Lecture - 22.2

Generative Adversarial Networks – Architecture

Okay? So, now I have just with that we have seen an overall idea behind gas so you have the generator you have the discriminator both of them are neural networks you have this minimax loss function run of the one part is with respect to theta the other part is with respect to Phi and you alternate between these two objectives of T and Phi Okay? Now what I'd not told you is that I just said that the generator is a complex transformation which is a neural network Right? But, what is this neural network in practice what do you use for that neural network whether you use a feed-forward neural network or use a convolutional neural network. So that's what we look at in this module, what's the architecture used for the generator what's the architecture used for the discriminator. Okay? and again there are various choices here,

Refer Slide Time: (1:04)

- We will now look at one of the popular neural networks used for the generator and discriminator (Deep Convolutional GANs)
- For discriminator, any CNN based classifier with 1 class (real) at the output can be used (e.g. VGG, ResNet, etc.)

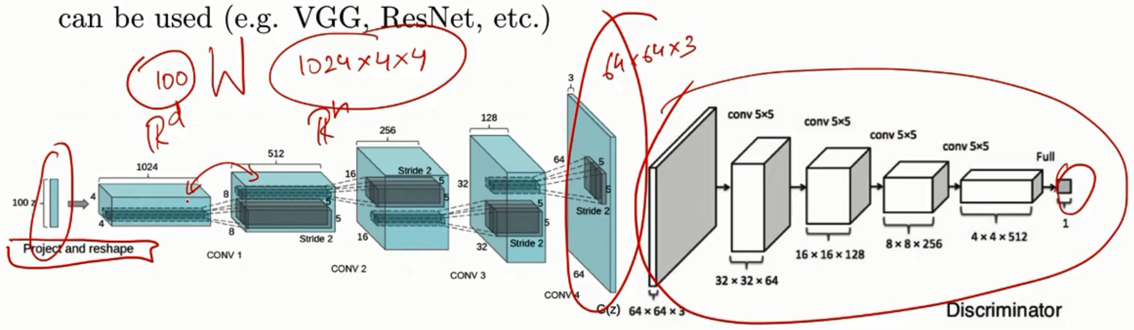


Figure: Generator (Redford et al 2015) (left) and discriminator (Yeh et al 2016) (right) used in DCGAN

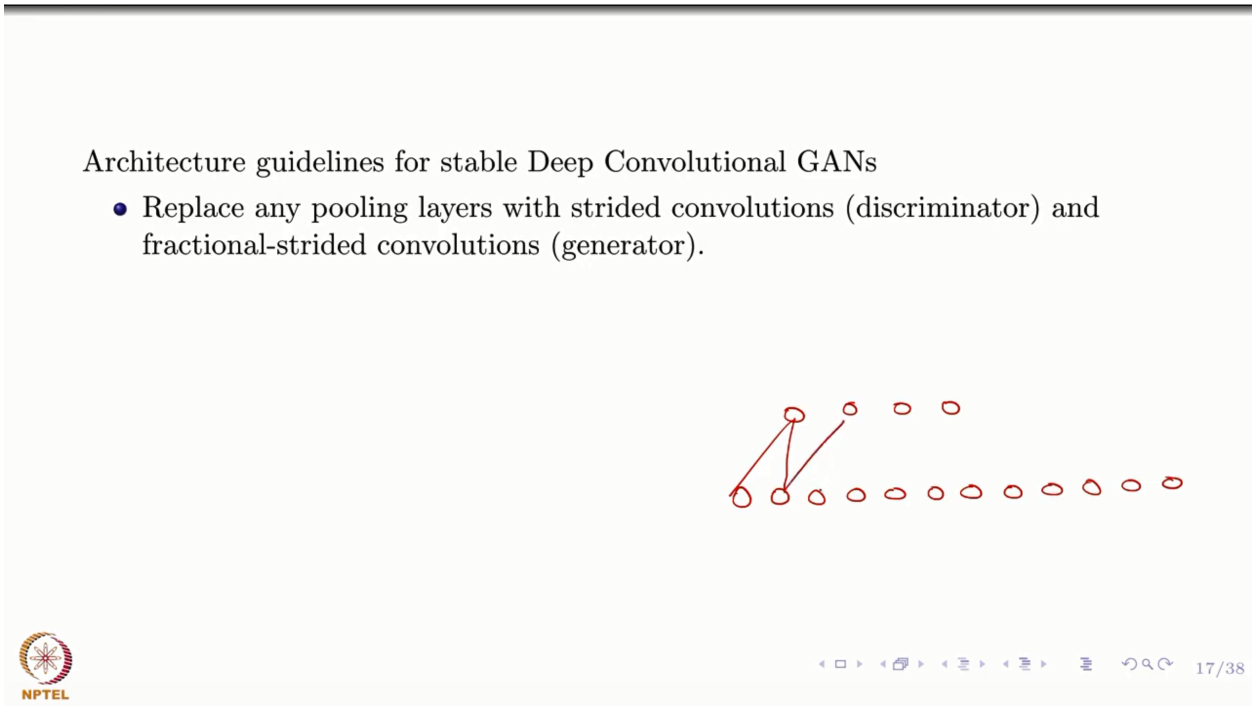
16/38

I'm going to talk about the most popular one which works very well right so that's something known as deep convolution grants so the good thing is for the discriminator you can use any CNN architecture that you want so pick up whatever is your favorite architecture VGG resonate whatever just use it as a discriminator what would be the output layer of the discriminator, how many neurons will it have one there is no SoftMax as we require an amazing it they'll just be one neuron that will give you a value 0 or 1 is that fine. Okay? So the discriminator will just be any convolutional neural network that you like any popular architecture with one output which tells you whether this is real or fake. Okay? But for the generator things are not so straightforward people experimented a lot with various architectures and this paper which is cited here actually came up with this a set of heuristics for what works well for the generator, and this is the network which they tried so let's look at it so you have this noise vector which is hundred dimensional. Okay? From here you need to go to say a 64 cross 64 cross 3 output. Right? So, this is an image of 64 + 64 cross 3 channels. So, this is your input, and this is your output so how do you go from the input to the output so use a series of so let's understand each of these.

The first thing that you do is see what is said here it's project and reshape. Okay? So what you're going to do is you're going to take this 100 dimensional vector multiply it by an appropriate W, so that you get an output of size 1024 cross 4 cross 4 what's the size of W going to be, that's the project part. Right? So, what's the size of W going to be, you have this as the input this is the output this is RD this is our n what's the dimension of W D cross n right is that fine but because n will just give you a vector again so what you do is you take that vector and reshape it into 4 cross 4 cross 1024 is that fine everyone gets this. Okay? Now what you do is you apply something known as transpose convolution because now remember we have to go from 4 curse 4 curse 1 0 2 4 to 64 cross 64 cross 3. So, we have to actually increase the size where is all the convolutional neural networks that we have seen they start with 64 cross 64 and come down to a smaller size so we have to do the reverse operation now. So you have to use something known as the transpose convolution so what's the transpose convolution, or in other words tell me how will you do this operation that you go from 4 cross 4 to 8 cross 8 shall give you some hints- the operation is transpose convolution you know how to take transpose of matrices you also know how to treat a convolutional neural network as a feed-forward neural network where a lot of entries in the weight matrix are 0.

So in the first figure if you can recall that we had seen in the convolution neural network where we are taking this M this timid converted into 16 inputs 4 cross 4 and then shown the second hidden layer and shown a feed complete feed forward neural network and then remove some edges from it. Right? Do you remember that vaguely of course you do?

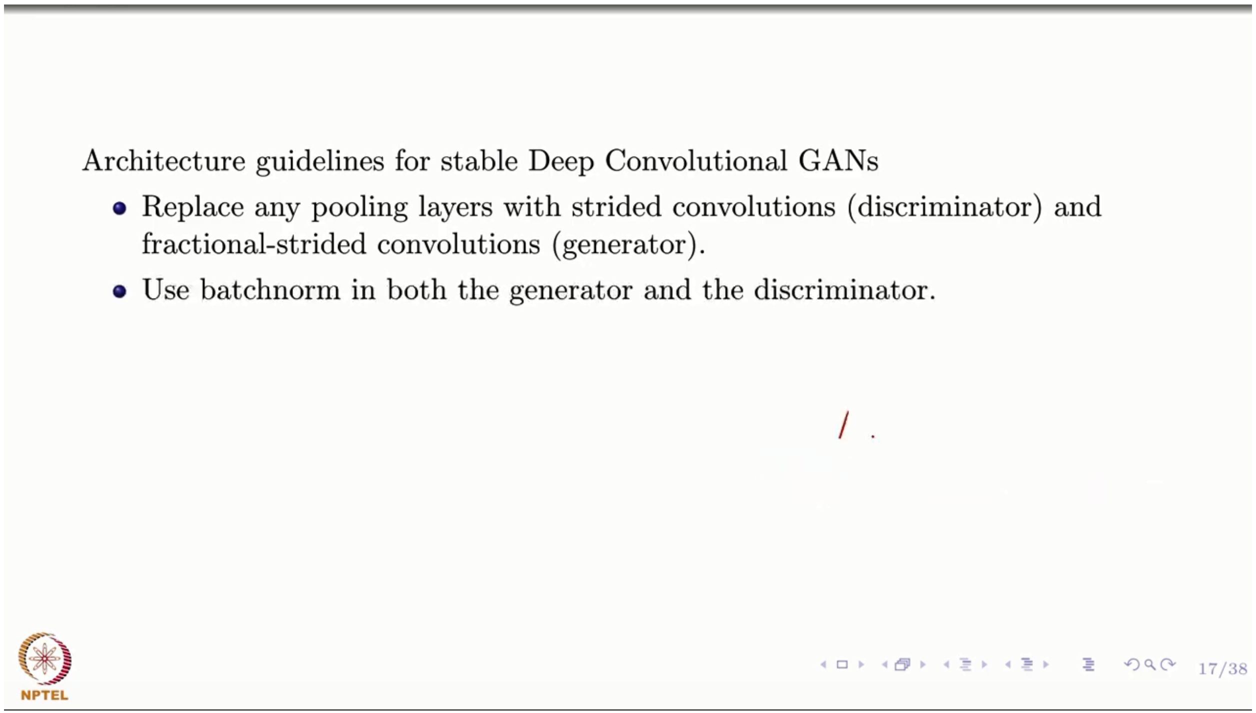
Refer Slide Time: (4:40)



Architecture guidelines for stable Deep Convolutional GANs

- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).

Refer Slide Time: (4:58)

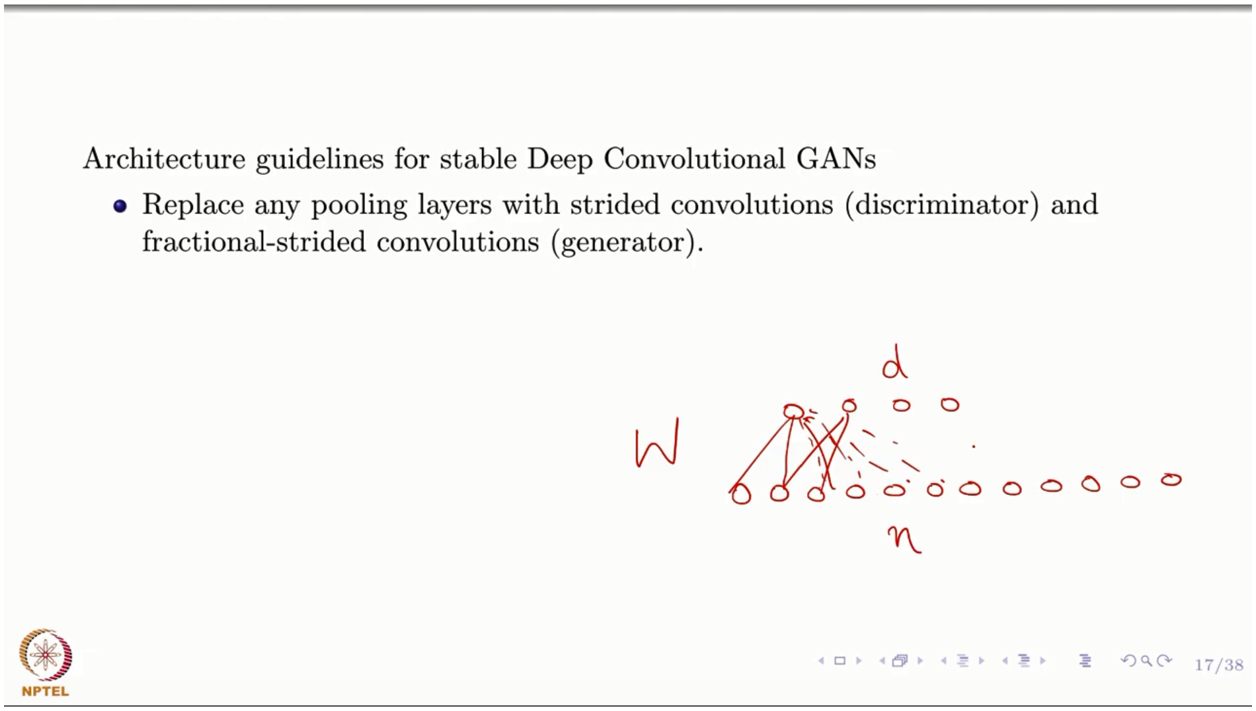


Architecture guidelines for stable Deep Convolutional GANs

- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).
- Use batchnorm in both the generator and the discriminator.

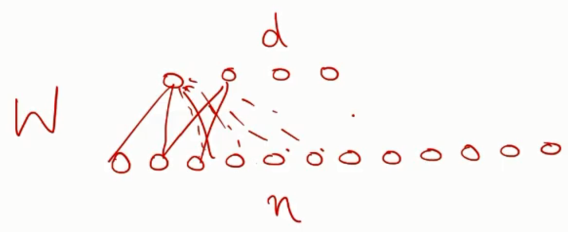
Okay? So, I'll in the interest of time right I'll just tell you what I mean by this, so you can always think of the convolutional neural network as a feed-forward neural network with sparse connections Right? So, this guy would only be connected to these two Okay? Maybe for example,

Refer Slide Time: (4:58)



Architecture guidelines for stable Deep Convolutional GANs

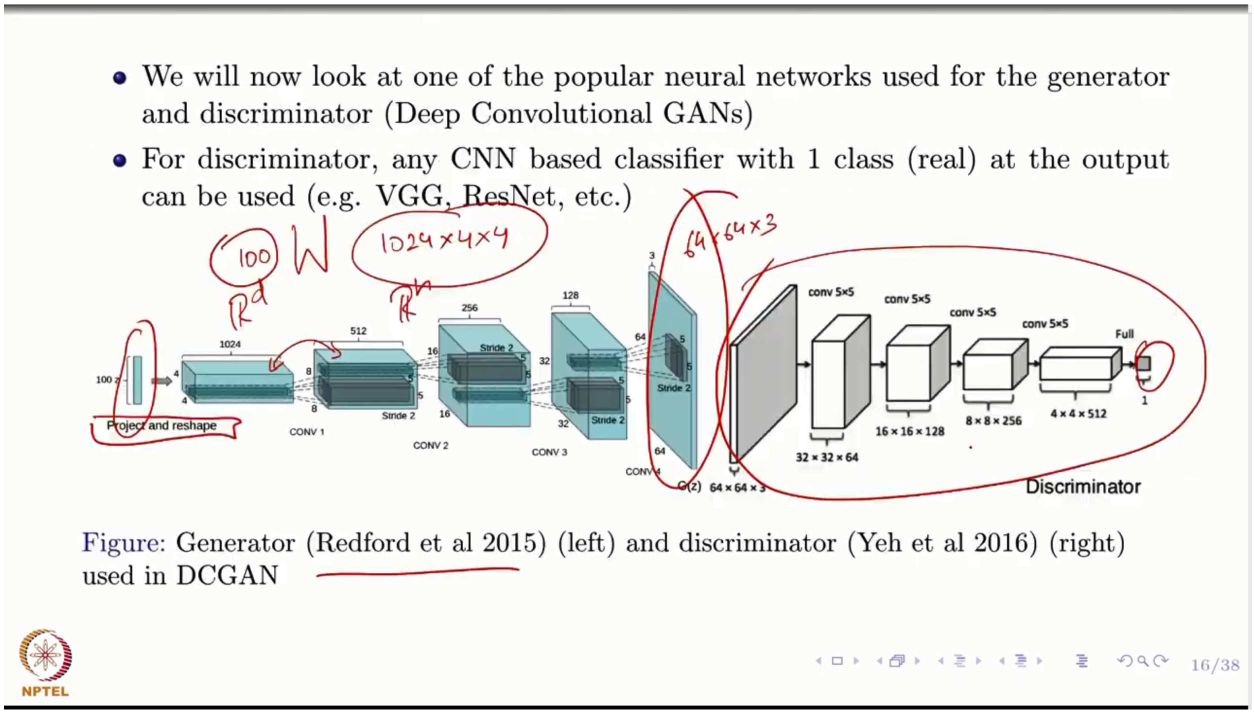
- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).



17/38

This guy would be connected to these two and so on. Right? But, the other way of saying this is that all of these are actually connected it just happens that these dotted lines which I am showing are zero weights is that fine then I can write this as a normal weight matrix which goes from an n-dimensional input to a d-dimensional input is that fine? Now what do I actually want to do I want to go from a d-dimensional input to an n-dimensional input what kind of a transformation can I do W transpose I take the d dimensional input applied W transpose to it I'll get an n-dimensional does that make sense avian gets that so that's the inefficient intuitive way of understanding it. In practice you will have these API stencil flow and so on which will implement this transpose convolution much more effectively. Right? Because if we do it this way the way I said they're going to do a lot of these zero multiplications. Right? which does not make sense you know that the output is going to zero but you are still doing those multiplications. Right? So, actually it will not be implemented it like that but the way to understand a transpose convolution is that you treat the convolution as a matrix multiplication operation, a very sparse matrix multiplication operation just take the transpose of that matrix to go from d to n everyone gets this piece raise your hands if you get this. Okay? Good.

Refer Slide Time (6:21)



- We will now look at one of the popular neural networks used for the generator and discriminator (Deep Convolutional GANs)
- For discriminator, any CNN based classifier with 1 class (real) at the output can be used (e.g. VGG, ResNet, etc.)

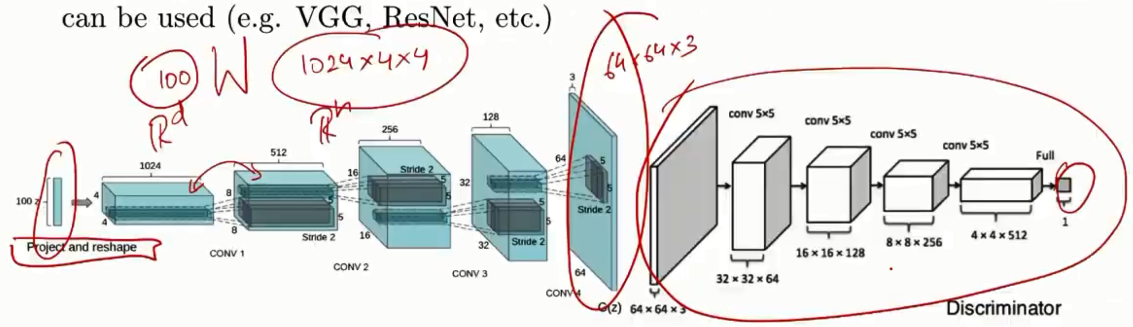
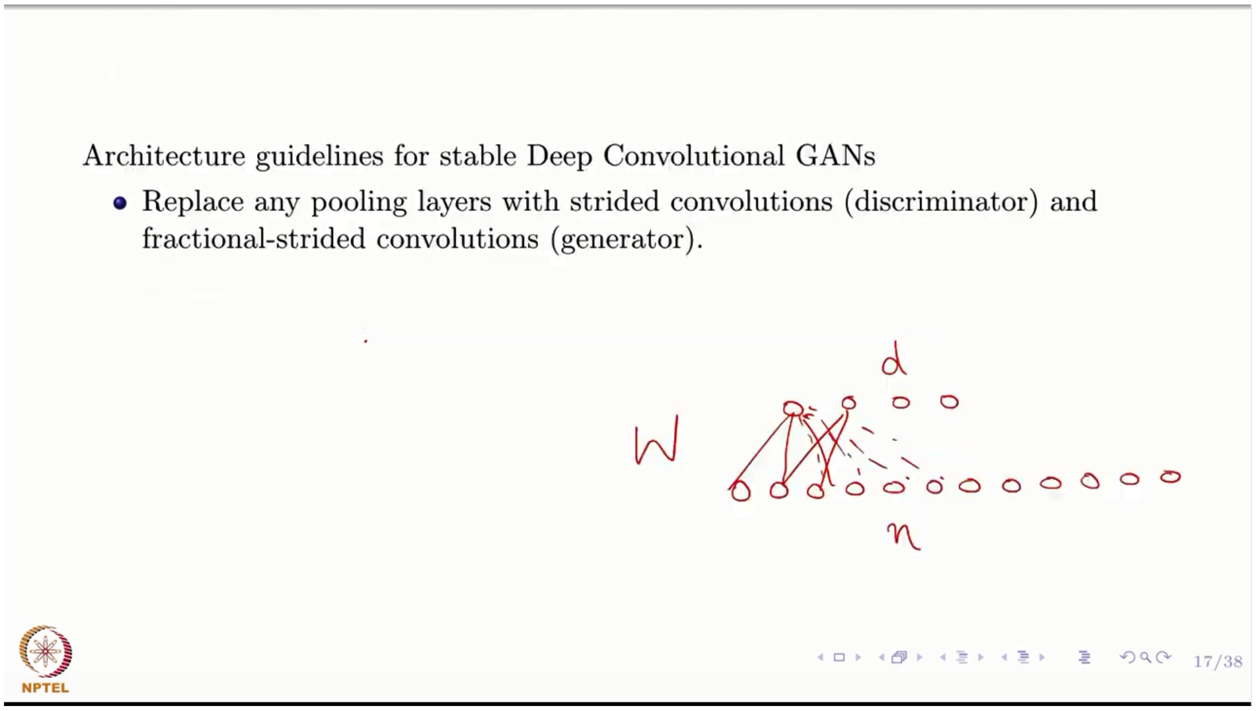


Figure: Generator (Redford et al 2015) (left) and discriminator (Yeh et al 2016) (right) used in DCGAN

So, that's what these inverse transfers sorry, transpose convolution operations are and that's how you grow from 4 cross 4 to 64 cross 64 but how do you reduce the depth from 1024 to 3 how will you do that how did you increase the depth from 3 to 1024. In a normal condition in your network by increasing the number of feature maps or number of filters So, now I will keep decreasing the number of filters. Okay? So, here originally you had 1 0 2 4 in the next layer you'll just have 512 filters and so on till you go to three does that make sense. Okay? So, this is what the generator looks like this is a standard architecture for DC Gant's and this works very well in practice and the discriminator as I said can be any convolutional neural network

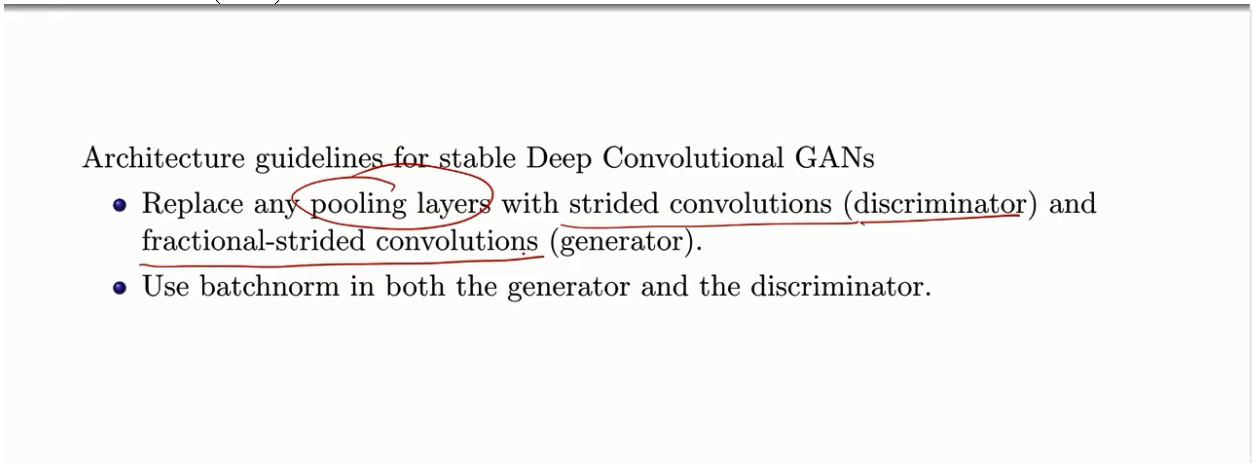
Refer Slide Time: (7:06)


Architecture guidelines for stable Deep Convolutional GANs

- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).

Okay?

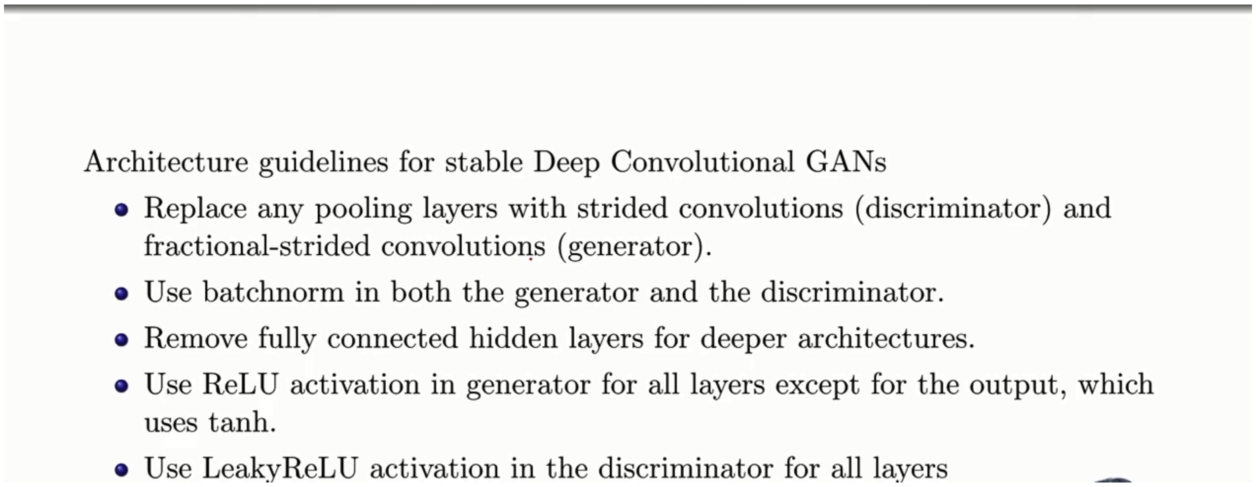
Refer Slide Time (7:07)


Architecture guidelines for stable Deep Convolutional GANs

- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).
- Use batchnorm in both the generator and the discriminator.

And here are some straightforward guidelines for having a stable deep convolutional neural net the stable deep convolutional GANs. So if you go online and read a lot of literature about GANs one constant complain is that it's very hard to train them the training is not very stable, you start getting nans you start the disk media learns very fast but the generator lags behind and all sorts of things happen. Right? So, that's why there's a lot of emphasis in coming up with stable architectures which learn well so DC gas was one such interesting and important contribution where they came up with an architecture which works very well and here are the standard guidelines that they had, so for the discriminator you do not use any pooling layers and instead you strided convolutions what does that mean what does a pooling layer do compression, what does trading do again compression. Right? Because you apply straight they are not going over every pixel so you're going to get a smaller output, so instead of doing max pooling you do strided convolution and for the discriminator sorry for the generator use fractional strided convolutions what does that mean, that's the transpose convolution operation that we just spoke about Right? For the generator you will have to use this transpose convolution operation. Okay.

Refer Slide Time: (8:21)



Architecture guidelines for stable Deep Convolutional GANs

- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).
- Use batchnorm in both the generator and the discriminator.
- Remove fully connected hidden layers for deeper architectures.
- Use ReLU activation in generator for all layers except for the output, which uses tanh.
- Use LeakyReLU activation in the discriminator for all layers

Use batch norm for both the generator and the discriminator, remove fully connected layers from the deeper architecture and use ReLU as the activation function for except for the output, which is going to be tan edge and, use leaky ReLU for the describes.

It all looks like very Blackmagic red you can use this use this and so on but this is come out after the lot of experimentation and these are the configurations or these are the choices which led to the most stable training. Right? So, you could just take this off the shelf and try to implement it and it should work better. Okay?