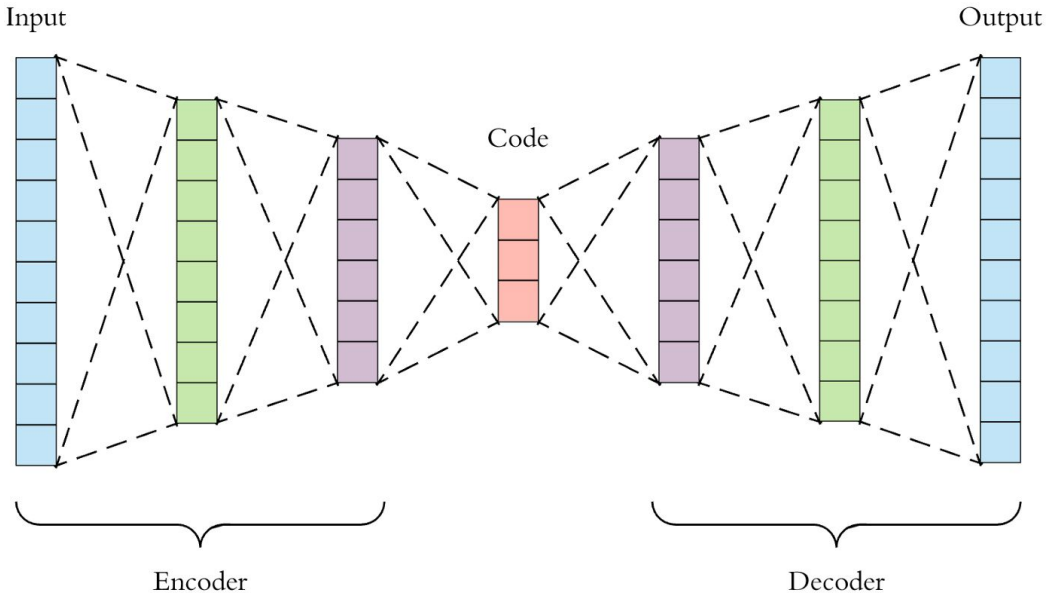
TP n°2 : Deep Learning

Generative models

Yohan ISMAEL & Théo BERILLON

Les auto encodeurs

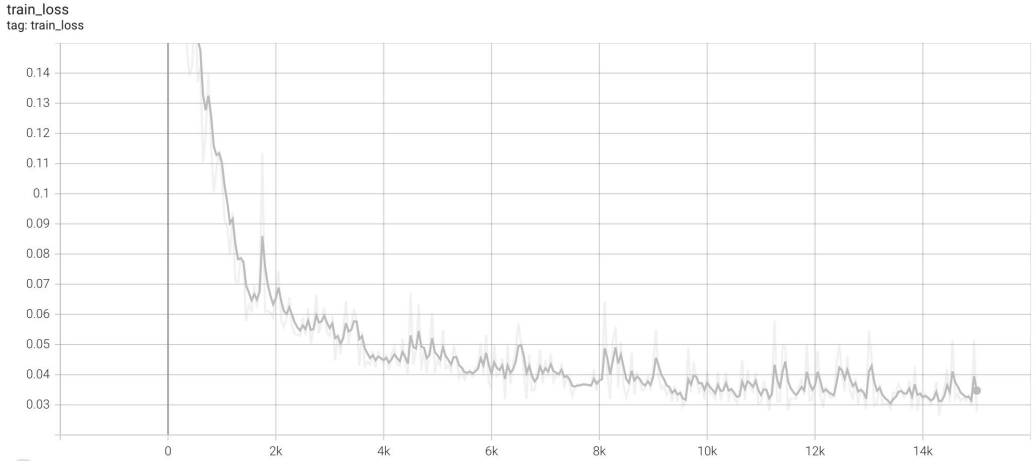


On va utiliser un auto encodeurs avec les données MNIST du TP précédent (image représentant des chiffres)

La sortie donne l'image d'entrée dans une représentation moins coûteuse

L'image d'entrée passe dans un réseau de neurones encodeur jusqu'à arriver à un espace de dimension faible appelé espace latent. De cet espace on réalise l'opération inverse en faisant passer les données de cet espace dans un réseau décodeur jusqu'à obtenir une sortie de même dimension que l'entrée.

Entraînement de l’auto encodeur



test_loss	0.020402422174811363

On utilise la fonction de perte L1, moins précise mais qui permet de garder toutes les fréquences de l’image. Le résultat sera en moyenne moins bon qu’avec une fonction de perte L2 mais plus convaincant visuellement

Avec 10 cycles, l’erreur d’entraînement est de 0.03 tandis que l’erreur de test de 0.02

Résultats de l’auto encodeur

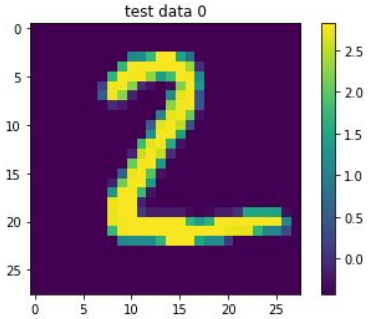


Image du dataset

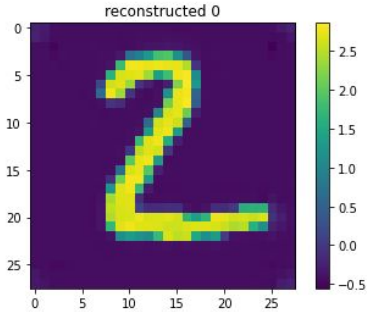
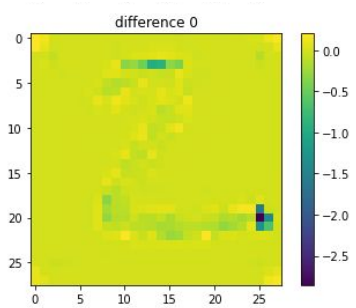


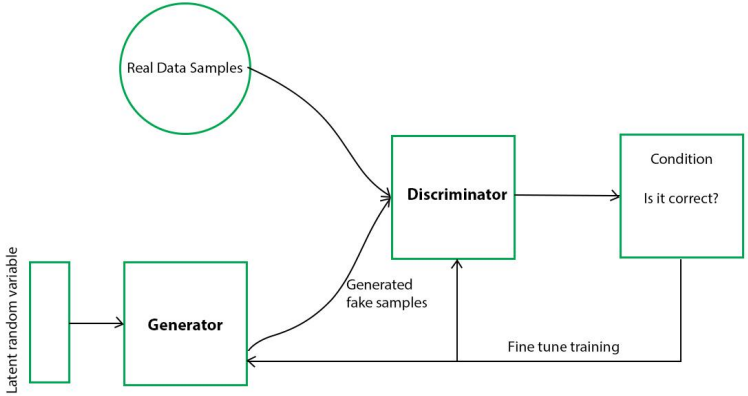
Image reconstruite



Différence entre image d’entrée et de sortie

L’auto encodeur n’étant pas parfait (erreur de test de 0.02), l’image de sortie n’est pas exactement de même qualité que l’image de départ.

Les GANs



Le principe est de mettre en opposition des fausses données créées à partir d'un générateur et des vraies données. Les fausses données et les vraies données sont les entrées du discriminateur qui doit reconnaître les vraies des fausses. Ainsi on entraîne deux réseau :

- Le générateur
- Le discriminateur

Le problème

Les données sont issues d'un dataset de Google Map représentant deux images :

- Une image satellite
- Un calque représentant le plan de l'image satellite

Le but est de créer des calques correspondant à des images satellite. Ainsi à l'entrée du discriminateur, on aura la même image satellite avec un vrai calque et un faux calque correspondant.

Ainsi on va conditionner le GAN en ajoutant cette image satellite en condition du générateur et du discriminateur.

Remarque : On aurait pu réaliser le problème avec un auto encodeur où l'on utilise le décodeur du calque et l'encodeur de l'image satellite.

Modèle

Générateur :

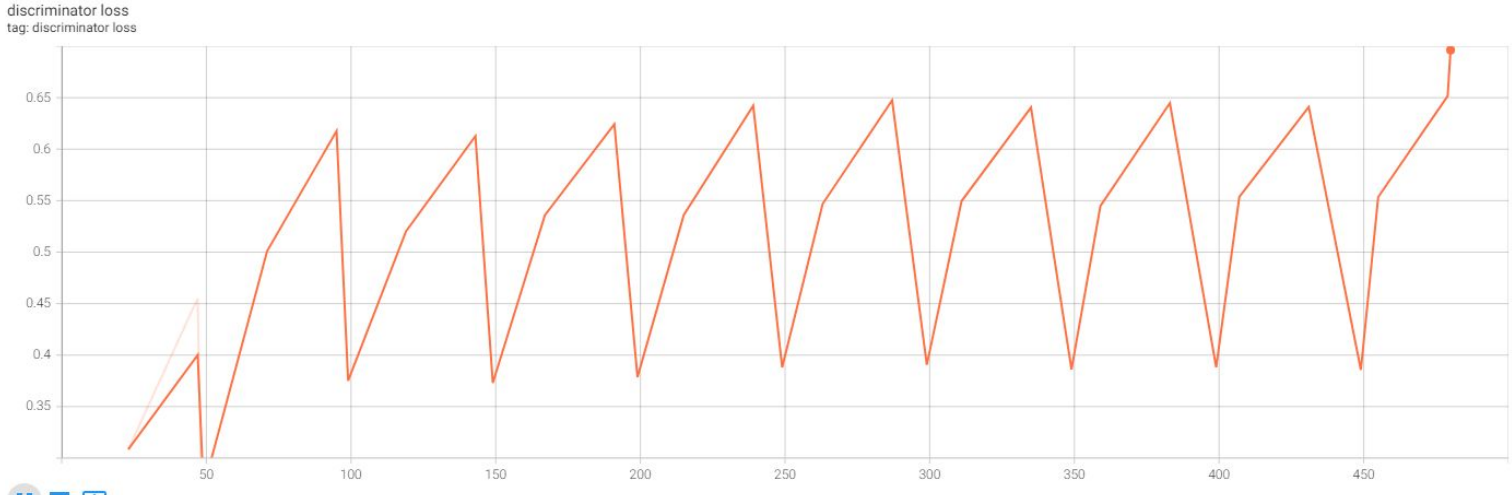
- Auto encodeur

```
(encoder): Encoder(
  (encoder): ModuleList(
    (0): ConvDown(
      (model): Sequential(
        (0): Conv2d(1, 8, kernel_size=(3, 3), stride=(1, 1))
        (1): BatchNorm2d(8, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (2): Dropout2d(p=0.5, inplace=False)
        (3): LeakyReLU(negative_slope=0.2)
      )
    )
    (1): ConvDown(
      (model): Sequential(
        (0): Conv2d(8, 16, kernel_size=(3, 3), stride=(1, 1))
        (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (2): Dropout2d(p=0.5, inplace=False)
        (3): LeakyReLU(negative_slope=0.2)
      )
    )
    (2): ConvDown(
      (model): Sequential(
        (0): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1))
        (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (2): Dropout2d(p=0.5, inplace=False)
        (3): LeakyReLU(negative_slope=0.2)
      )
    )
  )
)
(decoder): Decoder(
  (decoder): ModuleList(
    (0): ConvUp(
      (model): Sequential(
        (0): ConvTranspose2d(32, 16, kernel_size=(3, 3), stride=(1, 1))
        (1): LeakyReLU(negative_slope=0.2)
      )
    )
    (1): ConvUp(
      (model): Sequential(
        (0): ConvTranspose2d(16, 8, kernel_size=(3, 3), stride=(1, 1))
        (1): LeakyReLU(negative_slope=0.2)
      )
    )
    (2): ConvUp(
      (model): Sequential(
        (0): ConvTranspose2d(8, 1, kernel_size=(3, 3), stride=(1, 1))
        (1): LeakyReLU(negative_slope=0.2)
      )
```

Discriminateur :

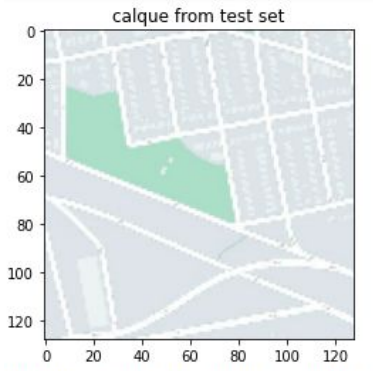
```python
class DiscriConv(nn.Module):
  def __init__(self, in_channels, out_channels, kernel_size):
    super().__init__()
    self.model = nn.Sequential(nn.Conv2d(in_channels = in_channels,
                                         out_channels = out_channels,
                                         kernel_size = kernel_size,
                                         stride = 2,
                                         padding = 1),
                                         nn.BatchNorm2d(out_channels),
                                         nn.LeakyReLU(0.2, inplace=True))

  def forward(self,x):
      return self.model(x)

class Discriminator(nn.Module):
  def __init__(self, in_channels, out_channels):
    super(Discriminator, self).__init__()
    self.main = nn.Sequential(DiscriConv(in_channels, 32,3),
                              DiscriConv(32, 64,4),
                              DiscriConv(64, 128,4),
                              # 128x4x4
                              nn.Conv2d(in_channels = 128,
                                        out_channels = out_channels,
                                        kernel_size = 3,
                                        bias=False))

  def forward(self, input):
    return self.main(input)
```

Entraînement du cGAN

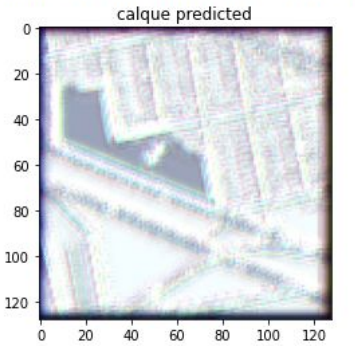


On remarque que l'erreur d'entraînement du discriminateur monte et descent, on peut supposer facilement qu'elle monte lorsque le générateur c'est entrainé et descend lorsque sait reconnaître les images générer

Résultats du cGAN



Calque du dataset



Calque issu du générateur

On voit la non précision du générateur grâce à l'image reconstituée. Cependant, on peut remarquer la même forme que l'image de départ mais avec des couleurs saturées et une imprécision élevée.