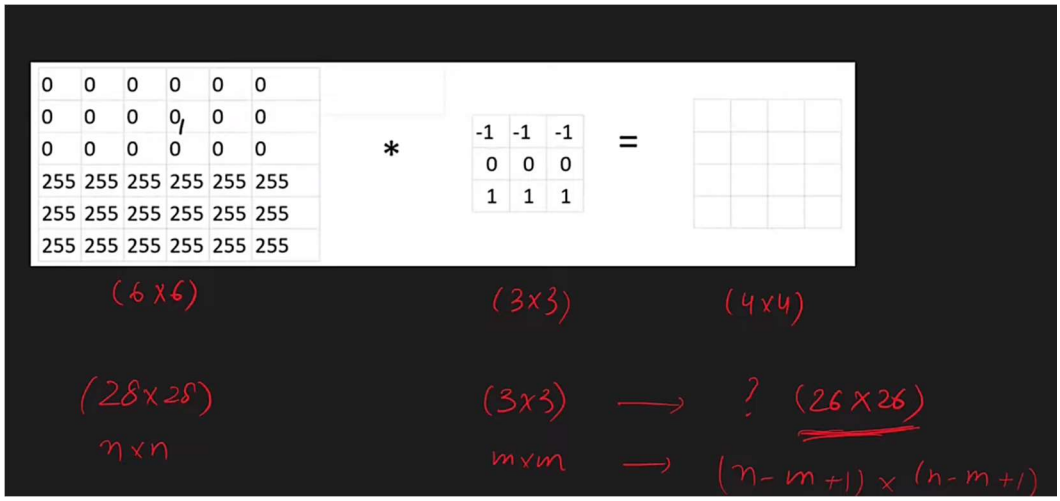
For Gray Scale Image

Image 28x28 and Filter 3x3 – So Result 26x26



If image is 6x6 and Filter is 3x3 then what is the result of Feature matrix

Image 6x6 = nxn

Filter 3x3 = mxm

Formula: (n-m+1) x (n-m+1) = (6-3+1) x (6-3+1) = 4x4

If image is 28x28 and Filter is 3x3 then what is the result of Feature matrix

Image 28x28 = nxn

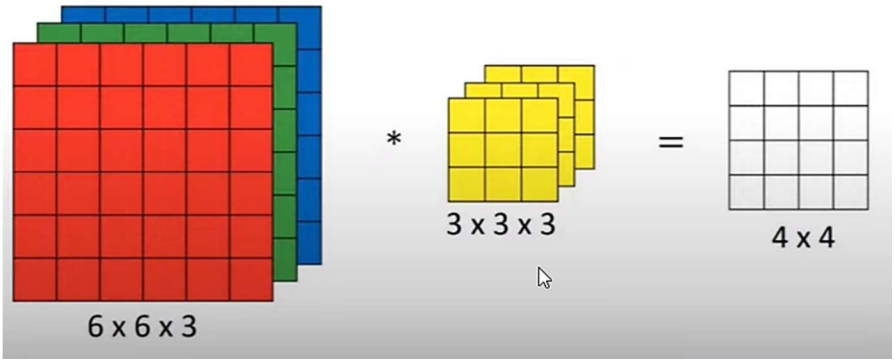
Filter 3x3 = mxm

Formula: (n-m+1) x (n-m+1) = (28-3+1) x (28-3+1) = 26x26

To work on image – in pre-processing image should be similar size by rescaling

For Color Images

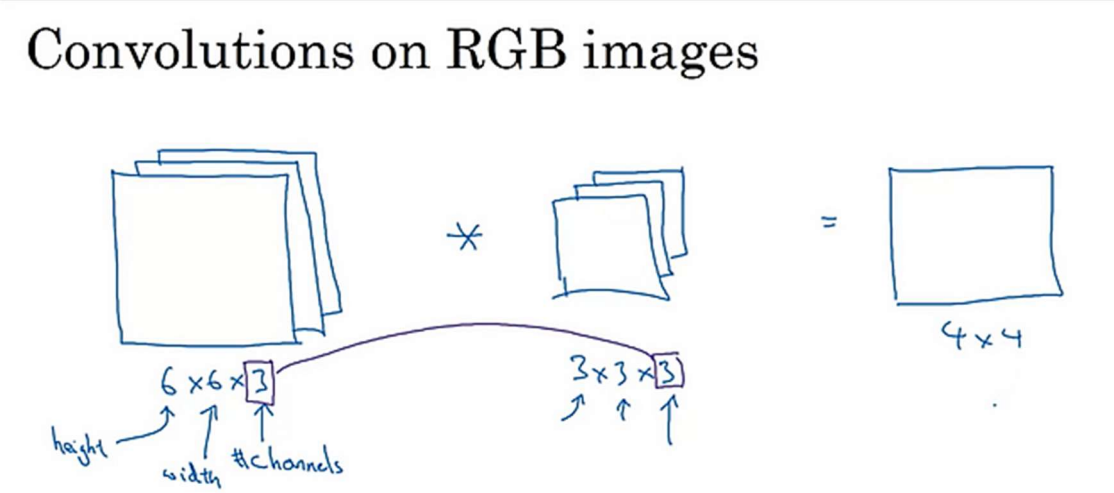
Image 6x6x3 and Filter 3x3x3 – So Result 4x4



Note: In color images with image size third number represents channel (which means number of colors i.e. rgb)

6 rows, 6 columns, 3channels (color red, green, blue) – 6x6x3

Reference: https://medium.com/swlh/convolutional-neural-networks-part-3-convolutions-over-volume-and-the-convnet-layer-91fb7c08e28b

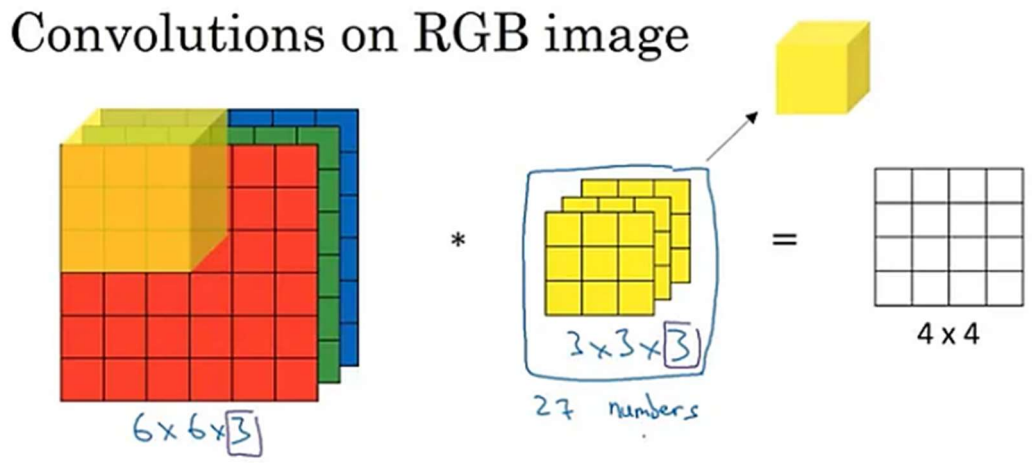


Note: Convolving a 6 by 6 by 3 volume with a 3D filter (a 3 by 3 by 3 filter).

An RGB image is represented as a 6 by 6 by 3 volume, where the 3 here responds to the 3 color channels (RGB). So the filter itself will also have 3 layers corresponding to the red, green, and blue channels.

From above figure, notice that:

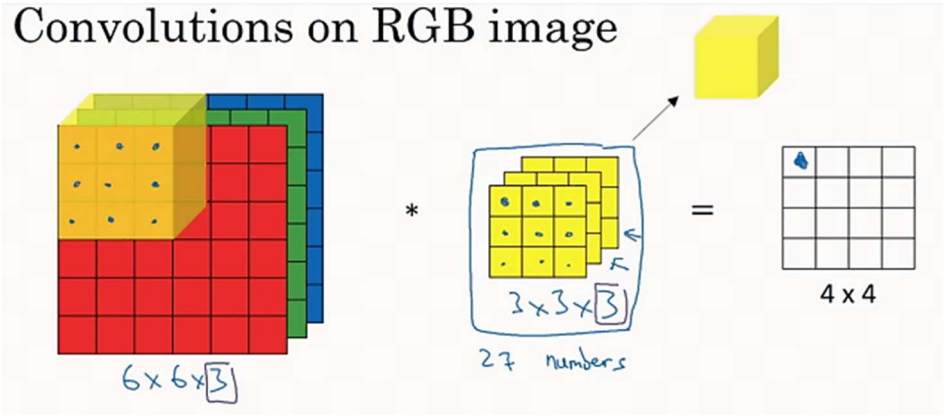
- The first 6 is the height of the image,
- The second 6 is the width, and
- The 3 is the number of channels.

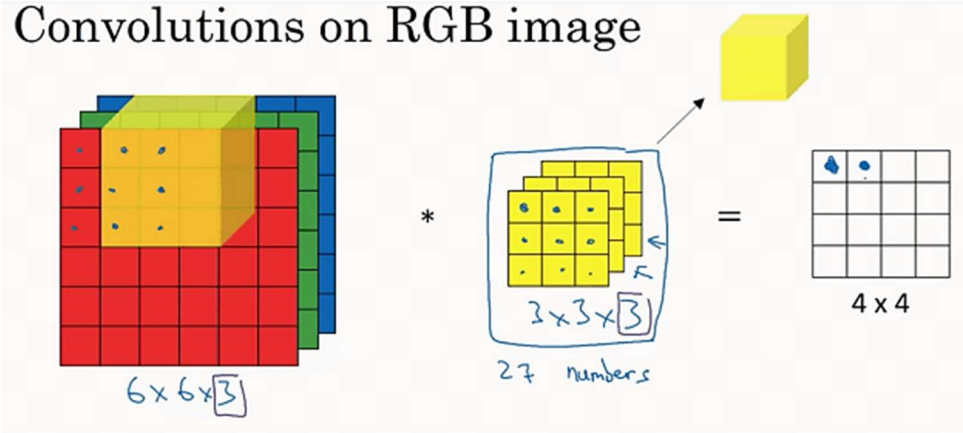


A representation of a 3D convolution.

Here's a 6 by 6 by 3 image, and a 3 by 3 by 3 filter.

Notice that the 3 by 3 by 3 filter has 27 numbers, or 27 parameters, that's three cubes.





Note: Take 9 values collectively apply filter and then get one summarized value. Do same for the rest.

So how do you convolve this RGB image the 3D filter?

First take each of the 27 numbers of the filter and multiply them with the corresponding numbers from the red, green, and blue channels of the image,i.e

Take the first 9 numbers from red channel, then the 3 beneath it to the green channel, then the three beneath it to the blue channel, and multiply it with the corresponding 27 numbers that gets covered by this yellow cube show on the left.

Then add up all those numbers and this gives you this first number in the output, and then to compute the next output you take this cube and slide it over by one, and again, due to 27 multiplications, add up the 27 numbers, that gives you this next output, do it for the next number over, for the next position over, that gives the third output and so on.

Feature Map

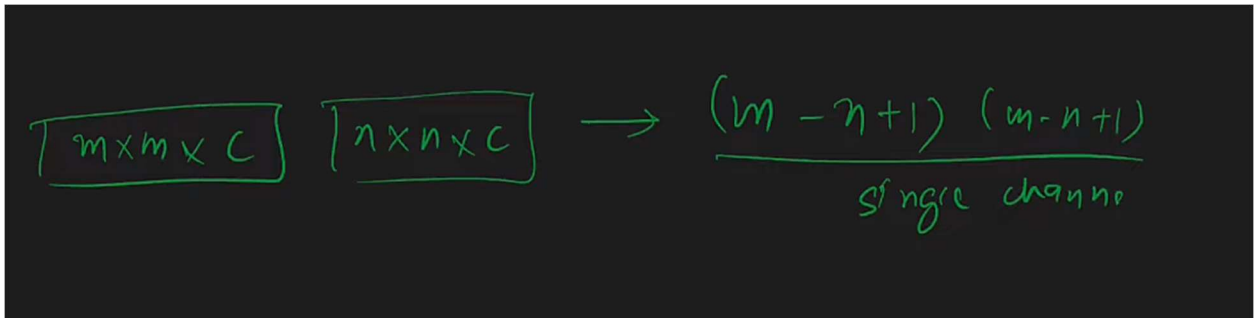
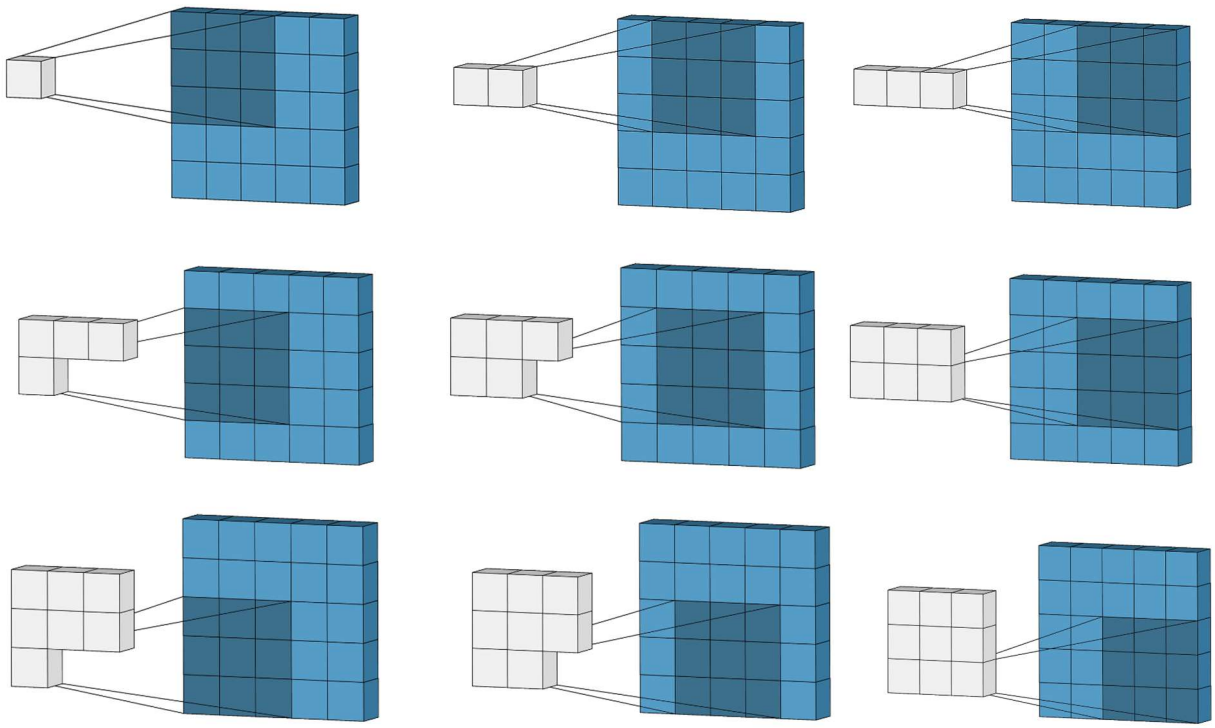


Image reading process



Above process shows that filters reads central information multiple times, but left and right or top and bottom edges read only once. So it create biasness that means model did not train accurately. To resolve this we need:

- Padding
- Stride

Convolutional Neural Network

Convolutional Neural Networks (CNNs or ConvNets) are specialized neural architectures that are predominantly used for several computer vision tasks, such as image classification and object recognition

Convolutional neural networks have three main kinds of layers, which are:

- Convolutional layer
- Pooling layer
- Fully-connected layer

Note: The first convolutional layer may be followed by several additional convolutional layers or pooling layers; and with each new layer.

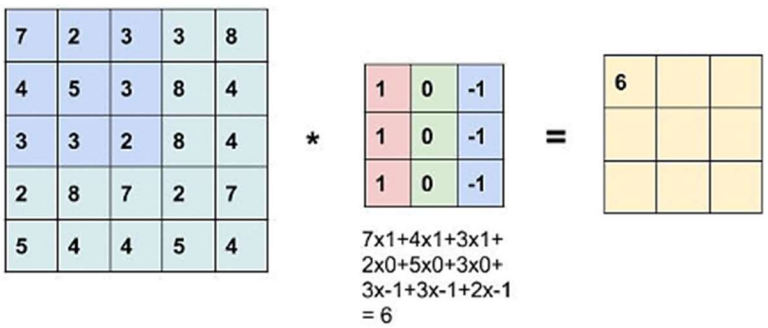
The convolutional layer includes **input data**, **a filter**, and a **feature map.**

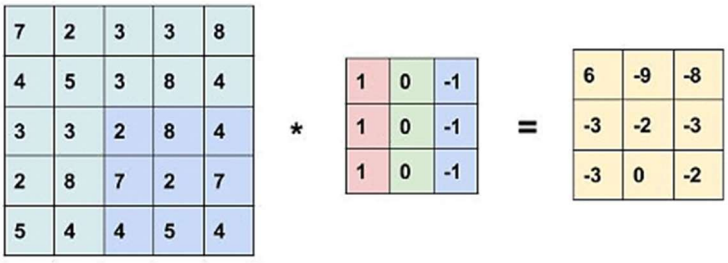
Example: The image is made up of a matrix of pixels in 3D, representing the three dimensions of the image: height, width, and depth.

The filter — which is also referred to as kernel — is a two-dimensional array of weights, and is typically a 3×3 matrix.

Subsequently, the filter shifts by a stride, and this whole process is repeated until the kernel slides through the entire image, resulting in an output array.

The resulting output array is also known as a ***feature map, activation map, or convolved feature***





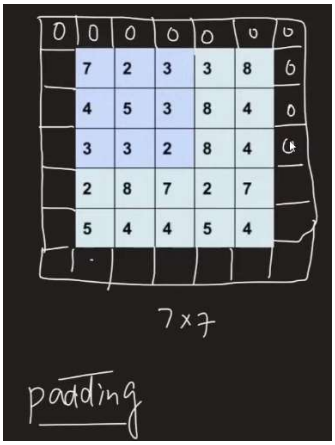
Executing filter one by one

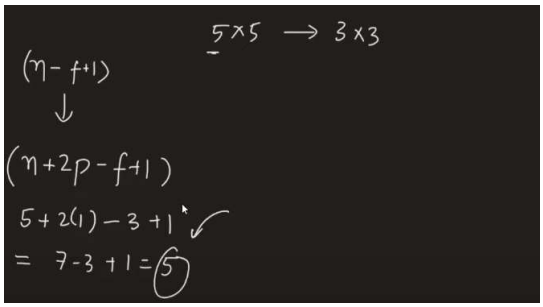
It shows that edges are reading one time, while center parts are reading multiple time that creates biasness. To resolve this add ***padding***.

Padding

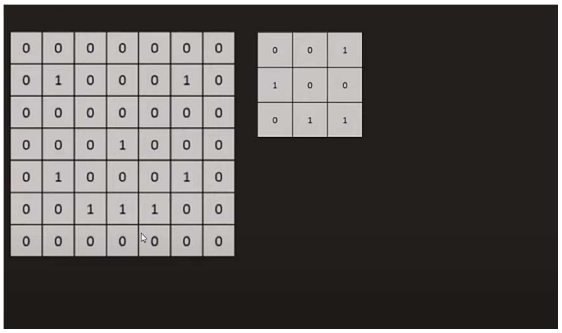
Add extra layer of 0 in top-bottom-left-right. Due to padding filter size change**.**

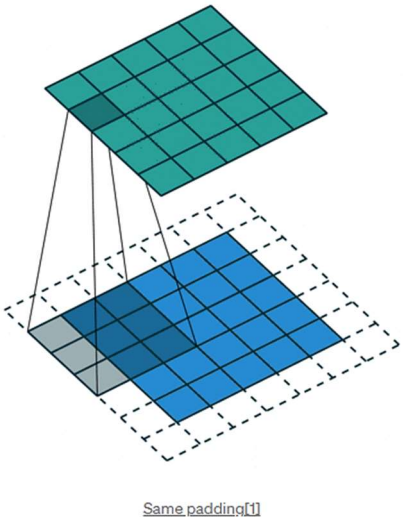
Before introducing padding





After introducing padding





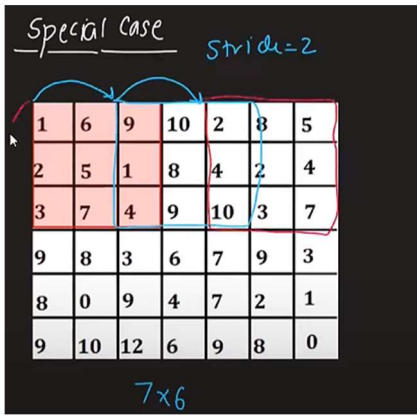
Same padding[1]

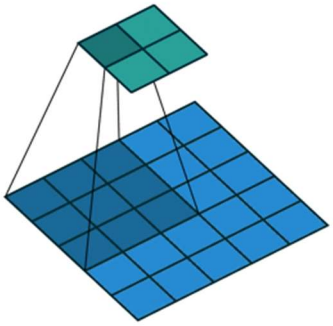
Note:

- In zero padding – we add layer.
- In valid padding – we drop layer

Padding change filter size which create error. To resolve this introduce ***stride*** which means jump. Normally, we add 2x2 stride.

Formula: n+2p-f+1 (here f = filter size, p = padding layers)





A stride 2 convolution[1]

This is commonplace in convolutional neural networks, where the size of the spatial dimensions are reduced when increasing the number of channels.

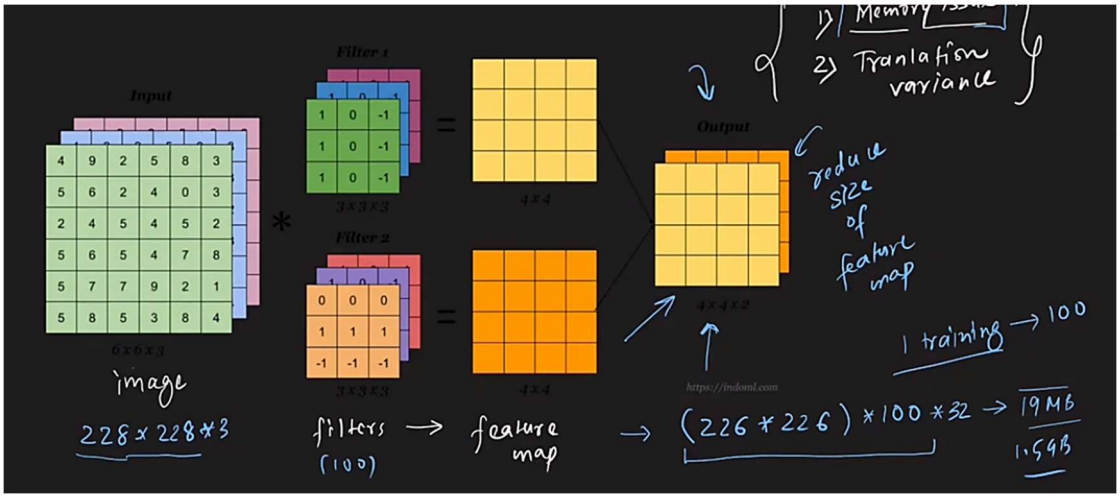
The idea of the stride is to skip some of the slide locations of the kernel.

- A stride of 1 means to pick slides a pixel apart, so basically every single slide, acting as a standard convolution.
- A stride of 2 means picking slides 2 pixels apart, skipping every other slide in the process, downsizing by roughly a factor of 2,
- A stride of 3 means skipping every 2 slides, downsizing roughly by factor 3, and so on.

Note:

- Normally, we add 2x2 stride.
- Add Padding and stride together.
- Stride reduces size
- This process works on 2D gray scale images properly. But not on 3D colored images.

Problem with Convolution

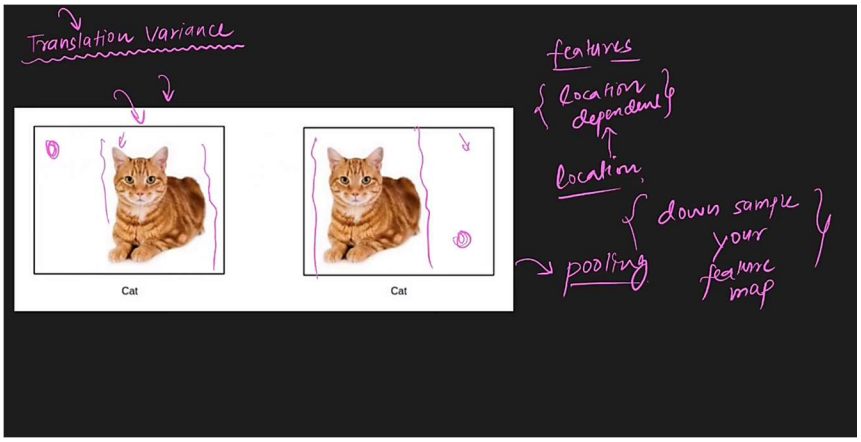


If we have rgb image or large image and we use padding and stride so what are the issues came? To resolve this we need small feature size

Memory – Padding increases the size of the image, which in turn increases the number of computations and the amount of memory needed. This can be particularly problematic with deep networks or large batch sizes.

Translation variance – location dependence error – If the network is too dependent on the specific location of features due to the padding, it might not generalize well to different positions of the same feature in the image.

Feature Size and Stride – A large padding can increase the size of the feature maps but might introduce artifacts or increase computational overhead.



To resolve issues:

Downsampling/ Downsizing: Consider using pooling layers (like max pooling) to reduce the spatial dimensions gradually and control memory usage.

To reduce feature matrix size (means memory issue reolve)

After each convolution operation, we have the application of a Rectified Linear Unit (ReLU) function, which transforms the feature map and introduces nonlinearity.

Then add Pooling.

Note: After convolution layer pooling layer added

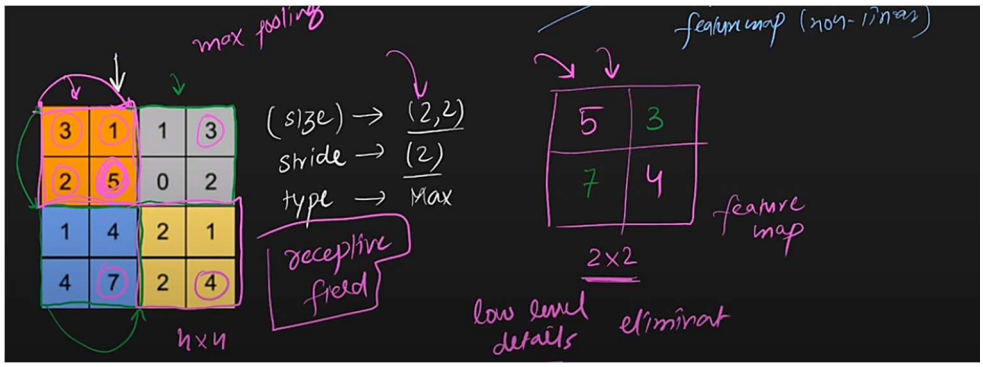


Image: Max Pooling (picks larger number from all)

Max Pooling – brings 4boxes result in one box (by picking maximum value from all four values)

Min Pooling – brings 4boxes result in one box (by picking minimum value from all four values)

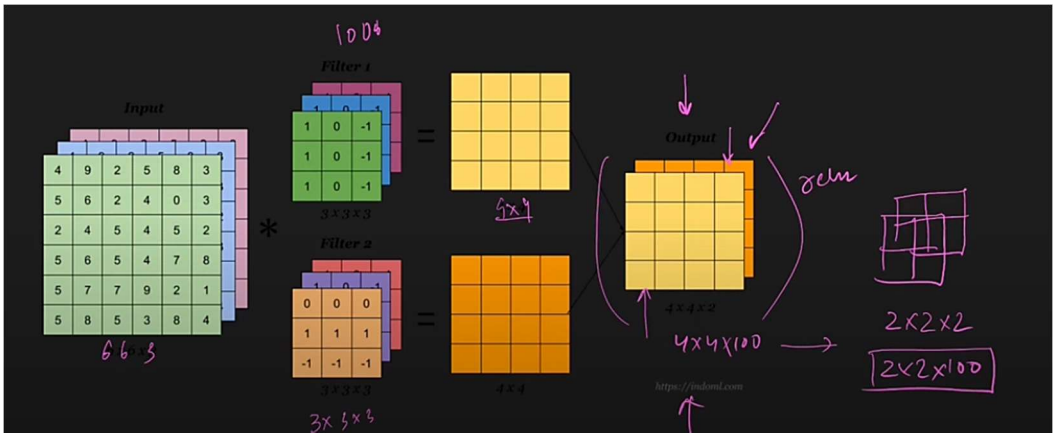
Global Positioning Pooling – In whole feature matrix pick only one number that is largest of all (matrix 1x1)

Average Pooling – brings 4boxes result in one box (by picking average value from all four values)

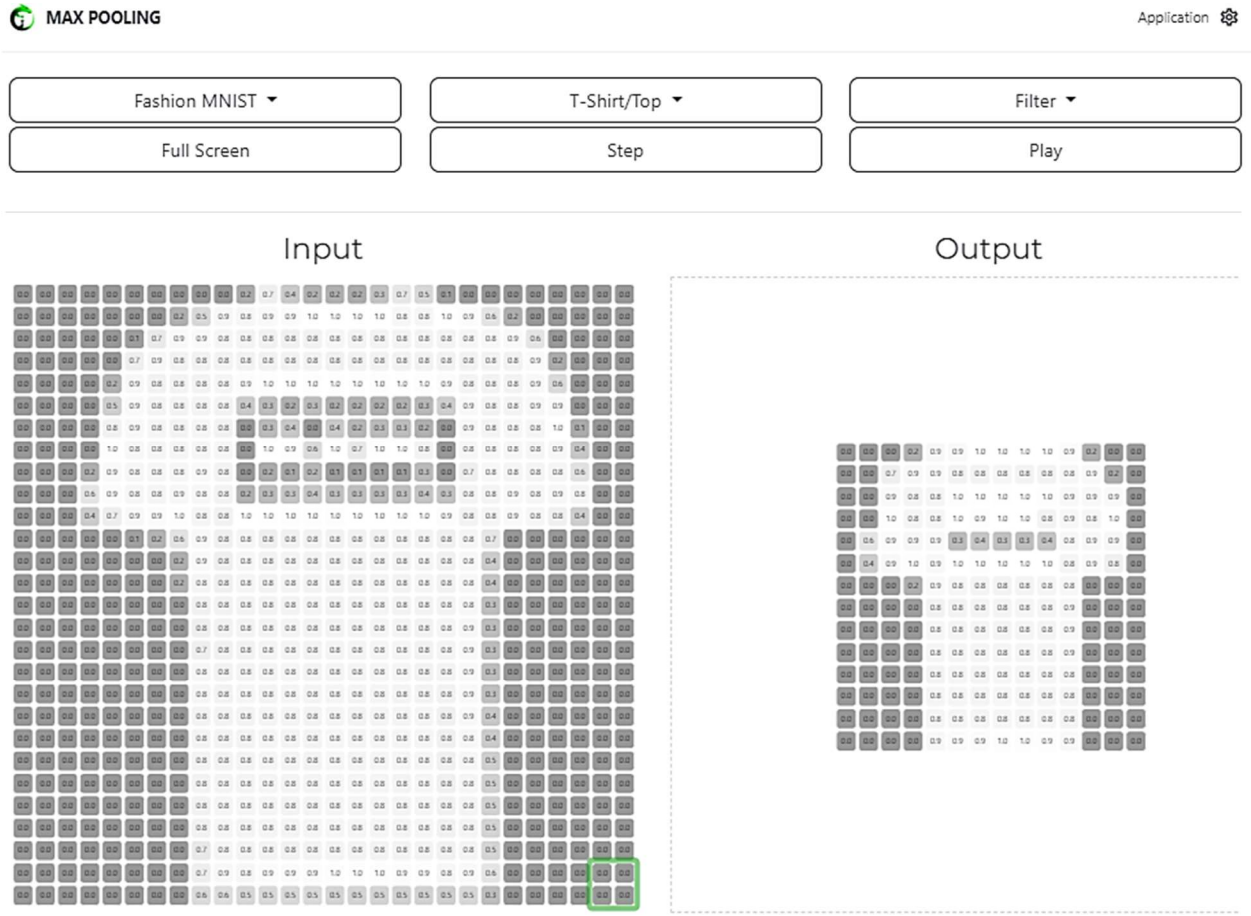
Adaptive Pooling – Its primary goal is to output a fixed-size feature map regardless of the size of the input image, making it extremely useful for tasks where the input image size varies but a consistent input size is required for the following layers, such as fully connected layers in a classification task.

Adaptive Average Pooling is a powerful tool in CNNs that allows for flexible and adaptable pooling of input data, making it easier to handle inputs of varying sizes and ensuring consistent output sizes for further processing.

Problem on volumes



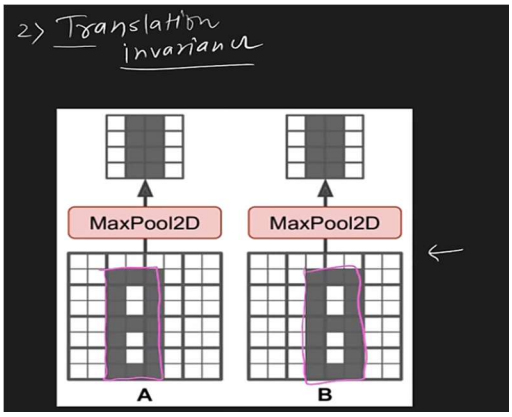
Reference: https://deeplizard.com/resource/pavq7noze3

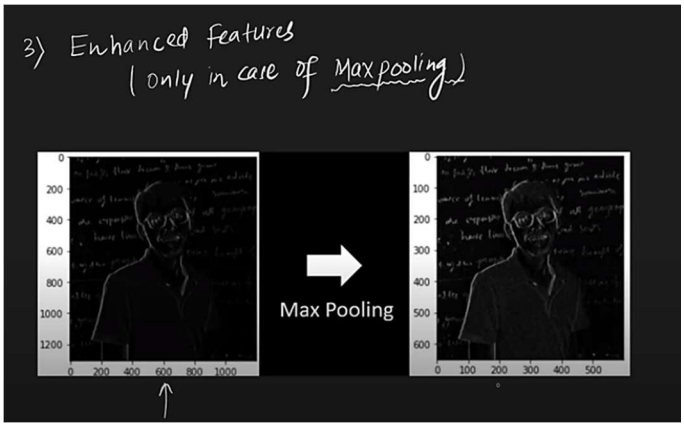


In Image pool down only center values of shirt.

Advantages of max pooling:

- Pooling also reduces translation inversion problem.
- Enhanced features only in case of max pooling





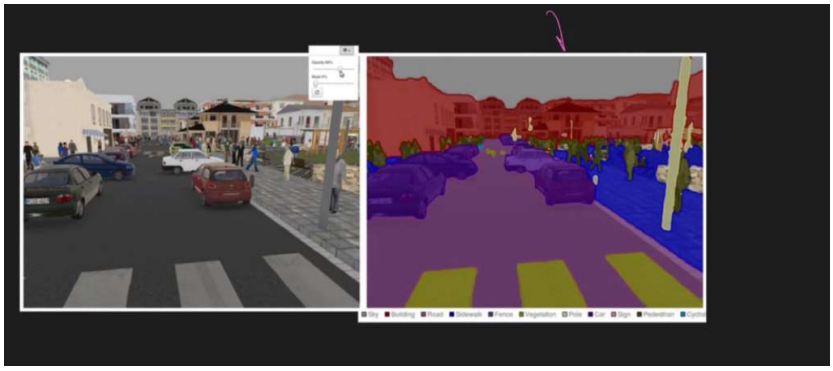
Retains Important Features: Preserves the most prominent features of the input by selecting the maximum value, which helps in capturing significant patterns.

Reduces Dimensionality: Effectively reduces the spatial dimensions of the feature maps, leading to decreased computational load and fewer parameters.

Improves Translation Invariance: Helps the network become more invariant to small translations of features in the input.

Disadvantages of max pooling:

Image Segmentation – spaces remove but in image everything matters.



Loss of Information: Discards all but the maximum value in each pooling window, potentially losing detailed information about other values.

Not Suitable for All Tasks: May not be ideal for tasks where preserving the average or overall pattern of features is important.

Reference: https://medium.com/latinxinai/convolutional-neural-network-from-scratch-6b1c856e1c07

Besides the weights in the filter, we have other three important parameters that need to be set before the training begins:

Number of Filters:

This parameter is responsible for defining the depth of the output. If we have three distinct filters, we have three different feature maps, creating a depth of three.

Stride:

This is the distance, or number of pixels, that the filter moves over the input matrix.

Zero-padding:

This parameter is usually used when the filters do not fit the input image. This sets all elements outside the input matrix to zero, producing a larger or equally sized output. There are three different kinds of padding:

- **Valid padding:** Also known as no padding. In this specific case, the last convolution is dropped if the dimensions do not align.
- **Same padding:** This padding ensures that the output layer has the exact same size as the input layer.
- **Full padding:** This kind of padding increases the size of the output by adding zeros to the borders of the input matrix.

Note: After each convolution operation, we have the application of a Rectified Linear Unit (ReLU) function, which transforms the feature map and introduces nonlinearity.

Pooling Layer

The pooling layer is responsible for reducing the dimensionality of the input. It also slides a filter across the entire input — without any weights — to populate the output array. We have two main types of pooling:

Max Pooling: As the filter slides through the input, it selects the pixel with the highest value for the output array.

Average Pooling: The value selected for the output is obtained by computing the average within the receptive field.

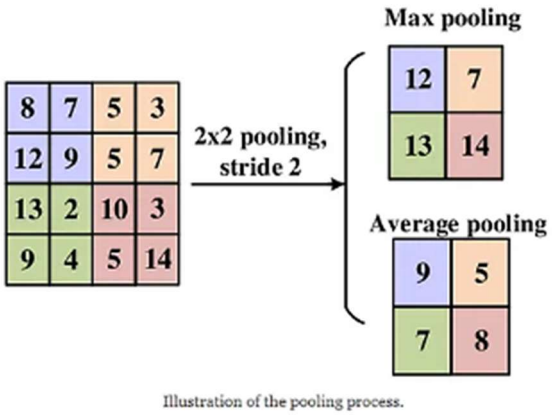


Illustration of the pooling process.

The pooling layer serves the purpose of reducing complexity, improving efficiency, and limiting the risk of overfitting

Note: While both convolutional and pooling layers tend to use ReLU functions, fully-connected layers use the Softmax activation function for classification, producing a probability from 0 to 1.

Types of Pooling

Max Pooling: Selects the maximum value from each pooling window to retain prominent features.

Average Pooling: Computes the average value from each pooling window to smooth features and reduce detail.

Global Average Pooling: Averages all values in the feature map to produce a single value per map, used for dimensionality reduction.

Global Max Pooling: Takes the maximum value from the entire feature map to reduce each map to a single value, preserving the most significant feature.

Min Pooling: Selects the minimum value from each pooling window to capture the least prominent features.

L2-Norm Pooling: Computes the L2 norm (Euclidean norm) of values in the pooling window to measure feature magnitude.

Fractional Pooling: Pools with non-integer stride to reduce feature maps to non-integer dimensions for more control over down-sampling.

Adaptive Pooling: Adjusts pooling window size dynamically to produce a fixed output size regardless of input size.