LEMANS SCHOOL OF AI | SESSION 4.03

SESSION D'INITITATION AUX

GENERATIVE ADVERSARIAL NETWORKS

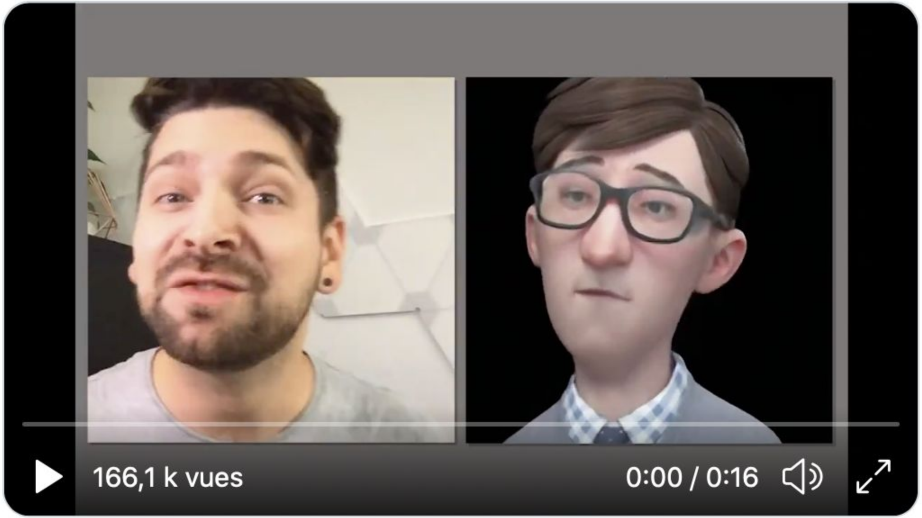
JEUDI 15 OCTOBRE 2020 | 18H30 | LE MANS INNOVATION

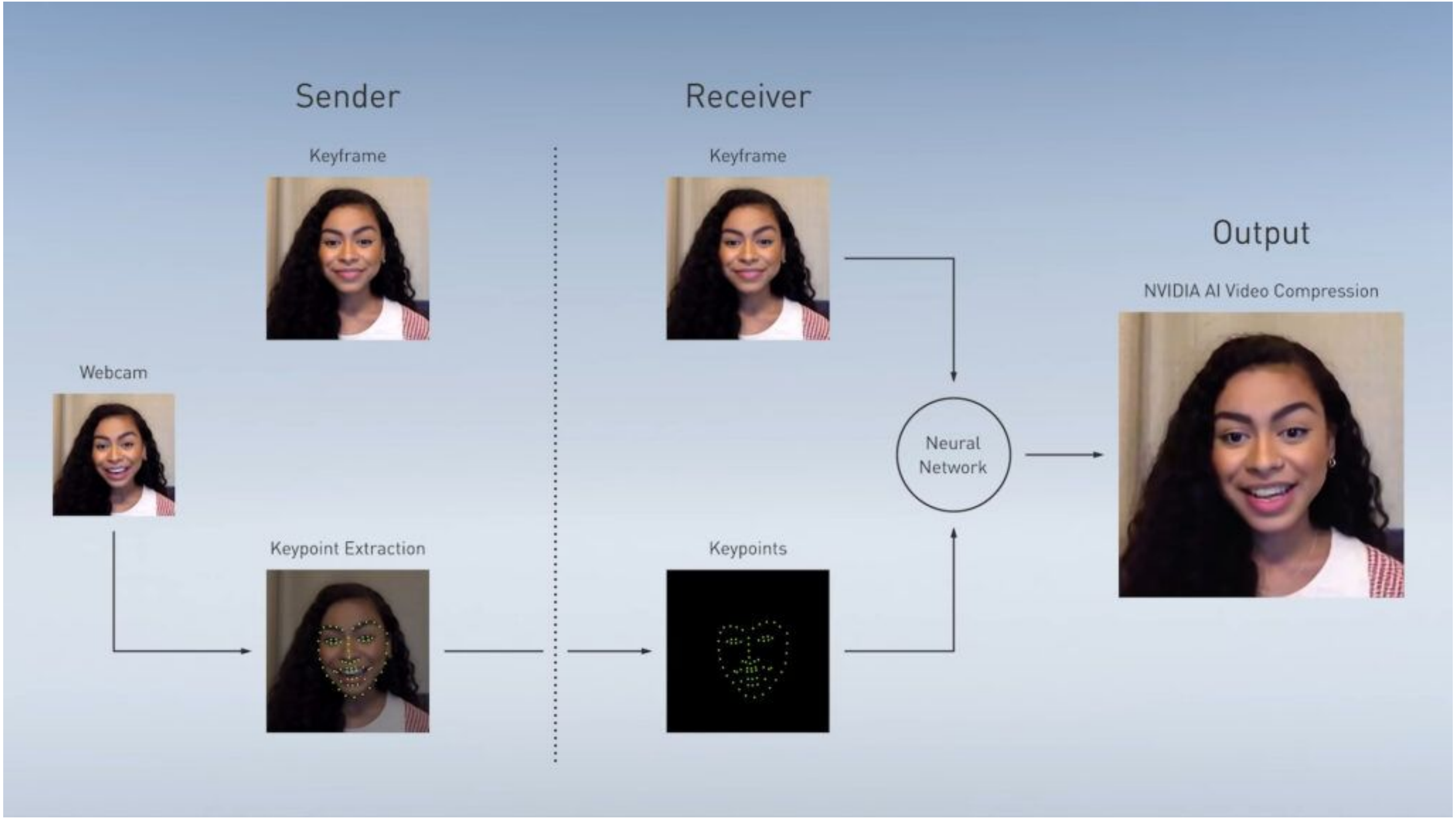
LEMANS
SCHOOLOFAI

ARTIFICIAL INTELLIGENCE STUDYGROUP MADE BY DOERS FOR DOERS IN LE MANS

NVIDIA @nvidia · 7 oct.

NVIDIA Maxine will be making an appearance on @BBCClick with @thisisFoxx this weekend! Can't wait that long? Get a sneak peek of how this new #AI platform transforms video conferencing in the #GTC20 keynote: nvda.ws/3jEH5Zn




Sender
Receiver
Keyframe
Keyframe
Output
NVIDIA AI Video Compression
Webcam
Neural Network
Keypoint Extraction
Keypoints

Sommaire

1/ intro (5 min)

2/ exemples d’application (15 min)

3/ comment ça marche (25 min)

4/ un programme exemple (15 min)

5/ quelques considérations (15min)

intro

La technologie **GAN** est introduite en 2014

par **Ian Goodfellow** (OpenAI Institute -> Google -> Apple (directeur ML))

Yann LeCun

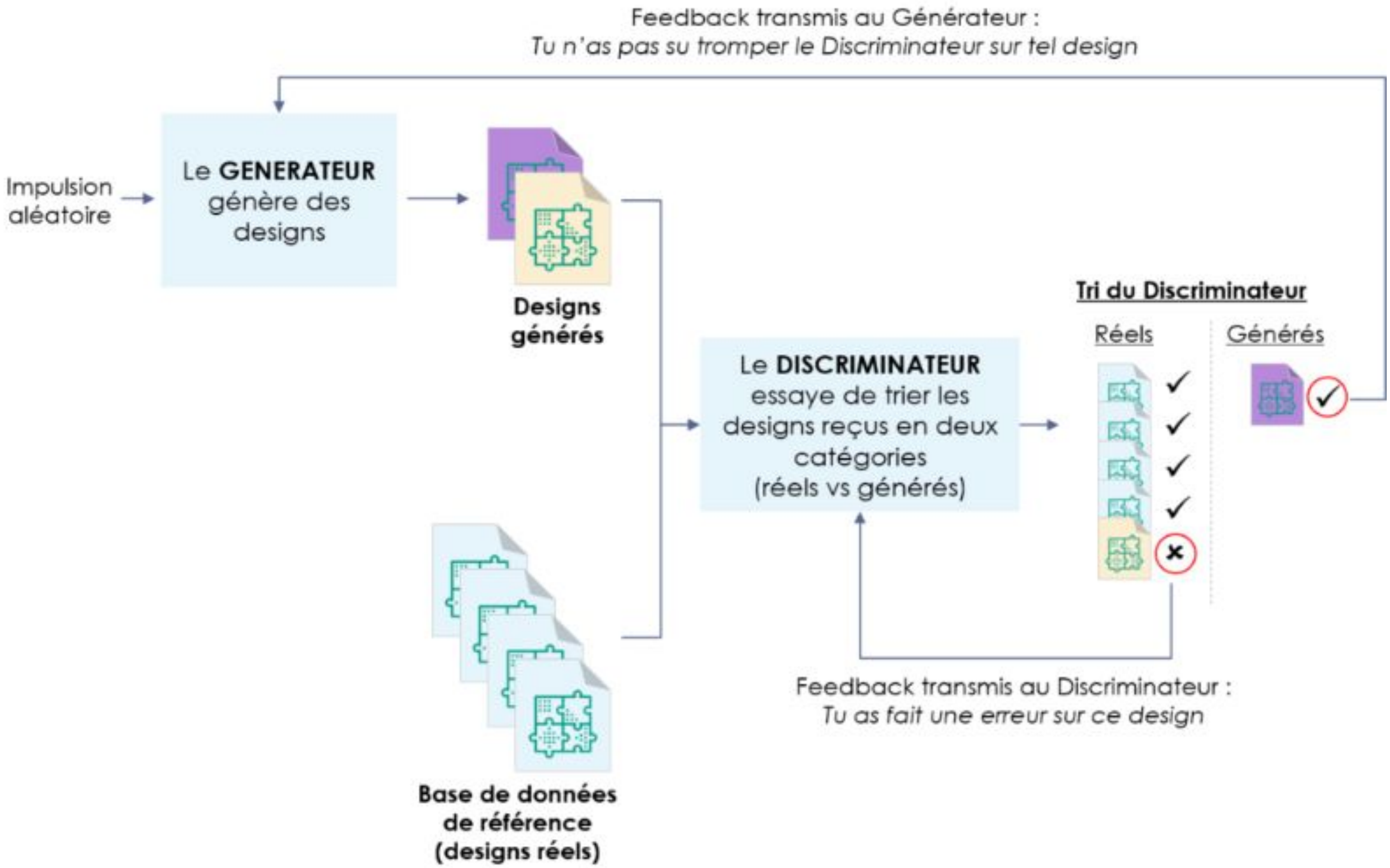

« l’idée la plus intéressante des 10 dernières années dans le domaine du Machine Learning ».

de quoi s'agit-il ?

- **GAN** = type de modèle de DL

- la particularité : compétition entre deux réseaux de neurones :

  un **générateur** contre un **discriminateur**.

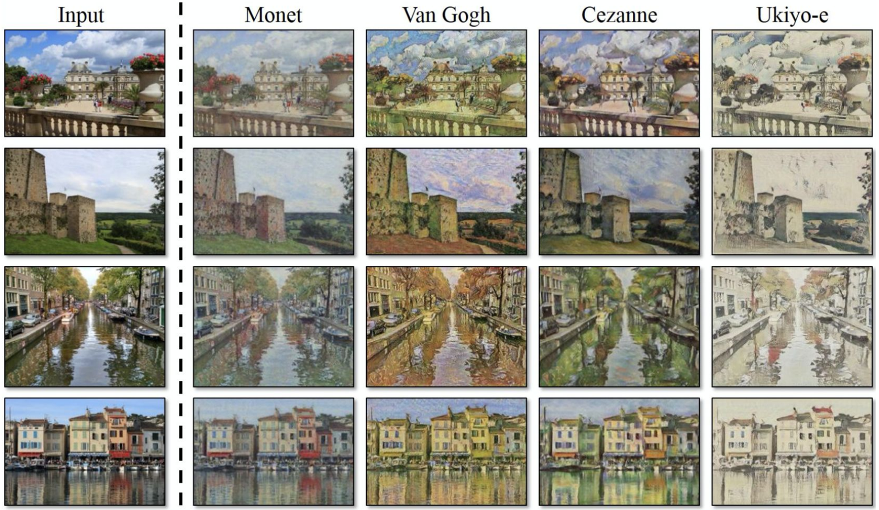

Feedback transmis au Générateur :
Tu n'as pas su tromper le Discriminateur sur tel design
Impulsion aléatoire
Le **GENERATEUR** génère des designs
Designs générés
Le **DISCRIMINATEUR** essaye de trier les designs reçus en deux catégories (réels vs générés)
Tri du Discriminateur
Réels
Générés
Base de données de référence (designs réels)
Feedback transmis au Discriminateur :
Tu as fait une erreur sur ce design

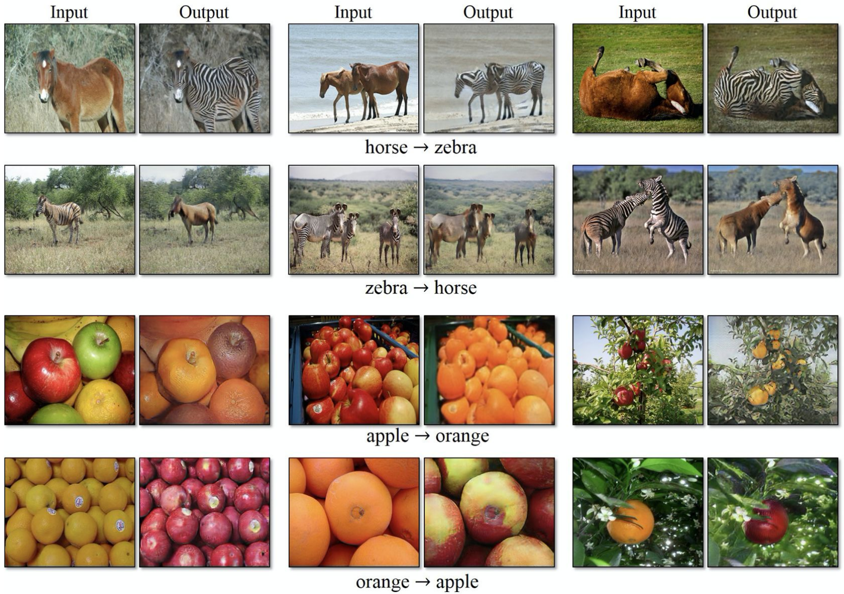
quelques exemples d’application

StyleGAN : génération d'images

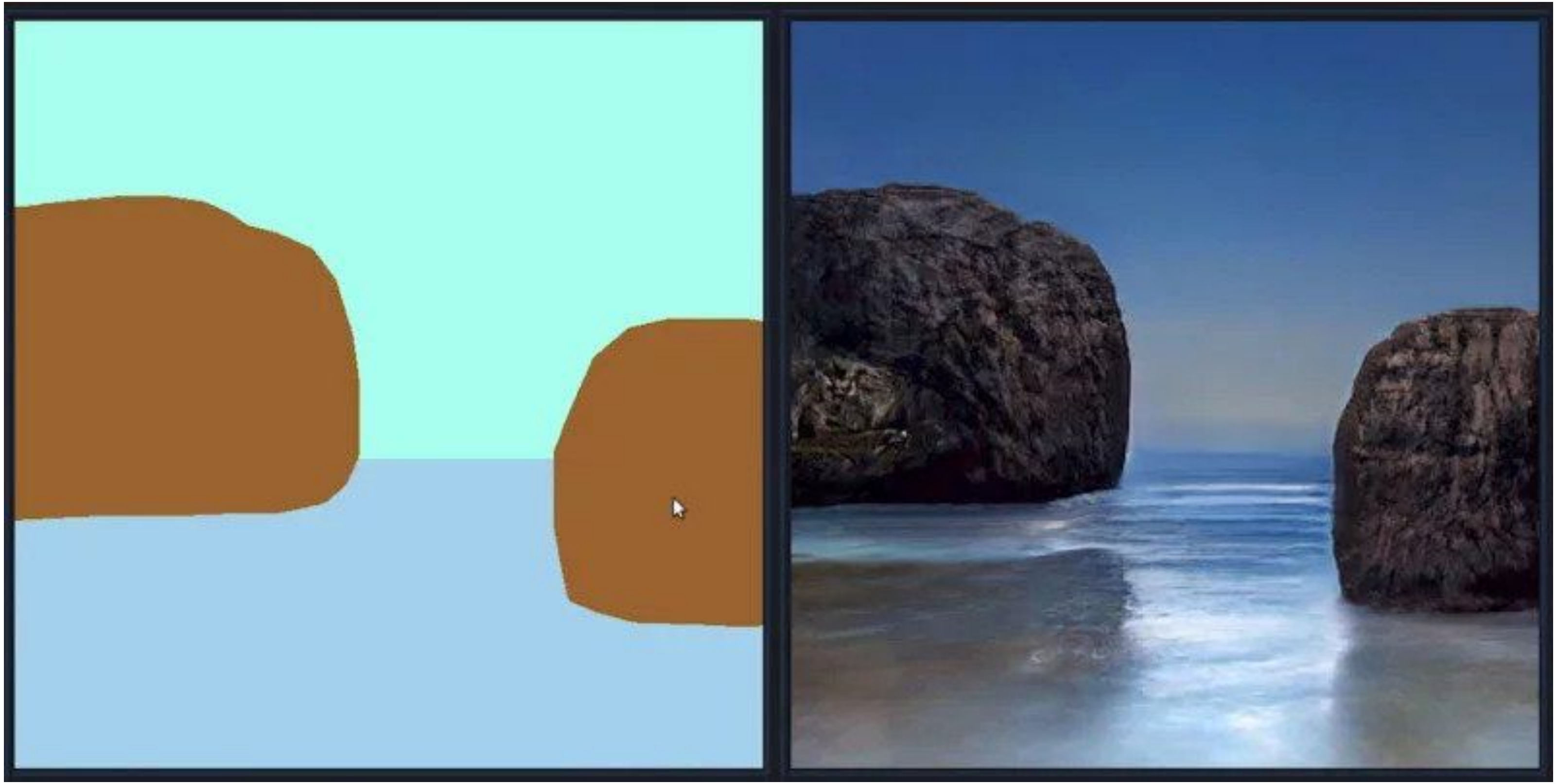
https://thispersondoesnotexist.com/

CycleGAN : transfert d'images/vidéos





GauGAN - NVIDIA




Input
Output
Input
Output

3D GAN

http://3dgan.csail.mit.edu

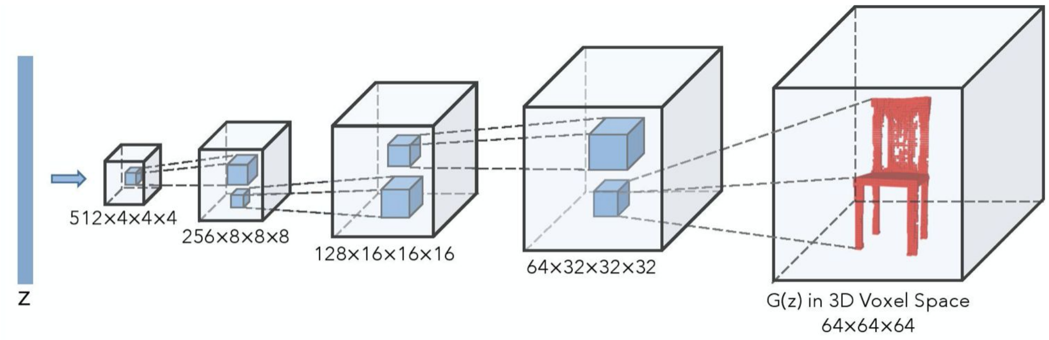


Figure 1: The generator of 3D Generative Adversarial Networks (3D-GAN)

Figure 2: Shapes synthesized by 3D-GAN

Entreprise utilisant les GANs

Adobe

Nouvelle génération de Photoshop

Augmentation de données

Superresolution

Filtres sur les images

Génération de texte

vidéo

retirer le bruit d’une image


DeepFake - Jim Carey
Ctrl Shift Face
Original - Jack Nicholson
The Shining

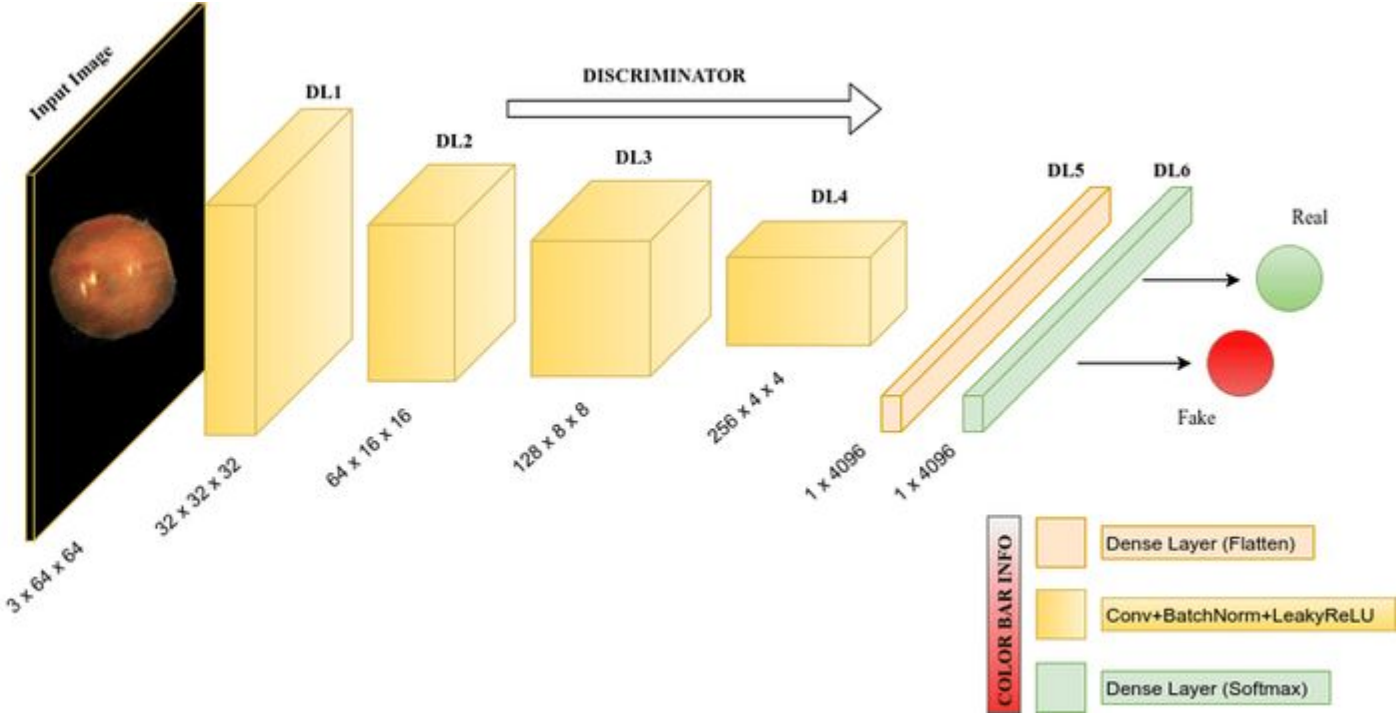
https://colourise.sg/


Pour votre santé, évitez de manger trop gras, trop sucré, trop salé. www.mangerbouger.fr
SCREENCAST MATIC

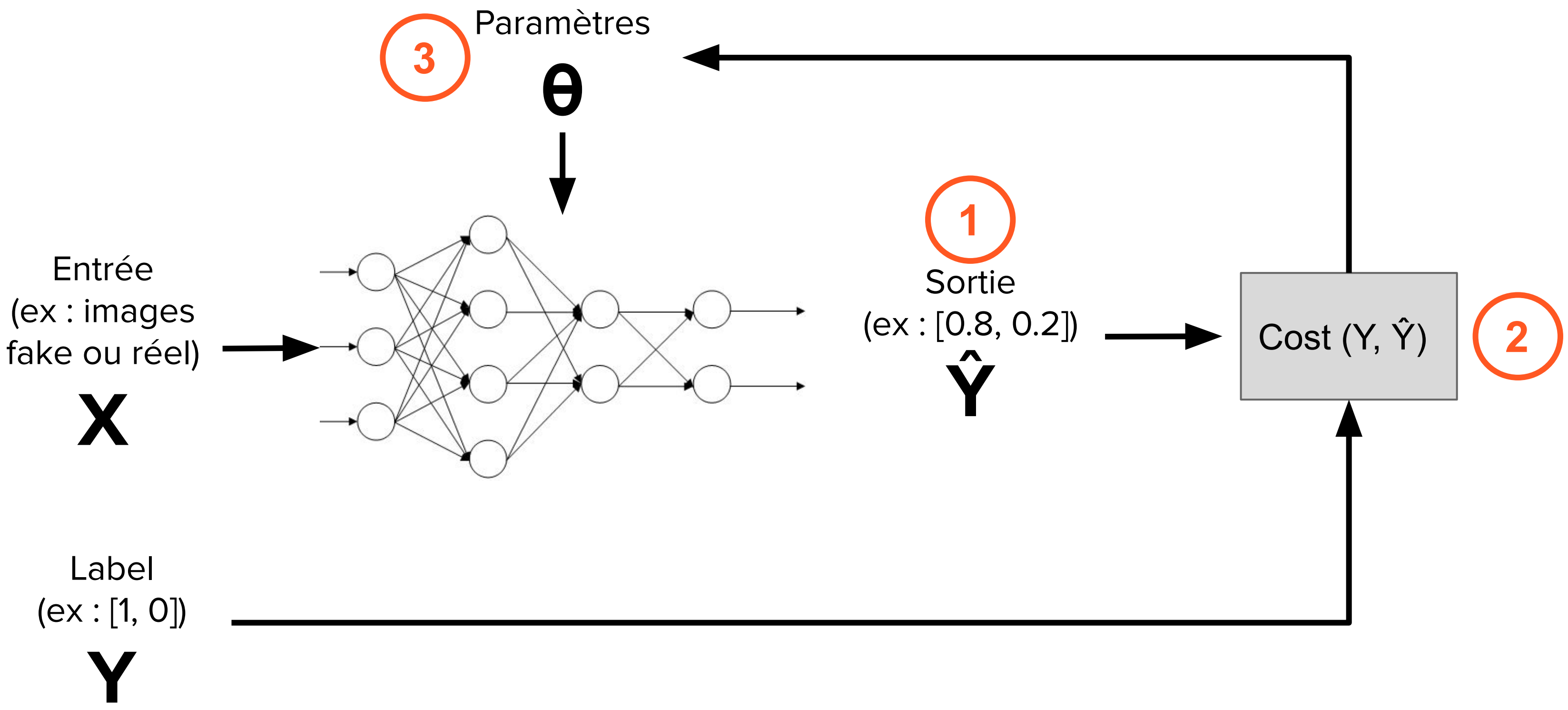

AUTO-ECOLE
Pour votre santé, mangez au moins cinq fruits et légumes par jour. www.mangerbouger.fr

comment ça marche

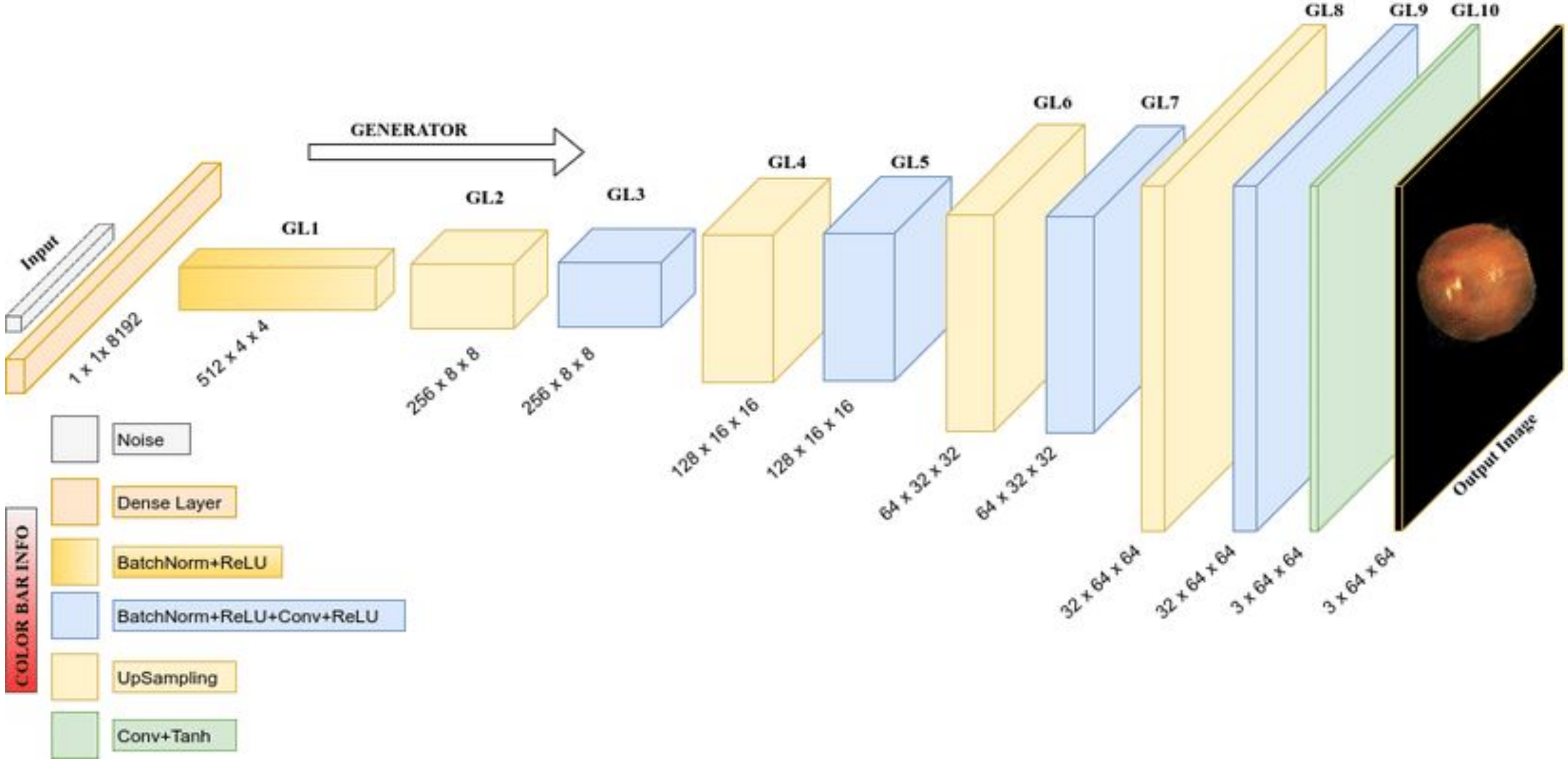
Réseaux à convolution (discriminateur)



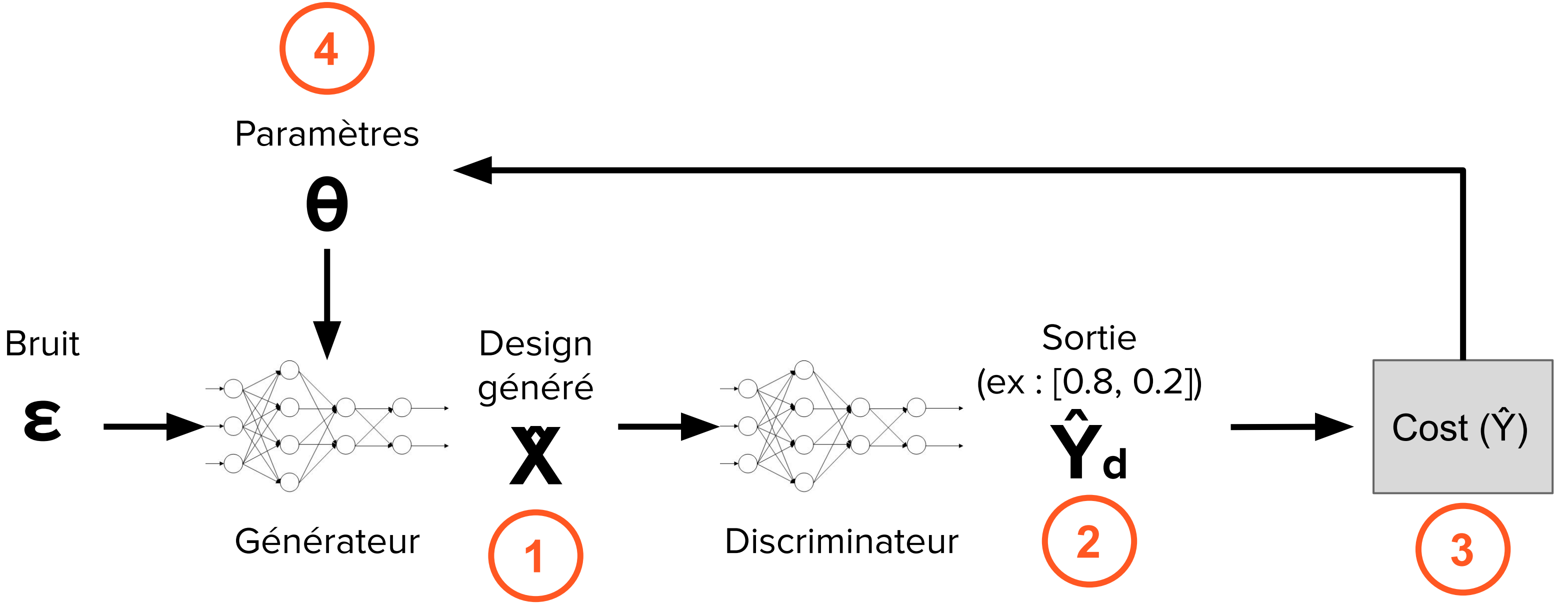
Entrainement du discriminateur



Réseaux à déconvolution (générateur)



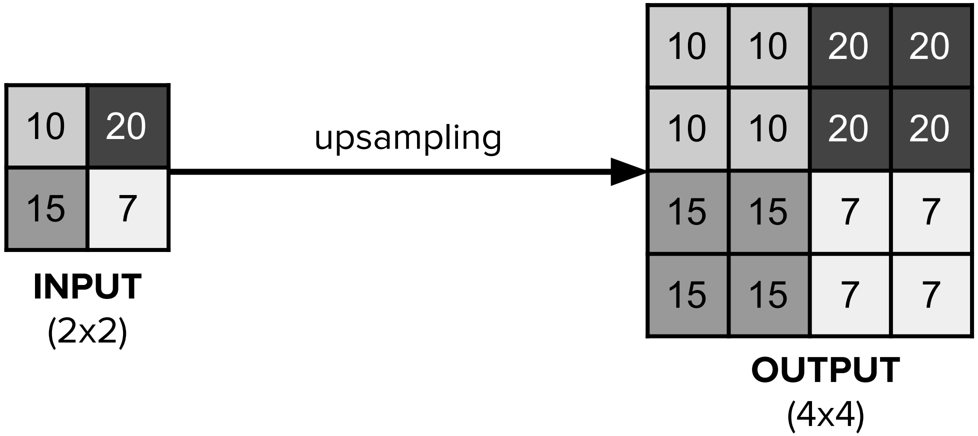
Entrainement du générateur



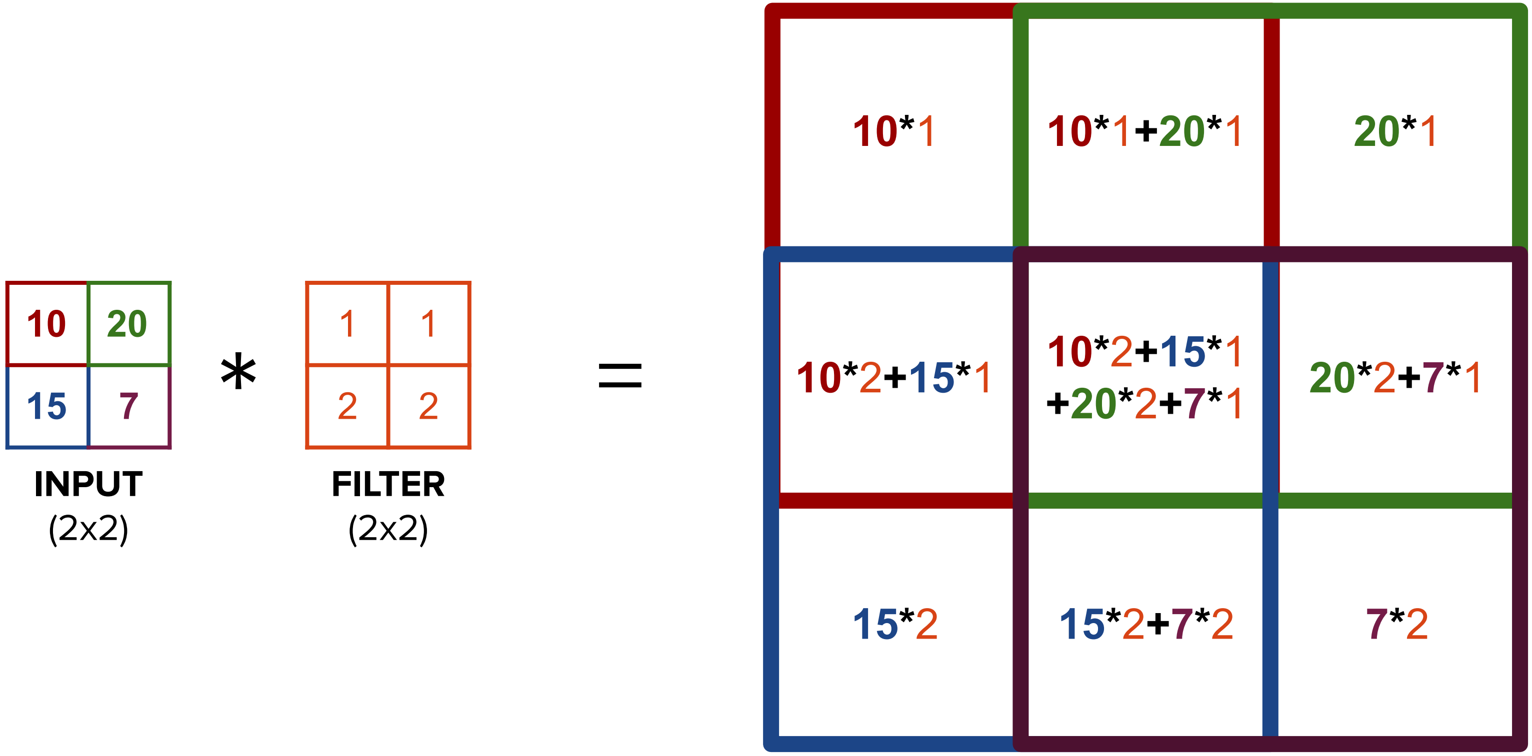
Objectif pour :

- le Générateur : Ŷd = [1, 0]
- le Discriminateur : Ŷd =[0, 1]

Upsampling techniques (le plus proche voisin)



Upsampling techniques (déconvolution)



pas = 1

Binary cross-entropy cost function

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m} \overbrace{\boxed{[y^{(i)} \log h(x^{(i)}, \theta)}}^{\text{real } y^{(i)}=1} + \overbrace{\boxed{(1 - y^{(i)}) \log(1 - h(x^{(i)}, \theta))]}}^{\text{fake } y^{(i)}=0}$$

Prediction

Label

Features

Parameters

$h(x^{(i)}, \theta)$	$\log h(x^{(i)}, \theta)$	$\log(1 - h(x^{(i)}, \theta))]$
~0	- ∞	~0
~1	~0	-∞

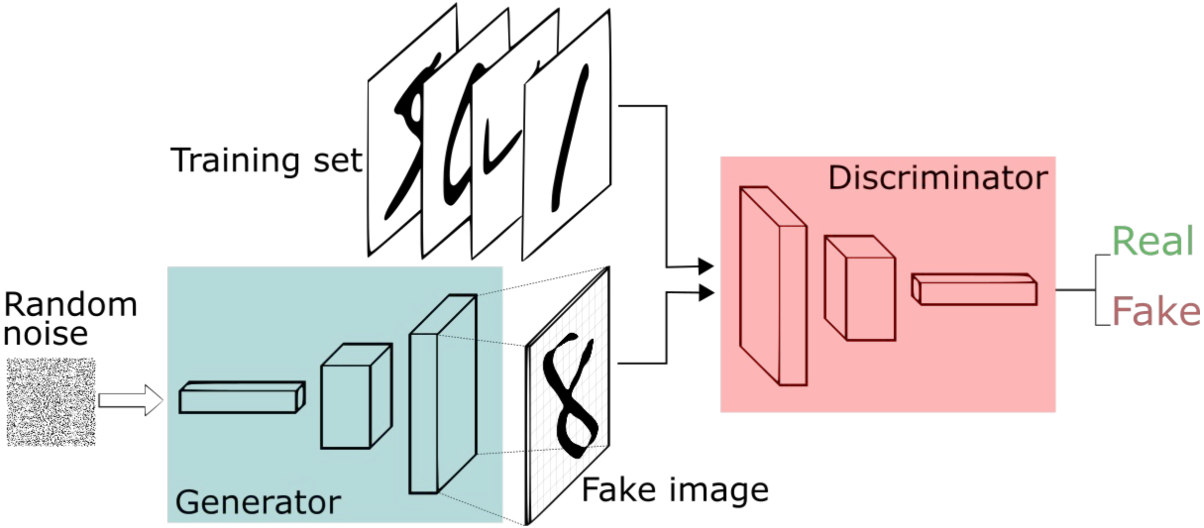
Objectif pour :

- le Générateur : J(θ) plus grand possible
- le Discriminateur : J(θ) plus proche de 0 possible

un programme exemple

mon premier GAN

- avec Keras, tensorflow
- dataset de chiffres manuscrits de MNIST
- objectif : générer une image synthétique d’un chiffre



le générateur

entrée :

100-dimensional noise

sortie :

vector of the size 784

(28x28 the original size

of the images)

```
class Generator(keras.Model):

    def __init__(self, random_noise_size = 100):
        super().__init__(name='generator')
        #layers
        self.input_layer = keras.layers.Dense(units = random_noise_size)
        self.dense_1 = keras.layers.Dense(units = 128)
        self.leaky_1 =  keras.layers.LeakyReLU(alpha = 0.01)
        self.dense_2 = keras.layers.Dense(units = 128)
        self.leaky_2 = keras.layers.LeakyReLU(alpha = 0.01)
        self.dense_3 = keras.layers.Dense(units = 256)
        self.leaky_3 = keras.layers.LeakyReLU(alpha = 0.01)
        self.output_layer = keras.layers.Dense(units=784, activation = "tanh")

    def call(self, input_tensor):
        ## Definition of Forward Pass
        x = self.input_layer(input_tensor)
        x = self.dense_1(x)
        x = self.leaky_1(x)
        x = self.dense_2(x)
        x = self.leaky_2(x)
        x = self.dense_3(x)
        x = self.leaky_3(x)
        return  self.output_layer(x)

    def generate_noise(self,batch_size, random_noise_size):
        return np.random.uniform(-1,1, size = (batch_size, random_noise_size))
```

fonction de coût pour le générateur

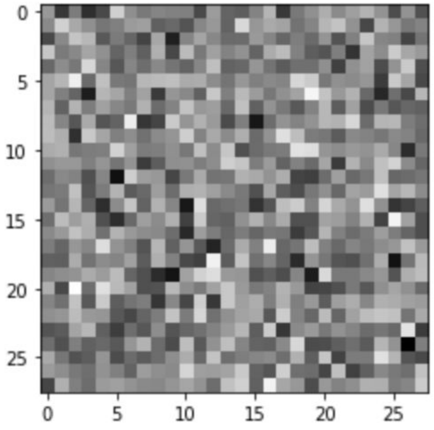
```
cross_entropy = tf.keras.losses.BinaryCrossentropy(from_logits = True)
```

```
def generator_objective(dx_of_gx):
    # Labels are true here because generator thinks he produces real images.
    return cross_entropy(tf.ones_like(dx_of_gx), dx_of_gx)
```

Génération sans entraînement

```
generator = Generator()
fake_image = generator(np.random.uniform(-1,1, size =(1,100)))
fake_image = tf.reshape(fake_image, shape = (28,28))
plt.imshow(fake_image, cmap = "gray")
```

```
<matplotlib.image.AxesImage at 0x7f65e2a7e198>
```



le discrimi -nateur

entrée :

784-dimensional

*vector (28*28 = 784)*

sortie :

1 neurone

a fake or a real image

```
class Discriminator(keras.Model):
    def __init__(self):
        super().__init__(name = "discriminator")

        #Layers
        self.input_layer = keras.layers.Dense(units = 784)
        self.dense_1 = keras.layers.Dense(units = 128)
        self.leaky_1 =  keras.layers.LeakyReLU(alpha = 0.01)
        self.dense_2 = keras.layers.Dense(units = 128)
        self.leaky_2 = keras.layers.LeakyReLU(alpha = 0.01)
        self.dense_3 = keras.layers.Dense(units = 128)
        self.leaky_3 = keras.layers.LeakyReLU(alpha = 0.01)

        # This neuron tells us if the input is fake or real
        self.logits = keras.layers.Dense(units = 1)
    def call(self, input_tensor):
          ## Definition of Forward Pass
        x = self.input_layer(input_tensor)
        x = self.dense_1(x)
        x = self.leaky_1(x)
        x = self.leaky_2(x)
        x = self.leaky_3(x)
        x = self.leaky_3(x)
        x = self.logits(x)
        return x
```

fonction de coût pour le discriminateur

```python
def discriminator_objective(d_x, g_z, smoothing_factor = 0.9):
    """
    d_x = real output
    g_z = fake output
    """
    # If we feed the discriminator with real images,
    # we assume they all are the right pictures --> Because of that label == 1
    real_loss = cross_entropy(tf.ones_like(d_x) * smoothing_factor, d_x)

    # Each noise we feed in are fakes image --> Because of that labels are 0
    fake_loss = cross_entropy(tf.zeros_like(g_z), g_z)

    total_loss = real_loss + fake_loss

    return total_loss
```

remarque : smoothing_factor is to avoid overfitting

Entrainement

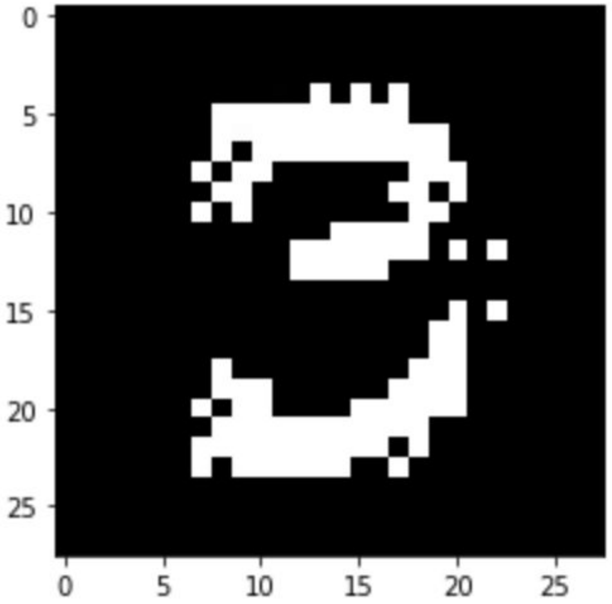
```
BATCH_SIZE = 256
BUFFER_SIZE = 60000
EPOCHES = 300
```

```
@tf.function()
def training_step(generator: Discriminator, discriminator: Discriminator, images:np.ndarray , k:int =1, batch_size = 32):
    for _ in range(k):
        with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
            noise = generator.generate_noise(batch_size, 100)
            g_z = generator(noise)
            d_x_true = discriminator(images) # Trainable?
            d_x_fake = discriminator(g_z) # dx_of_gx

            discriminator_loss = discriminator_objective(d_x_true, d_x_fake)
            # Adjusting Gradient of Discriminator
            gradients_of_discriminator = disc_tape.gradient(discriminator_loss, discriminator.trainable_variables)
            discriminator_optimizer.apply_gradients(zip(gradients_of_discriminator, discriminator.trainable_variables)) # Takes a list of gradient and variables pairs


            generator_loss = generator_objective(d_x_fake)
            # Adjusting Gradient of Generator
            gradients_of_generator = gen_tape.gradient(generator_loss, generator.trainable_variables)
            generator_optimizer.apply_gradients(zip(gradients_of_generator, generator.trainable_variables))
```

résultat

```
fake_image = generator(np.random.uniform(-1,1, size = (1, 100)))
plt.imshow(tf.reshape(fake_image, shape = (28,28)), cmap="gray")
```

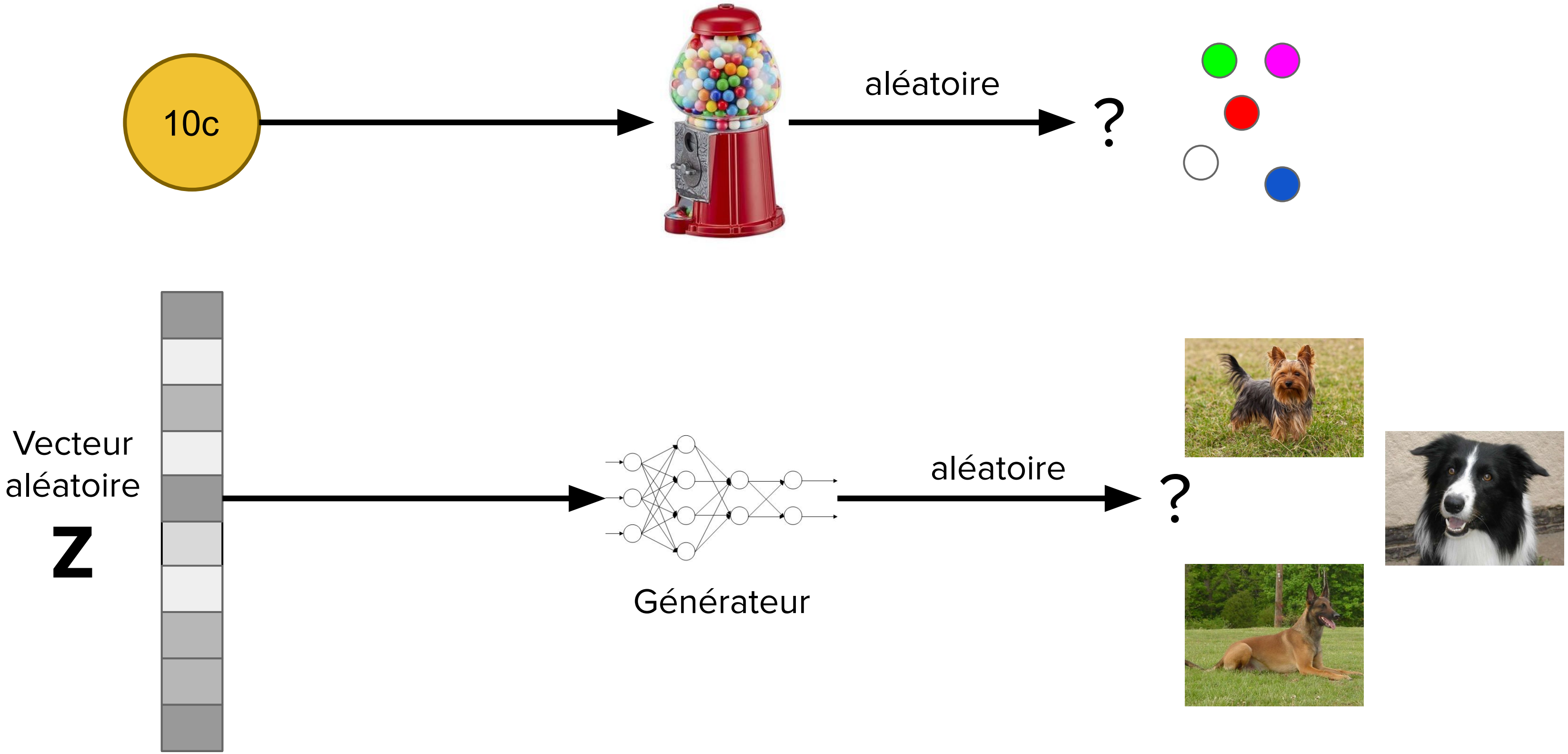
```
<matplotlib.image.AxesImage at 0x7f65e0f7db70>
```



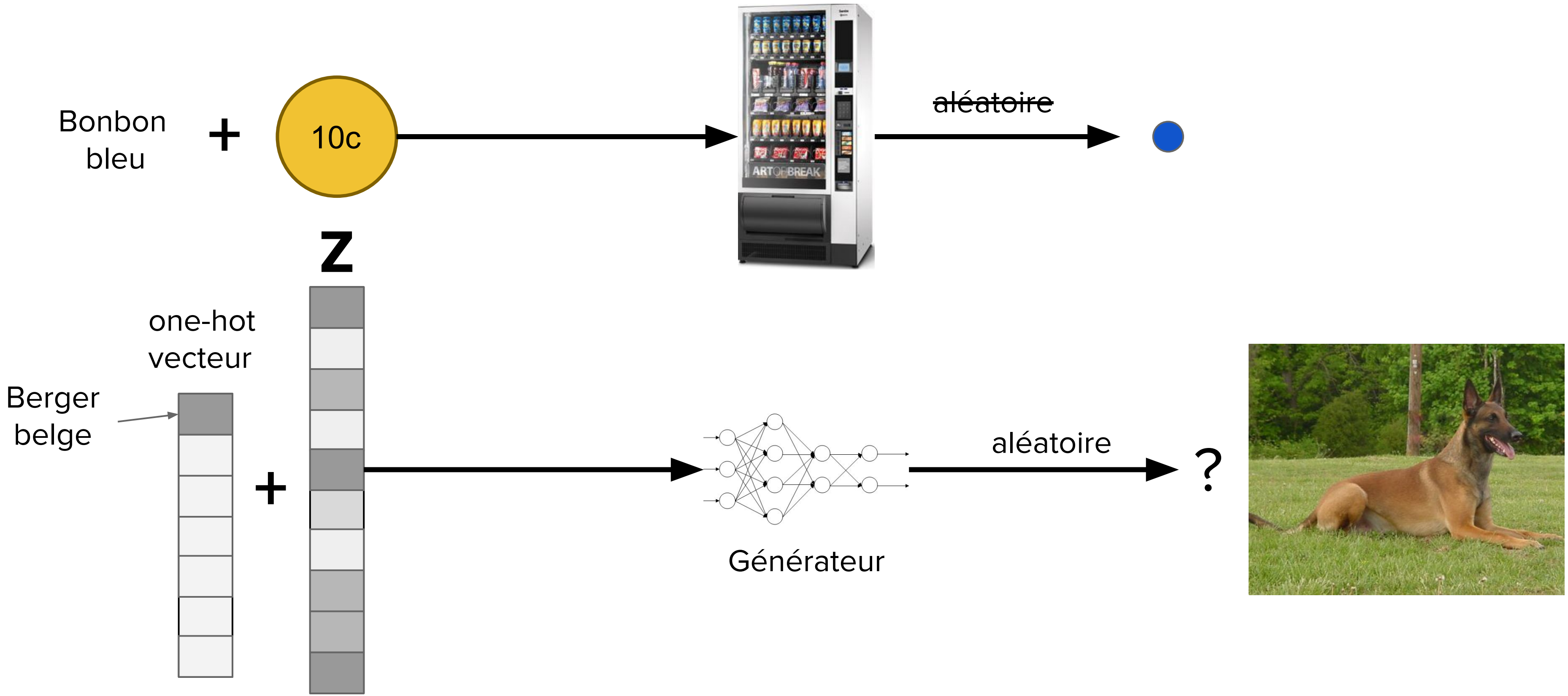
quelques considérations …

Conditional GANs

Unconditional GANs



Conditional GANs

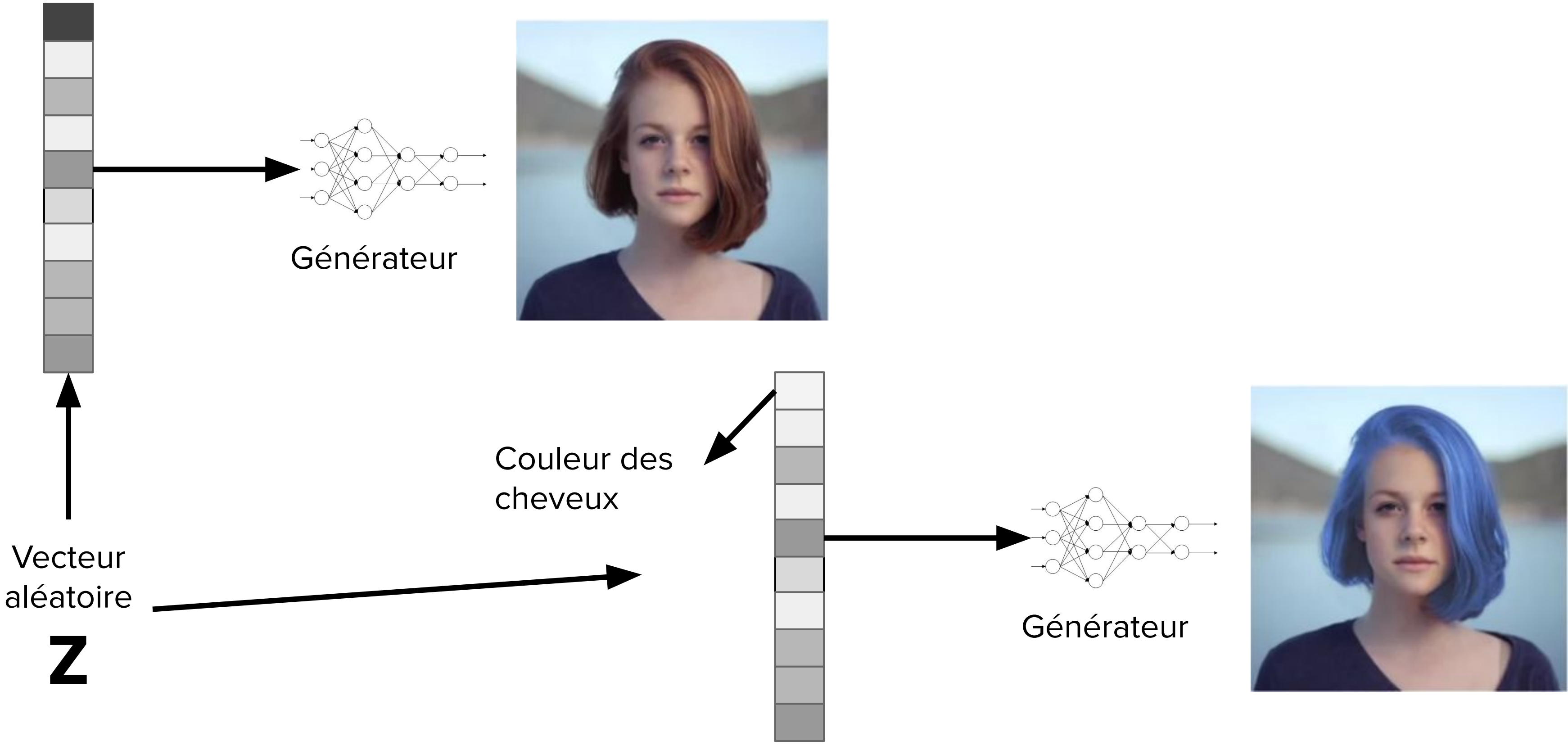


Conditional vs Unconditional GANs

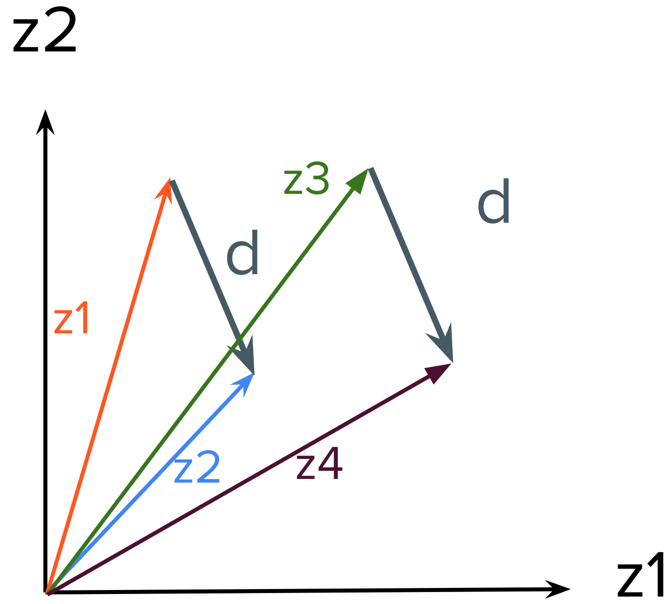
Conditional	Unconditional
• Génère la classe qu'on souhaite	• Génère une classe aléatoirement
• Les données d'entraînement doivent être annotées	• Les données d'entraînement ne doivent pas être libellées

Controllable GANs

Controllable GANs



Z-space and controllable generation



<u>Z-space avec les vecteurs aléatoires z</u>

g(z1)

g(z3)

g(z2)

g(z3+d)
= g(z4)

Controllable vs Conditional GANs

Controllable	Conditional
• Génère des images avec la caractéristique qu'on souhaite	• Génère la classe qu'on souhaite
• Les données d'entraînement ne doivent pas être annotées	• Les données d'entraînement doivent être annotées
• On manipule le vecteur ε en entrée	• On concatène le vecteur ε avec une vecteur de classe

et pour la session 2

sur la base d’un exemple, on rentrera dans le détail de l’implémentation d’un modèle :

- les limites
- les problèmes rencontrés et leurs solutions