VRIJE
UNIVERSITEIT
BRUSSEL

Image Processing

Single image super-resolution

Author

Rami Soussi

Contents

INTRODUCTION

Even today, image super-resolution is still one of the fundamental challenges in image processing.

In this project, I will make use of interpolation-based methods and learning-based methods in order to increase the resolution of the images.

I will give description of the two different methods.

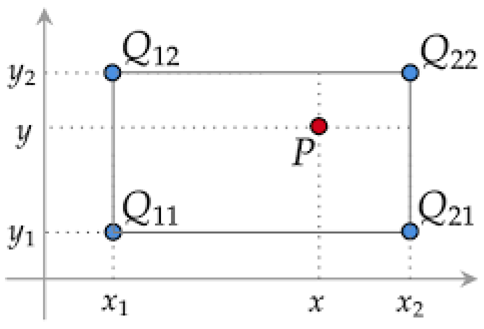
As a conclusion the difference between the results of both methods will be made.

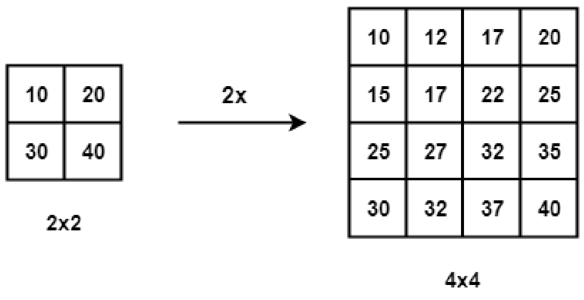
A – Interpolation-based method

1. Traditional interpolation algorithms

1.1. Bilinear interpolation :

This algorithm takes the linearly weighted average of the four nearest pixels around the destination pixel.





10	20
30	40

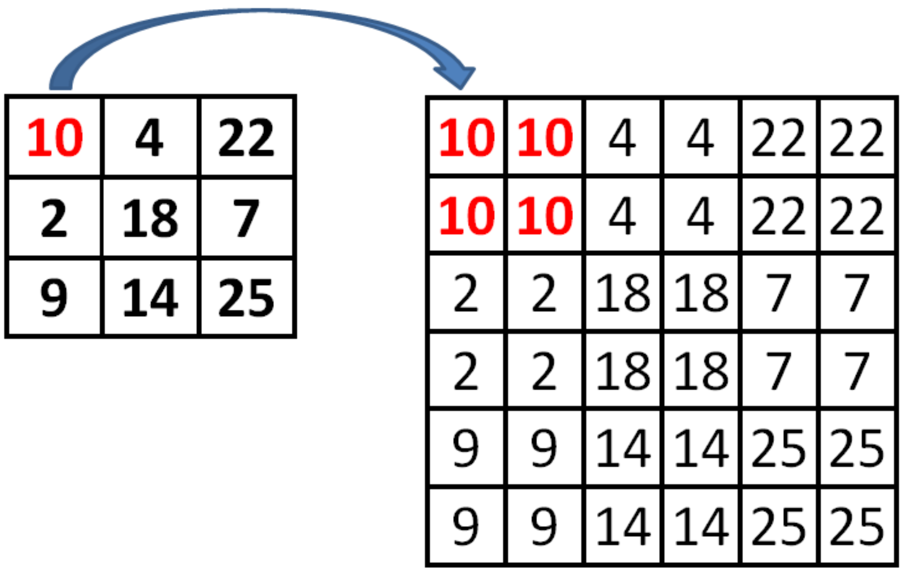
2x2

10	12	17	20
15	17	22	25
25	27	32	35
30	32	37	40

4x4

1.2. Nearset neighbor interploation :

The nearset neighbor algorithm simply copies the nearest pixel (nearest neighbour sampling)

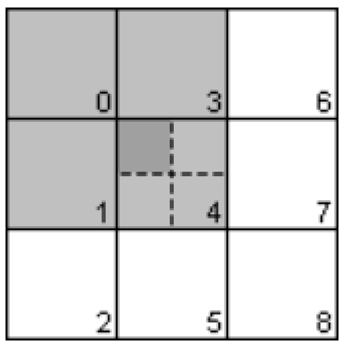


10	4	22
2	18	7
9	14	25

10	10	4	4	22	22
10	10	4	4	22	22
2	2	18	18	7	7
2	2	18	18	7	7
9	9	14	14	25	25
9	9	14	14	25	25

2. directionally averaging scaling algorithm

2.1. Algorithm:

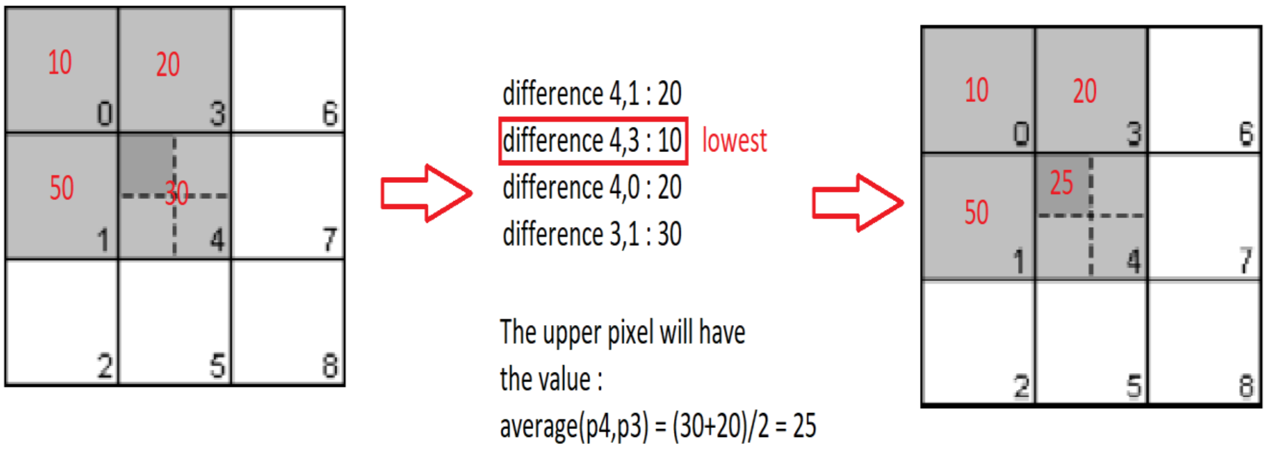


This algorithm calculates four gradients for each source pixel. The figure above represents nine pixels from the source image. To calculate the target pixel for the upper left quadrant for the middle source pixel, the algorithm considers the source pixels 0, 1, 3 and 4.

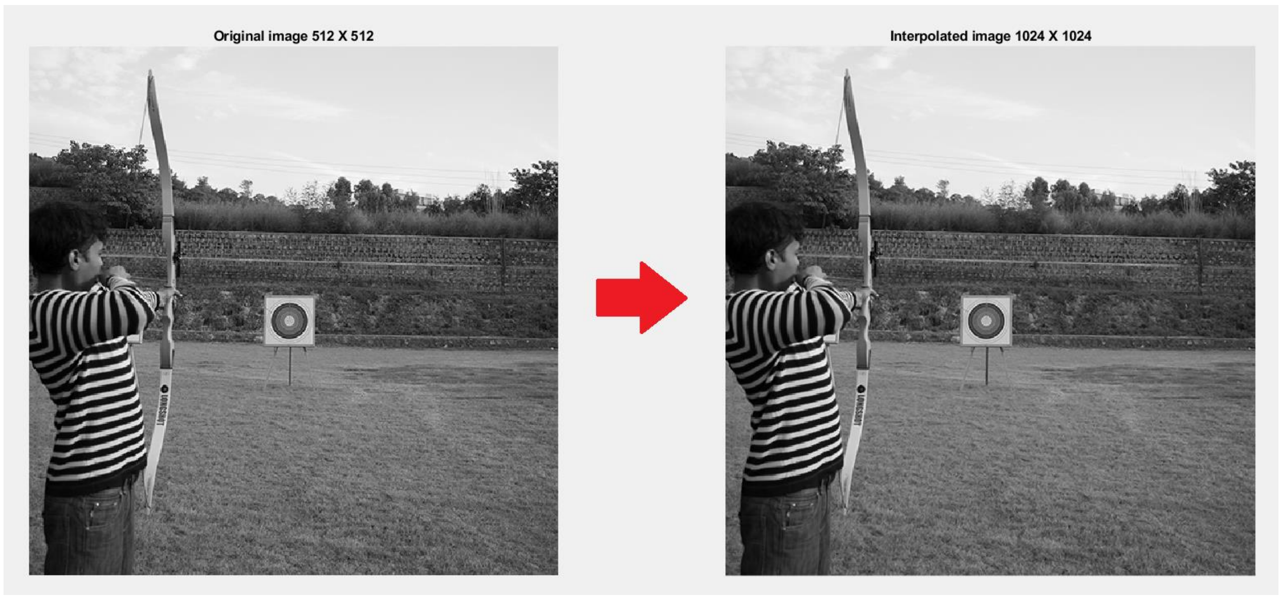
Of all source pixels under consideration, it interpolates between the pair of pixels with the lowest difference. The upper left target pixel may be:

1. interpolated between source pixels 4 and 1,
2. interpolated between source pixels 4 and 3,
3. interpolated between source pixels 4 and 0,
4. interpolated between source pixel 4 and the average of source pixels 1 and 3.

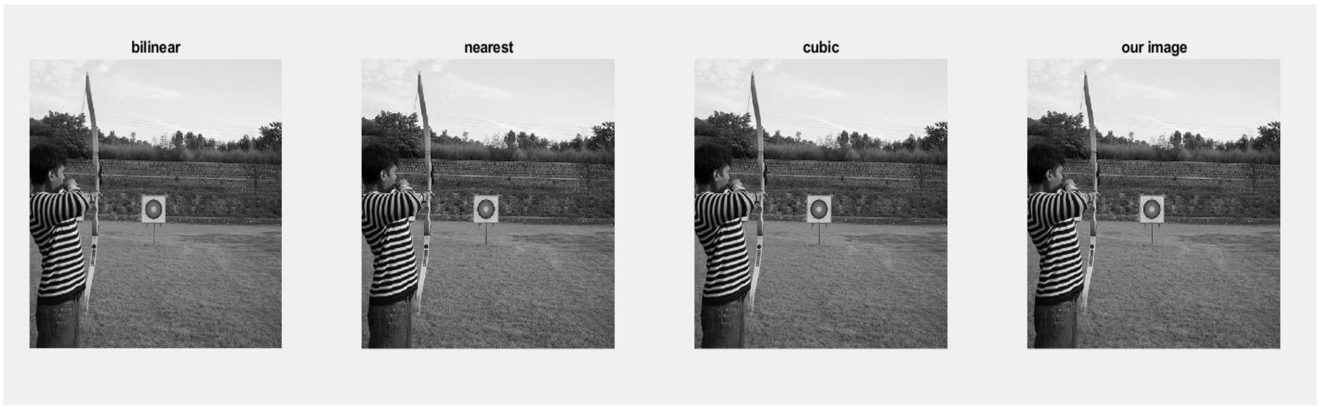
2.2. Example :



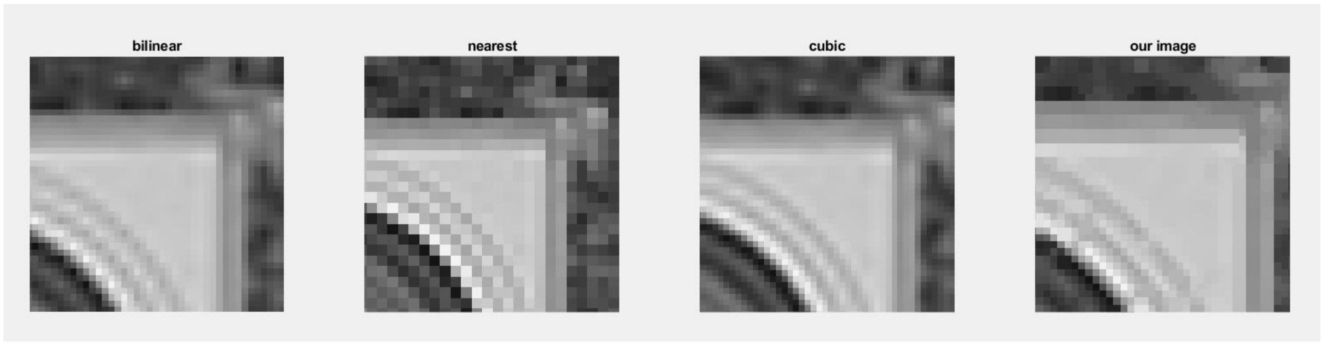
2.3. Results on gray-scale images :



Comparison between interpolations :



I used zoom to see the difference between pixels :



Comparaison between psnr :

Interpolation method	bilinear	nearest	cubic	Our method
PSNR	32.0975	28.3648	32.0605	32.7491

We remark that our new interpolation method has the best noise reduction.

2.4. Results on coloured images :

In order to use this interpolation method with RGB images, we can convert neighboured pixels to grey scale, then calculate the distance between the grey scale values. Then, we assign the averaged RGB source pixels to the result image.

PSNR = 28.0430

B - Learning-based method

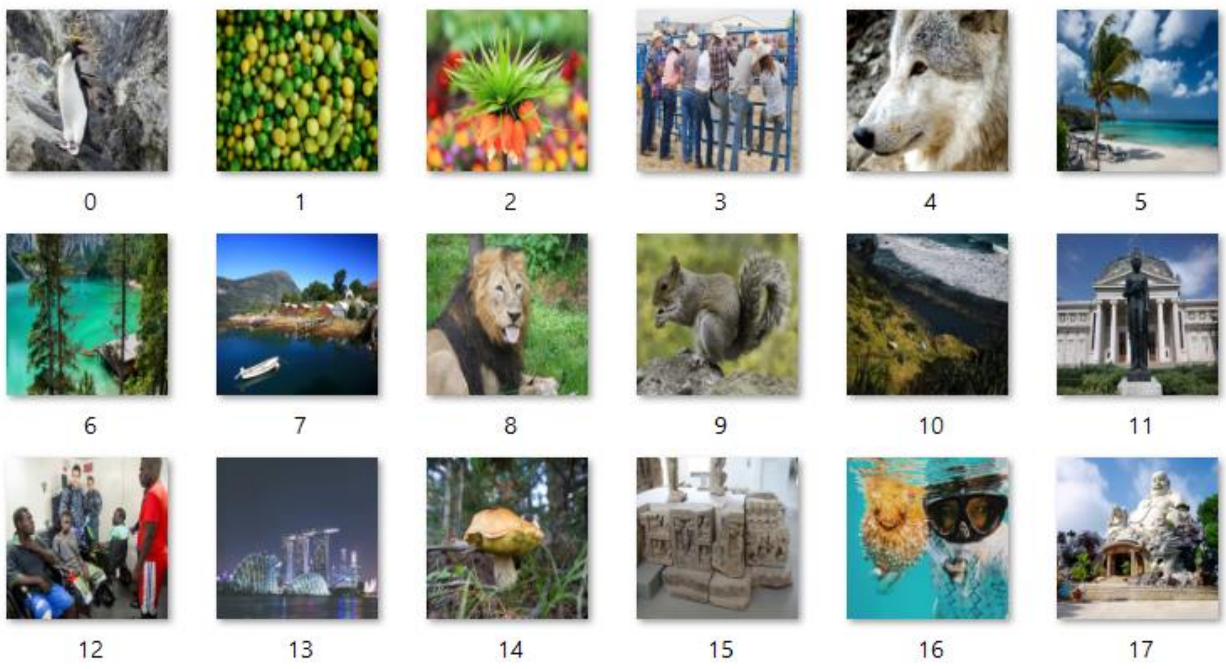
1. Working environment :

1.1. Libraries :

```python
import matplotlib.pyplot as plt
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
from PIL import Image
```

1.2. Dataset :

The dataset I used is a set of 100 low resolution images having dimensions 96X96, and the corresponding 100 high resolution images having dimensions 384X384.



2. Convolutional neural network:

2.1. Definition :

A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other.

2.2. Layers :

```
self.relu = nn.ReLU(inplace=True)

self.conv1 = nn.ConvTranspose2d(3, 64, 2, padding = 1, stride=2)

self.conv2 = nn.ConvTranspose2d(64, 64, 2, stride=1)

self.conv3 = nn.ConvTranspose2d(64, 64, 2, stride=2)

self.conv4 = nn.ConvTranspose2d(64, 3, 3, stride=1)
```

* **ConvTranspose2d** : this layer applies a 2D transposed convolution operator over an input image composed of several input planes.

For this layer, we have the following relations between input and output dimensions:

- Input: $(N, C_{in}, H_{in}, W_{in})$
- Output: $(N, C_{out}, H_{out}, W_{out})$

$$H_{out} = (H_{in} - 1) \times \text{stride}[0] - 2 \times \text{padding}[0] + \text{dilation}[0] \times (\text{kernel\_size}[0] - 1) + \text{output\_padding}[0] + 1$$

$$W_{out} = (W_{in} - 1) \times \text{stride}[1] - 2 \times \text{padding}[1] + \text{dilation}[1] \times (\text{kernel\_size}[1] - 1) + \text{output\_padding}[1] + 1$$

* **Relu** : this layer is a Rectified Linear Unit. it applies the activation function defined as the positive part of its argument :

f(x) = max (0,x)

To train our model, we start by fixing its loss function and optimizer.

2.3. Loss function :

In the backward propagation, we have to use a loss function that compares the output of the model to the target.

I used the mean square error that calculates the differences between the input and output values of pixels and gives their mean.

2.4. Optimizer :

After evaluating the error of the output, we need an optimizer function that updates the coefficients in the CNN layers.

I used the Adam optimizer.

2.5. Implementation :

```python
class ConvNet(nn.Module):
    def __init__(self, num_classes):
        super(ConvNet, self).__init__()

        self.relu = nn.ReLU(inplace=True)

        self.conv1 = nn.ConvTranspose2d(3, 64, 2, padding = 1, stride=2)

        self.conv2 = nn.ConvTranspose2d(64, 64, 2, stride=1)

        self.conv3 = nn.ConvTranspose2d(64, 64, 2, stride=2)

        self.conv4 = nn.ConvTranspose2d(64, 3, 3, stride=1)

    def forward(self, x):

        x = np.transpose(x,(0,3, 1, 2))
        x = x.reshape([-1,3,96,96])
        x = torch.tensor(x,  dtype=torch.float)


        out = self.relu(self.conv1(x))
        out = self.relu(self.conv2(out))
        out = self.relu(self.conv3(out))
        out = self.relu(self.conv4(out))

        return  out

model = ConvNet(10)
```

For the implementation, we used the set of layers as a class named ConvNet. ConvNet class inherits from nn.Module which is the base class for all neural network classes.

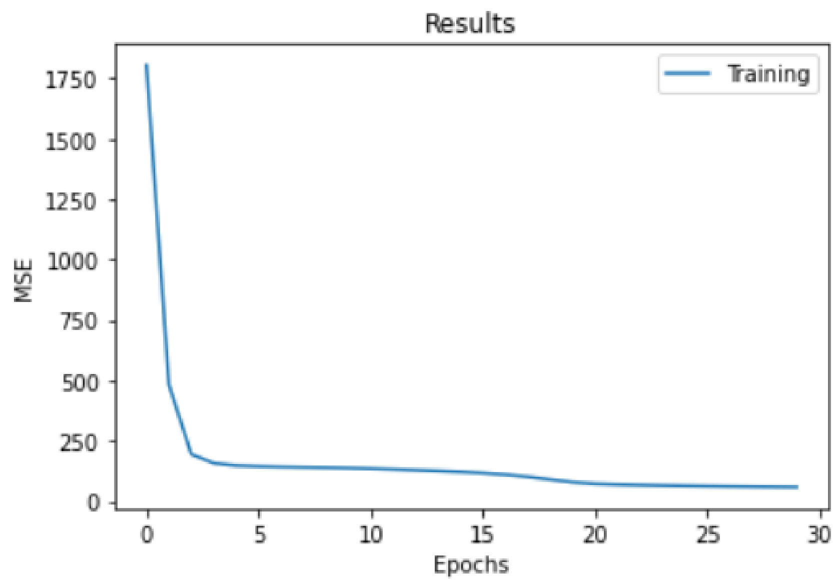
In order to train the CNN, I divided my dataset into train set and test set.

After that, I divided the train dataset into batches. Each batch is composed of 8 samples.

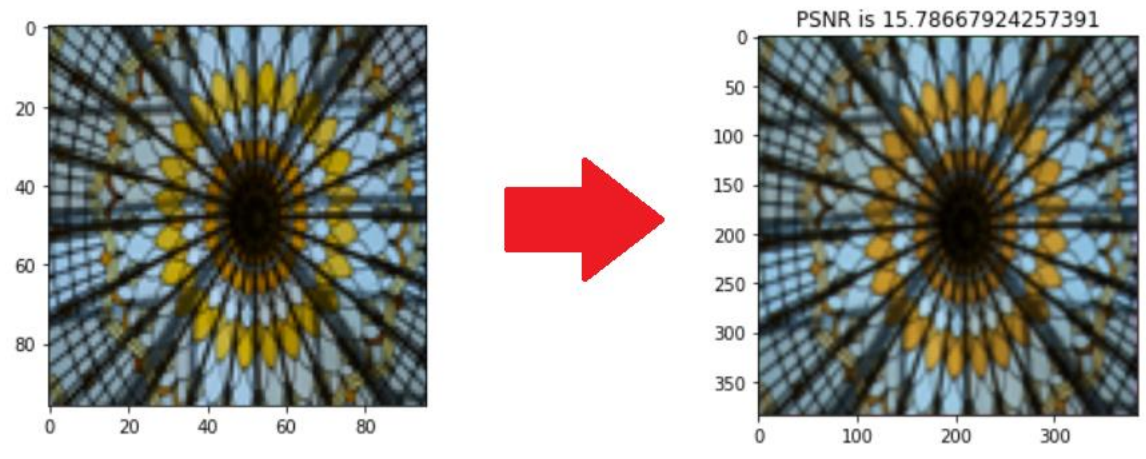
I used 30 epochs to train the dataset, which means that the model have to pass through the entire training data 30 times.

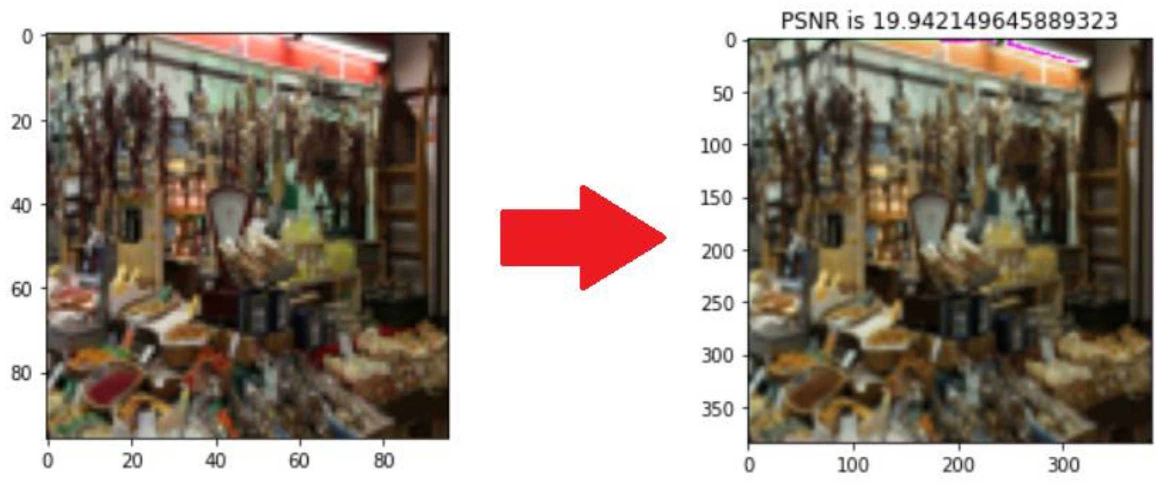
3. Results :

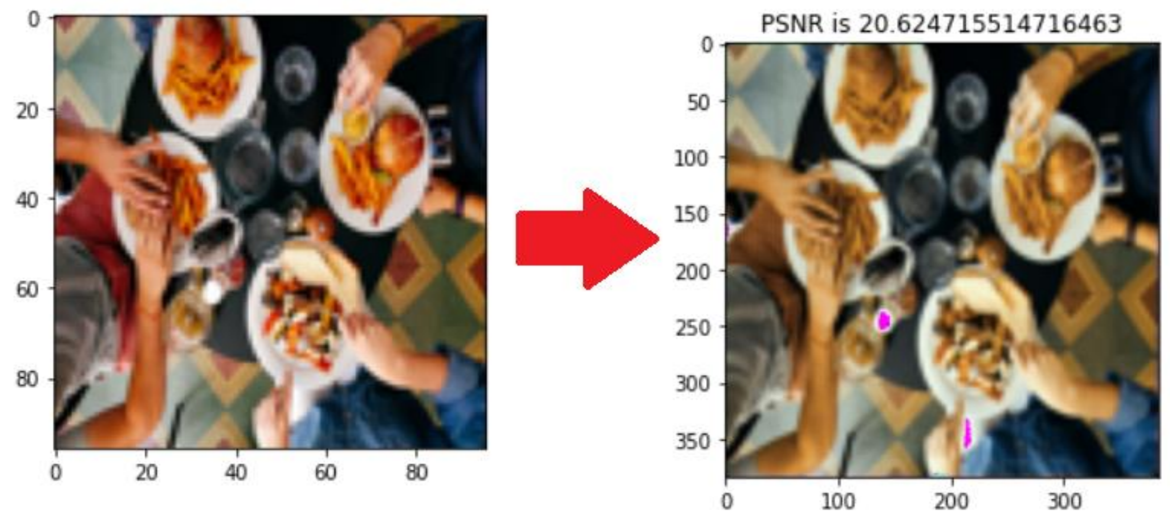
After 30 epochs, I get the following MSE values :



We remark that the MSE is decreasing, which means that the model is learning to upsample the input images and get a result close to the target.







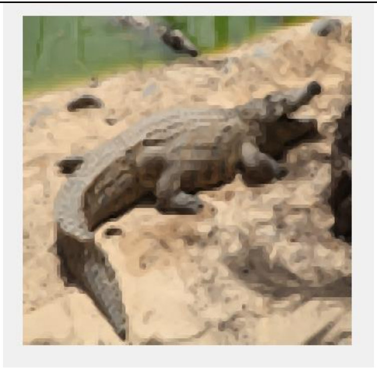
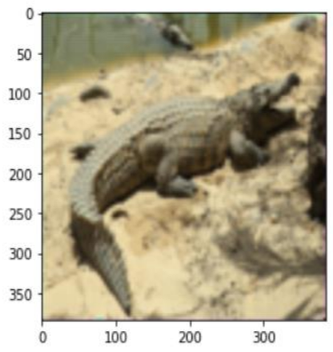
I calculated the PSNR of the test dataset composed of 12 images :

```
-------------------PSNR of image 0 ----------------------------
15.78667924257391
-------------------PSNR of image 1 ----------------------------
19.942149645889323
-------------------PSNR of image 2 ----------------------------
24.014136980705914
-------------------PSNR of image 3 ----------------------------
19.388100272785458
-------------------PSNR of image 4 ----------------------------
20.9214336722337
-------------------PSNR of image 5 ----------------------------
20.45464373703063
-------------------PSNR of image 6 ----------------------------
22.334150735778053
-------------------PSNR of image 7 ----------------------------
27.713018857552573
-------------------PSNR of image 8 ----------------------------
21.60630353551072
-------------------PSNR of image 9 ----------------------------
21.746230926024054
-------------------PSNR of image 10 ----------------------------
23.835699979783072
-------------------PSNR of image 11 ----------------------------
20.624715514716463
```

We remark that the PSNR of the test dataset vary between 15 and 24.

C - Comparison between the two methods

I used the two method on the same image and I have the following results:

	Interpolation-based	Learning-based
RESULT IMAGE		
PSNR	21.8848	21.7462

We found that the results of the two methods are close to each other.

Conclusion

As a conclusion, we found that the two types of methods give satisfactory results.

The interpolation techniques are very known and widely used in image processing tasks, however, the deep learning techniques are relatively new and they need a big training and testing dataset to give better results and avoid overfitting.

References

[1] https://www.compuphase.com/graphic/scale2.htm#CARRATO1

[2] https://pytorch.org/docs/stable/generated/torch.nn.ConvTranspose2d.html

[3] https://www.kaggle.com/akhileshdkapse/super-image-resolution