Report Computer Vision

Project 3

Yoann Le Duault, ER1562

I. Dataset

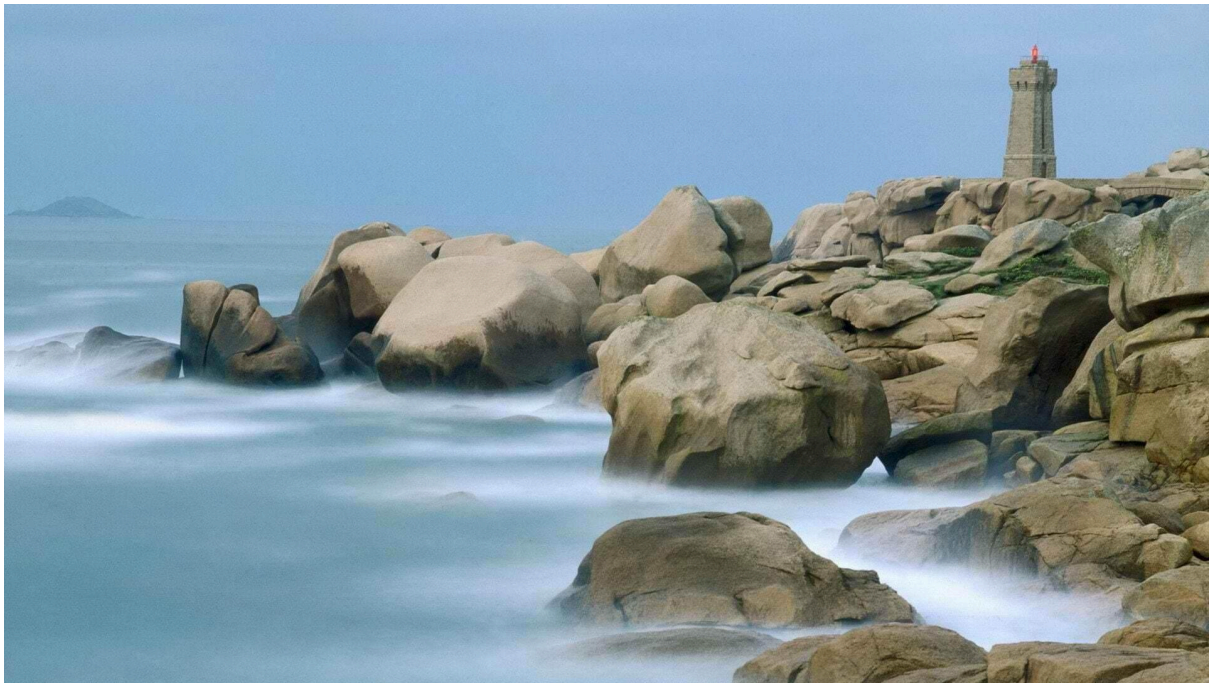
The selected dataset is the *Benchmarking Ultra-High-Definition Image Super-resolution (ICCV2021)*. It contains 8099 images in 4K (original images), and in various inferior resolutions corresponding to different degradations. Since the 4K images are prohibitively big, we will use 2x bicubic degraded images as a dataset.

These images are in full HD (1920 x 1080p). To complete the dataset, we add 2901 images in full HD. Part of these images (2378 images) are from the AM-2K dataset and the last 523 are images that were downloaded directly from the web (identifiable by their naming convention: *Brittany*).

Examples :

4K data

AM-2K



Brittany

II. Problem

We propose to solve the Super resolution problem. This problem consists of using algorithms to increase the resolution of images with coherent and probable added pixels.

Most of the methods to solve this problem seem to rely on the use of a neural network similar to U-net, consisting of multiple blocks of convolution with skip connections that are fed to an upscaling convolutional layer to generate the upscale image.

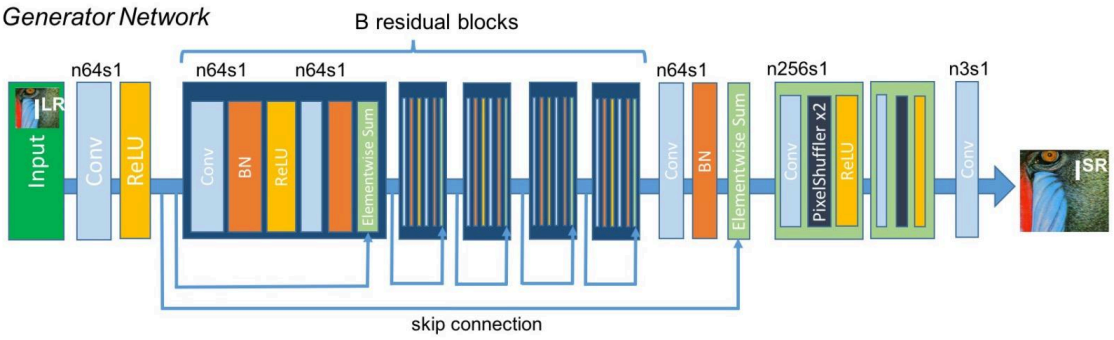
Super resolution can be used to artificially increase the quality of camera (typically with smartphone) without using mechanical/optical parts, to process visual data of poor quality like SAR images (*Synthetic-aperture radar* with satellites) or update old media to modern standard of resolution (see *Valkyria Chronicles 3 Extra Edition AI upscaled* for example)

III. Architectures

We have implemented two architectures for solving the super resolution problem.

A. ResNet

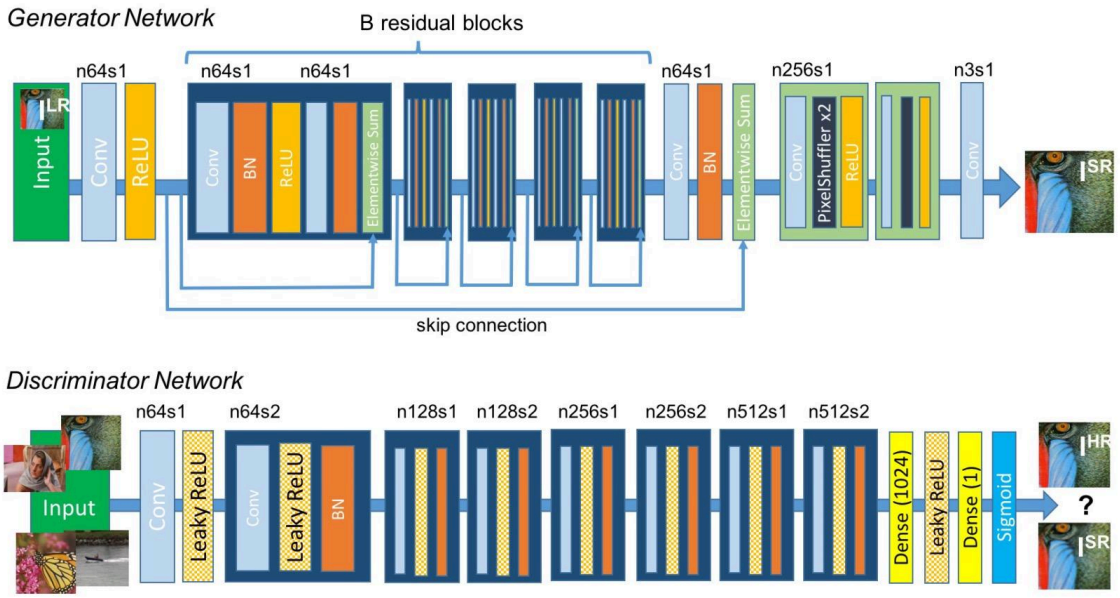
The first implementation is a ResNet model. Its architecture is the following :
(BN stands for batch normalization)



Our model uses 16 residual blocks. Each block is composed of a convolution layer, then batch normalization, then ReLU then another convolution then BN then an element wise sum between the result of the block and the image before the block (with the skip connection).

B. Gan

The GAN model is an improvement on the ResNet. We use the same architecture for the generator, provide a discriminator and use VGG19 for feature descriptions.



Since we are using a GAN, we need a way to convert our image into feature descriptions. For this, we use the 5-4 layers of VGG19.

IV. Model analysis and hyperparameters

	Memory usage	Nb trainable parameters	Nb non-trainable parameters
ResNet	5.93 MB	1549443	4224
GAN	95.74 MB	25114948	7936

V. Training

The training is done in the project notebook. We generate an infinite dataset (from the TF Dataset class) by repeating the data in it. The dataset is not directly fed to networks, we undergo data augmentation by cropping part of the image (96x96 crop), rotating, flip, adding noise by using a mapping method on the dataset (resulting in random transformation for each image batch called).

The only modification to make to retrain the model is the path to the input images and to the output folders. Since we use Colab, these paths are defined with the absolute path “/content/drive/MyDrive/”.

When training, we use checkpoints to keep the weights in case of an error/disconnection. It is very important to ensure that these are correctly stored, because these networks are time consuming to train.

Hyperparameters :

For the training of the ResNet, we make 20 epochs of 1000 batches. The main reason for this choice is the time taken to train.
The article also uses a batch of 16 images of 96x96p so we keeped it that way.

For the training of the GAN, we make 1000 epochs of 200 steps, again for time reasons.

Prediction result :

For using these models, we implemented a Gradio interface that takes a low resolution and depending on the button clicked, generates the upscaled image using the corresponding model.
To launch the GUI, just call in a terminal the file run_gui.py as follow *./run_gui.py* with python3 installed. A terminal window should give you a url link to the interface. Simply enter the URL into a web browser.

If you encounter difficulties, there is a notebook with the same name that contains the necessary to manually use the algorithms.

VI. Metrics, loss and evaluation

We use two metrics to measure the efficiency of models :

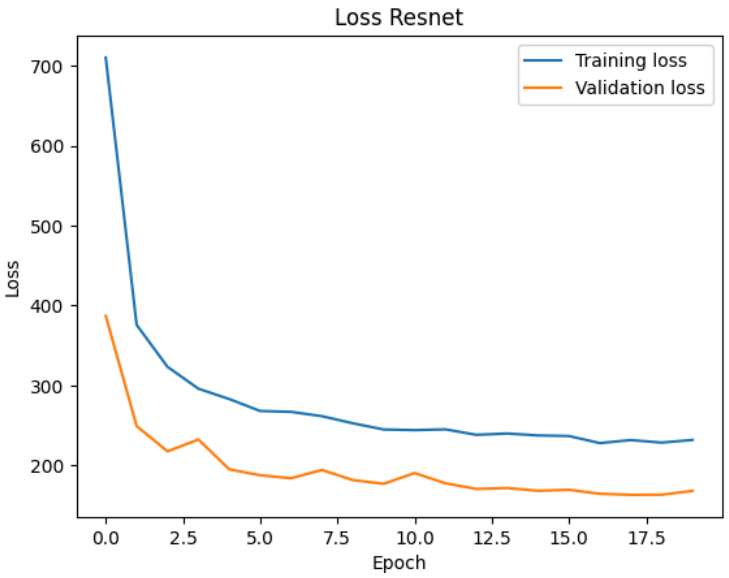
- PSNR (Peak Signal to Noise Ratio)
- SSIM (structural similarity)

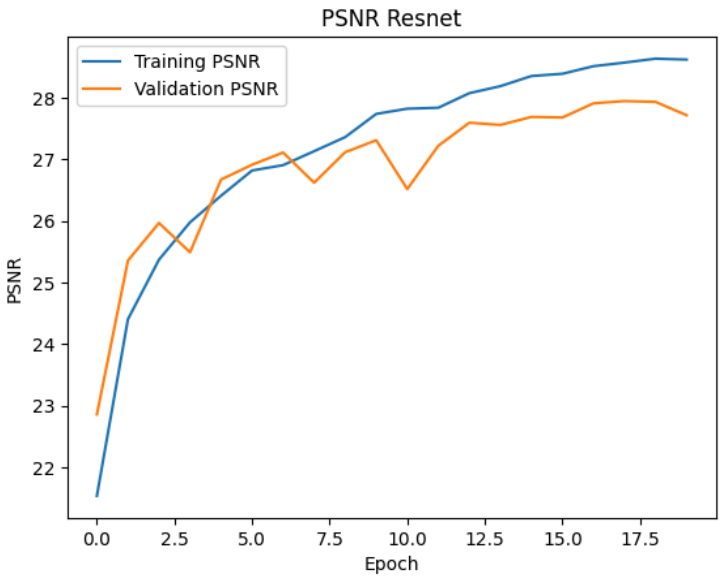
Additionally, for the optimizer, we use the Keras implementation of Adam, with a decay of learning rates with the GAN.

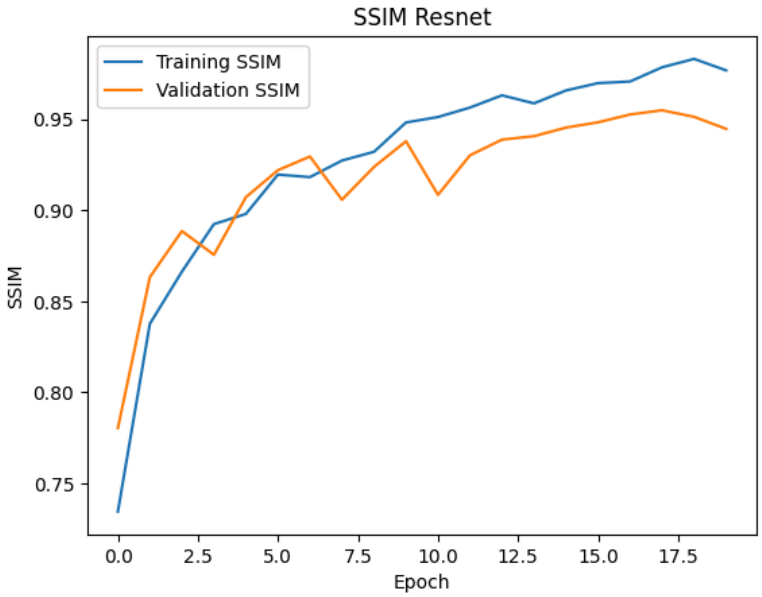
The losses function are as follows :

- For the ResNet, we use MSE
- For the GAN generator, we use MSE on the VGG19 features vectors for content loss and binary cross entropy for the loss on the generator. The two are combined into a perception loss by the formula : content_loss + 0.001 * generator_loss. The gradient is applied to this perception loss.
- For GAN discriminator, we use binary cross entropy (combine with sigmoid activation layer)

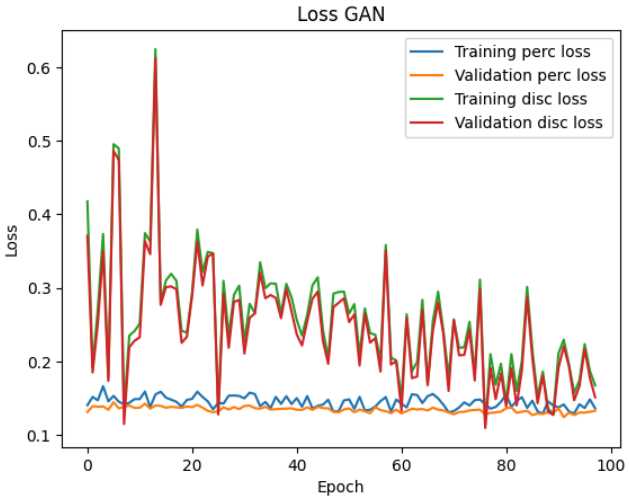
Plots of training and validation ResNet

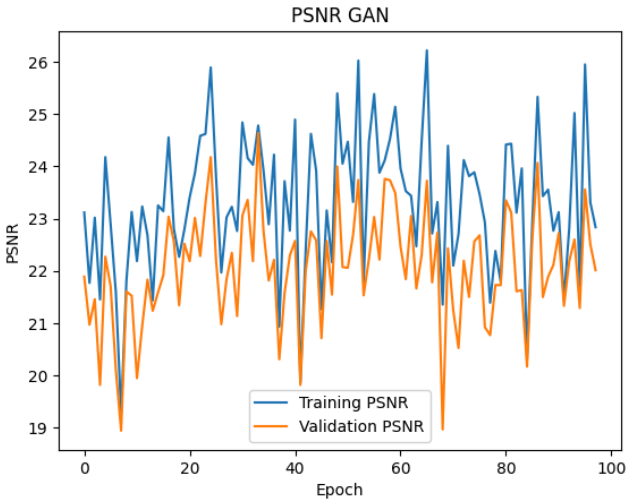


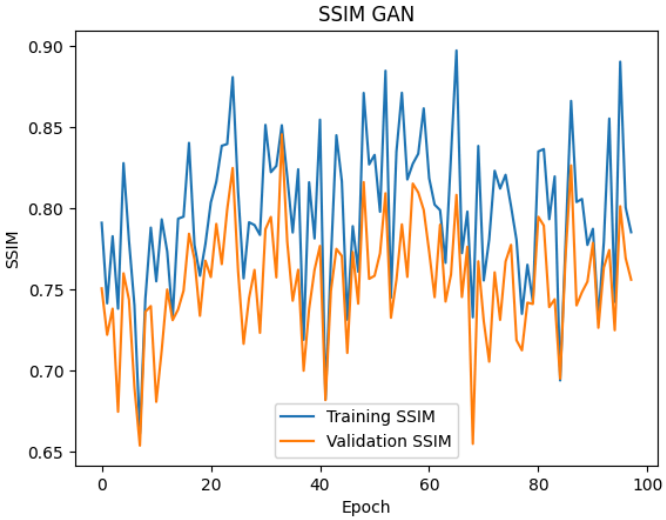




Plots of training and validation GAN







VII. Comparison of models

Original image not use in training:

ResNet

PSNR average value on test set : 24.1112 dB
SSIM average value on test set : 0.7183

Example image:

GAN

PSNR average value on test set : 18.2438 dB
SSIM average value on test set : 0.5464

Example image:

Analysis

We can see that the two models can produce images of the right resolutions.

The ResNet leads to an image slightly blurred, erasing the details of sharp color gradient. It is clearly a consequence of using MSE loss, the network will try to obtain images with pixels matching with the HR image, leading to creating pixels as average of the surrounding pixels.

The Gan leads to an image with sharper features than the ResNet, because it uses features from VGG19 and adversarial training. We can remark that the generated image are somewhat stylized (very visible close to the image, at distance, it is not so visible), probably a consequence of the relatively short training of the network compared to the common training time in scientific article (ranging from few days to several weeks, using very powerful GPUs)

VIII. Time (training and prediction)

The training has been done using a Colab T4 GPU instance.

	Training time	**Prediction time**
ResNet	5h12min	between 5-10s for each image
GAN	6h25min	between 5-10s for each image

IX. Bibliography

Articles :

1. **Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network**, Christian Ledig, Lucas Theis, Ferenc Husz´ar, Jose Caballero & al. , *https://arxiv.org/pdf/1609.04802.pdf*
2. **Benchmarking Ultra-High-Definition Image Super-resolution**,Kaihao Zhang , Dongxu Li, Wenhan Luo & al., *https://openaccess.thecvf.com/content/ICCV2021/papers/Zhang_Benchmarking_Ultra-High-Definition_Image_Super-Resolution_ICCV_2021_paper.pdf*

GitHub/Dataset:

1. **Benchmarking Ultra-High-Definition Image Super-resolution** repo : https://github.com/HDCVLab/UHD4K_UHD8K?tab=readme-ov-file

2. **AM-2K dataset** : https://drive.google.com/drive/folders/1SReB9Zma0TDfDhow7P5kiZNMwY9j9xMA

X. Table of completed items

Completed items	Points
Problem : Super resolution	3
Model : ready architecture trained from scratch	1
Model additional point : subsequent model	1
Dataset : at least 10000 photos	1
Dataset : > 500 own photos	1
Training : Data augmentation	1
Tool : GUI Gradio	1
TOTAL	**9**

XI. Github repository

The project has been uploaded in the following github repository :

https://github.com/yleduault/CV-project3

The requirements for running the program are indicated in the requirements.txt file.