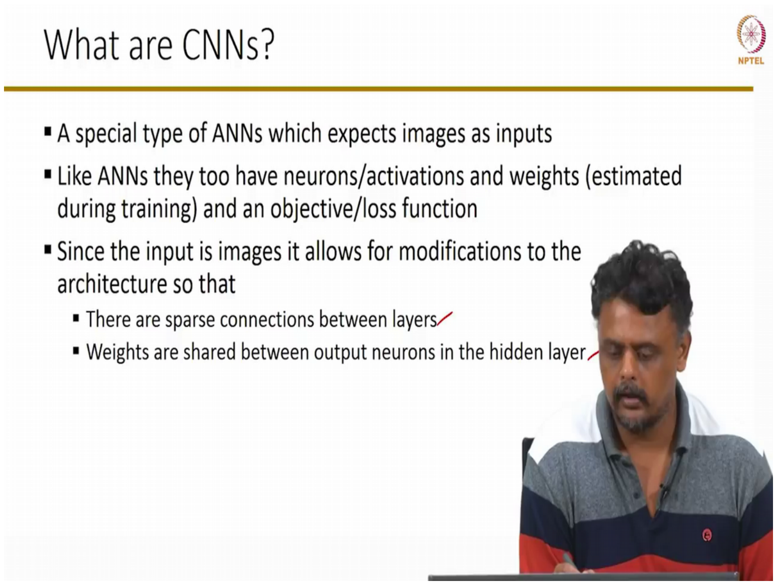
Machine Learning for Engineering and Science Application
Professor Ganapathy Krishnamurthi
Department of Engineering Design
Indian Institute of Technology Madras
Introduction to Convolution Neural Networks (CNN)

(Refer Slide Time: 00:14)

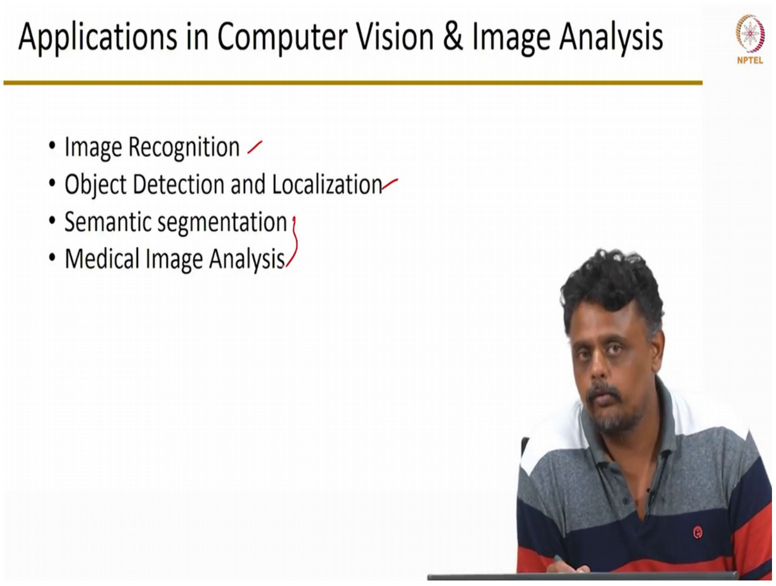


What are CNNs?

- A special type of ANNs which expects images as inputs
- Like ANNs they too have neurons/activations and weights (estimated during training) and an objective/loss function
- Since the input is images it allows for modifications to the architecture so that
  - There are sparse connections between layers
  - Weights are shared between output neurons in the hidden layer

Welcome to the series of lectures on convolution neural networks. CNN are basically a special class of artificial neural network that you see in regular neural network which expect images as input. They are designed to work on images mostly to handle computer vision problems like artificial neural networks that you have seen before the regular artificial neural network, these network also have weights, neurons and bias units and the weights in these CNN are also estimated by optimising an appropriate objective function because input to these networks or images it allows for 2 things one is fast connections we will see what those are as we progress as well as parameter sharing and because the images are used as input these 2 concepts are possible basically it is possible they have sparse connection as well as sharing of weights between the output neurons in a layer.

(Refer Slide Time: 1:32)

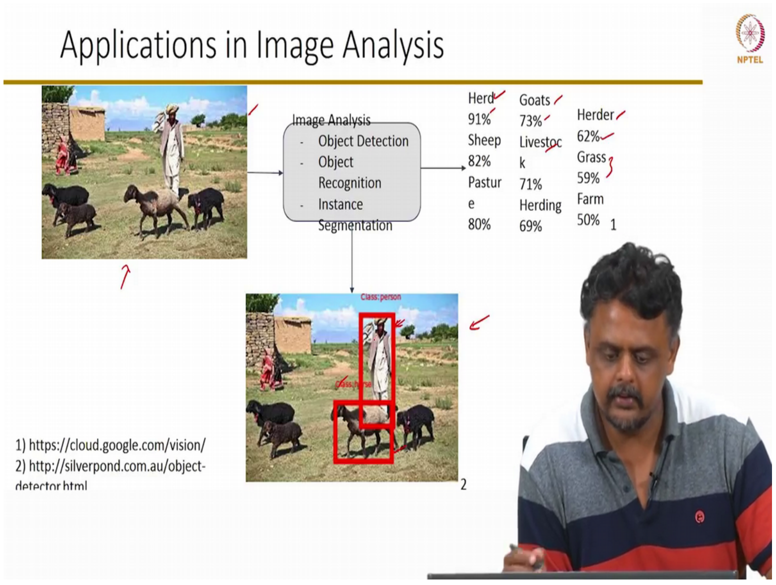


Applications in Computer Vision & Image Analysis

- Image Recognition
- Object Detection and Localization
- Semantic segmentation
- Medical Image Analysis

CNN in the recent past and of course especially since deplaning has taken off has found used applications basically in image recognition, object detection and localisation, semantic segmentation and medical image analysis these are some of the areas where CNN have shown extremely good performance on many benchmark problems and are now being tried out for many commercial applications.

(Refer Slide Time: 2:00)



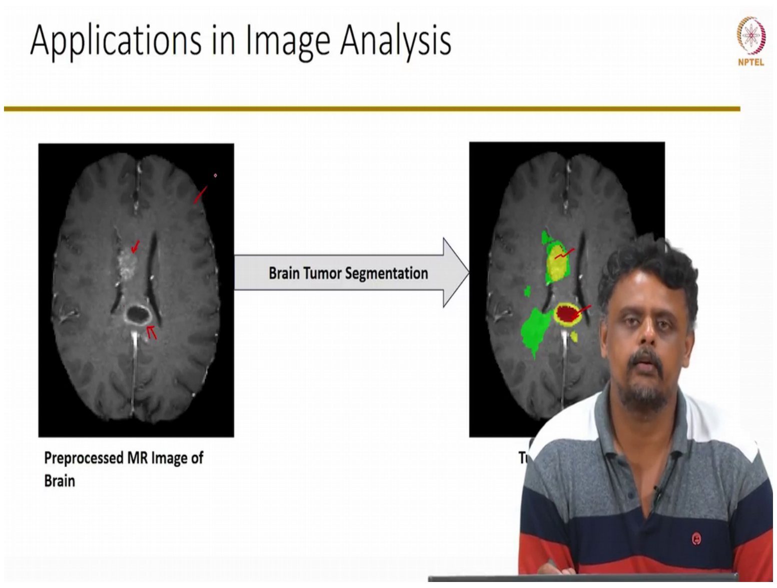
Applications in Image Analysis

1) https://cloud.google.com/vision/
2) http://silverpond.com.au/object-detector.html

So for instance let us look at this particular application, so this is an image from the wild its images showing rural scene somewhere and give this image is given as input to the Google cloud vision program it is really available and you can try it out then it automatically gives you describe the image as saying it is a herd there is a 91 percent probability that it has goads them and there is actually a herder in there. It also identifies grass, it also says livestock in this case and that this guy is the man in the pictures is actually herding, so this level of detail is possible this is (())(2:46) performance is possible with current sceneries.

Here is an output from another CNN from Silver Pond it is an object detector, so it is able to identify the man in the picture as well as the goat. However it does label it wrongly as I think a horse okay, so it is able to localise as well as identify the objects okay it is a typical application that you would do with… in a computer vision and this not too hard to think of some very… What everybody now talking about is self-driving cars you can see that if you have vision system in our self-driving car and you can and it performs well that you can identify obstacles or you can identify a pedestrian or signals or lights, zebra crossing et cetera and act accordingly. So this is a typical application in computer vision.

(Refer Slide Time: 3:44)

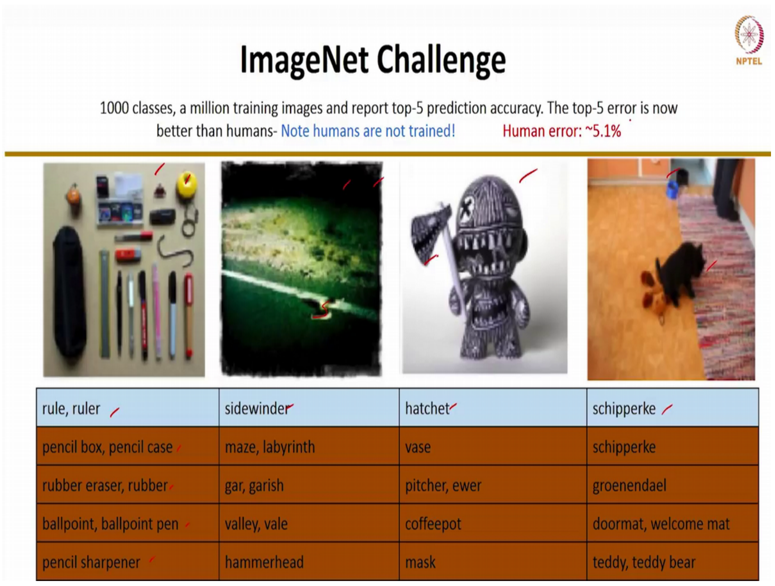


Applications in Image Analysis

CNN also finds applications in image analysis and in this case we are looking specifically at medical image analysis, so if you look at this image here on the left it is an image of a brain MR image of the brain it is been pre-processed to some extent, so you can see some abnormalities here and there okay and what you see on the right here are basically the pixel labelling task done by a CNN where it correctly identifies or fairly correctly identifies regions that appear abnormal.

The various color schemes here corresponds to different types of classes within the abnormality itself. There is a huge advantage in terms of at least in medical image analysis wherein you know this is just one slice in a brain, so a typical medical image where there are hundreds of such slices going through a particular anatomy and going through them manually and labelling each of these boxes by hand is pretty much (())(5:01) and very error prone task , so this can actually serve us huge support for a radiologist who can who look at these kind of images every day for interpreting them and diagnosing patients.

(Refer Slide Time: 5:15)



ImageNet Challenge

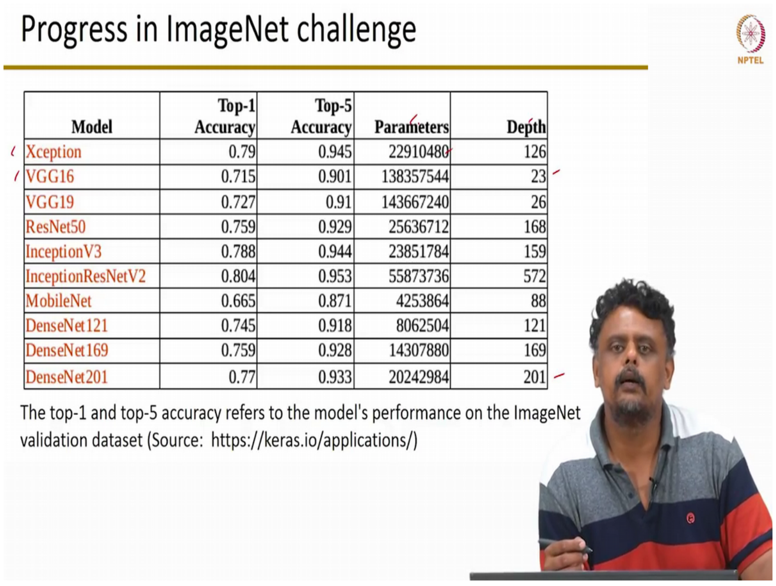
1000 classes, a million training images and report top-5 prediction accuracy. The top-5 error is now better than humans- Note humans are not trained! Human error: ~5.1%

rule, ruler	sidewinder	hatchet	schipperke
pencil box, pencil case	maze, labyrinth	vase	schipperke
rubber eraser, rubber	gar, garish	pitcher, ewer	groenendael
ballpoint, ballpoint pen	valley, vale	coffeepot	doormat, welcome mat
pencil sharpener	hammerhead	mask	teddy, teddy bear

So what we have here representative images from the image net database, now this image net challenge, visual recognition challenge is been going on for quite a few years now basically the challenge organisers make available to you millions of images drawn from the wild from Internet and labelled by expert as belonging to one of thousand categories and the challenges to create a visual machine learning on the (())(5: 42) system that when given an input image from a test set, you again containing several hundreds of thousands or millions of images. A test image is able to correctly classify it okay.

So shown here are several images, so this is actually image of sidewinder I have already marked it with red and this is actually marked as a hot shaped that is the correct level the blue shows the correct label and this is again schipperke, I actually do not know what that is but anyway, so these are some of the prediction made by typical network which are staying on the database. The challenge is that the correct clash should be among the top 5 prediction of your system okay, so over the last few years these CNNs have proven to have outperformed many other systems, AI systems trained for this task. Human error rate itself is around 5 to 6 percent and there are now large CNNs deep CNNs which out-performed this.

(Refer Slide Time: 6:58)



Progress in ImageNet challenge

Model	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	0.79	0.945	22910480	126
VGG16	0.715	0.901	138357544	23
VGG19	0.727	0.91	143667240	26
ResNet50	0.759	0.929	25636712	168
InceptionV3	0.788	0.944	23851784	159
InceptionResNetV2	0.804	0.953	55873736	572
MobileNet	0.665	0.871	4253864	88
DenseNet121	0.745	0.918	8062504	121
DenseNet169	0.759	0.928	14307880	169
DenseNet201	0.77	0.933	20242984	201

The top-1 and top-5 accuracy refers to the model's performance on the ImageNet validation dataset (Source: https://keras.io/applications/)

So for instance if you look at the accuracy of some these are the names given to a different Convolution Neural Networks by the authors who build them or the people who worked on them. Now the top 5 accuracy (())(7:08) is pretty impressive it is almost about 95 to 96 percent which is approaching human top 5 accuracy okay. We will see what this means the parameter of the network, the depth we will see later what it means but parameter of the network are basically the number of weights in the network. They range from million to 140 million okay so overtime people have started with started off with a very large number of rates and over time the network have trim down they have gone deeper but they have manage also to reduce the number of parameters in the network and also improve the top 5 accuracy.

(Refer Slide Time: 7:51)

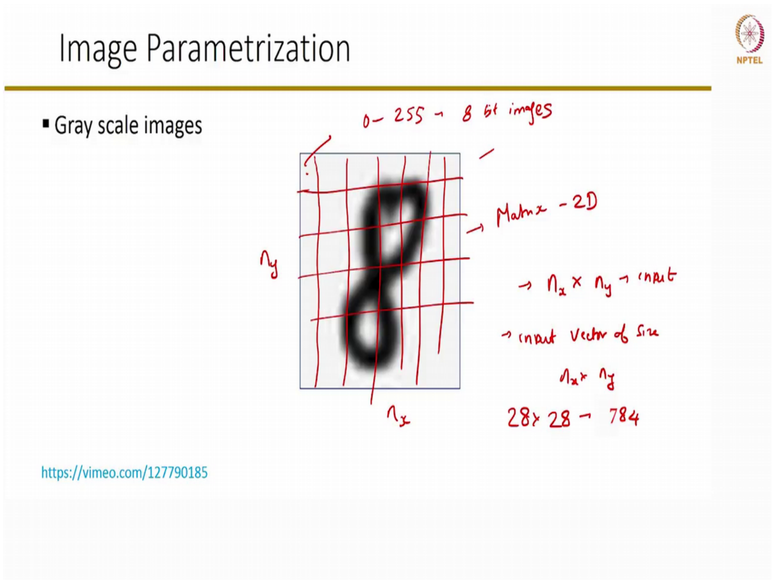


Image Parametrization

- Gray scale images

https://vimeo.com/127790185

Okay so before we move onto what CNN's do and how to build your own CNN, let us look at how images are parameterised okay, so we saw earlier that CNN's take images as input, so what does it mean so you know that for an artificial neural network the input is usually a vector, vector of values or labels or some categorical label if you want but as far as CNN's is concerned the Convolution Neural Networks is concerned the image is an input, so there are different types of images grey scale as well as RGB, so what do you mean by saying images input.

So if you take grey scale image here is a digit, image of a digit 8, so you can think of the image as being made up of… so this is actually a matrix so this is actually a matrix it is a 2D matrix and image is made up of pixels with each pixels having a particular numerical value, so this whole image is actually a 2-D matrix and if you can think of it as like a grid I can draw some crude bread like structures here and it is very close view of the image, so at each grade point there is a numerical value associated with it.

So that is the pixel value so in a typical image for instance the image is that you take with your camera the values of the pixel range from 0 to 255 they refer to as 8 bit images and the dimensionality of the input is basically the size of your 2-D matrix, so you might have x pixels on the x-axis and ny pixels on ny axis, so your image input is of size nx times ny, this is your input. If you want to think (())(9:58) in terms of artificial neural network your regular artificially neural network you have an input vector of size nx times ny.

The given example EMIS database has images of digits which are of size 28 by 28 pixels so that means it is a 4 vector of size 784, so as far as greyscale images is concerned you can think of them as a 2-D matrix dimensions nx cross ny depending upon how many pixels are there along the x or y axis and if you (())(10:48) in terms of regular ANN the size of the input is basically the total number of pixels in the image.

(Refer Slide Time: 10:59)

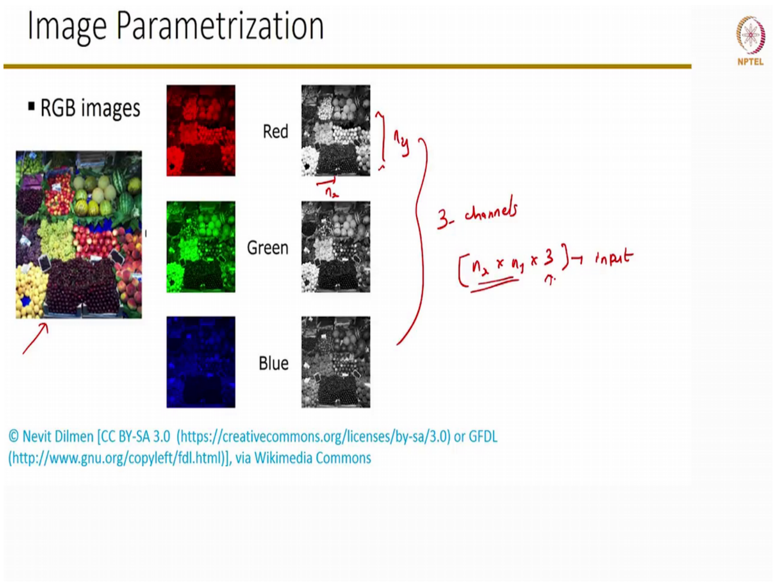


Image Parametrization

- RGB images

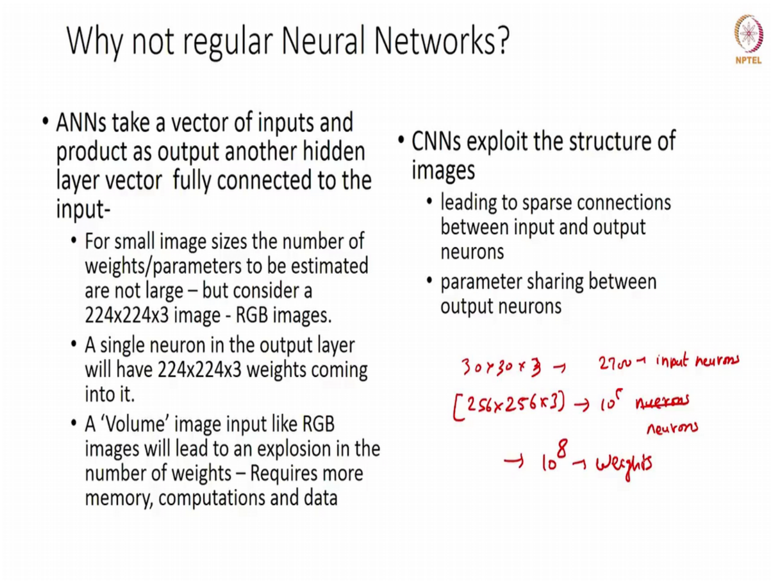
Red
Green
Blue
n_y
n_x
3- channels
$[n_x \times n_y \times 3] \rightarrow$ input

© Nevit Dilmen [CC BY-SA 3.0 (https://creativecommons.org/licenses/by-sa/3.0) or GFDL (http://www.gnu.org/copyleft/fdl.html)], via Wikimedia Commons

Now by images we also mean RGB images in that case let us take a color RGB mean basically typically color images if you take an RGB image we can extract the individual channels in RGB image. So an RGB image consist of 3 channel R, G and B. Each channel itself is… each channel is an image and the pixel values in each of the channel it is range from 0 to 255, so what we get as an RGB see as visualize as an RGB image is basically the combination of these RGB pixel values.

So if you want to give a CNN and RGB images as input which means that so you have for every image nx pix ny, nx times ny pixels as the input, however there are also 3 channels as they are called. What is the terminology typically used in CNN’s? So your input is basically nx cross and ny cross 3, size of your input. Once again as before if you want to think in terms of regular ANN’s you have to rasterise the image and making into vector of size nx times ny times 3. So the CNN basically takes if you can generalise CNN basically takes a volume as input by volume I mean that you have a pixel array of given size nx ny however there can be multiple such array which gives rise to volume. CNN takes as volume as input and assigns that volume to a particular class label based on your object (())(12:45).

(Refer Slide Time: 12:49)



Why not regular Neural Networks?

- ANNs take a vector of inputs and product as output another hidden layer vector fully connected to the input-
  - For small image sizes the number of weights/parameters to be estimated are not large – but consider a 224x224x3 image - RGB images.
  - A single neuron in the output layer will have 224x224x3 weights coming into it.
  - A 'Volume' image input like RGB images will lead to an explosion in the number of weights – Requires more memory, computations and data
- CNNs exploit the structure of images
  - leading to sparse connections between input and output neurons
  - parameter sharing between output neurons

$30 \times 30 \times 3 \rightarrow$ 2700 $\rightarrow$ input neurons

$[256 \times 256 \times 3] \rightarrow 10^5$ ~~nuerons~~ neurons

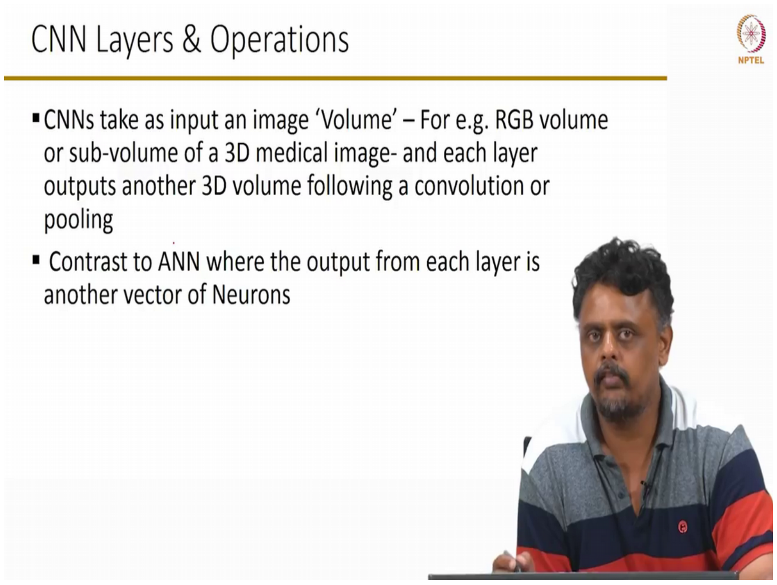
$\rightarrow 10^8 \rightarrow$ weights

So since we can rasterise these images so we can read out these pixels values one at a time and form a vector and then why not we just go ahead and use a regular artificial known network. Now if these images are small let us say if we have a 32 by 32 image or 30 by 30 images for the sake of calculation so you have 30 times 30 image. There are 3 channels okay so basically 2700 input neurons for a regular ANN okay so you will have over 2700 neurons, however most regular sized images of the order of 220 or 256 times 256 and if you have 3 channels, so this is already of the order of 10 raise to 4 and 10 raise to 5 neurons okay.

So which means that if you are (())(13:58) we want to get a hidden layer of 1000 neurons which will give rise to 10 raise to 8 weights this is a very conservative estimate because sometimes the images are (())(14:11) just 1000 by 1000 or 512 by 512 especially medical images are quite large, so as the size of your input image is increase ANN's do not scale very well so they are unable to handle such…the number of weights or there is (())(14:26) of number of weights that has to be estimated using ANN that means that proportionally a large number of data points are required.

Another aspect of why you should not use ANN this because ANN's once because if ANN's takes as input vector so which means that even if you are given an image we have 2 victories it by rasterising it and in that process we will lose the spatial structure of the data okay, so images have spatial structure which is what we want to exploit by using a CNN and in the process we also as we saw earlier exploit 2 more things we get as sparsely connected network which means that there will not be as many weights as artificial neural networks. In addition there is also a parameter sharing which again reduces the number of weights and the parameter sharing which in turn enables us to exploit the local connectivity of neural network.

(Refer Slide Time: 15:28)



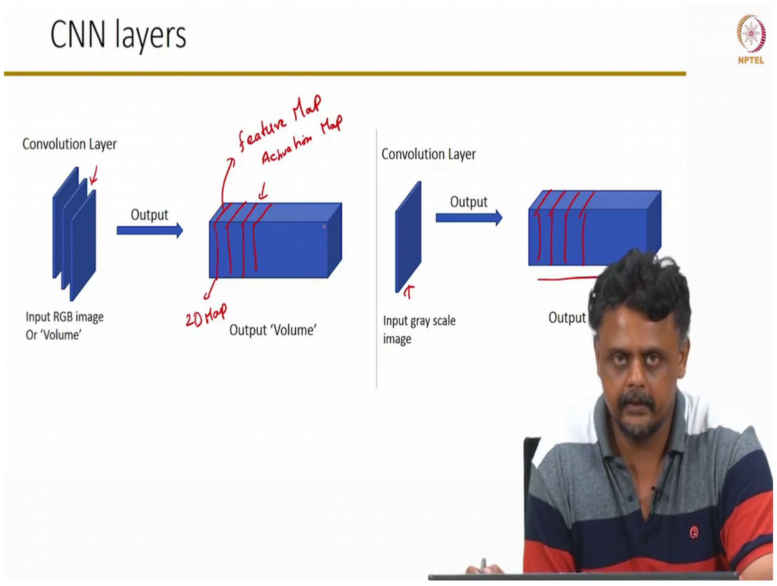
CNN Layers & Operations

- CNNs take as input an image 'Volume' – For e.g. RGB volume or sub-volume of a 3D medical image- and each layer outputs another 3D volume following a convolution or pooling
- Contrast to ANN where the output from each layer is another vector of Neurons

The parameter sharing enables us to exploit the local connectivity in an image, so what does a CNN consist of okay, CNN's like ANN's consists of sequence of hidden layers but these hidden layers are basically convolutions or pooling, so we will see what these are in later slides but it is an alternation of convolution and pooling layers followed by a series of fully connected layers just like in artificial neural networks leading to a classification layer. This is a typical structure of a convolution neural network.

Now why do we have this kind of structure, what is this convolution doing here? What does it mean? What does convolution accomplish? We will examine that okay before we go there will also look at…just to summarise CNN's take as input must (())(16:30) saw earlier images but these images can have multi channels, so simple example being a RGB image which has 3 channels, so they take as input a volume and in each layer in a CNN outputs are volume irrespective of whether it is a convolution or a pooling layer okay this refrain from contrast to ANN's where the output is based on every layer is basically another vector of neurons.

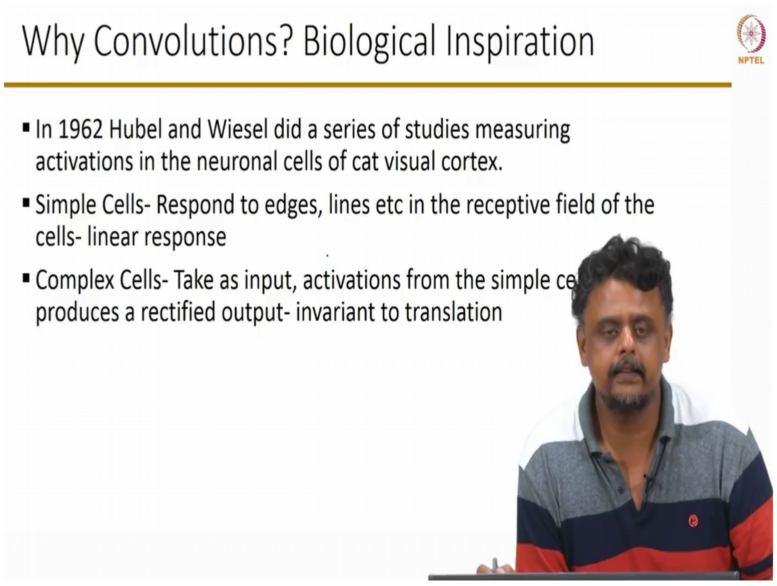
(Refer Slide Time: 17:05)


CNN layers

Just to summarise again in visual form of convolution layer takes us so in this case the input is an RGB image and we will not know we will not look at what the operations are right now but the output from a layers is basically a volume here where if you can slice the volume this way okay then have multiple outputs or multiple 2-D outputs, each of these are 2-D map. Each of these 2-D outputs is often referred to as a feature map or an activation map so the number of output features map or the activation maps is entirely within our control we will see how that can be defined okay and the size of the 2-D map the nx, ny (())(18:15) 2-D map again is determine by the operations we perform.

Similarly even for a single input channels, so this is for 3 input channels so irrespective of the size number of channels in your input you can have multiple channels in your output this is true of every layer, so for instance if you take this layer this can again undergo another convolution leading to even more higher number of activation maps being output we will see that in some popular CNN architectures later on.

(Refer Slide Time: 18:53)



Why Convolutions? Biological Inspiration

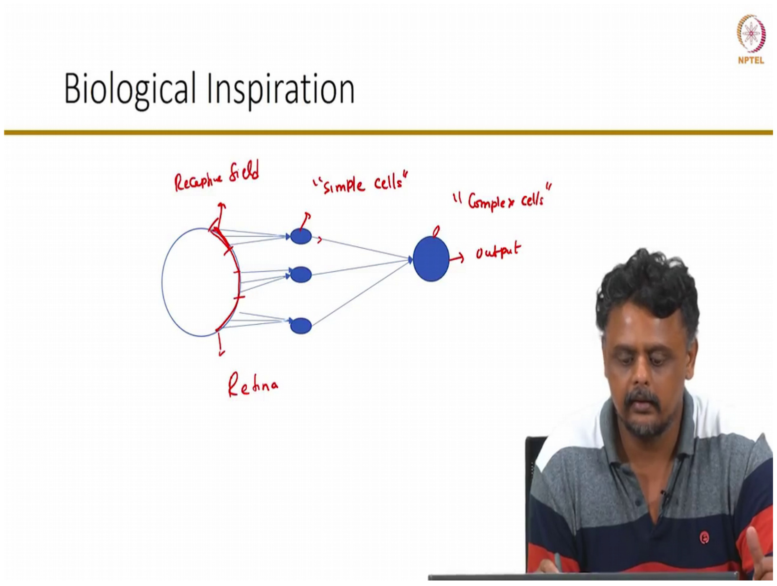
- In 1962 Hubel and Wiesel did a series of studies measuring activations in the neuronal cells of cat visual cortex.
- Simple Cells- Respond to edges, lines etc in the receptive field of the cells- linear response
- Complex Cells- Take as input, activations from the simple ce produces a rectified output- invariant to translation

So why convolutions or what do these convolutions accomplish, what are they inspired by? We will look at what exactly convolution do but before that…so in 1960s Hubel and his colleagues did series of experiments measuring the activations or signals from neurons in the primary visual cortex of cats okay. We will not go into exactly how they did the experiments but these are biologist they knew what they were doing.

They eventually won a Nobel Prize for this work, they found out that in the primary visual cortex, the primary visual cortex in turn we will see later will take its input from the retina. The retina is where what we see is projected so our eyes, the lens of our (())(19:30) projects whatever we see on to the retina, so the primary visual cortex and the signal from the retina goes to the primary visual cortex. So the primary visual cortex they found out have 2 types of cells, neuron cells, a simple cells.

Simple cells of course they get their signals from the rods and cones in the retina and the simple cells respond to adjust, so they have very good response to adjust of different orientations and it was a linear response okay and then there are other types of cells called complex cells which seems to take input from the simple cells, so a linear combination of input from the simple cell and had an nonlinear response. One aspect of it was that it was insensitive to translations, so you can move an edge across the eye or have a project an edge on the retina of the eye and move it across the retina but the output from a complex cell would be the same that is what it means. So they concluded that the visual cortex have these 2 types of cells and this behaviour they characterised okay.

(Refer Slide Time: 20:41)

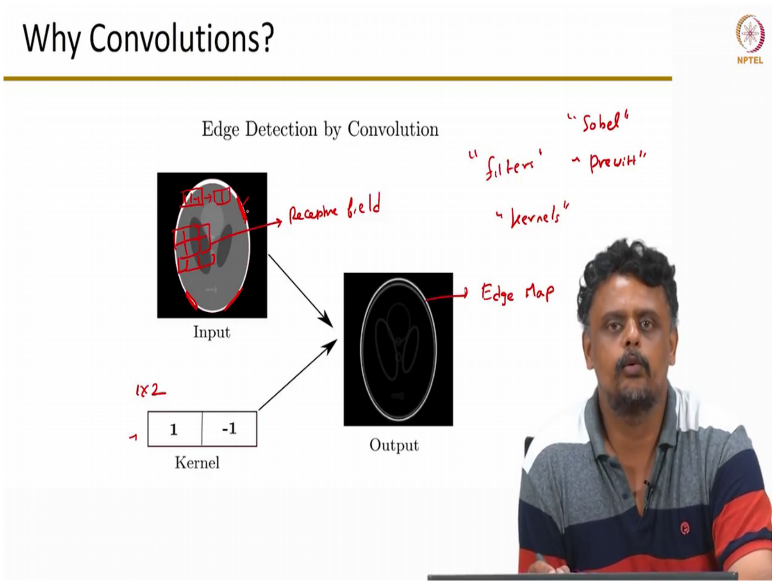


Biological Inspiration

So this was the inspiration for…so if you want to look at it, so this let us say is your eyeball and this is your retina right here. The simple cells here take as input signals from a particular region this is the region, small region here for this cell, so this is called the receptive field of that particular cell or group of cells, receptive field okay. So what it does is it does a linear combination of the signals from the receptive field and provides an output okay.

Similarly there are bunch of these simple cells each has its own receptive field in the retina of the eye and they in turn provide an output and this complex cell takes a weighted combination of these inputs and provides an output and it is nonlinear output okay from the cells. So this is the inspiration behind CNN's, so they try to mimic this vision this is of course the general wisdom that goes around but (())(22:04) that is not a very good understanding… There is a lot of progress in this field that how vision system works but this is a very simplified like I said cartoon version in my case that is what I have done here cartoon version of how the vision system works.

(Refer Slide Time: 22:22)


Why Convolutions?

Edge Detection by Convolution

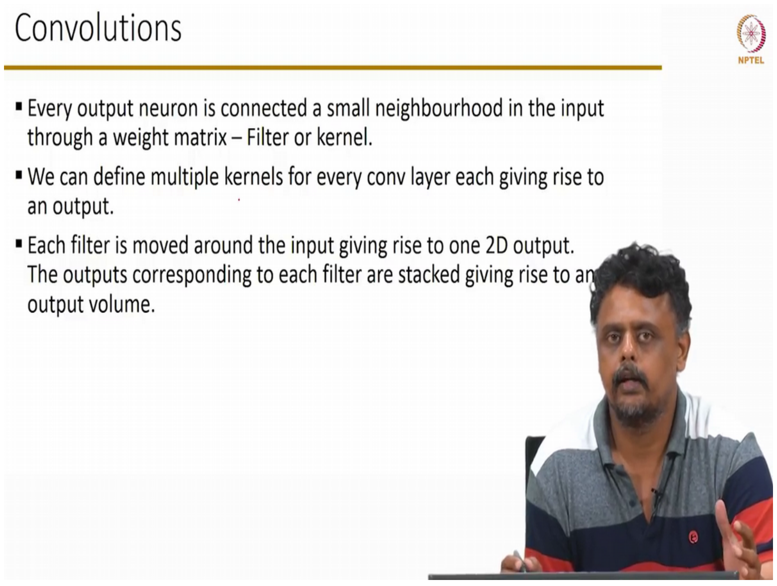
Another way of looking at it is why do you use convolution is that if you go back to signal processing or conventional image processing techniques it is well-known that if you have an image is we can define filters or what you call kernels okay, you can have filter or filter kernels as they are called ill which if you operate on an image should be able to extract different features from it, so in this case I have shown a very simple kernel say 1 row to column is 1 cross 2 filter kernel, so all you have to do is superimpose it on the image, so this is 1 and minus 1 multiply with the underlying pixel values and add them.

You can see that by applying this filter kernel and of course translate and do the same thing everywhere okay. So if you do that all across the image then you will get the edge map. Now it is possible to construct by hand different filters like these which will highlight edges at different orientations. So there are multitudes of filters so for instance there are sobel filters or prewitt filters and so on which has been already which mean they have… It is well-known in I mean in traditional image processing literature that these filters can use to highlight adjust an image which is very similar to what we saw with how the simple cells work is basically remove this kernel so the receptive field for this kernel is about 1 cross 2 okay.

So we can make 3 cross 3 so bell filter so the receptive field for that filter is a 3 cross 3 region in the image let us say if you define a 3 cross 3 filter and this will be receptive field of the filter, so the idea is if we define enough filters then we will get a variety of edge map this is just one… we can call this the first convolution layer and then we do more combination of those edge maps to get may be higher order description of the image, so what these filter kernels exploits is that the edges in an image are similar everywhere, so for instance if you have a vertical edge somewhere in an image, in one region of the image let us say there is an edge in this case there is an edge here.

I have a similar edge here too the orientations are different, right so you can have an edge here which is very similar to the edge (())(25:17) pointed out like this edge here and this edge here are the same very similar. So if I define a filter that picks out an edge at that angle I will say that is an 45 degree edge than I can use it all over the picture to highlight that particular edge okay, so this is what is parameter sharing is all about in the sense that for every… for identifying a particular feature I do not need to define a new filter for every region in the image. It is the same filter that I can apply for wherever there is an edge at this particular angle that filter will pick it okay. This is another way of looking at convolution neural network okay.

(Refer Slide Time: 26:01)

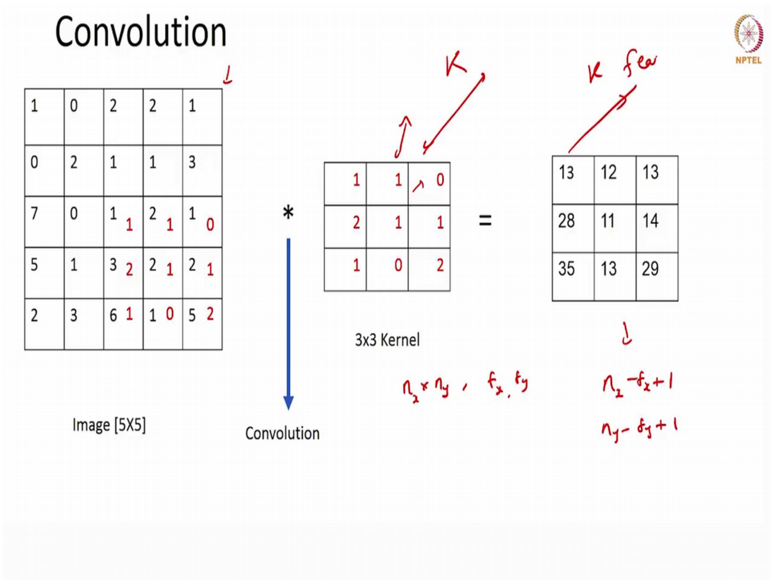

Convolutions

- Every output neuron is connected a small neighbourhood in the input through a weight matrix – Filter or kernel.
- We can define multiple kernels for every conv layer each giving rise to an output.
- Each filter is moved around the input giving rise to one 2D output. The outputs corresponding to each filter are stacked giving rise to an output volume.

So what does convolution accomplish? So like we saw the convolution neural networks same structure as artificial neural networks there is an input layer forward by sequence of hidden layer and then there is output. Now the output of any hidden layer and in for general you can think of input as also as a layer in the network, so the output of every layer, the neuron in the output of every layer is connected to a small neighbourhood in the input that is what the convolution kernel accomplishes, the filter (())(26:36) that is what it accomplishes and the connection is through a weight matrix which we call the filter or a kernel okay.

For every convolution layer we can just define multiple filter kernels and the way it works is that we move the filter kernel around the image at every region and every position that we move it around to, we multiply with the underlying pixel values and add them of up so it is a sum of products, so which gives rise to a corresponding output, so since we can define multiple filters in every layer we can stack the output of each of the filters by obtained by applying each of the filters in the input and giving rise to another volume of hidden neurons.

(Refer Slide Time: 27:31)



So let us just look at how a typical convolution works, so let us just look at how a typical convolution works, so what we see on the left her is your toy image can call it, it is a 5 by 5 image and on the right is your 3 by 3 convolution kernel okay, so how does one actually perform the convolution that is what we are going to see.

So it is very simple all you have to do is superimpose the convolution kernel starting at some point at the top left part of the image you can start from anywhere typically top left corner of the image multiply so just it will be 1 cross 1 times 1 plus 0 times 1 plus 2 times 0 plus 1 times 1 plus 2 times 1 so on and so forth and you multiply and you add so it is a sum of products, sum of the corresponding elements which means that the corresponding elements in the image with the filter weights.

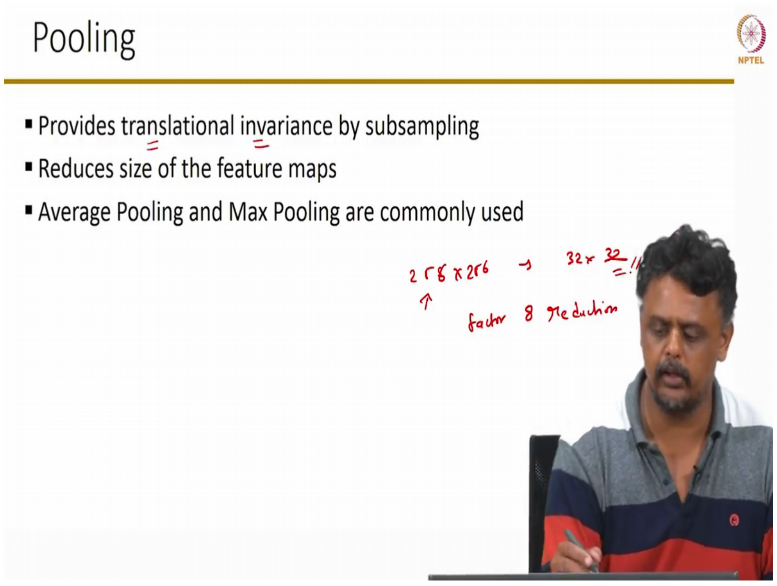
So that give rise to one element in your output feature, so next step is to slide the kernel to the right or by 1 pixel and perform a similar operation, we can keep doing that because once we hit the edge of the image, so now if we go any further then the filter will not fit completely in the image, so we will stop there and then we move 1 pixel down and continue to do so and at every point we placed the kernel, we multiply with the underlying pixel values add and then obtain the corresponding output there.

So as we move through we see that at every position we perform the same operation and once again (())(29:24) here and if you go down any further if you move the… shift the filter down any further then it will not fit inside the image, so we stopped right there will. So in general if you have an image or input of size nx times ny and your filter kernel is of size fx or fy. Your output size will be nx minus fx plus 1 and ny minus fy plus 1 okay so this is the basic output. So you see that as we do the convolution the output size keeps decreasing and there is a way to stop that, we will see how that is done in a more systematic way but this is typically what happens when you do convolutions.

So at every convolution layer you will define a multitude of these filters, so when I say define you really do not know so because these are the weights right these are similar these are the equivalence of the weights in your artificial neural network, so that is what we typically estimate in artificial neural network, the weights of the network by optimising an objective function. In this case we will determine the members of the filter kernel again by optimising a suitable objective function.

So at every layer given an input we can define many such filters kernels so we will define K filter kernel K can be any…very large number so for (())(30:59) there are instances there are networks which define 512 such filters in every layer okay and so the output would be K feature maps of each of this particular size, so typically in every layer each of the filter maps are of same size even though there are exceptions which we will again look at later and later videos, so typically you also… For instance you would… 1 layer would contain you would define 3 by 3 filters about 256 of them. All filters would be of size 3 by 3, so all the feature maps would be of the output feature maps would be of the same size but they will be about K (())(31:42).

(Refer Slide Time: 31:46)



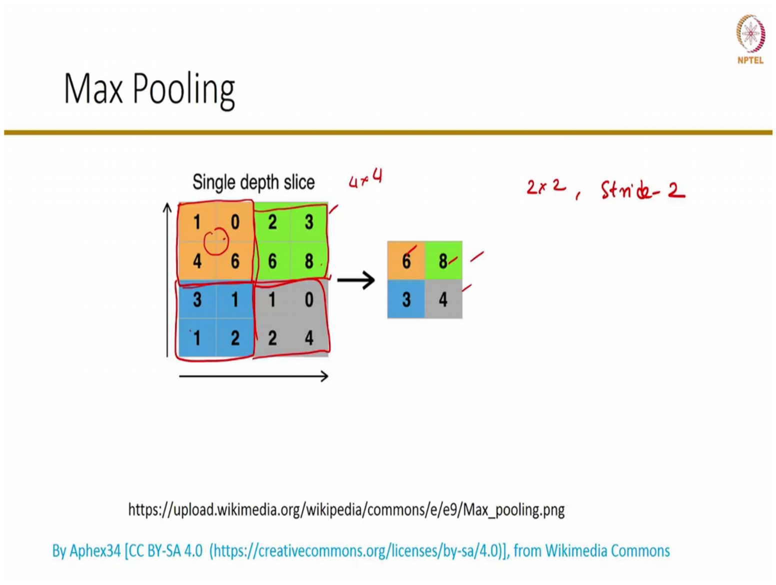
Pooling

- Provides translational invariance by subsampling
- Reduces size of the feature maps
- Average Pooling and Max Pooling are commonly used

Now in this another operation which we define there was pooling, so following convolution or a series of convolution there is also pooling. What does this pooling accomplish? One of the major advantages it supposedly gives is the translational invariance, so it is very easy to visualise. If you have an object in your picture and you are trying to localise it let us say you are using the neural network, now if keep subsampling the picture let us say you subsample the picture from 256 cross 256 to 32 by 32 right? Almost a factor of 8 reduction factor 8 reduction okay and almost or exactly a factor of 8.

So let us say if the object moves around inside this image, inside the 256 by 256 image. If it moves less than 8 or 16 pixels you hardly see a motion in the 32 by 32 image, so basically what this pooling performs is a subsampling operation. It reduces the size of your feature maps and as you build more and more layers it comes to a point where in very large motion of…so the network kind of becomes invariant to very large motion of the object you are trying to detect in your main image. Typically average pooling and Max pooling are commonly used.

(Refer Slide Time: 33:21)



Max Pooling

https://upload.wikimedia.org/wikipedia/commons/e/e9/Max_pooling.png

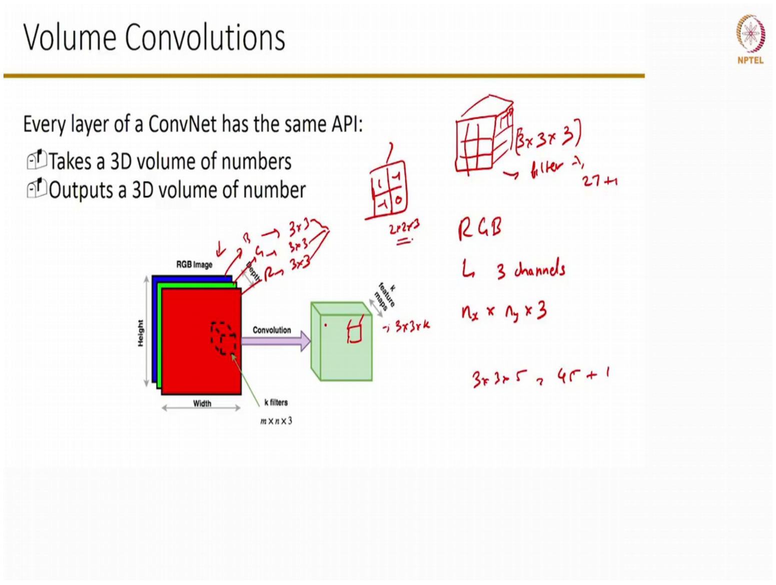
By Aphex34 [CC BY-SA 4.0 (https://creativecommons.org/licenses/by-sa/4.0)], from Wikimedia Commons

We will look at Max pooling average pooling should be a value obvious, so let us take this feature map of size 4 by 4, this is of size 4 by 4. A typical Max pooling operation would be to... It is again you can think of it as using a kernel, so you define a 2 by 2 kernel here okay and what max pooling does is to look at the maximum value inside this 2 by 2 space, right so here it is 6 and the way Max pooling is done unlike we saw that for convolution you just slide the filter kernel by one, here you slide the…you do not have overlap.

So you would side so that there is no overlap you can also skip we see that (())(34:02) is tried and there is no overlap and in this window 8 is the maximum. Similarly here is 3 in this window 3 and in this window 4 is the maximum, so if you do Max pooling of size 2 by 2, right with a stride in this case, the stride is how many pixel do you move before you do another Max pooling operation, in this case you move 2 pixels so the stride of 2 then you will half the size of the feature map.

So this is the function of the Max pooling so if you want to be very systematic about it you would try to retain the size of your feature map when doing convolution we will see how we can do that and we will try to…any subsampling will be done during the pooling operation. For average pooling instead of the Max value you will take the mean of the values inside this 2 by 2 neighbourhood.

(Refer Slide Time: 35:17)



Volume Convolutions

Every layer of a ConvNet has the same API:

- Takes a 3D volume of numbers
- Outputs a 3D volume of number

So we saw that… we mention earlier that the neural network, the volume metric convolution or volume convolutions are done okay, so what do you mean by that? So because whatever you have seen so far we are usually defining the filter kernels to be of a 2 by 2…as a 2 dimensional matrix right, so for instance we have a 3 by 3 filter kernel right? So fx and fy so what do you mean by volume matrix convolution? How does it work? So let us take this particular example, so input let us just look at the input layer it generalises to all layers, so input layers is an RGB image we have 3 channels, so RGB so input is 3 channels, so basically our input size is nx times ny times 3 okay.

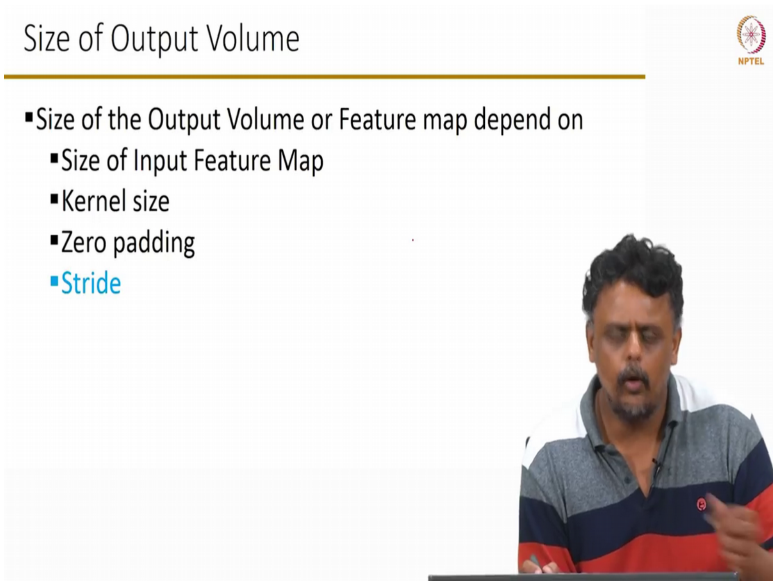
This 3 is each of them the R, G and B channel and each of them is an image, now when you perform a convolution, so let us say we define a 3 by 3 convolution and it need not be a 3 by 3 can even do a 5 by 5 convolution just to prevent any…if you are uncomfortable with 3 by 3 you can use 5 by 5 kernel it is easy to draw that, so this is your filter let us say you are doing a 3 by 3 convolution on your input image which has 3 channels. Now even though we say 3 by 3 the filter itself would be defined as a 3 by 3 times 3, so there will be a filter which operates on the blue, green and red.

Each of them would be operated on by a 3 by 3 concurrently okay, so a neuron in one of the output feature maps is the sum of all the outputs right, so we saw earlier so when we go back to these slides but in video we see that we saw earlier that we superimpose the filter kernel on a region at a location in the image multiply with the underlying pixel value to get one output this is on a 2-D feature map, 2-D input and 1 filter kernel. Now we have in this case 3 channels, so the filter kernel itself is 3 by 3 by 3. If we had 5 channels as input then the filter kernel would be 3 by 3 times 5 with 45 here it is 27 values here it will be 45 values that we are…plus a bias units if you want to include the bias okay.

This is how volume metric convolution are done, the filter acts across the channels or across the volume okay, so in this case the output let us say we have defined K filters of size or particular size then it will give you K feature maps okay. Now if I do 3 by 3 convolution we are actually using 3 by 3 times K size filter to operate on this feature, so it will be 3 by 3 but it will act across the feature maps. So this is what we refer to as volume convolutions that it takes… the number of input channels can be variable.

So typically you will only define 3 by 3 or 5 by 5 filters but it is implicit that those filters also act across the channels depending on how many channels you have as input and all the values in the filter kernel will be unique, so in the sense do not think that if you define a 3 by 3 kernel let us say or in this case for simplicity 2 by 2 kernel, let us say you define it like this something of that sort okay, so this is not duplicated across the channels right so you will have… so if we have 3 input channels there will be times 3, so this is 2 by 2 times 3 this is the size of your filter kernel and this have as many unique elements as determined by your back (())(40:01). So their weights will be estimated like that way okay. This is important point understand because many beginners really falter here sometime to understand this okay.

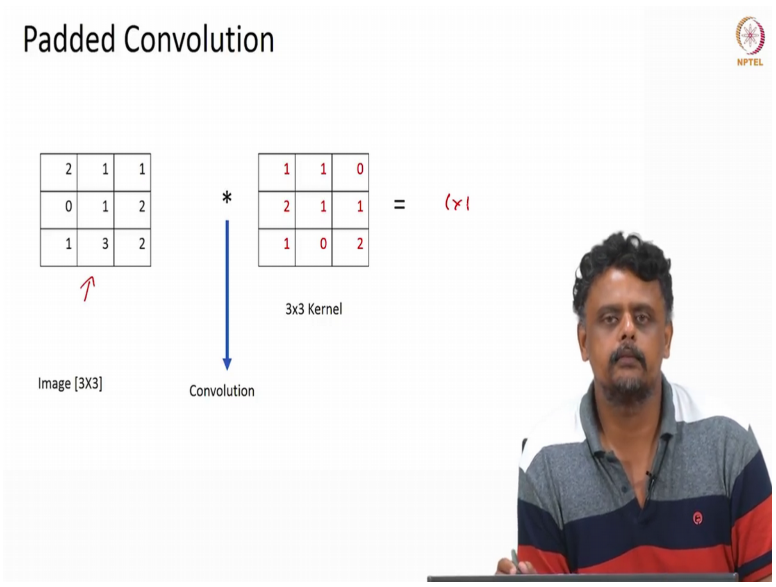
(Refer Slide Time: 40:15)


Size of Output Volume

- Size of the Output Volume or Feature map depend on
  - Size of Input Feature Map
  - Kernel size
  - Zero padding
  - Stride

So how do we determine the size of the output volume and we have seen some hints so far, we will just do it very systematically. See size of the output volume or the feature map it depends on the size of the input and we saw that, the size of the filter kernel we are using, how much zero padding we do we will see why we use zero padding and the stride in the network okay.

(Refer Slide Time: 40:43)



Padded Convolution

2	1	1
0	1	2
1	3	2

$*$

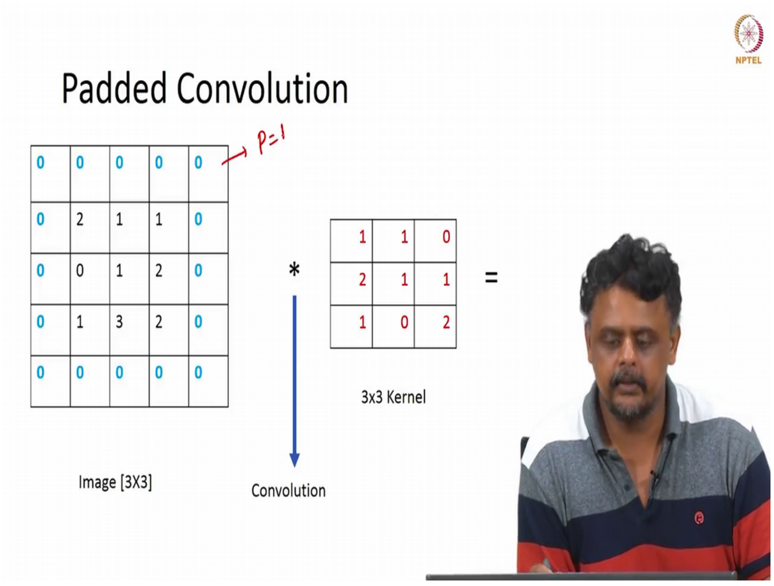
1	1	0
2	1	1
1	0	2

= 1x1

So padded convolution, why would you want to do padded convolution? So we saw earlier that when we do a convolution with the 3 by 3 kernel the size of the output was less than the size of the input okay. Suppose we want to preserve the size, the reason is if you have multiple convolutions layers…as you do more and more convolution at some point the size of the feature maps will become so small that you cannot do any more filtering, so it actually adds restrictions on how deep you can go okay.

So in order to avoid that we sometimes would like to do padded convolutions, so all we have to do this is our input feature map size again we will just operate with one feature map at a time. It automatically applies to a volume, input volume so it is very difficult to show it on screen that way, so we have input image of size 3 by 3. This very simple so if you apply this 3 by 3 kernel on this 3 by 3 image your output will be 1 cross 1 that is typically what we get but then you cannot do subsequent filtering on top of that it becomes difficult.

(Refer Slide Time: 42:04)



Padded Convolution

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

$\rightarrow p=1$

Image [3X3]

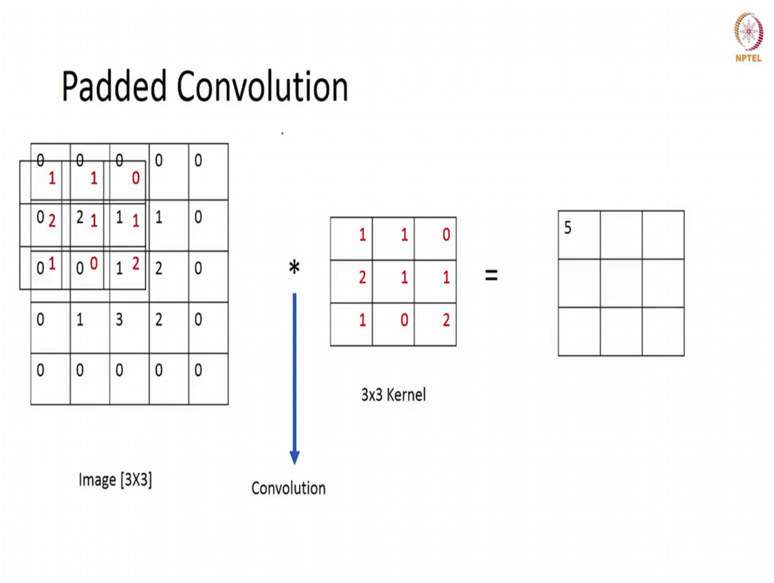
*

Convolution

1	1	0
2	1	1
1	0	2

3x3 Kernel

=



Padded Convolution

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

Image [3X3]

*

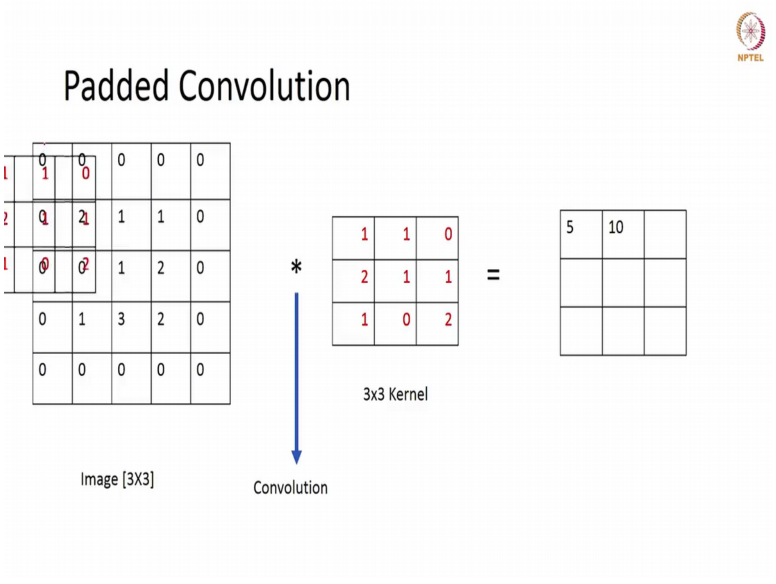
Convolution

1	1	0
2	1	1
1	0	2

3x3 Kernel

=

5		



Padded Convolution

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

Image [3X3]

*

Convolution

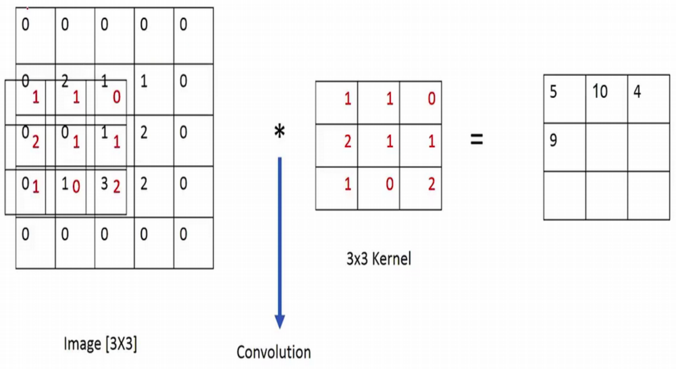
1	1	0
2	1	1
1	0	2

3x3 Kernel

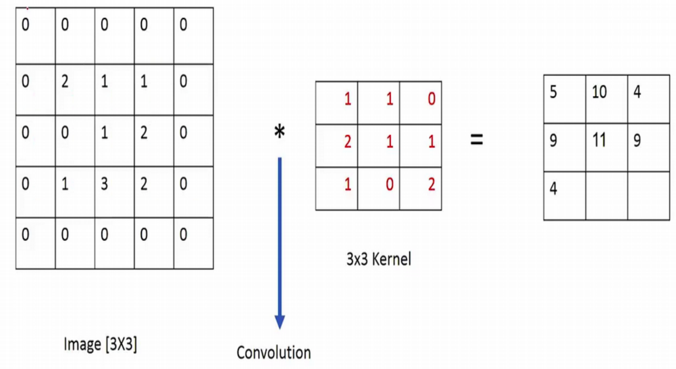
=

5	10	

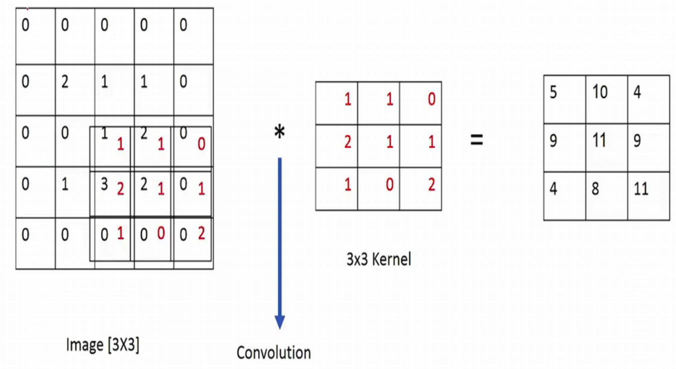
Padded Convolution

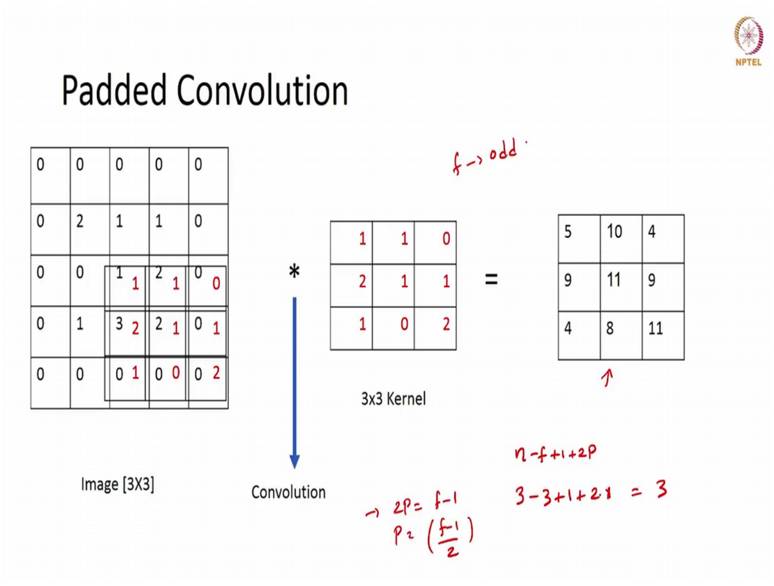


Padded Convolution



Padded Convolution



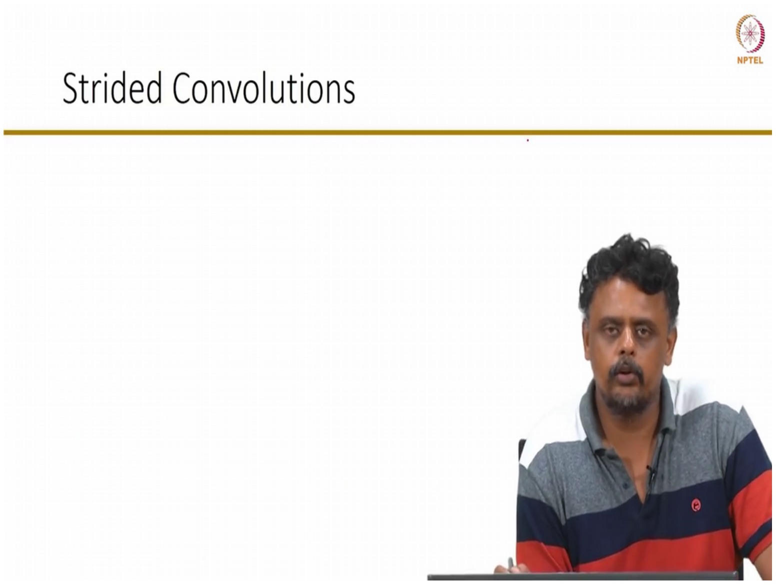


So to preserve the size of a feature map, so you will padded with zeros all around, so this corresponds to padding of 1 which means that you will pad with one on the left right top and bottom of the picture and if you do a convolution with this then you see that your output it is very similar to what we did earlier you will position your filter kernel at the top left and then move it around like you do for regular convolution except that now you have added 0 everywhere but then by doing this by adding zeros everywhere on the edges we get an output feature map of size 3 by 3 okay.

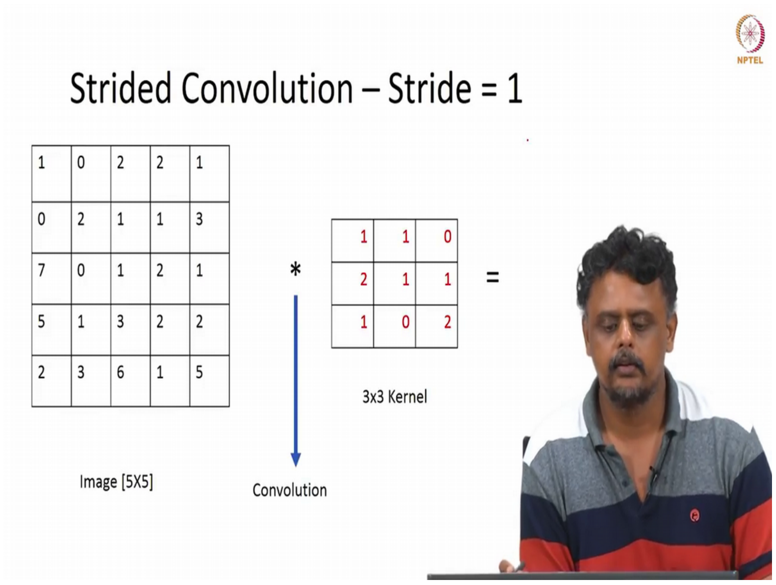
So this is the idea behind padded convolution is that it helps to preserve the size of your input of course you can add more zeros to it, it will be just slightly larger feature map you will get there but you can add more zeros but typically it is done to preserve the size of your feature map and zero padding is determine buy the size of your feature kernel, so in general we will just use square feature maps as input, so if size of your feature map on one axis is N okay we saw that if we use a filter kernel of size F then the size of the output is n minus f plus 1 okay.

So if we have a padding of size P in this case P equal to 1, n minus f plus 1 let us just write this more clearly alright, so output is n minus the size of the feature kernel plus 1 and plus 2p okay, so in this case our original n was 3 our filter kernel size was also 3 plus 1 plus padding is one, so 2 times 1 so we will get 3. So if you want to preserve the size of the feature map following the convolution then you would want 2p f minus 1 or p is f minus 1 divided by 2 which is another reason why we would like our f to be odd. It is easier to otherwise you will get some fraction of values and you have to do rounder or slower or several adjustment have to be made down streams you can work through that, so typically will work with odd size filter kernel, 3 by 3, 5 by 5, 7 by 7 so on and so forth so that this calculations are simplified.

(Refer Slide Time: 44:57)



Strided Convolutions



Strided Convolution – Stride = 1

1	0	2	2	1
0	2	1	1	3
7	0	1	2	1
5	1	3	2	2
2	3	6	1	5

Image [5X5]

* Convolution

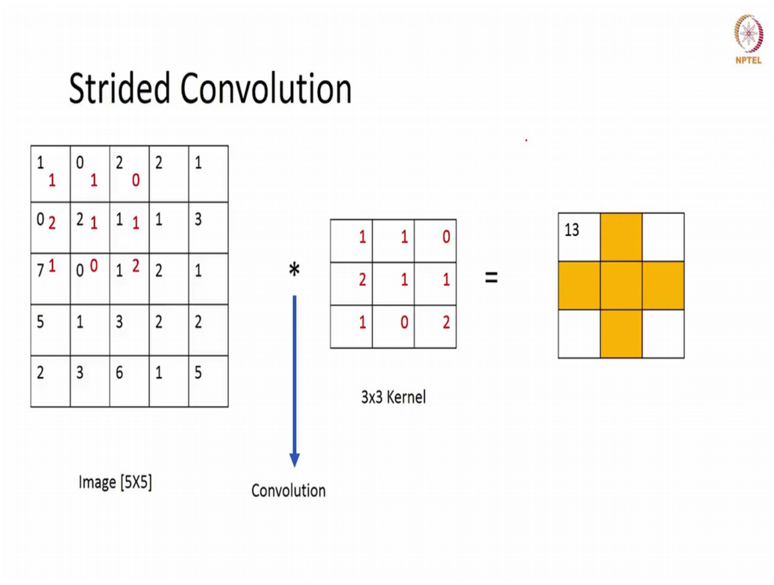
1	1	0
2	1	1
1	0	2

3x3 Kernel

=

Another operation that we typically do or strided convolutions again we saw that with the convolution the size of the feature map reduces, however it reduces gradually if you shift the feature map by one every time, so the slide of 1 is typically what by 1 pixel every time, so the stride of 1 is typically what we do, however if you have let us say a very large feature maps and we want to subsample it quickly otherwise memory becomes an issue then you do stride at convolution. It is very simple strided convolutions are very simple to understand except just that you skip a few pixel instead of moving the kernel on the image one pixel at a time you would skip multiple pixel at a time and how many ever pixel you skip is basically the stride you are using.

(Refer Slide Time: 45:47)



Strided Convolution

1 1	0 1	2 0	2	1
0 2	2 1	1 1	1	3
7 1	0 0	1 2	2	1
5	1	3	2	2
2	3	6	1	5

Image [5X5]

*

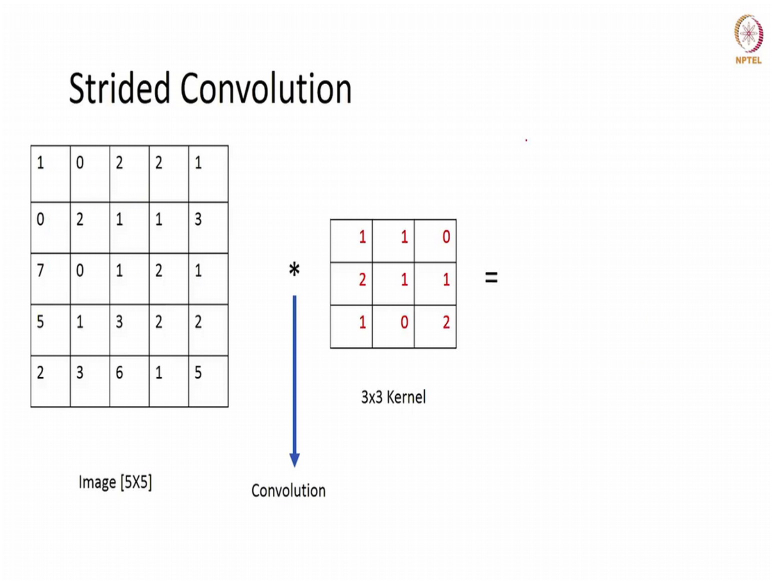
Convolution

1	1	0
2	1	1
1	0	2

3x3 Kernel

=

13	🟧	
🟧	🟧	🟧
	🟧	



Strided Convolution

1	0	2	2	1
0	2	1	1	3
7	0	1	2	1
5	1	3	2	2
2	3	6	1	5

Image [5X5]

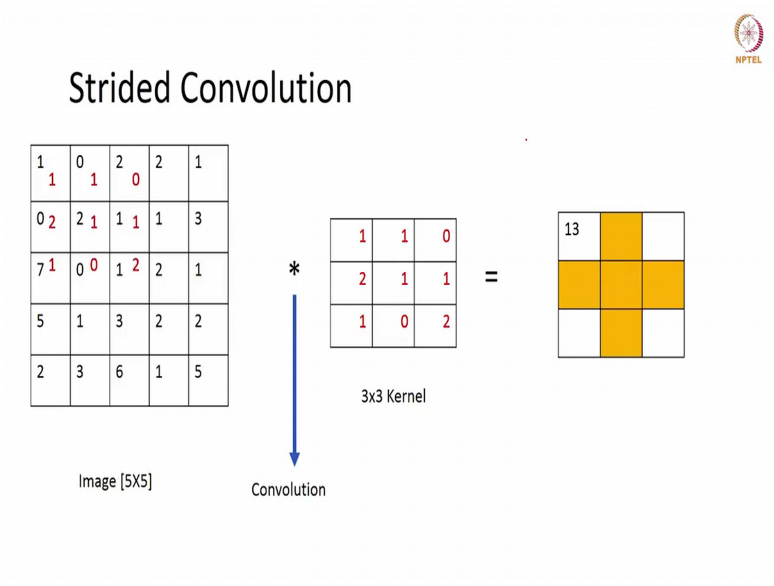
*

Convolution

1	1	0
2	1	1
1	0	2

3x3 Kernel

=



Strided Convolution

1 1	0 1	2 0	2	1
0 2	2 1	1 1	1	3
7 1	0 0	1 2	2	1
5	1	3	2	2
2	3	6	1	5

Image [5X5]

*

Convolution

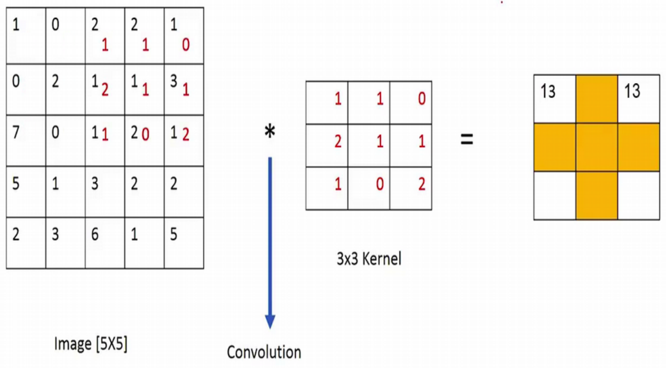
1	1	0
2	1	1
1	0	2

3x3 Kernel

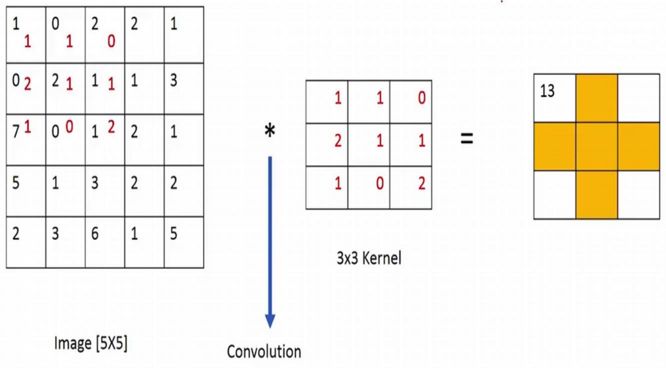
=

13	🟧	
🟧	🟧	🟧
	🟧	

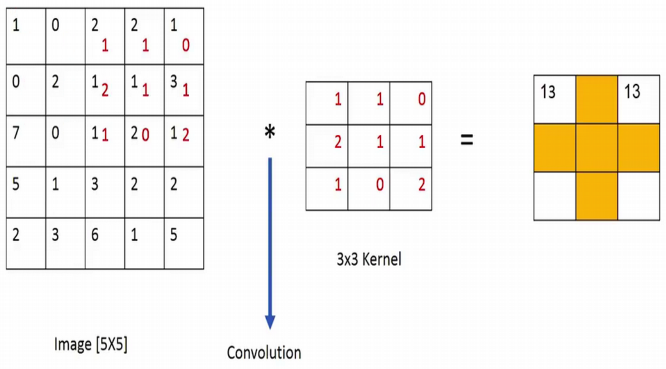
Strided Convolution



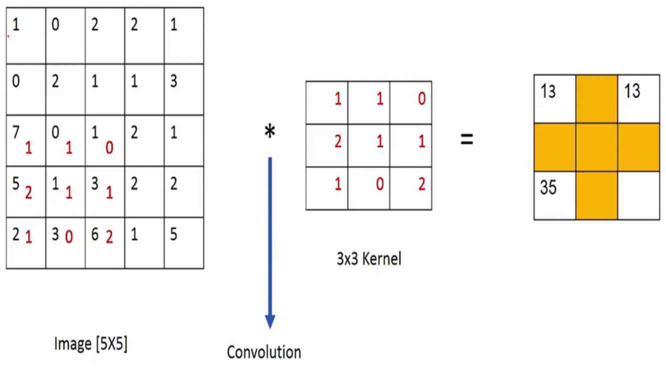
Strided Convolution



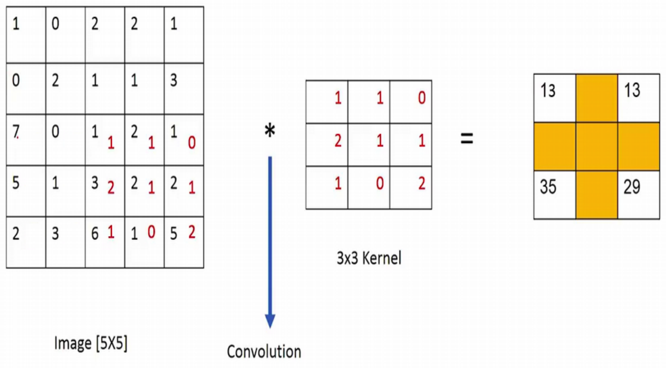
Strided Convolution



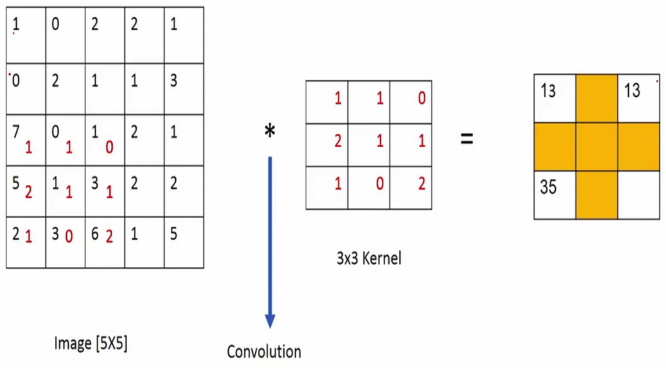
Strided Convolution



Strided Convolution



Strided Convolution



Strided Convolution

1	0	2	2	1
0	2	1	1	3
7	0	1	2	1
5	1	3	2	2
2	3	6	1	5

Image [5X5]

*

Convolution

1	1	0
2	1	1
1	0	2

3x3 Kernel

=

13	🟧	13
🟧	🟧	🟧
35	🟧	29

Strided Convolution

1	0	2	2	1
0	2	1	1	3
7	0	1 1	2 1	1 0
5	1	3 2	2 1	2 1
2	3	6 1	1 0	5 2

Image [5X5]

*

Convolution

1	1	0
2	1	1
1	0	2

3x3 Kernel

=

13	🟧	13
🟧	🟧	🟧
35	🟧	29

Strided Convolution

1 1	0 1	2 1	2	1
0 1	2	1 1	1	3
7	0	1 1	2 1	1 0
5	1	3 2	2 1	2 1
2	3	6 1	1 0	5 2

Image [5X5]

*

Convolution

1	1	0
2	1	1
1	0	2

3x3 Kernel

=

$w^T x \rightarrow$ non-linearity

Relu

13	🟧	13
🟧	🟧	🟧
35	🟧	29

$$\frac{n-f+2p}{s}+1$$

$$\frac{5-3+0}{2}+1 \rightarrow \underline{2}\ //$$

So stride 1 is what we saw earlier is typically how we would go but for stride let us say 2 we saw that for if you let us say if we want to do convolution stride of 2 then we saw… we usually start from the top left right here okay and we just looked that we skipped this okay at side of 2, so it moved 1 2 pixels and then you get the corresponding output there. Once again as you go down you will skip 1 2 and you will position the kernel there and of course you would do one more skip to get there, so typically you should have obtained 3 by 3 output but because of the stride you will get a 2 by 2 okay, so the general formula for strided convolution again giving the size of your feature map is n, square n.

You know that n minus f plus 2p divided by the size of your stride plus 1 okay, so in this case it is very easy to verify. Your input was 5 feature map the filter kernel size is 3. There is no padding stride off 2 plus 1 you get 2 by 2 feature maps, so you subsample we quickly typically okay. You typically work with odd sized odd number filter kernel 3 by 3… not odd number filter kernel sorry, the size of the filter kernel is pretty odd number so it is convenient to make this calculations because you have to make sure that at some point you do not run into this factional value and it will be difficult to resize your features maps.

So we typically work with this odd size filters, so that it enables you to do these calculations and end up with whole numbers rather than fractions okay, so to summarise briefly what we have looked at so far? We have looked at convolution neural networks, we see that they have… they are made up of a sequence of convolution and Max pooling layers leading to a fully connected layer and the decision layer.

Convolutions are basically done by defining filter kernel is at every layer in your network, so every layer you define k filter kernels and the convolution is done by moving the filter kernel across the image at every point multiplying with the underlying pixel value and adding to get the output. Max pooling is done to reduce the size of your feature maps, so we repeat these layers eventually leading to a fully connected and the decision layer, so we have not seen those yet I have also not mention that there is a non-linearity there of course because this is a neural network, so following a convolution you usually do a point wise nonlinearity.

So what do I mean by that let us say for instance, so in this case I have done a convolution with a stride. I would take these values, so this is what you would typically see in your ANN as a transpose right where in this case that W is the… Or the elements of your filter kernel and the x comes from the pixel values here, so you would pass it through a nonlinearity. Like a (())(49:49) for instance, so every activation in the output feature map will be put through a nonlinearity so that layer is always there.

So convolution followed by nonlinearity is typically done that is a usual thing, so you have a…so if you…in the deep learning (())(50:08) you will call it a linear layer which is basically w transpose x as we saw that, so are just some of products followed by a nonlinearity, so convolution nonlinearity followed by Max pooling and so on, so this typical sequence of operations that are done in a convolution neural network.

So we will in the next few videos look at a typical construction of a convolution neural network. See what the layers are and how we define these layers, how we would define number of kernels in every layer and what it would look like and how we would progress to let us say a fully connected layer and then to a distant layer or can we skip the fully connected layers or not? That is another aspect that you would like to look at, so this will cover in the next series of lectures.