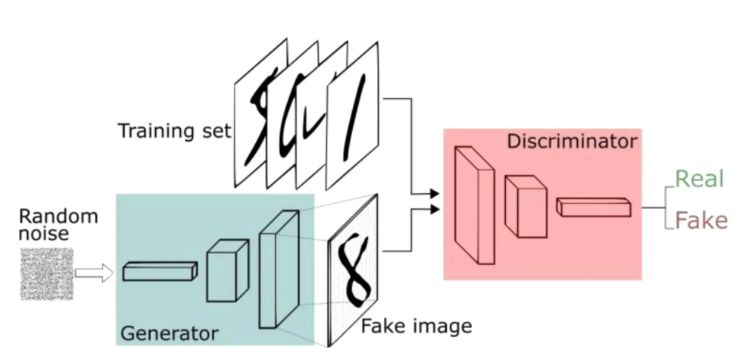
Generative Adversarial Networks (GANs)

Generative Adversarial Networks (GANs) are a type of machine learning model that generate realistic data by pitting two neural networks against each other: a generator and a discriminator. The generator produces samples, and the discriminator tries to distinguish between the generated samples and real data. Through this adversarial training process, the generator learns to create data that is difficult for the discriminator to differentiate, leading to high-quality synthetic data.



**Key Concepts:**

**Generator (G):**Takes random noise as input and produces a sample that resembles the training data. Its goal is to deceive the discriminator.

**Discriminator (D):**Classifies input samples as either real or fake (generated by the generator). It tries to improve its ability to distinguish between real and fake data.

**Adversarial Training:**The generator and discriminator are trained simultaneously, with the generator trying to fool the discriminator and the discriminator trying to improve its ability to detect fake data.

**Iterative Learning:**GANs operate on the principle of iterative learning, where both networks are updated over time, with feedback loops driving improvement.

**How GANs Work:**

* **Input:** The generator receives a random input (e.g., noise vector).
* **Generation:** The generator creates a sample based on the input.
* **Discrimination:** The discriminator receives both real data samples and the generated samples from the generator.
* **Classification:** The discriminator classifies each sample as either real or fake.
* **Feedback:** The results of the discriminator's classifications are used to update both the generator and discriminator.
* **Iteration:** Steps 1-5 are repeated iteratively, with the generator learning to produce more realistic samples and the discriminator learning to detect them.

**Standard GAN loss function (min-max GAN loss)**

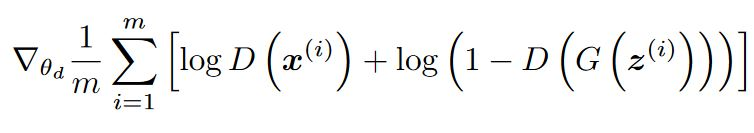
The standard GAN loss function, also known as the min-max loss, was first described in a 2014 paper by Ian Goodfellow et al., titled “Generative Adversarial Networks“.



* The generator tries to minimize this function while the discriminator tries to maximize it. Looking at it as a min-max game, this formulation of the loss seemed effective.
* In practice, it saturates the generator, meaning that the generator quite frequently stops training if it doesn’t catch up with the discriminator.
* The Standard GAN loss function can further be categorized into two parts: Discriminator loss and Generator loss.

**Discriminator loss**

While the discriminator is trained, it classifies both the real data and the fake data from the generator. It penalizes itself for misclassifying a real instance as fake, or a fake instance (created by the generator) as real, by maximizing the below function.

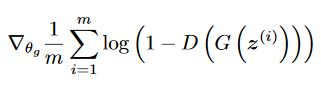


* log(D(x)) refers to the probability that the generator is rightly classifying the real image,
* maximizing log(1-D(G(z))) would help it to correctly label the fake image that comes from the generator.

**Generator loss**

While the generator is trained, it samples random noise and produces an output from that noise. The output then goes through the discriminator and gets classified as either “Real” or “Fake” based on the ability of the discriminator to tell one from the other. The generator loss is then calculated from the discriminator’s classification – it gets rewarded if it successfully fools the discriminator, and gets penalized otherwise.

The following equation is minimized to training the generator:



**Benefits of GANs:**

* **Data Augmentation:** GANs can be used to generate synthetic data, which can be used to augment existing datasets, especially when real data is scarce.
* **Image Generation:** GANs are known for their ability to generate high-quality images, which has applications in various fields.
* **Video Synthesis:** GANs can be used to create synthetic videos.
* **Scientific Simulations:** GANs can be used to create realistic simulations of physical phenomena.

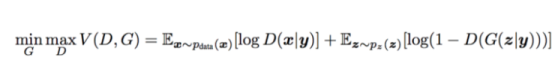
**Challenges:**

* **1. Mode Collapse:**The generator might produce only a limited set of samples, rather than a diverse range.
* **2. Training Instability:**Training GANs can be challenging, as the generator and discriminator can become unstable and fail to converge.
* **3. Vanishing Gradients**
* This phenomenon happens when the discriminator performs significantly better than the generator. Either the updates to the discriminator are inaccurate, or they disappear. One of the proposed reasons for this is that the generator gets heavily penalized, which leads to saturation in the value post-activation function, and the eventual gradient vanishing.

**Conditional Generative Adversarial Network (CGAN)**

This version of GAN is used to learn a multimodal model. It basically generates descriptive labels which are the attributes associated with the particular image that was not part of the original training data. CGANs are mainly employed in image labelling, where both the generator and the discriminator are fed with some extra information y which works as an auxiliary information, such as class labels from or data associated with different modalities. The conditioning is usually done by feeding the information y into both the discriminator and the generator, as an additional input layer to it.

The following modified loss function plays the same min-max game as in the Standard GAN Loss function. The only difference between them is that a conditional probability is used for both the generator and the discriminator, instead of the regular one.



Why conditional probability? Because we are feeding in some auxiliary information(the green points), which helps in making it a multimodal model, as shown in the diagram below:

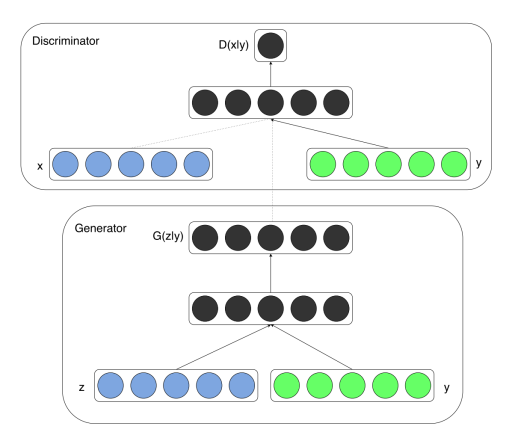


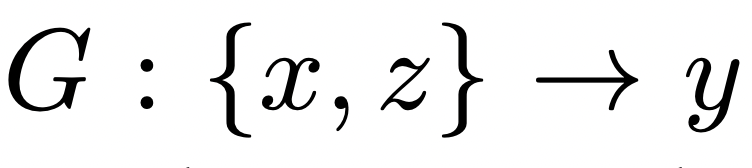
Figure 1: Conditional adversarial net

This medium article by Jonathan Hui delves deeper into CGANs and discusses the mathematics behind it.

**Pix2Pix**

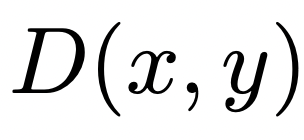
Pix2Pix GAN is a conditional GAN (cGAN) that was developed by Phillip Isola, et al. Unlike vanilla GAN which uses only real data and noise to learn and generate images, cGAN uses real data andnoise as well as labels to generate images.

In essence, the generator learns the mapping from the real data as well as the noise.



The generator G combines the learnt real data x and the random noise z to output y, which is the fake data.

Similarly, the discriminator not only learns from the “real data” example it has seen, but also from the labels that help it understand what is real and what is fake.

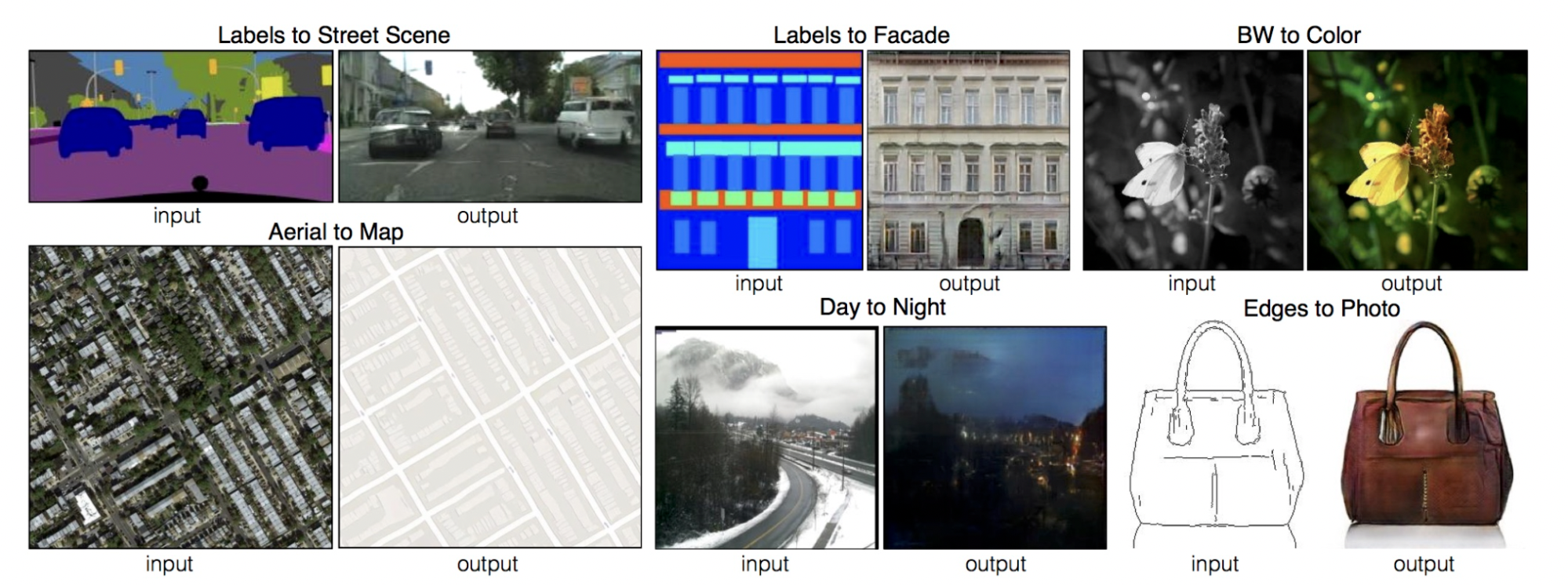


The discriminator uses, then, two sources of information to improve its ability to tell real from fake: x (the real data) and y (the label saying “real” or “fake.”)

This setting makes cGAN to be suitable for image-to-image translation tasks, where the generator is conditioned on an input image to generate the corresponding output image. In other words, the generator uses a condition distribution (or data) such as a guide or a blueprint to generate a target image (see the image below).



The model generates realistic building facades (right column) based on input segmentation maps (left column), with comparisons to the actual ground truth images (center column)



The idea with Pix2Pix relies on the dataset provided for the training. It is a pair-to-pair image translation with training examples {x, y} having a correspondence between them.

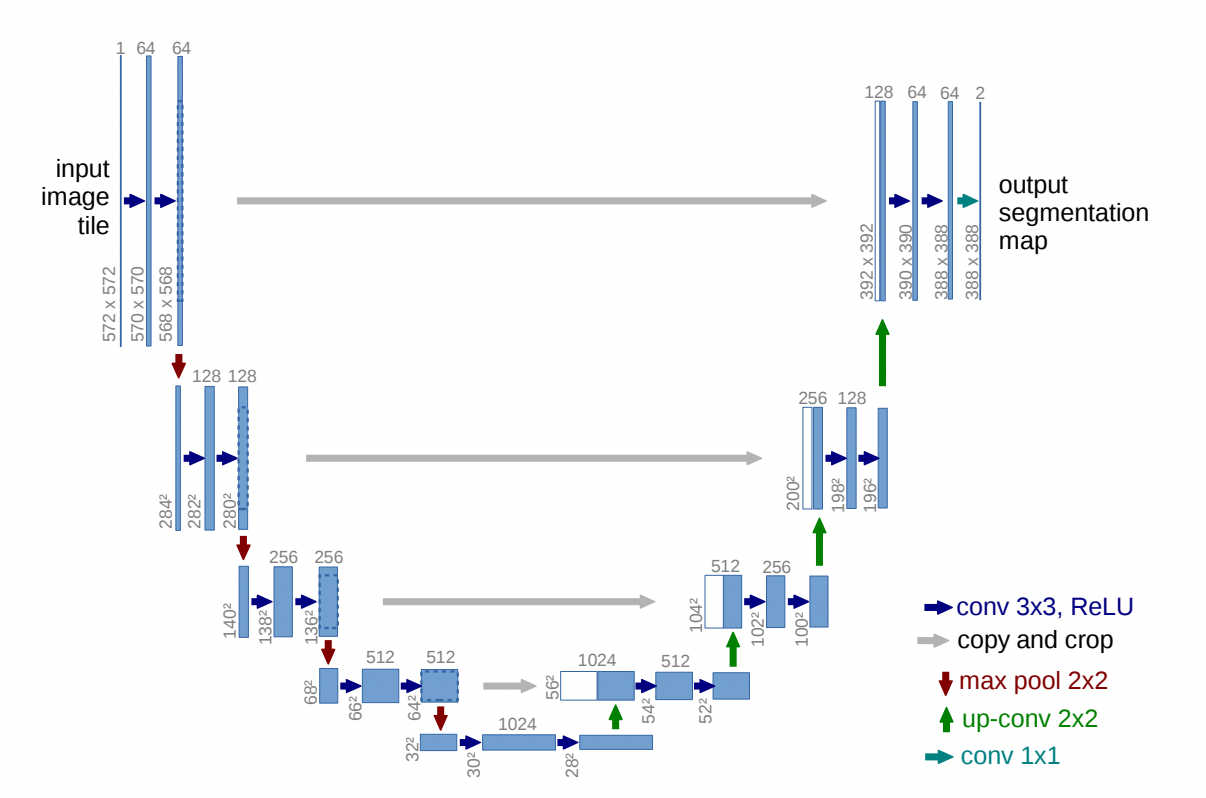
**Pix2Pix network architectures**

The pix2pix has two important architectures, one for the generator and the other for the discriminator, namely U-net and patchGAN.

**U-Net generator**

As mentioned before, the architecture used in pix2pix is called U-net. U-net was primarily developed for biomedical image segmentation by Ronneberger et. al. in 2015.

*U-Net generator: A symmetric encoder-decoder structure with down-sampling through max pooling (red arrows) and up-sampling via transposed convolutions (green arrows). Skip connections (gray arrows) connect layers of matching spatial dimensions in the encoder and decoder, preserving spatial information for segmentation in the output map.*



U-Net consists of two major parts:

* A contracting path made up of convolutional layers (left side) which downsamples the data while extracting information.
* An expansive path made of up transpose convolution layer (right side) which upsamples the information.

Let’s say our downsampling has three convolutional layers C\_l(1,2,3), then we have to make sure that our upsampling has three transpose convolutional layers C\_u(1,2,3). This is because we want to connect the corresponding blocks of the same sizes using a skip connection.



Skip connection architecture: This diagram illustrates the use of skip layers between encoder (C\_l1, C\_l2, C\_l3) and decoder (C\_u1, C\_u2, C\_u3) blocks, with a bottleneck in the center to keep the feature dimensions at each stage. This retains the spatial details across the network

**Downsampling**

During downsampling, each convolutional block extracts spatial information and passes the information to the next convolutional block to extract more information until it reaches the middle part known as the bottleneck. Upsampling starts from the bottleneck.

**Upsampling**

During upsampling, each transpose convolutional block expands information from the previous block while concatenating the information from the corresponding downsampling block. By concatenating information, the network can then learn to assemble a more precise output based on this information.

This architecture can localize, i.e. it can find the object of interest pixel by pixel. Furthermore, U-Net also allows the network to propagate context information from lower resolution to higher resolution layers. This allows the network to generate high-resolution samples.

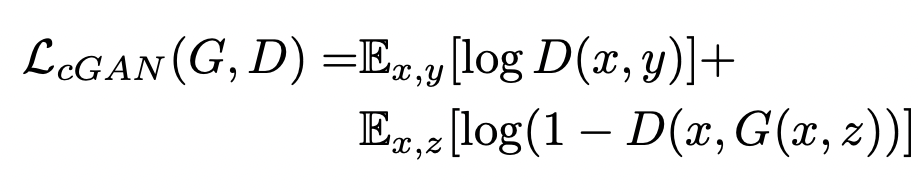
**Markovian discriminator (PatchGAN)**

The discriminator uses PatchGAN architecture. This architecture contains several transposed convolutional blocks. It takes an NxN part of the image and tries to find whether it is real or fake. N can be of any size. It can be smaller than the original image and it is still able to produce high-quality results. The discriminator is applied convolutionally across the whole image. Also, because the discriminator is smaller i.e. it has fewer parameters compared to the generator, it is faster.

PatchGAN can effectively model the image as a Markov random field, where NxN is considered an independent patch. Therefore, PatchGAN can be understood as a form of texture/style loss.

**Loss function**

The loss function is:



The equation above has two components: one for the discriminator and the other for the generator. Let’s understand both of them one by one.

In any GAN, the discriminator is trained first in every iteration so that it can recognize both real and fake data. Essentially,

D(x,y) = 1 i.e. real and,

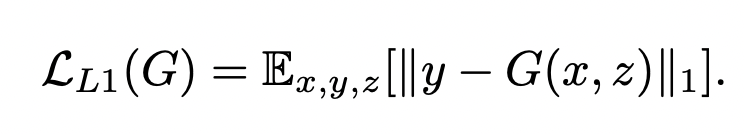
D(x,G(z)) = 0 i.e. fake.

It is worth noting that G(z) will also produce fake samples and thus its value will be closer to zero. In theory, the discriminator should always classify G(z) as zero only. Therefore the discriminator should maintain a maximum distance between real and fake i.e. 1 and 0 in every iteration. In other words, the discriminator should maximize the loss function.

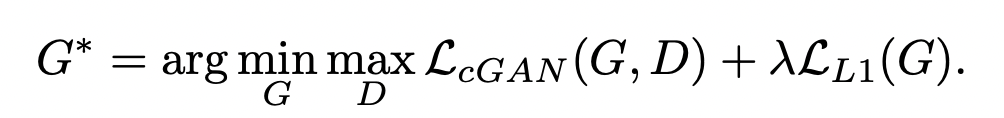
After the discriminator, the generator is trained. The generator i.e. G(z) should learn to produce samples that are closer to the real samples. To learn the original distribution it takes help from the discriminator i.e. instead of D(x, G(z)) = 0, we change D(x, G(z)) = 1.

With the alteration in labeling, the generator now optimizes its parameter concerning the parameter belonging to the discriminator with ground truth labels. This step ensures that the generator can now yield samples that are close to real data i.e. 1.

The loss function is also mixed with an L1 loss so that the generator not only fools the discriminator but also produces images near the ground truth. In essence, the loss function has an additional L1 loss for the generator.



Therefore, the final loss function is:



It is worth noting that the L1 loss can preserve low-frequency details in the image, but it will not be able to capture high-frequency details. Hence, it will still produce blurry images. To tackle this problem, PatchGAN is used.

**Optimization**

The optimization and training process is similar to vanilla GAN. However, the training itself is a difficult process since the objective function of GAN is more concave-concave rather than convex-concave. Because of this, it is difficult to find a saddle point and this is what makes training and optimizing the GANs difficult.

As we saw previously, the generator is not trained directly but through the discriminator. This essentially limits the optimization of the generator. If the discriminator fails to capture high dimensional spaces then it is very certain that the generator will fail to produce good samples. On the other hand, if we can train the discriminator in a much more optimal way then we can be assured that the generator will be trained optimally as well.

In the early stages of training, G is untrained and weak to produce good samples. This makes the discriminator very powerful, so instead of minimizing log(1 − D(G(z))), the generator is trained to maximize log D(G(z)). This creates some sort of stability in the early stages of the training.

Other ways to tackle the instability are:

* Using spectral normalization in every layer of the model
* Using Wasserstein loss which calculates the average score for real or fake images.

**Source:**

<https://neptune.ai/blog/gan-loss-functions>

<https://neptune.ai/blog/pix2pix-key-model-architecture-decisions>