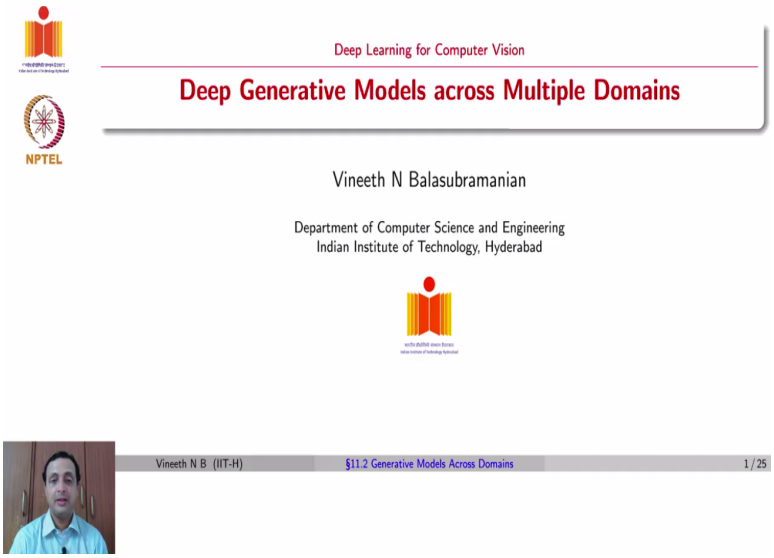
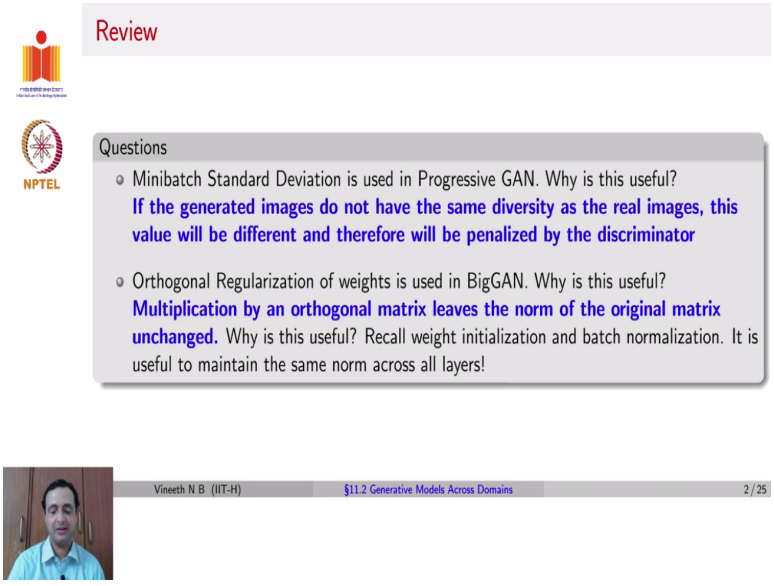
Deep Learning for Computer Vision
Professor Vineeth N Balasubramanian
Department of Computer Science and Engineering
Indian Institute of Technology, Hyderabad
Deep Generative Models Across Multiple Domains

(Refer Slide Time: 00:12)



In this lecture, we will talk about an interesting use case of GANs, which is Generating Images across Domains.

(Refer Slide Time: 00:26)



Before we get there, let us answer the questions that we left behind. One of the questions was Minibatch standard deviation is used in Progressive GAN. Why is this useful? I hope you had a chance to try to find this. The answer is, in Minibatch standard deviation, the standard deviation at each spatial location in a feature map across a Minibatch is concatenated in a later layer of a discriminator.

The standard deviation gives an idea of the diversity of the images generated in a given Minibatch. If this diversity is significantly different from the diversity in the real images from a real dataset, that would incur a penalty, and the generator would learn to generate diverse images. And that is the main idea of including this in Progressive GAN. The second question was, why is Orthogonal Regularization of weights used in BigGAN?

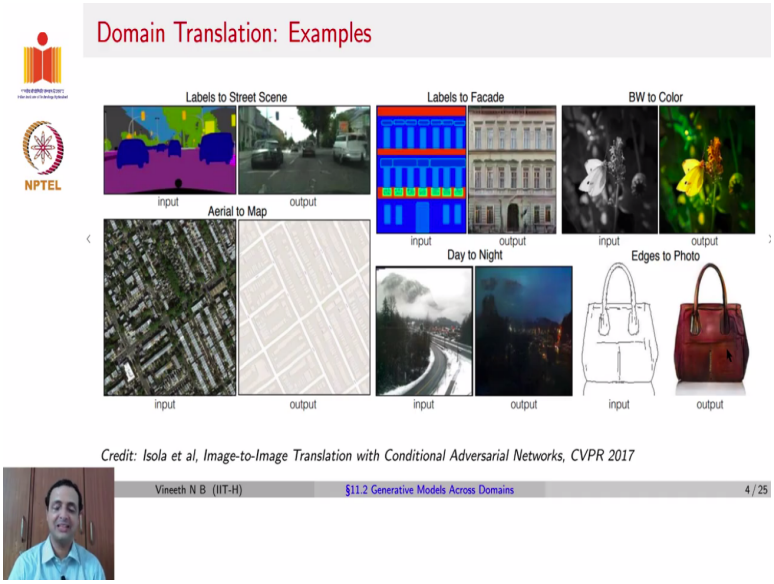
The answer comes from linear algebra. Multiplication of a matrix by an orthogonal matrix leaves the norm of the original matrix unchanged. Why is this useful? You have to recall Weight Initialization and Batch Normalization. It is useful and important to maintain the same norm across all layers and orthogonal regularization is a method that tries to achieve this during training.

(Refer Slide Time: 02:22)

With that, let us move on to using GANs for generation across Domains, a task known as Domain Translation. The goal here is, given an image from a source domain, we would like to

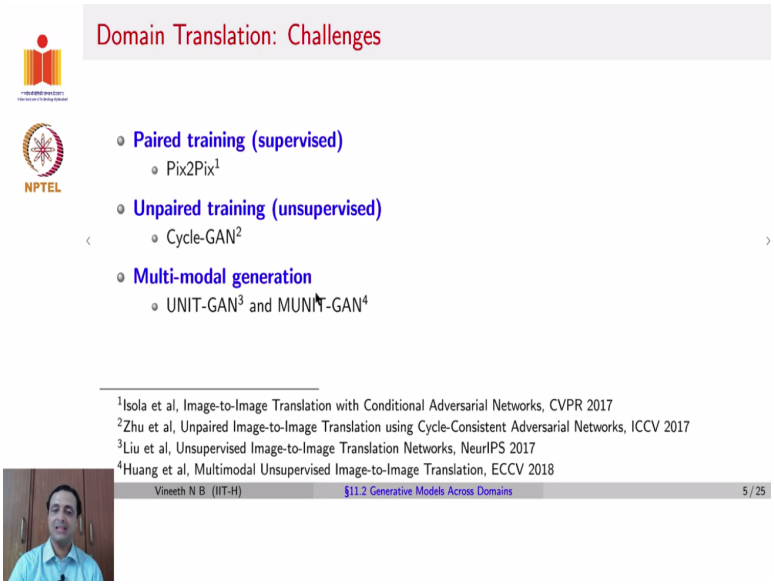
generate an image in a target domain. We would like to learn this function G, which takes us from source to target. You could look at this as a variant of GANs, where the input is not a noise vector but a source domain image. There is more to it than just changing the input. Examples of use cases could be to take a Male image to change it to Female, to go from sketches to photos, to take a scene, and transform a summer scene to winter, and so on.

(Refer Slide Time: 03:21)



Here are some examples of how Domain Translation can be used. Here is an example of going from Semantic Segmentation Labels to a Street Scene. Similarly, Labels to a Facade, Black and White to Color, Day to Night, could be very useful for Autonomous Navigation or Self Driving datasets. Going from an Aerial View to a Google map, the output is from a Sketch to a Photo. All of these are examples of domain translation.

(Refer Slide Time: 04:01)



So, there are a few settings under which different methods have been proposed, which we will focus on in this lecture. In the first setting, known as Paired Training or the supervised setting, you are given images from both domains in a paired manner. The goal is to train a GAN to translate for a new image from one domain.

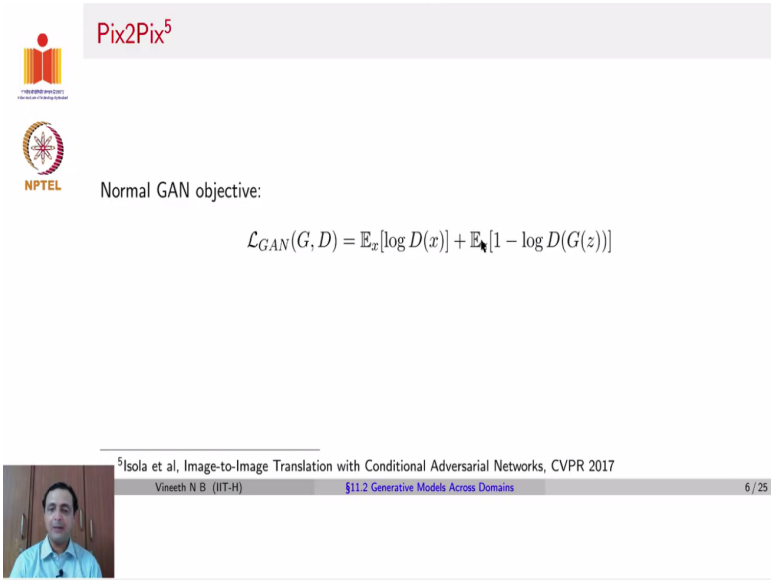
So, for every sketch, you are also given its corresponding photo in your dataset. This is the first and simplest setting. In the second setting, we will talk about Unpaired training or unsupervised, where you have a set of sketches, and you have a set of photos. They are not necessarily paired. So, you do not know if, for a given sketch, the corresponding photo is there in the dataset or not.

But you still have to learn to go from a sketch to a photo, and we call this Unpaired image to Image Translation. Finally, we will talk about Multi-modal generation, where you can go seamlessly between domains, where the popular methods are UNIT and MUNIT. Let us see each one in detail.

(Refer Slide Time: 05:27)

The first method is for Paired translation. The popular method here is Pix2Pix. Pix2Pix defines an image to image translation task as predicting pixels from pixels, and that is why the name Pix2Pix. It provides a framework to perform all such tasks.

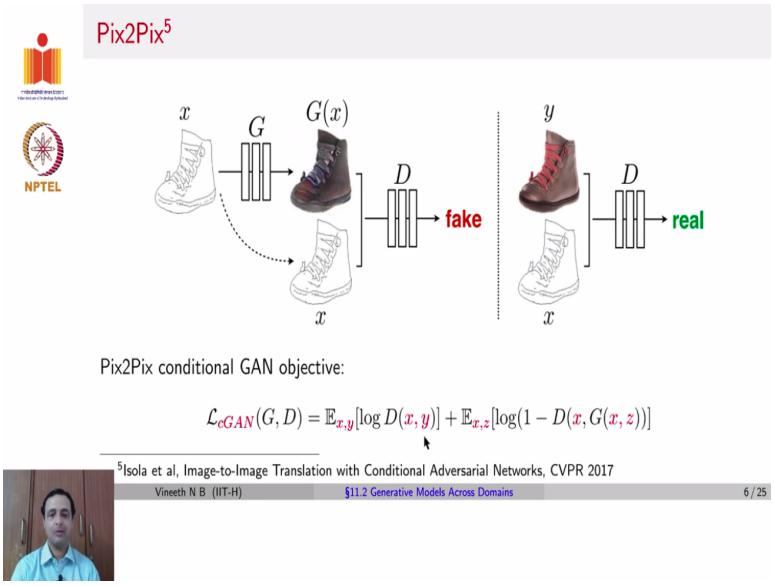
(Refer Slide Time: 05:53)



Pix2Pix builds upon the standard GAN objective. Recall that the standard GANs objective is to maximize the likelihood of the discriminator and minimize the fooling rate of the generator. In

contrast, the generator tries to maximize the second term. However, when adapting this for the image to image translation tasks, the objective changes slightly.

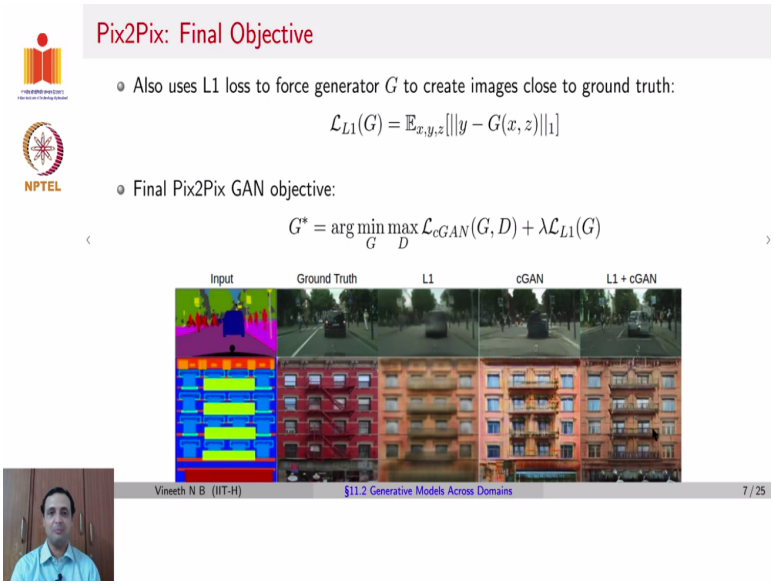
(Refer Slide Time: 06:23)



Pix2Pix defines this as a conditional GAN objective. Instead of just an image x coming from the real domain, you have an image x and the corresponding image y from another domain. In a standard GAN, given an image and given the generated image, one would have to see which is fake and real. In Pix2Pix, given an image and given another image, which may not be exactly similar to $G(x)$. The discriminator has to tell whether this is a correct translation or not. So, the conditional GAN objective now is given by the discriminator, which has to maximize the probability of x and y, assuming they are the correct paired images from sketches and photos to be real. So, that is something the discriminator has to do.

And the second term is where given x, a sketch, for example, and z, a latent vector, $G(x, z)$ generates a photo. So, you could consider G to be given input from one domain. In this case, sketch G generates an image from the other domain. The discriminator's job is to take the original sketch and the generated photo and see if both can be classified to be fake. The generator would want a log of 1 minus that quantity to be high. That is the min-max game generator, and the discriminator would play here. In addition to the Vanilla objective, you now have the x, y tuples to manage inside the GAN objective.

(Refer Slide Time: 08:34)

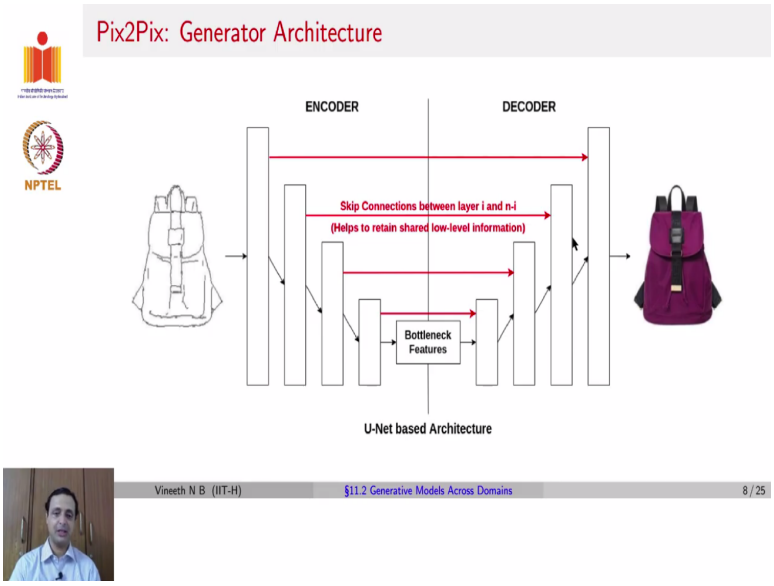


In addition to doing this, Pix2Pix also introduces an L_1 objective to ensure that the generated image matches the original expected photo from the second domain. How is this done? The generator also tries to minimize the L_1 loss, which is the sum of absolute values of each element between y which is the image from domain **two** in our case, could be a photo and $G(x, z)$, x again is a sketch, an input from domain **one**.

And z is the latent noise vector given as input to the generator. $G(x, z)$is the generated image from domain **two**. We would like to match y as closely as possible, captured by this term in the loss function. The overall objective now becomes the standard min-max GAN objective, which is captured in the first term, represented as cGAN or conditional GAN plus some coefficient λ times the L_1 loss that forces the generated images to be close to ground truth.

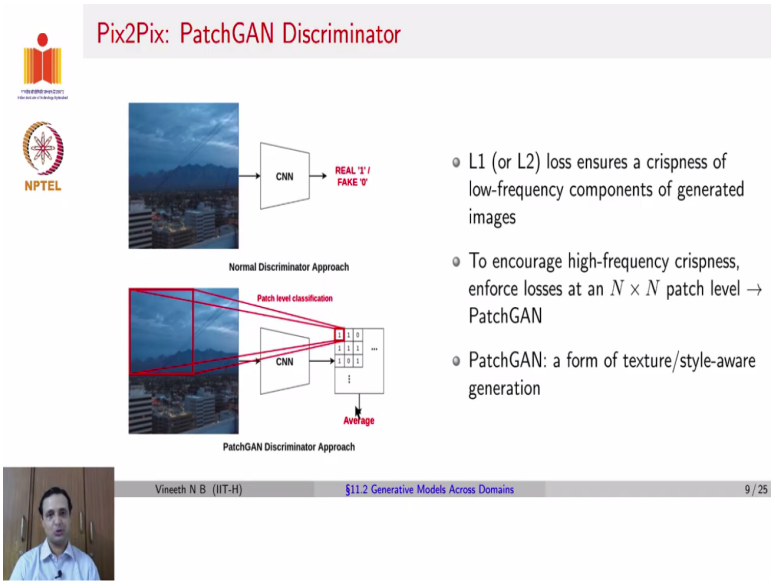
With this objective Pix2Pix, obtains fairly impressive results. A couple of examples are shown, where the input is a Semantic Segmentation mask, which is one domain, and the aim is to generate the scene image, which is the other domain. One can see that using only with the L_1 loss; the generation is very blurry. Using conditional GAN, the generation improves, and when the two are put together, the generation has a fair good amount of detail and is close to the ground truth image for this example. You see a similar observation even with the second image, where one goes from the Semantic Segmentation mask to the Actual Facade picture.

(Refer Slide Time: 10:51)



The generator architecture in Pix2Pix resembles a U-Net-based architecture. The encoder reduces the dimensions of layers until a set of bottleneck features, which are then upsampled to get the final dimension of the input image or the desired image as the output. Similar to U-Net, there are skipped connections. These connections go from each layer in the encoder to its corresponding mirror layer in the decoder, similar to what we saw in U-Net for Semantic Segmentation.

(Refer Slide Time: 11:34)

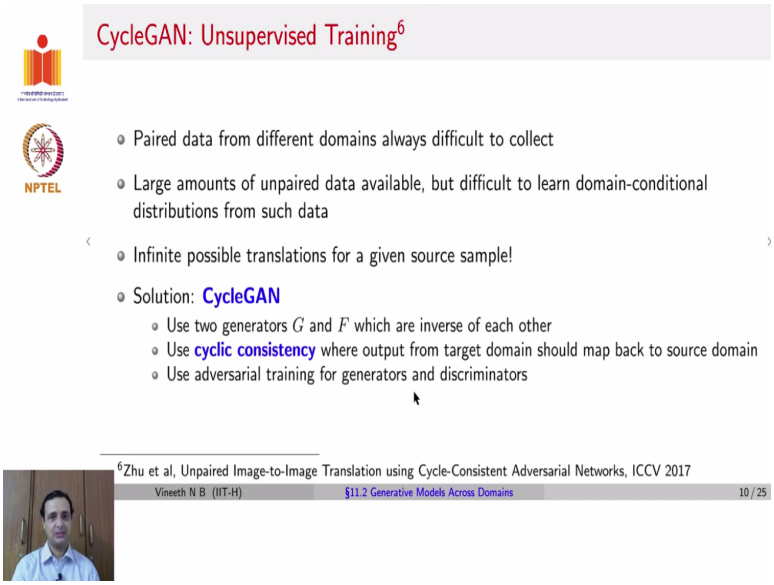


In addition, Pix2Pix also uses what is known as a PatchGAN Discriminator. In a standard discriminator of a GAN, even if L1 or L2 loss is used as a regularizer the way Pix2Pix introduced it. This ensures the crispness of low-frequency components of generated images. So, you do get crispness in the output of certain large objects in the image. If one wanted finer details, you need to do better than the L1 loss at the image level.

The PatchGAN Discriminator introduces L1 loss at a patch level between the generated image and the original expected ground truth or the input. So, in this case, there is an enforcement of a patch level classification of the generated image, comparing it to the ground truth and saying whether it is real or fake. This is done for all patches in the generated image. The average is taken to decide whether the generated image is real or fake, which is then used in the loss to backpropagate and train the generator.

One could also look at PatchGAN as a form of texture or style aware generation so that finer details or textures in the image can be generated better using a local patch wise discriminator approach.

(Refer Slide Time: 13:21)

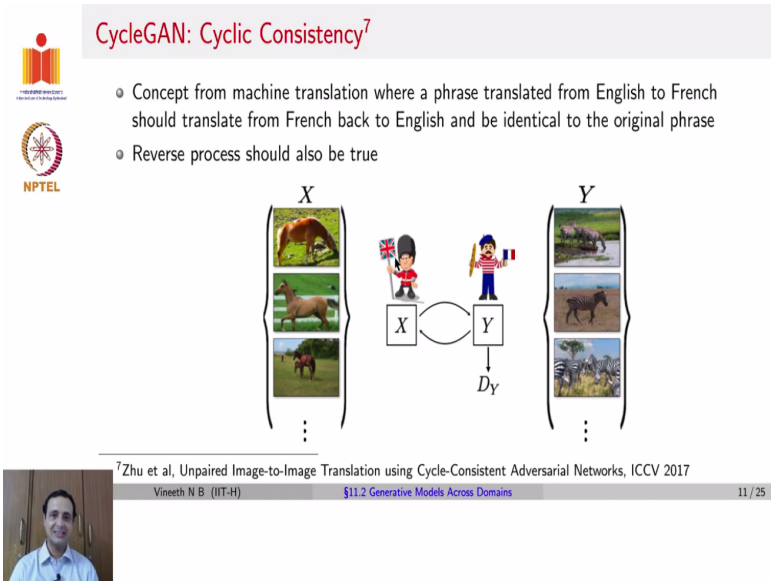


A second domain translation method or image to image translation method is Unpaired image to image translation, an Unsupervised approach called CycleGAN, a popular approach again. CycleGAN is premised on the observation that pair data from different domains can be

challenging to collect. It is not very easy for every sketch to obtain its corresponding photo and thus build a dataset. On the other hand, what may be more accessible is you could get large amounts of Unpaired data where you have a set of sketches and a set of photos. You may not necessarily have a paired photo for each sketch. They could just be loosely different sets. In this case, it becomes challenging to learn domain conditional distributions the way Pix2Pix learned to generate these images. The challenge here is you could have infinite possible translations for a given source sample.

Because the pairing is not known in the training data. Given a sketch, there could be infinite ways to transform this sketch into a photorealistic image. How do you handle this? That is where CycleGAN comes into the picture. It uses two generators, G and F, which are intended to be inverse functions of each other. It uses a concept called Cycle consistency, where the idea is that the output from the target domain should also map back to the source domain and match the input image. It uses adversarial training for generators and discriminators to achieve this.

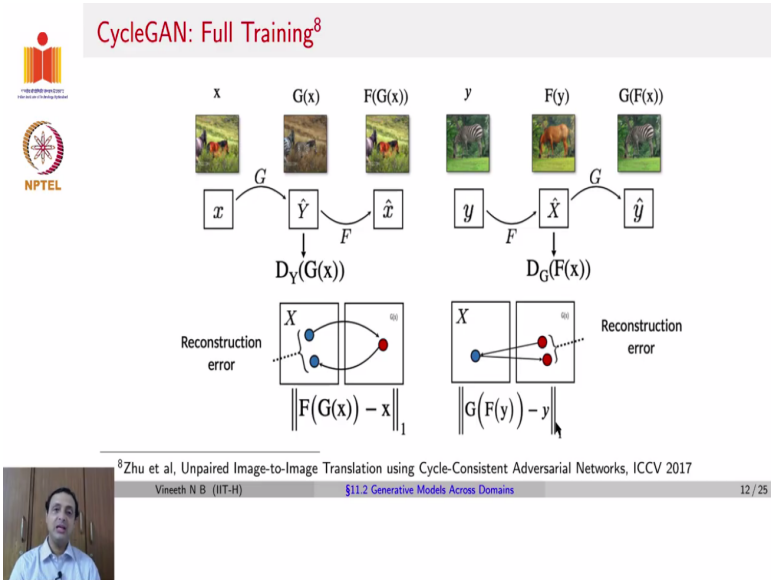
(Refer Slide Time: 15:32)



Let us see what Cyclic Consistency mean. The name CycleGAN comes from this use. Cyclic Consistency is similar to a concept from machine translation, where a phrase translated from English to French should also translate back from French to English and get you back the original sentence in English. This would ensure that the translation is complete and the generated French sentence can recover from the original sentence.

One would want the reverse process also to be true. If you start with French, go to English, then the generated English sentence should get back the French sentence. It is shown pictorially in the slide. So, given an image from one of the images from this input domain X, you can go to an image from the second domain Y. If you generate back the image of the original domain from Y, you should get back the original image you started with. This is the key idea of Cyclic Consistency in CycleGANs.

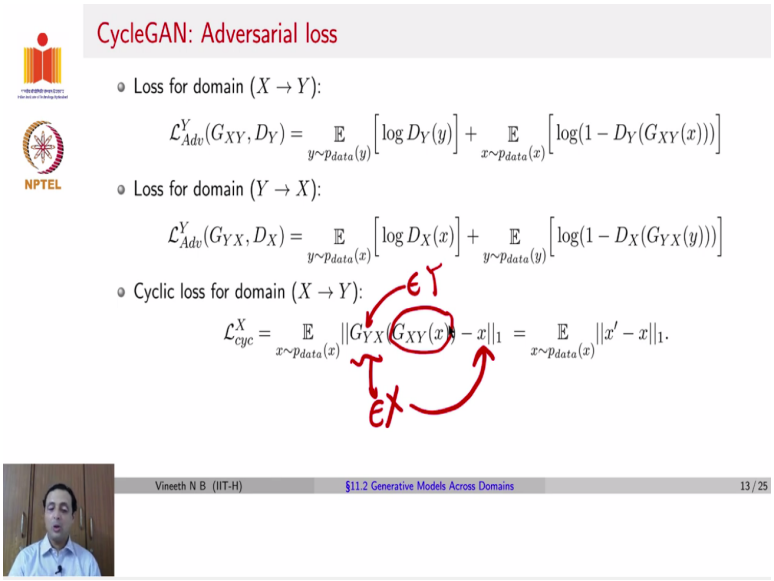
(Refer Slide Time: 17:00)



The way this is implemented is, given an input image, coming from Domain 1, G generates the version of the image in Domain 2, given by G(x). This input is given to F, F(G(x)), which should be close to x. The L1 norm gives the loss function to minimize $||F(G(x)) - x||_1$.

Similarly, if you start from the second domain, y, generate an image in the first domain, which is given by $F(y)$. Then apply the transformation $G(F(y))$, the reconstruction error to be minimized is given by $||G(F(y)) - y||_1$, the L1 norm, which should be close to 0. These are the criteria that help CycleGAN work, even with unpaired images.

(Refer Slide Time: 18:15)



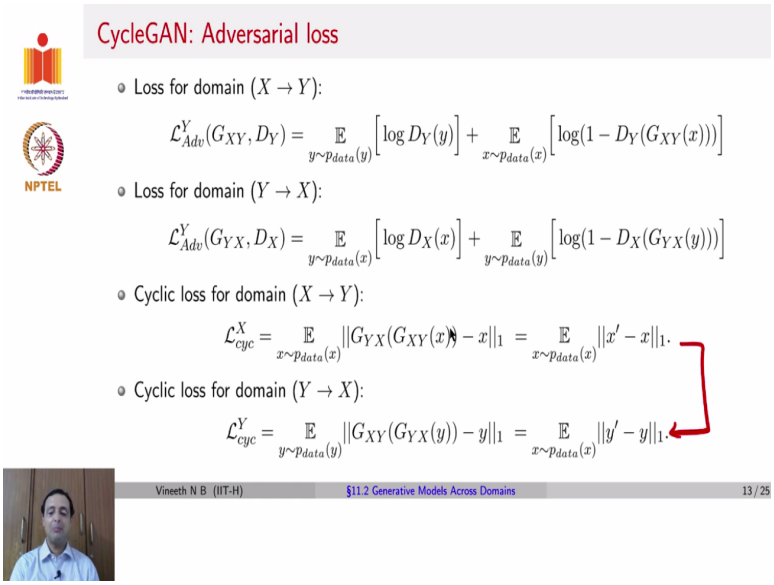
Adversarial losses give the loss functions. Let us elaborate on each of them. So, to go from domain X to domain Y, the adversarial loss is to maximize the discriminator's output on Y, and the discriminator would want $D_Y(G_{XY}(x))$ to go to 0 and the generator would want $1 - D_Y(G_{XY}(x))$ to go to 0 or $D_Y(G_{XY}(x))$ to go to 1. So, this would ensure that the image generated from X in the Y domain looks like a real Y to the discriminator D_Y.

This is the first Adversarial loss. Similarly, one could have a reverse adversarial loss for an image going from domain Y to domain X. The second loss is simply the converse or the complement of the first loss function. The only difference is the input is an image from domain Y, and the output is an image from X given by generator G_{YX}.

While these two losses give domain specific losses to go from X to Y and Y to X. We then have the Cyclic loss for domain X to Y, which, as we already mentioned, is given by $G_{XY}(x)$, which gives you output in domain Y and then considering G_{YX}, which is the generator going from Y to X of this value in domain Y. So, remember, this quantity gives you an element in domain Y, and then applying the generator that goes from Y to X gives you back an element in domain x.

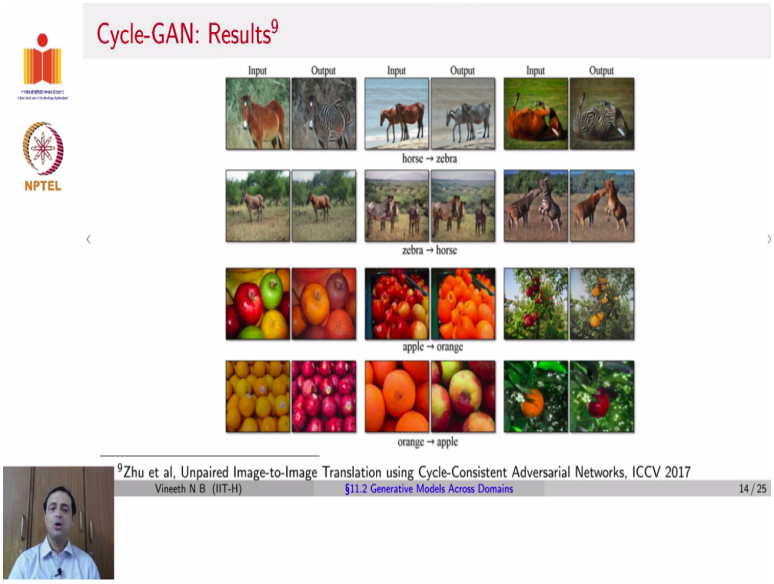
You would want this final generated output to be close to the x that you started with. So, the L1 loss tries to minimize the loss between the final reconstruction in the original domain and the ground truth that we started with.

(Refer Slide Time: 20:56)



We also need a similar cyclic loss to go from Y to X, which is a complement of the loss from X to Y. These four losses are put together to help train the CycleGAN.

(Refer Slide Time: 21:10)

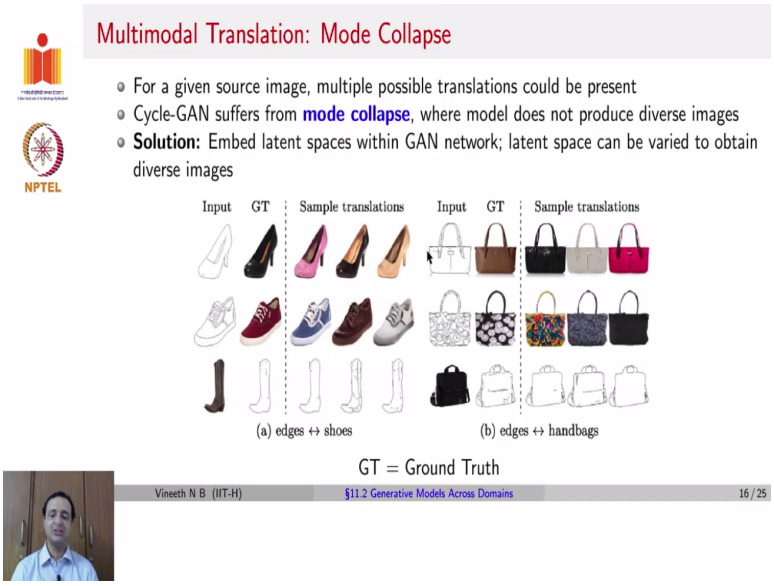


And with this, Cycle-GAN gives impressive results, where given an input image, it gives an output image from another domain. You can see different kinds of images.

(Refer Slide Time: 21:28)

To make this a little bit more tangible and clearer. Here is an input image, and the output image shows the input image in different artist styles, such as Van Gogh, Monet, Cezanne, etc. You can see here that the styles of the artists remain the same. But for the input image, which is translated in that artist's style.

(Refer Slide Time: 21:55)

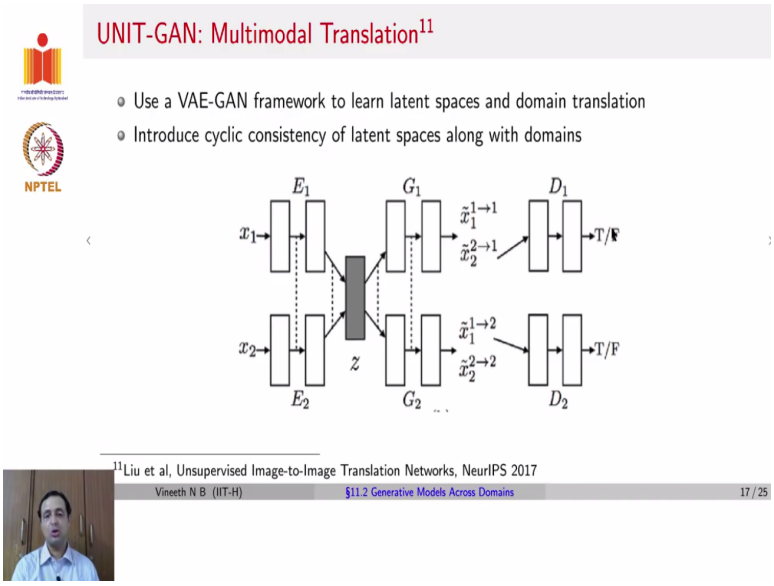


However, CycleGAN has one problem because it is possible to have multiple possible translations for a given source image. This leads to what is known as Mode Collapse, where the model may not be able to produce diverse images because CycleGAN could just generate one image from which one can retrieve the original image from the source domain and still get a low loss. It does not explicitly try to generate different kinds of images in the second domain or the target domain, for that matter.

So, the solution to address the mode collapse problem in CycleGANs is to embed latent spaces inside the GAN framework. How do we do that? By combining VAEs and GANs. Recall that VAEs introduce a latent space that is learned through an encoder-decoder framework. So, we will talk about these methods now that embed latent spaces inside the GAN framework to address this Mode collapse problem.

What we ideally want is, given an input edge image or a sketch, we would want several kinds of domain translations. You may want a Pink shoe, a Black shoe or a Beige shoe. Based on certain changes in a latent variable, which is learned through a VAE. You can see applications of such an approach in, say, Fashion and Apparel purchase so on and so forth.

(Refer Slide Time: 23:55)



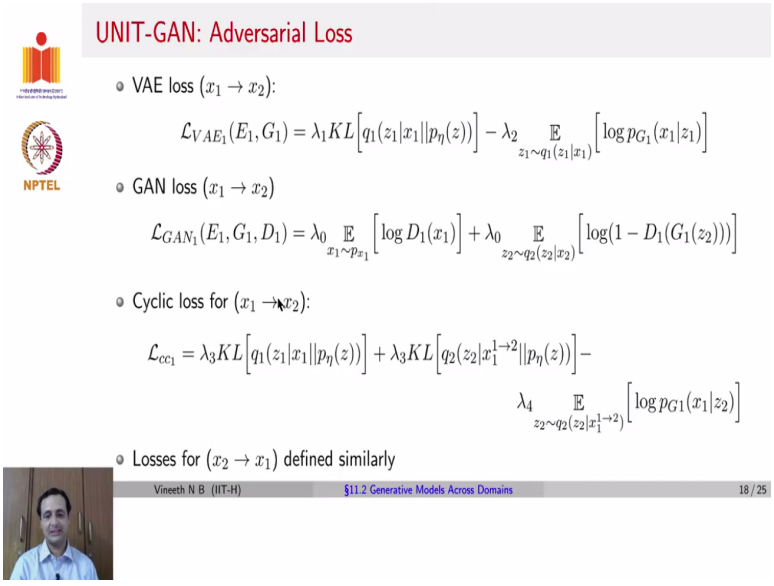
One of the earliest methods in this direction is known as UNIT. UNIT stands for Unsupervised Image to Image Translation Network, which was published in NeurIPS 2017. UNIT uses a

VAE-GAN framework to learn latent spaces and domain translation simultaneously. So, in addition to the cycle consistency that we saw with CycleGANs, UNIT-GAN introduces cycle consistency, even at the latent space level.

Let us see this with the loss functions. But let us first try to understand the entire setup before we go there. The UNIT architecture is based on this illustration. Given two inputs x_1 and x_2, from the two domains, you have corresponding encoders E_1 and E_2 for each domain, which gives a latent vector in the same space. The dimension of the latent vector for both domains is the same.

Given a latent vector from that shared space, you have two generators, G_1 corresponding to Domain 1 and G_2 corresponding to Domain 2. This gives us four possibilities of generations, $x_{1\to 1}$, where the input is from Domain 1 and the output generated is also from Domain 1. $x_{2\to 1}$, where the input is from Domain 2, but the output is from Domain 1. Similarly, $x_{1\to 2}$and $x_{2\to 2}$. All these images are then passed to a discriminator corresponding to each domain to say whether the generation is true or false.

(Refer Slide Time: 25:56)



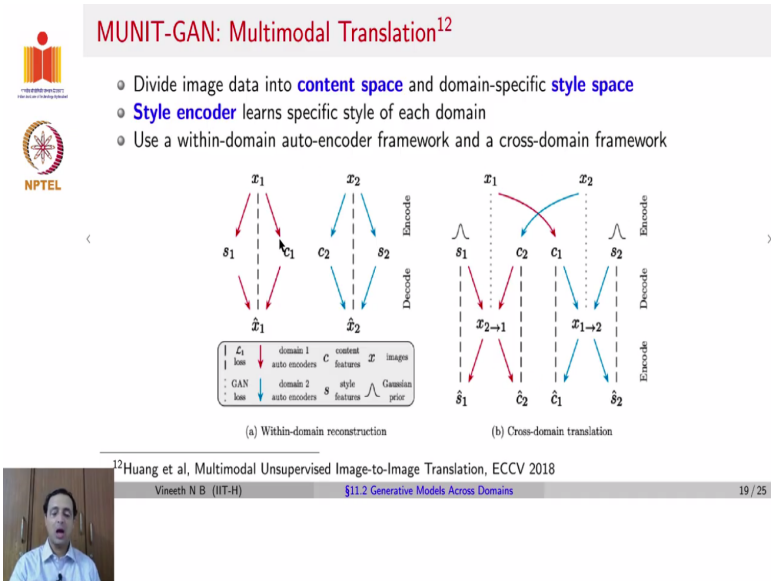
The loss functions are given by both the VAE loss and the GAN loss since UNIT is a VAE-GAN framework. A KL-divergence gives the VAE loss for domain $x_1 \to x_2$, between the approximate

posterior, $q_1(z_1 | x_1 || p_\eta(z))$ and the log-likelihood of generating x_1 given z_1 using the generator G_1, $log\ \ p_{G_1}(x_1 | z_1)$. We take the negative log-likelihood, which needs to be minimized.

Similarly, you have the GAN loss, which corresponds to x_1 coming from p_{x_1}, $log\ D_1(x_1)$, which is maximized by the discriminator D_1 and $log(1 - D_1(G_1(z_2)))$, which is a sample drawn from the second domain, but the generation is of the first domain is minimized by $G_1$and maximized by D_1. You would have a similar loss for $x_2 \rightarrow x_1$ for VAEs and GANs to complete this picture.

In addition, you also have a Cyclic loss for $x_1 \rightarrow x_2$, given by a KL divergence between the approximate posterior q_1 of z_1 given x_1 and the prior. Similarly, q_2 of z_2 given x_1 that translates from 1 to 2, and along with that the prior. Also, the negative log-likelihood of G_1 generating x_1 given z_2. This would be the Cyclic loss from x_1 to x_2, and one would again have the x_2 to x_1 defined similarly. All of these can be carefully understood as extensions of GANs and VAEs. Keeping in mind that one needs to ensure generation across domains.

(Refer Slide Time: 28:49)



An extension of UNIT-GAN is the MUNIT-GAN. In MUNIT-GAN, the image data, the latent space of the image data, is divided into a content space and a domain-specific style space. The

idea here is that each domain has a certain style. We talked about this with CycleGANs, where you could have each artist's style be a different domain. The style encoder then tries to transfer the content in a different domain to the style in that particular domain.

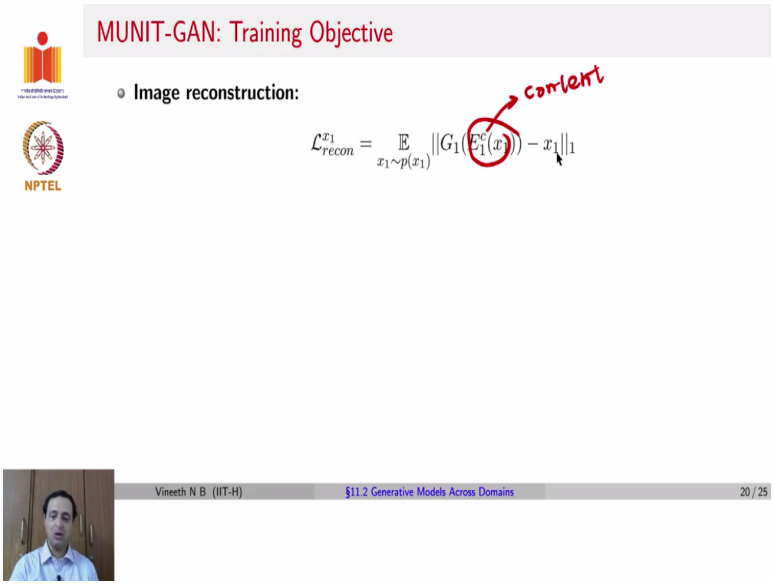
This is implemented using a within-domain autoencoder framework, as well as a cross-domain framework. Let us see this in some more detail here. So, you have $x_1$and x_2, which are images in two different domains. The latent variable is now divided into two parts, s_1 and c_1, which would have been $z_1$in UNIT. s_1 corresponds to the Style Latent's. So, you could now imagine a latent vector z divided into two parts.

Not necessarily equal, so the latent vector could be, say, 100 dimensional, 40 dimensions could correspond to the style, and 60 dimensions could correspond to content, just as an example. So, s_1 corresponds to the style of the first domain, c_1 is a set of latent dimensions corresponding to the content of that particular domain. Similarly, c_2 and s_2 for the second domain. Otherwise, you could now consider these two as individual variational autoencoders for Domain 1 and Domain 2.

This is the Within-domain Autoencoder framework. In the Cross-Domain framework, one gives x_1 as input, which now leads to c_1 , which is the content latent of x_1. Similarly, the content latent of x_2 is c_2. c_2 is now combined with s_1 that forms a new latent. The decoder is applied to this new concatenated latent to get an $x_{2\to1}$, which is a translation from the second domain to the first domain.

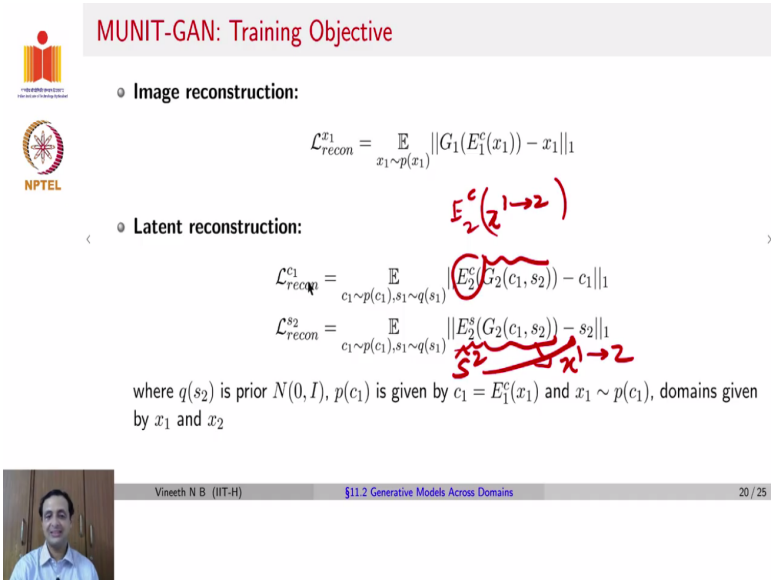
Similarly, the content variable of Domain 1 with the style variable of Domain 2 gives us $x_{1\to2}$, which is a translation from domain 1 to 2. Now, to complete the cycle in the latent variable space, one could again take these x's and derive their latent variables through the encoder of a variational autoencoder, which would give us $\hat{s}_1$, $\hat{c}_2$, $\hat{c}_1$ and $\hat{s}_2$. Let us keep this structure and entire illustration in mind when we look at the loss functions.

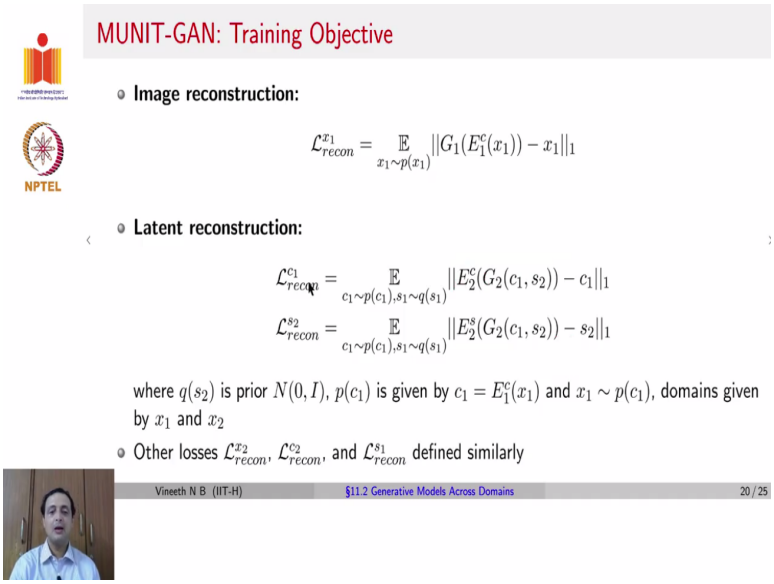
(Refer Slide Time: 32:13)



So, you have an Image Reconstruction Loss, given by L_1loss, $|| G_1(E^c{}_1(x_1)) - x_1||_1$. Remember, $E^c{}_1(x_1)$ gets us the content latent of the encoder of x_1. Remember, this is the latent content of the latent variables of the encoder of the first domain. The generator of Domain 1 is applied to that, which gives a reconstruction in that same domain, which we would like to be close to x_1. This is a simple Image Reconstruction Loss.

(Refer Slide Time: 32:57)



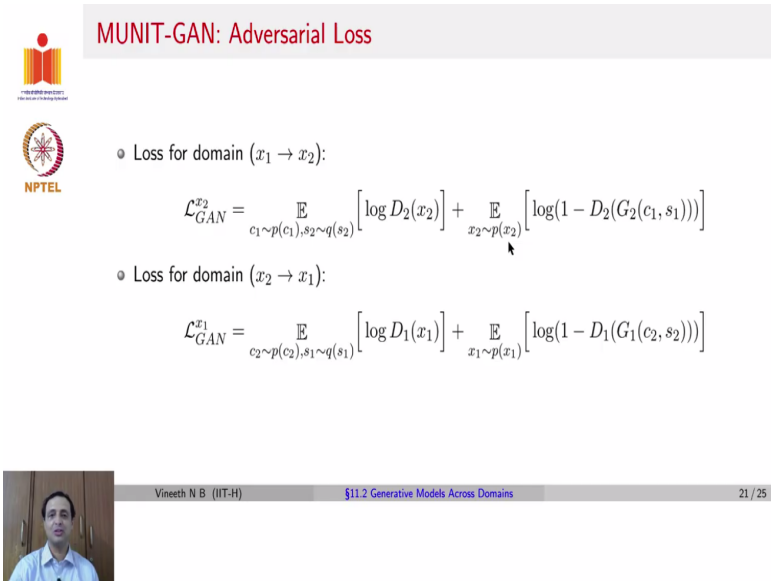


We also have the Latent Reconstruction Loss, where we would like to ensure the latent's reconstruct themselves. This is given by L_1loss, $||\,E_2{}^c(G_2(c_1, s_2)) - c_1||_1$. Let us try to understand this. Given the content from Domain 1 and the style from Domain 2, when one applies G2, we get an output; what would that output be? That output would be $x_{1\to 2}$. Now, given this x, when the encoder of the second domain is applied, which is what $E_2{}^c$ corresponds to.

One would want the content variable to be close to the content of the first domain, which is given by c_1 reconstruction loss. One would have a similar reconstruction loss for s_2 . Let us see that too. Given c_1 and s_2, once again, one gets a construction of $x_{1\to 2}$. When the second encoder is applied to this, one expects to retrieve s_2, given by $\hat{s}_2$ if you recall in the earlier diagram.

One would want this to be close to the original s_2 , and the L1 loss tries to capture this Latent Reconstruction. You would similarly have terms for the corresponding s_1 reconstruction and c_2 reconstruction.

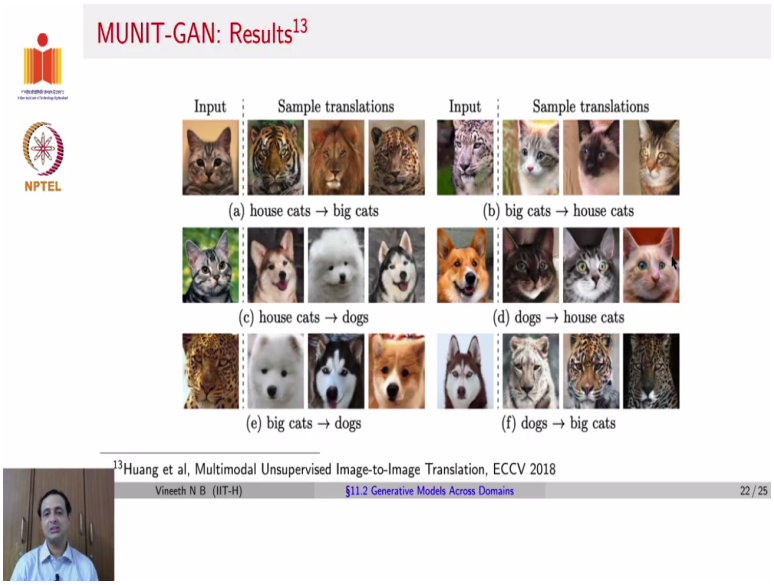
(Refer Slide Time: 34:46)



Finally, Adversarial Loss also tries to ensure that $D_2(x_2)$ is maximized by the discriminator D_2. Similarly, $(1 - D_2(G_2(c_1, s_1)))$ is maximized by the discriminator and minimized by the generator in the second domain. This is the standard GAN loss for the second domain. One would similarly define a GAN loss for the first domain, complementing the same loss function.

In case some of this is hard to follow, I would recommend going through these equations carefully. This is an extension of Vanilla GAN across two domains, and there is no further complexity.

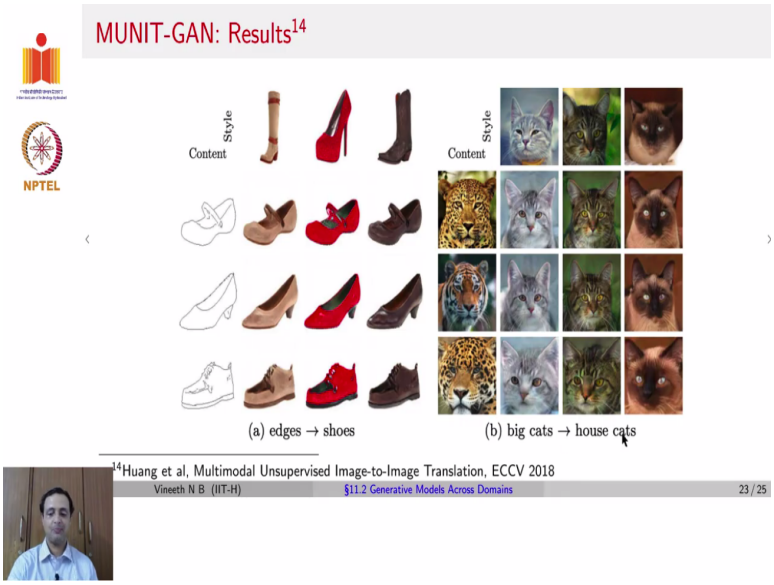
(Refer Slide Time: 35:40)



With these loss functions, MUNIT-GAN shows impressive results of translations from one domain to the other. This time with more diversity and variety by changing the latent values, given an input image from one domain, in this case, cats. If one wants to generate larger jungle cats or big cats, you could now get several translations by playing with the latent vector with the latent style vector of the second domain.

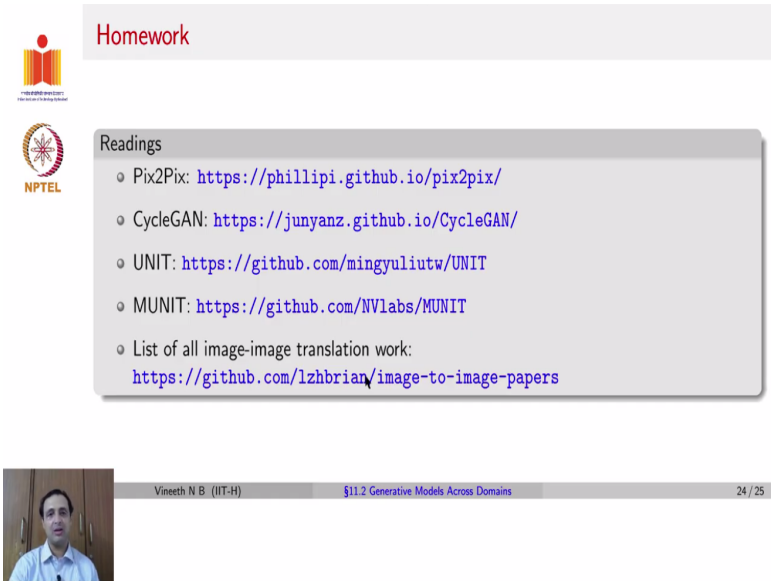
You can see that with different examples, you get several varieties of outputs in the second domain for a given input, which can be obtained by changing the style vector of the second domain. Remember, the style vector of the latent in the second domain can be interpolated to change these kinds of outputs in the generation.

(Refer Slide Time: 36:45)



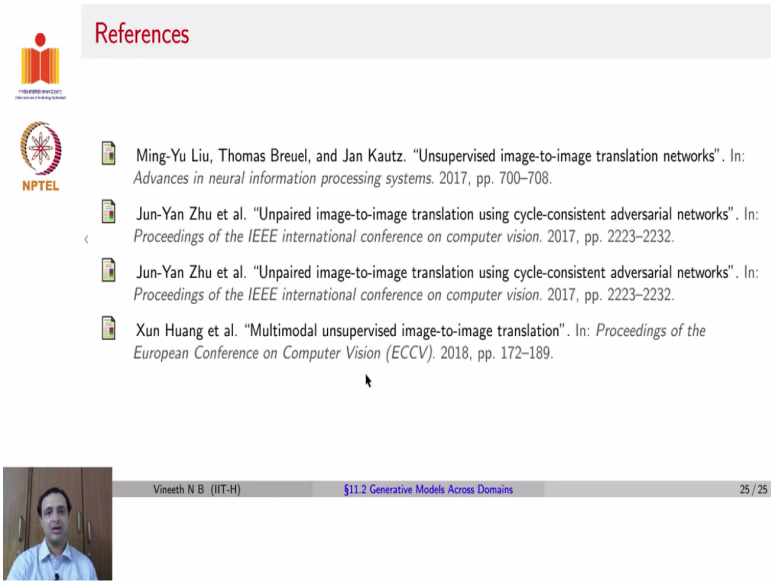
And as wanted when we started, this also allows us to vary style and content gradually across the generations. So, you can see an example that the content comes from these sketches, and the style comes from these images. In each case, the colour and style are of the second domain, while the content comes from the first domain. You can see in subfigures a and b that the first set of images go from edges to shoes and the second set of images from big cats to house cats.

(Refer Slide Time: 37:31)



That concludes this lecture. Each link provides more details of the paper and the code if you would like to know more. The link at the end has a list of all image to image translation work if you would like to understand them more.

(Refer Slide Time: 37:53)



Here are references.