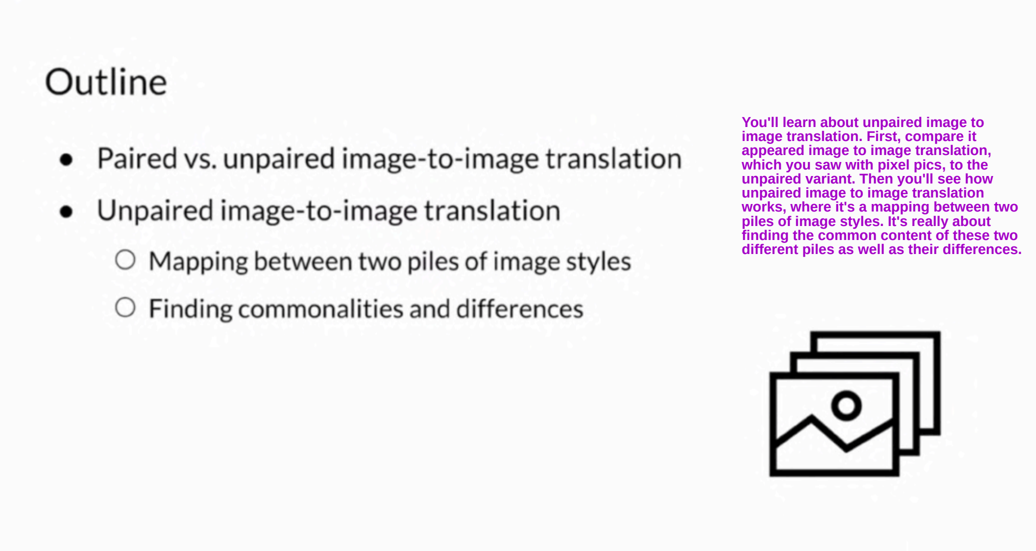
deeplearning.ai

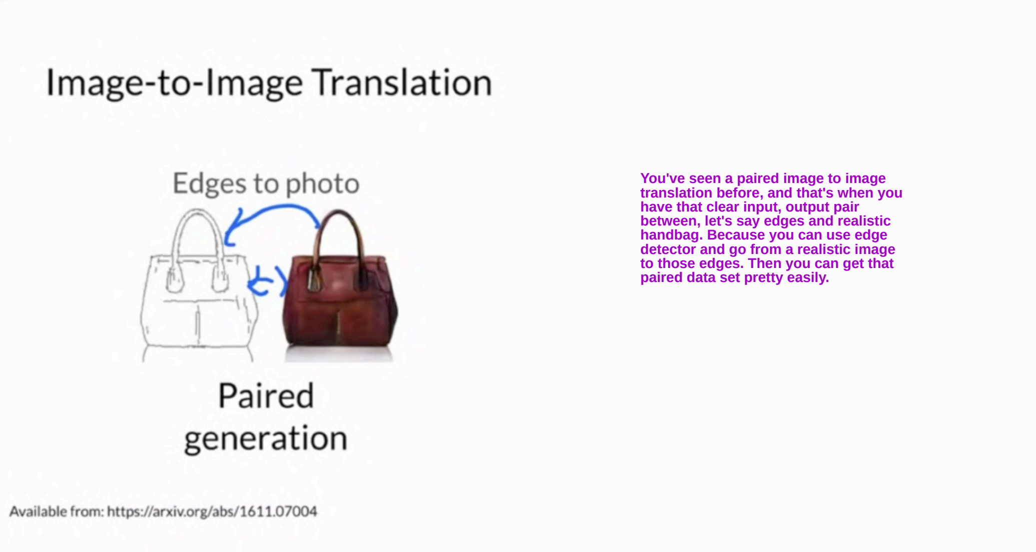
# Unpaired Image-to-Image Translation



## Outline

- Paired vs. unpaired image-to-image translation
- Unpaired image-to-image translation
  - ○ Mapping between two piles of image styles
  - ○ Finding commonalities and differences

You'll learn about unpaired image to image translation. First, compare it appeared image to image translation, which you saw with pixel pics, to the unpaired variant. Then you'll see how unpaired image to image translation works, where it's a mapping between two piles of image styles. It's really about finding the common content of these two different piles as well as their differences.



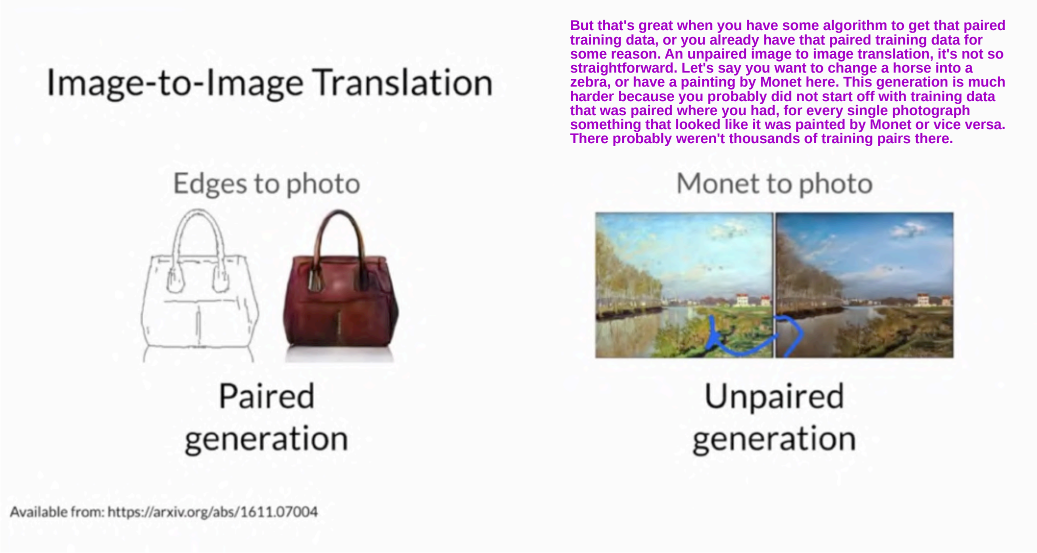
## Image-to-Image Translation

Edges to photo

Paired generation

Available from: https://arxiv.org/abs/1611.07004

You've seen a paired image to image translation before, and that's when you have that clear input, output pair between, let's say edges and realistic handbag. Because you can use edge detector and go from a realistic image to those edges. Then you can get that paired data set pretty easily.



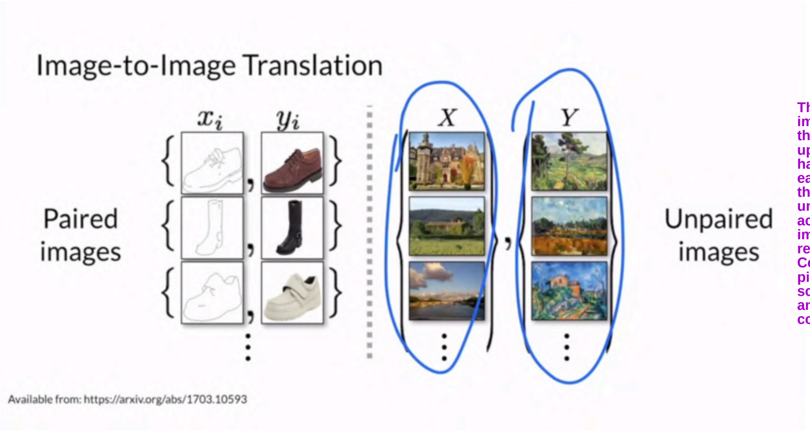
## Image-to-Image Translation

Paired generation

Unpaired generation

Available from: https://arxiv.org/abs/1611.07004

But that's great when you have some algorithm to get that paired training data, or you already have that paired training data for some reason. An unpaired image to image translation, it's not so straightforward. Let's say you want to change a horse into a zebra, or have a painting by Monet here. This generation is much harder because you probably did not start off with training data that was paired where you had, for every single photograph something that looked like it was painted by Monet or vice versa. There probably weren't thousands of training pairs there.



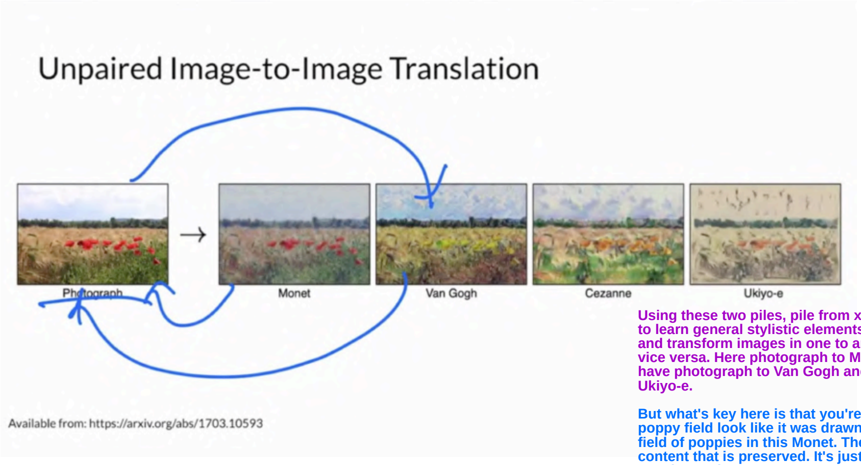
## Image-to-Image Translation

Paired images

Unpaired images

Available from: https://arxiv.org/abs/1703.10593

The paradigm between these two image to image translation tasks, is that for one of them, you have these paired images. You pair up xi and yi so here i is probably 0, 1, 2. You have all these different pairs that match onto each other, but you don't necessarily have this correspondence all the time, and so an unpaired image to image translation, you actually just have two piles of two different image styles, x and y. One pile which is maybe realistic photos, the other maybe it's Monet or Cezanne or someone else. Or you can have a pile of winter looking images versus summer scenes or a pile of horses in one or zebras in another, where you don't have a one-to-one correspondence at all.



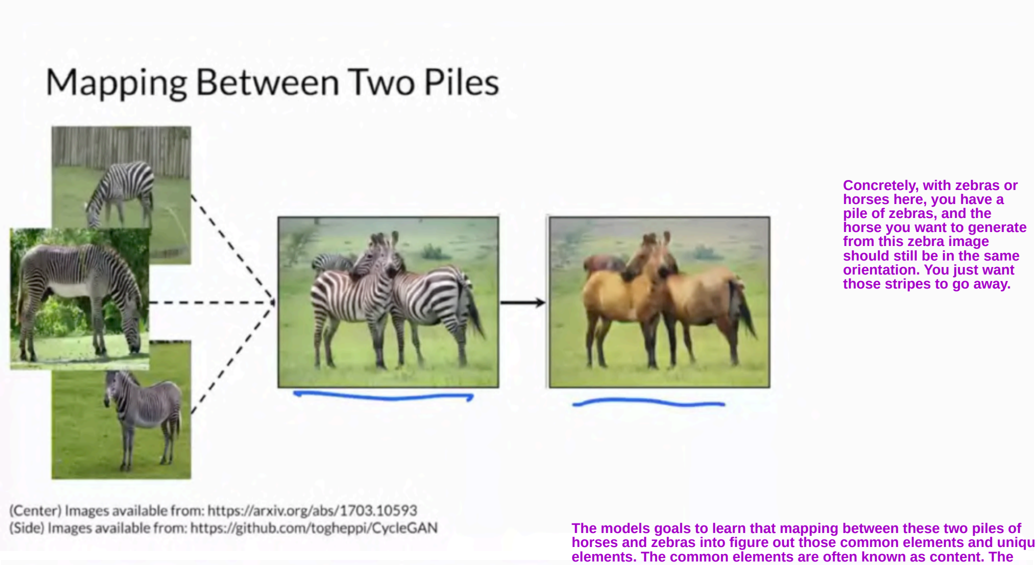
## Unpaired Image-to-Image Translation

Available from: https://arxiv.org/abs/1703.10593

Using these two piles, pile from x and y, you want your model to learn general stylistic elements from one pile to the other and transform images in one to another and sometimes also vice versa. Here photograph to Monet and maybe you can also have photograph to Van Gogh and vice versa to Cezanne and Ukiyo-e.

But what's key here is that you're making this photo of a poppy field look like it was drawn by say Monet, but it's still a field of poppies in this Monet. There's still some type of content that is preserved. It's just the stylistic elements that are changed.

That's pretty key in thinking about this translation task because there are commonalities and stylistic differences, unique things about each pile where you want to be able to tease out what's common, and keep those common elements, and only transfer those unique elements that are to each pile.



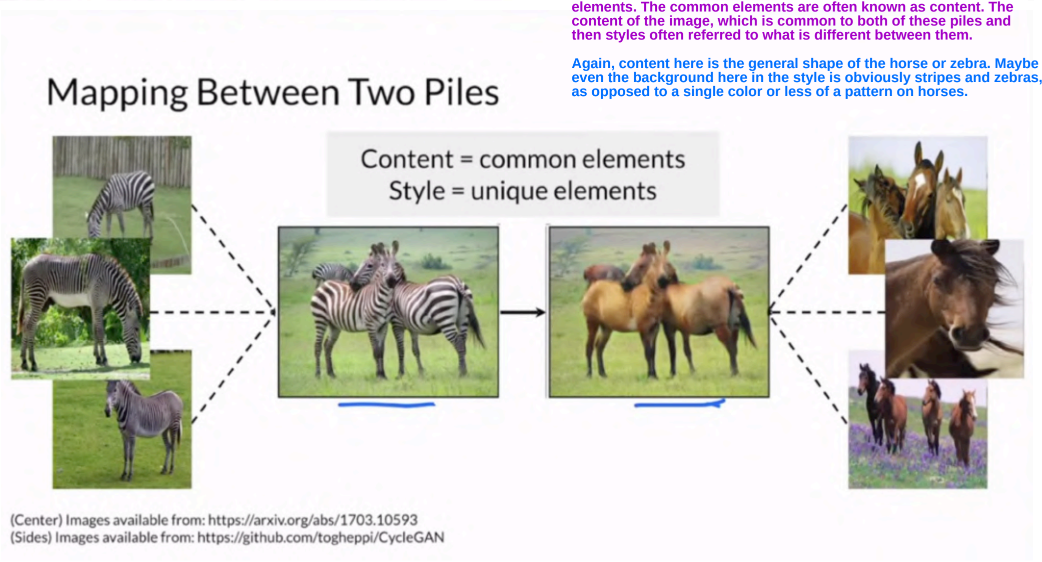
## Mapping Between Two Piles

(Center) Images available from: https://arxiv.org/abs/1703.10593
(Side) Images available from: https://github.com/togheppi/CycleGAN

Concretely, with zebras or horses here, you have a pile of zebras, and the horse you want to generate from this zebra image should still be in the same orientation. You just want those stripes to go away.

The models goals to learn that mapping between these two piles of horses and zebras into figure out those common elements and unique elements. The common elements are often known as content. The content of the image, which is common to both of these piles and then styles often referred to what is different between them.

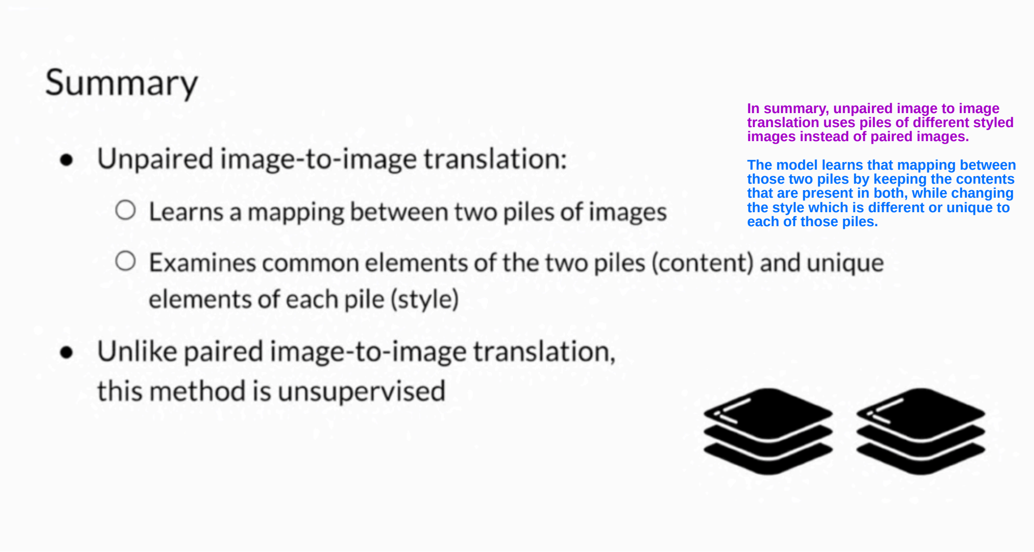
Again, content here is the general shape of the horse or zebra. Maybe even the background here in the style is obviously stripes and zebras, as opposed to a single color or less of a pattern on horses.



## Mapping Between Two Piles

Content = common elements
Style = unique elements

(Center) Images available from: https://arxiv.org/abs/1703.10593
(Sides) Images available from: https://github.com/togheppi/CycleGAN

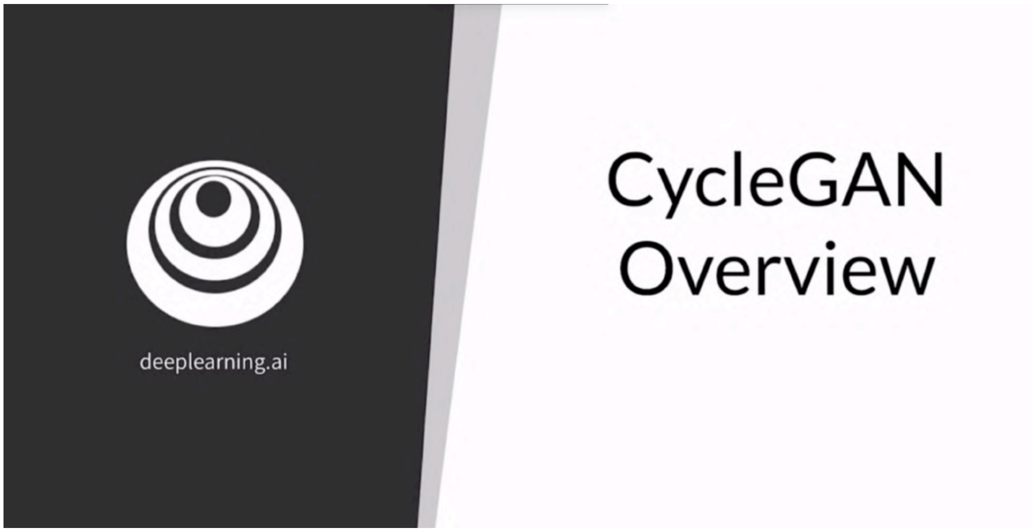


## Summary

- Unpaired image-to-image translation:
  - ○ Learns a mapping between two piles of images
  - ○ Examines common elements of the two piles (content) and unique elements of each pile (style)
- Unlike paired image-to-image translation, this method is unsupervised

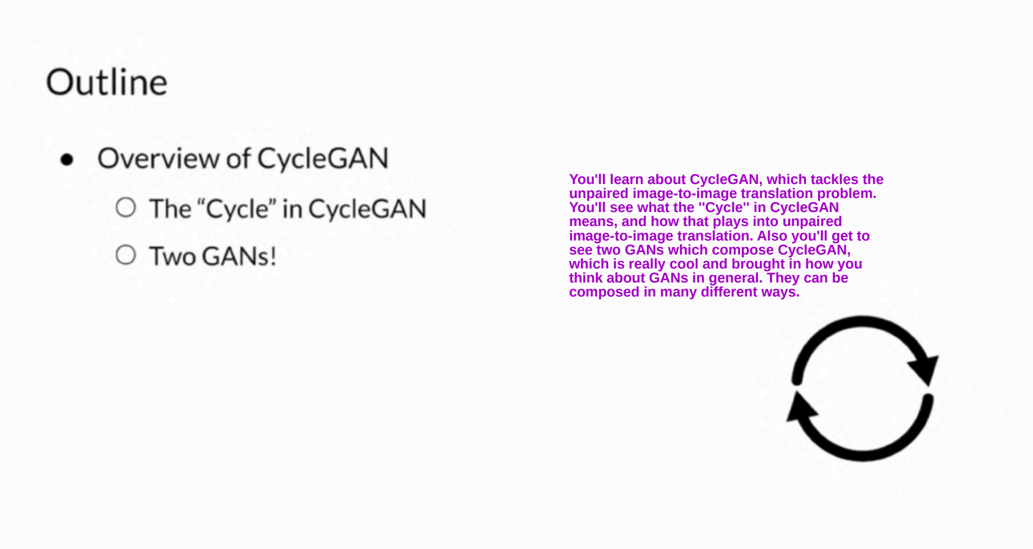
In summary, unpaired image to image translation uses piles of different styled images instead of paired images.

The model learns that mapping between those two piles by keeping the contents that are present in both, while changing the style which is different or unique to each of those piles.



deeplearning.ai

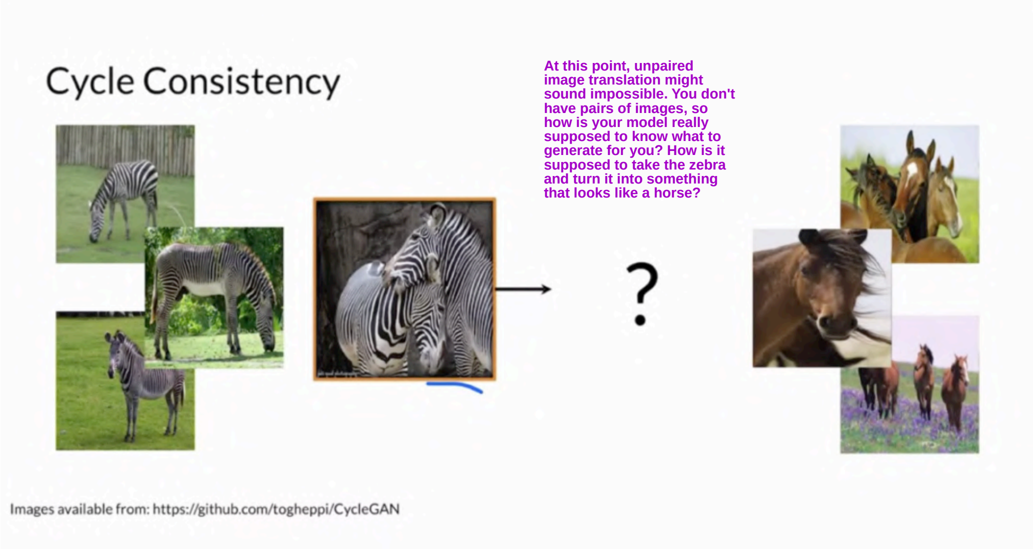
# CycleGAN Overview



## Outline

- Overview of CycleGAN
  - ○ The "Cycle" in CycleGAN
  - ○ Two GANs!

You'll learn about CycleGAN, which tackles the unpaired image-to-image translation problem. You'll see what the "Cycle" in CycleGAN means, and how that plays into unpaired image-to-image translation. Also you'll get to see two GANs which compose CycleGAN, which is really cool and brought in how you think about GANs in general. They can be composed in many different ways.



## Cycle Consistency

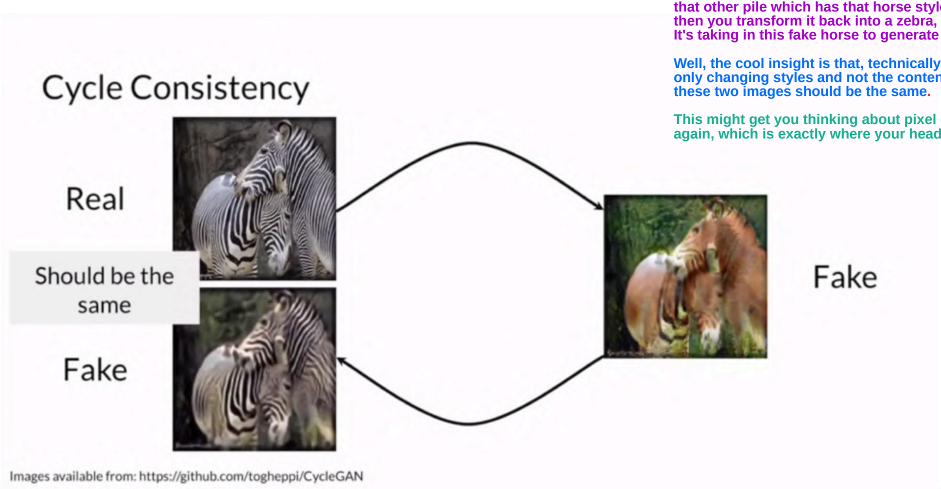
At this point, unpaired image translation might sound impossible. You don't have pairs of images, so how is your model really supposed to know what to generate for you? How is it supposed to take the zebra and turn it into something that looks like a horse?

Images available from: https://github.com/togheppi/CycleGAN

Well, there's this really cool insight that makes CycleGAN work. That's if you go form an image from one pile, say there's a real zebra, and you transform it into the style from that other pile which has that horse style going on, and then you transform it back into a zebra, and it's still fake. It's taking in this fake horse to generate another fake zebra.

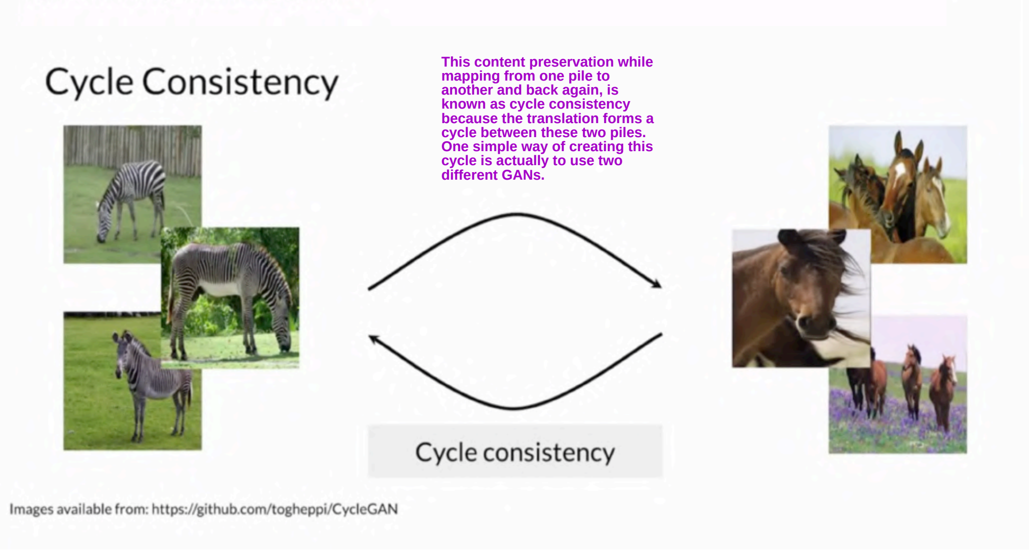
Well, the cool insight is that, technically because you're only changing styles and not the content of the image, these two images should be the same.

This might get you thinking about pixel distance loss, again, which is exactly where your head should be going.



## Cycle Consistency

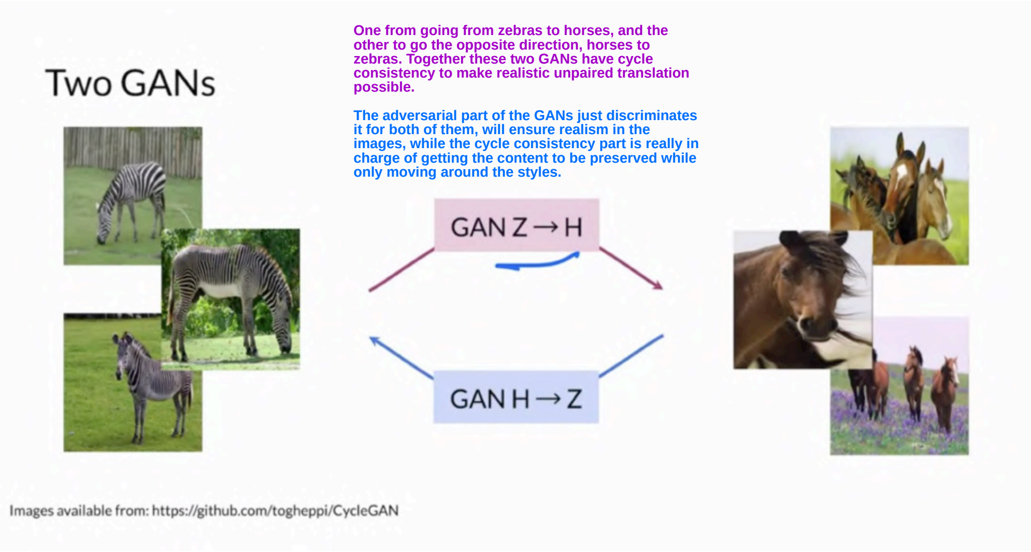
Images available from: https://github.com/togheppi/CycleGAN



## Cycle Consistency

This content preservation while mapping from one pile to another and back again, is known as cycle consistency because the translation forms a cycle between these two piles. One simple way of creating this cycle is actually to use two different GANs.

Images available from: https://github.com/togheppi/CycleGAN



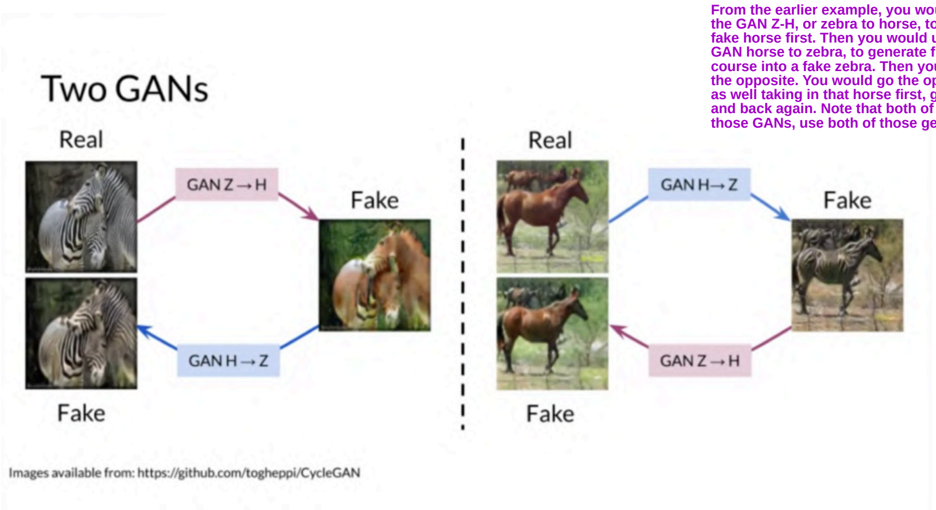
## Two GANs

One from going from zebras to horses, and the other to go the opposite direction, horses to zebras. Together these two GANs have cycle consistency to make realistic unpaired translation possible.

The adversarial part of the GANs just discriminates it for both of them, will ensure realism in the images, while the cycle consistency part is really in charge of getting the content to be preserved while only moving around the styles.

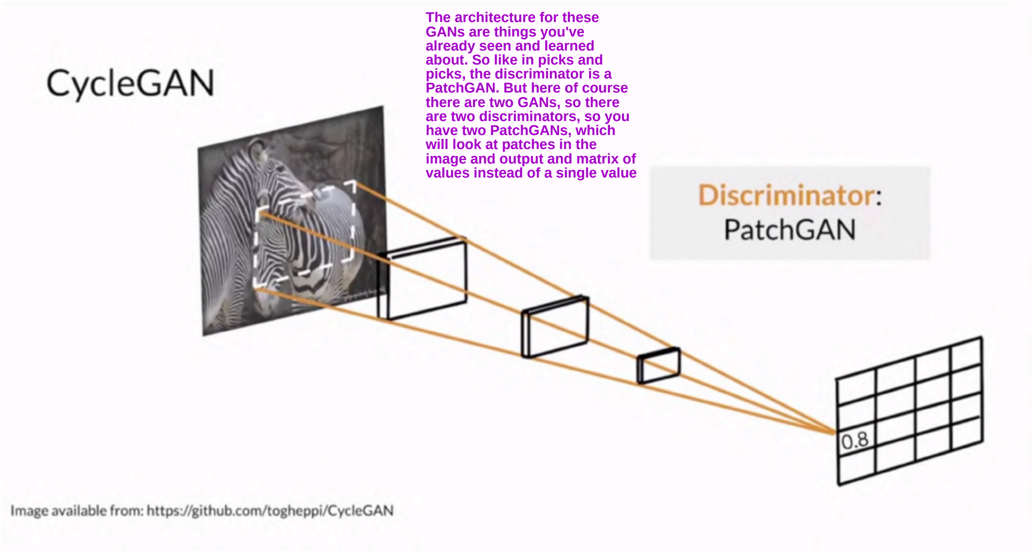
Images available from: https://github.com/togheppi/CycleGAN

From the earlier example, you would actually use the GAN Z-H, or zebra to horse, to generate that fake horse first. Then you would use your second GAN horse to zebra, to generate from that fake course into a fake zebra. Then you would also do the opposite. You would go the opposite direction as well taking in that horse first, going to zebra, and back again. Note that both of them use both of those GANs, use both of those generators.



## Two GANs

Images available from: https://github.com/togheppi/CycleGAN

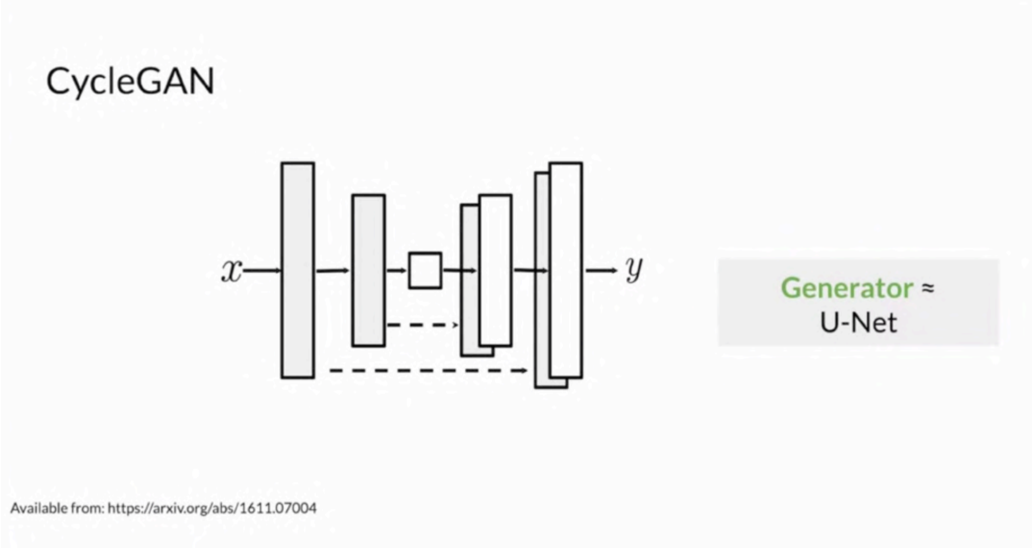


## CycleGAN

The architecture for these GANs are things you've already seen and learned about. So like in picks and picks, the discriminator is a PatchGAN. But here of course there are two GANs, so there are two discriminators, so you have two PatchGANs, which will look at patches in the image and output and matrix of values instead of a single value

**Discriminator:** PatchGAN

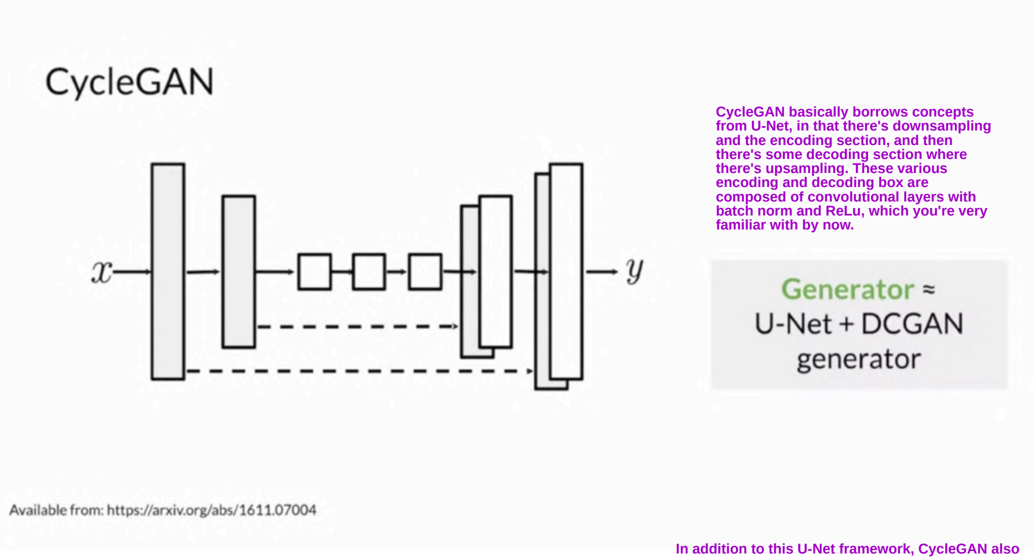
Image available from: https://github.com/togheppi/CycleGAN



## CycleGAN

**Generator** ≈ U-Net

Available from: https://arxiv.org/abs/1611.07004



## CycleGAN

Generator ≈ U-Net + DCGAN generator

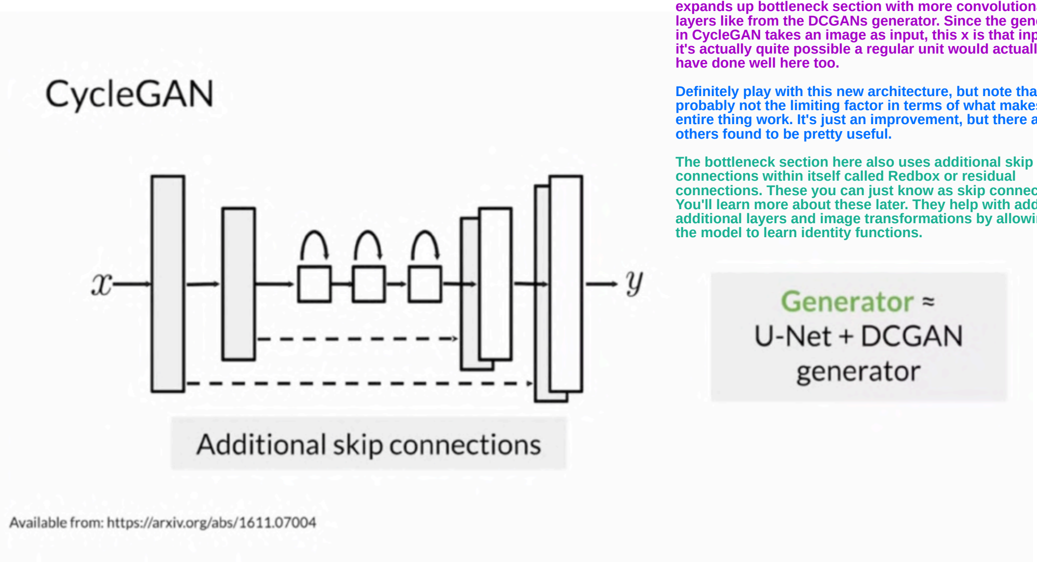
Available from: https://arxiv.org/abs/1611.07004

CycleGAN basically borrows concepts from U-Net, in that there's downsampling and the encoding section, and then there's some decoding section where there's upsampling. These various encoding and decoding box are composed of convolutional layers with batch norm and ReLu, which you're very familiar with by now.

In addition to this U-Net framework, CycleGAN also expands up bottleneck section with more convolutional layers like from the DCGANs generator. Since the generator in CycleGAN takes an image as input, this x is that input, it's actually quite possible a regular unit would actually have done well here too.

Definitely play with this new architecture, but note that it is probably not the limiting factor in terms of what makes this entire thing work. It's just an improvement, but there are others found to be pretty useful.

The bottleneck section here also uses additional skip connections within itself called Redbox or residual connections. These you can just know as skip connections. You'll learn more about these later. They help with adding additional layers and image transformations by allowing the model to learn identity functions.

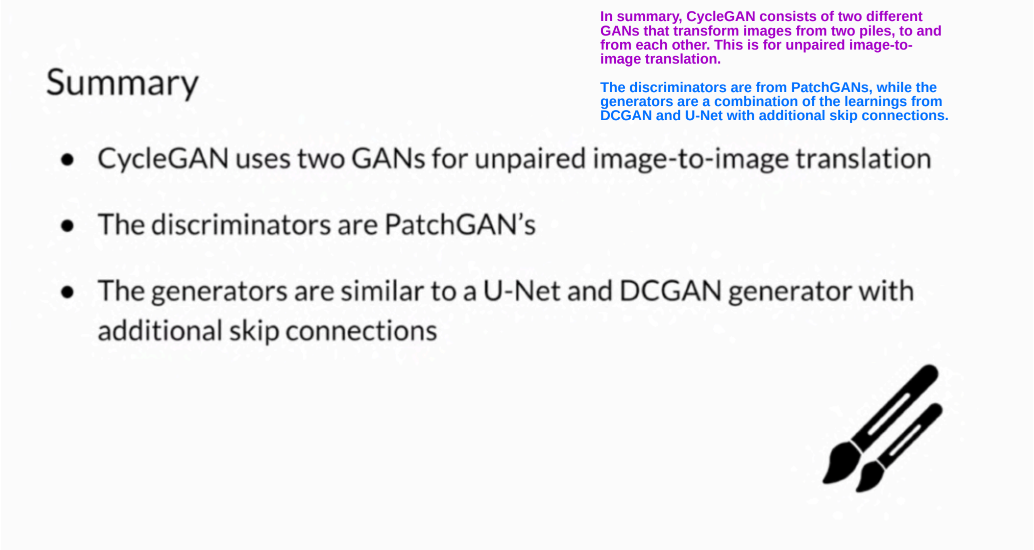


## CycleGAN

Additional skip connections

Generator ≈ U-Net + DCGAN generator

Available from: https://arxiv.org/abs/1611.07004



## Summary

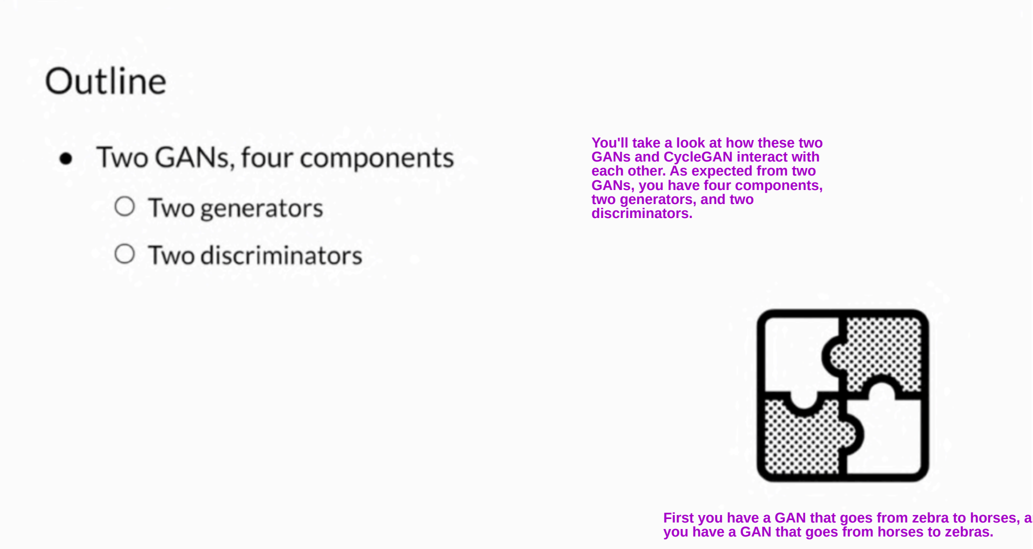
- CycleGAN uses two GANs for unpaired image-to-image translation
- The discriminators are PatchGAN's
- The generators are similar to a U-Net and DCGAN generator with additional skip connections

In summary, CycleGAN consists of two different GANs that transform images from two piles, to and from each other. This is for unpaired image-to-image translation.

The discriminators are from PatchGANs, while the generators are a combination of the learnings from DCGAN and U-Net with additional skip connections.

deeplearning.ai

# CycleGAN: Two GANs



## Outline

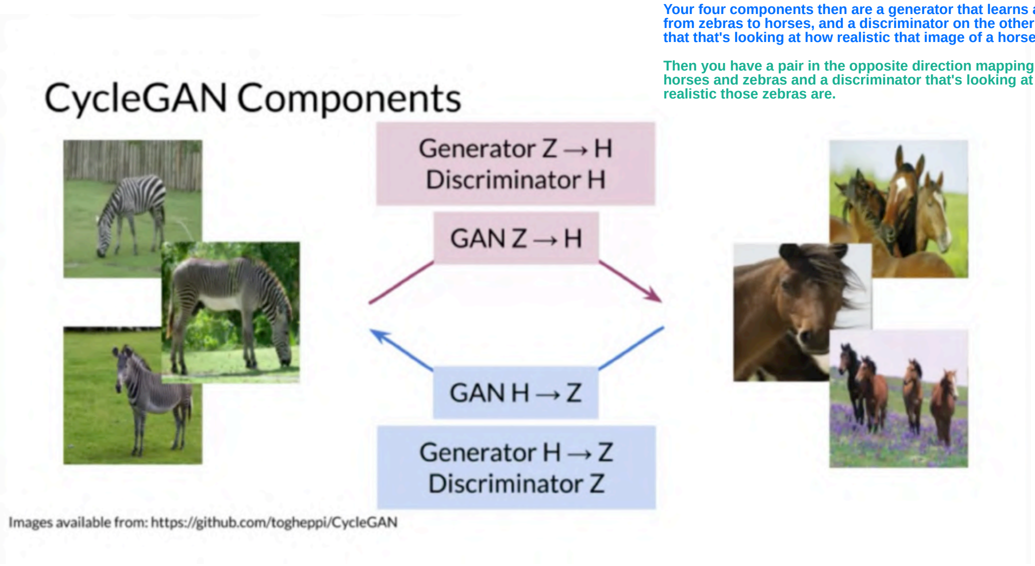
- Two GANs, four components
  - ○ Two generators
  - ○ Two discriminators

You'll take a look at how these two GANs and CycleGAN interact with each other. As expected from two GANs, you have four components, two generators, and two discriminators.

First you have a GAN that goes from zebra to horses, and then you have a GAN that goes from horses to zebras.

Your four components then are a generator that learns a mapping from zebras to horses, and a discriminator on the other end of that that's looking at how realistic that image of a horse is.

Then you have a pair in the opposite direction mapping from horses and zebras and a discriminator that's looking at how realistic those zebras are.



## CycleGAN Components

Generator Z → H
Discriminator H

GAN Z → H

GAN H → Z

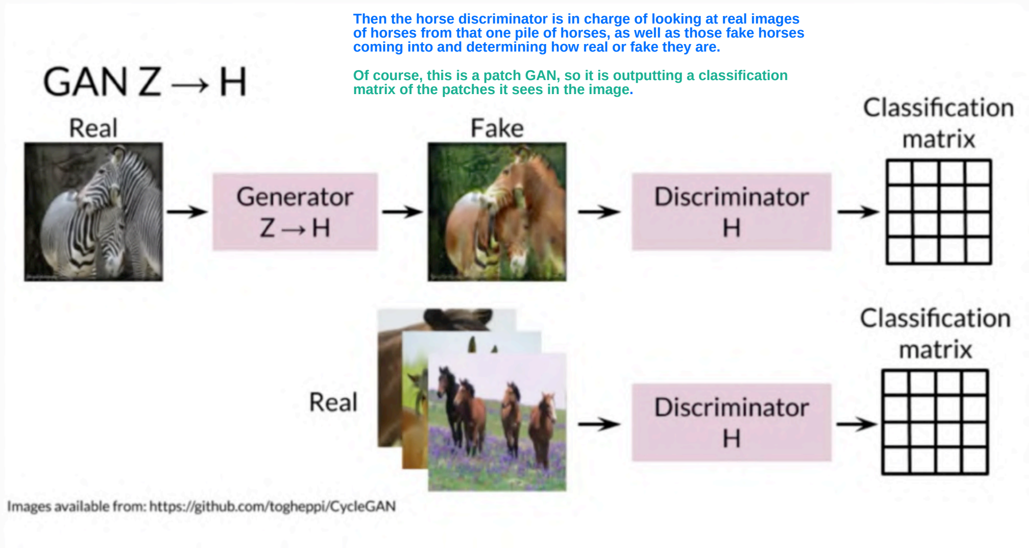
Generator H → Z
Discriminator Z

Images available from: https://github.com/togheppi/CycleGAN

Starting in one direction what this looks like is taking in a zebra image, mapping that into a horse using the zebra to horse generator.

Then the horse discriminator is in charge of looking at real images of horses from that one pile of horses, as well as those fake horses coming into and determining how real or fake they are.

Of course, this is a patch GAN, so it is outputting a classification matrix of the patches it sees in the image.



## GAN Z → H

Real

Generator Z → H

Fake

Discriminator H

Classification matrix

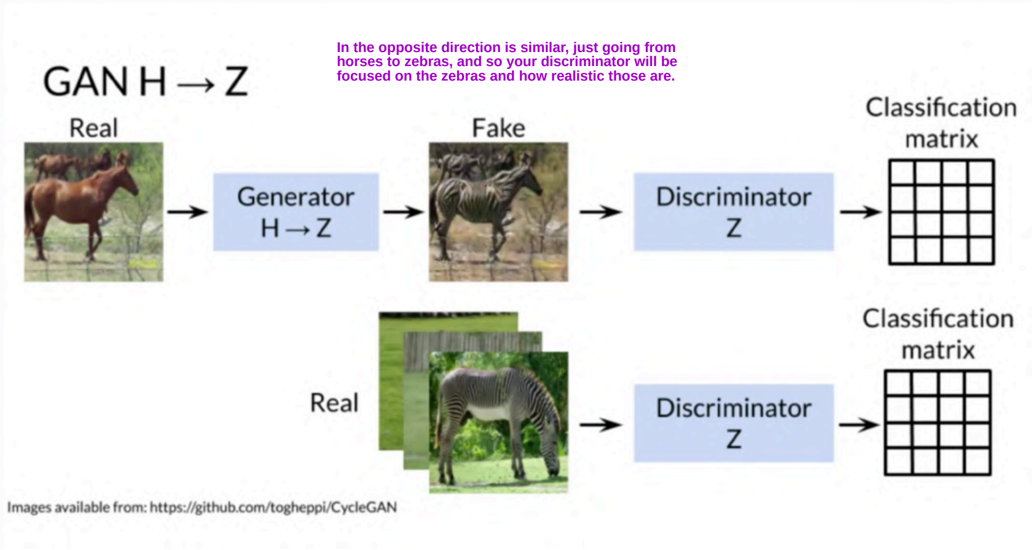
Real

Discriminator H

Classification matrix

Images available from: https://github.com/togheppi/CycleGAN

In the opposite direction is similar, just going from horses to zebras, and so your discriminator will be focused on the zebras and how realistic those are.



## GAN H → Z

Real

Generator H → Z

Fake

Discriminator Z

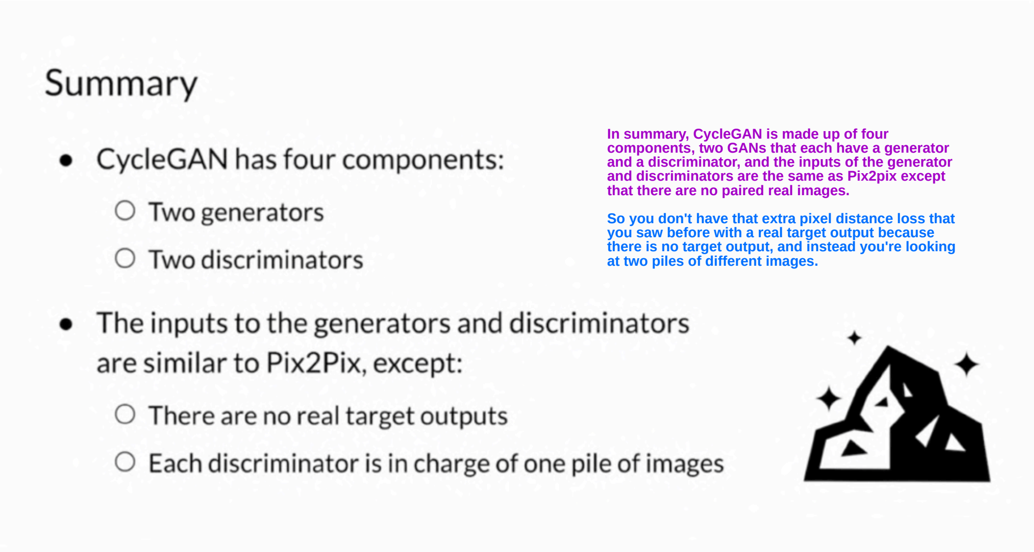
Classification matrix

Real

Discriminator Z

Classification matrix

Images available from: https://github.com/togheppi/CycleGAN



## Summary

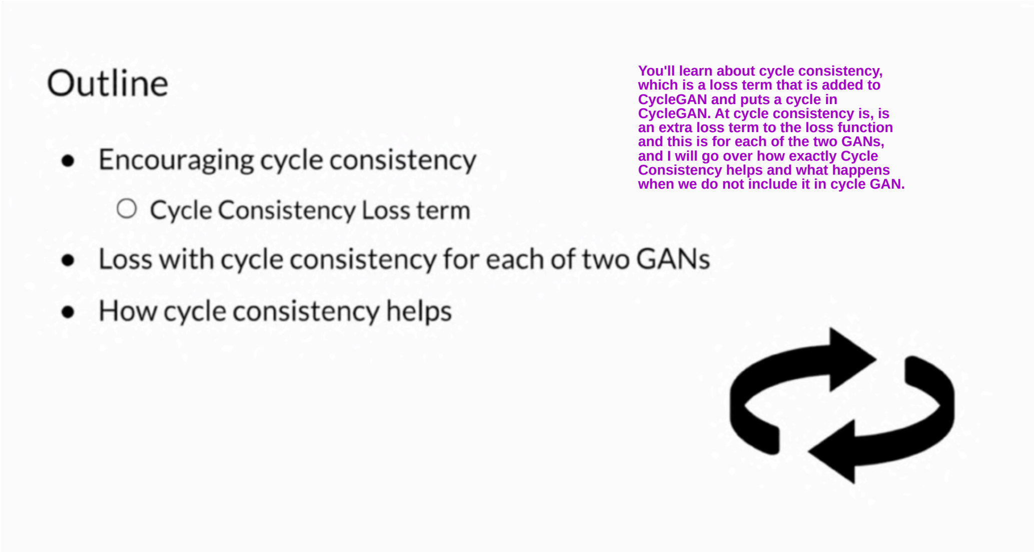
- CycleGAN has four components:
  - ○ Two generators
  - ○ Two discriminators
- The inputs to the generators and discriminators are similar to Pix2Pix, except:
  - ○ There are no real target outputs
  - ○ Each discriminator is in charge of one pile of images

In summary, CycleGAN is made up of four components, two GANs that each have a generator and a discriminator, and the inputs of the generator and discriminators are the same as Pix2pix except that there are no paired real images.

So you don't have that extra pixel distance loss that you saw before with a real target output because there is no target output, and instead you're looking at two piles of different images.

deeplearning.ai

# CycleGAN: Cycle Consistency



## Outline

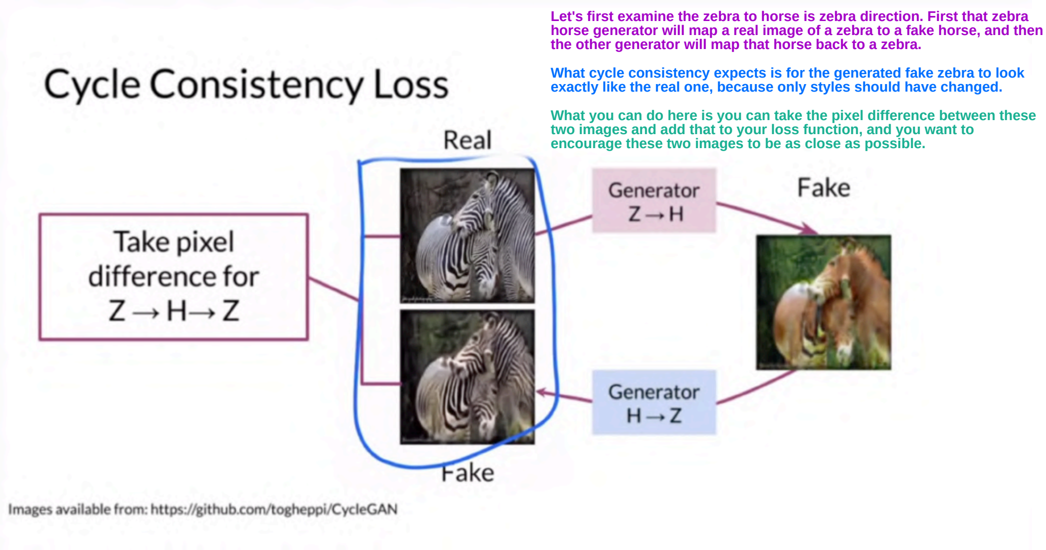
- Encouraging cycle consistency
  - ○ Cycle Consistency Loss term
- Loss with cycle consistency for each of two GANs
- How cycle consistency helps

You'll learn about cycle consistency, which is a loss term that is added to CycleGAN and puts a cycle in CycleGAN. At cycle consistency is, is an extra loss term to the loss function and this is for each of the two GANs, and I will go over how exactly Cycle Consistency helps and what happens when we do not include it in cycle GAN.

Let's first examine the zebra to horse is zebra direction. First that zebra horse generator will map a real image of a zebra to a fake horse, and then the other generator will map that horse back to a zebra.

What cycle consistency expects is for the generated fake zebra to look exactly like the real one, because only styles should have changed.

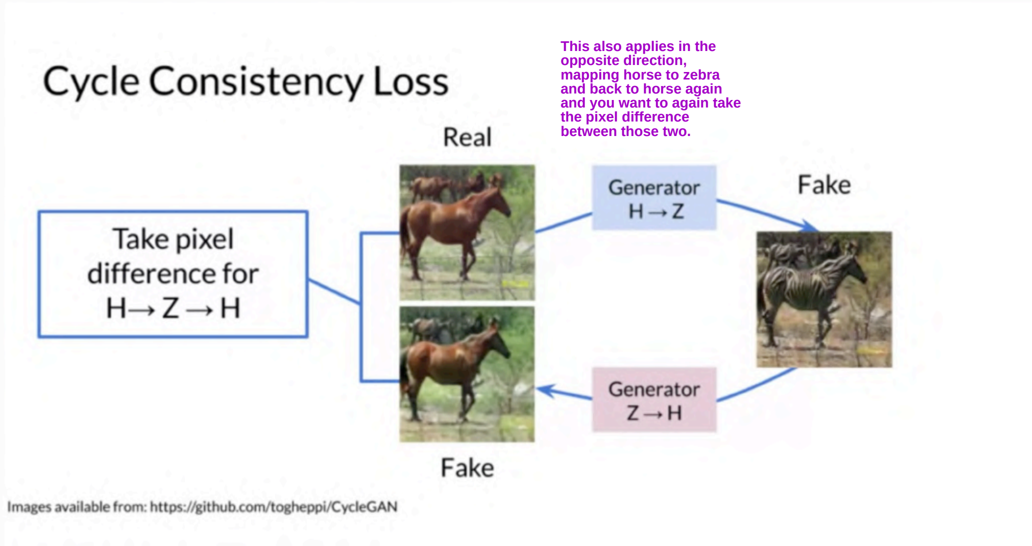
What you can do here is you can take the pixel difference between these two images and add that to your loss function, and you want to encourage these two images to be as close as possible.



Cycle Consistency Loss

Images available from: https://github.com/togheppi/CycleGAN

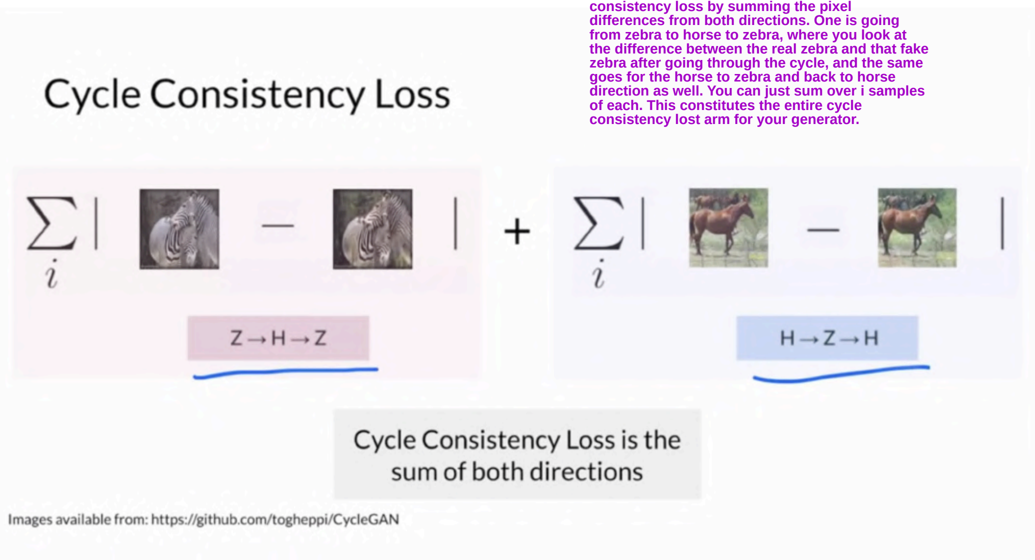
This also applies in the opposite direction, mapping horse to zebra and back to horse again and you want to again take the pixel difference between those two.



Cycle Consistency Loss

Images available from: https://github.com/togheppi/CycleGAN

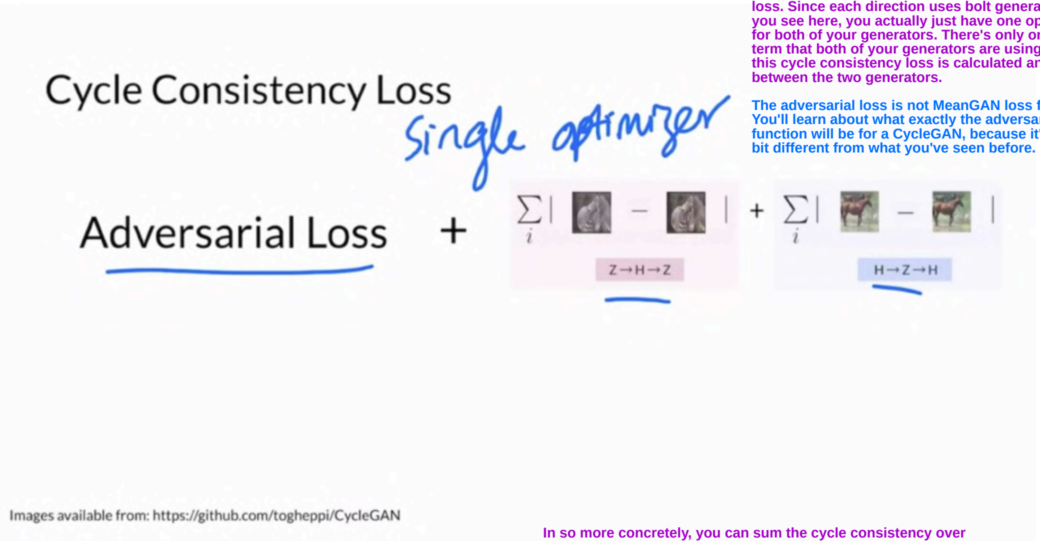
You can now construct the entire cycle consistency loss by summing the pixel differences from both directions. One is going from zebra to horse to zebra, where you look at the difference between the real zebra and that fake zebra after going through the cycle, and the same goes for the horse to zebra and back to horse direction as well. You can just sum over i samples of each. This constitutes the entire cycle consistency lost arm for your generator.



Cycle Consistency Loss

Cycle Consistency Loss is the sum of both directions

Images available from: https://github.com/togheppi/CycleGAN



## Cycle Consistency Loss

single optimizer

Adversarial Loss + $\sum_{i} | \quad - \quad | + \sum_{i} | \quad - \quad |$

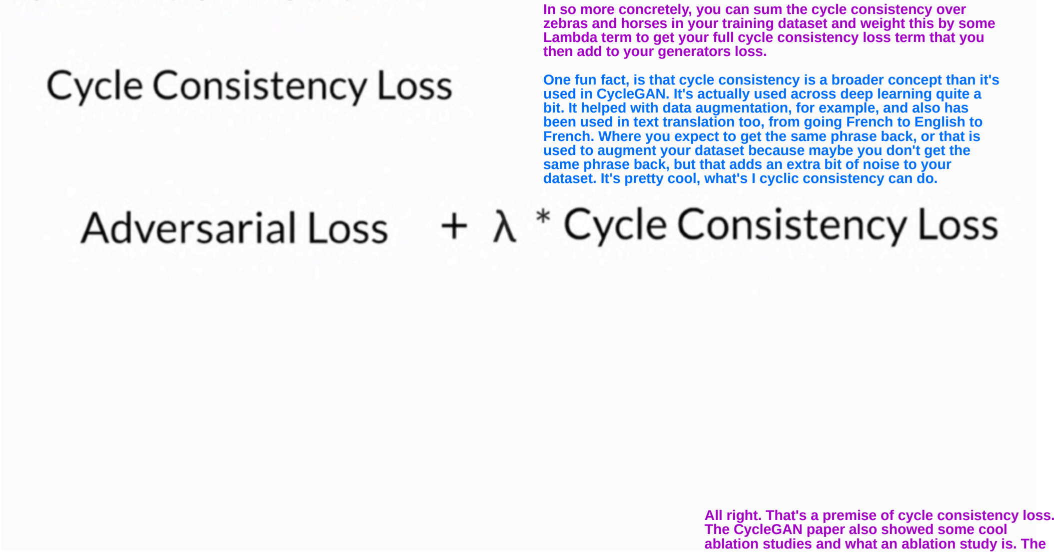
Z→H→Z

H→Z→H

Images available from: https://github.com/togheppi/CycleGAN

How you add this as you add this to your adversarial loss. Since each direction uses bolt generators that you see here, you actually just have one optimizer for both of your generators. There's only one loss term that both of your generators are using. From this cycle consistency loss is calculated and shared between the two generators.

The adversarial loss is not MeanGAN loss function. You'll learn about what exactly the adversarial loss function will be for a CycleGAN, because it's a little bit different from what you've seen before.

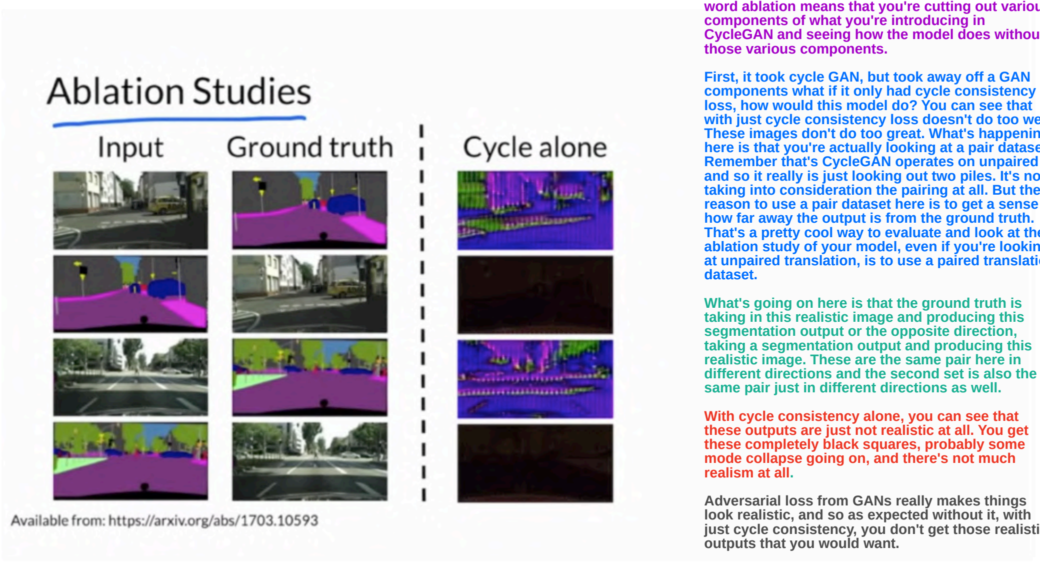


## Cycle Consistency Loss

Adversarial Loss + λ * Cycle Consistency Loss

In so more concretely, you can sum the cycle consistency over zebras and horses in your training dataset and weight this by some Lambda term to get your full cycle consistency loss term that you then add to your generators loss.

One fun fact, is that cycle consistency is a broader concept than it's used in CycleGAN. It's actually used across deep learning quite a bit. It helped with data augmentation, for example, and also has been used in text translation too, from going French to English to French. Where you expect to get the same phrase back, or that is used to augment your dataset because maybe you don't get the same phrase back, but that adds an extra bit of noise to your dataset. It's pretty cool, what's I cyclic consistency can do.



## Ablation Studies

Input | Ground truth | Cycle alone

Available from: https://arxiv.org/abs/1703.10593

All right. That's a premise of cycle consistency loss. The CycleGAN paper also showed some cool ablation studies and what an ablation study is. The word ablation means that you're cutting out various components of what you're introducing in CycleGAN and seeing how the model does without those various components.

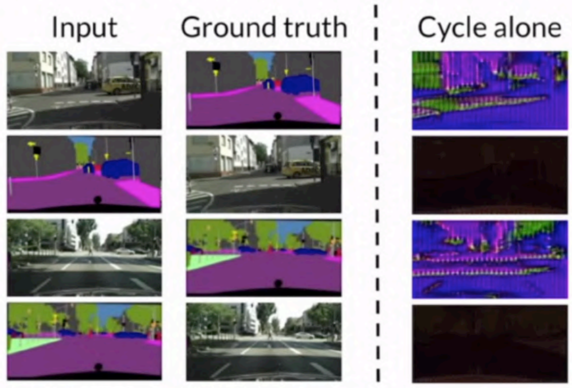
First, it took cycle GAN, but took away off a GAN components what if it only had cycle consistency loss, how would this model do? You can see that with just cycle consistency loss doesn't do too well. These images don't do too great. What's happening here is that you're actually looking at a pair dataset. Remember that's CycleGAN operates on unpaired and so it really is just looking out two piles. It's not taking into consideration the pairing at all. But the reason to use a pair dataset here is to get a sense of how far away the output is from the ground truth. That's a pretty cool way to evaluate and look at the ablation study of your model, even if you're looking at unpaired translation, is to use a paired translation dataset.

What's going on here is that the ground truth is taking in this realistic image and producing this segmentation output or the opposite direction, taking a segmentation output and producing this realistic image. These are the same pair here in different directions and the second set is also the same pair just in different directions as well.

With cycle consistency alone, you can see that these outputs are just not realistic at all. You get these completely black squares, probably some mode collapse going on, and there's not much realism at all.

Adversarial loss from GANs really makes things look realistic, and so as expected without it, with just cycle consistency, you don't get those realistic outputs that you would want.

## Ablation Studies

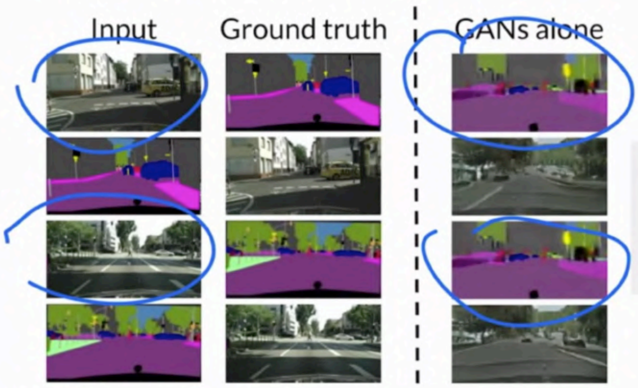


Adversarial loss from GANs really makes things look realistic, and so as expected without it, with just cycle consistency, you don't get those realistic outputs that you would

Without Adversarial GAN Loss, outputs are not realistic

Available from: https://arxiv.org/abs/1703.10593

## Ablation Studies



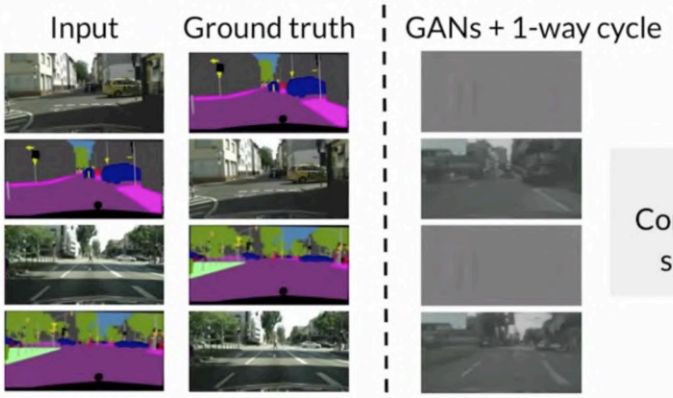
All right. But what about if you only have GANs and there's no cycle consistency? Well, the outputs actually look fairly realistic. They look pretty good, except for you might note that there is a little bit of mode collapse going on.

These two have very different input images, but the segmentation masks that are produced from them are very similar and there's a little bit of mode collapse going on, and that's obviously not ideal. These realistic images, while somewhat realistic much better than those black squares, still look a little bit mode collapsing they have all the same features, is not as diverse as the real images.

Without Cycle Consistency Loss, outputs show signs of mode collapse

Available from: https://arxiv.org/abs/1703.10593

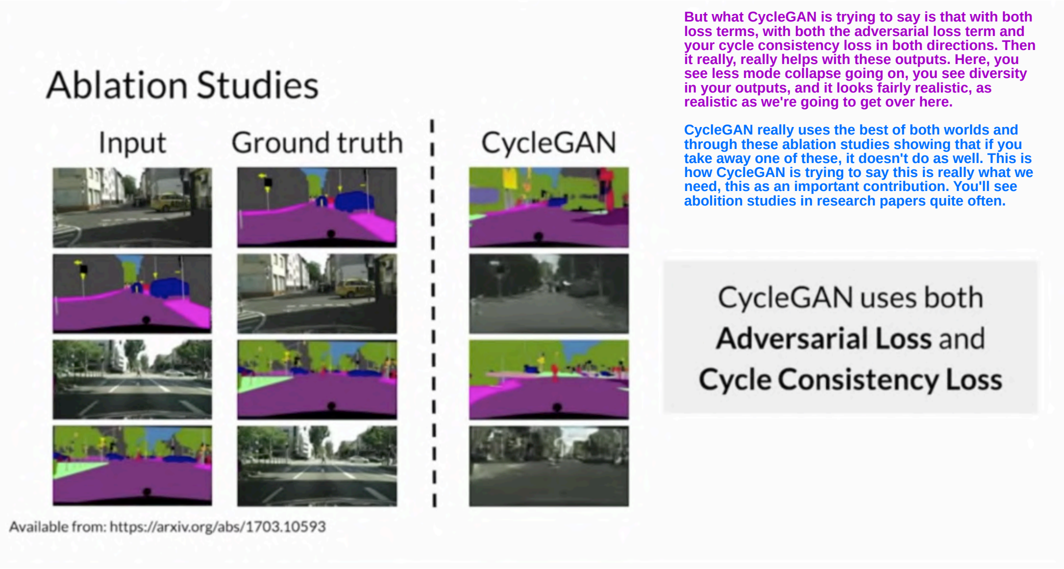
## Ablation Studies



Interestingly, you could use half of cycle consistency, but that also isn't enough for the GAN to learn diverse and quality mappings. You can see that without cycle consistency loss, the fall loss in both horses to zebras the horses and zebras the horses the zebras, and then the output still see some motor collapse to it's not great.

Without **full** Cycle Consistency Loss, outputs see mode collapse too

Available from: https://arxiv.org/abs/1703.10593



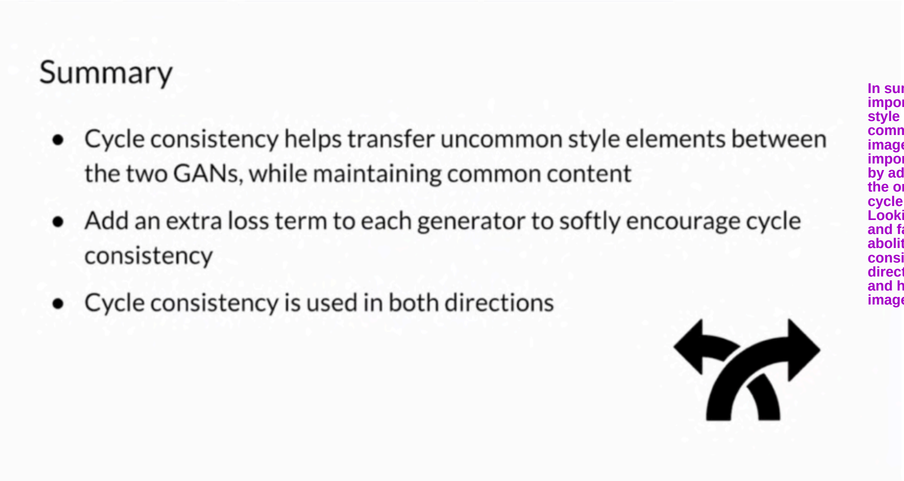
## Ablation Studies

CycleGAN uses both **Adversarial Loss** and **Cycle Consistency Loss**

Available from: https://arxiv.org/abs/1703.10593

But what CycleGAN is trying to say is that with both loss terms, with both the adversarial loss term and your cycle consistency loss in both directions. Then it really, really helps with these outputs. Here, you see less mode collapse going on, you see diversity in your outputs, and it looks fairly realistic, as realistic as we're going to get over here.

CycleGAN really uses the best of both worlds and through these ablation studies showing that if you take away one of these, it doesn't do as well. This is how CycleGAN is trying to say this is really what we need, this as an important contribution. You'll see abolition studies in research papers quite often.

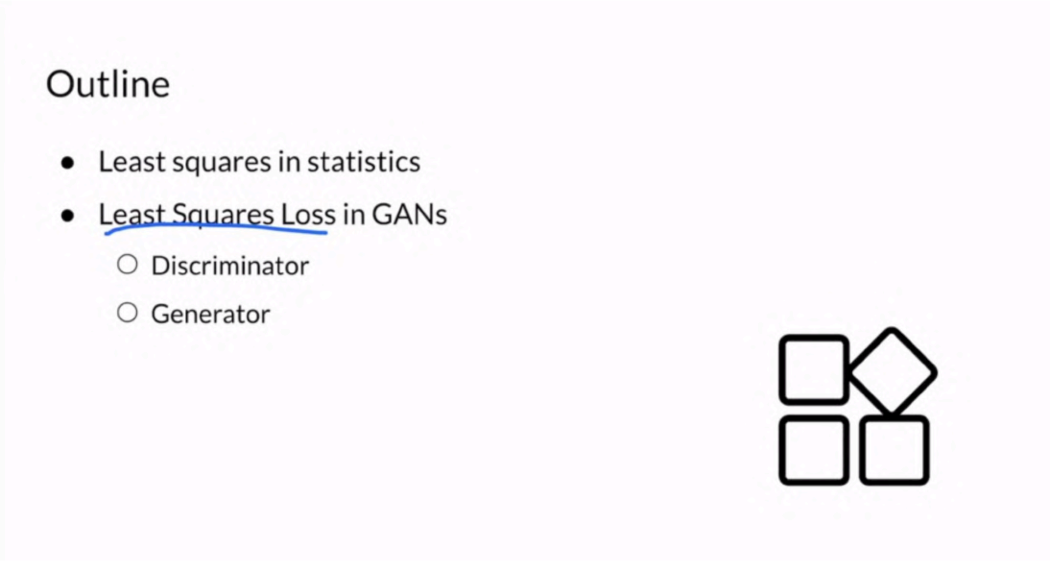


## Summary

- Cycle consistency helps transfer uncommon style elements between the two GANs, while maintaining common content
- Add an extra loss term to each generator to softly encourage cycle consistency
- Cycle consistency is used in both directions

In summary, cycle consistency is important in transferring on common style elements while maintaining common content across those images, and it is a really, really important loss term. This can be done by adding that pixel distance loss to the or adversarial loss to encourage cycle consistency in both directions. Looking at fake zebra to real zebra and fake horse to real horse. The abolition studies show that the cycle consistency loss term in both directions help prevent mode collapse and help with this uwieldy, unpaired image to image translation task.

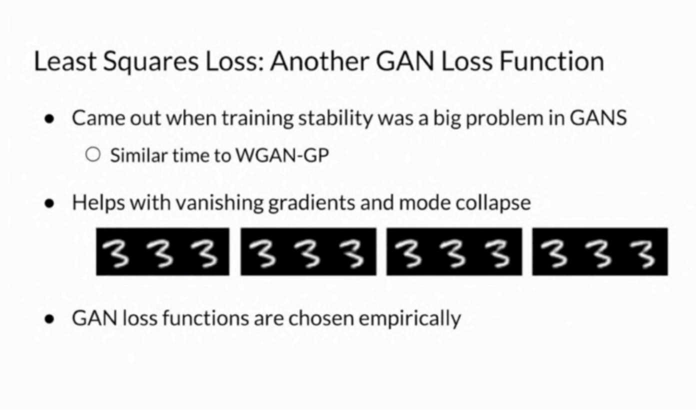
# CycleGAN: Least Squares Loss



## Outline

- Least squares in statistics
- Least Squares Loss in GANs
  - ○ Discriminator
  - ○ Generator

ıe



## Least Squares Loss: Another GAN Loss Function

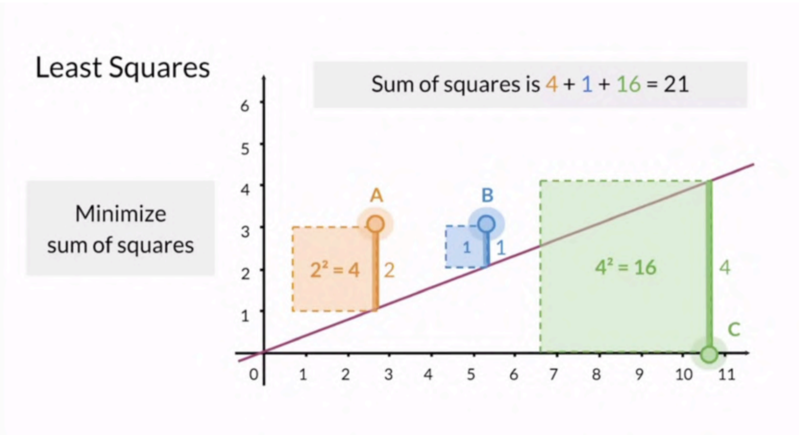
- Came out when training stability was a big problem in GANS
  - ○ Similar time to WGAN-GP
- Helps with vanishing gradients and mode collapse
- GAN loss functions are chosen empirically

Quick background on least squares loss and GANs. It came out at a similar point as WGAN-GP, which was of course on week three. And this was a time when training stability in GANs was a big problem.

Least Squares loss is used to help with training civility, namely vanishing gradient problems that you saw from BCE loss, but then cause mode collapse and other issues resulting in the end of learning, which is the worst thing you could possibly get.

Note that before we dive in, there are a ton of GAN Loss Functions, particularly to help with training stability, and often people experiment empirically with this since some datasets actually fare better with one loss-function versus another even on the same model. Remember that this could also introduce bias as people have different views on what is "better".

Something to think about is that training time is also a huge component to how people drink GANs. You might have seen that WGAN-GP was a little bit slower because the calculation is a bit more complex. You might have observed that in your assignment. That also sometimes pushes people to use a certain loss function over another as well.



## Least Squares

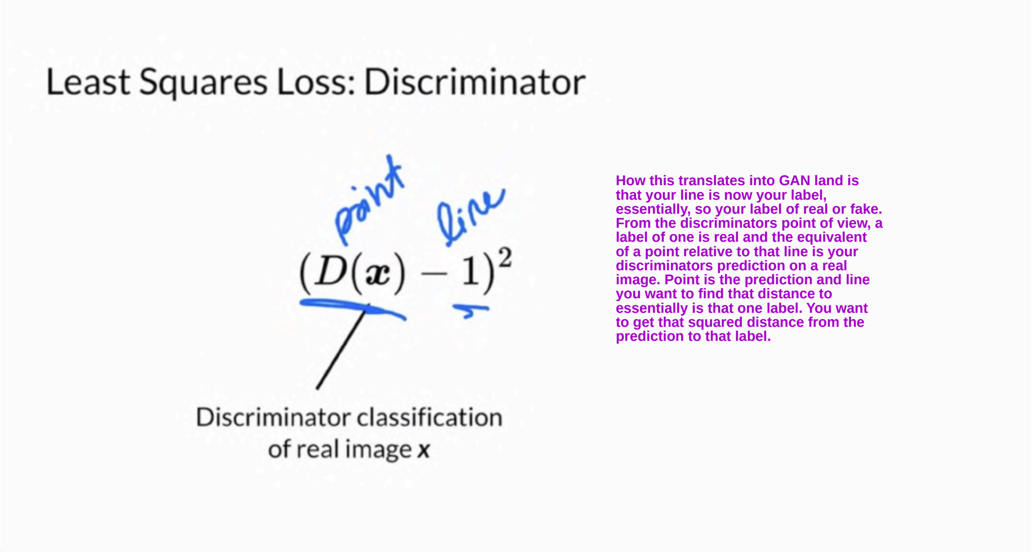
Minimize sum of squares

Sum of squares is 4 + 1 + 16 = 21

Least squares is a pretty simple concept of from statistics. It's a method that minimizes the sum of squared residuals. What that means is that it tries to find the best fit line that has the smallest sum of squared distances between that line and all the points.

Here's an example with three different points and you want to fit a line through them. Okay, cool. You can do that by finding each points distance from a line. You draw a line first, you find the distance from a point to that line. Specifically, you want to take a look at their squared distances and you want to minimize that squared distance, so you want to find the best fit line. When you draw a line like this, this is going to have a much larger square distance than this other line that you see here in purple.

You get the sum of squares as the total distance or cost for each line that you tried to do, you minimize that sum of squares to find that best fit line. That is what that means by minimizing the sum of the squared residuals, which is just that squared distance you see between the line and each point. By taking the sum of all of those squares and minimizing that value, you get the best fit line you can. This is a really simple concept to find the best fit line across a few points.

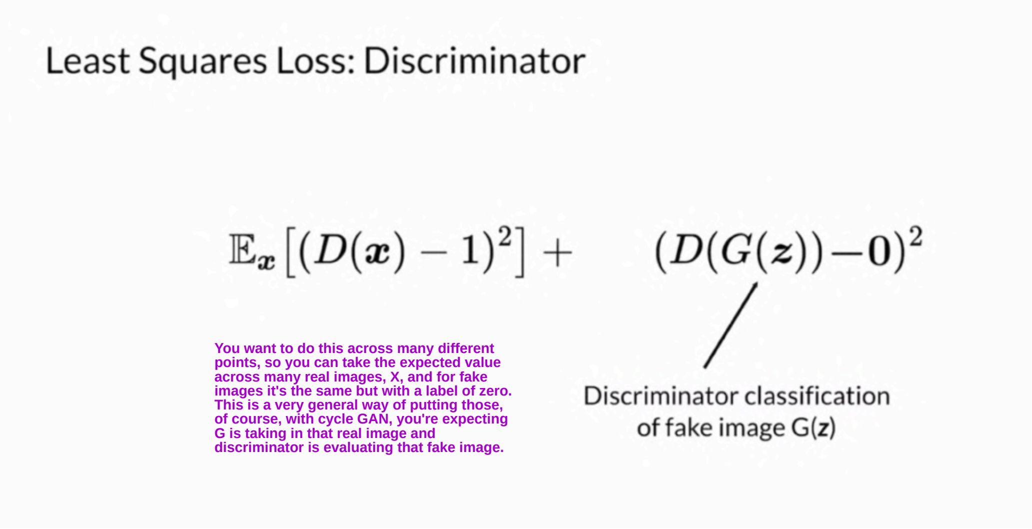


## Least Squares Loss: Discriminator

$$\big(\underline{\overset{\text{point}}{D(\boldsymbol{x})}} - \underline{\overset{\text{line}}{1}}\big)^2$$

Discriminator classification of real image ***x***

How this translates into GAN land is that your line is now your label, essentially, so your label of real or fake. From the discriminators point of view, a label of one is real and the equivalent of a point relative to that line is your discriminators prediction on a real image. Point is the prediction and line you want to find that distance to essentially is that one label. You want to get that squared distance from the prediction to that label.

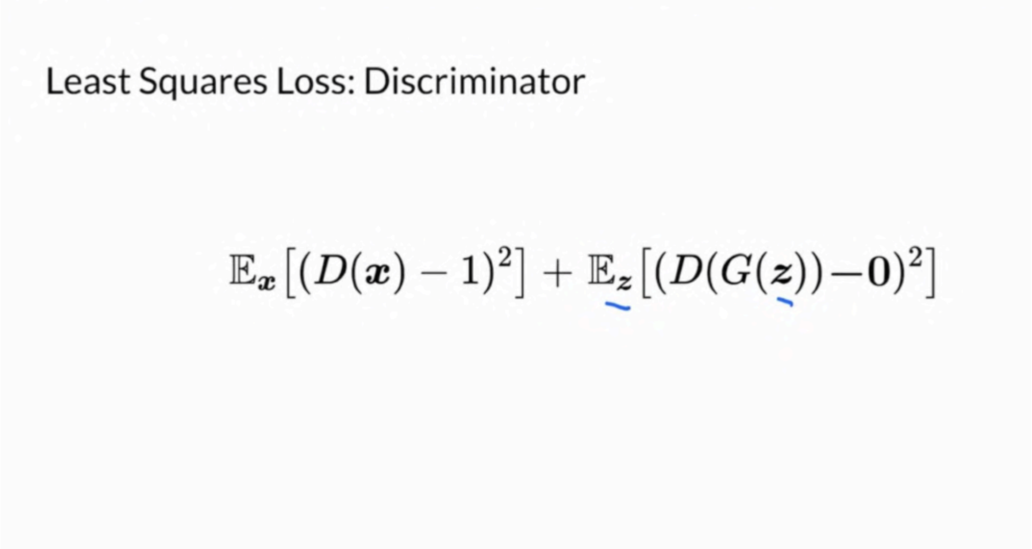


## Least Squares Loss: Discriminator

$$\mathbb{E}_{\boldsymbol{x}}\left[(D(\boldsymbol{x})-1)^2\right] + \quad (D(G(\boldsymbol{z}))-0)^2$$

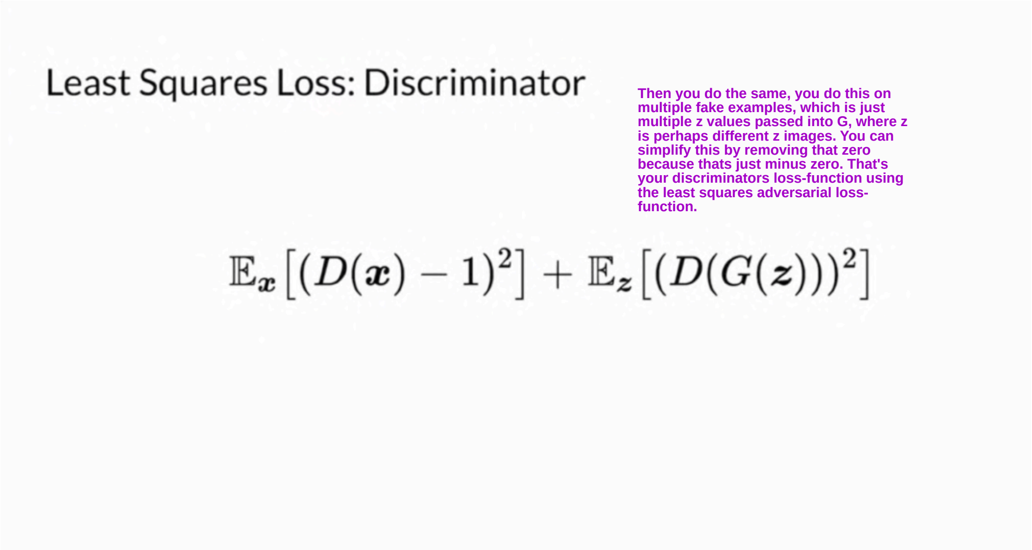
Discriminator classification of fake image G(***z***)

You want to do this across many different points, so you can take the expected value across many real images, X, and for fake images it's the same but with a label of zero. This is a very general way of putting those, of course, with cycle GAN, you're expecting G is taking in that real image and discriminator is evaluating that fake image.



## Least Squares Loss: Discriminator

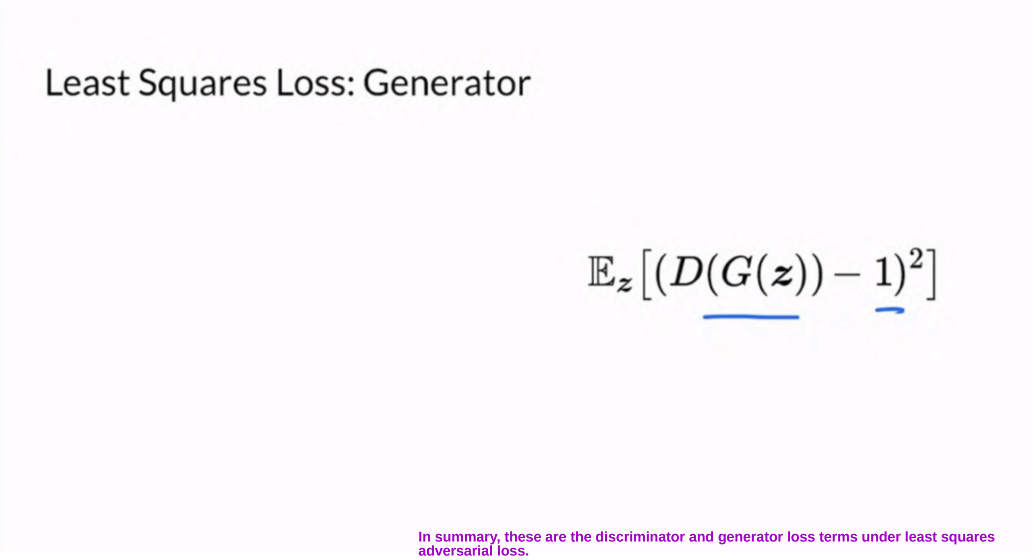
$$\mathbb{E}_{\boldsymbol{x}}\left[(D(\boldsymbol{x})-1)^2\right] + \mathbb{E}_{\underline{\boldsymbol{z}}}\left[(D(G(\underline{\boldsymbol{z}}))-0)^2\right]$$



## Least Squares Loss: Discriminator

Then you do the same, you do this on multiple fake examples, which is just multiple z values passed into G, where z is perhaps different z images. You can simplify this by removing that zero because thats just minus zero. That's your discriminators loss-function using the least squares adversarial loss-function.

$$\mathbb{E}_{\boldsymbol{x}}\left[(D(\boldsymbol{x})-1)^2\right]+\mathbb{E}_{\boldsymbol{z}}\left[(D(G(\boldsymbol{z})))^2\right]$$



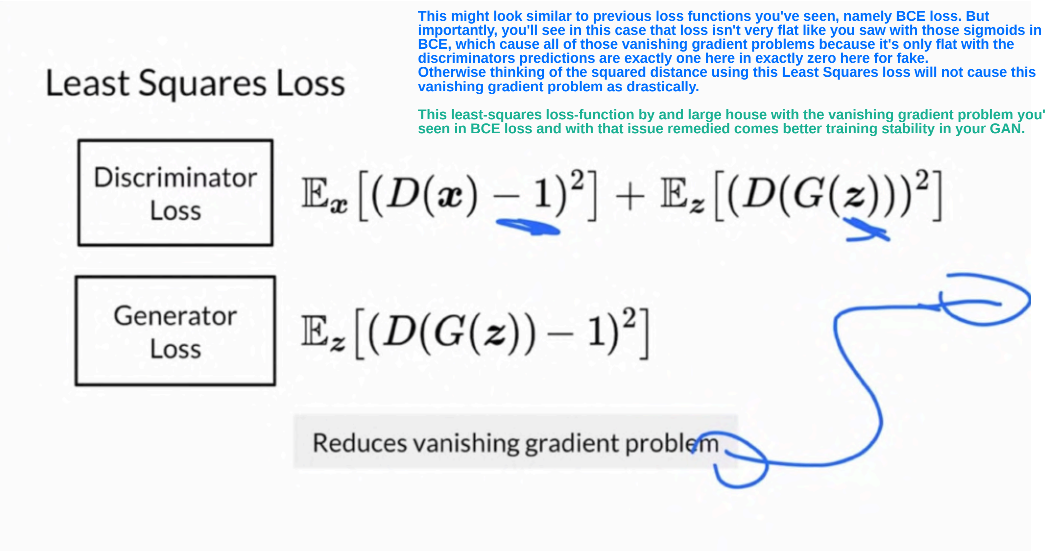
## Least Squares Loss: Generator

$$\mathbb{E}_{\boldsymbol{z}}\left[(D(G(\boldsymbol{z}))-1)^2\right]$$

In summary, these are the discriminator and generator loss terms under least squares adversarial loss.

This might look similar to previous loss functions you've seen, namely BCE loss. But importantly, you'll see in this case that loss isn't very flat like you saw with those sigmoids in BCE, which cause all of those vanishing gradient problems because it's only flat with the discriminators predictions are exactly one here in exactly zero here for fake.
Otherwise thinking of the squared distance using this Least Squares loss will not cause this vanishing gradient problem as drastically.

This least-squares loss-function by and large house with the vanishing gradient problem you've seen in BCE loss and with that issue remedied comes better training stability in your GAN.



## Least Squares Loss

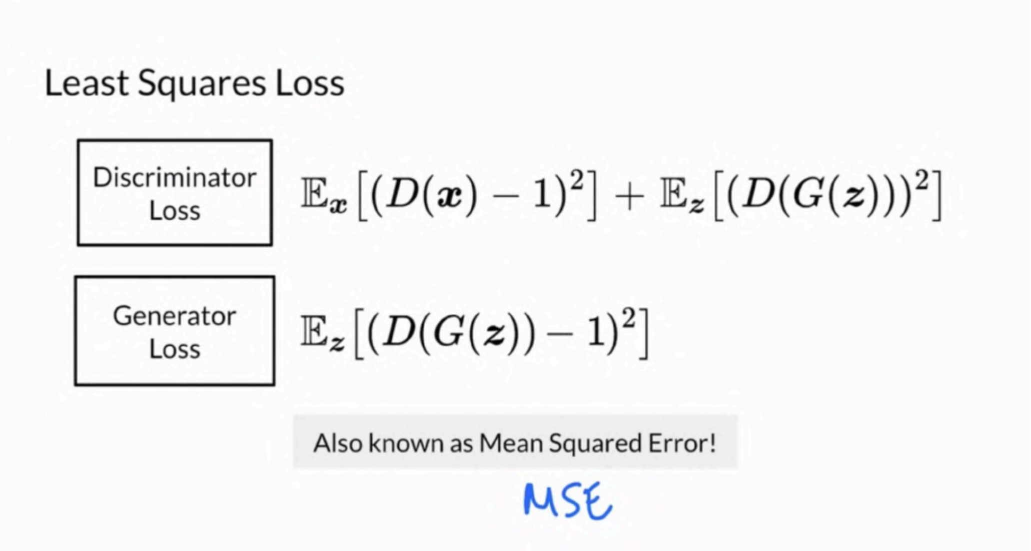
Discriminator Loss

$$\mathbb{E}_{\boldsymbol{x}}\left[(D(\boldsymbol{x})-1)^2\right]+\mathbb{E}_{\boldsymbol{z}}\left[(D(G(\boldsymbol{z})))^2\right]$$

Generator Loss

$$\mathbb{E}_{\boldsymbol{z}}\left[(D(G(\boldsymbol{z}))-1)^2\right]$$

Reduces vanishing gradient problem



## Least Squares Loss

Discriminator Loss

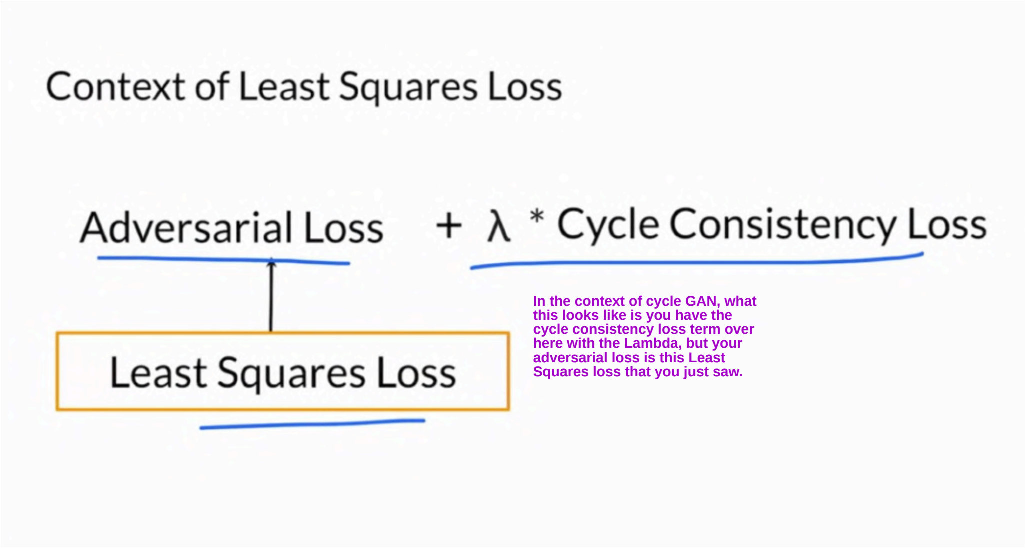
$$\mathbb{E}_{\boldsymbol{x}}\left[(D(\boldsymbol{x})-1)^2\right] + \mathbb{E}_{\boldsymbol{z}}\left[(D(G(\boldsymbol{z})))^2\right]$$

Generator Loss

$$\mathbb{E}_{\boldsymbol{z}}\left[(D(G(\boldsymbol{z}))-1)^2\right]$$

Also known as Mean Squared Error!

MSE

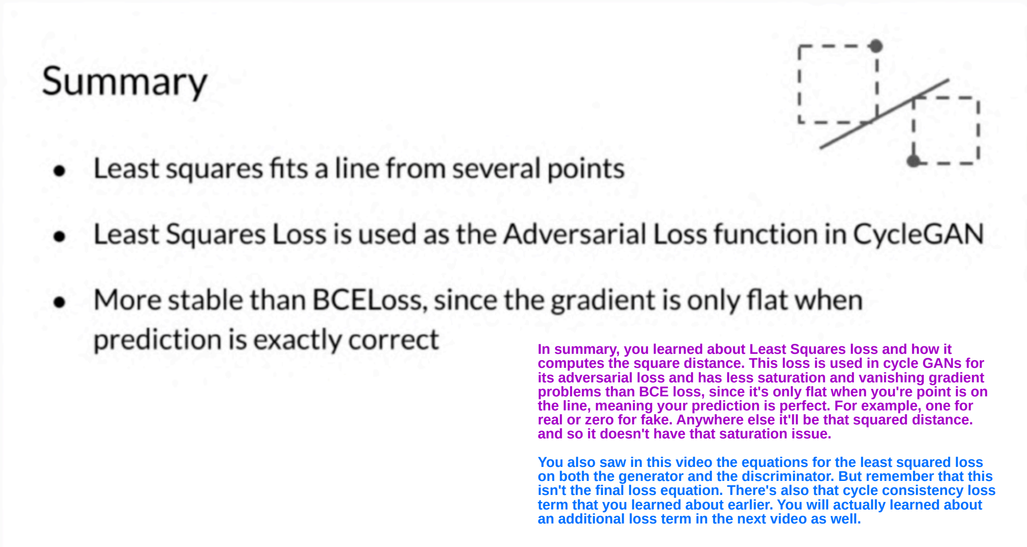


## Context of Least Squares Loss

Adversarial Loss + λ * Cycle Consistency Loss

Least Squares Loss

In the context of cycle GAN, what this looks like is you have the cycle consistency loss term over here with the Lambda, but your adversarial loss is this Least Squares loss that you just saw.

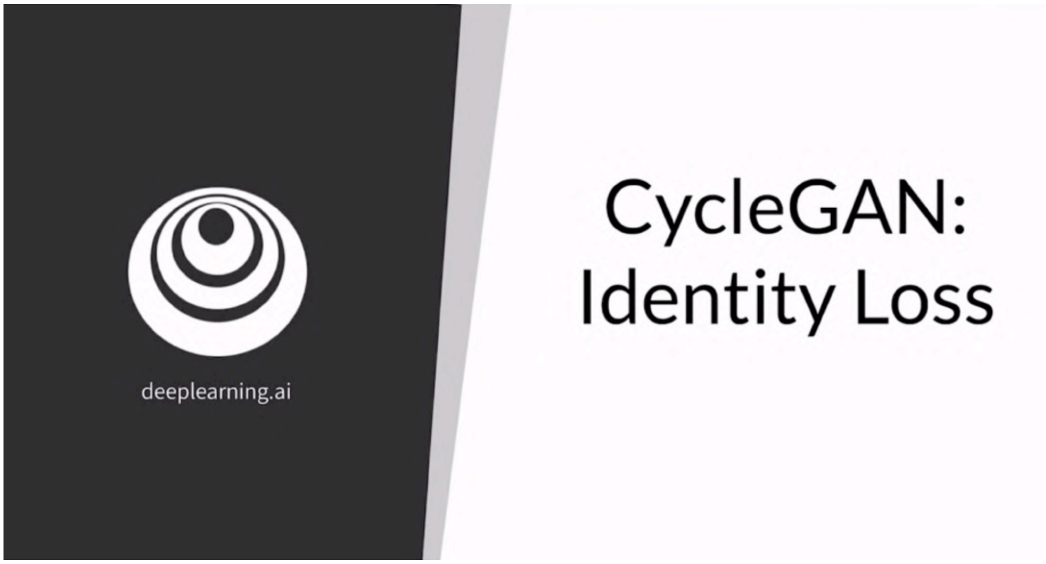


## Summary

- Least squares fits a line from several points
- Least Squares Loss is used as the Adversarial Loss function in CycleGAN
- More stable than BCELoss, since the gradient is only flat when prediction is exactly correct

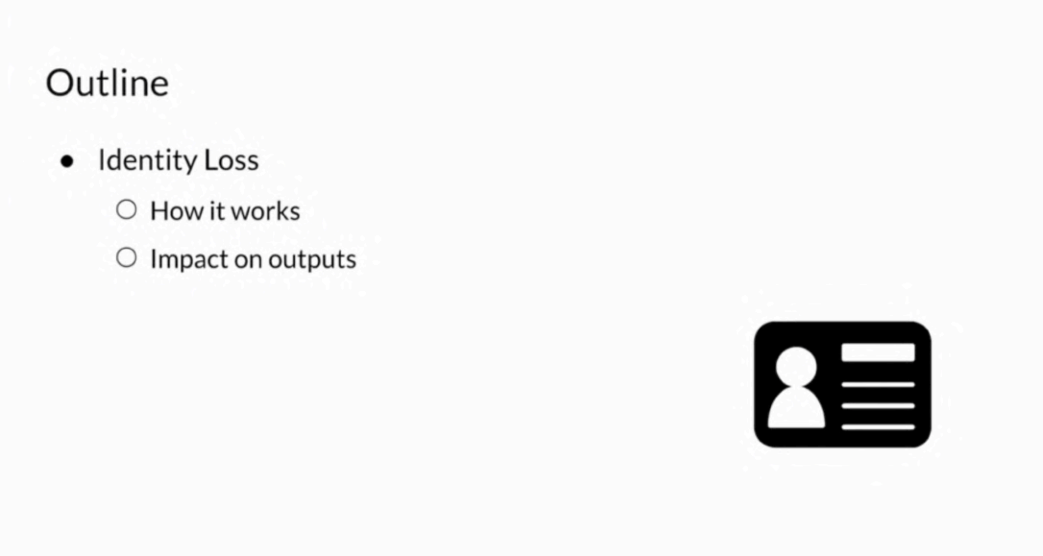
In summary, you learned about Least Squares loss and how it computes the square distance. This loss is used in cycle GANs for its adversarial loss and has less saturation and vanishing gradient problems than BCE loss, since it's only flat when you're point is on the line, meaning your prediction is perfect. For example, one for real or zero for fake. Anywhere else it'll be that squared distance. and so it doesn't have that saturation issue.

You also saw in this video the equations for the least squared loss on both the generator and the discriminator. But remember that this isn't the final loss equation. There's also that cycle consistency loss term that you learned about earlier. You will actually learned about an additional loss term in the next video as well.



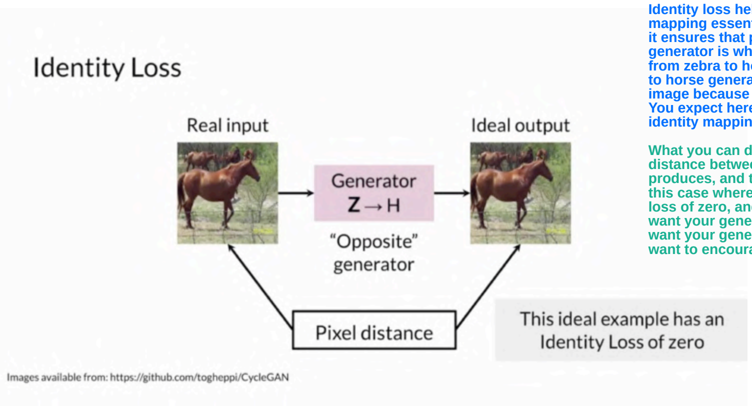
deeplearning.ai

# CycleGAN: Identity Loss



## Outline

- Identity Loss
  - ○ How it works
  - ○ Impact on outputs



## Identity Loss

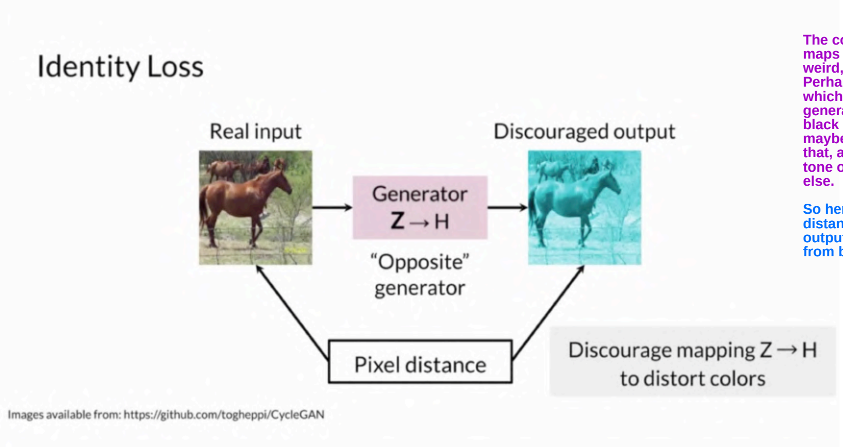
This ideal example has an Identity Loss of zero

Images available from: https://github.com/togheppi/CycleGAN

In addition to adversarial loss in the cycle consistency loss term, there's another pixel difference loss term that you could use. This one again is optional called identity loss.

Identity loss helps preserve color in images and have your mapping essentially make more sense. What that means is that it ensures that putting an image like this horse into this opposite generator is what I'm calling it, which is the generator that goes from zebra to horse. When you put this real input into this zebra to horse generator, it should ideally output the exact same image because it's already a horse style, it's not a zebra, right? You expect here is a zebra to horse generator will apply an identity mapping or essentially no change in input to the output.

What you can do for your loss term is that you can get the pixel distance between the real input, and whatever the generator produces, and the output, and add that to your loss function. In this case where the pixel distance is zero, the identity loss has a loss of zero, and so that's the idea. This is exactly what you want your generator to be doing. It's already a horse. You don't want your generator to transform it into any other thing, so you want to encourage this behavior.



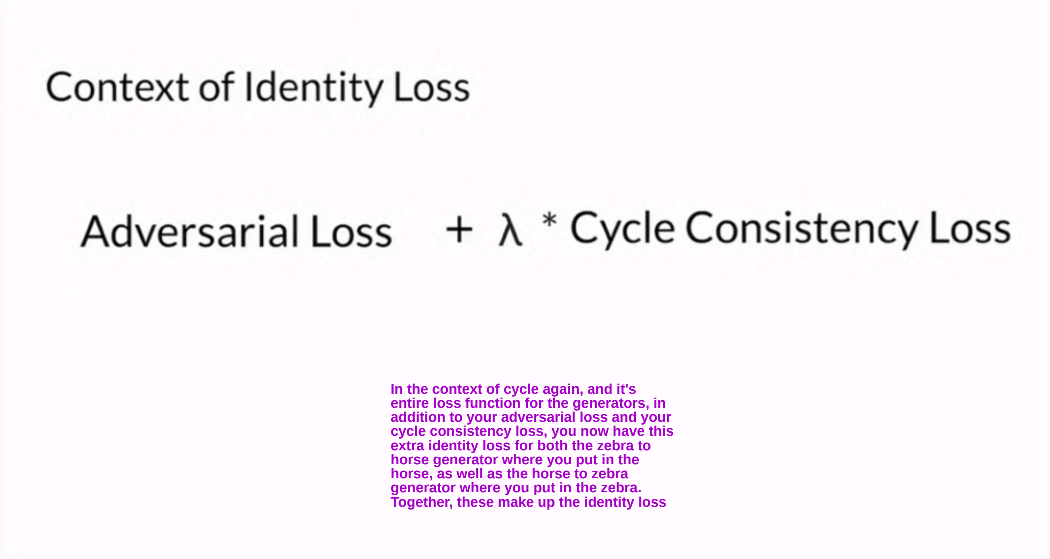
## Identity Loss

Discourage mapping Z → H to distort colors

Images available from: https://github.com/togheppi/CycleGAN

The converse is, if the generator then maps this horse image to something weird, it changes the image in some way. Perhaps it changes the color, for example, which is not uncommon because the generator is mapping between maybe black and whitish color to a brown, and so maybe it found some other way of doing that, and it will try to make the general tone of the input image into something else.

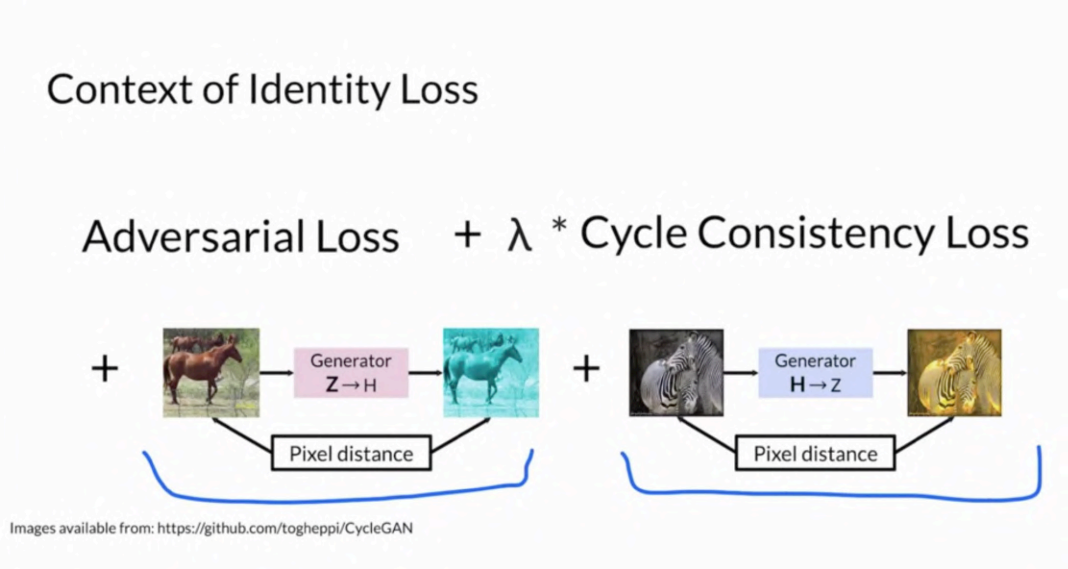
So here you want to take the pixel distance between this input and this output here, and discourage this mapping from being anything but identity.



## Context of Identity Loss

Adversarial Loss $+ \ \lambda \ *$ Cycle Consistency Loss

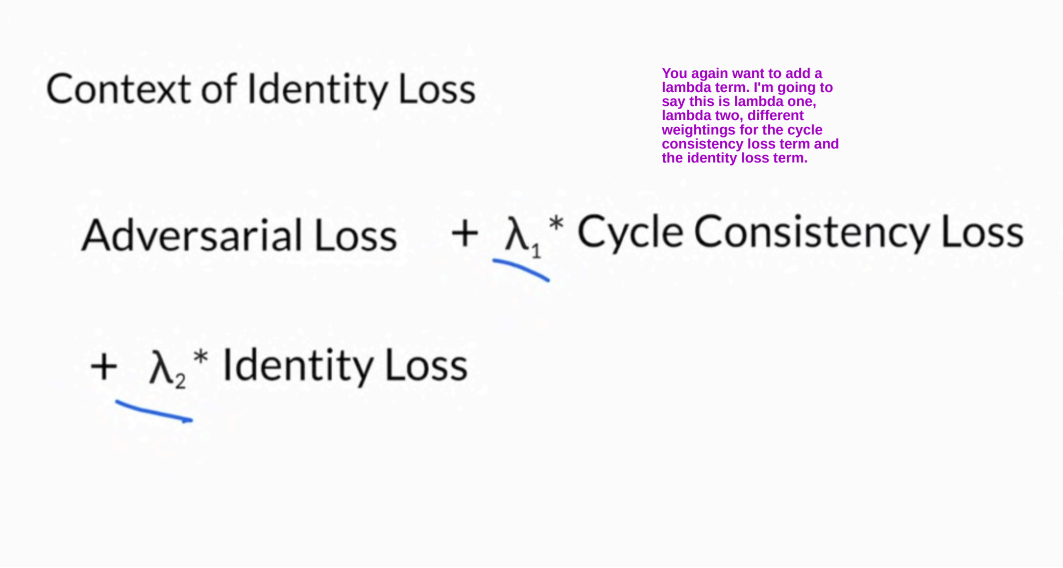
In the context of cycle again, and it's entire loss function for the generators, in addition to your adversarial loss and your cycle consistency loss, you now have this extra identity loss for both the zebra to horse generator where you put in the horse, as well as the horse to zebra generator where you put in the zebra. Together, these make up the identity loss



## Context of Identity Loss

Adversarial Loss $+ \ \lambda \ *$ Cycle Consistency Loss

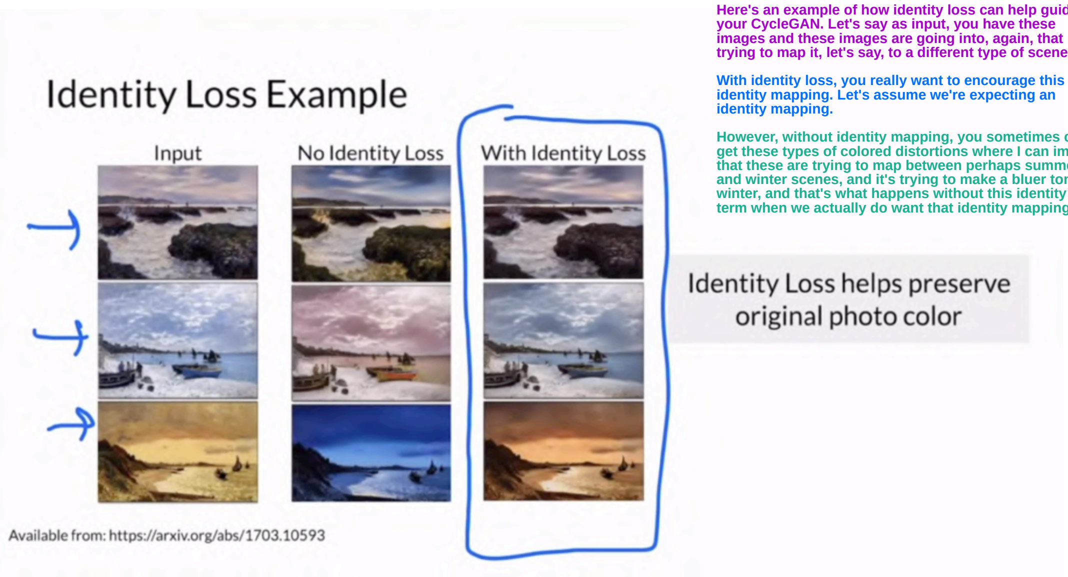
Images available from: https://github.com/togheppi/CycleGAN



## Context of Identity Loss

$$\text{Adversarial Loss} + \lambda_1 * \text{Cycle Consistency Loss} + \lambda_2 * \text{Identity Loss}$$

You again want to add a lambda term. I'm going to say this is lambda one, lambda two, different weightings for the cycle consistency loss term and the identity loss term.



## Identity Loss Example

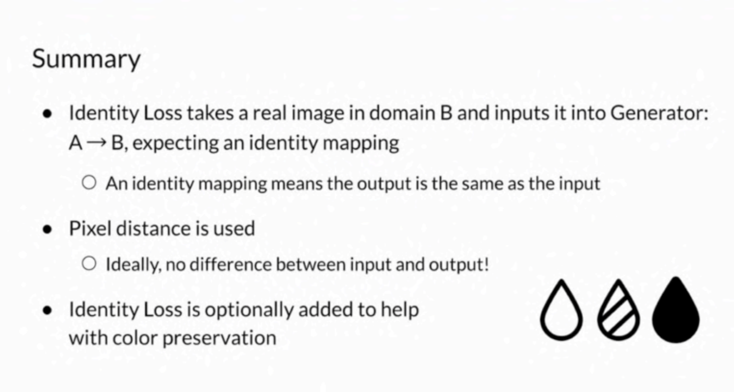
Identity Loss helps preserve original photo color

Available from: https://arxiv.org/abs/1703.10593

Here's an example of how identity loss can help guide your CycleGAN. Let's say as input, you have these images and these images are going into, again, that is trying to map it, let's say, to a different type of scenery.

With identity loss, you really want to encourage this identity mapping. Let's assume we're expecting an identity mapping.

However, without identity mapping, you sometimes can get these types of colored distortions where I can imagine that these are trying to map between perhaps summer and winter scenes, and it's trying to make a bluer tone for winter, and that's what happens without this identity loss term when we actually do want that identity mapping.



## Summary

- Identity Loss takes a real image in domain B and inputs it into Generator: A → B, expecting an identity mapping
  - ○ An identity mapping means the output is the same as the input
- Pixel distance is used
  - ○ Ideally, no difference between input and output!
- Identity Loss is optionally added to help with color preservation

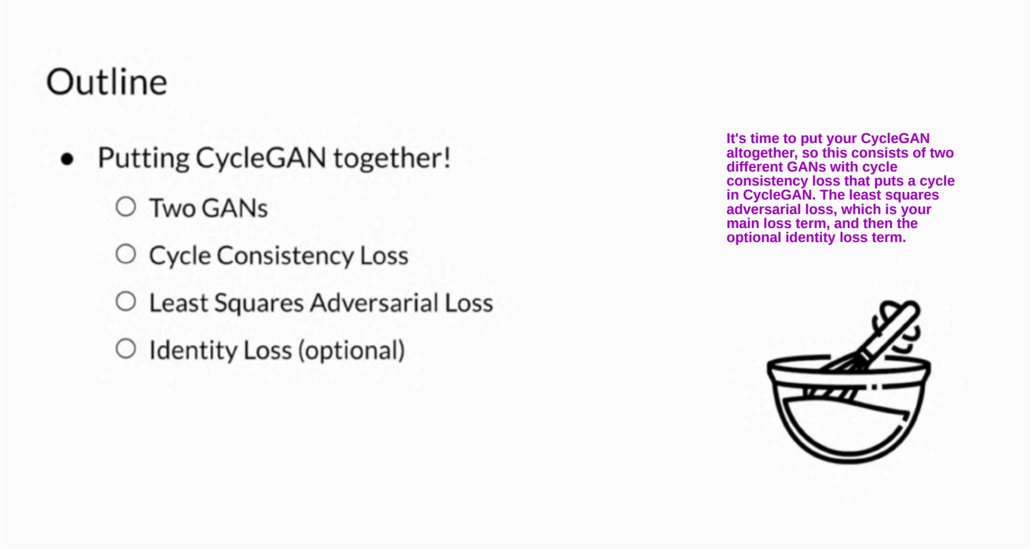
In summary, identity loss takes a real image in one of those piles and inputs it into the opposite generator. What you expect is actually an identity mapping, because that input image that you're putting into your generator, already has the styles that the generator is trying to map it to.

Pixel distance is used to determine this, and ideally there's no difference between input and output. The identity loss is zero.

Identity loss is overall and optional loss term that has been shown to help with colored preservation, in some tasks. The main reason why it's optional is that, it's been shown in many cases, to be very helpful, but in other cases not to make much of a difference.

deeplearning.ai

# CycleGAN: Putting It All Together



## Outline

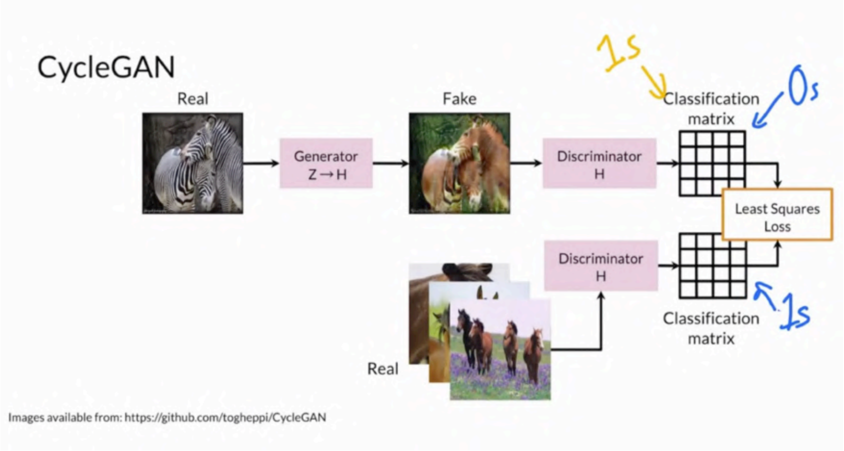
- Putting CycleGAN together!
  - ○ Two GANs
  - ○ Cycle Consistency Loss
  - ○ Least Squares Adversarial Loss
  - ○ Identity Loss (optional)

**It's time to put your CycleGAN altogether, so this consists of two different GANs with cycle consistency loss that puts a cycle in CycleGAN. The least squares adversarial loss, which is your main loss term, and then the optional identity loss term.**

**First, the process with an example going from a real zebra to a fake horse here, but it's also the same the other way around, just the opposite way. And so to start, you first input your zebra image and you get a fake horse using your generator that maps from zebra to horse.**

**Your horse discriminator then looks at this fake image as well as real images. It doesn't know which one's which, and outputs a classification matrix of how real or how fake it thinks those patches of those images are.**

**What's used here is least squares loss, and specifically for reals, this classification matrix is full of ones. And for fakes it's full of zeros, and that's how you compute the least squares adversarial loss. And that's for the discriminator and for the generator, this classification matrix is actually all ones.**

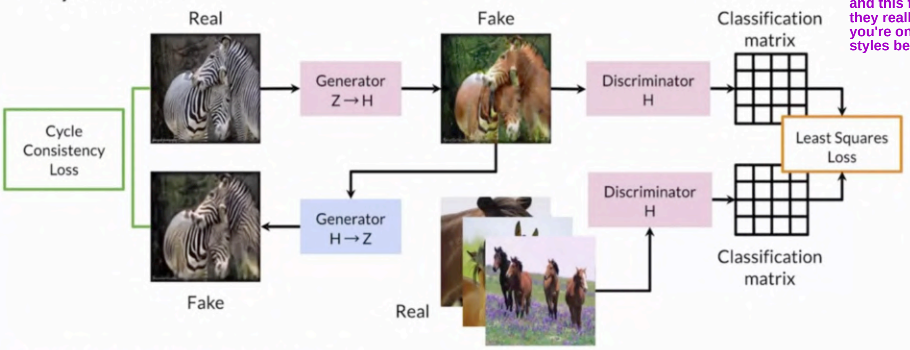


## CycleGAN

Images available from: https://github.com/togheppi/CycleGAN

So in addition to this least squares adversarial loss, you also want to take your fake horse and feed it through the other generator that's going from horses to zeros to generate this fake zebra such that you can then compute the cycle consistency loss in this direction. And you do that by taking the pixel difference between the real input and this fake generated zebra because they really should look the same as you're only supposed to be transferring styles between these two generators.

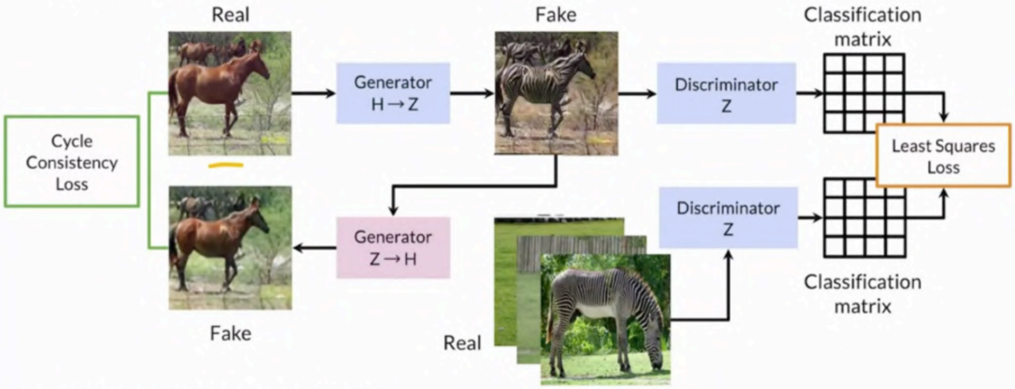
## CycleGAN


Real
Fake
Classification matrix
Generator Z→H
Discriminator H
Cycle Consistency Loss
Least Squares Loss
Generator H→Z
Discriminator H
Classification matrix
Fake
Real

Images available from: https://github.com/togheppi/CycleGAN

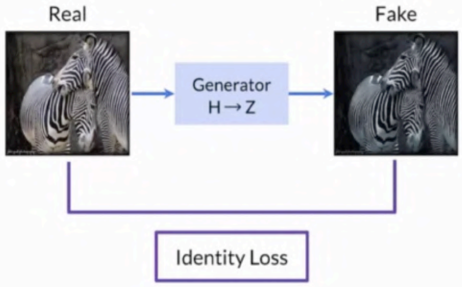
And again, the same is done in the opposite direction as well, and so this is going from horses and zebras. Then back to horses for the cycle consistency loss and then of course their zebra discriminator here will also apply.

## CycleGAN


Real
Fake
Classification matrix
Generator H→Z
Discriminator Z
Cycle Consistency Loss
Least Squares Loss
Generator Z→H
Discriminator Z
Classification matrix
Fake
Real

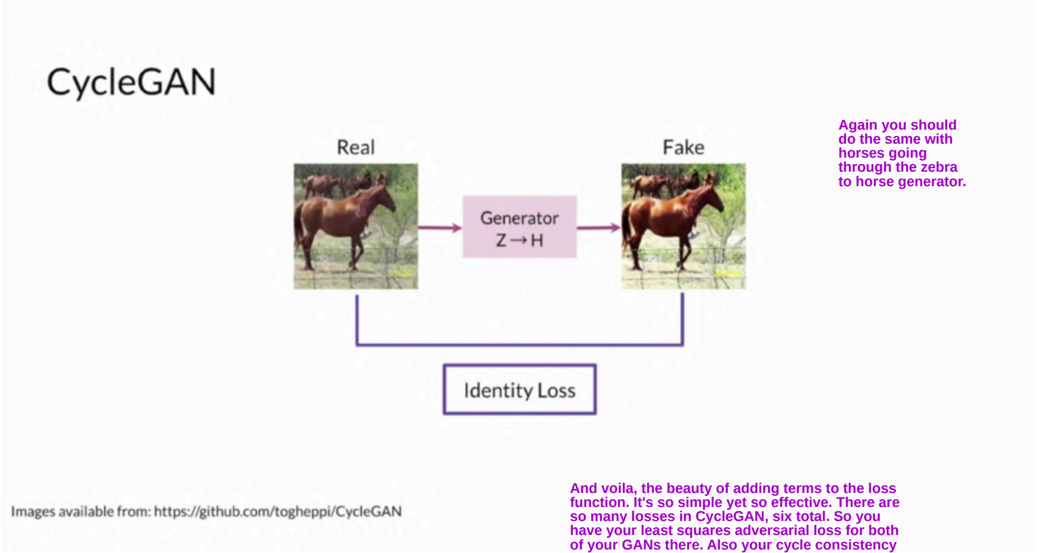
Images available from: https://github.com/togheppi/CycleGAN

## CycleGAN


Real
Fake
Generator H→Z
Identity Loss

If you do choose to use identity loss for your task, you also want to put your zebra through the opposite generator from horses and zebras. To get a zebra you take the pixel difference here and you save that as the identity loss

Images available from: https://github.com/togheppi/CycleGAN



## CycleGAN

Real

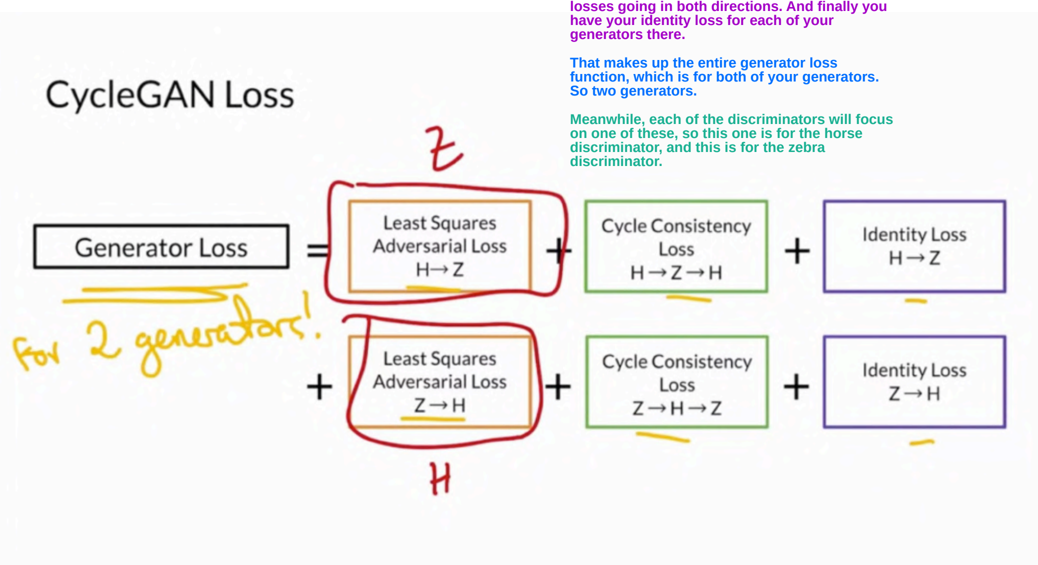
Generator
Z → H

Fake

Identity Loss

Images available from: https://github.com/togheppi/CycleGAN

Again you should do the same with horses going through the zebra to horse generator.



And voila, the beauty of adding terms to the loss function. It's so simple yet so effective. There are so many losses in CycleGAN, six total. So you have your least squares adversarial loss for both of your GANs there. Also your cycle consistency losses going in both directions. And finally you have your identity loss for each of your generators there.

That makes up the entire generator loss function, which is for both of your generators. So two generators.

Meanwhile, each of the discriminators will focus on one of these, so this one is for the horse discriminator, and this is for the zebra discriminator.

## CycleGAN Loss

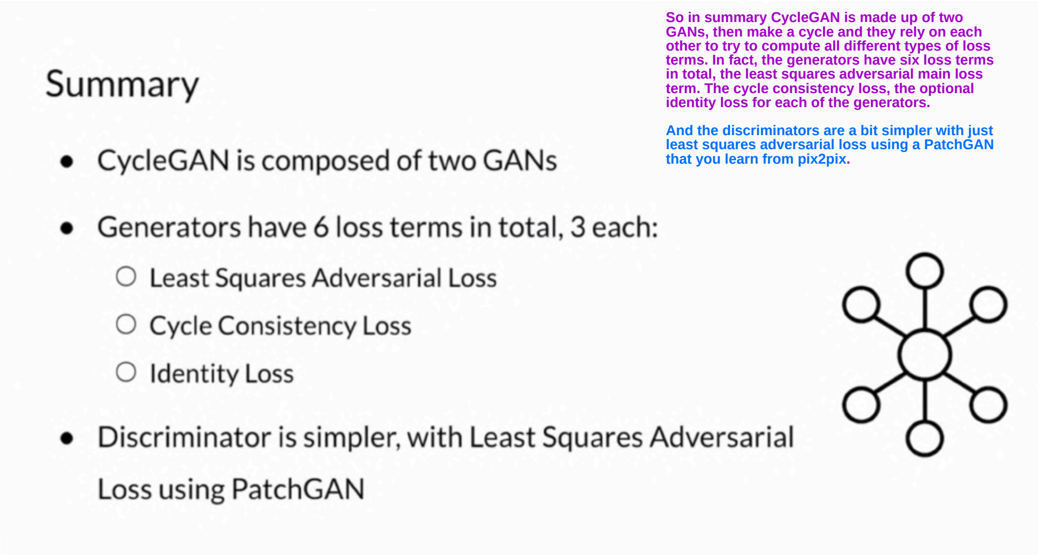
Generator Loss = Least Squares Adversarial Loss H → Z + Cycle Consistency Loss H → Z → H + Identity Loss H → Z

+ Least Squares Adversarial Loss Z → H + Cycle Consistency Loss Z → H → Z + Identity Loss Z → H

Z

H

For 2 generators!

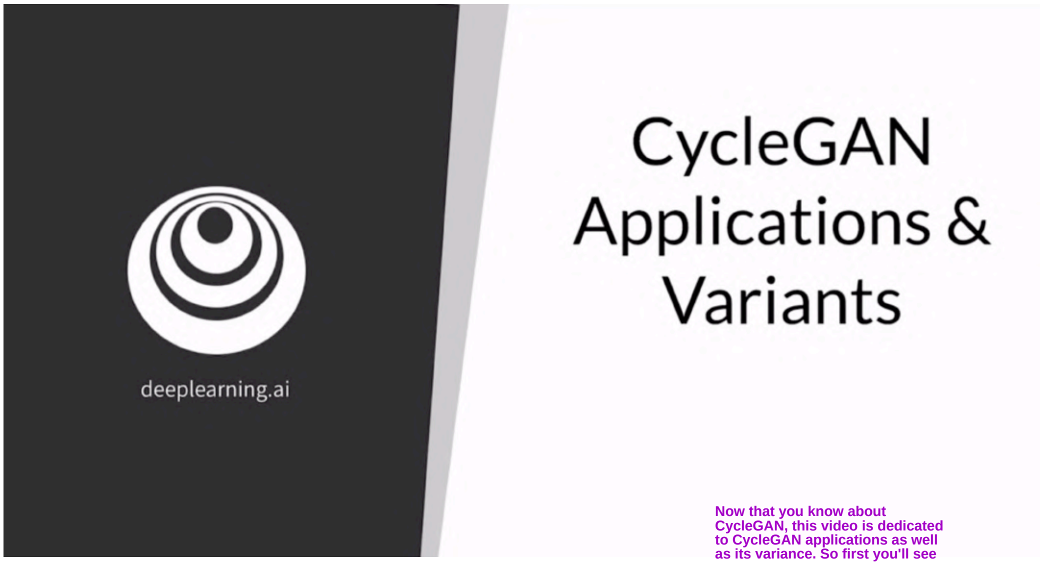


## Summary

- CycleGAN is composed of two GANs
- Generators have 6 loss terms in total, 3 each:
  - ○ Least Squares Adversarial Loss
  - ○ Cycle Consistency Loss
  - ○ Identity Loss
- Discriminator is simpler, with Least Squares Adversarial Loss using PatchGAN

So in summary CycleGAN is made up of two GANs, then make a cycle and they rely on each other to try to compute all different types of loss terms. In fact, the generators have six loss terms in total, the least squares adversarial main loss term. The cycle consistency loss, the optional identity loss for each of the generators.

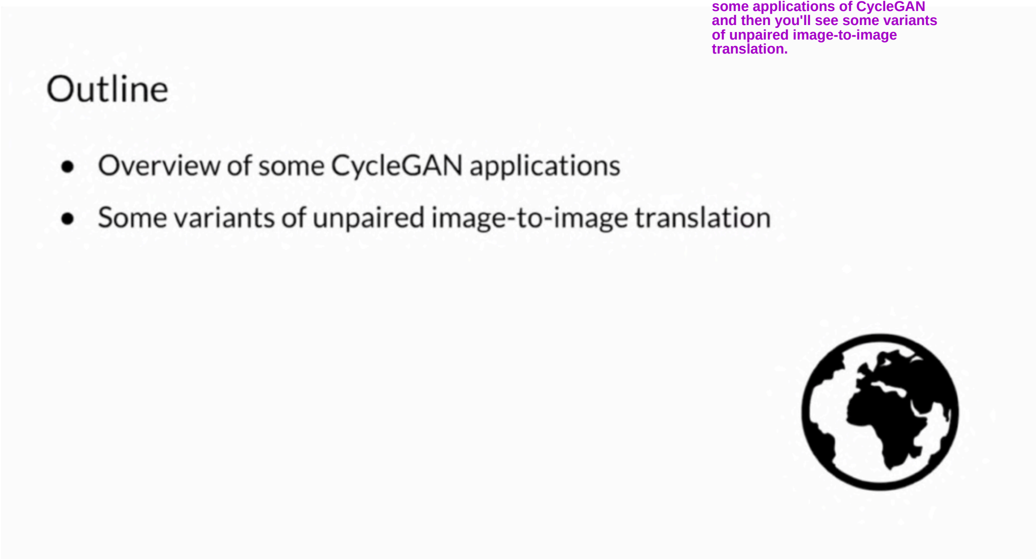
And the discriminators are a bit simpler with just least squares adversarial loss using a PatchGAN that you learn from pix2pix.



# CycleGAN Applications & Variants

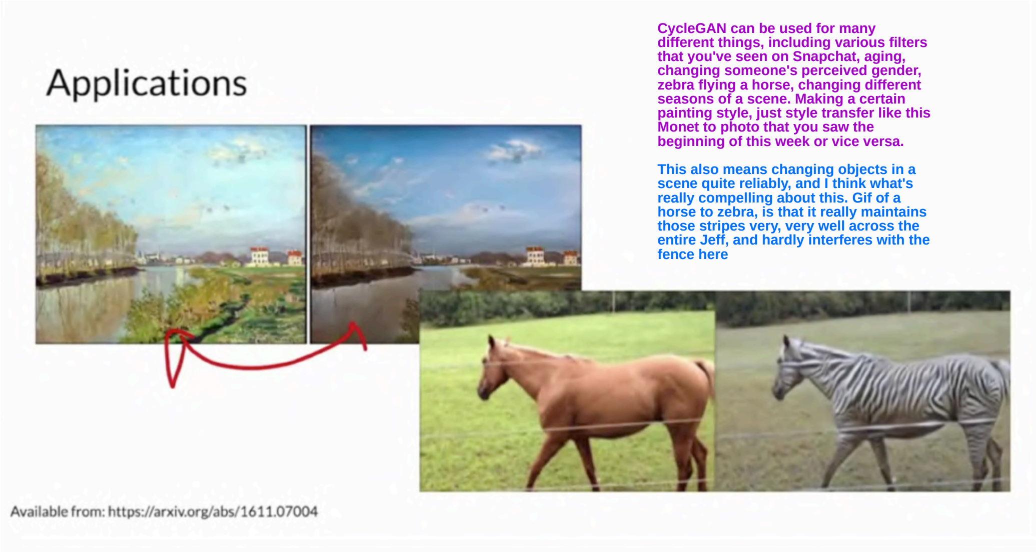
deeplearning.ai

Now that you know about CycleGAN, this video is dedicated to CycleGAN applications as well as its variance. So first you'll see some applications of CycleGAN and then you'll see some variants of unpaired image-to-image translation.



## Outline

- Overview of some CycleGAN applications
- Some variants of unpaired image-to-image translation



## Applications

CycleGAN can be used for many different things, including various filters that you've seen on Snapchat, aging, changing someone's perceived gender, zebra flying a horse, changing different seasons of a scene. Making a certain painting style, just style transfer like this Monet to photo that you saw the beginning of this week or vice versa.

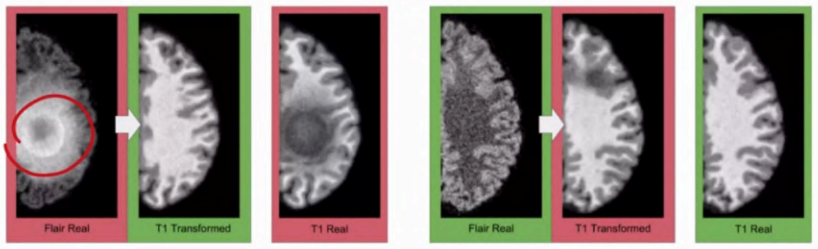
This also means changing objects in a scene quite reliably, and I think what's really compelling about this. Gif of a horse to zebra, is that it really maintains those stripes very, very well across the entire Jeff, and hardly interferes with the fence here

Available from: https://arxiv.org/abs/1611.07004

Beyond the immediate media applications CycleGAN is also often used in data augmentation. For example, it's really hard to get paired images of someone who has a tumor now, but then also getting an image of when they did not have that tumor. That kind of pressions is not something we can probably predict right now. And so this might be the way to go, to be able to get some data augmentation here so you can of course hallucinated tumor, hallucinate away a tumor to then get those paired examples for downstream applications.

For classification for tumor segmentation for monitoring, the growth of that tumor which is really important in medical applications. So here you first see removing a tumor using a CycleGAN which removes this large spot and then here you are adding a tumor.

## Applications



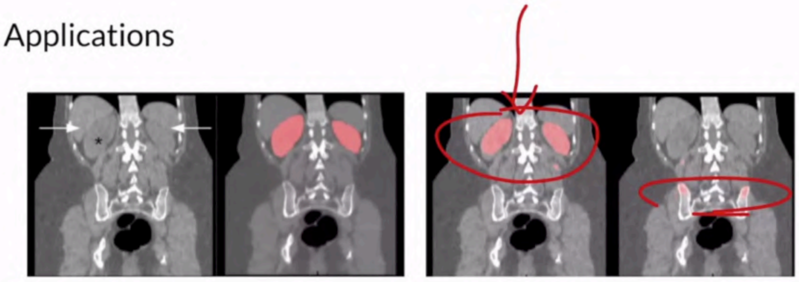
(a) A translation removing tumors

(b) A translation adding tumors

Available from: https://arxiv.org/abs/1805.08841

For data augmentation in general CycleGAN is able to generate extremely realistic examples. For example, here you have a CT scan and you have an expert who's made the correct segmentation. CycleGAN is able to kind of hallucinate those segmentations quite well while using Standard Augmentation, you're not able to get there at all. In fact, you're not segmenting the right place at all there.

## Applications



CT

Expert

CycleGAN

Standard Augmentation

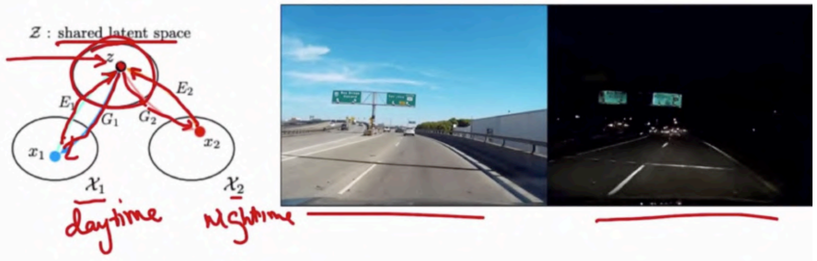
Available from: https://www.nature.com/articles/s41598-019-52737-x.pdf

CycleGAN isn't the only model out there that is able to do unpaired image-to-image translation. There is a variant that called UNIT, which stands for Unsupervised Image-to-Image Translation, which you can think of unpaired image-to-image translation as being unsupervised cause you don't have labels for it.

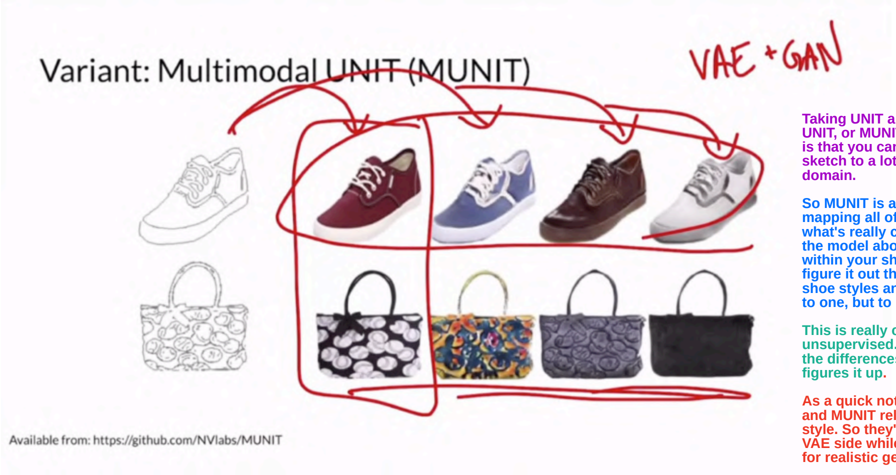
Here's a gif between a daytime in a nighttime scene of a car driving. And the key insight in this model is known as this shared latent space, which basically states that given a noise vectors Z in this latent space. It can generate an image in domain X1, so it can generate that image, and there's a mapping back to that latency, for example. And the same latency also is able to map to another domain saying X2 and also map back to that latent again.

You can imagine these two domains as X1 being that day-time car drive, and this one X2 being the night time domain. And so X1 and X2 are really just the styles in the shared latent space is saying, hey we can share the same link in the same Z and still produce those two different styles. We can solve the same content and still produce those two different styles that can map back and forth, not just in one direction.

## Variant: UNIT



Available from: https://github.com/mingyuliutw/UNIT



Variant: Multimodal UNIT (MUNIT)

VAE + GAN

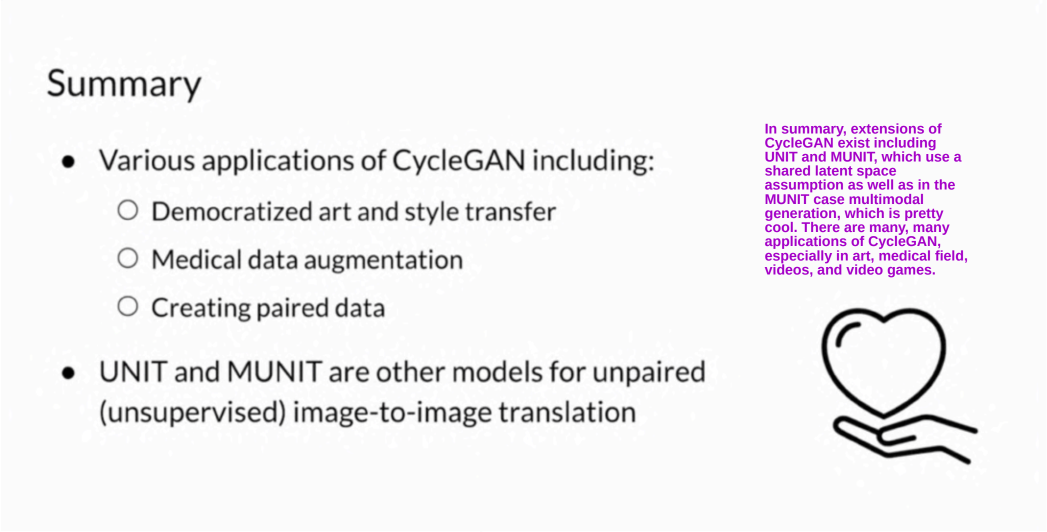
Available from: https://github.com/NVlabs/MUNIT

Taking UNIT a step further is Multimodal UNIT, or MUNIT, and what multimodal means is that you can go from one domain of a sketch to a lot of modes in the second domain.

So MUNIT is able to find not only this one mapping all of these others as well. And what's really cool is you actually never tell the model about all of these different styles within your shoe pile, for example. It's able to figure it out that you have all these different shoe styles and is able to map this not only to one, but to many.

This is really cool because it is unsupervised. You don't have to label what the differences between the shoes are, it figures it up.

As a quick note, both these techniques UNIT and MUNIT relying on encoding an image or style. So they're getting inspiration from the VAE side while also using GAN components for realistic generation.



Summary

- Various applications of CycleGAN including:
  - ○ Democratized art and style transfer
  - ○ Medical data augmentation
  - ○ Creating paired data
- UNIT and MUNIT are other models for unpaired (unsupervised) image-to-image translation

In summary, extensions of CycleGAN exist including UNIT and MUNIT, which use a shared latent space assumption as well as in the MUNIT case multimodal generation, which is pretty cool. There are many, many applications of CycleGAN, especially in art, medical field, videos, and video games.

# (Optional) The CycleGAN Paper

Compelled to learn more about CycleGAN? Take a look at the original paper!

Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks (Zhu, Park, Isola, and Efros, 2020): https://arxiv.org/abs/1703.10593

# (Optional) CycleGAN for Medical Imaging

Intrigued by the application of CycleGANs in the medical field? See how they can be used to help augment data for medical imaging!

Data augmentation using generative adversarial networks (CycleGAN) to improve generalizability in CT segmentation tasks (Sandfort, Yan, Pickhardt, and Summers, 2019): https://www.nature.com/articles/s41598-019-52737-x.pdf

# Works Cited

All of the resources cited in Course 3 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:

- Image-to-Image Translation with Conditional Adversarial Networks (Isola, Zhu, Zhou, and Efros, 2018): https://arxiv.org/abs/1611.07004
- Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks (Zhu, Park, Isola, and Efros, 2020): https://arxiv.org/abs/1703.10593
- PyTorch implementation of CycleGAN (2017): https://github.com/togheppi/CycleGAN
- Distribution Matching Losses Can Hallucinate Features in Medical Image Translation (Cohen, Luck, and Honari, 2018): https://arxiv.org/abs/1805.08841
- Data augmentation using generative adversarial networks (CycleGAN) to improve generalizability in CT segmentation tasks (Sandfort, Yan, Pickhardt, and Summers, 2019): https://www.nature.com/articles/s41598-019-52737-x.pdf
- Unsupervised Image-to-Image Translation (NVIDIA, 2018): https://github.com/mingyuliutw/UNIT
- Multimodal Unsupervised Image-to-Image Translation (Huang et al., 2018): https://github.com/NVlabs/MUNIT

From the notebooks:

- PyTorch-CycleGAN (2017): https://github.com/aitorzip/PyTorch-CycleGAN/blob/master/datasets.py
- Horse and Zebra Images Dataset: https://people.eecs.berkeley.edu/~taesung_park/CycleGAN/datasets/horse2zebra.zip