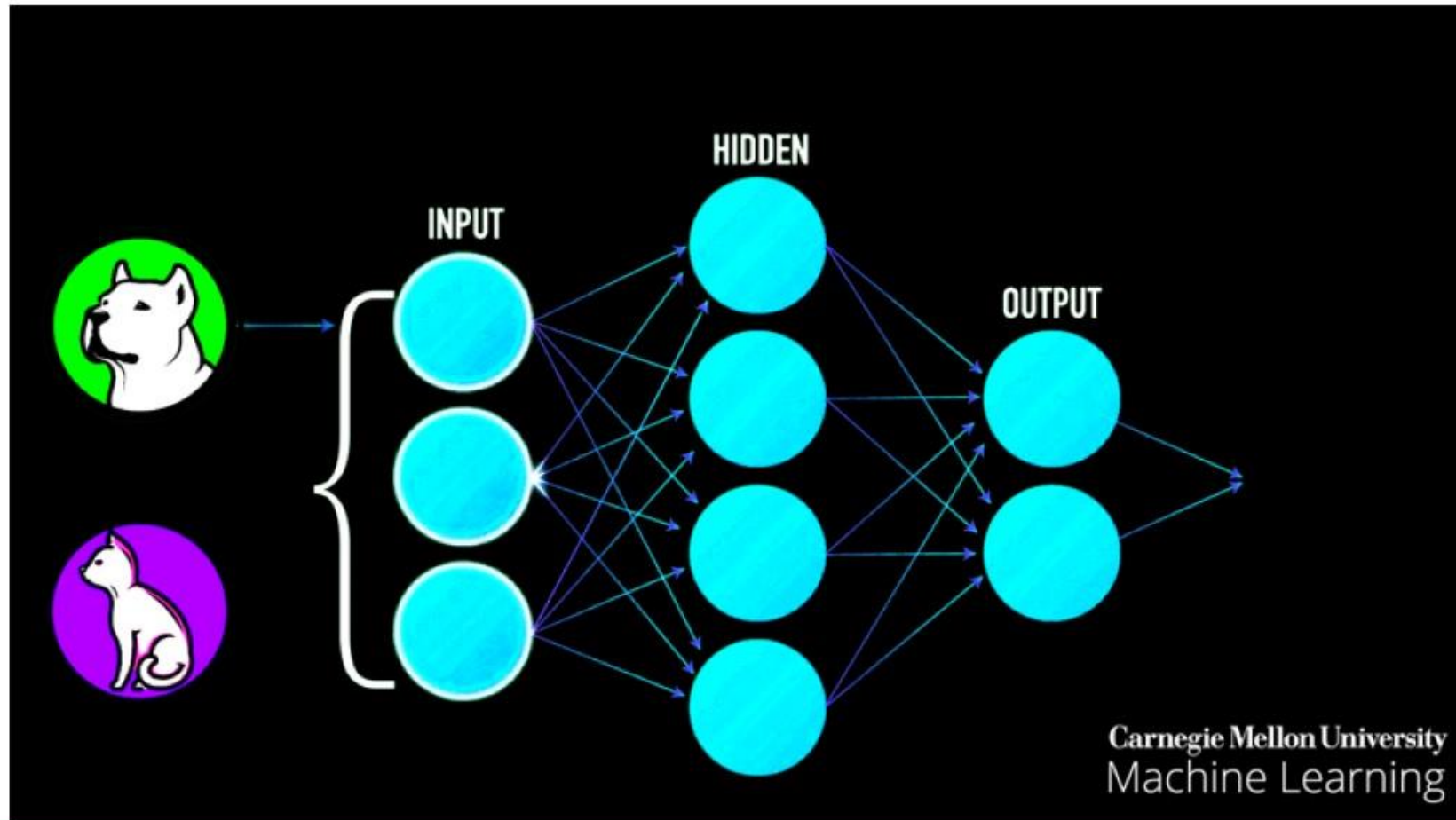
A little girl in a pink hat is blowing bubbles.



Object Detection

localizing and
classifying multiple
objects within images

- Image Classification



Neural Style Transfer

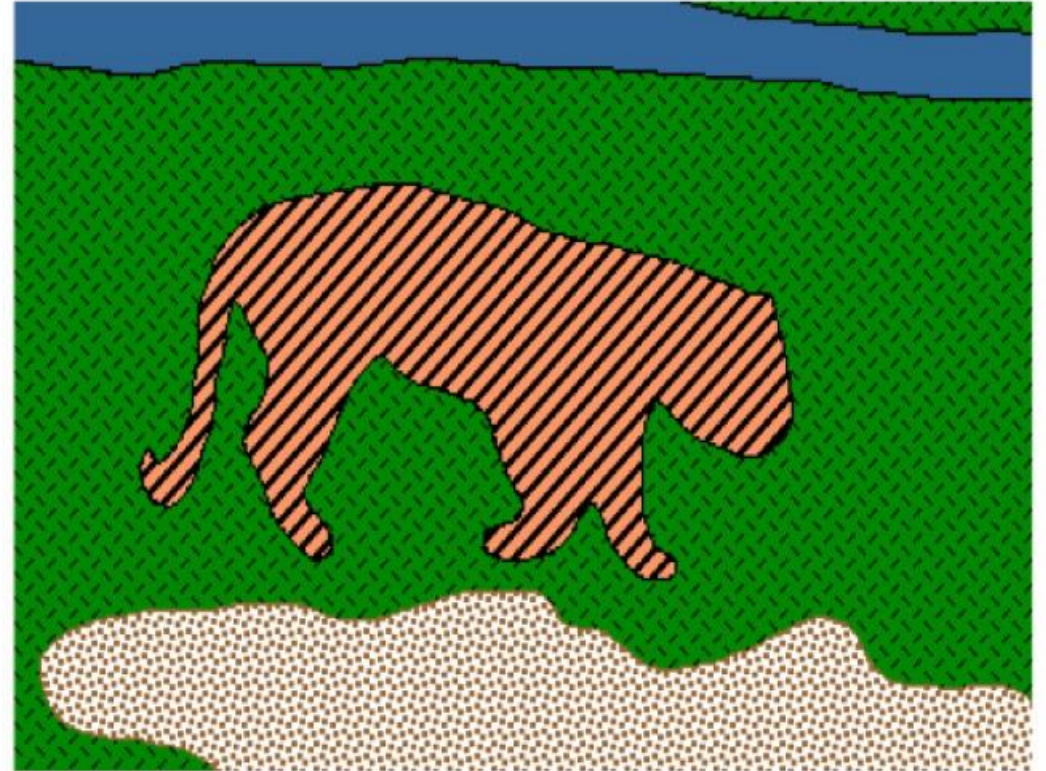
Art Generation and Style Transfer: CNNs can generate new artistic images or transform images to adopt the style of another image.



- **Facial Recognition:** CNNs are widely used in facial recognition systems.
- **Medical Image Analysis:** CNNs are valuable in medical imaging for tasks like tumor detection, disease classification, and segmentation of organs or structures within medical images.
- **Autonomous vehicles:** Analyzing real-time visual data for navigation and object detection. CNN can process sensor data, such as images from cameras or lidar scans, to detect and track objects, predict their motion, and make decisions for safe navigation

- **Video Analysis:** CNNs can analyze videos by processing frames over time. Applications include action recognition, video surveillance, and video captioning.
- CNN is used by **Google** for its image search function, **Facebook** for its automatic tagging algorithms, **Amazon** for its product suggestion functions.

- Image segmentation: Assigning a label to each pixel in an image.



- **Image Segmentation.** It is the process of dividing an image into different regions based on the characteristics of pixels.
- For instance,
 - Film making: The software behind green screens implements image segmentation to crop out the foreground and place it on a background for scenes that cannot be shot or would be dangerous to shoot in real life.
 - Image segmentation is also used to track objects in a **sequence of images** and to classify terrains, like petroleum reserves, in satellite images.
 - Some medical applications of segmentation include the identification of injured muscle, the measurement of bone and tissue, and the detection of suspicious structures to aid radiologists (**Computer Aided Diagnosis, or CAD**).

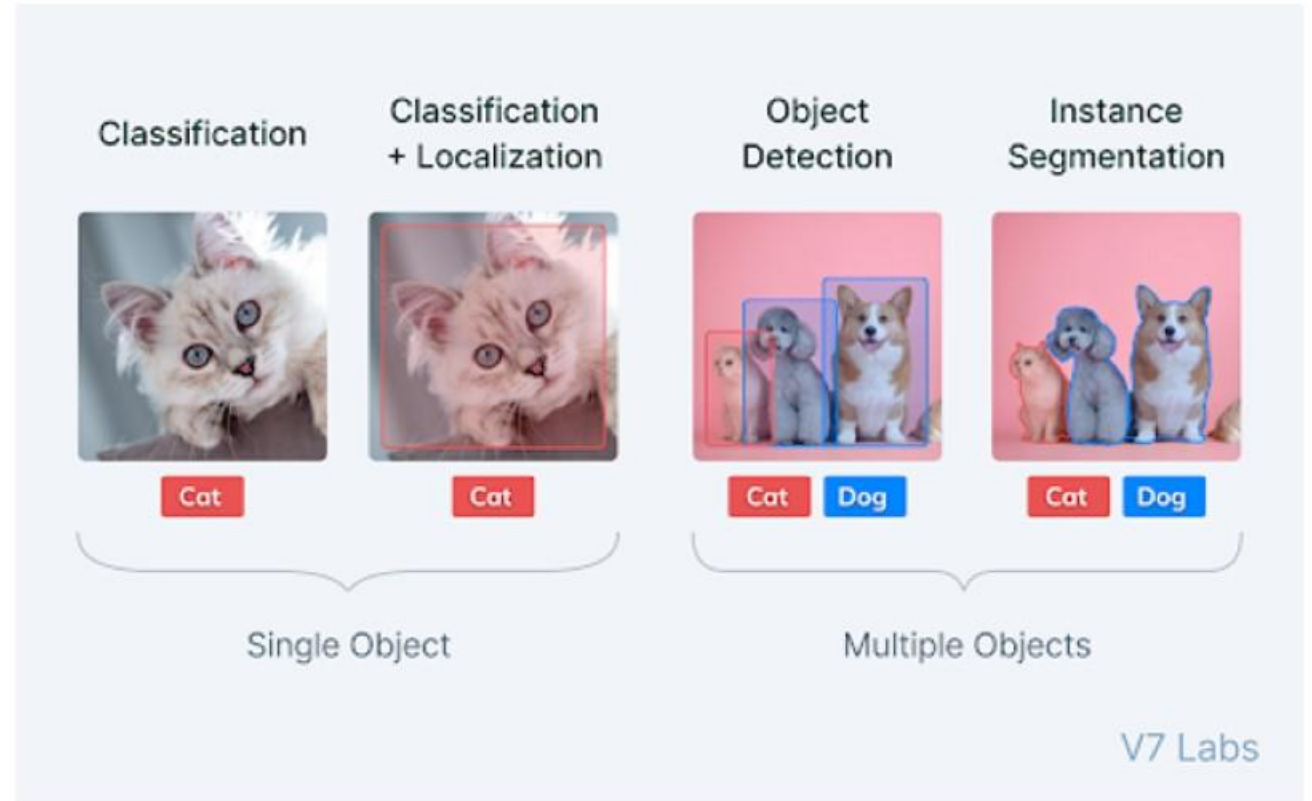
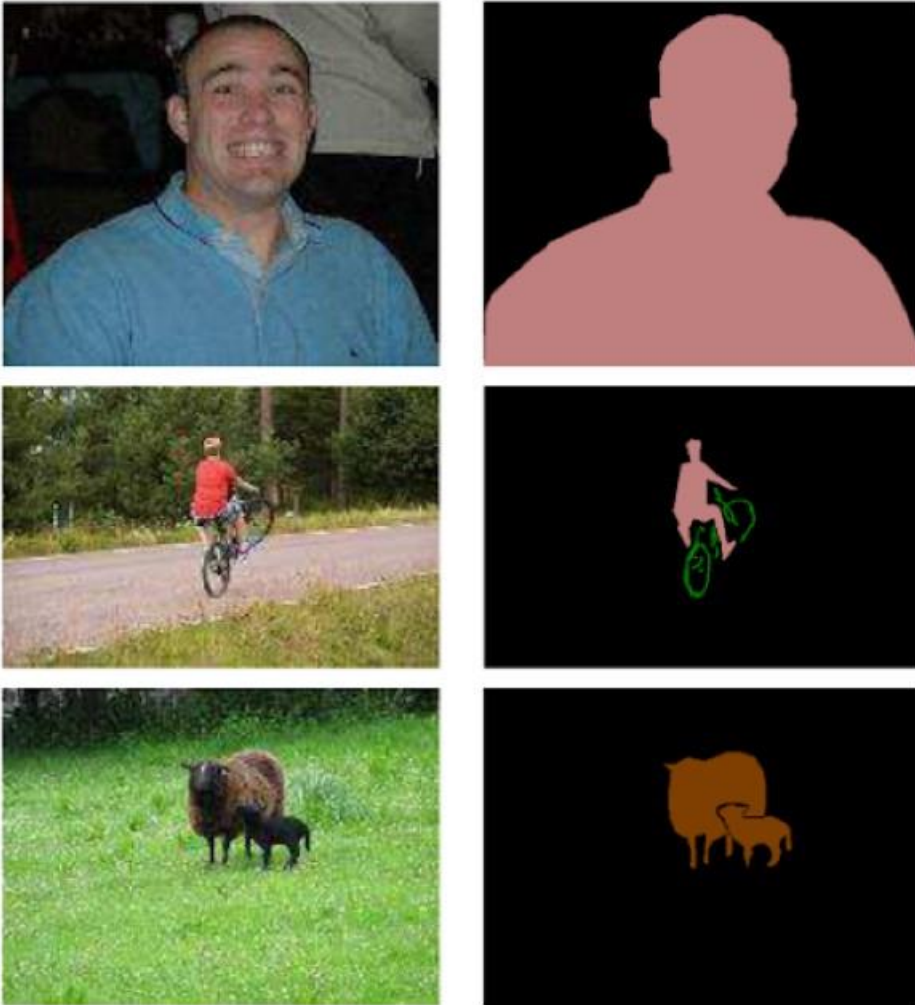
Image Segmentation.



Label any object in seconds with
Auto-Annotation

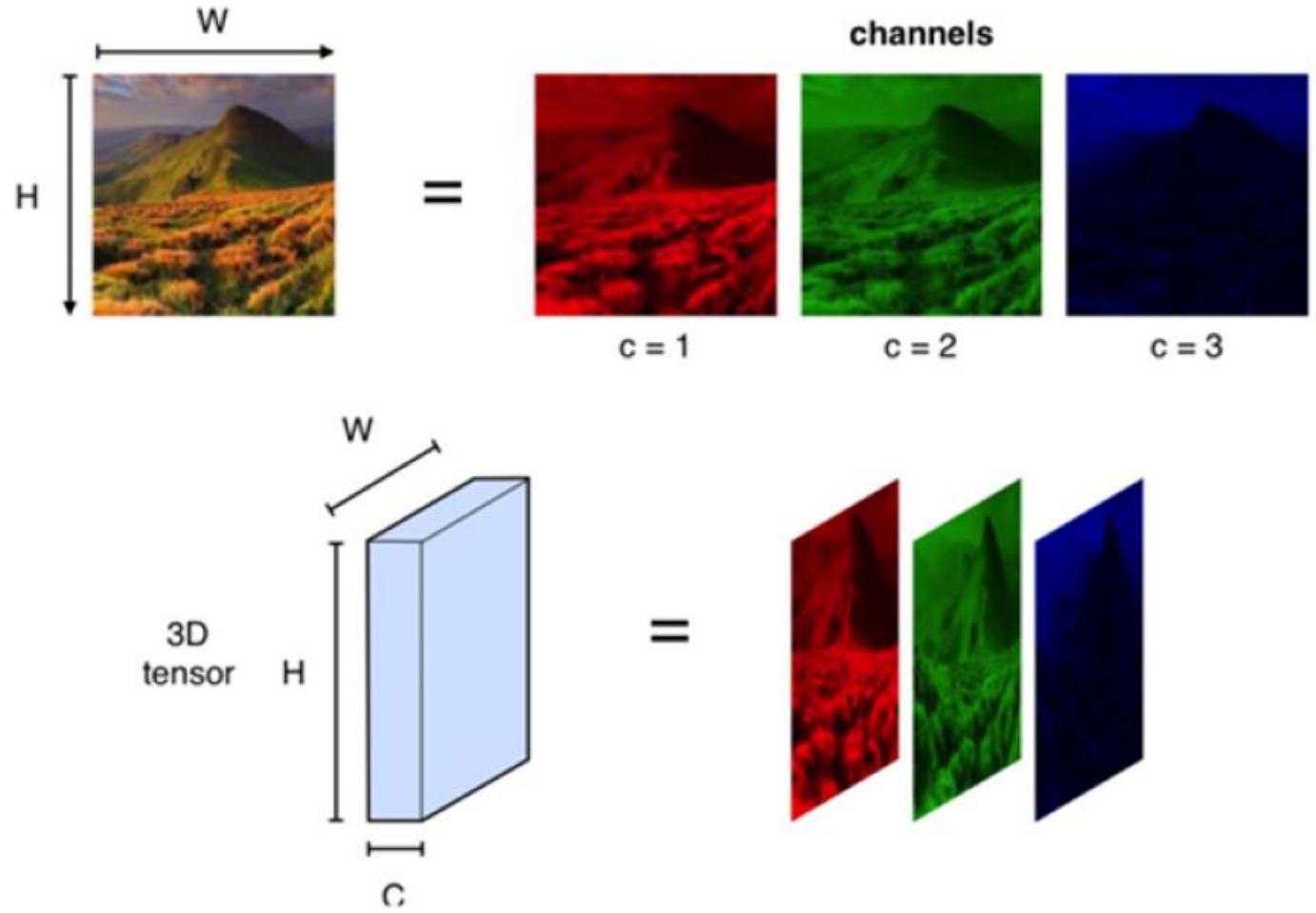
<https://www.v7labs.com/blog/image-segmentation-guide>

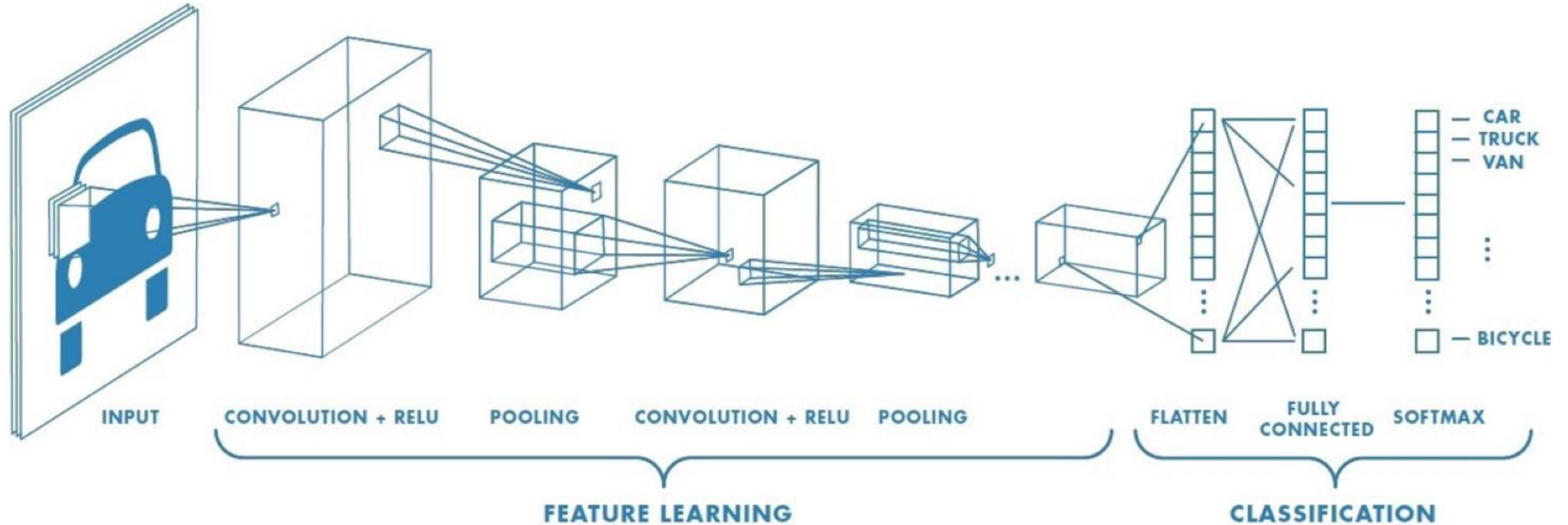
Applications of CNN



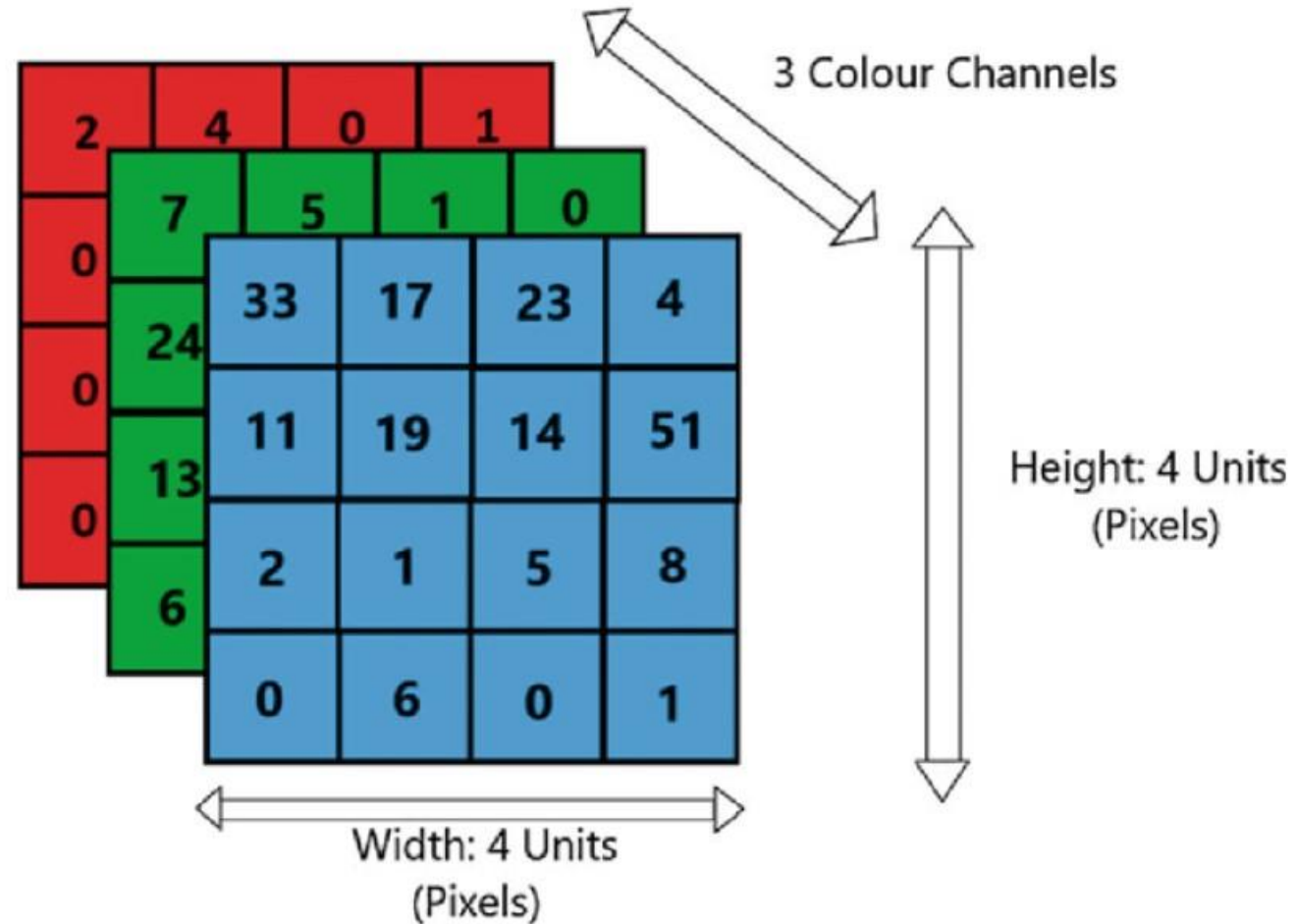
Input Layer

- The input to a convolution layer is a 3-dimensional array(**tensor**)





Input Layer(Tensor)



Input Layer

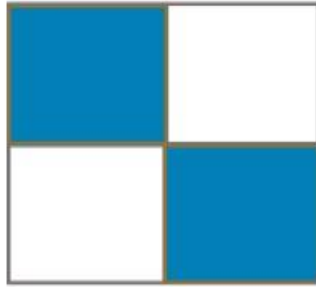


image for the symbol \

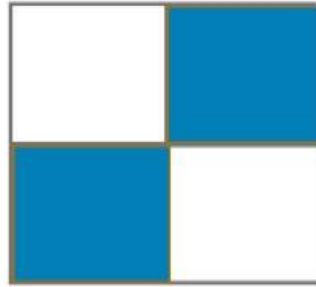
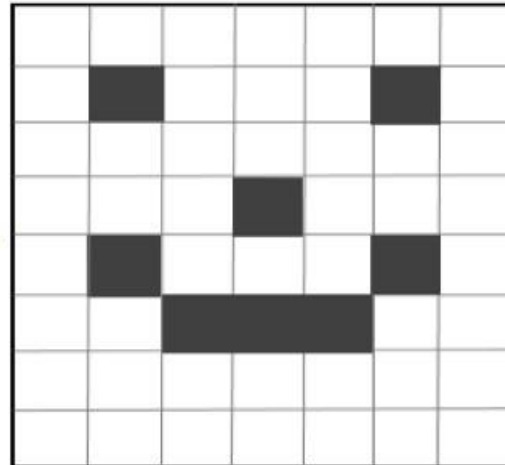


image for the symbol /



Real Image



Represented in the form of
black and white pixels

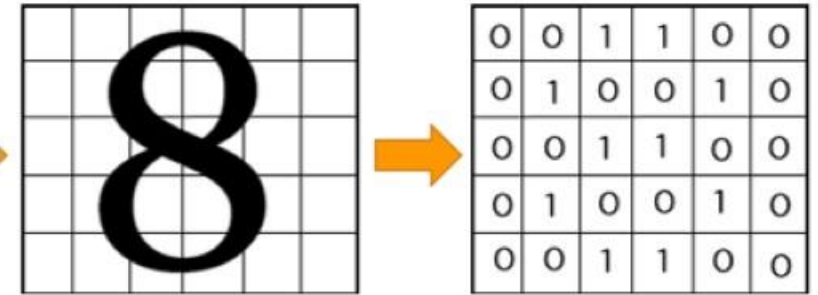
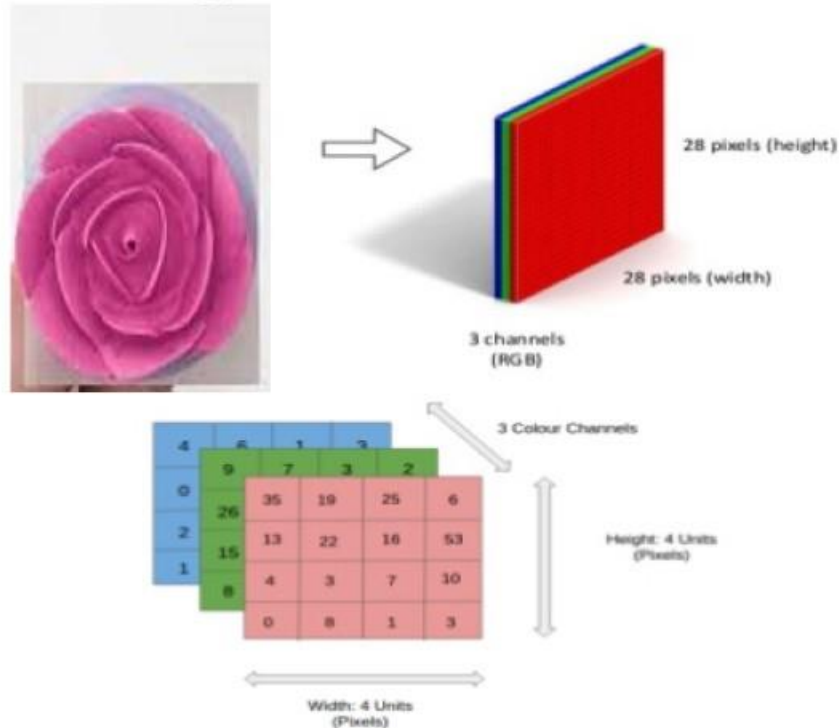


0	0	0	0	0	0	0
0	1	0	0	0	1	0
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	1	0	0	0	1	0
0	0	1	1	1	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

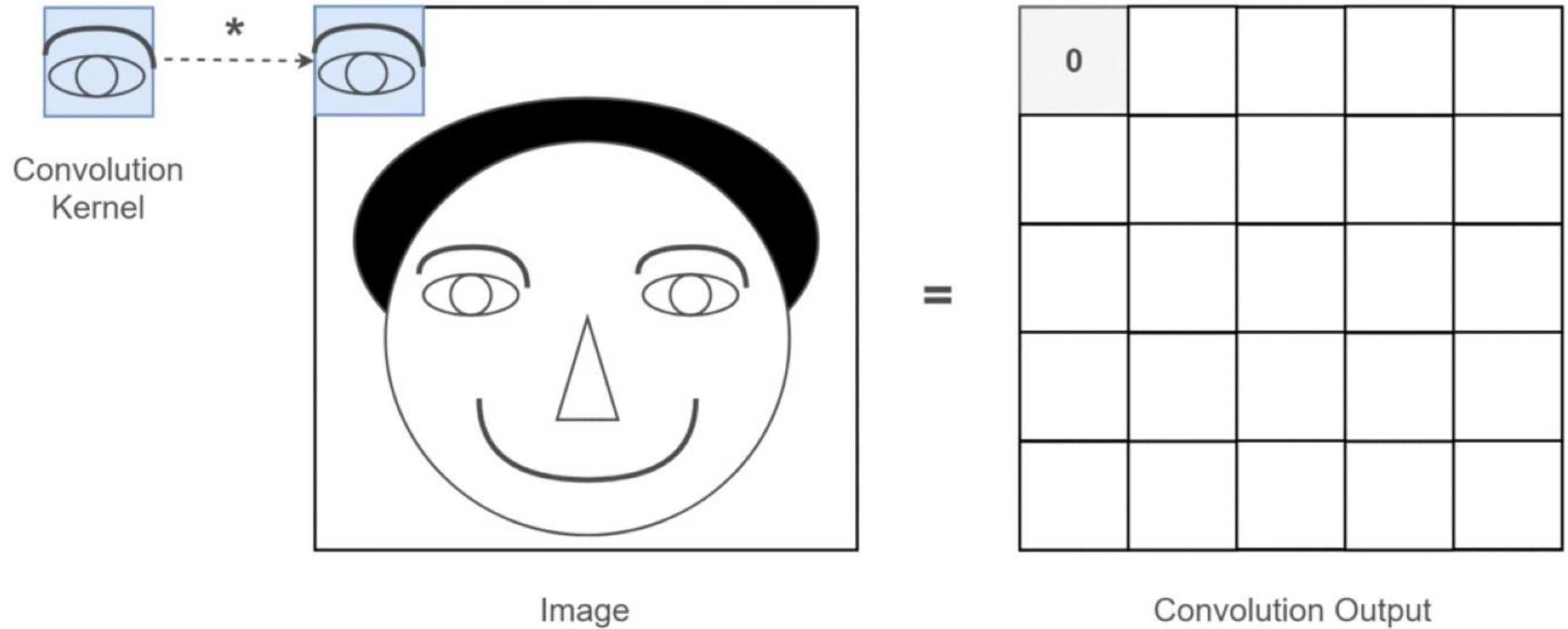
Image represented in the
form of a matrix of numbers

- Every image is represented in the form of an array of pixel values.

color image is 3rd-order tensor



Convolution Layer



- Data = [5,3,2,5,9,7]
- b = [1,2,3]
- In convolution operation, the arrays are **multiplied element-wise**, and the product is summed to create a new array, which represents $a*b$.
- The first three elements of the matrix a are multiplied with the elements of matrix b. The product is **summed** to get the result.
- The next three elements from the matrix a are multiplied by the elements in matrix b, and the product is **summed** up.
- This process continues until the convolution operation is complete.

Convolution Layer

$a * b$

Multiply the arrays
element wise

Sum the product

[5, 6, 6]

17

$a = [5, 3, 2, 5, 9, 7]$

$b = [1, 2, 3]$

$a * b = [17,]$

Convolution Layer

$a * b$

Multiply the arrays
element wise

Sum the product

$a = [5, 3, 2, 5, 9, 7]$

$b = [1, 2, 3]$

$[5, 6, 6]$

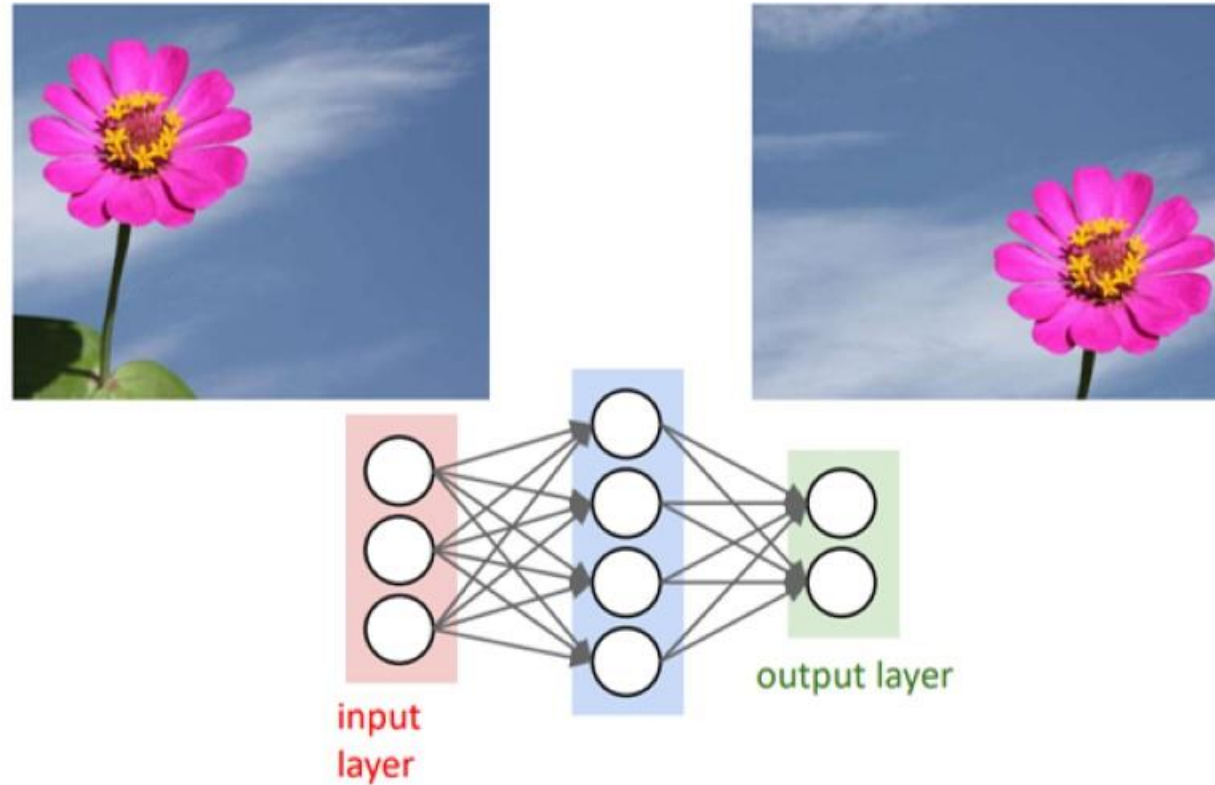
$[3, 4, 15]$

17

22

$a * b = [17, 22]$

Is there a flower in any of these images



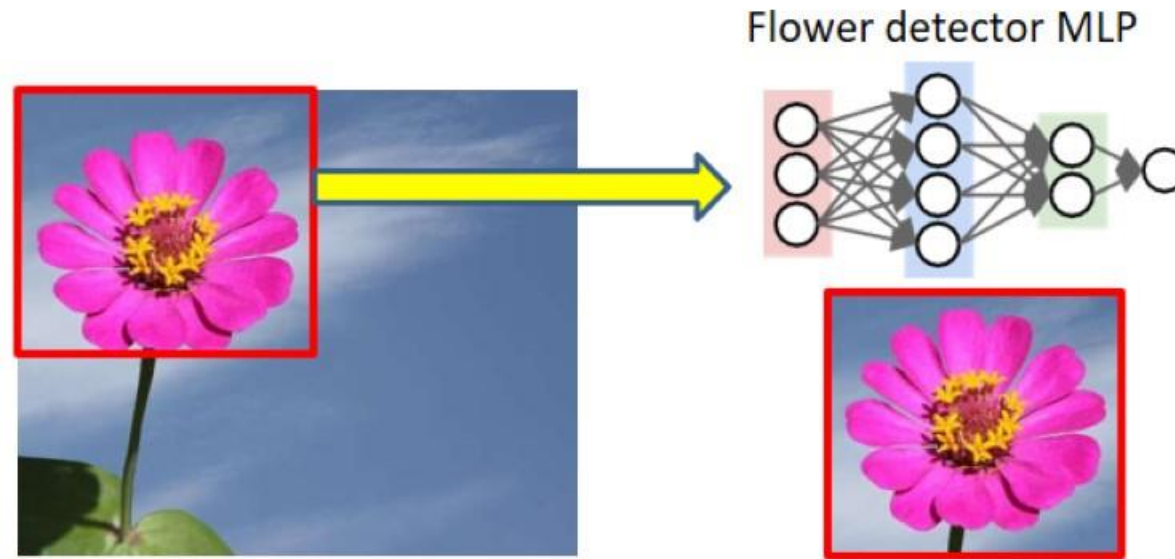
- Will an MLP that recognizes the left image as a flower also recognize the one on the right as a flower?

<https://deeplearning.cs.cmu.edu/S23/>

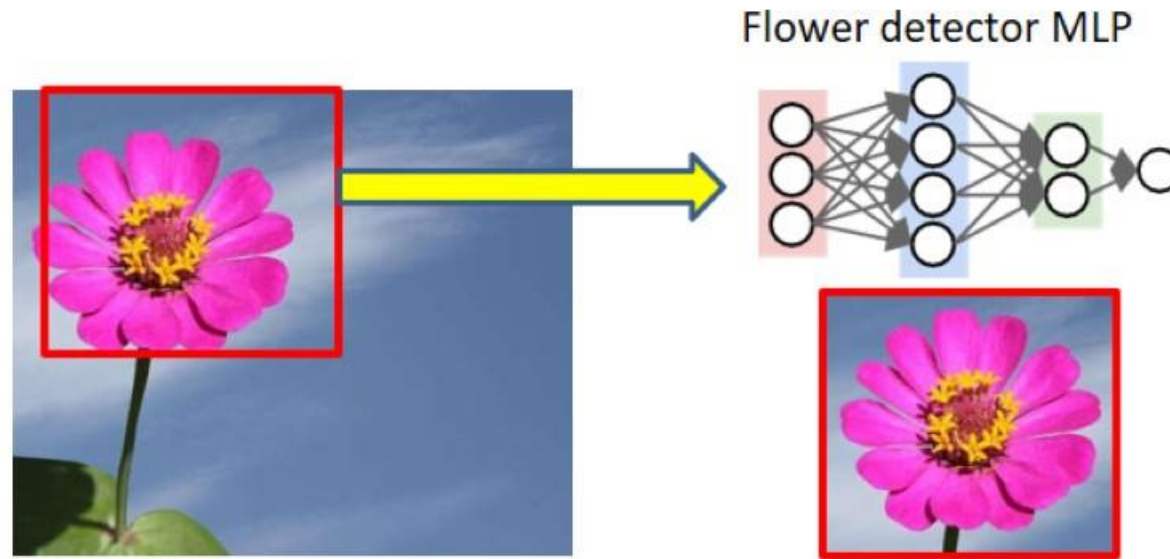
- Need a network that will “fire” regardless of the precise location of the target object
- The need for *shift invariance*
- In many problems the *location* of a pattern is not important
 - **Only the presence of the pattern**
- Conventional MLPs are sensitive to the location of the pattern
 - Moving it by one component results in an entirely different input that the MLP won’t recognize
- Requirement: Network must **be shift invariant**
- **Scan for the desired object – “Look” for the target object at each position**

<https://deeplearning.cs.cmu.edu/S23/>

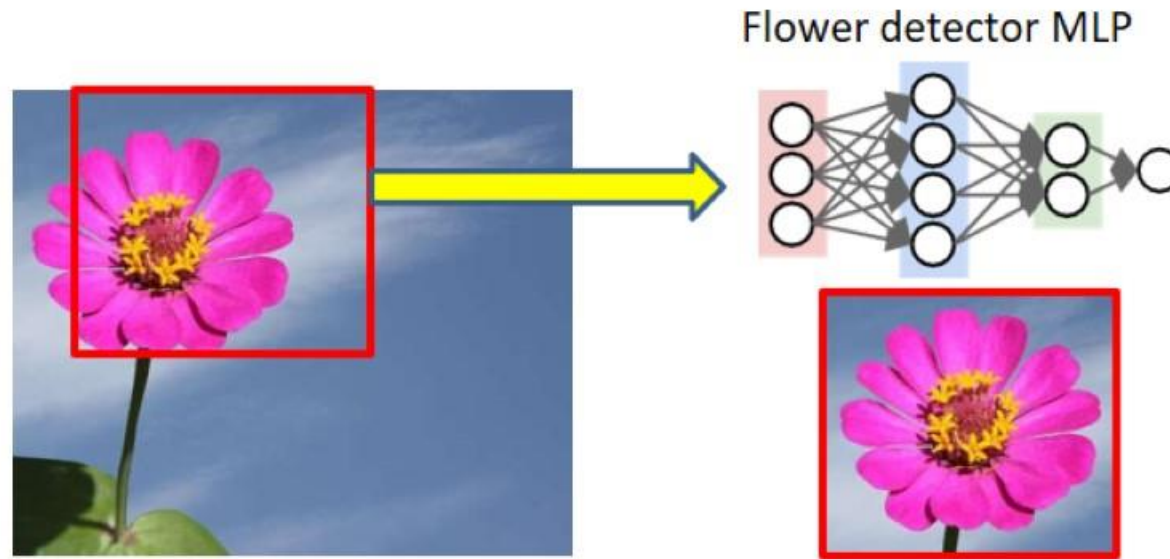
Solution: Scan



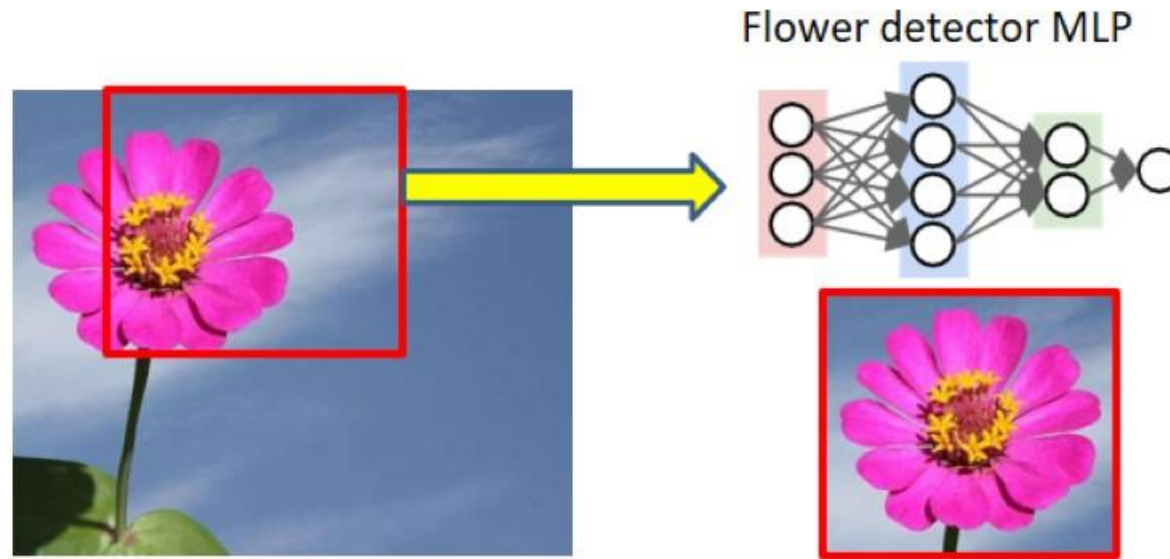
- *Scan* for the desired object
- At each location, the entire region is sent through the MLP



- *Scan* for the desired object
- At each location, the entire region is sent through the MLP

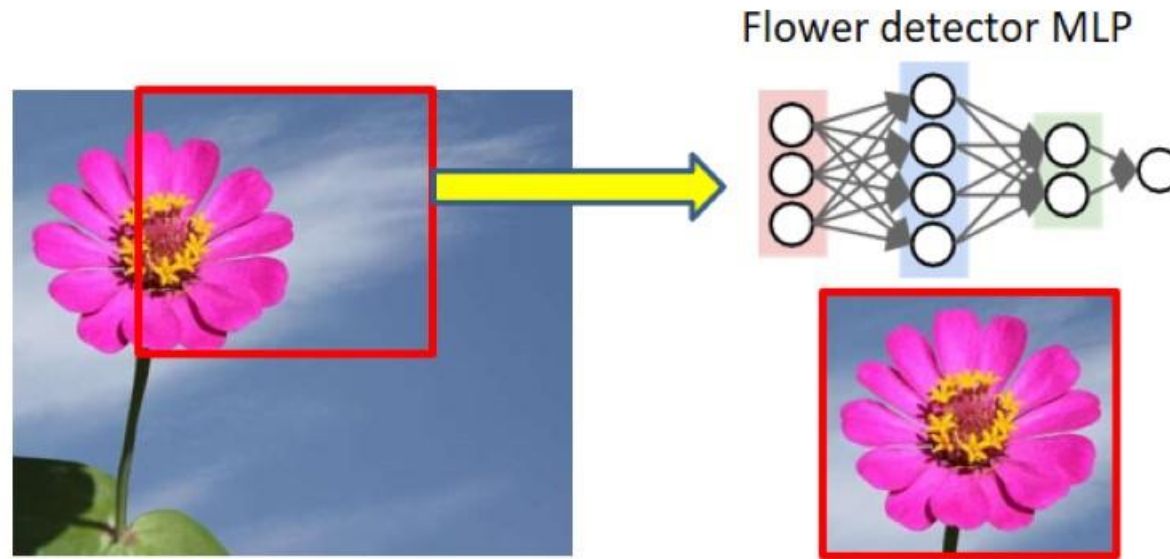


- *Scan* for the desired object
- At each location, the entire region is sent through the MLP

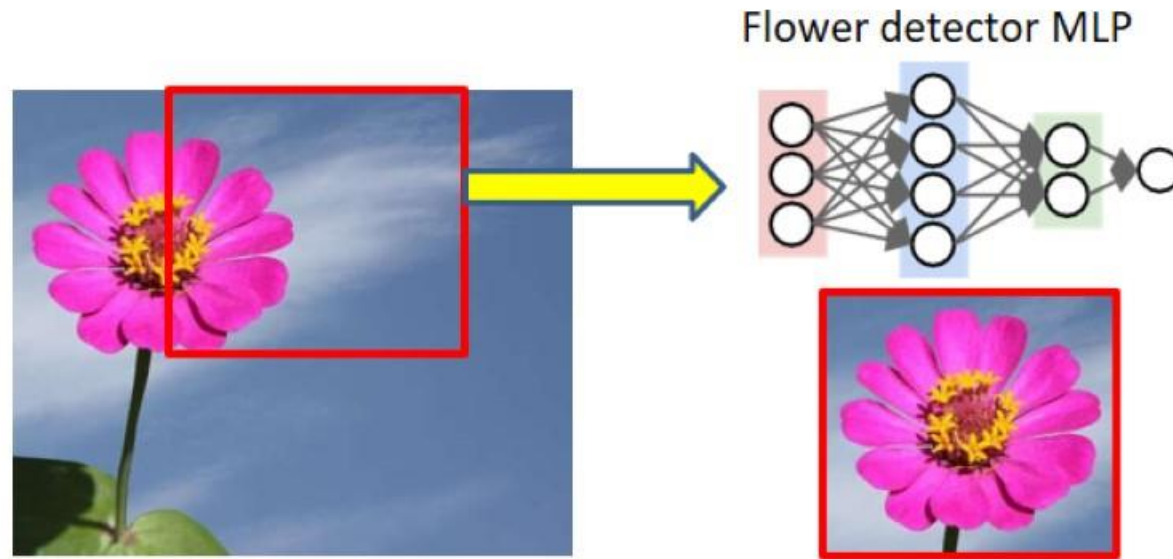


- *Scan* for the desired object
- At each location, the entire region is sent through the MLP

Solution: Scan



- *Scan* for the desired object
- At each location, the entire region is sent through the MLP



- *Scan* for the desired object
- At each location, the entire region is sent through the MLP

Convolution Layer

3 ¹	0 ⁰	1 ⁻⁰	2 ⁻⁰	7 ⁻⁰	4 ⁻¹
1 ¹	5 ⁰	8 ⁻⁰	9 ⁻⁰	3 ⁻⁰	1 ⁻¹
2 ¹	7 ⁰	2 ⁻⁰	5 ⁻⁰	1 ⁻⁰	3 ⁻¹
0 ¹	1 ⁰	3 ⁻⁰	1 ⁻⁰	7 ⁻⁰	8 ⁻¹
4	2	1	6	2	8
2	4	5	2	3	9

*

1	0	-1
1	0	-1
1	0	-1

=

-5	-4	0	8
-10	-2	2	3
0	-2	-4	-7
-3	-2	-3	-16

Vertical edge detection examples

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0



*

1	0	-1
1	0	-1
1	0	-1



=

0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0



0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10



*

1	0	-1
1	0	-1
1	0	-1



=

0	-30	-30	0
0	-30	-30	0
0	-30	-30	0
0	-30	-30	0



Convolution layer

Kernel/Filter, K,
represented in the color
yellow. We have selected **K**
as a **3x3x1** matrix.

1 _{x1}	1 _{x0}	1 _{x1}	0	0
0 _{x0}	1 _{x1}	1 _{x0}	1	0
0 _{x1}	0 _{x0}	1 _{x1}	1	1
0	0	1	1	0
0	1	1	0	0

Image

4		

Convolved
Feature

Convolution Layer

0	0	0	0	0	0	...
0	156	155	156	158	158	...
0	153	154	157	159	159	...
0	149	151	155	158	159	...
0	146	146	149	153	158	...
0	145	143	143	148	158	...
...	...	...	...	...	...	...

Input Channel #1 (Red)

0	0	0	0	0	0	...
0	167	166	167	169	169	...
0	164	165	168	170	170	...
0	160	162	166	169	170	...
0	156	156	159	163	168	...
0	155	153	153	158	168	...
...	...	...	...	...	...	...

Input Channel #2 (Green)

0	0	0	0	0	0	...
0	163	162	163	165	165	...
0	160	161	164	166	166	...
0	156	158	162	165	166	...
0	155	155	158	162	167	...
0	154	152	152	157	167	...
...	...	...	...	...	...	...

Input Channel #3 (Blue)

-1	-1	1
0	1	-1
0	1	1

Kernel Channel #1



308

1	0	0
1	-1	-1
1	0	-1

Kernel Channel #2



-498

0	1	1
0	1	0
1	-1	1

Kernel Channel #3



164

+

+

+ 1 = -25

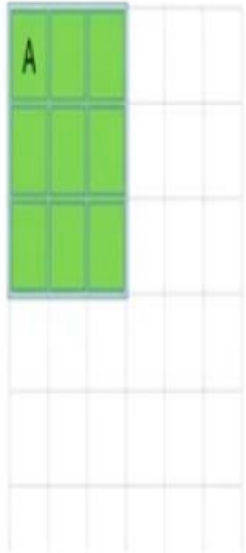
Bias = 1

Output

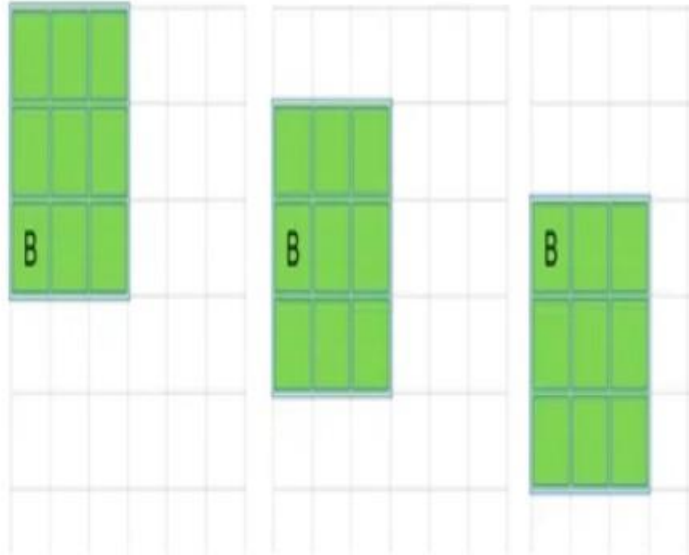
-25				...
				...
				...
				...
...	...	...	...	...

Padding

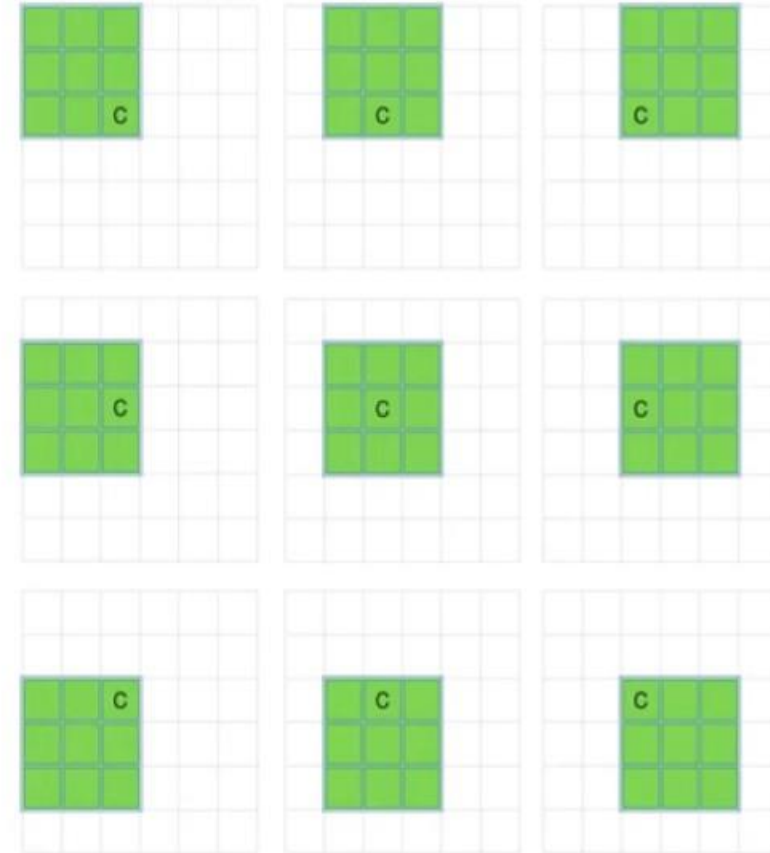
Corner Pixel



Edge Pixel



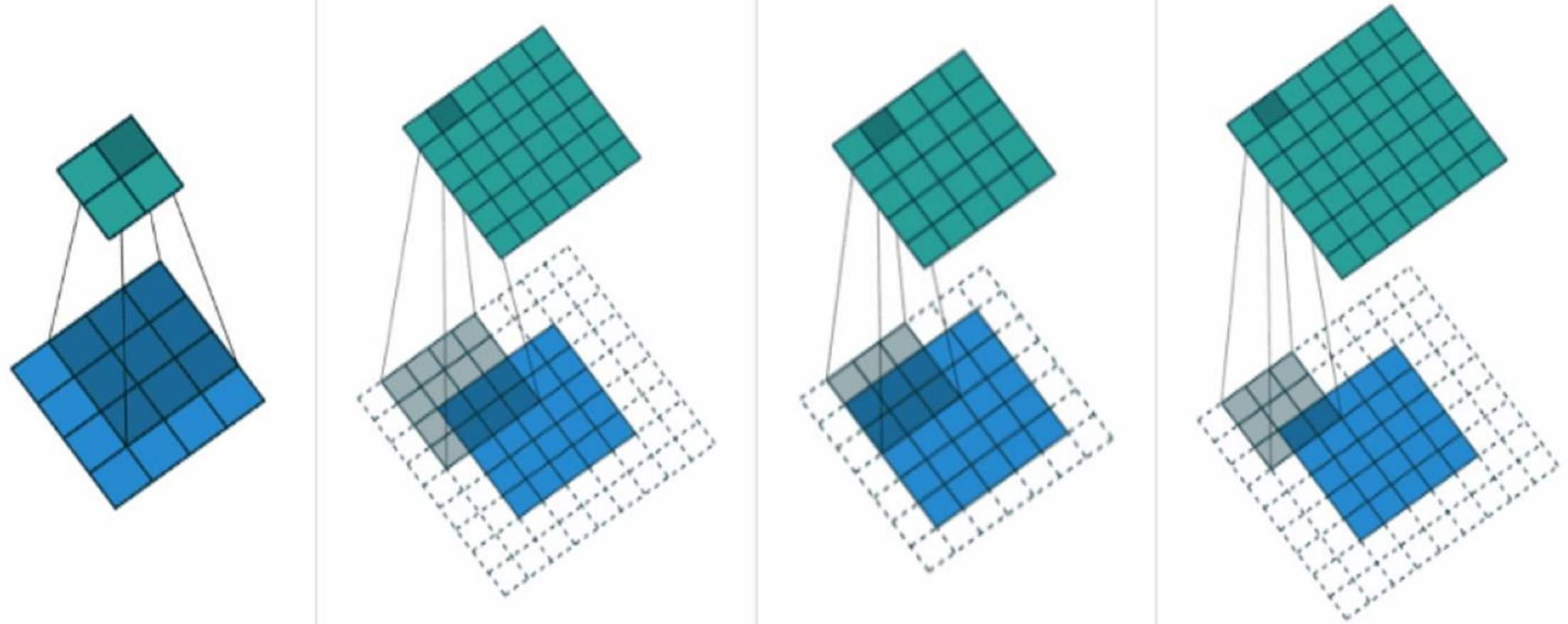
Middle Pixel



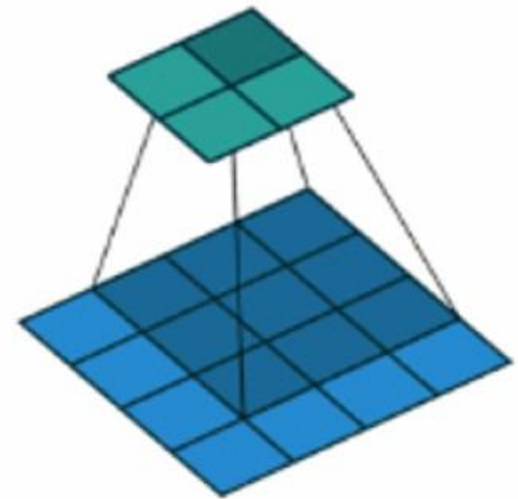
- Every time we apply a convolutional operation, the size of the **image shrinks**
- Pixels present in the **corner** of the image are used only a few number of times during convolution as compared to the central pixels. Hence, we do not focus too much on the corners since that can lead to information loss
- To overcome these issues, we can pad the image with an **additional border**, i.e., we add one pixel **all around the edges**.

- Padding is a technique used to preserve the spatial dimensions of the input image after convolution operations on a feature map. Padding involves adding extra pixels around the border of the input feature map before convolution.
 - **Valid Padding:** In the valid padding, no padding is added to the input feature map, and the output feature map is smaller than the input feature map. This is useful when we want to reduce the spatial dimensions of the feature maps.
 - **Same Padding:** In the same padding, padding is added to the input feature map such that the size of the output feature map is the same as the input feature map. This is useful when we want to preserve the spatial dimensions of the feature maps.

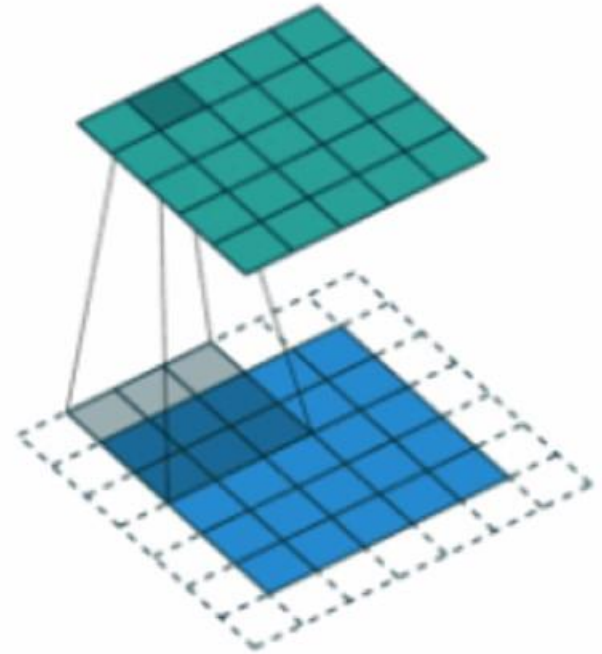
Padding



- The input image has 4x4 pixels and the filter has 3x3. There is **no padding**, which is called '*valid*.'
- The result becomes 2x2 pixels data **$N-F+1$**
 $(4-3+1 = 2)$.
- Output data is **downsized**



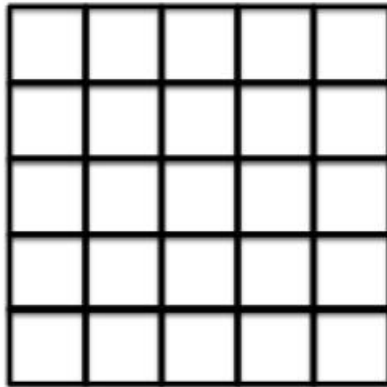
- **Same Padding:** There is **one layer padding**. The input image has 5x5 pixels and the filter has 3x3.
- So the result gets 5x5 pixels ($N+2*P-F+1$)
- ($5 + 1*2 - 3 + 1 = 5$)
- **Same** size as the input image.
- We can even make the outcome bigger than the input data, but the **Valid** and **Same** cases are used the most.



Same Padding

Padding is to add **extra pixels** outside the image. And zero padding means every pixel value that you add is zero.

Image

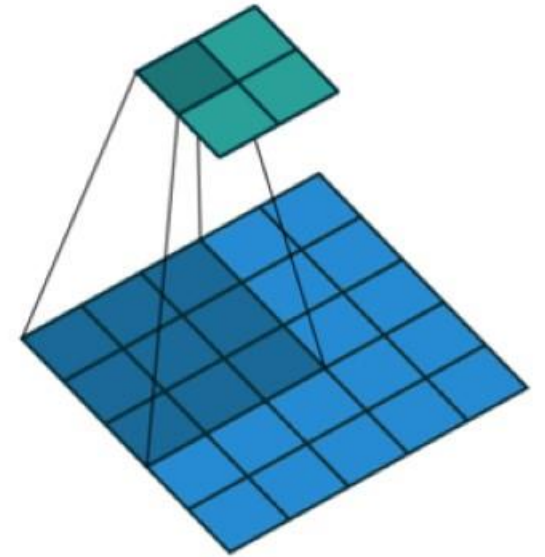


Image

0	0	0	0	0	0	0
0						0
0						0
0						0
0						0
0						0
0	0	0	0	0	0	0

<https://medium.com/machine-learning-algorithms/what-is-padding-in-convolutional-neural-network-c120077469cc>

- In the context of convolutional neural networks (CNNs), the term "**stride**" refers to the **step size** by which the **filter/kernel** moves across the input feature map during the convolution operation. It determines the amount of spatial shift between successive applications of the filter.



Padding

- We have seen that convolving an input of 6 X 6 dimension with a 3 X 3 filter
- *Input: $n \times n$*
- *Filter size: $f \times f$*
- *Output: $(n - f + 1) \times (n - f + 1)$*

Solution

- results in 4 X 4 output.

- Suppose we choose a stride of 2. So, while convoluting through the image, we will take two steps – both in the horizontal and vertical directions separately. The dimensions for stride s will be:
- Input: $n \times n$
- Padding: p
- Stride: s
- Filter size: $f \times f$
- Output: $[(n + 2p - f)/s + 1] \times [(n + 2p - f)/s + 1]$
- Stride helps to **reduce the size of the image**.

Advantages of Stride:

Computational Efficiency: Using a larger stride reduces the number of operations required to process an input image. By skipping pixels, the computational cost can be significantly reduced, making it faster to process large datasets.

Dimension Reduction: Applying stride in convolutional layers can reduce the spatial dimensions of the output feature maps. This reduction can be desirable in certain cases, such as when dealing with large input sizes or when aiming to downsample the data.

Advantages of Stride:

Translation Invariance: Stride can improve the translation invariance property of convolutional neural networks (CNNs). Translation invariance means that the model's predictions are unaffected by the position of objects within the input. By using stride, the model can capture the same features at different locations, enhancing its ability to generalize across different positions.

Drawbacks of Stride:

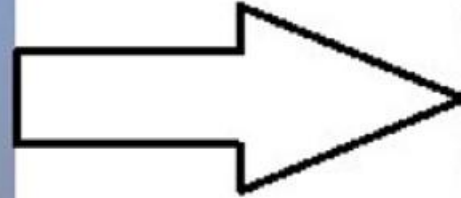
Information Loss: When using a larger stride, some spatial information is inevitably lost

Reduced Spatial Resolution: Stride can cause a reduction in the spatial resolution of the output feature maps. This reduction can be disadvantageous in tasks where preserving detailed spatial information is important, such as object detection or image segmentation.

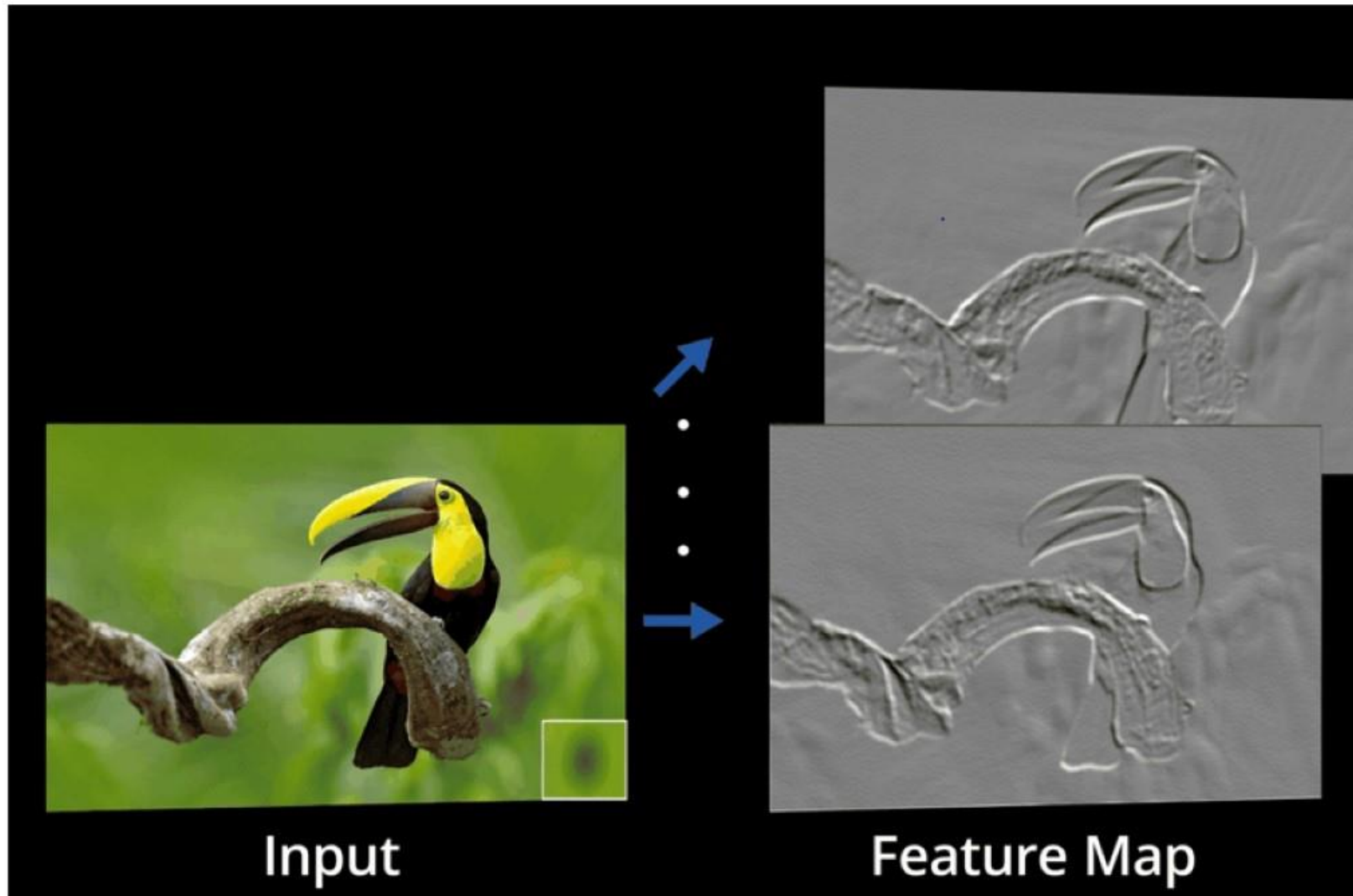
Overlooking Small Objects: If the stride is too large, the receptive field of the model may become too coarse, making it difficult to detect or classify small objects within the input.

Convolution layer

- extracting valuable features from an image



Convolution layer



- Once we get an output after convolving over the entire image using a **filter**, we add a bias term to those outputs and finally **apply an activation function to generate activations**. This is one layer of a convolutional network. Recall that the equation for one forward pass is given by:

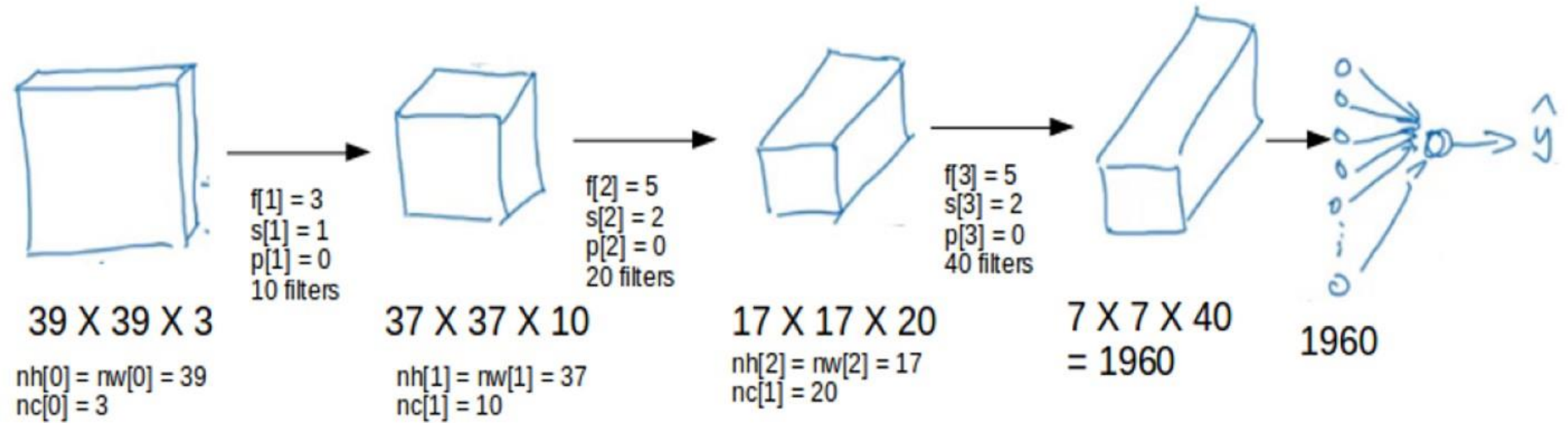
- $z[1] = w[1] * a[0] + b[1]$

- $a[1] = g(z[1])$

One Layer of a Convolutional Network

- input $(6 \times 6 \times 3)$ is $a^{[0]}$ and filters $(3 \times 3 \times 3)$ are the weights $w^{[1]}$. These activations from layer 1 act as the input for layer 2, and so on.
- Clearly, the number of parameters in case of convolutional neural networks is independent of the size of the image. It **essentially depends on the filter size**.
- Suppose we have **10 filters**, each of shape $3 \times 3 \times 3$. What will be the number of parameters in that layer?
 - Number of parameters for each filter $= 3 \times 3 \times 3 = 27$
 - There will be a **bias term** for each filter, so total parameters per filter $= 28$
 - As there are 10 filters, the total parameters for that layer $= 28 \times 10 = 280$

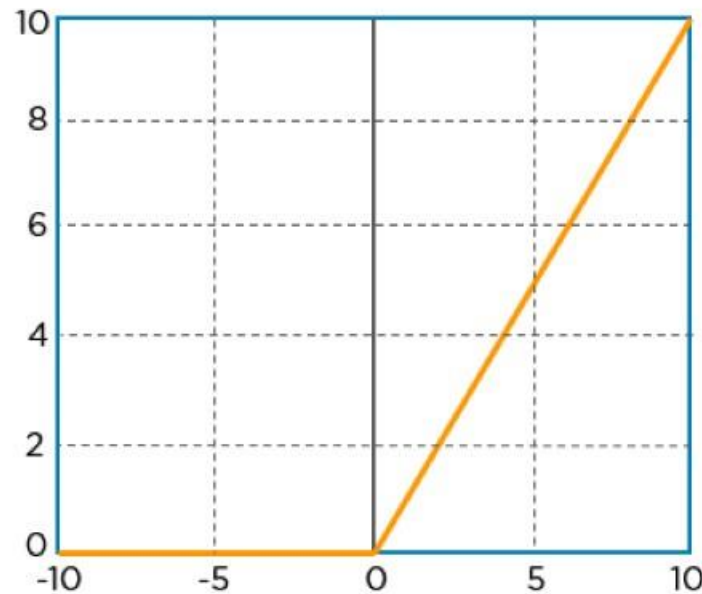
One Layer of a Convolutional Network



$$\begin{aligned} nh[1] &= rw[1] = (n + 2p - f) / s + 1 \\ nh[1] &= rw[1] = (39 + 0 - 3) / 1 + 1 \\ nh[1] &= rw[1] = 37 \end{aligned}$$

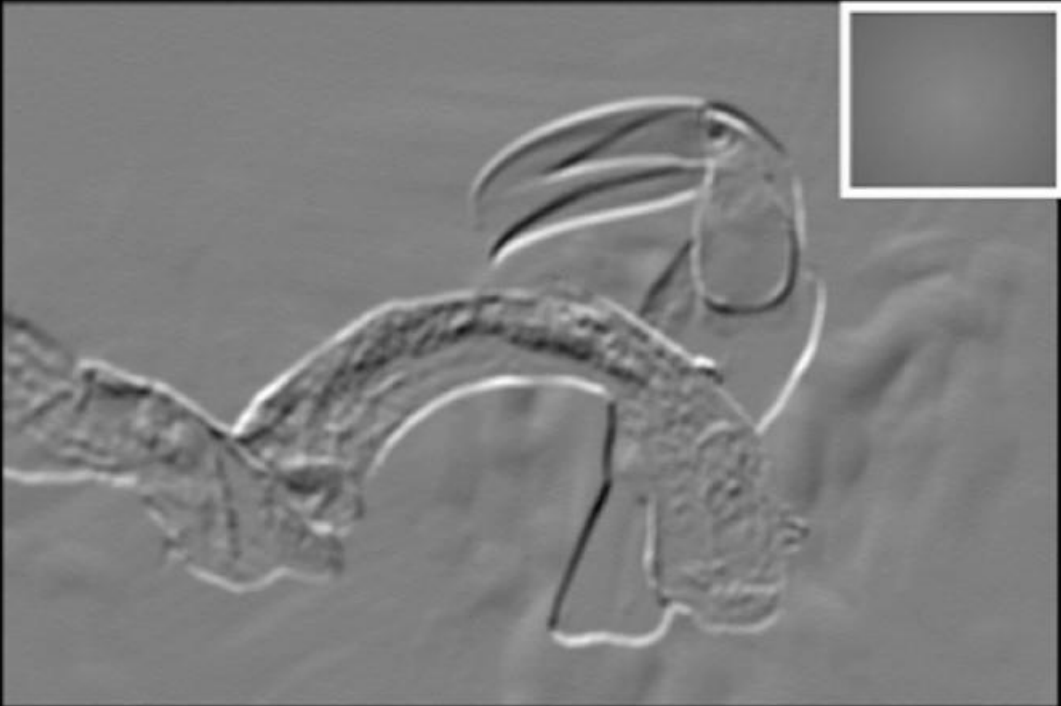
- $f[l]$ = filter size
- $p[l]$ = padding
- $s[l]$ = stride
- $n_{[c]}^{[l]}$ = number of filters

- ReLU stands for the rectified linear unit.
- ReLU performs an element-wise operation and sets all the negative pixels to 0.



$$R(z) = \max(0, z)$$

Input Feature Map

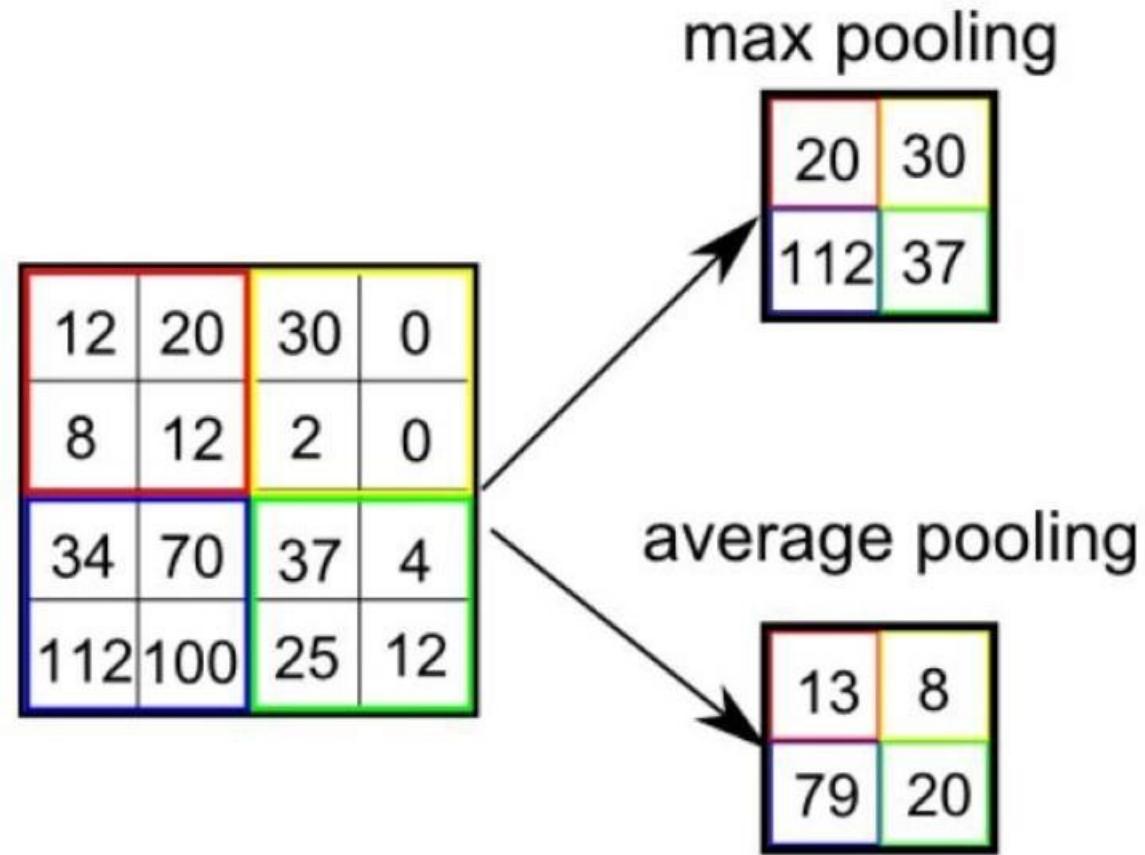


- Pooling layer is responsible for reducing the spatial size of the Convolved Feature. This is to decrease the computational power required to process the data through dimensionality reduction. Furthermore, it is useful for extracting dominant features which are rotational and positional invariant, thus maintaining the process of effectively training of the model.

3.0	3.0	3.0
3.0	3.0	3.0
3.0	2.0	3.0

3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1

Pooling Layer

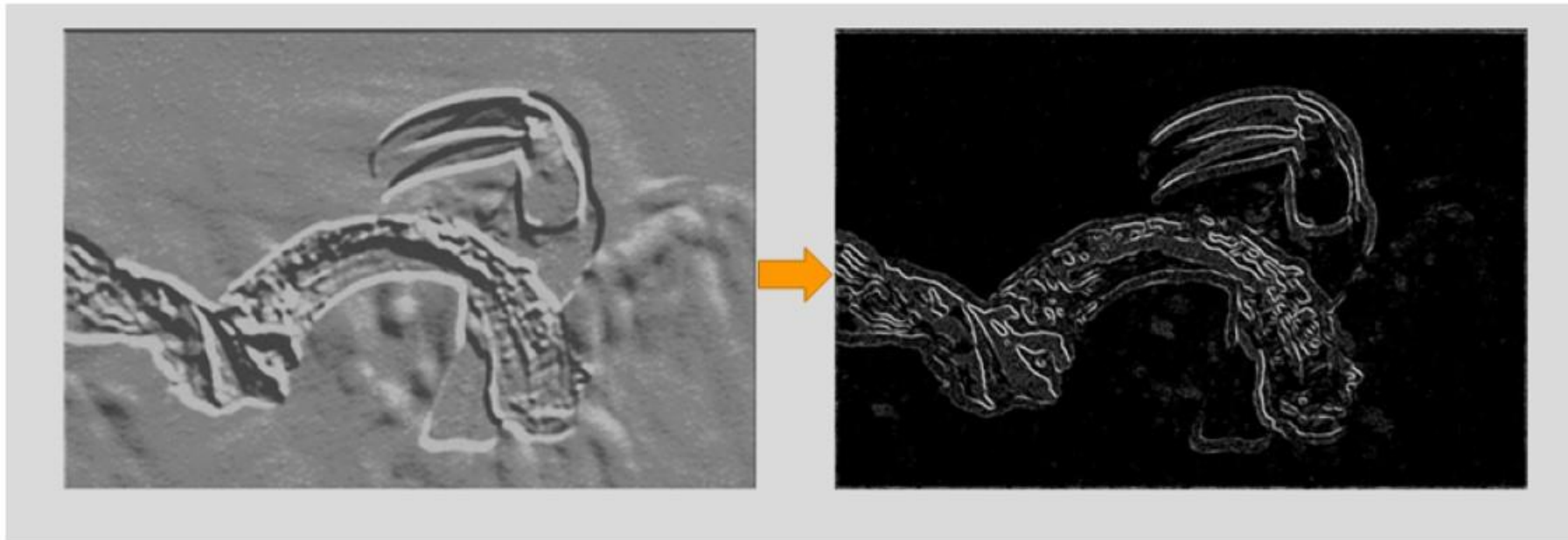


There are two types of Pooling:

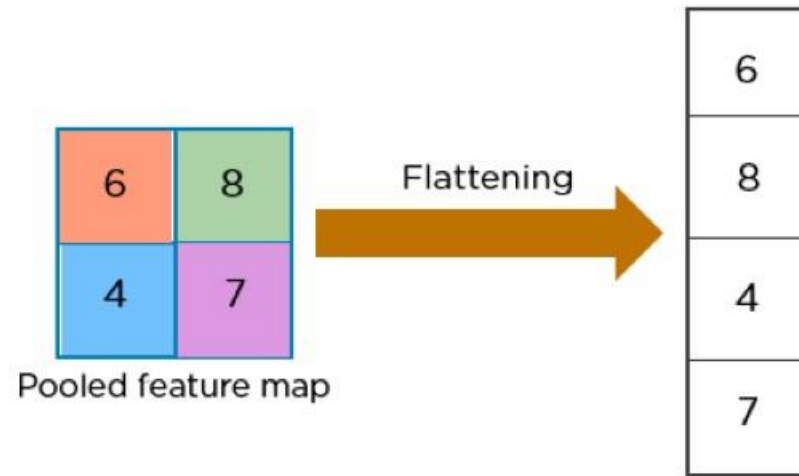
- **Max Pooling:** returns the maximum value from the portion of the image covered by the Kernel. down sample the feature maps
- **Average Pooling.** Average Pooling returns the average of all the values from the portion of the image covered by the Kernel.
- Max Pooling also performs as a **Noise Suppressant**. It discards the noisy activations altogether and also performs de-noising along with dimensionality reduction. On the other hand, Average Pooling simply performs dimensionality reduction as a noise suppressing mechanism. Hence, we can say that **Max Pooling performs a lot better than Average Pooling.**

Pooling Layer

- The pooling layer uses various filters to identify different parts of the image like edges, corners, body, feathers, eyes, and beak.



- Flattening is used to convert all the resultant 2-Dimensional arrays from pooled feature maps into a single long continuous linear vector.

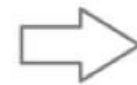


- Adding a Fully-Connected layer is a (usually) cheap way of learning non-linear combinations of the high-level features as represented by the output of the convolutional layer. The Fully-Connected layer is learning a possibly non-linear function in that space.

Why ConvNets over Feed-Forward Neural Nets?

- why not just flatten the image (e.g. 3x3 image matrix into a 9x1 vector) and feed it to a Multi-Level Perceptron for classification purposes?
- in cases of extremely basic binary images, the method might show an average precision score while performing prediction of classes but would have little to no accuracy when it comes to complex images having pixel dependencies throughout.
- Translation invariance of CNN :if the input data is shifted or translated, a translation-invariant model should still be able to recognize and classify the underlying patterns correctly.

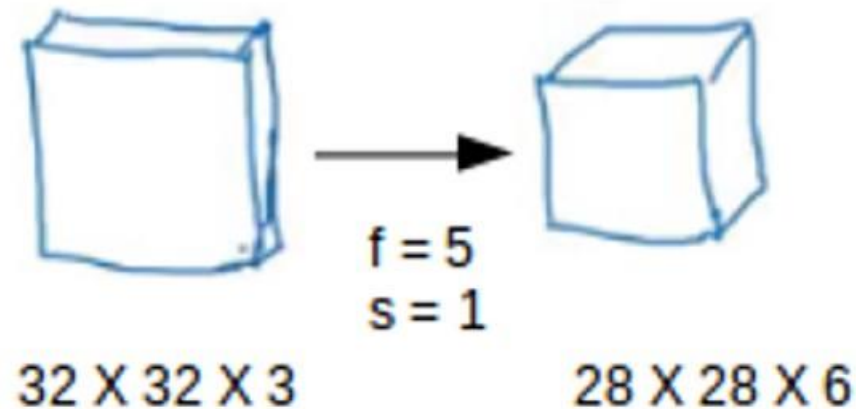
1	1	0
4	2	1
0	2	1



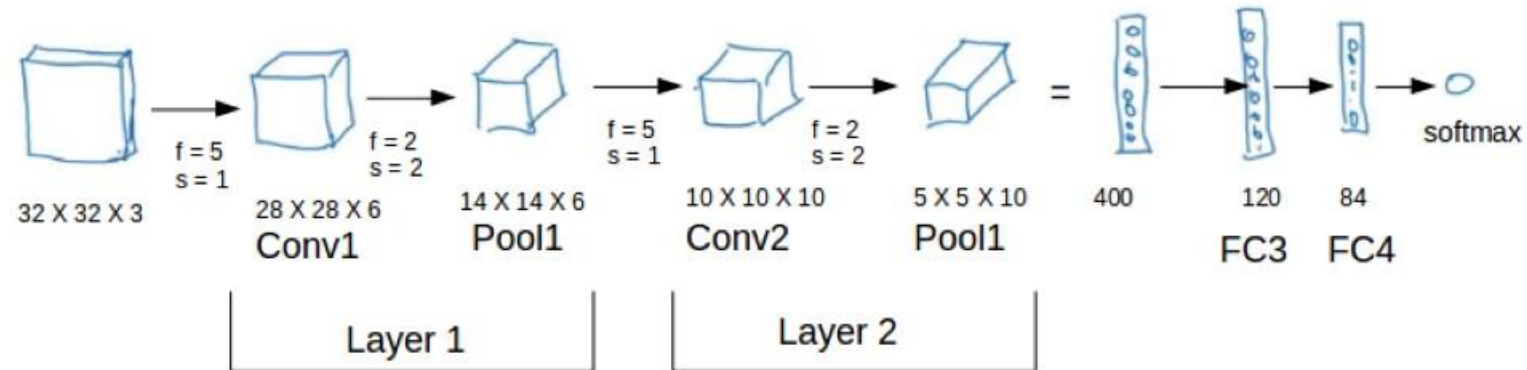
1
1
0
4
2
1
0
2
1

Why Convolutions?

- There are primarily two major advantages of using convolutional layers over using just fully connected layers:
 1. Parameter sharing
 2. Sparsity of connections



CNN Example

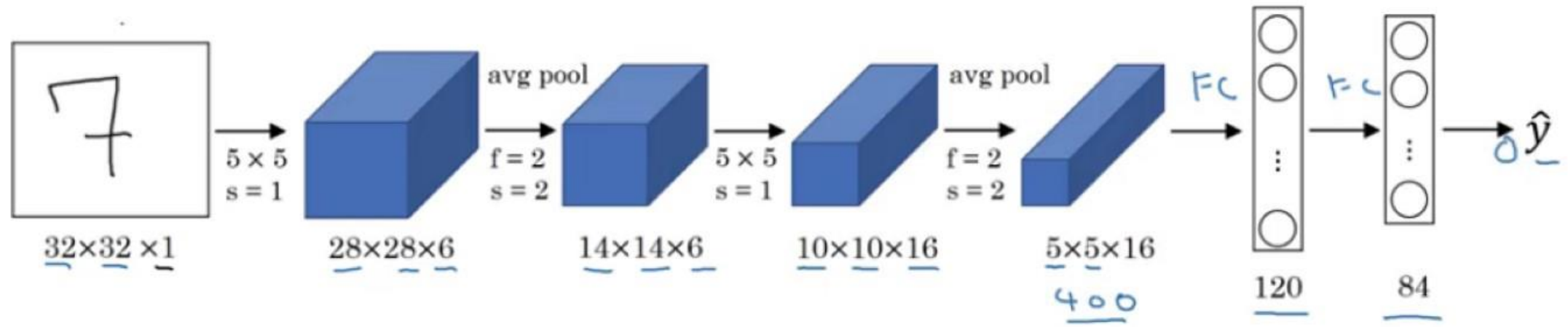


Why Convolutions?

- If we would have used just the fully connected layer, the number of parameters would be $= 32*32*3*28*28*6$, which is nearly equal to 14 million! Makes no sense, right?
- If we see the number of parameters in case of a convolutional layer, it will be $= (5*5 + 1) * 6$ (if there are 6 filters), which is equal to 156. Convolutional layers reduce the number of parameters and speed up the training of the model significantly.
- In convolutions, we share the parameters while convolving through the input. The intuition behind this is that a feature detector, which is helpful in one part of the image, is probably also useful in another part of the image. So a single filter is convolved over the entire input and hence the parameters are shared.
- The second advantage of convolution is the sparsity of connections. For each layer, each output value depends on a small number of inputs, instead of taking into account all the inputs.

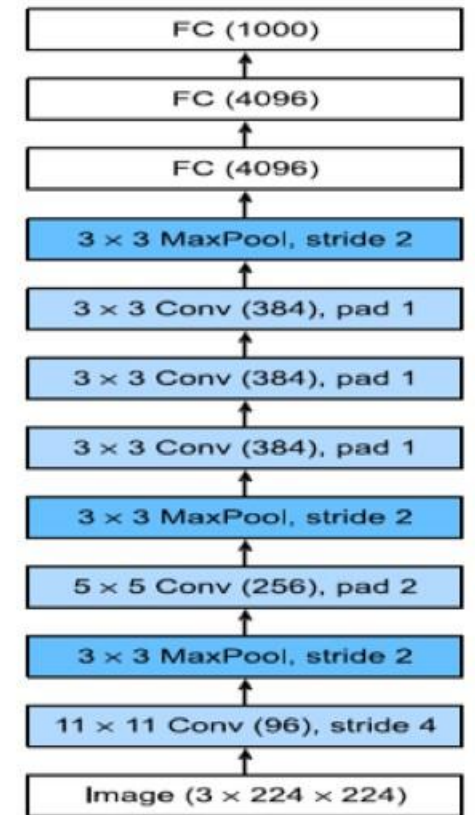
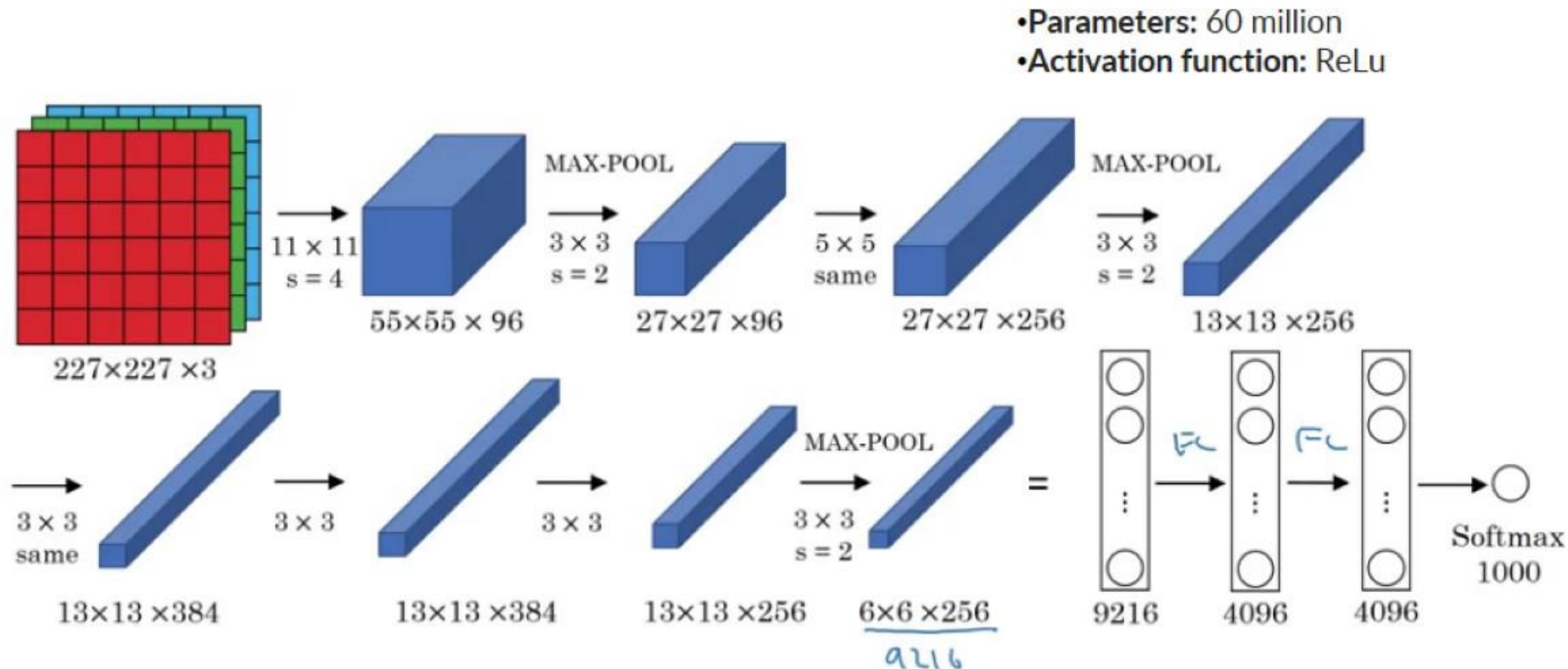
There are various architectures of CNNs :

1. LeNet
2. AlexNet
3. VGGNet
4. GoogLeNet
5. ResNet
6. ZFNet
7. Siamese Network
8. RasNet
9. DenseNet



- **Parameters:** 60k
- **Layers flow:** Conv -> Pool -> Conv -> Pool -> FC -> FC -> Output
- **Activation functions:** Sigmoid/tanh and ReLu

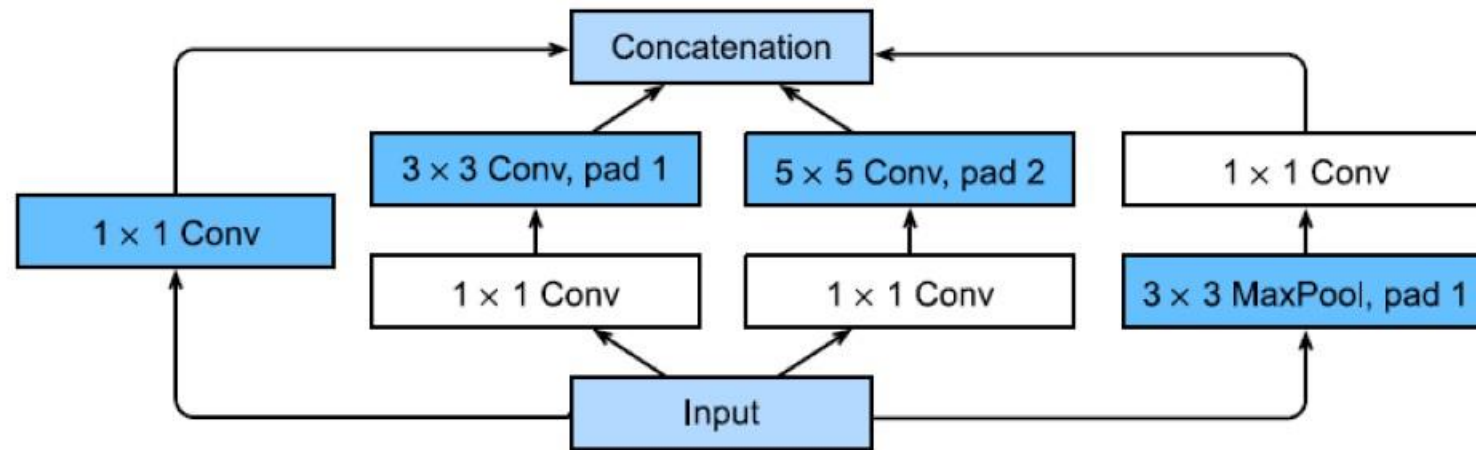
- AlexNet, which employed an 8-layer CNN, won the ImageNet Large Scale Visual Recognition Challenge 2012 by a phenomenally large margin.

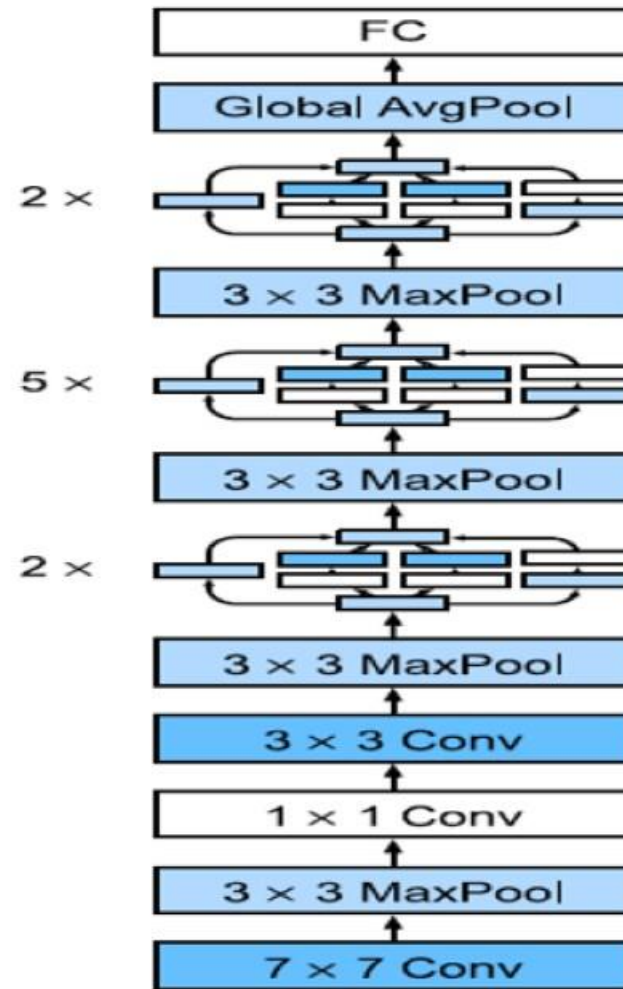


-
- The diagram illustrates the VGG-16 architecture, showing the flow of data through various layers. It starts with an input image of size $224 \times 224 \times 3$. This is followed by two convolutional layers, each with 64 filters of size 3×3 , resulting in a feature map of size $224 \times 224 \times 64$. This is then followed by a pooling layer, resulting in a feature map of size $112 \times 112 \times 64$. This is followed by two more convolutional layers, each with 128 filters of size 3×3 , resulting in a feature map of size $112 \times 112 \times 128$. This is then followed by a pooling layer, resulting in a feature map of size $56 \times 56 \times 128$. This is followed by three convolutional layers, each with 256 filters of size 3×3 , resulting in a feature map of size $56 \times 56 \times 256$. This is then followed by a pooling layer, resulting in a feature map of size $28 \times 28 \times 256$. This is followed by three convolutional layers, each with 512 filters of size 3×3 , resulting in a feature map of size $28 \times 28 \times 512$. This is then followed by a pooling layer, resulting in a feature map of size $14 \times 14 \times 512$. This is followed by three convolutional layers, each with 512 filters of size 3×3 , resulting in a feature map of size $14 \times 14 \times 512$. This is then followed by a pooling layer, resulting in a feature map of size $7 \times 7 \times 512$. This is followed by two fully connected layers, each with 4096 units, resulting in a feature map of size 4096×1 . Finally, a Softmax layer with 1000 units is used for classification.

The diagram illustrates the VGG architecture. On the left, a 'VGG block' is shown, which consists of two identical stages. Each stage contains a '3 x 3 Conv, pad 1' layer followed by a '2 x 2 MaxPool, stride 2' layer. On the right, the full 'VGG' network is shown. It starts with two 'VGG block' units, followed by three fully connected (FC) layers: 'FC (4096)', 'FC (4096)', and 'FC (1000)'. Arrows indicate the flow of data from bottom to top.

- The basic convolutional block in GoogLeNet is called an Inception block, likely named due to a quote from the movie Inception (“We need to go deeper”), which launched a viral meme.





- In the context of convolutional neural networks (CNNs), the terms "kernel" and "filter" are often used interchangeably, but they refer to slightly different concepts.
- a **kernel** is a **small matrix of weights used for convolution**
- a **filter** is a **collection of kernels** used in a layer of a CNN to extract multiple features from the input data.

LAB 1

Dr. Shahira Ahmed

LAB 2

Dr. Shahira Ahmed



LECTURE SUMMARY

- ◉ Convolutional Neural Networks (CNNs) are a type of deep learning model commonly used for image recognition and computer vision tasks.
- ◉ CNNs utilize specialized layers, such as convolutional layers, pooling layers, and fully connected layers, to extract features from input images and make predictions.



LECTURE SUMMARY

- Convolutional layers apply a set of learnable filters to the input image, enabling the network to capture local patterns and spatial dependencies.
- Pooling layers reduce the spatial dimensions of the input, helping to extract invariant features and improve computational efficiency.



LECTURE SUMMARY

- ◉ Fully connected layers connect every neuron in one layer to every neuron in the next layer, allowing the network to learn high-level representations and make predictions based on the extracted features.
- ◉ various architectures of CNNs
- ◉ Lab 1
- ◉ Lab 2

*Thank
you*

