Unit - 2

UNIT II Convolutional Networks : Convolution operation - Motivation - Pooling - Convolution and Pooling as strong prior - Efficient convolution algorithms - Unsupervised features - Sequence Modeling: Recurrent and Recursive Nets - LSTM Networks - Applications - Computer Vision - Speech Recognition - Natural Language Processing.

Convolutional Neural Networks

Convolution Process

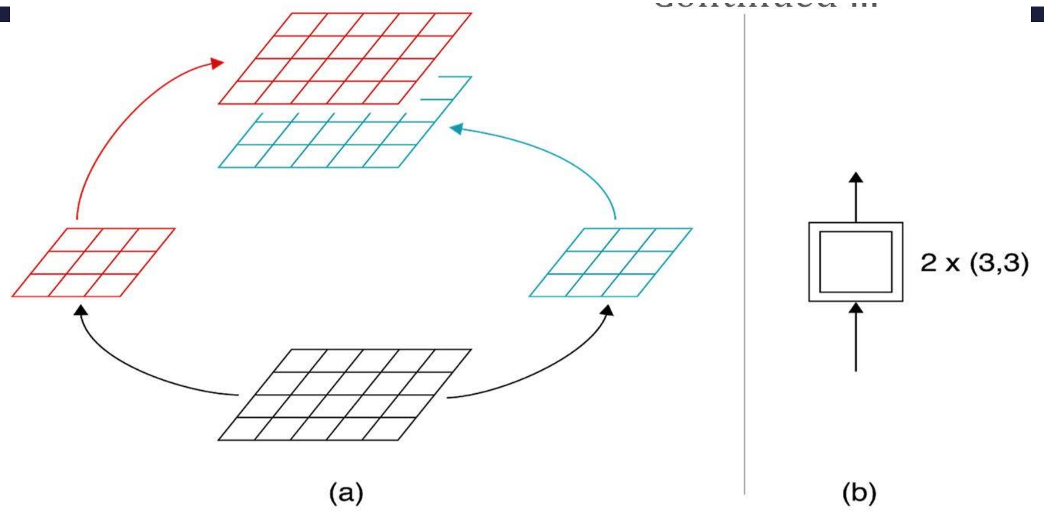
Introduction to Convolution

•A Convolution layer is most famously used for processing 2D images

Eg. Convolutional Neural Networks (CNN) are used to identify faces in photos

•To get a general sense, let's look at Convolution on a 2D image
•To do convolution we will create a tiny square filter, say 3x3 size.
•Each grid (or cell) in the filter is called a '*weight*'
•Move the tiny filter over the entire *input image*

Convolution



Steps to filter input image

For every pixel in the input image:

- Centre the 3x3 filter *over* it
- Multiply the value of each *weight* in the filter with the *value of the pixel* in input image below it.
- Add up those values
- Put the result for that pixel in the *output image*

- Convolution is a mathematical operation -> Combo of Multiplication and Addition

- Convolution in 1D is done with two lists of numbers (of equal lengths)
  - List 1 - input values (say pixel values) - ***I***
  - List 2 - weight values (weights of a filter) – ***W***
- Say ***I*** and ***W*** are lists of length ‘***n***’
- Do elementwise multiplication of list1 and list2
- Add the outputs in above step to get the convolution result

Conv = $\boldsymbol{I}(1) * \boldsymbol{W}(1) + \boldsymbol{I}(2) * \boldsymbol{W}(2) + \boldsymbol{I}(3) * \boldsymbol{W}(3) + \,.\,.\, + \boldsymbol{I}(n) * \boldsymbol{W}(n)$

A simple exercise on Convolution

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

Input image

To illustrate convolution,
take a 6 X 6 input image

Convolve input image with
a 3 X 3 filter

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

6 X 6 image

$*$

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

3 X 3 filter

1.Let’s take the first 3 X 3 sub-matrix from the 6 X 6 image and multiply it with the filter.

2.The first element of the 4 X 4 output will be the sum of the element-wise product of these values, i.e.

3*1 + 0 + 1*-1 + 1*1 + 5*0 + 8*-1 + 2*1 + 7*0 + 2*-1 = -5.

3.To calculate the second element of the 4 X 4 output, we will shift our filter one step towards the right and again get the sum of the element-wise product.

4.Similarly, we will convolve over entire image and get a 4 X 4 output as shown at bottom right

5. So, convolving a 6 X 6 input with 3 X 3 filter gave an output of 4 X 4.

Convolution is elementwise product and addition

3^{1}	0^{0}	1^{-1}
1^{1}	5^{0}	8^{-1}
2^{1}	7^{0}	2^{-1}

0^{1}	1^{0}	2^{-1}
5^{1}	8^{0}	9^{-1}
7^{1}	2^{0}	5^{-1}

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

Convolution case 1: Single filter to single channel Image (SF-SCI image)

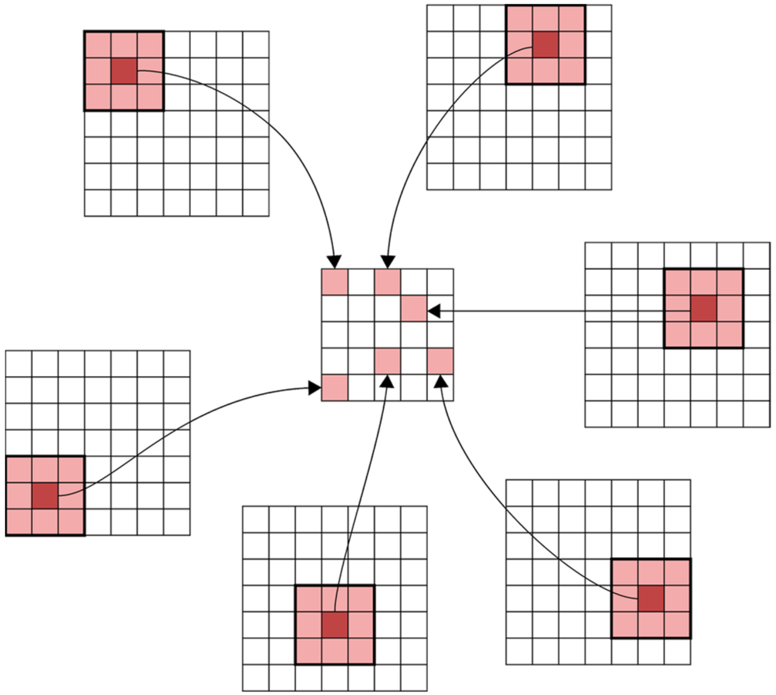


Fig : Convolution on a 2D image

Convolution case 1: Single filter to 2D image (SF-SCI, i.e. grayscale image)

- See previous slide
- To convolve an image with a filter, we move the filter across the image and apply it at each position.
- The resulting value then becomes the value for that pixel in the result.
- Previous figure shows some positions of the filter in the input, and the positions where their computed values go into the output.
- Note that because the filter can't extend past the edges, the input is 7 by 7 but the output is only 5 by 5.
- As a general rule, if a matrix (n,n) is convolved with a (f,f) filter, result is (n-f+1 , n-f+1) matrix.

Convolution

Process (Continued ...)

CASE 2 : Multiple filters to a 2D Image (grayscale image)

•We can extend above idea to use multiple filters (for detecting different features) on a 2D image

Eg: Suppose we have a 2D image and we want to look for

- Baseballs – use a filter called filter 2
- Eyeballs – use a filter called filter 1

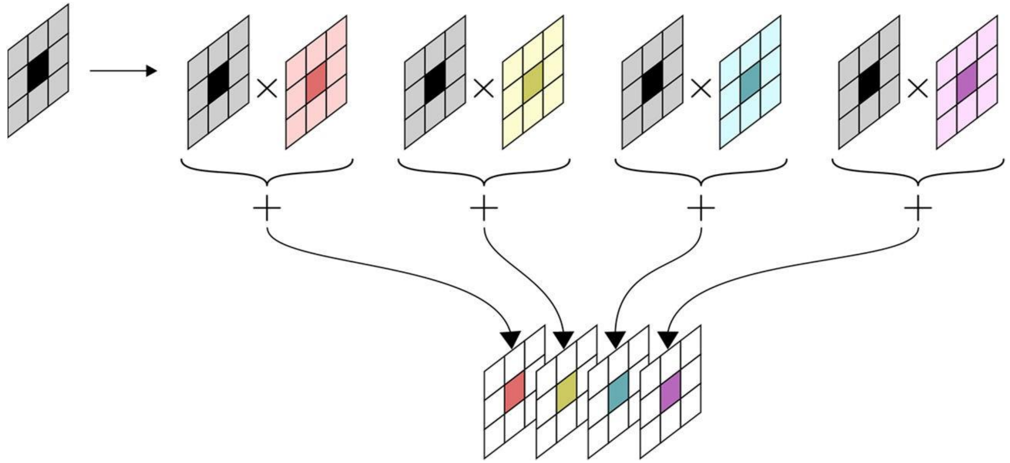
Each filter looks for its own features

- Volleyballs – use a filter called filter 3
- Meatballs – use a filter called filter 4

Convolution

Case 2 Multiple filters to a 2D Image (Continued ...)

- Run each filter over the input image, independently
- Result : four output images, one output for each filter



- If we used 7 filters, then output would be a new image with 7 channels

Convolutional
Neural Networks

Convolutions over 3D images & Multi-channel inputs

Convolving with 3D images

- Let's next look at convolving with 3D images (color images) and higher dim inputs
- How do we convolve a filter with 3D images?
- Before looking at convolution, note the important point below: No of channels

  in each filter = No of channels in input
- We can have any number of filters that convolve with am multi-channel image, but each filter must obey the above rule.

Convolution Process (Continued …) Handling multi-channel inputs

CASE 3 : Handling 3D or multi-Channel Images (such as color image)

- Key thing to note is that the input has multiple channels, say, like 3 channels for a RGB colour image.
- Then, each filter must also have same number of channels, i.e. each filter must have 3 channels for convolving with a RGB color image

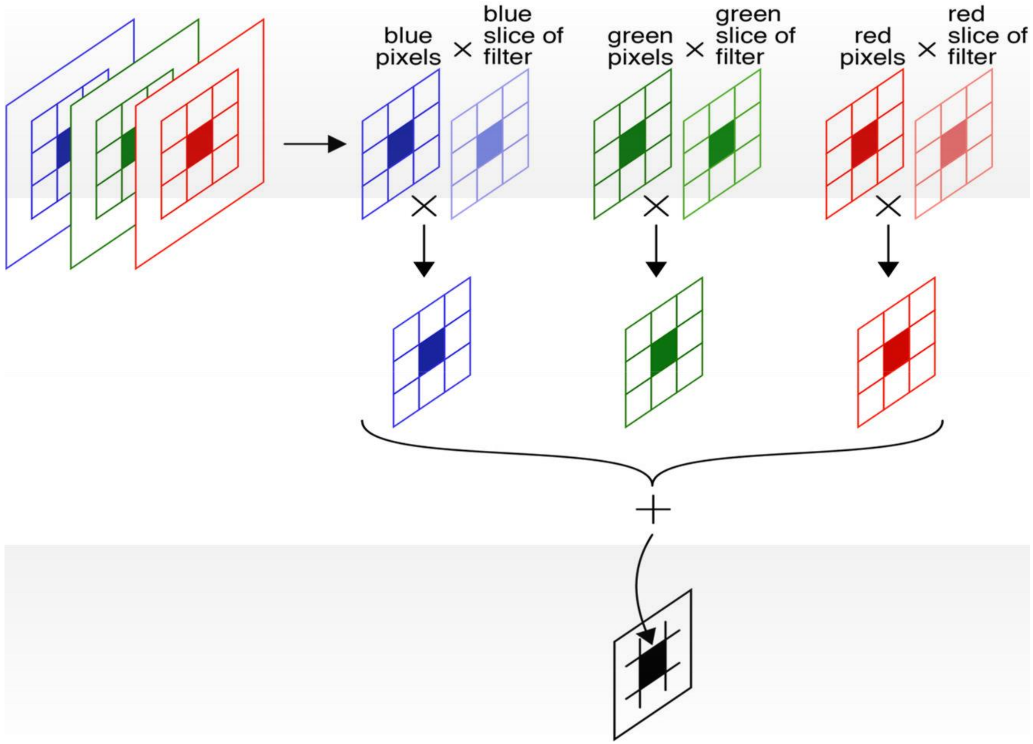


Fig : convolution with 3D images

Convolving an RGB image with a 3 by 3 by 3 kernel.

We pull out the 9 red, green, and blue pixel values for the 3 by 3 footprint, and multiply those elements with the corresponding slice of the kernel.

We can add up all the results to produce a single value.

Convolutions Over 3D Vol

How do we apply convolution on this image?

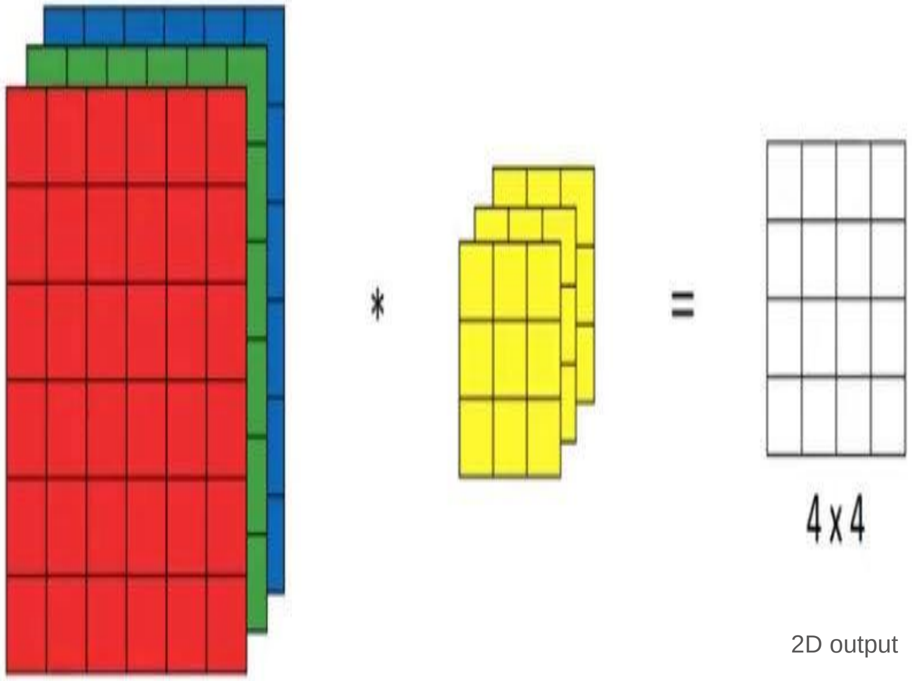
Input: 6 X 6 X 3
Filter: 3 X 3 X 3

Height x Width x Channels

Keep in mind that the number of channels in the input and filter should be same.

This will result in output of 4 X 4, single channel

(ie., only 2D output)



3D input

2D output

First element of the output = sum of the element-wise product of the first 27 values from the input (9 values from each channel) and the 27 values from the filter.

After that, we convolve over the entire image.

Convolution

Process (Continued ...)

CASE 4 : Convolving *multiple* filters with multi-channel input.

Example: Input has 6 channels

•We want to apply 4 filters of size 3 x 3

•Each of the 4 filters should have 6 channels (to match with number of input channels)

•Output of each filter -> Single channel deep

•Final output has 4 channels (see figure in next slide)

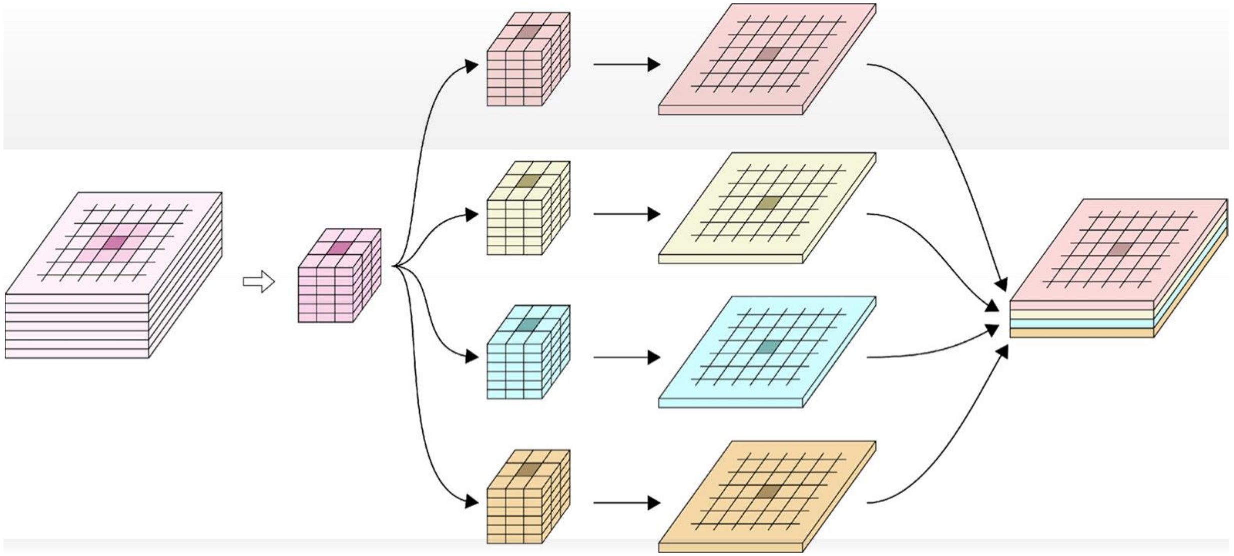


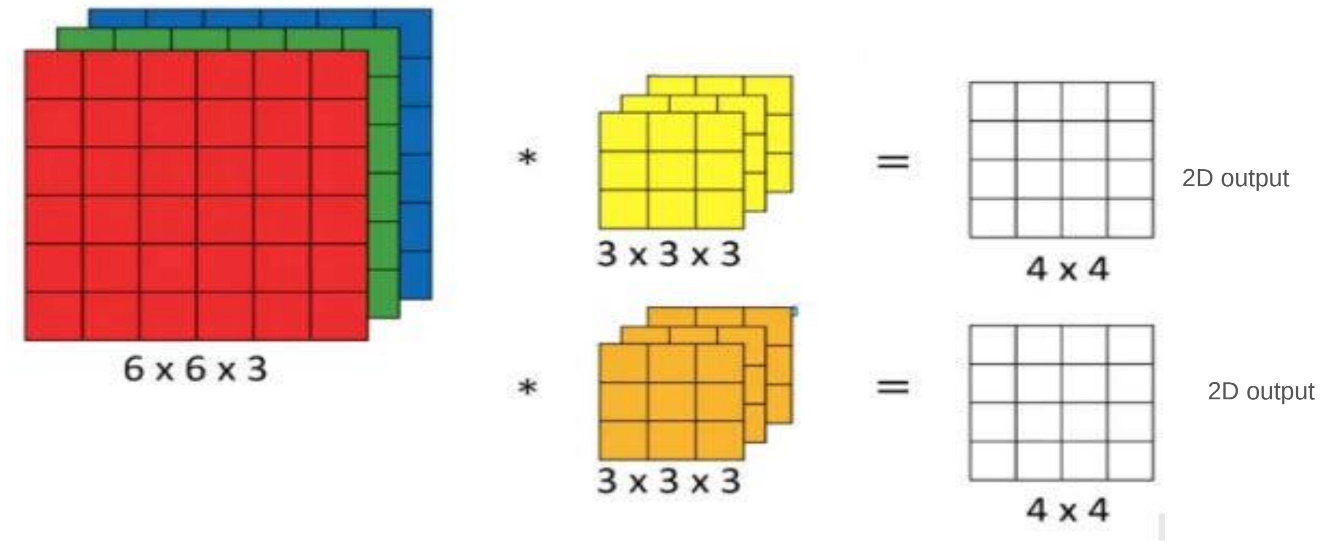
Fig : Multi-filter multi-input convolution

When we convolve filters with an input, each filter must have as many slices as the input.
Here the input is 6 channels deep, so each filter is 6 channels deep.
The 4 filters each create an output of 1 channel, so the final output has 4 channels.

Multiple Filters

Multiple filters can be used to detect multiple features or edges.

So lets learn how to convolve with multiple filters



Issues with convolution

1. Every time we apply a convolutional operation, the size of the image shrinks

2. Pixels present in the corner of the image are used only a few number of times during convolution as compared to the central pixels.

That is, convolution throws away a lot of information that are in the edges.

Padding can avoid image shrinking

To overcome these issues, we can pad the input image (before convolution) with additional pixel(s) around the image.

The padding amount *p* is the number of rows/columns we will insert at top, bottom, left and right of the original image.

Example: Let p=1 around our 6 x 6 input image. So, we add a ring around image.
This means that the input image will now be an 8 X 8 matrix (instead of a 6 X 6 matrix).
Convolving padded image with a 3 X 3 filter results in 6 X 6 matrix This is the same size as input image.

This is how padding helps !

Padding

- Near the edge of image, the filter's footprint can *'fall off'* the image side. There are no input values there.

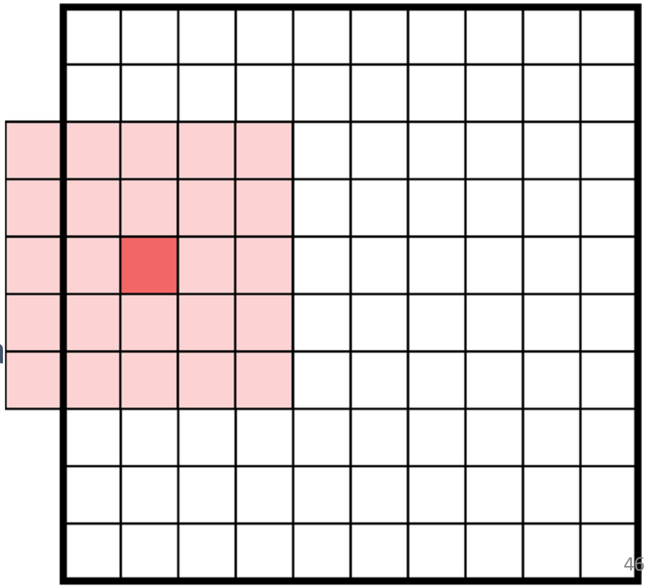
- How to convolve at these positions?

- Two choices :

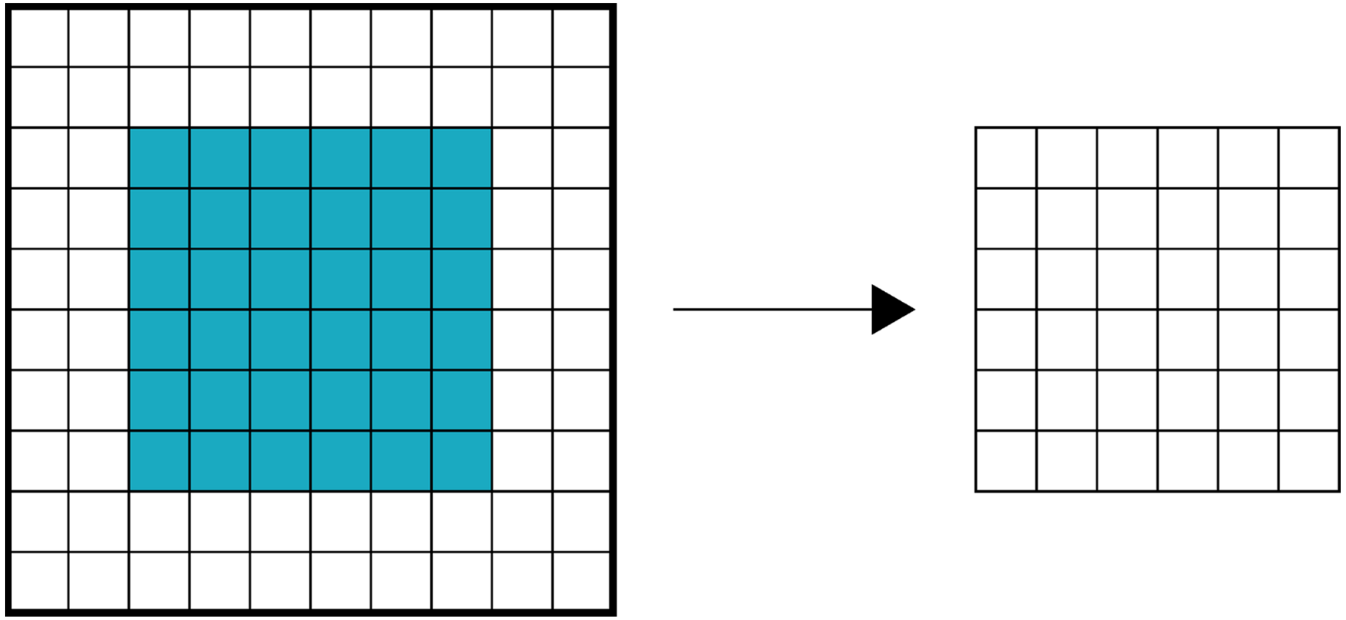
a) Disallow this case

  - Place footprint where its entirely within ima
  - This makes output smaller in size

 - Like shown on next slide

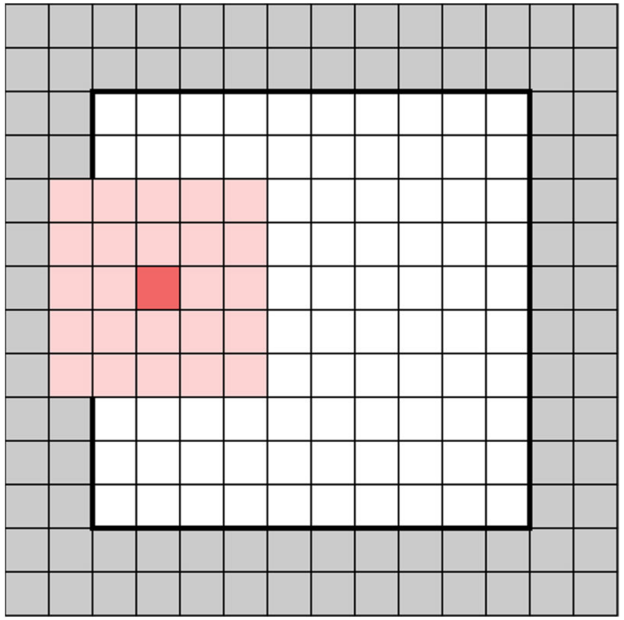


This is what happens without padding



Placing footprint where its entirely within image, makes the output smaller in size

Padding (contd.)



b) Use padding

- Add a border of “extra” pixels around the outside of image
- All these “extra” pixels have some values, typically, zeros
- This choice is called as <u>zero-padding</u>
- See adjacent Fig. for zero padding
- The grey borders are padded areas, and have zero values.
- Using padding, we can make output size equal to input size, or even larger (see later)!

Output size

Suppose the sizes are **Input:** n X n **Padding:** p
Filter size: f X f
Then, the output size is (n+2p-f+1) X (n+2p-f+1)

Example: If n = 6, f = 3, and p = 1 Then the output image be of size ?
Each side of output will be of size= n+2p-f+1 = 6+2-3+1 = 6
Hence, we maintain the size of the image (before and after convolution) by appropriate amount of padding

This is called as “same” padding

Two types of padding

There are two common choices for padding:

Valid: It means no padding. If we are using valid padding, the output will be (n-f+1) X (n-f+1)

Same: Here, we apply padding so that the output size is the same as the input size, i.e.,

$n+2p-f+1 = n$, so, $p = (f-1)/2$ for “same padding”

Stride

- When we sweep a filter over an image, we can move or stride the filter more than one image pixel to the right or down .
- That is, we could skip over pixels horizontally or vertically,or both
  - Input scanning with filters can skip over pixels.
- What’s the use of striding?
- Striding is a fast way to reduce size (or resolution) of an image, in order to speed up later blocks in network.

Strided Convolutions

Stride helps to reduce the size of the image, a particularly useful feature.

Example: Suppose we choose a stride of 2.

So, while convoluting through the image, we will take two steps – both in the horizontal and vertical directions,separately.

Stride (contd.)

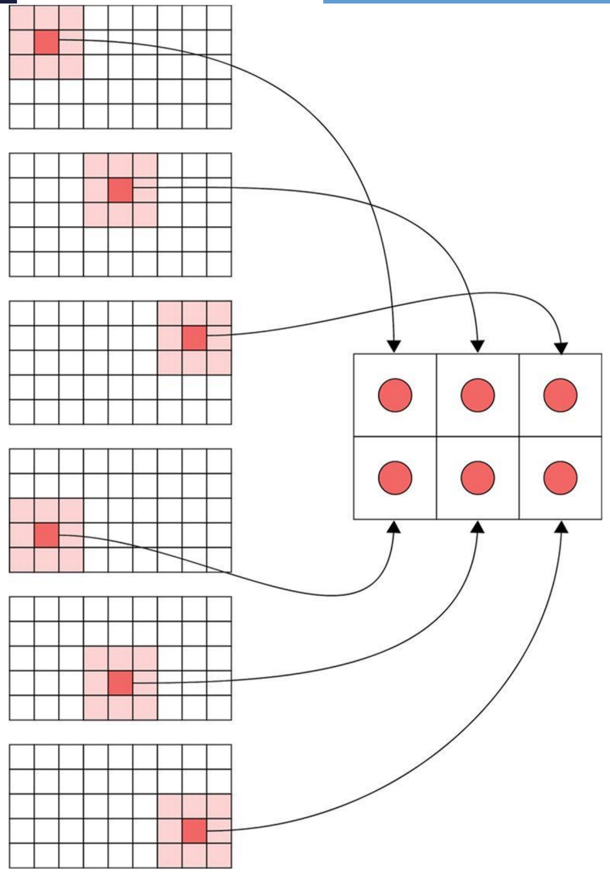
Problem :

○For a 5 by 9 input image, if we use a stride = 3 horizontally, and stride = 2 vertically, then what's the output going to look like?

Answer:

- Since stride = 3 horizontally, we move the filter to the right by 3 pixels on each horizontal step, and then move down by 2 lines on each vertical step.
  ○Output pixels are still assembled as before.
  ○Result: new (output) image is ⅓th size of original image horizontally, and ½ size of original image vertically.

- Result in adjacent figure



Transposed Convolution

- Let's look at a technique to make output size *larger* than input
- These are called as "upsampling" techniques
- If a convolution step does "upsampling", i.e, it makes output size larger than input, then it is called as "transposed convolution".
- "Upsampling" using fractional striding (or transposed convolution) increases image resolution, eg. from 100x100 to 300x300 pixels.

- <u>Example</u> (to make output larger than input image)
  - All we have to do is to pad (or surround) the input with enough number of rings of zeros (depending on filter size, etc.)

  - An instance on the next slide!

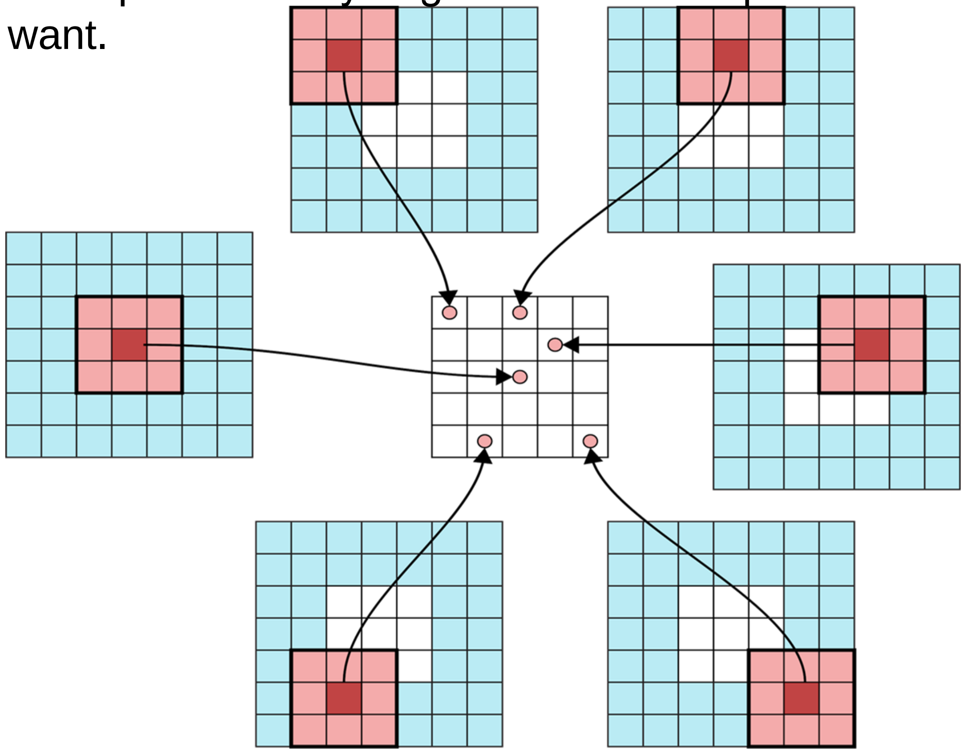
Transposed convolution (contd.)

Input image = 3 X 3

Filter size = 3 X 3 Required output size = 5 X 5

Soln: Pad image with 2 rings of zeros.

We can make output even larger (with more rings of zeros), but that'll produce only rings of zeros in output - which we rarely want.

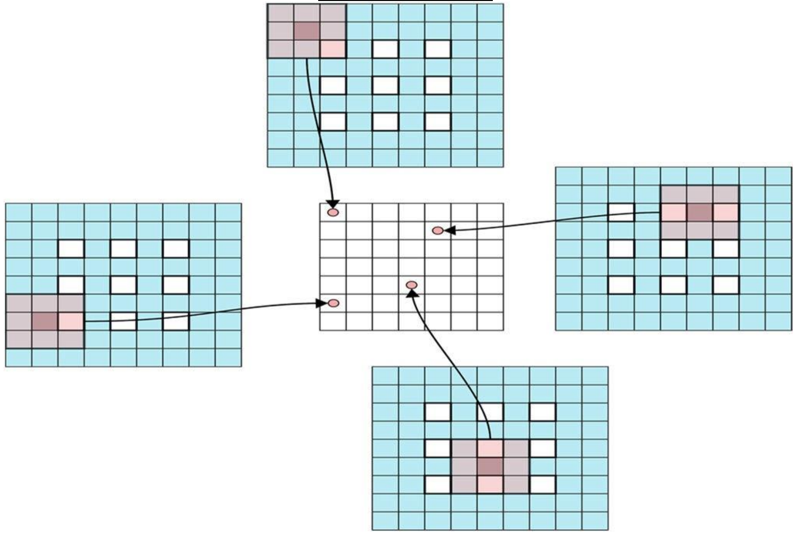


Note: Transposed convolution can also be described as *fractional striding* (the above example is a fractional stride of ⅓.

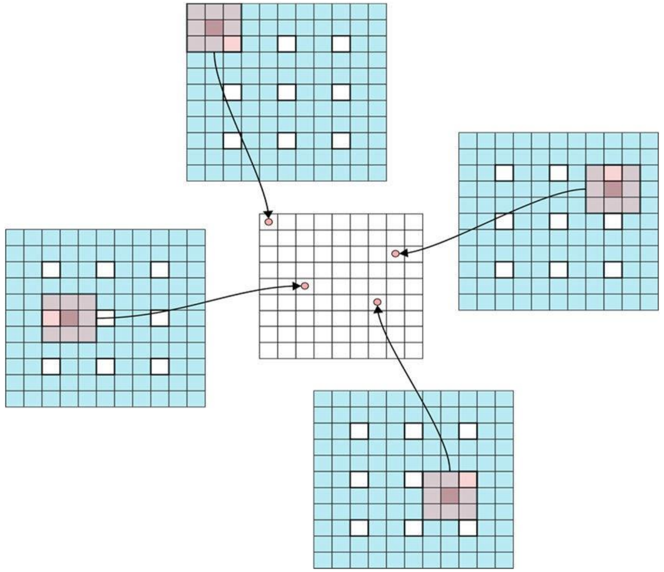
Dilated Convolution

- Another way to get a larger output size is to spread out input images by inserting padding - both around and between input elements.

- This is called Dilated Convolution.



Input 3 X 3 ⇒ Output 7 X 7



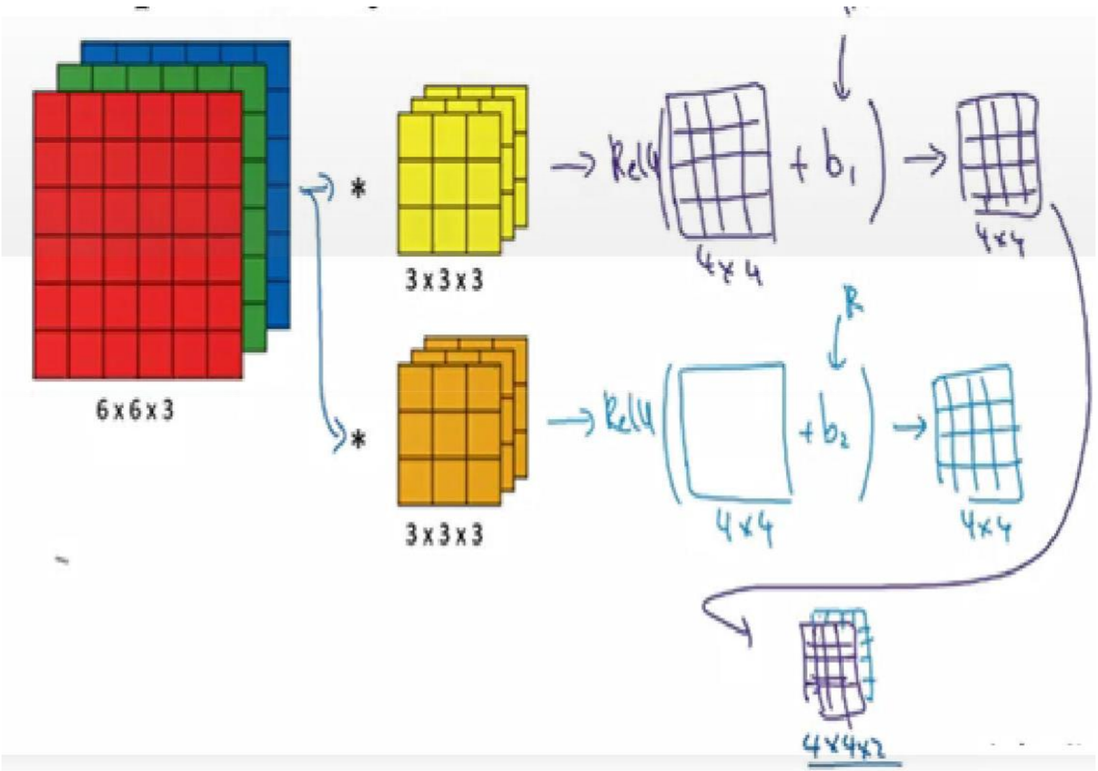
Input 3 X 3 ⇒ Output 9 X 9

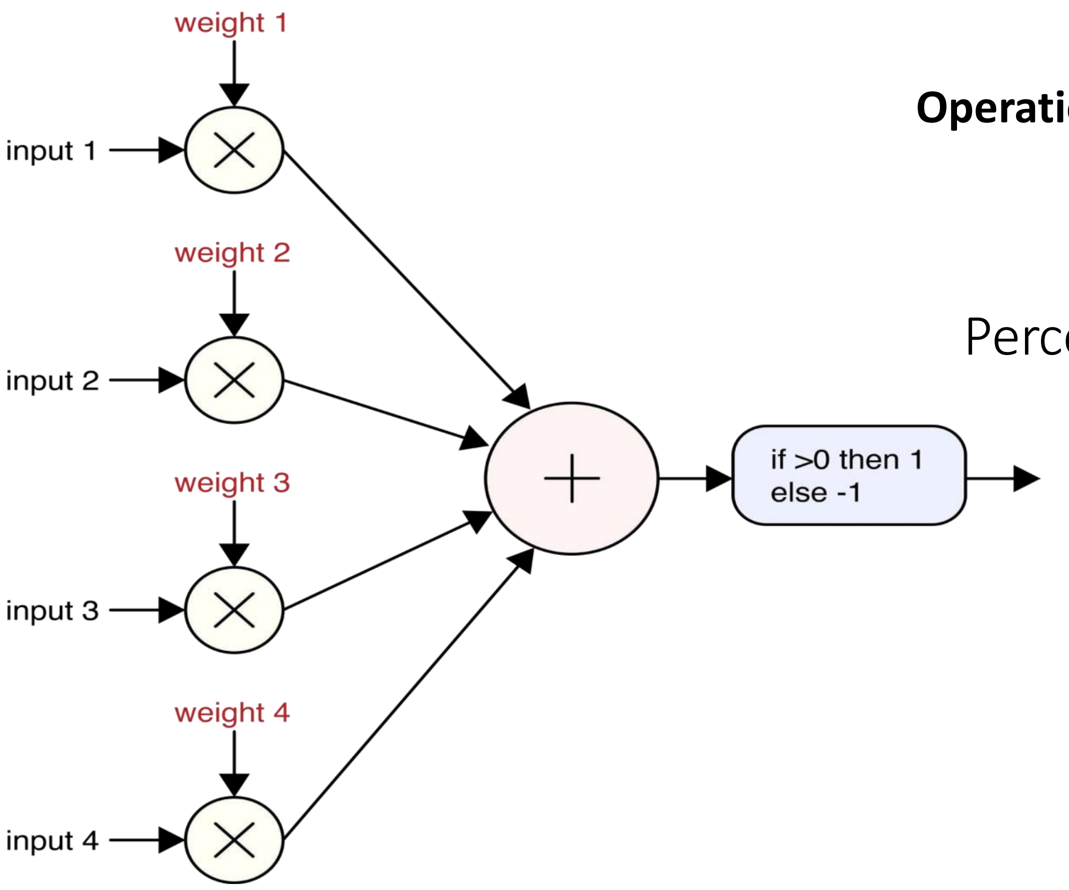
What happens in One Layer of a ConvNet

After getting an output by convolving over entire image using a filter, do

- Add a bias term to those outputs
- Apply an activation function to generate activations.

This is what happens in one layer of a convolutional network.





Operations

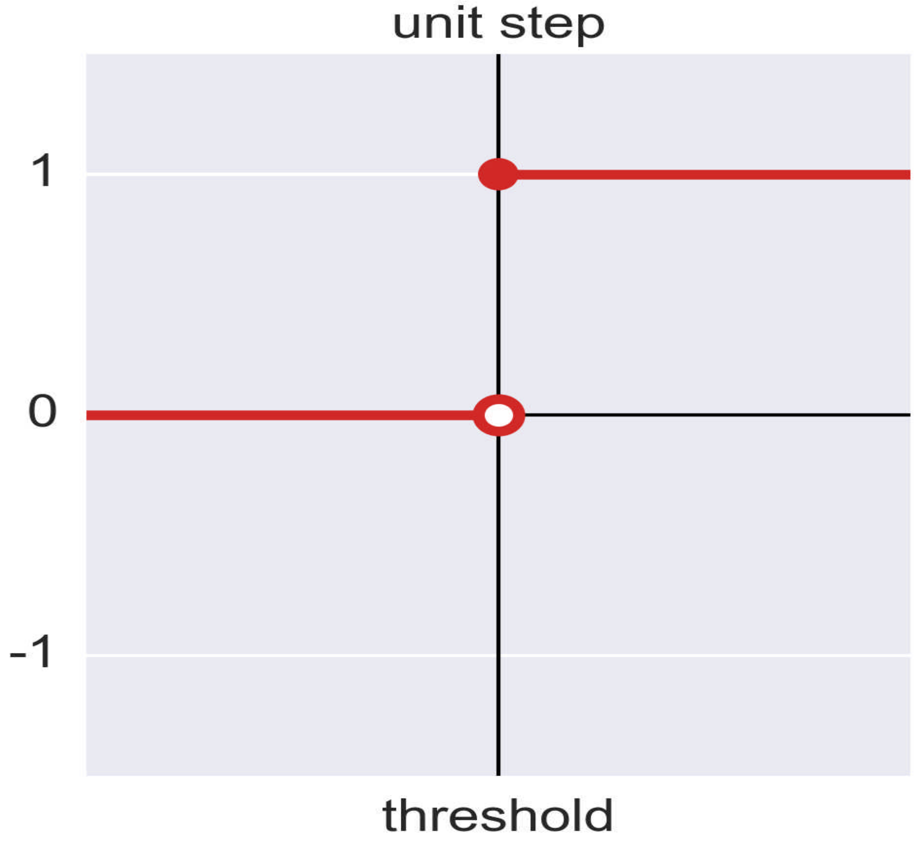
Perceptron mimics the functional behaviour of neuron

A schematic view of a perceptron. Each input is a single number, and it's multiplied by a corresponding real number called its weight.

The results are all added together, and then tested against a threshold.

If results are positive, perceptron outputs +1, else −1.

Activation Function



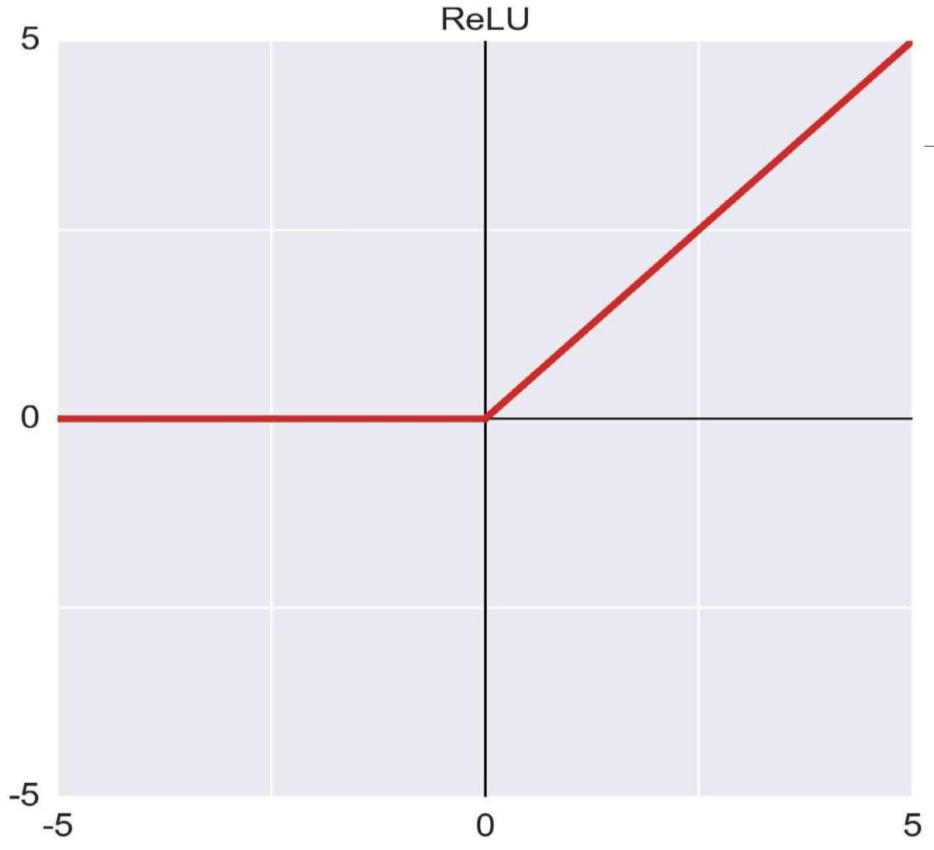
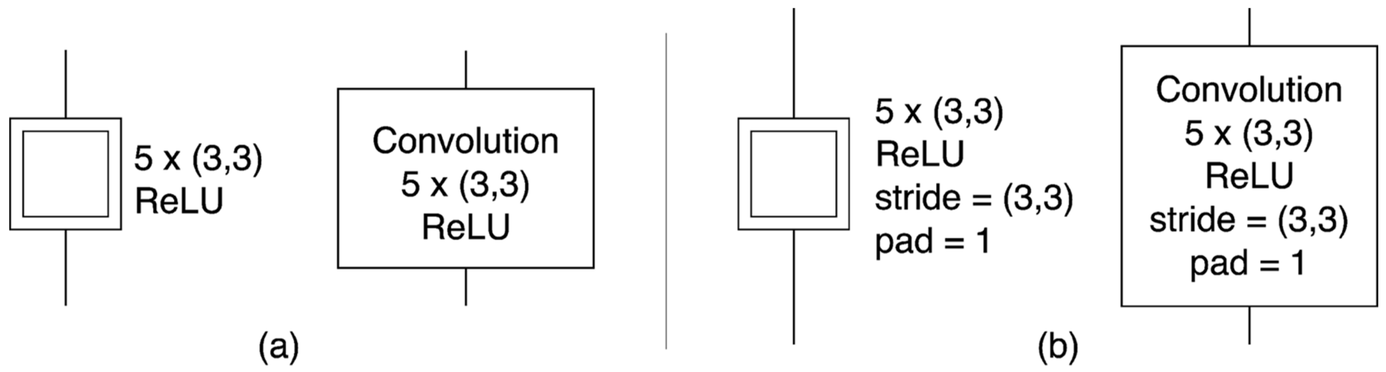


Diagram of a convolution layer



Means we are using 5 filters, each 3 by 3

Activation fn is ReLU

A convolution layer in schematic form box-and-label-form

Pooling Layers

- A pooling layer lets us to change the size of image (or data) flowing through the neural network

- Often used with image input, to reduce the image size so as to speed up the processing.

- Also introduces some kind of "robustness" in processing Example
- Input image of size 512 x 512 = Lots of pixels !

To reduce the size of the image, use pooling after the convolution layer Pooling is

explained in the following figure

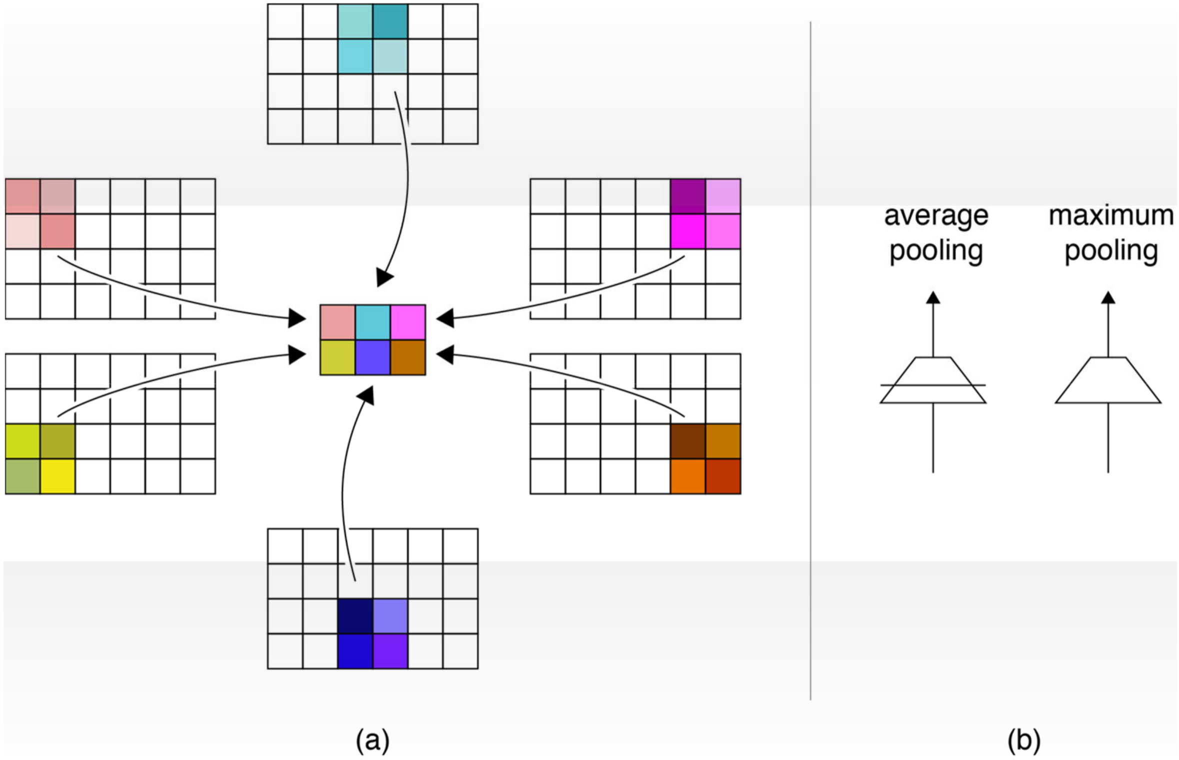


Fig : Pooling process – two types (average and max pooling)

Pooling process (contd.)

See the pooling process in previous slide.

(a) When we apply pooling to a 2D image, we gather small blocks (often squares) and use some version of their data (usually either the average value or the largest one) as the value we place into a new, smaller image.

(b) Our schematic symbols for pooling. The symbol suggests a reduction in the length of the side of an input.

(a) The two versions distinguish average and maximum pooling.

Pooling layers

Pooling layers are generally used for

- ❑ Reducing the size of the inputs and hence speeding up computations.

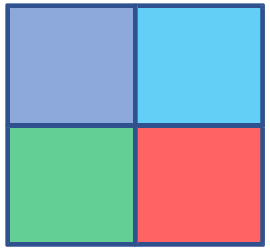
- ❑ Giving some kind of robustness to feature detection

Max pooling

Idea: Max pooling says that if feature is detected anywhere in this block, then keep a high number Pooling block of size 2 X 2

Input Image of size 4 X 4

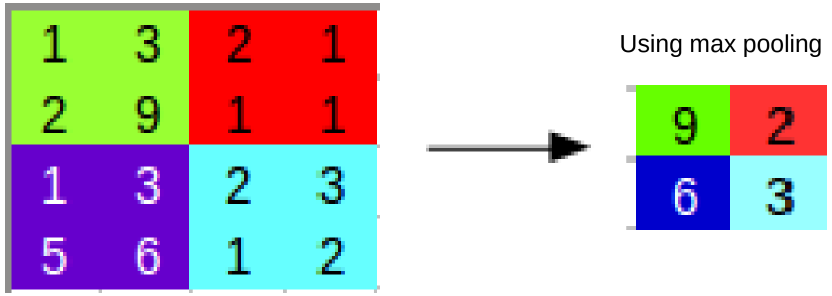
1	3	2	1
2	9	1	1
1	3	2	3
5	6	1	2



Pooling block is size 2X2.

Stride s = 2.

For every consecutive 2 X 2 block, we take the max number. We get below result.



Average pooling

Apart from max pooling, we can also apply average pooling.
Here, instead of taking the max of the numbers, we take their average.

1	3	2	1
2	9	1	1
1	4	2	3
5	6	1	2

$\longrightarrow$

Using Average pooling

3.75	1.25
4	2

Creating a Network of Layers

- ❑ Various types of layer in a convolutional network: Convolution, Pooling, Fully connected

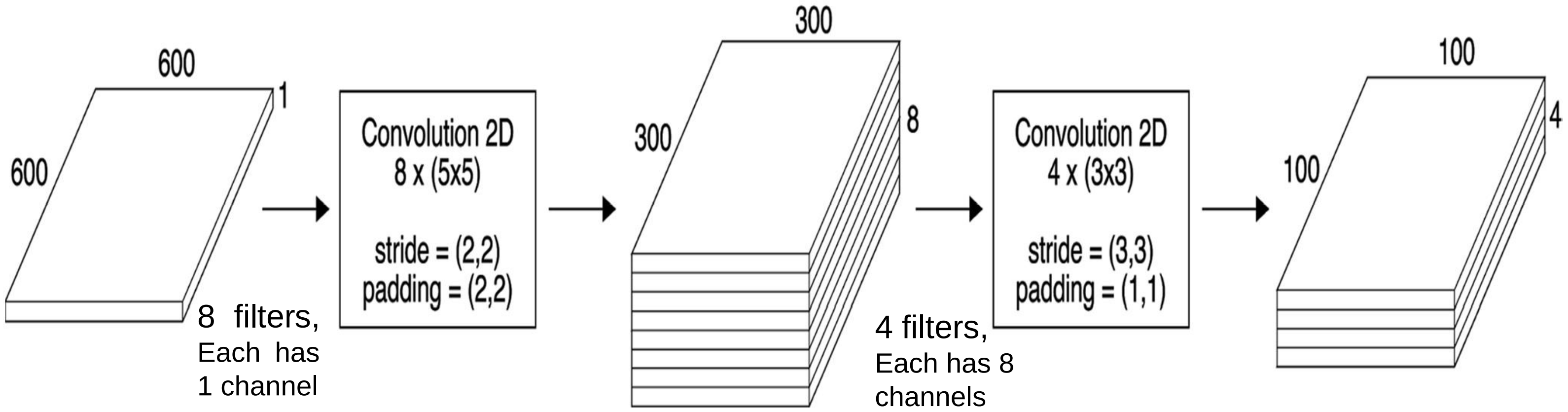
- ❑ There are also a number of hyperparameters that we can tweak while building a convolutional network.

- ❑ These include the number of filters, size of filters, stride to be used, padding, etc.

- ❑ **Downsampling using striding during convolution is often better than convolution + pooling.**

- ❑ Notice that as we go deeper into the network,
  - ❑ Size of the image shrinks
  - ❑ Number of channels increases.

Example - Creating a series of convolutions



Creating a series of convolutions

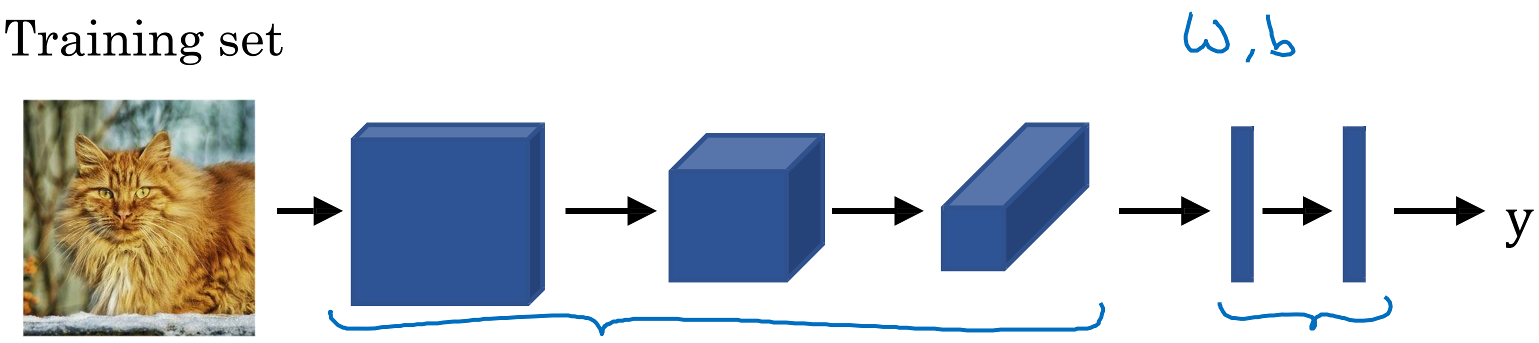
Input: 600 by 600 image, with just 1 channel (greyscale)

First convolution: 8 filters (each has one channel, as input has one channel) of 5 X 5 size. Image is padded by two rings of zeros all around before convolution. Take stride=2 in vertical and horizontal directions during convolutions with each filter. Output is 300 by 300 by 8 (8 channels in output as we used 8 filters)

Second convolution:. Pad the image box from first layer with one ring of zeros (as padding =1). Use 4 filters (each filter has 8 channels) of size 3 by 3. Do strided convolution using stride = 3 in both directions. Resulting output is 100 by 100 by 4. Each stage uses a lower-resolution version of the previous stage, so it can work with larger collections of features without requiring larger filters.

Training CNN

Training set



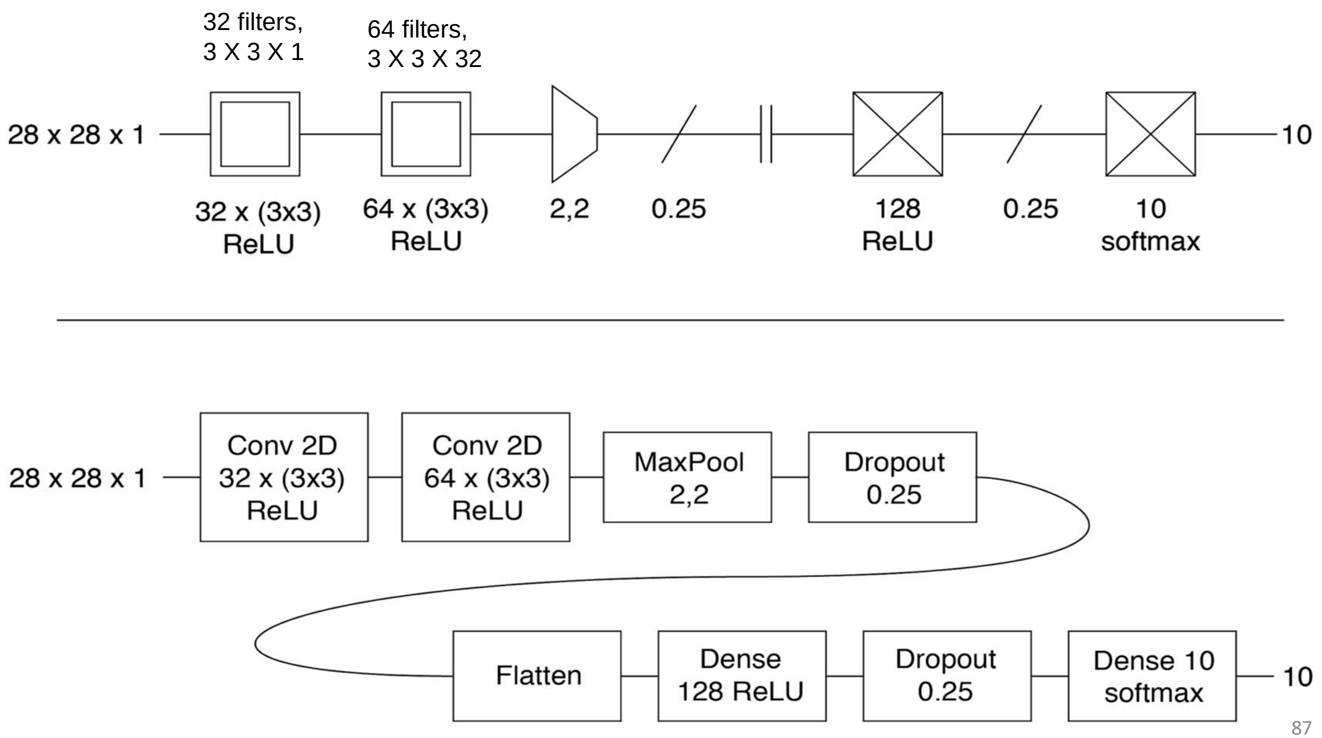
Hyperparameters of a Convolution layer

- Parameters of a convolution layer
  - How many filters?
  - What’s there footprint?
  - Striding? How much striding?
  - What activation function?
  - Padding? How much padding?
- Output of a convolution layer
  - Output has one slice (or channel) for each filter (filter is designed to get a feature from input)
  - Output of a convolution layer is a feature map

Initializing filter weights

- How to initialize filter weights
  - If two filters have identical weights, then they are symmetrical.
  - Symmetrical filters do the same job, so waste of resources.
  - Avoid forming symmetrical filters ⇒ don’t initialize two filters with same values.
  - That is, go for symmetry breaking by initializing every filter with different values.
  - Different approaches :
    1. Use small random numbers, eg [-0.01, 0.01] for initializing each filter.

Image Classifier from Keras ML library



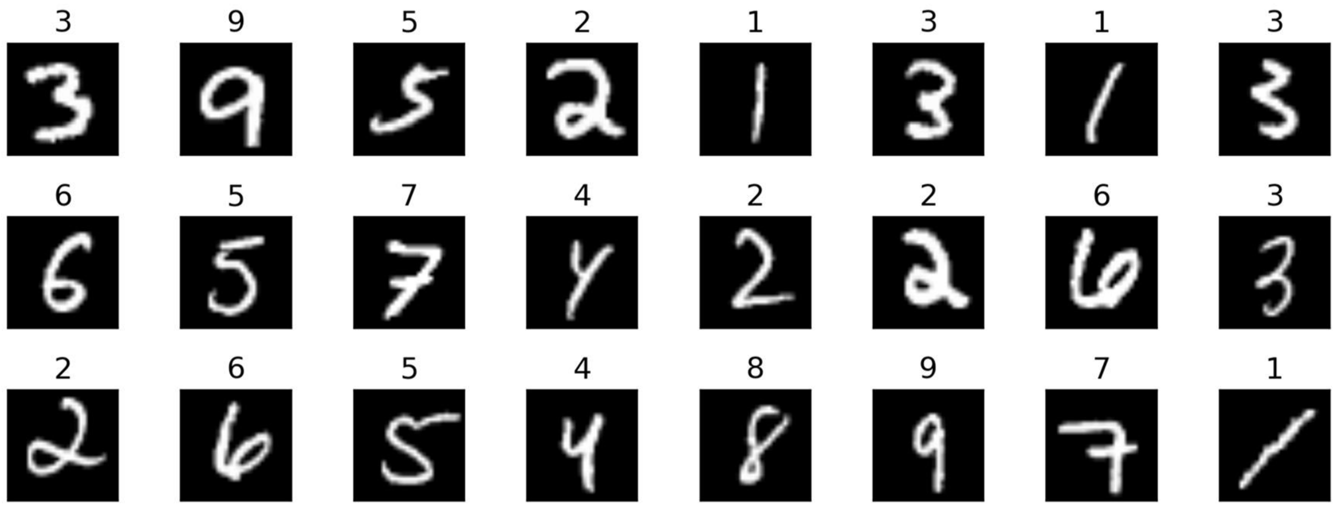
Example

- Input to convnet: MNIST image, resolution 28 X 28 X 1
- First convolutional layer:
  - 32 filters, size 3 X 3, each filter's output through ReLU activation
  - Each filter dimension is 3 X 3 X 1 (only 1 channel, as input is greyscale image of 1 channel)
  - Each filter initialized by Keras library using Glorot initialization
  - Stride s = 1 (default), padding p = 0 (default)
  - So, as p = 0, output image will not have outermost ring of pixels of input
  - So, output size = 26 X 26 X 32 (32 channels in output, as 32 filters are used)
- Second convolutional layer:
  - 64 filters, each of size 3 X 3
  - As input to this layer has 32 channels, so each of 64 filters will have 32 channels. Hence, each filter is created with size 3 X 3 X 32
  - Stride and padding are default (s = 1, p = 0)
  - As no padding, we lose another ring outside of input image
  - So, output is of size 24 X 24 X 64 (64 signifies the number of filters used in this layer).

- Max pooling layer:
    - Block size 2 X 2
    - For every non-overlap 2 X 2 block in input, max pooling layer outputs just one value containing maximum value in this block
    - So, output of this max pooling layer is 12 X 12 X 64 (pooling doesn't change the number of channels)
- Dropout layer: (here, dropout =0.25, i.e., 25%)
    - No operations, only that some 25% of neurons in the previous computation layer (i.e, the second convolutional layer) will be temporarily disabled
    - This should help avoid overfitting
    - Output of dropout layer is of same size as its input, i.e, 12 X 12 X 64
- Flattening layer:
    - Flattens the input to this layer (12 X 12 X 64) to a big list of numbers (12 X 12 X 64 = 9216 numbers)

- First fully connected (FC) layer:
    - Of 128 neurons, gives 128 outputs
    - Dropout layer:
    - With 25 % dropout values - temporarily disconnects 25 % of neurons in first FC layer at the start of epoch
    - Second fully connected (FC) layer:
    - Of 10 neurons, gives 10 outputs
    - Softmax step:
    - 10 outputs from previous (second) FC layer are converted to probabilities
    - Final output:
    - Convnet's prediction of probability that input image is the corresponding digit (0-9)

- <u>Results of Example Convnet for MNIST Digit classification Problem</u>:
    - Train for 12 epochs on standard MNIST data
    - Accuracy is 99% on both training and validation datasets - avoided overfitting
    - Predictions on MNIST <u>Validation</u> set are shown below.
    - Perfect Job!



Convolutional Neural Networks

Some Classic Networks

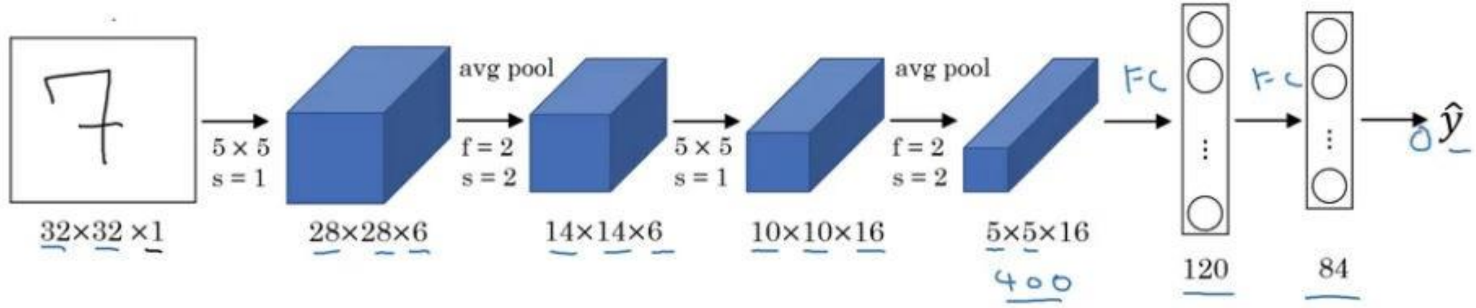
Classic Networks

In this section, we will look at the following classic networks:

1. LeNet-5
2. AlexNet
3. VGG16

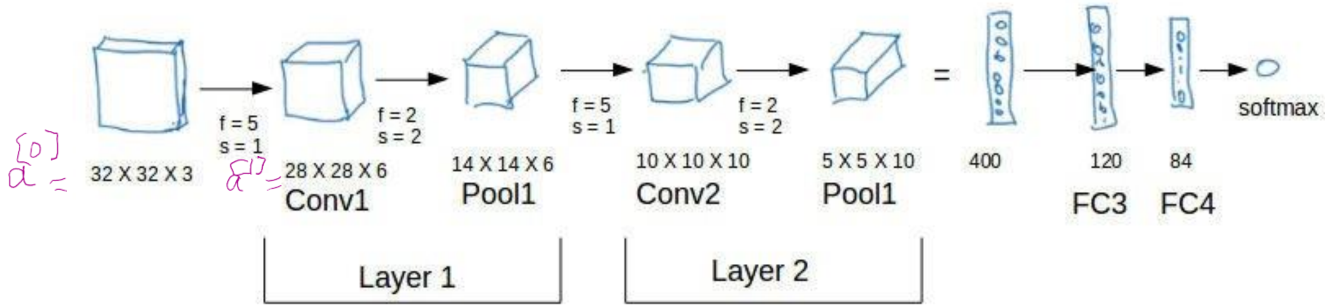
LeNet-5 (LeCun et al., 1998

- ❑Goal: Identify handwritten digits in a 32 x 32 x1 gray image.
- ❑Published in 1998. It was used for HW digits recognition in US banks.
- ❑It takes a 32 X 32 X 1 (grayscale) image as input.
- ❑Image is processed through a combination of convolution and pooling layers, then fully connected layers, and lastly, classified into corresponding classes.
- ❑The Layers flow is Input -> Conv -> Pool -> Conv -> Pool -> FC -> FC -> Output
- ❑Total number of parameters in LeNet-5 are 60k
- ❑Activation functions: Sigmoid/tanh (and ReLu in modern versions)



LeNet-5

LeNet-5 has a combination of convolution and pooling layers at the beginning, a few fully connected layers at the end, and finally a softmax classifier to classify the input into various categories.

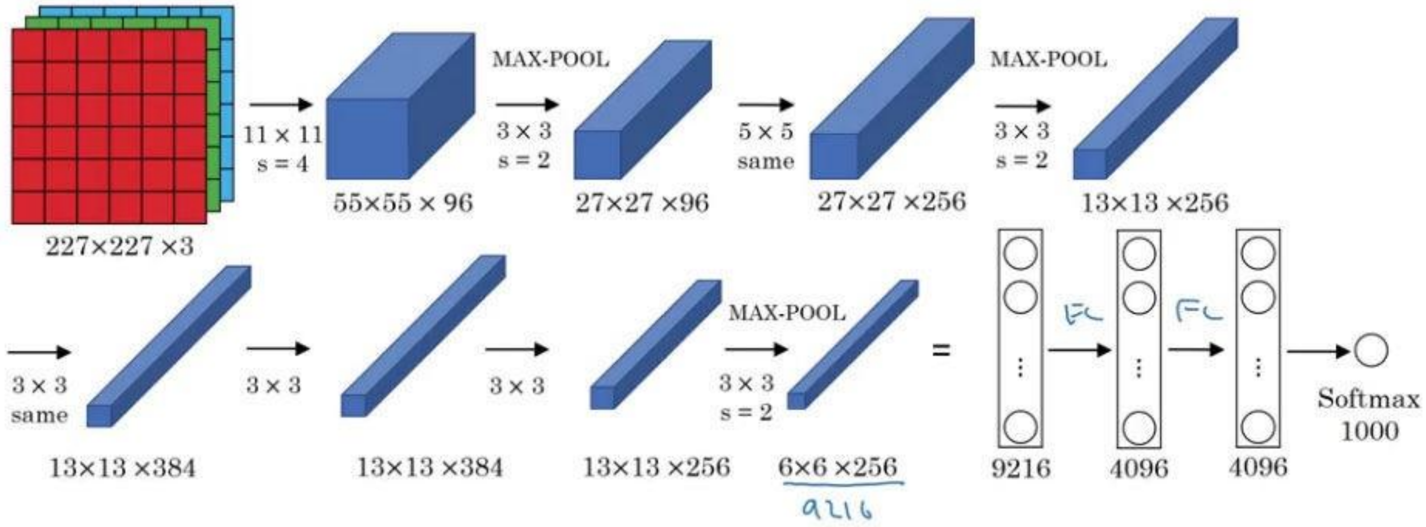


The height and width of the input shrinks as we go deeper into the network (from 32 X 32 to 5 X 5) and the number of channels increases (from 3 to 10).

There are a lot of hyperparameters in this network which we have to specify as well.

AlexNet

- Named after Alex Krizhevesky. (Krizhevesky et al. 2012)
- This paper convinced computer vision researchers that deep learning is so important
- Goal: ImageNet challenge, which classifies images into 1000 classes
- Similar to LeNet-5, but has more convolution and pooling layers
- Parameters: 60 million (compare with 60K of LeNet-5)
- Activation function: ReLu



Third Example Convnet
VGG16 (Visual Geometry Group, 16 Computational Layers)

- A modification of AlexNet (Simonyan& Zisserman 2015)
- Much bigger and more powerful convnet; can identify 1000 objects in color photos
- Training data set has 1.2 million images, each manually labelled
- Authors (Simon Yan et.al, 2014) gave all weights and how data was preprocessed
- Focus is on having only these blocks:
    - Convolution layers that have 3 X 3 filters with a stride of 1 (and same padding).
    - Max pool layer is used after each convolution layer with f= 2 and s= 2.
- Parameters: 138 million

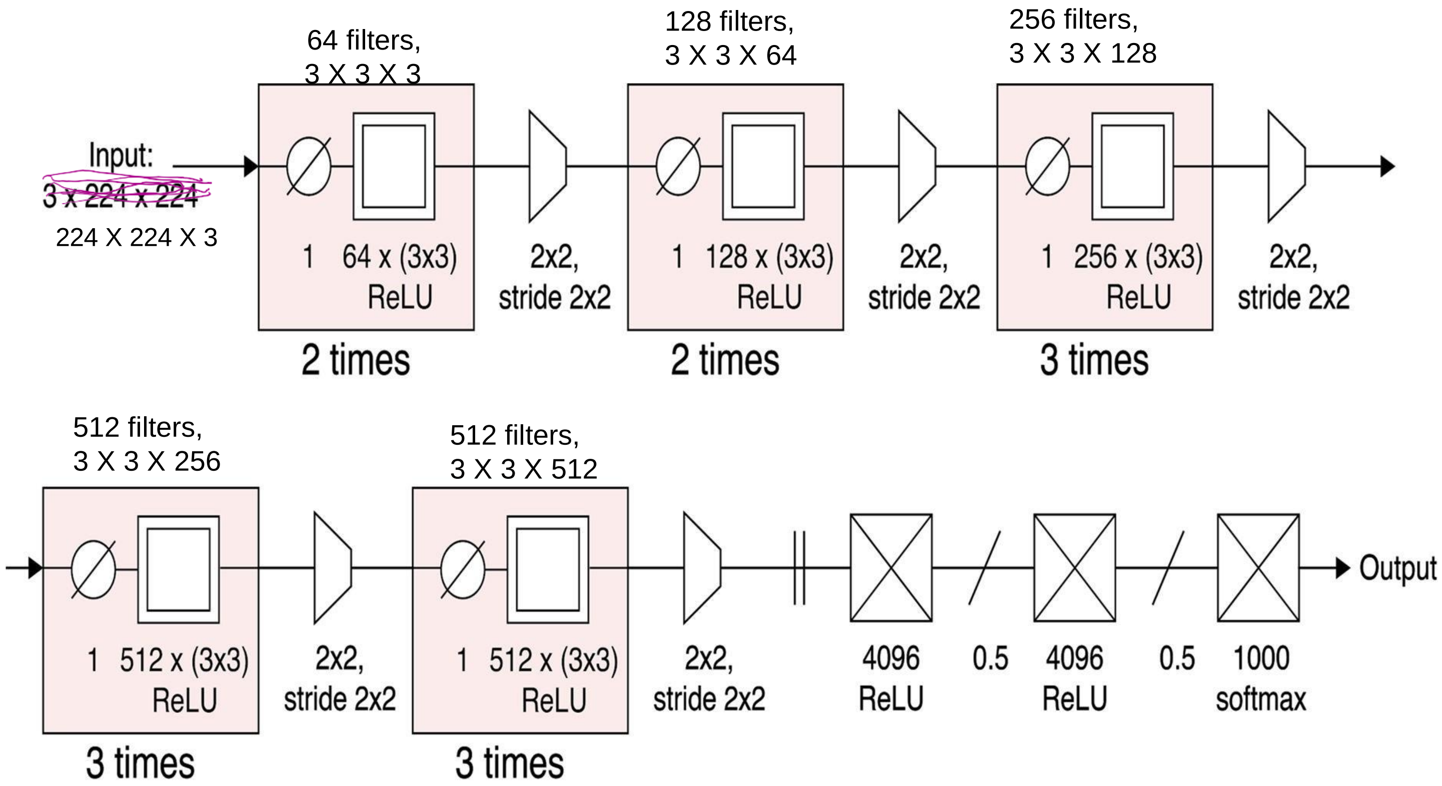
VGG16 architecture

- Overview of Architecture:

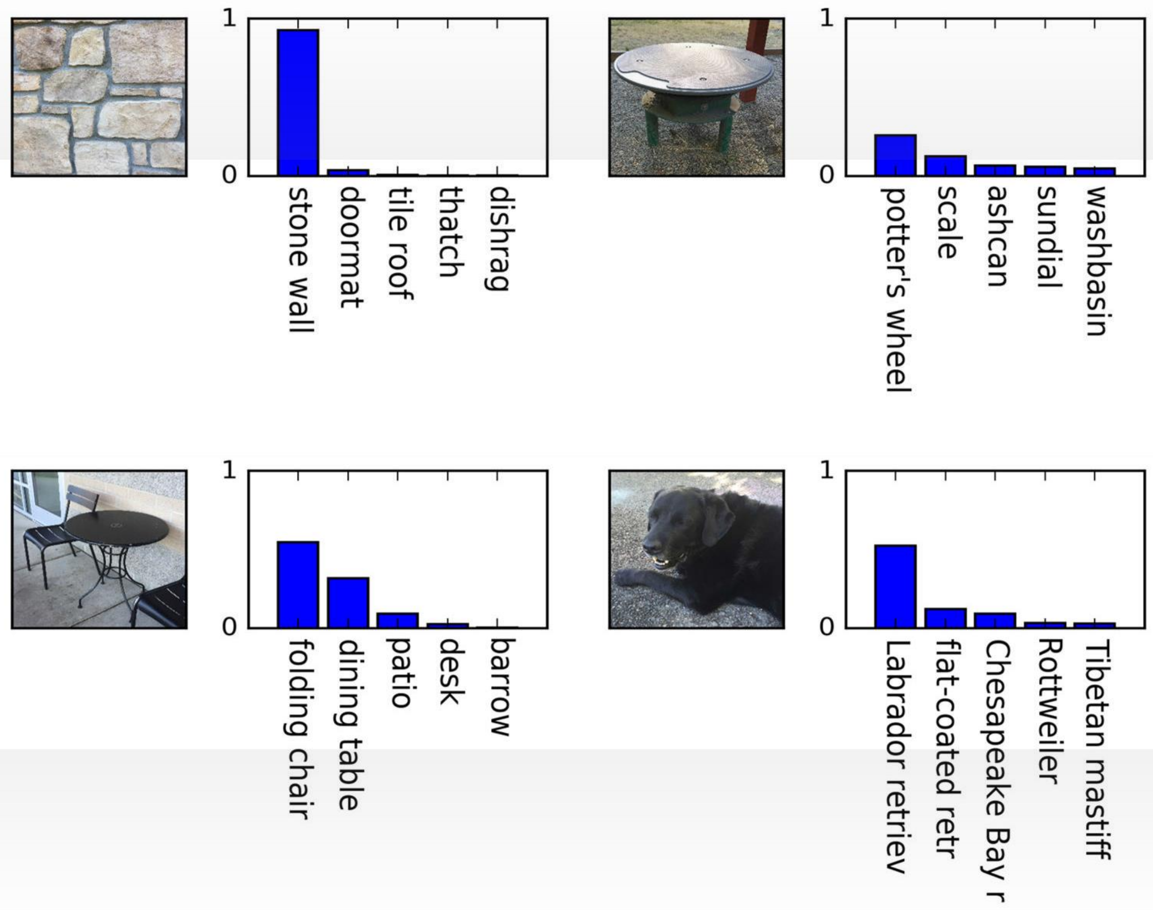
- VGG16 has several convolutional layers (with zero padding), max pooling layers 2 X 2 (p=2), striding of 2 (s= 2), flattening layer, dropout layers, and FC layers towards the very end of the net.

- The convolutional layers appear in sequence of 2 or 3 of same repeated layers.

VGG16 Architecture



Results of VGG16



Evaluation Metrics

Table: A sample test set with model predictions.

ID	Target	Pred.	Outcome	ID	Target	Pred.	Outcome
1	spam	ham	FN	11	ham	ham	TN
2	spam	ham	FN	12	spam	ham	FN
3	ham	ham	TN	13	ham	ham	TN
4	spam	spam	TP	14	ham	ham	TN
5	ham	ham	TN	15	ham	ham	TN
6	spam	spam	TP	16	ham	ham	TN
7	ham	ham	TN	17	ham	spam	FP
8	spam	spam	TP	18	spam	spam	TP
9	spam	spam	TP	19	ham	ham	TN
10	spam	spam	TP	20	ham	spam	FP

Binary Prediction

For binary prediction problems there are 4 possible outcomes:

- True Positive (TP)
- True Negative (TN)
- False Positive (FP)
- False Negative (FN)

The structure of a confusion matrix

		Prediction positive	Prediction negative
Target	positive	*TP*	*FN*
	negative	*FP*	*TN*

Misclassification rate

$$\text{Misclassification rate} = \frac{\text{Number incorrect predictions}}{\text{Total Predictions}}$$

$$\text{Misclassification rate} = \frac{(2+3)}{(6+9+2+3)} = 0.25$$

A confusion matrix for the set of predictions shown in Table

		Prediction	
		'spam'	*'ham'*
Target	*'spam'*	6	3
	'ham'	2	9

Misclassification accuracy

$$\text{classification accuracy} = \frac{(TP + TN)}{(TP + TN + FP + FN)}$$

$$\text{classification accuracy} = \frac{(6 + 9)}{(6 + 9 + 2 + 3)} = 0.75$$

Classification accuracy

$$\text{misclassification accuracy} = \frac{(FP + FN)}{(TP + TN + FP + FN)}$$

$$\text{misclassification accuracy} = \frac{(2 + 3)}{(6 + 9 + 2 + 3)} = 0.25$$