[Music]

Welcome back to the lecture on convolutional networks, we will start to look with how a typical convolutional neural network architecture looks like. And you see it here, so you can see you have the input and this is this pixel variant of an image and after that follows a number of layers that is defined here as convolution and Pooling. So essentially it's not pre specified how many such levels you should have it could be one level, one convolution one pooling on convolution pooling but it could so that's up to the design of the network and essentially it depends a little on how complex the image is, how many objects, how many features because essentially if you want to model many features and you have many levels of abstraction that were to capture we need more levels. So we will go into the details shortly I just give you the broad picture now and so after that and one can say that that phase is supposed to do the feature learning, so the idea is that the system after this these faces have captured the relevant features that you can see in the image. So then the next step the final step is given that the features are articulated then we want to analyze this, one typical task is to classify the potential objects that can be there given the feature map and that part is more or less like the traditionally Neural networks that we started to look at in the beginning of the week and the only thing we need to do first typically is because the output of these first phases is normally a multi-dimensional matrix but as you may remember for a traditional artificial neural network then we normally want one feature vector. So actually and typically there is a stage in between called flatten when we take the multi dimension matrix and create one dimensional vector and then based on that vector we do the traditional classification. So this is more or less the architecture.

So there are a few terms that are important here when we go deeper into this different phases. So convolution you already heard you've got the mathematical background there's also some campaign called Filter, a lot of people talk about Kernels. I prefer personally filter because Kernel is used in so many fashions as you remember from earlier lectures is a little confusing actually, there is a concept called Stride, there's a concept called Padding all that have to do very much how you how you handled this convolution layer. If the concept of Feature map and of course that's the map of features at gradually is built up and so on, but we will go through all these concept and systematically in the following slides. So we will start with what is most characteristic for convolutional Network, this is the convolution and Pooling phases and these

phases together one can characterize as the feature learning phase. The feature learning phase is a network consisting an arbitrary number of pairs of convolution and pooling layers, and the number of rows of these pairs of layers are engineering decisions for typical problem settings but in general later deeper levels handles more abstract or high-level features or patterns in analogy with our assumed model of the functioning of the human visual cortex, that also have these layers that systematically looks at more complex features. Yeah so now we will focus on what happened in one layer and we assume that functionality will be the same independently how many this pairs of layers we add. So always for this kind of tasks is very important to look at what we start with so here we assume we got images with a certain format in this RGB model, so in this case we assume we have an example that are 32 times 32 pixels and they have a 3 pixel depth to handle the colors shades. Let us now look at the convolution layer. A convolution layer as typically many filters that are going to be processed on the input image right in parallel most typically in sequence. Now we will look at what happens when we apply one filter to the input image, and as you understand what we do now is an I was a slightly abstract equivalent of what you we did when we did these small examples of convolution because essentially the idea that the way to regard what happens is that we have like the only engine we have two functions we have the input the whole input image which is one entity we have one we have a filter and we are going to view the input image from the perspective of that filter and map that onto another array so you can say that the resulting the output from this face is a new array which somehow should be considered the convolution of the input array we look at and the filter. And it's the filter in a way that reshapes in the same way as the J function in the mathematical function the J function really rare reshapes F so here the filter reshapes the input and gives us the output. So and of course in every of these steps one has to choose a particular filter and a few things to think about them so first of all but to think about the size and obviously filter is a sub area so we have a rectangular input area and a filter should be some subset of that and normally we talk about filters of a limited size they could be 5 times 5, it could be 5 minus 3 times 3 and so on. And the idea is that also in an analogy with how we calculated the mathematical convolution, we should let actually this filter slide across in a systematic fashion and across the whole input array, and so let us disregard the color dimension for a moment I mean they have course had to be handled but they just shouldn't disturb the main process for the moment. So let's forget them and let's assume that every pixel has one value a number. So when we slide the filter across the total input image

we take the dot product between each filter element and each corresponding element of the sub area of the input array that we are at the moment and then we get a scalar out of that with a number of the one number out of that. And as you understand when we move a filter around there are so many positions the filter can be in and normally depending on the size of the filter there are the size of the filter decides how many position can be normally those position are fewer than that what we started from. So when we handled one position of the filter we calculate the number and we put that number in the new output array, and then we let the filter slide we do the same computation for that position, we put it in the output etc. So when we have done that systematically for all positive positions of the sliding then we have a complete feature output map. And the normal thing is that we have a stride which is low which it means that when we slide we take one step or two steps or three steps or stridor one we just move this filter window consecutively but we couldn't have a bigger step which means that we will go more rapidly but the first ever then also fewer filter measurements which will also then in lower the number of elements in the into the feature map. So just to summarize then, what we have is a filter and that's kind of a measurement from a certain perspective we applied we create this output matrix and the size array and the size of the output array depends of course on the size of the filter and the number the size of the this slice actually.

So let's look at another example maybe go through this because it's I think it's important to understand this basic step because it's the heart of this method. So here is another example we have a seven point seven array that gives us 49 elements, the filter is shown in the middle the filter size is three and that is black and lines so now we move from the beginning we get rid of the color schemes and the stride is one and this filter and this is a new thing which we didn't talk about last slide, that of course there have to be a certain pattern in in the future because the pattern in the feature this is the what defines the measurement we want to do. So in this case it's a filter that is supposed to find diagonal patterns in the input image. So the output because of the size of the filter and the stride in the same sense that in the earlier example we get a smaller output array actually a 5 point 5 with 25 elements and here you can see I mean you can do it yourselves so you can place the filter in such a position and then you make a paradise multiplication do you will say that those modifications adds up to the figure four, so therefore you get a four in out matrix. So this is what's going on and of course there are in a realistic example the matrix would pixel is great and there are many many steps in this so it's always a

complex machine learning, but I hope I've given you the picture now that the core mechanisms here are pretty straightforward and not difficult to understand.

There is another concept that you hit when you read about this kind of systems and that's the concept of padding so I can wonder what that means. I mean the principle very simple, so you have a image input array and depending on you who you choose your filter size and how and the stride length it can be so that you have difficulties in an appropriate way by the sliding a cover all aspects because it may be so that you cannot make such slides that every element it's taken into account. So one way then is to actually extend the image frame with some dummy elements on the border. Of Couse this is not informative because it's just put there to get some space but if you do that then you may counteract this is the state of the affair that you cannot really measure every corner of the image. It's more of an engineering decision from case to case whether you think this is important, if it's necessary, if it's beneficial and so on. So there is no clear, it's a context-dependent choice and that's the concept of padding that's really augmenting the image frame image with the neck strap frame for the reasons I gave you.

As I already told you in each convolution layer there may be many filters that mean them maybe which means there are many kind of measurements you want to make on in parallel on this input image and these are handled separately, so there's the same process for each filter and each filter produce an output array and as you can see from the picture on the right what we then get when we are finished we'll go in through the old sets of filter it weighs like a sandwich of output arrays, so we get a bigger array actually in the end. The pooling subsampling layer is a complimentary step, so I said earlier normally the convolution step is always followed by something step and the purpose of the pooling layer is to use the complexity of the array you created. So because if you have many filters looking for many different kinds of features and that will give you potentially a large array coming out from the convolution step and typically you don't want to reduce that. There is an analogy here with what we discussed in an earlier week of overfitting because if when we talk about decision tree we said that okay if decision tree becomes too complicated if you killed so too many nodes, too many branches it's more likely that it will do overfitting on the data it's going to test. And it's the same here there is some parallel here that you'll get a to complex array structure here it it's also a risk to get the similar phenomenon. So the pooling layer operates on each feature Maps also as we said out from the

convolution step comes portfolio or sandwich of these feature maps and then so in the pooling layer you do the pooling operation on each feature map independently. Also here you use filters but for another purpose somehow but actually you typically also here define a filter with a certain size and then you move that filter around on the surface of the array you have at hand and then of course you need to define what kind of matching should take place between that filter and some sub-region of the area you're going to study and one very much used approach is to take just pick the biggest value. So out of the out of the big array you apply the filter window and then for the sub area you cover you pick that pixel with a with a largest figure and that figure is what you take out as the result from that measurement and as in the same way as did with convolution you move this filter around with a certain stride also so as you can see in the example here to the right ,you have a four times four window and you have a with selective filter of size two in both dimensions and we have a stride of two which means actually that there are only four positions that this filter can have. So when we so then actually when we go around we place the filter over which one of these four positions and if we have this max pooling approach we just pick the biggest number in each position. An alternative approach is average pooling then you take the average value in each little square and you have pop that result here. So the output here and given this example is a two times two matrix instead over four times four. So that's also pretty straightforward. There are two phenomenon or two aspects of the neuron structures used in this networks for the convolution and pooling labels. So two phenomena is Weight sharing and Local connectivity, and both these phenomenal aspects are actually attempts to fight the complexity. So for example it's assumed that when you apply a filter you should apply it in more or less in the same fashion doesn't matter where on a input array you apply it the behavior should be the same, so therefore normally you make restrictions on the weight so you say for the weights of the neurons involved should be the same doesn't matter whether the filters employed here and here, neurons involved should keep the same weights in these cases. So therefore in a normal artificial neural network the weights could be to all something (23:04) separate, so there are no restrictions on connects a little memory (23:08) but in this case at least for these subsets of weights for this subset of neurons you can imply the restriction that should be the same. So the other thing local connectivity is also as you know that in a general ANN the neural connections any neuron can be connected to any other neuron, but here because we have this layer structure and also certain neurons applied in a certain filter then it's also normal that pretty strictly restrain

the connectivity among the neurons involved, so post these things together contribute to making the system more efficient. When leaving the convolution and pooling layers and before entering the fully connected layers the output of these previous layers is flattened. By this is meant that the dimensions of the input array from early phases are flattened out to one large dimension. So you can see a small example to the right but you can also have a bigger one you have a 3d array with the shape of 10 times 10 times 10, so when flattened would become a 1d array with thousand elements. So this is pretty straight forward. So now we have something that is easily and normally it can be put as input to a kind of standard artificial neural network.

After the flatten we move to the fully connected layers. So in contrast to the earlier layers who primarily did the feature extraction, the fully connected layers are supposed to do typically classification or something of that kind. The fully connected layers takes as input the flattened array represent the activation match of high level features from earlier arrays and outputs an N dimensional vector typically. So N is the number of classes that the program has to choose from, so for example if the task is digit classification and N would be ten since there are 10 digits. And the fully connected layers determine which is best correlate to a particular class. So and you can have different kinds of activation functions in input layer, you can have the ones we already discussed but there's also something called Softmax activity function and that's particularly useful for handling multiple class classification problem. So essentially what that kind of activity function ensures is that the outcome, the output in this vector is a sum of numbers that equals to 1, so it's the probability it gives you a probability for whether the instance you look at, found an instance in the image is a member of one of these classes so that can be pretty handy.

Finally here there are few comments on the activation functions that could be used, so many times you use different kind of activation functions in different parts of this kind of complex system. So if we look at state-of-the-art it's very usual too common to use something called Rectified Linear Unit activation function in the convolution and pooling layers. It's a very straightforward function and easy to handle, there are no negative things with your doing that. For the for the output you can do different things, you can use what was already mentioned Sigmoid functions, Hyperbolic functions if you have a binary classification problem. For multiple class by that classification problem the Softmax function is very popular because you get a normalized input in terms of probabilities for class membership. Sometimes people use also

Gaussian functions but that's not the most common, you can also have an identity activation function which is recommended for regression problems.

Here we will we go through another example and it's also mentioned this example it's a pretty well-known example of a CNN, actually one of the first real systematically carried out systems of this kind. So it's from 1990, so it's pretty old given the use of this sub area and it's a system where you were the purpose is to analyze handwritten machine printed characters actually, I mean they are an image form so there are infinite pixels so and it's still an image recognition problem. So this is rightly you can see here this is more of a repetition of course you can see that there are two of these pairs of convolution and subsampling layers and then it's a full connection area so there one flattening layer, two fully connected layers and finally there is Softmax classifier applied.

So here you can see you also see clearly how this dimensions vary I mean this is of course a key issue here to understand how the size of this array so how complex the problem you create. So here it's a grayscale 32 times 32 grayscale image that's what you start from and then you are you have a convolution layer with a filter of size 5 and stride of 1. So then out of that you get the matrix 28 times 28 times 6. And then actually you apply a second layer where you reduce those dimensions with two pooling layer. So after that, you have a different kind of convolution, you have a different kind of pooling, so then in the end you are down to something that is 5 times 5 times 16 actually. And then you have two fully connected layers an output layer and in the there is a mapping on the digits that is 0 to nine.

So you can see as a summary of the system you can see the different layers employed, you can see the kind of feature maps produced the sizes the filter kernel is the equivalent synonym to filter, so you can see the filter supplied you can see the strides, you can see the kind of activation functions. So hyperbolic tangent is a variant of sigmoid. So obviously here in this very early system it wasn't so popular to use this array very new function at that point so still it was the fashion to use Sigma related functions. So this was an example again of the same scheme and finally I will give you some timeline for what happens in this area as I said the pretty young area, maybe one can say that LeNet the one example I showed you lately is more or less the point where this really took off. There were a number of precursors in the period before and actually this area really got famous only at the point where you've got more powerful hardware to support

this system because these systems are very complex and takes a lot of computing power so and the existence and the knowledge to apply special hardware for running these systems made a big change and I would say that the big success story is this 2012 system Alexnet by Alex Krizhevsky. So that was convolutional neural networks, I hope you bought a fare picture it may be not the simplest area but it's very important component in this whole realm of artificial neural networks. So thank you for attention, at the end of this lecture the next lecture is 6.9 will and this week and the lecture is about deep learning and some recent developments thank you.