Convolutional Neural Networks

Hamidreza Baradaran Kashani

Deep Neural Networks

List of Contents

Image Processing

- **MNIST Dataset:** a large dataset of 28x28 grayscale images of hand-written digits. Each image depicts one of the numbers 0 through 9.

- Using deep learning, we can take a data-driven approach to training an algorithm that discover patterns that distinguish one number from another.

Image Processing

- **How computers interpret images?** Any gray scale image is interpreted by a computer as an array.

- Gray scale images are 2D and RGB images are 3D arrays.

- Each grid cell is called pixel. Each pixel has a numerical value.

- Images in the MNIST dataset are understood by a computer as a 28x28 array.

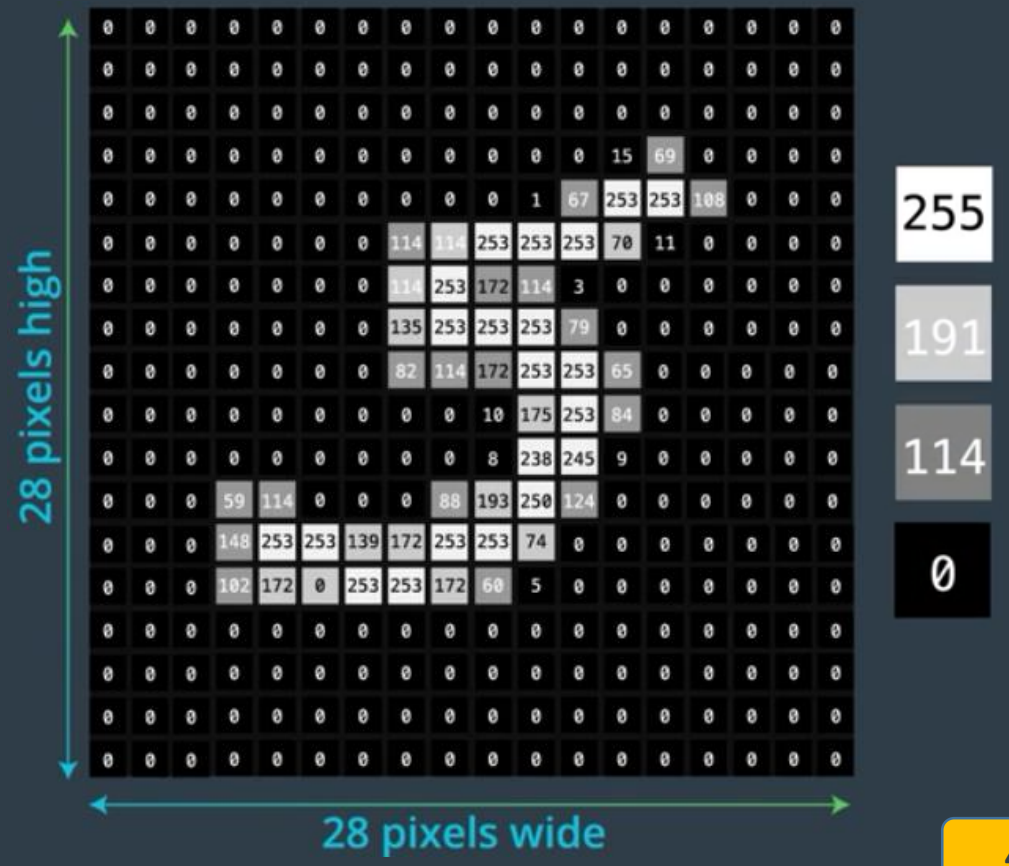


Image Processing

- In a gray scale image, white pixels are encoded as the value 255, and black pixels are encoded as 0. Gray pixels fall somewhere in between. This value is called intensity.

- In fact the intensity is a measure of light and dark, can be used to detect the shape of objects in an image.
  - ✓ As a pixel gets lighter, it's value is being closer to 255.

- For example when we try to distinguish a person from a background in an image, we can look at the contrast that occurs where the person ends and the background begins.

Image Processing

- Contrasts define a shape boundary that separates the objects.
- Abrupt changes in intensity show the edges of an object.
  - ✓ Changes from a dark to light area
  - ✓ Changes from a light to dark area
- Specific image filters can detect big changes in intensity and show the edges.
- Color images have similar numerical representations for each pixel color.
  - ✓ For example in a RGB image, there are 3 pixel colors called channels. Each channel can have a value between 0 and 256.

Image Processing

- **Pre-processing the images:** each image should be rescaled so that each image has values in a range from 0 to 1.

- To go from a range [0-255] to [0-1], we should divide every pixel value by 255. This step called normalization.

- Normalizing the pixel values helps the gradient calculations stay consistent and not get so large that they slow down or prevent a network from training.

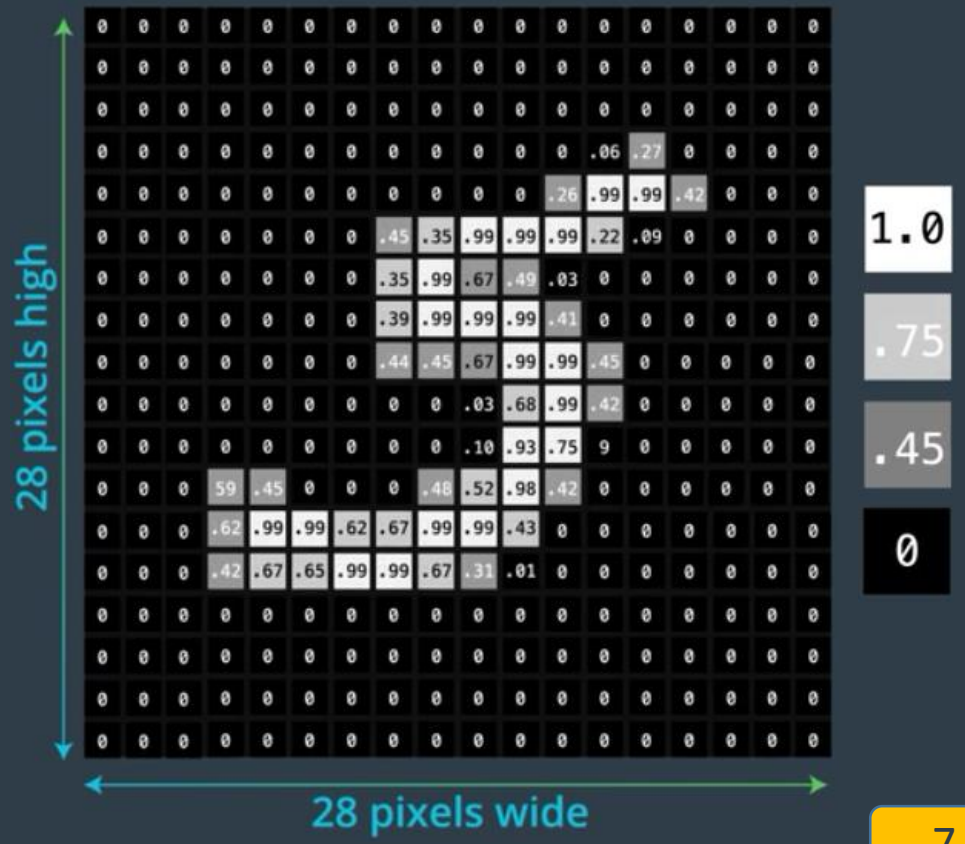


Image Processing

- **Image classification:** one of the major tasks in the field of image processing is image classification.
  - ✓ We want to create and train a neural network for discovering patterns in our training images and distinguish them based on their category.

- The most important option for the task of classification is MLP.

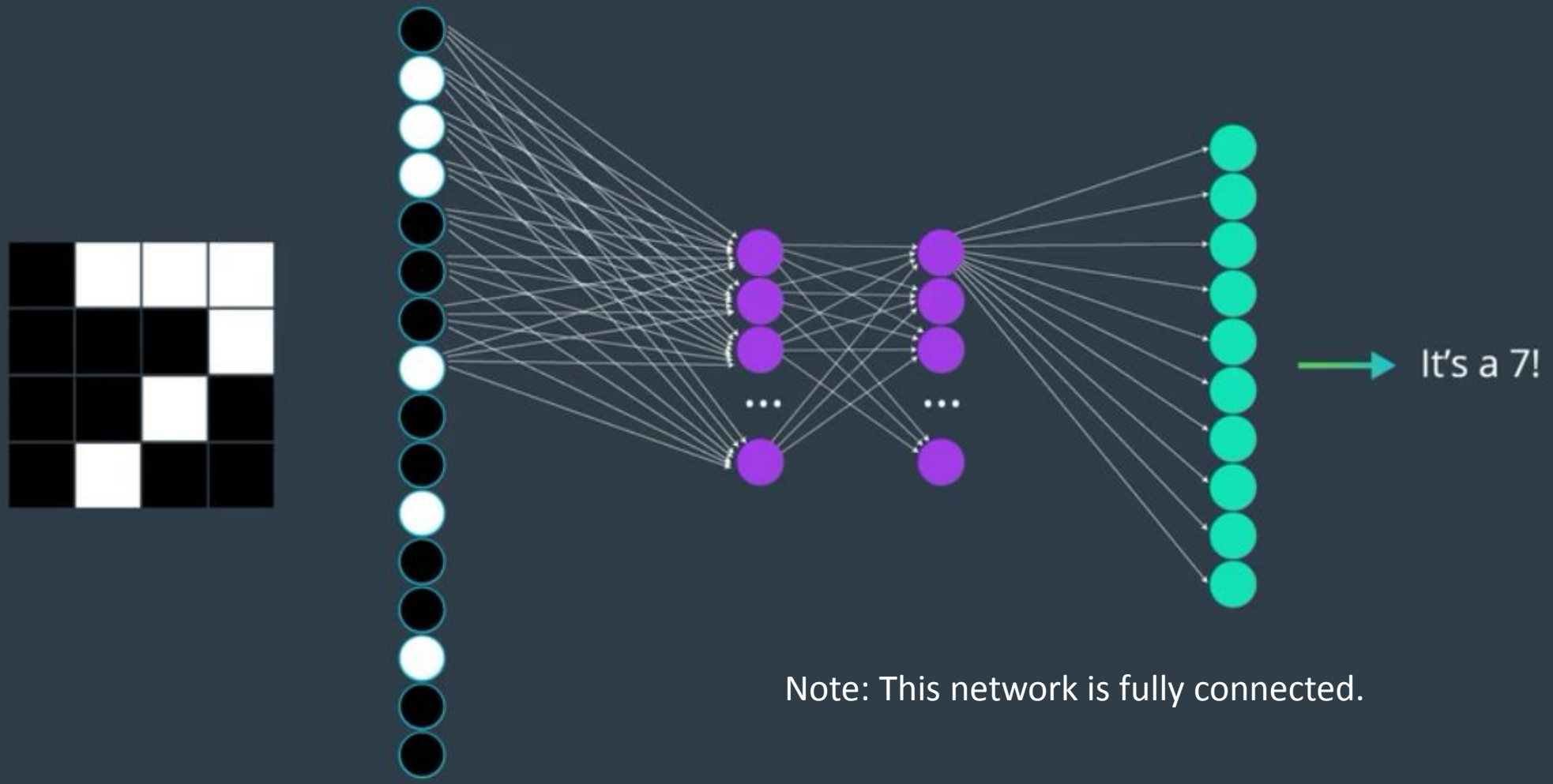
- Since MLP only takes vectors as input, we have to first convert any image array into a vector. This process is called flattening.

Image Processing

- For example in a 4x4 image, instead of representing as a 4x4 matrix, we can construct a vector with 16 entries.

- Then we should input our 16-dimensional vector into an MLP in order to classify it.

Image Processing



Note: This network is fully connected.

What is CNN?

- In the case of MNIST dataset, where all images of digits are roughly the same size and are centered in a 28x28 pixel grid, MLP has a good performance in image classification.

- What if we have to work with images where the digits could appear anywhere within the grid?

What is CNN?

- In fact when we work with real-world messy image data, the performance of MLP in image processing tasks (like classification) is weak.
  - ✓ The most important reason for this weakness is that MLP does not attend the spatial features.

- We saw that in order to feed an image to an MLP, the image must be converted to a vector (flattening operation). The MLP then treats this converted image as a simple vector with no spatial structure.
  - ✓ It has no knowledge that these numbers were originally spatially arranged in a grid

What is CNN?

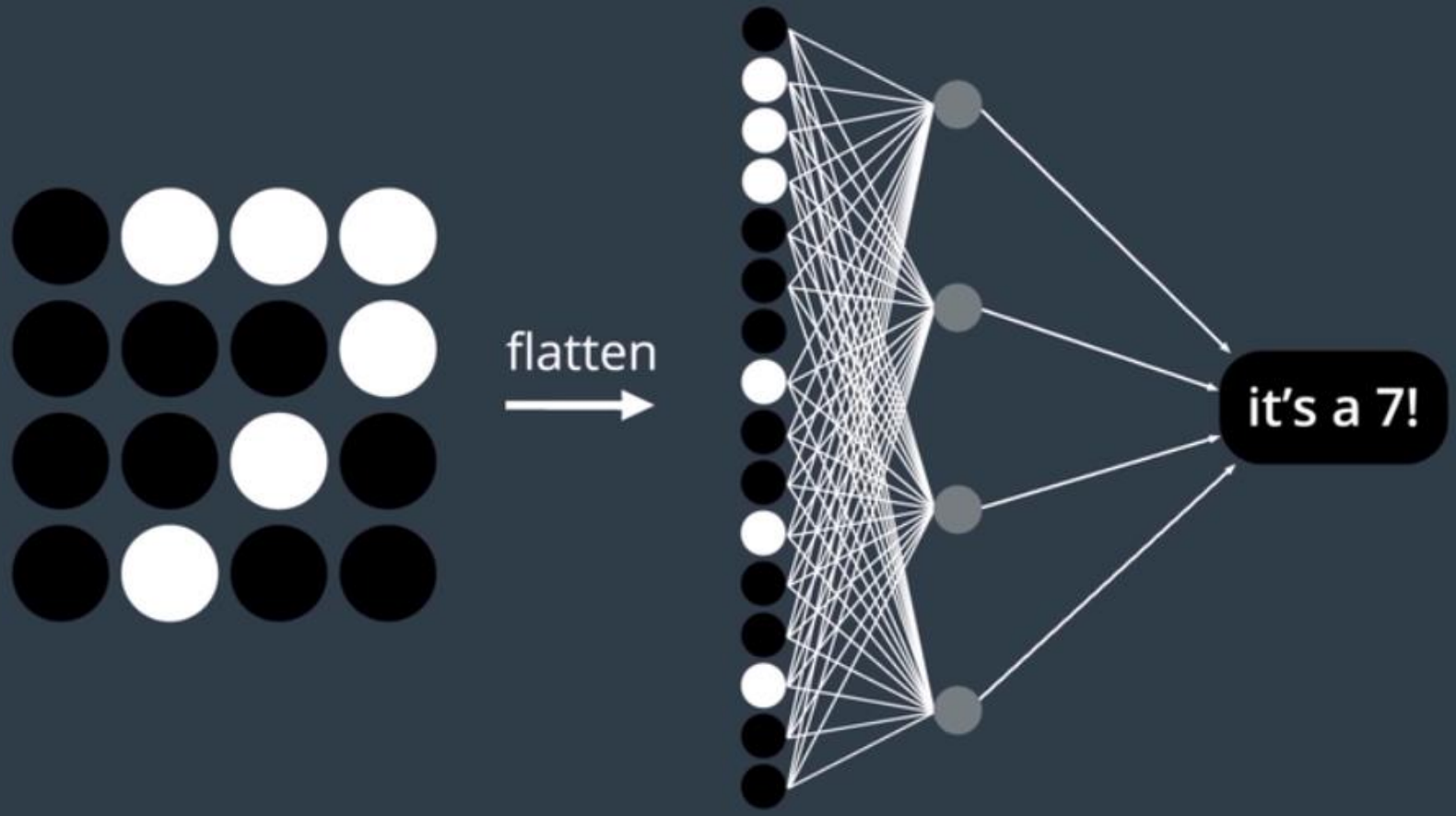
- We need a mechanism that can capture the patterns in multi-dimensional data. Due to that CNNs have been introduced!
- A **convolutional neural network** is a special kind of neural network that can remember **spatial information**.
- MLPs can only look at individual inputs but CNNs can look at an image as a whole or in patches and analyze groups of pixels at a time.
- Unlike MLPs, CNNs understand the fact that image pixels that are closer in proximity to each other are more heavily related than pixels that are far apart.

What is CNN?

- Unlike MLPs, CNNs accept matrices as input.
- MLPs only use fully connected layer, so they have a lot of parameters and their computational complexity is high. but CNNs use sparsely connected layers.
- In an MLP, every hidden node is connected to every pixel in the original image and every hidden node is responsible for gaining and understanding of the entire image all at once.

What is CNN?

- The schematic of an MLP:

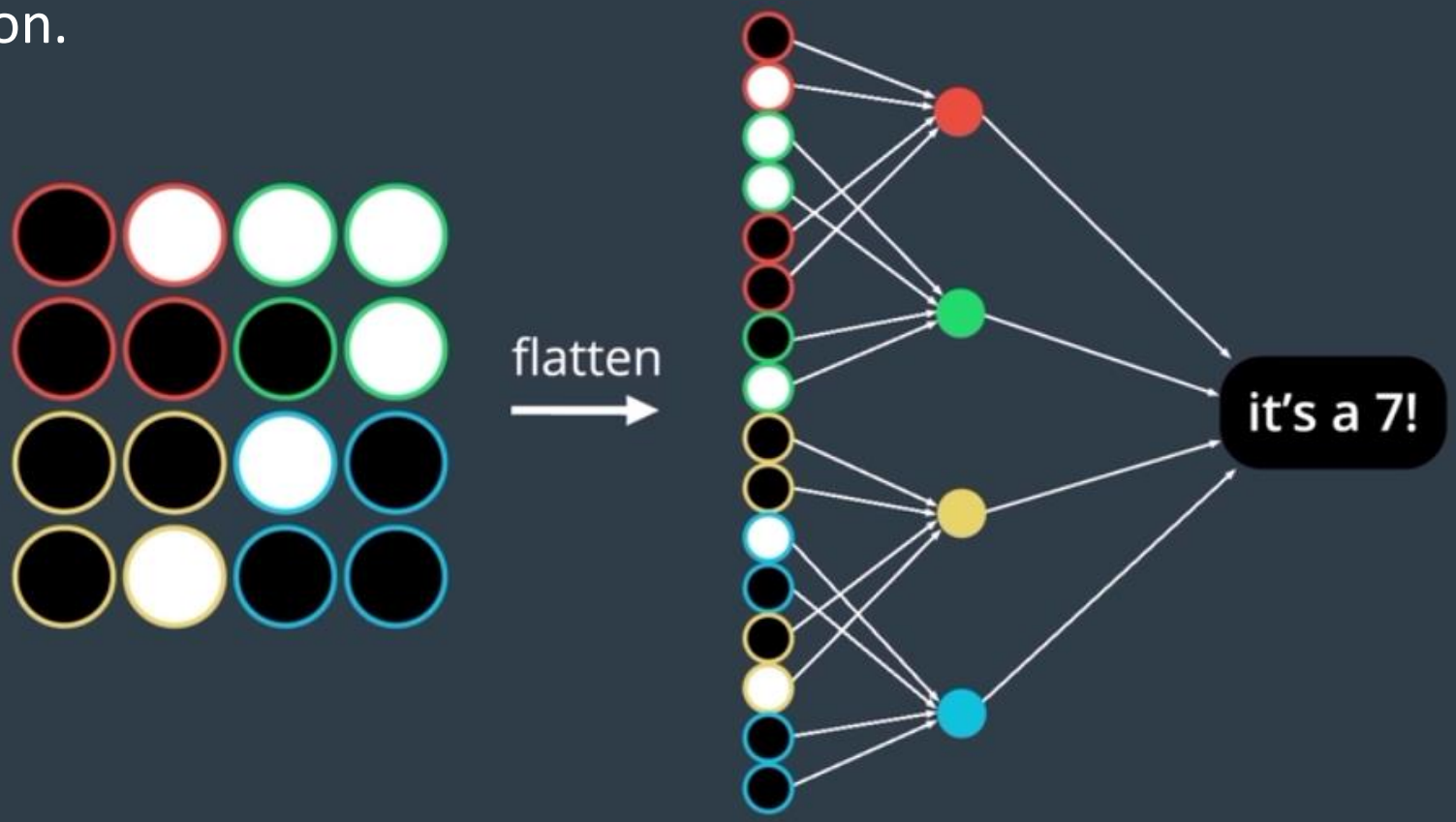


What is CNN?

- In CNNs, we break the image into regions and then connect each hidden node to only the pixels in one of these regions.
- In the example shown in the next slide, we divide the 16x16 image to 4 regions depicted by different colors (**Note: these colors are not color channels!**) and each hidden node sees only a quarter of the original image.
- With this regional breakdown and the assignment small local groups of pixels to different hidden nodes, every hidden node finds patterns in only one of the four regions in the image.

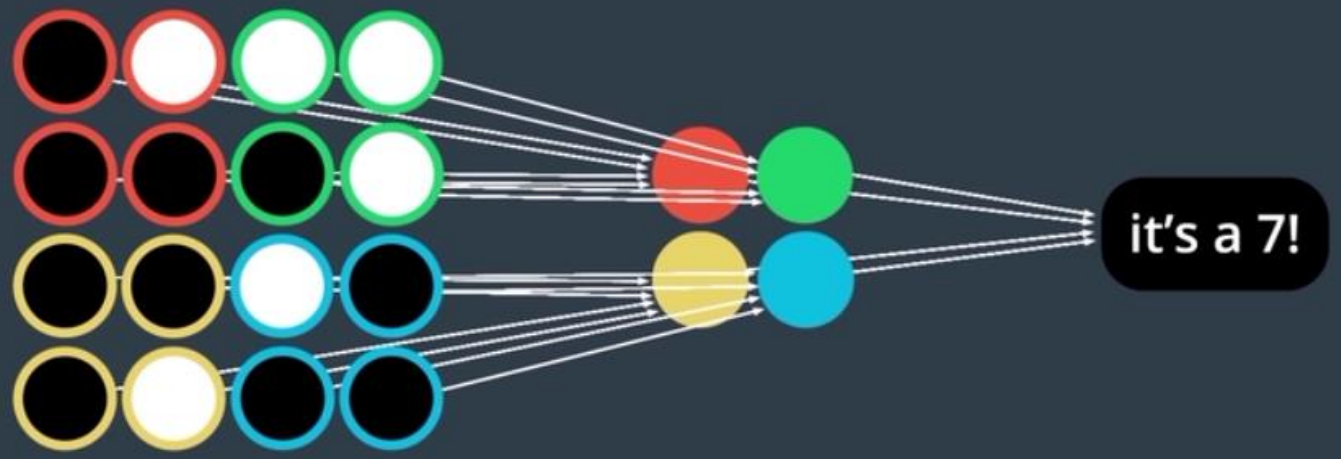
What is CNN?

- Each hidden node still reports to the output layer where the output layer combines the findings for the discovered patterns that learned separately in each region.



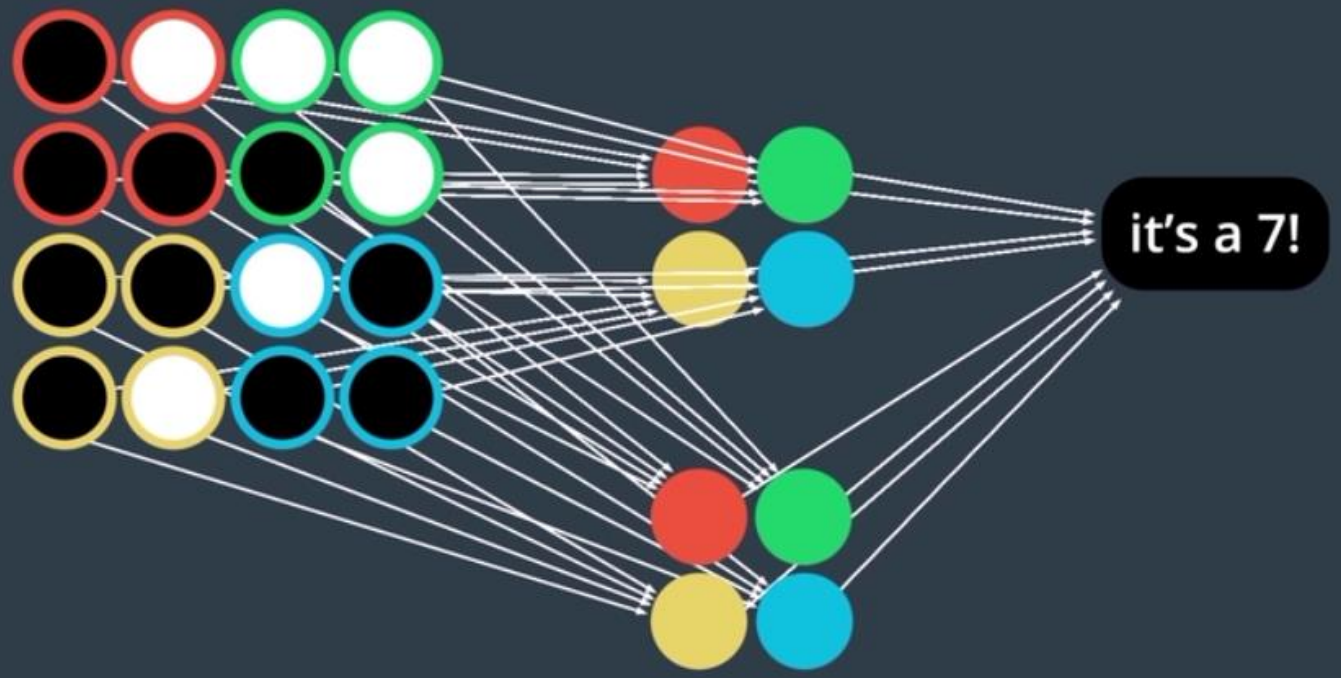
What is CNN?

- Therefore layers in a CNN is locally connected that uses far fewer parameter than a densely connected layer and has less potential to overfit.
- We can rearrange each of these vectors as a matrix:



What is CNN?

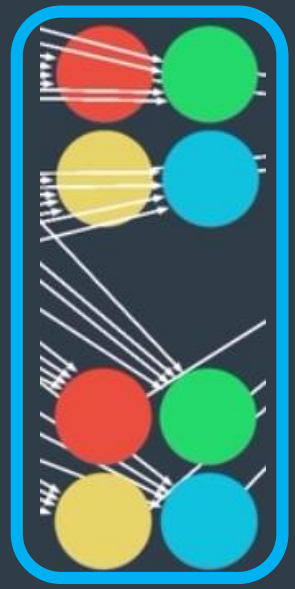
- We could also expand the number of patterns that we are able to detect by introducing more hidden nodes.



What is CNN?

- The red nodes in the hidden layer is only connected to red nodes in the input layer. This fact is the same for other colors.

- We now have 2 collections of hidden nodes, where each collection contains nodes responsible for examining a different region of the image. Each collection is called a filter (or kernel).



What is CNN?

- It is shown to be useful to have each of the hidden nodes within a filter share a common group of weights. This property is known as parameter sharing.

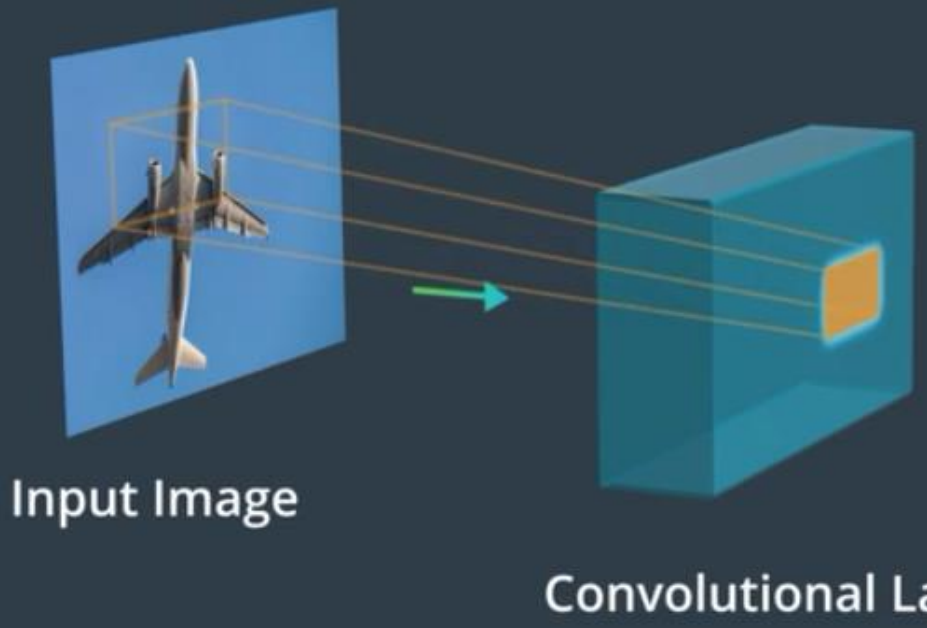
- The idea behind parameter sharing is that every pattern that is relevant towards understanding the image could appear anywhere within the image.

  - ✓ For example in the image classification task, we want our neural network to say it's a cat and It doesn't matter exactly where the cat is.

What is CNN?

- **How CNNs can remember spatial information?** By convolutional layers.

- **Convolutional layer:** a layer that applies a series of different image filters (convolutional kernels) to an input image.

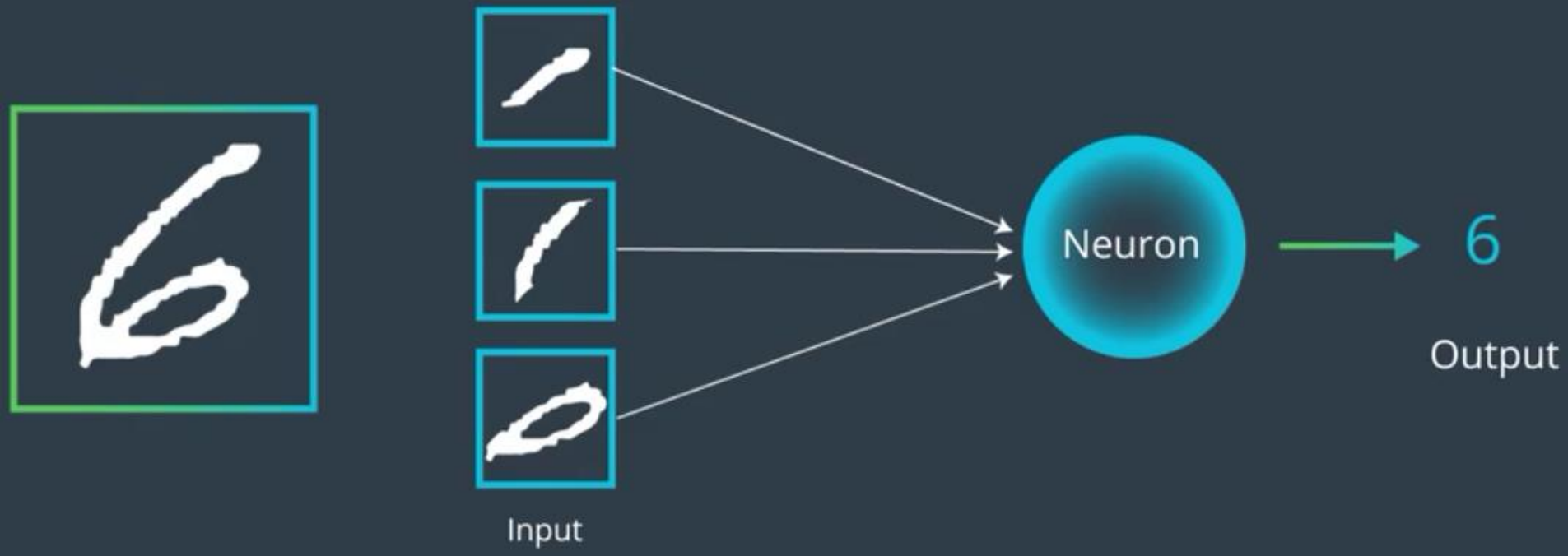


What is CNN?

- Each filter (kernel) extracts different features. Some of them may extract the edges of objects, the others may focus on the colors, etc.
- The resulting filtered images have different appearances.
- Later layers in a CNN will learn how to combine different color and other spatial features to produce an output like a class label (if the task is image classification).

What is CNN?

- For example, in the task of classifying digits, a CNN should learn to identify spatial patterns like the curves and lines in the number 6 to make it distinct from other digits.



Convolution

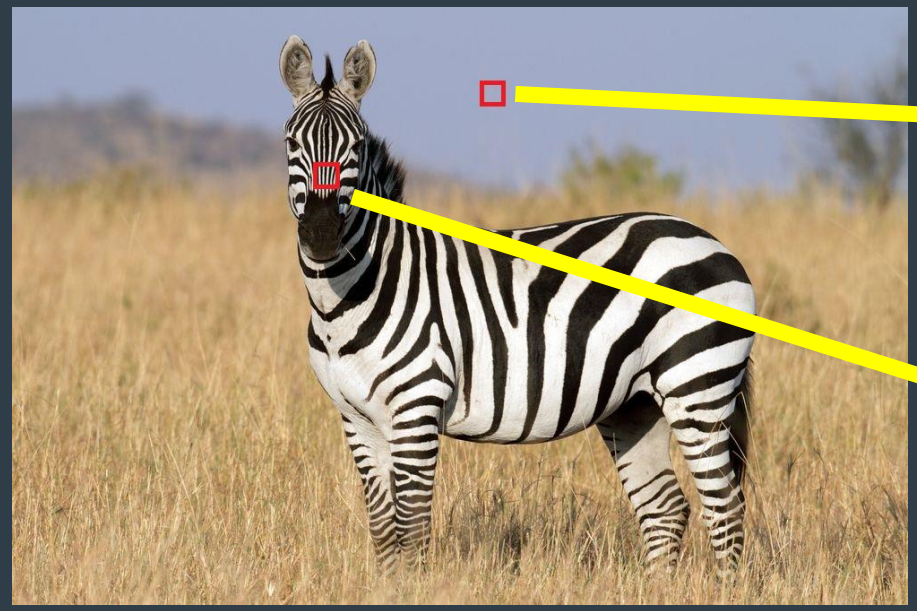
- **Filters:** in image processing, filters are used:
  - ✓ to remove unwanted or irrelevant information (like noise) in an image
  - ✓ to amplify features like object boundaries or other distinguishing traits

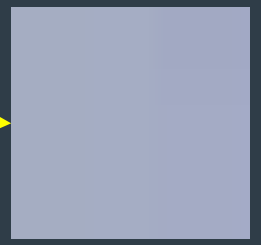
- One of the most important filters in image processing is high-pass filter.

- High-pass filters are used to make an image appear sharper and enhance high-frequency parts of an image (like the object edges).

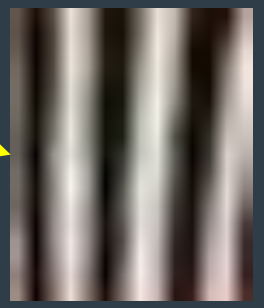
- High-frequency regions are areas where the levels of intensity in neighboring pixels rapidly change from very dark to very light and vice versa.

Convolution





Low-frequency



High-frequency

- Since we are looking at patterns of intensity, the filters operate on gray-scaled images to display patterns of lightness and darkness in a simple format.

Convolution

- What will happen if we apply a high-pass filter to an image?
  - ✓ Since there is no change (or a little change) in large areas of dark (or light), the output pixels will be black.
  - ✓ But the intensity change in high-frequency regions (like the edges) is a lot, so the filter will emphasize these areas.

- Mathematically, a filter is a matrix (grids of numbers). It can modify an image through convolution operation.

- An input image f(x,y) is convolved with this filter. Because of that, the filter also known as convolution kernel or simply **kernel**.

Convolution

- **Convolution Kernel:** an important operation in computer vision field that is the basis for convolutional neural networks.

- It takes a kernel (filter) and passes it over an image pixel by pixel transforming based on what these numbers are.

- By changing the numbers in kernel matrix we can create many different effects from edge detection to blurring an image.

Convolution

- For every pixel in the grayscale image, we put our kernel over it so that the pixel is in the center of the kernel.
- Each element of the kernel should be multiplied by the corresponding pixel in the image and finally the results will be summed up to get a new pixel value.

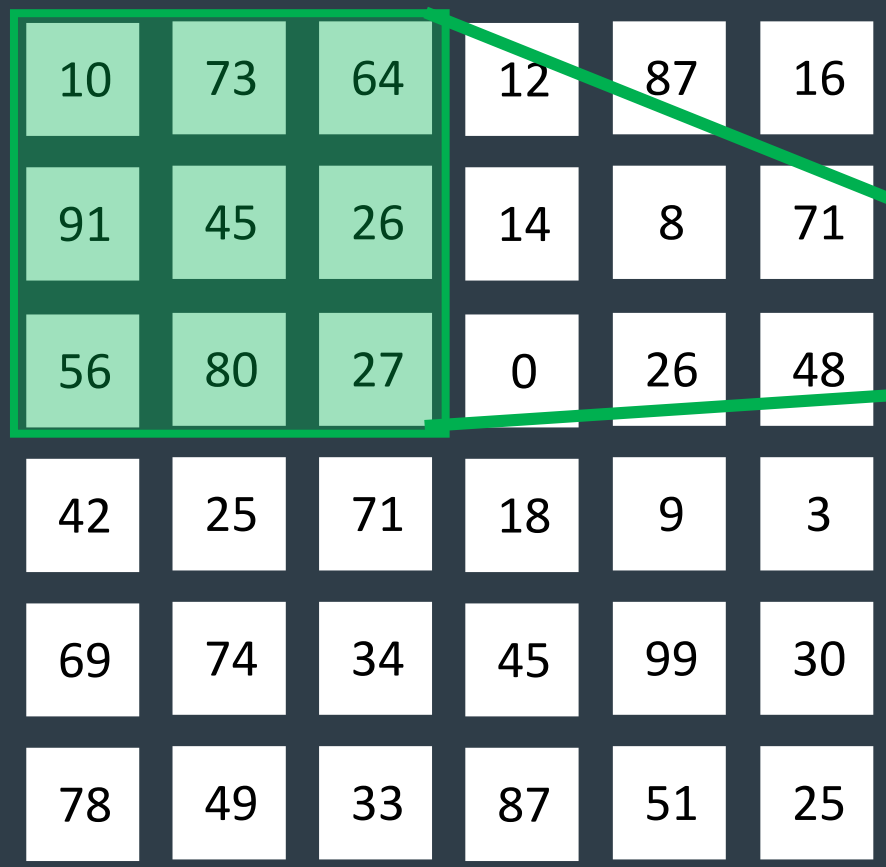
Convolution

- We will walk through an example of convolution operation. In this example, we have a 3x3 kernel. This kernel is a popular kernel which is called high-pass filter and is used for the task of edge detection.
- The image that is supposed to be convolved with the kernel is a 6x6 image.
- This kernel will slide on the image to produce a 4x4 image (ignore padding for now).

0	-1	0
-1	4	-1
0	-1	0

10	73	64	12	87	16
91	45	26	14	8	71
56	80	27	0	26	48
42	25	71	18	9	3
69	74	34	45	99	30
78	49	33	87	51	25

Convolution



$0 \times 10 + (-1) \times 73 + 0 \times 64 +$
$(-1) \times 91 + 4 \times 45 + (-1) \times 26 +$
$0 \times 56 + (-1) \times 80 + 0 \times 27 = -90$

Convolution

$$0 \times 73 + (-1) \times 64 + 0 \times 12 + (-1) \times 45 + 4 \times 26 + (-1) \times 14 + 0 \times 80 + (-1) \times 27 + 0 \times 0 = -46$$

-90 -46

Convolution

$0 \times 64 + (-1) \times 12 + 0 \times 87 +$
$(-1) \times 26 + 4 \times 14 + (-1) \times 8 +$
$0 \times 27 + (-1) \times 0 + 0 \times 26 = 10$

Convolution

$$0 \times 12 + (-1) \times 87 + 0 \times 16 + (-1) \times 14 + 4 \times 8 + (-1) \times 71 + 0 \times 0 + (-1) \times 26 + 0 \times 48 = -166$$

-90	-46	10	-166

Convolution

10	73	64	12	87	16
91	45	26	14	8	71
56	80	27	0	26	48
42	25	71	18	9	3
69	74	34	45	99	30
78	49	33	87	51	25

$$0 \times 91 + (-1) \times 45 + 0 \times 26 + (-1) \times 56 + 4 \times 80 + (-1) \times 27 + 0 \times 42 + (-1) \times 25 + 0 \times 71 = 167$$

-90	-46	10	-166
167			

Convolution

10	73	64	12	87	16
91	45	26	14	8	71
56	80	27	0	26	48
42	25	71	18	9	3
69	74	34	45	99	30
78	49	33	87	51	25

$$0 \times 18 + (-1) \times 9 + 0 \times 3 + (-1) \times 45 + 4 \times 99 + (-1) \times 30 + 0 \times 87 + (-1) \times 51 + 0 \times 25 = 261$$

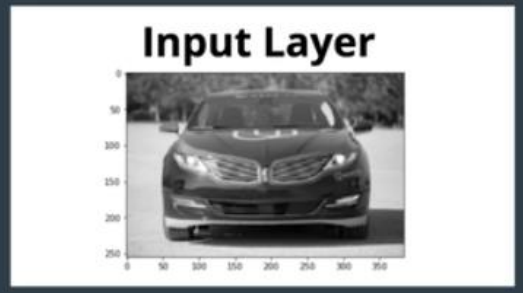
-90	-46	10	-166
167	.	.	.
.	.	.	.
.		.	261

Convolution

- The value for each pixel in the output image is a weighted sum of neighboring pixels in the input image at the same location. These weights are determined by the coefficients in the kernel.
- The filter that has been discussed, was a specific filter for identifying the edge patterns. Other filters may have different coefficients, but the operation is the same.
- Filters can have any size but they are often squared with an odd side length (like 3x3 and 7x7).

Convolution

- Suppose we have a convolutional layer with 4 filters applying to a grayscale car image. Each filter is 4x4 and should be convolved with the input image.



Filter 1

-1	-1	+1	+1
-1	-1	+1	+1
-1	-1	+1	+1
-1	-1	+1	+1

Filter 2

+1	+1	-1	-1
+1	+1	-1	-1
+1	+1	-1	-1
+1	+1	-1	-1

Filter 3

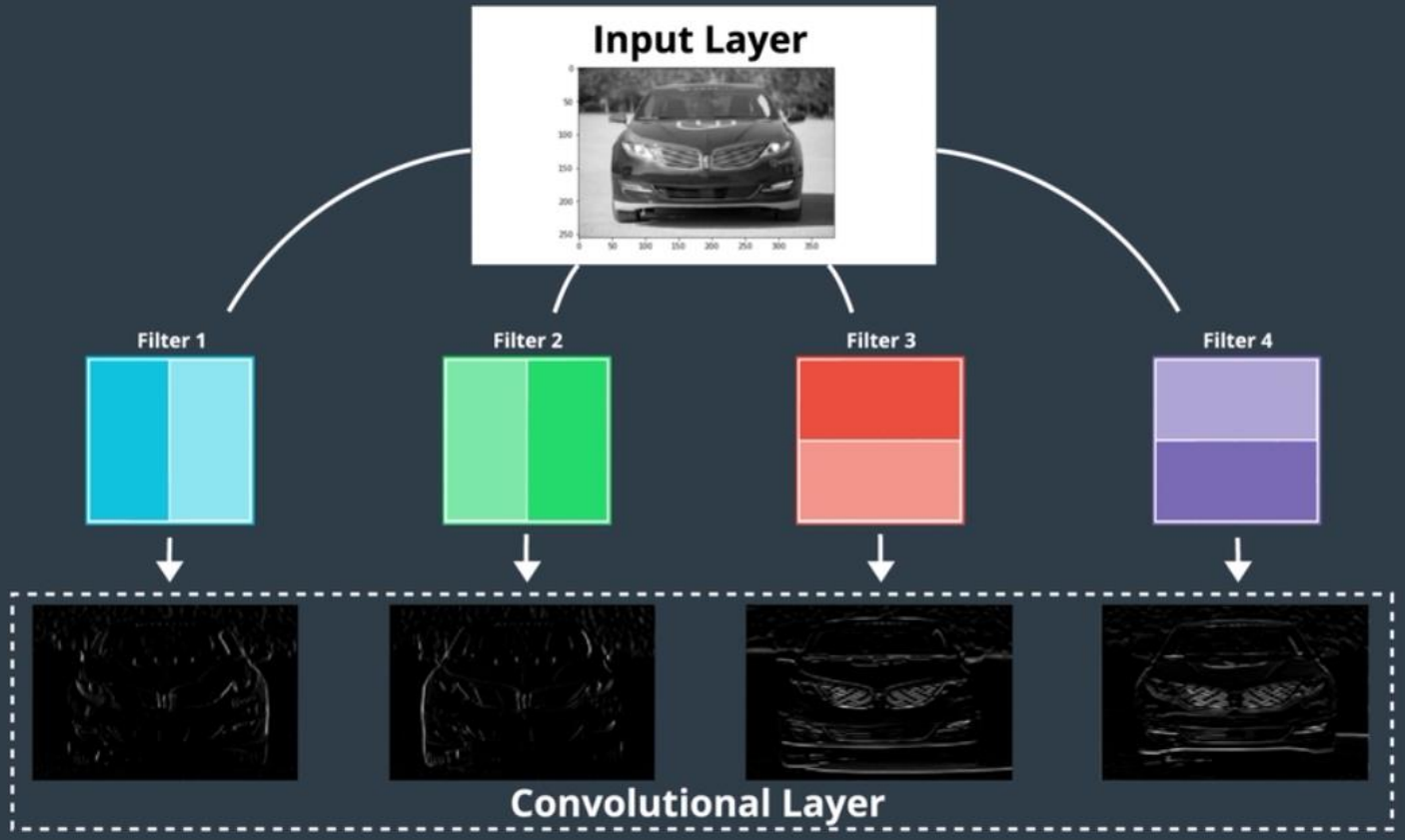
-1	-1	-1	-1
-1	-1	-1	-1
+1	+1	+1	+1
+1	+1	+1	+1

Filter 4

+1	+1	+1	+1
+1	+1	+1	+1
-1	-1	-1	-1
-1	-1	-1	-1

Convolution

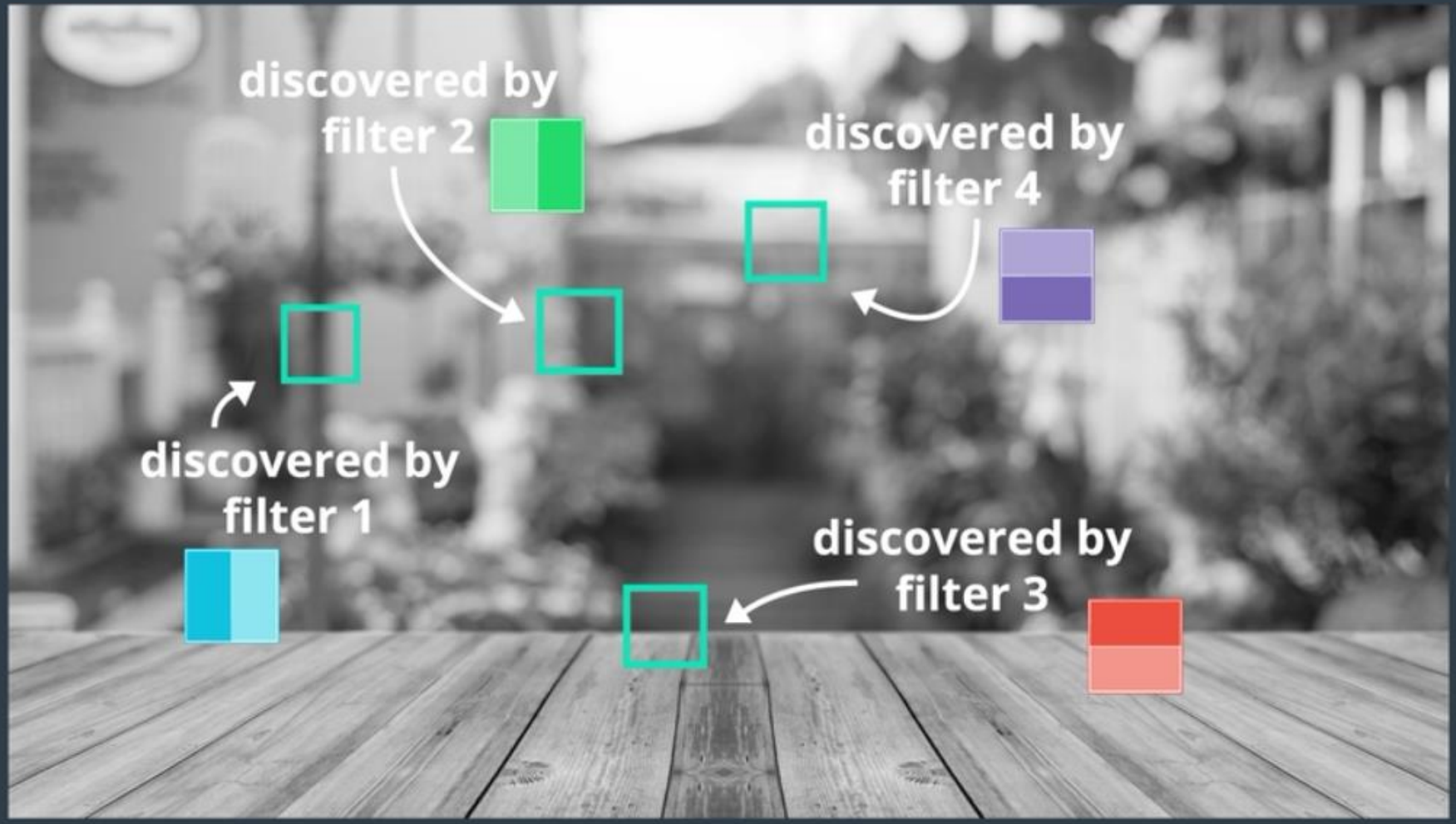
- The filtered images (output of filters) are also called **feature maps**. Each of these feature maps show a much simpler image with less information.



Convolution

- In this case, the first two filters discover vertical edges and the last two filters detect horizontal edges in the image.
- Lighter values in the feature map mean the pattern was detected by the filter in the image.
- The first filter detects the right edge of the car while the second filter captures the left edge of the car.
- In another example in the next slide, we show that what patterns can these edge detector filters detect in an image.

Convolution



Convolution

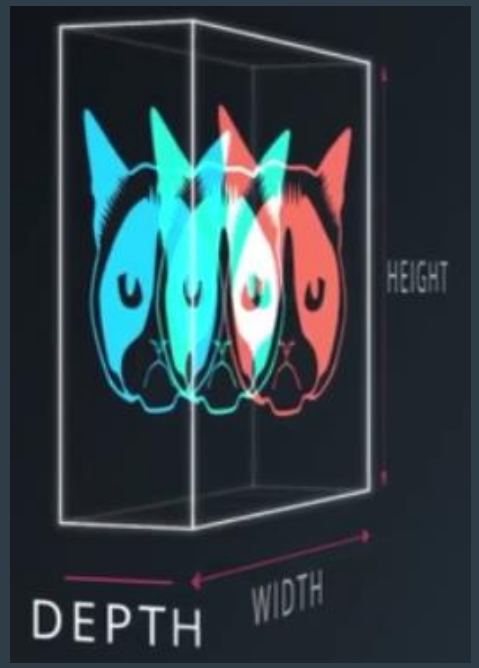
❑ **Color images:**

- As we saw, grayscale images are interpreted by the computer as a 2D array with height and width.

Convolution

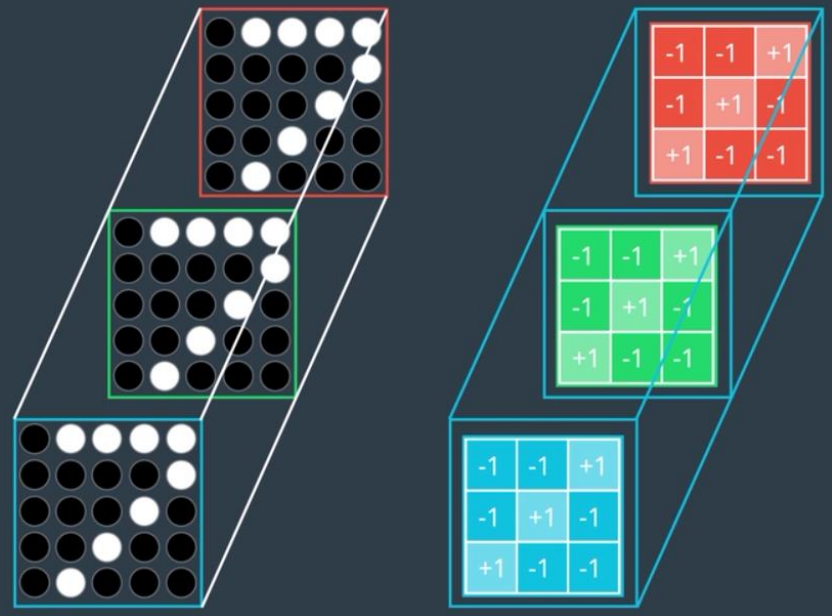
- Color images are interpreted by the computer as a 3D array with height, width and depth.
  - In the case of RGB images, the depth is 3.

- The RGB 3D array is a stack of three 2D matrices (Red matrix, Green matrix and Blue matrix).



Convolution

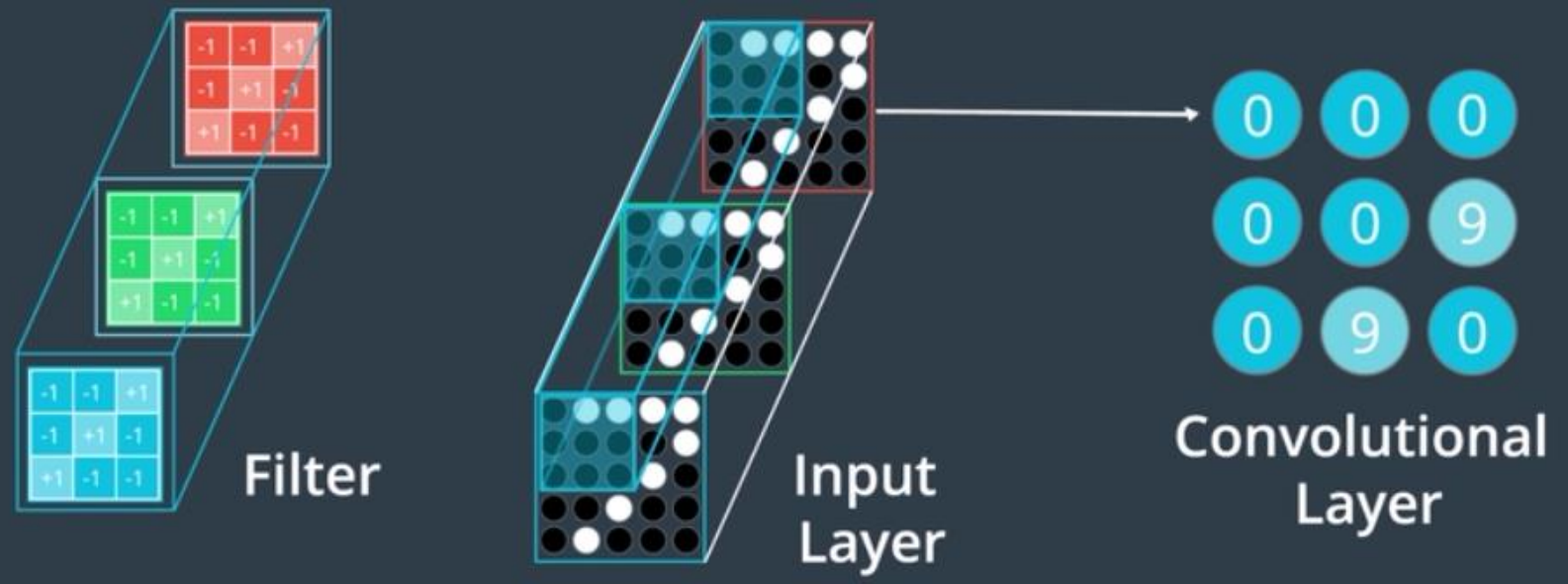
- **Convolution on a color image:** as with grayscale images, we move a filter horizontally and vertically across the image.
  - ✓ The difference is that now the filter is three dimensional to have a value for each color channel at each horizontal and vertical location in the image array.



Convolution

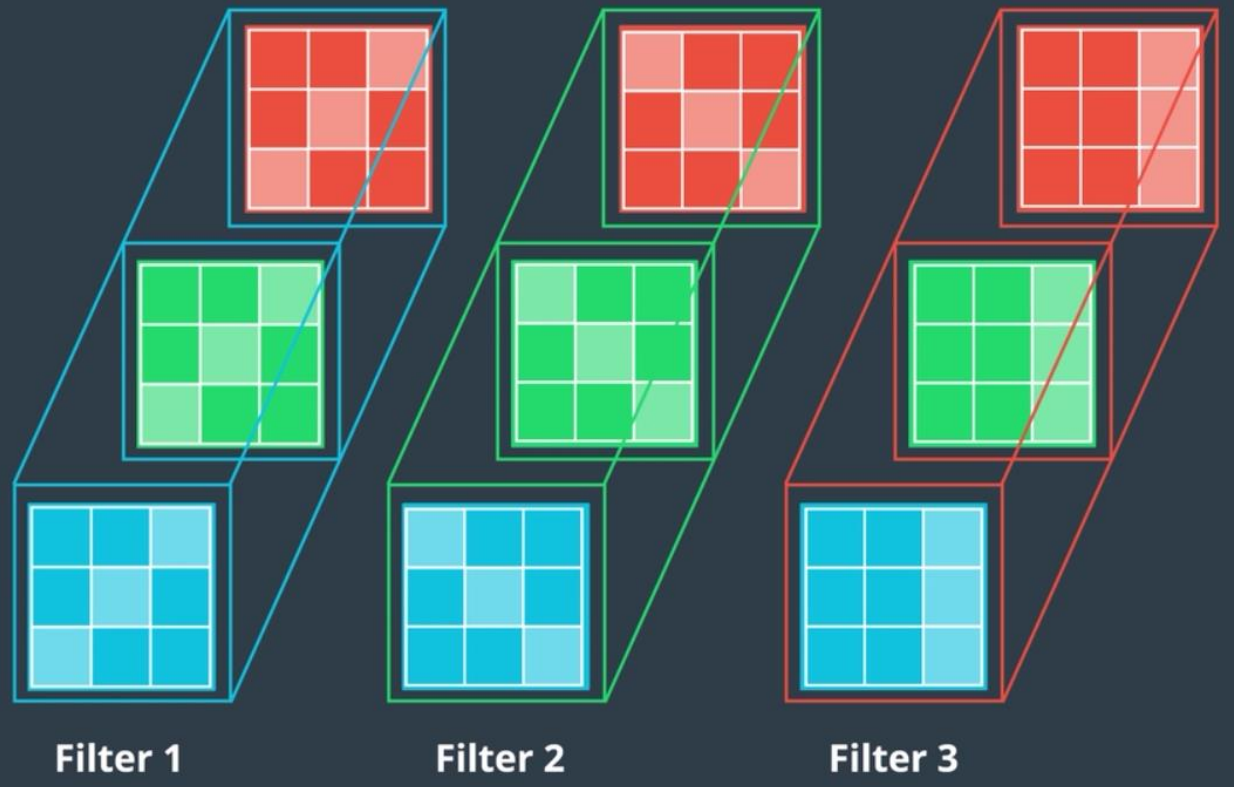
- Just as the color image that is a stack of three 2D matrices, the filter is also a stack of three 2D matrices.
  - ✓ Both the color image and the filter have red, green and blue channels.
- To obtain the value of the nodes in the feature map corresponding to this RGB filter, we perform the same calculations (is shown in the next slide).
  - ❖ Notice that these calculations are for a single filter in a convolutional layer.

Convolution



Convolution

- In a convolutional layer we can have more than one filter.
- For example here we have 3 RGB filters that each filter is a 3D array.

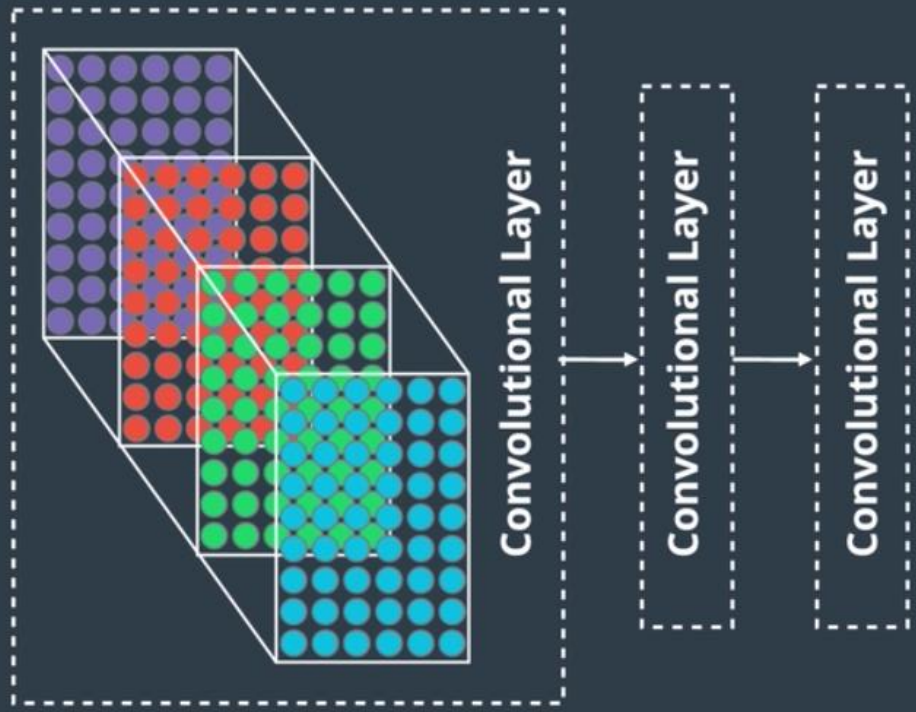


Convolution

- **more convolutional layers:** a convolutional layer can capture patterns in the original image. we can stack the outputs (feature maps) of a convolutional layer to get a 3D array.

- We can then use this 3D array as input to another convolutional layer to discover **patterns within the patterns** that we discovered in the first convolutional layer.

✓ and again we can discover patterns within the patterns within…



Convolution

- We can control the behavior of a convolutional layer by specifying the number of filters and the size of each filter.
- To increase the number of nodes in a convolutional layer, we could increase the number of filters.
- To increase the size of the detected patterns, we could increase the size of our filters.

Convolution in PyTorch

CONV1D

CLASS torch.nn.Conv1d(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=True*, *padding_mode='zeros'*, *device=None*, *dtype=None*) [SOURCE]

Applies a 1D convolution over an input signal composed of several input planes.

In the simplest case, the output value of the layer with input size (N, C_{in}, L) and output $(N, C_{\text{out}}, L_{\text{out}})$ can be precisely described as:

$$\text{out}(N_i, C_{\text{out}_j}) = \text{bias}(C_{\text{out}_j}) + \sum_{k=0}^{C_{in}-1} \text{weight}(C_{\text{out}_j}, k) \star \text{input}(N_i, k)$$

where $\star$ is the valid cross-correlation operator, N is a batch size, C denotes a number of channels, L is a length of signal sequence.

Convolution in PyTorch

CONV2D

CLASS torch.nn.Conv2d(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=True*, *padding_mode='zeros'*, *device=None*, *dtype=None*) [SOURCE]

Applies a 2D convolution over an input signal composed of several input planes.

In the simplest case, the output value of the layer with input size (N, C_{in}, H, W) and output $(N, C_{\text{out}}, H_{\text{out}}, W_{\text{out}})$ can be precisely described as:

$$\text{out}(N_i, C_{\text{out}_j}) = \text{bias}(C_{\text{out}_j}) + \sum_{k=0}^{C_{\text{in}}-1} \text{weight}(C_{\text{out}_j}, k) \star \text{input}(N_i, k)$$

where $\star$ is the valid 2D cross-correlation operator, N is a batch size, C denotes a number of channels, H is a height of input planes in pixels, and W is width in pixels.

Convolution in PyTorch

Example of Conv1D

```
m = nn.Conv1d(16, 33, 3, stride=2)
input = torch.randn(20, 16, 50)
output = m(input)
```

Example of Conv2D

```
>>> # With square kernels and equal stride
>>> m = nn.Conv2d(16, 33, 3, stride=2)
>>> # non-square kernels and unequal stride and with padding
>>> m = nn.Conv2d(16, 33, (3, 5), stride=(2, 1), padding=(4, 2))
>>> # non-square kernels and unequal stride and with padding and
dilation
>>> m = nn.Conv2d(16, 33, (3, 5), stride=(2, 1), padding=(4, 2),
dilation=(3, 1))
>>> input = torch.randn(20, 16, 50, 100)
>>> output = m(input)
```

Stride & Padding

❑ Stride:

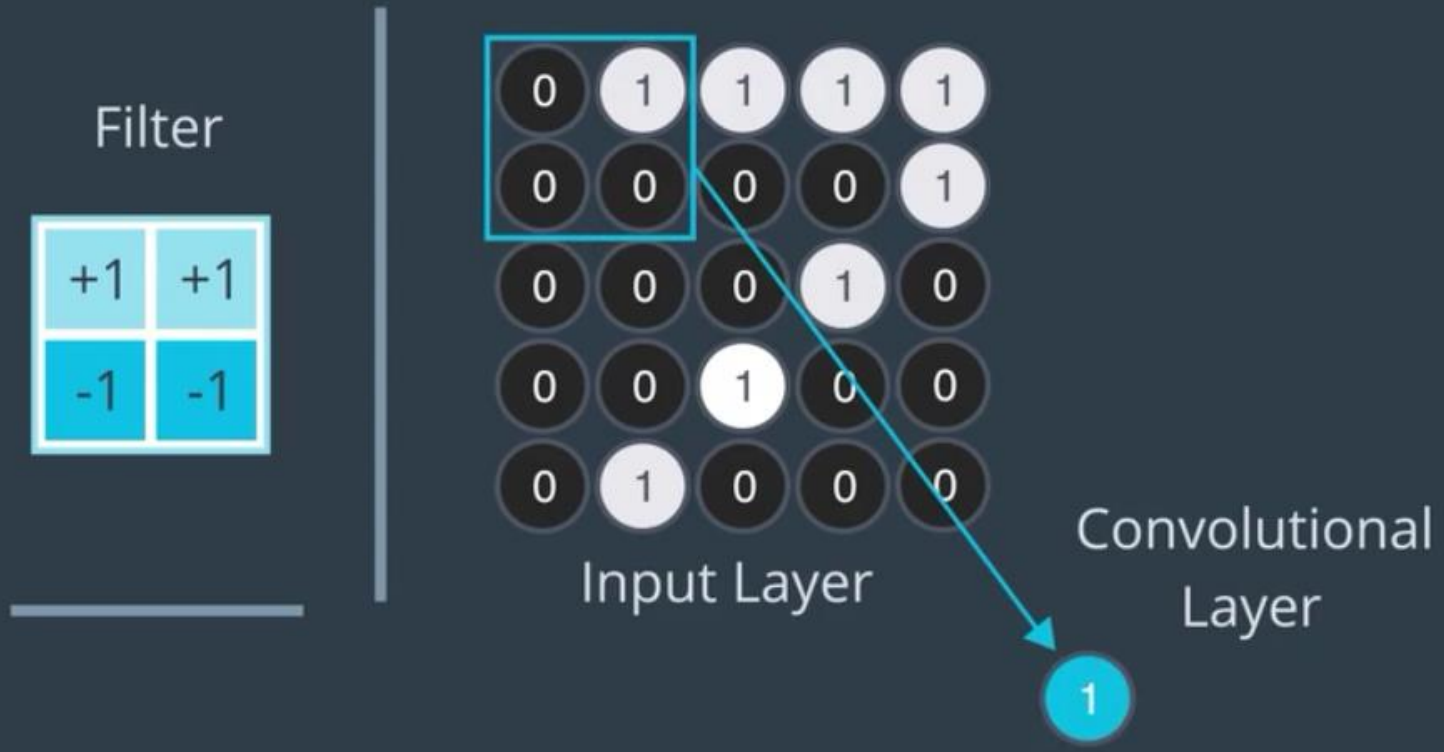
- There are more hyperparameters in a convolutional layer. One of these hyperparameters is the stride of the convolution.
- The stride is the amount by which the filter slides over the image.
- If the stride is s, we slide horizontally across the image s pixel at a time until we reach the end of a row, then we slide s pixel vertically and repeat the horizontal process for the next row. This operations should be done until the whole image will be covered.

Stride & Padding

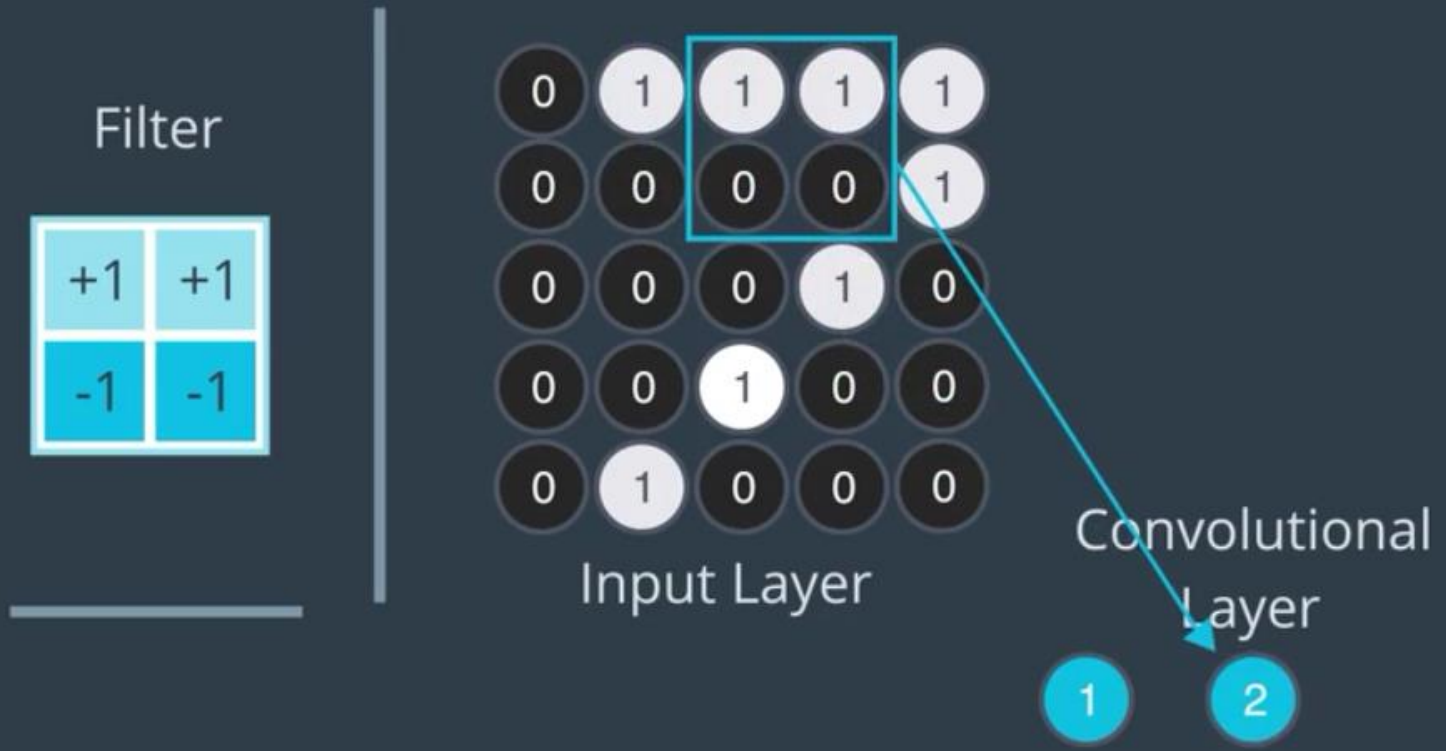
- The stride in the example that we discussed in the convolution section was 1.
- Consider an example of a 5x5 grayscale image and a filter with the height and width of 2.

- We show the convolution operation with stride 2, step by step.

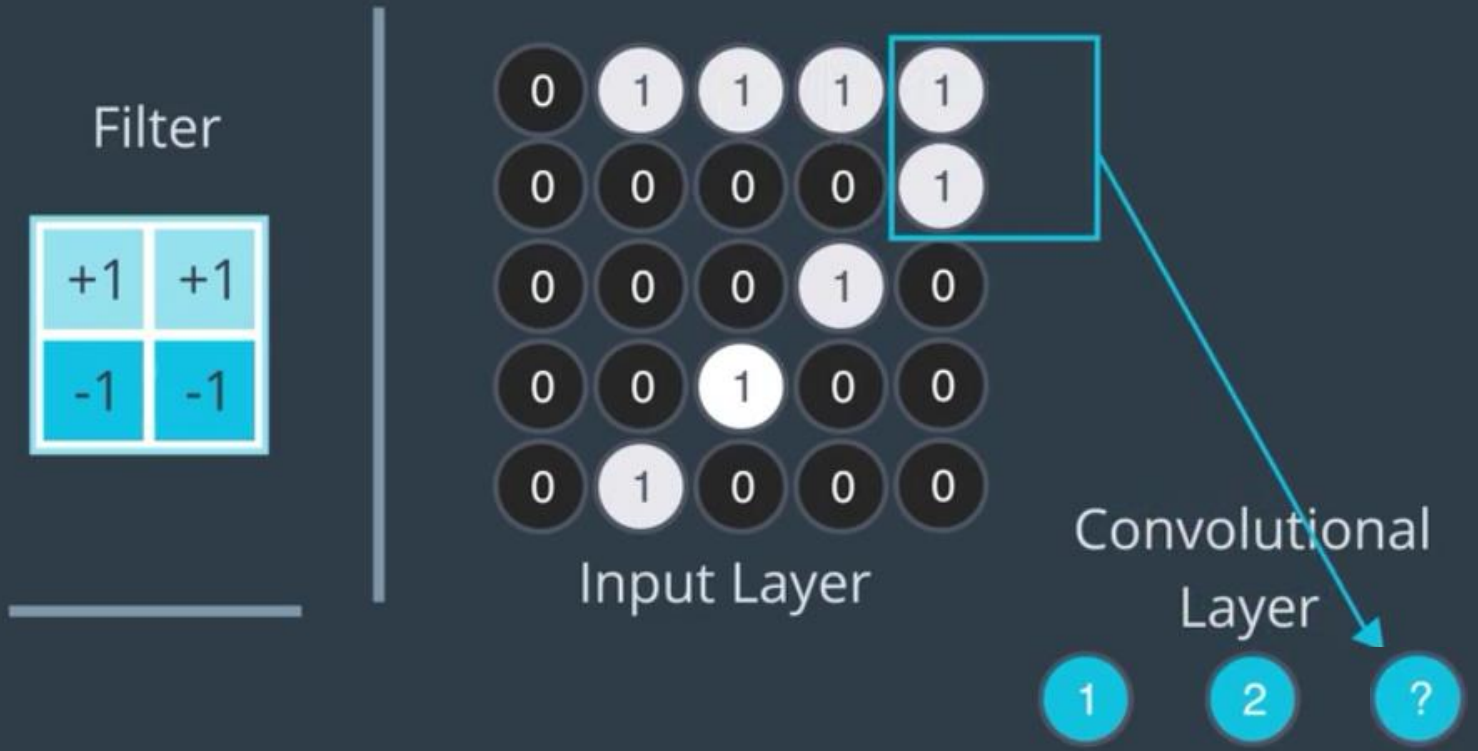
Stride & Padding



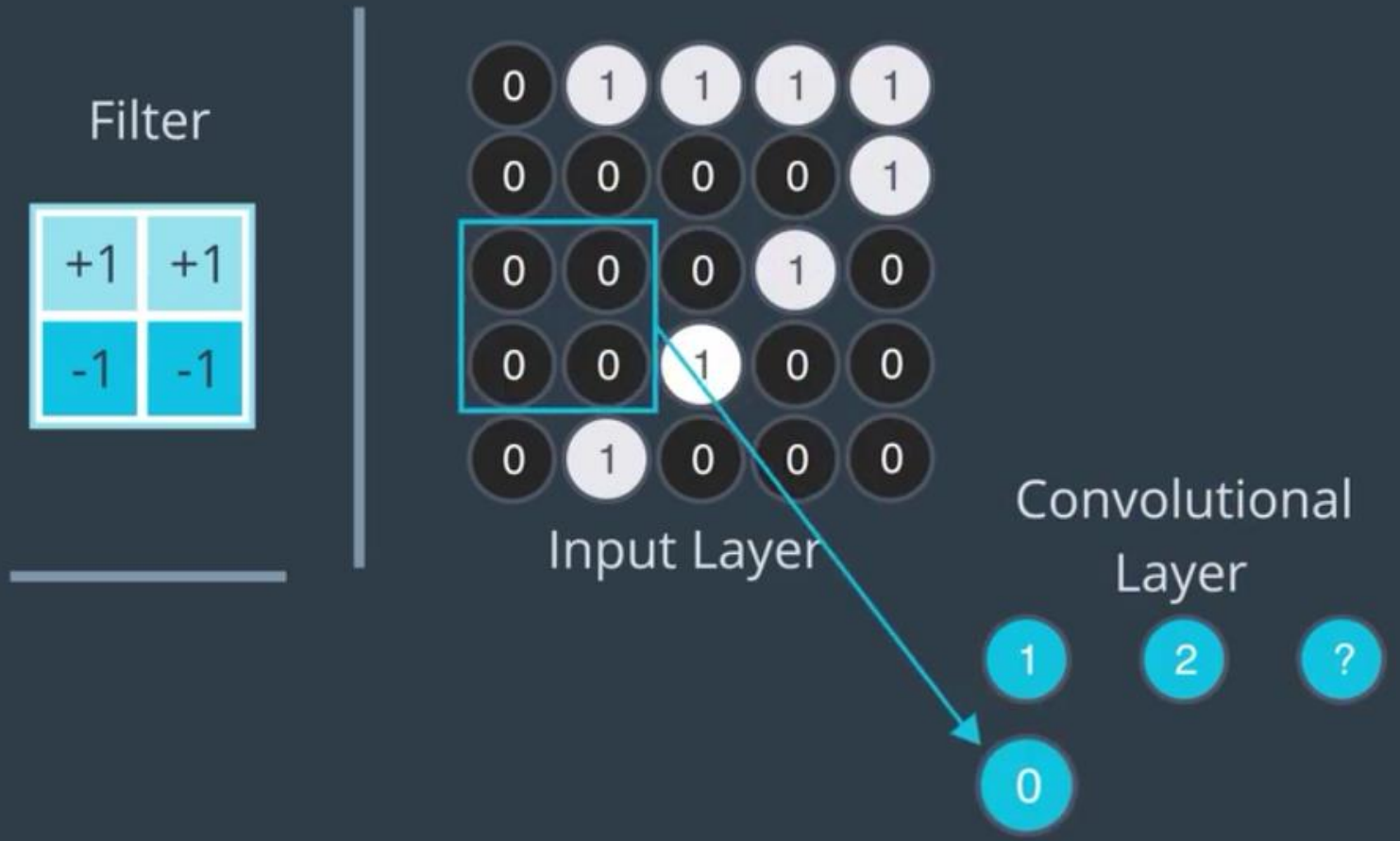
Stride & Padding



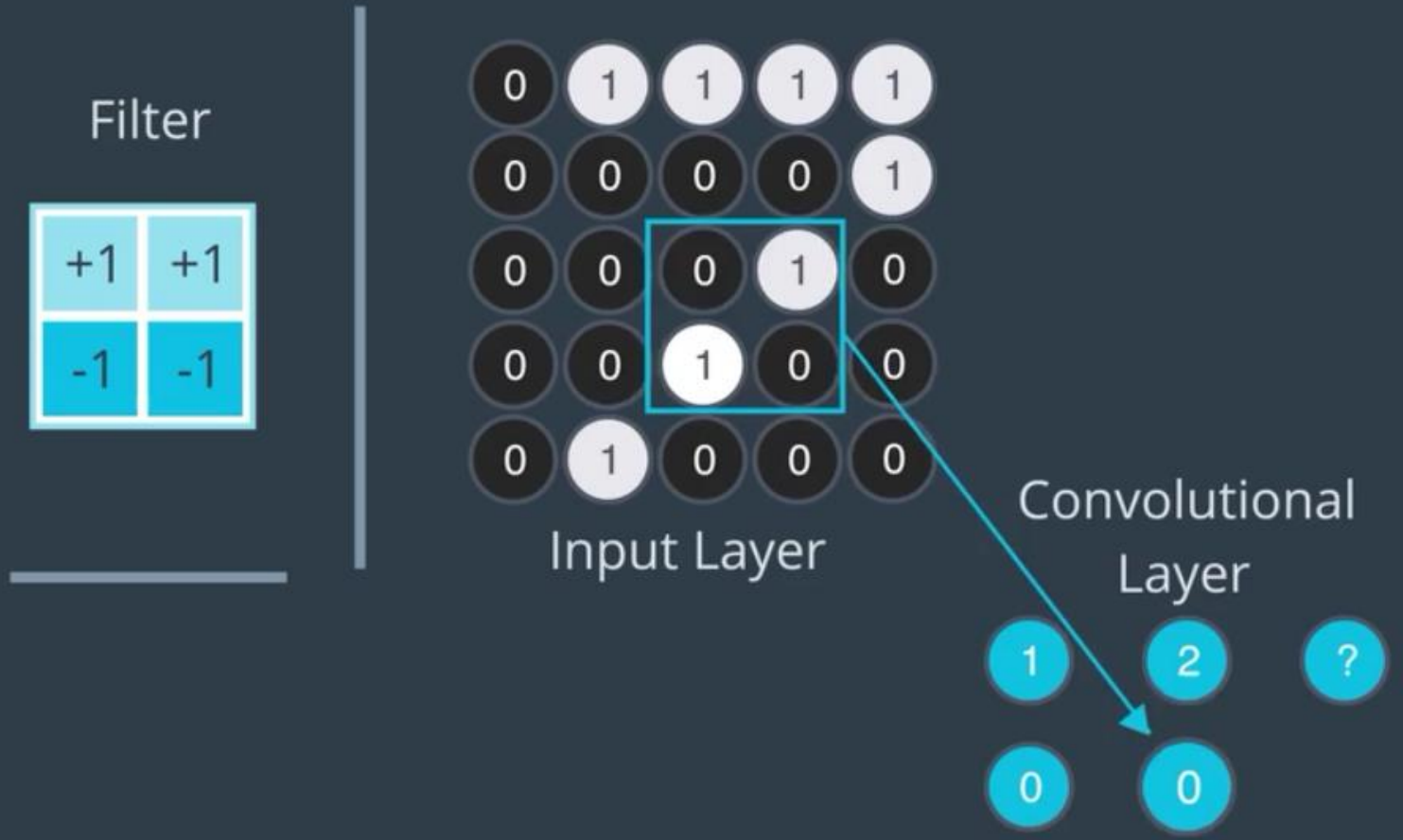
Stride & Padding



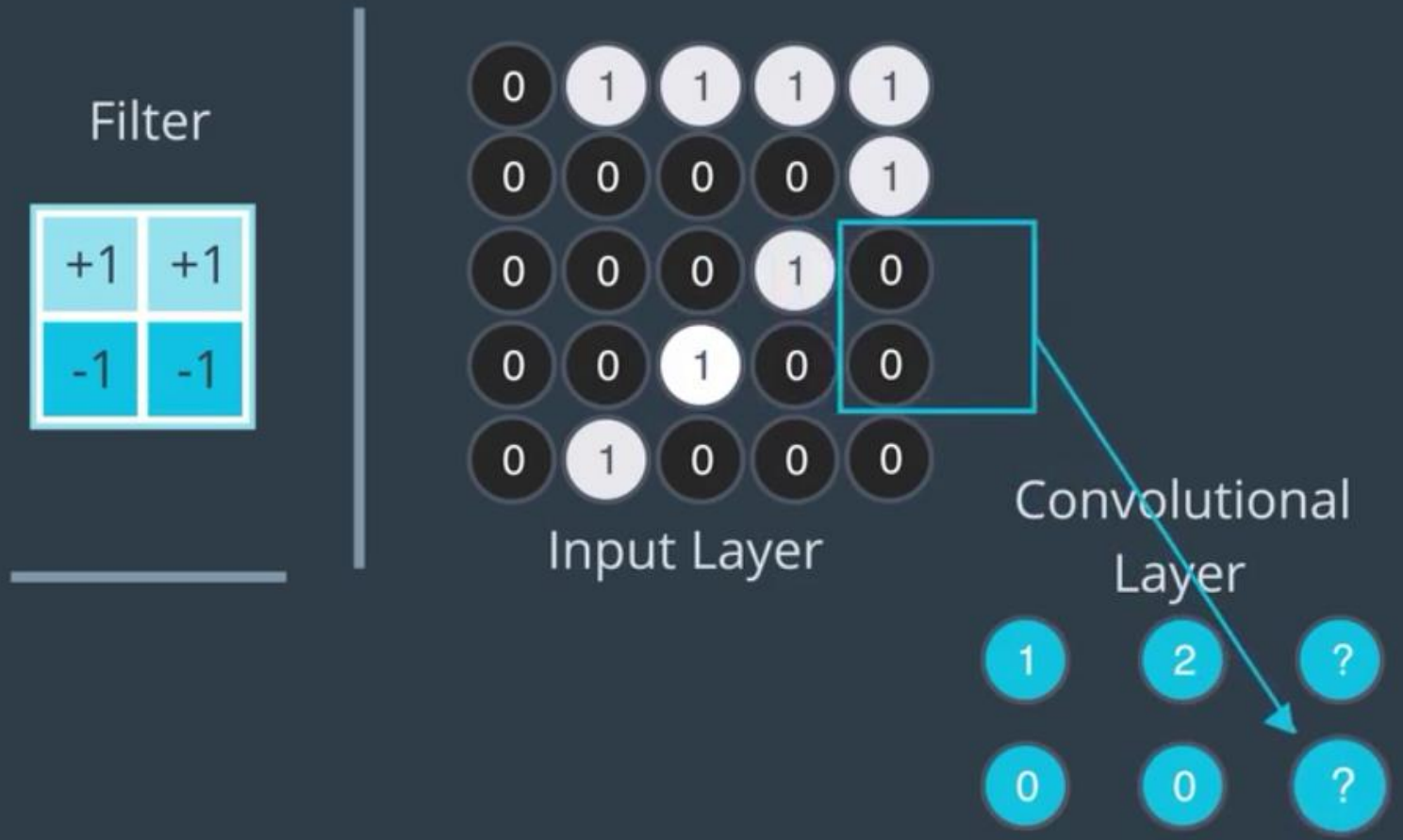
Stride & Padding



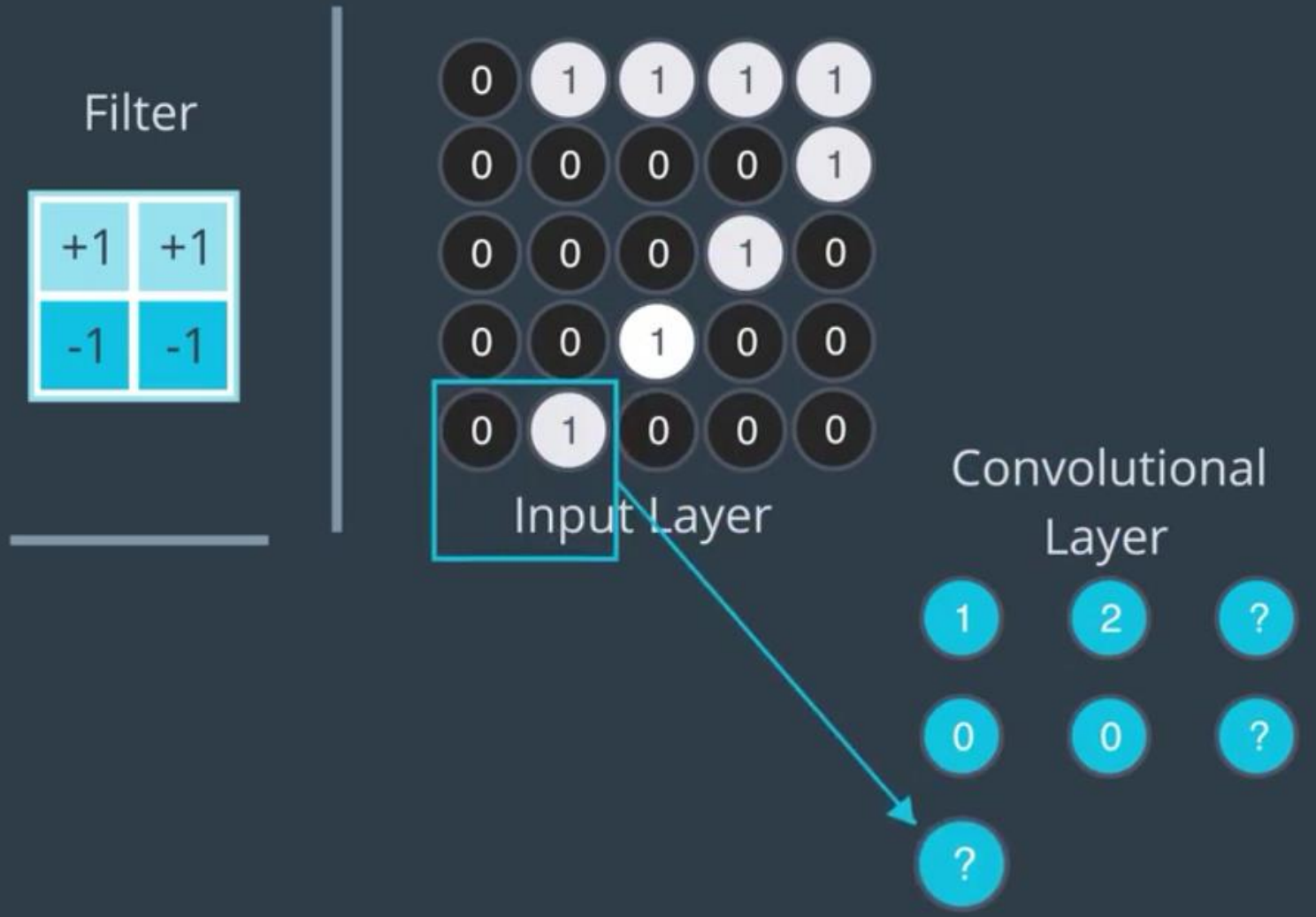
Stride & Padding



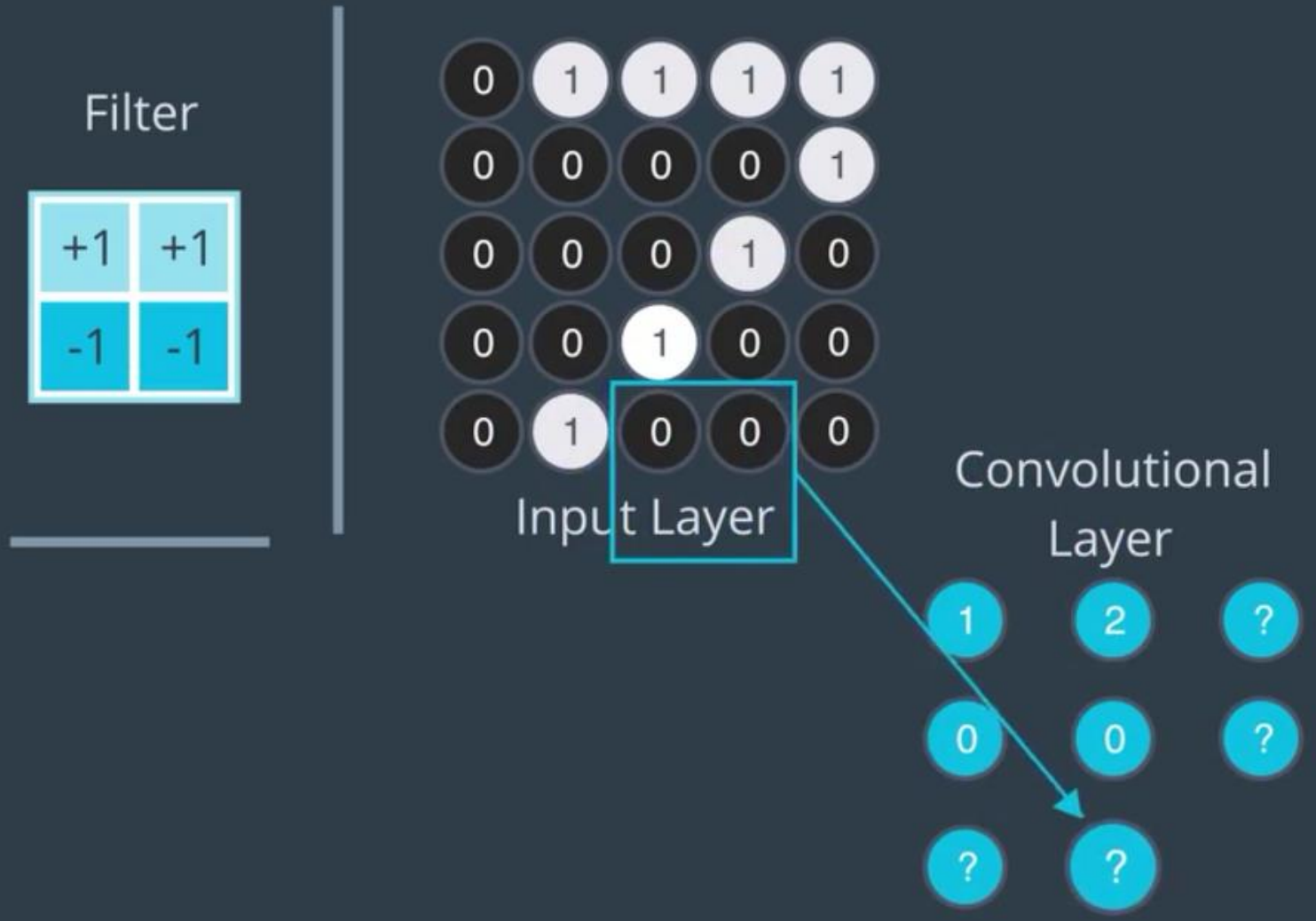
Stride & Padding



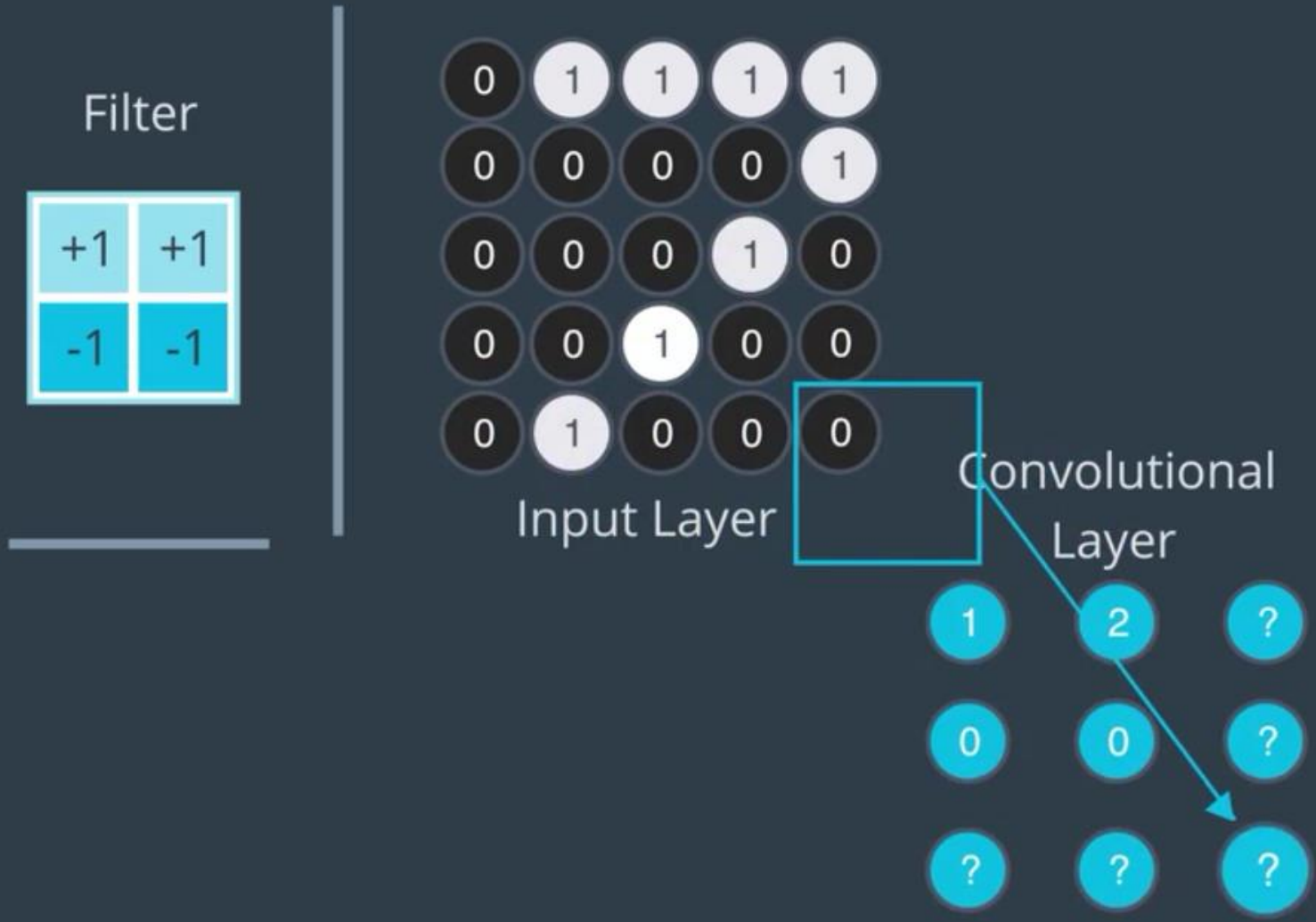
Stride & Padding



Stride & Padding



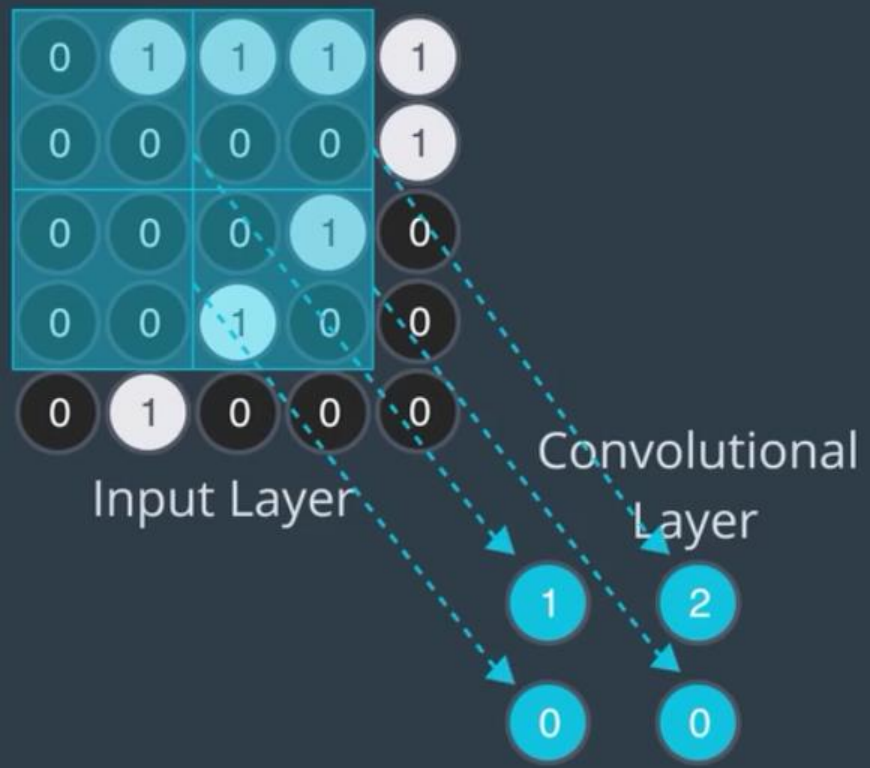
Stride & Padding



Stride & Padding

- But what about the edges? In other words, when the filter goes outside the image (but still has overlap with the image), what is the result of the convolution?

- One possible option is just removing them! But if we choose this option, it is possible that our convolutional layer has no information about some edges of the image.

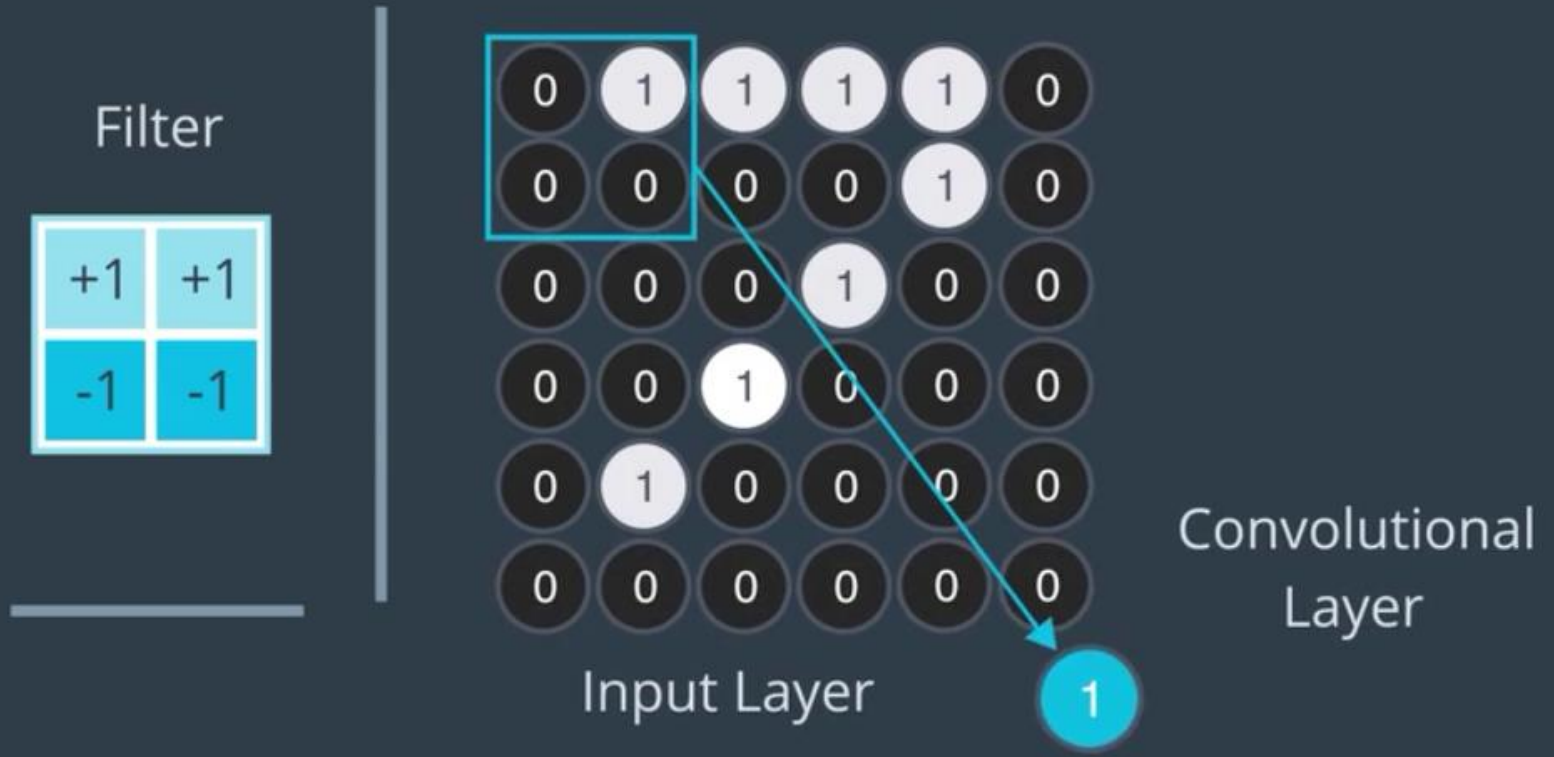


Stride & Padding

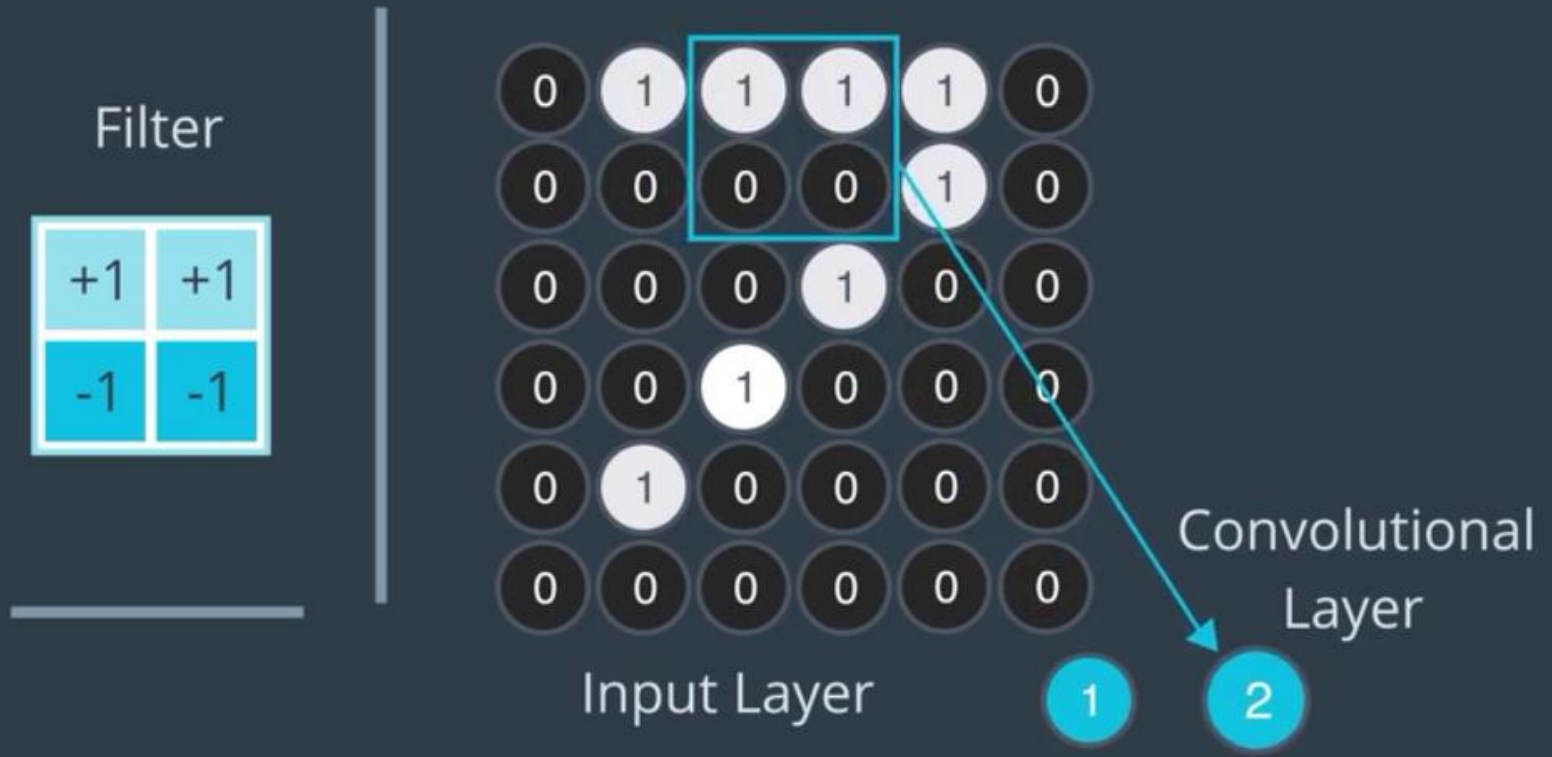
❑ Padding:

- The better option is padding the image with zeros to give the filter more space to move.
- By padding, we can now perform the convolution and make sure that every region in the image has its contribution in the output.
- We show the convolution operation with stride 2 and this time with padding, for the same example:

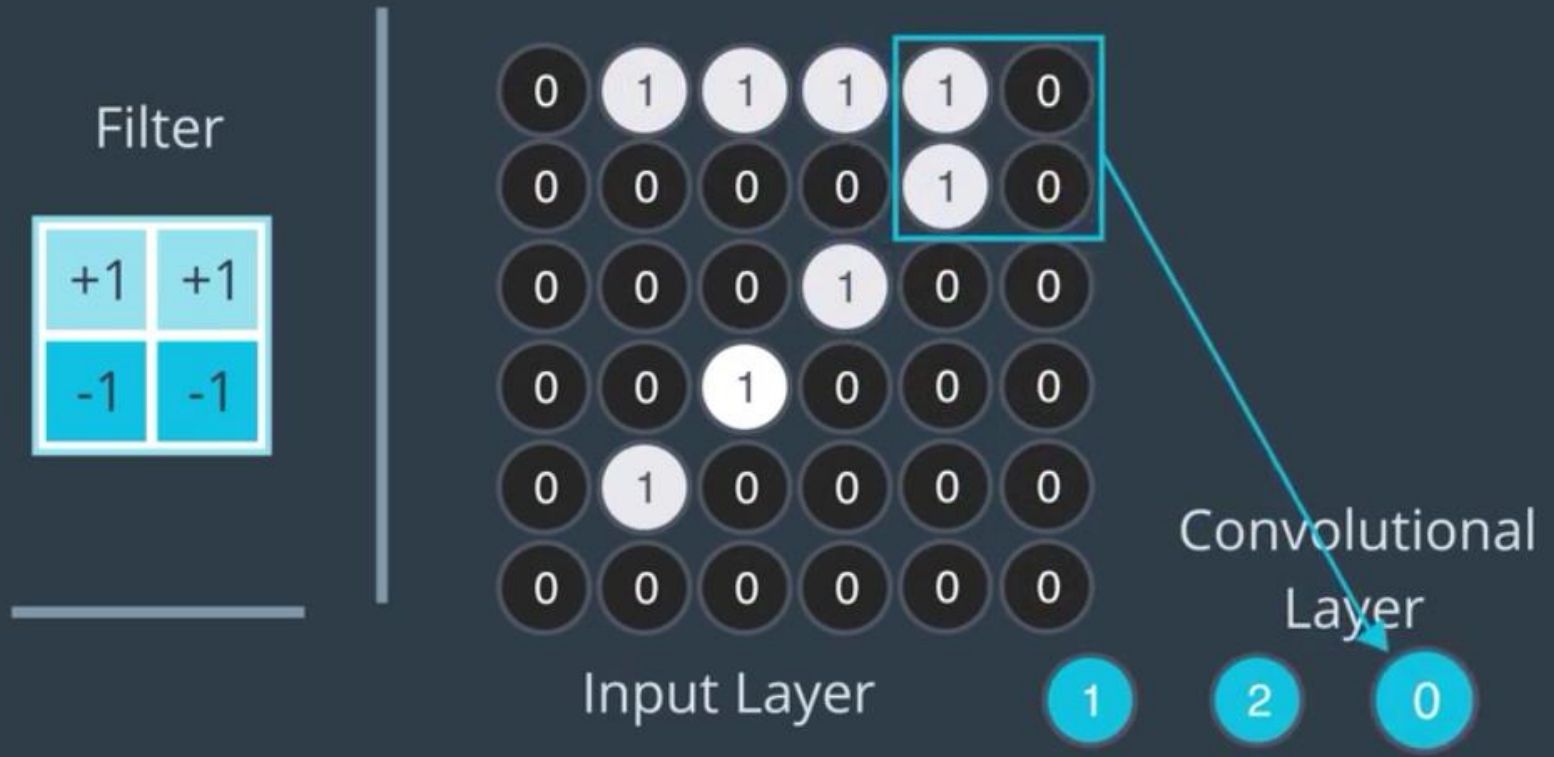
Stride & Padding



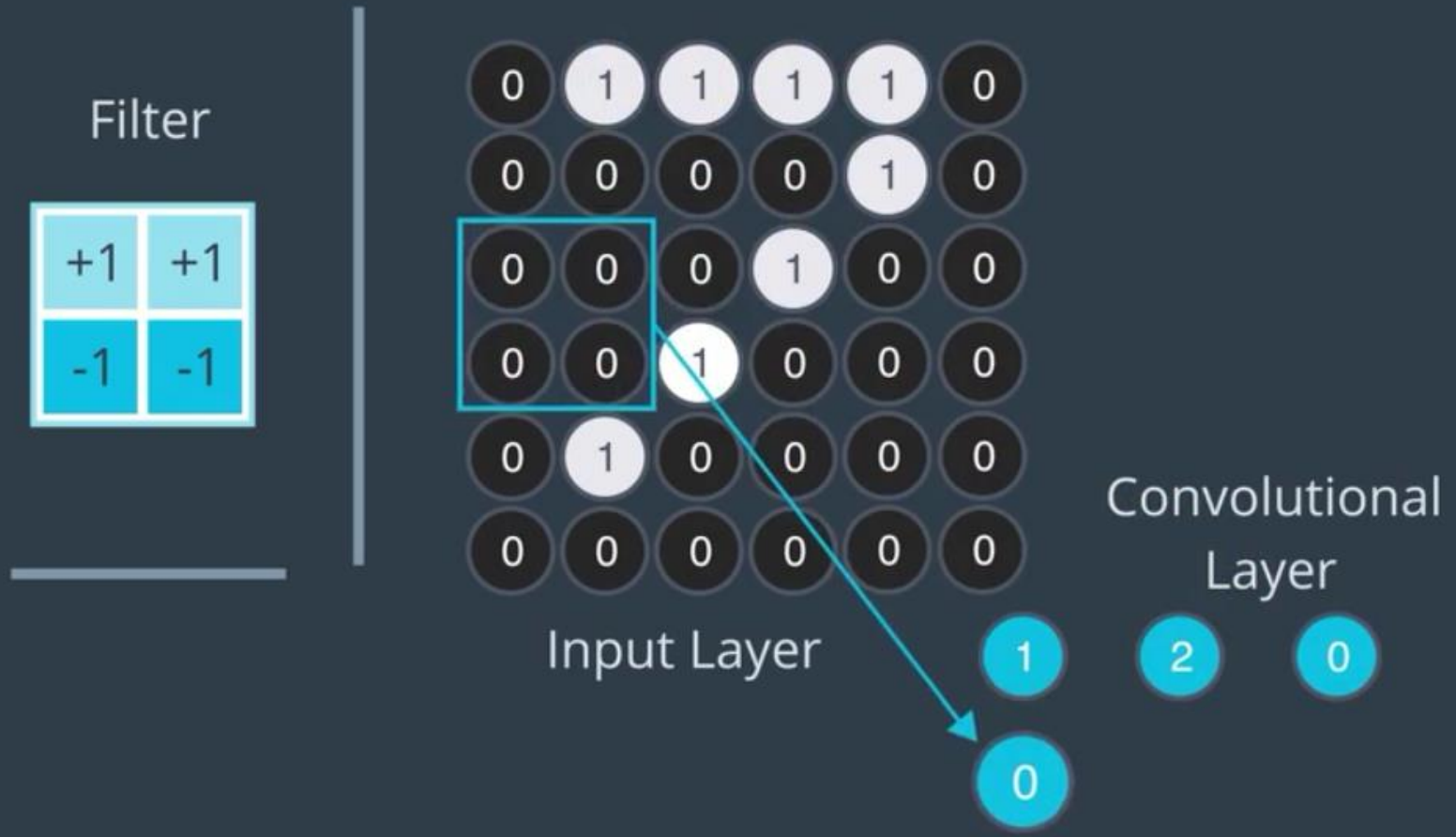
Stride & Padding



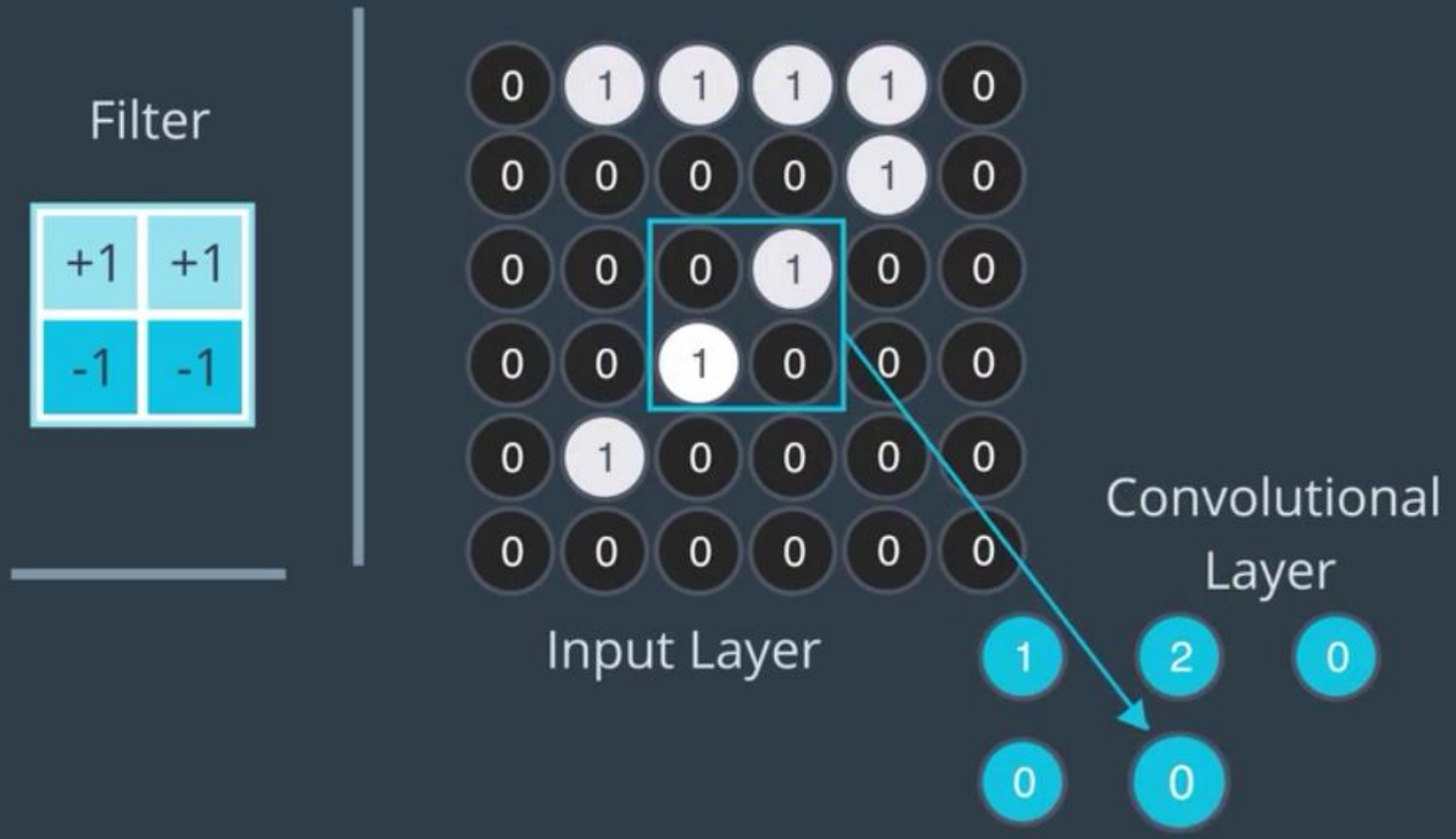
Stride & Padding



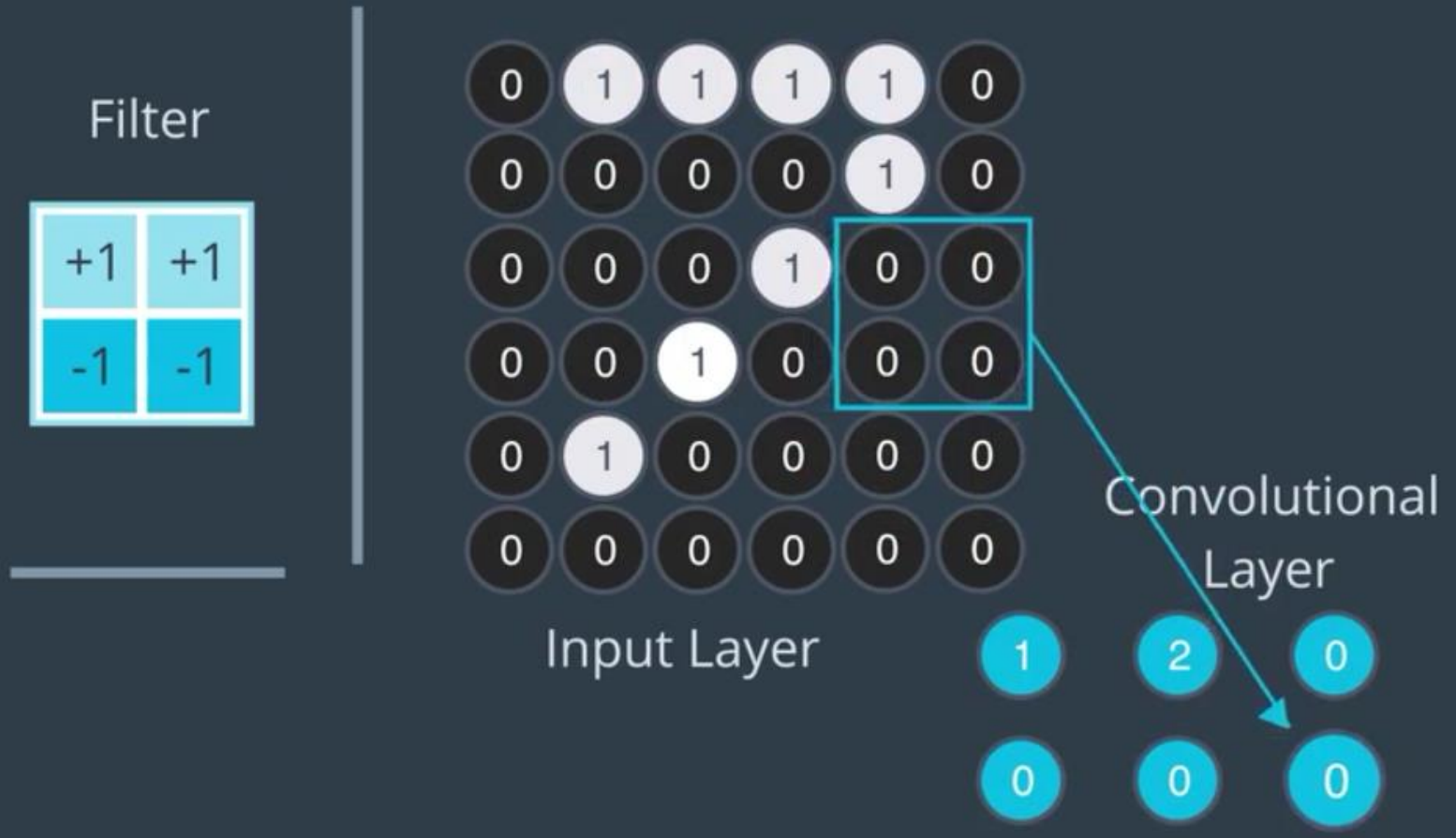
Stride & Padding



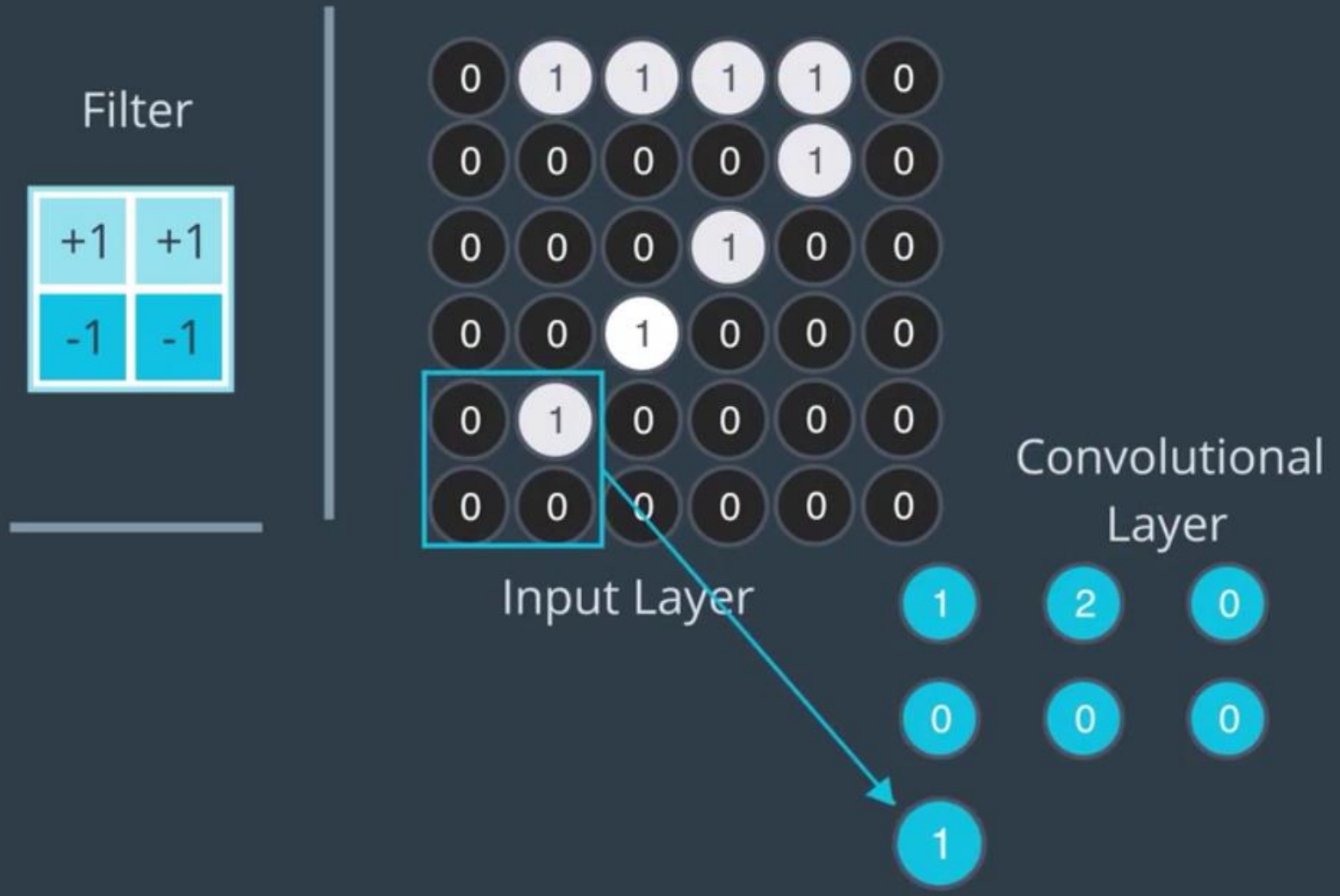
Stride & Padding



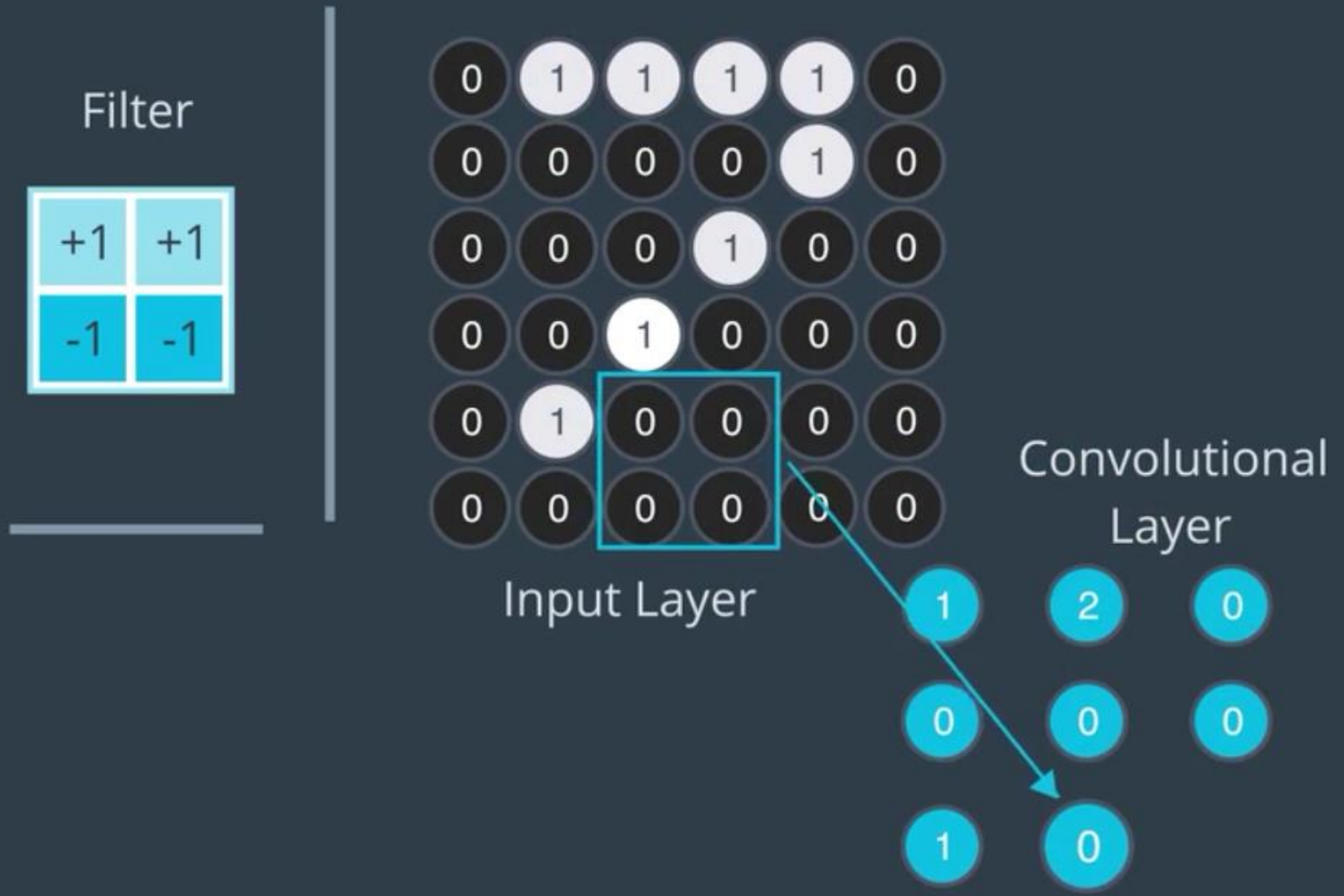
Stride & Padding



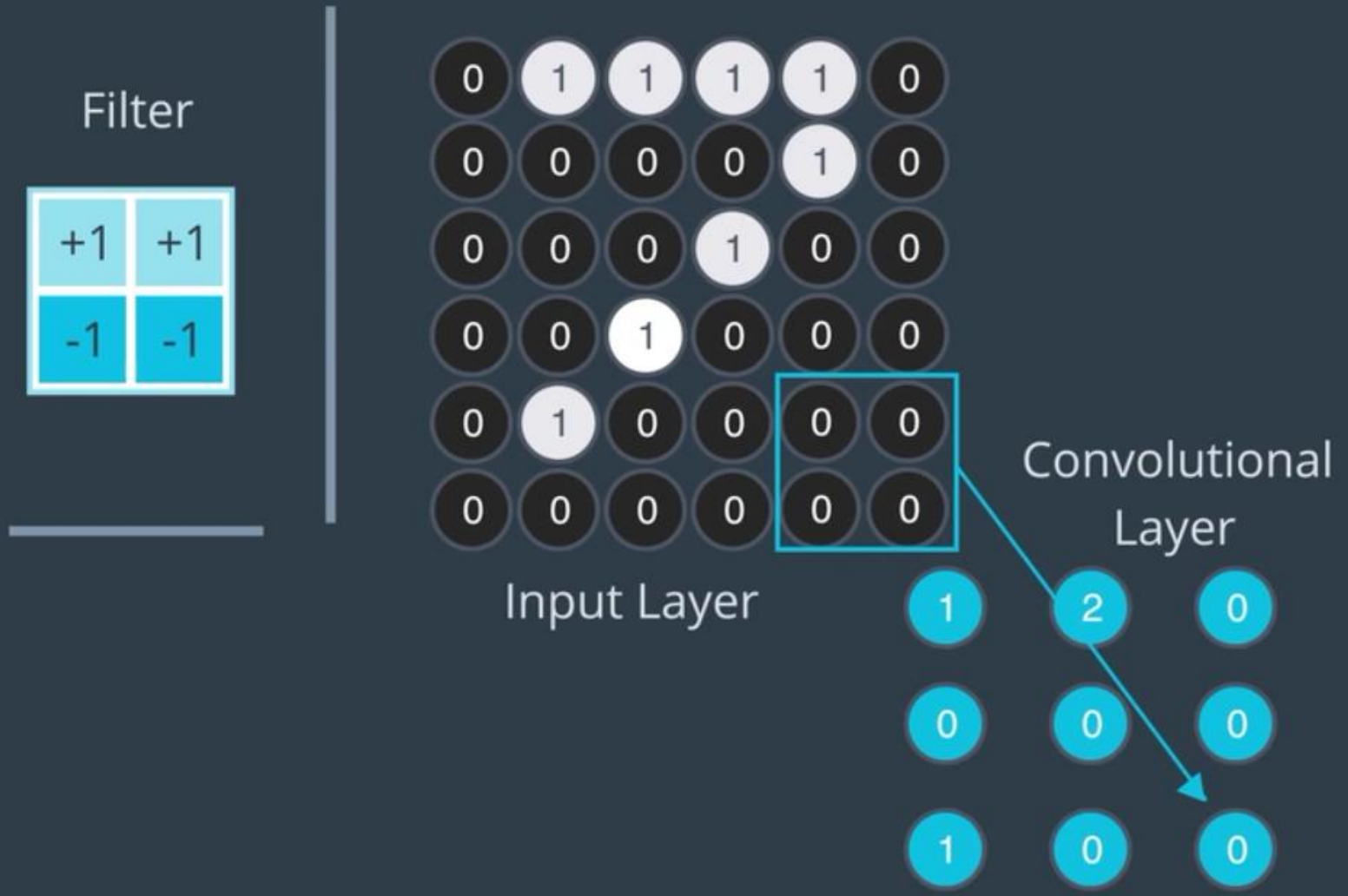
Stride & Padding



Stride & Padding



Stride & Padding



Stride & Padding

- A stride of one with appropriate padding makes the convolutional layer the same width and height as the input image.
- A stride of two with appropriate padding makes the convolutional layer, half the width and height of the input image.

Pooling

- As we saw, the output of a convolutional layer is a stack of feature maps where we have one feature map for each filter of that layer.
- A complicated dataset with many object categories will require a large number of filters, each responsible for finding a pattern in the image.
- More filters means a bigger stack, which means that the dimensionality of our convolutional layers can get large.
  - ✓ **Higher dimensionality ⇒ more parameters ⇒ may overfit**

Pooling

- The role of pooling layers within a convolutional neural network is reducing the dimensionality.
- Pooling layers often take convolutional layers as input.

Pooling

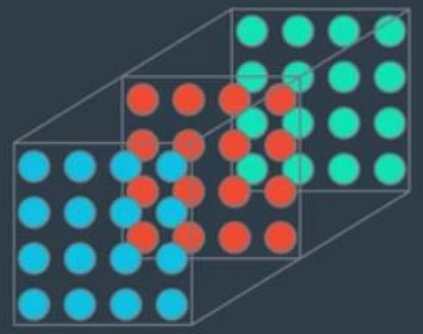
- In the pooling layer as convolutional layer we slide a rectangular window on the image (the output of convolutional layer) with some stride.
- There are 2 types of pooling layers: **max pooling** and **average pooling**
- **Max pooling operation:** takes a feature map (the result of convolution) and a window size (for example 3x3) as input and slides the window over the feature map horizontally and vertically just like the convolution operation.

Pooling

- The result of max pooling layer is calculated by just taking the maximum of the pixels contained in the window.
- **Max pooling layer:** takes a stack of feature maps as input and performs max pooling operation for each feature map separately.

➢ We will walk through an example of max pooling layer with 3 feature maps and window size=2 and stride=2.

Pooling

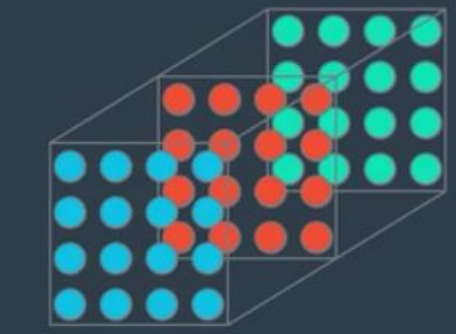


Convolutional Layer

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

Max Pooling Layer
WINDOW SIZE: 2X2
STRIDE: 2

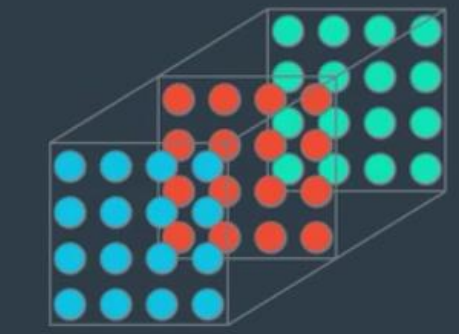
Pooling



Convolutional Layer

Max Pooling Layer
WINDOW SIZE: 2X2
STRIDE: 2

Pooling



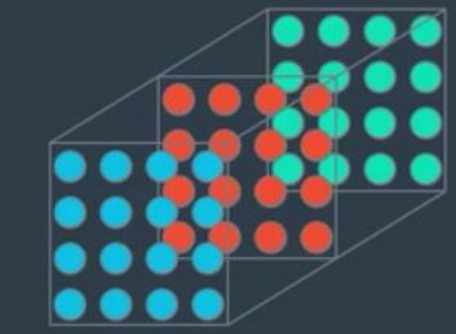
Convolutional Layer

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

9	8

Max Pooling Layer
WINDOW SIZE: 2X2
STRIDE: 2

Pooling



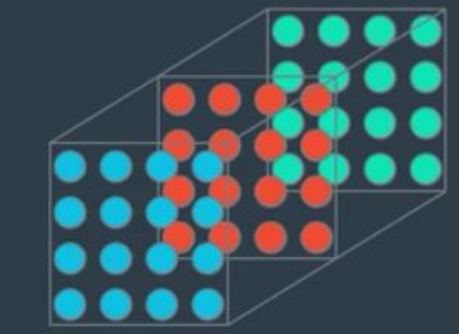
Convolutional Layer

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

9	8
7	

Max Pooling Layer
WINDOW SIZE: 2X2
STRIDE: 2

Pooling



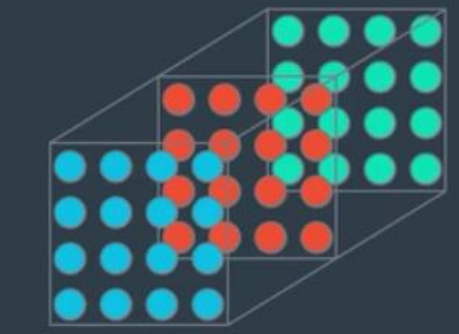
Convolutional Layer

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

9	8
7	9

Max Pooling Layer
WINDOW SIZE: 2X2
STRIDE: 2

Pooling



Convolutional Layer

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

9	8
7	9
9	7
8	9
7	9
8	9

Max Pooling Layer
WINDOW SIZE: 2X2
STRIDE: 2

Pooling

- The output of a max pooling layer is a stack with the same number of feature maps as input, but each feature map has been reduced in width and height.

Pooling

- The other type of pooling is average or mean pooling which is totally similar to max pooling except that average pooling return the average of values instead of their maximum.
- For the task of image classification and image segmentation, usually the max pooling is better because it notices better to the most important details about edges and other features in an image.
- For tasks like image smoothing, average pooling layer is preferable.

Training a CNN

- The process of training in a CNN and an MLP is the same.
- In a CNN, the numbers in each filter is learnable. For example, a 5x5 filter has 26 (25 weights and 1 bias) parameters that should be learned during the backpropagation.
- In the beginning, the filters (weights) are randomly generated and the patterns be detected using these (maybe) weak filters.
- Then based on the loss value the filters are updated at each epoch to take on values that minimize the loss.

Training a CNN

- We will not specify the values of the filters or tell the CNN what kind of patterns it needs to detect, these will be learned from the data.

- Since our CNN only accept a fixed size input and our images in the dataset may have different size, we should pick an image size and resize all of our images to that same size before applying to the CNN.
  - ✓ This is another preprocessing step like normalization.

Training a CNN

- Four types of layers (including the activation) can be seen in a CNN:

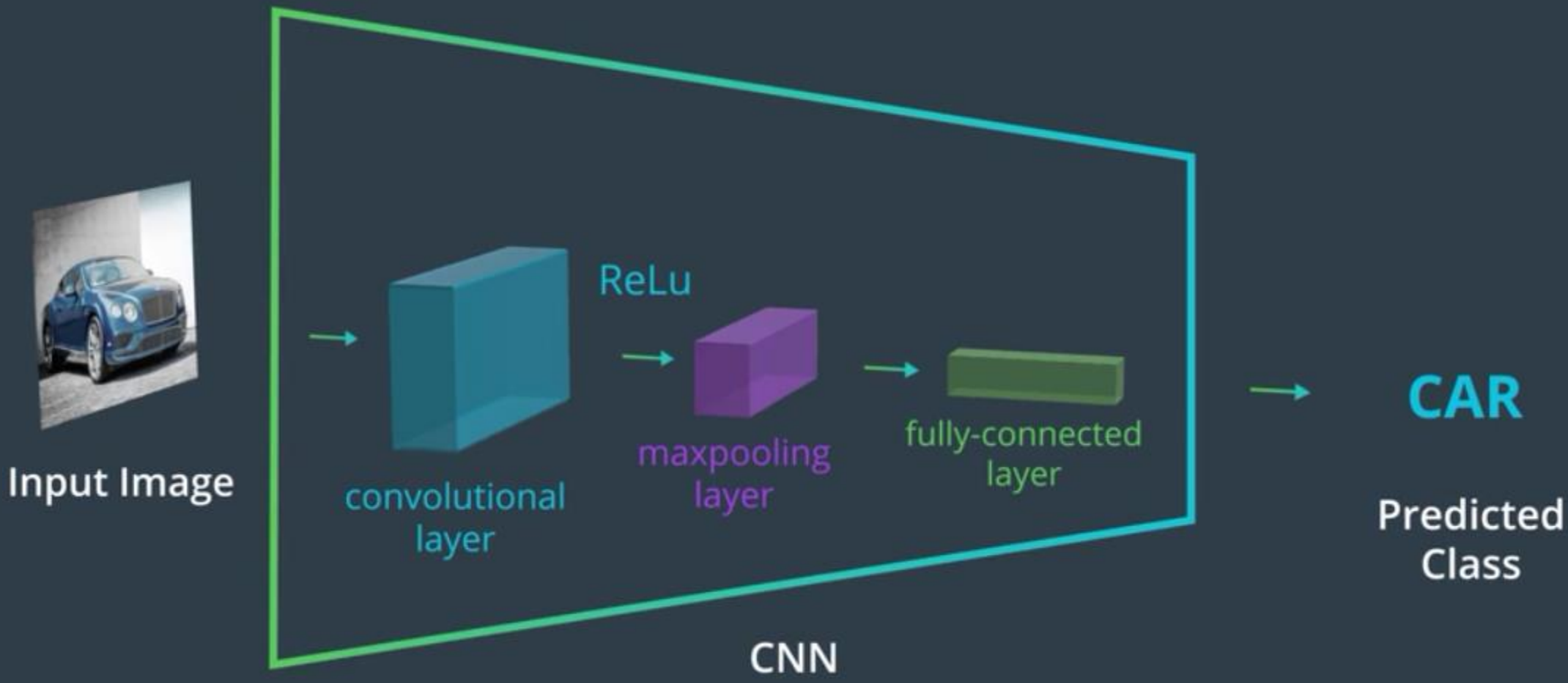
1. Convolutional layer detects regional patterns in an image using a series of image filters.

2. Typically a ReLU activation function is applied to the output of these filters to provide nonlinearity in their output values.

3. Pooling layer appears after convolutional layers to reduce the dimensionality of our input arrays.

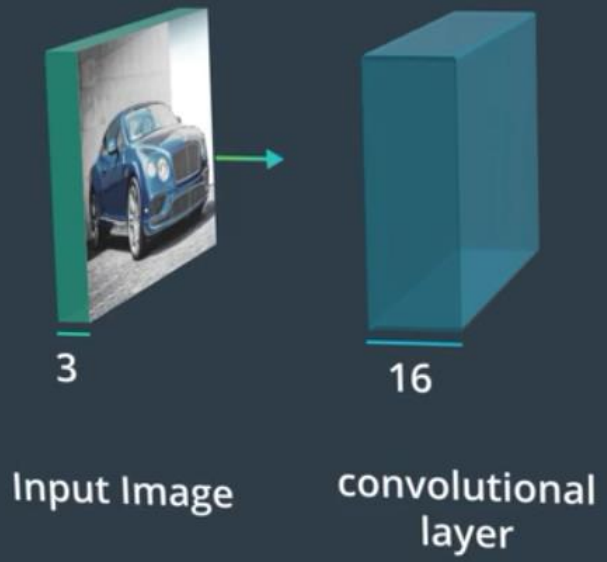
4. Finally some fully connected layers are employed to map the features extracted by convolutional layers to a more appropriate representation.

Training a CNN

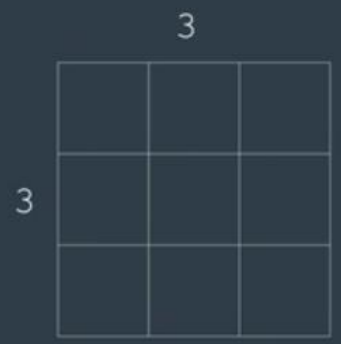


Training a CNN

- Each of the convolutional layer requires to specify a number of hyperparameters.
  - ✓ For example this convolutional layer accepts input with depth 3 (RGB image) and produces 16 filtered images based on a 3x3 kernel:




```
self.conv1 = nn.Conv2d(3,16,
                3,stride = 1, padding = 1)
```

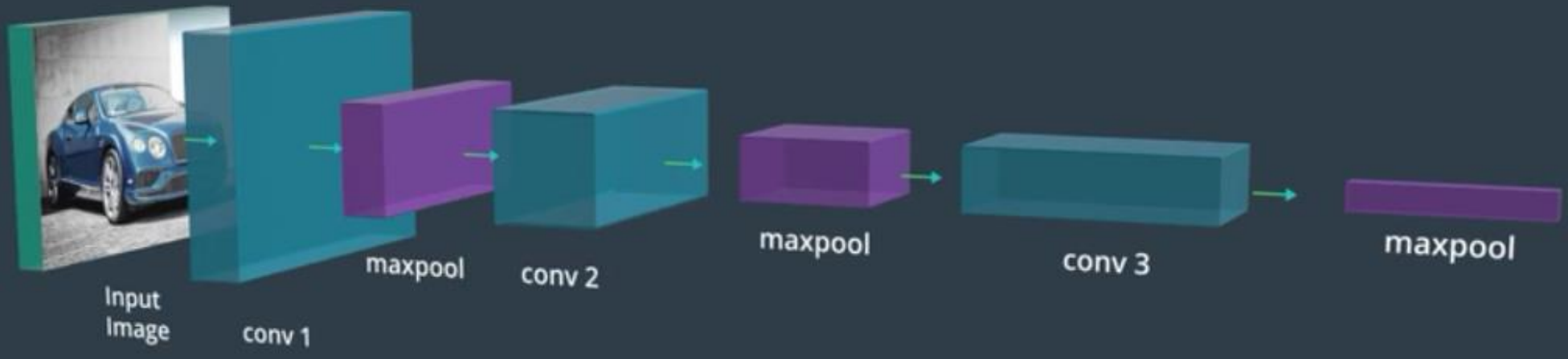


Training a CNN

- For deciding the depth or the number of filters in a convolutional layer, often we want to increase the number of filters in sequence.
  - ✓ For example the first conv has a depth of 16, the second has a depth of 32, the third 64 and so on.

- Since with this setting as we get deeper the width and height become larger, we add some pooling layers between the conv layers to decrease the dimensions and discard some irrelevant spatial information.
  - ✓ A pooling layer needs a window size and a stride to be defined.

Training a CNN

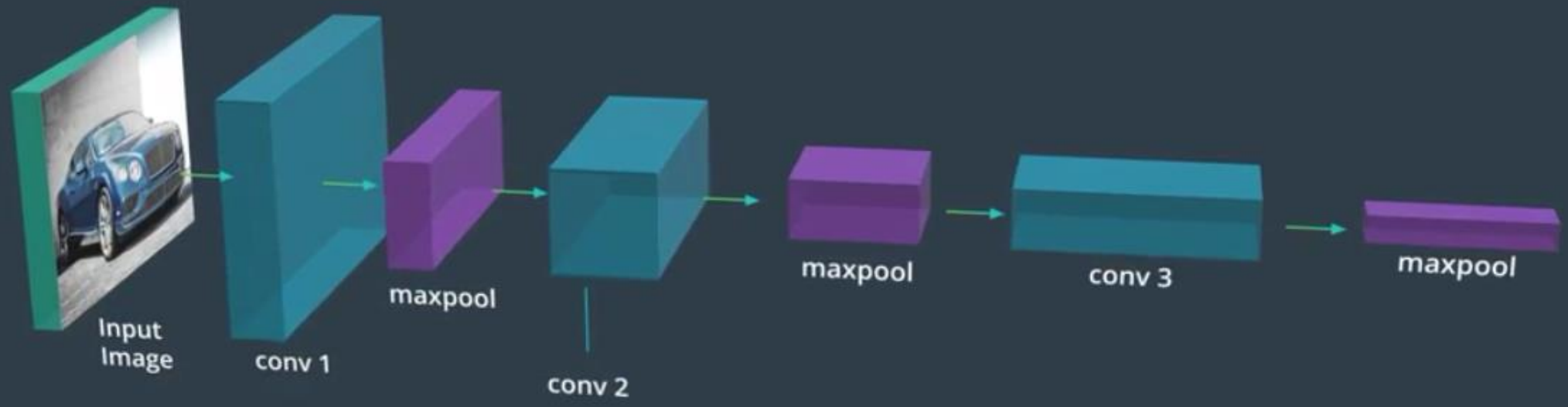



```
self.conv1 = nn.Conv2d(3, 16, 3, padding = 1)
self.conv2 = nn.Conv2d(16, 32, 3, padding = 1)
self.conv3 = nn.Conv2d(32, 64, 3, padding = 1)

self.maxpool = nn.MaxPool2d(2, 2)
```

Training a CNN

- As the network gets deeper,
  - ✓ it is extracting more and more complex patterns and features that help identify the content and the objects in an image.
  - ✓ It is discarding some spatial information about features like a smooth background that do not identify the image.



Pre-trained CNNs

- Suppose that we want to train a classifier that can distinguish between different types of cats that are very similar to each other:
  - ✓ One obvious solution is to collect a bunch of data and train a model on this dataset. But there is a problem, if our selected model is complicated, we have to collect too many data that is a difficult work.

- If we can not collect too many data, we have to simplify our model and therefore the performance may decrease.

- Of course solutions like data augmentation can help us, but a more powerful solution is introduced for these kind of problems, called **transfer learning**.

Pre-trained CNNs

- Transfer learning is to transfer knowledge learned from the source dataset to the target dataset.
  - ✓ A special kind of transfer learning is called **fine-tuning** that we pre-train a model on a source data and then tune it (train it more) on the desired target data.

- In our example, we can train a complicated model on ImageNet dataset as the source and then train more on our collected dataset that contains specific desired categories of cats as the target.

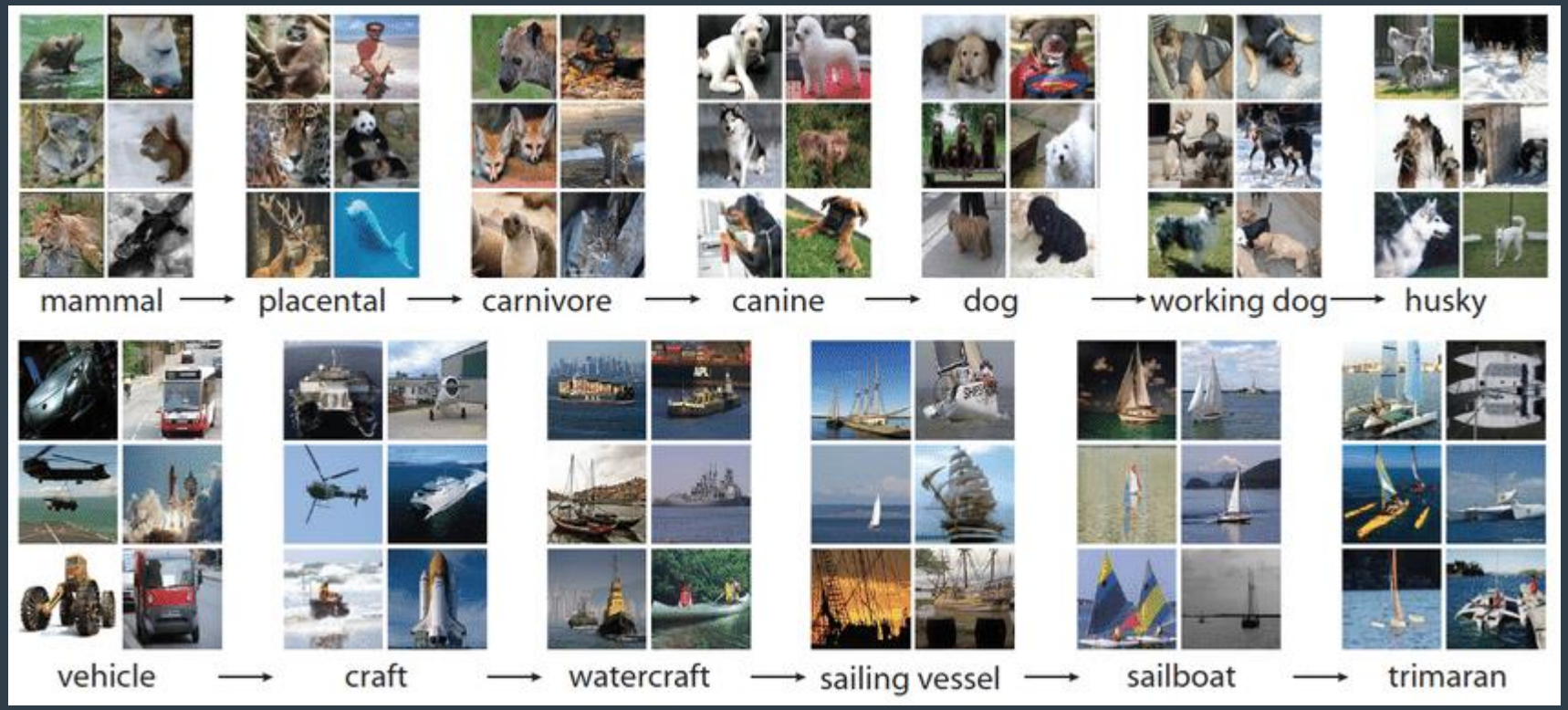
Pre-trained CNNs

- ImageNet dataset is the most widely used image dataset, which has more than 10 million images and 1000 categories. This dataset is usually used as the source dataset to apply transfer learning in image processing tasks.

- In our example, although most of the images in the ImageNet have nothing to do with cats, but when our model train on this data set, it will be able to extract more general image features which can help identify edges, textures, shapes and object composition.
  - ✓ These features may also be effective for recognizing cats in the target data.

ImageNet

A snapshot of two root-to-leaf branches of ImageNet

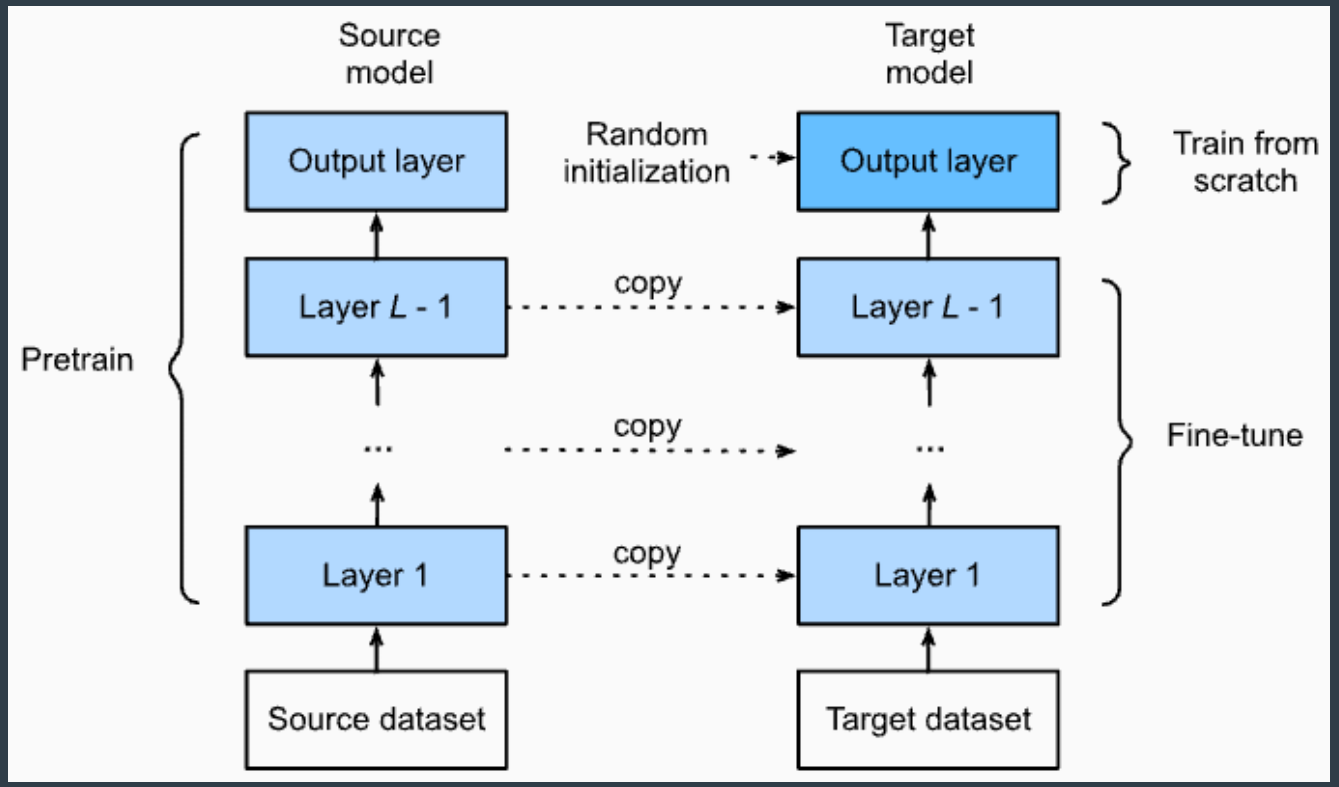


Pre-trained CNNs

- Fine-tuning steps:
  1. Pre-train a neural network on a source dataset (e.g. the ImageNet) and call it the source model.
  2. Create a neural network called the target model. This model copies all designs and parameters of the source model except the output layer.
  3. Add an output layer to the target model, whose number of outputs is the number of categories in the target dataset. Then randomly initialize the model parameters of this layer.
  4. Train the target model on the target dataset. The output layer will be trained from scratch, while the other parameters are fine-tuned based on the parameters of the source model or are frozen.

Pre-trained CNNs

- The process of fine-tuning:



Pre-trained CNNs

- In fact the common parameters between the source and the target model are the **knowledge** transferred from the source dataset.
- In recent years, too many pre-trained deep neural network models are introduced based on ImageNet which show good performance in different tasks.
- We will introduce the most popular pre-trained CNNs that are widely used in recent years including: GoogLe Net (Inception Net V1), VGG Net and ResNet.

Pre-trained CNNs

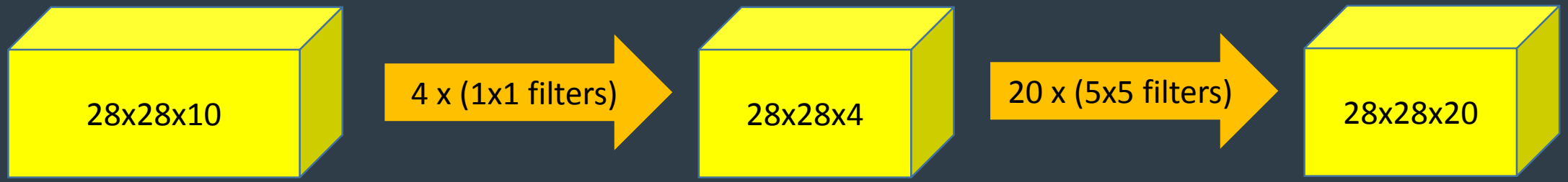
❑ GoogLe Net (Inception Net V1):

- This architecture was the winner at ILSVRC 2014 and used techniques such as 1x1 convolutions in the middle of the architecture and global average pooling.

➢ 1x1 convolution: this convolution is used to decrease the number of parameters and operations. For example if we want to perform 5x5 convolution having 20 filters without using 1x1 conv as intermediate we have 5x5x20x10=5000 weights:

Pre-trained CNNs

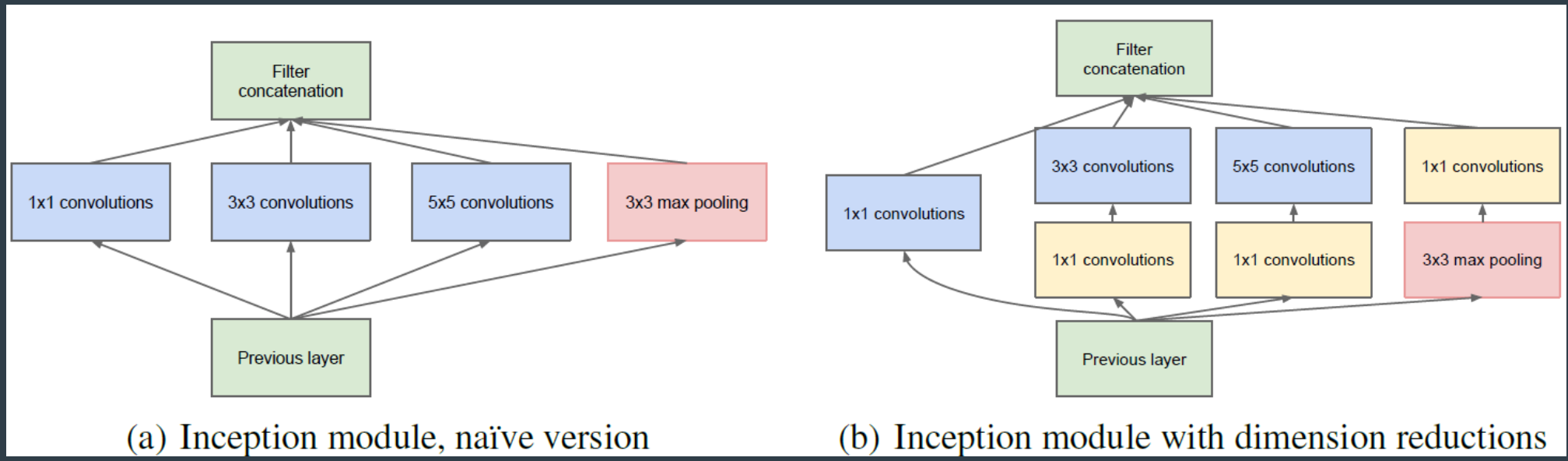
But if we use 1x1 conv as intermediate, we have 1x1x4x10 + 5x5x20x4 = 2040 weights:



- Global Average Pooling: this operation is used at the end of the network and takes a feature map of 7x7 and averages it to 1x1. in fact it converts each feature map matrix to a number.

Pre-trained CNNs

- Inception module: in this module, a 1x1, a 3x3 and a 5x5 convolution and a 3x3 max pooling performed in a parallel way at the input. The output of these are stacked together to generate the final output.



Pre-trained CNNs

- The idea behind this module is that conv filters of different sizes will handle objects at multiple scale better.

➢ Auxiliary classifier for training: these intermediate classifier branches in the middle of the architecture help in avoiding vanishing gradient and also provide regularization.

- these classifiers are used during training only and the generated loss of these layers will be added to the total loss with a weight of 0.3

Pre-trained CNNs

- This architecture takes image of size 224x224 with RGB color channels. All the convolutions inside, uses ReLU as activation function.

- GoogLe Net is 22 layers deep and is designed to be run on individual devices even with low computational resources.

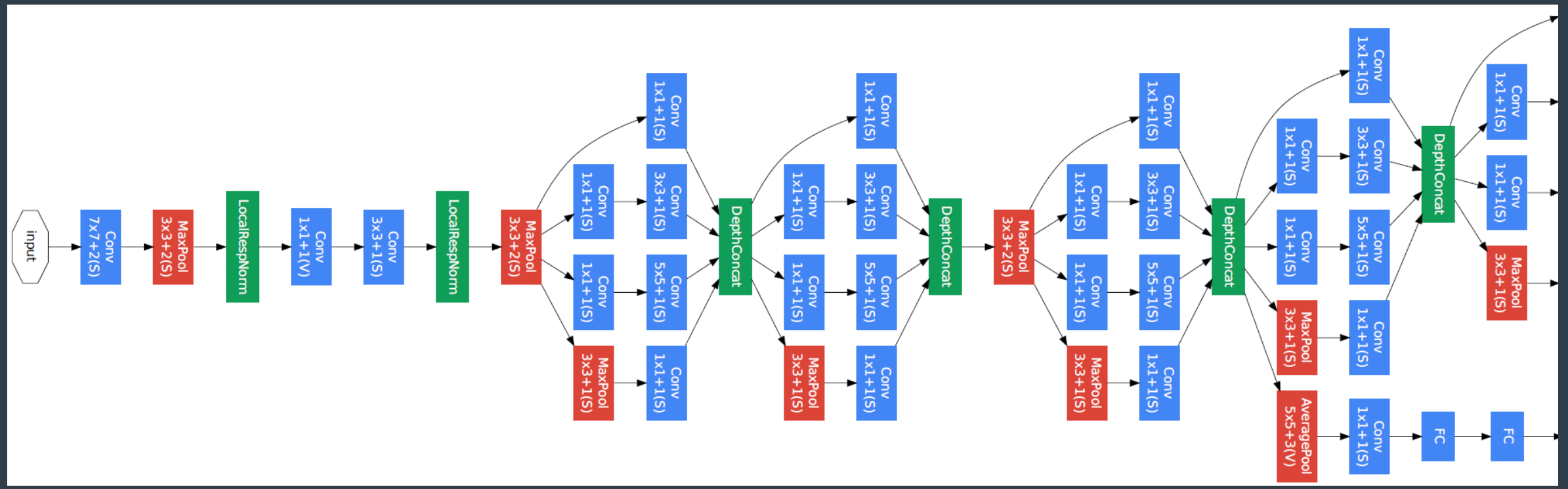
Pre-trained CNNs

GoogLe Net

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

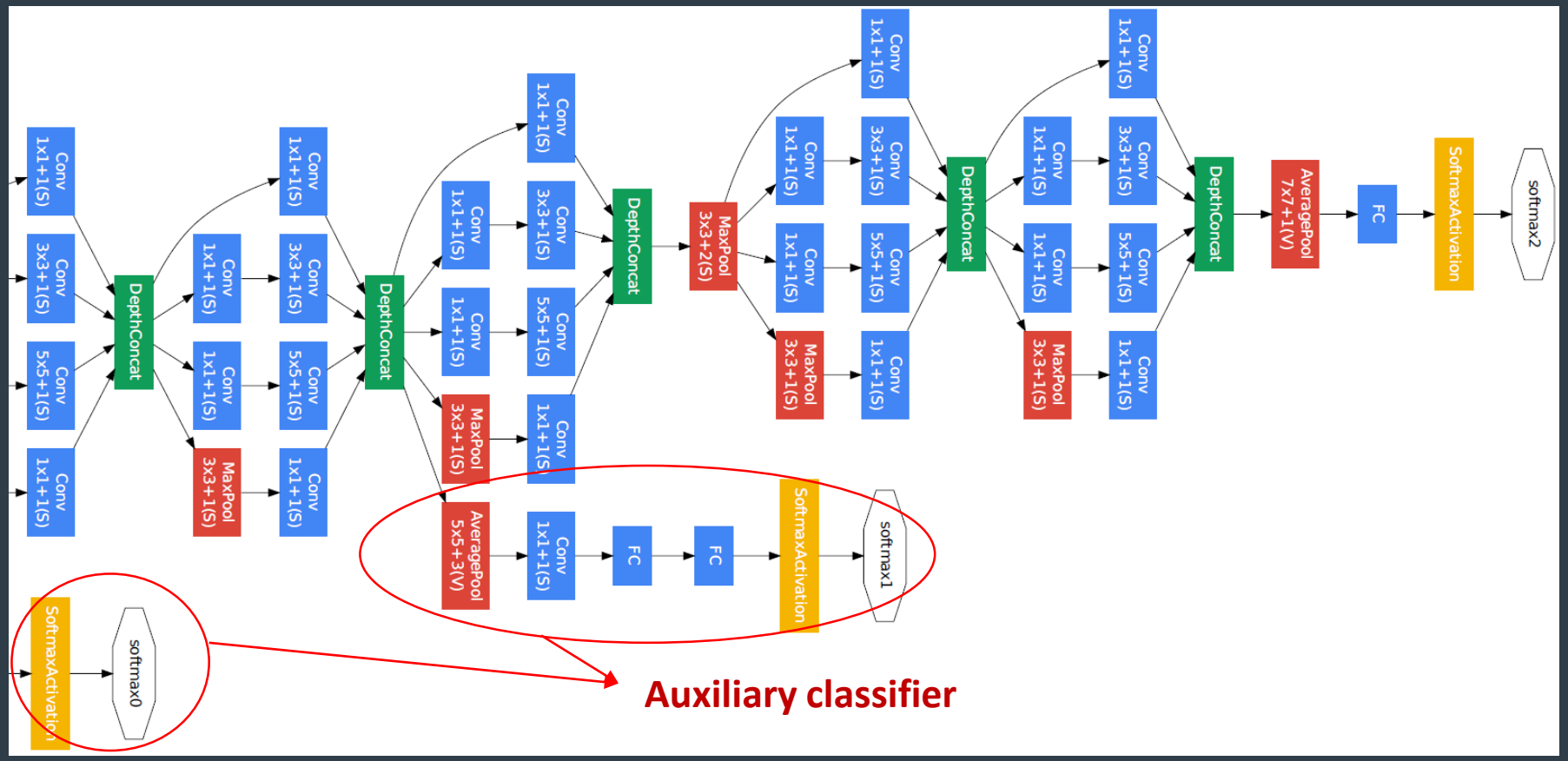
Pre-trained CNNs

- Google Net architecture:



Pre-trained CNNs

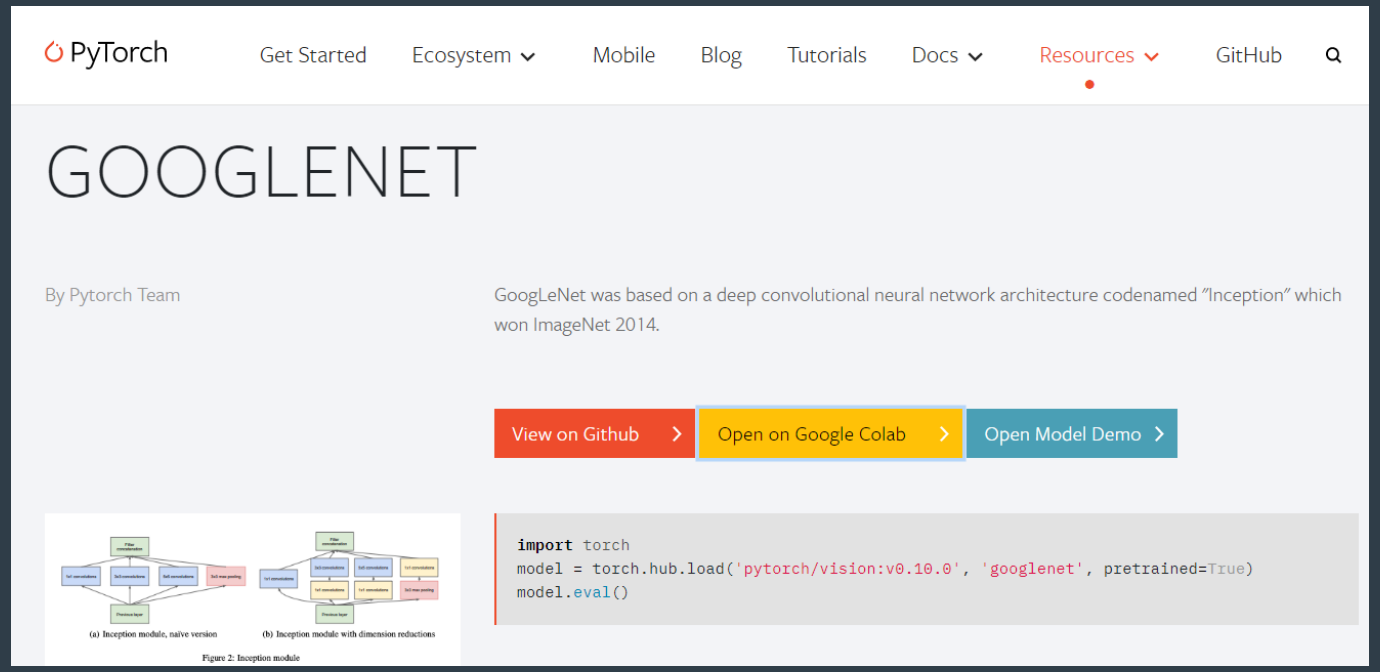
- Google Net architecture (Cont.):



Google Net in PyTorch

https://pytorch.org/hub/pytorch_vision_googlenet/

```
import torch
model = torch.hub.load('pytorch/vision:v0.10.0', 'googlenet', pretrained=True)
model.eval()
```



Pre-trained CNNs

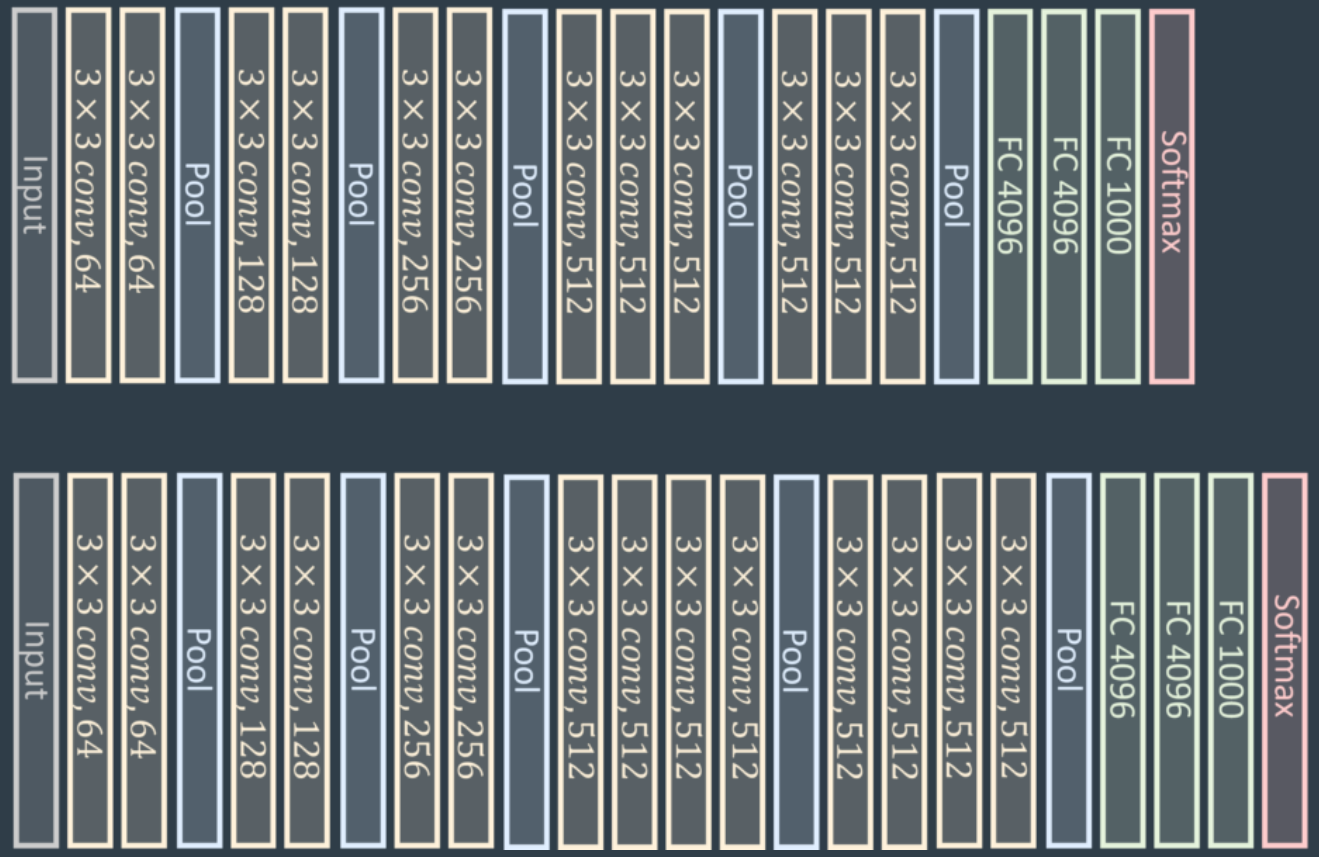
❑ VGG Net:

- This architecture was ILSVRC 2014 runner-up. In order to deepen the number of network layers and to avoid too many parameters, a small 3x3 kernel (in comparison with AlexNet with 7x7 kernels) is used in all layers.

- The most popular versions of VGG are VGG-16 with 16 layers and VGG-19 with 19 layers.

- Both VGG-16 and VGG-19 include 5 sets of convolutional layers, each followed by a max pooling and 3 dense layers followed by Softmax. The depth of conv layers in each set is 64, 128, 256, 512 and 512 respectively.

Pre-trained CNNs



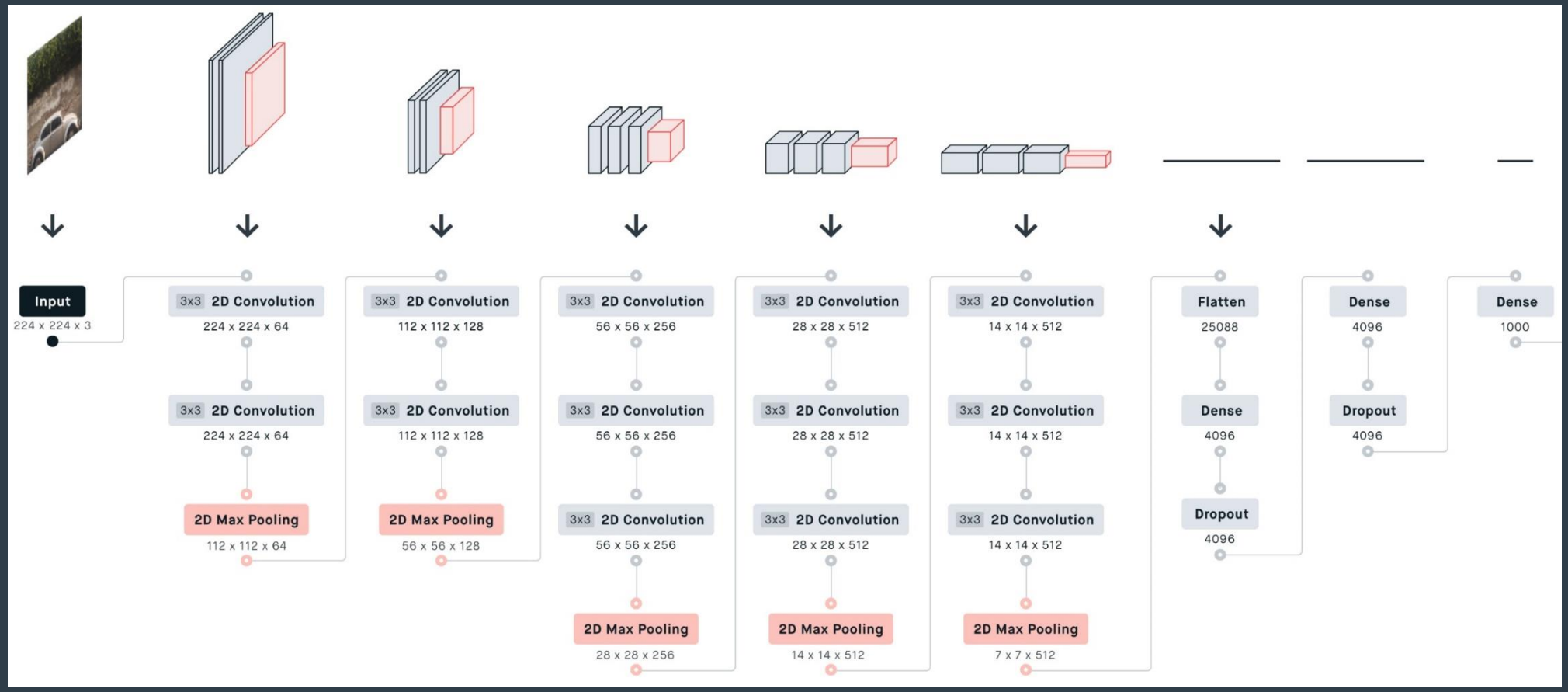
Pre-trained CNNs

- VGG-16 architecture:

Layer		Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	224 x 224 x 3	-	-	-
1	2 X Convolution	64	224 x 224 x 64	3x3	1	relu
	Max Pooling	64	112 x 112 x 64	3x3	2	relu
3	2 X Convolution	128	112 x 112 x 128	3x3	1	relu
	Max Pooling	128	56 x 56 x 128	3x3	2	relu
5	2 X Convolution	256	56 x 56 x 256	3x3	1	relu
	Max Pooling	256	28 x 28 x 256	3x3	2	relu
7	3 X Convolution	512	28 x 28 x 512	3x3	1	relu
	Max Pooling	512	14 x 14 x 512	3x3	2	relu
10	3 X Convolution	512	14 x 14 x 512	3x3	1	relu
	Max Pooling	512	7 x 7 x 512	3x3	2	relu
13	FC	-	25088	-	-	relu
14	FC	-	4096	-	-	relu
15	FC	-	4096	-	-	relu
Output	FC	-	1000	-	-	Softmax

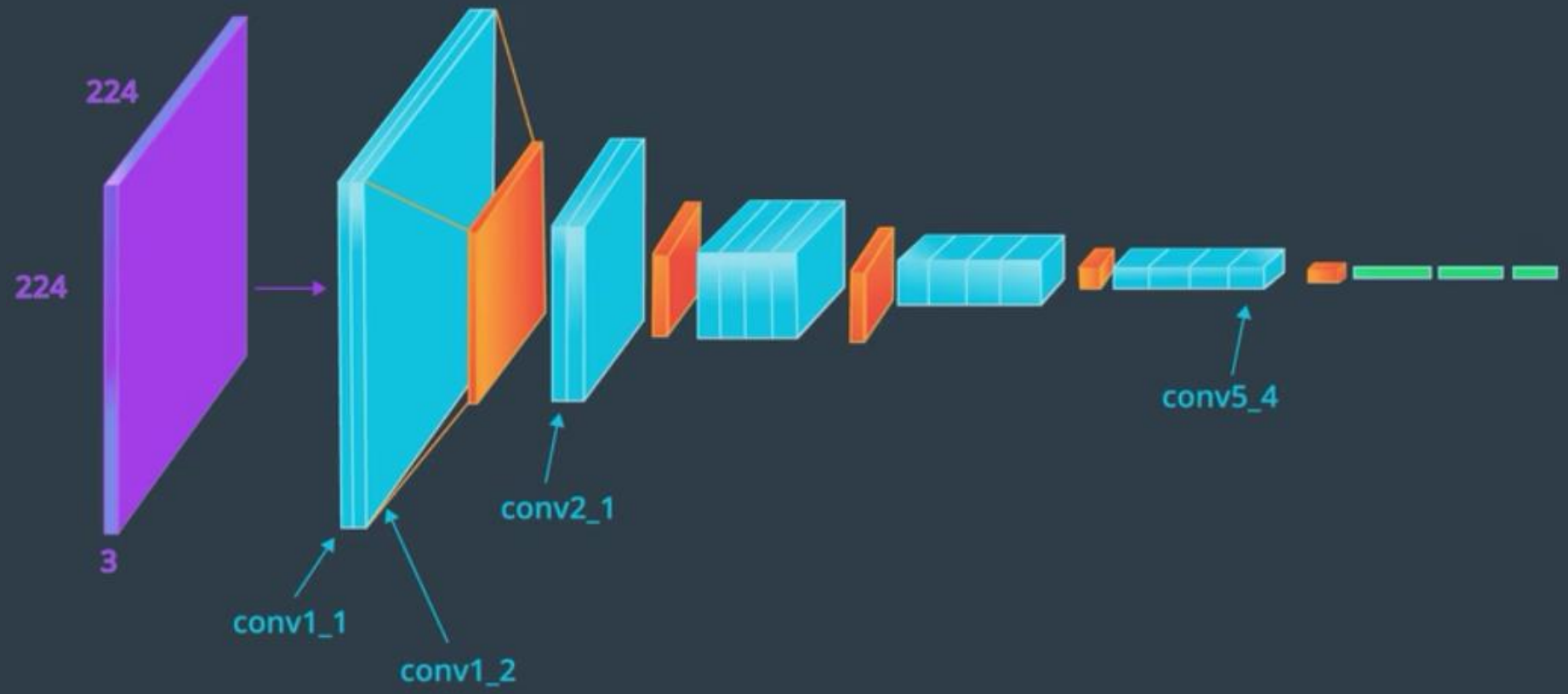
Pre-trained CNNs

- VGG-16 architecture:



Pre-trained CNNs

- VGG-19 architecture:

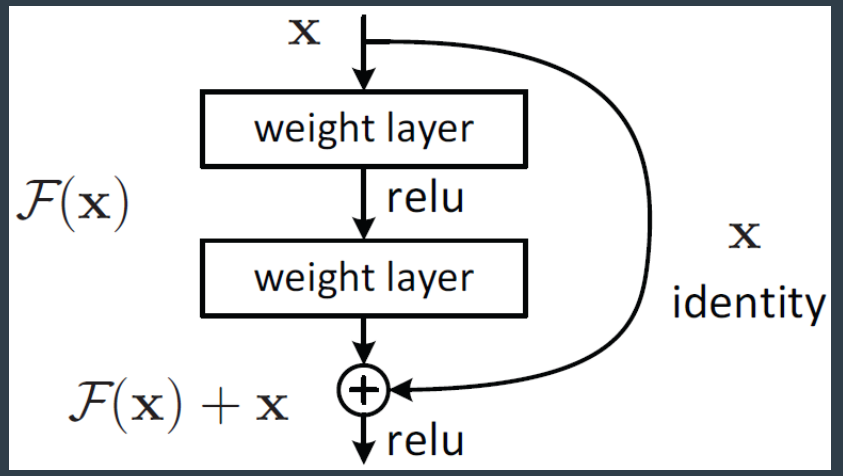


Pre-trained CNNs

❑ ResNet:

- This architecture was the winner at ILSVRC 2015 and had an important innovation called skip (residual) connection.

➢ Skip connection (residual block): this block connects activations of a layer to further layers by skipping some layers in between.



Pre-trained CNNs

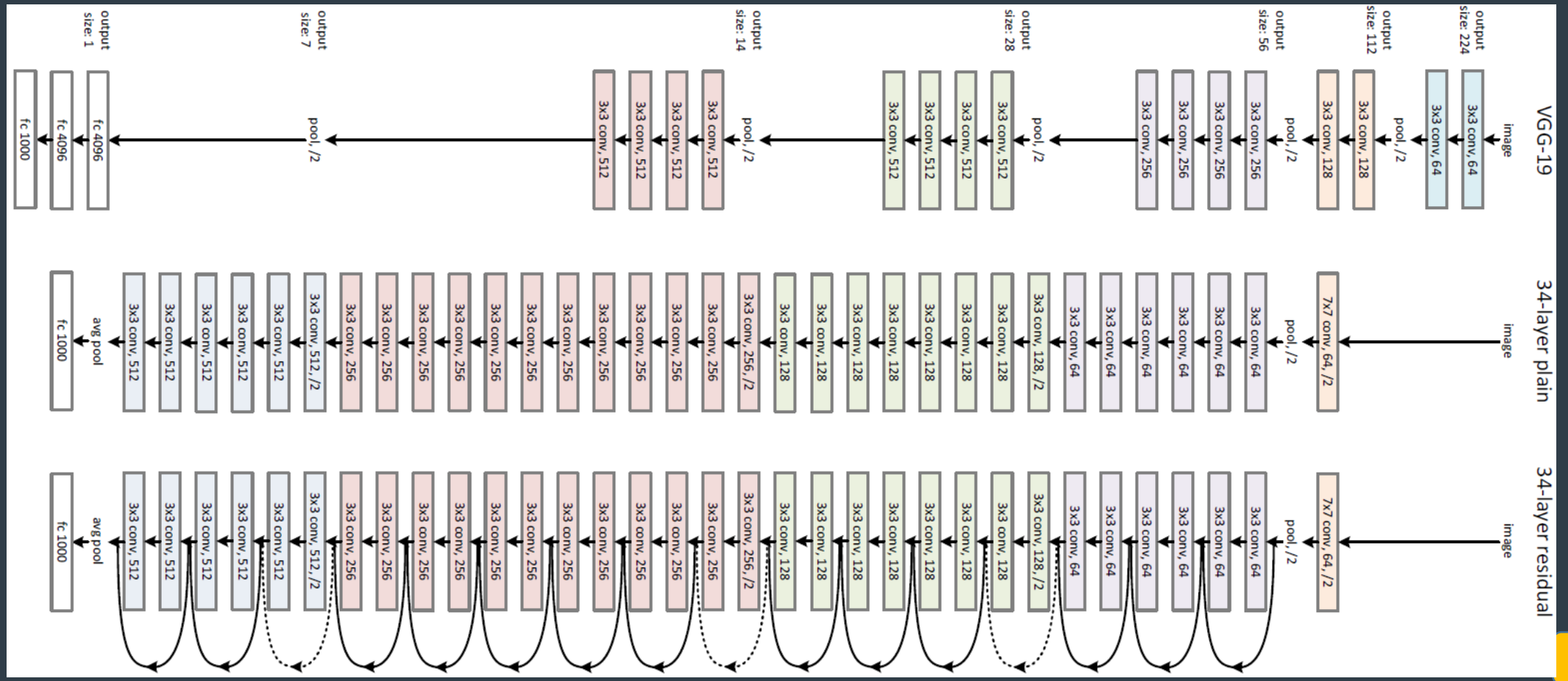
- ResNet is made by stacking these residual blocks together. The idea behind this network is instead of having layers learning the underlying mapping (say H(x)), we allow the network to fit the residual mapping (F(x)=H(x)-x).

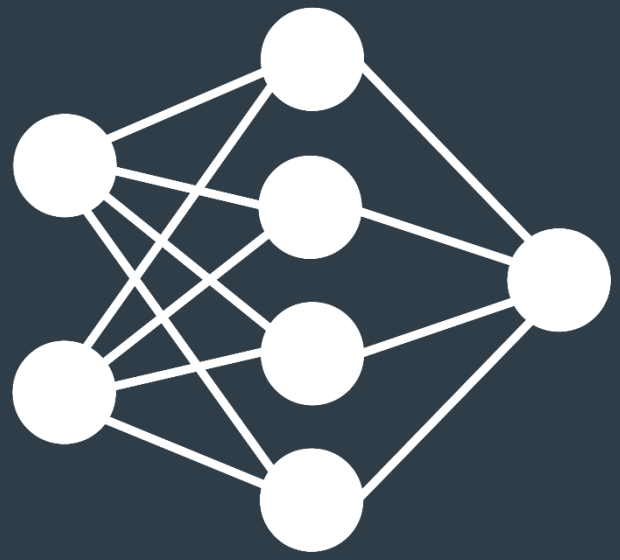
- The advantage of adding skip connection is that if any layer hurt the performance of architecture then it will be skipped by regularization.
  - ✓ This result in training a very deep neural network without the problems caused by vanishing/exploding gradients.

- ResNet uses a 34-layer plain network architecture inspired by VGG-19, but the residual connections are also added.

Pre-trained CNNs

- ResNet in comparison with VGG-19:





Thanks for your attention

End of chapter 9

Hamidreza Baradaran Kashani