

Medical Image Analysis
Professor. Ganapathy Krishnamurthi
Department of Engineering Design
Indian Institute of Technology, Madras
Lecture 43

Introduction to CNNs

(Refer Slide Time: 00:14)



Medical Image Analysis

Introduction to CNNs

Hello and welcome back. So, in this week we are going to look at convolutional neural networks or otherwise CNNs as they are referred to as. So, these are the kinds of networks that are specially used for analyzing grid-like data, specially images. And so, they have wide applicability in medical image analysis. And many of the state-of-the-art research that you see in especially image segmentation arise from a CNN like CNN algorithms.

So, there are CNN architectures, we will see what that means, developed specifically for image analysis, which do very well for all kinds of images from radiology to history of pathology. So, we are going to briefly look at what CNNs, what kind of algorithm CNNs are, and how to use them in the context of medical image analysis.

(Refer Slide Time: 01:10)



So, here is the organization for this week's lectures. So, we are going to look at introduction, which is basically the motivation for having a convolutional neural network, especially in the context of image data. We will look at what are the operations that go into a convolutional network. So, we have looked at feed forward neural networks. And we know the operations involved there, here there is some slight differences. We will look at what they are and the consequences of these operations.

And we will also take a brief look at some training tips, kind of important factors to keep track of when you are training a neural network. And you look at some architectures, the ones that are often used when designing CNN architectures for medical image analysis. So, these are architectures which were mostly developed for the image recognition.

And these are of course adapted for other variety of other tasks, even like segmentation. And so, we look at these architectures and some of the salient features of this aspect just to get an understanding of how they work.

(Refer Slide Time: 02:21)

Fully Connected To Convolution

fnw

- ANNs take a vector of inputs and produce 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 $224 \times 224 \times 3$ image - RGB images. *10^6 pesim*
- A single unit in the output layer will have $224 \times 224 \times 3$ 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

Feed forward neural network

hidden layer

So, some fully connected, from fully connected to convolutions. So, what do they mean? What does it mean? So, if we consider a artificial neural network, what it does, or ANN as it is called. We also refer to it as feed forward neural network, you must have seen this feed forward, this is what we refer to. It takes a vector of inputs.

Now these inputs, this vector of inputs, as we call them, we sometimes refer to them as neurons in intermediate layers. Each element of this vector we referred to as a neuron, or a unit, and these are used interchangeably throughout this lecture. So, an artificial neural network, which is basically a feed forward neural network takes a vector of inputs and produces as output, another hidden layer vector fully connected to the input.

So, what does that mean? It means that so for instance, if you have, I will just draw something very simple. Let us say three inputs. We call them, x_1 , x_2 , x_3 , and then let us say you have three outputs, the output layer, we call them h_1 , h_2 , and h_3 , each of them has a hidden unit. And this is a hidden layer, this whole thing is a hidden layer. This is a hidden layer.

So, now, we know that when we say fully connected, we mean that if you take h_1 is connected to all of its inputs in the previous layer, take h_2 it is connected to all of its inputs in the previous layer. If you take h_3 , once again, it is connected to all of its inputs and the this layer.

So, which means that if you have n_1 inputs, and n_2 in the outputs, you need a $n_1 \times n_2$ matrix of weights. So, with n_1 , n_2 being the total number of elements. So, but let now, let us consider,

let us say, an input, which is actually an image. For instance, let us say RGB images, these are images that you find on the internet that you take with your cell phones, etcetera.

These are the size $224 \times 224 \times 3$. So, if you just say, in fact, this is basically of the order of 10^4 features. So, if you take consider each pixel has a feature that is about 10^4 features, 10^4 to 10^5 features depending. So, now, if you look at this, then we need to connect, if you ever try to analyze these images, let us say the task of image recognition using a fully connected feed forward neural network.

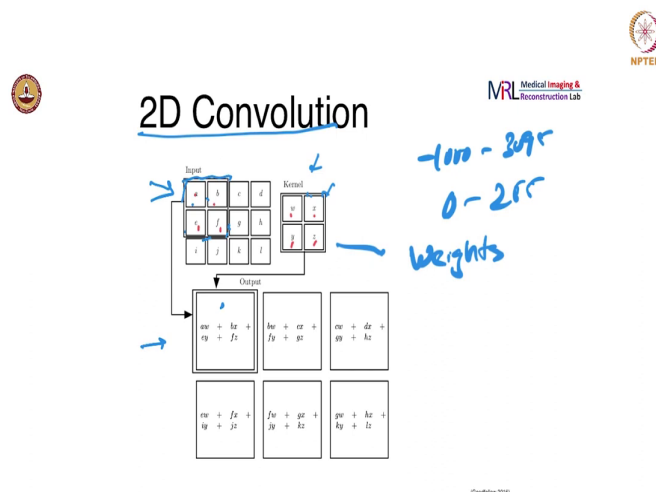
And if you take let us say 1 neuron, which is I am going to draw this here, if you take this one neuron, let us say there are just 10^4 neurons here, we say one neuron and output layer it has so many weights coming, just 10^5 weights coming into it alone. So, if you think about it, then you can see that the weight matrix, which takes which maps this input to the output layer, no matter how many neurons you make it to be, it is going to be a very large weight matrix.

So, it means there is an explosion in the number of weights, typically means that we ended up requiring more memory and computations. And of course, since there are more weights, we need more data to estimate. So, this is again a very nominal example. You can see just for one neuron, if you, how do we, if you rasterize this picture, let us say we take an image, images you see it as a grid and then you rasterize it, like you let it leave a flat on it like a vector and this out like a vector, now, we can have a fully connected neural network, which takes this as a feature input.

Entire image is like a vector of features, every pixel corresponding to a feature. And now if you even to produce one output neuron, you need as many connections as these, it is about 10^5 weights. So, imagine if you have a hidden layer, which have like 1000's of neurons, so you see the number of weights just explodes.

So, that is the issue with using a fully connected neural network to analyze data which has grid like structure. So, this is in the form of a grid. So, you can lay out the image pixel in the form of a grid primarily because there is correlation between nearby pixels. So, if you take an image, this pixel definitely has correlation with all the surrounding pixels, so it is meaningful to put them in the form of a grid. So, the proximity has meaning. So, a lot of the such data exists where you can put them in a grid and where it has meaning.

(Refer Slide Time: 07:17)



So, let us look at how typical output is calculated in a convolutional neural network. And this process is what is known as convolution, that is where the name comes from. So, we will look at 2D inputs. So, we have this image, this grid like input, and the values of at each of the grid points is basically a b, c, d, e, f, as shown by this represented by a b, c, d, etcetera, these have numerical values.

Of course, if you take a picture or an image, then each of these are nothing but the pixel values. So, we saw that for CT image, pixel values will go from the, voxel values will go from (-1000 to 3095). If you have a grayscale image, they will go from (0 to 255). So, those will be the values taken up by a b, c, d, etcetera. This filtered kernel, which basically these are the weights of your network.

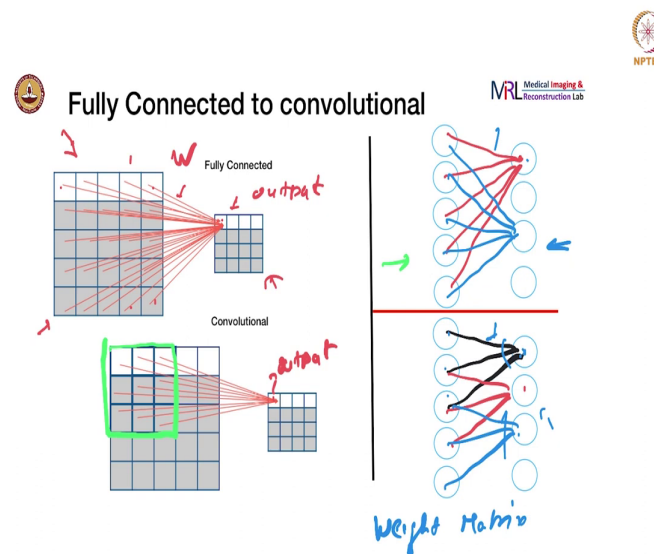
And it is actually in the form of very small matrix. And these are the order of 2×2 , 3×3 or 5×5 matrices. And we will see why it is arranged that way. And we can, of course, we will also see how we can represent this in the form of a typical feed forward neural network also. So, in the case of a convolutional neural network, here is how the output is calculated.

So, you would superimpose this kernel, filter kernel as it is called, on top of the image at various locations. And in fact, you would do this in a systematic way, by sliding this across the image at every position at one pixel at a time or two pixels at a time in horizontal and vertical directions would slide this kernel across the image. At every position, you would multiply with the corresponding elements of the input, add them to get the output.

So, for instance, if you take this particular output unit neuron, its value is basically $a.b$, sorry not $a.b$, $a.w+b.x+e.y$, like I did, and f time, plus f times z . So, you are going to superimpose this kernel on top the input, multiply the corresponding elements and add them up to get a particular outcome. And you do that by sliding this kernel across the input.

So, we will look at a slightly more detailed numerical examples in later slides. This gives you an idea of how this works. Of course, we can also represent this in the form of a feed forward neural network. We will see that also in the subsequent slides.

(Refer Slide Time: 09:50)



So, from fully connected to convolutional. So, I just want to show you this graphic, how we can understand this. So, if you are thinking of grid like inputs, and this is a grid is a 5×5 grid of input, you can call it a small image. And we want to, let us say calculate. So, this is the smaller box, here is your output. This is one of the hidden layers, if you can call it, the entire matrix is one hidden layer.

And if you do the fully connected way, then what you would have is a weight arising from every one of these pixels or grid points to the output. And of course, they are connected to the output through weights w . So, every output unit here in this grid is connected to all the input, all the elements in the input grid and through weights. So, there are as many weights as there are elements in the grid.

Now, when they form, when you come to convolutional neural networks, what we try to do is the sparse connection, we will see that because this is output, here, this output unit is connected to only a subset of the input. So, in this case, is connected to a 3×3 set of input.

So, you can see where the kernel comes from. So, last time, we saw 2×2 kernel, so which means that this is being operated upon by a 3×3 kernel to produce this output.

So, how do we visualize this in the context of feed forward network configuration. So, let us say this is the input layer right here, and this is the output layer. So, in the feed forward neural network configuration, what you will have for every output is connected to all its inputs, all units in the input layer. So, that will just show you for one more. So, that is typical configuration, that you would see this.

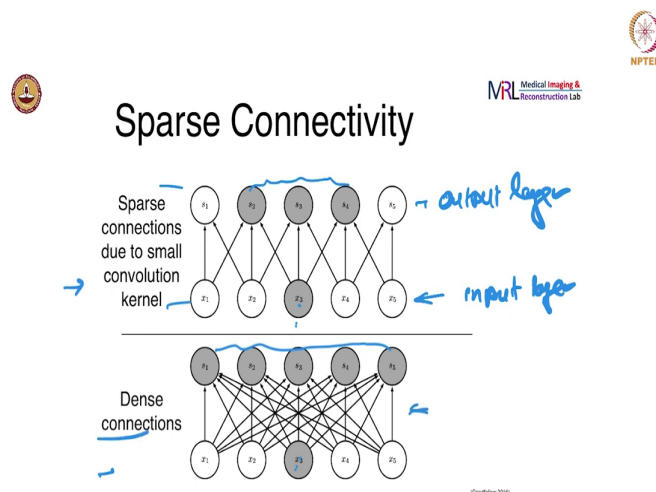
And once again, and the red and blue indicate a unique set of weights. Of course, there are in a sense, every red is different, slightly different. But this is one set of weights, which only connects to this neuron to our output unit. So, in the case of a convolutional neural network, you would represent this this way. So, for instance, this output is only connected to these three, and maybe this output will be connected to these three inputs, and so on and so forth.

So, this is not, again, this is not for 2D, this is in 1D. But still you can, if you rasterize this input and lay it out in a proper fashion, we can still illustrate it in this fashion. But just for the sake of understanding, you understand now for this output you are only connected to these three. Once again, I am assuming, I am assigning a unique set of weights to each one of these outputs. And we will see that that is necessarily not the case.

And that is where this concept called parameter sharing comes in. So, this is fast connection, because now see that this output is only produced by three inputs. So, if you write down the weight metrics, so if you write down the weight matrix itself, it is a good exercise, write down the weight matrix for this configuration. And you will see that there will be a lot of zeros in it. That is the way to understand. That is why it is a sparse connection.

And of course, there is parameter sharing because these sets of black weights, blue weights need not be different. So, we will see more about that in the later slide. So, that this is the general understanding of how we analyze grid data, just that we do not connect all the inputs to one output unit. When you are analyzing grid data, we are just looking at subsets of it. And they are connected using bunch of weights, which we call referred to as filter kernels.

(Refer Slide Time: 13:55)

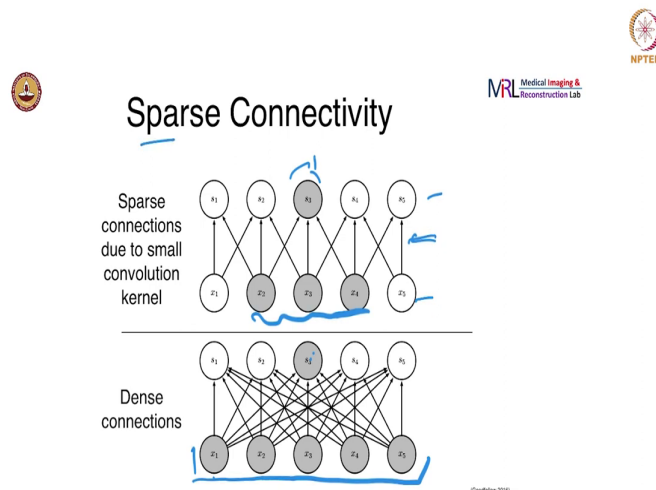


So, I will talk more about this sparse connectivity, the sparse connectivity due to a small convolutional kernel. We will look at what a convolution kernel is again in more detail, but for this purpose we just see this, but this is just a bunch of weights that take a set of inputs to the output unit.

So, from the point of view of the input, you see that even as I drawn in the previous slide, you see this x_3 in the input layer, so this is the input layer and this is the output layer. And they have these units in the input and the output. So, if you see from the input layer point of view, this particular input feature or unit is only connected to a very small subset of the output units.

And that happens because of the way we do these connections as you saw on the previous slide. And but if you look at the typical feed forward neural network we will see that every unit in the input is connected to all the units in the output. That is typical for in the case of feed forward configuration this is called, this is densely connected, as it is referred to here. And these are sparsely connected.

(Refer Slide Time: 15:16)

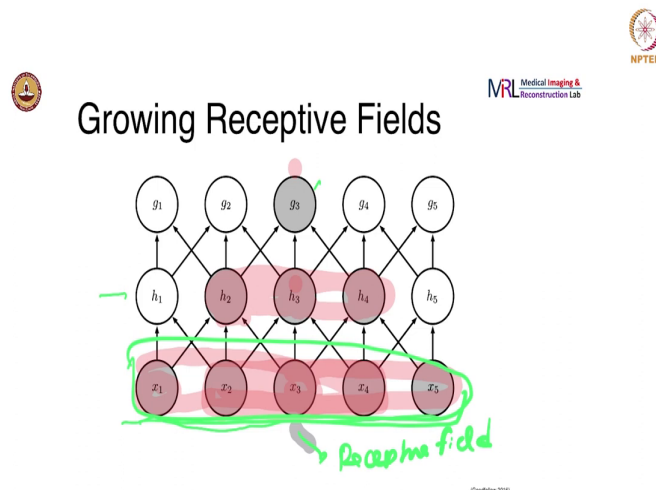


The other way to look at this from the output point of view. So, if you take one output neuron here, it is only connected to a subset of the input. So, just through the weights, of course, when I say connected, I mean the output is a linear combination of a subset of the inputs. In this case, just three of them.

So, but if you consider, let us say, feed forward neural network fully connected, you will see that they are connected, that all of the input is connected to each one of the outputs. So, s_3 is for instance, gets its inputs from all of these five, there are only five units in the input. So, this we again saw a few slides ago some of these pictures.

So, this connectivity, like I said, if you write the weight matrix for this connection, for this layer, for mapping X layer to the S layer, you will see that that weight matrix has a lot of zeros in it. So, that is the understanding of how, that is how you understand a sparse connectivity.

(Refer Slide Time: 16:23)

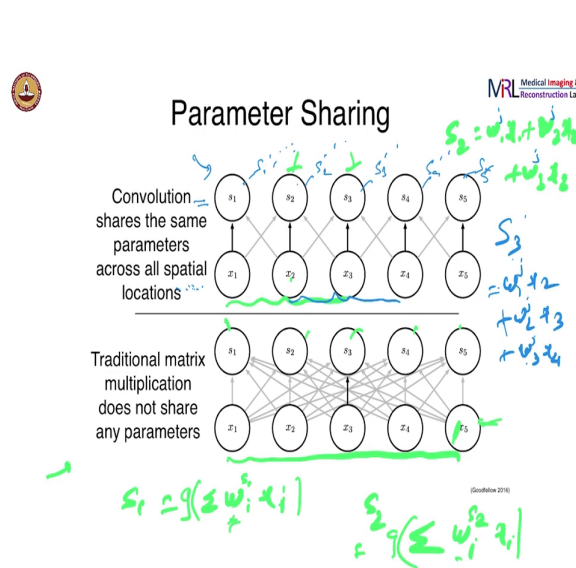


And we will also see one more concept that we keep running into often is this receptive field. So, since we are only connecting the output neuron or output unit to a very small subset of the input, so if you, for instance, if you see let us consider this h_3 for instance, h_3 is connected to x_2, x_3 and x_4 , so these three. So, this is the receptive field, this is the receptive field. Now, for h_3 , this is the receptive field.

Now, if you consider a fully connected network, then for every output unit the entire x_1 to x_5 is the receptive field. But let us now go one more layer deep and we will do the same kind of connections. You will see that for instance g_3 , g_3 is receptive field if we call it that is just again these three. So, now, but if you consider, if you look at h_2, h_3 and h_4 , see h_2 actually does look at x_1 to x_3 , gets its inputs from x_1 to x_3 and h_4 gets inputs from x_3 to x_5 .

So, technically g_3 's receptive field is this entire input, it looks at all of us. This is very useful concept to know from the point of view of image analysis. So, this is 1D examples, might not be so clear. But so, if all the way to understand this, that if you have a convolutional neural network working on an input image, and as you go deeper into the network, every output unit in any one of the layers, let us say, if you go to 5-10 layers, there is actually getting input from a very large piece of the image, a section of the image. So, that is a good point to note.

(Refer Slide Time: 18:43)



The parameter sharing is something I mentioned in passing once again to understand this. So, the usual, if you look at the traditional feed forward neural network, we saw that if you take s_1 , for instance, s_1 gets his input from x_1 to x_5 . So, basically you have a bunch of weights. So, $s_1 = \sum W_i^{s1} X_i$. And, if you look at $s_2 = \sum W_i^{s2} X_i$ that also is based on some, instead of w_i 's, I will just maybe, if I use a subscript here w_i^{s1} .

And s_2 is also a linear combination $s_2 = \sum W_i^{s2} X_i$. Again, I have not shown the non-linearity. So, you would actually have a point wise non-linearity, some g , which I am not mentioning, but just to understand where the inputs come from, so, every one of these output units is a linear combination followed by non-linearity of its input units. And the linear combination is effected by these weight vectors, once again, you can put them into a matrix form also, where each of these weight vectors are unique, this w_i^{s1} 's and w_i^{s2} 's are a different set of weights.

Now, for the convolutional neural networks, they share the same parameters across all spatial locations, which means, so for instance, if you take s_2 , let us say, then its inputs are from x_1 , x_2 and x_3 . So, $s_1 = w_1 \cdot x_1 + w_2 \cdot x_2 + w_3 \cdot x_3$. So, we just write this down here quickly s_1, s_2 is

$w_1 \cdot x_1 + w_2 \cdot x_2 + w_3 \cdot x_3$ Now, if you go to s_3 , we can actually write, just one second, say s_3 .

Now, s_3 in this case can be written as, this we can use the same combination. $s_3 = w_1 \cdot x_2 + w_2 \cdot x_3 + w_3 \cdot x_4$. So, it is the same set of weights, you just translate along this vector to multiply and produce this. Once again, I am ignoring the non-linearity, those things remain the same when you go from a fully connected to convolutional network.

So, the processing is the same, the linear combination followed by non-linearity that process remains the same from feed forward networks convolutional networks, the only thing that we are doing is we saw earlier is that we are doing sparse connections in the sense not, we are not taking all the inputs to produce one output, we are only doing a subsection of the inputs. And then the weight vectors are shared across different subsets of inputs.

So, x_1, x_2, x_3 is multiplied by a set of weights followed by non-linearity to get s_2 . You can use the same set of weights to multiply x_2, x_3, x_4 for the non-linearity to produce s_3 , so on and so forth. So, we can get, we can use these weights again and again over different input units. So, which means that for imaging, or image analysis, this translates to sliding the weight matrix filter kernel that we saw across the same weight matrix across the image to get an output.

We will have a more detailed illustration later, but just for understanding. So, this gives you another, also another angle. So, now, we have used one set of weights w_1, w_2, w_3 to produce this outputs s_1, s_2, s_3, s_4 and s_5 . Of course, we are ignoring the, let us assume that there are some neurons here to the left of x_1 and some neurons, which we will not consider them at this point.




But we are just used one bunch of weights w_1, w_2, w_3 , to get s_1, s_2, s_3, s_4 , and s_5 . Also, possible to do another set of weights. So, if you saw, we can call them, we can use different subscripts. So, let us say we will subscript this as some j . So, we can have j , here the j does not refer to a layer or anything, it is just one set of weights, they0 set of weights.

So, we can have other sets of weights, three weights, which can give rise to another set of activations. So, this can be, that can be another set of activation. So, we can have as many sets of activations or output units by choosing a different set of weights of 3, so we can just keep doing this. So, one thing that you have to do in a convolutional neural network is to determine how many such sets of weights, or my filter kernels that you want to define, but that is a hyper parameter, so called for a convolutional neural network.

So, once again notwithstanding parameter sharing I can have multiple such sets of weights. So, you can say okay, how does this help, you can see that to produce this, so if you think about this, if you are using fully connected neural network, and this is 5×5 , you need 25 weights to produce one set of outputs. But in this case, we just needed three weights to get an entire new output layer, hidden layer with 5 hidden units.

And then in fact, if you keep, if you do 5, 6 sets of such weights, you can get multiple sets of hidden units. So, that is the idea behind parameter sharing. And that is also how you define multiple layers in one layer like you can. So, every layer in a convolutional neural network will have multiple sets of such activations. And that is what you would call a layer in a convolutional neural network.

(Refer Slide Time: 25:17)



CNNs- From Fully connected to convolutional Layers

- A typically fully connected feed forward neural network- All units/element(s) in previous layer are connected to an unit in the current layer
- In a CNN, only a subset of units in the previous layer connect to an unit in the current layer- **Sparse Connections**
- The same set of weights connect unit(s) in the present layer to units in the previous layer- **Weight Sharing**
- The weights are typically visualised as **filter kernels**- a small matrix (2x2,3x3, 5x5 etc) that can be applied to units in the previous layer to generate units in the current layer.
- **Hierarchical Learning**

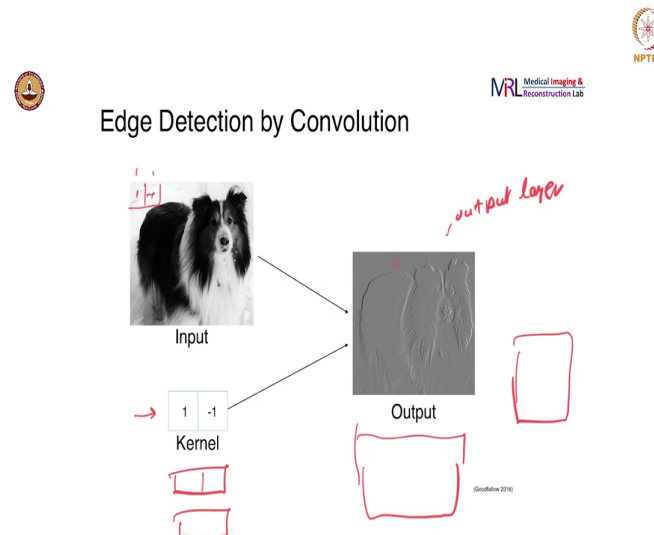
So, just to summarize intermediate summary in the for a feed forward neural network, you have full connections, where every output element is connected to all the input elements. For a CNN only, a subset of inputs is connected to an output. So, that is why you get sparse connections. Another way of looking at it is that your weight matrix there will be lots of 0's. The same set of weights connect a multiple output to different inputs.

So, you can just basically think of it as sliding that weight vector across the input vector to get different outputs. These weights are typically visualized as filter kernels, if you are looking at 2D or multiple dimensional input. So, illustration, so far have all been on a vector of inputs. But if you can think of a 2D input, then these weights can also be looked at like a small matrix or a filter kernel, that you slide over the image to get a particular output, activation map or hidden units.

One aspect of CNN, which we will look at later is that CNN's enable the so-called hierarchical learning wherein you learn features, which when composed give rise to a different set of higher order features. We will look at that later. But this is just something that

comes from the architecture itself, that is basically your, because your outputs are compositions of functions that this gives rise to the hierarchical learning.

(Refer Slide Time: 26:55)



So, if you want to understand what these filter panels do, again, we will have much more detailed example next. But just in the context of an actual input, let us say our input is this dog image, this is our input vector. And this is the first layer we are defining this weight. Once again, you would randomly initialize your weights just like you do for linear regression, as well as for the feed forward neural network.

And in this case, we just fixed the weights to be $(1, -1)$. And if you actually superimpose the weights $(1, -1)$ multiply with the underlying pixel values, add them, add the products, so, you will get one output there, if you do that, what this does is to highlight the edges of the image, you get this edge map. Now, this is your output, this is one output layer. Now, what I said was now you can do multiple sets of such filters.

So, you can have multiple such filter kernels and each one of them will produce a similar output. You stack them all up to get this output layer in a convolutional neural network. The other reason why this example is illuminating is because you see that these filter kernel in this case, you can actually learn this, you can actually learn this edge detector. So, this $(1, -1)$ turns out to be edge detector.

So, similarly, when you train a neural network, especially the earlier layers, you will get these primitive image processing operations done automatically, edge detectors, kernel detectors, blob detectors, etcetera. The compositions of these in the further layers will end up detecting

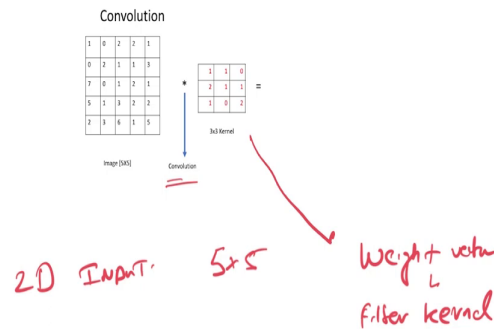
entire, in this case ears, eyes, nose, etcetera. So, that is why it is also referred to as hierarchical learning.

(Refer Slide Time: 28:44)



CNN- Convolution

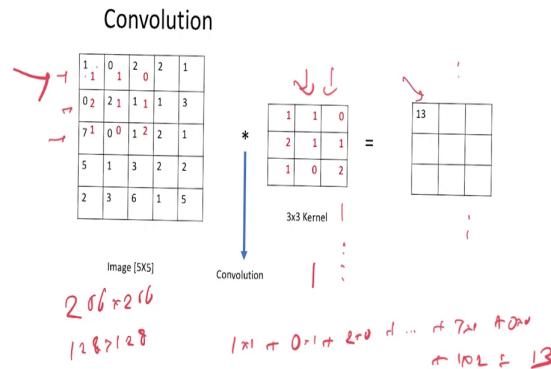
MRI Medical Imaging & Reconstruction Lab



So, we will just illustrate convolution. So, we are going to look at a 2D input, a very small picture of size 5×5 , even the larger picture, let us say 256×256 , 224×224 is the same operation. Our filter weight, in this case, I will say I will call this weight vector, but here, it is actually a weight matrix. I should not call this weight matrix, but it is a small filter, or shape filter kernel.

So, what it does is acts on the input and gives you a particular output. So, this operation, which I am going to describe is referred to as a convolution, but not to be confused with the convolution which is in signal processing literature there. The process is slightly different. So, let us not worry about there, how it is done there. But rather in this context, let us see what convolution means as you go step by step.

(Refer Slide Time: 29:48)



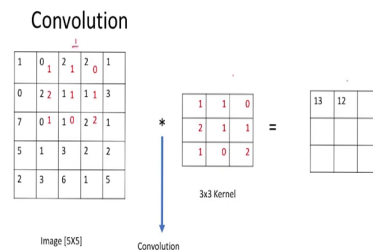
So, what you do is you will superimpose, you will superimpose this kernel on top of the image. And zoom in, maybe it helps you, yes. You will superimpose this kernel on this image. And you would multiply corresponding elements. So, it is 1×1 , just write this down, it is $1 \times 1 + 0 \times 1 + 2 \times 0$. That is the first row. Again, second row computation, so on and so forth.

Third row, plus $7 \times 1 + 0 \times 0 + 1 \times 2$, this is the third row of computation, that will give you 13. So, this is the computation, this is what is the convolution that way to understand convolution that as it happens in a CNN. So, this is your filter kernel or these sets of weights in a feed forward neural network these will be called weights. Here, you just call them as filter kernel.

So, you would do this operation by sliding this filter kernel over the input. In this case our input is small, it is 5×5 . But you typically get much larger input through 256×256 or 128×128 . And you can resize them and so if you find that it takes too much memory. So, for this particular input, you are superimposing the kernel on top of the image starting from the top left corner and multiplying the corresponding underlying elements adding them to get one output here.

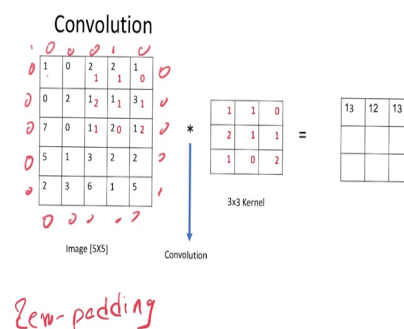
Once again, these are just the weights that you estimate in a feed forward neural network. And we can define multiple such kernel. When I say define you initialize multiple such kernels, you realize that these are what are learned because the weights are learned in while training a neural network. Similarly, for a convolutional neural network, these filter kernels are learned. So, you can have multiple such weights, matrices or weight or filter kernel which operate on a convolution to produce multiple such output feature maps as they are called.

(Refer Slide Time: 32:02)



Similarly, we will walk through this slowly, so you can move this once again, the next. So, what you have done here is translated this filter kernel 1 pixel to the right, did the same operation and got this value.

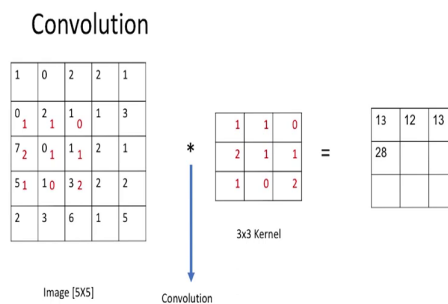
(Refer Slide Time: 32:21)



Similarly, you can do this for all positions. You can move it to the right, for rightmost now we are at the rightmost. Now, here is something that to pay attention to. So, here we are not doing anything special to this image. So, it is not uncommon to do this zero padding. So, what you would do is you would add a bunch of zeros to the boundary of the image or some other value which is meaningful. So, that absolutely you can push the kernel further.

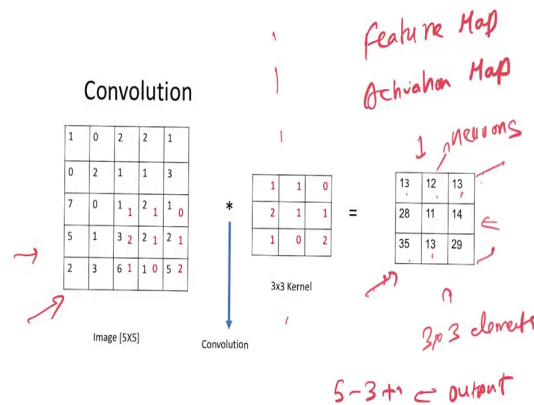
So, typically, the valid convolutions as they are called, you would start from the top left. And you always have the filter kernel completely inside the image. So, if top left, as soon as you hit the point beyond which you cannot slide the filter kernel, you stop, and then you come to the next row. So, but if you do want to keep going on, then you would have to zero pad appropriately so that you can move the filter kernel as far as you want to.

(Refer Slide Time: 33:26)



Similarly, here, if you look, you have now after go hitting this rightmost pixel, you are now moved one step down, put one step down, one pixel down, and then you continue doing the same operation.

(Refer Slide Time: 33:52)



So, you can keep doing this until we get your final output. One thing you have to notice is that this output, you will notice that the number of, so you start off with 5×5 . Now, your output has only 3×3 elements. So, this convolution automatically because you did not pad with zeros, the automatically reduces the number. So, you start off with 5×5 , you will lose in this case, I know you lost two elements when you do the convolution.

So, this basically is where if you start with 5×5 , of course, you are looking at always square inputs and square filter kernels. So, you would basically, just basically $5-3+1$, that is the size of your output. So, the dimensions of your image minus the dimensions of your filter kernel plus 1 that will be the size of the output if you do not 0 pad.

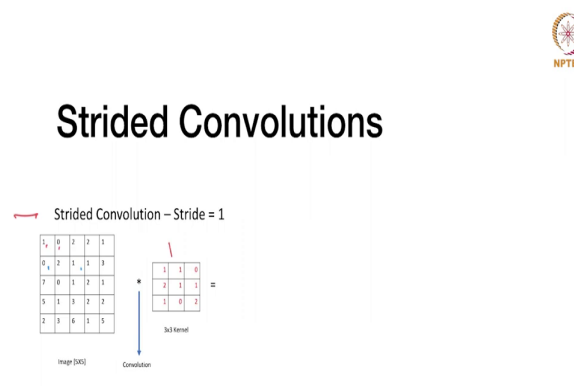
So, if you want to preserve the size, you would have to had a zero to the edges so that and as many zeros as it is required to make sure that the input and output are of the same size. That is a trick that people do in order to preserve the so-called resolution of the output. Now, this output here you would often refer to as a feature map or an activation map, either are fine. Similarly, individual elements here, units here, you would call them neurons or activations, because it is an activation or hidden units which we all of these are acceptable terms.

But typically, you would say this is a feature map, you would call this an input feature map, this as an output feature map. These are all terms which are used interchangeably in the convolutional neural network jargon, but essentially, that is it is just basically our output layer. In a feed forward neural network, these should be laid out like a vector.

But here, they are still retain the grid structure of the input. The idea here is to retain the grid structure of the input, because grid structure like we discussed earlier had has some meaning. Pixels nearby, possibly belong to the same object, if you are talking about an image, and probably share the same properties. So, that spatial location information has to be maintained. And one way to maintain that is to put them on a grid.

So, the intermediate layers or in the form of grids also. And you can have, you can define as many filter kernels as you want. And correspondingly, as many output maps will be generated.

(Refer Slide Time: 36:54)

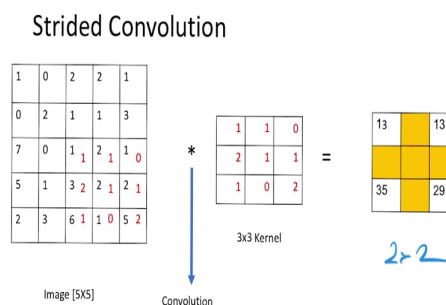
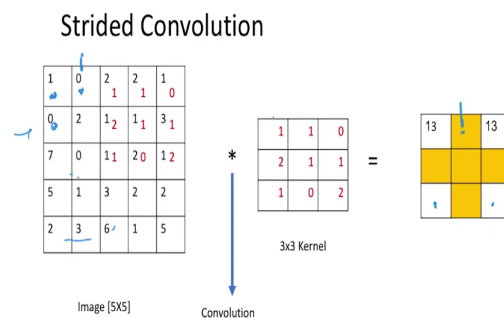


So, if one way of, another way of doing this to reduce computations in successive layers, because you eventually want one output if you are doing regression, or let us say a vector of 3 or 4 outputs depending on if you have usually the k outputs if you are doing classification. But if you look at the way we have been computing, it seemed like it will take forever to get to that point, because you have fairly large images 256×256 .

And even if you, and typically you would 3×3 or 5×5 kernels, because larger kernel means that you will need more computations, then at that rate at which you are doing like, you are losing about two, size of two on either dimension, as you do convolutions without zero padding. It will take a lot of layers of computations to get to the output, and you would, and without increasing number of computations, one way to do that is to do a so-called strided convolution.

So, the previous one, we were sliding the filter kernel one pixel at a time, here, you would stride them two pixels at a time. So, you would skip a pixel, you skip a pixel, and that is a stride of one. And of course, you can skip as many pixels as you want. And that will once again give a corresponding output. So, let us do this that way.

(Refer Slide Time: 38:19)

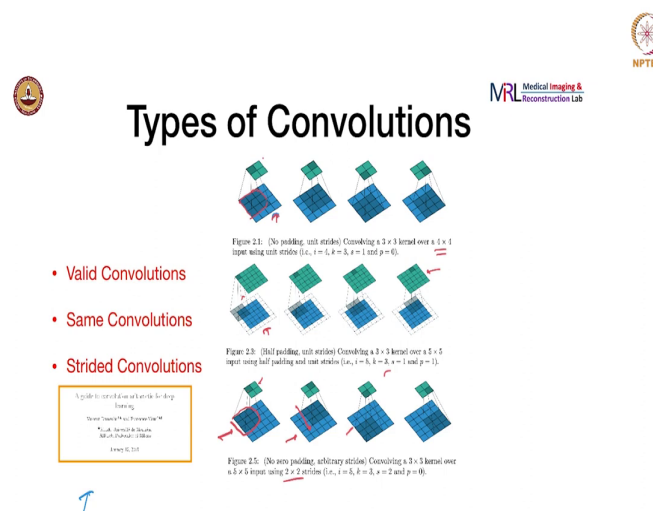


You will see that, in this case, the outputs, you can do the same, if you stride one, you have missed out on instead of 3×3 , you will end up getting a 2×2 output, because you are left alignment was with 1, let me show you zoomed in. So, you skipped this pixel, you skipped positioning the top left corner of your filter kernel at this pixel, but rather skip this and move here. So, which means that in your output feature map, this particular pixel is skipped.

Similarly, when you go move down, you will skip this row and then superimposed from here, and then we will move to the right, you will again once again skip one column. So, if you do that, you are finally you are output. And keep that again, I urge you to actually do this by hand you can compute and verify yourself is that you will get this output.

So, you will get 2×2 output. So, your stride and it also reduces the size of your output feature map. So, that is one way to of course, quickly, subsample your inputs so that your computations are not huge. So, that the weight and this was one way of doing that. So, this is the first operation in a convolutional neural network, the convolutional operation that has a lot of these variations.

(Refer Slide Time: 39:47)



There are different types of convolutions. That is one we have, I have just showed you those. And this is again, taken from this particular publication. It is a very, actually very informative publication, I urge you to all to go through this. So, there are three types that we looked at in summary, we looked at the so-called valid convolutions, which is when you would superimpose a filter kernel on top of the feature map or the input feature map, slide it one pixel across a time in the horizontal and the vertical directions, till you no longer can strike because you will go out of the picture.

And every time you would multiply the underlying elements, take a summation, do the non-linearity, get an output. So, in this case, the light blue, use a different color, light blue is the input feature map superimposed filter kernel is shown here, it is a 3×3 . And in every


location, you would, again form the output, the green is the output feature map. So, for a 3×3 kernel striding over a 4×4 input using unit strides.

So, if you read this paper, he has a very nice nomenclature developed, I am not going to explain this here. So, I would like for you to go read it, it is not too hard to understand. Similarly, if you have padding, if you do a padding of one is what is shown here, pad with one set, we can pad with zeros, or just replicate the broader values. And if you would stride similarly, as you did, then the output feature map, you see that output, the output size is maintained.

So, you had a 5×5 , input, now if you pad one which means that you pad one on either side, so that you should do padding and that means that your output size is maintained. But if you do not do zero padding, but then if you do a stride 2×2 stride then your output ends up being 2×2 . So, 2×2 , not 2×2 , stride of 2, which means that you actually skipped a pixel.

So, that is what if you see but from this image to that image, so you have initially superimposed it here. I did you do the usual computation to get this rain feature map, but then you kind of skipped this column. You can do that once again, for this particular output feature map, you keep this row. So, this way you can actually subsample your feature map, there is one way of doing this, strided convolutions.

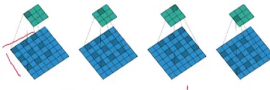
(Refer Slide Time: 42:37)



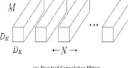
Type of Convolutions

- Dilated Convolutions
- Depthwise Convolutions
- Depthwise Separable Convolutions

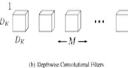
Figure 5.1: (Dilated convolution) Convolve a 3×3 kernel over a 7×7 input with a dilation factor of 2 (i.e., $i = 7$, $k = 3$, $d = 2$, $s = 1$ and $p = 0$).



(i) Standard Convolution Filters



(ii) 1 x 1 Convolutional Filter, called Factorized Convolution or Depthwise Separable Convolution



(i) Depthwise Convolutional Filters

Mishkin, D. Efficient Convolutional Neural Networks for Mobile Vision Applications

Authors: G. Simen, Minghui Du, Rui Chen, Shengyu Chen
Editor: Wang, Editor: Wang, Editor: Wang, Editor: Wang, Editor: Wang

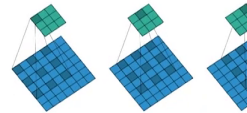
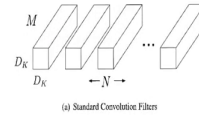
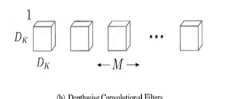


Figure 5.1: (Dilated convolution) Convolution with a dilation factor of 2 (i.e., $i = 7$, $k = 3$, $d = 2$)

- Dilated Convolutions
- Depthwise Convolutions
- Depthwise Separable Convolutions



(a) Standard Convolution Filters



(b) Depthwise Convolutional Filters

MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications
 Andrew G. Howard, Mengde Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias S. Sengul, Hartwig Adam, Google Inc.

Type of Convolutions

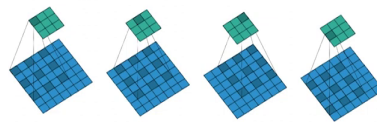
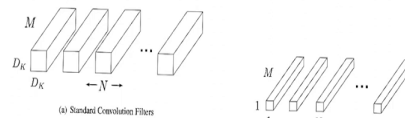


Figure 5.1: (Dilated convolution) Convolution with a dilation factor of 2 (i.e., $i = 7$, $k = 3$, $d = 2$, $s = 1$ and $p = 0$).

olutions
 nvolution
 le Convolutions



(a) Standard Convolution Filters



(b) Depthwise Convolutional Filters

works for Mobile Vision
 Dmitry Kalenichenko
 Hartwig Adam

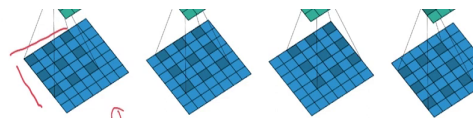
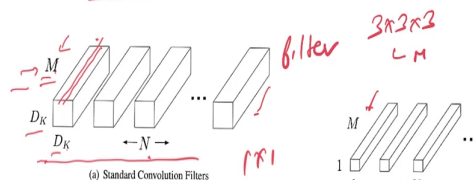
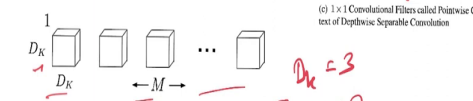


Figure 5.1: (Dilated convolution) Convolution with a dilation factor of 2 (i.e., $i = 7$, $k = 3$, $d = 2$, $s = 1$ and $p = 0$).

IS
 lutions



(a) Standard Convolution Filters



(b) Depthwise Convolutional Filters

(c) 1×1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

$D_K = 3$
 $(3 \times 3 \times 1)$
 $3 \times 3 \times 128$

/vision
 00



So, we look at now two different sets of convolutions, types of convolutions once again. So, these are the dilated depth wise and depth by separable convolutions, some of these images are taken from this publication MobileNets. Just zoom in just to see. Now, if you see this, first one is the dilated convolution. Wherein so you are convolving a 3×3 filter kernel over a 7×7 input with a dilation factor of 2, which means that you are taking the filter kernel and inserting 0 rows and 0 columns.

So, that way, if you look at it for the same 3×3 filter, we get a larger receptive field. That is one of the other concepts that we looked at initially, and I illustrated it with a 2D, sorry with a 1D example. So, in this case, it is a 2D example, you can see that if you use a 3×3 kernel, you have a 3×3 neighborhood that each output is looking at a 3×3 neighborhood in the image.

On the other hand, when you do these dilated convolutions, then also you are inserting the zeros in here. So, that your filter kernel actually operates for a slightly larger region. Now here, what is the assumption, the assumption here is that since these are very small 3×3 , 5×5 regions, you would expect that in the image, these regions are fairly uniform.

So, when you introduce these zeros in the filter kernel, you are actually not missing out much instead there is no pixel to pixel strong fluctuations in the image in the input feature map. So, when you do that, and you get for the same thing as three, you get a slightly larger receptive field.

And again, here all the other routes apply you can stride more or you can stride less and you can zero pad, all of these are possible, but the advantage is over several layers if the third or the fourth layer your output neurons or output activation maps are looking at a much larger field in the input feature maps, several layers down, and this is one trick that has been used very successfully in many cases.

The other types of convolution we are going to look at are the depthwise convolutions and the depthwise separable convolutions. These images are again, once again dimension are taken from this MobileNets paper. So, let me just zoom in a bit just for you to see clearly the paper as well as the pictures that were taken from them. So, if you look at a standard convolution filter, D_K , let me just red this D_K , $D_K \times D_K \times M$.

So, where does this M come from? And there are N such filters. So, each of them is a filter. Now, you might be wondering where that comes from. So, like I mentioned earlier for every

layer, you can define as many filters as you want, filter kernels as you want. And each of these filter kernels will give rise to an output feature map, or output layer or output feature map, output activation map.

Now, you can stack them all up. So, if you define M such filter kernels, then your output will have M such feature map. So, in general, if you say for convolutional neural network, your input is actually a cube, basically, you can have as many feature maps in your input set. And each of them has a certain size.

So, in this, so, if there are M feature maps in your input, and if you actually define your filter kernel to be D_K , $D_K \times D_K$, which means that it is actual sizes $D_K \times D_K \times M$. So, the filter itself actually extends across the length, across the channel dimension. So, if you stack up the feature maps, the dimensions along which we stack them up is called the channel dimension.

So, there is, though as the number of feature maps increase, that means that you have more and more channels. So, to give you an example, RGB image has three channels. So, on an input RGB image, so, for instance, RGB are three channels which means when you say you have 3×3 filters, you actually mean that you have a $3 \times 3 \times 3$ filter for an RGB image.

So, let us say you define M such filters, then you will have M feature maps on your output. And which means that in subsequent layer, when you are actually operating on those feature maps, your filter would be let us say, if you D_K is 3, then your actual filter size is $3 \times 3 \times M$, so this is your filter size.

That is what happens in a standard convolutional neural network is that your filter has a depth which is equal to the number of channels in your input. So, for instance, like I said RGB, RGB image as input your filter actually has a channel dimension of 3. So, when you say 3×3 filter, which means $2 \times 3 \times 3$ filter. Similarly, here in this case M channels your filter has size $3 \times 3 \times M$.

And you would add, so that means the number of computations actually increase. If you think about it. Let us say you define, you have defined 128 filters in your previous layer, then you have a current layer as input you have 128 feature maps, so you have your filter size is actually $3 \times 3 \times 128$. So, if you do 256 such filters that is a lot of computation.

So, to decrease the computation, to make computations more efficient, of course, this paper shows that why they are efficient, you would actually do this so called depthwise

convolutions because you operate on each one of these activation maps like shown here in the pictures by using just a simple 3×3 . So, you will define M , 3×3 filters.

And you just operate them across. So, for each one channel, each one of the input channels, you will have one filter. And your output, so if you have M filters, you will have M channels as output, that is what is shown here. So, this is one form of reducing your computational burden. So, you will have M outputs. So, and if you want to combine information across channels, then you would do this 1×1 filters N of them.

So, because you want to get the same dimension. So, basically the standard convolution filter, if you want to get N feature maps as output, you will define N filter kernels each of size $D_K \times D_K \times M$, so that your output has N feature maps. So, then in this particular formulation, where you do depthwise convolution filters, what you would do is, since you know that there are M feature maps in your input.

So, you will define M filter kernels, but these are your standard size 3×3 , let us say D_K is 3, you get only 3×3 . And then when you operate on them, each of these M feature maps you will get M output feature maps corresponding to each one of them. And of course, you will stack them all up and you do this so-called 1×1 convolution.

So, 1×1 convolution is basically the same operations as we saw with the 3×3 , except that you just have a vector, that is all. So, because it is of size $1 \times M$. So, you will operate that across the image, across the channels. But of course, you can define N such 1×1 filters and you get n outputs.

So, what we are trying to do is we are trying to operate mix the information or take nonlinear combinations of the pixel, voxel values in plane using these regular convolutions. And then across channels you want to take nonlinear combination using this 1×1 convolutions. So, do not get flustered by this 1×1 convolution because the same as 3×3 , except that your filter size is 1, that is it, you have a vector. So, if you have M channels to work with then your filter size is $1 \times 1 \times M$. The computations remained the same.