Special Applications

- Face Verification and Recognition
- Neural Style Transfer
- Generative Adversarial Networks

Face Verification and Recognition

- **Face Verification** - "is this the claimed person?".

  - For example, at some airports, you can pass through customs by letting a system scan your passport and then **verifying** that you (the person carrying the passport) are the correct person.
  - A cell phone that unlocks using your face is also using **face verification**. This is a **1:1 matching problem**.

- **Face Recognition -** "who is this person?".

For example, the video lecture showed a face recognition video of Baidu employees entering the office without needing to otherwise identify themselves. This is a **1:K matching problem**.

Using a CNN to recognize (classify) persons is not a promising approach because:

- you don't have a large database with lots of images for each person
- **if a new person should be recognized, the CNN should be trained again**

One-shot learning: learning from one example to recognize the person again

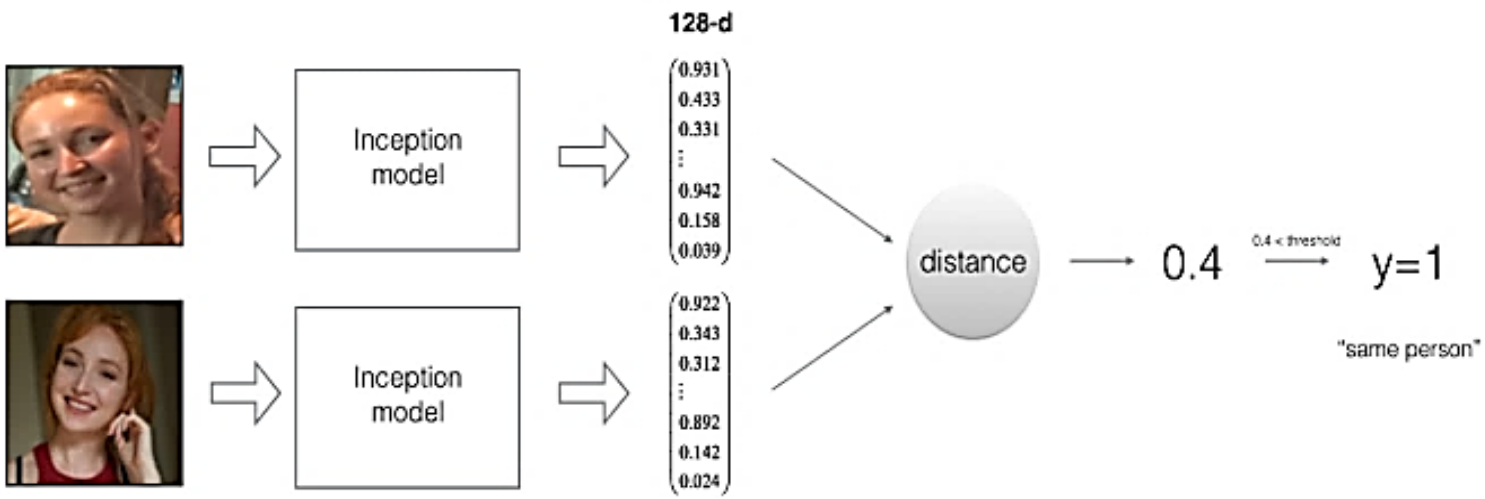
Learning a "similarity" function

d(img1, img2) = degree of difference between two images

Face verification

If d(img1, img2) ≤ threshold; same person

If d(img1, img2) > threshold; different person

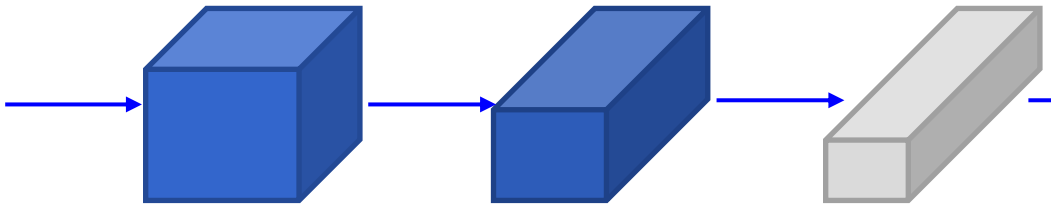


Face recognition Repeat Face verification for all images in the database

[Andrew Ng, Face recognition & Neural style transfer, https://www.coursera.org/learn/convolutional-neural-networks/home/week/4]

Siamese network

$x^{(1)}$



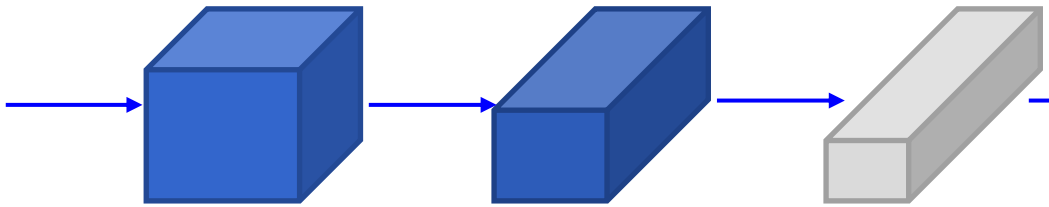
$f(x^{(1)})$

128 features

Same CNN to extract 128 features

Siamese network

$x^{(2)}$



$f(x^{(2)})$

By using a 128-neuron fully connected layer as its last layer, the model ensures that the output is an **encoding vector** of size 128.

$$d(x^{(1)}, x^{(2)}) = \left\| f(x^{(1)}) - f(x^{(2)}) \right\|_2^2$$

Parameters of CNN define the encoding $f(x^{(i)})$

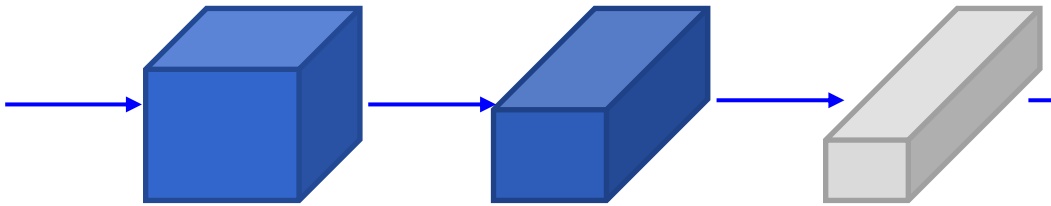
Learn parameters of CNN so that: $d(x^{(1)}, x^{(2)})$ { is **small** for the **same person**; is **large** for **different persons**

[Andrew Ng, Face recognition & Neural style transfer, https://www.coursera.org/learn/convolutional-neural-networks/home/week/4]

[Y. Taigman, M. Yang, M. Ranzato and L. Wolf, "DeepFace: Closing the Gap to Human-Level Performance in Face Verification," 2014 IEEE Conference on Computer Vision and Pattern Recognition, Columbus, OH, 2014, pp. 1701-1708, doi: 10.1109/CVPR.2014.220.]

Siamese network

$x^{(1)}$

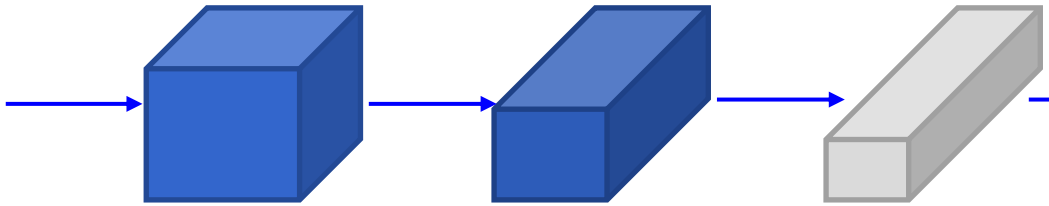


$f(x^{(1)})$

Same CNN to extract 128 features
Siamese network

128 features

$x^{(2)}$



$f(x^{(2)})$

Learn parameters of CNN so that: $d(x^{(1)}, x^{(2)})$ { is **small** for the **same person**; is **large** for **different persons**

Siamese Neural Networks use one shot classification which means that only one example for each class is needed to train the model.

[Andrew Ng, Face recognition & Neural style transfer, https://www.coursera.org/learn/convolutional-neural-networks/home/week/4]

[Y. Taigman, M. Yang, M. Ranzato and L. Wolf, "DeepFace: Closing the Gap to Human-Level Performance in Face Verification," 2014 IEEE Conference on Computer Vision and Pattern Recognition, Columbus, OH, 2014, pp. 1701-1708, doi: 10.1109/CVPR.2014.220.]

Learning objective - Triplet loss function

So, an encoding is a good one if:

- The encodings of two images of the **same person are quite similar** to each other.
- The encodings of two images of **different persons are very different.**

The **triplet loss function** formalizes this and tries

- to "push" the encodings of two images of the same person (Anchor and Positive) closer together,
- while "pulling" the encodings of two images of different persons (Anchor, Negative) further apart.

Anchor (A)

Positive (P)

Anchor (A)

Negative (N)

Training will use triplets of images (A, P, N):

- A is an "Anchor" image (reference image) - a picture of a person.
- P is a "Positive" image - a picture of the same person as the Anchor image.
- N is a "Negative" image - a picture of a different person than the Anchor image.

These triplets are picked from our training dataset. We will write $(A^{(i)}, P^{(i)}, N^{(i)})$ to denote the i-th training example.

You'd like to make sure that an image $A^{(i)}$ of an individual is closer to the Positive $P^{(i)}$ than to the Negative image $N^{(i)}$) by at least a margin α:

$$|| f(A^{(i)}) - f(P^{(i)}) ||_2^2 + \alpha < || f(A^{(i)}) - f(N^{(i)}) ||_2^2$$

You would thus like to minimize the following "triplet cost":

$$J = \sum_{i=1}^{m} \Big[\underbrace{|| f(A^{(i)}) - f(P^{(i)}) ||_2^2}_{(1)} - \underbrace{|| f(A^{(i)}) - f(N^{(i)}) ||_2^2}_{(2)} + \alpha \Big]_+$$

Here, we are using the notation "$[z]_+$" to denote $max(z, 0)$.

[F. Schroff, D. Kalenichenko and J. Philbin, "FaceNet: A unified embedding for face recognition and clustering," 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Boston, MA, 2015, pp. 815-823, doi: 10.1109/CVPR.2015.7298682]

[Andrew Ng, Face recognition & Neural style transfer, https://www.coursera.org/learn/convolutional-neural-networks/home/week/4]

Today's face recognition systems, especially the commercial ones, are trained on **very large datasets**.

Datasets of a million images are not uncommon, some companies are using 10 million images, and some companies 100 million images.

These are huge datasets, even by modern standards, and they are not easy to acquire.

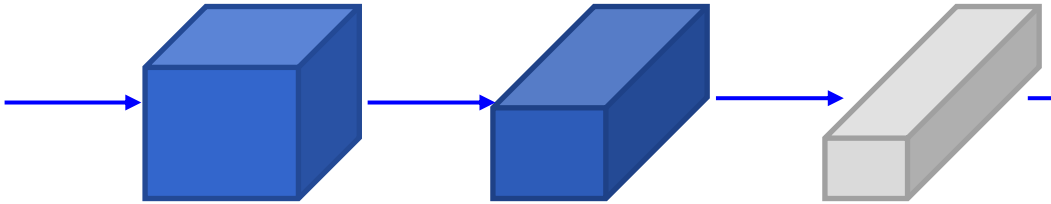
Fortunately, some of these companies have trained these large networks and posted parameters online.

So, rather than trying to train one of these networks from scratch, this is one domain where because of the share data volume sizes, it might be useful to **download someone else's pre-trained model**, rather than do everything from scratch yourself.

[Andrew Ng, Face recognition & Neural style transfer, https://www.coursera.org/learn/convolutional-neural-networks/home/week/4]

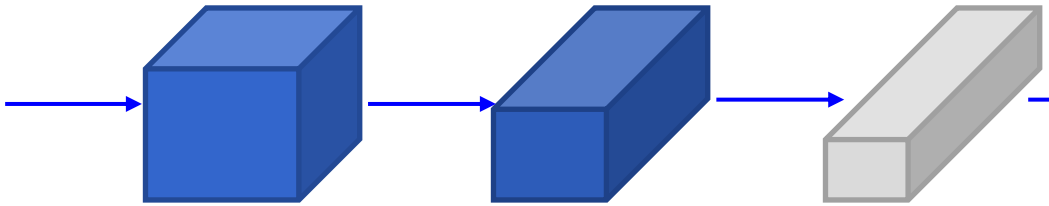
Face verification as a binary classification problem

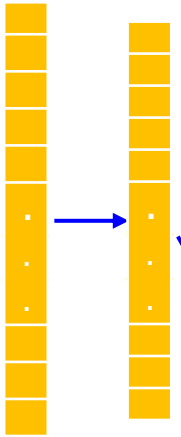
$x^{(1)}$

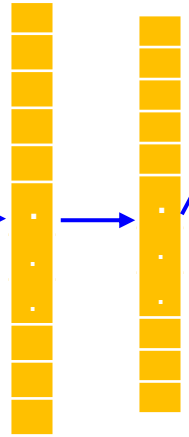


Siamese network

$x^{(2)}$







$\hat{y}$

Logistic regression unit

$$\hat{y} = \begin{cases} 1 \\ 0 \end{cases}$$

	x	y
Training dataset:		1
		0

[Andrew Ng, Face recognition & Neural style transfer, https://www.coursera.org/learn/convolutional-neural-networks/home/week/4]

[Y. Taigman, M. Yang, M. Ranzato and L. Wolf, "DeepFace: Closing the Gap to Human-Level Performance in Face Verification," 2014 IEEE Conference on Computer Vision and Pattern Recognition, Columbus, OH, 2014, pp. 1701-1708, doi: 10.1109/CVPR.2014.220.]

Neural style transfer

Neural style transfer is an optimization technique used to take two images:

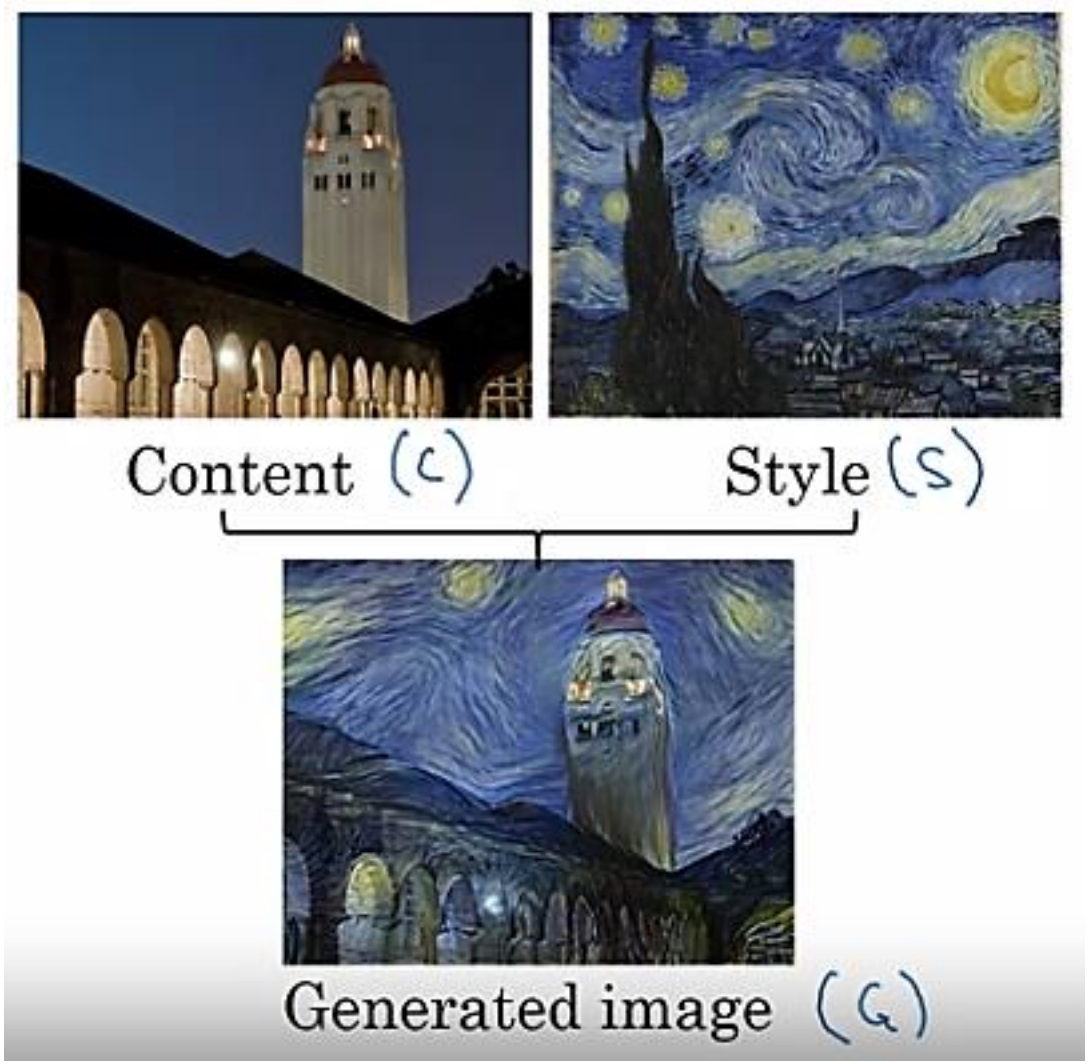
- ✓ a ***content*** image
- ✓ a ***style reference*** image (such as an artwork by a famous painter)

and blend them together so

the output image looks like the content image, but "painted" in the style of the style image.

[Andrew Ng, Face recognition & Neural style transfer, https://www.coursera.org/learn/convolutional-neural-networks/home/week/4]

Neural style transfer



Vincent Vann Gogh style

Starry Night

[Andrew Ng, Face recognition & Neural style transfer, https://www.coursera.org/learn/convolutional-neural-networks/home/week/4]

In Neural Style Transfer, **it optimizes a cost function to get the pixel values**!

Neural Style Transfer (NST) is one of the most fun techniques in deep learning. It merges two images, namely:

a **"content" image (C) and a "style" image (S),**

to create a "generated" image (G).

The generated image G combines the "content" of the image C with the "style" of image S.

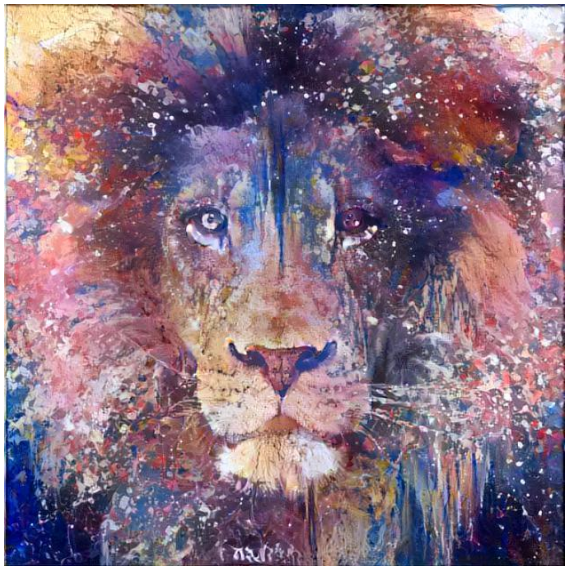
Neural Style Transfer (NST) uses a **previously trained convolutional network** and builds on top of that.

The idea of using a network trained on a different task and applying it to a new task is called transfer learning.

The model has already been trained on the very large database, and thus has learned to recognize a variety of

- low-level features (at the shallower layers)
- high-level features (at the deeper layers).

Create Something Amazing - NightCafe Creator

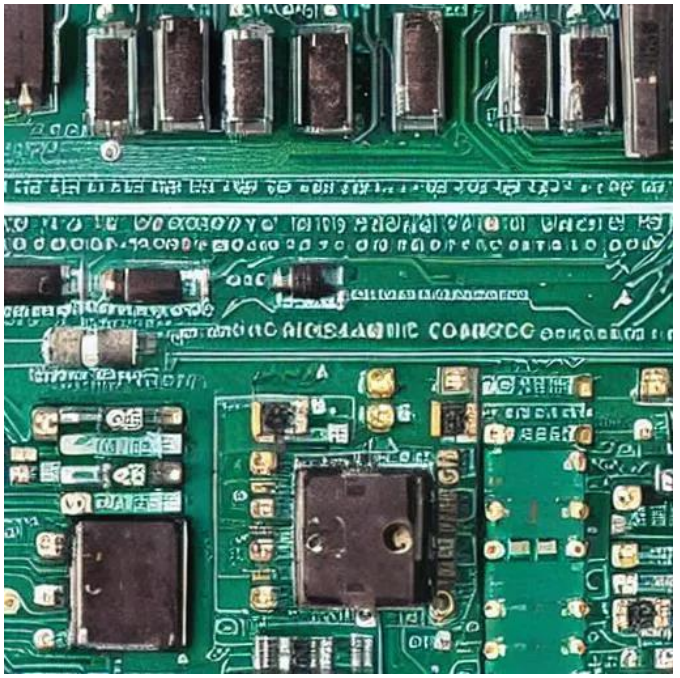


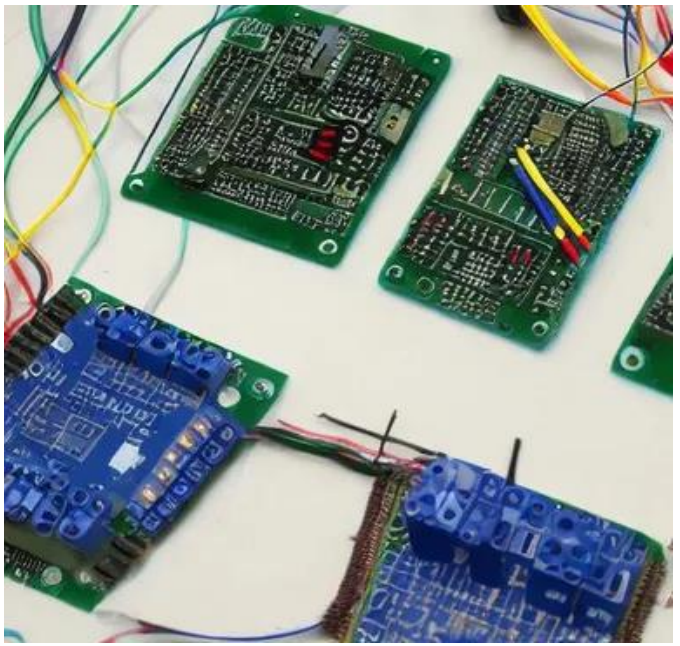
Content **Style** **Generated image**

Text + Model image =>
New image created by AI

Electronic circuit

+

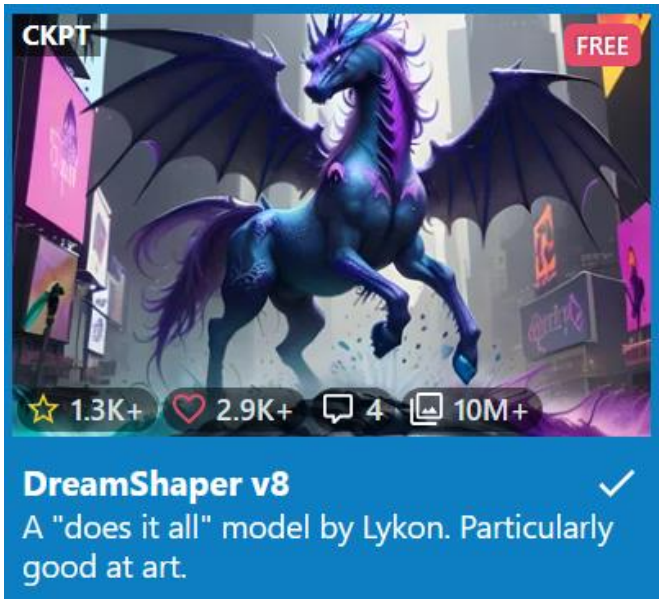


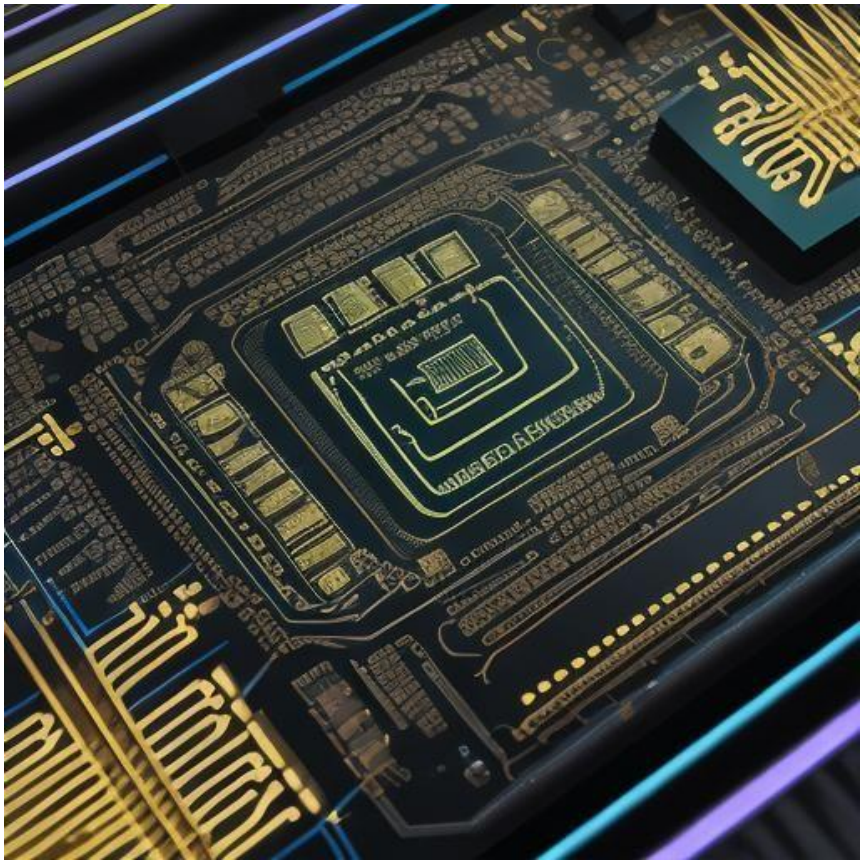


[Create Something Amazing - NightCafe Creator]

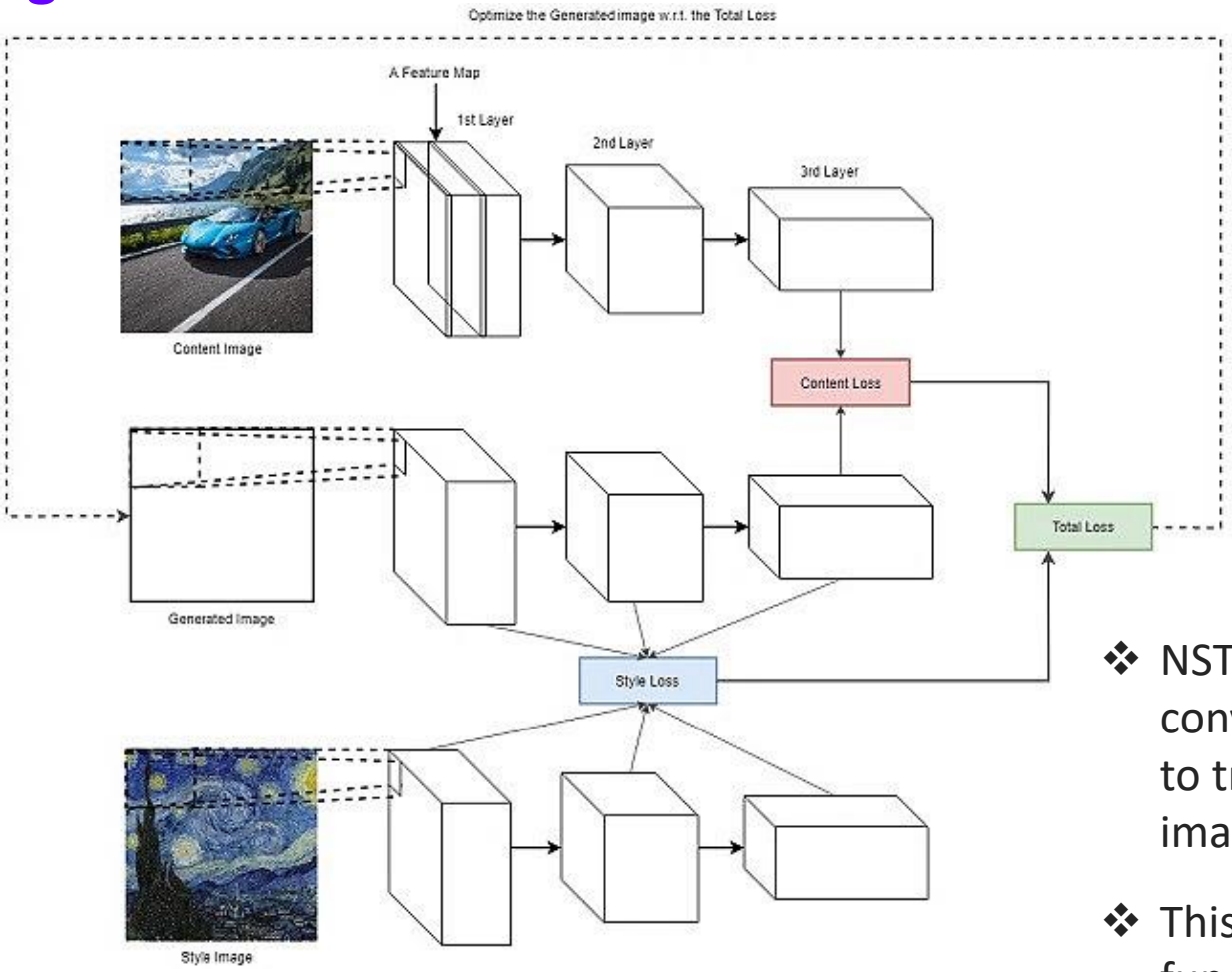
Electronic circuit

+





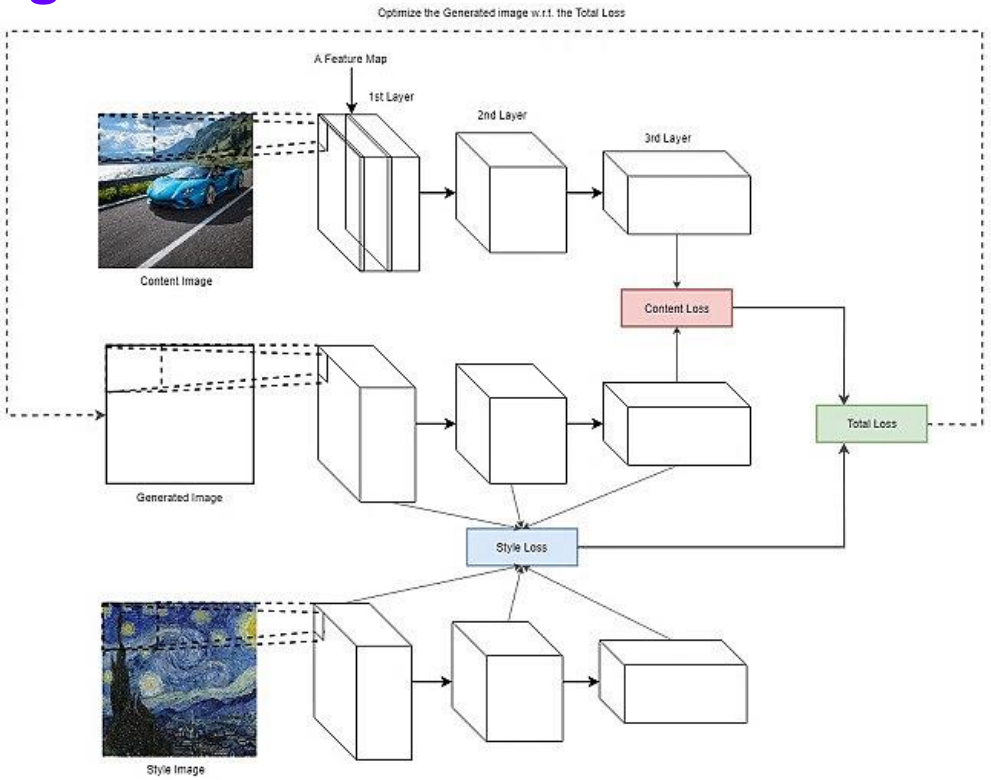
High level architecture of a NST model



Thushan Ganegedara, Intuitive Guide to Neural Style Transfer, Jan 8, 2019, https://towardsdatascience.com/light-on-math-machine-learning-intuitive-guide-to-neural-style-transfer-ef88e46697ee

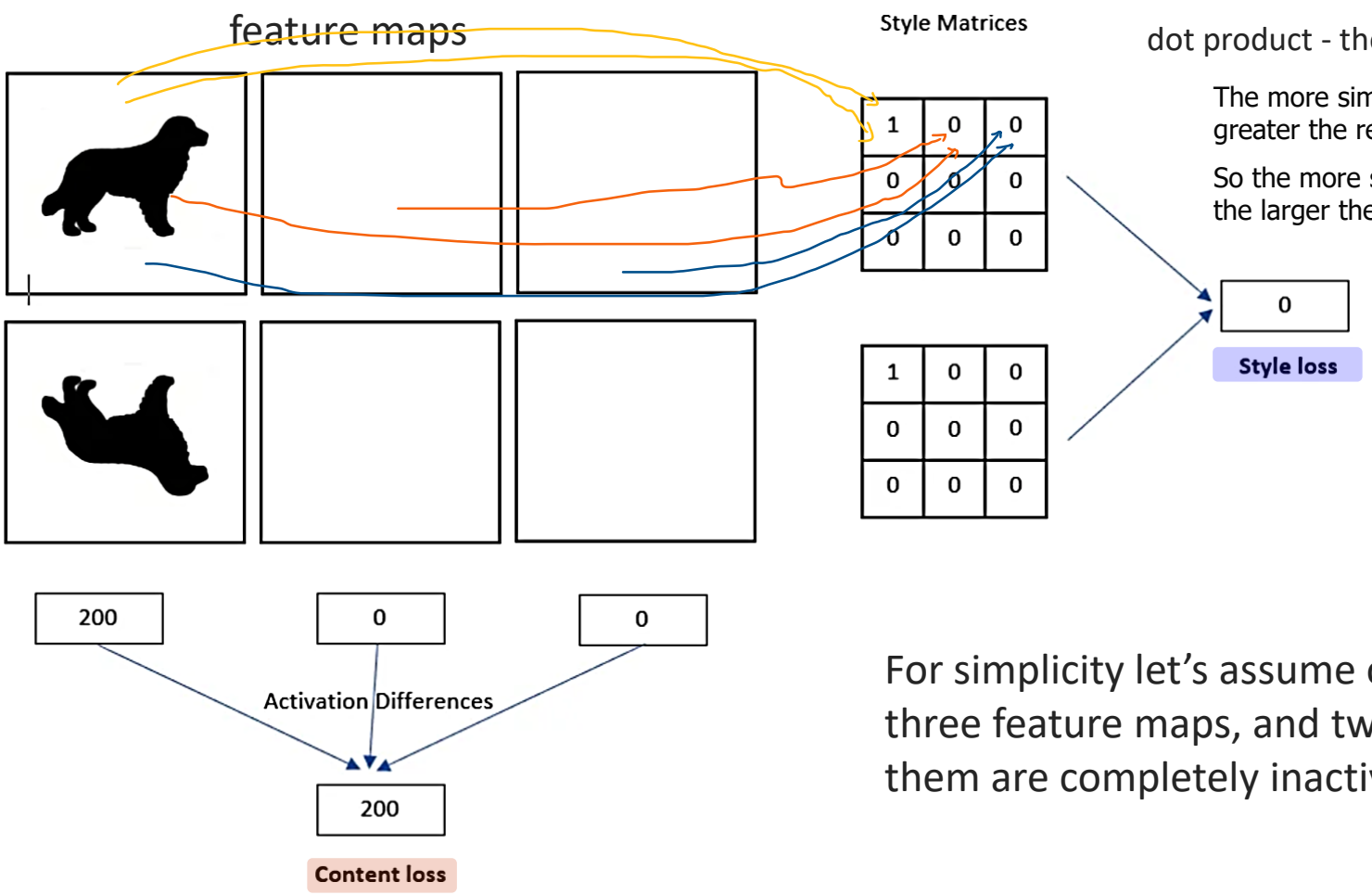
- NST employs a pretrained convolution neural network (CNN) to transfer styles from a given image to another.
- This is done by defining a loss function that tries to **minimise the differences between a content image, a style image and a generated image**.

High level architecture of a NST model



- The Content loss function ensures that the activations of the higher layers are similar between the content image and the generated image.
  - for two images to have the same content, they should have similar activations in the higher layers.

- The Style loss function makes sure that the correlation of activations in all the layers are similar between the style image and the generated image
  - style information is measured as ***the amount of correlation between feature maps*** in a given layer
  - a loss is defined as the difference of correlation between the feature maps computed by the generated image and the style image



For simplicity let's assume only three feature maps, and two of them are completely inactive.

We have one feature map set where the first feature map looks like a dog, and in the second feature map set, the first feature map looks like a dog upside down.

- we haven't lost style information between two feature map sets (Style loss = 0).
- the content is quite different. (Content loss = 200)

Generative Adversarial Networks (GANs)

[Jason Brownlee, A Gentle Introduction to Generative Adversarial Networks (GANs), July 19, 2019, https://machinelearningmastery.com/what-are-generative-adversarial-networks-gans/]

- Generative modeling is an **unsupervised learning task** in machine learning
- It involves automatically **discovering and learning the regularities or patterns in input data** in such a way that the model can be used to
  - **generate new examples that plausibly could have been drawn from the original dataset.**

- GANs are a clever way of **training a generative model** by framing the problem as a **supervised learning problem with two sub-models**:
  - **the generator model** that we train to **generate new examples**,
  - **the discriminator model** that tries to classify examples as either **real** (from the domain) or **fake** (generated).

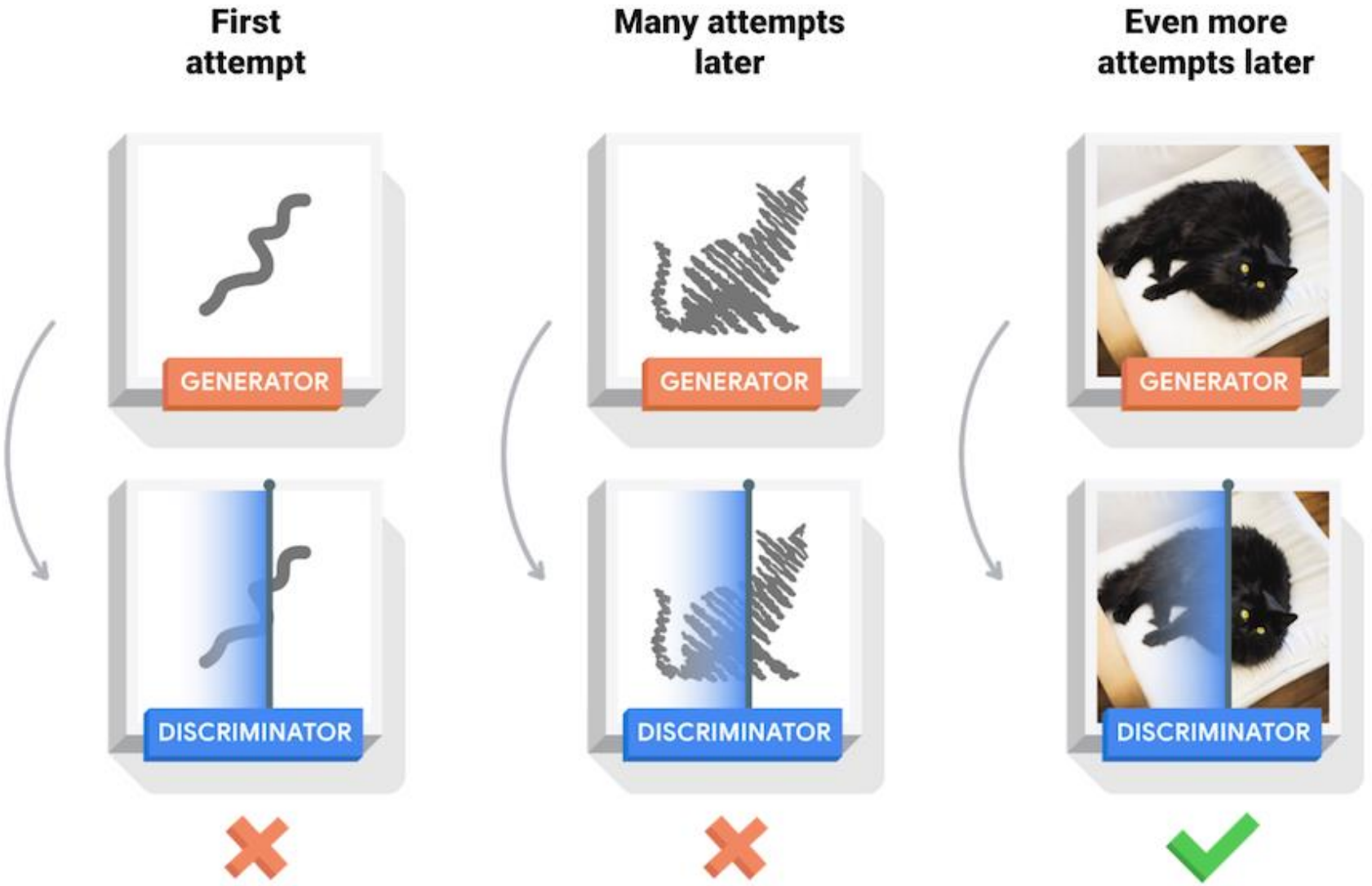
- The two models are **trained together in a zero-sum game, adversarial**, until the discriminator model is fooled about half the time, meaning the **generator model is generating plausible examples**.

GENERATOR
"The Artist"
A neural network trying to create pictures of cats that look real.

DISCRIMINATOR
"The Art Critic"
A neural network examining cat pictures to determine if they're real or fake.

Thousands of real-world images labeled "CAT"

[dcgan.ipynb - Colaboratory (google.com)]

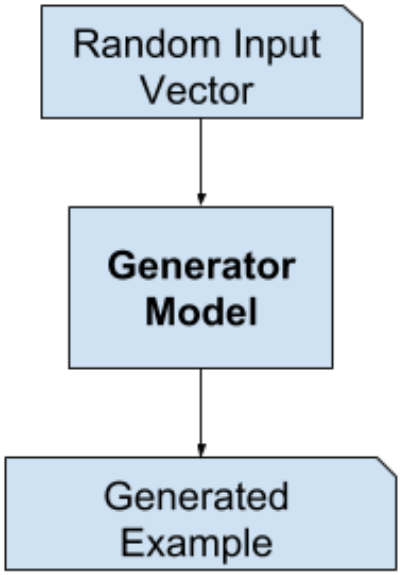

First attempt
Many attempts later
Even more attempts later
GENERATOR
DISCRIMINATOR
GENERATOR
DISCRIMINATOR
GENERATOR
DISCRIMINATOR

The Generator Model

The generator model takes a fixed-length random vector as input and generates a sample in the domain.

The vector is drawn from a Gaussian distribution, and the vector is used to seed the generative process.

After training, points in this multidimensional vector space will correspond to points in the problem domain, forming a compressed representation of the data distribution.



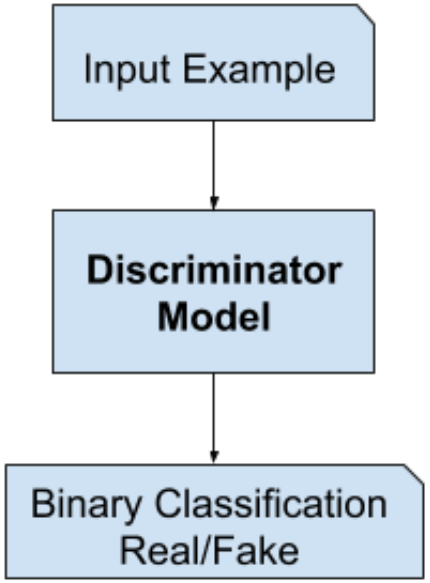
The Discriminator Model

The discriminator model takes an example from the domain as input (real or generated) and predicts a binary class label of real or fake (generated).

The real example comes from the training dataset. The generated examples are output by the generator model.

The discriminator is a normal (and well understood) classification model.

After the training process, the discriminator model is discarded as we are interested in the generator.



GANs as a Two Player Game

Generative modeling is an unsupervised learning problem, although a clever property of the GAN architecture is that the training of the generative model is framed as a supervised learning problem.

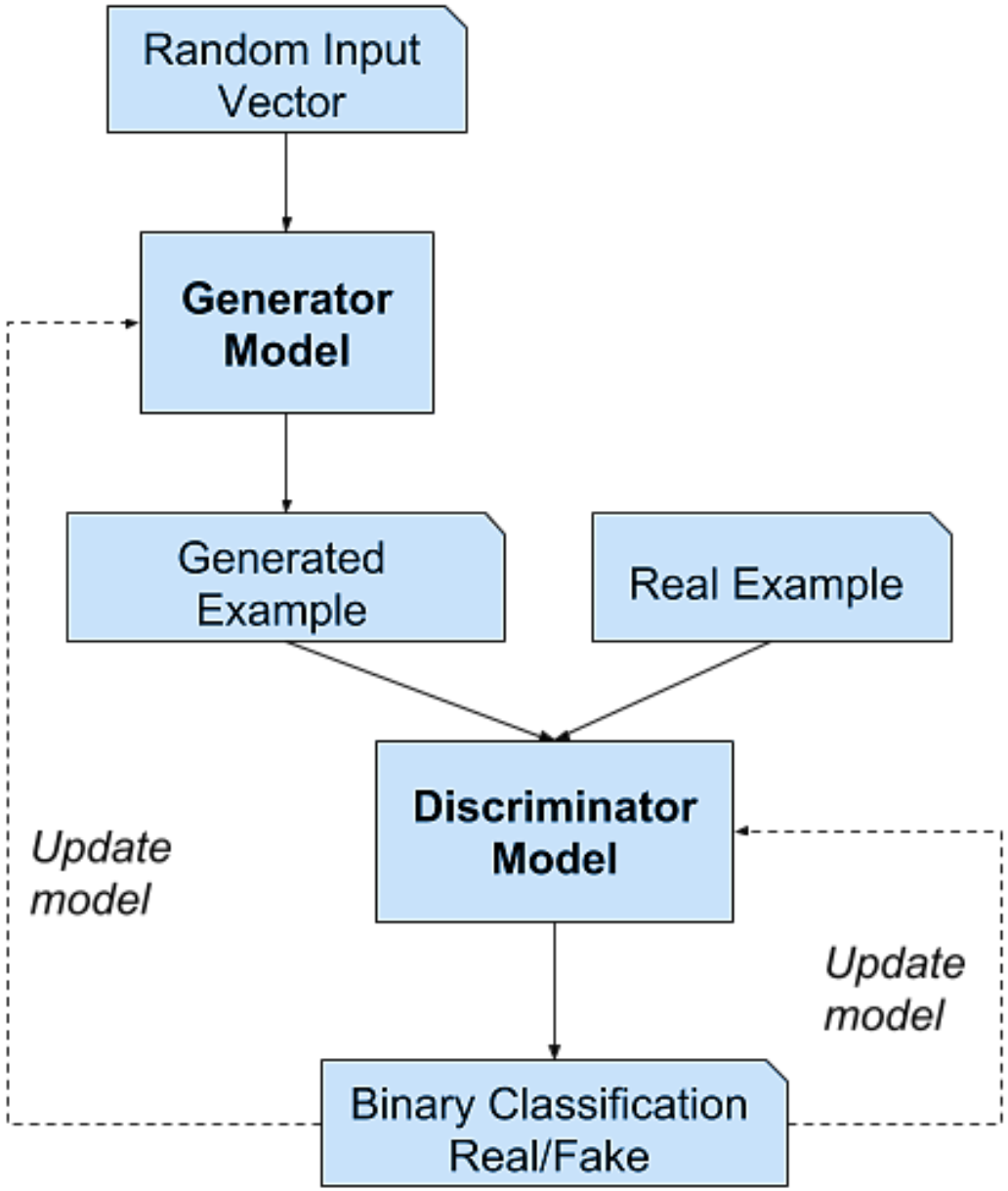
The two models, the generator and discriminator, are trained together.

The generator generates a batch of samples, and these, along with real examples from the domain, are provided to the discriminator and classified as real or fake.

The discriminator is then updated to get better at discriminating real and fake samples in the next round, and importantly, the generator is updated based on how well, or not, the generated samples fooled the discriminator.

1. The training loop begins with generator receiving a random seed as input.
2. That seed is used to produce an image.
3. The discriminator is then used to classify real images (drawn from the training set) and fakes images (produced by the generator).
4. The loss is calculated for each of these models, and the gradients are used to update the generator and discriminator.

Example of the GAN Model Architecture



In this case, zero-sum means:

- ✓ when the discriminator successfully identifies real and fake samples, it is rewarded or no change is needed to the model parameters, whereas the generator is penalized with large updates to model parameters.

- ✓ when the generator fools the discriminator, it is rewarded, or no change is needed to the model parameters, but the discriminator is penalized, and its model parameters are updated.

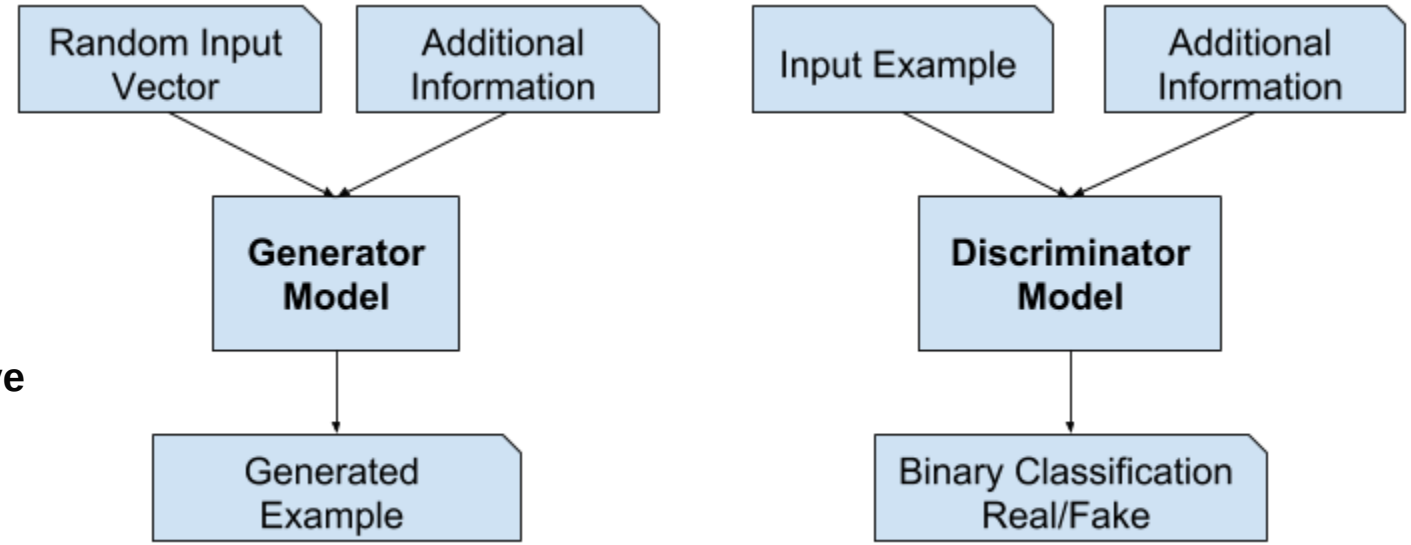
Conditional GANs

An important extension to the GAN is in their use for **conditionally generating an output.**

The generative model can be trained to generate new examples from the input domain, where the input is provided with (conditioned by) some additional input.

The additional input could be a class value, such as male or female in the generation of photographs of people, or a digit, in the case of generating images of handwritten digits.

Example of a Conditional Generative Adversarial Network Model Architecture



- Perhaps the most compelling application of GANs is in conditional GANs for tasks that require the generation of new examples. Jan Goodfellow indicates three main examples:

• **Image Super-Resolution**: generates high-resolution versions of input images.

• **Creating Art**: creates new and artistic images, sketches, painting, and more.

• **Image-to-Image Translation**: translate photographs across domains, such as day to night, summer to winter, and more.

- GANs have been able to generate photos so realistic that humans are unable to tell that they are of objects, scenes, and people that do not exist in real life.

Example of the Progression in the Capabilities of GANs From 2014 to 2017. Taken from The Malicious Use of Artificial Intelligence: Forecasting, Prevention, and Mitigation, 2018.

Why Generative Adversarial Networks?

One of the many major advancements in the use of deep learning methods in domains such as computer vision is a technique called **data augmentation.**

Data augmentation results in better performing models, both increasing model skill and providing a regularizing effect, reducing generalization error.
It works by creating new, artificial but plausible examples from the input problem domain on which the model is trained.

The techniques are primitive in the case of image data, involving crops, flips, zooms, and other simple transforms of existing images in the training dataset.

Successful generative modeling provides an alternative and potentially more domain-specific approach for data augmentation.
In fact, data augmentation is a simplified version of generative modeling, although it is rarely described this way.

GAN Lab. Play with Generative Adversarial Networks (GANs) in your browser!

https://poloclub.github.io/ganlab/

Deep Convolutional Generative Adversarial Network

https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tutorials/generative/dcgan.ipynb