Master 2 Imagine

Projet Image

Compte rendu Nº6

Mhamad Awwad

Albena Stefanova

Restauration d’images bruitées par CNN

Table des matières :

I. Avancement global

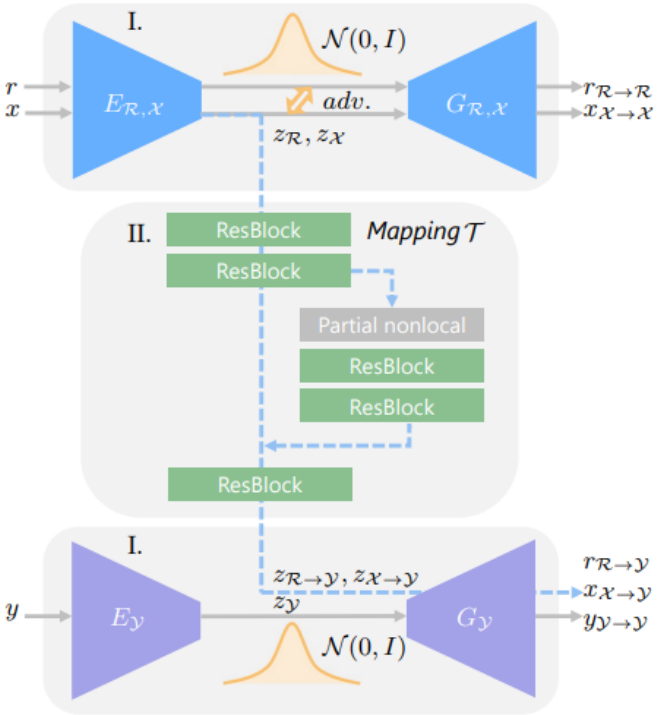
Pendant cette semaine nous avons partagé le travail en deux:

- Générer une base de données avec des dégradations structurées et non-structurées de 9 000 images.
- Creer un VAE simple - https://colab.research.google.com/drive/1--ZGVCnm89MaNV1h-pkDeJ_Pn7QYJj3L?usp=sharing
- Creer un deuxieme VAE plus avancé - https://github.com/Albena-S/Projet-Image/blob/master/Jupyter/Variational%20Autoencoder%201%20-%20DatabaseX.ipynb
- Creer un modele de GAN simple.

Nous avons réussi à finir l'implémentation de la méthode de débruitage avec VAE, comme prévu.

II. Tâches menées sur la période écoulée

II.1. Réalisation d'un Système VAE1 - GAN - VAE2

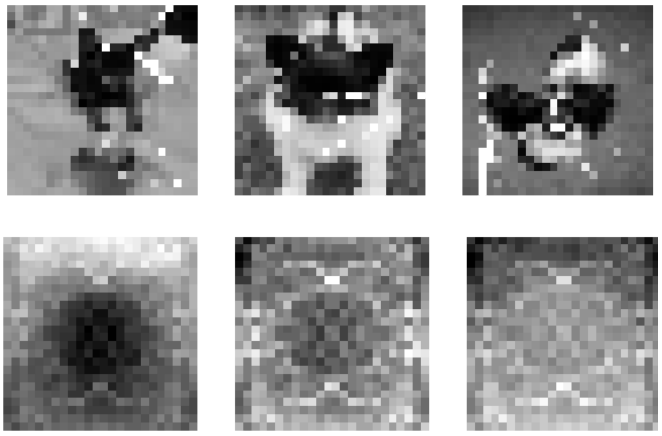


D'après l'article déjà mentionné, le système de restauration des images anciennes consiste à suivre plusieurs chemins:

on fait entraîner d'abord deux VAE : VAE1 pour les images en photos réelles r ∈ R et les images synthétiques x ∈ X , avec leur gap de domaine fermé en entraînant conjointement un discriminateur contradictoire (adversarial discriminator) ; VAE2 est réalisé pour des images propres y ∈ Y.
Avec les VAE, les images sont transformées en espace latent. Ensuite, nous apprenons le mappage qui restaure les images corrompues en images propres dans l'espace latent.
La réalisation des VAE1 et VAE2 est la même, mais les données sont différentes pour chacun d'eux.

- Implémentation d'un VAE

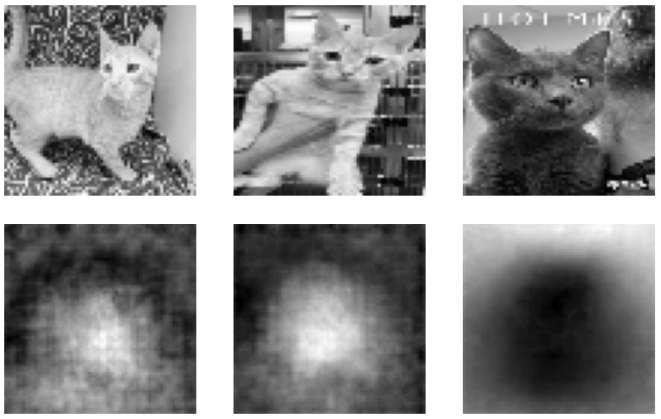
Remarque :
On a réalisé une première version de VAE en se basant sur le code de M. François Chollet, créateur de Keras. En adaptant le code pour notre base des images dégradées on a eu ces résultats :



On voit que les résultats semblent plus ou moins cohérents :

- *La prediction de chien noir(premiere image) est noir au centre et blanc aux extremites.*
- *Pareils pour les deux autres chiens.*

Les résultats ne sont toujours pas très bons par rapport à ce qu'on s'attendait. Une des raisons possible était que la base était toujours trop petite. On a alors récupéré une base de 50 000 images avec que des chats cette fois.
Les résultats ont été un peu plus satisfaisants.



On voit la ressemblance mais on imagine de meilleures prédictions. C’est pour cette raison qu’un a décidé d'implémenter de manière propre un VAE qu’on détaillera ci-dessous.

Nous allons maintenant expliquer toutes les étapes de construction de notre VAE.

Le VAE est composé de 2 composants principaux: l’encodeur et le décodeur.
l'encodeur est simplement un groupe de couches, ils pourraient être des couches entièrement connectées ou conventionnelles qui vont prendre l'entrée et ils vont la mapper sur une distribution de deux vecteurs distincts : l'un représentant la moyenne de la distribution et l'autre représentant l'écart type de cette distribution. Cette représentation est appelée “espace latent”.
Puis à partir de l’espace latent, il va essayer de reconstruire l'entrée en utilisant à nouveau des couches entièrement connectées ou convolutionnelles.
Alors pour réaliser un VAE on a besoin de faire un encodeur et un décodeur.
Avant faire ca il faut préparer les données:

on a demandé aux auteurs de l'article, s' il y a une possibilité de nous donner leurs base de données dont ils ont bases pour faire leur système, mais ils n'ont pas répondu. Pour cela, et après avoir pris les conseils et les indications de monsieur Puech et monsieur Dibot, on a implémenté une base de code qui génère des images dégradées en utilisant plusieurs modèles de masque et on a généré à peu près 9000 images dégradées.

D’abord on a récupéré le dossier base données qui contient nos images d'entraînement.

```
image_dir = "dataset_X/"
images = [os.path.join(image_dir, image) for image in
os.listdir(image_dir)]
print(len(images))
```

On a défini une fonction preprocess qui prend une image et elle la lit, decode, redimensionne et remodèle.

```
image_size = 64
@tf.function
def preprocess(image):
    image = tf.io.read_file(image)
    image = tf.io.decode_jpeg(image)
    image = tf.cast(image, tf.float32)
    image = tf.image.resize(image, (image_size, image_size))
    image = image / 255.0
    image = tf.reshape(image, shape = (image_size, image_size, 1,))
    return image
```

Puis on a sauvegardé les images dans un tensorflow BatchDataset "training_dataset" d’un lot de taille 128.

```
batch_size = 128
training_dataset = tf.data.Dataset.from_tensor_slices((images))
training_dataset = training_dataset.map(preprocess)
training_dataset = training_dataset.shuffle(1000).batch(batch_size)
print(type(training_dataset))
```

Et ensuite, pour échantillonner un vecteur z (espace latent), il faut calculer la moyenne et l'écart type du perte. (z_mean, z_var).

```
class SamplingLayer(keras.layers.Layer):
    '''A custom layer that receive (z_mean, z_var) and sample a z vector'''

    def call(self, inputs):

        z_mean, z_log_var = inputs

        batch_size = tf.shape(z_mean)[0]
        latent_dim = tf.shape(z_mean)[1]

        epsilon = keras.backend.random_normal(shape=(batch_size,
latent_dim))
        z = z_mean + tf.exp(0.5 * z_log_var) * epsilon
        return z
```

Maintenant, on a utilisé la classe SamplingLayer d'échantillonnage, pour implémenter un encodeur, qui compresse les images d'entrée par des couches conventionnelles.

```
encoder_input    = tf.keras.Input(shape=(64,64,1))
x         = layers.Conv2D(32, 3, strides=2, padding="same",
activation="relu")(encoder_input)
x         = layers.Conv2D(128, 3, strides=2, padding="same",
activation="relu")(x)
x         = layers.Conv2D(256, 3, strides=2, padding="same",
activation="relu")(x)
x         = layers.Conv2D(512, 3, strides=2, padding="same",
activation="relu")(x)
x         = layers.Flatten()(x)
x         = layers.Dense(1024, activation="relu")(x)

z_mean    = layers.Dense(latent_dim, name="z_mean")(x)
z_log_var = layers.Dense(latent_dim, name="z_log_var")(x)

z         = SamplingLayer()([z_mean, z_log_var])

encoder = tf.keras.Model(encoder_input, [z_mean, z_log_var, z],
```

```
name="encoder")
encoder.summary()
```

Le décodeur va prendre le résultat produit par l'encodeur, et essayer de faire une reconstruction en se basant encore sur des couches conventionnelles.

```
decoder = Sequential()
decoder.add(Dense(1024, activation = selu, input_shape = (latent_dim,
)))
decoder.add(BatchNormalization())

decoder.add(Dense(8192, activation = selu))
decoder.add(Reshape((4,4,512)))

decoder.add(Conv2DTranspose(256, (5,5), activation = LeakyReLU(0.02),
strides = 2, padding = 'same'))
decoder.add(BatchNormalization())

decoder.add(Conv2DTranspose(128, (5,5), activation = LeakyReLU(0.02),
strides = 2, padding = 'same'))
decoder.add(BatchNormalization())

decoder.add(Conv2DTranspose(64, (5,5), activation = LeakyReLU(0.02),
strides = 2, padding = 'same'))
decoder.add(BatchNormalization())

decoder.add(Conv2DTranspose(32, (5,5), activation = LeakyReLU(0.02),
strides = 2, padding = 'same'))
decoder.add(BatchNormalization())

decoder.add(Conv2DTranspose(1, (5,5), activation = "sigmoid", strides =
1, padding = 'same'))
decoder.add(BatchNormalization())

decoder.summary()
```

Pour de meilleurs résultats de reconstruction du vecteur latent, il faut que le modèle final de la combinaison de l'encodeur et du décodeur, va avoir une méthode de calcul du perte. cette méthode et le résultat de l'addition de 2 méthodes:
Perte de reconstruction: cette perte compare la sortie du modèle avec l'entrée du modèle. Cela peut être les pertes que nous avons utilisées dans les auto-encodeurs.
Perte latente : cette perte compare le vecteur latent à une distribution gaussienne de moyenne nulle et de variance unitaire.

```
def reconstruction_loss(y, y_pred):
```

```
    return tf.reduce_mean(tf.square(y - y_pred))

def kl_loss(mu, log_var):
    loss = -0.5 * tf.reduce_mean(1 + log_var - tf.square(mu) -
tf.exp(log_var))
    return loss

def vae_loss(y_true, y_pred, mu, log_var):
    return reconstruction_loss(y_true, y_pred) + (1 / (64*64)) *
kl_loss(mu, log_var)
```

Combinaison de l'encodeur et de décodeur dans un modèle:

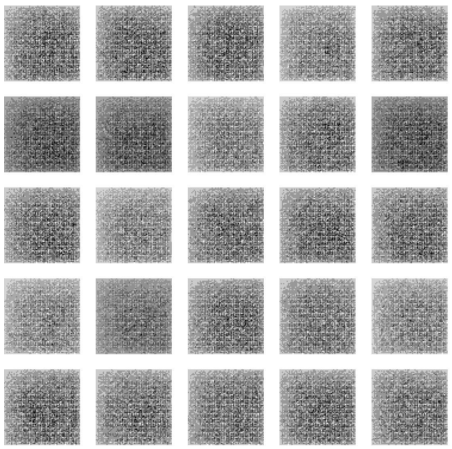
```
mu, log_var, z = encoder(encoder_input)
reconstructed = decoder(z)
model = Model(encoder_input, reconstructed, name ="vae")
loss = kl_loss(mu, log_var)
model.add_loss(loss)
model.summary()
print(z)
```

et finalement l'entraînement du VAE:

```
from tensorflow.keras.optimizers import Adam

random_vector = tf.random.normal(shape = (25, latent_dim,))
save_images(decoder, 0, 0, random_vector)

mse_losses = []
kl_losses = []
optimizer = Adam(0.0001, 0.5)

epochs = 100
# execution_epoch(epochs,optimizer,model,training_dataset)
for epoch in range(1, epochs + 1):
    print("Epoch: ", epoch)
    for step,training_batch in enumerate(training_dataset.take(11)):
        with tf.GradientTape() as tape:
            reconstructed = model(training_batch)
            y_true = tf.reshape(training_batch, shape = [-1])
            y_pred = tf.reshape(reconstructed, shape = [-1])
            mse_loss = reconstruction_loss(y_true, y_pred)
            mse_losses.append(mse_loss.numpy())
            kl = sum(model.losses)
            kl_losses.append(kl.numpy())
            train_loss = 0.01 * kl + mse_loss
```

```
            grads = tape.gradient(train_loss, model.trainable_variables)
            optimizer.apply_gradients(zip(grads,
model.trainable_variables))

            if step % 10 == 0:
                save_images(decoder, epoch, step, random_vector)
            print("Epoch: %s - Step: %s - MSE loss: %s - KL loss: %s" %
(epoch, step, mse_loss.numpy(), kl.numpy()))
```

Après un entraînement, des image dégradées, de 100 époques et 11 pas de chaque époque, on a eu les résultats suivants (affichage d'une grille qui comprend 5✕5 images restaurées):

	
Epoque 0	Epoque 100

- Implementation de GAN

L'encodeur de VAE1 va avoir 2 données d'entrée, qui sont les image anciennes réelles et les image artificiellement dégradées, et ca produira 2 espaces latents qu'on va aligner et dont l'écart de domaine est examiné plus par un réseaux antagonistes génératifs (Generative Adversarial Networks ou GAN).
On a implémenté, pour tester comment ça marche, un pseudo code d'un GAN qui prend deux espaces latents, et les aligne dans une même espace latent.

```
input_data_x = ...
input_data_z = ...
input_data = tf.concat([input_data_x, input_data_z], axis=1)
```

```
latent_data_x = vae.encoder.predict(input_data_x)
latent_data_z = vae.encoder.predict(input_data_z)
```

Les 2 composants principaux pour un GAN sont le générateur et le discriminateur.

1. générateur: entraîné pour générer de nouveaux résultats
2. discriminateur: classe les entrées comme réalistes ou fausses. Il tente d'identifier si une entrée provient de l'ensemble de données d'origine du modèle de générateur.

```
discriminator = tf.keras.Sequential([
    layers.Input(shape=(latent_dim,)),
    layers.Dense(128, activation='relu'),
    layers.Dense(1, activation='sigmoid')
])

generator = tf.keras.Sequential([
    layers.Input(shape=(latent_dim,)),
    layers.Dense(128, activation='relu'),
    layers.Dense(latent_dim)
])
```

On fait entraîner le GAN sur les espaces latents des deux domaines.
Le GAN est entraîné sur des vecteurs de uns et de zéros comme labels. De cette façon, le discriminateur apprendra à distinguer les données réelles des données générées.
Les vecteurs de uns et de zéros sont utilisés comme labels pour les données d'entraînement des deux domaines, les uns représentant les données générées et les zéros les données réelles.

```
gan = tf.keras.Model(generator.input, discriminator(generator.output))
gan.compile(optimizer='adam', loss='binary_crossentropy')

ones = tf.ones(lat_shape=(latent_data_x.shape[0], 1))
zeros = tf.zeros(shape=(latent_data_z.shape[0], 1))

gan.fit(latent_data_x, ones, epochs=10)
gan.fit(latent_data_z, zeros, epochs=10)
```

et enfin la génération du nouveau espace latent

```
generated_data_in_latent_space = gan.generator.predict(random_noise)
```

prediction

```
generated_images_x_z = vae.decoder.predict(generated_data_x)
```

II.2. Methode

Étant donné que l'article est écrit en anglais et que nous comprenons aussi bien l'anglais que le français, peut-être même un peu mieux, c'est pour cette raison que nous avons décidé de faire une explication beaucoup plus détaillée des méthodes expliquées dans l'article anglais.
L'article propose des algorithmes basés sur des connaissances de base dans le domaine de l'IA. Pour nous, presque tout était une découverte et en essayant de comprendre au mieux toutes les notions et idées, nous avons écrit cette explication détaillée. À chaque fois que des notions nous étaient inconnues, nous avons ajouté des explications sur ces méthodes.

Nous allons présenter la partie 3. Méthodes de l'article 'Old Photo Restoration via Deep Latent Space Translation'.

- Part 3. Method

Old photo restoration needs a different approach compared to conventional image restoration. Firstly, old photos contain more complex degradation. There is a domain gap between synthetic and real photos and as such, a CNN cannot generate well realistic photos by purely learning from synthetic data. Also, defects of old photos are a compound of multiple degradations and all they require different strategies for restoration.

Overall, old photos contain two types of defects :

- Unstructured, such as film noise, blurriness and color fading, which are corrected by **making use of surrounding pixels within the local patch**.
- Structured, such as scratches and blotches , which are corrected by **inpainting by considering the global context**.

- 3.1 Restoration via latent space translation

The paper presents old photo restoration as an image translation problem, which is a process of converting an image from one domain to another domain. In our case **clean images and old photos as images from distinct domains and we wish to learn how to pass from one to the other**.
More precisely, we have three domains :

- The real photo domain R, with r representing images from this domain so $r \in R$.
- The synthetic domain X where images suffer from artificial degradation, with x representing images from this domain so $x \in X$.

- The corresponding ground truth domain *Y* that comprises images without degradation, with *y* representing images from this domain so $y \in Y$.

A good question to ask is 'Why not supervised learning?'. The answer would be because there is no direct pairing from real photos *r* to clean images *y*.
The solution is to decompose the translation in two stages :

- Each domain is mapped to its corresponding latent spaces so
$$E_{\mathcal{R}} : \mathcal{R} \mapsto \mathcal{Z}_{\mathcal{R}}, E_{\mathcal{X}} : \mathcal{X} \mapsto \mathcal{Z}_{\mathcal{X}}, E_{\mathcal{Y}} : \mathcal{Y} \mapsto \mathcal{Z}_{\mathcal{Y}}$$
where *E* is the encoder and *Z* the latent space.
Also because synthetic images and real old photos are both corrupted, sharing similar appearances, we need to align their latent space into the shared domain by enforcing constraints. Therefore we have
$$\mathcal{Z}_{\mathcal{R}} \approx \mathcal{Z}_{\mathcal{X}}$$
- Then image restoration is learnt in the latent space. By using the pairs {x,y} we learn the translation from the latent space of corrupted images, $\mathcal{Z}_{\mathcal{X}}$, to the latent space of ground truth, $\mathcal{Z}_{\mathcal{Y}}$, through the mapping
$$T_{\mathcal{Z}} : \mathcal{Z}_{\mathcal{X}} \mapsto \mathcal{Z}_{\mathcal{Y}}$$
can be reversed to *Y* through generator
$$G_{\mathcal{Y}} : \mathcal{Z}_{\mathcal{Y}} \mapsto \mathcal{Y}.$$

All of this was to reach the conclusion that **real old photos *r* can be restored by**

$$r_{\mathcal{R}\to\mathcal{Y}} = G_{\mathcal{Y}} \circ T_{\mathcal{Z}} \circ E_{\mathcal{R}}(r)$$

- Domain alignment in the VAE latent space (Stage 1)

We assume that *R* and *X* are encoded into the same latent space.
We find that one way to examine and reduce domain gap in a VAE is to use an **adversarial discriminator.** This is a type of neural network that is often used in generative adversarial networks (GANs) to distinguish between samples from different domains.
The general idea is to train the adversarial discriminator to distinguish between samples from the different domains, and then use this information to improve the performance of the VAE. This can help the VAE better capture the characteristics of the data from each domain and **reduce the gap** between them.
In our case, we are training a VAE on a dataset of images from two different domains, R and X. We should train the adversarial discriminator to distinguish between images from the two domains. This can help the VAE learn to generate images that are more realistic and less easily distinguishable from real images.

In the First stage of the algorithm we need to create two VAEs :

- VAE1

Old photos {r} and synthetic images {x} share VAE1, with the encoder $E_{\mathcal{R},\mathcal{X}}$ and generator $G_{\mathcal{R},\mathcal{X}}$.

- VAE2
  The ground true images {y} are fed into the second one, VAE2 with the encoder-generator pair $\{E_{\mathcal{Y}}, G_{\mathcal{Y}}\}$.

Images can be reconstructed by sampling from the latent space.

Now let's see more details about VAE1.
Why is VAE used rather than vanilla autoencoder ? Because VAE features denser latent representation due to the KL regularization, which helps to produce closer latent space for {r} and {x} with VAE1 thus leading to smaller domain gap.
To optimize VAE1 with data {r} and {x} the article proposes to use Differentiable stochastic sampling.
What is Stochastic sampling ? Stochastic sampling is used in VAEs to generate multiple samples from the latent space. This is done by sampling from a distribution over the latent space, rather than using the mean of the distribution as the latent representation. This can help the VAE **generate more diverse and interesting samples**, as it allows the model to explore different regions of the latent space. In our case of image sampling, stochastic sampling can help the model generate multiple variations of a given image, rather than just one fixed output. This can **make the generated images more realistic and less repetitive**.

The objective with {r} is defined as:

$$\begin{aligned}\mathcal{L}_{\text{VAE}_1}(r) = \ & \text{KL}(E_{\mathcal{R},\mathcal{X}}(z_r|r)||\mathcal{N}(0,I)) \\ & + \alpha\mathbb{E}_{z_r\sim E_{\mathcal{R},\mathcal{X}}(z_r|r)}\left[\|G_{\mathcal{R},\mathcal{X}}(r_{\mathcal{R}\to\mathcal{R}}|z_r) - r\|_1\right] \\ & + \mathcal{L}_{\text{VAE}_1,\text{GAN}}(r)\end{aligned}$$

Objective for {x} is similar. Here we don't understand very well the notations and what they mean. The first function, *KL* is a standard Kullback-Leibler Divergence, in code `tf.keras.losses.KLDivergence()`.
The last loss function is least-square loss. The authors reference the LSGAN , from the article "Least squares generative adversarial networks", to address the well-known over-smooth issue in VAEs, further encouraging VAE to reconstruct images with high realism. However, we do not understand well how to use LSGAN, which is required in almost all loss in the paper.
We do not understand very well the second parameter either.

As proposed above, to further narrow the domain gap in this reduced space, the authors use an adversarial network to examine the residual latent gap. Concretely, another discriminator $D_{\mathcal{R},\mathcal{X}}$ is trained, which differentiates $\mathcal{Z}_{\mathcal{R}}$ and $\mathcal{Z}_{\mathcal{X}}$, and whose loss is defined as :

$$\begin{aligned}\mathcal{L}^{\text{latent}}_{\text{VAE}_1,\text{GAN}}(r,x) = \ & \mathbb{E}_{x\sim\mathcal{X}}[D_{\mathcal{R},\mathcal{X}}(E_{\mathcal{R},\mathcal{X}}(x))^2] \\ & + \mathbb{E}_{r\sim\mathcal{R}}[(1 - D_{\mathcal{R},\mathcal{X}}(E_{\mathcal{R},\mathcal{X}}(r)))^2]\end{aligned}$$

Meanwhile, the encoder $E_{\mathcal{R},\mathcal{X}}$ of VAE1 tries to fool the discriminator with a contradictory loss.
What is a contradictory loss? This function is used to calculate the difference between the output of the discriminator for real and fake samples, and this difference is used to update the weights of the generator and discriminator in order to improve the performance of the model. In this way, the generator and discriminator are able to learn from each other and improve their performance over time. **The contradictory loss ensures that *R* and *X* are mapped to the same space.**
The the total objective function for VAE1 becomes more complicated, taking into consideration all of the above and looks like this :

$$\min_{E_{\mathcal{R},\mathcal{X}},G_{\mathcal{R},\mathcal{X}}} \max_{D_{\mathcal{R},\mathcal{X}}} \mathcal{L}_{\text{VAE}_1}(r)+\mathcal{L}_{\text{VAE}_1}(x)+\mathcal{L}^{\text{latent}}_{\text{VAE}_1,\text{GAN}}(r,x)$$

- **Restoration through latent mapping (Stage 2)**

With the latent code,a specific point in the latent space that represents a given input, captured by VAEs, in the second stage, we leverage the synthetic image pairs {x, y} and propose to **learn the image restoration by mapping their latent space** (the mapping network *M* in Figure 3 of the document and Fig.1 in this report).
At this stage, we solely need to train the parameters of the latent mapping network *T* and fix the two VAEs. The loss function of that network is :

$$\mathcal{L}_{\mathcal{T}}(x,y) = \lambda_1 \mathcal{L}_{\mathcal{T},\ell_1} + \mathcal{L}_{\mathcal{T},\text{GAN}} + \lambda_2 \mathcal{L}_{\text{FM}}$$

One of the components is a feature loss function.
What is a feature loss ? That is a type of loss function that is used to encourage the encoder and decoder to produce feature representations that are similar to those of the real data. This helps to improve the quality of the generated data and reduce the reconstruction error.

- 3.2 Multiple degradation restoration

For now the latent restoration is only concentrated on local features due to the limited receptive field of each layer. The restoration of structured defects requires inpainting. The authors propose to enhance the latent restoration network by incorporating a **global branch which consists of a nonlocal block that considers global context**. As the original block proposed in the article "Non-local neural networks" is unaware of the corruption area, the nonlocal block proposed by the authors of the current paper explicitly utilizes the mask input so that the pixels in the corrupted region will not be adopted for completing those areas.
The algorithm uses *F*, an intermediate feature map, and *m* a binary mask downscaled to the same size, where 1 represents the defect regions to be inpainted and 0 represents the intact regions.

The relations between two locations in the feature map *F* can be calculated by comparing the feature information at those locations and determining how similar they are. This relation, affinity, is denoted by $s_{i,j}$ and :

$$s_{i,j} = (1 - m_j) f_{i,j} / \sum_{\forall k} (1 - m_k) f_{i,k}$$

The partial nonlocal finally outputs a weighted average of correlated features for each position.
The global branch is specifically for inpainting and **is fused with the local branch under the guidance of the mask.** In this way, **the two branches constitute the latent restoration network,** which is capable of dealing with multiple degradation in old photos.

- 3.3 Defect Region Detection

Since the global branch of our to-be restoration network requires a mask *m*, we train a **scratch detection network in a supervised way** by using a mixture of real scratched dataset and synthetic dataset to generate the mask automatically. The model uses a cross-entropy loss to minimize the difference between the predicted mask. The loss function is the following :

$$\begin{aligned}\mathcal{L}_{CE} = \mathbb{E}_{(s_i, y_i) \sim (\mathcal{S}, \mathcal{Y})} \Bigg\{ & \alpha \sum_{h=1}^{H} \sum_{w=1}^{W} -y_i^{(h,w)} \log \hat{y}_i^{(h,w)} \\ & - (1 - \alpha) \sum_{h=1}^{H} \sum_{w=1}^{W} (1 - y_i^{(h,w)}) \log(1 - \hat{y}_i^{(h,w)}) \Bigg\}\end{aligned}$$

As scratch regions are a small portion, a parameter α is used to calculate the positive/negative proportion of y_i as :

$$\alpha_i = \frac{[y_i = 1]}{[y_i = 1] + [y_i = 0]]}$$

This sums up the heart of this paper.

III. Tâches prévues

Pour la semaine prochaine on a comme objectif premièrement de finir l'étape 1 de l'article :

- VAE1 pour les images en photos réelles $r \in R$ et les images synthétiques $x \in X$, avec leur gap de domaine fermé en entraînant conjointement un discriminateur contradictoire (adversarial discriminator).
- VAE2 pour des images propres $y \in Y$.

Deuxièmement, on va faire un poster.