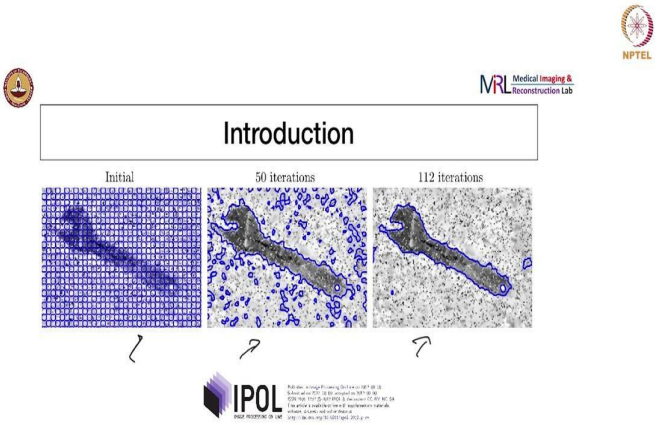
Medical Image Analysis
Professor Ganapathy Krishnamurthi
Department of Engineering Design
Indian Institute of Technology Madras
Lecture 47

Semantic Segmentation

Hello, and welcome back. So, we will now proceed to talk about semantic segmentation using deep neural networks for this week, and that will be the last of the lectures for this week. And for next week, we will talk about a generative model.

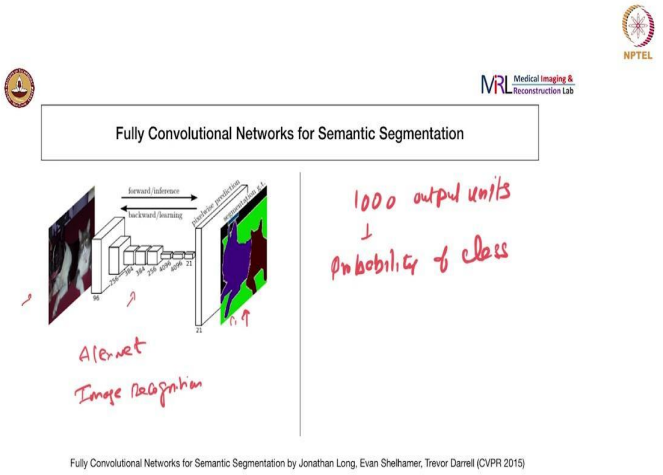
(Refer Slide Time: 00:33)



So we have all seen this kind of segmentation that we introduced in one of the earlier weeks, where we used a contour to initialize and then the contour slowly converged around the object of interest. So very clearly, there was a foreground and background object as seen here, where lots of little circles have been initialized, and over time, they all converge to the object of interest.

So this is the kind of segmentation that we have all worked with. So with deep neural networks, the segmentation takes out, the process is different. And that is, and we will look at what that is.

(Refer Slide Time: 01:17)

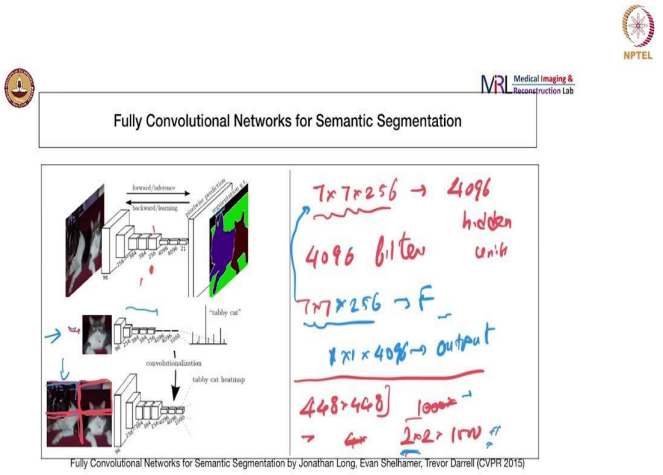


So let us consider this, Alex net architecture is typically used for image recognition. So one of the earliest papers to do semantic segmentation, which is basically labeling every pixel in the input image as belonging to a set of classes, one class or the other. So we know that the image recognition networks, just like the Alex net over here, typically give you one, one output per class, which means that, let me clarify what I mean.

What I am saying is that, if you have, if you are trying to classify an object as belonging to one of those classes, then the output layer typically has 1000 output units. Each of them is a value, which corresponds to the probability of belonging to that class. But what we want instead is this map from here, which is pretty much the same size as the input and with every pixel, given a class label or probability of belonging to a class.

And that is, if there are 3 classes, then you will have a 3 such images in the output with each image corresponding to probability of a certain class. So we will explain that more of that in the later slides. So the idea is now to take a object recognition based network and use it for image segmentation, where then every pixel of the input limit is labeled as belonging to one class, or the other was the number of classes being provided, depending on the data.

(Refer Slide Time: 03:12)



So how does that work? So let us take a look at this. Alex net architecture, so we know that for instance, this 256 Right, so basically, this layer here, the beginning Alex net is of size 7×7 ×256 And it is fully connected to the next layer, which has 4096 hidden units. The idea up here, to get to fully convolutional network is to convert all the fully connected layers into convolutional layers.

So how we can convert fully connected layers convolutional layers, so in this case, if you take this layer which has it is actually has 256 feature maps, each feature map is of size 7×7. So then if we define 4096 filters, each of size 7×7, but then they will automatically, run across the channel dimension.

So, each filter is actually of size 7×7×256. And if we have 4096 filters, then the output of this is the filter size. If this filter acts on this input, then the output is of size 1×1. So your output would be 1×1×4096 this is your output. So then, so while that we have done this, what is the advantage of doing this. So we know that for in the case of a image recognition, the input size is approximately 224+224.

Now if that input goes through this typical Alex net, the output is a vector of size 1000. And each element belongs, indicating the probability of belonging to that class. Now, if we take this, let us say this consider this input image, which I am going to say is twice the size of this image, twice the size of the image that you typically give input to this image recognition network.

So you would have instead of 224, if this was 224+224, your input would then be 448×448. Now, if we have converted the network to fully convolutional nets, so fully convolutional layers, then instead of 496×496×1, since I have indicated, means I have doubled the size of the input, then your output, eventually, if we use these convolutions properly, then output will be 1000 cross I should say, right another way 2×2×1000.

So the output would be maps of size 2×2000 of them. And these 2×2 indicate, so if we can split, so I am going to do this very crudely is a 2×2 feature map and the output, there are 1000's such feature maps, and each of these feature maps would indicate the probability of a particular class, let us say cat or dog.

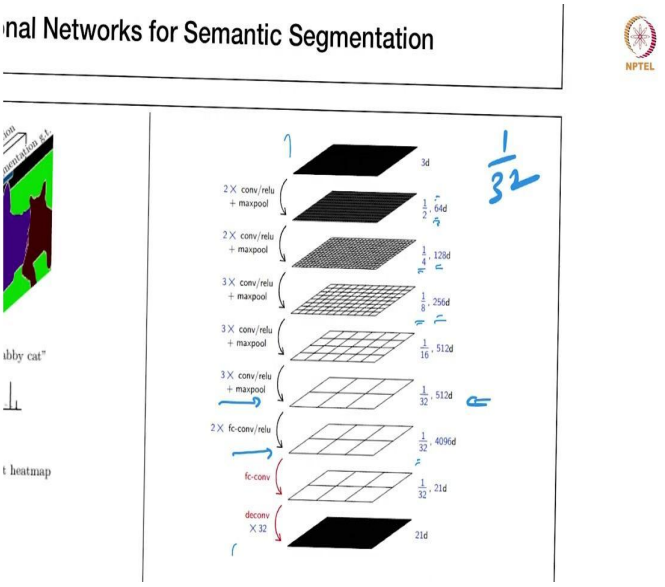
So very easy way of understanding, where this is 2×2 come from is that we split this input into 2×2, then the probability of this so if we have, let us say, 1000, there is the first 2, cat and dog, as you get the first feature 2×2 feature map, it will say whether this big super pixel block here is cat or dog probability, one should check cat or dog probability, let us say the first feature map is cat.

Then this particular block, what is the probability of it being cat, this particular block? What is the probability of that being cat, so on and so forth? And if you go to the second feature map, let us say it is corresponding the class dog, then it will do the same thing except that now the probability values would indicate that of being a dog. Now, one clear problem we have seen is that, this is a very coarse feature.

Coarse feature map, so but we would like, like the probability for each pixel as belonging to a particular class, which is what gives you the segmentation map, correct. So then the way to do that would be to up sample this. So this up sample process is often mistakenly called dconvolution, of course, but the better thing would be to call it transposed convolutions. Or strided convolutions or fractionally stated convolutions.

Transpose convolutions would make sense, we will see what those are over the next few slides, but what you do is basically you take this 2×2 and up sample it to the size of the image in order to get a fairly high resolution map.

(Refer Slide Time: 08:25)



So how would you go about doing that? So I am just giving you in this paper what they did. So they had the crude, this rule of thumb. So if you look at the Alex, Alex net, if you have an input of size 224×224, you would, if you come to this 256, this size would become, after a few layer this becomes, this size is 7×7, there are, of course, 256 channel, and this is 224× 224, let us say. So, there is a factor of 32 reduction, so it becomes $\frac{1}{32}$.

So that is typically what this indicated here. So the idea is, if you have a input of a certain size, by the time it comes around here, or let us say even in this particular layer, you have it is been down sampled by a factor of 32. So, this is a typical architecture. So, these numbers tell you the size, so let me zoom in a little bit, to show you. So, this is 3 dimensional, 3d is 3 channels of certain size.

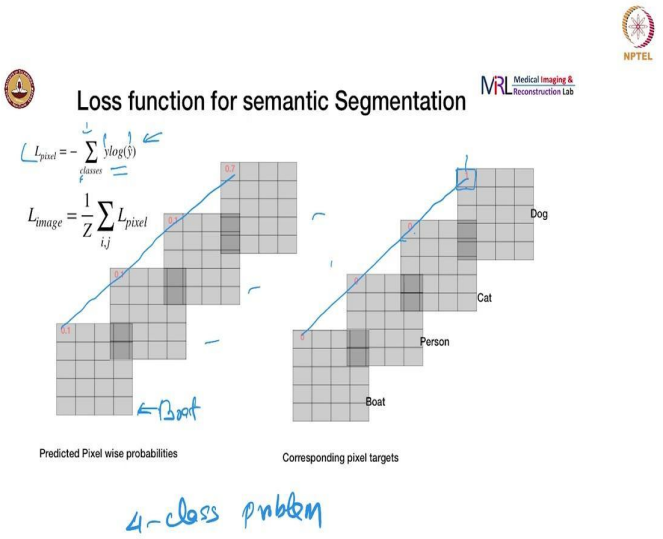
And then, if you do 2 convolution layers, non-linearity followed by max pooling, you would end up with one half the size sixth and of course, you can have as many outputs as there are filters, which in this case is 64. Similarly, you do another set of convolutions max pooling, you do lose another factor of 2 up sampling so your size is one forth, then you double the number of feature maps.

Similarly, one half the size again, double the number of feature maps, you do that till you get to $\frac{1}{32}$. So, this is the typical loss in the resolution size as you go through the neural network, a convolutional neural network from the input to the output side. But this paper came out in

2015 so they were using some of the best performing networks on the image recognition challenge which was basically Alex net, VGG etc.

And the all of them had this typical format wherein if you come to, after a certain number of layers towards the end of the, towards the output layer your size is about $\frac{1}{32}$. So, then the idea is how do we get very high quality up sampled segmentation maps?

(Refer Slide Time: 10:53)



So, before we go any further, but I want to show you, what would the loss function, how would it calculate the loss function? The loss function is calculated per pixel. So, you would do this, the equivalent of this binary cross entropy which is ylog(y) or category cross entropy if you want to call it and this is computed per pixel. So, how would you interpret it? So, your ground truth will tell you what category the particular pixel is.

So your ground truth will have the, let us suppose you are working on a, in this case a 4 class problem, 4 classes both person cat and dog, so let us see your ground truth, let us say the object you are looking for in that image of the 4 classes into the pixels corresponding to, you write down the pixels corresponding to dog.

So let us say this, I am just going to take the example of one pixel. This we know for sure is dog, the others are, if that is dog then it cannot be anything else. So cat is 0 person is… this is the ground truth with one the label one correspond to this pixel here says that that pixel

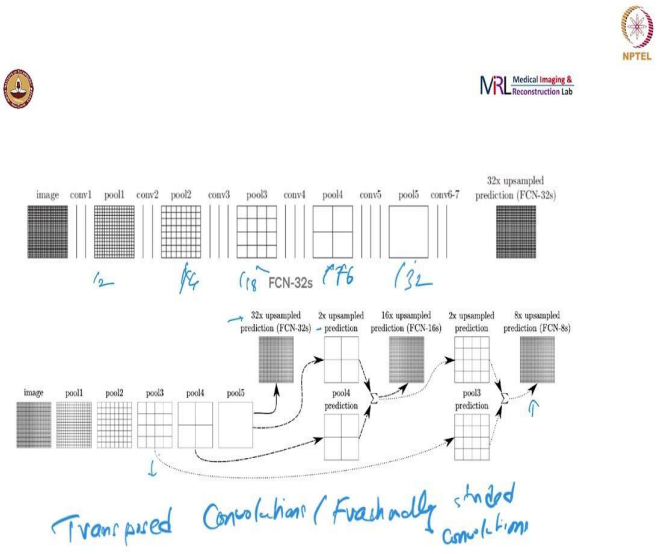
belongs to dog. Your output like I said, will have as many feature maps as there are classes. So, in this case, there are 4 classes there are 4 feature maps.

And each feature map, the pixel values in each pixel map, each feature map is the probability of that corresponding class. So, we say this is a boat class. So, each of the pixel values will tell you the probability of that pixel belonging to boat. Similarly, for the person, cat and dog. So, you would use these to calculate your pixel wise loss functions. that is typically what you do.

And of course, this summation is over the classes, for every pixel you will have the categorical cross entropy over the classes and you would it totally and you will take the average loss over the entire image. So, but this is how we would interpret the output. You will have as many feature maps as there are labels classes and the feature map the pixel values correspond to the probability of that pixel belonging to that class.

So, then how would you do this, sum over classes. So, if you look at this pixel, then you take this out as a vector of output and this would be a vector of your targets and you would use that to compute this pixel, pixel wise categorical percent. You have that for every pixel, you would do that for all the pixels and then you would sum over for the entire image. So, y here is a 4 vector, $\hat{y}$ here is also a 4. So, this is how you would calculate the loss function for semantic segmentation.

(Refer Slide Time: 13:58)



So, now that your converted all the fully connected layers to convolutional layer, so all of them are convolutional layers now, and whatever we saw that even with that architecture, the final layer, the output feature maps are very coarse in the sense, it is a factor of 32, size reduction is a factor of 32 because of the sequence of convolution and pooling layers. So, one way that this problem was solved in that paper that we, I was referring to is by using the intermediate layers also to provide predictions.

So, for instance, pooling 5 layer after up sampling will, can be used to provide an 32x up sample prediction. Now, you can take the pooling 5 layer, do a 2x up sample, up sampling, you take the pooling 4 layer, of course, once you 2x up sample, it pool 4, We are about the same size as pool 5 once you apply sample this by 2x because every time this pooling layer is just a factor of 2 reduction in size.

So, then between 2 successive pooling layers you have a factor of 2 up sampling. So, pool 5 and pool 4 you can combine and then you can 16x up sample to provide a prediction. And of course, you can keep doing that, you can sum them and use that. Similarly, you can take that summation, up sample it one more time take the pooling 3, third pooling layer combine that and provide one more up sample prediction.

So, basically, I mean of course, I urge you to go to the paper as well as follow this up samplings properly, but the idea is you take intermediate outputs from the network and use that to provide high resolution information in the decision layers. So that you get a decent high resolution segmentation map, so this is one of the earlier techniques for fully convolutional networks.

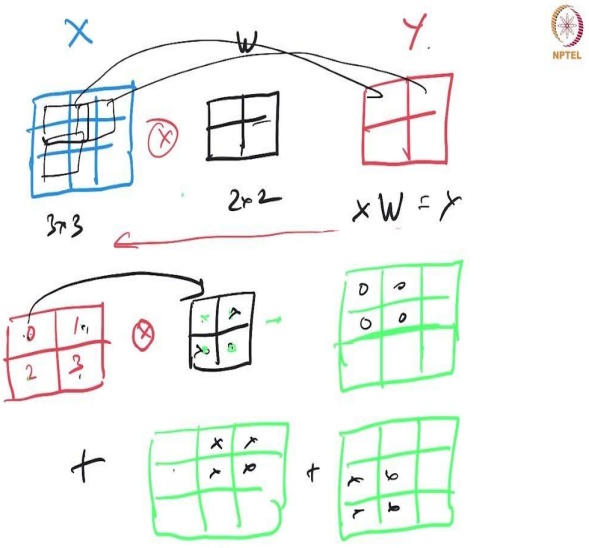
Now, when I say up sampling, I am taking this pool 5 layer or pool, let us say here and say this is factor of 2, this is a factor of 4, let us say this is a factor of 8 and 16 factor then here is factor of 32 reduction in size, how do we up sample it? So, what is this up sampling? How is this done? Now, this up sampling is accomplished by what we call transposed convolutions.

So, why is it called a transpose convolution? We will look at it briefly. Again it is also sometimes referred to us dconvolution which is which is technically not right, because dconvolution is used in a signal passing literature in a different sense, so let us not use that but of course in the deep learning community dconvolution filters are I guess acceptable, but transpose convolutions or fractionally strided convolutions are used.

And there is 2 ways to do this, one is you can just do some high resolution up sampling, interpolation. You can do spline interpolation, cubic spline interpolation all that, B spline interpolation all that and that you do not have to learn the parameters. However, if you do this transposed convolution etc, you will learn them, subsequently they will be slightly better than just using, layer splines or layer interpolation or anything of that sort.

So what why is it called transposed convolution etc, we will see. So we are going to be looking at these transposed convolutions, how will you get the name from and how we do that?

(Refer Slide Time: 17:53)



So let us say your input feature map is of size 3×3, is the input feature map call it X, and then you have your filter kernel, which is your weight matrix W, which is of size 2. So this is 3×3, 2×2, your output will be 2 crossed, this is your Y. So, if you do a cross correlation of X with W, X, W will give you Y as the output.

So the idea is now is we want to up sample, so what do we have to do? What is this operation which when applied in this direction will give rise to the X? So the way you will do this one is you will superimpose this matrix here. And that will give you one, one. And then of course, you can do this every time, you with other one. Similarly, you will superimpose it here, give you the third and fourth, so on.

The idea of the strided convolutions works slightly differently. So what you would do is, you would wait. So let us say we take this Y. This is typically what we do? And we want to turn

this into 3×3. So you will take every element here, let us say I have 0, 1, 2, 3. And so your strided convolution I will also denote that by, your transpose convolution also denote like with this symbol.

With let us say the same kind of 2×2 filter would be to multiply this every element of this input with the filter. So you want to wait the filter with this element that will give you in this case, let me redraw this, your output will look like this. So you will actually end up with a 3×3. This is not a good diagram, so let me draw this for one element.

So I am not going to give you the elements itself, but your output in this case will turn out to be 3×3, if you do it, the way to do that is we have this 3×3 output, the way you would construct it is we will take this element and the input, multiply each one of the elements here with it. So then, of course, this will all be zeros, so then you will get, in this case, you will get 4 zeros. And then plus, you make one more same size.

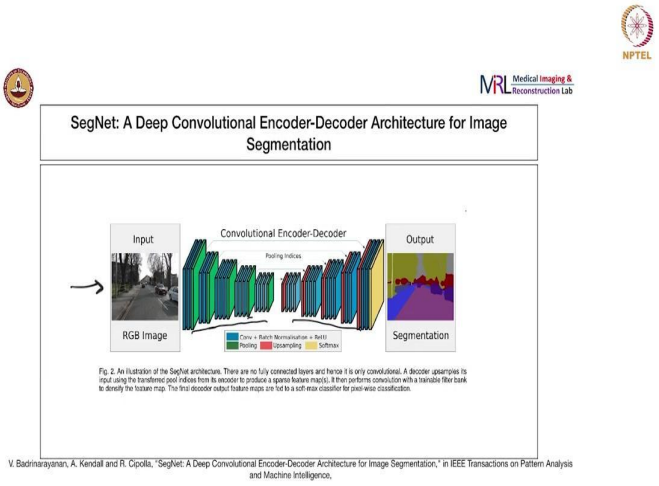
Now we are going to be multiplying with one with each one of these elements. So, but you will put it here, the output here, the one multiply with this one multiply with that one multiply with this and that that goes directly here. Similarly, you will have do just one more, just to show what is happening. And you have this here if you just draw for instance. So you take 2, multiply this element, this element, this and this, all these 4, so you will get 4 numbers here.

Similarly, you take 3 and do and then you and then you would add them all up, see that there are overlap regions where you just sum them up, overlap regions where you sum them up. So this is with a stride of 1. If we increase the stride, then you will get a slightly larger output feature. Here is 3×3, we get 4×4. So you can now one way of looking at is why is it called a transpose convolution.

So if you can think of the up going using X and W to get Y, think of it as a matrix multiplication, vector matrix multiplication, so X is a vector, W is a weight matrix, you can rewrite this 2×2 a kernel in the form of weight matrix, so that you can operate it on X, which you will treat like a vector, then your output is Y.

Now, of course, when you go the other way, that is you are going in this direction from Y towards X, then it is basically equivalent to multiplying your input Y with a W^T. So you can that is one way of understanding what is going on internal? So that is why it is called transpose convolution, but however the actual computation are done this way.

(Refer Slide Time: 22:51)



So, having said that, there are of course many other ways of getting your high resolution output segmentation map. So, one other technique, which is uses the Encoder Decoder Architecture. So, again, all of these have their origins in the computer vision community. So all the inputs are typically the grayscale images, 2 degrees scale images, or RGB images with 3 channels. So it is very similar to a convolutional network, you have an input, you have a bunch of convolutional layers.

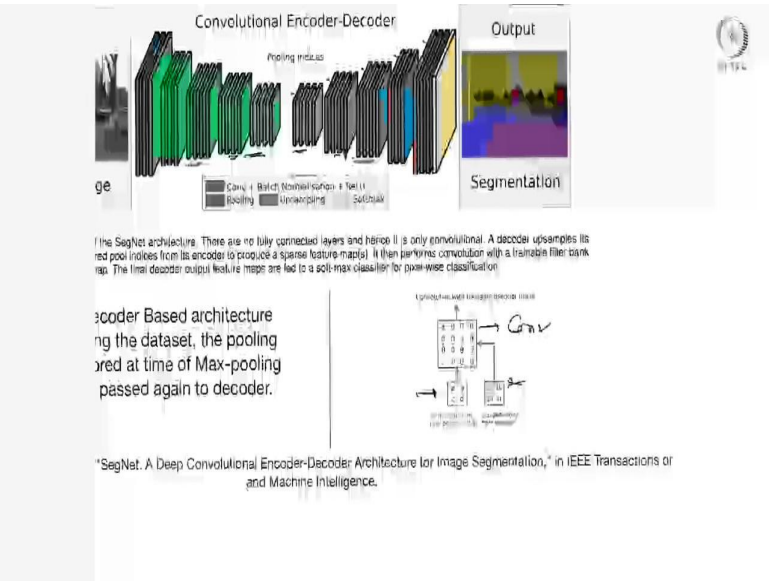
And of course, convolution followed by max pooling, but every time you do a max pooling, you would remember the indices which were, which survived like in sense right, if you have 2×2 max pooling, you would take only a maximum out of that. So, other 3 elements have gone, but you will remember the indices of the one, index of the one element that you kept.

So, as you go, so as you go from left to right, this piece is called the encoder, this is called a decoder, what the encoder does is do a succession of convolution and max pooling operators till it comes to a certain size and there is no, after this point what you do is you would transfer the, you would start doing the up sampling, the way you would do the up sampling is you would since you are, you will try to mirror the encoder side.

So on the decoder side, you will initialize your feature map with the inputs with the larger size feature map, and it will only populate those locations from which you acquired the max pooling outputs. So let us see what that means? And then you do the convolution. And for

every stage, every time you want to increase the resolution, you would once again populate that larger size feature map based on the indices that you had from the encoding layer.

(Refer Slide Time: 24:47)



So let us look in slightly more detail. So, are there are no fully convoluted layers again, connected layers, all of them are convolutional. A decoder lab sample the input using the transferred pixel indices, transferred pool indices from its encoder producer sparse feature map, then you would perform convolutions with a trainable filter bank to make it more dense.

So it is only remember only copying the indices not the value. So let us say, for instance, here, here it is, I will just zoom in. So for instance, let us, if you look at this particular output, this is the decoder size output, the decoder, so decoder output on the decoder size, the feature map first feature map we get is ABCD.

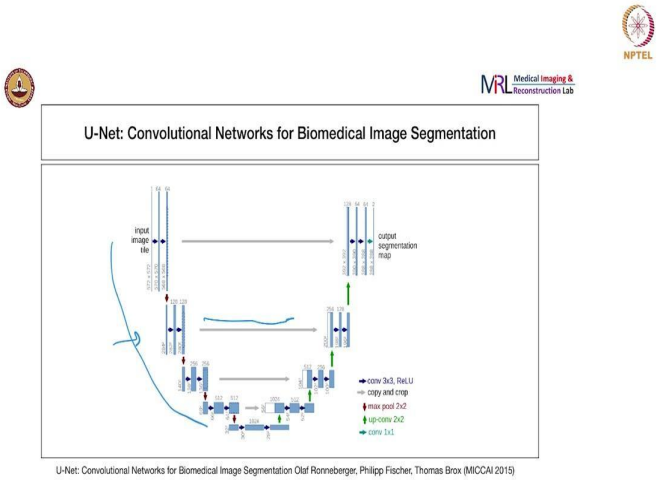
So now you want to map to a slightly higher filter panel. So from the corresponding stage in the let us say that you are here, and you want to go to this size. So you would go to this stage, find out the indices, from which the max pooling operator took the values to get to the lower size feature maps. And then you would put these elements into those areas into those indices.

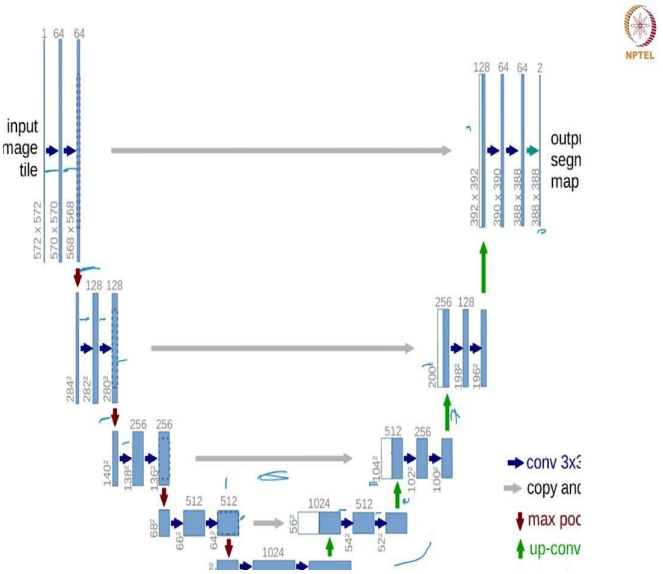
And then now you do convolutions on this, this is typical 3×3, 4×4 convolutions depending on the size of your filter columns. So you would do as many convolutions as you would like. And then once again, you would, let us say, from here to there, if you want to go from here to there, then let us say you will take the inputs, they take this feature map, find the values here,

of course, you have a list of values there. And since you want to go to higher resolution feature map here, let us use a different color.

So high resolution feature map here, you would go back to the same level there. Find out which indices were taken to get to the lower resolution feature map, and then populate the values, they are only in those indices, and then continue to do convolutions. That is a typical way that is one of the encoder decoder networks that was involved for some time.

(Refer Slide Time: 27:06)





Of course, when it comes to biomedical image segmentation, the most commonly used network architecture is the U-net architecture. So, this is again, like a workhorse of biomedical image segmentation, pretty much if you use, any kind of architecture is

accommodated here. So we saw a bunch of new architectures that were for image classification, right object detection, etc.

So Alex net, VGG net, we saw Res net when then dense net, so all of these can be accommodated in these layers. As long as you can have a an game this also has an encoding and decoding layer, except the one advantage it has is it can it also have the skip connection, so called integrated by the gray arrows, which will transfer the feature maps and concatenate them indeed the decoder site. So that is what gives rise to that U-net architecture.

So I will just follow briefly through a one, the encoder side, a couple of maps, and a couple of map of a decoder side just to give you an idea of how this works. So here we will zoom in a bit just to see clearly, so, I just so that you can see one or 2 of these. So, this was used for first microscopy images, so this image is slightly larger than what you might have seen. So, the first image, of course, this is a grayscale images or you can also use RGB images as this filter columns and change to accommodate all the channels.

So, the input image tile is 572×572. Then you use 3×3 convolutions if you like, that will give you valid conclusions that is why you have a decrease in the size. So, 572-3+1 will give you 570. So, a couple of such things, and then this dotted region is basically you would crop it, you would crop it, and that crop is what is concatenated there, but we will not worry about that.

So, after a couple of convolutional layers where you have 64 filters each, you have a down sampling layer, this down sampling is typically by a factor of 2. And then of course, a succession, so this is the down sampling here. And then we have 2 successful 3×3 convolution 64 filters each. Once again, you would have, after you down sample you have a couple of 3×3 convolution 128 filters.

The same sequence as you have seen before, you would have increased the number of filters, but the size of your feature maps would decrease, by the down sampling every time. So once you do that, you will come to this part, where the U happens? Just move that. So here, you have once again, the red arrows indicate down sampling, right?

So every time you have a red arrows it is down sampling by a factor of 2, and the blue arrows indicates convolutions is 3×3. Of course, the non-linearity I always never mentioned. But it is implicit, non-linearity is of course there. And so once you come to the certain point, you

would just in this case, you will see if it come to this 512, feature maps 64^2, you have a down sampling 32^2, the size of the feature map again, 32^2, then you have a convolution 3×3 convolution get you 30^2 1024 of them.

And then you have a 28^2 feature, size feature map here. So that is one pathway. The other pathway goes from 512 feature maps 64^2 to there is you just copy, this is where you do the copy and crop. So you have copied and cropped 56^2 area from here, and concatenated it with the, within a feature map from this from this pathway.

So you get about 1024 feature maps. So then in this way, this is the up sampling pathway. So you have a sequence once again of convolutions. And then there is an up convolution once again. So this up convolution can be your strided convolutions to get in this case from 52 to 104. Now, you see that why we have this cropping operation, as we come to this layer, the corresponding encoding layer, this filter size maps or the feature maps are big, so you would crop them to the size that is appropriate.

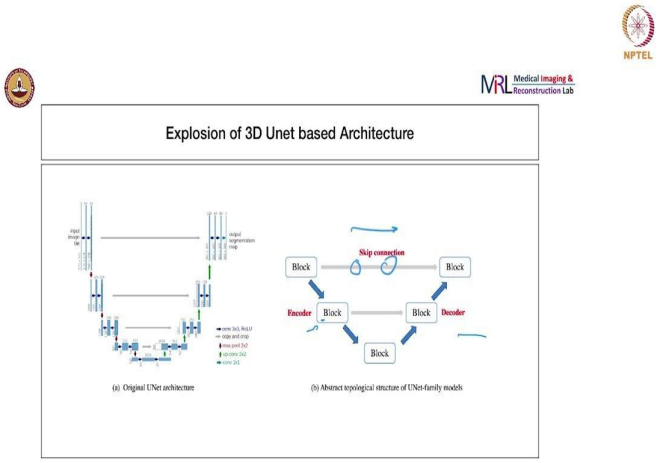
So for instance here, once you go from 52 up sample, you go 104^2 feature maps. But here, it is 136^2, so you would crop 104^2, and concatenate, once again do convolution and up sample. Here, you will crop and concatenate so on and so forth. So, this is a typical pathway. And all the way to the top where in the end you will have an output segmentation map. Here once again, there are not as many segmentation maps in the output the final layer as there are classes.

So let me move that to see in this case for this problem that they are demonstrating in this paper, 2 classes, and foreground and background. So there will be a one feature map output, which is the probability of foreground and other feature map probability of background, of 2 class whichever we want to look at it. So this is the typical architecture for all U-net related things.

So in the sense that your input again can be of any dimension, so we are going to look at 3D in a short while. So and then followed by a sequence of convolutions, down sampling, convolutions down sampling, etc. And then at this U, at this U layer, this part where it is, where the, where it is flattened down, that is you will start to crop and concatenate on the way up. So the cropping and concatenation from the encoding to decoding layer is just to provide, information for better up sampling and better prediction.

So that is a because highlight of the high resolution information is there in the encoding pathway, which is lost when you try to decode, so you need to give that input in order for better decoding.

(Refer Slide Time: 33:33)



So that is like a 1000's of variants of this seemingly and have this 3D U-net. I have, have not shown the 3D U-net base, but typically, this is one input channel here, there could be multiple input channels, not necessarily one input channel. So that is that is where the 3D comes from, we will look at that later.

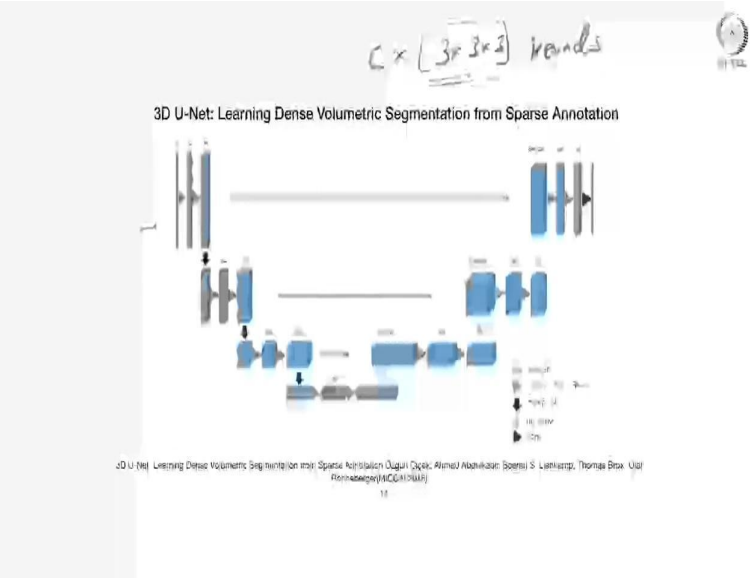
But the general idea is you have these U-net families, right, there is a skip connection. Just point out here, the skip connections always encoder block and a decoder. And these blocks are basically can be any kind of sequence of convolutions, up sampling, down sampling, batch norm, dense net. So, for each block can be a dense net block, each block can be a Res net block.

Each here there is a block there, the competitions can be Res net like, or dense net, like Res net, to so many other so many different types of architectures. They can all be plugged in here as modules and of course, the idea is as you go down, you are losing resolution. As you go up, you are building a solution. Of course, the skip connections themselves, they have variants of it.

So you can put multiple layers here, you can have, a bunch of layers, convolutional layers here, or you can just, copy and crop and copy, or you can do a bunch of 1×1 convolutions to

combine across the feature map. So all of these are possibilities when you do the skip connection also. So this is a general abstraction for the various U-net based architectures that are out there. Some variation of this system is these typically employed to get here, to get better results, depending on the problem.

(Refer Slide Time: 35:18)



So we are just going to look at the so called 3D U-net. So it is basically the 3D counterpart of your 2D U-net that we have looked at so far. So what we mean by 2D U-net? 2D U-net is basically your conclusions or 2D. But of course, we have seen that even in 2D convolutions your filter extends across the channel. So when we see 2D U-net, the input is 2D, there is only one channel typically.

And that is our input, and that, even if there are 3 channels, each channel is 2 dimensional. That will be the best explanation for 2D inputs. So as far as the RGB image has 3 channels, but it is still a 2D problem, because each channel is 2 dimensional. But, for a 3D U-net, you can have multiple channels, but each channel itself is 3 dimensional.

So this particular paper, which I have cited here, which describes this 3D U-net, is describes a situation where there are 3 channels, but each channel is a 3D volume. So it is not just a 2D, each channel is not 2D at the input side, then you are convolutions or your convolution kernels or 3 dimensional. So instead of a 3×3 kernel you would define, it would say 3×3×3 kernels.

And the interesting part is that this of course, will extend across, I put it here channels. So this becomes a tensor, becomes difficult to roll this in your head. But that is the idea behind the 3D U-net, everything else remains the same. Now, you have to understand that when you do the first set of convolutions, you are, your convolutional kernel. Again, subsequently, in all of the convolutions this convolution kernel extends across the channels where each channel is 3D.

So that is the idea. The channels themselves are 3 dimensional, your basic filter kernel is 3D, and of course, all the operations remain the same, everything else remains the same. Of course this, the idea behind doing that is that a lot of the medical images have make sense when they are viewed as 3D objects. So for instance, when you are looking at an organ, like heart, or even if you are looking at a tumor or a pathology, in this case a tumor, you would notice that the organ is a 3D structure.

So, if you view as successive cuts through the organ, which is what we call slices, remember earlier classes about how we can make 3D volumes from CT or MR imaging. The idea is, the successive slices 2D slices in a volume of correlated, so it is best to view them as 3D structures rather than 2D structure. So, when you do the correlation, when you do the convolution and the convolution kernels have to be 3D, because the so called I know the translation invariance, the stationarity applies in 3D sensia.

But of course, it is complicated, the computations are complicated by the presence of extra channels in the input, each channel having, have a 3D form. So again, once again, this paper was used to analyze microscope images, but this is basically sectional microscopy where you have multiple sections in an image. And they show that by now they had very few images, but they were able to get good results.

So, once again, I urge you to read the paper, but this is a general idea. Now, there are all the software packages like Torch and Tensor Flow, which have no predefined 3D convolutional kernels that you can use like modules and if you want to do a rapid prototyping. So, we have looked at a couple of architectures only and look, so do the encoder decoder architecture. We also looked at U-net.

I mean, U-net is the most commonly used architecture or there are variants of it everywhere. But that is that is the most common use of course, you will incorporate everything in this architecture, for instance, there are transformers incorporated into this architecture, etc, called

Trans U-net. There are different variations of this, like I mentioned, skip connections can have, again can have their own, convolutional layers, and so on and so forth.

So, this is again just a brief overview of semantic segmentation. But in medical image analysis this is like the most common task. So for all pathologies you need segmentation, organ segmentation, etc and for population studies. So, this is a very common task and there are lots of research papers on this subject. What I have shown you is like one of the basic architectures that are commonly used, and you can have to build on top of that.

And of course, I have not mentioned many other recent developments in semantic segmentation architectures or even image recognition architectures. And they have been again incorporated into; the image recognition architectures have been incorporated into segmentation architectures etc. Different tricks, different ways to combine high resolution low resolution feature maps, all of that is there but I urge you to read the literature to keep track of that.

But once you, but the general principle is the same, you have an encoder decoder kind of situation always and how you make those pathways are exact is, determines how good your outputs in general are. So this concludes our week in terms of semantic segmentation or volumetric segmentation, if you want to call it for medical image analysis, I briefly touched upon some of the architectures that are commonly used.

You will also have a tutorial to attend to show some basic parsing techniques in the context of a deep learning. Thank you.