Chapter - 3

**Computer Vision**

**GAN: Image Creation**

The idea behind GANs, How do GANs work

Applications of GANs

**3.1 Introduction**

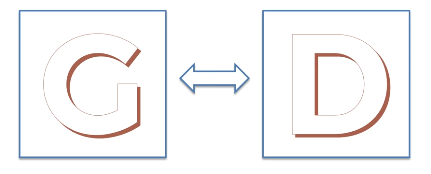
In this chapter we will learn about Generative Adversarial Network.

* What we will learn in this chapter:
  + - * + The Idea Behind GANs
        + How Do GANs Work (Step 1, 2, 3)
        + Applications of GANs

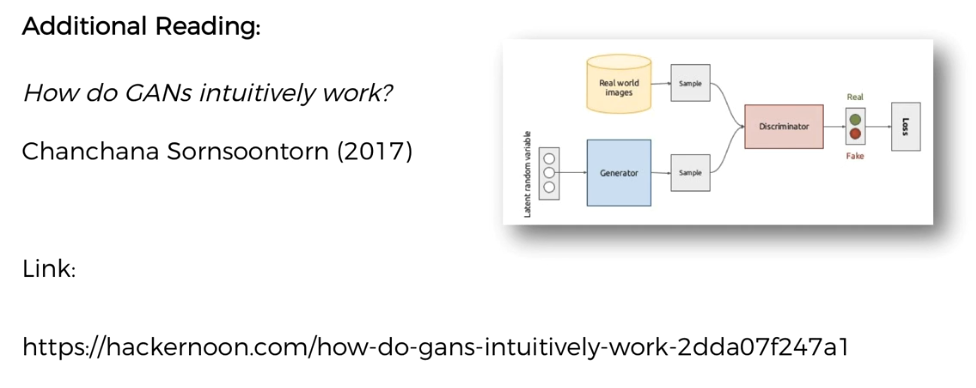
**3.2 The idea behind GANs**

GANs were invented by **Ian Goodfellow** when he was in his late 20s. GAN stands for ***Generative Adversarial Network***. We'll talk about what GANs are, how they're structured and how they're trained up.

* For now we talk about the philosophy. Why were they created in the first place?
* The reason why GANs were constructed is because:
* Neural networks are generally good at predicting things or classifying things and solving problems. *But they do not create something for itself*.
* So AI researchers wanted to create a type of neural network that can create for itself and that's why they came up with this idea of GAN or ***Generative Adversarial Network***.
* GANs can actually generate/create images that never actually ever existed before. Those objects never existed or built by humans. You can think about it as GANs *can have an IMAGINATION*.
* GANs learn about the world, our objects that we have/use or animals or anything, and then they can *create new versions* of those *objects*.
* How GANs work: GANs have two components. And these two components are *constantly in touch*.
* GENERATOR
* DISCRIMINATOR

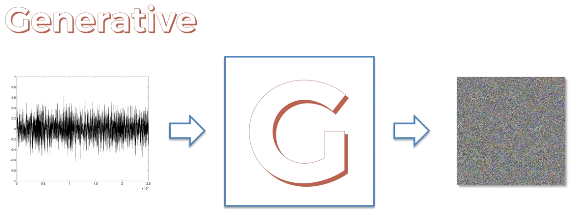


* The GENERATOR generates images and the DISCRIMINATOR then assesses those images and *tells* the generator: whether or not those images are likely to be *similar* to the *real world* that already exists.
* And when you're training this up you are bringing up these two GENERATOR and DISCRIMINATOR together from scratch.
* You're Not creating something that already knows everything about the world and generating things and the discriminator tell the generator if that looks like a real object or not.
* The real fact is that they start from SCRATCH (from zero) and then they learn together.
* For instance, generator generates some images, say: images of chair, and then the discriminator will look at the images that the generator generated and will look at some images of real chair to compare the two images. Then it will learn for itself what is a chair what isn't. It gain that knowledge and therefore it will know what is a chair what isn't.
* DISCRIMINATOR will also give Feedback to the GENERATOR saying Generated chairs are not really chairs because of some missing features.
* So DISCRIMINATOR and GENERATOR were kind of playing a *game* *against* each other but they're *learning together*.
* The GENERATOR is always trying to create images which look like real objects to fool the DISCRIMINATOR. But the DISCRIMINATOR always has real images of actual object to compare to in order to learn better. And then also give feedback to the GENERATOR.
* In that sense they learn together and grow together and eventually this whole network learns how to generate better images that are indistinguishable from the real world objects.
* Though DISCRIMINATOR and GENERATOR kind of working against each other, but as a whole the network, it's getting better and more robust.
* In nutshell, GANs learning about our world and then creating objects or images of objects which actually are very similar to what we have in our world but never actually existed.
* Additional readings: if you'd like some additional reading there's a great blog post by Chanchana Sornsoontorn (2017). It's on hackernoon.com and you can check it out there.
* It will talk you through the intuition behind GANs.



**3.3 How do GANs work (Step 1)**

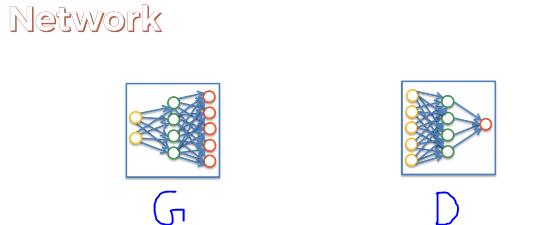
* Here we'll discuss about how GANs work. GAN stands for Generative Adversarial Networks and we're going to go through these letters one by one.
* After that, we'll get to the **training** **part** of the GAN.
* G- Generative: G stands for generative and this is a model which takes a random noise-signal as input and then it can output an image.



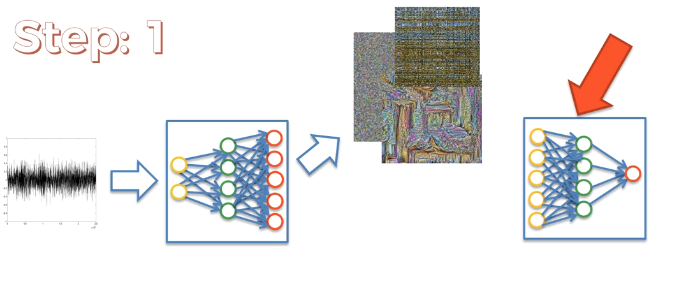
* A-Adversarial: The adversarial part stands for the discriminator. It is the opponent of the generator.
* You can think of the Generator as like a thief or somebody who's trying to forge dollar bills and discriminator as the detective.
* D the Discriminator, is model which is **capable** of **learning** about *objects, animals or people* or basically *certain features*.
* For instance if you show it lots of images of dogs and then you show it lots of non-dogs it will be able to *discriminate* between *dogs* and *non-dogs*.
* Ideally if you show the discriminator of a random non-dog image it'll give zero a zero probability and for real dog it will give 100% probability.

|  |  |
| --- | --- |
|  |  |
|  |
| Training | Detection |

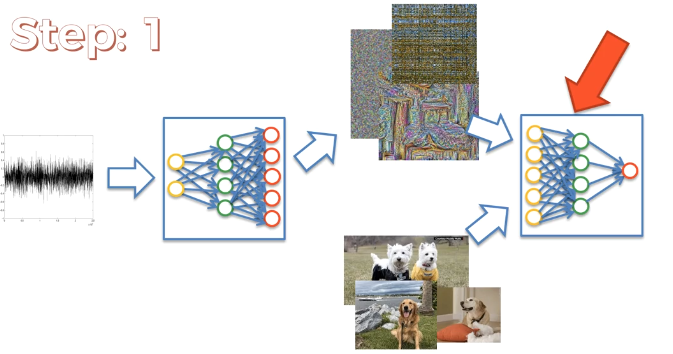
* Neural Networks: These two models, **Generator** and **Discriminator** are actually **Neural-Networks**. Since NNs are capable of learning, the weights of those synapses (blue-arrows) that's where information is captured.
* These weights are going to be updated during training of the neural networks.



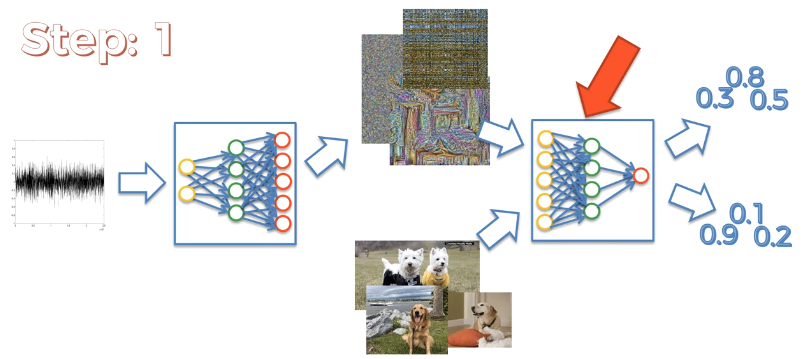
* Training Step-1: We'll going to do it three times, in the practical tutorials are going to do with 100 or 1000 of times.
* Here we will see how the networks Evolve over the iteration. So every step is an iteration.
* Don't confuse steps with EPOCHs.
* STEP-1: we input a random noise signal into our GENERATOR (it is some kind of random initialization).
* Then the Generator GENERATES some images. At the beginning, those images are completely useless and completely random.



* Train the DISCRIMINATOR: At this stage what we want to train the *discriminator*.
* We want the *discriminator* to be able to *distinguish* between for instance Dogs and Non-Dog.
* So we need to give the *discriminator* some Real-World Dog images.
* Hence, to train the *discriminator* we're going to input a batch of non-dog images and a batch of dog images.
* Now we'll see what probabilities this *discriminator* will spit out.



* Note that, the discriminator knows nothing at this stage, it hasn't been trained at all. This is the very first step.
* None of these two models (generator & discriminator) have been trained so the discrimination might spit out numbers like following.
* It may give some high probability to the random non-dog image at the beginning, and low probability to dog images.

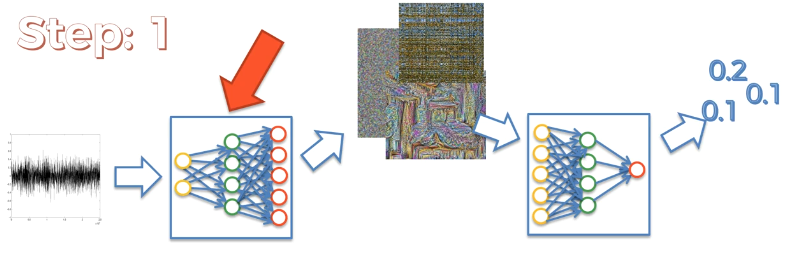


* That's because *discriminate* has never seen a *dog* before.
* It will fix these *mistakes* in the following *iterations* of training.

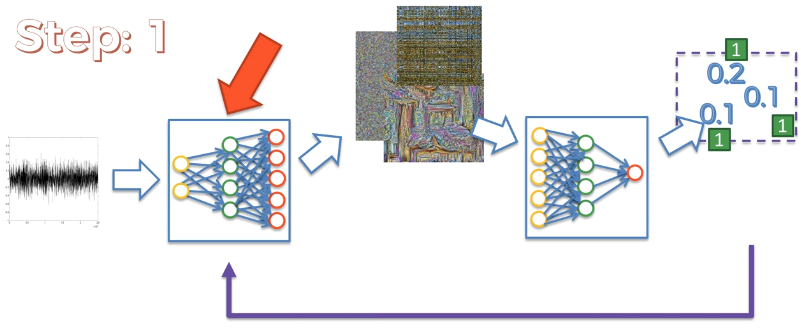
|  |  |
| --- | --- |
| * Next, we take these probabilities and we compare with the real world values (**0** for *Non-Dog* and **1** for *Dogs*) with respect to real-images. * We calculate the ERROR. And then the CUMULATIVE ERROR will be calculated. |  |

|  |  |
| --- | --- |
| * The Cumulative ERROR will be Back Propagated through the network of the Discriminator. * The weights on the network are a being updated. * Now the discriminator has learned from its mistakes and next time it's going to do a better job. |  |

* Train the generator: We're going to take the same batch of images that are created by the generator.
* We use the *same Generated Images* that were used for the *Training* of the *Discriminator*. Ian Goodfellow recommends in his paper that it worked fine.
* We're going to run them through the DISCRIMINATOR again but this time we don't need any dog images because the way that the GENERATOR learns is:
* By trying to *trick* *the* *discriminator*
* Based on whether it succeeds or not, it will update its weight.
* So it doesn't need to compare to any dogs or non-dogs it just needs to try and trick the discriminator.
* Then the discriminator will provide an output probability. Since the *Discriminator* is *trained* now, the output is different (it can distinguish between dogs and non-dogs more accurately).
* We can see, for non-dog image, the output are now close to **0**.



* They're not just absolutely ***0***, because it takes time for the *DISCRIMINATOR* to learn (to train).
* This is the reason, why these **two models (GENERATOR & DISCRIMINATOR) should be training at the same time together**.
* Both Generator & Discriminator have to train together:
* Because if the Discriminator was very Well Trained (training the discriminator separately) and then and the Generator was only Being Trained now than the *discriminator* doesn't *leave* any *chances* for the *generator*.
* If the *DISCRIMINATOR* is too smart for the *GENERATOR* has no chances of tricking the *DISCRIMINATOR*. That would be not good for anybody.
* We're going to take these values and we're going to compare them to what it should be.
* Now we compare these values with **1 (Dog)** instead of **0**, because now our *objective* is to *train the GENERATOR* to generate an image of Dog to trick the *DISCRIMINATOR*.
* So **once** we compare the output values to **1**, the **generator** can see those values are much **less** than **1**, i.e it can recognize, that *it's generating images that are not close to a Real-Dog*.
* Then we calculate the ERROR. We'll take the aggregate ERROR and it will be back propagated through the network of the *GENERATOR*.



* The GENERATOR now updates its Weight in a way that it will *generate better image next-time* (to improve the result).

(The back propagation in both cases consist SGD, which normally happens in any NN.)

**3.4 How do GANs work (Step 2)**

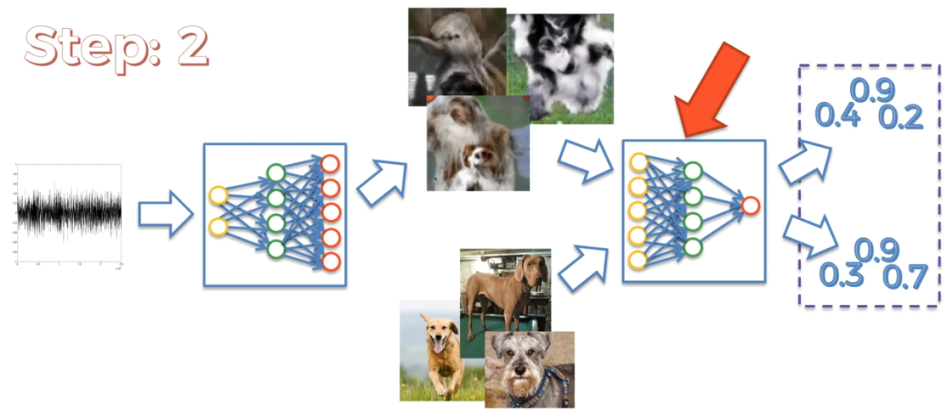
Step two is basically the same thing. But now the **networks** have **trained** **bit** **more**. So it's going to reiterate the whole process (as in first step) and understand it better.

* Step Two:

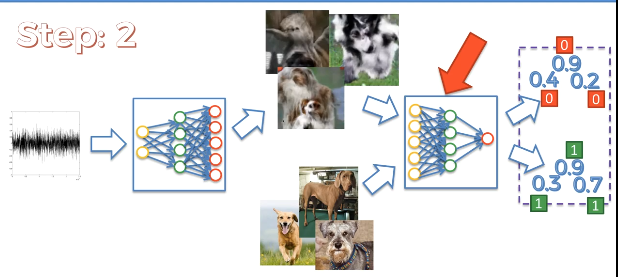
1. **NOISE** goes into the **GENERATOR**.
2. **GENERATOR** generates images which are **not random any more**, they are taking shape of **Dogs** (or some weird animals).

* We'll understand how they generate or understand these things. But basically it's through:

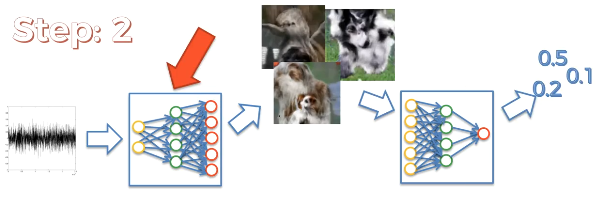
1. The *Back Propagation*
2. The calculation of error & understanding the mistakes.
3. Then adjusted its weight to make better generation.
4. TRAIN the DISCRIMINATOR: Then we're going to **train** the **DISCRIMINATOR** again:
5. Again we use some a new batch images of Real dogs.
6. We put Real-Dogs images along with Non-Dogs images into the DISCRIMINATOR.
7. Then it will output some values:



1. It will then compare the values to the actual numbers (**1** for dog **0** for non-dog). To calculate ERROR.



1. Discriminator is then Calculates the Error and *back-propagates* the *Error* through *its network* and therefore learns from that and the *weights* are updated. So that next time it'll do a better job at discriminating dogs versus non-dogs.
2. TO train the GENERATOR: We're going to use the *same* *generated* image.
3. Put them through the network of the Discriminator, then it output some values. Now the *values* are *more accurate* because the Discriminator is trained two times already. However these are not dropping to complete zero (because it needs more training).



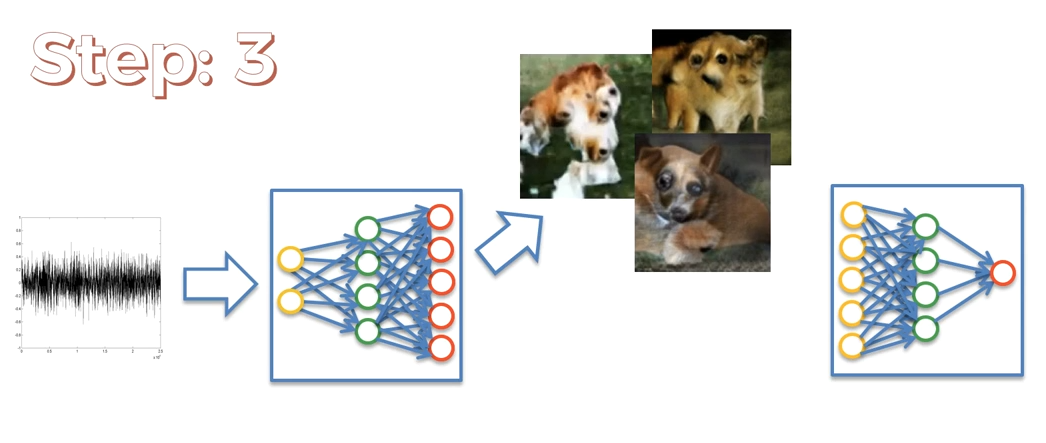
1. To train the *generator*, we again take these values we calculate *ERRORS* (we want these values to be 1, because *generator* wants to trick the *discriminator*).
2. Then these values are **back-propagated** through to the **GENERATORS** network and the weights are updated.

* Note that, these networks are huge (for our learning purpose we drawn them small).
* The **main point** is that, there's the Communication happening between the GENERATOR and DISCRIMINATOR.
* So the generator creates these images, it try to fool the discriminator.
* Discriminator has real images to examine the generated images from ***GENERATOR*** and ***DISCRIMINATOR*** get trained by comparing them.
* DISCRIMINATOR then output the probabilities for the generated image.
* GENERATOR then use those probability to learn its mistakes and then generate better images next time.

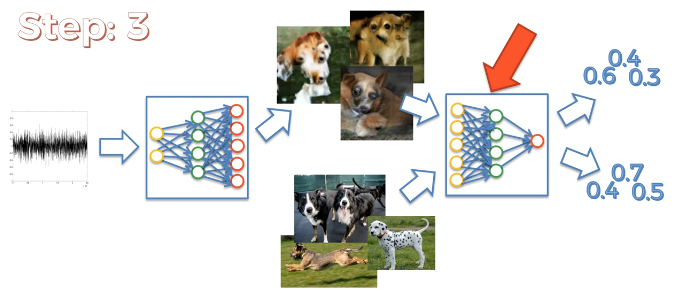
For example Discriminator learns to detects the eyes of a dog, but generated images of dogs has no-eyes or just only black-dots instead of eyes (the detected features also could be nose Or Paws or tails).

**3.5 How do GANs work (Step 3)**

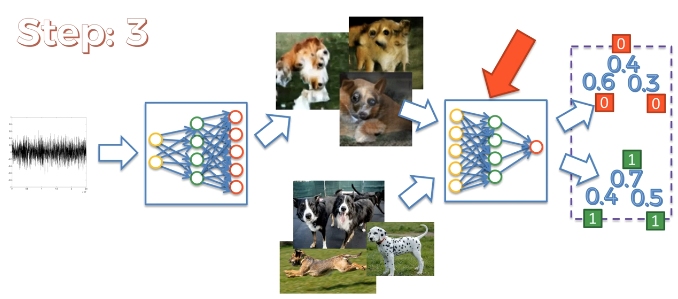
* Third Step:
* Noise goes into the generator,
* GENERATOR generates dogs (with eyes this time, they have too many eyes) we can also notice some other features (ears, paws, nose etc). But it's learned from the mistakes it made previously.

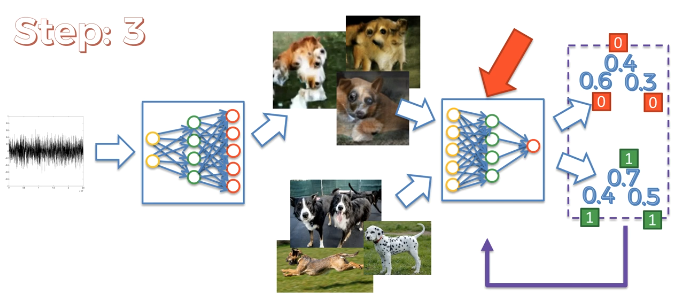


* Train DISCRIMINATOR: To train the discriminator and we're going to need another batch of real dog images along with generated images.
* Then we've got some *Top-values* and *Bottom-values* and we can see these values are slowly converging to 1 (Dogs), means that *generated* *dogs* values are *getting closer* to the values for the *real-dogs*.
* It is becoming harder for the DISCRIMINATOR to discriminate between Dogs and non-Dogs.
* So these generated images are getting closer to more realistic dogs.

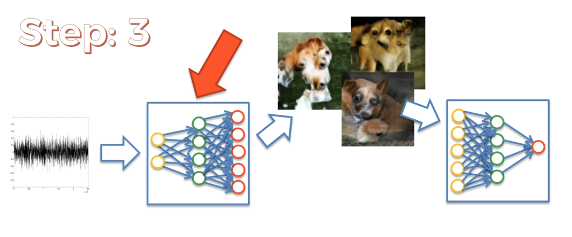


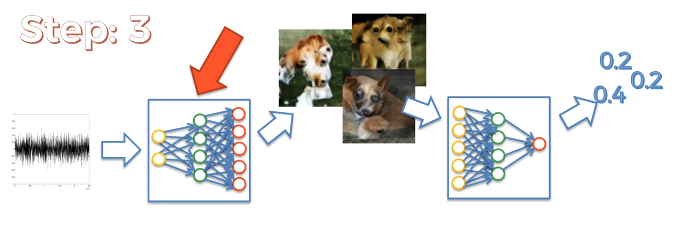
* Then we compare these values with real values ***0*** and ***1***, we calculate the ***errors***.
* That error is going to be back propagated through the Network of the DISCRIMINATOR.
* Weights are going to be updated.



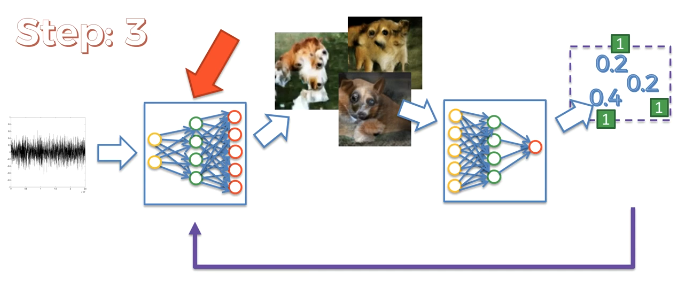


* Train the GENERATORS:
* We're going to put same generated images into the network.
* We get an output (improved) from Discriminator.

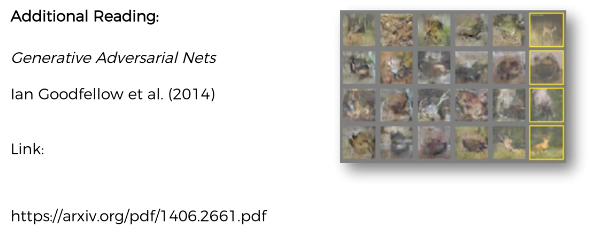




* We **calculate** the **Error** comparing with **1** (Dog).
* Error Back-Propagated through the network of the generator and the weights are updated. And GENERATOR getting trained.



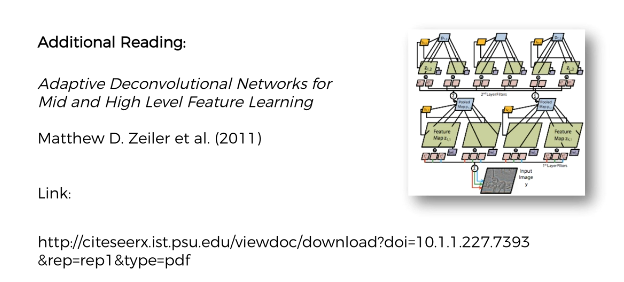
* We've discussed just three steps, in reality, there are 100 of 1000 steps in multiple epoch.
* **Additional Paper:**
* It is the original paper by *Ian Goodfellow* called *Generative Adversarial Nets (2014),* which you can find in archive.



* **DECONVOLUTIONAL Neural Network (Note):**
* What we've discussed hare is the **Deconvolutional Neural Network (opposite of CNN)**. We've not discussed this type of networks yet.

|  |  |
| --- | --- |
| **CNN** | **DNN - Deconvolutional Neural Network.** |
|  |  |

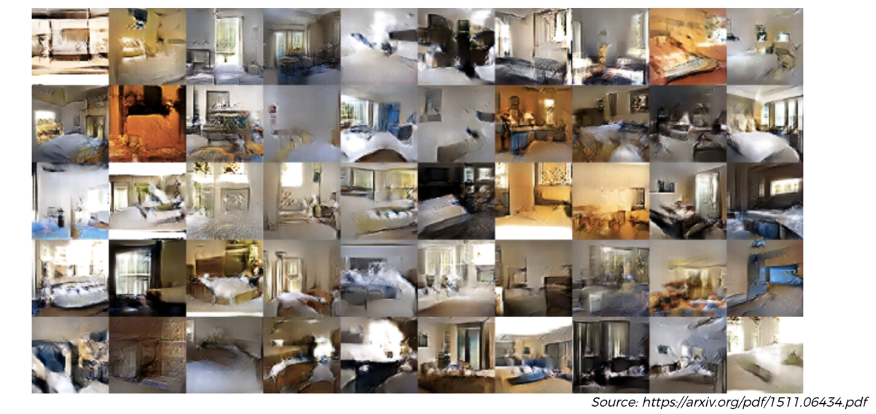
* **More on Deconvolutional Neural Network:**
* Adaptive Deconvolutional Networks for Mid and High Level Feature Learning by Matthew D. Zeiler et al. (2011).



**3.6 Applications of GANs**

GANs can be used for many different things. For instance:

* Generating images
* Image modification
* Super resolution
* Assisting artists
* Photo realistic images
* Speech generation
* face Ageing/De-ageing.
* **Generating images:**
* By looking at this can you say what this is: They're all images of one certain thing. But after a few epochs these will be more clear.



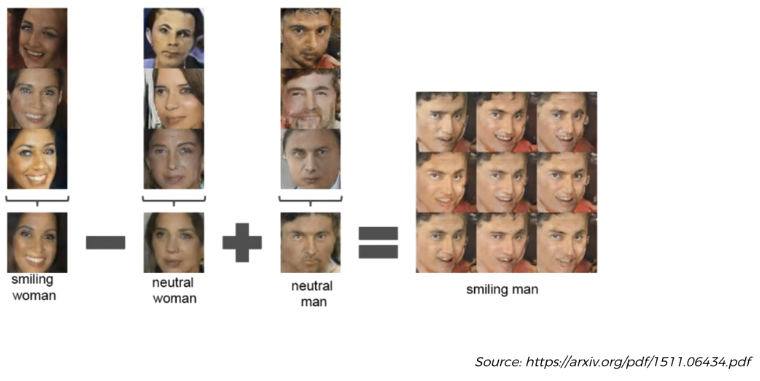
* However these are all generated images after one epoch.
* So after 12 or more epochs we get following result:



Now it is clear that, these are the images of Bed-Rooms.

This is one of the things that GAN's are doing really well, some of these bedroom's looks so realistic.

* **Image Modification:** Let's have a look at this example.



* We've got this from the same paper and here we got a smiling woman on the left (these are images generated by GAN, all of them).
* Here we got a neutral woman on in the middle and then you go to a neutral man.
* All smiling women are actually assigned to a vector. All neutral woman and neutral man are assigned to corresponding vectors. And we can do arithmetic's on them.
* Basically if we subtract the features of a neutral woman from smiling women and add the result to neutral man we get smiling men.

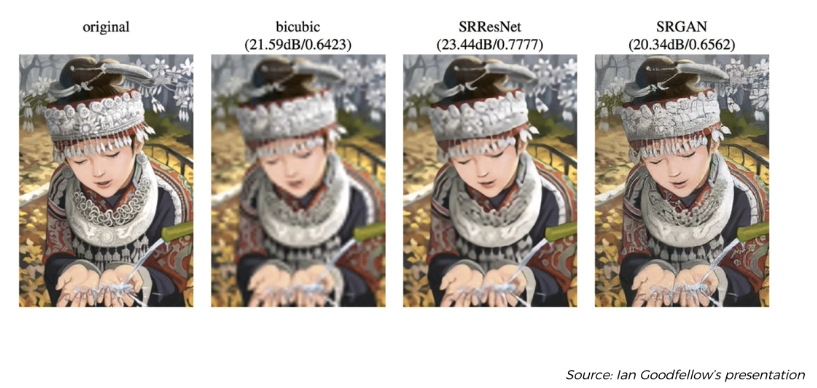
Very interesting And Creepy as well.

* Now here's another example. Men's with glasses subtract Men's without glasses (completely different man as you can see). And then you add a woman without glasses and you get a woman with glasses.

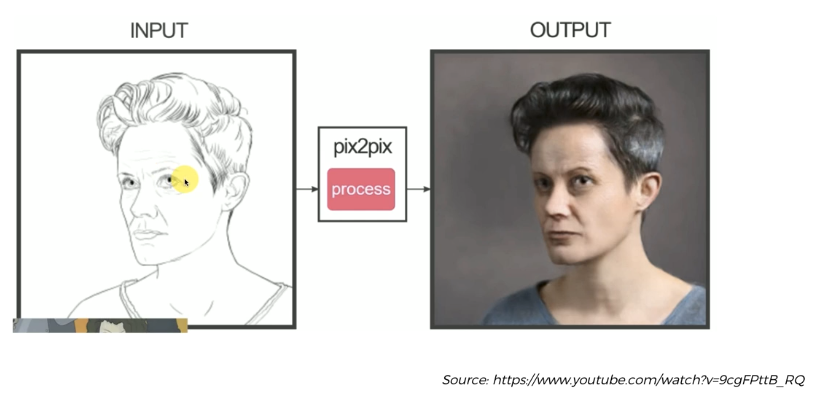
|  |  |
| --- | --- |
| * These images never really existed, they are all GAN generated images. * Now the point is, if we do the same thing using pixels, the result would be not good. We could get overlay of different images like follows: |  |



* **Super Resolution:**
* Here's an example from the *Ian Goodfellows presentation*, where they had an original image, then they reduced the resolution so made it blurry (more grainy and less detail).
* And then they applied different algorithms to increase their resolution back. One is SRResNet and another from SRGAN.
* Also notice that the pattern is messed up in the GAN applied image. But the resolution is good.

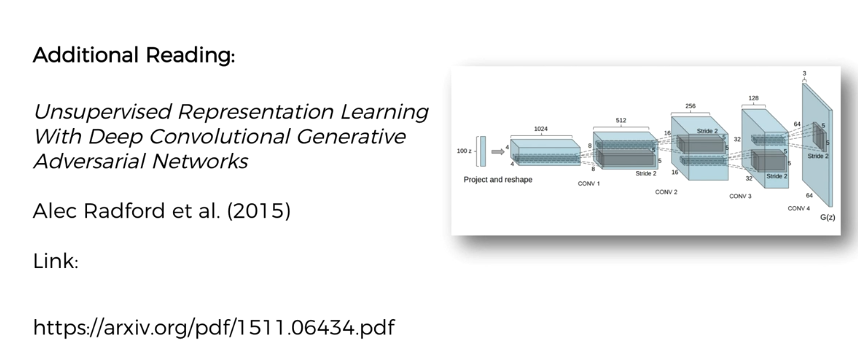


* **Photo-Realistic Images**
* One example is **Pix2Pix**. If you draw something with just on line and then you apply GAN, literally it just you get a photo realistic image.



|  |  |
| --- | --- |
| * **Face Aging:**   It generates what you will look like when you aged. The network is trained to extract the features of aging. |  |

* **Additional Readings:**



* Called Unsupervised Representation Learning With **Deep Convolutional Generative Adversarial Networks** by Alec Radford et al. (2015). DC-GANs are a step above an enhancement on top of GANs whether networks go even deeper.
* And you can find this paper on archive there and it's actually going to a lot of those examples are we talked about.
* YouTube video reference: called Artist Vs Pix2pix:Is This Humor Or Horror.

