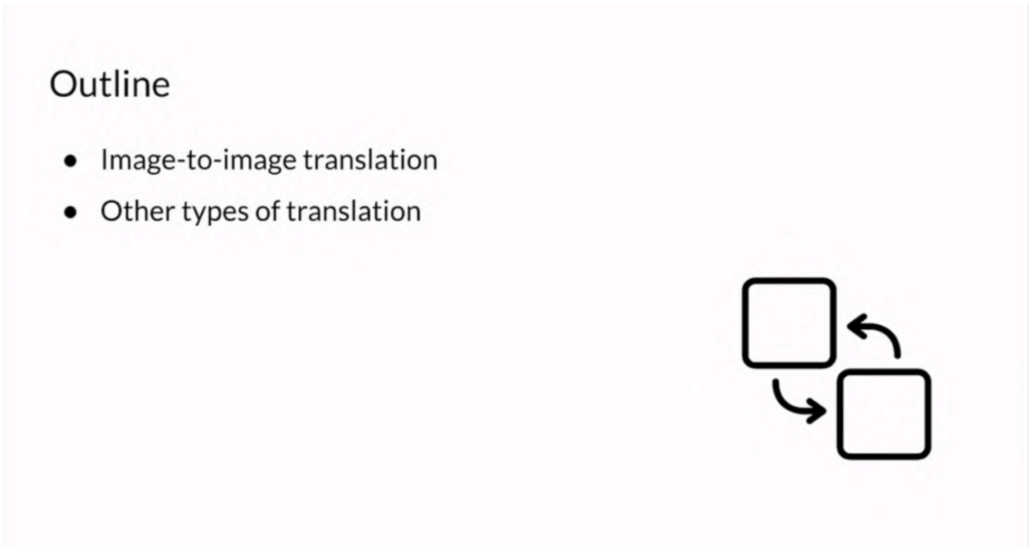
deeplearning.ai

Image-to-Image Translation



Outline

- Image-to-image translation
- Other types of translation

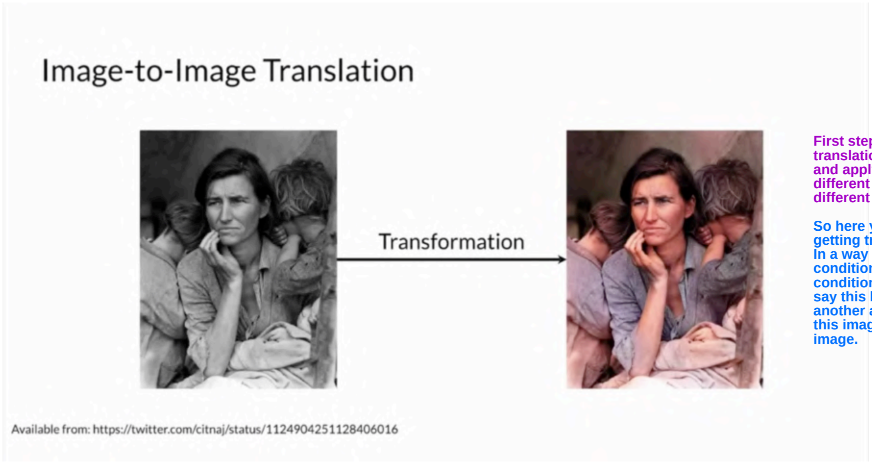


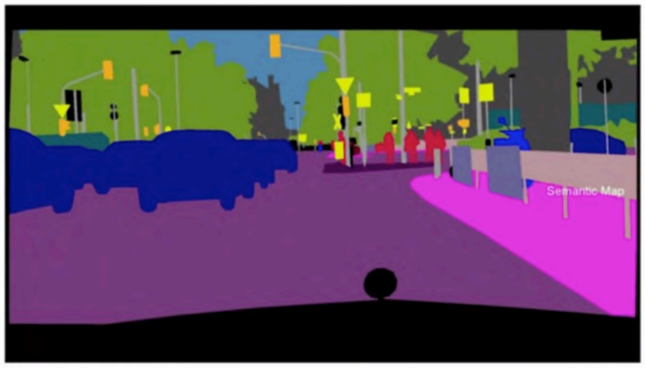
Image-to-Image Translation

Available from: https://twitter.com/citnaj/status/1124904251128406016

First step, what's image-to-image translation? Well, imagine taking an image and applying a transformation to it to get a different image, like translating across different styles of an image.

So here you see this black and white image getting transformed into a colored image. In a way this is actually a type of conditional generation, but it's conditioning on the content of one image, say this black and white image to create another and that's saying, conditioning on this image give me a colored style of that image.

Image-to-Image Translation

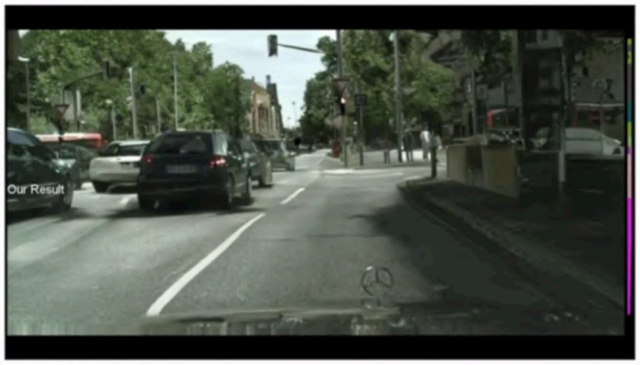


Available from: https://github.com/NVIDIA/pix2pixHD

Another example image-to-image translation task is going from a segmentation map. These are segmentations of this road, of a car or pedestrians, of the sidewalk, of trees. Going from that to a realistic photo that maps onto trees where it's labeled trees, road where it's labeled road, and car where it's labeled car. It's going from one domain to another.

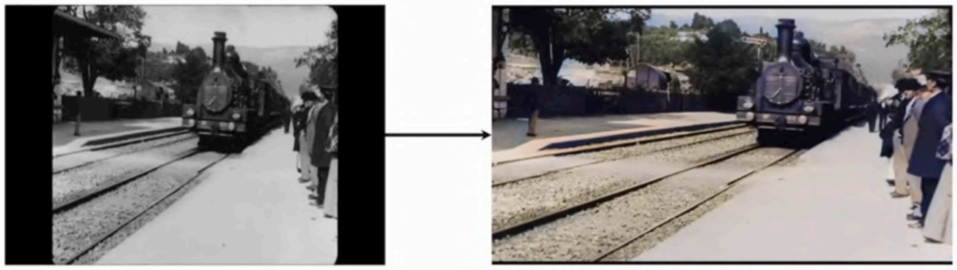
You can imagine that with the semantic segmentation map, you can get realistic views that look very different. It doesn't have to just be corresponding to this one realistic image, this photo here.

Image-to-Image Translation



Available from: https://github.com/NVIDIA/pix2pixHD

Image-to-Image Translation

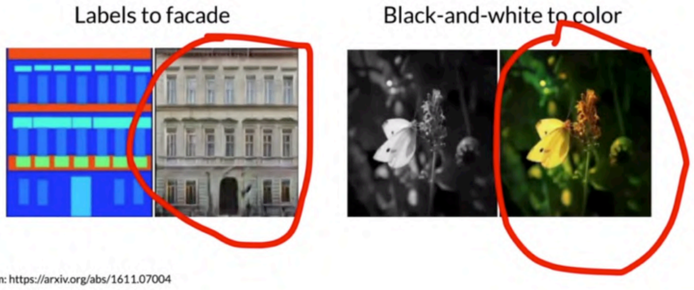


Available from: https://youtu.be/3RYNThid23g

Another cool image-to-image translation task, or perhaps here's video to video, where you have frames of a video go map onto the frames of another video. It is essentially image-to-image translation, but for many images, many frames. You can take this black and white trained film from back in the day and make this old film 4K with some realistic color on it as well.

This is actually not just adding realistic color but going from low resolution to a high resolution, also known as super-resolution. An image-to-image translation is really where GANs have been able to shine because they're able to create these very realistic images in here videos, every single frame looks very realistic still.

Paired Image-to-Image Translation



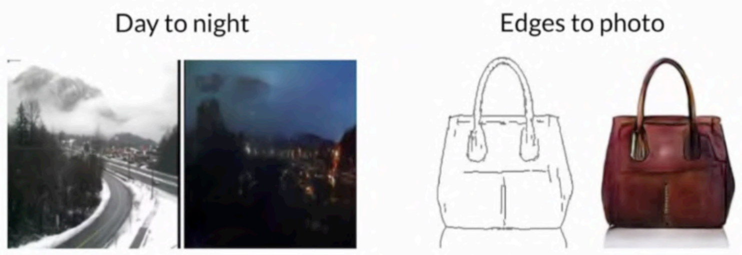
Available from: https://arxiv.org/abs/1611.07004

Now diving deeper into a type of image-to-image translation that you'll be exploring first.

The first is paired image-to-image translation. What that means is that you are pairing the input and then an output. This means that for your training dataset, every single input example that you might have, you have a corresponding output image or target image that contains the contents of that input image with a different style. So it maps one-on-one. Basically what you do is you condition on the input to get the output image. As you can probably tell the output pairs which are the second images in each of these, the facade in this building and the colored image and this butterfly, they're not the only type of image that could be generated from that black and white photo or that segmentation map that you see. You can imagine a lot of different buildings that could be generated from those labels and a lot of different butterfly types that can be generated from that black and white image.

So the paired output image is not necessarily the ground truth, but a ground truth, a possibility.

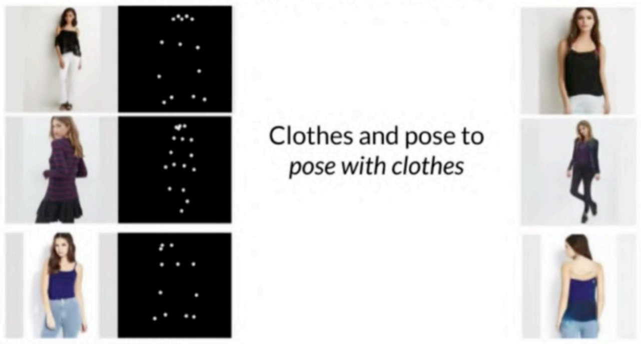
Paired Image-to-Image Translation



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

Here are other examples going from day to a night time photo, also edges to a photo and you can actually get edges to create this paired training data by just using an edge detection algorithm, which is a typical computer vision algorithm. That's pretty cool because you can easily create this paired dataset. Then you again can actually go from any kind of edge that you do draw to a realistic photo after it's been trained.

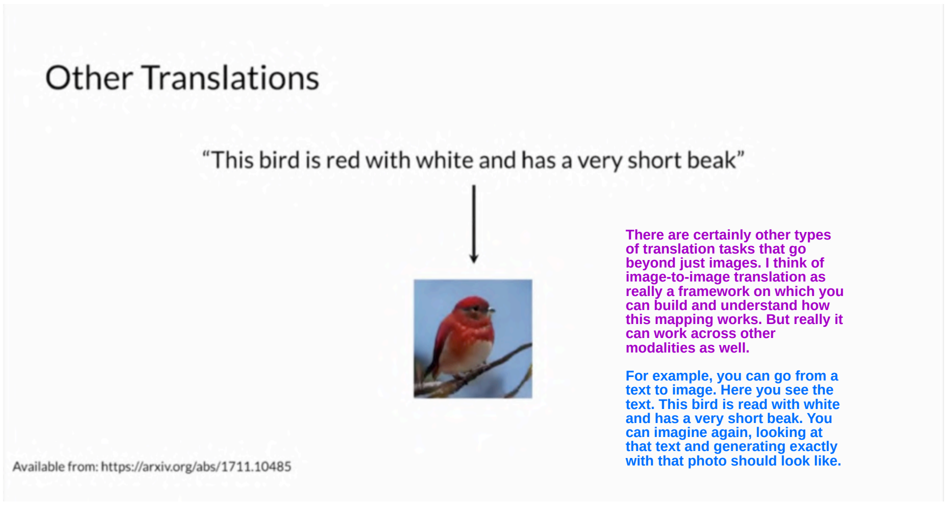
Paired Image-to-Image Translation



Available from: https://arxiv.org/abs/1705.09368

Then image-to-image translation can actually get even more complex.

Instead of conditioning on a single input image, you can take as input models wearing different clothes and also a point-wise map of where they should be standing, their pose. Those dots represent different poses. Then from these two pieces of information, you want your game to generate that person in a different pose. That's what you see on the right here.



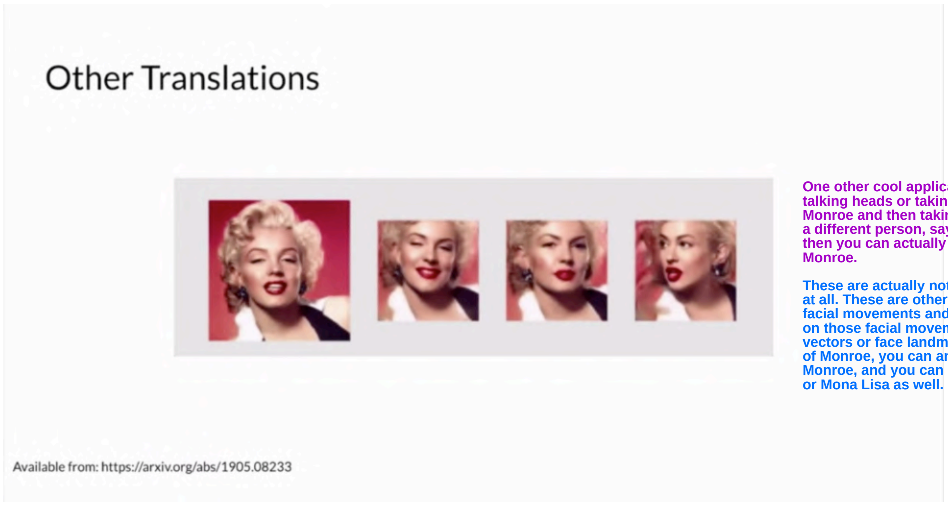
Other Translations

"This bird is red with white and has a very short beak"

Available from: https://arxiv.org/abs/1711.10485

There are certainly other types of translation tasks that go beyond just images. I think of image-to-image translation as really a framework on which you can build and understand how this mapping works. But really it can work across other modalities as well.

For example, you can go from a text to image. Here you see the text. This bird is read with white and has a very short beak. You can imagine again, looking at that text and generating exactly with that photo should look like.

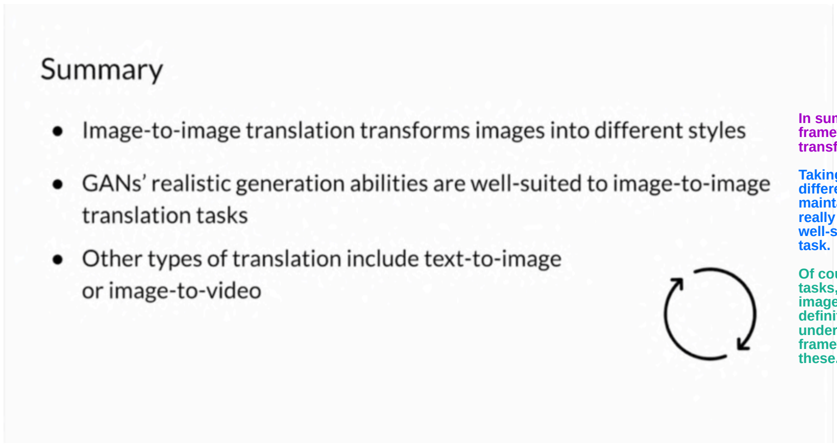


Other Translations

Available from: https://arxiv.org/abs/1905.08233

One other cool application is neural talking heads or taking an image, say, of Monroe and then taking face vectors from a different person, say me or you, and then you can actually speak through Monroe.

These are actually not Monroe speaking at all. These are other people making facial movements and then conditioned on those facial movements, those face vectors or face landmarks plus this image of Monroe, you can animate this image of Monroe, and you can do this for Einstein or Mona Lisa as well.



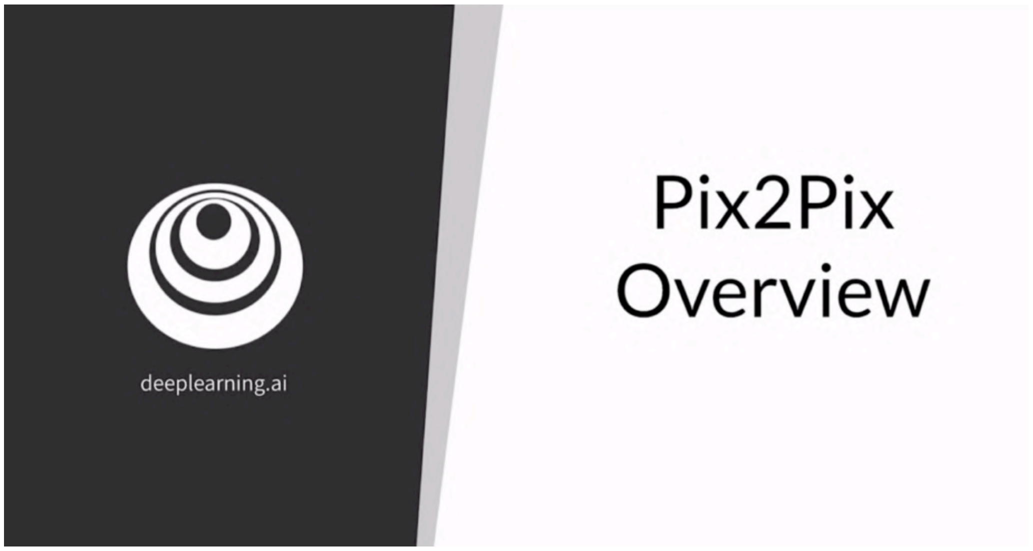
Summary

- Image-to-image translation transforms images into different styles
- GANs' realistic generation abilities are well-suited to image-to-image translation tasks
- Other types of translation include text-to-image or image-to-video

In summary, image-to-image translation is a framework of conditional generation that transforms images into different styles.

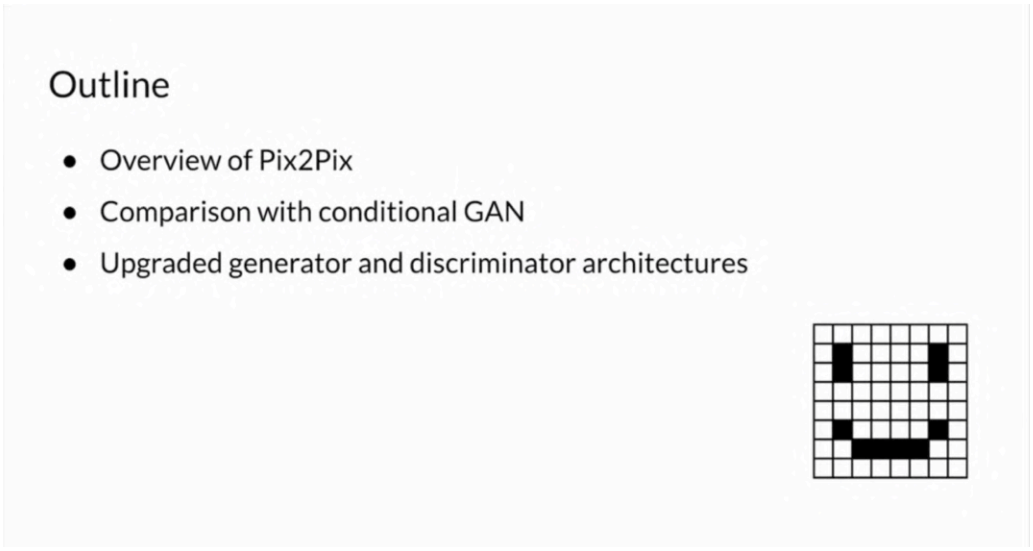
Taking in an image and transforming it to get a different image of a different style, but maintaining that content. Because GANs are really good at realistic generation, they are really well-suited for this image-to-image translation task.

Of course, there are other types of translation tasks, such as text to image or image to video or image plus faced mark to video. These are definitely things you can explore and understanding the image to image translation framework will set you up to understanding all of these.



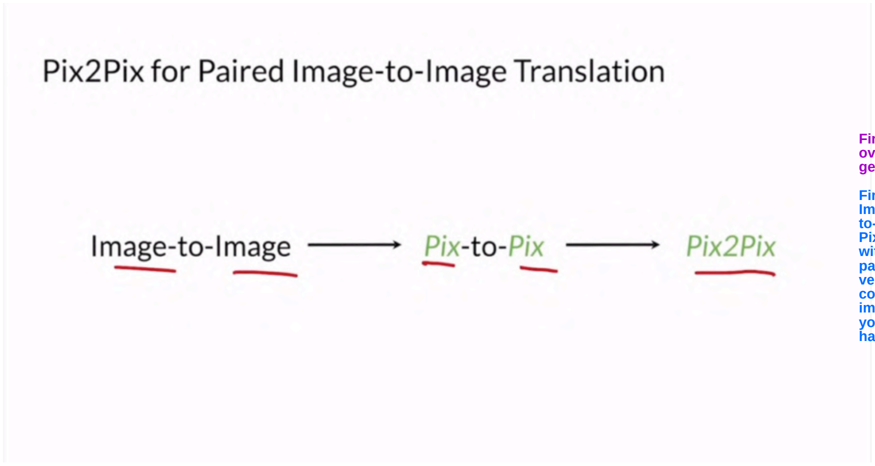
Pix2Pix Overview

deeplearning.ai



Outline

- Overview of Pix2Pix
- Comparison with conditional GAN
- Upgraded generator and discriminator architectures

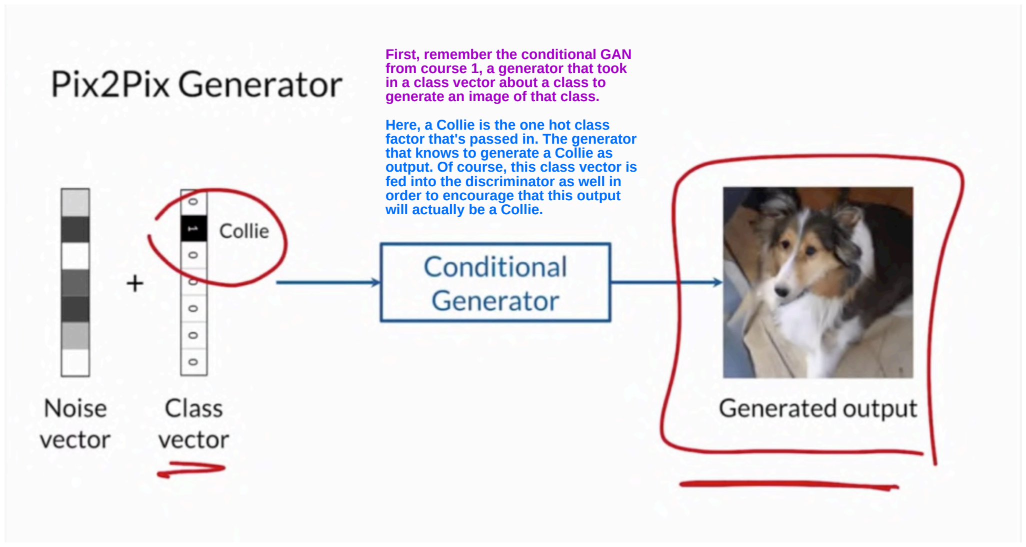


Pix2Pix for Paired Image-to-Image Translation

Image-to-Image → Pix-to-Pix → Pix2Pix

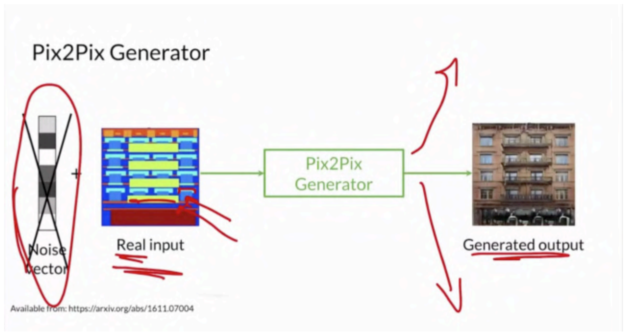
First we'll begin by getting an overview of Pix2Pix, and its unique generator and discriminator design.

First, how it was named. Image-to-Image translation goes from image-to-image, and then you see Pix-to-Pix, pix as an image and then Pix2Pix with the number two. This was a paper from UC Berkeley and it was a very successful use of a type of conditional GAN to perform paired image-to-image translation, where you condition on the input image and have a direct output pair.



First, remember the conditional GAN from course 1, a generator that took in a class vector about a class to generate an image of that class.

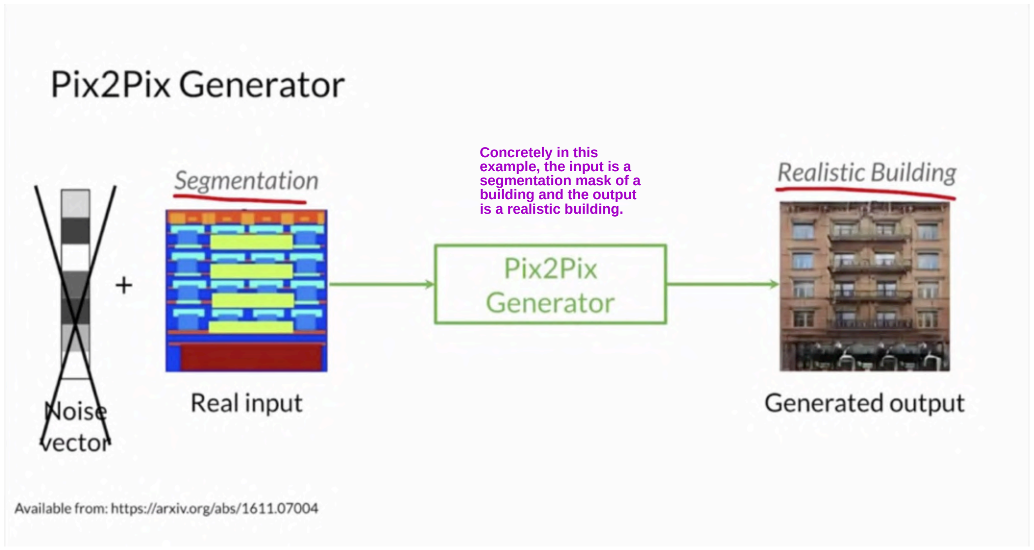
Here, a Collie is the one hot class factor that's passed in. The generator that knows to generate a Collie as output. Of course, this class vector is fed into the discriminator as well in order to encourage that this output will actually be a Collie.



Pix2Pix is similar, but instead of a class vector, you actually pass in an entire image as the input. This image would be say, a segmentation map of a building. Note that this is a real input. Segmenting some type of building where these medium blue squares are windows versus the things on top of the windows, balconies are green and the general facade is this deep blue. Different colors correspond to different classes within that image.

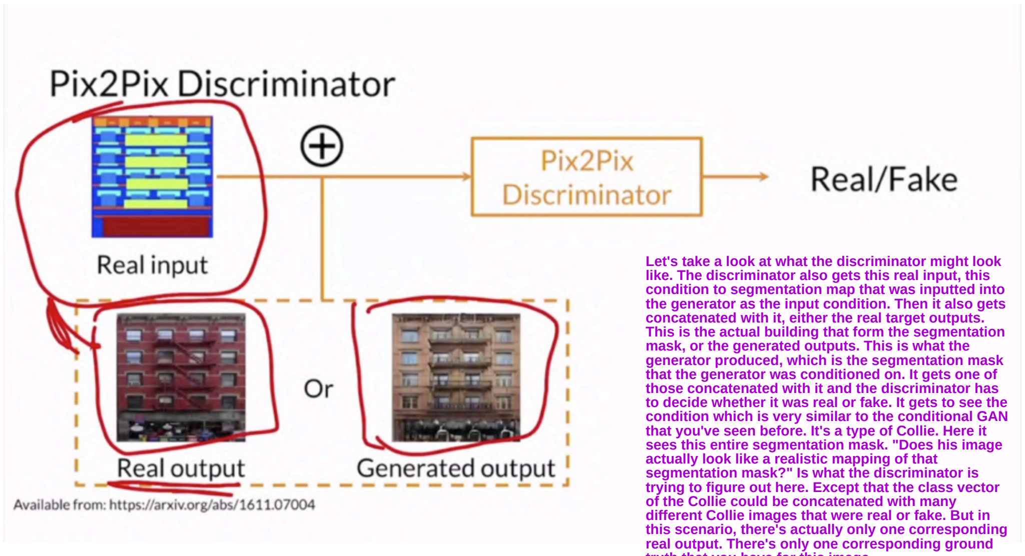
Something you might see is that the noise vector here is being crossed out, and it's actually because the authors found that the noise vector didn't make a huge difference to the generator's output. Typically, what the noise vector was used for was so that the generator could generate all these different outputs. But what they found was that the noise vector didn't actually make that much of a difference in terms of what the generated output did look like. This is likely due to the fact that there's a paired output image that this generator is trying to get at, and you'll see this very shortly.

Instead of noise, they actually found that they could add some stochasticity into the network using dropout. Dropout, just randomly plugs out nodes in certain layers in your neural network as it's training, so different nodes can learn different things. As a result, this adds some randomness to your outputs as it's training, though it's not as drastic as you saw with a variety of different dogs when you input a different noise vectors as that input before.



Concretely in this example, the input is a segmentation mask of a building and the output is a realistic building.

Pix2Pix Discriminator

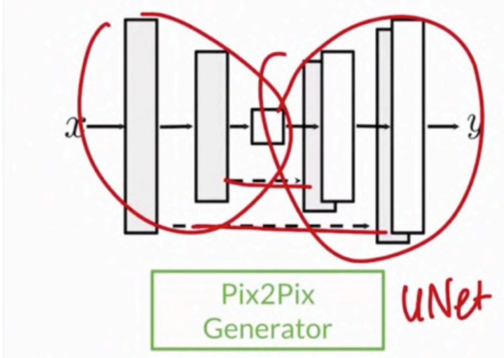


Available from: https://arxiv.org/abs/1611.07004

Let's take a look at what the discriminator might look like. The discriminator also gets this real input, this condition to segmentation map that was inputted into the generator as the input condition. Then it also gets concatenated with it, either the real target outputs. This is the actual building that form the segmentation mask, or the generated outputs. This is what the generator produced, which is the segmentation mask that the generator was conditioned on. It gets one of those concatenated with it and the discriminator has to decide whether it was real or fake. It gets to see the condition which is very similar to the conditional GAN that you've seen before. It's a type of Collie. Here it sees this entire segmentation mask. "Does his image actually look like a realistic mapping of that segmentation mask?" Is what the discriminator is trying to figure out here. Except that the class vector of the Collie could be concatenated with many different Collie images that were real or fake. But in this scenario, there's actually only one corresponding real output. There's only one corresponding ground truth that you have for this image.

That's why under this paradigm with a paired output image, the authors found that adding random noise doesn't actually change with the model learns.

Pix2Pix Upgrades

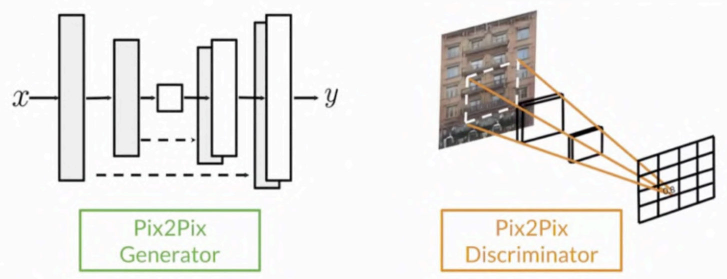


Based on: https://arxiv.org/abs/1611.07004

Something pretty exciting happens here. Both the generator and discriminator will get an upgrade, and the generator will first become what's called a U-Net. A U-Net is typically used for segmentation. It's the type of encoders, so you see it encodes an image and then a decoder after that, followed by some skip connections in between and I'll go into this a bit more in a following video.

Pix2Pix Upgrades

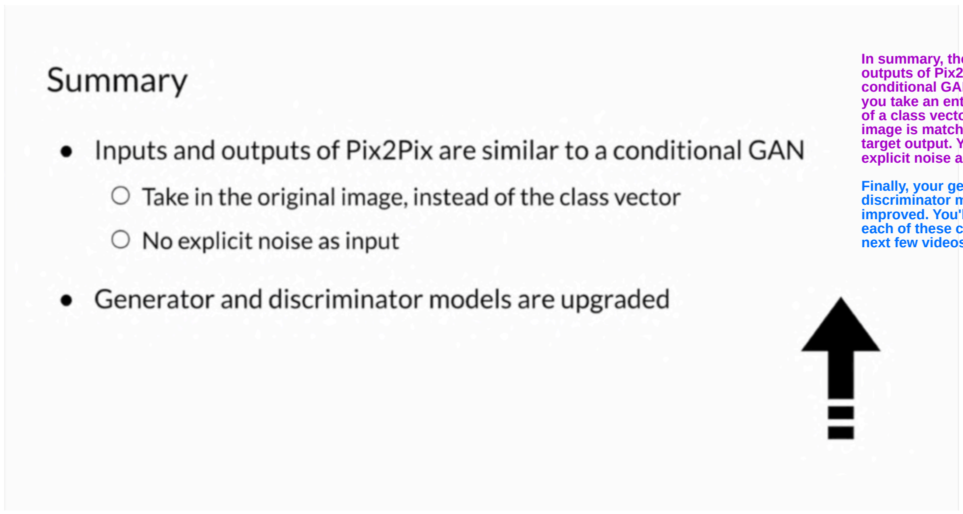
Goal is still to produce realistic outputs!



(Left) Based on: https://arxiv.org/abs/1611.07004
(Right) Based on: https://arxiv.org/abs/1803.07422

Then the discriminator will become a patch can. What that means is that it gives more feedback on different parts of the image. Instead of saying something is real or fake for an entire image, is this real or fake for different patches of that image. A lot of real fake determinations in a matrix as an output.

This means that the discriminator will give a lot more feedback back to the generator. The goal, of course, is still to produce realistic outputs from this GAN training scheme.

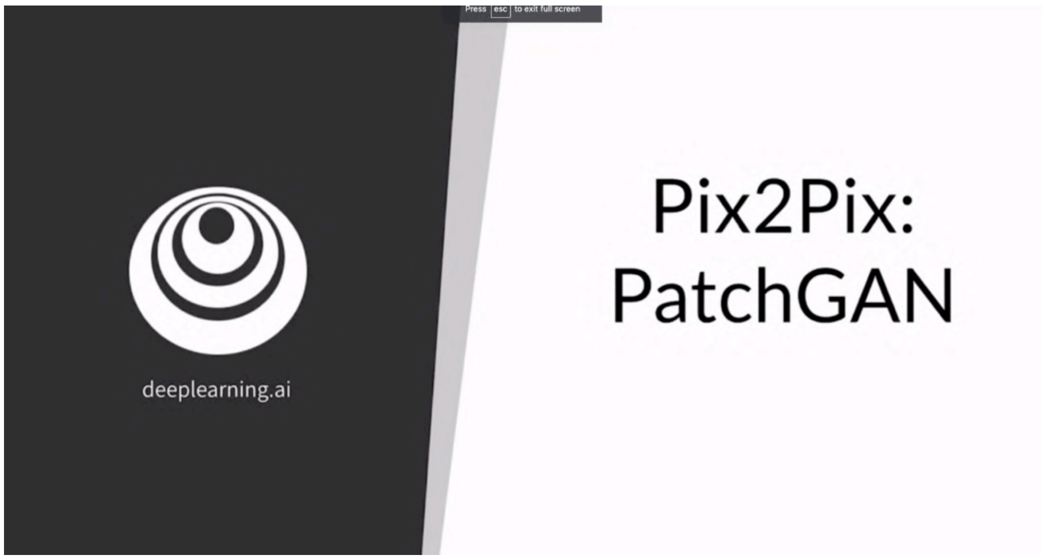


Summary

- Inputs and outputs of Pix2Pix are similar to a conditional GAN
    - Take in the original image, instead of the class vector
    - No explicit noise as input
- Generator and discriminator models are upgraded

In summary, the inputs and outputs of Pix2Pix are similar to a conditional GAN, except for now you take an entire image instead of a class vector, and that paired image is matched one-to-one to a target output. You also have no explicit noise as your input.

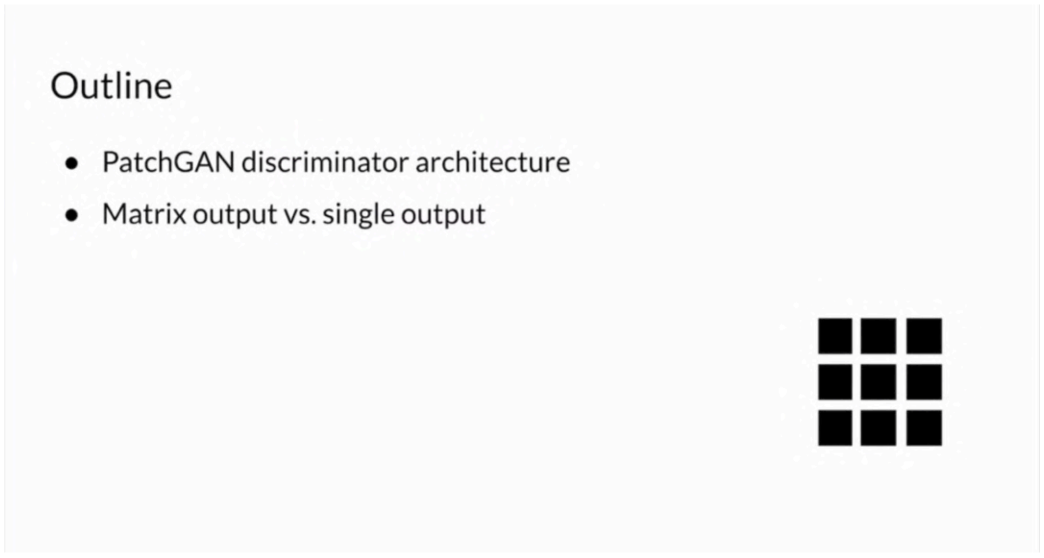
Finally, your generator and discriminator models are heavily improved. You'll get to explore each of these components in the next few videos.



Press esc to exit full screen

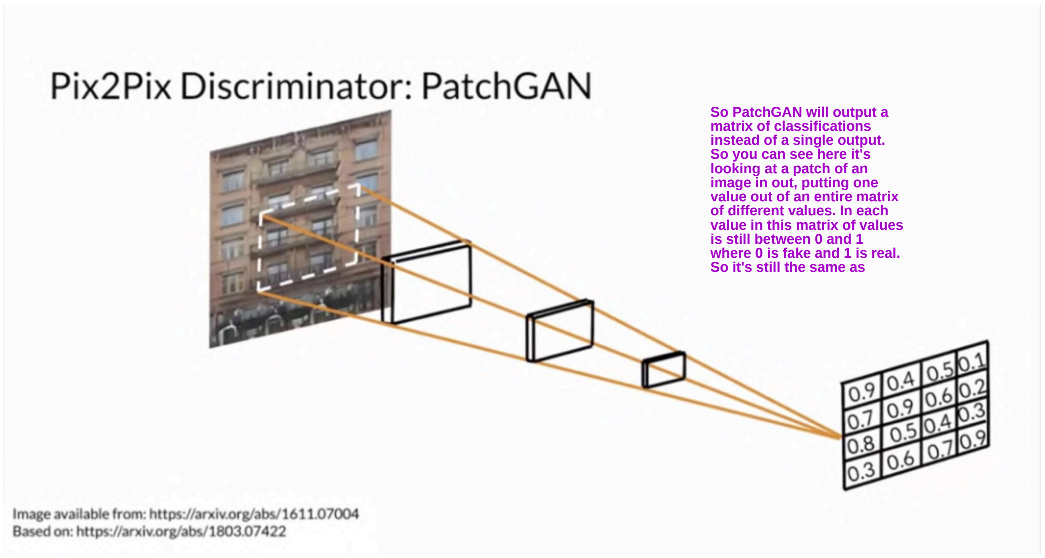
deeplearning.ai

Pix2Pix: PatchGAN



Outline

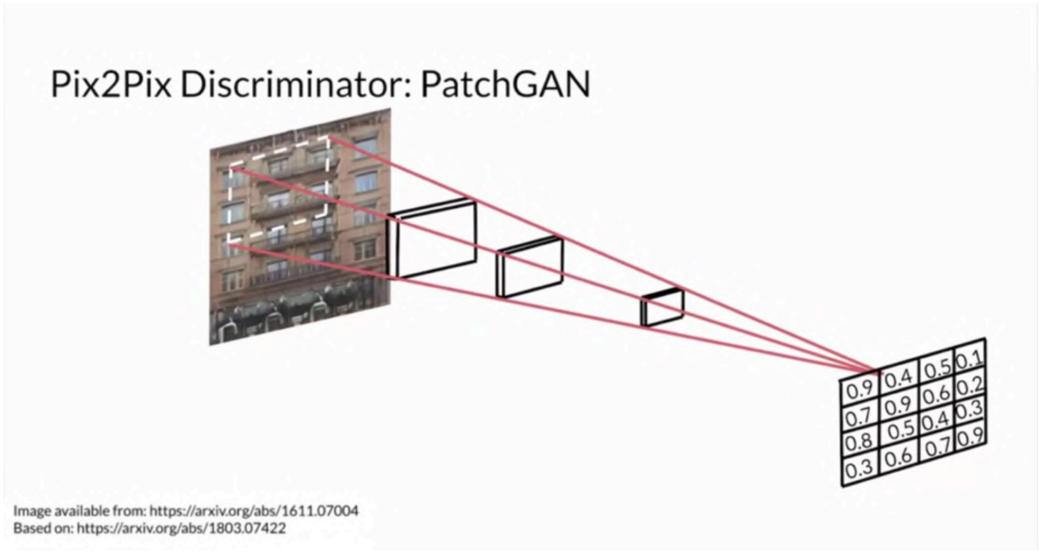
- PatchGAN discriminator architecture
- Matrix output vs. single output



Pix2Pix Discriminator: PatchGAN

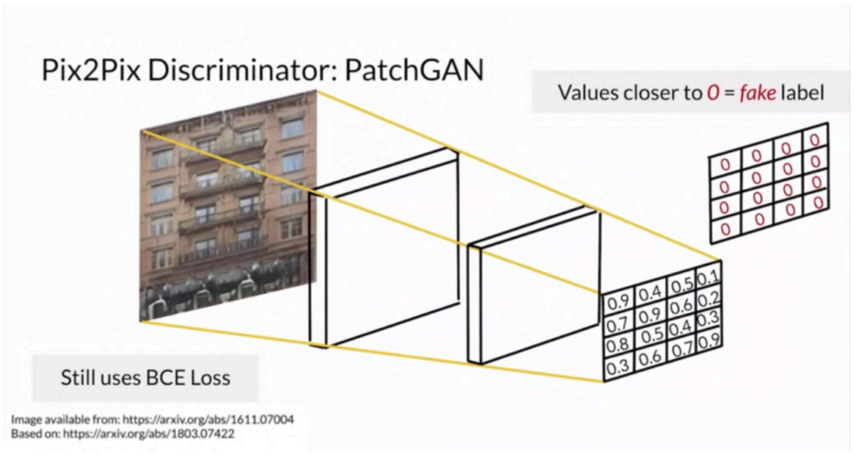
Image available from: https://arxiv.org/abs/1611.07004
Based on: https://arxiv.org/abs/1803.07422

So PatchGAN will output a matrix of classifications instead of a single output. So you can see here it's looking at a patch of an image in out, putting one value out of an entire matrix of different values. In each value in this matrix of values is still between 0 and 1 where 0 is fake and 1 is real. So it's still the same as



Pix2Pix Discriminator: PatchGAN

Image available from: https://arxiv.org/abs/1611.07004
Based on: https://arxiv.org/abs/1803.07422



Pix2Pix Discriminator: PatchGAN

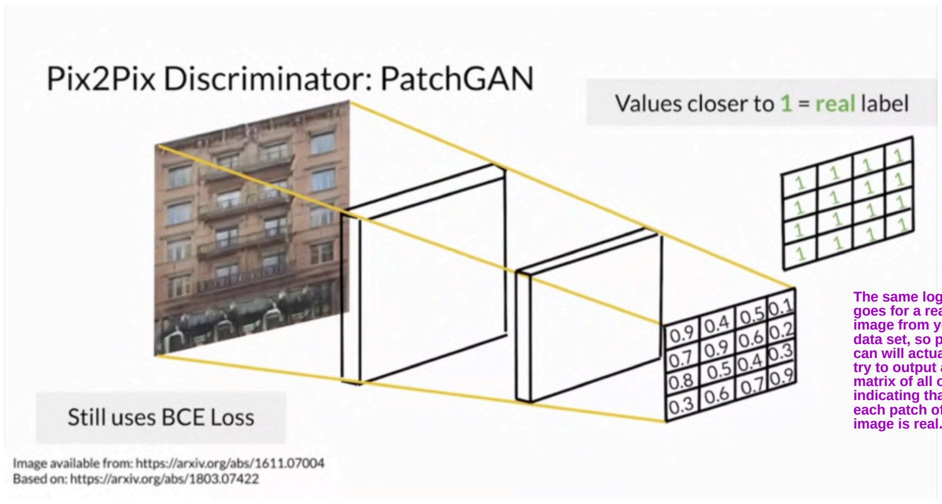
Values closer to *0* = *fake* label

Still uses BCE Loss

Image available from: https://arxiv.org/abs/1611.07004
Based on: https://arxiv.org/abs/1803.07422

So by sliding its field of view across all the patches in the input image, the PatchGAN will then give feedback on each region or patch of the image. And because it outputs the probability of each patch being real, it can be trained with BCE loss still.

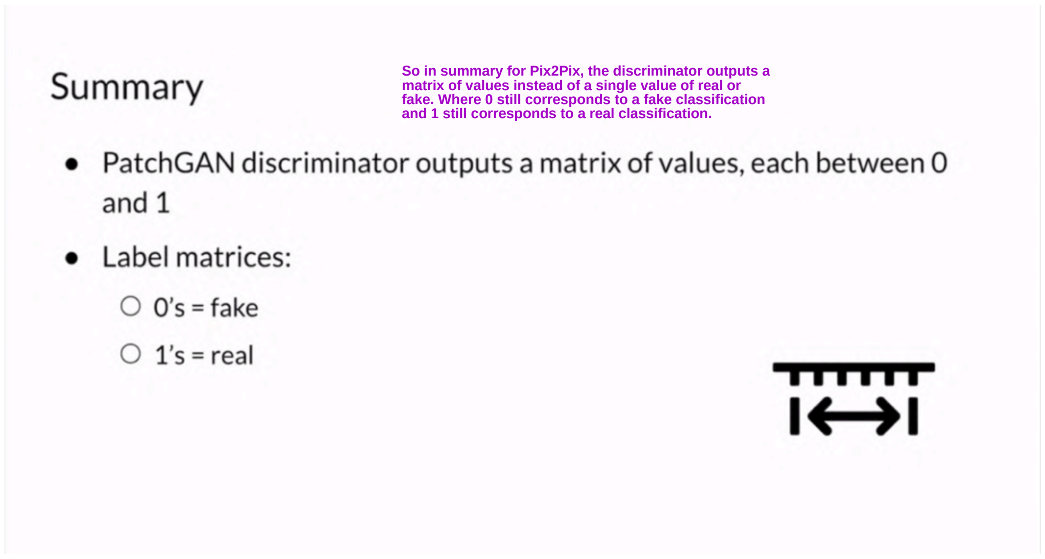
So for a fake image from the generator, what this means is that the PatchGAN should try to output a matrix of all zeros. So the label for it, the corresponding label for it here is this matrix of all zeros. Meaning yes every single patch of this image is fake.



Pix2Pix Discriminator: PatchGAN

Image available from: https://arxiv.org/abs/1611.07004
Based on: https://arxiv.org/abs/1803.07422

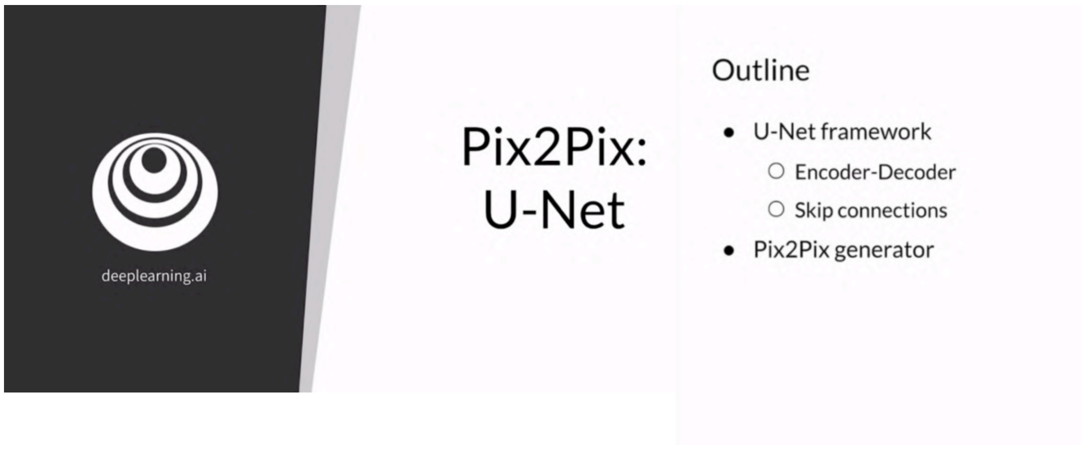
The same logic goes for a real image from your data set, so patch can will actually try to output a matrix of all ones indicating that each patch of the image is real.



Summary

So in summary for Pix2Pix, the discriminator outputs a matrix of values instead of a single value of real or fake. Where 0 still corresponds to a fake classification and 1 still corresponds to a real classification.

- PatchGAN discriminator outputs a matrix of values, each between 0 and 1
- Label matrices:
  - 0's = fake
  - 1's = real



deeplearning.ai

Pix2Pix: U-Net

Outline

- U-Net framework
  - Encoder-Decoder
  - Skip connections
- Pix2Pix generator

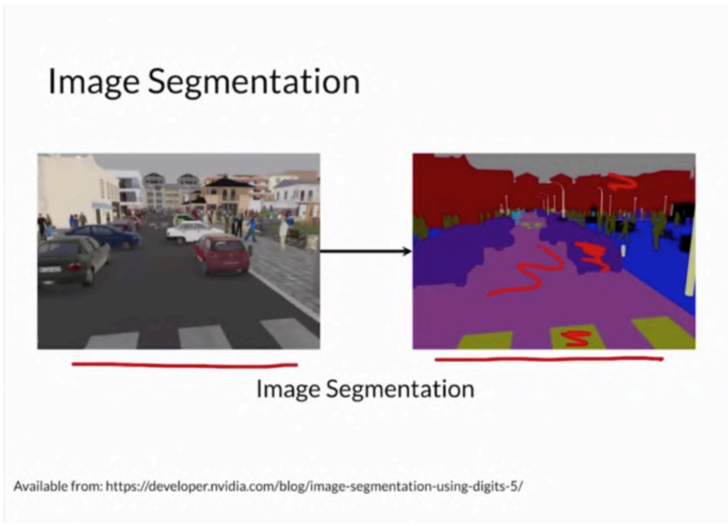


Image Segmentation

Image Segmentation

Available from: https://developer.nvidia.com/blog/image-segmentation-using-digits-5/

So U-Net in general has been a very successful computer vision model for image segmentation. So what segmentation is is taking a real image and getting a segmentation mask or labels on every single pixel that image in terms of what object it is. So you're labeling cards, you're labeling crosswalks, you're labeling trees, you're labeling the road, you're labeling pedestrians. Hopefully this is for self-driving car application.

The segmentation task is very much an image-to-image translation task, but there is a correct answer in terms of what each pixel is and which class each pixel belongs to. So a pixel on a car is definitely going to be a car, and you can't segment it any other way. But when it comes to generating things or what Pix2Pix is really good at, it's something without a correct answer.

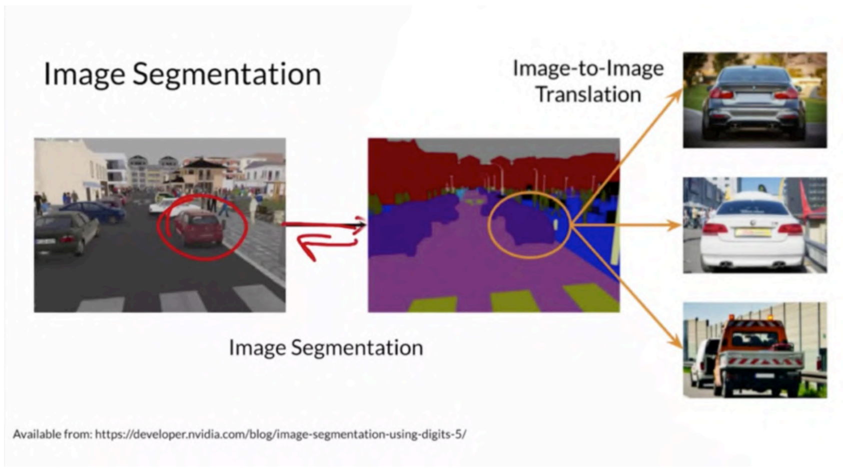
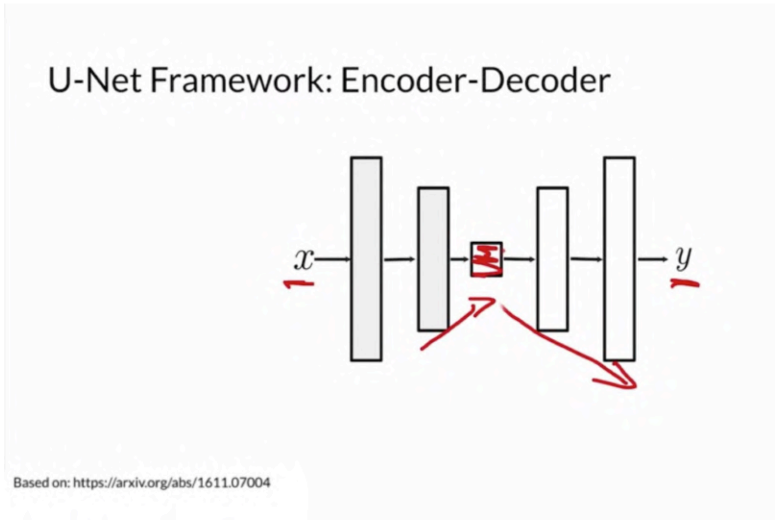


Image Segmentation

Image Segmentation

Available from: https://developer.nvidia.com/blog/image-segmentation-using-digits-5/

So this image of this car here, there's no correct answer in terms of what that car could be. I mean, it could be this car back here in that image, but since it's just a segmentation mask, realistically it could be a different car, any of these other cars. All of these cars are technically correct if you were to look at it as a human, right? So this is also an image-to-image translation task, and now what's just important to note is that Pix2Pix uses the U-Net architecture for the generator architecture. Because it's good at taking in an input image and mapping it to an output image. And typically it's used for just image segmentation, but Pix2Pix wants to use it for this generation task as well. And what Pix2Pix does is largely be able to go back and forth between these two.



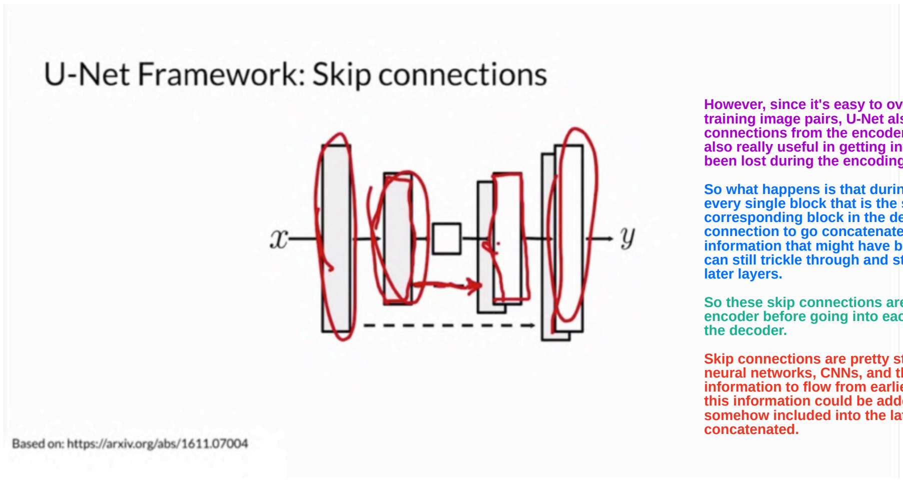
U-Net Framework: Encoder-Decoder

Based on: https://arxiv.org/abs/1611.07004

So remember that the traditional generator architecture takes in a tiny noise vector. The U-Net generator actually takes in an entire image, so that's what this x is here. So this means that the generator has to be a lot beefier with convolutions to handle that image input.

So the architecture of the Pix2Pix generator is this encoder-decoder structure, where first you encode things. And you can imagine an encoder being close to a classification model because you take in an image, and then you output maybe a value here of what class it is, a cat, or a dog. And then you can also think of this middle layer as being a bottleneck where it is able to embed the information in this image. So all that important information in that image is compressed into this little bit of space, just so you can get those high level features, those important features, and to then decode it an output y, another image.

This might remind you of an auto-encoder except for an auto-encoder, you won't want to be as close as possible to x, and here you don't want that. You want y to be a different style conditioned on x.



U-Net Framework: Skip connections

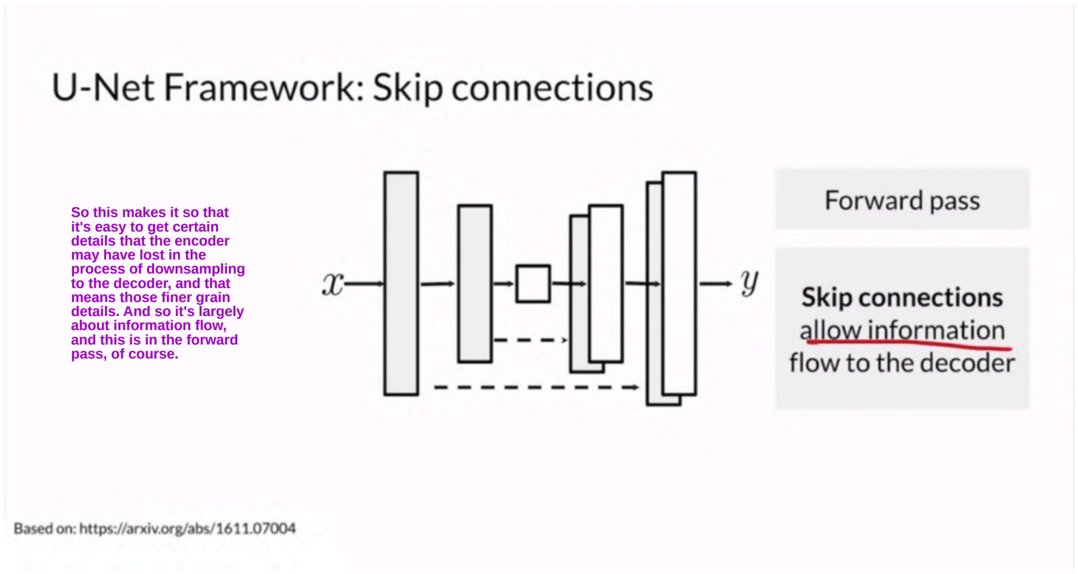
Based on: https://arxiv.org/abs/1611.07004

However, since it's easy to overfit these networks to your training image pairs, U-Net also introduces skip connections from the encoder to decoder, and this is also really useful in getting information that might have been lost during the encoding stage.

So what happens is that during the encoding stage, every single block that is the same resolution as its corresponding block in the decoding stage get this extra connection to go concatenate with that value. Such that information that might have been compressed too much can still trickle through and still get to some of these later layers.

So these skip connections are concatenated from the encoder before going into each convolutional block in the decoder.

Skip connections are pretty standard in convolutional neural networks, CNNs, and they largely just allow information to flow from earlier layers to later layers. In this information could be added or concatenated, but somehow included into the later layers, and here it's concatenated.



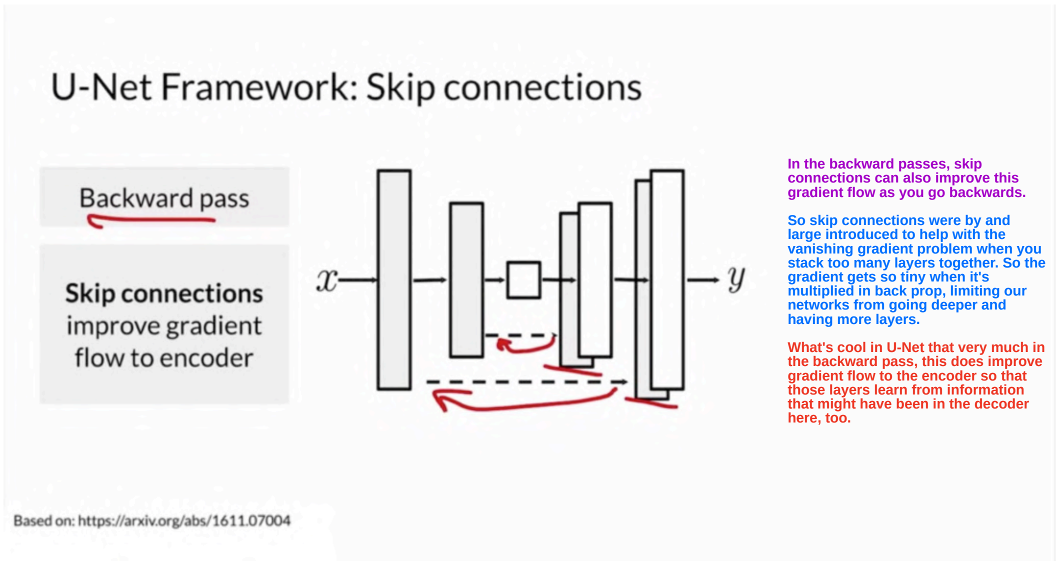
U-Net Framework: Skip connections

Forward pass

Skip connections allow information flow to the decoder

Based on: https://arxiv.org/abs/1611.07004

So this makes it so that it's easy to get certain details that the encoder may have lost in the process of downsampling to the decoder, and that means those finer grain details. And so it's largely about information flow, and this is in the forward pass, of course.



U-Net Framework: Skip connections

Backward pass

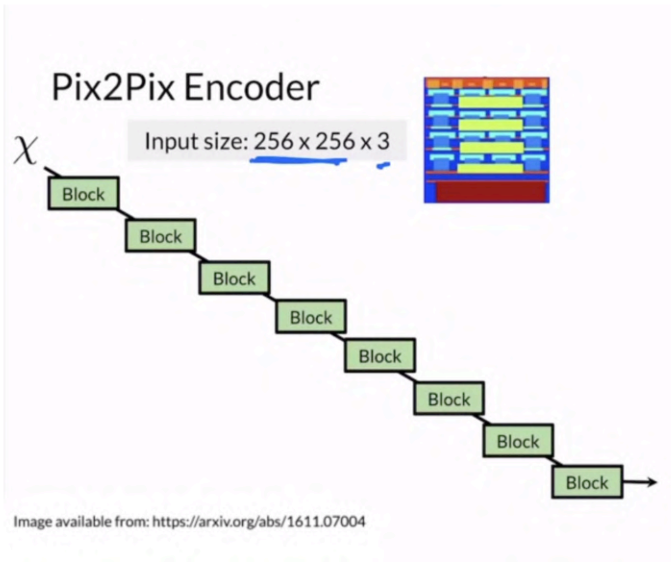
Skip connections improve gradient flow to encoder

Based on: https://arxiv.org/abs/1611.07004

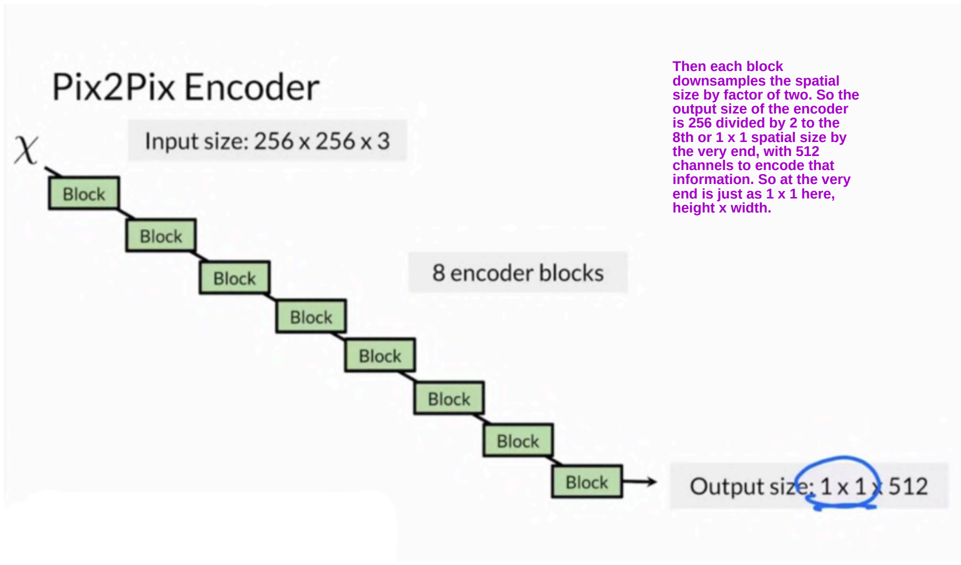
In the backward passes, skip connections can also improve this gradient flow as you go backwards.

So skip connections were by and large introduced to help with the vanishing gradient problem when you stack too many layers together. So the gradient gets so tiny when it's multiplied in back prop, limiting our networks from going deeper and having more layers.

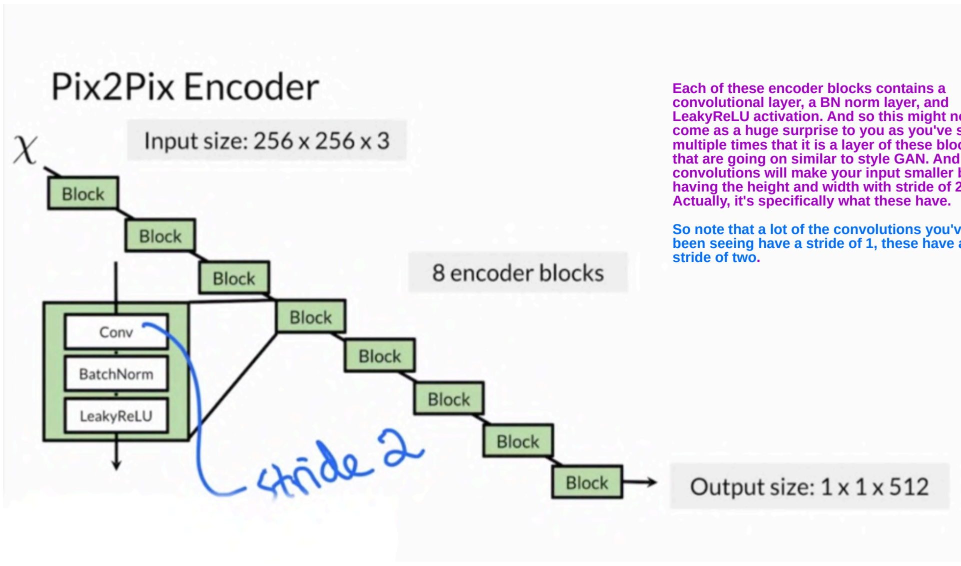
What's cool in U-Net that very much in the backward pass, this does improve gradient flow to the encoder so that those layers learn from information that might have been in the decoder here, too.


Pix2Pix Encoder
Input size: 256 x 256 x 3
χ
Block
Block
Block
Block
Block
Block
Block
Block
Image available from: https://arxiv.org/abs/1611.07004

So first, you have your encoder, which takes in an image x, of size, let's say 256 by 256 height width, with three channels of color, RGB. And in this example of this segmented image that is inputted as essentially that conditioned image, and it goes through all of these blocks. It goes through eight encoder blocks to compress that input.

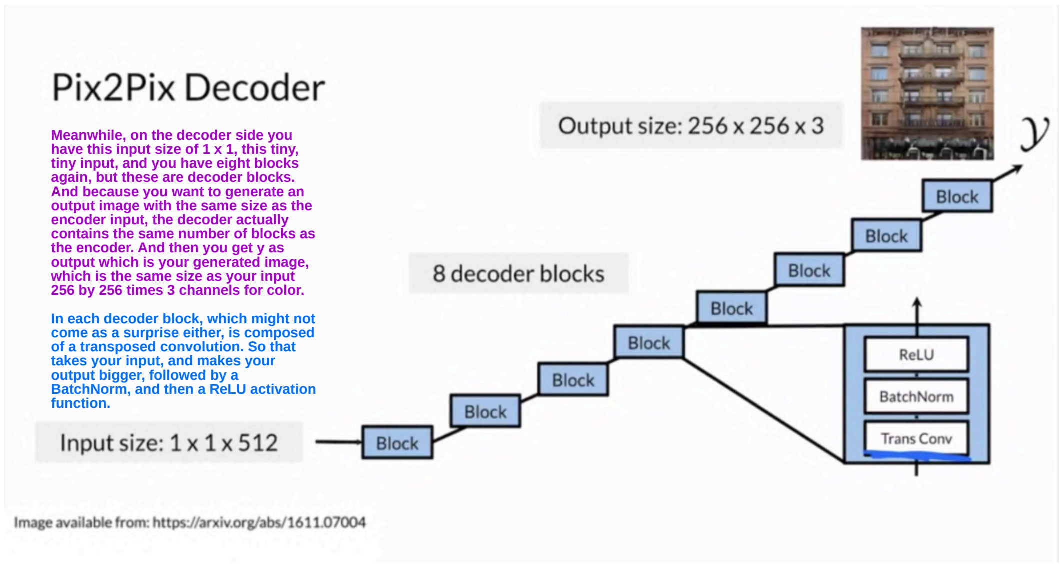


Then each block downsamples the spatial size by factor of two. So the output size of the encoder is 256 divided by 2 to the 8th or 1 x 1 spatial size by the very end, with 512 channels to encode that information. So at the very end is just as 1 x 1 here, height x width.



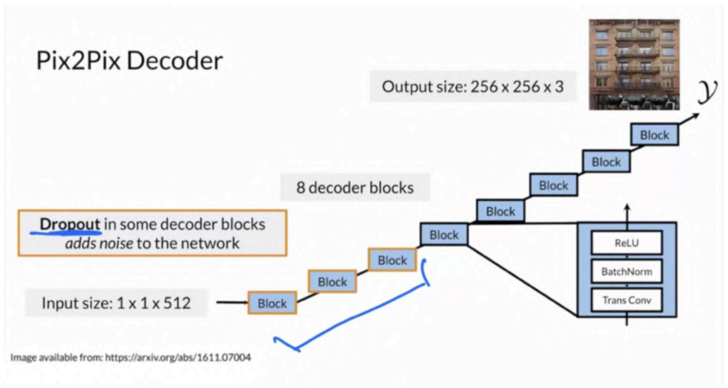
Each of these encoder blocks contains a convolutional layer, a BN norm layer, and LeakyReLU activation. And so this might not come as a huge surprise to you as you've seen multiple times that it is a layer of these blocks that are going on similar to style GAN. And so the convolutions will make your input smaller by having the height and width with stride of 2. Actually, it's specifically what these have.

So note that a lot of the convolutions you've been seeing have a stride of 1, these have a stride of two.



Meanwhile, on the decoder side you have this input size of 1 x 1, this tiny, tiny input, and you have eight blocks again, but these are decoder blocks. And because you want to generate an output image with the same size as the encoder input, the decoder actually contains the same number of blocks as the encoder. And then you get y as output which is your generated image, which is the same size as your input 256 by 256 times 3 channels for color.

In each decoder block, which might not come as a surprise either, is composed of a transposed convolution. So that takes your input, and makes your output bigger, followed by a BatchNorm, and then a ReLU activation function.

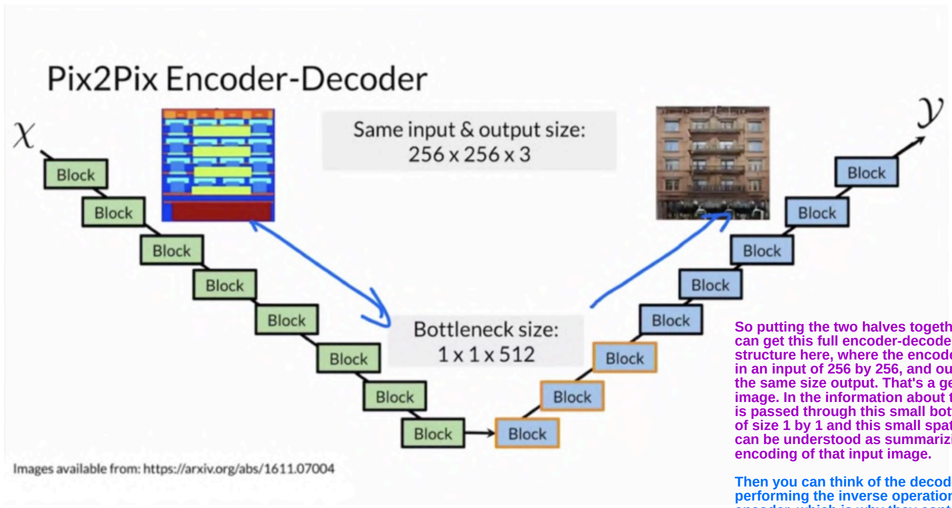

Pix2Pix Decoder
Output size: 256 x 256 x 3
y
8 decoder blocks
Dropout in some decoder blocks adds noise to the network
Block
ReLU
BatchNorm
Trans Conv
Input size: 1 x 1 x 512
Image available from: https://arxiv.org/abs/1611.07004

Dropout is added to this network, but it's actually just added to the first three blocks of this decoder.

Dropout randomly. Disabled different neurons at each iteration of training to allow different neurons to learn, and this is so that the same neurons aren't always learning the same thing.

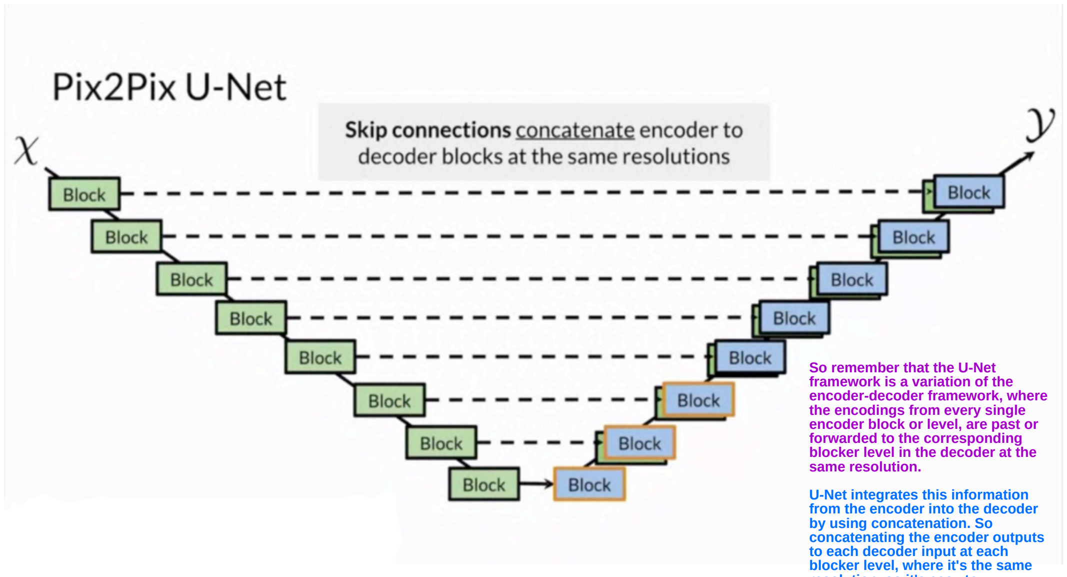
This stochasticity, this noise added to the network. Please note that this is only present during training, and as with all uses of Dropout, it is typically turned off during inference or test time. And that inference neurons are actually scaled by this inverse dropout probability to maintain the same kind of distribution that they expect.

That's not super important to know, but just know that Dropout does add some kind of noise to this model. Remember that we're taking away the noise as input right now, and so this is where stochasticity does seep into this model architecture, but only during training. During inference, you're not going to see that type of stochasticity of course, nor be able to inject that type of noise like that.


Pix2Pix Encoder-Decoder
x
Same input & output size: 256 x 256 x 3
y
Block
Bottleneck size: 1 x 1 x 512
Images available from: https://arxiv.org/abs/1611.07004

So putting the two halves together, you can get this full encoder-decoder structure here, where the encoder takes in an input of 256 by 256, and outputs the same size output. That's a generated image. In the information about the input is passed through this small bottleneck of size 1 by 1 and this small spatial size can be understood as summarizing the encoding of that input image.

Then you can think of the decoder as performing the inverse operations as the encoder, which is why they contain the same number of blocks, or eight blocks.



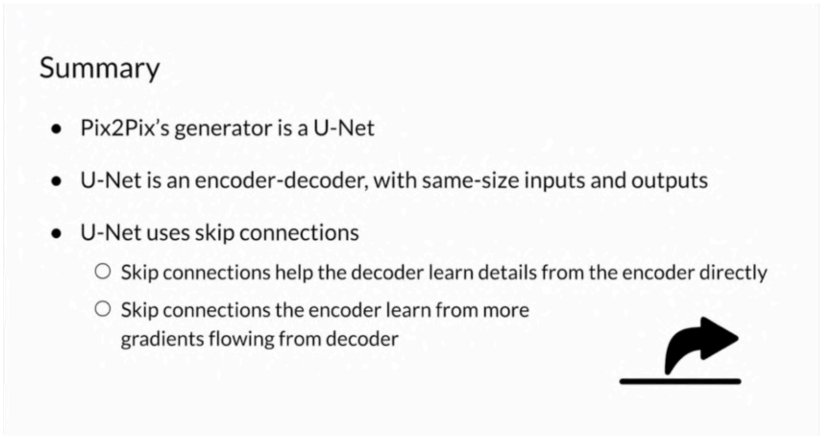
Pix2Pix U-Net

Skip connections concatenate encoder to decoder blocks at the same resolutions

So remember that the U-Net framework is a variation of the encoder-decoder framework, where the encodings from every single encoder block or level, are past or forwarded to the corresponding blocker level in the decoder at the same resolution.

U-Net integrates this information from the encoder into the decoder by using concatenation. So concatenating the encoder outputs to each decoder input at each blocker level, where it's the same resolution, so it's easy to concatenate them.

So in summary, Pix2Pix uses a unit for its generator, and a U-Net is an encoder-decoder framework that uses skip connections that concatenate same resolution or same block or same level. Feature maps to each other from the encoder to the decoder, and this helps the decoder learn more details from the encoder directly, in case there are finer details that are lost during the encoding stage. And the skip connections also help in the backwards layer, of course to help more gradient to flow from the decoder back to the encoder.

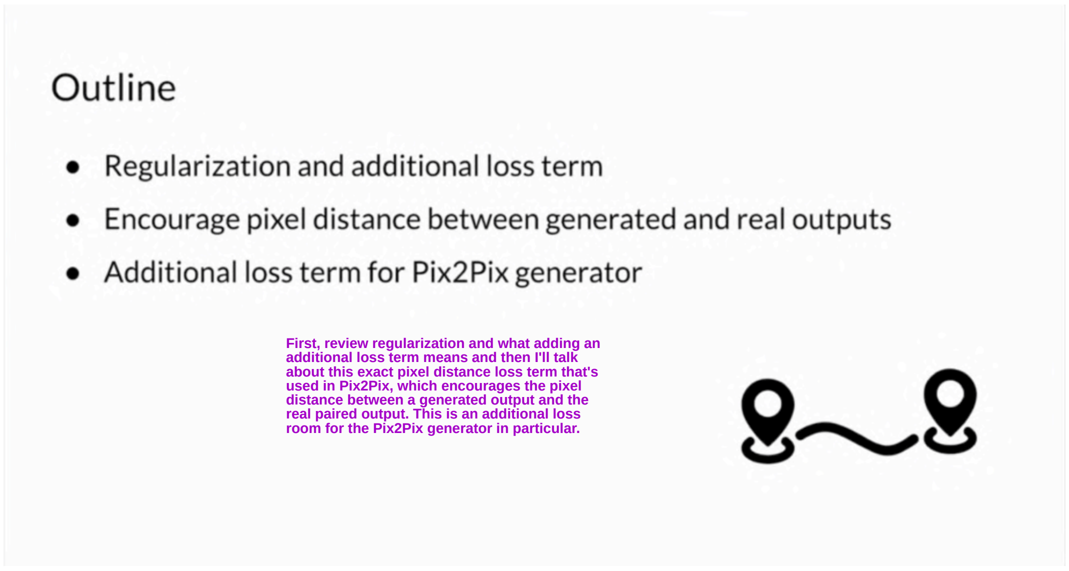


Summary

- Pix2Pix's generator is a U-Net
- U-Net is an encoder-decoder, with same-size inputs and outputs
- U-Net uses skip connections
  - Skip connections help the decoder learn details from the encoder directly
  - Skip connections the encoder learn from more gradients flowing from decoder

deeplearning.ai

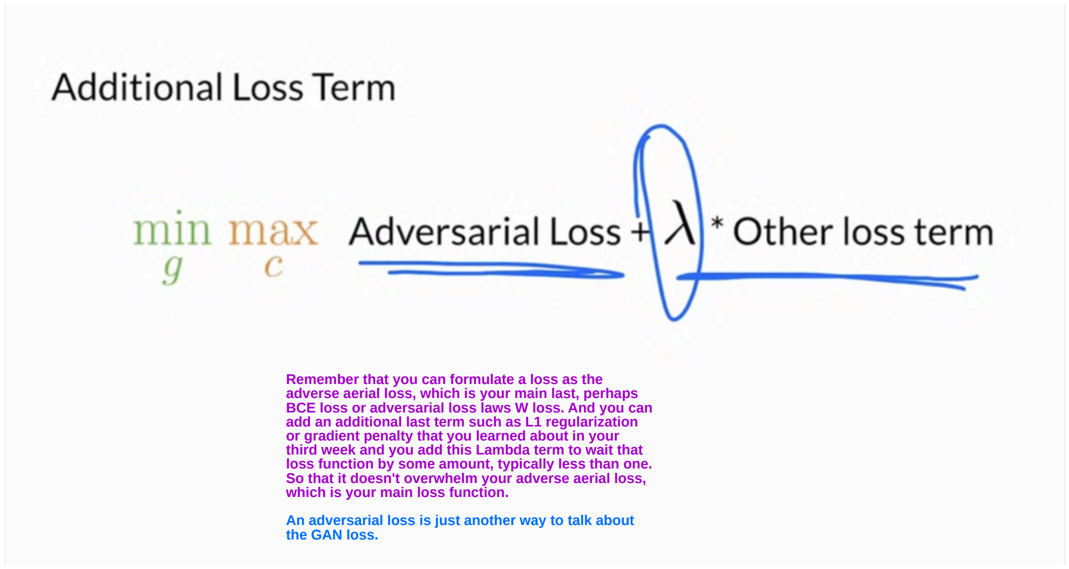
Pix2Pix: Pixel Distance Loss Term



Outline

- Regularization and additional loss term
- Encourage pixel distance between generated and real outputs
- Additional loss term for Pix2Pix generator

First, review regularization and what adding an additional loss term means and then I'll talk about this exact pixel distance loss term that's used in Pix2Pix, which encourages the pixel distance between a generated output and the real paired output. This is an additional loss room for the Pix2Pix generator in particular.

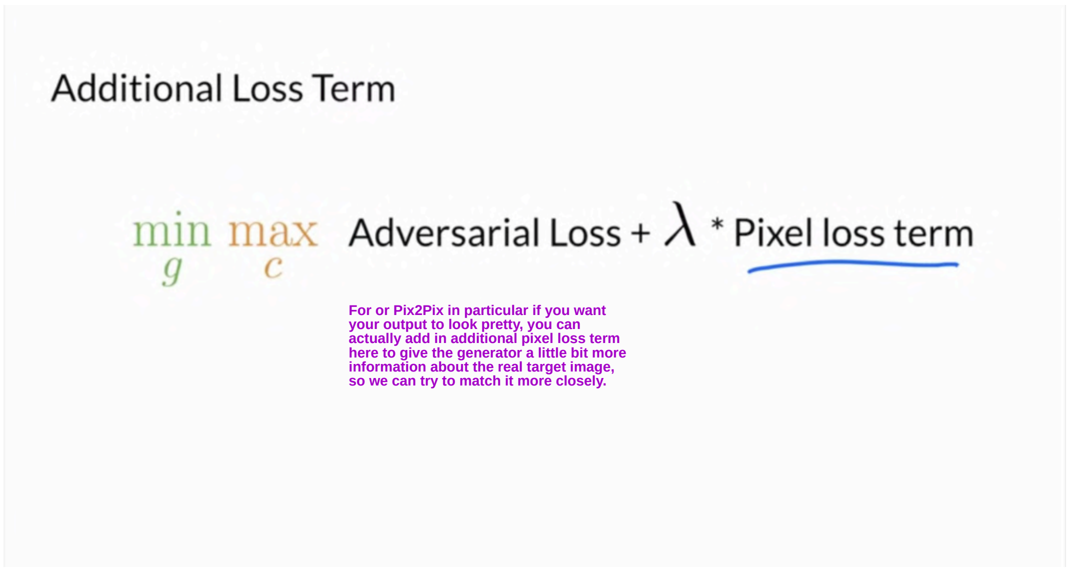


Additional Loss Term

$$\min_{g}\max_{c}\ \underline{\text{Adversarial Loss}} + \boxed{\lambda} * \underline{\text{Other loss term}}$$

Remember that you can formulate a loss as the adverse aerial loss, which is your main last, perhaps BCE loss or adversarial loss laws W loss. And you can add an additional last term such as L1 regularization or gradient penalty that you learned about in your third week and you add this Lambda term to wait that loss function by some amount, typically less than one. So that it doesn't overwhelm your adverse aerial loss, which is your main loss function.

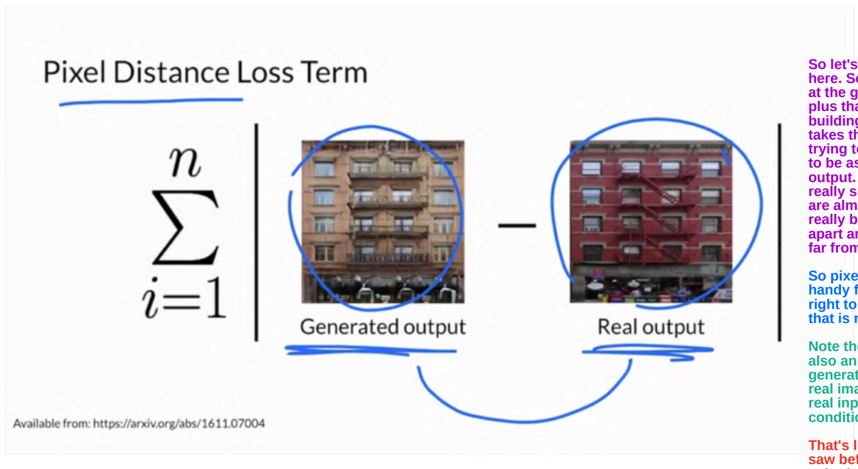
An adversarial loss is just another way to talk about the GAN loss.



Additional Loss Term

$$\min_{g}\max_{c}\ \text{Adversarial Loss} + \lambda * \underline{\text{Pixel loss term}}$$

For or Pix2Pix in particular if you want your output to look pretty, you can actually add in additional pixel loss term here to give the generator a little bit more information about the real target image, so we can try to match it more closely.



Pixel Distance Loss Term

$$\sum_{i=1}^{n} \left| \text{Generated output} - \text{Real output} \right|$$

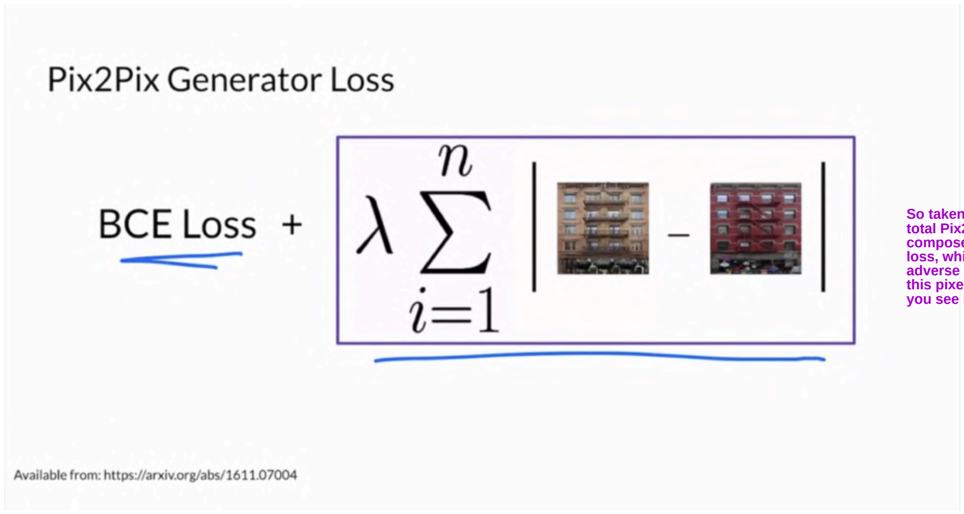
Available from: https://arxiv.org/abs/1611.07004

So let's take a look at what that means here. So the pixel distance last term looks at the generated output from the generator plus that real target output. This is the real building here is just the generated and it takes the pixel difference between the two, trying to encourage the generated output to be as close as possible to the real output. Because of this, pixel distance is really small. That means that the images are almost exactly the same, and if it's really big, that means they're really far apart and their generated image is really far from the real target output.

So pixel distance laws comes in a lot of handy for helping with this realism. Now, right to get exactly close to something that is real.

Note there's something tricky here. This is also an added layer supervision where the generator kind of quote unquote sees the real image now, right? It not only sees the real input image, which is just for conditioning.

That's like the one hot class factor you saw before, but now it implicitly sees the paired output in some way, even though this is a very softway and mainly makes those samples look super nice.

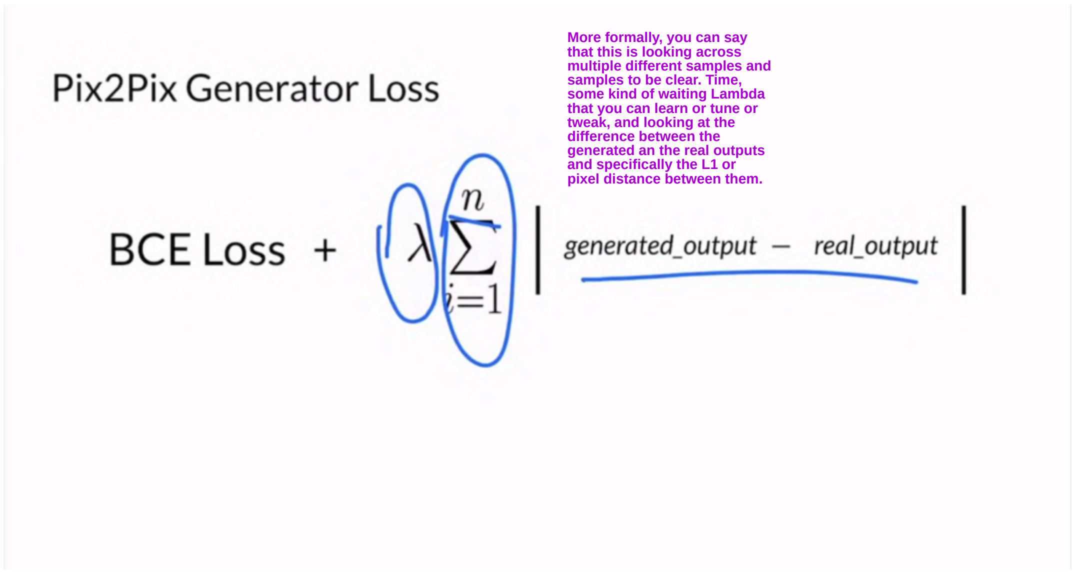


Pix2Pix Generator Loss

$$\text{BCE Loss} + \boxed{\lambda \sum_{i=1}^{n} \left| \quad - \quad \right|}$$

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

So taken together, they total Pix2Pix generator is composed of the BCE loss, which is that adverse aerial loss plus this pixel distance loss you see here.



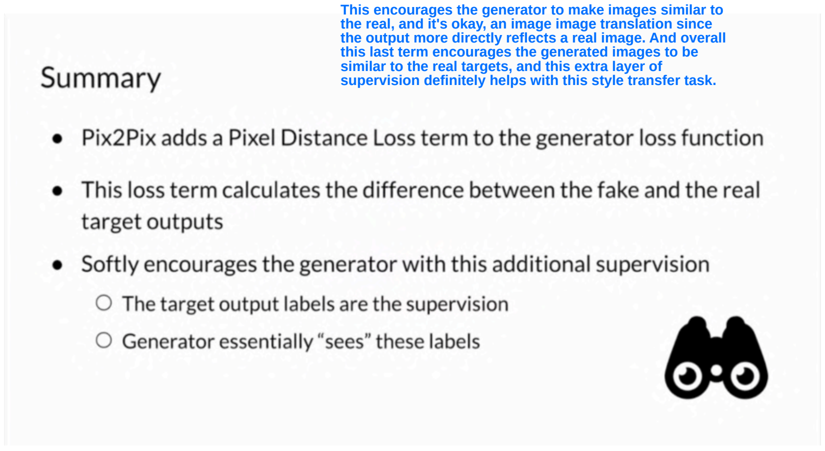
Pix2Pix Generator Loss

More formally, you can say that this is looking across multiple different samples and samples to be clear. Time, some kind of waiting Lambda that you can learn or tune or tweak, and looking at the difference between the generated an the real outputs and specifically the L1 or pixel distance between them.

$$\text{BCE Loss} + \lambda \sum_{i=1}^{n} \left| \mathit{generated\_output} - \mathit{real\_output} \right|$$

So in summary Pix2Pix lets the generator quote unquote, see the reels by adding an L1 regularization term to his loss function that takes the pixel difference between the real target output in the fake one.

This encourages the generator to make images similar to the real, and it's okay, an image image translation since the output more directly reflects a real image. And overall this last term encourages the generated images to be similar to the real targets, and this extra layer of supervision definitely helps with this style transfer task.

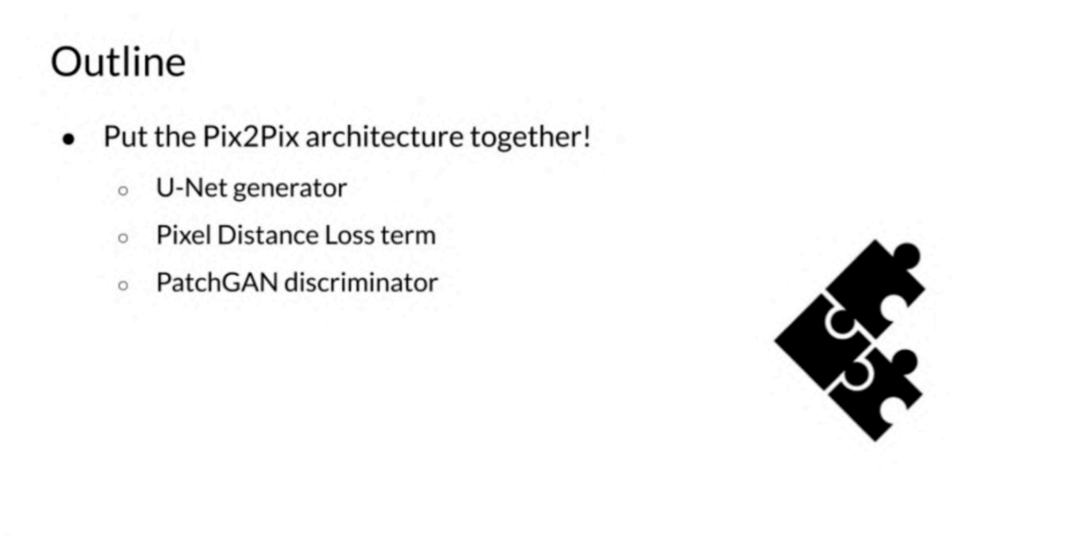


Summary

- Pix2Pix adds a Pixel Distance Loss term to the generator loss function
- This loss term calculates the difference between the fake and the real target outputs
- Softly encourages the generator with this additional supervision
  - The target output labels are the supervision
  - Generator essentially "sees" these labels

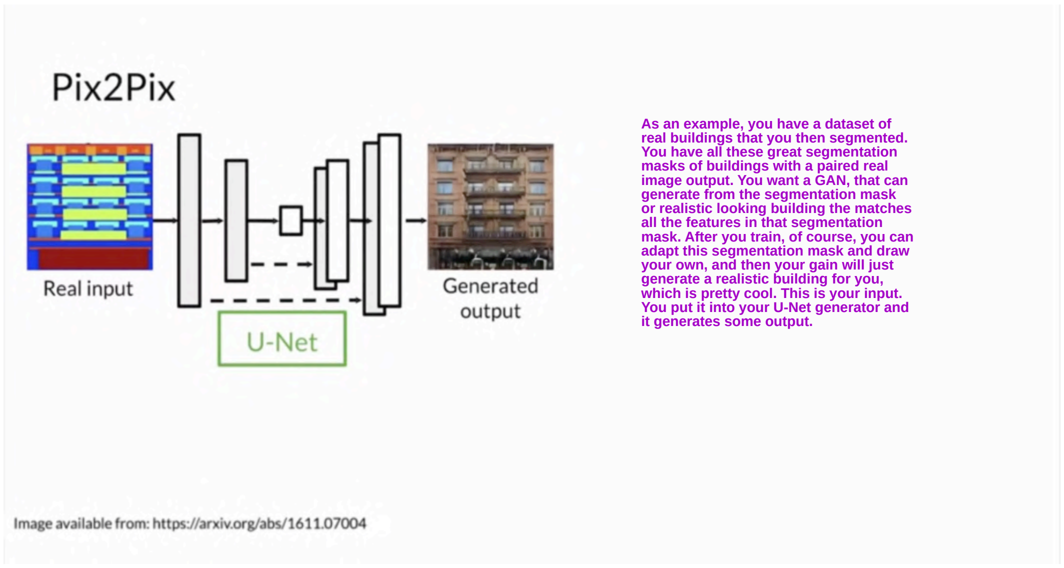
deeplearning.ai

Pix2Pix: Putting It All Together

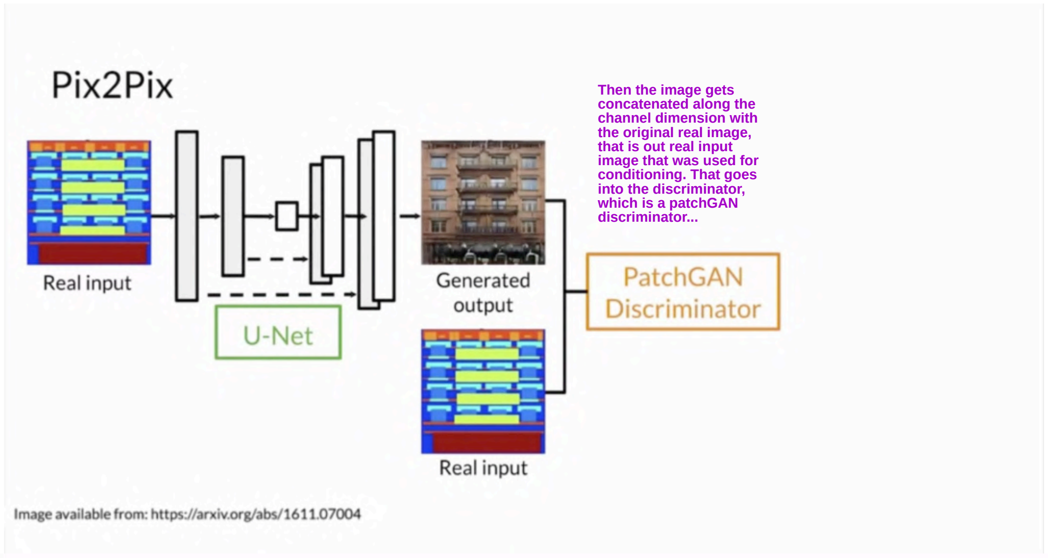


Outline

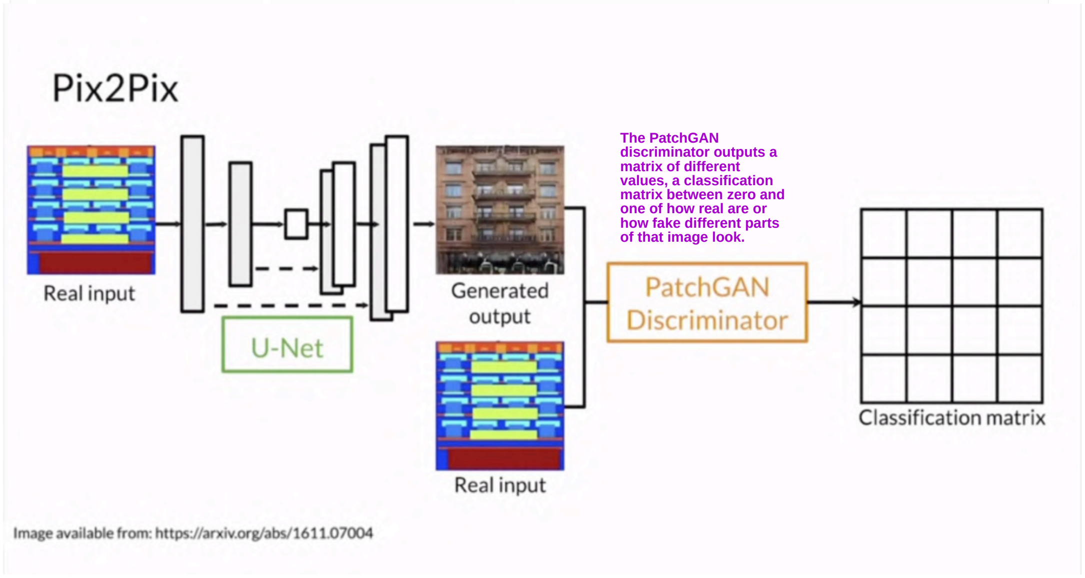
- Put the Pix2Pix architecture together!
  - U-Net generator
  - Pixel Distance Loss term
  - PatchGAN discriminator



As an example, you have a dataset of real buildings that you then segmented. You have all these great segmentation masks of buildings with a paired real image output. You want a GAN, that can generate from the segmentation mask or realistic looking building the matches all the features in that segmentation mask. After you train, of course, you can adapt this segmentation mask and draw your own, and then your gain will just generate a realistic building for you, which is pretty cool. This is your input. You put it into your U-Net generator and it generates some output.

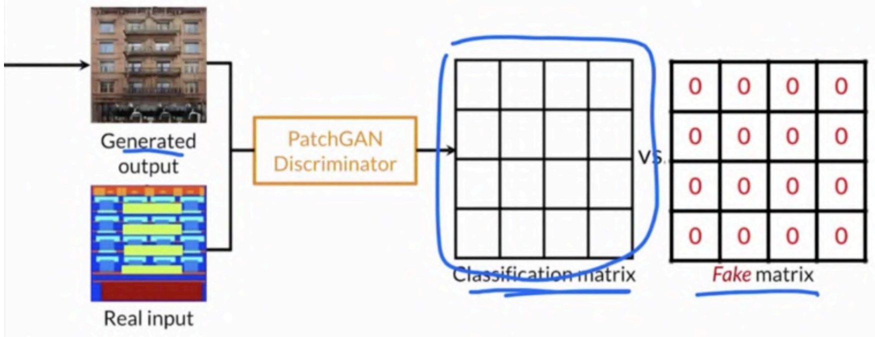


Then the image gets concatenated along the channel dimension with the original real image, that is out real input image that was used for conditioning. That goes into the discriminator, which is a patchGAN discriminator...



The PatchGAN discriminator outputs a matrix of different values, a classification matrix between zero and one of how real are or how fake different parts of that image look.

Pix2Pix: Discriminator Loss



Then for the discriminator's loss, the output of concatenating the generated output with the real input will be compared to the fake label matrix, which is a matrix of all zeros because a discriminator will succeed if it classified every single patch of that image looked fake, was zero.

Image available from: https://arxiv.org/abs/1611.07004

Then on a real outputs, so let's say there's a real output instead, then the discriminator will want to compare its classification matrix with the real label because they want to get as close as possible to all ones in this classification matrix in his predictions. Again, as input, these were concatenated along the channel dimension.

Now for the generator's loss, it's still the discriminator looking at the generated output concatenated with that real input along the channel dimension. But instead of a matrix of all zeros assimilable is looking at the real labeled matrix of ones because the generator wants a discriminator to think that every single patch of it's generated image looks real.

Pix2Pix: Discriminator Loss

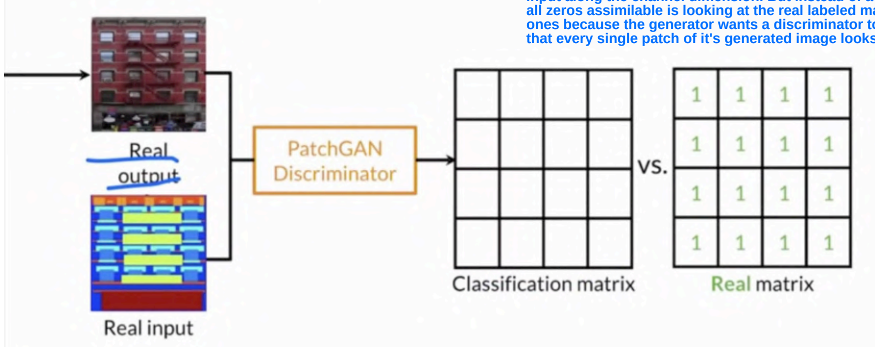


Image available from: https://arxiv.org/abs/1611.07004

But that's not it for the generator's loss. That's just using BCE loss between these two matrices.

The generator also has this pixel distance loss term that's looking at how different it's generated output is to the real target output multiplied by some waiting Lambda there. That's the full generator loss.

Pix2Pix: Generator Loss

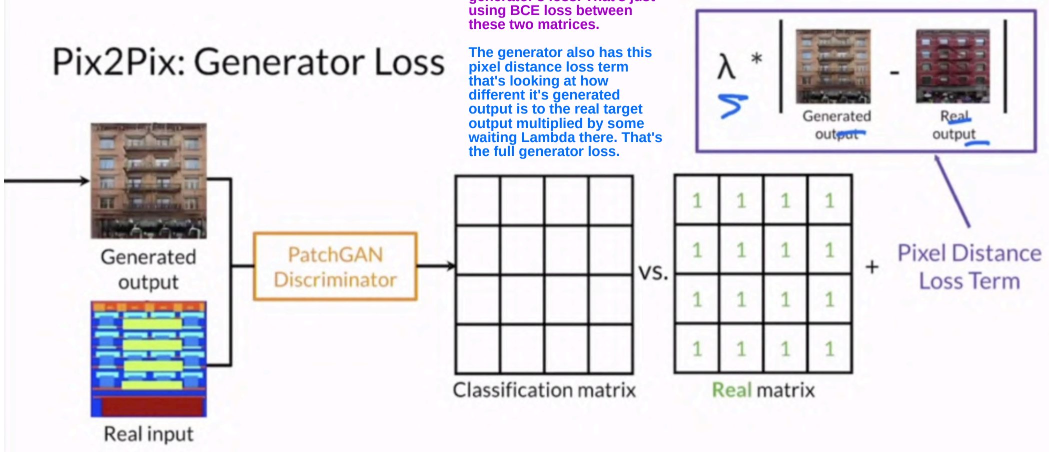
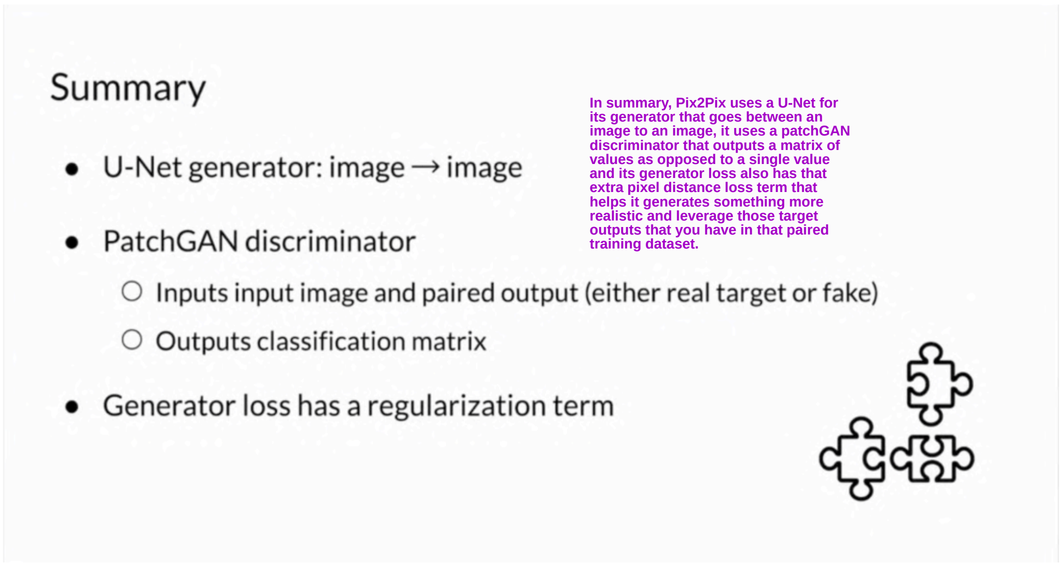


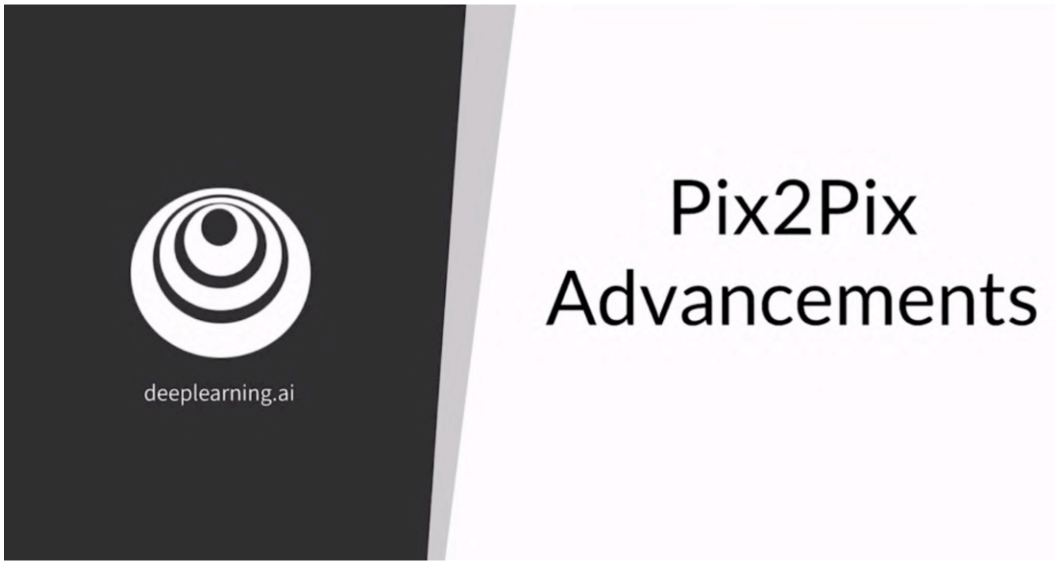
Image available from: https://arxiv.org/abs/1611.07004



Summary

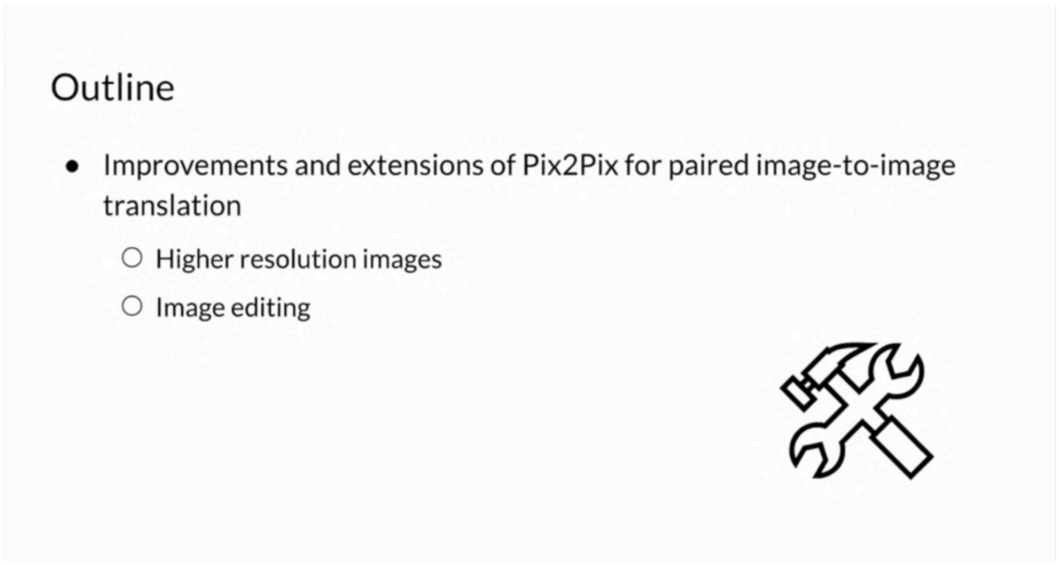
- U-Net generator: image $\rightarrow$ image
- PatchGAN discriminator
  - Inputs input image and paired output (either real target or fake)
  - Outputs classification matrix
- Generator loss has a regularization term

In summary, Pix2Pix uses a U-Net for its generator that goes between an image to an image, it uses a patchGAN discriminator that outputs a matrix of values as opposed to a single value and its generator loss also has that extra pixel distance loss term that helps it generates something more realistic and leverage those target outputs that you have in that paired training dataset.



deeplearning.ai

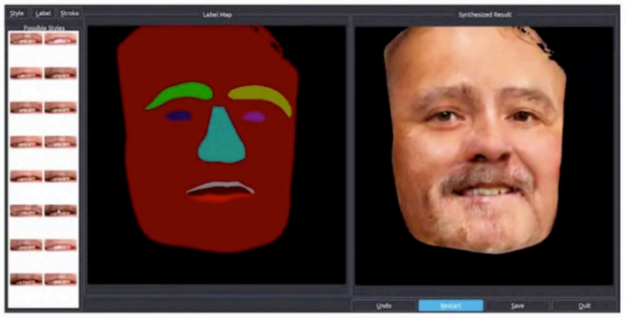
Pix2Pix Advancements



Outline

- Improvements and extensions of Pix2Pix for paired image-to-image translation
  - Higher resolution images
  - Image editing

Pix2PixHD



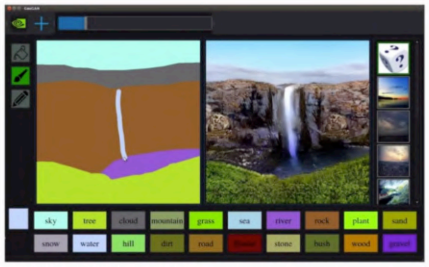
Available from: https://github.com/NVIDIA/pix2pixHD

First another model came out from the same group at Berkeley and also NVIDIA that works on higher resolution images, so it's called Pix2PixHD, high def. Really what's cool about Pix2PixHD is that it operates on much higher resolution images and includes a lot of modifications that make it significantly better. So I do recommend checking out the Pix2PixHD paper and an optional notebook if you want to be doing image-to-image translation.

What's shown here is something super cool which you can also use in the regular Pix2Pix. Basically what you can do is you can have the segmentation mask of someone's face and you can adapt that mask however you want and be able to generate a different type of face. And actually I think what's being done here is it's not even changing the mask. You actually just have lots of different possible faces that you can generate from this one mask, so that's pretty cool too. You can definitely do the version where you adopt the mask using Pix2Pix.

Another really really cool model that's come out is called GauGAN by NVIDIA. And this is a pun on an artist named Paul Gauguin. And GauGAN is super cool in terms of its application. I still remember the demo that they gave at a conference. Basically what you do, you can draw sketches here on the left and indicate what kind of class they are. So you can say like hey I want some Sky up here, I want this one line of water and then it's able to generate this realistic photo for you. And what's cool about GauGAN is that it uses some of the advances that you've seen in StyleGAN. And it actually uses adaptive instance normalization to take in the segmentation map, and use AdaIN, adaptive instance normalization for informing styles again.

GauGAN



Available from: https://blogs.nvidia.com/blog/2019/03/18/gaugan-photorealistic-landscapes-nvidia-research/

Summary

- Pix2PixHD and GauGAN are successors of Pix2Pix
- They are designed for higher resolution images
- They highlight opportunities for image editing using paired image-to-image translation
  - Pix2Pix can do this too, of course!

(Optional) The Pix2Pix Paper

Want to know more about image-to-image translation and the research behind the components of Pix2Pix? Take a look at the original paper!

Image-to-Image Translation with Conditional Adversarial Networks (Isola, Zhu, Zhou, and Efros, 2018): https://arxiv.org/abs/1611.07004

(Optional Notebook) Pix2PixHD

https://colab.research.google.com/github/https-deeplearning-ai/GANs-Public/blob/master/C3W2_Pix2PixHD_(Optional).ipynb

Please note that this is an optional notebook, meant to introduce more advanced concepts if you're up for a challenge, so don't worry if you don't completely follow!

In this notebook, you will learn about Pix2PixHD, which synthesizes high-resolution images from semantic label maps. Proposec in High-Resolution Image Synthesis and Semantic Manipulation with Conditional GANs (Wang et al. 2018), Pix2PixHD improves upon Pix2Pix via multiscale architecture, improved adversarial loss, and instance maps.

(Optional Notebook) Super-resolution GAN (SRGAN)

https://colab.research.google.com/github/https-deeplearning-ai/GANs-Public/blob/master/C3W2_SRGAN_(Optional).ipynb

Please note that this is an optional notebook meant to introduce more advanced concepts. If you're up for a challenge, take a look and don't worry if you can't follow everything. There is no code to implement—only some cool code for you to learn and run!

In this notebook, you will learn about Super-Resolution GAN (SRGAN), a GAN that enhances the resolution of images by 4x, proposed in Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network (Ledig et al. 2017). You will also implement the architecture and training in full and be able to train it on the CIFAR dataset.

(Optional) More Work Using PatchGAN

Want to see how a GAN can fill-in cropped-out portions of an image? Read about how PGGAN does that by using PatchGAN!

Patch-Based Image Inpainting with Generative Adversarial Networks (Demir and Unal, 2018): https://arxiv.org/abs/1803.07422

(Optional Notebook) GauGAN

https://colab.research.google.com/github/https-deeplearning-ai/GANs-Public/blob/master/C3W2_GauGAN_(Optional).ipynb

Please note that this is an optional notebook meant to introduce more advanced concepts. If you're up for a challenge, take a look and don't worry if you can't follow everything. There is no code to implement—only some cool code for you to learn and run!

In this notebook, you will learn about GauGAN, which synthesizes high-resolution images from semantic label maps, which you implement and train. GauGAN is based around a special denormalization technique proposed in Semantic Image Synthesis with Spatially-Adaptive Normalization (Park et al. 2019)

Works Cited

All of the resources cited in Course 3 Week 2, 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:

- DeOldify... (Antic, 2019): https://twitter.com/citnaj/status/1124904251128406016
- pix2pixHD (Wang et al., 2018): https://github.com/NVIDIA/pix2pixHD
- [4k, 60 fps] Arrival of a Train at La Ciotat (The Lumière Brothers, 1896) (Shiryaev, 2020): https://youtu.be/3RYNThid23g
- Image-to-Image Translation with Conditional Adversarial Networks (Isola, Zhu, Zhou, and Efros, 2018): https://arxiv.org/abs/1611.07004
- Pose Guided Person Image Generation (Ma et al., 2018): https://arxiv.org/abs/1705.09368
- AttnGAN: Fine-Grained Text to Image Generation with Attentional Generative Adversarial Networks (Xu et al., 2017): https://arxiv.org/abs/1711.10485
- Few-Shot Adversarial Learning of Realistic Neural Talking Head Models (Zakharov, Shysheya, Burkov, and Lempitsky, 2019): https://arxiv.org/abs/1905.08233
- Patch-Based Image Inpainting with Generative Adversarial Networks (Demir and Unal, 2018): https://arxiv.org/abs/1803.07422
- Image Segmentation Using DIGITS 5 (Heinrich, 2016): https://developer.nvidia.com/blog/image-segmentation-using-digits-5/
- Stroke of Genius: GauGAN Turns Doodles into Stunning, Photorealistic Landscapes (Salian, 2019): https://blogs.nvidia.com/blog/2019/03/18/gaugan-photorealistic-landscapes-nvidia-research/

From the notebooks:

- Crowdsourcing the creation of image segmentation algorithms for connectomics (Arganda-Carreras et al., 2015): https://www.frontiersin.org/articles/10.3389/fnana.2015.00142/full
- U-Net: Convolutional Networks for Biomedical Image Segmentation (Ronneberger, Fischer, and Brox, 2015): https://arxiv.org/abs/1505.04597