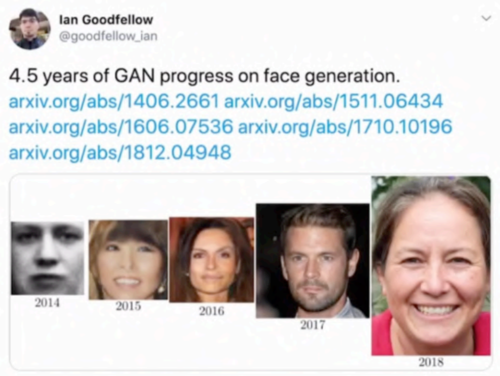
GAN Improvements

deeplearning.ai

Outline

- How GANs have improved
- State of the art methods for improving GANs performance

GANs Over Time



Ian Goodfellow
@goodfellow_ian

4.5 years of GAN progress on face generation.
arxiv.org/abs/1406.2661 arxiv.org/abs/1511.06434
arxiv.org/abs/1606.07536 arxiv.org/abs/1710.10196
arxiv.org/abs/1812.04948

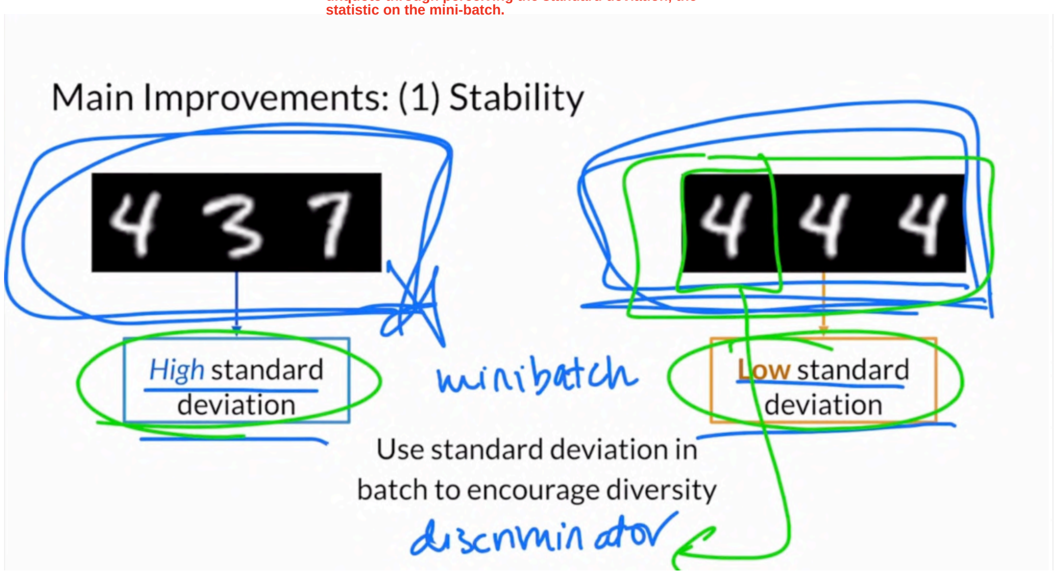
https://arxiv.org/abs/1406.2661
https://arxiv.org/abs/1511.06434
https://arxiv.org/abs/1606.07536
https://arxiv.org/abs/1710.10196
https://arxiv.org/abs/1812.04948

The first major improvement was to GAN training stability. You might recall from the section on mode collapse. An Unstable GAN typically means that training at longer won't help it. In this case, the generator here is stuck in some local minimum, and it's only generating one thing and here it's only generating 4s, when it really should be generating a diversity of things.

You've seen many ways to prevent mode collapse. But one interesting thing is to take a look at these two samples where you want this thing on the left with a lot of variation and there's high standard deviation across samples. But, and again, that has mode collapse when you look at a mini-batch of, say, three samples compared to the one that you want here on the left, you see that there is very low standard deviation, meaning that there is very little variation across these three samples.

Each of these three samples from your mini-batch can be pretty indicative of how much variation there is inherent to your GAN. As a result, the standard deviation of these images can show when your generator is approaching mode collapse because its standard deviation will be low.

To mitigate this, perhaps you can pass this information to the discriminator and it can then punish the generator or rather give feedback back to the generator to try and keep it from collapsing. Because now if this is given to the discriminator, instead of the discriminator just seeing one sample at a time, it will now see all of these and realize there might be a little bit of mode collapse here going on. Of course it doesn't see all those samples, it sees them quote unquote through perceiving the standard deviation, the statistic on the mini-batch.



Main Improvements: (1) Stability

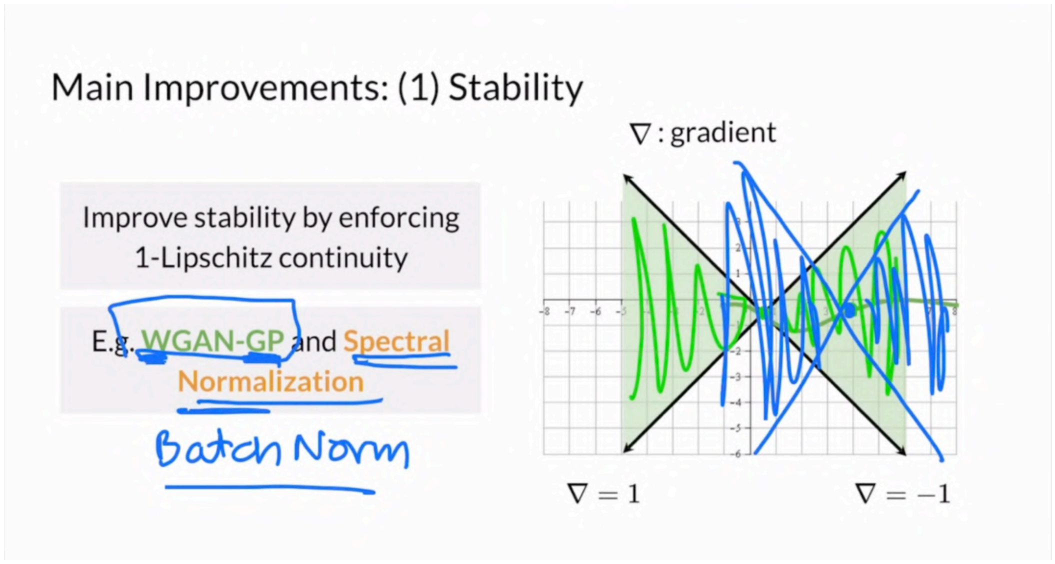
Use standard deviation in batch to encourage diversity

Another way researchers have improved stability in GANs is by enforcing one Lipschitz continuity when using Wasserstein loss or W loss. This prevents a model from both burning too fast and also keeping in valid W distance, which you've seen in Week 3 of this specialization, where you want to make sure the gradient of your function remains in these green triangles at every single point of this function. This is observing this point and observing a point out here would yield a different set of triangles, and you want to make sure you're function stays within those. That means it's not growing more than linearly.

You've seen this with WGAN-GP, which is using Wasserstein loss, which can be abbreviated by this W and maintains one Lipschitz continuity by using the Gradient Penalty or GP here.

Another method to enforce this is also to use Spectral Normalization on the weights. This is similar to a concept you might be more familiar with, known as Batch Norm or Batch Normalization. Another method is called Spectral Normalization, which in some senses they are similar to Batch Normalization, in the sense that it normalizes the weights and it's often instantiated as some layer, so Spectral Norm layer. This is another type of weight normalization like Batch Norm and it stabilizes training by keeping values within a certain range.

I won't go into the details because it gets pretty mathematical on what a Spectral Norm is, but it doesn't maintain this one Lipschitz continuity. Don't worry too much about it. There are open-source implementations of this method.

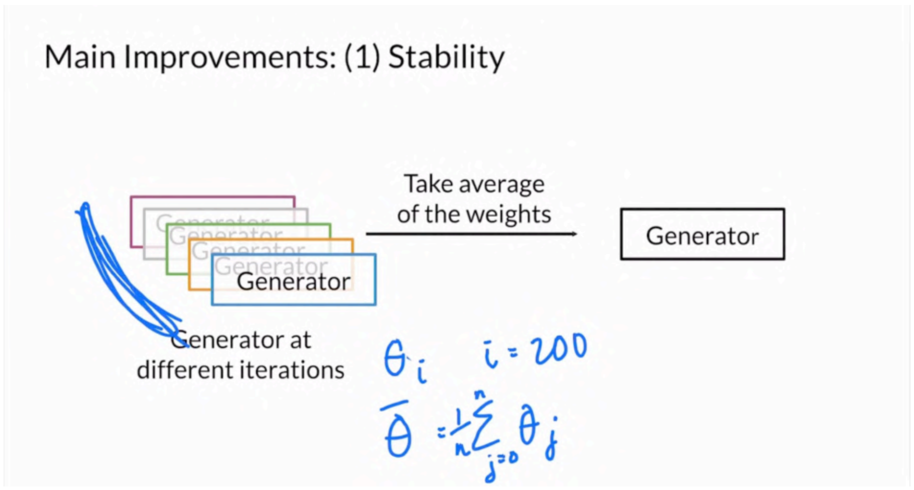


Main Improvements: (1) Stability

Improve stability by enforcing 1-Lipschitz continuity

E.g. WGAN-GP and Spectral Normalization

Batch Norm



Main Improvements: (1) Stability

Generator
Generator
Generator
Generator
Generator
Take average of the weights
Generator
Generator at different iterations

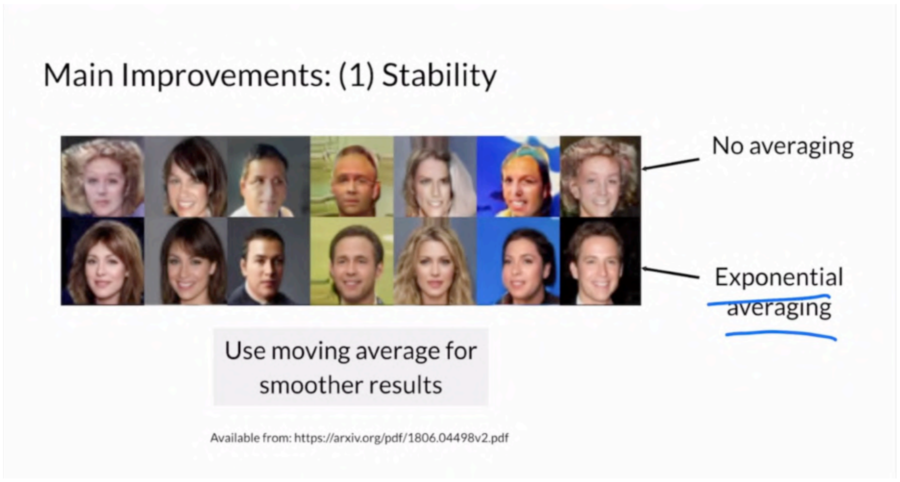
$\theta_i \quad i = 200$

$\bar{\theta} = \frac{1}{n}\sum_{j=0}^{n} \theta_j$

A final way researchers have tried to stabilize GANs is by using two techniques that you'll learn more about in the following videos in StyleGAN.

The first is using a Moving Average that takes the average weights over several checkpoints. Instead of describing the weights of one generator, which would be typically Theta_i, where i equals, let's say Step 200, wherever you find the best generator checkpoint.

Instead you want the average Theta, which I'll put a bar here, which will equal the sum of j to n Theta_j. Instead you want the average of all of these parameters.



Main Improvements: (1) Stability

Use moving average for smoother results

Available from: https://arxiv.org/pdf/1806.04498v2.pdf

This gives you much smoother results and is done during training, but also can be applied when saving a model and then sampling from it.

Here you can see the differences between averaging and not averaging. Without averaging, you can see that maybe that generator checkpoint was in some local minimum or it just wasn't very smooth versus using some averaging. Here, the exponential average can yield much smoother result and the differences are quite apparent here.

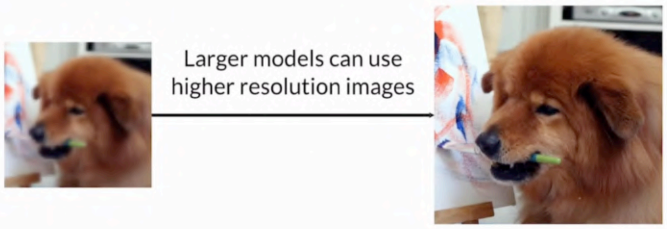
Main Improvements: (1) Stability

Progressive growing gradually trains GAN at increasing resolutions

The second technique is called Progressive Growing. It gradually trains your generator on increasing image resolutions. It can generate high-quality images. You'll learn more about this in the coming videos on StyleGAN.

Now that you've learned some ways, researchers have increased GAN training stability so they can train for longer and improve. You'll see some other areas where GANs have advanced as well.

Main Improvements: (2) Capacity



Larger models can use higher resolution images

Another major improvement to GANs was their capacity. Generators and discriminators have gone a lot wider and deeper, such as deep convolutional GANs, DC-GANs that you learned about earlier in Week 2.

These models can be larger because of improved hardware like better GPUs in higher resolution data sets, like the FFHQ data set, which is a set of face images from Flickr that are really high resolution to a trained StyleGAN, the state of the StyleGAN.

Some architectures had begun catering to these higher resolution data sets as well.

Main Improvements: (3) Diversity

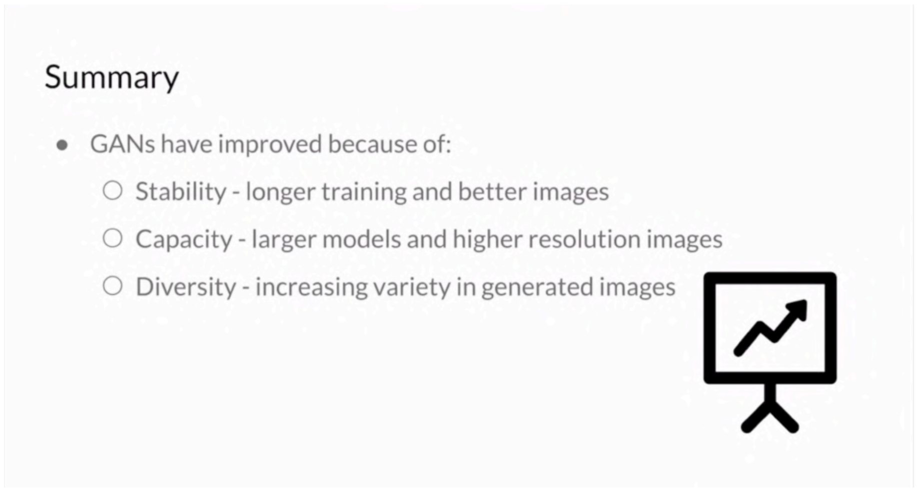
minibatch std dev can help, too!

Available from: https://github.com/NVlabs/stylegan

The final major improvement to GANs has also been their diversity and their output. They have had increasing variation in their generated images and that's fantastic because we want to model all the possible human faces here, not just generate one face. Remember that this is also a key criterion that you now evaluate your GANs on.

One of the ways diversity has been improved is by having larger data sets with more coverage because having more variety in the images being trained on, lead to the same in the output as well. Faces you see here are actually generated by StyleGAN, and StyleGAN has a few architectural changes to increase diversity that you'll see in the next couple of sections as well.

I do want to point out here that training stability, and I do want to point out here that the mini-batch statistics that you saw previously using the mini-batch standard deviation can certainly help too. I do want to know that the method you saw before using mini-batch standard deviation to perhaps help with the stability and training can also help with increasing and improving diversity by avoiding more collapse and turning towards how much variation real images have.



Summary

- GANs have improved because of:
  - Stability - longer training and better images
  - Capacity - larger models and higher resolution images
  - Diversity - increasing variety in generated images

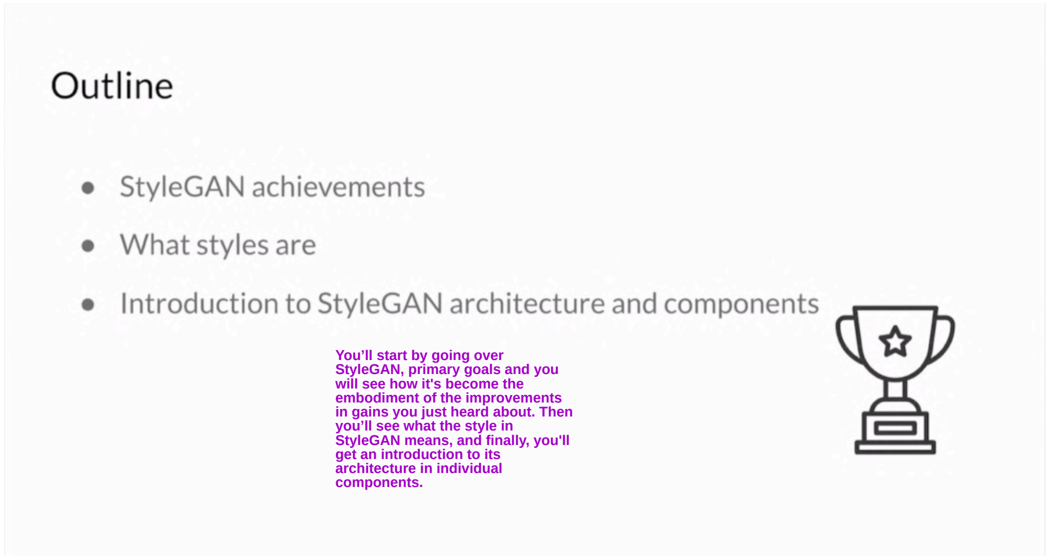
In summary, GANs have advanced largely from improved training, stability, capacity, and diversity. A stable GAN with less chance of mode class lets you train your model for longer and larger models and higher resolution images will improve the quality of your outputs.

A few changes will increase the diversity in your generated images, and you'll see some of these in the following sections on StyleGAN. One of the best GANs today.

deeplearning.ai

StyleGAN Overview

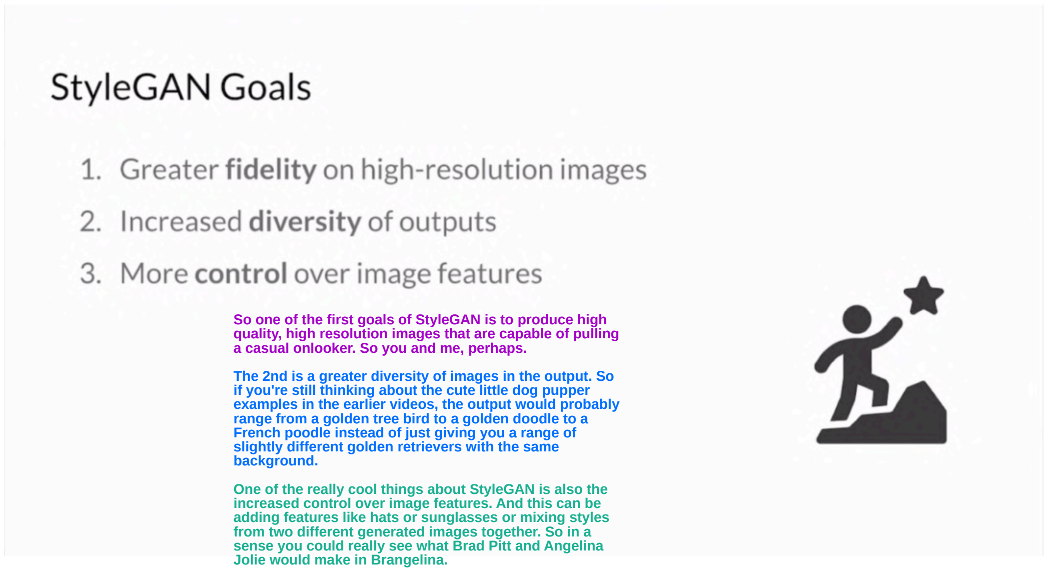
In the previous section you heard about how GAN have improved over the past few years. In this video you'll learn about StyleGAN relatively new architecture that's considered to not only be the state of the art GAN right now, but an inflection point improvement in GAN, particularly in its ability to generate extremely realistic human faces.



Outline

- StyleGAN achievements
- What styles are
- Introduction to StyleGAN architecture and components

You'll start by going over StyleGAN, primary goals and you will see how it's become the embodiment of the improvements in gains you just heard about. Then you'll see what the style in StyleGAN means, and finally, you'll get an introduction to its architecture in individual components.



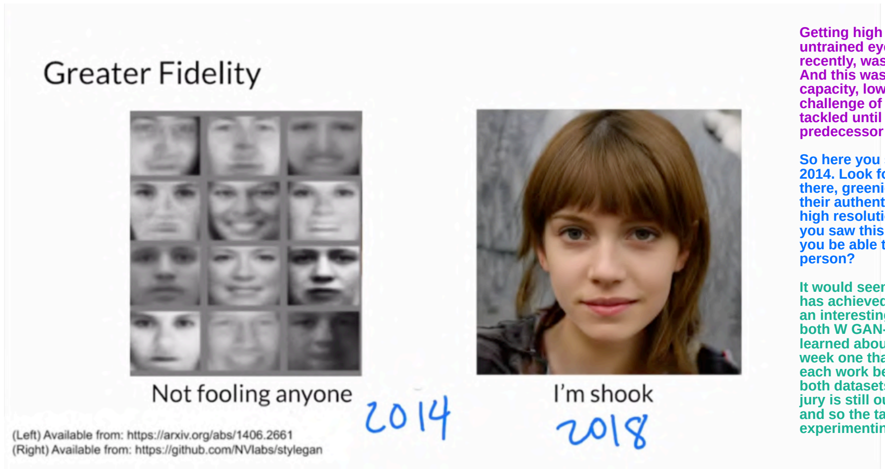
StyleGAN Goals

1. Greater **fidelity** on high-resolution images
2. Increased **diversity** of outputs
3. More **control** over image features

So one of the first goals of StyleGAN is to produce high quality, high resolution images that are capable of pulling a casual onlooker. So you and me, perhaps.

The 2nd is a greater diversity of images in the output. So if you're still thinking about the cute little dog pupper examples in the earlier videos, the output would probably range from a golden tree bird to a golden doodle to a French poodle instead of just giving you a range of slightly different golden retrievers with the same background.

One of the really cool things about StyleGAN is also the increased control over image features. And this can be adding features like hats or sunglasses or mixing styles from two different generated images together. So in a sense you could really see what Brad Pitt and Angelina Jolie would make in Brangelina.



Greater Fidelity

Not fooling anyone 2014

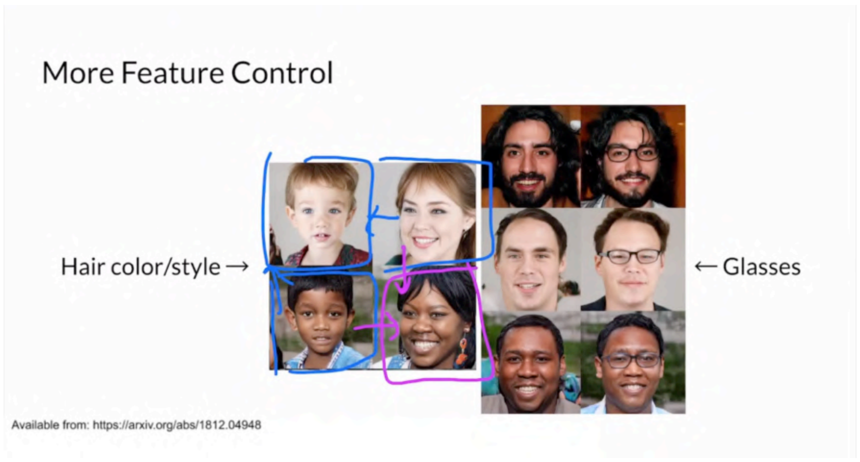
I'm shook 2018

(Left) Available from: https://arxiv.org/abs/1406.2661
(Right) Available from: https://github.com/NVlabs/stylegan

Getting high resolution fidelity that fools the untrained eye is a huge feat that up until recently, was much more difficult to achieve. And this was mainly due to smaller model capacity, lower resolution datasets, and also the challenge of high resolution just really wasn't tackled until ProGAN, which is StyleGAN predecessor that was launched in 2018.

So here you see these generated faces from 2014. Look for the most part like bad sketches there, greenie an unlikely to convince anyone of their authenticity. Now look at this more recent high resolution face generated by StyleGAN, if you saw this image without any context, would you be able to guess that she wasn't a real person?

It would seem pretty apparent that's StyleGAN has achieved its goal of greater fidelity, but on an interesting side note, StyleGAN has tried both W GAN-GP loss from week three that you learned about, and the original GAN loss in week one that you learned about and found that each work better on a different data set. And both datasets were high resolution faces, so the jury is still out, but really both probably work, and so the takeaway is that it's all about experimenting.



More Feature Control

Hair color/style →

← Glasses

Available from: https://arxiv.org/abs/1812.04948

Finally, StyleGAN also wants to increase the control you have over your image feature. And this can be mixing the style of 1 image into another like you can see here where this face here is a mix from the below and to the right where you can see hair color and styles coming from the picture from the right and various facial features coming from the face on the bottom.

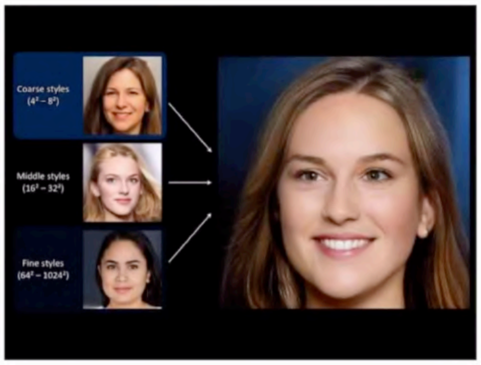
And then you see here, on the bottom right hand corner this woman is a mixing of styles from the woman above, as well as styles from the boy on the left.

Control could also mean adding accessories such as glasses and style. StyleGAN accomplishes this by disentangling the latent space which you'll be learning about in more detail here shortly.

Style in GANs

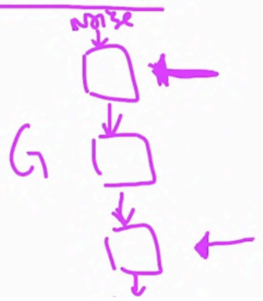
Can be any variation in the image, from larger, coarser styles to finer, more detailed styles

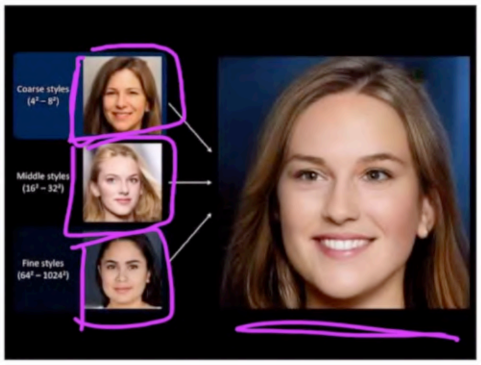
The word style in the context of image generation is pretty much any variation in the image. And you can think of these variations as generally representing the look and feel of different levels in the image. And these different levels could mean larger, coarser styles such as general face shape or facial structure to finer, more detailed styles which are like color their hair or placement of certain wisps of hair.


Coarse styles ($4^2 - 8^2$)
Middle styles ($16^2 - 32^2$)
Fine styles ($64^2 - 1024^2$)

Style in GANs

Can be any variation in the image, from larger, coarser styles to finer, more detailed styles

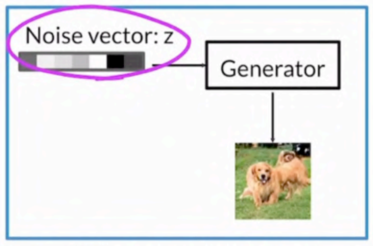



Coarse styles ($4^2 - 8^2$)
Middle styles ($16^2 - 32^2$)
Fine styles ($64^2 - 1024^2$)

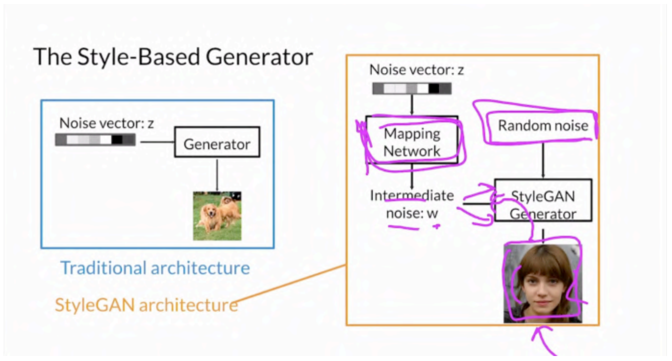
The StyleGAN generator, Interestingly enough, is made up of blocks where earlier blocks roughly aligned with coarser features, like facial structure or pose. So the earlier layers here and nearer to the output, finer details styles such as hair color, eyebrow shape, are controlled.

So you can imagine this woman here could be composited or change in some ways based on core styles from the woman on top medium styles from the woman in the middle and finer grain styles from the woman on the bottom here.

The Style-Based Generator



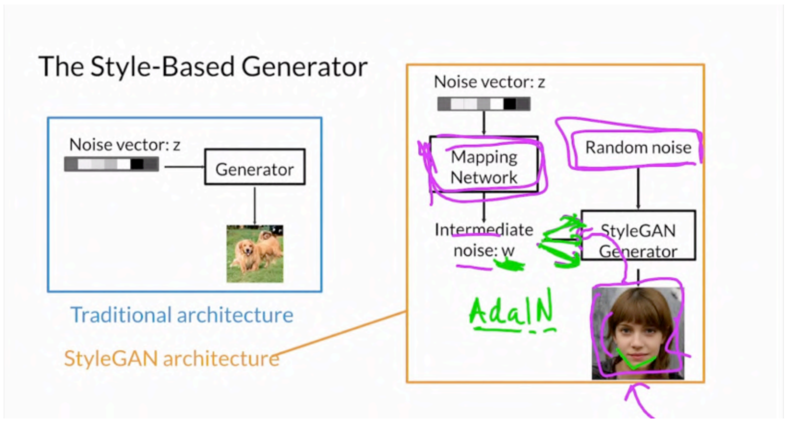
Traditional architecture



In a traditional GAN generator, you taken this noise vector into the generator and the generator then outputs an image

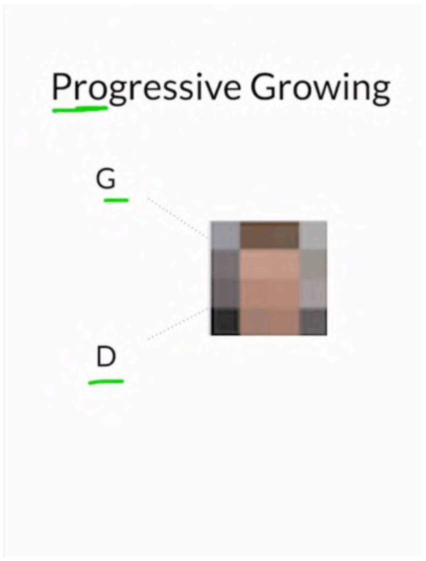
Now in StyleGAN, the noise vector is handled a little bit differently, so instead of feeding in the noise vector directly into the generator, it goes through a mapping network to get an intermediate noise vector W, say. That then gets injected into the StyleGAN generator, actually multiple times to produce this image. And also there's this extra random noise that's passed in to add various stochastic variation onto this image. And that could just be moving this wisp of hair in different ways. So smaller variation that is random noise could then help produce and for this random noise there's no learning aspect of It is largely just uncorrelated random gaussian noise. Where you perturb the values coming out of convolutions, various layers a little bit.

However, this mapping network, which you'll see shortly, is very important and consists of learnable parameters as well. So backdrop will go all the way from the discriminator, back here through the generator and through this mapping network.



And something that's important to know is that W is not just imputed easily into the StyleGAN generator. Instead, styles are extracted from the W value. This intermediate noise value and then added to various points in the StyleGAN generator and an earlier points of the StyleGAN, generator W will inform course or styles that you saw previously. Such as general face shape and injecting W into layer layers will control styles of finer grained things such as hair color. And this injection of this intermediate noise into all of these layers of StyleGAN is done through an operation called at a in or adaptive instance normalization.

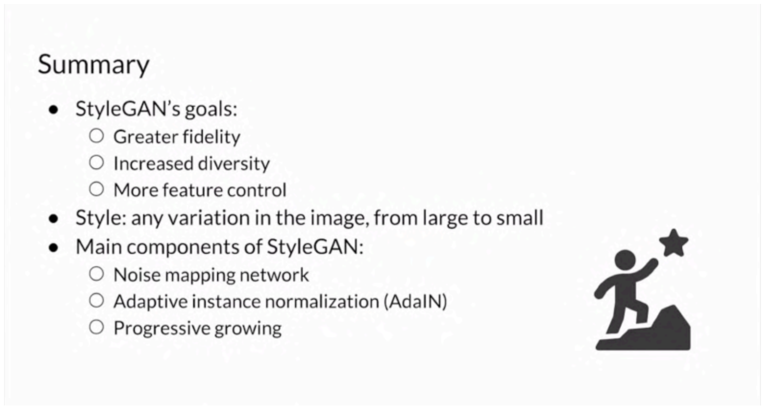
This is the type of normalization similar to batch normalization, except that after it normalizes in some way, it tries to apply some kind of style which is just statistics of an image based on this W coming in.



The 3rd and final important component of StyleGAN is progressive growing. Which slowly grows the image resolution being generated by the generator and evaluated by the discriminator over the process of training. And progressive growing originated with program organ with progressive growing. And it's not unique to StyleGAN, but it is its predecessor and the others didn't know that it did help with training on higher resolution images. So during the training process, both the generator and discriminator start with a small low resolution image. And the goal servers trained the generator to be able to generate something easier than a high resolution face, as in this small blurry image that's in the right ballpark. You know there's still some face pixels going on.

When the models are stable, they can then beat sanded to double the height and width. So it's slightly harder task now, slightly less blurry, needs to look a little more face like, but still easier than a super high res face.

This continues until the desired image resolution is reached. In this doubling occurs at scheduled times during the training process, but let's tricky is not. The doubling can't be too abrupt, but must be more gradual in order to ease the generator into generating those larger images. The main takeaway is that you gotta go slow if you want to grow.



Summary

- StyleGAN's goals:
  - Greater fidelity
  - Increased diversity
  - More feature control
- Style: any variation in the image, from large to small
- Main components of StyleGAN:
  - Noise mapping network
  - Adaptive instance normalization (AdaIN)
  - Progressive growing

So now you're acquainted with StyleGAN, and some of its amazing achievements. which include greater fidelity on higher resolution images, increase diversity in their outputs, and more control over image features like hair color accessories.

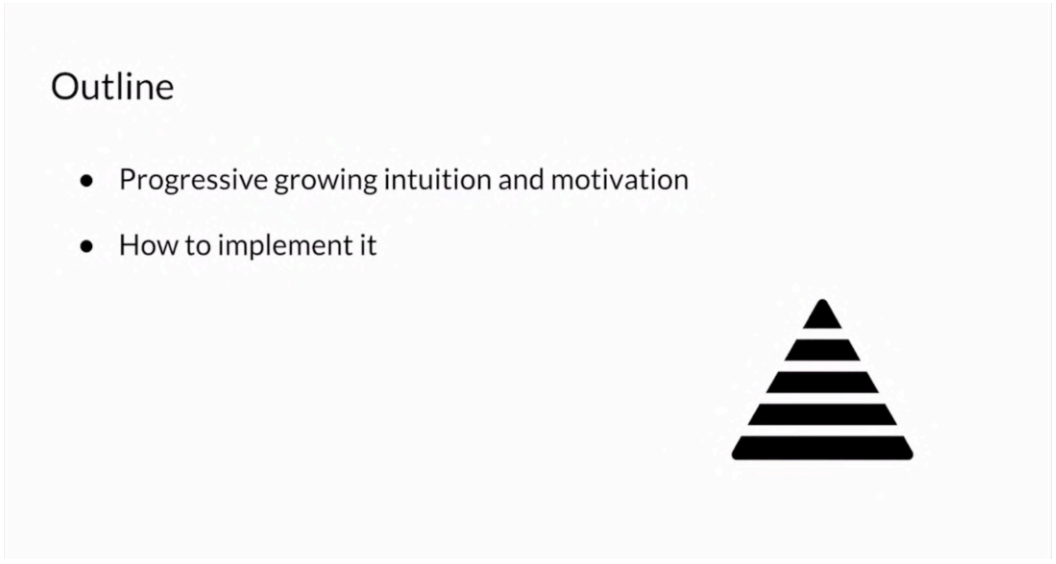
You saw a bit about what style means in the context of image generation, which is to say the general texture or look and feel of different levels of the image. And these vary depending on their location in the generator. From large core styles like the shape of a face to finer styles like hair color.

Finally you were introduced to the StyleGAN architecture, and you've seen how its main components that noise mapping network, adaptive instance normalization set it apart from a more traditional GAN.

Progressive growing isn't limited to use in StyleGAN, but it does help with more stable training of higher resolution images. And this was just a high level introduction you're about to get a deeper dive into each of these components and how they work next.

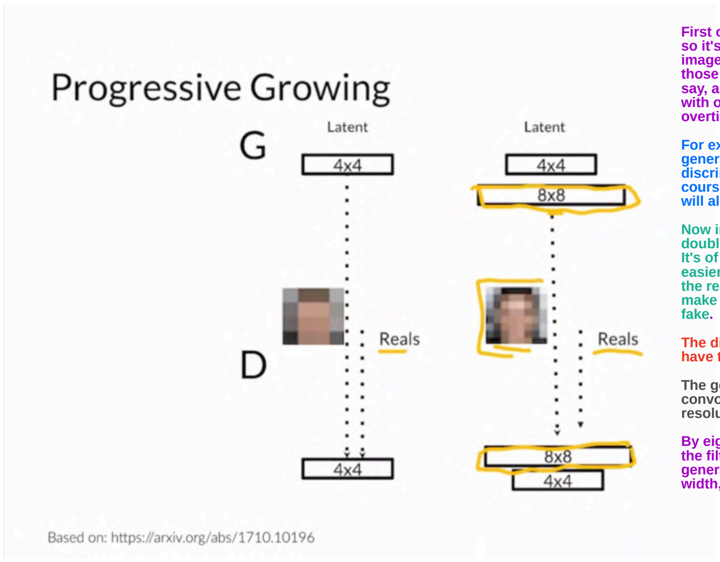
Progressive Growing

deeplearning.ai



Outline

- Progressive growing intuition and motivation
- How to implement it



Progressive Growing

Based on: https://arxiv.org/abs/1710.10196

First off, progressive growing and StyleGAN is trying to make it so it's easier for the generator to generate higher resolution images by gradually training it from lower resolution images to those higher resolution images. Starting with an easier task of say, a very blurry image for it to generate a four by four image with only 16 pixels to then a much higher resolution image overtime.

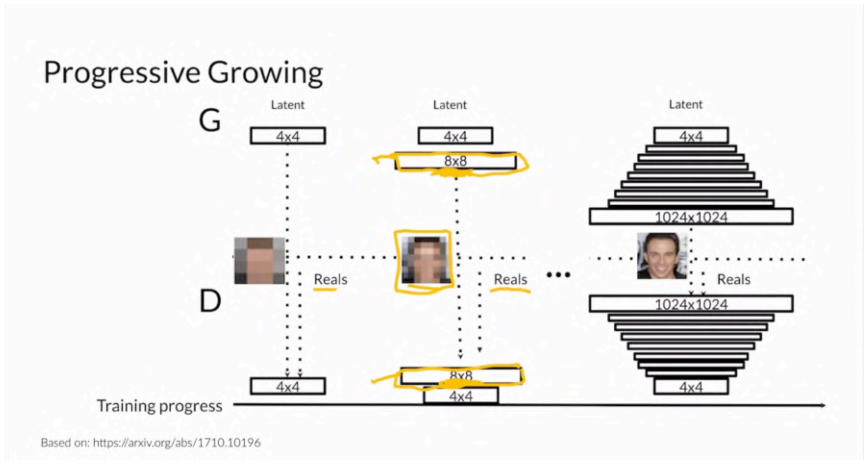
For example, this is what it would start out as where the generator just needs to generate this four by four image and the discriminator needs to evaluate whether it's real or fake. Of course, to make it not so obvious what's real or fake, the reals will also be downsampled to a four by four image.

Now in the next step of progressive growing, everything is doubled, so the image now generated is an eight by eight image. It's of much higher resolution image than before, but still an easier task than a super high resolution image, and of course, the reals are also down-sampled to an eight by eight image to make it not so obvious which ones are real and which ones are fake.

The discriminator will now taken this eight by eight image and have this extra convolutional layer to take that in.

The generator conversely, will also have this eight by eight extra convolutional layer that will then be able to generate this higher resolution image.

By eight by eight here for the convolutional layer, I don't mean the filter size, I actually mean the expected output here on generator and input here for the discriminator of heightened width, we're not looking at channels here.

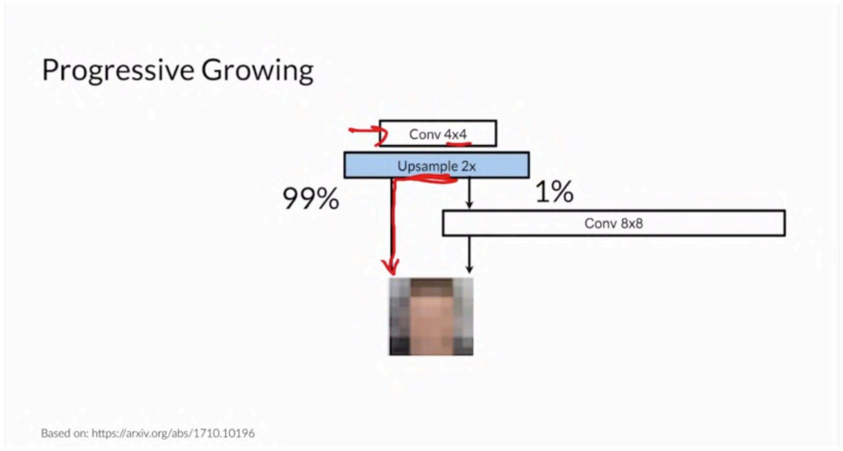


Progressive Growing

Based on: https://arxiv.org/abs/1710.10196

Then finally, over time, over and perhaps lots of different growing periods and these are scheduled intervals during training. The generator will be able to generate super high-resolution images and the discriminator will look at that higher resolution image against real images that will also be at this high resolution, so no longer downsampled and be able to detect whether it's real or fake.

Progressive Growing in Action

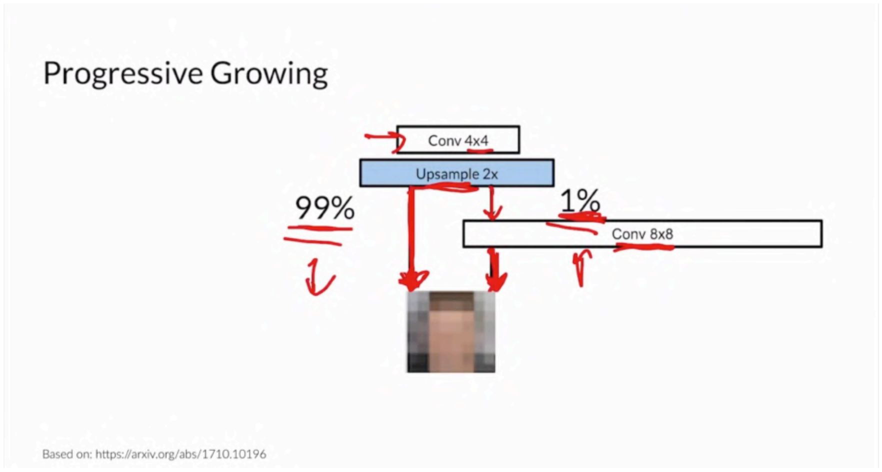


Progressive Growing

Based on: https://arxiv.org/abs/1710.10196

Now this progressive growing isn't as straightforward as just doubling in size immediately at this scheduled intervals, it's actually a little bit more gradual than that.

What happens initially when you first want to generate a doubled size image is that instead of a four by four image here, where I'm saying conv four by four is a height width. You actually want to upsample the image and upsampling, that could be using nearest neighbors filtering. These are not learned parameters, it's just very basic upsampling, and all you do is you upsample it to this image here.



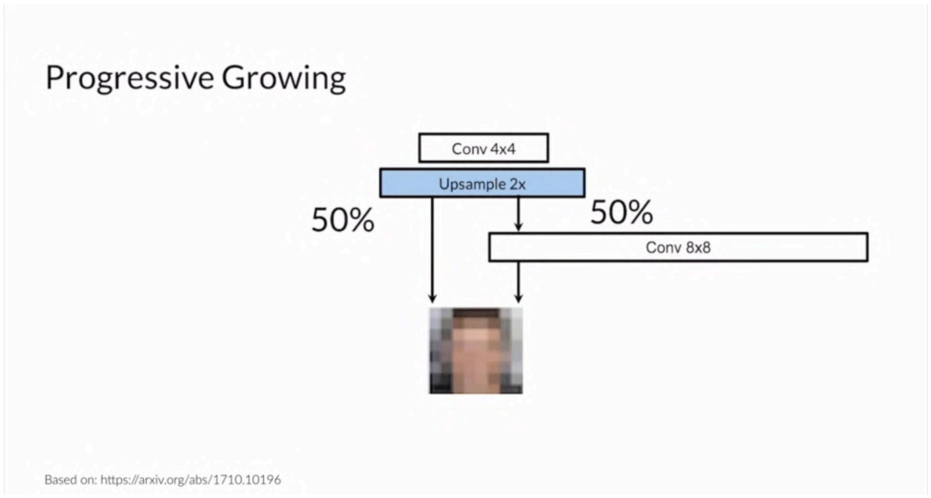
Progressive Growing

Based on: https://arxiv.org/abs/1710.10196

All you're doing is upsampling and then at the next step, maybe you do 99 percent upsampling and you do one percent of taking this upsampled image into a convolutional layer that produces an eight by eight resolution image so that you have some learned parameters, but you only weight this direction by one percent.

You do all this learning and then you still way is image as 99 percent from the upsampling purely and one percent from these learned parameters.

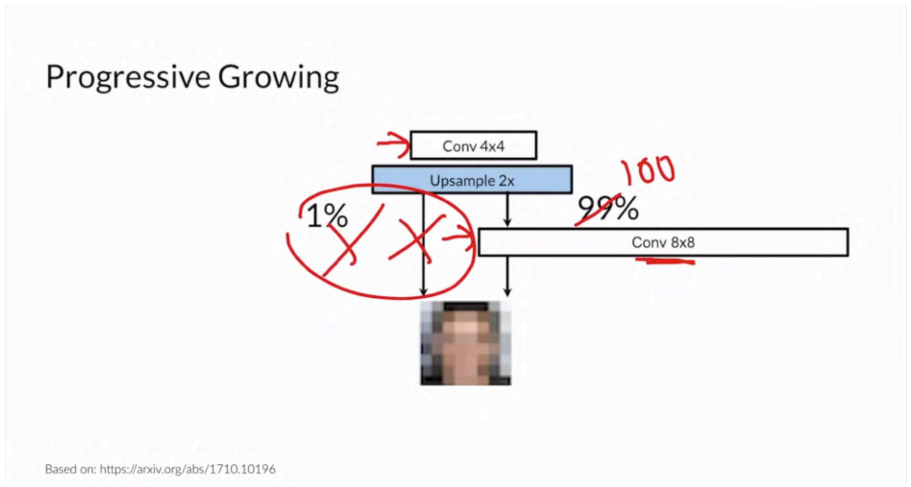
Then over time, you start to decrease, it's 99 percent and you want to increase this one percent, so you rely more and more on these learned parameters.



Progressive Growing

Based on: https://arxiv.org/abs/1710.10196

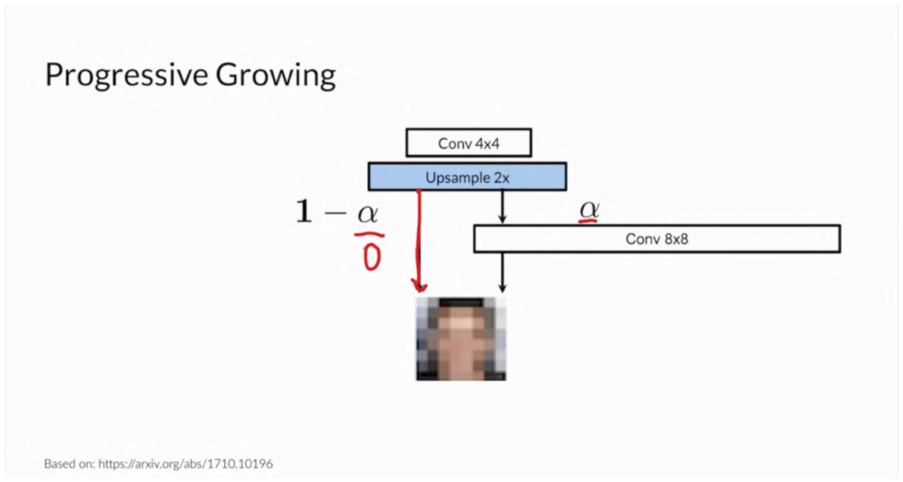
Let's say you have 50 percent, 50-50, and you're relying more on learned parameters and you see this image starting to look perhaps more like a face and less like just upsampling from nearest Neighbors upsampling



Progressive Growing

Based on: https://arxiv.org/abs/1710.10196

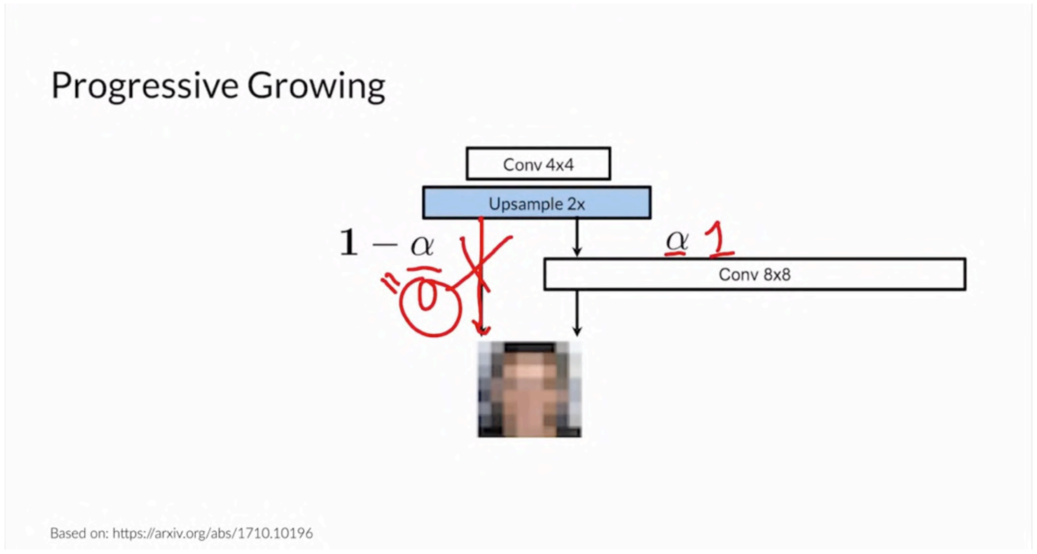
Then over time even more until you completely do not rely on this upsampling by itself and you just rely on these learned parameters. At some point this will be a 100 percent and this will just be part of the network that you see like this guy. This will just be part of the network and this section will just not be there anymore.



Progressive Growing

Based on: https://arxiv.org/abs/1710.10196

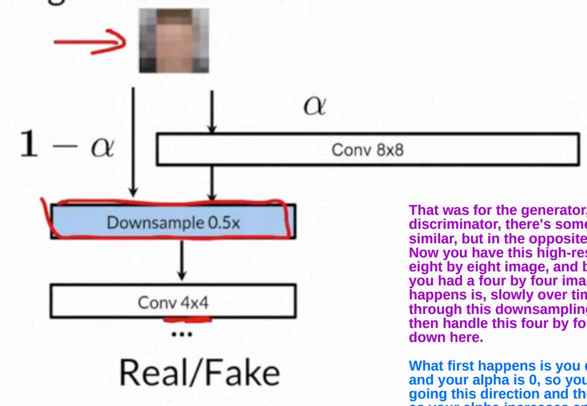
More generally, you can think of this as an alpha parameter that grows over time where alpha starts out as 0 and this is just completely one, so completely this way. Then alpha grows like 0.1, 0.2, 0.3 all the way up to one. When an alpha is one then this will equal 0, this whole thing over here, so you don't care about this, and this will equal one.



Progressive Growing

Based on: https://arxiv.org/abs/1710.10196

Progressive Growing: Discriminator



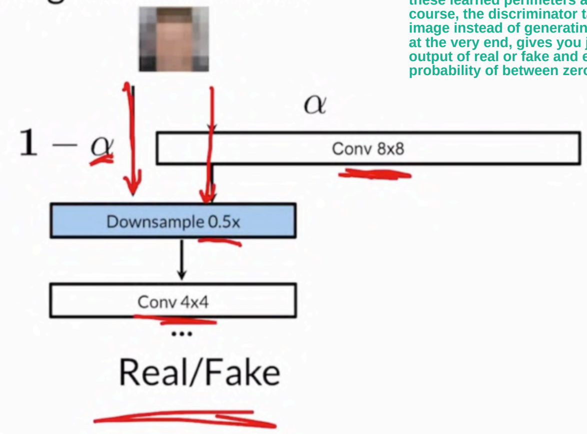
Based on: https://arxiv.org/abs/1710.10196

That was for the generator, for the discriminator, there's something fairly similar, but in the opposite direction. Now you have this high-resolution eight by eight image, and before maybe you had a four by four image. What happens is, slowly over time, you go through this downsampling layer to then handle this four by four image down here.

What first happens is you downsample and your alpha is 0, so you're only going this direction and then over time as your alpha increases and this is the same alpha perimeter as you saw in the generator, you'll go through these learned parameters with a comv eight by eight, taken as input and then downsampled here and then into your four by four.

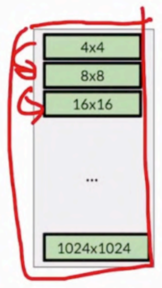
Essentially, what's happening in the discriminator is the reverse and with the same alpha that gradually uses these learned perimeters as well. Of course, the discriminator takes in the image instead of generating the image at the very end, gives you just one output of real or fake and essentially a probability of between zero and one.

Progressive Growing: Discriminator



Based on: https://arxiv.org/abs/1710.10196

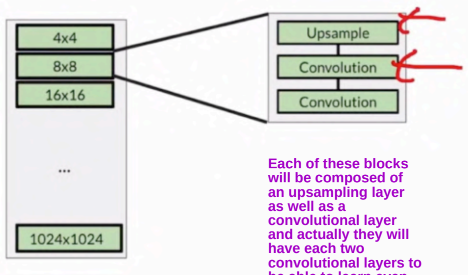
Progressive Growing in Context



Progressive growing in the context of StyleGAN essentially is a series of all of these blocks that progressively grow into higher and higher resolution outputs and the StyleGAN generator essentially will look like this.

Based on: https://arxiv.org/abs/1812.04948

Progressive Growing in Context



Each of these blocks will be composed of an upsampling layer as well as a convolutional layer and actually they will have each two convolutional layers to be able to learn even more.

Based on: https://arxiv.org/abs/1812.04948

Summary

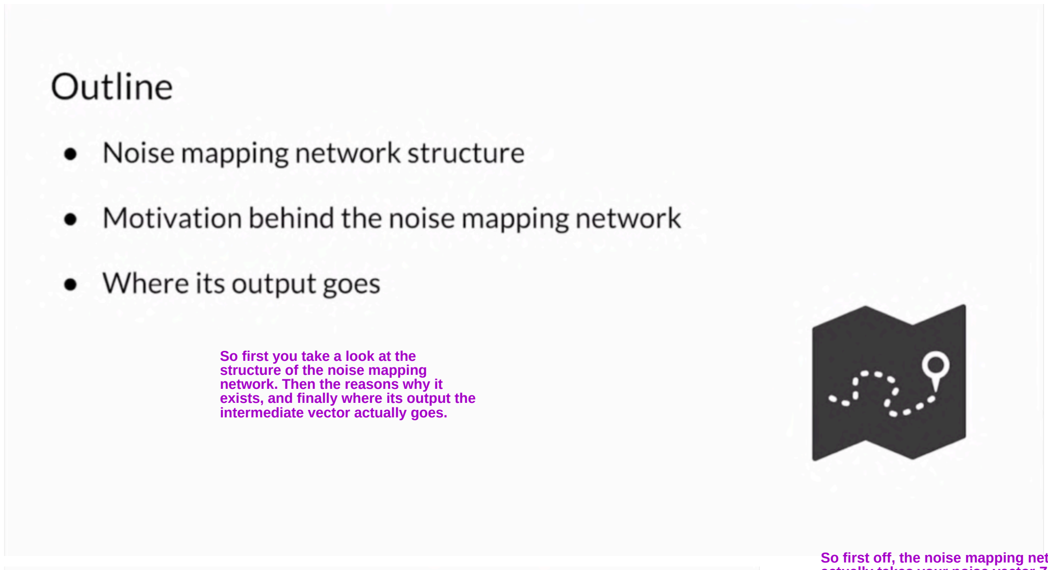
- Progressive growing gradually doubles image resolution
- Helps with faster, more stable training for higher resolutions

In summary, progressive growing gradually will double that image resolution so that it's easier for your StyleGAN to learn higher resolution images over time. Essentially, this helps with faster and more stable training.

deeplearning.ai

Noise Mapping Network

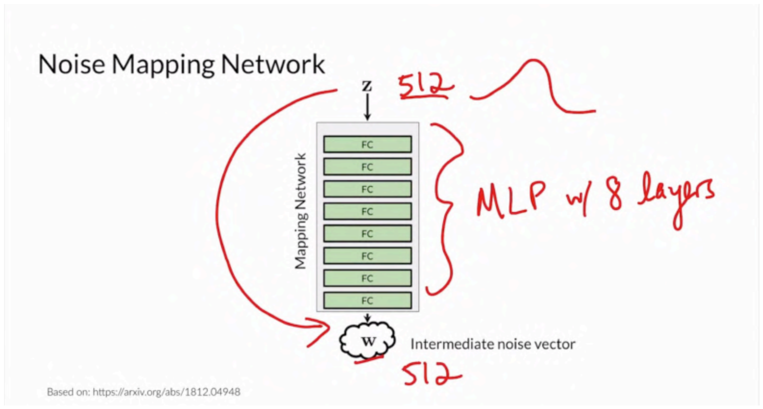
Now you'll learn about the Noise Mapping Network, one of style games, unique components. And methods for adding the noise vector as input, which will help control styles later on.



Outline

- Noise mapping network structure
- Motivation behind the noise mapping network
- Where its output goes

So first you take a look at the structure of the noise mapping network. Then the reasons why it exists, and finally where its output the intermediate vector actually goes.



Noise Mapping Network

z
512
FC
Mapping Network
MLP w/ 8 layers
w
Intermediate noise vector
512
Based on: https://arxiv.org/abs/1812.04948

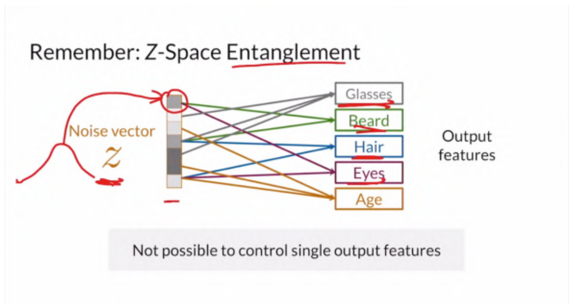
So first off, the noise mapping network actually takes your noise vector Z and maps it into an intermediate noise vector W.

Tis noise mapping network is composed of eight fully connected layers with activations in between, also known as a multilayer perceptron or MLP.

So it's a pretty simple neural network that takes your Z noise vector, which is 512 in size. And maps it onto your W intermediate noise factor, which is still 512 in size, so it just changes the values.

So you still do the same thing with Z where you sample Z from a normal distribution for all 512 values. And then you put it now through this network to get this intermediate W noise vector.

The motivation behind this is that mapping your noise vector will actually get you a more disentangled representation.



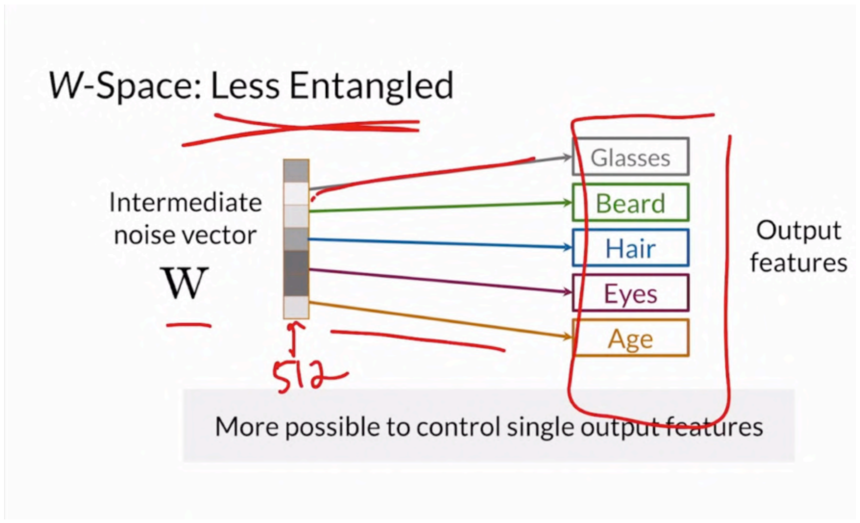
Remember: Z-Space Entanglement

So as a quick reminder, entanglement in your Z-Space is when your noise vector doesn't really map onto output features in a one-to-one way. Where when you change one of these Z vector values you can actually change a lot of different features in your output. And this is bad because this doesn't allow for a lot of fine grained control or feature level control that you would like in your resulting image. Because it wouldn't be great if you're trying to change the eyes of someone and you suddenly change their beard. Whether they have a beard at all in that image.

The reason why Z-Space where the noise vectors from is often entangled is because the real data that you know has a certain probability density. Meaning that there is a probability of the sample of certain image having glasses on, or not having glasses on. Having a beard, having a certain color of hair, eyes, or certain age associated with them. There's this probability density around the actual features.

But because Z has is normal prior, where you draw all your values from this normal distribution. It can be hard for it to map this normal distribution to the correct density distribution of all your output features. And because you're expecting random noise vectors to basically be able to model this entire space of having glasses, having a beard, what color hair, eyes. Like all different types of images, all different types of features associated with those images. If you're expecting the noise vector the Z-Space to do that, that's actually quite challenging for it. And it will try to find some way to twist itself to try to map onto these desired features. And it often twists itself in a way that probably doesn't quite make sense to us. And so this is kind of unrealistic for it to just be completely disentangled, so be completely one-to-one mapping here.

It's kind of an unrealistic thing for it to achieve, so it will probably and very necessarily learn a complex entangled set of mappings to achieve that. To be able to map on to all those output features we would like it to map onto and generate.



W-Space: Less Entangled

Intermediate noise vector W

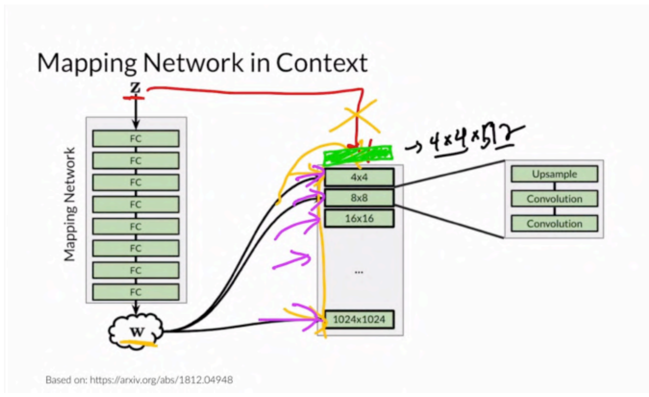
Output features

More possible to control single output features

However, having this intermediate mapping for the noise vector can give it a little bit of room. So it can allow for that space, and now this is called W-Space. To match the real data's density and this makes it easier to learn a one-to-one mapping, that's disentangled.

So essentially the noise vector doesn't need to be as fixed to the statistics of the training data. But can instead learn these factors of variation that are now linear and easier to generate from. And this will reduce the chance of your styles being too closely correlated. And this ultimately helps to control a map features. And I wouldn't say W-Space is actually this perfect, as shown in this image here.

This is just a graphic, it's just less entangled than Z-Space, is what it's been count. And as a quick note, these vectors you see here are just size seven, there are actually 512 here.



Mapping Network in Context

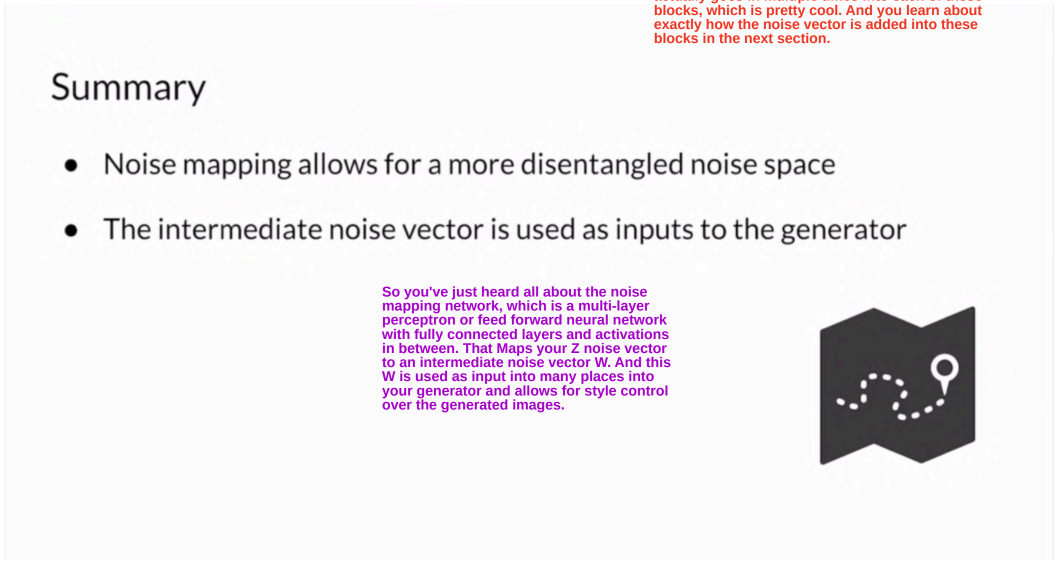
Based on: https://arxiv.org/abs/1812.04948

So where does this noise mapping network fold into the context of this dialogue in architecture? So you have learned about progressive growing, where the output doubles in size every now and then. And this is what the generator architecture kind of looks like. And the noise mapping network actually comes in here as input into all of these blocks.

So you'd probably note that were Z normally would fit in here, this is no longer the case. And instead it's going through this mapping network and W actually is inputed at multiple different places along the network. And you'll see in the next video how exactly it's added into all these places. And how W's influence differs at different points that its inputed. Its influence differs where it's inputed earlier versus later. And know that W doesn't actually go in the beginning of the network unlike what Z did in the original gan set up. And this is because the authors are style, again, actually found no noticeable difference from adding it there. So we put in W there or not was basically the same thing. So instead of inputting W at the very beginning, what the model starts with is actually just a constant value.

So with the network you actually starts with is just a constant value. So basically just a fixed value that doesn't change for any image you generate. And it's actually not just one fixed value, this is of size 4 x 4 x 512. As it goes in and then grows where it's 4 x 4 to start height width, and then a 512 channels. And again, this is the same value for every image being generated.

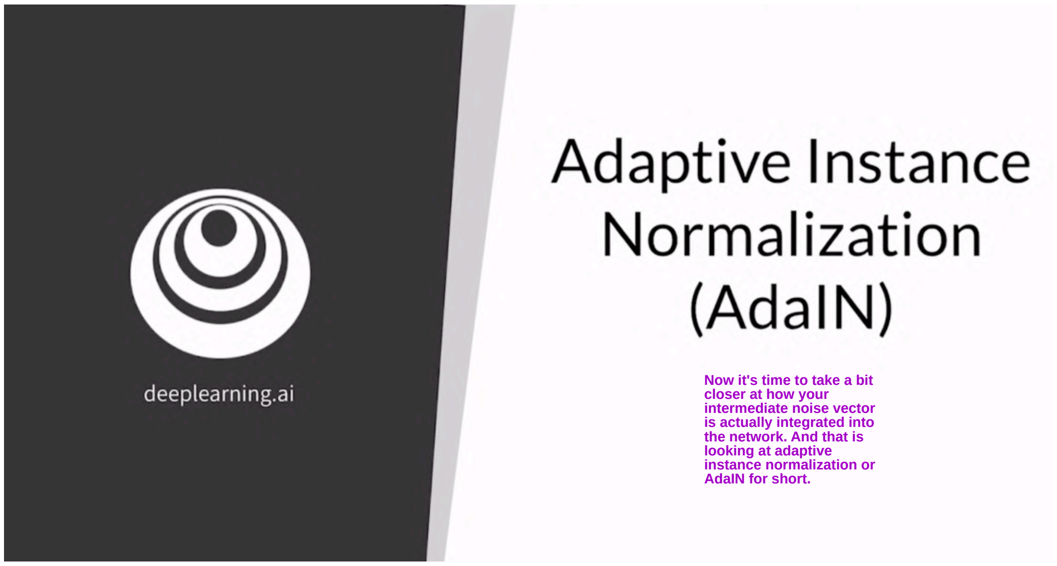
So any changes to any image being generated will occur where the noise vector is being introduced. Which is everywhere, which is all of these blocks actually in every intermediate block here. And it actually goes in multiple times into each of these blocks, which is pretty cool. And you learn about exactly how the noise vector is added into these blocks in the next section.



Summary

- Noise mapping allows for a more disentangled noise space
- The intermediate noise vector is used as inputs to the generator

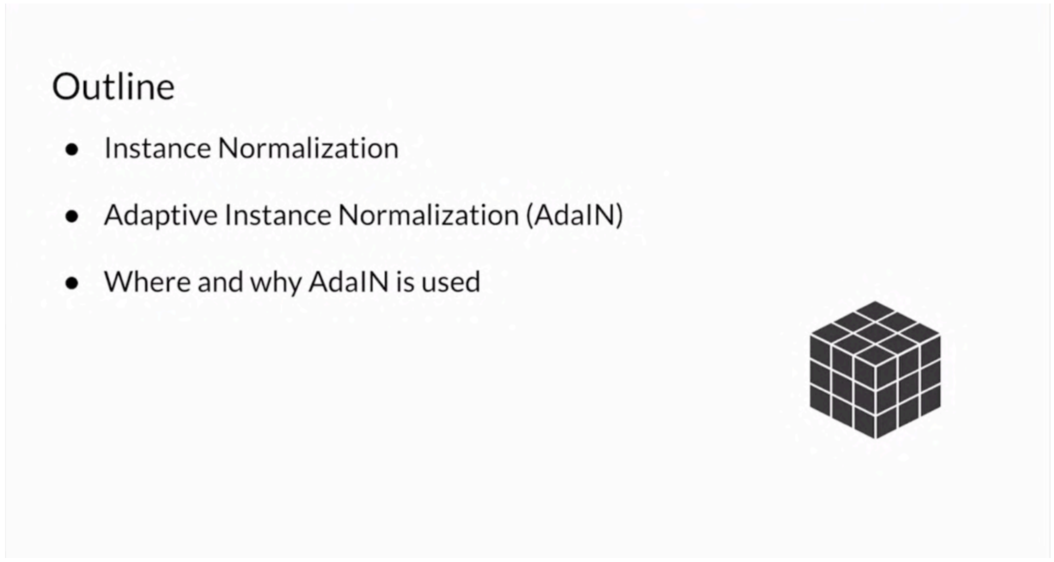
So you've just heard all about the noise mapping network, which is a multi-layer perceptron or feed forward neural network with fully connected layers and activations in between. That Maps your Z noise vector to an intermediate noise vector W. And this W is used as input into many places into your generator and allows for style control over the generated images.



deeplearning.ai

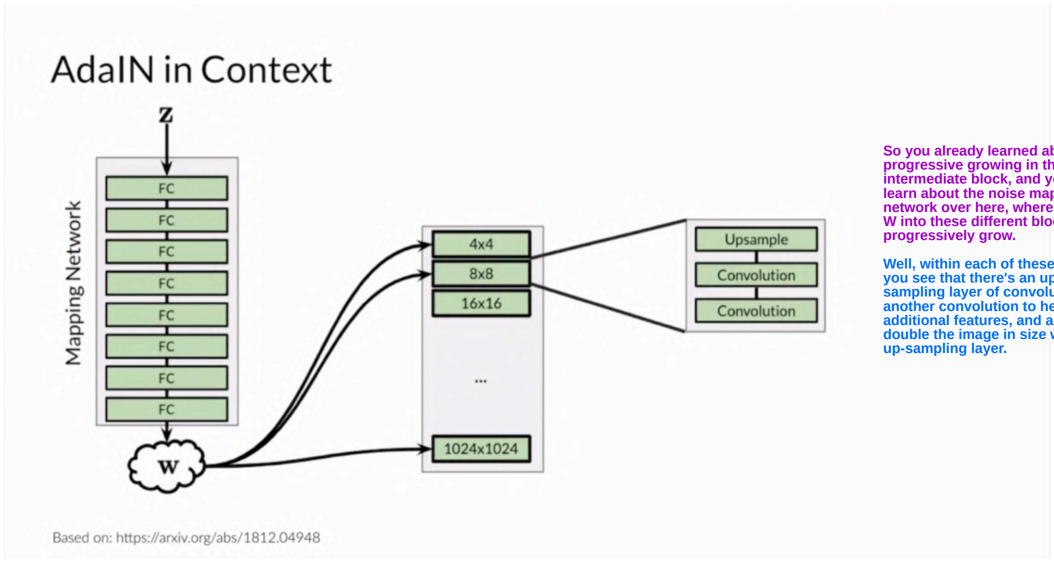
Adaptive Instance Normalization (AdaIN)

Now it's time to take a bit closer at how your intermediate noise vector is actually integrated into the network. And that is looking at adaptive instance normalization or AdaIN for short.

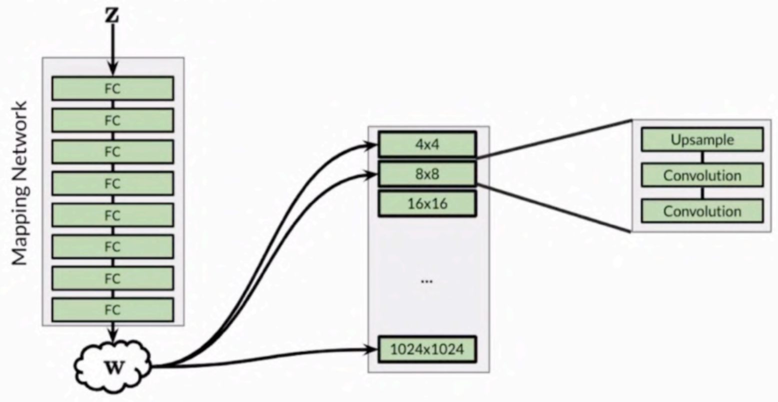


Outline

- Instance Normalization
- Adaptive Instance Normalization (AdaIN)
- Where and why AdaIN is used



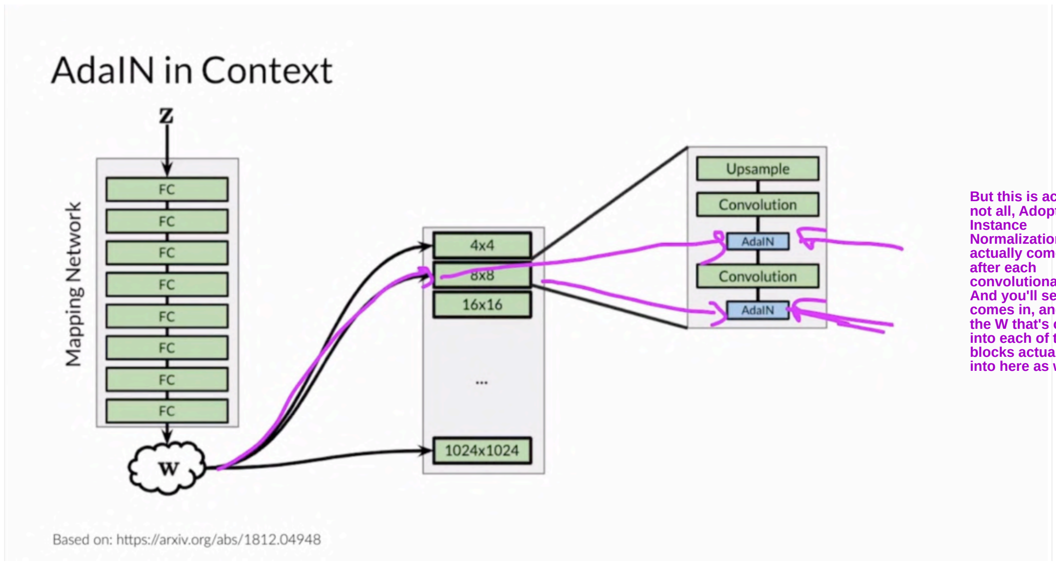
AdaIN in Context



Based on: https://arxiv.org/abs/1812.04948

So you already learned about progressive growing in this intermediate block, and you also learn about the noise mapping network over here, where it injects W into these different blocks that progressively grow.

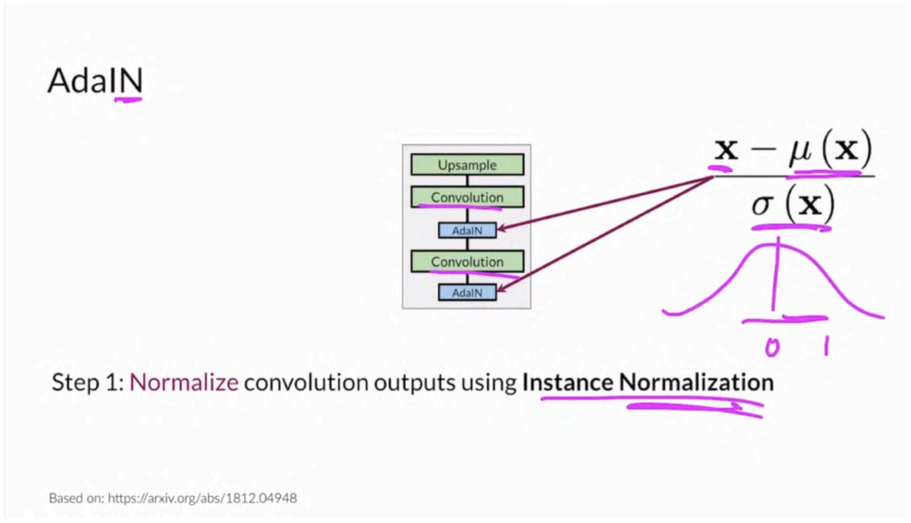
Well, within each of these blocks, you see that there's an up-sampling layer of convolution and another convolution to help learn additional features, and also to double the image in size with the up-sampling layer.



AdaIN in Context

Based on: https://arxiv.org/abs/1812.04948

But this is actually not all, Adoptive Instance Normalization, AdaIN actually comes in after each convolutional layer. And you'll see how it comes in, and how the W that's coming into each of these blocks actually goes into here as well.



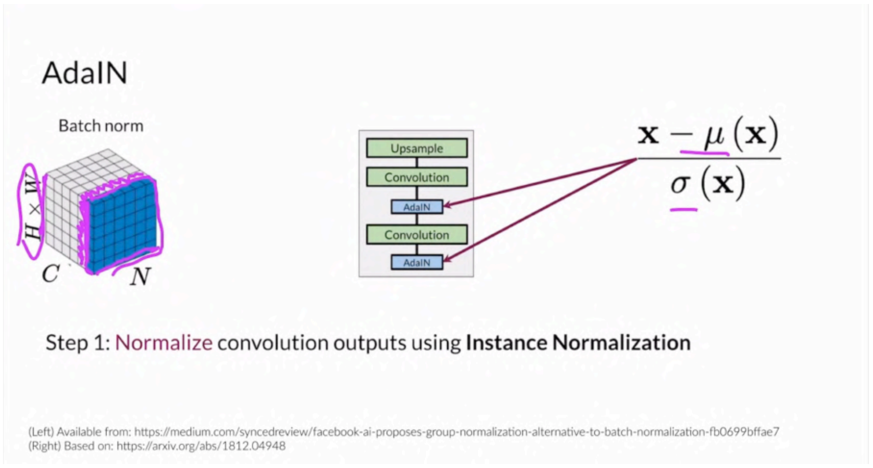
AdaIN

$$\frac{\mathbf{x} - \mu(\mathbf{x})}{\sigma(\mathbf{x})}$$

Step 1: Normalize convolution outputs using **Instance Normalization**

Based on: https://arxiv.org/abs/1812.04948

So first, let's focus on just as one block here and the first step of adoptive instance normalization will be the in part or instance normalization part. And what happens here is, well, if you remember with normalization is it takes the outputs from your convolutional layers X, and it puts it at a mean of 0 and a standard deviation of 1. And it does this by getting the mean of those values, and also the standard deviation to then center it around 0 with a standard deviation of y



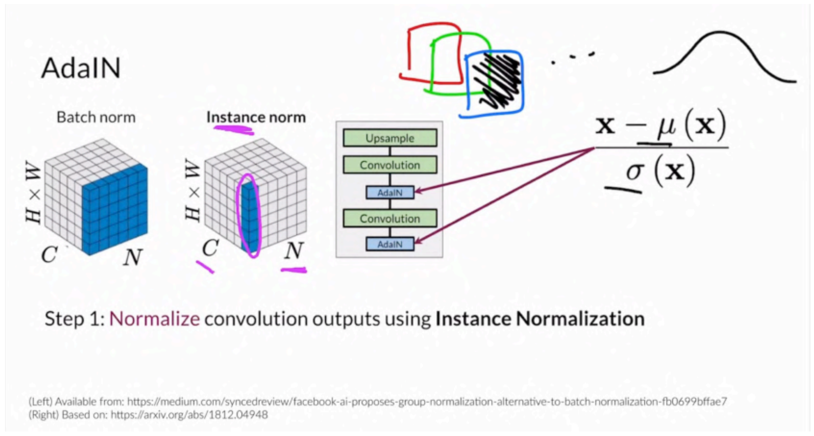
AdaIN

$$\frac{\mathbf{x} - \mu(\mathbf{x})}{\sigma(\mathbf{x})}$$

Step 1: Normalize convolution outputs using **Instance Normalization**

(Left) Available from: https://medium.com/syncedreview/facebook-ai-proposes-group-normalization-alternative-to-batch-normalization-fb0699bffae7
(Right) Based on: https://arxiv.org/abs/1812.04948

But that's not it, because it's actually not based on the batch necessarily, which you might be more familiar with batch norm. Where batch norm you look across the height and width of the image, which is along this axis that you see highlighted in blue here. You look at our channel, so among RGB, you only look at R for example, and you look at all N examples in the mini batch. And then, you get the mean and standard deviation based on all of these highlighted blue cells for one channel in one batch. And then you also do it for the next batch, and also the next channel.



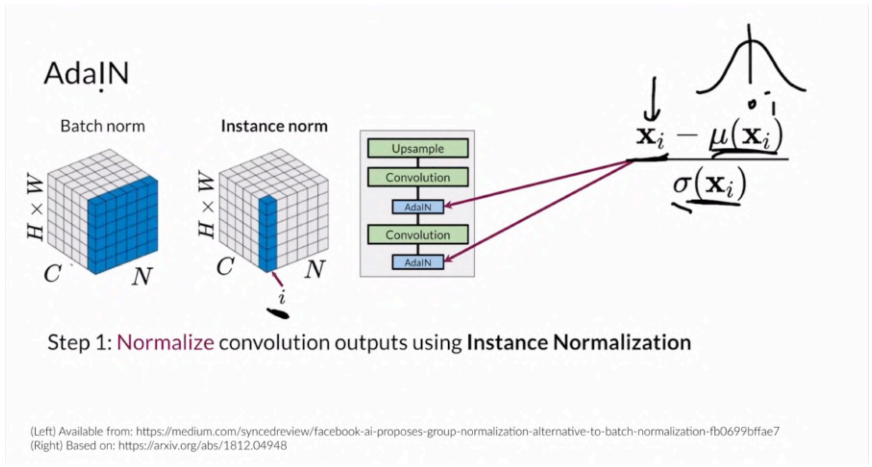
AdaIN

$$\frac{\mathbf{x} - \mu(\mathbf{x})}{\sigma(\mathbf{x})}$$

Step 1: Normalize convolution outputs using **Instance Normalization**

(Left) Available from: https://medium.com/syncedreview/facebook-ai-proposes-group-normalization-alternative-to-batch-normalization-fb0699bffae7
(Right) Based on: https://arxiv.org/abs/1812.04948

But instance normalization is a little bit different. Instance normalization, comparing it using the same graph here, actually only looks at one example or one instance, so an example is also known as an instance. So it doesn't look across statistics of the entire batch, it only looks at one example, and again only one channel of that example. So if you had an image with channels RGB, that would just be looking at, let's say the B here, which is just as blue channel, and getting the mean and standard deviation only from that blue channel. Nothing else, not additional images at all, just getting the statistics from just that one channel, one instance. In normalizing those values in there based on its mean and it standard deviation.



AdaIN

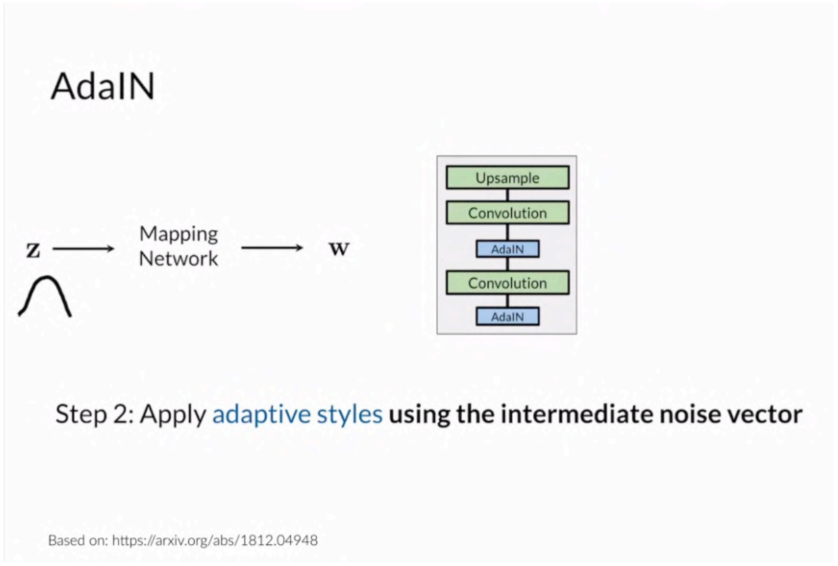
$$\frac{\mathbf{x}_i - \mu(\mathbf{x}_i)}{\sigma(\mathbf{x}_i)}$$

Step 1: Normalize convolution outputs using **Instance Normalization**

(Left) Available from: https://medium.com/syncedreview/facebook-ai-proposes-group-normalization-alternative-to-batch-normalization-fb0699bffae7
(Right) Based on: https://arxiv.org/abs/1812.04948

So to represent that in this equation, we actually call an instance i hear, so it'll be xi or that instance, and the mean or mu here over that instance as well stand deviation.

And again, here it just means every value in that instance will be normalized to a mean of 0 and a standard deviation of 1. So that's the first step of adaptive instance normalization, so that's the instance normalization part.



AdaIN

Step 2: Apply adaptive styles **using the intermediate noise vector**

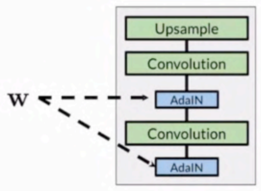
Based on: https://arxiv.org/abs/1812.04948

Where the adaptive part comes in is to ally adaptive styles to this now normalized set of values. And the instance normalization here probably makes a little bit more sense than nationalization, because it really is about every single sample you are generating, as opposed to necessarily the batch or normalizing across a batch, for example.

Okay, so the adoptive styles are coming from your intermediate noise vector w which you heard about being inputted into multiple areas of the network. And so adaptive instance normalization is where w will come in.

So you have your original vector which was sampled from a normal distribution for each of its values. It was sent through this noise mapping network that you learned about earlier, which is a multilayer perceptron to get your intermediate noise vector w.

AdaIN

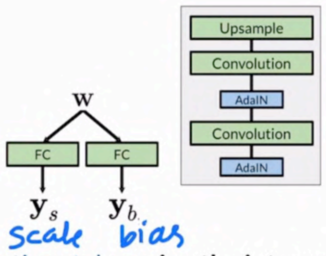


So w inform styles which are then imported into AdaIN, but actually how that happens where it informs those styles is actually not directly inputting w there.

Step 2: Apply adaptive styles **using the intermediate noise vector**

Based on: https://arxiv.org/abs/1812.04948

AdaIN

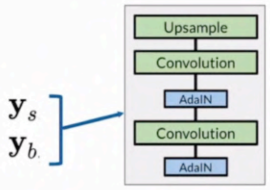


Instead it goes through learned parameters, such as two fully connected layers, and produces two parameters for us. One is ys that stands for scale, and the other is yb, which stands for bias, you might be able to guess what these terms are four. So scaling parameters for this in the bias parameter or the shift parameter is the other one

Step 2: Apply adaptive styles **using the intermediate noise vector**

Based on: https://arxiv.org/abs/1812.04948

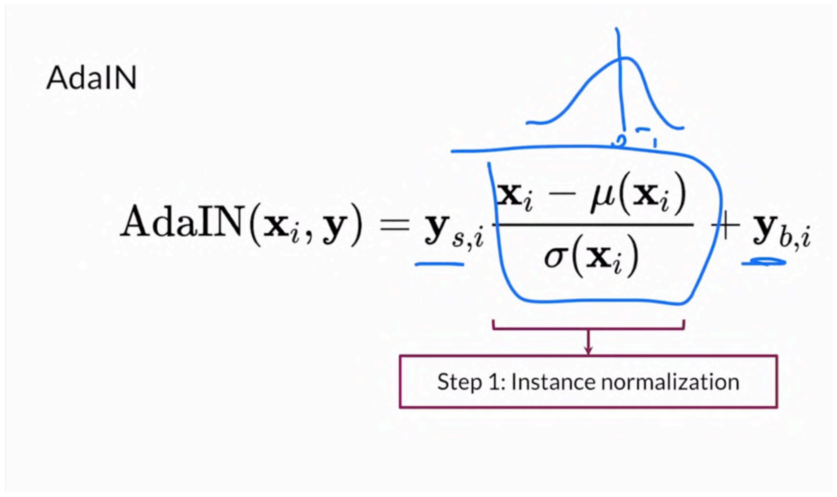
AdaIN



These statistics are then imported into the AdaIN layers. So that is put in there and then another set of these values will be put into the other one.

Step 2: Apply adaptive styles **using the intermediate noise vector**

Based on: https://arxiv.org/abs/1812.04948

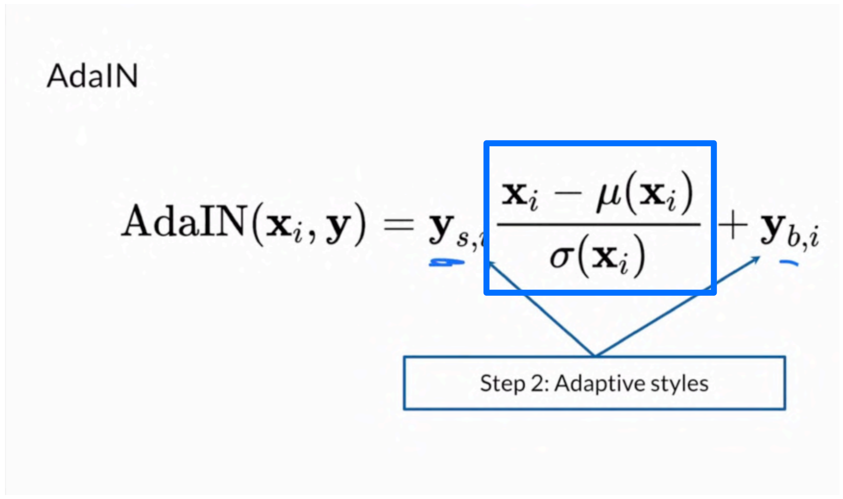


AdaIN

$$\mathrm{AdaIN}(\mathbf{x}_i, \mathbf{y}) = \mathbf{y}_{s,i} \frac{\mathbf{x}_i - \mu(\mathbf{x}_i)}{\sigma(\mathbf{x}_i)} + \mathbf{y}_{b,i}$$

Step 1: Instance normalization

So exactly how they're put in ys and yb is that after you do this instance normalization step, which is this middle part, then you want to multiply your values which are now normalized around 0 and stand deviation of 1. This is similar to batch norm by the way, then you want to reshift and rescale your values based on these statistics that are extracted from the intermediate noise factor.



AdaIN

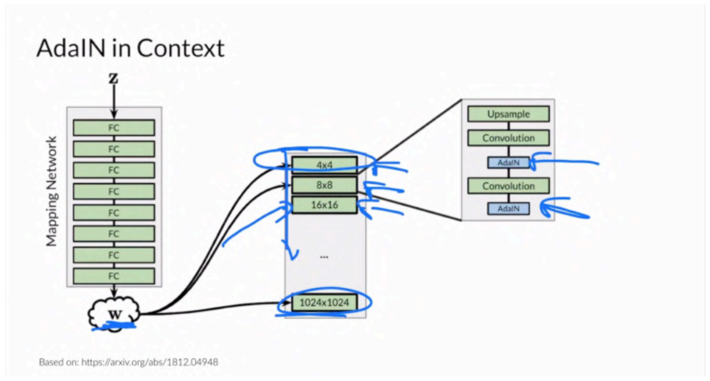
$$\mathrm{AdaIN}(\mathbf{x}_i, \mathbf{y}) = \mathbf{y}_{s,i} \frac{\mathbf{x}_i - \mu(\mathbf{x}_i)}{\sigma(\mathbf{x}_i)} + \mathbf{y}_{b,i}$$

Step 2: Adaptive styles

So the second step is getting these adoptive styles, and their adoptive, because your w can change or these values extracted will change.

So looking at this, style really comes down to just re scaling and re shifting values to a certain range, mean, and standard deviation. So you can think of it as an image that has some kind of content and with different ys and yb values. It will be like Picasso drew, or like Monet drew that image, but it will have the same content, right? It'll still be a face or a puppy field, but it would be a different style. So shifting your values to different ranges means standard deviations will actually get you those different styles.

Note that there is again i here for the instance because this is instant normalization still, so you have this ys and yb value just for that instance. And know that before you're applying these styles, essentially this middle instance normalization part is trying to undo any style related information that was originally there. So just getting some kind of content so that you can effectively ally these styles next in the second step.



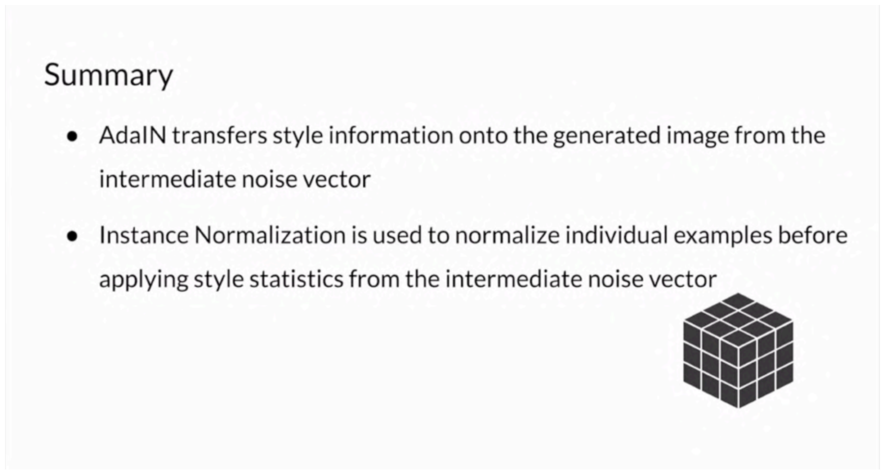
AdaIN in Context

Based on: https://arxiv.org/abs/1812.04948

All right, so zooming out of a bit, we are just adding in fit in again. Well, the generator is made up of lots of blocks where earlier blocks actually roughly aligned with coarser features and litter blocks with finer details. And this is pretty consistent across all neural networks.

So here what's interesting is that AdaIN is used at every single block here in the generator to essentially get the style information from w into those feature maps. So w is added into all of these blocks, and when it's added in into these earlier blocks, it'll be those coarser details that are changed or affected by this w style. Whereas in these later blocks, it'll be finer details that are informed by w.

Because the normalization step AdaIN, Adaptive Instance Normalization renormalizes these statistics back to a mean of 0, and a standard deviation 1. Because this occurs at every single one of these blocks, this means that every block will control styles at that block. And at the next block, it will be overwritten by the next AdaIN in the normalizes those previous outputs, and that even happens within the block you see here. And this allows for control over the model in terms of what's generated, and in terms of what's kind of style is being generated. Imagine injecting different styles at different blocks to allow for either course or finer grained controls, and that will be in the upcoming section, so w might not just be w here.



Summary

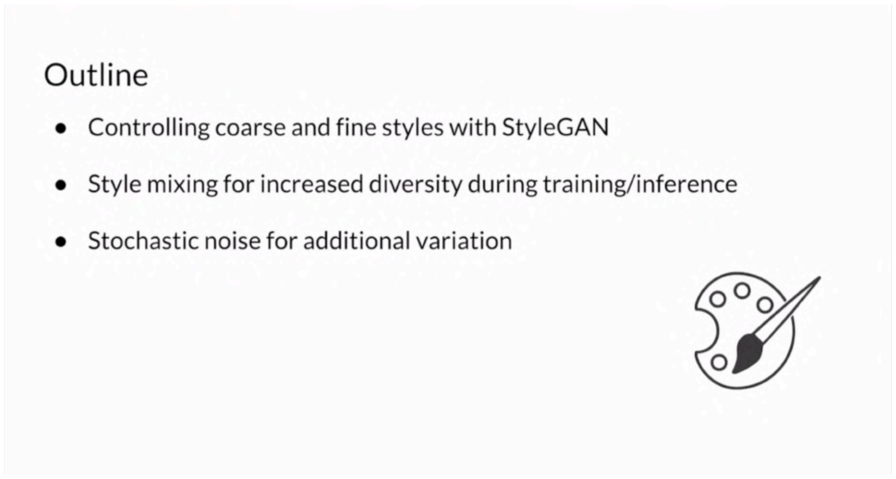
- AdaIN transfers style information onto the generated image from the intermediate noise vector
- Instance Normalization is used to normalize individual examples before applying style statistics from the intermediate noise vector

So in summary, adaptive instance normalization are what transfers style information from the intermediate noise vector w onto your generated image. Instance normalization essentially normalize each instance. While the adoptive part in adoptive instance normalization is able to ally the different styles from the intermediate noise vector w onto that image or onto that intermediate feature map. In the following video, you see exactly how the style part comes in, and maybe how to adopt your w a little bit more so that you can control

Style Mixing & Stochastic Noise

deeplearning.ai

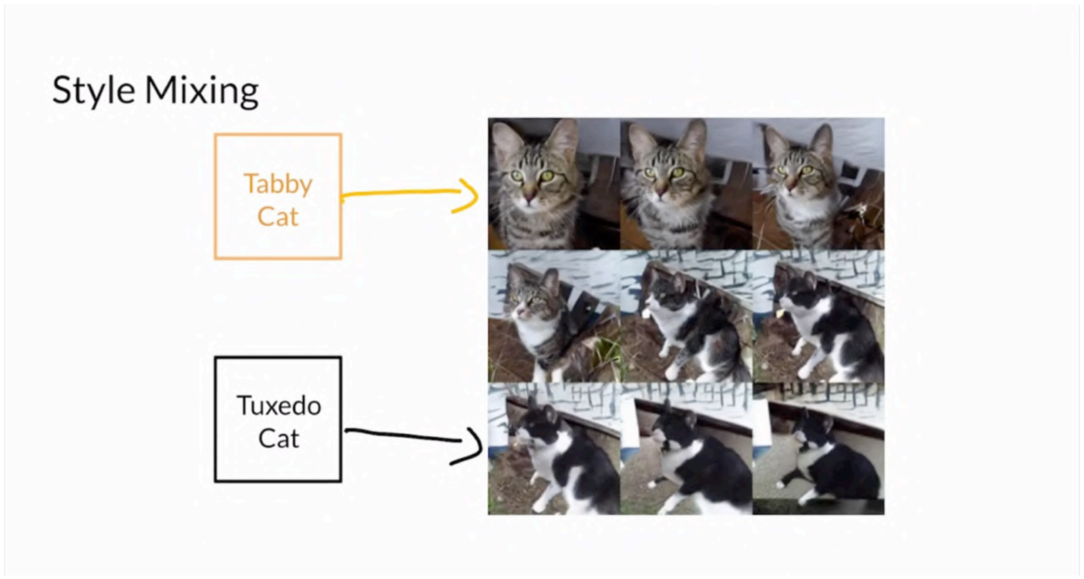
In this section, you learned about style mixing, which mixes the different intermediate noise vectors, as well as stochastic noise, which adds a little bit more extra noise into your model and your images.



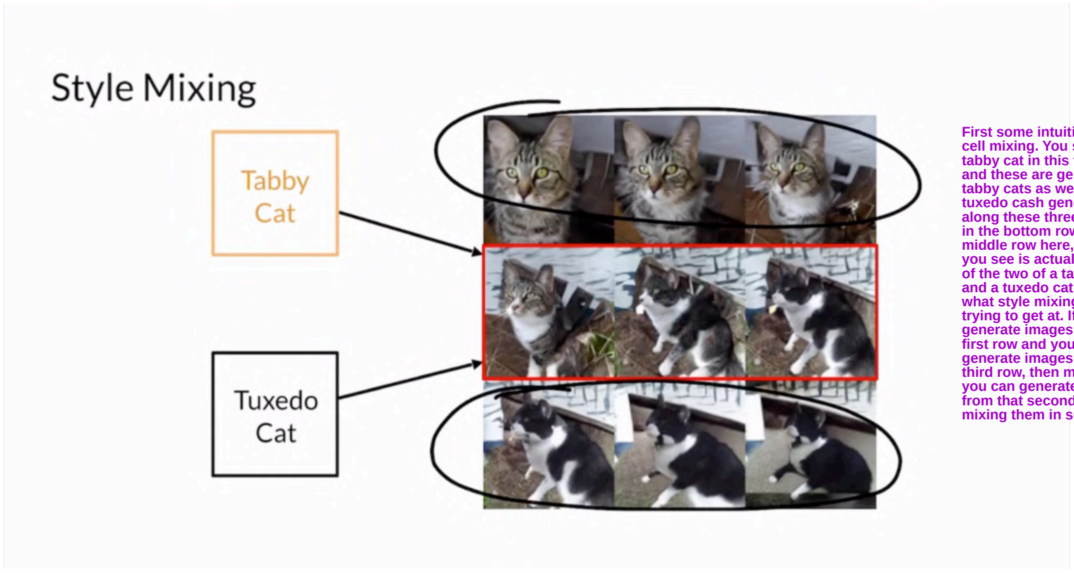
Outline

- Controlling coarse and fine styles with StyleGAN
- Style mixing for increased diversity during training/inference
- Stochastic noise for additional variation

In this section, you'll learn about controlling coarse and fine styles with StyleGAN, using two different methods. The first is style mixing for increased diversity during training and inference, and this is mixing two different noise vectors that get inputted into the model. The second is adding stochastic noise for additional variation in your images. Adding small finer details, such as where a wisp of hair grows.

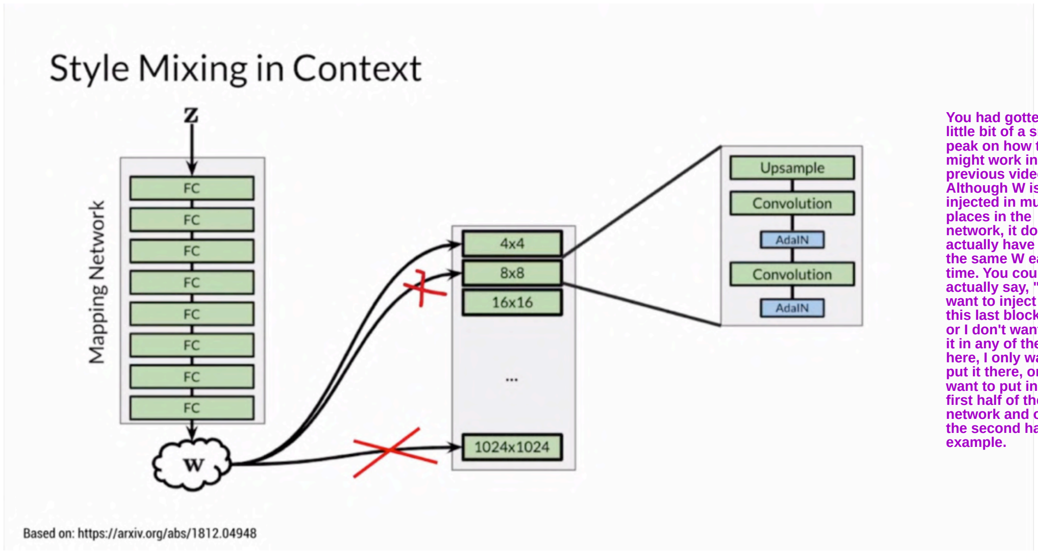


Style Mixing



Style Mixing

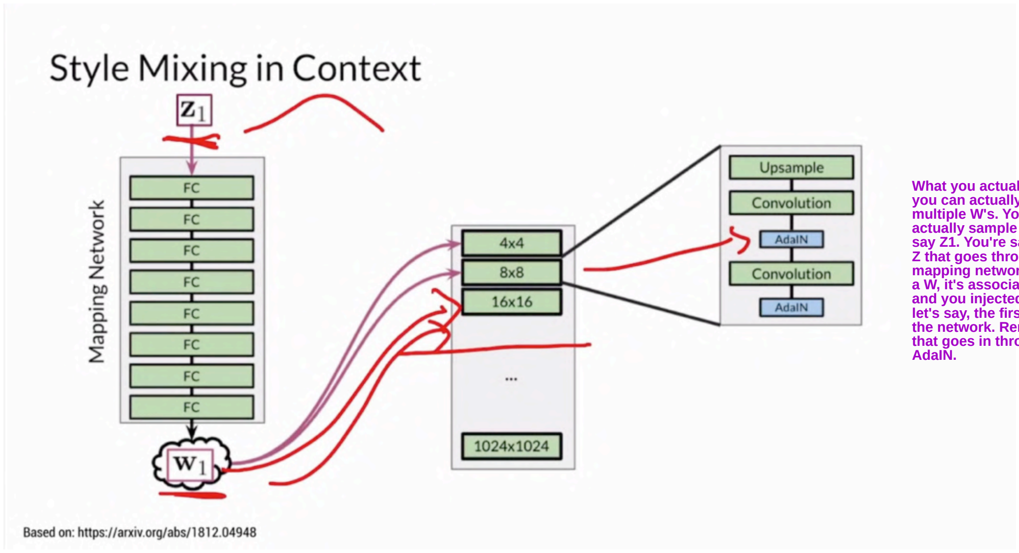
First some intuition on cell mixing. You see a tabby cat in this first row, and these are generated tabby cats as well as a tuxedo cash generated along these three images in the bottom row. In the middle row here, what you see is actually a mix of the two of a tabby cat and a tuxedo cat. This is what style mixing is trying to get at. If you can generate images from the first row and you can generate images from the third row, then maybe you can generate images from that second row by mixing them in some way.



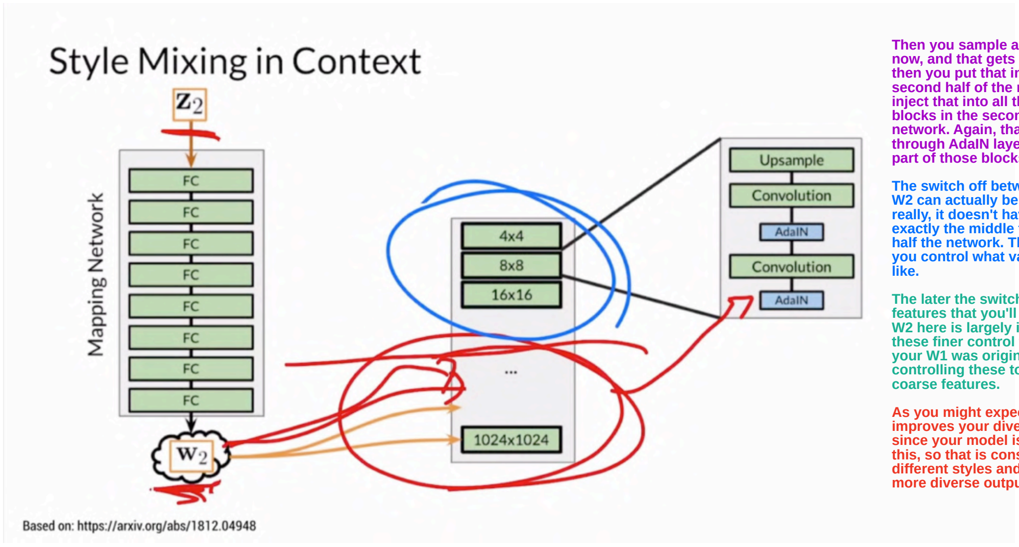
Style Mixing in Context

Based on: https://arxiv.org/abs/1812.04948

You had gotten a little bit of a sneak peak on how this might work in a previous video. Although W is injected in multiple places in the network, it doesn't actually have to be the same W each time. You could actually say, "I don't want to inject W into this last block here or I don't want to put it in any of these here, I only want to put it there, or I only want to put in the first half of the network and only in the second half," for example.



What you actually do is you can actually have multiple W's. You can actually sample a Z, let's say Z1. You're sampling a Z that goes through the mapping network, you get a W, it's associated W1, and you injected that into, let's say, the first half of the network. Remember that goes in through AdaIN.

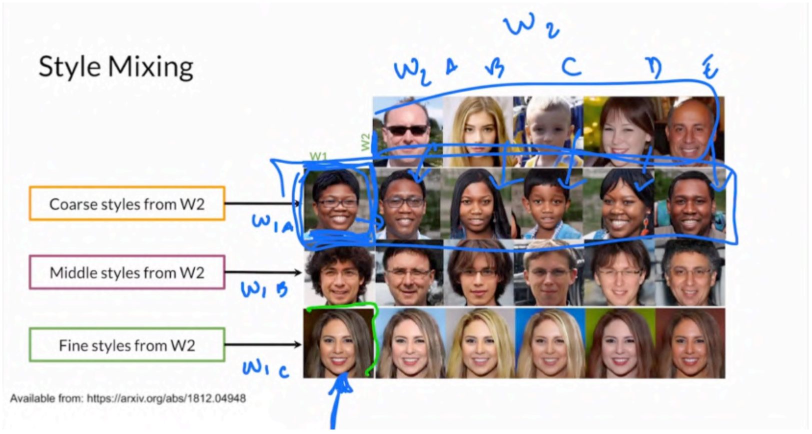


Then you sample another Z, Z2 now, and that gets you W_2, and then you put that in to the second half of the network. You inject that into all the different blocks in the second half of the network. Again, that goes through AdaIN layers that are part of those blocks.

The switch off between W1 and W2 can actually be at any point really, it doesn't have to be exactly the middle for half and half the network. This can help you control what variation you'd like.

The later the switch, the finer the features that you'll get from W2. W2 here is largely informing these finer control features, and your W1 was originally controlling these top more coarse features.

As you might expect, this improves your diversity as well since your model is trained like this, so that is constantly mixing different styles and it can get more diverse outputs.

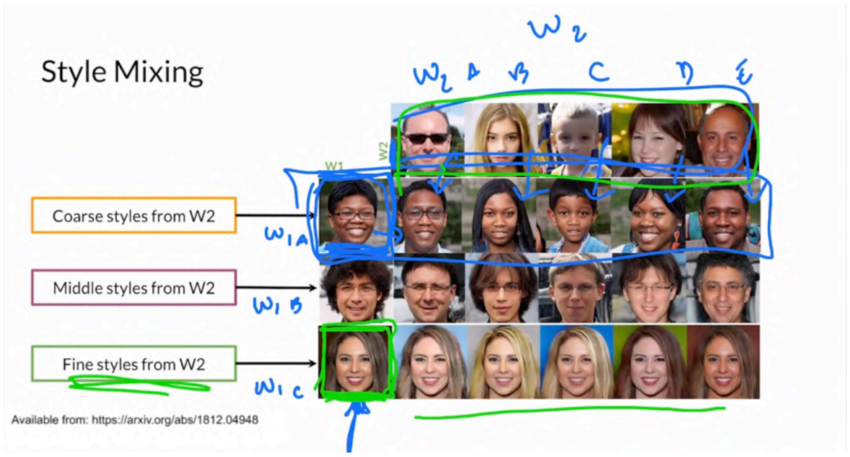


Let's take a look at what that might look like. Here's an example using generated human faces from StyleGAN. Here in this first column, are our images generated by W1, and you can imagine this being W1A, this is W1B, and this is W1C. These are definitely not the same W's generating these three images. But let's call them all W1 for now. W2 will be on this first row here, where these are all W2, A, B, C, D, E. These are all W2 here.

What's interesting is that this first row here, if you only look at this, is actually getting coarse styles from W2, so from the top row here, it's getting those coarse styles, and so you can see rough face shape and maybe some type of gender-related thing, and then it's getting finer styles and maybe middle styles from this W1 image right here", just as one imagery here, that's being applied.

Each of these are a mix of these two different intermediate noise vectors that generated those associated images from the top here, and there's one image here.

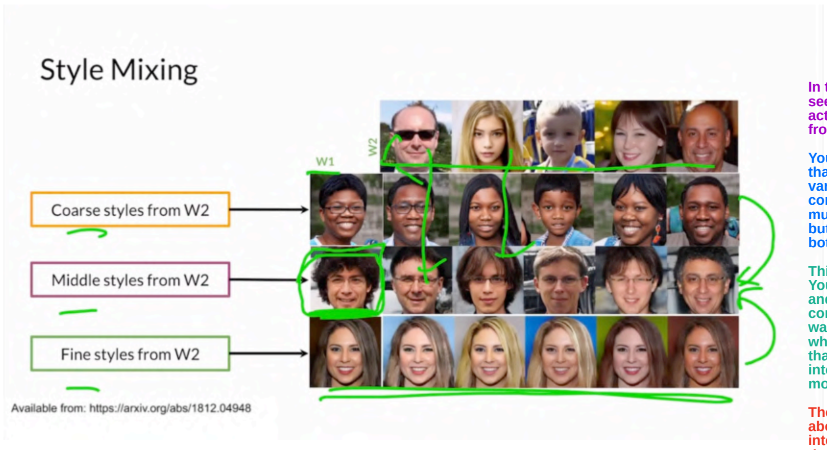
Style Mixing



Available from: https://arxiv.org/abs/1812.04948

Then if you look at this last rows, we're skipping the second row for now, if you look at this last row with this woman here, you're actually getting only fine styles from W2, so you're only getting fine styles from the top here, and what you can see is that those finer styles are just informing much smaller and slider variation to this person, and this person takes on all the coarse and even middle styles of this original vector that generated this face. These all look pretty similar.

Style Mixing



Available from: https://arxiv.org/abs/1812.04948

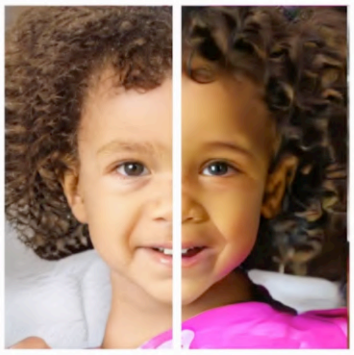
In the middle here, it's a mix of both. You see the vector that generated this image is actually being performed with middle styles from the ones up top up here.

You see you definitely much more variation than the bottom over here, but you see variation as informed by these top images compared to the first row. You see it being much more similar to this original image, but of course not so address against this bottom here.

This is what the style mixing is all about. You can input different W1, W2 vectors, and you can get these mixes and you can control the degree to which how much you want of one image versus another, and what type of styles coarse, middle or fine, that you want from each of these intermediate vectors. It's definitely much more than just coarse, middle, and fine.

These are just three broad ways of thinking about it because you're injecting your W intermediate noise vector many different times, much more than three times into your StyleGAN generator.

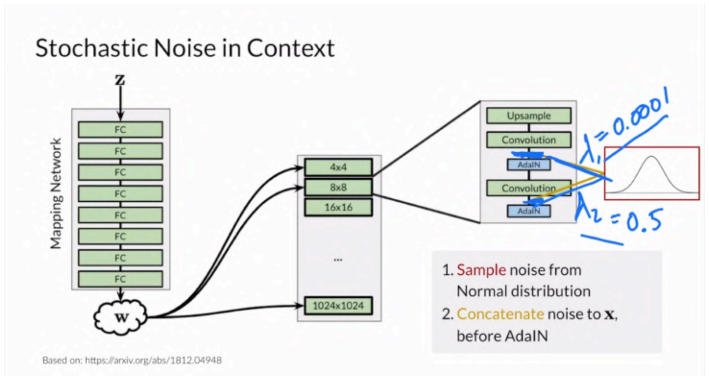
Stochastic Variation



Available from: https://arxiv.org/abs/1812.04948

That was really cool. But what about slider variations that don't require you to mix two different images, that don't require you to necessarily say, "I want the style from that person versus this person. I want to just perturb or see differences with this one thing I generated." StyleGAN has this too, which is adding this additional noise to the model, which will add stochastic variation to your image.

That's shown here. What's really cool is that this first half is injecting random noise into the finer layers, the later layers of your model, and that gets the person to have these much smaller curls in the hair and these eyebrows that are more wispy, versus injecting that noise in earlier layers of the network where there is more coarse variation that's expected from the noise, and you get these larger curls on the image and a smoother eyebrow it seems here.



Stochastic Noise in Context

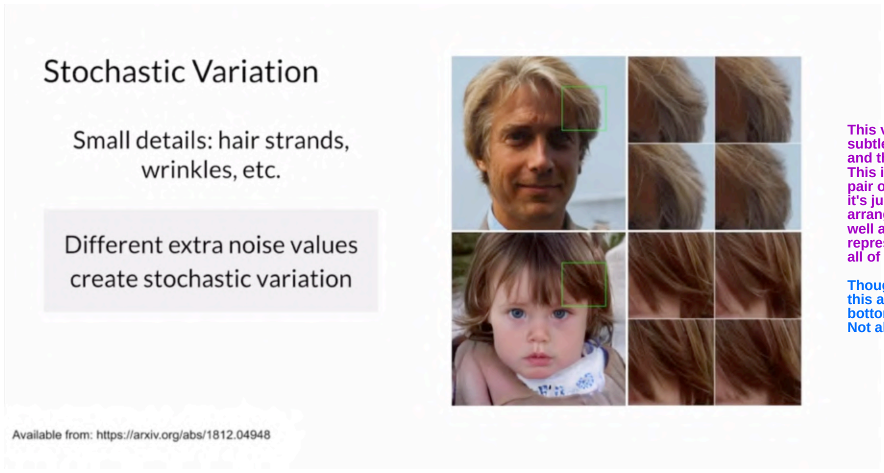
1. Sample noise from Normal distribution
2. Concatenate noise to $\mathbf{x}$, before AdaIN

Based on: https://arxiv.org/abs/1812.04948

In order to do that, is actually a separate process of injecting noise, so it has nothing to do with Z or W here. With our noises is that it'll actually be added in before your adaptive instance normalization. But first you want to sample noise from a normal distribution, so you just sample completely random values from this, and then those noise values are then concatenated to your X, which is your convolutional feature map output before it goes into adoptive instance normalization AdaIN. That's just adding some stochasticity, some randomness into your image, into those values.

The degree to which this noise actually affects these convolutional outputs is controlled by a factor, let's call it Lambda 1 here and Lambda 2. These are learned value. Lambda 1 could be, let's say 0.00001 so it doesn't affect it that much, and let's say Lambda 2 is 0.5 and, so does matter a lot, it is added a lot onto that convolutional output.

These are not real, actual values of how much it necessarily does scale it. It is a learned value in terms of how much this does help.



Stochastic Variation

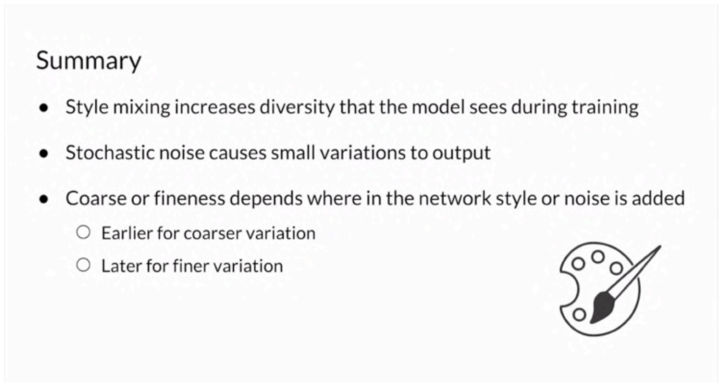
Small details: hair strands, wrinkles, etc.

Different extra noise values create stochastic variation

Available from: https://arxiv.org/abs/1812.04948

This variation can even change super subtle things that I think is so cool and that's changing this wisp of hair. This is a zoom in into this person's pair over here that's generated, and it's just so slight in terms of the arrangement of this person's hair as well as this person's hair, which represent, StyleGAN's ability to model all of that.

Though I will say and I do pick up on this a lot, is that this baby at the bottom here doesn't look very real. Not all outputs looks super real.



Summary

- Style mixing increases diversity that the model sees during training
- Stochastic noise causes small variations to output
- Coarse or fineness depends where in the network style or noise is added
  - Earlier for coarser variation
  - Later for finer variation

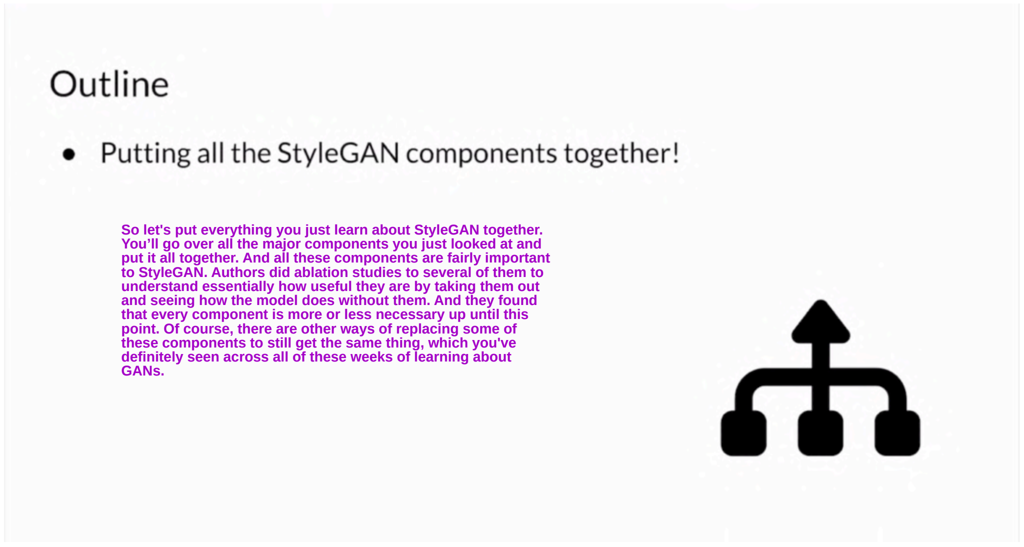
To recap style mixing with your intermediate noise vectors, can increase the diversity that the model sees during training and allow you to control coarse or finer styles.

Stochastic noise is another way to inject variation into your output.

Also the coarse or fineness depends on where in the network your style mixing or noise is added in earlier for coarser variation and later for finer variation, which is pretty consistent across neural networks, including classifiers.

deeplearning.ai

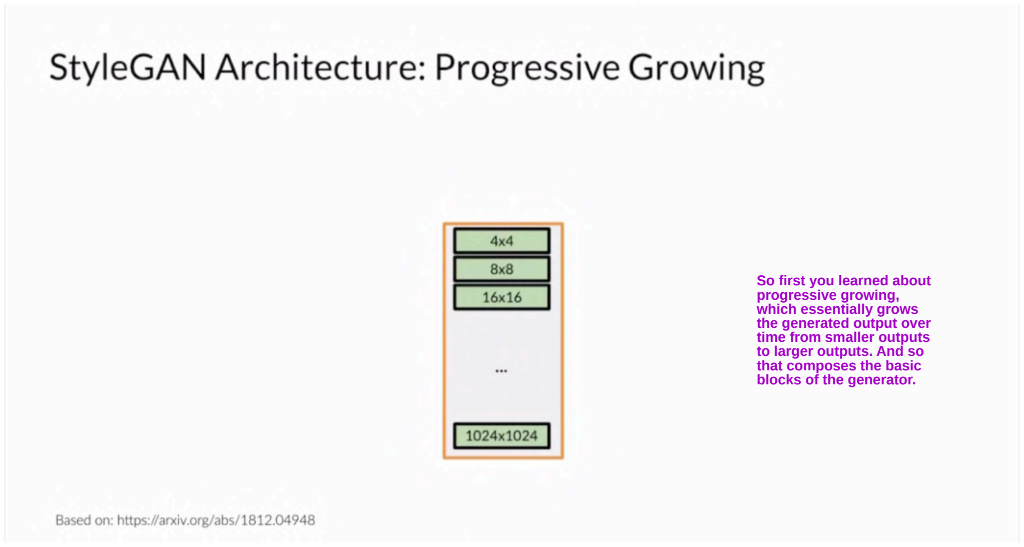
Putting It All Together



Outline

- Putting all the StyleGAN components together!

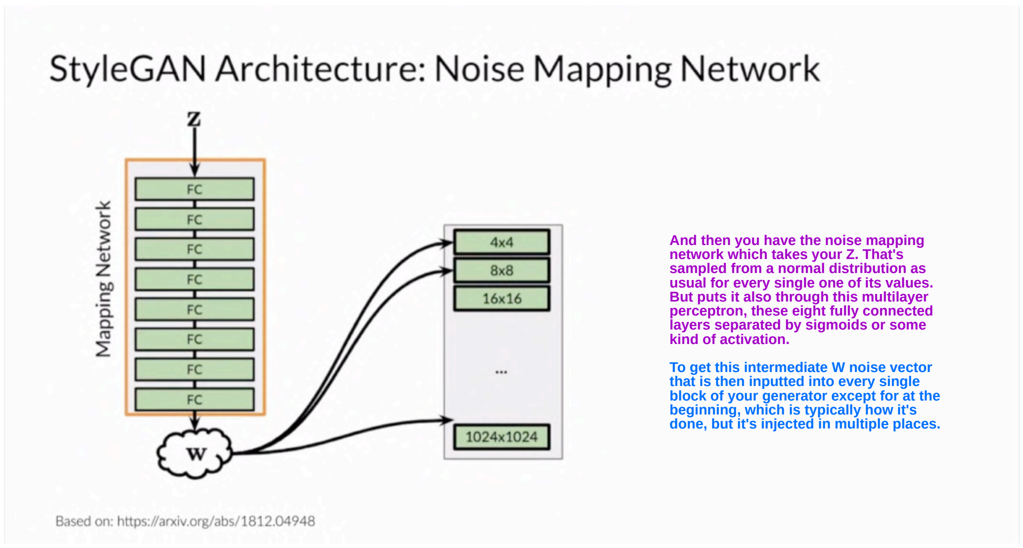
So let's put everything you just learn about StyleGAN together. You'll go over all the major components you just looked at and put it all together. And all these components are fairly important to StyleGAN. Authors did ablation studies to several of them to understand essentially how useful they are by taking them out and seeing how the model does without them. And they found that every component is more or less necessary up until this point. Of course, there are other ways of replacing some of these components to still get the same thing, which you've definitely seen across all of these weeks of learning about GANs.



StyleGAN Architecture: Progressive Growing

So first you learned about progressive growing, which essentially grows the generated output over time from smaller outputs to larger outputs. And so that composes the basic blocks of the generator.

Based on: https://arxiv.org/abs/1812.04948

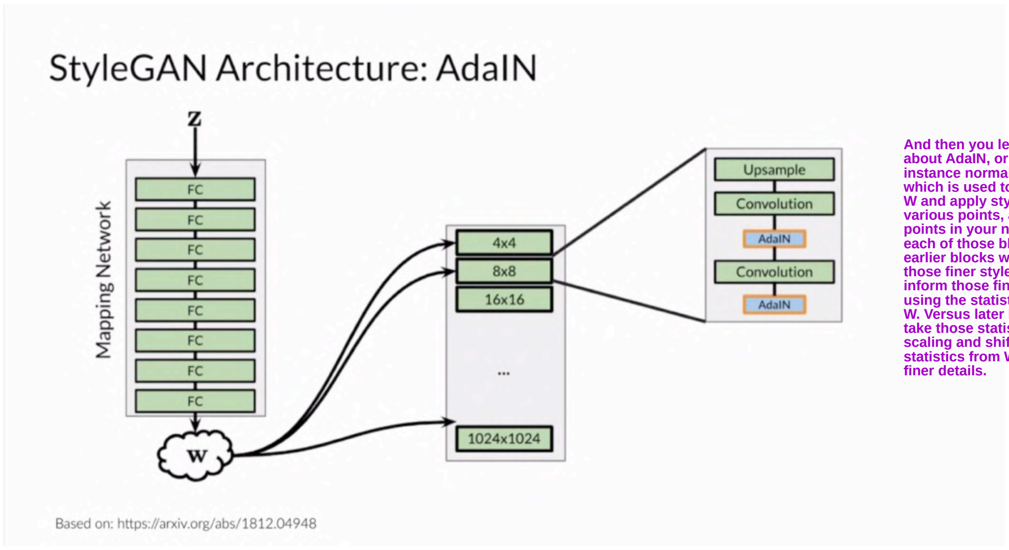


StyleGAN Architecture: Noise Mapping Network

Based on: https://arxiv.org/abs/1812.04948

And then you have the noise mapping network which takes your Z. That's sampled from a normal distribution as usual for every single one of its values. But puts it also through this multilayer perceptron, these eight fully connected layers separated by sigmoids or some kind of activation.

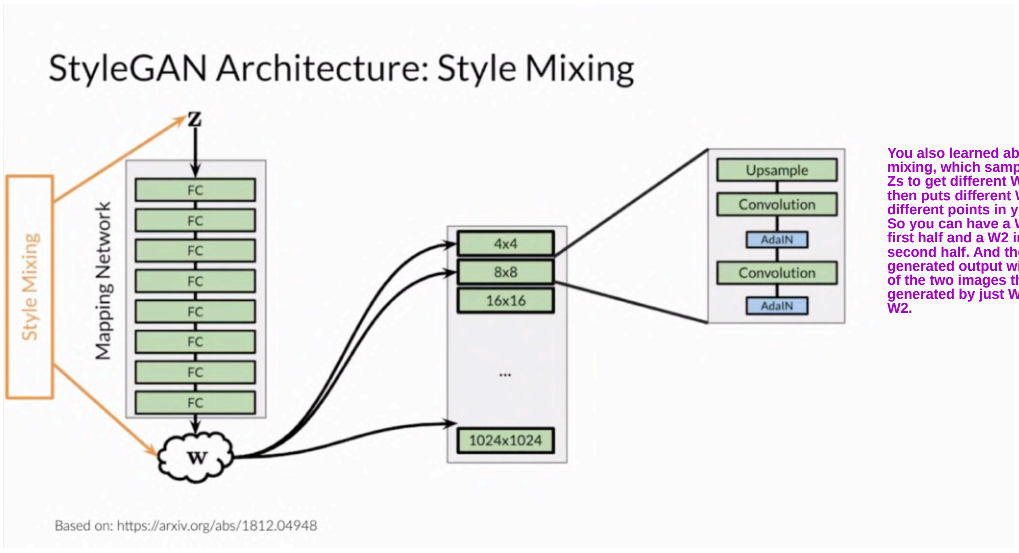
To get this intermediate W noise vector that is then inputted into every single block of your generator except for at the beginning, which is typically how it's done, but it's injected in multiple places.



StyleGAN Architecture: AdaIN

Based on: https://arxiv.org/abs/1812.04948

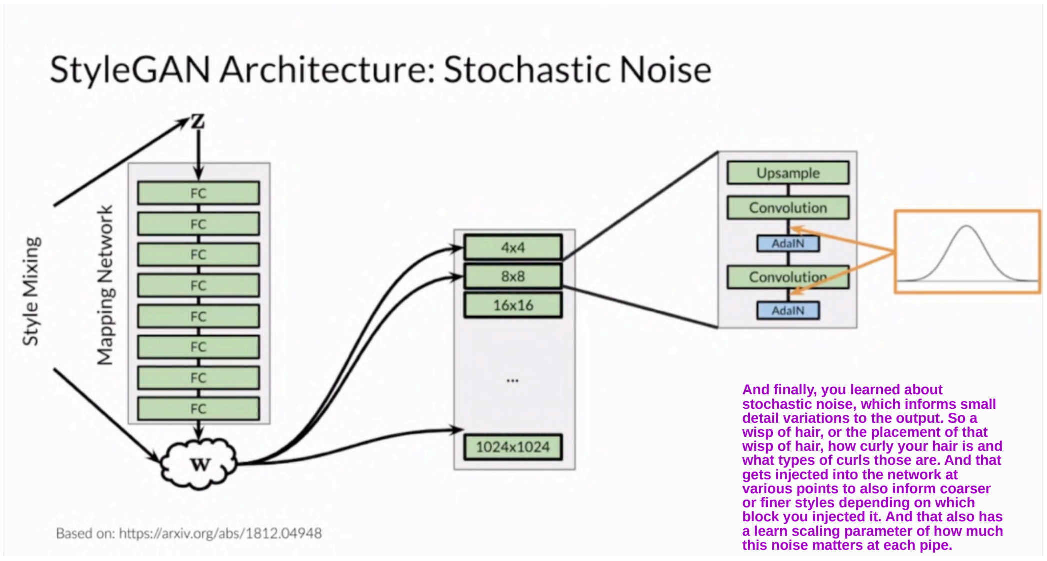
And then you learned about AdaIN, or adaptive instance normalization, which is used to take your W and apply styles at various points, at various points in your network in each of those blocks. And earlier blocks will control those finer styles or inform those finer styles using the statistics from W. Versus later blocks will take those statistics, so scaling and shifting statistics from W to inform finer details.



StyleGAN Architecture: Style Mixing

Based on: https://arxiv.org/abs/1812.04948

You also learned about style mixing, which samples different Zs to get different Ws, which then puts different Ws at different points in your network. So you can have a W1 be in the first half and a W2 in your second half. And then your generated output will be a mix of the two images that were generated by just W1 or just W2.



StyleGAN Architecture: Stochastic Noise

And finally, you learned about stochastic noise, which informs small detail variations to the output. So a wisp of hair, or the placement of that wisp of hair, how curly your hair is and what types of curls those are. And that gets injected into the network at various points to also inform coarser or finer styles depending on which block you injected it. And that also has a learn scaling parameter of how much this noise matters at each pipe.

(Optional) The StyleGAN Paper

Amazed by StyleGAN's capabilities? Take a look at the original paper! Note that it may take a few extra moments to load because of the high-resolution images.

A Style-Based Generator Architecture for Generative Adversarial Networks (Karras, Laine, and Aila, 2019): https://arxiv.org/abs/1812.04948

(Optional) StyleGAN Walkthrough and Beyond

Want another explanation of StyleGAN? This article provides a great walkthrough of StyleGAN and even discusses StyleGAN's successor: StyleGAN2!

GAN — StyleGAN & StyleGAN2 (Hui, 2020): https://medium.com/@jonathan_hui/gan-stylegan-stylegan2-479bdf256299

Works Cited

All of the resources cited in Course 2 Week 3, in one place. You are encouraged to explore these papers/sites if they interest you! They are listed in the order they appear in the lessons.

From the videos:

- Generative Adversarial Networks (Goodfellow et al., 2014): https://arxiv.org/abs/1406.2661
- Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks (Radford, Metz, and Chintala, 2016): https://arxiv.org/abs/1511.06434
- Coupled Generative Adversarial Networks (Liu and Tuzel, 2016): https://arxiv.org/abs/1606.07536
- Progressive Growing of GANs for Improved Quality, Stability, and Variation (Karras, Aila, Laine, and Lehtinen, 2018): https://arxiv.org/abs/1710.10196
- A Style-Based Generator Architecture for Generative Adversarial Networks (Karras, Laine, and Aila, 2019): https://arxiv.org/abs/1812.04948
- The Unusual Effectiveness of Averaging in GAN Training (Yazici et al., 2019): https://arxiv.org/abs/1806.04498v2
- Progressive Growing of GANs for Improved Quality, Stability, and Variation (Karras, Aila, Laine, and Lehtinen, 2018): https://arxiv.org/abs/1710.10196
- StyleGAN - Official TensorFlow Implementation (Karras et al., 2019): https://github.com/NVlabs/stylegan
- StyleGAN Faces Training (Branwen, 2019): https://www.gwern.net/images/gan/2019-03-16-stylegan-facestraining.mp4
- Facebook AI Proposes Group Normalization Alternative to Batch Normalization (Peng, 2018): https://medium.com/syncedreview/facebook-ai-proposes-group-normalization-alternative-to-batch-normalization-fb0699bffae7