Single Image Super-Resolution GAN

Nicolas THOU[1]

[1] *nicolas.thou@student.uliege.be (s198327)*

I. INTRODUCTION

Single image Super-resolution imaging is a field of research that is already tackle by researchers for few years. This techniques for enhancing the quality of a low resolution image is used in medical application, in some radar and medical applications. Mapping a low resolution image (LR) to a high resolution image (HR) induce a loose of information in comparison to the original image. The main issue raised in this field of researcher is the reconstruction of natural image where unfortunately we don't reach a satisfying mapping that reconstruct accurately the original image we aim for. The research question that is clearly raised in this field is the following one : **From a low resolution image, how can we generate fine structure details to obtain a high resolution image ?** One interesting work has been proposed by Christian Ledig et al from Twitter where they used the power of a GAN in order to achieve a perceptually satisfying reconstruction. Indeed, my project is based on this paper where I have to code and study the SRGAN (Super Resolution Generative Adversial Network). The code of this project has been influenced by Kailash Ahirwar [2] from which I obviously modify. Last but not least, this project is also a way to understand deeply each tricks and choice of the code of SRGAN. The architecture of neural network follows what is proposed on the article [1]. The main goal of the project is to code a neural network that generate details which leads to a better results than approximating many intractable details from a low resolution image. Along this report, LR will mean Low resolution and HR will mean High Resolution.

II. BACKGROUND

A. Generative Adversarial Network

Generative Adversarial Network (GAN) is the fight between two networks where both trains in order to compete each other. It is an unsupervised model where we don't have ground truth in order to compare our outputs. This architecture is composed by 2 main neural network which are the Generator G and the Discriminator D. The Generator tends to approximate as close as possible the data distribution of the dataset that we want to generate. The Discriminator is a classifier that tends to recognize between an original image and a generated image. The idea is based on a minimax two-player game. [3] In the following subsection I will emphasize the explanation of the architecture of the generator. Indeed, the discriminator is only an extractor features followed by a fully connected layer in order to classify images. The method to accomplish a satisfying training between the both is to train independently the discriminator and the generator in the way that G fools the discriminator D and the latter unrecognized the true image and the fake image. The best probability for a discriminator is supposed to be equal to 0.5 where he cannot distinguish between the two distribution, one from the training set the other from the latent space produce by the generator.

1. Generator

From a randomly generated vector of number, the generator has to generated a vector with the same distribution as the training set. The generator try to minimize the binary cross entropy loss of the Discriminator. He aims to fool the discriminator. When we create a generator, we have to specify his goal for instance generation of texts, images, or sounds. In our case, the generator will take as input the low resolution image and the original image (The original image is used for the loss function). Then, his goal is to upscale the low resolution image. The core idea of this generator remains on the last layers which use **transpose convolution**. From a feature map, we want to output an image. The transposed convolution is simply the inverse operation of a convolution layer. In this project, we use the same architecture of [8] as introduce in the Fig 1.

For the SRGAN, the generator use a **sub-pixel convolution** that upscales the LR feature map using the Nearest Neighbor interpolation and then employs a standard convolution layer to output the HR space. It has the same computational complexity than a transposed convolution (even faster) and has more parameters and thus can generalize more. At the end of the multiple transformations from upscaling feature maps, the last layer has to output the HR image by a **periodic shuffling**. This function take as input all the channel input, and output the 3 channel RGB of the HR image. [10] The second interpretation of the sub-pixel convolution is given by [9], where we map the feature map to a sub pixel space by adding row, columns of zero between pixel and a padding of zero. Then, a kernel maps the sub pixel space and output directly the HR image. But we can notice that the filter used in the sub pixel space can be break into sub filters as shown in the Fig 1 (filter of different colors represent the sub filters). Those filters act independently because of the row and column of zero. Finally we can learn an upscaling mapping thanks to the weights of the filters. This convolutional layer is motivated by the com-

putanional cost in comparison to a classic deconvolution layer.

2. Least Squares GAN

Using the Least Squares Generative Adversarial Networks (LSGAN) for implementing the loss function for the discriminator is very powerful in terms of perceptual quality. In fact, researchers noticed the instability of GANs learning is partially caused by the objective function. Specifically, minimizing the objective function of regular GAN suffers from vanishing gradients, which makes it hard to update the generator. LSGANs can tackle this problem because it penalizes samples based on their distances to the decision boundary, which generates more gradients to update the generator. [4]

For the discriminator D, a true label equal to 1 and a fake label equal to 0, the loss function is described as follow :

$$min_D L(D) = \frac{1}{2} \mathop{\mathbb{E}}_{x \sim p_{data}(x)} [(D(x) - 1)^2] + \frac{1}{2} \mathop{\mathbb{E}}_{y \sim p_{data}(y)} [(D(G(y)))^2] \quad (1)$$

In fact, using the sigmoid cross entropy loss allows the generator to produce images that are on the correct side of the decision boundary. However, those images are far from the real data distribution. So the generator will achieve to fool the discriminator but the discriminator won't give a signal to show the discriminator how correct is the generated image. It will only give a feedback on whether or not the generated image is classify as a real example. With the least square loss, the more the generated image is far from the boundary, the more the generator is penalized. And it happens even if the generated image is on the right side of the boundary decision. Then the fake example becomes more closer to the real data. The least square loss rely on the distance between the generated image and the decision boundary. The latter has to cross the real data, then making the generated image closer to the decision boundary will lead them to be closer to the real data distribution. In [5], they show that instead of minimizing the Jensen-Shannon divergence, the use of least square loss leads us to minimize the Pearson χ^2 divergence between $p_d + p_g$ and $2p_g$. For GAN, the measure of similarity between the distribution of the generated data and the true data (so that we can optimize the parameters our generation model) needs various statistical divergences. A statistical divergence $D(P\|Q)$ between two probability distributions P and Q maps two distributions to $\mathbb{R}$. The two distributions are very similar if $D(P\|Q)$ is really close to zero.

B. SRGAN Loss

The interesting part of [1] is there contributions about the losses function. The authors have the idea to use the VGG Loss which is more invariant to changes in pixel spaces. In fact, the mean square error loss enhance the quality of an image by averaging over multiple solutions. However, this technique has a lack of details for the reconstruction images. So they use a weighted loss between the binary cross entropy loss that pushes the generator to approximate the distribution of natural image and the VGG loss that allows us to obtain better perceptual similarity instead of similarity in pixel space. The losses function are summarized in the following equations : The first equation (Eq 2) is the minimization of the loss from the discriminator in order to be closer to the distribution of the dataset. D_{θ_D} is the probability of the discriminator if the argument is a natural image. $G_{\theta_G}(I^{LR})$ is the generated image from the Low resolution image I^{LR} .

$$l^{SR}_{Gen} = \sum_{n=1}^{N} = -\log D_{\theta_D}(G_{\theta_G}(I^{LR})) \quad (2)$$

The second equation is VGG Loss function. Instead of comparing each pixel of an image, we compare the feature map of the two images, from a VGG-19 pretrained. In the code, we use the feature map from the layer 9 in VGG-19 and we compare the two features map with a size (64×64). So this loss compute the euclidean distance between features. (Eq 3)

$$l^{SR}_{VGG/i,j} = \frac{1}{W_{i,j} H_{i,j}} \sum_{x=1}^{W_{i,j}} \sum_{y=1}^{H_{i,j}} [\Phi_{i,j}(I^{HR})_{x,y} - \Phi_{i,j}(G_{\theta_G}(I^{LR}))_{x,y}]^2 \quad (3)$$

Finally, the SRGAN follows the weighted loss function :

$$l^{SR} = l^{SR}_{VGG/i,j} + 10^{-3} \cdot l^{SR}_{Gen} \quad (4)$$

So we pay much more attention to the perceptual similarity than to approximate the data distribution. As we don't want to generate some fake images, we want instead to reconstruct some details. The generator doesn't take as input a random vector but the LR image. That could explain why the authors of [1] put more weight on the $l^{SR}_{VGG/i,j}$ loss function.

C. Peak Signal-to-Noise Ratio (PSNR)

Peak Signal-to-Noise Ratio measure the difference between the original image and the reconstructed one in decibel. It calculates the ratio between the maximum

fluctuation in the input image data type and the MSE. For example, if the input image has a double-precision floating-point data type, then MAX is 1. If it has an 8-bit unsigned integer data type, MAX is 255. I recall that the mean squared error (MSE) is computed as the average of the squared intensity differences between a source image and a distorted image.

$$\mathbf{PSNR}(x,y) = 20 \cdot log_{10} \frac{MAX}{\sqrt{MSE}}$$

The higher the PSNR, the better the quality of the compressed, or reconstructed image. [7]

D. Structural Similarity Index (SSIM)

Structural Similarity Index is used in order to measure the quality of an image by evaluated three features of an image : luminance, contrast and structure. This metric aims to keep only the structure of an image independent from the luminance and contrast by taking only a local luminance and contrast of the image (illumination). If we compare two images $\mathbf{x}$ and $\mathbf{y}$ we need the following equations to compute the SSIM metric :

$$\mathbf{l}(x,y) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1}$$

$$\mathbf{c}(x,y) = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2}$$

where

$$C_1, C_2 = K_1L^2, K_2L^2$$

with K_1, K_2 a small constant and L the dynamic range of the pixel values (255 for 8-bit grayscale image).

And finally the structure comparison as :

$$\mathbf{s}(x,y) = \frac{\sigma_{xy} + C_3}{\sigma_x\sigma_y + C_3}$$

The authors of [6] use the standard deviation (the square root of variance) as an estimate of the signal contrast (σ_x, σ_y). The local means are μ_x, μ_y and the cross covariance is σ_{xy}.

Then, the SSIM is the combination of the three measure :

$$\mathbf{SSIM}(x,y) = \mathbf{l}(x,y)^\alpha \cdot \mathbf{c}(x,y)^\beta \cdot \mathbf{s}(x,y)^\gamma$$

where the coefficients in the power allow us to adjust the relative importance of the three components. Moreover, the three components are relatively independent. For instance, the change of luminance and/or contrast will not affect the structures of images [6].

As PSNR Metric, the higher the SSIM, the better the quality of the compressed, or reconstructed image (range 0 to 1).

III. METHODS

A. The models

The model I want to focus on first is the generator. After 16 Residual blocks, I use two Upsampling2D layers following each one by a convolutional layer and an Activation function Relu. The last layer is a Conv2D layer following by an Activation tanh function. I don't use directly a Conv2DTranspose layer which would replace the third previous block of layers. You can notice that the last convolutional layer has only 3 filters in order to generate an image of 3 channels (RGB channels).

For the Discriminator, I stack 8 convolutional blocks for the extractor features and put 2 Dense layers as classifier. The VGG-19 is pretrained on imagenet. We use his hidden layer as an output because we need the feature map from this model in order to compare via MSE the feature map of the generated image and the HR image.

The architecture of residual block follow the paper [7] where they show that adding batch normalization after the short cut hurts the performance of a Resnet. In fact, it would normalize the entire block and each batch normalization apples it's own distortion. So it would modify the original input. [7]

B. The research protocol

1. *Training*

In this project, I try to code and understand how I can train a generator that enhance quality of an image. After having train an adversarial model where I have to first train the discriminator, then to fix his weights and use his prediction in order to train the generator. To optimize our adversarial network, I alternate between one gradient descent step for D, then one step for G. I use the Adam optimizer for both networks G and D. In my experiment, I use batch size equal to 16 because of memory limitation per cpu, 1 batch per epoch and 6264 epochs. Normally, the product of the number of batches per epoch and the batch size has to be equal to the size of the training dataset in order to let every image to have a chance to be process. But I chose to have more epochs so as to have a plot of the loss functions more interpretable. The learning rate is set to 0.0002 and the exponential decay rate for the 1st moment estimates is set to 0.5, and for the 2nd moment estimates is set to 0.999. VGG is a pre-trained model and I don't need to train it. The discriminator use the mean square error as loss function (Cf Least Squared GAN). The generator has a weighted sum of the loss binary cross entropy (weight set to 0.001) and the VGG Loss (weight set to 1) following the article [1]. Low resolution images have dimension (64, 64) and for an x4 upscaling, I have the original image and the high resolution image with a dimension set to (256, 256). The

fake and real labels for the discriminator are set to a vector of 0 (fake) and 1 (real). In my project, the generator act in a same way as a deconvolution layer, but in fact, I stack 2 upsampling layer which use **nearest neighbor** algorithm to fill the row and columns of vectors. It simply double the row and columns. And every upsampling layer has to be followed by a convolutional layer which interpret and fill the upsampled feature map by fine details. I recall that the filters (that contains weights) learn how to reconstruct the details of a low resolution image. Moreover, the layers have been initialized by the Xavier Initialization. It initialize the weight over a uniform distribution in range of $-limit; +limit$ such that :

$$limit = \sqrt{\frac{6}{fan - in_{input} + fan - out_{output}}}$$

where $fan - in_{input}$, $fan - out_{output}$ are the number of input/output units in the weight tensor. We apply data augmentation with a random horizontal flip each time after resizing the batch of images.

The training is composed in different parts and follow the below algorithm 1, where the testing algorithm is more straight forward and follow the Algorithm 2. Moreover I present some results generated during the training in the section *Results*.

In order to downscale the image of the training set and then rescale it to a 256×256 size I use the bilinear interpolation method (openCV). This technique is widely use in computer vision in order to rescale images. From an orgininal image, I downscale the image, the algorithm remove one row and one column over two. It's common to downscale an image by subsampling. And then, to avoid the aliasing problem we upscale the image by bilinear interpolation where the algorithm interpolate the value of each unknown points by the 4 nearest neighbors. It's still not as well as the original image. So bilinear interpolation is 2 linear interpolation vertically and horizontally with respect to the unknow point, then by substitution and combination of equations we can obtain the final color of the unknown point. This works by interpolating pixel color values, introducing a continuous transition into the output even where the original material has discrete transition. For the upscaling part with bilinear interpolation, it is only used for comparing many method and the SRGAN. Only the downscaling part is useful for the training of SRGAN.

I normalize the image between -1 and 1 as the tanh function at the end of the generator output value between -1 and 1. The benefit of normalize image is to make the training less sensitive to the weights. Indeed, the weights are small value, and could not have an impact if they work on value between 0 and 255.

2. Testing

The test for the SRGAN use the two metrics described above i-e Peak Signal-to-Noise Ratio and Structural Similarity Index implemented by Tensorflow *ssim* and *psnr*. I test 100 images which I downscale and then try to output the high resolution image and compare it with the original one. And I compute the mean of the result metric obtain for those 100 images. After training the adversarial model, I test the generator over a batch of image to compute the average PSNR and SISM and compare the nearest neighbour, the bicubic interpolation, the bilinear interpolation, and the SRGAN method. The results is summarized on the table (Fig 2). We can see from the results and also from the generated image that GAN is better perceptually in comparison to the classic methods. I have also tested to put some image of a group of people, a car and some text. I notice the fact that the SRGAN doesn't generate good text. The car is still blurry and the group of people is better than the car and the text, in particularly because the SRGAN has been trained on distribution of face of humans. You can see those results in the section *Results* in the following pages.

However, we have to mention that the ability of PSNR to capture perceptually relevant differences, such as high texture detail, is very limited as it is defined based on pixel-wise image differences. PSNR does not necessarily reflect the perceptually better SR result. [1].

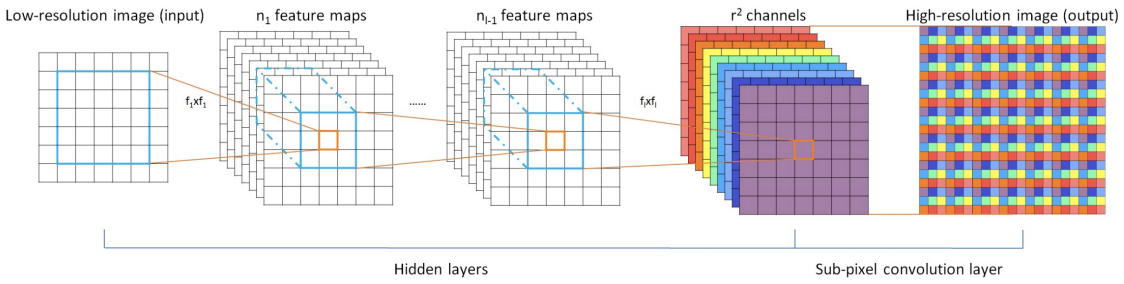


FIG. 1: After extracting the feature map, increase the resolution from LR to HR only at the very end of the network and super-resolve HR image from LR feature maps

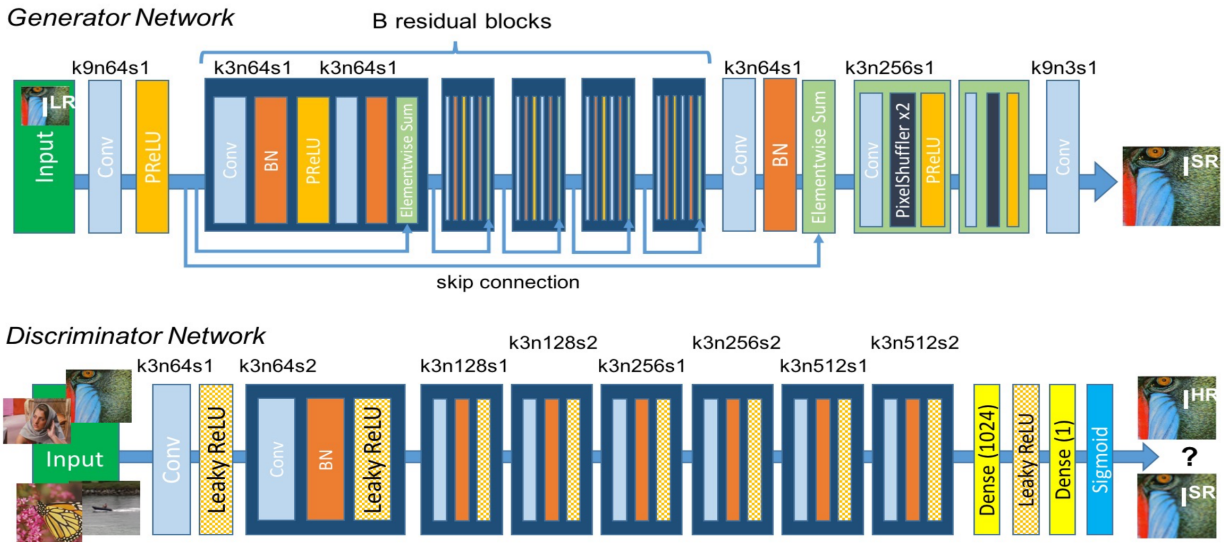


FIG. 2: Architecture of the Generator (top) and the Discriminator (down) with corresponding kernel size (k), number of feature maps (n) and stride (s) indicated for each convolutional layer. The generator has B equal to 16 residual blocks

Algorithm 1 Training algorithm for the adversarial model

Result: training adversarial model
vgg ← build_vgg()
discriminator ← build_discriminator()
generator ← build_generator()
for epochs until number_of_epoch **do**
  for batch until number_of_batch **do**
    Training the Discriminator
    high_resolution_images, low_resolution_images ← shuffle_convert_low_high(celebA dataset)
    generated_images ← generator(low_resolution_images)
    Compute the least square loss and back propagated
    Training the Generator
    high_resolution_images, low_resolution_images ← shuffle_convert_low_high(celebA dataset)
    Fix the weights of the Discriminator
    image_features_target ← vgg.predict(high_resolution_images)
    Compute the VGG Loss and the Binary Cross Entropy and back propagated
  end for
end for

Algorithm 2 Testing Algorithm for the generator

```
Result: Testing the generator
generator = build_generator()
generator ← weight saved during training
for i from 1 to 10 do
  original_list, low_list ← 10 random images in test set
  for original, low in original_list, low_list do
    high_resolution_image = generator(low)
    PSNR_list_generator ← PSNR(high_resolution_image, original)
    SSIM_list_generator ← SSIM(high_resolution_image, original)
    high_resolution_image_neireast = neireast_neighbor(low)
    PSNR_list_neireast ← PSNR(high_resolution_image_neireast, original)
    SSIM_list_neireast ← SSIM(high_resolution_image_neireast, original)
    high_resolution_image_bicubic = bicubic(low)
    PSNR_list_bicubic ← PSNR(high_resolution_image_bicubic , original)
    SSIM_list_bicubic ← SSIM(high_resolution_image_bicubic , original)
    high_resolution_image_bilinear = bilinear(low)
    PSNR_list_bilinear ← PSNR(high_resolution_image_bilinear, original)
    SSIM_list_bilinear ← SSIM(high_resolution_image_bilinear, original)
  end for
end for
For the 4 methods, return mean(PSNR_list), mean(SSIM_list)
```

Metric values during test				
Metric	Nearest Neighbour	Bilinear Interpolation	Bicubic	SRGAN
PSNR	20.34	23.31	23.53	**23.77**
SSIM	0.664	0.768	0.767	**0.802**

FIG. 3: Table showing the PSNR and SSIM metric for 4 methods. Those results are from the test over 100 images.

C. Analyze the data

The interpretation of the losses that I obtain along the training is showed in 3 plots. I show the loss function and the accuracy of the discriminator. The last plot is the loss function of the generator. From the accuracy plot of the discriminator, we can notice that I never reach the optimal discriminator where D has to output 0.5. It seems that the discriminator at the beginning decrease because the generator improve on the generation of super resolution image. But after 500 epochs, the discriminator increase his accuracy to 1 on average and becomes really strong for 5000 epochs. Until 5800 epoch, the accuracy seems to reach the optimal discriminator, but we should train the SRGAN more longer to be sure that we reach a stage.

For the SRGAN Loss function, it decreases really fast and converge since the begining. However, when the loss function of the discriminator show a peak, the loss function of the SRGAN shows up also with a peack (for instance at 2300 epochs). So if the discriminator reaches a constant level for his loss since 6000 epochs to a value of 0.25, then the loss of the SRGAN will be also impacted. And it begins the classic issue of Convergence during training for GANs. However, as the generator rely more on the VGG Loss, since 6000 epochs, even if the loss function of the discriminator increase to 0.25, it seems that the SRGAN Loss is quiet resistant about the error of the discriminator.

IV. CONCLUSION

To conclude this project, the Single Image Super Resolution research through SRGAN is satisfying in comparison to classical method. Notice that the state of art of Super Resolution Image outperform the SRGAN. But, from this project we can have a good intuition for the new state of art models. All the new models don't use GAN. This project still shows me the power of GANs. I don't really know if the GAN can be improved because the discriminator is way too much trained. At a certain point of the training, the discriminator has an accuracy of 1. So it doesn't provide enough signal to train the generator. But on the other hand, the SRGAN Loss is given by the VGG Loss, and we see that the Loss function of the adversarial model decreases and stabilizes. It means that the difference of the features map between a generated image and an original image is not really different according to the mean square error. So that could explain the quality of the generated images from the model. In order to answer to research question, the SRGAN is a model that trained a generator in a way that it wants to fool a discriminator. So the fighting between two neural networks lead the SRGAN to reach a point where the generator can generate images with a lot of details. Moreover, the quality of the outputs remains on the VGG Loss where the generator is penalized if the generated image has a different feature map (through VGG) than the feature map of the original image. Ultimately, I show that the SRGAN cannot be generalized over different type of image than faces (train on human faces) as the text or even the car. For the group of people and the car, it is still hard because the goal is to reproduce fine structure details and not to output an overall image which is perceptually correct. We want more than an average, but we seek to obtain the details of an image (for instance, numberplate of the blue car).

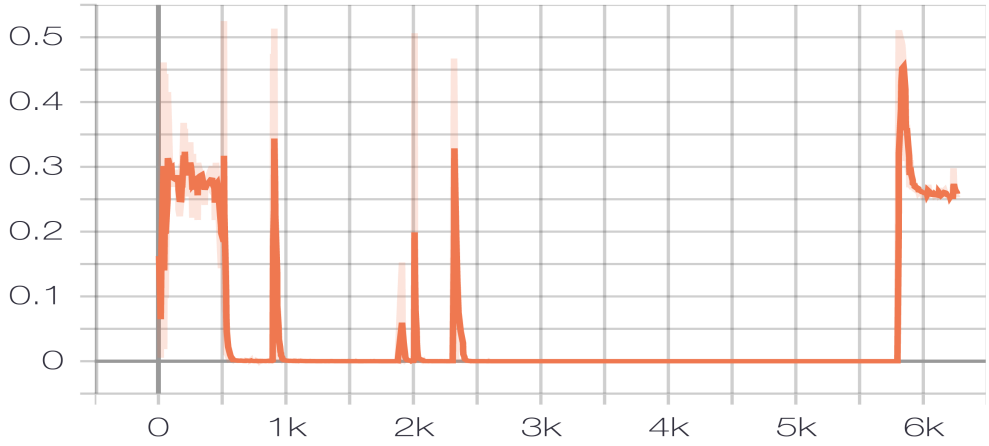


FIG. 4: Loss function for the discriminator (MSE)

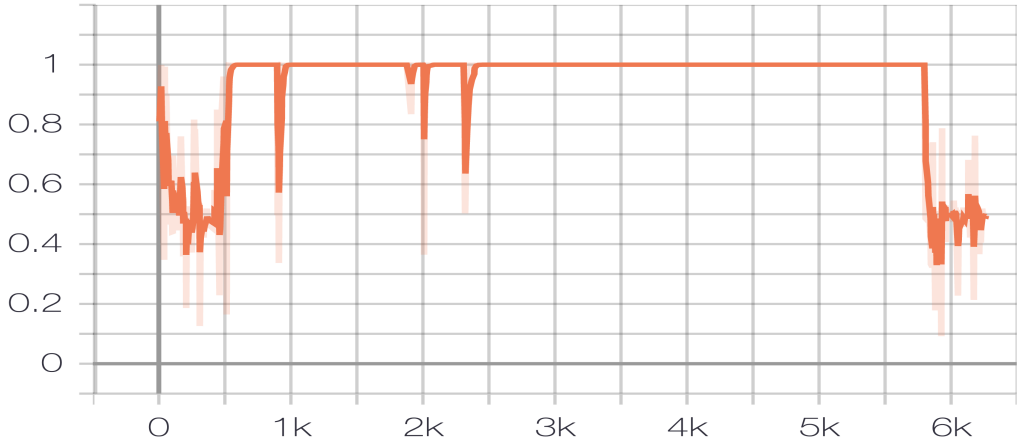


FIG. 5: Accuracy for the discriminator

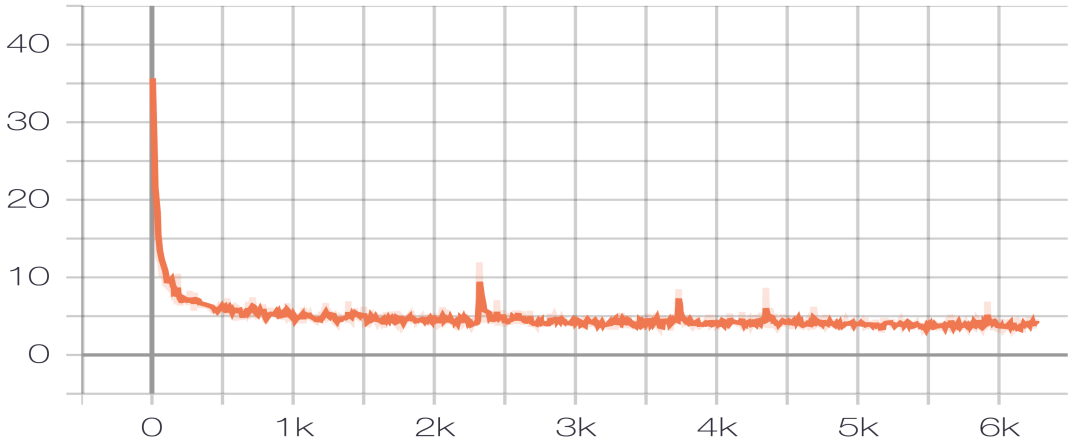


FIG. 6: Loss function for the adversarial loss (weighted loss of VGG Loss and Binary Cross Entropy)

V. RESULTS

This section show, in the page 11, some results along the training of the adversarial model. We can interpret some of the generated images with respect to their particularities. We have to keep in mind that the main goal is to reproduce details. So an output that seems similar than the original but without details is not satisfying.

VI. ANNEX

A. VGG convolution neural network

The VGG-19 is the extractor features used during the project for the VGG Loss. The model uses during the project is a pre-trained model on ImageNet. I took from VGG the third convolutional layer in the fifth convolutional block. A convolutional block is composed by 4 convolutional layers and one max pooling layer.

B. Residual block - Skip-connection

Residual block has the particularity to be really deep and use skip connection (also called short cut) in order to avoid the vanishing gradient problem. The vanishing gradient problem occurs in deep networks, since with the chain rule during back propagation a lot of gradients are multiplied. If only one of those gradients is very close to zero, the whole product becomes very small. It is the contrary of the exploding gradient problem where the value of certain gradients is very large and gives the main effect on the update since it is multiple magnitudes larger. Using residual blocks allow to train much deeper neural networks and to back-propagated the values until the earlier layer without becoming too small. The discriminator uses a chain of **16** residual blocks. The residual block I use in this project is composed by two [convolution layer, batch normalization] where for the last pair there is no activation.

C. Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers

Since the computer has a discret level of precision that it can display, if we add value of pixel that is higher than 255 for integers for instance :

$$224 + 21 = 245$$

$$224 + 7 = 252$$

$$224 + 12 = 264 \dots 8$$

This means that the brightest pixel become the tiny pixel. However, with intensity clipping, we stop this phenomenon by setting the maximum value as 255 for example. Addition to this point do not roll over but clip. So 264 in the example will be equal to 255.

D. Training GAN issues

Training a GAN can hidde many issue. We have to handle the discriminator training in one hand and the training of the generator on the other hand with a fix discriminator. Moreover we are not able to identify well the convergence of a GAN. If the discriminator is too trained, then the generator won't be update as the gradient has low value close to zero. On the other hand, if the generator is too trained, the generator won't have accurate feedback to produce details of the real data. Moreover, when we reach the point where the discriminator has an accuracy of 50%, the discriminator feedback gets less meaningful over time. [4] The discriminator classification becomes random and the feedback for the generator could corrupted the generation images and its own quality may collapse. For a GAN, convergence is often a temporary, rather than stable. [4]

1. Mode Collapse

Usually you want your GAN to produce a wide variety of outputs. You want, for example, a different face for every random input to your face generator. However, if a generator produces an especially plausible output, the generator may learn to produce only that output. In fact, the generator is always trying to find the one output that seems most plausible to the discriminator. If the generator starts producing the same output (or a small set of outputs) over and over again, the discriminator's best strategy is to learn to always reject that output. But if the next generation of discriminator gets stuck in a local minimum and doesn't find the best strategy, then it's too easy for the next generator iteration to find the most plausible output for the current discriminator. Each iteration of generator over-optimizes for a particular discriminator, and the discriminator never manages to learn its way out of the trap. As a result the generators rotate through a small set of output types. This form of GAN failure is called mode collapse [4]

E. Normalization

To normalize an image between two values a and b such that every scalar of the tensor representing the image is in the range [a ,b], we can use the following formula

$$x' = (b - a)\frac{x - min(x)}{max(x) - min(x)} - a$$

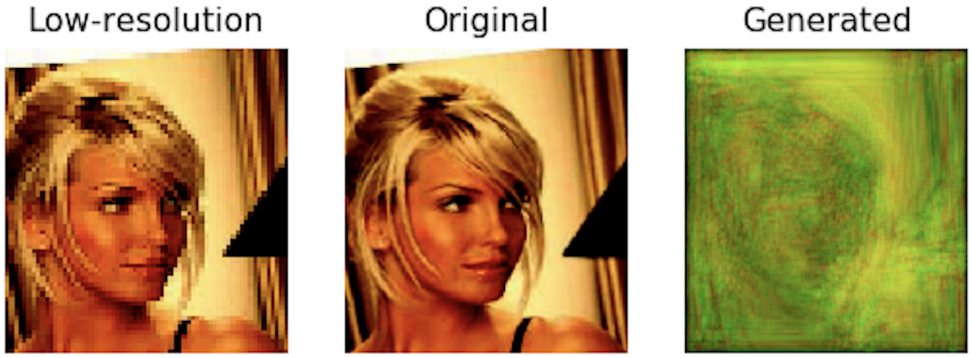
F. Model

We use Input() where its purpose is to instantiate a Keras tensor. A Keras tensor is a TensorFlow symbolic tensor object, which we augment with certain attributes that allow us to build a Keras model just by knowing the inputs and outputs of the model. For instance, if a, b and c are Keras tensors, it becomes possible to do: model = Model(input=[a, b], output=c). It is according to the Keras library.

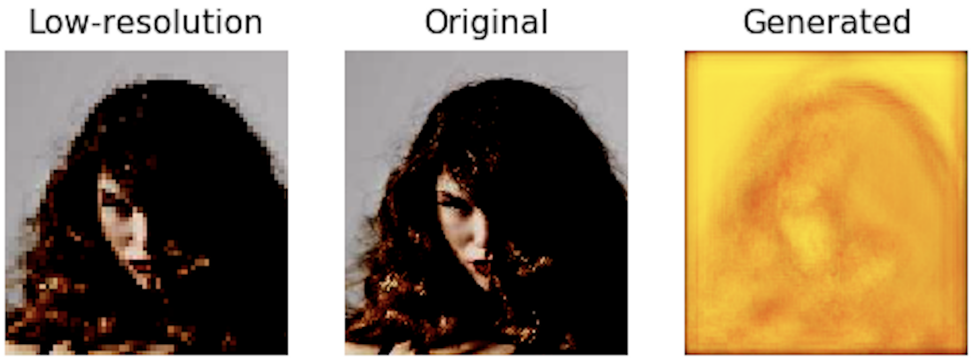
VII. BIBLIGRAPHY

Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network - Christian Ledig et al [1]
https://github.com/kailashahirwar/Generative-Adversarial-Networks-Projects/tree/master/Chapter05 [2]
Generative Adversarial Nets - Ian J. Goodfellow et al [3]
https://developers.google.com/machine-learning/gan/training [4]
Least Squares Generative Adversarial Networks - Xudong Mao et al [5]
Image Quality Assessment: From Error Visibility to Structural Similarity - Zhou Wang et al [6]
https://fr.mathworks.com/help/images/ref/psnr.html [7]
Training and investigating Residual Nets - February 4, 2016 by Sam Gross and Michael Wilber [7]
Real-Time Single Image and Video Super-Resolution Using an Efficient - Sub-Pixel Convolutional Neural Network - Wenzhe Shi and al [8]
Is the deconvolution layer the same as a convolutional layer? - Wenzhe Shi et al [9]
Checkerboard artifact free sub-pixel convolution - Andrew Aitken et al [10]

FIG. 7: This is the early generation of high resolution image. The generator is obviously not trained.

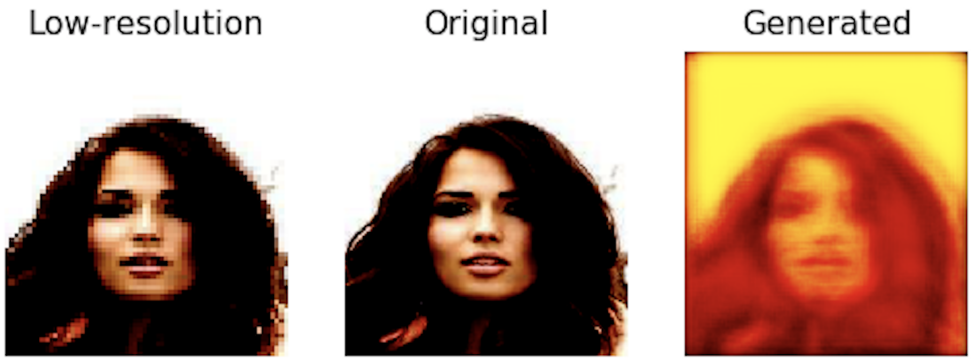


(a)

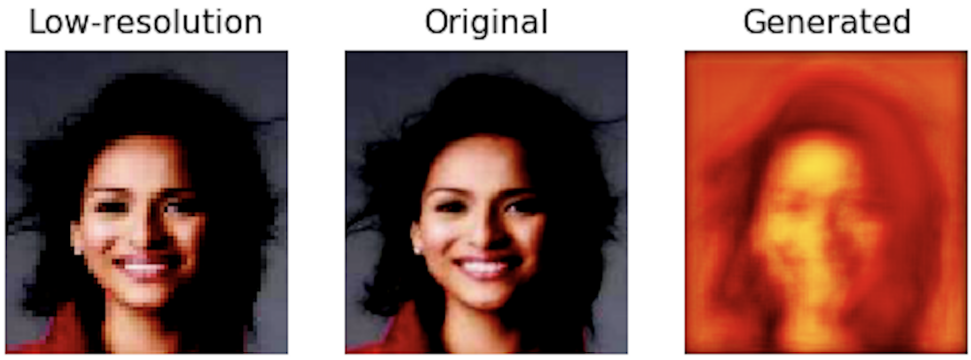


(b)

FIG. 8: For those images (a) and (b), we start to notice some shape, and edge. (Train over 5 epochs)

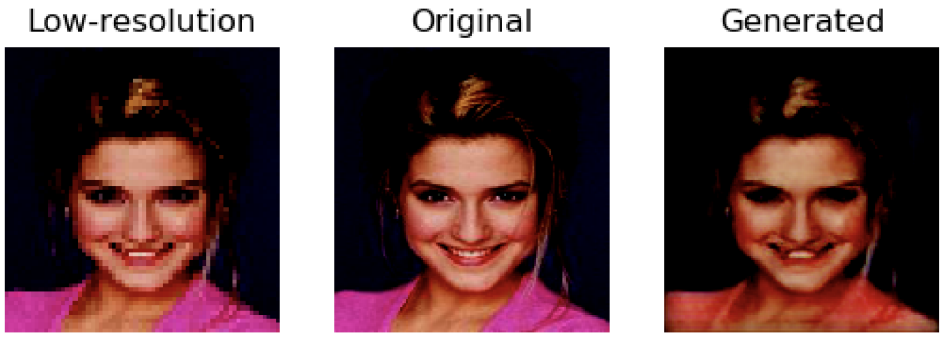


(a)

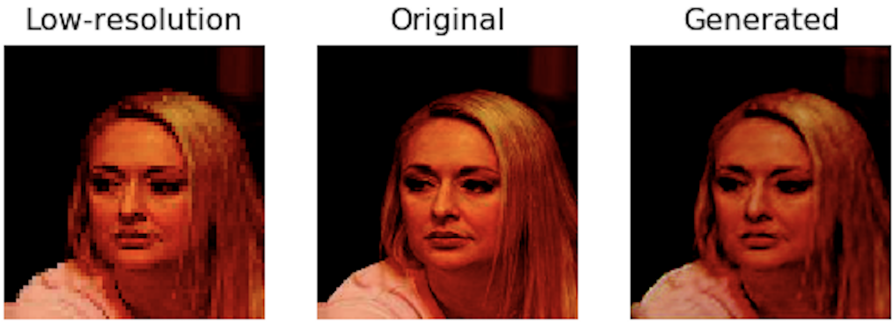


(b)

FIG. 9: The generator distinguish two colors to enhance the shape of the person.(Train over 20 epochs)



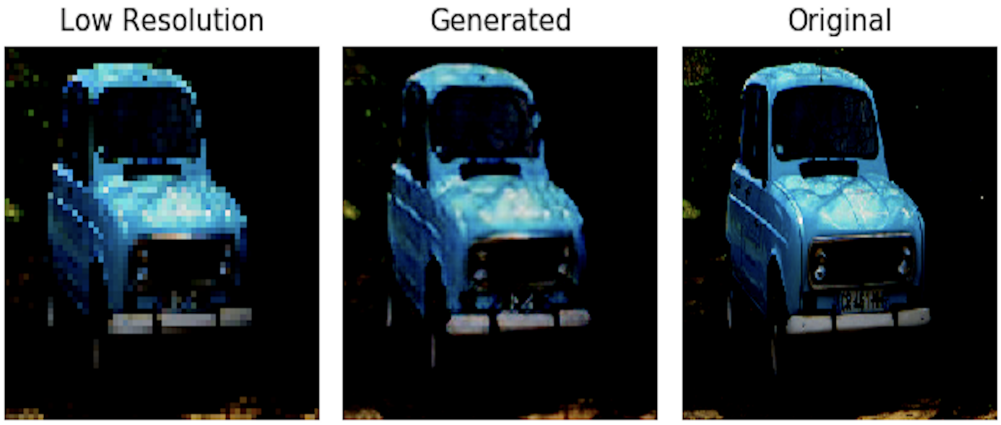
(a)



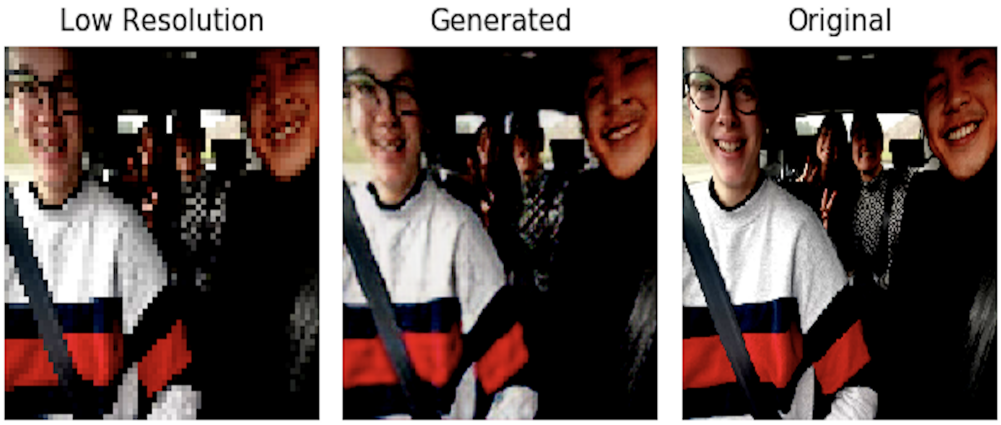
(b)

FIG. 10: The (a) image has a lack of details, for instance the eyes are still black. (100 epoch) The second block of photos (b) present generate a person with details but is still perceptually as low as the resolution of the LR image

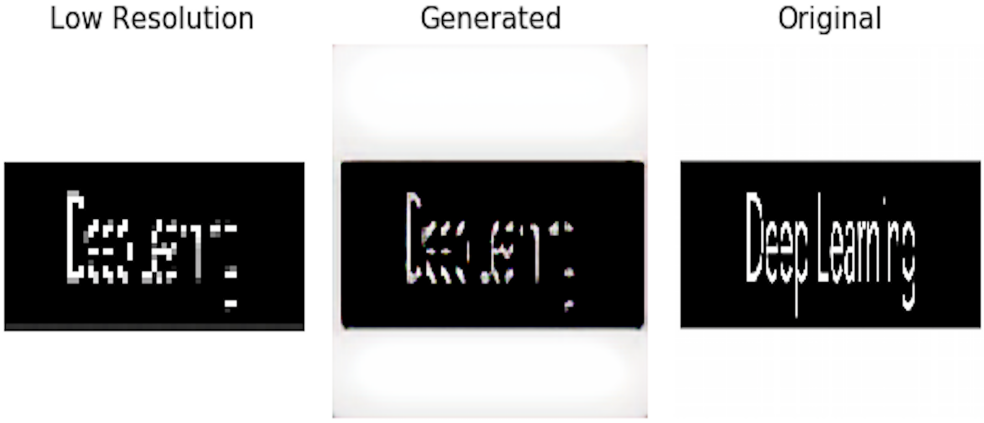
FIG. 11: After more than 6000 epochs, we can obtain high resolution reconstruction. The SRGAN is much more accurate than the other methods at the scope of hair and reproduce with a higher quality the details



(a)



(b)



(c)

FIG. 12: (a) The car generated has a perceptual good quality but obviously, it doesn't have details.(b) For the group of people, it seems that the SRGAN is used to see faces of human as the generated photo is perceptually good. (c) For the text, SRGAN is not able to reconstruct something. However not only, there are many possibilities from the low resolution image but also, the SRGAN doesn't train on dataset with image that have this kind of distribution