M03S01- Introduction to Convolutional Neural Networks

Deep Learning & Neural Networks - Convolutional Neural Networks

Introduction

Convolutional Neural Networks, or CNNs, are neural networks specialised to work with **visual data**, i.e. images and videos (though not restricted to them). They are very similar to the vanilla neural networks (multilayer perceptrons) - every neuron in one layer is connected to every neuron in the next layer, they follow the same general principles of forward and backpropagation, etc. However, there are certain features of CNNs that make them perform extremely well on image processing tasks.

This module covers the working principles of CNNs, compare various CNN architectures and choosing the right architecture for specific tasks. **Transfer learning** covers use of large pre-trained networks for computer vision tasks. You will also be able to train CNNs using Python + Keras.

In this session

The first session covers the following topics:

1. The need for a new type of architecture for images
2. Reading digital images
3. Understanding convolutions
4. The structure and topology of convolutional neural networks
5. Feature Maps
6. Training the network

Prerequisites

There are no prerequisites for this session other than knowledge of the matrix multiplication and previous courses on Neural Networks and, Statistics and ML.

A Specialised Architecture for Visual Data

Convolutional Neural Networks, or CNNs, are specialised architectures which work particularly well with **visual data**, i.e. images and videos. They have been largely responsible for revolutionizing 'deep learning' by setting new benchmarks for many **image processing** tasks that were very recently considered extremely hard.

Common Challenges in Image Processing

Common task of **visual recognition** (like identifying a ‘cat’ or a ‘dog’) - trivial as it is for humans, is still a big challenge for algorithms. Let’s look at some of the challenges:

- Viewpoint variation: Different orientations of the image with respect to the camera.

Viewpoint variation

- Scale variation: Different sizes of the object with respect to the image size.

Scale variation

- Illumination conditions: Illumination effects.

Illumination condition

- Background clutter: Varying backgrounds.

Background clutter

CNNs - A Specialised Architecture for Visual Data

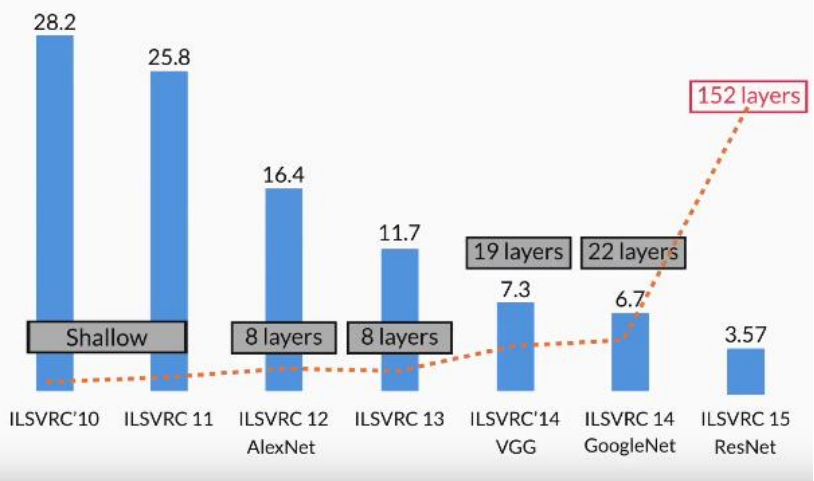
Although vanilla neural networks (MLPs) can learn extremely complex functions, their architecture does not exploit what we know about how the brain reads and processes images. For this reason, although MLPs are successful in solving many complex problems, they haven't been able to achieve any major breakthroughs in the image processing domain.

On the other hand, the architecture of CNNs uses many of the working principles of the **animal visual system** and thus they have been able to achieve extraordinary results in image-related learning tasks.

The ImageNet Challenge

CNNs had first demonstrated their extraordinary performance in the **ImageNet Large Scale Visual Recognition Challenge** (ILSVRC). The ILSVRC uses a list of about 1000 image categories or "classes" and has about 1.2 million training images. The original challenge is an **image classification** task.

You can see the impressive results of CNNs in the ILSVRC where they now outperform humans (having 5% error rate). The error rate of the ResNet, a recent variant in the CNN family, is close to 3%.

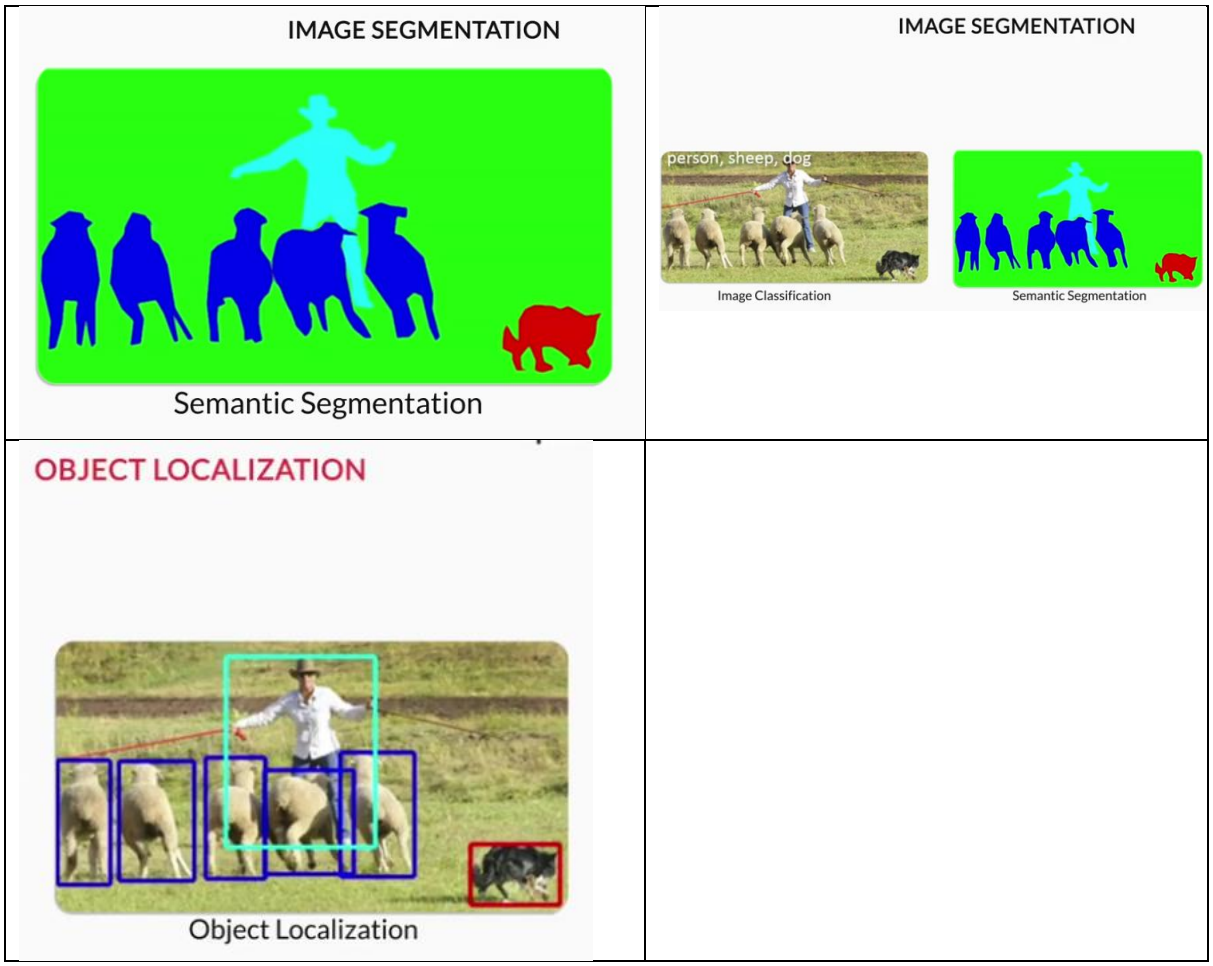


ImageNet challenge

Next segment covers different ways in which CNNs are used for image-processing tasks.

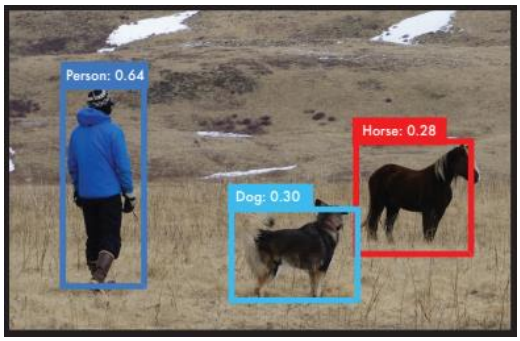
Applications of CNNs

This segment covers some common **types of image processing tasks** which can be solved using CNNs.



Some applications that we have discussed are:

- **Object localization** -Identifying the local region of the objects (as a rectangular area) and classifying them.



Object localisation

- **Semantic segmentation:** Identifying the exact shapes of the objects (pixel by pixel) and classifying them.

Semantic segmentation

- **Optical Character Recognition (OCR):** Recognise characters in an image. For e.g. for the top-left image, the output will be '1680'.

OCR

Let's see some more examples of image processing applications.

There are various other applications of CNNs in the healthcare sector. Many medical imaging applications used in radiology, cardiology, gastroenterology etc. involve classification, detection, and segmentation of objects which can be analysed using CNNs.

Next segment covers the motivation behind CNNs' architecture coming from the **visual system of mammals**.

Applications of CNN

Which of the following types of data can be analysed using CNNs? More than one options may be correct.

- [x] Images ✓

  Feedback :
  CNNs are most commonly used for analysing images.

- [x] Video ✓

  Feedback :
  A video is a series of images (or frames). CNNs are commonly used for processing videos.

- [x] Audio ✓

  Feedback :
  CNN can also be used for audio processing.

- [x] Text ✓

  Feedback :
  CNNs can also be applied to text, although their use is limited in this area.

Understanding the Visual System of Mammals - I

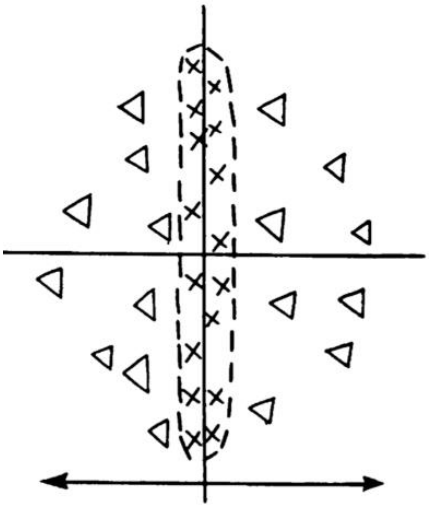
Architecture of CNNs is motivated by the **visual system of mammals**. This segment will discuss an influential paper named "Receptive field for single neurons in the cat's striate cortex" published by **Hubel and Wiesel**.

This was basically a bunch of experiments conducted to understand the visual system of a cat. In the experiments, spots of light (of various shapes and size) were made to fall on the retina of a cat and, using an appropriate mechanism, the response of the **neurons in the cat's retina** was recorded. This provided a way to observe which types of spots make some particular neurons 'fire', how groups of neurons respond to spots of certain shapes, etc.

Some of the important observations made in the study were:

1. Each neuron in the retina focuses on one part of the image and that part of the image is called the **receptive field of that neuron.**
2. There are **excitatory and inhibitory regions** in the receptive field. The neurons only 'fire' when there is a **contrast between the excitatory and the inhibitory regions.** If we splash light over the excitatory and inhibitory regions together, because of no contrast between them, the neurons don't 'fire' (respond). If we splash light just over the excitatory region, neurons respond because of the contrast.

The figure below shows a certain region of the receptive field of a cat. The inhibitory region (denoted by the triangular marks) is at the centre surrounded by the excitatory region marked by the crosses.



Receptive field of cat

1. The **strength of the response** is proportional to the summation over only the excitatory region (not inhibitory region). Later, you will study the **pooling layer in CNNs** which corresponds to this observation.

Next segment covers observations from this study that influenced the CNN architecture.

Receptive Field

Each neuron in the retina is trained to 'look at':

- The entire image
- **A particular patch (region) of the image** ✓

  Feedback :
  Each neuron is trained to look at only a certain patch of the image. This patch is called the receptive field of that neuron.

Understanding the Visual System of Mammals

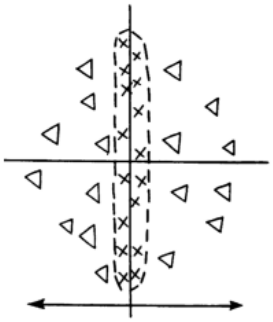
The excitatory and the inhibitory regions are:

- Regions in the receptive field which invoke a 'response' from all the neurons
- **Regions in the receptive field which invoke a 'response' from the neurons trained to focus on that receptive field** ✓

  Feedback :
  All neurons do not respond to light falling on a receptive field, only the ones trained to focus on that receptive field do.
- Regions in the input image which invoke a 'response' from all the neurons

Understanding the Visual System of Mammals

The following figure shows the excitatory and the inhibitory regions (crosses and triangles respectively) of a certain receptive field of a cat:



Which of the following shapes of light will invoke the strongest response by the neurons?

- A circular-shaped light spreading uniformly across the field (covering both the regions)
- A horizontal slit shaped light spreading across both the regions
- **A vertical slit shaped light falling only on the excitatory region** ✓

  Feedback :
  Neuron only 'fires' when there is a contrast between the excitatory region and the inhibiting region.
- An O-shaped light falling only on the inhibitory region

Understanding the Visual System of Mammals -II

Every neuron is trained to look at a particular patch in the retina, called the **receptive field** of that neuron.

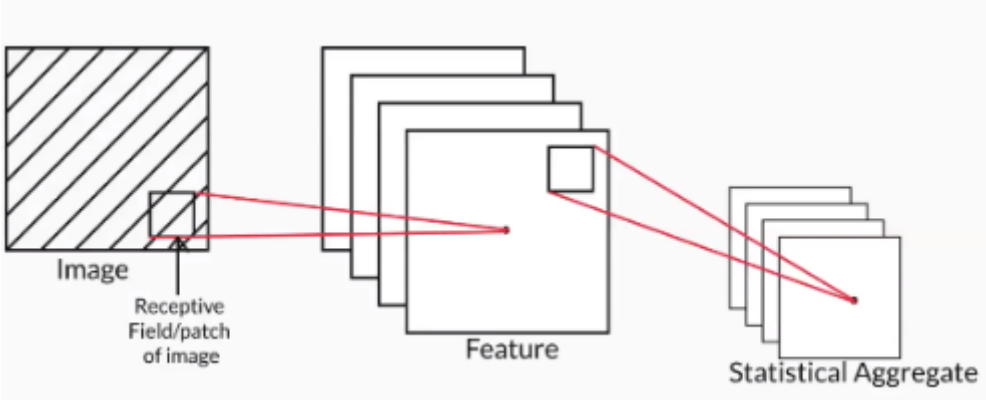
This raises some questions such as: Are the shapes and sizes of these receptive fields identical across neurons or do they vary? Do all the neurons 'see' the same 'features', or are some neurons specialised to 'see' certain features?

Let's seek answers to some of these questions, at a high level, how higher-level abstract 'features' such as 'movement' are detected by the visual system.

In this lecture, we studied two main observations from the paper:

- The **receptive fields of all neurons are almost identical** in shape and size
- There is a **hierarchy in the units:** Units at the initial level do very basic tasks such as picking raw features (such as horizontal edges) in the image. The subsequent units extract more abstract features, such as identifying textures, detecting movement, etc. The layers 'higher' in the hierarchy typically **aggregate the features** in the lower ones.

The image below illustrates the hierarchy in units - the first level extracts low-level features (such as vertical edges) from the image, while the second level calculates the statistical aggregate of the first layer to extract higher-level features (such as texture, colour schemes etc.).



Hierarchy in neurons

Using this idea, if we design a complex network with multiple layers to do **image classification** (for example), the layers in the network should do something like this:

- The first layer extracts raw features, like vertical and horizontal edges
- The second layer extracts more abstract features such as textures (using the features extracted by the first layer)
- The subsequent layers may identify certain parts of the image such as skin, hair, nose, mouth etc. based on the textures.
- Layers further up may identify faces, limbs etc.
- Finally, the last layer may classify the image as 'human', 'cat' etc.

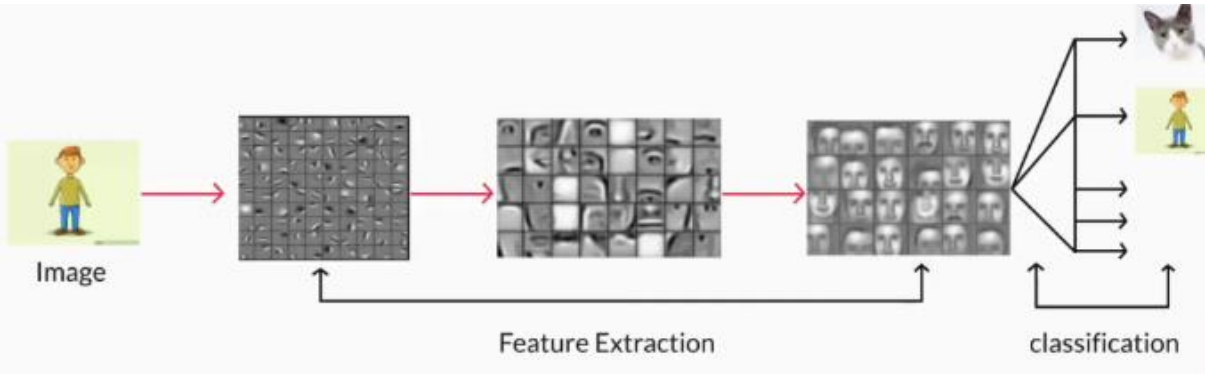


Image Classification

Apart from explaining the visual system, the paper also suggested that similar phenomena have been observed in the **auditory system** and touch and pressure in the **somatosensory system**. This suggests that CNN-like architectures can be used for **speech processing** and analysing signals coming from **touch sensors** or **pressure sensors** as well.

Let's have a look at some of the conclusions.

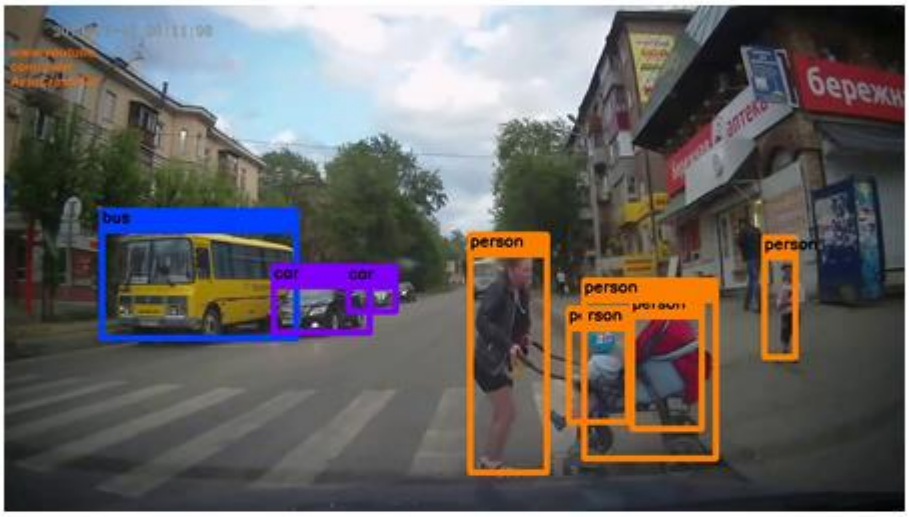
We have already discussed most of the key ideas of the CNN architecture through this paper. Summarising the main points below:

- Each unit, or neuron, is dedicated to its own **receptive field.** Thus, every unit is meant to ignore everything other than what is found in its own receptive field.
- The **receptive field** of each neuron is **almost identical** in shape and size.
- The subsequent layers compute the **statistical aggregate** of the previous layers of units. This is analogous to the 'pooling layer' in a typical CNN.
- Inference or the perception of the image happens at various **levels of abstraction**. The first layer pulls out raw features, subsequent layers pull out higher-level features based on the previous features and so on. Finally, the network gets an overall perception of an image in the last layer.

Layers in Your Visual Cortex While Driving

CNNs are similar to normal neural networks (MLPs) in that they have **layers of neurons arranged sequentially**. The paper suggested that the layers are arranged in a **hierarchical manner**, i.e. the layers further up (towards the right in usual notation) extract more 'abstract' features than the previous layers.

Let's assume that your visual cortex (the region of the brain that receives and processes visual information) has **five layers** (apart from the input) of neurons. Also, say you are driving, and in the process, you are doing **object detection** - i.e. **detecting** the area where an object is on the road or footpath (a pedestrian, a car, a traffic signal, a tree etc.) and **recognising** it:



Which of the following statements can be true? Mark all correct possibilities:

- [x] The first layer extracts basic features from the image, such as edges, while the second extracts textures, colour patterns, etc.

  Feedback:
  The initial layers extract basic patterns, while the latter ones extract higher-level patterns.

  ✓ Correct

- [x] The fourth layer may be detecting the area (the rectangular box) where an object is located, while the fifth (last) layer classifies it into one of the categories (person, car etc.)

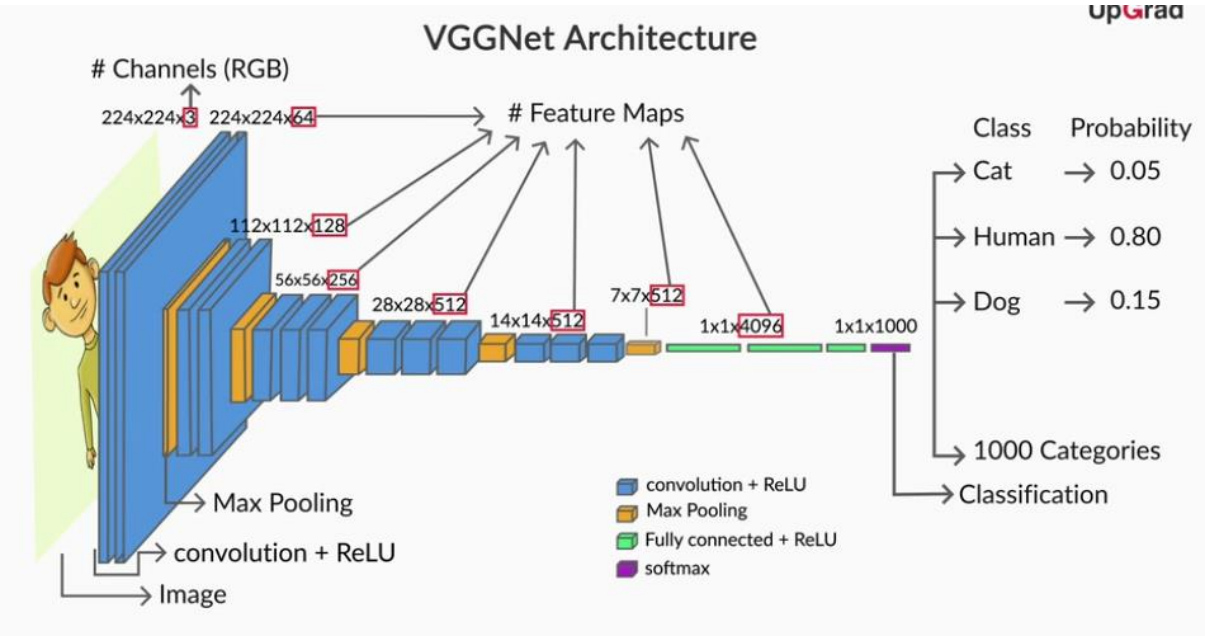
  Feedback:
  Classification of the object can only be done after finding where it is located.

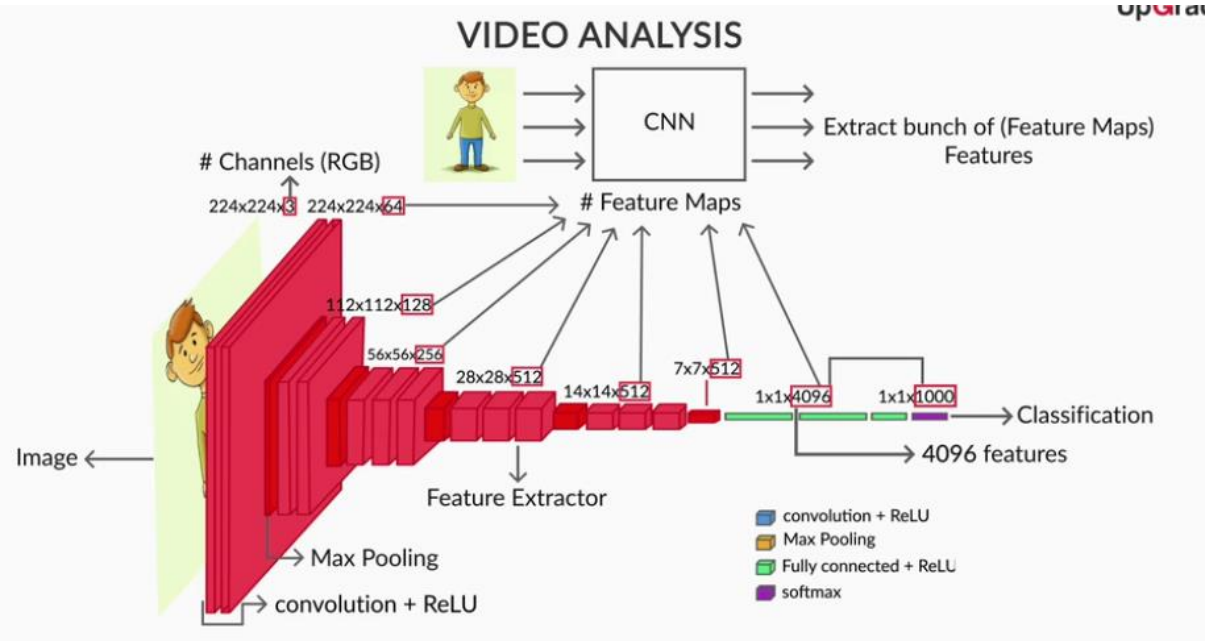
  ✓ Correct

- [ ] The second layer may be classifying the object into one of the categories (person, car etc.)

Introduction to CNNs

Let's dig a little deeper into CNN architectures now. In this segment, we will analyse the architecture of a popular CNN called **VGGNet.** Observing the VGGNet architecture will give you a high-level overview of the common types of CNN layers before you study each one of them in detail.

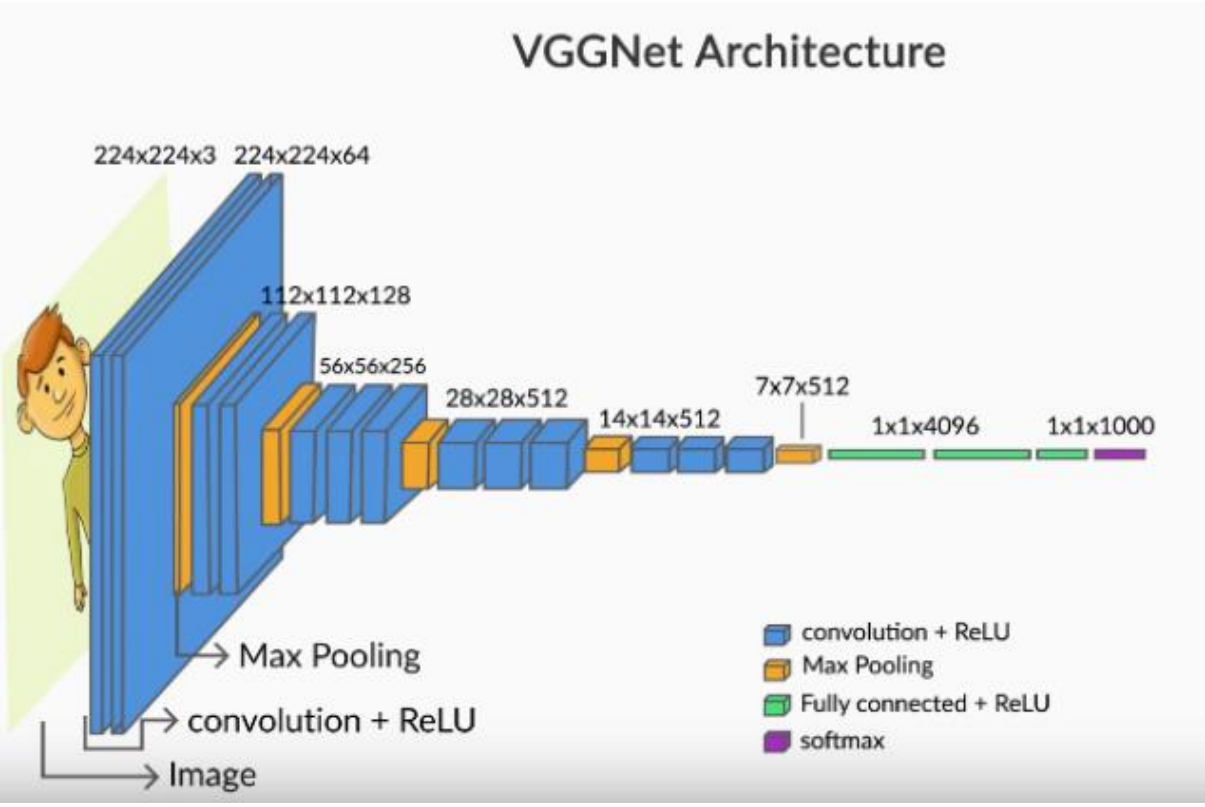




There are three main concepts in CNNs:

- Convolution, and why it 'shrinks' the size of the input image
- Pooling layers
- Feature maps

The VGGNet architecture is shown below.



VGGNet

The VGGNet was specially designed for the ImageNet challenge which is a classification task with 1000 categories. Thus, the softmax layer at the end has 1000 categories. The blue layers are the **convolutional layers** while the yellow ones are **pooling layers.**

Finally, the green layer is a **fully connected layer** with 4096 neurons, the output from which is a vector of size 4096.

The most important point to notice is that the **network acts as a feature extractor** for images. For example, the CNN above extracts a **4096-dimensional feature vector** representing each input image. In this case, the feature vector is fed to a softmax layer for classification, but you can use the feature vector to do other tasks as well (such as video analysis, object detection, image segmentation etc.).

Next is how to do **video analysis** using the feature vector extracted by the network.

Feature Extractor

Which of the following operation acts as a feature extractor?

- (●) **Convolution**

  Feedback :
  Convolution operation acts as a feature extractor.

- ( ) Softmax

Softmax Layer

What is the following is correct for last softmax layer in the CNN?

- [x] **Each class probability lies in the range 0-1**

  Feedback :
  Probability always lies between 0 and 1

- [x] **Sum of class probability is 1**

  Feedback :
  Sum of probability is 1

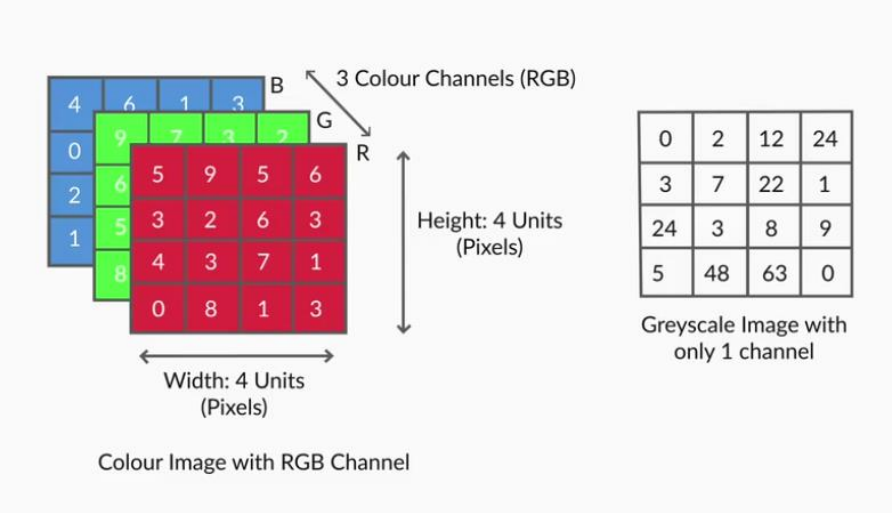
- [ ] Sum of class probability is 100

- [ ] Each class probability lies in the range 0-100

Reading Digital Images

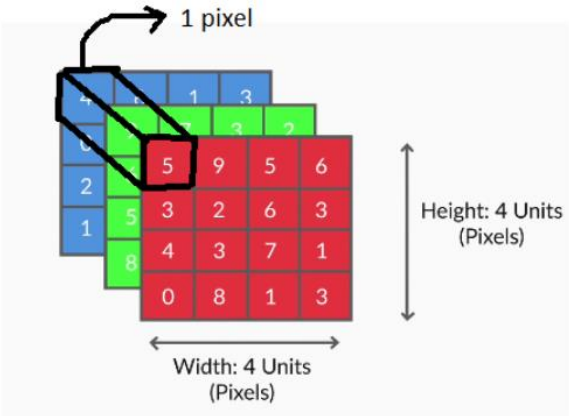
Before we dig deeper into the architecture of CNNs, let's understand what images are and how they are fed into CNNs.

Input to any neural network should be numeric. Fortunately, images are naturally represented as arrays (or matrices) of numbers. Let's study the typical structure of images.



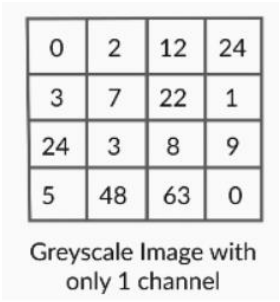
To summarize:

- Images are made up of **pixels**.
- A number between 0-255 represents the **colour intensity** of each pixel.
- Each pixel in a **colour image** is an array representing the intensities of red, blue and green. The red, blue and green layers are called **channels**.



Colour Image

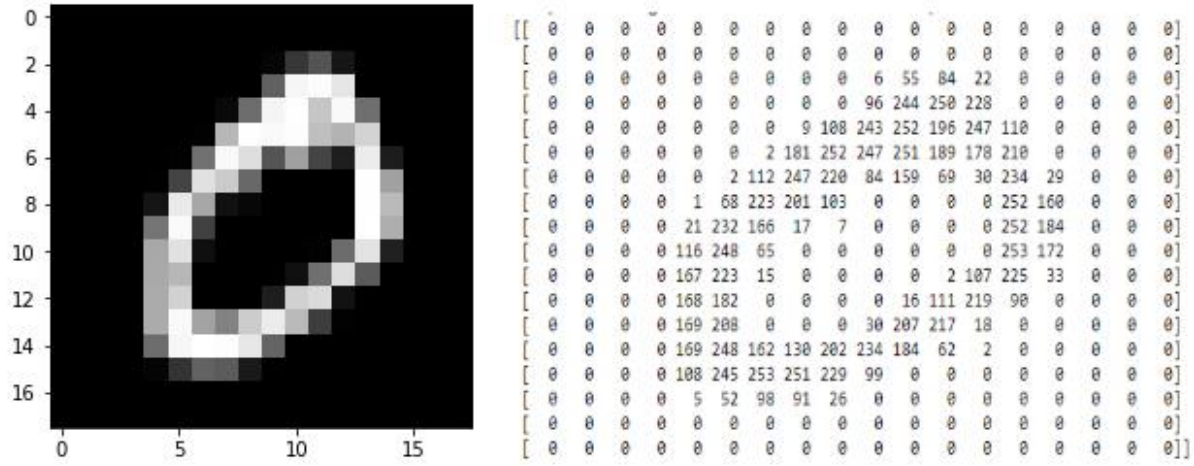
- In a **grayscale image** (a 'black and white' image), only one number is required to represent the **intensity of white**. Thus, grayscale images have **only one channel.**



Greyscale Image

Let’s see an example of how one would read images into Python.

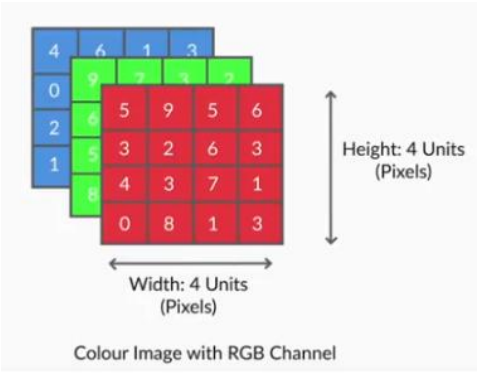
Consider this sample image of a 'zero' from the MNIST dataset.



18x18 greyscale image

- The height and width of this image are 18 pixels, so it is stored as an 18 x 18 array
- Each pixel's value lies between 0-255
- The pixels having a value close to 255 appear white (since the pixels represent the intensity of white), and those close to 0 appear black

Let’s see this for a colour image.



Colour image

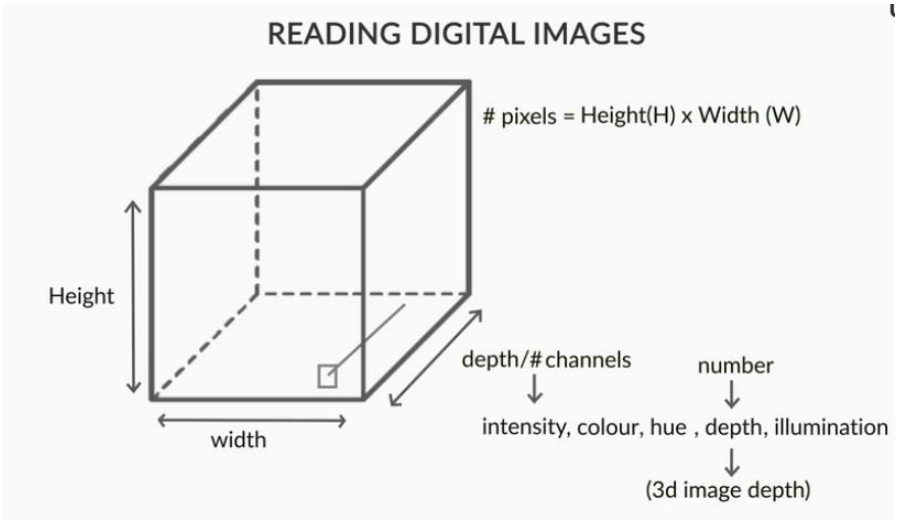
- The height and width of the image are 4 pixels.
- Here, three numbers make each pixel (representing RGB). So, there are **3 channels** here.
- The size of the matrix is thus 4 x 4 x 3

All colours can be made by mixing red, blue and green at different degrees of "saturation" (0-100% intensity). For example, a pure red pixel has 100% intensity of red, and 0% intensity of blue and green. So, it is represented as (255,0,0). White is the combination of 100% intensity of red, green and blue. So, it is represented as (255,255,255).

Why is the Range of Pixel Values 0-255?

Usually, 8-bits (1 byte) are used to represent each pixel value. Since each bit can be either 0 or 1, 8-bits of information allows for $2^8=256$ possible values. Therefore, the range of each pixel is 0-255.

Let's now quickly summarise what you have learnt about images.



Next few segments cover the architecture of CNNs in detail.

Pixels

How many total number of pixels are there in a grayscale image whose dimension is (250,150)?

- 250
- 150
- **37500** ✓

  Feedback :
  250x150 = 37500

  The number of pixels is height x width. Here, height is 250, and the width is 150. So, number of pixels is 250x150 = 37,500

Channels

How many channels are there in an image? Mark all correct statements:

- **If it is a grayscale image, number of channels is 1** ✓

  Feedback :
  A greyscale image has single channel.

- **If it is a colour image, and if we represent by RGB (Red, Green, Blue), the number of channels is 3** ✓

  Feedback :
  If it is a colour image, and if we represent by RGB (Red, Green, Blue), the number of channels is 3 for Red, Green and Blue.

- **If we represent an image in HSV (Hue, Saturation, Value) format, the number of channels is 3.** ✓

  Feedback :
  If we represent an image in HSV (Hue, Saturation, Value) format, the number of channels is 3 for Hue, Saturation and Value

- The number of channels is always 3.

Pixels

What is the total number of pixels in an image whose dimension is (250,150,3), where '3' represents the RGB channels?

- 250
- 150
- 1,12,500
- **37,500** ✓

  Feedback :
  Numbers of pixels is 'width x height', which is 250 x 150, independent of depth.

Values in channel

What is the range of possible values of each channel of a pixel if we represent each pixel by 8 bits?

- -254 to 255
- **0 to 255**

  Feedback :
  2^8(bits) = 256. So, the range is 0 to 255.

- 0 to 128

Reading Digital Images

In a grayscale image, which colours represent 0 and 255?

- **White-255, Black-0**

  Feedback :
  Lowest intensity - '0' is black and the highest intensity-'255' is white.

- Black-255, White-0

Pixels

What do the numbers signify in the pixel?

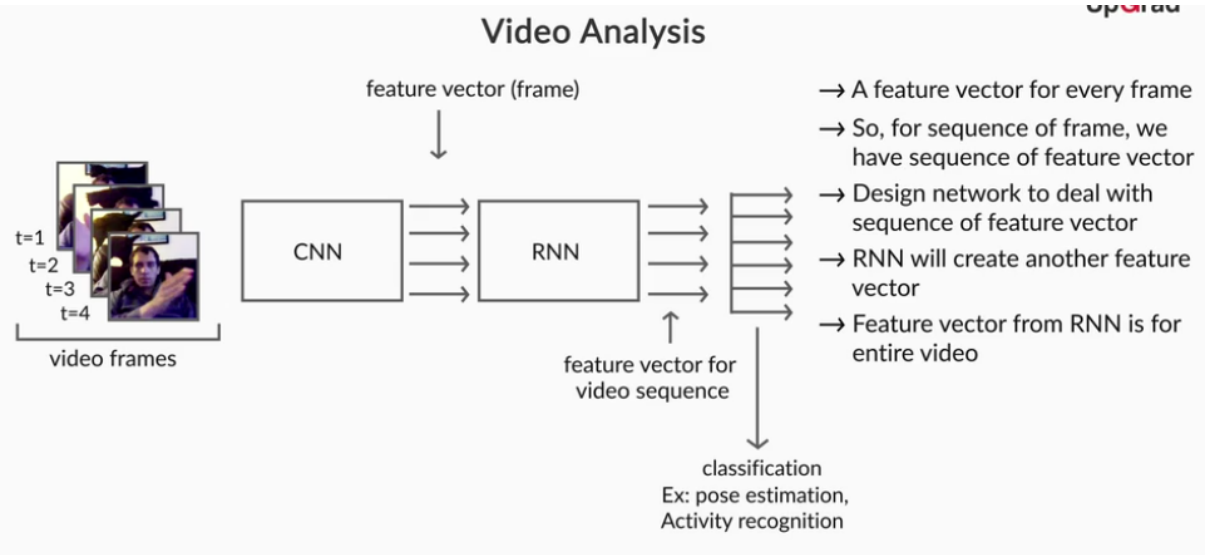
- **Intensity of a colour**

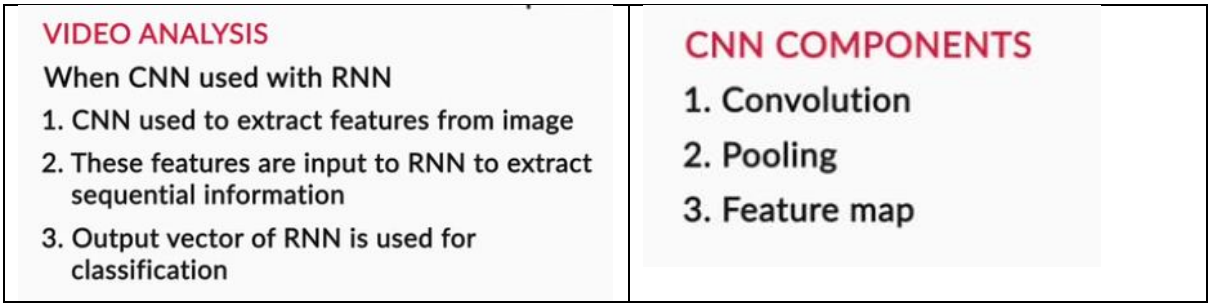
  Feedback :
  Each number in pixel represents intensity. If it's '0', it means black with no intensity and '255' means white with the highest intensity.

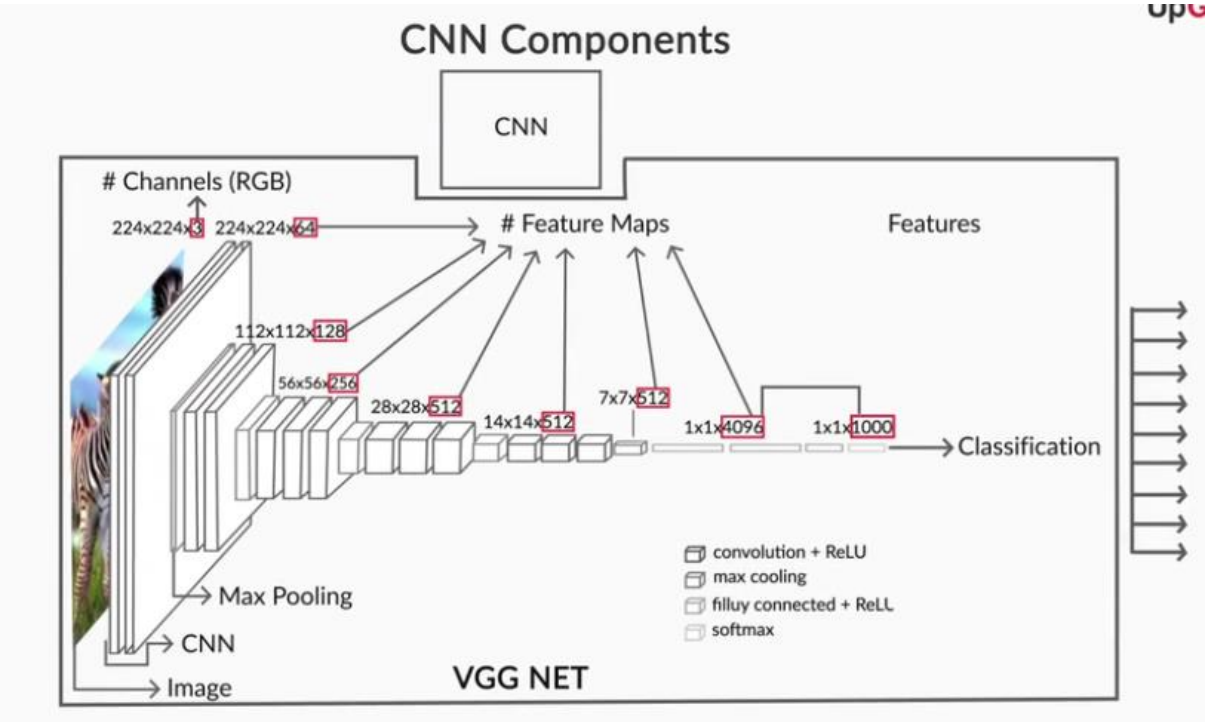
- Density of a colour

Video Analysis

This segment covers the process of **video analysis using CNNs**. A video is basically a sequence of frames where each frame is an image. CNNs can be used to extract features from an image. Let's now see how CNNs can be used to process a series of images (i.e. videos).







Let's summarise the process of video analysis using a CNN + RNN (Recurrent Neural Network) stack. RNNs are good at processing sequential information such as videos (a sequence of images), text (a sequence of words or sentences), etc.

For a **video classification** task, For a video of length 1 minute each, extract frames from each video at the rate of **2 frames per second** (FPS), to have 120 frames (or images) per video. Push each of these images into a convolutional net (such as VGGNet) and **extract a feature vector** (of size 4096, say) for each image. Thus, we have 120 feature vectors representing each video.

These 120 feature vectors, representing a video as a sequence of images, can now be fed sequentially into an RNN which classifies the videos into one of the categories.

A **CNN acts as a feature extractor** for images, and thus, can be used in a variety of ways to process images**.**

Next few segments will cover the main elements of CNNs in detail - convolutions, pooling, feature maps etc.

Video Analysis

Which of the following network is primarily used for processing sequential information?

- CNN
- RNN ✓

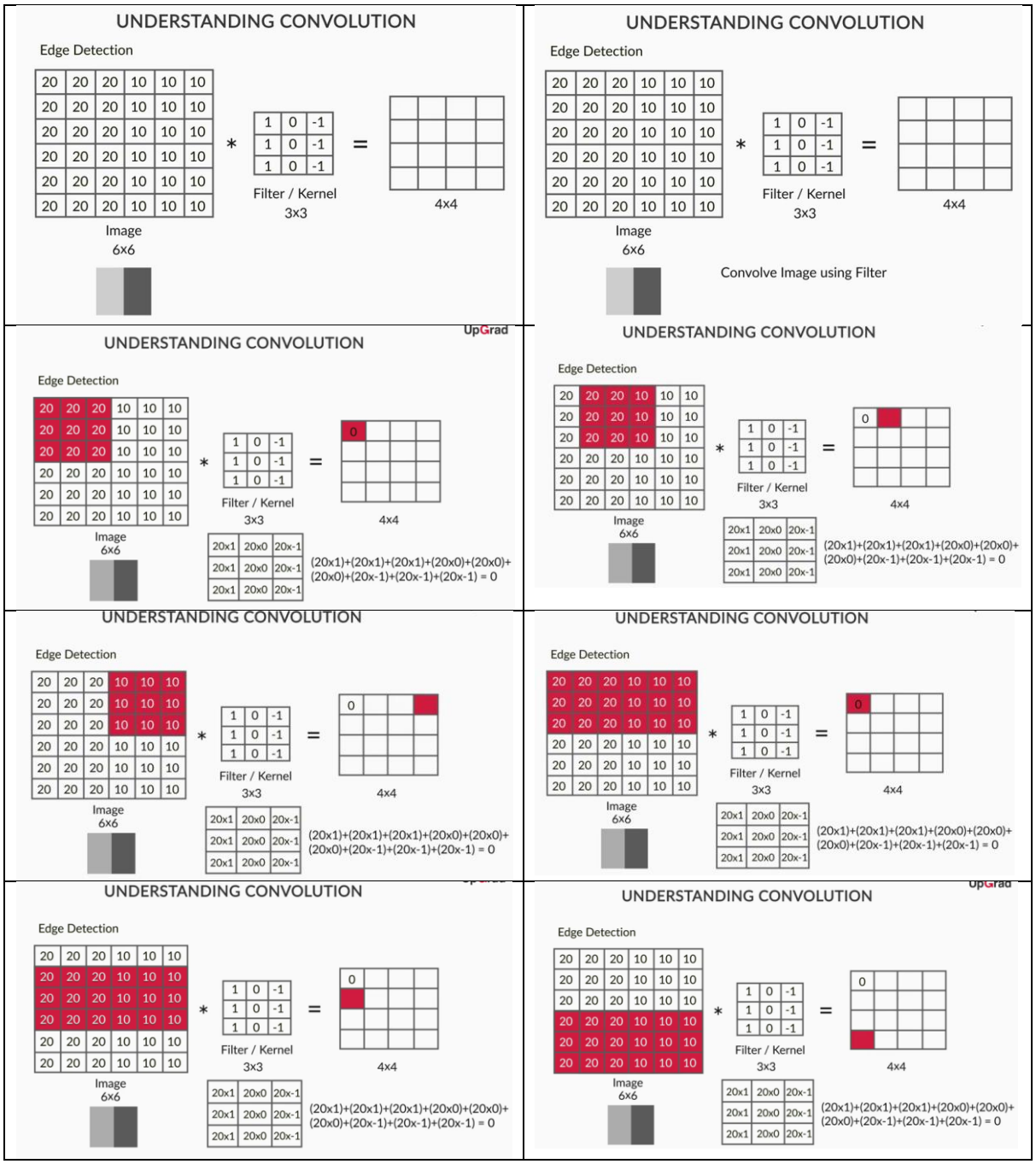
Feedback :
Recurrent Neural Networks are designed to process sequential information, such as videos (sequence of images), text (sequence of words or sentences), etc.

Understanding Convolutions - I

Three main terminologies related to the CNN architecture:

- Convolutions
- Pooling
- Feature Maps

Next few segments will go through each one of them in detail. Let's start by understanding how **convolutions** work.



Convolution

Mathematically, the convolution operation is the **summation of the element-wise product** of two matrices. Let's take two matrices, X and Y. If you 'convolve the image X using the filter Y', this operation will produce the matrix Z.

X

1	2	3
2	0	0
7	9	1

Y

3	2	0
3	0	1
0	5	2

Z

1x3=3	2x2=4	3x0=0
2x3=6	0x0=0	0x1=0
7x0=0	9x5=45	1x2=2

Finally, you compute the sum of all the elements in Z to get **a scalar number,** i.e. 3+4+0+6+0+0+0+45+2 = 60.

Understanding Convolution

Given an input matrix X of size (2,2) and filter matrix Y of size (2,2), find the output value after we perform convolution of X and Y.

X

1	4
0	9

Y

4	0
2	1

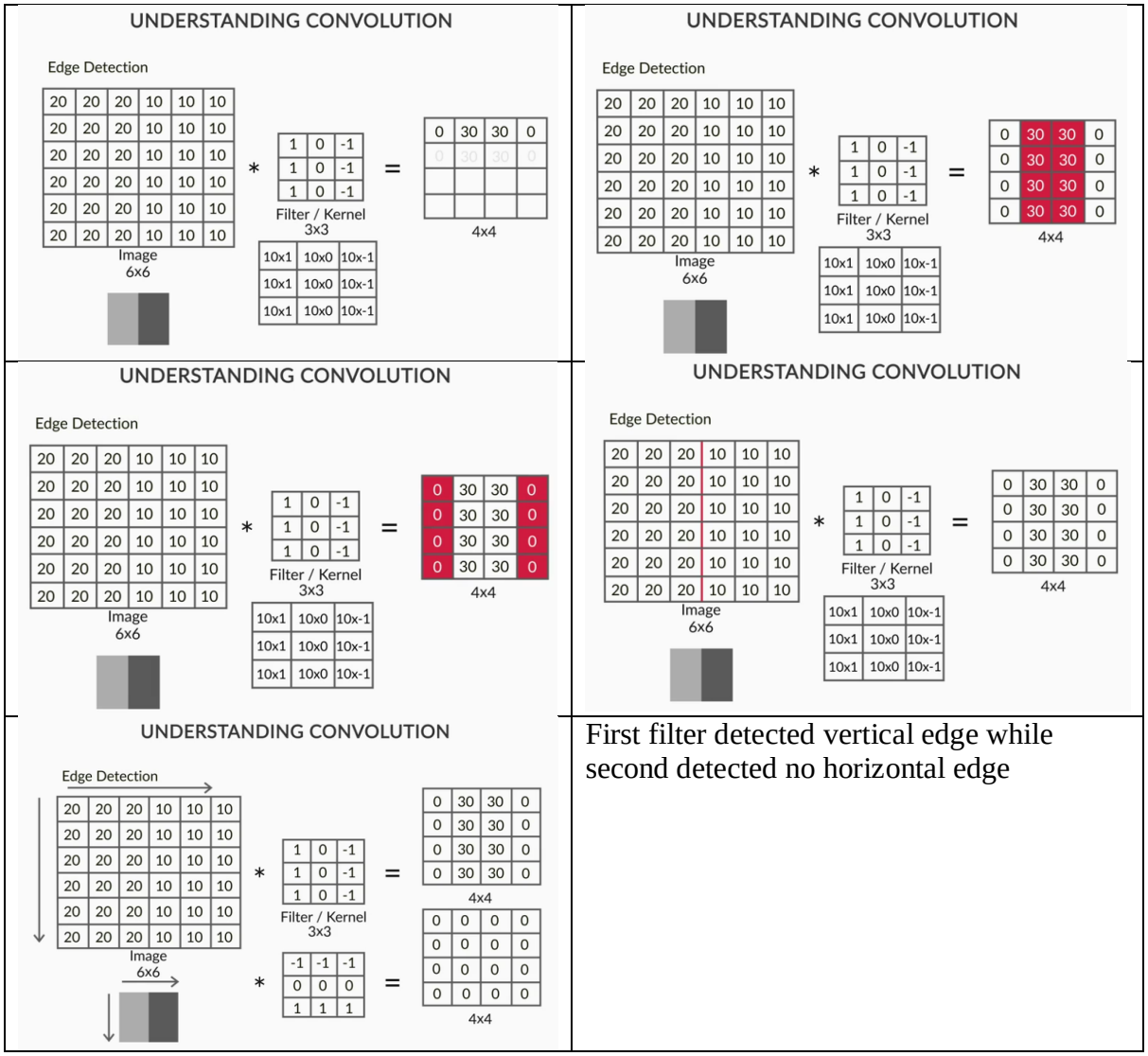
○ 10

◉ 13 ✓

Feedback :
Convolution of 2 matrices X and Y is : 1x4 + 4x0 + 0x2 + 9x1 = 13

Understanding Convolutions - II

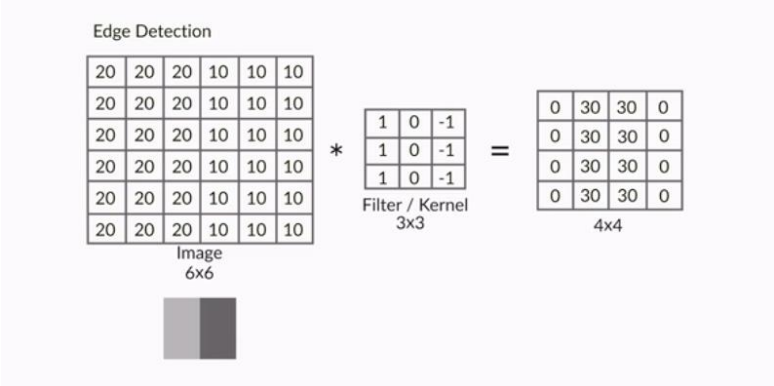
After the basic idea of filters and convolutions, let's continue our example from the previous page to understand how convolutions are used to **detect features** (such as vertical or horizontal edges) in an image.



First filter detected vertical edge while second detected no horizontal edge

This was an example of how the convolution operation (using an appropriate filter) detects certain features in images, such as horizontal or vertical edges.

In the convolution output using the first filter, only the middle two columns are nonzero while the two extreme columns (1 and 4) are zero. This is an example of **vertical edge detection**.



Vertical Edge Detection

Each column of the 4 x 4 output matrix **looks at exactly three columns** of the input image. The values in the four columns represent the amount of change (or gradient) in the intensity of the corresponding columns in the input image along the horizontal direction.

For e.g. the output is 0 (20 - 20 or 10 - 10) in the columns 1 and 4, denoting that there is no change in intensity in the first three and the last three columns of the input image respectively.

On the other hand, the output is 30 (20 - (-10)) in the columns 2 and 3, indicating that there is a gradient in the intensity of the corresponding columns of the input image.

Other Filters

The filter below is used for **horizontal edge detection**. This filter will be able to detect horizontal edges in an image.

Filter - Horizontal Edge Detection		
-1	-1	-1
0	0	0
1	1	1

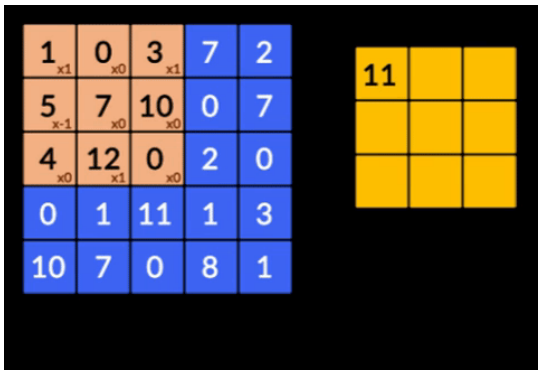
Convolution Example

Let's see one more example of a convolution operation. Consider the image 'I' shown below and convolve it with the 3 x 3 filter to produce a 3 x 3 array. The GIF below demonstrates the convolution operation.

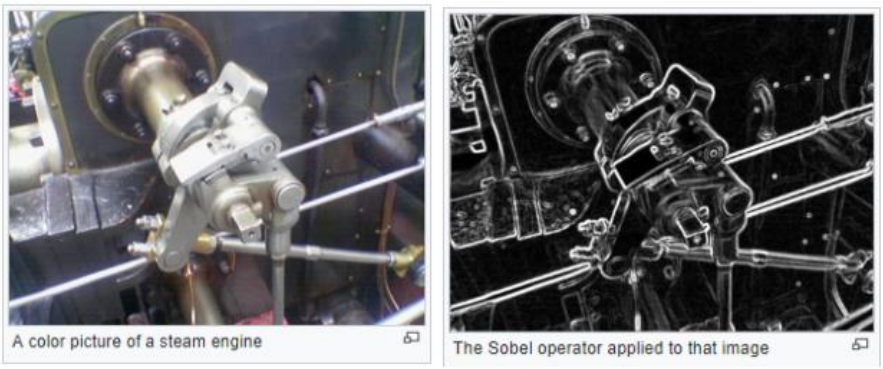
Image(I)				
1	**0**	**3**	**7**	**2**
5	**7**	**10**	**0**	**7**

Filter		
1	**0**	**1**
-1	**0**	**0**
0	**1**	**0**

4	12	0	2	0
0	1	11	1	3
10	7	0	8	1



Although we have only seen very simple filters, one can design arbitrarily complex filters for detecting edges and other patterns. For example, the image below shows **the Sobel filter** which can detect both horizontal and vertical edges in complex images.


A color picture of a steam engine

The Sobel operator applied to that image

Sobel Operator

We have discussed some simple examples of filters and convolutions, and you may have some questions such as 'can filters have arbitrary sizes', 'can any filter convolve any image', etc. In the next segment, we will be able to answer these questions using the concepts of **stride** and **padding.**

Convolution of X with Y

Given an input image X (3,3) and a filter Y of size (2,2), as shown below, what is the output matrix after convolution?

X

1	4	0
0	9	1
5	0	7

Y

1	-3
0	4

- ✓ | 25 | 8 |
  | -27 | 34 |
- | -2 | 8 |
  | -7 | 4 |

Output size

Given an input image of size (10,10) and a filter of size (4,4), what will be the size of the output size on convolving the image with the filter?

- (8,8)
- ✓ (7,7)

  Feedback :
  If we move (4,4) matrix on a matrix of size (10,10), there will be 7 vertical and 7 horizontal positions.
- (6,6)
- (5,5)

Diagonal Edge Detection

Which of the following filters can be used to detect a diagonal edge (an edge at an angle of 45 degrees from the x-axis) in an image? Choose all the correct options.

- [x] $\begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & -1 \end{bmatrix}$ ✓

  Feedback :
  A diagonal edge will have pixel values such that there is a gradient in the direction perpendicular to the 45-degree line, i.e. a gradient in pixel values from top-left to bottom-right. The filter should also have a gradient in this direction.

- [x] $\begin{bmatrix} 2 & 2 & 0 \\ 2 & 0 & -2 \\ 0 & -2 & -2 \end{bmatrix}$ ✓

  Feedback :
  A diagonal edge will have pixel values such that there is a gradient in the direction perpendicular to the 45-degree line, i.e. a gradient in pixel values from top-left to bottom-right. The filter should also have a gradient in this direction.

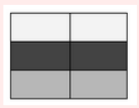
- [ ] $\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$

- [ ] $\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$

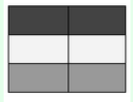
Matrix in form of Image

Which of the following image approximately represents the given matrix?

0	0	0	0	0	0
0	0	0	0	0	0
100	100	100	100	100	100
100	100	100	100	100	100
50	50	50	50	50	50
50	50	50	50	50	50

- ✗ 

  Feedback :
  Lower value of pixel means lower intensity and higher value for high intensity. Like '0' for black and '255' for white.

- ✓ 

  Feedback :
  Lower value of pixel means lower intensity and higher value for high intensity. Like '0' for black and '255' for white.

Vertical Edge Detection

Given an input image X, which of the following filters will detect an edge in the vertical direction? More than one options may be correct:

30	30	30	100	100	100
30	30	30	100	100	100
30	30	30	100	100	100
30	30	30	100	100	100
30	30	30	100	100	100
30	30	30	100	100	100

- [x]

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

  Feedback :
  Any matrix that takes the difference between the left and the right pixel can find the edge.

  ✓ Correct

- [ ]

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

  Feedback :
  Any matrix that takes the difference between the left and the right pixel can find the edge.

  ✓ Correct
  You missed this!

- [ ]

1	0	1
1	0	1
1	0	1

- [x]

1	0	-1
1	0	-3
1	0	-1

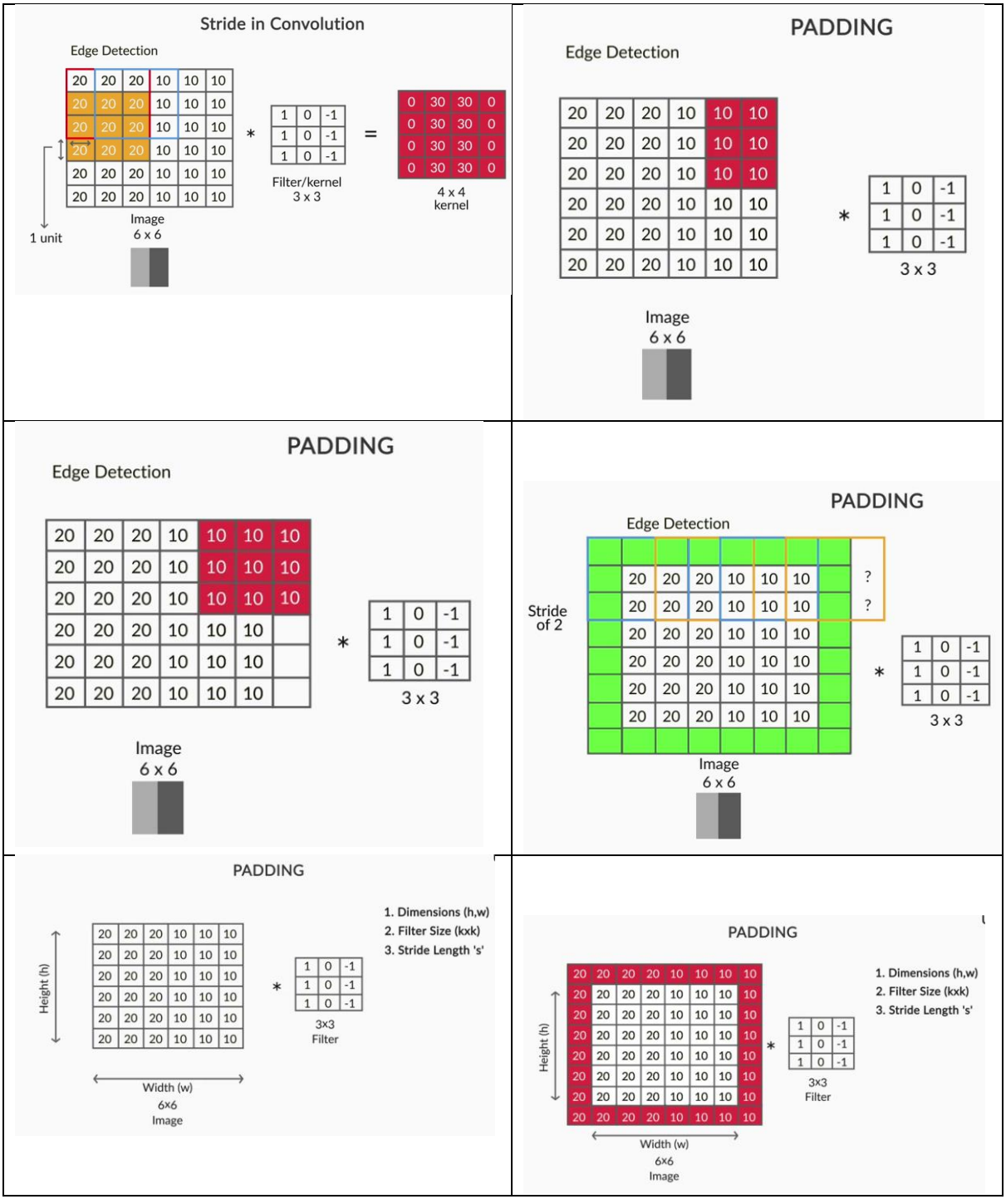
  Feedback :
  Any matrix that takes the difference between the left and the right pixel can find the edge.

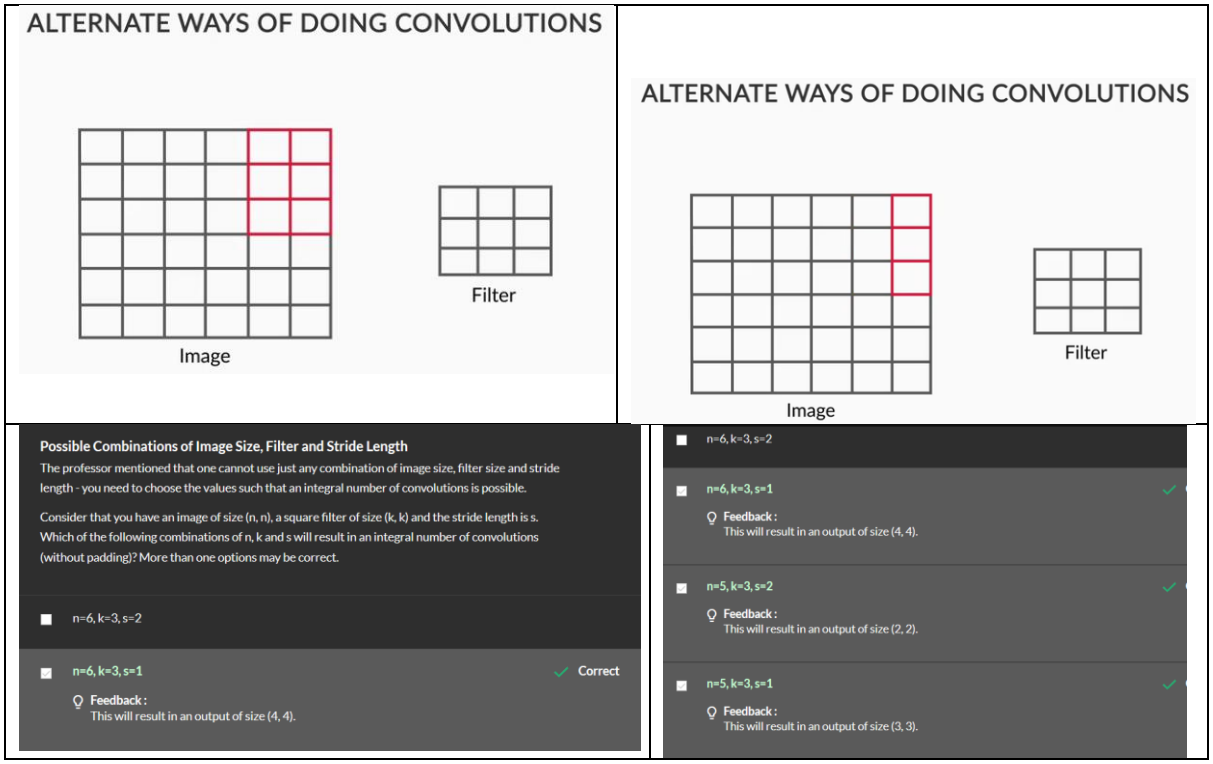
  ✓ Correct

Stride and Padding

While doing convolutions, each time we computed the element-wise product of the filter with the image, we had moved the filter by exactly one pixel (both horizontally and vertically). But filters can also be moved by an arbitrary number of pixels. This is the concept of **stride**.

The notion of strides will lead to - **padding.**





There is nothing sacrosanct about the stride length 1. If we do not need many fine-grained features for a task, we can use a higher stride length (2 or more).

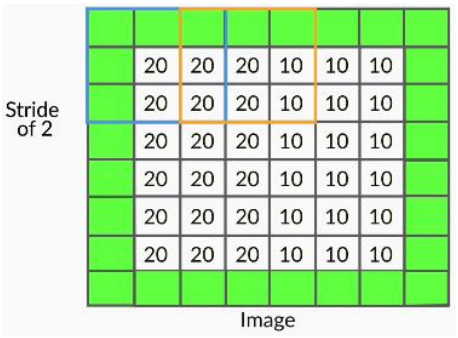
We cannot convolve all images with just any combination of filter and stride length. We cannot convolve a (4, 4) image with a (3, 3) filter using a stride of 2. We cannot convolve a (5, 5) image with a (2, 2) filter and a stride of 2.

To solve this problem, we use **padding.**

Padding

The following are the two most common ways to do padding:

- Populating the dummy row/columns with the pixel values at the edges
- Populating the dummy row/columns with zeros (zero-padding)



Stride and Padding

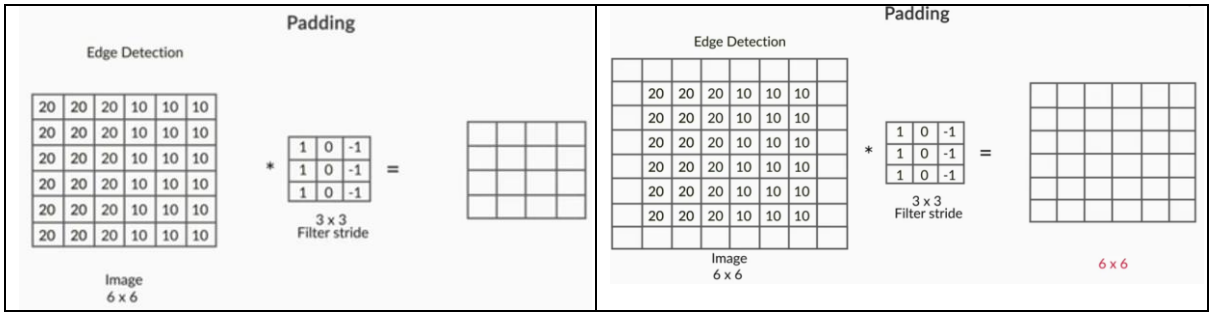
Notation:

Padding of 'x' means that 'x units' of rows/columns are added all around the image.

An alternate (less commonly used) way to do convolution is to shrink the filter size as you hit the edges.

When we convolve an image **without padding** (using any filter size), the **output size is smaller** than the image (i.e. the output '**shrinks**'). When we convolve a (6, 6) image with a (3, 3) filter and stride of 1, we get an output of (4, 4).

To **maintain the same size**, we can use padding. Let's see how padding maintains the image size.

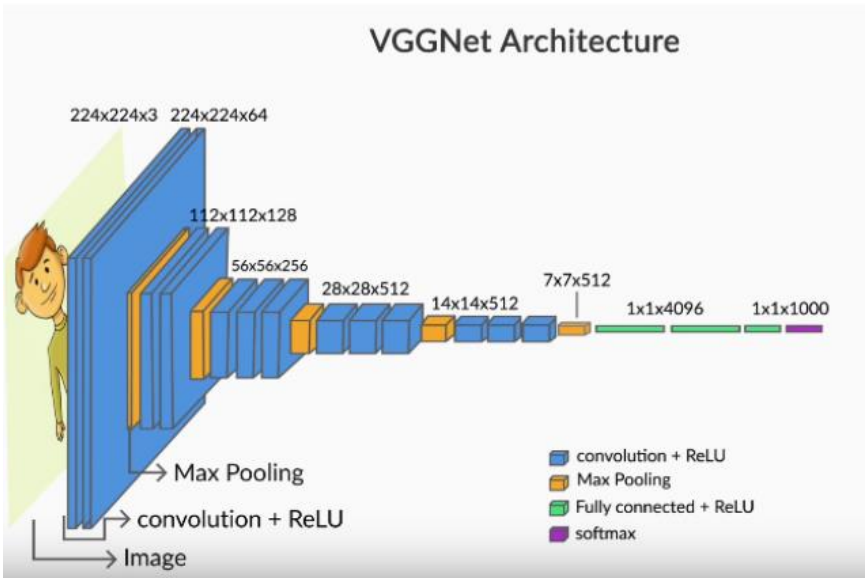


Doing convolutions without padding **reduces the output size**. **Only the width and height decrease (not the depth)** when we convolve without padding. The depth of the output depends on the number of filters used.

Why Padding is Necessary

Doing convolutions without padding will 'shrink' the output. For e.g. convolving a (6, 6) image with a (3, 3) filter and stride of 1 gives a (4, 4) output. Further, convolving the (4, 4) output with a (3, 3) filter will give a (2, 2) output. The size has reduced from (6, 6) to (2, 2) in just two convolutions. Large CNNs have tens (or even hundreds) of such convolutional layers (recall VGGNet), so we will be incurring massive 'information loss' as we build deeper networks!

This is one of the main reasons padding is important - it helps maintain the size of the output arrays and avoid information loss. Of course, in many layers, you actually want to shrink the output (as shown below), but in many others, you maintain the size of the output.



VGG Net

Input and Output of Convolutions

Given an image of size (n, n), a kernel of size (3, 3), no padding, and a stride length of 1, the output size will be:

- (n, n)
- (n-1, n-1)
- (n-2, n-2) ✓
  Feedback :
  Try convolving (3, 3), (4, 4), (5, 5) etc. images - you will note a pattern. In general, an (n, n) image will produce an (n-2, n-2) output on convolving with a (3, 3) filter.
- (n+1, n+1)

Stride and Padding

Given an image of size 5x5, filter size 3x3, stride 2, what is the padding required (on either side) to make the output size same as input size?

- 0 pixels on either side
- 1 pixels on either side
- 2 pixels on either side
- 3 pixels on either side ✓
  Feedback :
  3 pixels of padding is required around each edge of the image to make the output size the same as the image size.

The Need for Padding

You saw that doing convolutions without padding reduces the output size (relative to the input which is being convolved). The main reason this is not always beneficial is that:

- Maintaining the output size results in stable models
- There will be heavy loss of information (especially in deep networks) if all the convolutional layers reduce the output size ✓
  Feedback :
  If all the layers keep shrinking the output size, by the time the information flows towards the end of the network, the output would have reduced to a very small size (say 2 x 2) - which will be insufficient to store information for complex tasks.
- It does not matter much - one can build CNNs without any padding at all

A General Strategy for Maintaining the Output Size

Very often, you want to maintain the output size after convolution, i.e. if you have an (n, n) input, you want the output to be of size (n, n) as well. A general strategy to achieve this is:

- Use padding of 1 pixel (on both sides), a (3, 3) filter, and stride length 1 ✓
  Feedback :
  The output size is $\frac{n+2p-k}{s}+1$. With p=1, k=3, s=1, the output will always be (n, n). Going forward, you may find remembering this useful.
- Use padding of 2 pixels (on both sides) and a (3, 3) filter, and stride length 2
- Use zero padding and a (3, 3) filter, and stride length 1

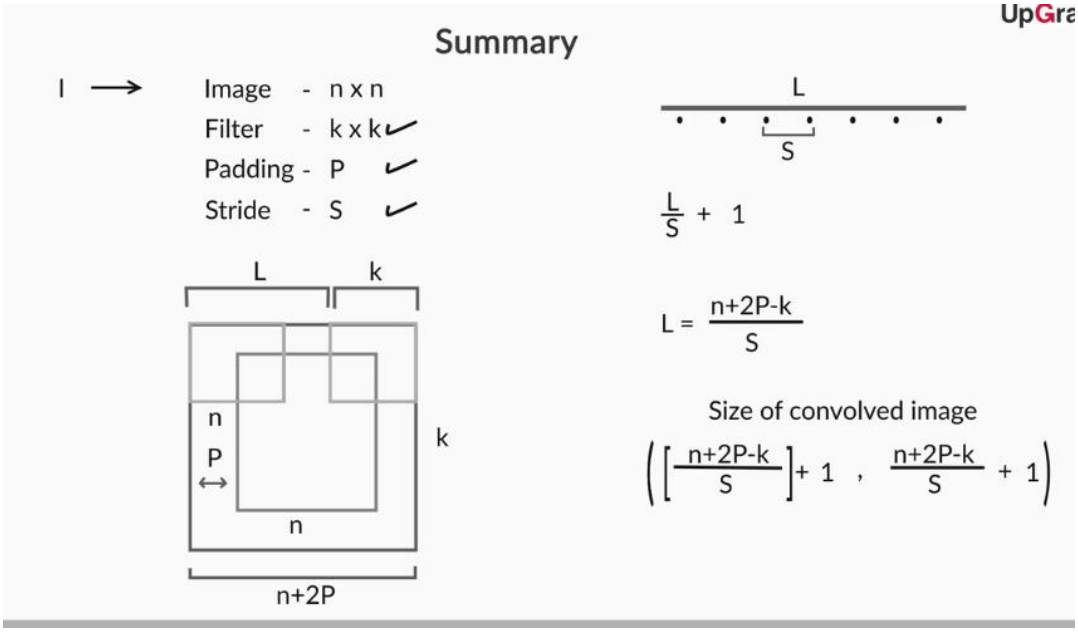
Until now, we have been computing the output size (using the input image size, padding and stride length) manually. Next segment covers generic formulas which will help reduce some of the manual work.

Important Formulas

This segment will go through useful formulas for calculating the output size using the input size, filter size, padding and stride length.

Fun Challenge

Before moving on to the lecture, you may want to try solving this problem yourself first: Given an image of size (n, n), a (k, k) filter, padding of p pixels on either side of the image, and stride length of S, derive a general formula for computing the output size after convolution.



Given the following size:

- Image - n x n
- Filter - k x k
- Padding - P
- Stride - S

After padding, we get an image of size (n + 2P) x (n+2P). After we convolve this padded image with the filter, we get:

$$\text{Size of convolved image} = \left(\frac{n+2P-k}{S}+1\right), \left(\frac{n+2P-k}{S}+1\right)$$

Until now, we have applied convolution only on 2D arrays, but most images are coloured and thus have multiple channels (e.g. RGB). Next segment will teach to convolve images with multiple channels.

Stride

Given an input image of size 224x224, a filter of size 5x5 and padding of 3, what are the possible values of stride S?

- [x] 1 ✓

  Feedback :
  (n+2P-k) should be divisible by stride 's'. So, (224+ 2x3 - 5) = 225. should be divisible by any possible values.

- [ ] 2

- [x] 3 ✓

  Feedback :
  (n+2P-k) should be divisible by stride 's'. So, (224+ 2x3 - 5) = 225 should be divisible by any possible . 225 is divisible by 3.

- [ ] 4

Padding

Given an input image of size 224x224, a filter of size 5x5 and stride of 2, what are the possible values of padding?

- [ ] 1
- [x] 2 ✗

  Feedback :
  (n+2P-k) should be divisible by stride 's'. So, (224 + 2xPadding -5) should be divisible by 2.

- [x] 3 ✗

  Feedback :
  (n+2P-k) should be divisible by stride 's'. So, (224 + 2xPadding -5) should be divisible by 2.

- [ ] Not possible ✓

  Feedback :
  (n+2P-k) should be divisible by stride 's'. So, (224 + 2xPadding -5) should be divisible by 2. This is not possible for any value of padding.

Output Size

The tables below enlist some combinations of the input size, filter size, stride length, padding and the output size. Match the rows in the left table corresponding to the outputs in the right table: (One or more options may be correct)

	Input	Filter	Stride	Padding
a	8x8	3x3	1	0
b	7x7	3x3	2	0
c	8x8	3x3	1	1

	Output?
d	8x8
e	6x6
f	3x3

- [ ] a->d, b->e, c->f
- [ ] a->e, b->d, c->f
- [x] a->e, b->f, c->d ✓

  Feedback :
  calculate the output size using ((n+2P-k)/S +1, (n+2P-k)/S) +1)

Padding and Output Size

Doing convolution without padding (assume that you are using a normal convolution with a k x k filter without shrinking it towards the edges etc.) :

- (•) Always reduces the size of the output ✓

  Feedback :
  Doing convolutions without padding always reduces the output size. You can see that from the formula as well: (n - k)/s will be less than n for all positive values of k.

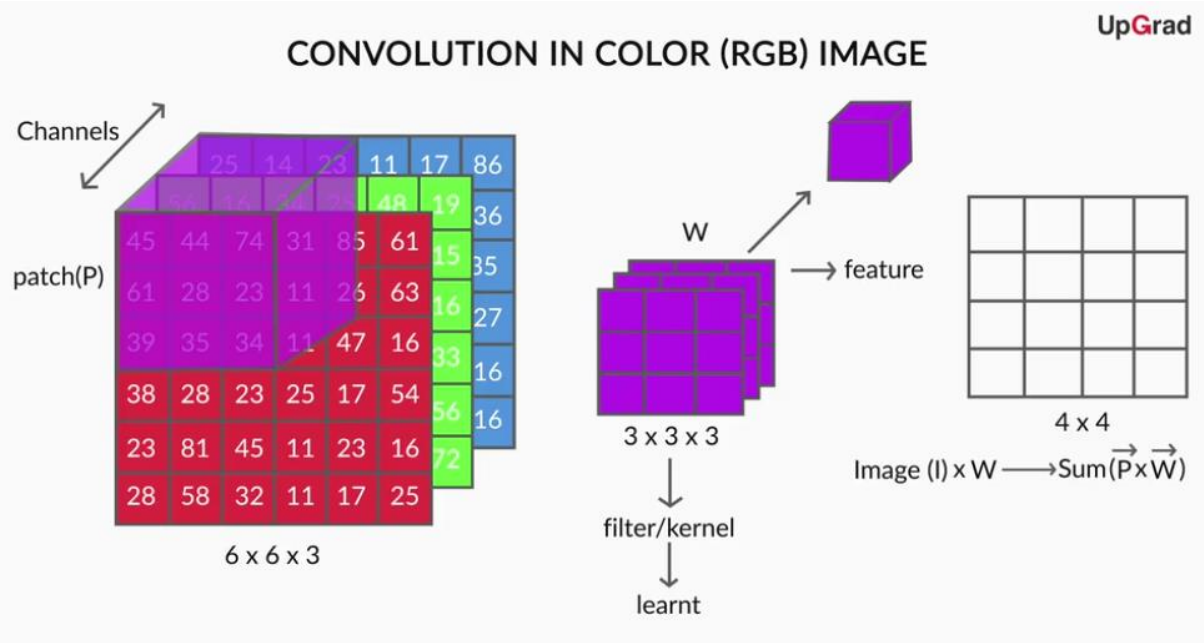
- ( ) Reduces the size of the output only when the stride length is greater than 1

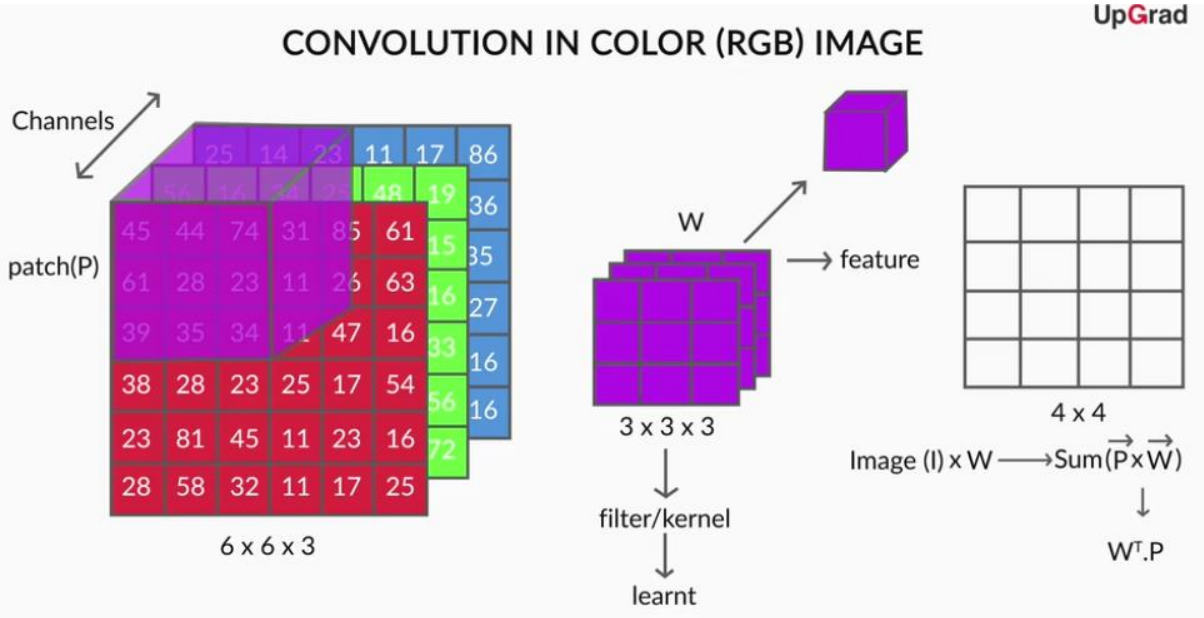
Weights of a CNN

So far, we have been doing convolutions only on 2D arrays (images), say of size 6x6. But most real images are coloured (RGB) images and are 3D arrays of size m x n x 3. Generally, we represent an image as a 3D matrix of size **height x width x channel.**

To convolve such images, use **3D filters**. Basic idea of convolution is still the same - take the element-wise product and sum up the values. The only difference is that now the filters will be 3-dimensional, for e.g. 3 x 3 x 3, or 5 x 5 x 3 (the last '3' represents the fact that the filter has as many channels as the image).

Let's now see how convolutions are performed on 3D arrays and what it is that a CNN 'learns' during **training**.





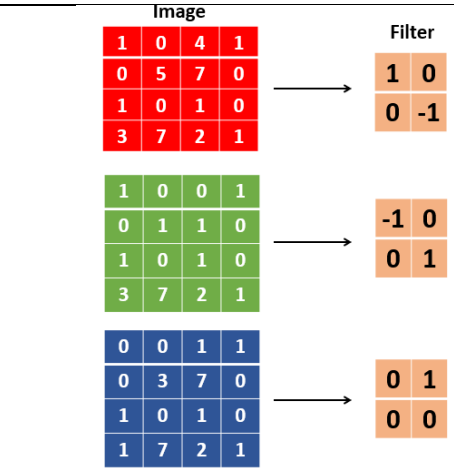
To summarise,

- We use **3D filters** to perform convolution on 3D images. For e.g. if we have an image of size (224, 224, 3), we can use filters of sizes (3, 3, 3), (5, 5, 3), (7, 7, 3) etc. (with appropriate padding etc.). We can use a filter of any size as long as the number of channels in the filter is the same as that in the input image.
- The **filters are learnt** during training (i.e. during backpropagation). Hence, the individual values of the filters are often called **the weights of a CNN.**

Comprehension - Weights and Biases

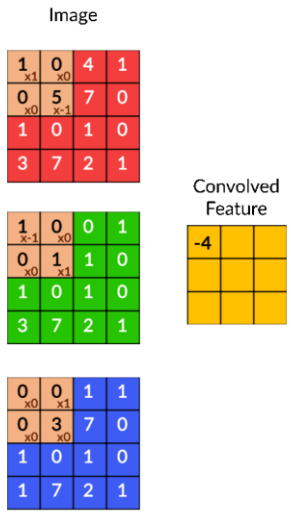
In the discussion so far, we have talked about only weights, but convolutional layers (i.e. filters) also have **biases**. Let's see an example to understand this concretely.

Suppose we have an RGB image and a (2, 2, 3) filter as shown below. The filter has three channels, and each channel of the filter convolves the corresponding channel of the image. Thus, each step in the convolution involves the element-wise multiplication of 12 pairs of numbers and adding the resultant products to get a **single scalar output**.



Channels in image and their corresponding Filter

The GIF below shows the convolution operation - note that in each step, a single scalar number is generated, and at the end of the convolution, a 2D array is generated:



We can express the convolution operation as a **dot product** between the weights and the input image. If you treat the (2, 2, 3) filter as a **vector** w of length 12, and the 12 corresponding elements of the input image as the vector p (i.e. both unrolled to a 1D vector), each step of the convolution is simply the **dot product** of w^T and p. The dot product is computed at every patch to get a (3, 3) output array, as shown here.

Apart from the weights, **each filter** can also have a **bias**. In this case, the output of the convolutional operation is a (3, 3) array (or a vector of length 9). So, the bias will be a vector

of length 9. However, a common practice in CNNs is that all the individual elements in the bias vector have the same value (called **tied biases**). For example, a tied bias for the filter shown above can be represented as:

$$w^T.x+b=\begin{bmatrix} sum(w^T.p_{11}) & sum(w^T.p_{12}) & sum(w^T.p_{13}) \\ sum(w^T.p_{21}) & sum(w^T.p_{22}) & sum(w^T.p_{23}) \\ sum(w^T.p_{31}) & sum(w^T.p_{32}) & sum(w^T.p_{33}) \end{bmatrix}+\begin{bmatrix} b & b & b \\ b & b & b \\ b & b & b \end{bmatrix}$$

$$=\begin{bmatrix} -4 & -5 & 5 \\ 3 & 11 & 6 \\ 0 & 1 & 0 \end{bmatrix}+\begin{bmatrix} b & b & b \\ b & b & b \\ b & b & b \end{bmatrix}$$

The other way is to use **untied biases** where all the elements in the bias vector are different, i.e. b_{11}, b_{12}, ... ,b_{mn}, but that is much less common than using tied biases.

Next segment will cover **feature maps.**

Convolution Operation

Given an image of size 128 x 128 x 3, a stride length of 1, padding of 1, and a kernel of size 3x3x3, what will be the output size?

- 128x128x3
- ✓ 128x128

  Feedback : *Though the input and filter are now 3D, the output of convolution will still be a 2D array. This is because, in each step of the convolution, the 9 elements of the filter will be 'dot product-ed' with the 9 elements of the 3D image, giving a single scalar number as the output.*
- 126x126x3
- 126x126

Weights in the Filters

Suppose you train the same network (i.e. the same architecture) for two different multiclass classification problems (such as classification of mammals and classification of flowers). Will the two resultant sets of weights be the same?

- The weights will be the same
- ✓ Weights will be different

  Feedback : *Weights will be different since each network will learn weights which are appropriate for the respective classification task.*

Number of Trainable Weights

What is the total number of trainable weights in a kernel/filter of size 3x3x3? Assume there are no biases.

- Zero - the weights are not trainable, they are specified as part of by the network architecture
- 9
- 3
- ✓ 27

  Feedback : *There are 27 weights in a (3, 3, 3) filter, which are all learnt during training.*

Number of Trainable Weights and Biases

What is the total number of trainable parameters in a kernel/filter of size 3x3x3? Assume that there is a single tied bias associated with the filter.

- Zero
- 9
- 27
- ✓ 28

  Feedback : *There are 27 weights and one bias in the (3, 3, 3) filter.*

Number of Operations in a Convolution

Given an image of size 3x3 and a kernel size 3x3, what will be the total number of multiplication and addition operations in convolution? Assume there is no padding and there are no biases (only weights). If there are m multiplication and n addition operations, the answer will be m+n.

- 15
- 15
- 17 ✓

  Feedback :
  There will be 3x3 multiplication operations and (3x3-1) addition operations (you need n-1 addition operations to add n numbers).
- 18

Number of Operations in 3D Convolution

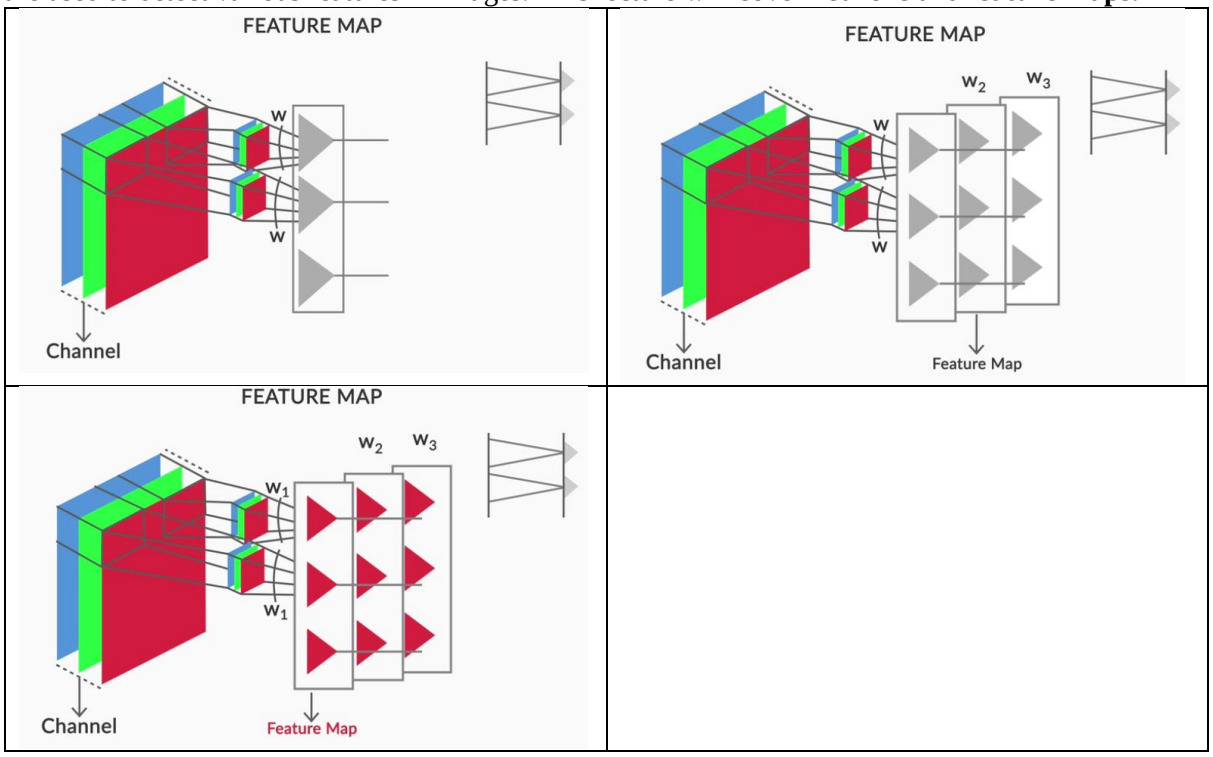
Given an image of size 3x3x3, a kernel of size 3x3x3, padding of 1, and stride length of 1, what will be the total number of multiplication and addition operations in the convolution? Assume there are no biases (only weights). If there are m multiplication and n addition operations, the answer will be m+n.

- 159
- 153
- 477 ✓

  Feedback :
  The output will be of size (n + 2p - k + 1) = (3, 3). In each step of the convolution, the 3 x 3 x 3 filter will be dot product-ed with a 3 x 3 x 3 array. This dot product will involve 27 pair-wise multiplications and then 26 addition operations, or 27+26 = 53 total operations. Since this operation will happen for each of the 3 x 3 cells in the output, the total number of operations is 53 x 9 = 477.

Feature Maps

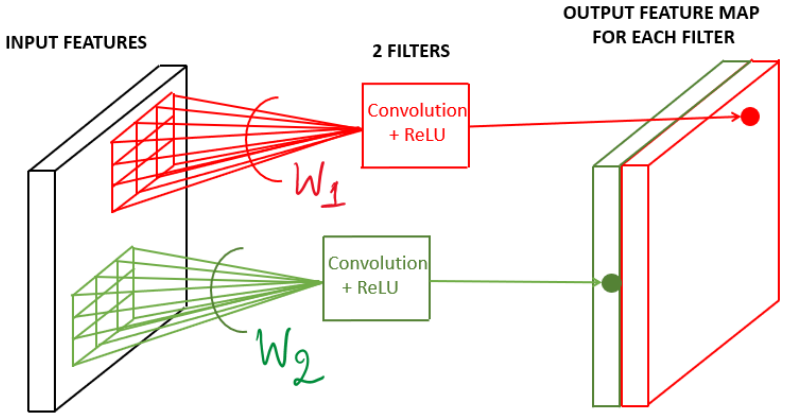
Values of the filters, or the weights, are learnt during training. Let's now understand how multiple filters are used to detect various features in images. This lecture will cover **neurons** and **feature maps.**



Important concepts and terms discussed above:

- **neuron** is a filter whose weights are learnt during training. A (3, 3, 3) filter (or neuron) has 27 weights. Each neuron looks at a particular region in the input (i.e. its 'receptive field').
- **feature map** is a collection of multiple **neurons** each of which looks at **different regions** of the input with the **same weights**. All neurons in a feature map extract the same feature (but from different regions of the input). It is called a 'feature map' because it is a mapping of where a certain feature is found in the image.

The figure below shows two neurons in a feature map (the right slab) along with the regions in the input from which the neurons extract features.



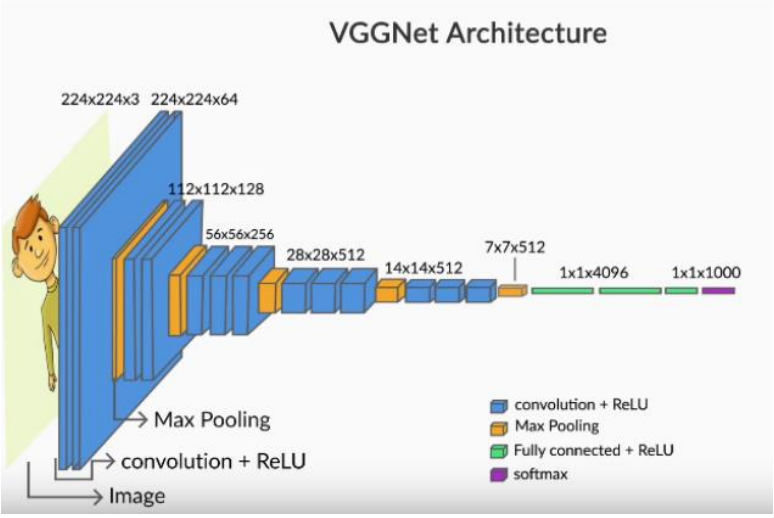
Feature Map

In the figure above, the two neurons produce two feature maps. Multiple such neurons convolve an image, each having a different set of weights, and each produces a feature map.

Comprehension - Feature Maps

Consider the VGGNet architecture shown below. The first convolutional layer takes the input image of size (224, 224, 3), uses a (3, 3, 3) filter (with some padding), and produces an output of (224, 224). This (224, 224) output is then fed to a **ReLU** to generate a (224, 224) **feature map**. 'feature map' refers to the (non-linear) **output of the activation function**, not what goes into the activation function (i.e. the output of the convolution).

Similarly, multiple other (224, 224) feature maps are generated using different (3, 3, 3) filters. In the case of VGGNet, 64 feature maps of size (224, 224) are generated, which are denoted in the figure below as the tensor 224 x 224 x 64. Each of the 64 feature maps try to identify certain features (such as edges, textures etc.) in the (224, 224, 3) input image.



VGG Net

The (224, 224, 64) tensor is the output of the **first convolutional layer.** In other words, the first convolutional layer consists of 64 (3, 3, 3) filters, and hence contains 64 x 27 trainable weights (assuming there are no biases).

The 64 feature maps, or the (224, 224, 64) tensor, is then fed to a **pooling layer**. Pooling layer is covered in the next segment.

Number of Feature Maps

If we use 32 filters (or kernels) of size 3x3x3 to convolve an image of size 128x128x3, how many feature maps will be created after the convolution?

- 1
- 3
- 0
- 32 ✓

Feedback :
We have already seen that each kernel of 3x3x3 will produce 1 feature map on an input of size 224x224x3. With 32 kernels, we will get 32 feature maps.

Output Size

Given an image of size 128x128x3, a stride of 1, padding of 1, what will be the size of the output if we use 32 kernels of size 3x3x3?

- 128x128x3
- 128x128x32 ✓

Feedback :
Each filter will produce a feature map of size 128x128 (with stride and padding of 1). Thus, 32 filters will produce 32 feature maps of size 128x128.

Output Size

Given an image of size 128x128x3, a stride of 1, zero padding, what will be the size of the output if we use 32 kernels of size 3x3x3?

- 128 x 128 x 3
- 126 x 126
- 128 x 128
- 126 x 126 x 32 ✓

Feedback :
The output of one kernel will be (128 - 3 +1) = 126 x 126, and thus, the output of 32 kernels will be 126 x 126 x 32.

Feature Map and Activation

Fill in the blank: The values in a feature map are ___ related to the weights of the filter generating the map.

- Linearly
- Non-linearly ✓

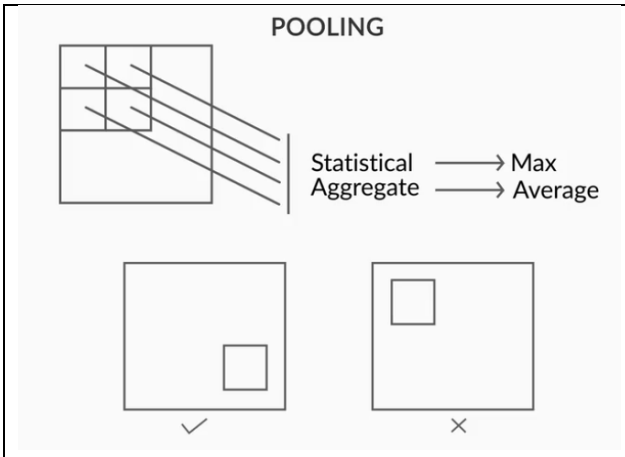
Feedback :
Feature map is the output from the activation function, which is usually non-linear (such as ReLU). That is, for a patch vector p and weight vector w, the values in the feature map will be $f(w^T . p)$ *where* f *is a non-linear activation function.*

Pooling

Earlier discussion on the experiments by Hubel and Wiesel observed the following statement:

- The strength of the response (of the retinal neurons) is proportional to the **summation** over the excitatory region.

After extracting features (as feature maps), CNNs typically **aggregate these features** using the **pooling layer.** Let's see how the pooling layer works and how it is useful in extracting higher-level features.



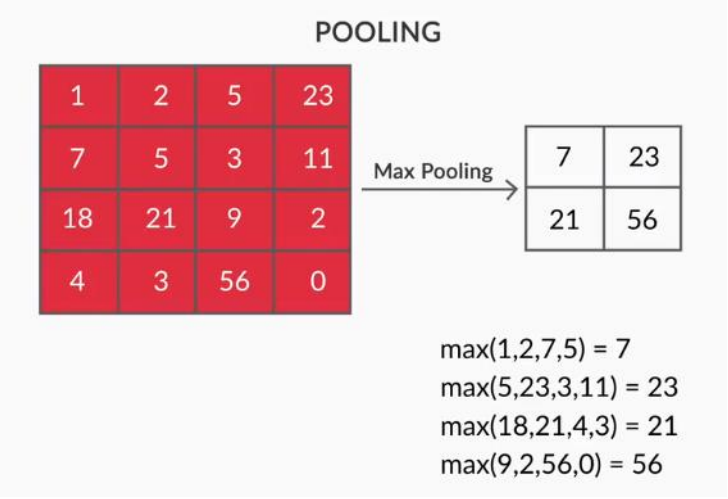
Pooling tries to figure out whether a particular region in the image has the feature we are interested in or not. It essentially looks at larger regions (having multiple patches) of the image and captures an **aggregate statistic** (max, average etc.) of each region. In other words, it makes the network **invariant to local transformations.**

The two most popular aggregate functions used in pooling are 'max' and 'average'.

- **Max pooling**: If any one of the patches says something strongly about the presence of a certain feature, then the pooling layer counts that feature as 'detected'.
- **Average pooling**: If one patch says something very firmly but the other ones disagree, the pooling layer takes the average to find out.

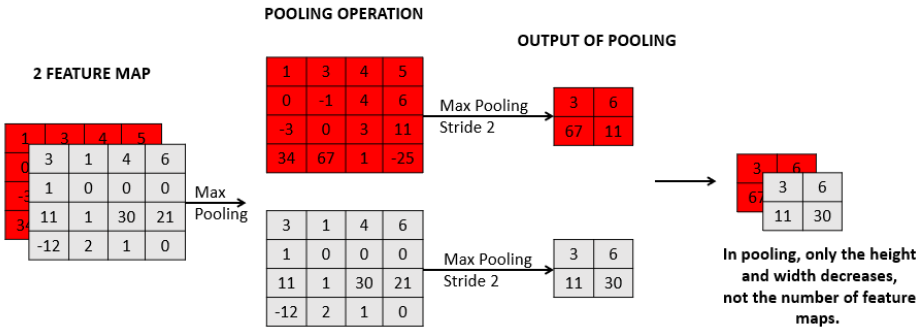
Let's now look at an example of max pooling and understand some potential drawbacks of the pooling operation.

Let's summarise the example of pooling used in the lecture:



Pooling

In the above figure, only the width and height of the input reduces. Let's extend this pooling operation to multiple feature maps:



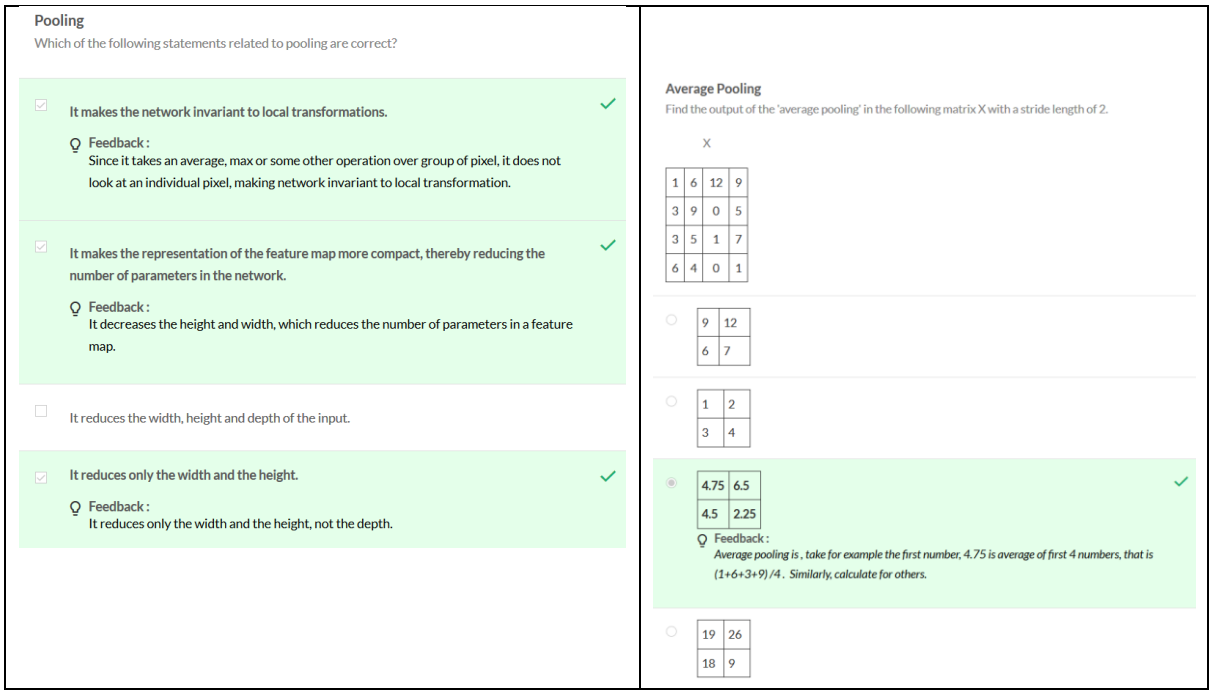
pooling in 3D

Pooling operates on each feature map independently. It reduces the size (width and height) of each feature map, but the number of feature maps remains constant.

Pooling has the advantage of making the representation more compact by **reducing the spatial size** (height and width) of the feature maps, thereby reducing the number of parameters to be learnt. On the other hand, it also **loses a lot of information**, which is often considered a potential disadvantage. Having said that, pooling has empirically proven to improve the performance of most deep CNNs.

Can we design a network without pooling? **Capsule networks** were designed to address some of these potential drawbacks of the conventional CNN architecture.

Next segment summarises all the concepts discussed till now.



Pooling parameters

How many trainable parameters are there in the pooling layer?

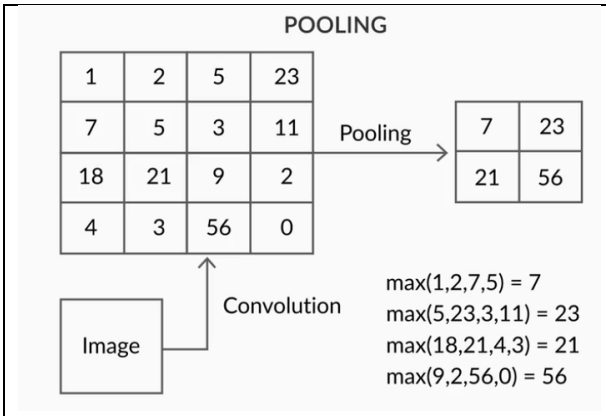
- 0 ✓

  Feedback :
  There are no parameters in pooling. The pooling layer just computes the aggregate of the input. For e.g. in max pooling, it takes max over group of pixels. We do not need to adjust any parameter to take max.

- 1
- 2
- It depends on the stride, padding and size of the input image.

Putting the Components Together

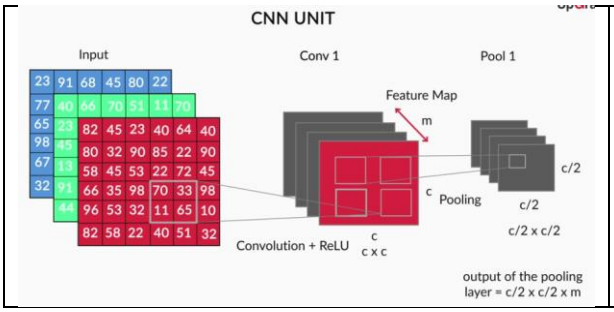
We have seen all the main components of a typical CNN - convolutions, feature maps, pooling layers etc. Let's now quickly summarise and put them together to get an overall picture of CNNs' architecture.

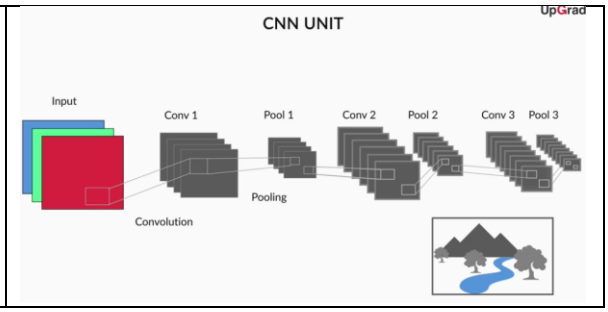


A **typical CNN layer (or unit)** involves the following **two components** in sequence:

- We start with an original image and do convolutions using multiple filters to get multiple feature maps.
- A pooling layer takes the statistical aggregate of the feature maps

Typically, deep CNNs have **multiple such CNN units** (i.e. feature map-pooling pairs) arranged sequentially. The following lecture will discuss this in detail.





A typical CNN has the following sequence of CNN layers:

- An input image is convolved using multiple filters to create **multiple feature maps**
- Each feature map, of size (c, c), is **pooled** to generate a (c/2, c/2) output (for a standard 2 x 2 pooling).
- The above pattern is called **a CNN layer or unit**. Multiple such CNN layers are stacked on top of one another to create deep CNN networks.

Pooling reduces only the height and the width of a feature map, not the depth (i.e. the number of channels). In m feature maps each of size (c, c), pooling operation will produce m outputs each of size (c/2, c/2).

Pooling Operation

Given an input of size 224x224x3 and stride of 2 and filter size of 2x2, what will be the output after the pooling operation?

- 112x112
- **112x112x3** ✓

  Feedback :
  Pooling will reduce only the height and the width, not the depth. So, depth of channel, here '3' will be there in the output. You can even calculate the output size by the formula ((n+2p-k)/s +1). In pooling, we don't pad the input and the value of padding will always be '0'.
- 224x224
- 224x224x3

A CNN Unit

What does a typical CNN 'unit' (also called a CNN 'layer') comprise of?

- A collection of feature maps
- A pooling layer
- **A collection of feature maps followed by a pooling operation** ✓

  Feedback :
  What we refer to as a CNN layer or unit is a collection of m feature maps each of which is pooled to generate m outputs. Typically, the output of pooling reduces the size to half, since the most common form of pooling is with a stride of 2.
- None of these

Size of Feature Maps

In a typical deep CNN, the size of each subsequent feature map reduces with the depth of the network. The size reduction is typically done in two ways - 1) convolution without padding or 2) by pooling.

What is the main reason we prefer a lower dimensional output of an image from the network?

- Lower dimensional outputs capture more information than a higher-dimensional one
- Lower dimensional outputs make the network easier to train (i.e. during backpropagation)
- **We want a compact representation of the image as the output, one which preferably captures only the useful features in the image** ✓

  Feedback :
  The reason we want a compact representation of the image is to get rid of all the redundancies in the image. For e.g. all the 224 x 224 x 3 pixels may not be required to do a classification or object detection task, just a smaller vector (say of length 5000) may be enough.
- Low dimensional outputs have empirically proven to be more useful than high dimensional ones

Summary

Basics of convolutional neural networks and their common applications in computer vision such as image classification, object detection, etc.

CNNs are not limited to images but can be extended to videos, text, audio etc.

The design of CNNs uses many observations from the **animal visual system,** such as each retinal neuron looks at its own (identical) **receptive field**, some neurons respond proportionally to the **summation over excitatory regions** (pooling), the images are perceived in a **hierarchical manner**, etc.

Images are naturally represented in the form of arrays of numbers. Greyscale images have a single channel while colour images have three channels (RGB). The number of channels or the 'depth' of the image can vary depending on how we represent the image. Each channel of a pixel, usually between 0-255, indicates the 'intensity' of a certain colour.

Specialised **filters, or kernels** can be designed to extract specific features from an image (such as vertical edges). A filter **convolves** an image and extracts features from each 'patch'. Multiple filters are used to extract different features from the image. Convolutions can be done using various **strides and paddings.**

The formula to calculate the output shape after convolution is given by:

$\left(\frac{n+2P-k}{S}+1\right), \left(\frac{n+2P-k}{S}+1\right)$, where

- The image is of size- n x n
- The filter is k x k
- Padding is P
- Stride is S

The filters are **learned during training** (backpropagation). Each filter (consisting of weights and biases) is called a **neuron**. Multiple neurons are used to convolve an image (or feature maps from the previous layers) to generate new **feature maps.** The feature maps contain the output of **convolution + non-linear activation** operations on the input.

A typical **CNN unit** (or layer) in a large CNN-based network comprises multiple filters (or neurons), followed by non-linear activations, and then a **pooling layer**. The pooling layer computes a statistical aggregate (max, sum etc.) over various regions of the input and reduces sensitivity to minor, local variations in the image. Multiple such CNN units are stacked together, finally followed by some fully connected layers, to form **deep convolutional networks.**

In the next session, you will learn to build and train CNNs using Python + Keras + GPUs.