Generative algorithms

Generative and **Discriminative** models are two different approaches that are widely studied in task of classification. They follow a different route from each other to achieve the final result. **Discriminative** models are widely popular and are used more comparatively to perform the task since they give better results when provided with a good amount of data. All the popular algorithms such as **SVM, KNN** etc. and popular network architectures such as **Resnet, Inception** etc. come under this.

The task of a **discriminative** model is simple, if it is shown data from different classes it should be able to discriminate between them. For example if I show the model a set of **dog and cat images** it should be able to say what is a dog and what is a cat by using discriminative features such as **eyes shape, ears size** etc.

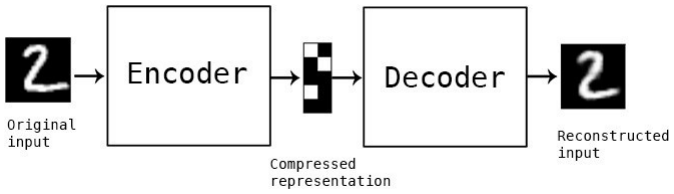
Generative model on the other hand has a much more complex task to perform. It has to understand the distribution from which the data is obtained and then needs to use this understanding to perform the task of classification. Also generative models have the **capability of creating data** similar to the training data it received since it has learnt the distribution from which the data is provided. For example if I show a generative model a set of dog and cat images now the model should understand completely what are the features that belongs to a certain class and how they can be used to generate similar images. Using this information it can do multiple things. It can compare the attributes to classify the image similar to how discriminative algorithm can classify an image. It can **generate** a new image which looks like one of the class images it has been provided for training.

> **Humans don't act like pure discriminators, we possess enormous generative capabilities**

Progress in generative algorithms is important because humans don't act just like pure discriminators, we have enormous generative or imaginative capabilities. If we give certain attributes such as **blue car on road** we can instantly generate a picture of that in our mind and we are looking at providing this kind of intelligence to machines.

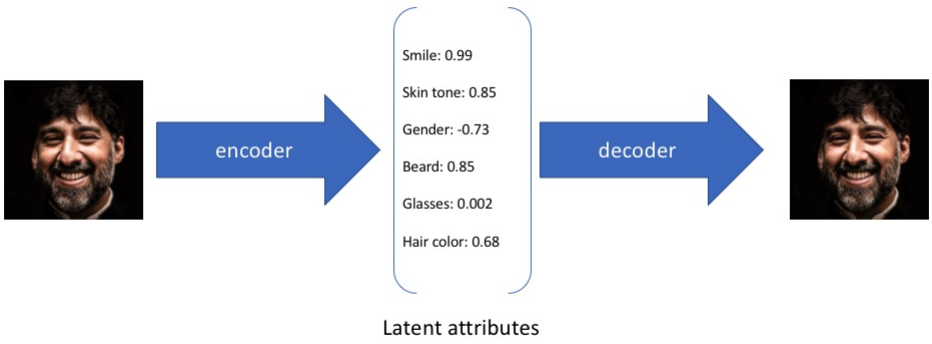
In the below content we will discuss about two famous generative algorithms **Autoencoders** and **Generative Adversial Networks**

Autoencoder



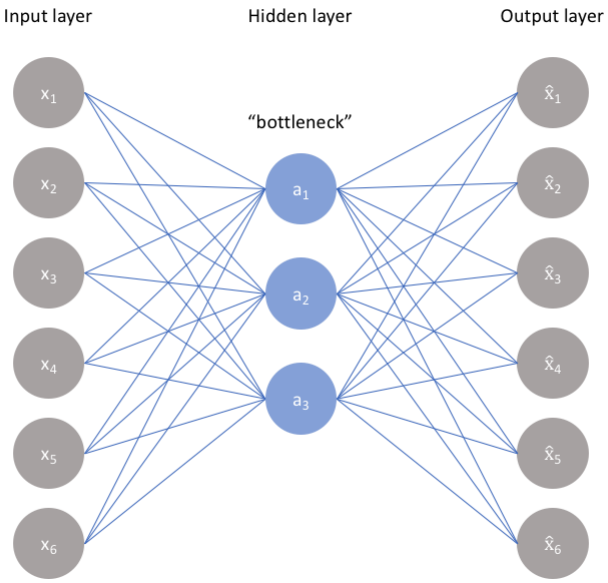
An **autoencoder** is a type of ANN used to learn efficient data codings in an unsupervised manner. The aim of an autoencoder is to learn a representation (encoding) for a set of data, typically for **dimensionality reduction.**

We as humans are pretty good at **visualizing** using few attributes. For example if we describe a human as tall, fair, bulky, no mustache, punjabi you can create a visualization based on these attributes. An autoencoder tries to achieve same thing. If I show an image of a person it learns all the attributes(known as **latent attributes**) such as the above needed to identify the person and then can use them to visualize/reconstruct the person.

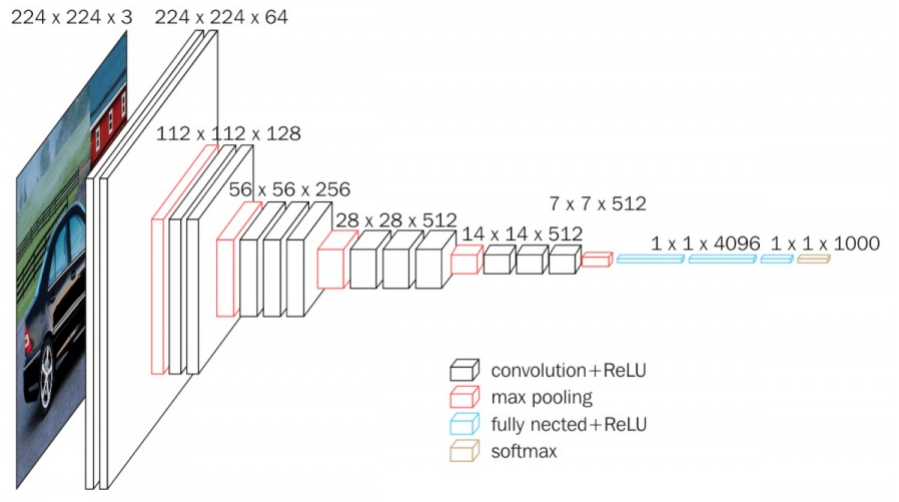


Autoencoder consists of 3 components

1. **Encoder**
2. **Bottleneck**
3. **Decoder**



Encoder is similar to any classification neural network such as **Resnet** etc. sans the prediction softmax layer. If we see the below figure of VGG network if we remove the final softmax layer the final 1000 values that we get can be thought of as 1000 latent attributes of an image.



Decoder is the opposite of encoder, it takes the latent attributes from the output of encoder and tries to reconstruct the image. This is done using deconv layers which can unsample the input

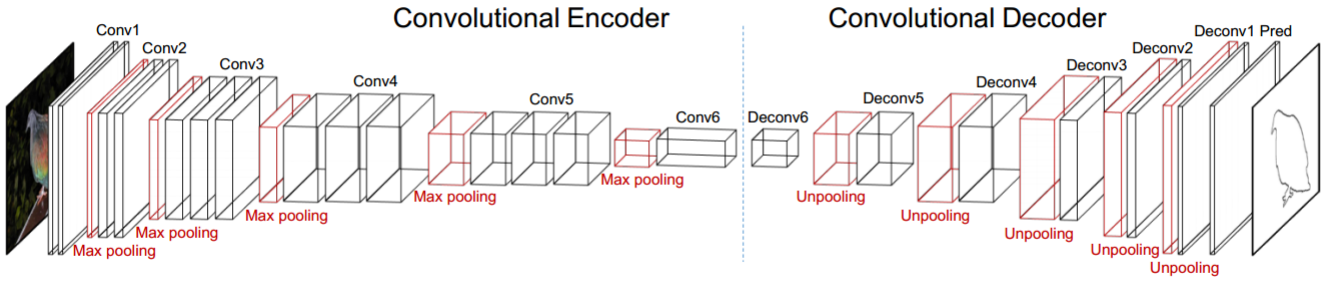


Figure 2. Architecture of the proposed fully convolutional encoder-decoder network.

Bottleneck is the latent vector that is output by the encoder and is upsampled by the decoder. It contains the **latent attributes** that are produced by the decoder such as the height, weight etc. described above.

The network of encoder and decoder is trained together using **backpropagation** to reduce the loss of reconstruction such as mean square error between the pixels.

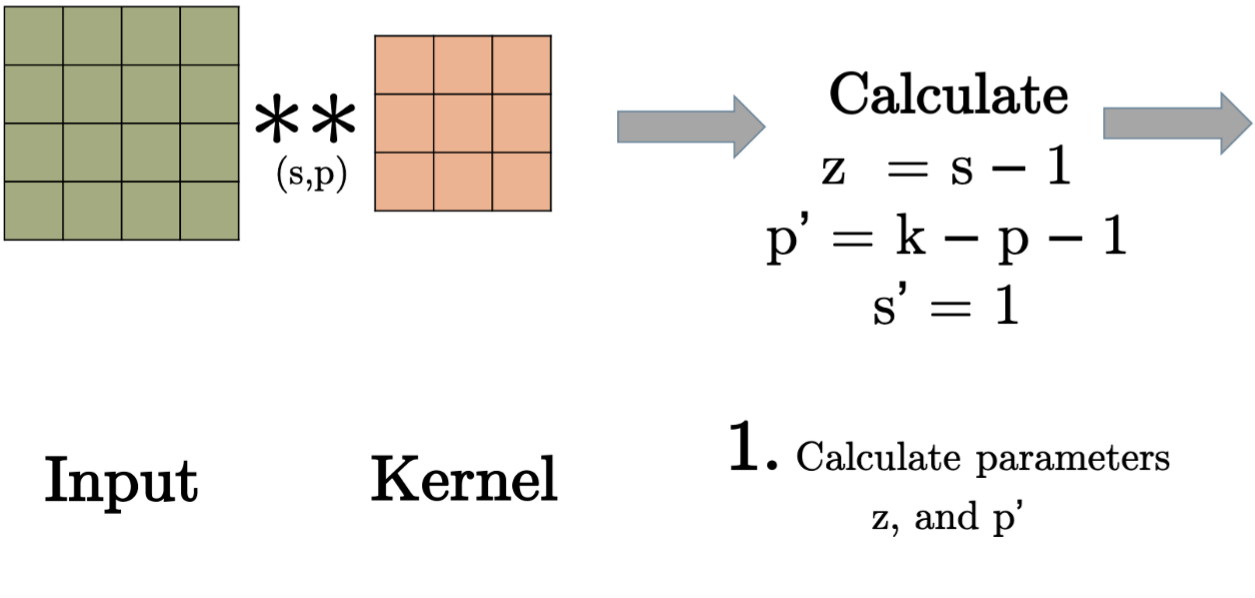
Transposed Convolutional Layer:

A transposed convolutional layer, on the other hand, is usually carried out for upsampling i.e. to generate an output feature map that has a spatial dimension greater than that of the input feature map. Just like the standard convolutional layer, the transposed convolutional layer is also defined by the padding and stride. These values of padding and stride are the one that hypothetically was carried out on the output to generate the input. i.e. if you take the output, and carry out a standard convolution with stride and padding defined, it will generate the spatial dimension same as that of the input.

Implementing a transposed convolutional layer can be better explained as a 4 step process

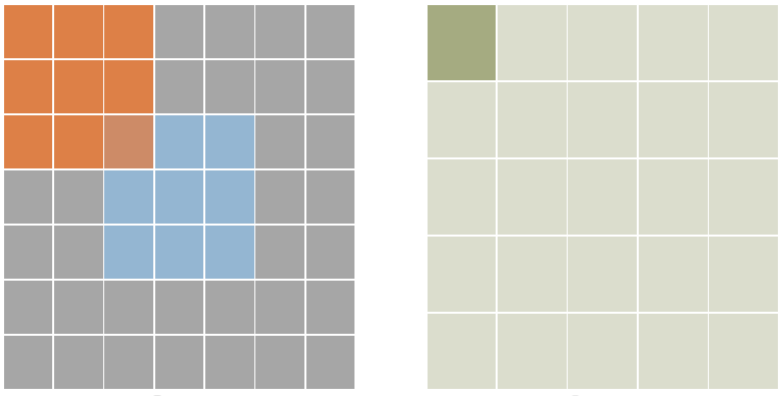
- **Step 1:** Calculate new parameters z and p'
- **Step 2:** Between each row and columns of the input, insert z number of zeros. This increases the size of the input to ***(2*i-1)x(2*i-1)***
- **Step 3:** Pad the modified input image with p' number of zeros
- **Step 4:** Carry out standard convolution on the image generated from step 3 with a stride length of 1

The complete steps can be seen in the figure below.



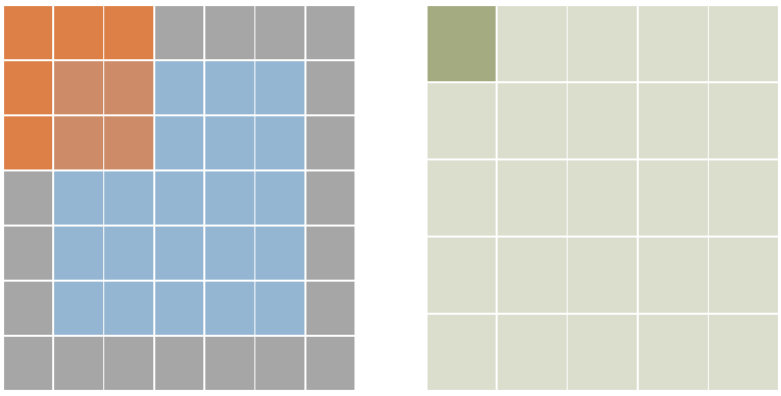
The animations below explain the working of convolutional layers for different values of stride and padding.

Type: transposed conv - Stride: 1 Padding: 0

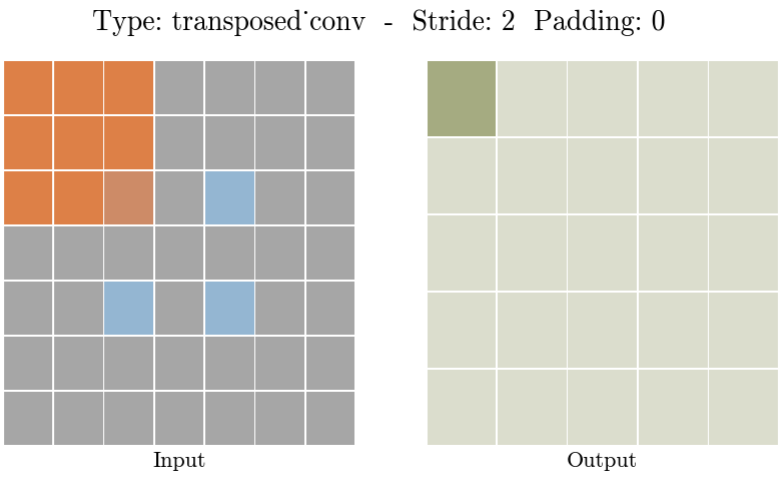


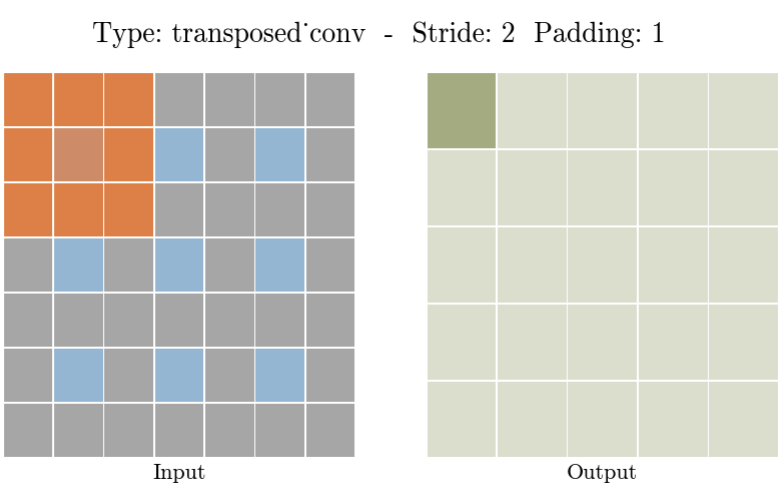
Input Output

Type: transposed conv - Stride: 1 Padding: 1



Input Output





For a given size of the input i, kernel k, padding p and stride s, the size of the output feature map o generated is given by

$$o = (i - 1) \times s + k - 2p$$

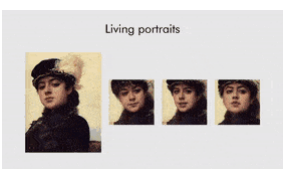
Applications of autoencoder :-

1. **Denoising images** :- Autoencoders can be used to remove noise from images. Since autoencoder learns the latent representation and not the noise it can remove the noise and give the clear image. It is trained by providing noisy images at input and we try to minimize reconstruction error with proper images at output
2. **Recommender systems** :- Netflix movie recommendation challenge winner used deep autoencoders
3. **Compression** :- As we have seen autoencoder converts an input to its latent space attributes and converts it back, it can be used for compression by making latent space much smaller in comparison to input.
4. **Dimensionality Reduction** :- Autoencoders can be used similar to **PCA** to reduce the feature space by mapping input to latent attributes and using them for modelling.
5. **Generation of data** :- A variant of autoencoders called **variational autoencoders** can be used to generate data similar to the distribution it is trained on.

Generative Adversial Networks

GAN's are another set of generative algorithms and are one of the primary reasons for producing so much hype in deep learning. Several applications have been made using GANs and a multitude of architectures have been researched upon which led to rapid development in the field of GAN's which can generate cool results which can make one wonder if it is real image or an image generated by GAN. For example below are the faces of person who have never existed in the real world. Looks pretty **cool** right. You can go https://thispersondoesnotexist.com . This website gives a realistic fake person on every refresh.

Recently Samsung published a paper in which a neural network takes just a picture and can produce a small video gif out of it. Through this they have got **Monalisa** alive. Think about what is in the future possibility. You can make you dead ancestors speak to you. Holy **awesome**



Until now we used to believe we can have faith of whatever we see or listen since they happened in real i.e **videos news must be true. Not anymore**. Now realistic fake videos or audio of the person can be generated like below. Now you can't even trust video news.

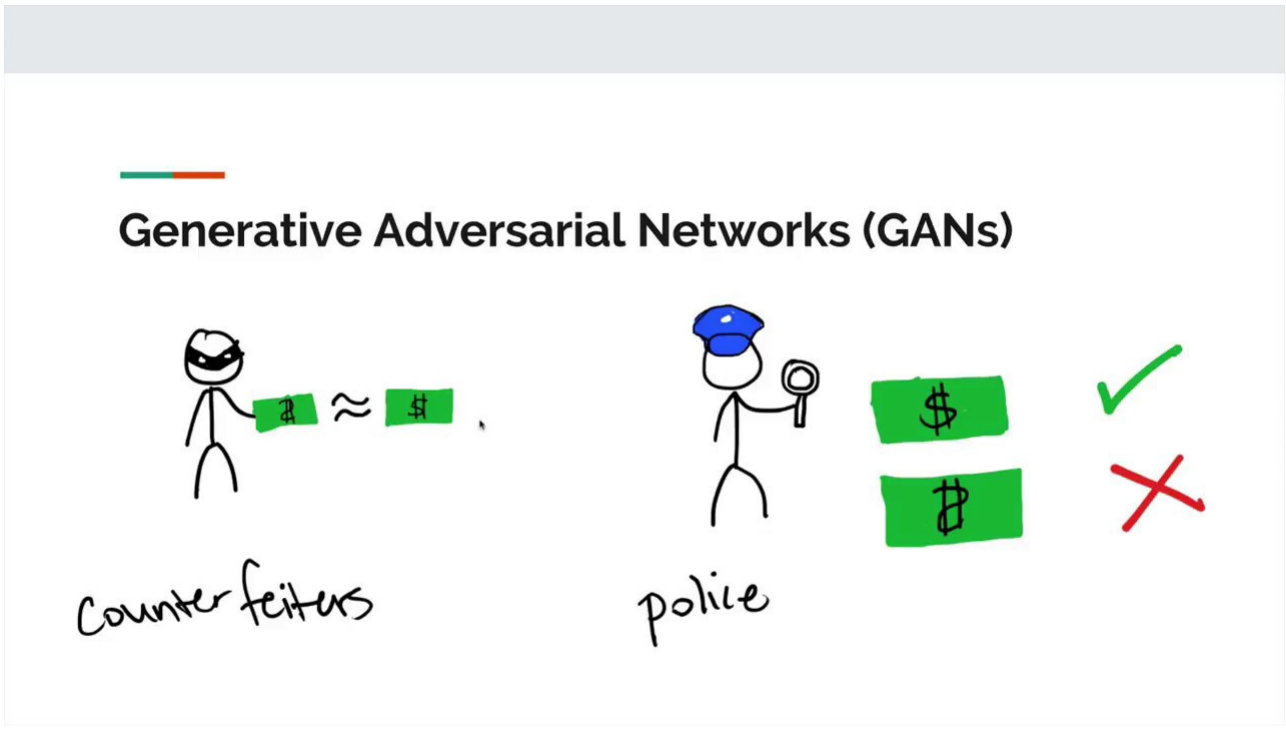
You can do domain transfer using a GAN as well. If you have an image of a horse you can **reimagine** as how it would look like if it is a zebra by using a GAN.

You can reimagine yourself playing out a protagonist character in a movie just like the below **guy transformed him into Leonardo Decaprio**. This is the next level of **Dubsmash**.

The possibilities are endless. You can create an entire movie without any real cast. Take a scene from real world and convert it to anime. Create fake persons for acting as models for donning the dresses in ecommerce website.

Working of GAN

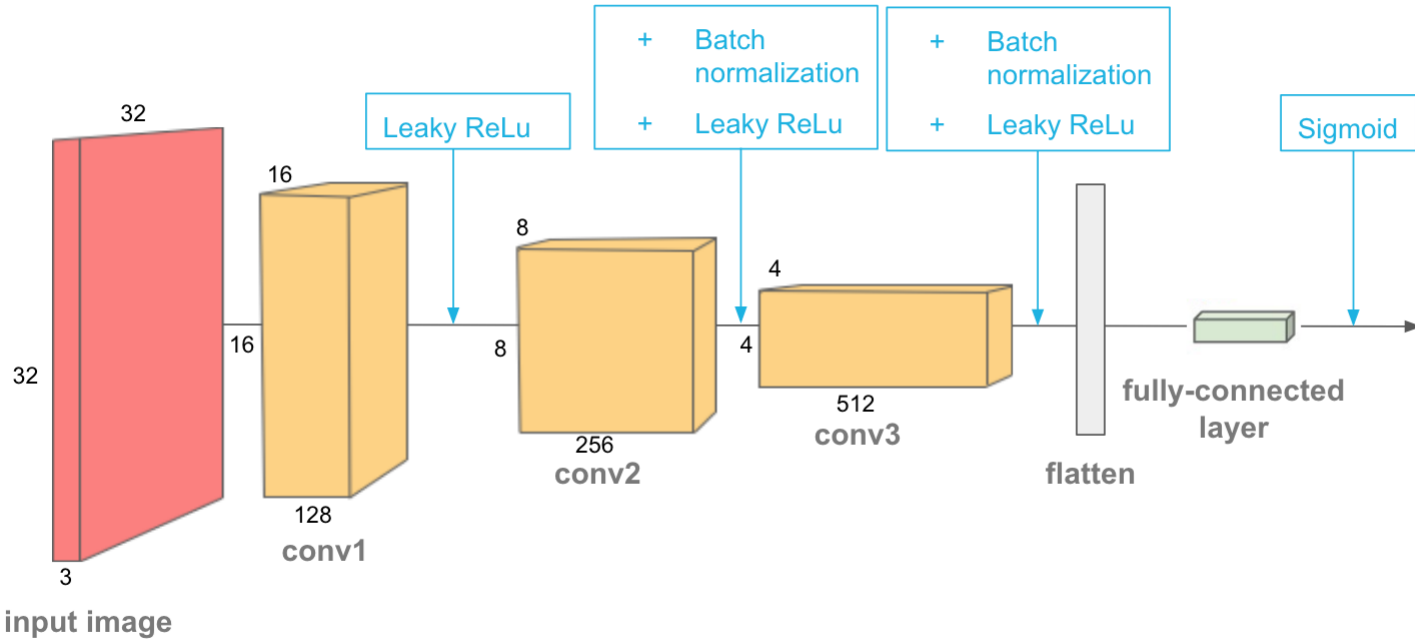
Now we will look into the theory behind how all this magic works. GAN consists of 2 neural networks(VAE from above has only a single neural network) which work with each other namely **Generator** and **Discriminator** . They act like **teacher-student, thug-cop**. The task of a Generator is simple as it name says it generates data for example an image which has to look like the real-world data. The task of Discriminator is to look at the data from Generator and discriminate it from real-world data i.e it should look at data generated from Generator and say it's **fake**.



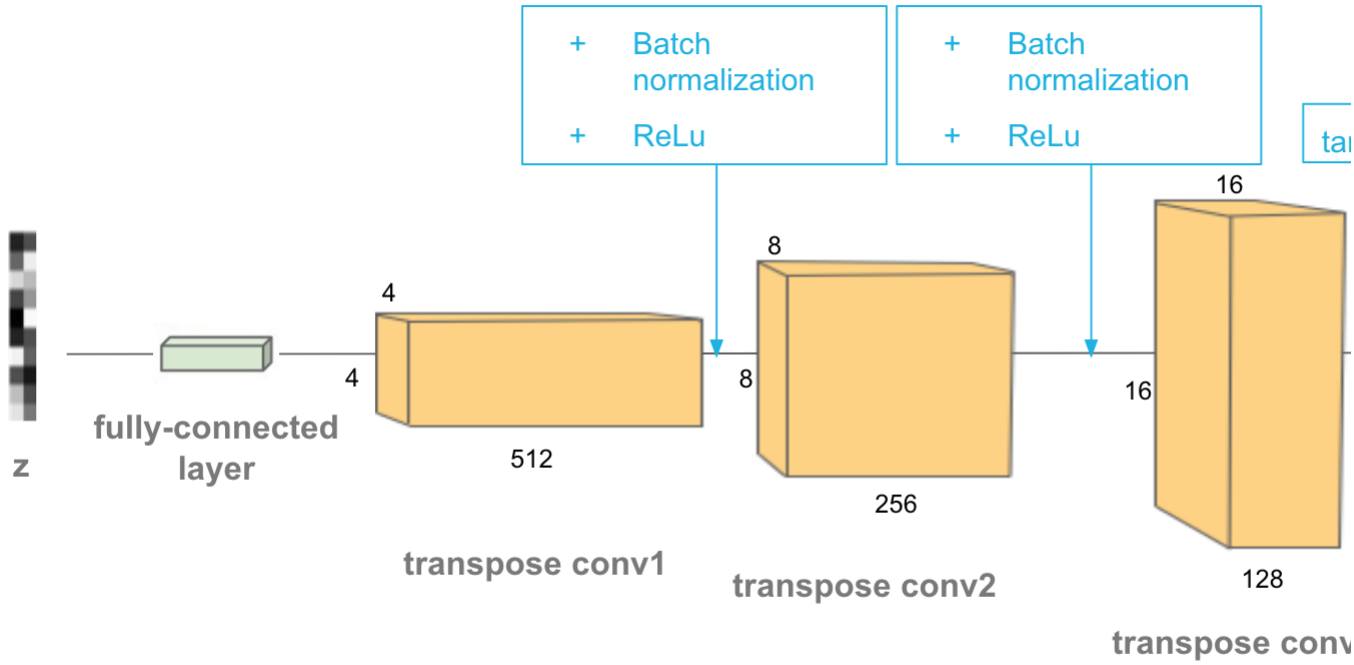
Now the **cat and mouse game** starts. Since discriminator says that the data is fake generator tries to better itself so that it can produce more realistic data which the discriminator can't judge as real or fake. Once this happens the discriminator knows that it is failing to properly discriminate so it will try to improve itself and next time it judges better. Now the ball is in the court of generator and this game of trying to overpower each other continues until a stage comes where the discriminator is completely confused whether the data from generator is real or fake. Now generator wins and we have been rooting for generator all along. But remember the **hero is as good as the villain** . **Avengers Endgame** was such a success because **Thanos** was such a menacing villain. So we want the discriminator to be the best and the generator needs to beat discriminator at its peak since then the victory is more sweeter.

There are multiple architectures and quite a lot of complex loss functions to make the GANs work and we will be looking at one of the most successful architecure **DCGAN** (Deep Convolutional GAN) which first introduced the usage of convolutional layers for GAN.

As said before GAN consists of two neural networks **Discriminator** and **Generator** . The architectures of these neural networks are similar to that of **Encoder** and **Decoder** in **VAE**. Discriminator is the neural network we are fairly familiar with, image as input which is sampled down with convolutional layers and finally we apply a softmax to get the output class, in this case we have only 2 classes **Fake or Not-Fake** . So common architectures like Resnet, Inception can used to model a Discriminator of DCGAN. Discriminator is trained by providing the real images as real class category and fake images given by generator as fake class category. So it is a 2 class classification problem.



Generator architecture looks opposite to that of Discriminator. It takes a linear vector and upsamples it similar to **Decoder in VAE**. The linear vector is generator by random sampling and just like in VAE we can think of it as attributes of latent space. Different random samplings generate different outputs.



Both the discriminator and generator are trained simultaneously in a **minimax** game. Discriminator tries to reduce the discriminative loss such as cross-entropy loss and Generator tries to oppose it by trying to increase the error. Unlike the general tasks like classification, detection etc. the **loss doesn't constantly decrease** since there are two opposing parties involved. Also we don't want either discriminator to overpower generator from the start or vice-versa since in that case it will be a one-sided game and the two networks on a whole wouldn't be learning as there is no competition, so we start from a stage where both are equally dumb i.e think of generator as generating random images and discriminator as randomly classifying images as fake or real. The networks slowly improve by competing with each other until it reaches a stage where generator completely fools the discriminator.

Training GANs is an art and there are a lot of hacks involved in stabilizing the process of training two networks simultaneously but that is the content for another article.

Loss Functions

Rather than having just a single loss function, here we need to define three function.

The Discriminator has two task

- **Discriminator** has to correctly label real images which are coming from training data set as "*real*".
- **Discriminator** has to correctly label generated images which are coming from **Generator** as "*fake*".

We need to calculate two losses for the Discriminator. The sum of the fake image and real image loss is the overall Discriminator loss

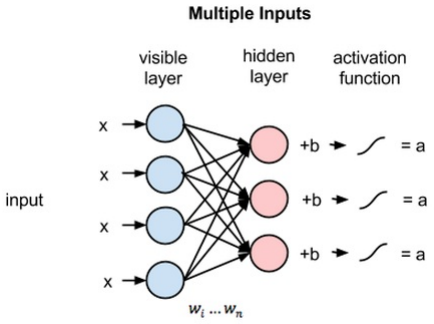
The generator network has one task

- To create an images that looks as *"real"* as possible to fool the **Discriminator**.

Restricted Boltzmann Machines

Boltzmann machines are stochastic and generative neural networks capable of learning internal representations and are able to represent and solve difficult combinatoric problems.

RBMs are a two-layered artificial neural network with generative capabilities. They have the ability to learn a probability distribution over its set of input. RBMs were invented by Geoffrey Hinton and can be used for dimensionality reduction, classification, regression, collaborative filtering, feature learning, and topic modelling.



RBM is a Stochastic Neural Network which means that each neuron will have some random behaviour when activated. There are two other layers of bias units (hidden bias and visible bias) in an RBM. This is what makes RBMs different from autoencoders. The hidden bias RBM produce the activation on the forward pass and the visible bias helps RBM to reconstruct the input during a backward pass. The reconstructed input is always different from the actual input as there are no connections among the visible units and therefore, no way of transferring information among themselves.