Outline

- Why evaluating GANs is hard
- Two properties: fidelity and diversity

In this lecture you'll learn about evaluating GANs. Starting with understanding a couple of criteria or properties you want your GAN to have.

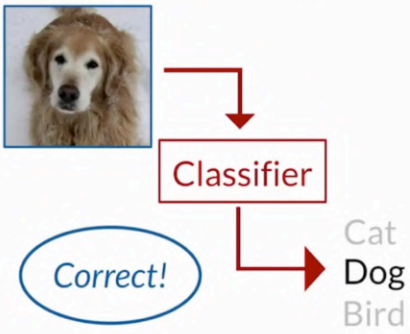
So first you'll see why it's challenging to evaluate a GAN. And then you'll learn about two of the most important properties, fidelity and diversity.

Why is evaluating GANs hard?

So evaluating GANs is similar to evaluating other models in that you typically take a model checkpoint, weights frozen at a certain point. And compare its output against some metrics, and these metrics could be used across models, not just your own for an independent evaluation.

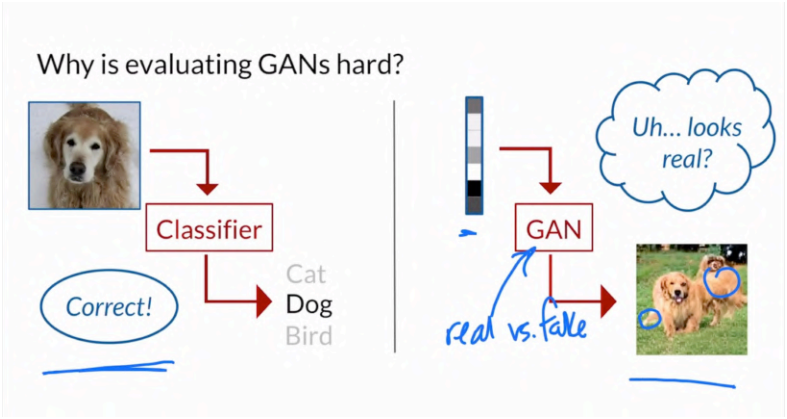
But evaluating GANs is a particularly challenging task in an active area of research that's seen a lot of progress recently.

Why is evaluating GANs hard?



So to illustrate, first in supervised learning with a classifier. There are tests you can pass on which have labels on how your images should be classified.

And based on these labels, you can evaluate if your model is right or wrong, if it's correct or not, you have a sense of correctness here. And usually there's like held out test set on which you can evaluate how right or wrong your model is. And this test set can be used to evaluate your model or other peoples models.



However, with a GAN you pass in some random noise to it and you get this fake image. But there's no concrete way of telling how realistic these generated images are. You don't know the exact pixels it's supposed to generate, you can't say. This pixel right here is slightly off or right there is supposed to be green. And so because there's that, there's no clear goal for what pixels you're supposed to generate given this noise vector going in.

So this model is more like an advanced artist learning how to paint masterpieces. As opposed to learning exact brush strokes in a known painting. Which is just this correctness from the classifier, because there's a clear right and wrong.

Additionally, the discriminator in this GAN which classifies real versus fake doesn't ever reach perfection. And often overfits to discriminating real versus fake images for its particular generator. So you might think wow, because its classifying between real and fake images maybe it could be useful for this. But no, it's overfitting to your generator, so it will likely think a lot of images from your particular generator. Even though they look realistic, are in fact fake because it can pick up on certain qualities. And so those certain qualities could be small, sometimes even perceptible things that the generator is producing. And because of this, there are no perfect or universal discriminators that can look at two generators. And say for sure that this one is better than the other

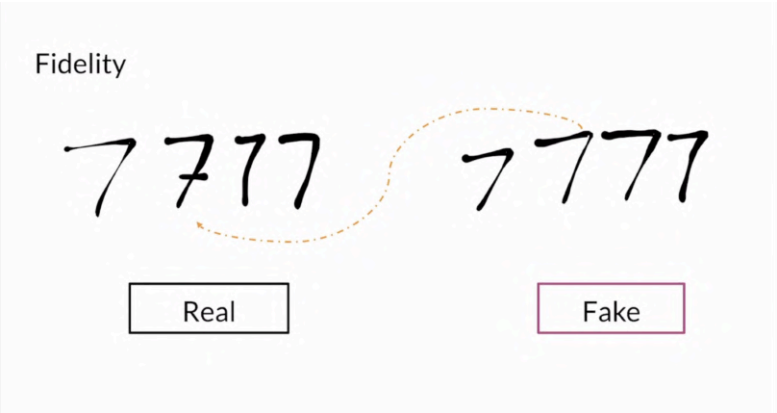
So then, how do you evaluate GANs? Well, you can start by first defining the desired criteria you want from these properties you might want. And one primary property, one primary objective is certainly fidelity or the quality of your generated images and how realistic they look. And you can think of quality overall and fidelity overall as its realism factor but also the crispness of the image. For example, a blurry face could still look realistic, but it wouldn't be high fidelity per say. So the fidelity of this picture of a person is pretty good here. But just generating a single image is not what you want from your generator. A good generator also produces a good variety of images, it wouldn't be very useful to just produce this one image here.

So the second property you typically care about is diversity. What's the range, what's a variety of images this generator is able to produce? Is it able to produce the diversity that's inherent in the training data set or the desired classroom modeling? That is, of all dogs in this case, can it model all types of dogs, all breeds of dogs, dogs in different places, in different positions? Or is it just going to generate a single, very realistic image, like with this face over here?

So that's why when you're evaluating again, it's important to consider not only fidelity, that quality of images which obviously is very important. But also this diversity aspect as well of what's the range of images it can generate. To get an idea of whether your GAN covers a good variety of what's expected and real.

So, all of these can be pretty tough, because how do you really evaluate or quantify whether something has sufficient diversity. When you don't want to necessarily memorize the training data set?

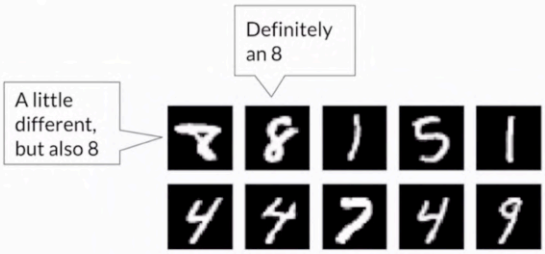
In summary, we have two properties, fidelity and diversity, and there on two axes. And sometimes you can think of them as trading one off for the other.



So for fidelity, you can consider how realistic your images are looking from your GAN. So think of it this way, so for each fake sample, how different is it from its nearest real sample. So you can also think about it more generally as, how far are 100 fakes from 100 reals to get a better representation?

You don't want a one hit wonder GAN, you don't want to miss good and bad high and low fidelity samples. Because those could differentiate this GAN from another GAN, you want a GAN that can consistently give you good results. And so there are a few ways you can do this comparison and you'll learn about some of them later in this course.

Diversity



On diversity, you want the generated images to cover the whole diversity, the variety of the real distribution. Meaning if a GAN is only generating the same single image, but it's very realistic, that's not a well performing model. You might remember that this is similar to what happens when there's mode collapse. So you want a GAN that can generate a variety of different images, such as these 8s written in different styles. And you can also measure and get a sense of the spread of say, 100 for examples to the spread of 100 real ones.

So when evaluating GANs, fidelity and diversity are both criteria you care about. By capturing fidelity and diversity, you can get a pretty good notion of how well you generator is generating fake images. And this could be just looking through how close your fake images are to your real images.

Summary

- No ground-truth = challenging to evaluate
- Fidelity measures image quality and diversity measures variety
- Evaluation metrics try to quantify fidelity & diversity

So in summary, it's challenging to evaluate again because there is no global discriminator. Giving a ground truth that would enable fair comparisons across GANs. And to evaluate a GAN, you want to consider fidelity or the quality of images. As well as the diversity or variety of those images coming out of your GAN. And with these properties, these criteria mind, you'll learn ways you can evaluate your GAN in the following videos.

Comparing Images

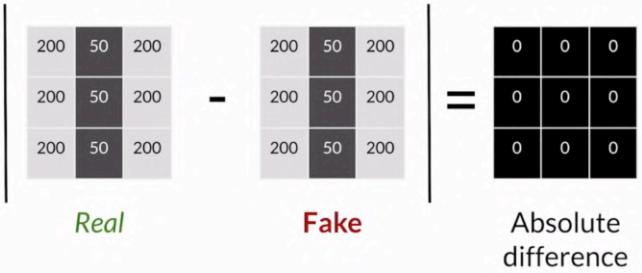
In this section you'll learn about a way to compare different images. For example real versus fake images in order to evaluate your GAN.

Outline

- Pixel distance
- Feature distance

Comparing images on fidelity and diversity can be challenging, because what exactly should you be comparing? First you explore what pixel distance is, a simple approach, but insufficient. Then you'll get to compare images using feature distance, which will give you greater reliability in your comparisons.

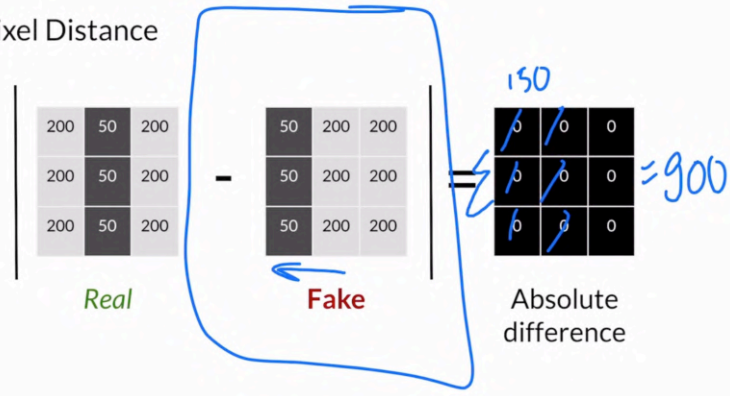
Pixel Distance



So probably these simple's approach to comparing images is looking at the differences between their pixels, and this is known as a pixel distance. So with the real image and a fake image here you can subtract their pixel values. Value from 0 to 255 in one image from the other and then get their absolute difference. So you're subtracting each pixel with the other and then you get each of their differences.

Perhaps then you can sum their differences. And here the total difference is 0s. So far this looks great because these two images are identical and we get 0 out here. So they look identical and the absolute difference is 0 so they have no distance from each other, perfect.

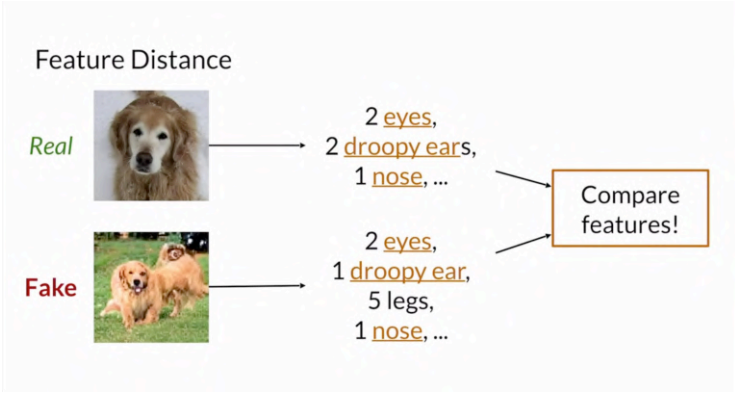
Pixel Distance



But pixel distance actually isn't that reliable. For example, imagine image shifted one pixel to the left here in the fake image. This would potentially have a huge pixel distance from the original image, or from this real image here, even though the images are really similar. Even imperceptibly different to your eye when looking at say, high resolution image with millions of pixels.

So if this image were super large and it only shifted one pixel, you probably wouldn't notice anything.

However, based on pixel distance, this absolute difference would then be huge. So then all of these values here would all evaluate to 150. And if you some this all together, you would get a huge pixel distance of 900.



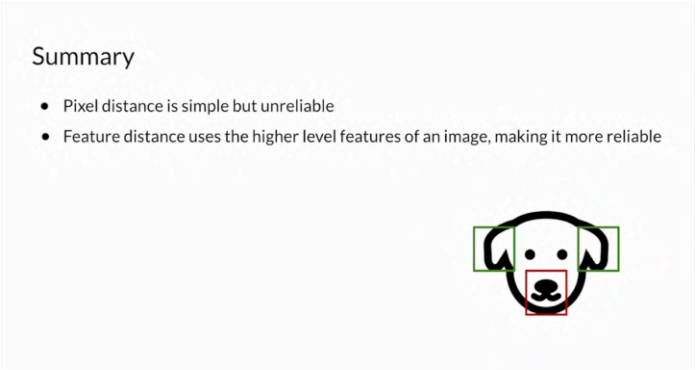
So one alternative is to look at the higher level features of your images instead of the pixels. So, does a dog have two eyes? Is the nose under the eyes is their fur? And this higher level semantic information would be less sensitive to small shifts.

So instead of looking at pixels directly, you could condense or explain the image using its features, like having two eyes, droopy ears and nose. And then you can compare the images by using these extracted features, okay? Now I'm going to compare these images at what I call the feature level, so looking at their features. And this is a neat trick for comparing higher level semantic information between images. And is pretty common outside of Gans as well.

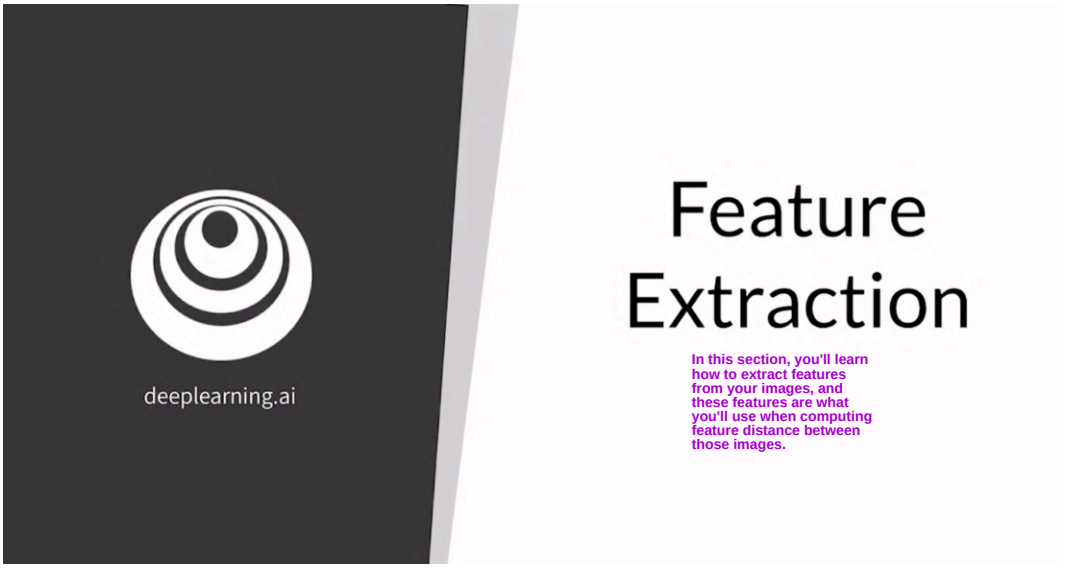
So by using these extracted features, your evaluation is less sensitive to small differences in the images. And in this example, although the background of the images are different, they're both still identifiable as dogs. With pixel distance, the fake image would be really far off from the real. And you'll see more details on how to get these features as well as calculate the feature distance in the following sections

And what's interesting here is that you can see that these two images are similar in some ways, having both having two eyes. But here this guy has two droopy ears, and this one only has one drew year, and this one seems to have five legs. So that seems kind of off on the fake side, and both of them have a nose.

So you can imagine a fake image that's further away from this real that only has one eye or two noses. Or one that's closer to your real image that doesn't have any legs and has two droopy ears. So that's how you're going to be getting distance between these images. It's add this feature level.



So in summary, when evaluating again comparing images with pixel distance is simple but too sensitive and unreliable. Feature distance is an alternative to that in words by extracting higher level features instead of pixels to compare your images. As a result, it's more reliable since it looks at higher level information.



In this section, you'll learn how to extract features from your images, and these features are what you'll use when computing feature distance between those images.

Outline

- Feature extraction using pre-trained classifiers
- ImageNet dataset

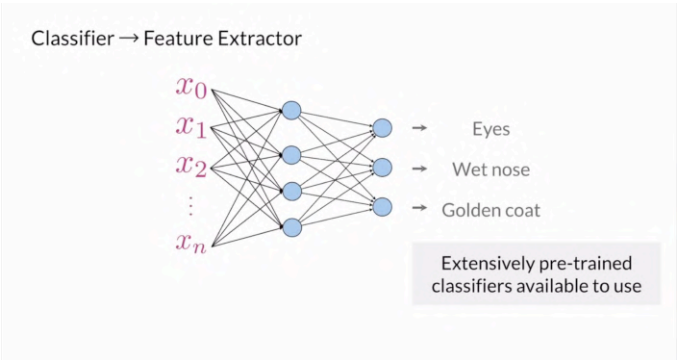
So specifically, you'll learn how to extract features using classifiers that have trained on so many images that these classifiers are generally regarded as being able to model natural images. Or photographs and these many images are typically from the famous RJ, data set called ImageNet, which I'll also touch upon in this section.

So in order to compute the feature distance between real and fake images, you first need a way to extract the features from those images. You can get a feature extractor by taking the weights of a pre trained classifier, typically on many images and ideally on data on classes that are somewhat related to the features you want to extract.

For example, if your classifier was classifying dogs and cats, then it would be a pretty good feature extractor for dogs and cats, but maybe not for office objects. In the weights of these pre trained models have essentially encoded features a lot of features as they have to, for example to classify dogs from plants. For example, it would have to have figured out what this wet nose Golden coat eyes are, as well as maybe also what a plant shape looks like and this might not be at the very last layer here. This might be an intermediate layer where there are more nodes downstream and they would have to have figured out some of these features along the way.

Although in reality, note that these features extracted are a bit more abstract and don't necessarily correspond to our notion of a wet nose or Golden coat or a plant shape.

You might be wondering if you need to build and train a you classifier each time you evaluate a GAN. Thankfully the answer is no as there are neural networks that have been pre trained on millions of images and hundreds or thousands of classes. You can plug in an used to extract features from an image input and these images are typically considered broad enough to be applicable to most natural images. That is, photos of the real world and so these pre trained classifiers are readily available for public use and offer a method that can be used across a variety of GAN, so all different GANs that you train. So these classifiers can classify many different classes and as a result they encode lots of relevant features in their network.

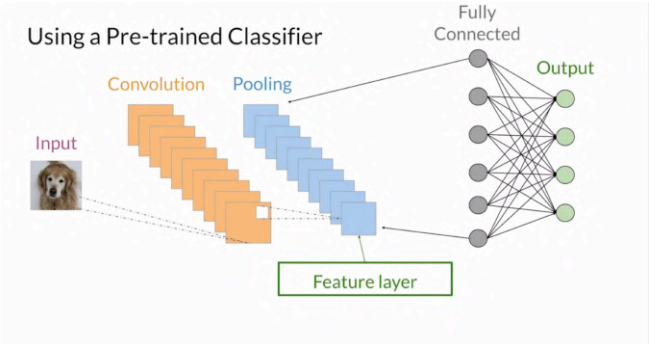


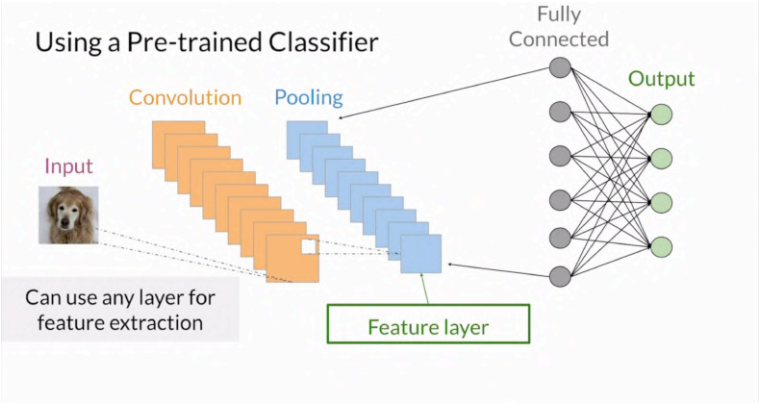
All right, so to use this pre trained classifier, you actually don't want the end task of classification at this output. You don't care about the actual classifier part, but the rest of the network is valuable. Its weights have learned important features that help with this classification task at the end.

So as a result you can lobby off this final classification layer and just grab the outputs from an earlier layer that contains useful information about the image you're putting through. So here you see the final layers of a generic convolutional neural network, or CNN a convolution, pooling. Fully Connected or linear layer with an activation right after that goes into your output predictions and you can usually lobby off this last fully connected layer.

This is because the most common place to get output features is the pooling layer before that last fully connected layer that's used for classification. Because at this layer, at this pooling layer, you have the most fine grained feature information. That is, this layer must have encoded quite a bit of information so we can then go on pass that information on to this fully connected layer to classify that image with just one more fully connected layer.

So you can then truncate the network as you saw with lobbying this last part off and use this last pooling layer. And this just means the values that come out of this layer, you take this essentially intermediate values out and these values. Let's say there are 100 of them come out from your let's say 500 by 500 output, and so those hundred values out here represent the features extracted from this model for this particular input image. And you can think of this layer as the feature layer because it is extracting those feature values. And you can also tell quite immediately that this feature layer will be outputting far fewer values than your input, so it condenses the pixel values in your input into these 100 features.

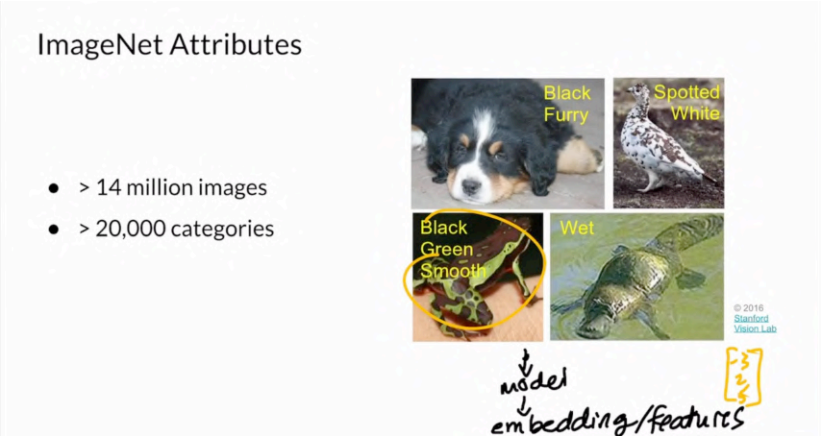




It's also okay to use an earlier layer for your feature layer, so using the last layer is just convention because it has the most information. But it also could be over fit to your data set and task like the classes for your classification out here.

So that means earlier layers are better for getting more primitive information, so as you go back along the network to an earlier convolution or typically pooling layer. And for those earlier layers, on one extreme, if you go to the earliest layer would probably just mean vertical edge detection and be able to extract features around the vertical edges in your input. Versus the last pooling layer out here, which will have features that apply to specific image classes, like whether there's a cat in the image or not, if the output is predicting a cat somewhere here.

So choosing what to call your feature layer is something you can experiment with, but I typically suggest starting with this last pooling layer because it will be trained on a huge data set with a broad task.

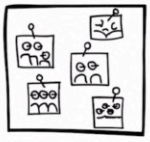


This dataset, essentially provides a plethora of information that can be meaningfully encoded in a classifier that is trained on it. In the features you extract from that classifier are sometimes known as ImageNet embeddings, because they embed and compress that information from an image. Into a smaller vector of information using the weights from a network trained on this ImageNet data set to then guide that embedding process.

And these features, these embeddings are just vectors that exist in what's more broadly known as feature space or embedding space. So you can imagine an image like this going in, and the output features are really just a vector coming out of negative three, two, five. A various values that represent a vector in a certain space, which then represent your features, and this again can also be known as an embedding.

Summary

- Classifiers can be used as feature extractors by cutting the network at earlier layers
- The last pooling layer is most commonly used for feature extraction
- Best to use classifiers that have been trained on large datasets -- ImageNet



So in summary, you saw how to get a feature extractor from a pre trained classifier by cutting the network and using weights from the layers before that output layer. And its most common to use that last pooling layer, but if you use an earlier layer, you can extract more permanent features. Such as vertical edges or natural looking patterns, using a classifier that was trained on large data sets such as ImageNet, which has millions of images. You can encode really meaningful information on your images as features. And then the following lectures you will learn about a popular classifier that uses ImageNet data and how it can be used to evaluate your GAN.

deeplearning.ai

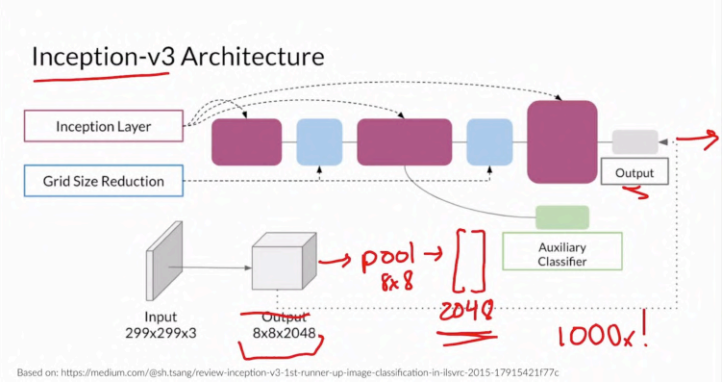
Inception-v3 and Embeddings

In this section you'll learn about the Inception-v3 Network, an intricate convolutional neural network classifier that can be trained on image.net.

Outline

- Inception-v3 architecture
- Comparing extracted feature embeddings

This section is about the Inception-v3 network, how to extract feature embeddings from it, and then compare those embeddings. This comparison can be used to evaluate GANs.

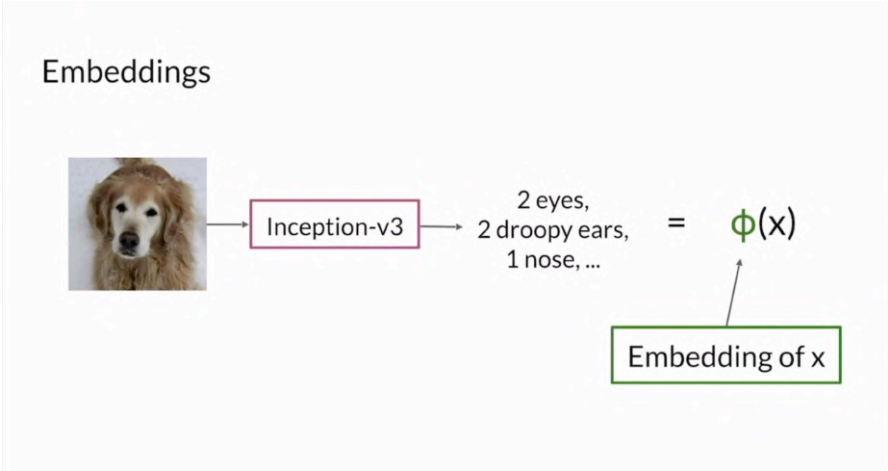


In the last section you learned that you can use a classifier as a feature extractor, specifically one that's been trained on the extensive image.net data set. The exact network you use can vary, but one of the most common ones who uses is Inception-v3, or Inception for short.

Inception is 42 layers deep, but amazingly cost and computationally-efficient, and has done well on classification tasks, as well as when particularly helpful as a feature extractor. And since you'll be using this as a feature extractor for comparing real versus generated images, I'll focus there.

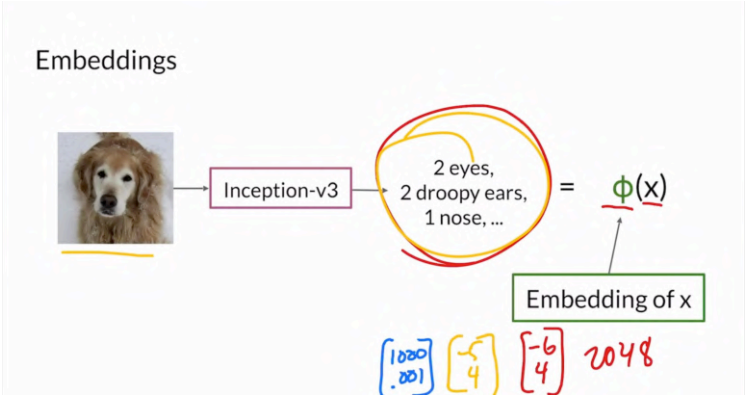
So this is a representation of the Inception-v3 network, and you can start by taking this classifier network, with the final fully connected layer for classification cutoff, and then using the last pooling layer. So classification is happening way out here off screen, and that's already been cut off, and here you see as output you get this 8x8x2048. This is actually not exactly the output, this is what you get out of your last convolutional layer. And then you put this into your last pooling layer with an 8x8 filter, and you get an embedding, a vector, of size 2048. And what's amazing is that you only get these 2048 values as your output, and this means that given an image it can condense the pixels of the image to just 2048 values to represent the salient features from that image. And I keep saying 2048 because it's really not a lot of values, compared to many images. Many images you see on the web are, let's say, 1024x1024 pixels with three channels for RGB color. Together, that's over 3 million pixel values, so the embedding size of 2048 is over 1000x smaller. That's 1000x fewer values needed to describe your image, so doing feature distance seems to be significantly better than pixel distance right now.

It's also useful that your feature extractor compresses information about your image, because that allows you to operate on fewer dimensions per image and will also greatly reduce the time it takes to compare a large number of images, which you will definitely be doing later on.



So using the Inception-v3 network trained to classify images on image.net, you can now extract features from your images to evaluate your gant. And again, that could be getting features like two eyes, two droopy ears, and one nose from this cute dog. And of course the actual features are a bit more abstract than these descriptions.

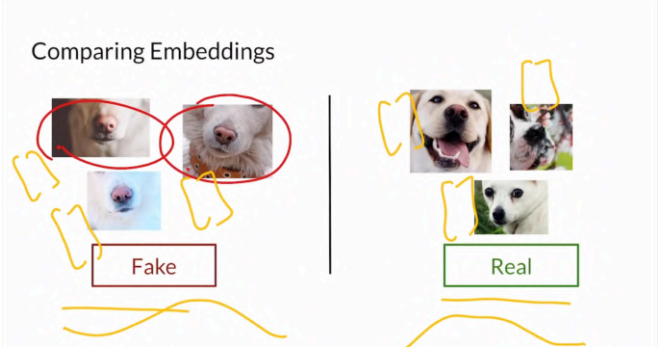
Notationally, this feature extraction model applies a function, or a mapping function, called phi, on an image X to extract its features. And X here can be a real or a fake image, and phi here is actually that inception network with the fully connected layer locked off.



To get these features of yours and to construct that embedding, which is just a vector of, again, 2048 values. And again, you learned that these extracted features are frequently called an embedding of an image, because they're condensed into this lower dimensional space and their placements in this lower dimensional space means something relative to each other. Where actually embeddings with similar features will be closer together, that is, take on similar values.

For example, if you had another dog coming in, that is fairly similar to this one, perhaps another golden retriever but it's in a different position or something, and you still extract these similar features, then perhaps its feature vector will be much closer to that original one. So it would have -5 and 4, and let's imagine the other one had -6 and 4. And so these two are fairly similar vectors.

Now, if you had an image coming in of a chair that looked very different with none of these features, then you would have a third feature vector that would be very far from these two. For example, 1000 here and .001. And I'm only showing two dimensions for these embeddings here, but remember they have 2048.



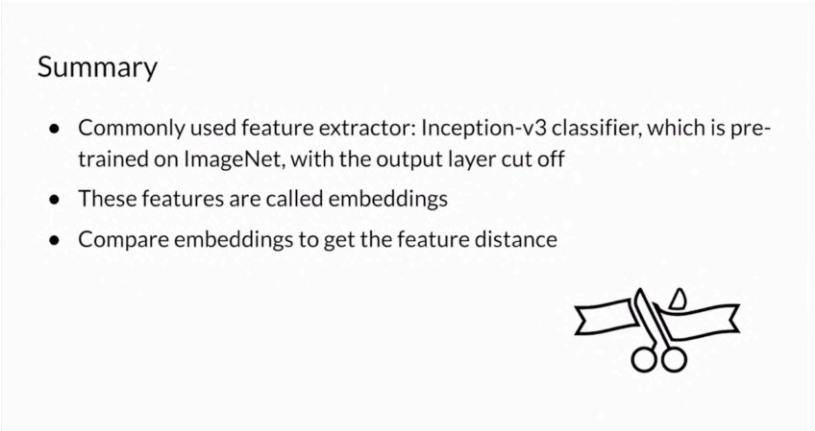
So for evaluating a GAN, the next step is to compare those embeddings, these extracted features, between reals and fakes. And that's typically several reels and several fakes, so you get a sufficient representation of images.

So let's say you have a few fake examples of dogs, and when you extract their features into an embedding you find that these embeddings, and these are pictorial representations of vector values, represent light colored dogs with pink noses. Meanwhile, the feature embeddings of your real images have also light colored dogs, but more black noses, and so comparing these images as features will be much more meaningful than comparing them as pixels.

And remember with simple pixel distance how a slight shift in pixels could actually make two images that are otherwise identical appear completely different? So these two fairly similar images based on their featured a sense would actually be very close because they're both light colored dogs, they both have pink noses. But in pixel distance they'd be really far apart, because you know this one pixel is very different from that one pixel, so they would be a world apart in that pixel distance.

So to get the feature distance, you could compare the features directly by subtracting them. So let's say you have vectors all around for all of these, and you can perhaps take the average of all the fake vectors, the average of all the reals, and subtract them. That's one way, and that's similar to how you did pixel distance.

Or you could get the Euclidean or cosine distance between various vectors. You could also consider that the set of reals and the set of fakes are some kind of distribution and see how far apart those distributions are. And in the next section, you'll learn how to calculate a feature distance between reals and fakes. That's a common method for evaluation, so stay tuned.



So now you know a fair amount about Inception as a classifier pre-trained on image.net. It can also be used as a feature extractor, by lopping off that final fully connected layer. And how those intermediate outputs from that last pooling layer can construct a feature embedding for your input image that can then be used to compare amongst different images.

Namely between real images and fake images, and getting a sense of how different they are in this feature space.

In this section, you'll learn about Frechet inception distance, or FID, the most popular metric for measuring the feature distance between real and generated images.

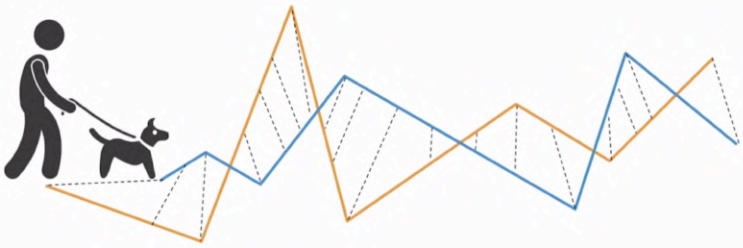
Outline

- Fréchet distance
- Evaluation method: Fréchet Inception Distance (FID)
- FID shortcomings

So first you learne about Frechet distance and then see how you can apply it to real and fake embeddings as Frechet inception distance, or FID, to see how far apart they are.

Finally, you'll learn about a few of FID's limitations. Evaluation is very much an open area in generative models research.

Fréchet Distance

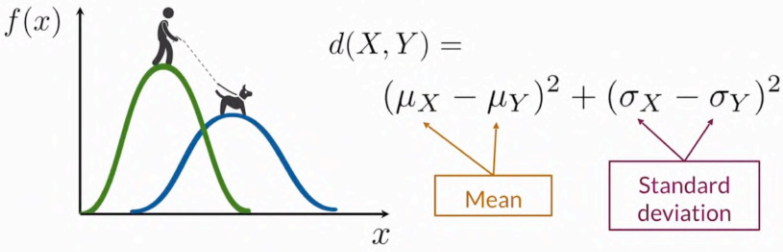


So Frechet distance named after the mathematician, Maurice Frechet is a distance metric that is used to measure the distance between curves and can be extended to comparing distributions as well.

The dog walker is a classic example used to illustrate Frechet distance, where the dog is on one curve and the walker is on the other. So the dog hears on blue and the walker here is on this orange. Each can go at their own speeds, but neither of them can go backwards. So keep that in mind. So to calculate the Frechet distance between these curves, you need to figure out the minimum leash length needed to walk the curves from beginning to end.

That is, what is the least amount of leash you can give your dog without ever having to give them more slack during the walk. So that's the intuition behind Frechet distance between two of these curves here.

Fréchet Distance Between Normal Distributions



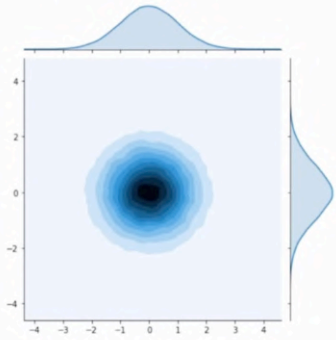
You can also calculate the Frechet distance between distributions. There are many distributions for which the Frechet distance between two distributions is analytically solved.

For example, there's a simple formula to calculate the Frechet distance between two single dimensional, normal distributions. Lets look at both of the distributions means, which is represented by Mu here, as well as their standard deviations represented by Sigma. So one distribution is X and one is Y. So you can imagine X and Y where X is the dog walker and Y is for the dog.

The mean gives you a sense of their center and the standard deviation gives you a sense of their spread.

Notationally, you can take the difference between the means and the difference between their standard deviations, and then square each of these differences to penalize values further away from each other and also as a notion of our distance in some way.

Multivariate Normal Distributions



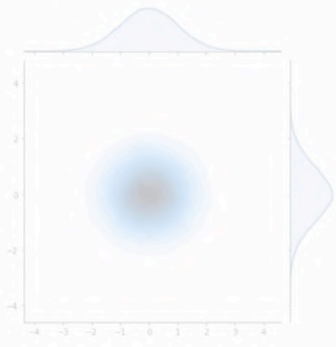
So keeping this in mind, let's take a quick detour to talk about multivariate normal distributions. Multivariate normal distributions generalize the idea of a normal distribution to higher dimensions. These extra dimensions allow you to model much more complex distributions in just one parameterize by a single mean and a single standard deviation.

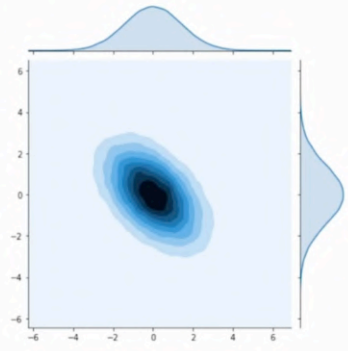
This is important because the features of your real data probably aren't distributed that nicely in just a single normal distribution. There are probably peaks in different areas, modes around certain features like a black nose for dogs or a tongue sticking out. A multivariate distribution is one to represent that. In a multivariate normal distribution has particularly nice properties. So it's often used.

So thus far, the distributions you've seen are univariate, meaning it's single-dimensional. So the easiest way to visualize a multivariate normal distribution is to have a univariate normal distribution for each dimension; here and here. These together construct this dark circle looking thing, which is actually pointing straight out at you in three dimensions, where the darker the values, the higher its altitude is.

So this resulting distribution illustrates how much each side varies based on the other, how much they affect each other. In here, both sides equally contribute to this center and they actually don't affect the other in doing so. So you have this nice round center, normal distribution peak coming out at you.

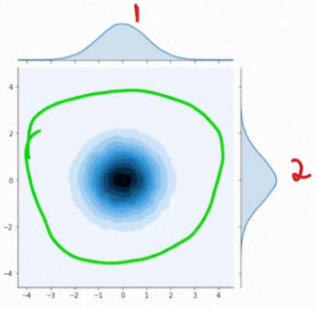
Multivariate Normal Distributions





But we can allow the dimensions to co-vary, meaning affect each other. This means that certain values in one dimension will cause values in another dimension to become more or less likely. So the resulting distribution could look more lopsided like this. Note that the normals here still look the same, even though they're affecting each other differently.

Multivariate Normal Distributions



σ^2 variance

$$\Sigma = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

Covariance matrix

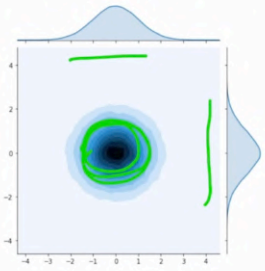
So in order to express that, you want to use a covariance matrix, which generalizes the idea of variants, which you've seen as Sigma squared, where Sigma on its own is the standard deviation and Sigma squared is the variance.

Remember that the variance quantifies the spread of this normal distribution. So covariance is aptly named because it measures the variance between two dimensions.

So along the diagonals of this covariance matrix, you see variants within a dimension. So for example, dimension one with dimension one and here dimension two with dimension two. Dimension one is over here, dimension two, it's over there.

But on the off diagonal you see these 0 values here. This 0 value measures dimension one to dimension two and this one is dimension 2 to dimension 1 and so what this is saying is that the ones here are telling you what the variance is within a dimension and it's just one and if these values were two for example, then you would see a much wider circle with much more spread and much lower peak because they've would be spread out much more.

Multivariate Normal Distributions



0's everywhere but the diagonal = all dimensions are *independent*

$$\Sigma = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

Covariance matrix

Now, a zero in any cell indicates no variance between those dimensions, so this covariance matrix, which has zeros in all of the off diagonal elements, means that the two dimensions are independent and again, if these values were different, say two or even if it's 0.5, then this peak would be even higher and the spread would be less but these two dimensions would still be independent.

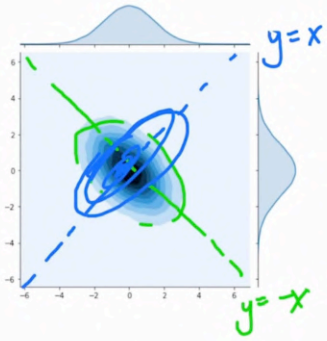
Technically here you don't need these covariance values assuming it's independent, to fully describe the spread of this distribution.

Multivariate Normal Distributions

Non-0's not on the diagonal = dimensions covary

$$\Sigma = \begin{pmatrix} 2 & +1 \\ +1 & 2 \end{pmatrix}$$

Covariance matrix



However, a covariance matrix with non zero off diagonal elements indicates covariance between the two dimensions and a negative value indicates a negative correlation and you can think of it as the direction here of how this correlation is going. This is along the diagonal y equals negative x into a positive value in this covariance matrix would actually mean the opposite here, like this, trending outwards towards you along y equals x instead and a value of 0.5 would concentrate the spread more around the center with less magnitude in the covariance while negative 0.5 would do the same in the opposite direction.

Multivariate Normal Fréchet Distance

Univariate Normal Fréchet Distance =

$$(\mu_X - \mu_Y)^2 + (\sigma_X - \sigma_Y)^2$$

Multivariate Normal Fréchet Distance =

$$\|\mu_X - \mu_Y\|^2 + \mathrm{Tr}\left(\Sigma_X + \Sigma_Y - 2\sqrt{\Sigma_X \Sigma_Y}\right)$$

Now that you have some intuition behind multivariate normal distributions, let's recall the formula for Frechet Distance on the univariate normal distribution and that was in single dimension space, so it's the difference between the means and the difference between the standard deviations squared to get notion of distance between two of these dimensions and you can actually generalize this to the multivariate case, which just means comparing between two multivariate normal distributions that you were just looking at and this is by using its covariance matrices, capital sigma here.

There are lots of parallels between the two formulas that can be generalized, now this might look daunting at first, so let's break that down.

Multivariate Normal Fréchet Distance

Univariate Normal Fréchet Distance =

$$(\mu_X - \mu_Y)^2 + (\sigma_X - \sigma_Y)^2$$

Multivariate Normal Fréchet Distance =

$$\|\mu_X - \mu_Y\|^2 + \mathrm{Tr}\left(\Sigma_X + \Sigma_Y - 2\sqrt{\Sigma_X \Sigma_Y}\right)$$

First, both formulas include the square of the distance between the means. The multivariate one simply takes the magnitude of the vector since it's no longer a single value and that's that norm here and so in the single dimension you can actually still write it like this, but it does evaluate to exactly this up here.

Multivariate Normal Fréchet Distance

Univariate Normal Fréchet Distance =

$$(\mu_X - \mu_Y)^2 + (\sigma_X - \sigma_Y)^2$$

$$(\sigma_X^2 + \sigma_Y^2 - 2\sigma_X\sigma_Y)$$

Multivariate Normal Fréchet Distance =

$$\|\mu_X - \mu_Y\|^2 + \mathrm{Tr}\left(\Sigma_X + \Sigma_Y - 2\sqrt{\Sigma_X \Sigma_Y}\right)$$

You can also expand the formula for the difference between those standard deviations in the univariate case, so if you expand this square here, you will get this term and you can see that you end up with something that is very similar to the multivariate formula, where Tr refers to the trace of a matrix, which is just the sum of its diagonal elements.

Multivariate Normal Fréchet Distance

Univariate Normal Fréchet Distance =

$$(\mu_X - \mu_Y)^2 + (\sigma_X^2 + \sigma_Y^2 - 2\sigma_X\sigma_Y)$$

Multivariate Normal Fréchet Distance =

$$\|\mu_X - \mu_Y\|^2 + \mathrm{Tr}\left(\Sigma_X + \Sigma_Y - 2\sqrt{\Sigma_X \Sigma_Y}\right)$$

$$\mathrm{Tr}\begin{pmatrix} 2 & -1 \\ -1 & 2 \end{pmatrix} = 4$$

For example, two, negative one, negative one, 2 that you saw before, the trace would then only look at these elements and take the sum of them. The trace of this would be four and note that the sum of the diagonal elements of say, sigma X would actually just be the sum of the variances, i.e the covariance that each dimension has with itself not with the other dimension. The covariance between a dimension in another dimension isn't really considered here when the trace is being used on these guys.

Also, everything seems squared in the univariate case. You have this square here and this is squared relative to this square root and that's because sigma is actually the standard deviation and sigma squared is the variance but this big sigma down here is the covariance and its generalization of sigma squared, the variance, so they do match and finally to be extra clear, this matrix square root down here gets the square root of the matrix not each individual element.

This multivariate normal Frechet inception distance, what you need to take away here is just that it's very similar and its very much just a generalization of the univariate case by looking at differences between both the centers of those distributions and means, as well as the spreads of their distributions, the variances or covariances in this case.

Fréchet Inception Distance (FID)

$$\text{FID} = \|\mu_X - \mu_Y\|^2 + \text{Tr}\left(\Sigma_X + \Sigma_Y - 2\sqrt{\Sigma_X \Sigma_Y}\right)$$

Real and fake embeddings are two **multivariate** normal distributions

X Real embeddings [] ×50,000 vs Y Fake embeddings [] ×50,000

Now, how is this useful? The multivariate normal distribution can approximately model the many modes in your image features, from your real image feature embeddings, you can construct a multivariate normal distribution and this will be based on many, say, 50,000 real embedding vectors looking at how they're distributed and fitting in multinomial distribution to them, which just means finding a mean and covariance matrix across all of these examples and so now you do the same for your fakes, you get their feature embeddings. You fit a multivariate normal distribution on it and then now you can compare the two multivariate normal distributions. The reals, which I'll call X, the fakes, which I'll call Y, and that just corresponds to these up here, where Mu X is the mean of the real embeddings, Mu Y is the mean of the fake embeddings, Sigma X is the covariance matrix of the real embeddings, and Sigma Y is the covariance matrix of the fake embeddings, which are then again used here.

Now, between the reals and the fakes, you can first apply embeddings to those images, fit a multivariate normal distribution to each, and then compute the FID score on them. This is called Frechet Inception Distance or FID and it's currently the most widely used GAN evaluation metric. You understood if Frechet distance and inception just refers to using the Inception-v3 network to extract those features, those embeddings for reals and for fakes.

Stepping back, FID looks at the statistics of the real multivariate normal distribution and the statistics of the fake multivariate normal distribution, where the statistics are just mean and covariance matrices, and calculates a how far apart those statistics are from each other. Where the closer those statistics are to each other, the closer the fake embeddings model the real embeddings.

Now, FID does assume that the embeddings take on a multivariate normal distribution. Not because it's always perfectly accurate, but mainly because it's approximate and easy to compute.

Fréchet Inception Distance (FID)

Lower FID = closer distributions

$$\text{FID} = \|\mu_X - \mu_Y\|^2 + \text{Tr}\left(\Sigma_X + \Sigma_Y - 2\sqrt{\Sigma_X \Sigma_Y}\right)$$

Real and fake embeddings are two **multivariate** normal distributions

Use *large sample size* to reduce noise

When all is computed here, you're left with a number that represents the difference between the real and the fake distributions. If the fake is close to the real, meaning your model is generating fake outputs close to the real, the lower this difference this number is.

A smaller value actually means that features in the reals and fakes are more similar to each other, so that means lower the FID, the better.

This can sometimes trip people up because other evaluation metrics lean towards higher values being better, but now for FID, lower is better because it's how far apart fakes are from reals.

Unfortunately, there isn't a super interpretable range for FID values. Wouldn't it be great if the values were between zero and one, and 0.5 meant half or fake? That's not the case for FID. But generally, the closer you are to zero, the better. Because at zero, your fakes are indistinguishable from the reals based on those statistics computed from many of their embeddings.

To give a sense of the scale of this computation, recall that there are typically 2048 dimensions in a feature vector or embedding from the inception network. FID typically is calculated over a large number of samples, say at least 50,000 reals and 50,000 fakes. Using a large number of samples reduces the noise and selection bias you'd have if comparing between say, just a couple of samples.

Shortcomings of FID

- Uses pre-trained Inception model, which may not capture all features

Additionally, the ImageNet pre-trained Inception-v3 model doesn't always get the features you want, or they don't always make sense for your generators task.

For example, if your GAN is trained on MNIST to generate handwritten digits, the inception-v3 model might not be able to detect meaningful features since ImageNet is composed of natural photos like dogs and humans, and quite frankly, a lot of dogs.

Shortcomings of FID

- Uses pre-trained Inception model, which may not capture all features
- Needs a large sample size

Also, as stated before, the sample size needs to be large for both sufficient coverage of samples and your FID score is biased. Which means that your FID score will change based on the number of samples you use, which isn't great because you're GAN isn't changing nor are the real samples, so the number of samples shouldn't impact the score. But lone and behold, FID is typically lower for larger sample sizes. The more simple you use, the better your GAN will seem to be, even if it actually isn't better, because you're using the same model to compare.

This isn't a good property to have in a metric, but there's certainly research trying to overcome this, creating an unbiased FID, for example, but that method isn't widely used. The large sample size also leads the FID being slow to run.

Shortcomings of FID

- Uses pre-trained Inception model, which may not capture all features
- Needs a large sample size
- Slow to run
- Limited statistics used: only mean and covariance

Lastly, there are a limited number of statistics gathered on those distributions. Only mean and covariance, which are the first few moments of a distribution or properties of that distribution. There are many other moments of a distribution, like skew and kurtosis that you might be familiar with.

Just remember that mean and covariance don't cover all aspects of the distribution, and the distribution also is likely not exactly multivariate normal. We're largely getting just mean and covariance, because we're assuming that the distribution is multivariate normal, but the distribution of your samples also is likely not exactly multivariate normal, but that assumption is made to make the computation of statistics and comparisons much easier.

These shortcomings unfortunately mean that you still need to qualitatively look over your samples to find the model you want and to debug your model. The problem is that existing evaluation metrics are not exactly the best indicator of progress now. Nevertheless, FID is still one of the most popular ways to evaluate your GAN because it's easy to implement and use.

Summary

- FID calculates the difference between reals and fakes
- FID uses the Inception model and multivariate normal Fréchet Distance
- Sample size needs to be large for FID to work well

In summary, you learned how FID calculates the difference between the real and fake embeddings by using Frechet distance. There are a few shortcomings to FID that make it, so you should still be babysitting your model and checking out it's samples instead of just comparing FID scores between saved model checkpoints during training, to decide which one is best.

In this section, you'll learn about another way to calculate the distance between reals and fakes using the Inception v3 model. This method was developed before FID and used to be widely used but now has been largely replaced by FID and is by enlarge more relevant to conditional generation.

Outline

- Another evaluation metric: Inception Score (IS)
  - Intuition, notation, shortcomings

That said, the inception score which is what it's called, is reported in many paper. It's so good to know what it's measuring and how it differs from FID.

Inception Model Classification

0.30 Cat

0.60 Dog

0.10 Bird

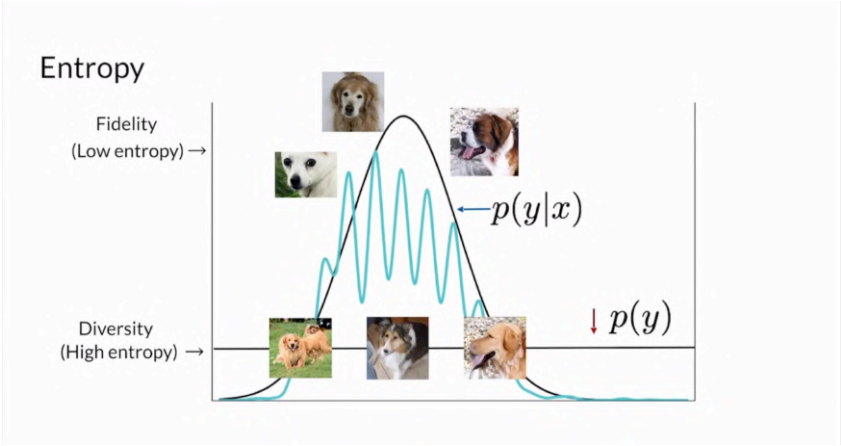
Fake

Remember the Inception v3 classifier pre-trained on ImageNet that was used for feature extraction, the inception score also uses this model. However, unlike FID, you use the classifier as is not as a feature extractor. You keep the classifier intact and don't use any of those intermediate outputs.

First you input a generated image and see what the inception v3 classifier says it's classified as. For example, maybe this image looks like it's 0.6 dog, 0.3 cat, and 0.1 bird. Intuitively a high value on one class, for example, a dog here with low values on the others suggests that this image clearly resembles one class over the others. That is, it is arguably high fidelity. You can observe this value for many samples to get a sense of how well overall this gain is able to produce clear objects based of course on the classifiers judgment. If this dog were 0.9, then this would do even better. It would have an even clearer dog, arguably.

Now because you want your gain also to produce diverse outputs, then looking across many samples, you would expect that many different classes would be generated. You would expect across all your samples that were trained on all of these classes, perhaps that dog appears and cat appears and bird also appears, and it's not just dog each time.

Again, this can be observed by using this classifier to then see what this distribution is. You'd want this diversity probability distribution over all the different classes to be nicely spread out and not spiky on a single class.

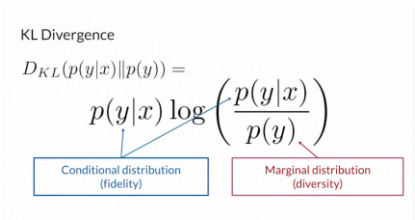


More formally, the first measure which looks at the distribution of classes given an image, is measuring the distribution p of y given x, where given your image x, what is the distribution over those classes given that classifier?

With images that have clear high fidelity generated objects, that distribution should have high pets with high probability are a few select classes, or even just a single class, and here maybe few select classes with overall very low probability on the remaining classes. What this is trying to approximate or signal is that there are clear objects being picked out in the generated images by your classifier.

This is known as having low entropy because the probability distribution is clustered at those select points and not scattered all about. Entropy in a sense is randomness, and this fidelity measurement corresponds to low entropy.

Now on the diversity front, you want lots of classes to be generated across your many samples. The distribution of classes across your samples should be high entropy, meaning it's not concentrated on a particular or a couple particular classes, which would be if you put your thinking cap on reminiscent of mode collapse because that means you're only a generating one type of thing and nothing else. This distribution is also known as the marginal label distribution or p of y. This is the distribution over all of your labels, all labels across your entire dataset or across a large sample. This might be confusing because you have high down here and you have low up here. But just keep in mind that these two are supposed to be very different, that is what's key here.



With this notion of fidelity being low entropy and of diversity being high entropy, you could combine their measurements into a SQL score, which is convenient because then you can score your model and say I'm getting better or I'm getting worse by getting a distance between that diversity are high-value minus fidelity, a low value.

The Inception scores specifically uses something called the KL divergence from p of y to p of y given x. KL divergence essentially tries to measure how much information you can get or gain on p of y given x, given just p of y. If you only have information on p of y, and so let's say p of y is really uniform on your different classes, it's just as uniform distribution, then what would you guess for your p of y given x? That would be very difficult. Meanwhile, if your p of y really spiky, then what would you guess for your p of y given X, you'd probably guess it was spiky around this area too. If p of y essentially isn't very informative like in this high entropy case because it's a lot of randomness and is all even across the classes, then you can't get much information on p of y given x, you've no idea what class it's from.

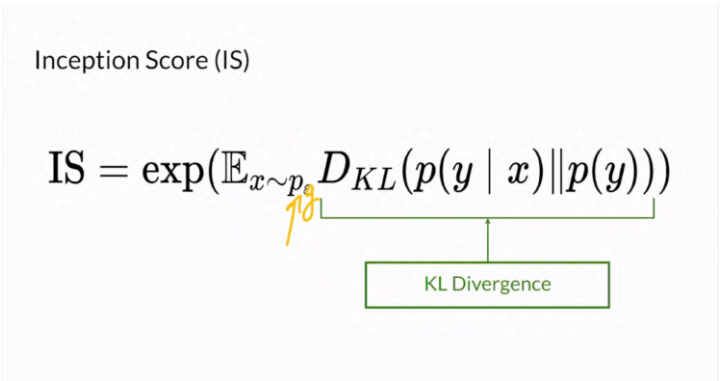
However, if p of y is full of peaks or full of one peak right here, then you probably have a clue of what p of y given x might be. A good guess at least probably that peak there. At the same time, if p of y given x is full of peaks, it's harder to guess at that exact distribution than if you're just guessing, It's even across all of the classes. It's easier to then guess for this guy that it is even across all the classes.

Now, that's some very basic intuition and no worries if you don't completely understand KL Divergence? For the purposes of understanding Inception Score, you can think of KL Divergence here as approximately getting how different the conditional label distribution for fidelity is, from the marginal label distribution for diversity, which is their relative entropy from one another.

There's a reason actually why it's not called KL distance, because KL Divergence isn't equal in the opposite direction. So with these two terms flipped, it's not actually an equal value, but it's a very close notion to that sense.

As a result, you'd expect to see a high KL Divergence, when your distributions are far apart, which is what you want. You want one to be high, you want the other to be low. In other words, basically, when fake images each have a distinct label and there's also a diverse range of labels among those fakes.

Notationally, it expands to this out here and feel free to dive more into KL Divergence outside of this course, it's a generally very useful concept for machine learning and is a core information theoretic component behind evaluating models against ground-truth. But essentially, it does evaluate to looking at the conditional distribution here and also the marginal distribution here.



The Inception Score then sums over all the images and averages over all the classes. You see this p of epsilon, perhaps better to say p of g, is samples from your generator and then at the end, there is an exponent. This is the same as doing e to the, this huge term up here and this exponent isn't really for calculating this value or important for KL Divergence in any way. It's actually just to give a nice human-readable score, like 100 for Inception Score and not 0.000001, which is not as useful and a very possible output of this inner term.

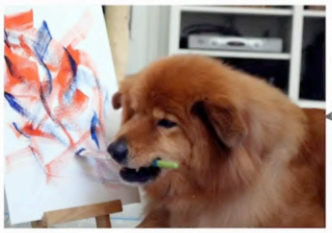
So mathematically the lowest possible value is 0 and the highest is infinity. But during implementation in this case, the lowest possible Inception Score will be 1 and the highest or best possible score can be the number of classes present or 1,000 in this case, because the Inception-v3 classifier is trained on ImageNet with a 1,000 classes.

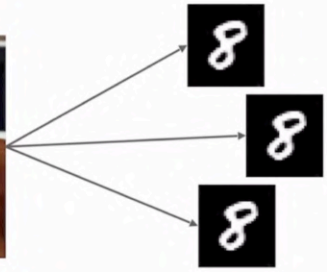
Now, higher the score the better here for Inception Score and that means the entropy of the conditional probability distribution is low, finding relevant objects and features, while the marginal probability distribution is high, finding a diverse set of features.

So typically when this score is low or bad, it's because both distributions either have low entropy, having high peaks, so no diversity or both distributions have high entropy with no clear objects found.

Shortcomings of IS

- Can be exploited or gamed
  - Generate one realistic image of each class





However, as you might have guessed, there are a lot of shortcomings with using the Inception Score.

First, the metric is easy to exploit or game on the diversity front, because if your GAN generates one single real image for each classifier class, so a 1000 images for each of the ImageNet classes then it actually gets a perfect score. But ideally, your GAN can do better than generate one image for each class. I'd like two golden retrievers, please. This is definitely a form of mode collapse, I can go undetected by Inception Score.

Shortcomings of IS

- Can be exploited or gamed
  - Generate one realistic image of each class
- Only looks at fake images
  - No comparison to real images

$$p(y|x) \qquad p(y)$$

Another big issue is that the Inception Score only looks at the generated samples. You might have noticed this, that I never mentioned the real samples here and it doesn't compare the generated samples to the real images. These proxy statistics might be a bit far off and idealistic and it's also dependent on the classifiers tasks and abilities.

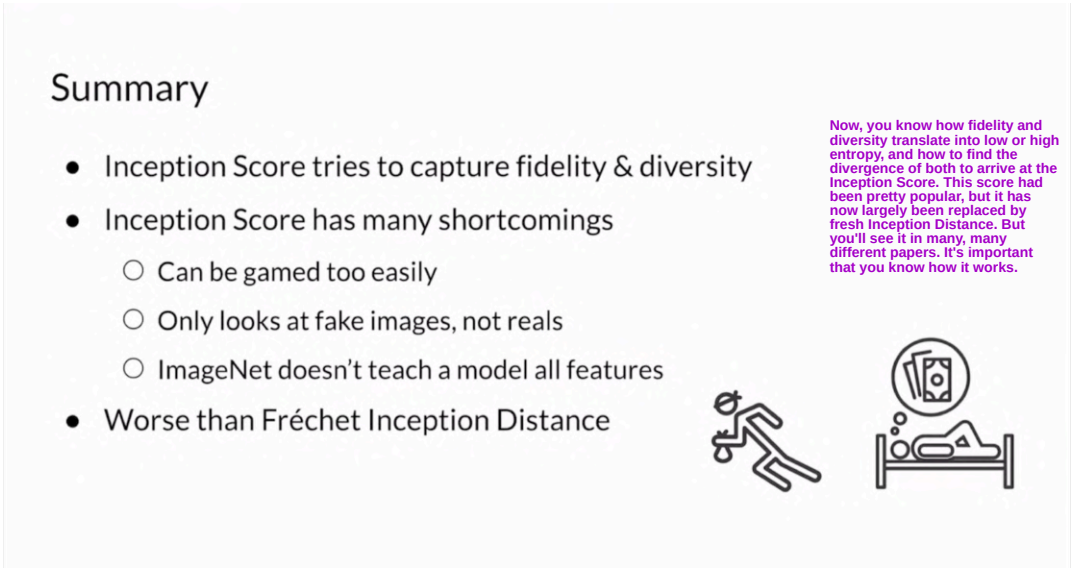
Shortcomings of IS

- Can be exploited or gamed
  - Generate one realistic image of each class
- Only looks at fake images
  - No comparison to real images
- Can miss useful features
  - ImageNet isn't everything

Finally, since the classifier is trained on ImageNet, the values and scores might be very imprecise in a way that's similar to FID's shortcoming.

For example, what if the generated images have a lot of objects in it? You'd expect high entropy in the different classes, in the classifier outputs such as a bedroom or office full of different items. Because it picks up on many different classes in that one image or what if there are a lot of features that aren't detected by the classifier because they're not relevant, such as a generator that only generates profiles of faces, when the classifiers trained for mostly dog breeds and other objects or even a generator that generates human faces with features in the wrong place. But that looks real because components of the face are clearly found by your classifier.

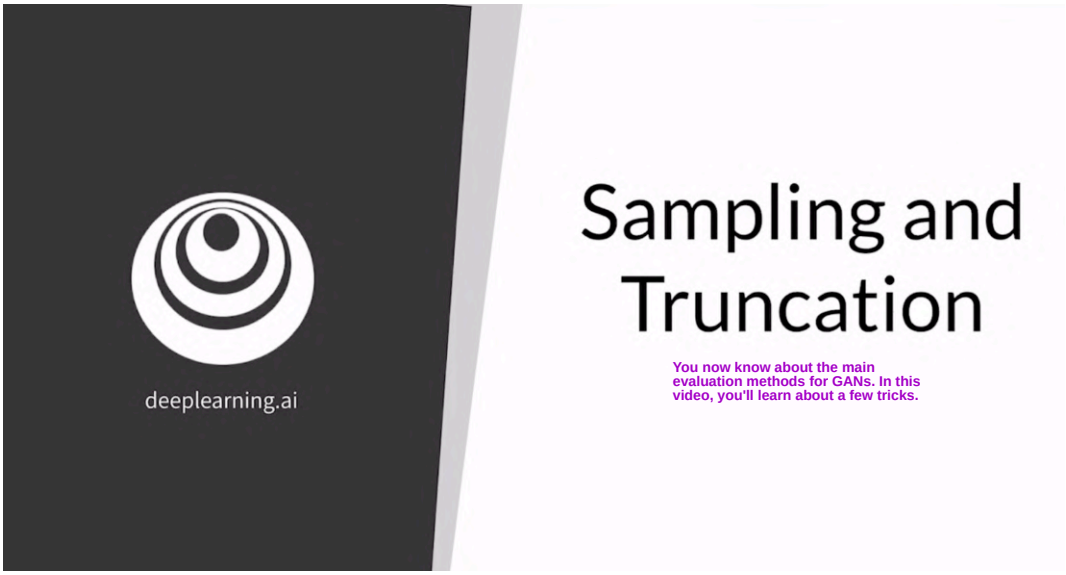
So spatial relationships in particular can pose a problem for the Inception Score. It only really makes sense when the training data-set is close to ImageNet and this is FID's shortcoming too. Despite all this, Inception Score still remains as one of the most widely used metrics for GANs after FID and is relevant mainly to conditional GANs that expect to clear classes.



(Optional) A Closer Look at Inception Score

Want to know more about why Fréchet Inception Distance has overtaken Inception Score? This paper illustrates the problems with using Inception Score.

A Note on the Inception Score (Barratt and Sharma, 2018): https://arxiv.org/abs/1801.01973

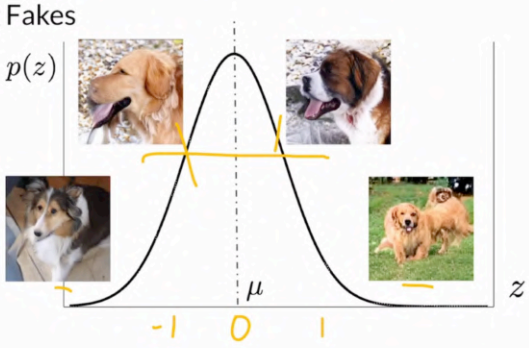


Outline

- Sampling reals vs. fakes
- The truncation trick
- HYPE!

In evaluating GANs, the statistics about the real versus the fake data sets matter in terms of sampling. You'll see how they differ and I'll discuss a trick you can do after training to shift towards either greater fidelity or diversity. Also get hyped because I'll be talking about the recently developed center for human evaluation referred to as HYPE.

Sampling Fakes

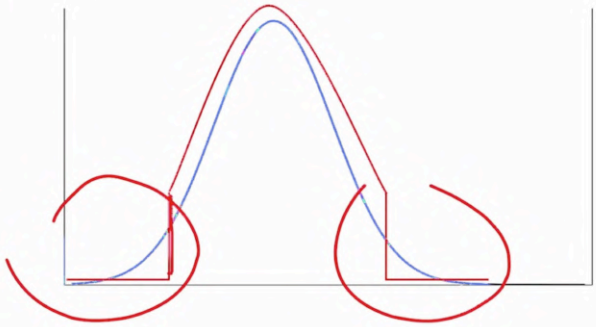


In addition to sample sizes, how do you select which images to sample when evaluating with FID? For reals you can just sample randomly in a uniform way but for fakes what's typically done is to sample z values based on the training distribution of z values or the prior distribution of your noise vectors p of z. Since you'll usually train your GAN with noise vectors using a normal prior, meaning a z that's selected from a normal distribution like the one you see here where it's centered around a Mu of zero and has a standard deviation of one. Vector values that are closer to the zero value will occur more than those further away during training.

As a result, when you sample using those values close to zero your resulting image would actually look pretty good but that's only fidelity because this also will occur at a loss of diversity. As you can see here, these two dogs look pretty good but they're less diverse and then the ones that are here might look a little bit more funky. You can adopt a fidelity and diversity by choosing where to sample. Sampling in the middle will get you more normal looking things. For better or worse, your sampling technique becomes actually an important aspect of evaluation and downstream use.

Because your evaluation metrics like FID or inception score operate on the samples of your model, not your model parameters. That means that evaluation on a GAN is very much sample dependent, so this does matter.

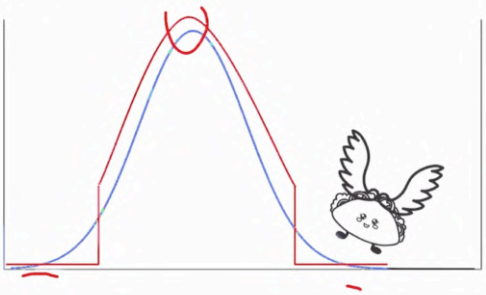
Truncation Trick



Following the observation on fidelity versus diversity, there's a really neat sampling trick that's used called the truncation trick. The truncation trick is exactly a trick because it's done after the model has been trained and it broadly trades off fidelity and diversity. What it actually does is truncate this normal distribution that you see in blue which is where you sample your noise vector from during training into this red looking curve by chopping off the tail ends here.

This means that you will not sample at these values out here past this certain hyperparameter and this hyperparameter determines how much of your tail you'll keep, so you can have one truncate up here. You can have one truncate out here so that you keep more of the curve

Truncation Trick



So if you wanted a higher fidelity which is roughly the quality and realism of your images, you want to sample around zero and truncate a larger part of your tails. However, this will decrease the amount of diversity in your generated images because this is where your funky flying tacos are generated and other weirder stuff that hadn't gotten much discriminator feedback during training. If you want greater diversity, then you want to sample more from the tails of your distribution and have a lower truncation value, a truncation hyperparameter value.

However, the fidelity of these images will also be lower because your generator isn't accustomed to getting its weights to make that noise vector into a beautiful realistic image. That is, it didn't get nearly as much feedback on its realism on those noise vectors sampled from these regions during training. Finally, you can definitely train your models on a different prior noise distribution from which you sample your noise vectors such as the uniform distribution where there is no concentration of possible values like up here. But the normal distribution is pretty popular in part because you can then use the truncation trick which is shown here to tune for the exact fidelity and diversity trade off you want. There really hasn't been a stark difference when people have experimented with different prior noise distributions.

As expected, a model's FID score will be higher which means it's worse when there's a lack of diversity or fidelity. The samples might not do well on FID when using the truncation trick, though using the truncation trick might conform to what you want in an application you're looking to apply your GANs to. Something downstream where you need higher fidelity images and you don't want the extra gunk.

HYPE and Human Evaluation

- Crowdsourced evaluation from Amazon Mechanical Turk
- $\text{HYPE}_{\text{time}}$ measures time-limited perceptual thresholds
- HYPE_{∞} measures error rate on a percentage of images
- Ultimately, evaluation depends on the type of downstream task

Available from: https://arxiv.org/abs/1904.01121

Speaking of using those truncated samples because they seem better to the human eye instead of FID, using people to evaluate an eyeball samples is still a huge part of evaluation and often an important part of the process of developing a GAN. What's cool is that there are methods that systematically evaluate the quality based on principled crowd sourcing and perception tasks.

In fact, one of the recently developed centers for GAN fidelity is one that I had invented in 2019 with other researchers from Stanford. Its name is a bit cheeky called HYPE for human eYe perceptual evaluation of generative models. HYPE displays a series of images one-by-one to crowd source evaluators on Amazon Mechanical Turk, it asks the evaluators to assess whether each image is real or fake

HYPE and Human Evaluation

- Crowdsourced evaluation from Amazon Mechanical Turk
- $\text{HYPE}_{\text{time}}$ measures time-limited perceptual thresholds
- HYPE_{∞} measures error rate on a percentage of images
- Ultimately, evaluation depends on the type of downstream task

Available from: https://arxiv.org/abs/1904.01121

One version of HYPE called HYPE time actually flashes images at you for different amounts of milliseconds to see at what threshold you can figure out an image is real or fake. The better your generative model is, the more time a human needs to decide whether that image is fake.

HYPE and Human Evaluation

- Crowdsourced evaluation from Amazon Mechanical Turk
- $HYPE_{time}$ measures time-limited perceptual thresholds
- $HYPE_{\infty}$ measures error rate on a percentage of images
- Ultimately, evaluation depends on the type of downstream task

Available from: https://arxiv.org/abs/1904.01121

HYPE infinity takes away that time threshold and actually it's just looking through images. But of course HYPE is predicated around having great quality control and being able to manage learning effects as an evaluator gets better and better at evaluating fake versus real images over time. Despite all this, ultimately evaluation will depend on the downstream task you want.

For example, if your GAN was to generate X-rays that had pneumonia in them, you want to make sure that your doctor would agree that there actually is pneumonia in there and maybe not an Amazon Mechanical Turk who doesn't have a doctor's degree or you probably don't want an ImageNet pre-trained classifier to extract features and to tell you whether or not that's close or not to a pneumonia X-ray image before you claim that your fake image is producing pneumonia on X-rays.

Summary

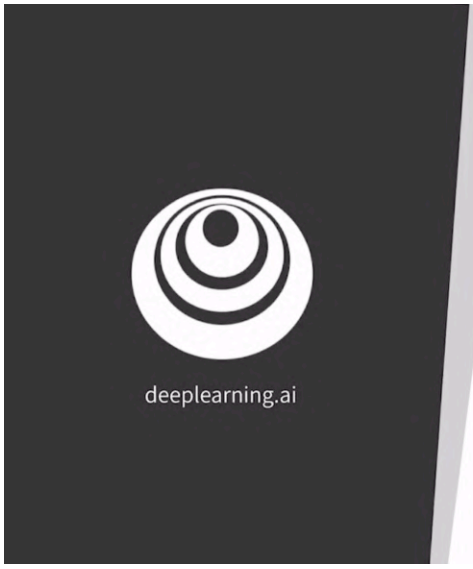
- Fakes are sampled using the training or prior distribution of z
- Truncate more for higher fidelity, lower diversity
- Human evaluation is still necessary for sampling

Now you know how fakes are sampled by using the prior distribution of z during training. During testing or inference time you also have a new trick up your sleeve, the truncation trick which allows you to truncate the tails of your sampling distribution a little more or a little less depending on whether you're interested in higher fidelity or higher diversity. As evident in the sampled images, automated evaluation metrics still don't capture exactly what we want but are a good approximation. That's why human perceptual evaluation still sets the benchmark and gold standard and it's also a quick way to evaluate images.

(Optional) HYPE!!

Intrigued about human evaluation and HYPE (Human eYe Perceptual Evaluation) of GANs? Learn more about this human benchmark in the paper! You may notice a familiar name among the authors ;)

HYPE: A Benchmark for Human eYe Perceptual Evaluation of Generative Models (Zhou et al., 2019):
https://arxiv.org/abs/1904.01121



Precision and Recall

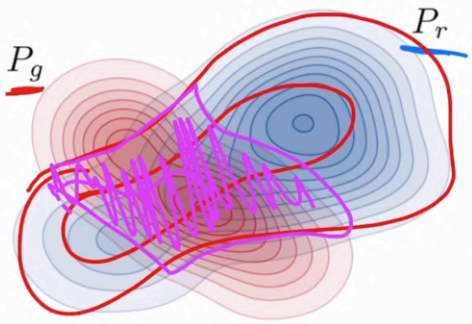
In the final part of this week's material, you'll learn about an evaluation concept you can use for GANs.

Outline

- Precision and recall in GANs evaluation
- Relating precision and recall to fidelity and diversity

Specifically, you'll learn about precision and recall in the context of GAN evaluation, though this will be applicable to all generative models and you'll gain some intuition for how they relate to fidelity and diversity.

Precision and Recall

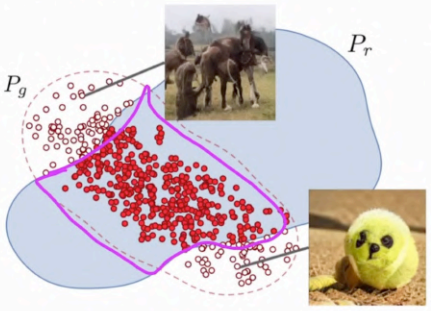


Additional evaluation metrics have recently emerged and few of them use a notion of precision and recall that I find particularly noteworthy and interesting, especially to break down fidelity and diversity into more familiar machine learning topics. If you happen to be familiar with the notion of precision and recall and classifiers, this is a cool extension to generative models such as GANs.

Imagine a space of all reals, the true distribution Pr here, where P stands for the probability distribution, r stands for reals. Over here in red, Pg is all of what the generator can generate, the fake distribution. The best thing that can happen for your GAN is for Pg to completely overlap Pr in every single way where there's no Venn diagram business or concentric circles. You don't even want to be a subset of Pr, you want to be completely Pr. Essentially for your red Pg, your generated distribution to be completely like Pr, your real distribution. You might know that there is this important intersection area and both precision and recall will be considering this intersection area.

Available at: https://arxiv.org/pdf/1904.06991.pdf

Precision



- Relates to fidelity
- Looks at overlap between reals and fakes, over how much extra gunk the generator produces (non-overlap red)

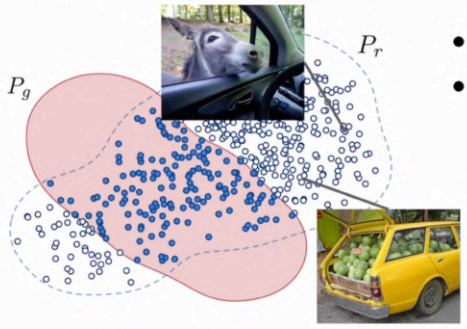
$$\frac{\text{overlap fakes}}{\text{all fakes}}$$

Diagram available at: https://arxiv.org/abs/1904.06991; Tennis dog available at: https://arxiv.org/abs/1809.11096

First precision, focus on the points within the dotted line. All of these points are generated samples and the ones that are filled in are generated samples that overlap with the real distribution, the blue and the ones that are not filled in are generated samples that don't overlap with the real distribution. Precision measures this intersection area, specifically, the fake examples that intersect with the reals divided by this whole dotted line area of all the possible things that generator can generate. Intuitively, this means you overlap where the generated images look pretty real because it's overlapping the real space divided by this entire space, which not only includes his real self, but also includes some samples that look very much fake like this tennis ball dog or I'm not even sure what that is, some horse thing, so just pure gunk. You don't want this extra gunk, so the more stuff you have out here, the worse your precision will be because your precision will be these overlap points divided by all fakes.

Your precision is essentially the fakes that look real over all of your fakes. You want your denominator to be as close as possible to that overlap. Perfect precision means that everything you generate will look real. But, that doesn't mean you cover this entire real thing. You can be a subset in here. Precision relates to fidelity because there's a notion of this extra fake stuff that's modeled but is unnecessary. The better your precision is, the higher the quality of all your generated samples are. The truncation trick can help with precision reducing your weird stuff. But it could also hurt recall too, because it could shrink all this generated part into something much smaller.

Recall



- Relates to diversity
- Looks at overlap between reals and fakes, over all the reals that the generator cannot model (non-overlap blue)

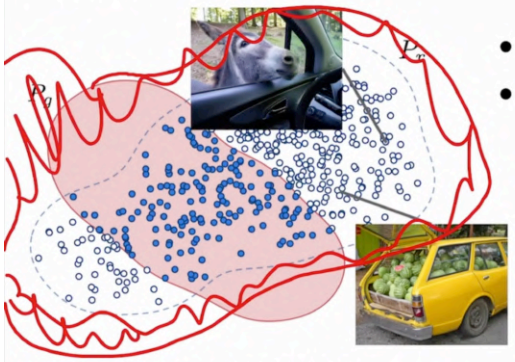
$$\frac{\text{overlap of reals}}{\text{all reals}}$$

Diagram available at: https://arxiv.org/pdf/1904.06991.pdf

On that topic of recall, recall on the other hand, is how much overlap there is divided by all of the real sample. Essentially, the mirror of precision. That includes the stuff that the generator is missing and isn't able to model. These points out here, which are real samples, but perhaps the generator was not able to model them. They look like pretty weird real samples too with this donkey looking in and this car full of watermelons. It's basically a measure of how well the generator can model all the reals and it ignores all that extra gunk that we were seeing before and that's not used in the measure of recall here.

Recall relates to diversity, because you can see if the generator models all the variation in the reals or not. Just to compare with precision, here it overlap of reals over all reals. Recall is trying to get a sense of how well the generator is able to model all possible real images. Models I've seen tend to be pretty good at recall meaning somewhere at some value of Z of that noise vector, they can actually generate every single image and the real dataset and perhaps more. The whole real distribution of all possible dogs or faces or whatever it is trying to model that is in your real dataset at the very least. But typically, these models, especially when they're large, produce a lot of extra gunk in the parameters that didn't get a lot of feedback from the discriminator because there are just too many parameters in the model, so some parts of it will inevitably produce gunk because it's already been able to model all the reals and doesn't need a cutback. What that'll look like is Pg looking more like this, being a superset of the reals. The state of the art models can often be bad at precision as opposed to recall.

Recall

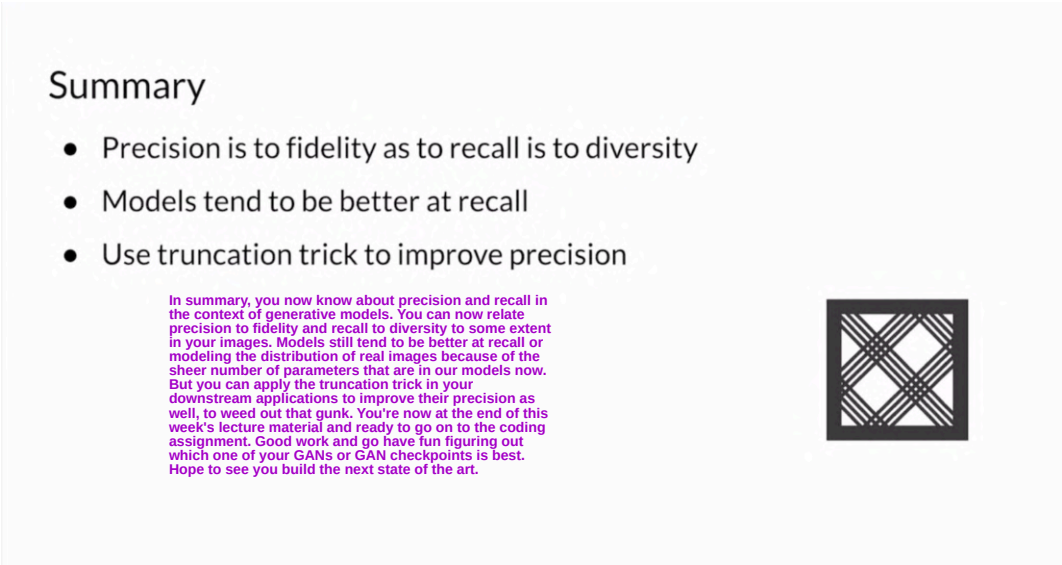


- Relates to diversity
- Looks at overlap between reals and fakes, over all the reals that the generator cannot model (non-overlap blue)

$$\frac{\text{overlap of reals}}{\text{all reals}}$$

Diagram available at: https://arxiv.org/pdf/1904.06991.pdf

This is why the truncation trick can come in handy for downstream applications to weed out all that extra gunk, which is everything outside of the dotted line then.



(Optional) More on Precision and Recall

Using precision and recall metrics on GANs pique your interest? See some specific methods in this paper! Note that you will learn all about StyleGAN in Week 3 of this course.

Improved Precision and Recall Metric for Assessing Generative Models (Kynkäänniemi, Karras, Laine, Lehtinen, and Aila, 2019): https://arxiv.org/abs/1904.06991

(Optional) Recap of FID and IS

Need a summary of FID and IS? Here are two great articles that recap both metrics!

Fréchet Inception Distance (Jean, 2018): https://nealjean.com/ml/frechet-inception-distance/

GAN — How to measure GAN performance? (Hui, 2018): https://medium.com/@jonathan_hui/gan-how-to-measure-gan-performance-64b988c47732

Works Cited

All of the resources cited in Course 2 Week 1, 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:

- StyleGAN - Official TensorFlow Implementation: https://github.com/NVlabs/stylegan
- Stanford Vision Lab: http://vision.stanford.edu/
- Review: Inception-v3 — 1st Runner Up (Image Classification) in ILSVRC 2015 (Tsang, 2018): https://medium.com/@sh.tsang/review-inception-v3-1st-runner-up-image-classification-in-ilsvrc-2015-17915421f77c
- HYPE: A Benchmark for Human eYe Perceptual Evaluation of Generative Models (Zhou et al., 2019): https://arxiv.org/abs/1904.01121
- Improved Precision and Recall Metric for Assessing Generative Models (Kynkäänniemi, Karras, Laine, Lehtinen, and Aila, 2019): https://arxiv.org/abs/1904.06991
- Large Scale GAN Training for High Fidelity Natural Image Synthesis (Brock, Donahue, and Simonyan, 2019): https://arxiv.org/abs/1809.11096

From the notebook:

- CelebFaces Attributes Dataset (CelebA): http://mmlab.ie.cuhk.edu.hk/projects/CelebA.html
- ImageNet: http://www.image-net.org/
- The Fréchet Distance between Multivariate Normal Distributions (Dowson and Landau, 1982): https://core.ac.uk/reader/82269844