Globally and Locally Consistent Image Completion

Retourné Grégoire, Counathe Pierre
MODAL INF 473V : Deep-Learning in Computer Vision

Abstract— **This is a project report about our implementation and results obtained with the Globally and Locally Consistent Image Completion method exposed in a 2017 research paper published by Satoshi Iizuka, Edgar Simo-Serra and Hiroshi Ishikawa. This is a school project for the INF437V MODAL : Deep Learning in Computer Vision. We implemented the project in Python, using mainly Pytorch. Afterwards we had our networks trained during one week on two different datasets.**
We present in this document:

1) **The issue and the point of the project**
2) **Our implementation, training algorithms and utility functions**
3) **Our choices for the training and datasets**
4) **Our results**

I. Issue and point of the project

Our project was to solve a very concrete problem using deep neural networks: the image completion problem. It is defined as followed: we give as an input an image with a hole (a bunch of pixels missing), and the network outputs the completed version of the image. The goal is to obtain, after thoroughly training the network, a realistic completion of the image, that could even fool certain human beings.

To do so we used a Generative Adversarial Network (GAN). It is composed of two distinct neural networks : the Generator (in our case the completion network) and the Discriminator (discrimination network). First, the Completion networks role is to complete the images. It is a deep neural network composed of convolutional and deconvolutional layers: it takes as input the image with holes and the location of the latter, it outputs an image totally completed. To recover the completed initial image, we then must extract the completed pixels in the holes sector of the output, and patch it onto the initial holed image.
On the other hand, the discrimination networks role is to accurately determine whether an image has been generated by the completion network or not. It takes as input an image with no holes (completed, or intact) and returns the probability that the image is real.

A priori, the only network we do care about is the completion one: it is solving the problem that we want to solve; we are already quite good at assessing whether an image is true. The discriminator does not seem crucial to the process. Indeed, we can train the completion network alone, making him very efficient pixel-wise, completing holey images with pixel values close to the real images. But that is not how human beings perceive an image: by adding a relatively small noise to images we obtain significantly differentiable images, although the pixel-wise difference is small. Hence, this approach fails, because we lack an established mathematically formula that can adequately quantify when two images are similar or not. This is where the discriminator becomes a game-changer.

Indeed, by training the discriminator to recognize completed images from real ones, it will genuinely learn what is different between a completed image and a real image, being able at last to indicate how the completion network should improve. In other words, the discriminator encodes the human definition of real images and can share its insights with the completion network. Consequently, to fine-tune the training process, we make both the generator and the discriminator compete. During the process, the generator will try to generate images that are pixel-wise coherent, and that fool the discriminator. The discriminator will always try to accurately predict if an image is completed or not. This game between both networks will at last enable the completion network to understand the underlying structure of the image it has to complete and make it complete images with outstanding accuracy.

II. Implementation

In this section, we will describe the different blocks of our implementation.

A. Models

1) Completion network: The different layers of the completion network are the following:

TABLE I
COMPLETION NETWORK LAYERS IN PYTORCH

Type	Kernel	Dilatation	Stride	Outputs
Conv2d	5 x 5	1	1	64
Conv2d	3 x 3	1	2	128
Conv2d	3 x 3	1	1	128
Conv2d	3 x 3	1	2	256
Conv2d	3 x 3	1	1	256
Conv2d	3 x 3	1	1	256
Conv2d	3 x 3	2	1	256
Conv2d	3 x 3	4	1	256
Conv2d	3 x 3	8	1	256
Conv2d	3 x 3	16	1	256
Conv2d	3 x 3	1	1	256
Conv2d	3 x 3	1	1	256
ConvTranspose2d	4 x 4	1	2	128
Conv2d	3 x 3	1	1	128
ConvTranspose2d	4 x 4	1	2	64
Conv2d	3 x 3	1	1	32
Conv2d	3 x 3	1	1	3

The activation function used inside the network is ReLU, and the output one is a sigmoid. A batchnorm layer is applied after each convolution layer.

The completion network follows an encoder-decoder structure and can be decomposed in three blocks. The first block is composed of the convolutions that allow the number of channels to rise. The second one includes convolutions that do not change the form of the passed image, and dilated convolutions that allow to compute each output pixel with a much larger input area, with the same amount of parameters. The last one is composed of deconvolution layers that allow the image to retrieve its initial size.

The *input* to this network is a batch of four channels images of size 256*256. The fourth channel is a binary mask, with ones where the pixels should be completed, and zeros where it is intact. Its *output* is a batch of three channels images of size 256*256.

2) Discriminator Network: The discriminator network is composed of a local and a global discriminator. Its role is to learn the properties of intact and completed images, so the completion network can learn from it later how to produce intact-like completed images. At the end of the network, the global and local discriminator output vectors are concatenated into a single layer.

The different layers of the discriminator network are the following:

TABLE II
LOCAL DISCRIMINATOR LAYERS IN PYTORCH

Type	Kernel	Stride	Outputs
Conv2d	5 x 5	2	64
Conv2d	5 x 5	2	128
Conv2d	5 x 5	2	256
Conv2d	5 x 5	2	512
Conv2d	5 x 5	2	512
FC	-	-	1024

TABLE III
GLOBAL DISCRIMINATOR LAYERS IN PYTORCH

Type	Kernel	Stride	Outputs
Conv2d	5 x 5	2	64
Conv2d	5 x 5	2	128
Conv2d	5 x 5	2	256
Conv2d	5 x 5	2	512
Conv2d	5 x 5	2	512
Conv2d	5 x 5	2	512
FC	-	-	1024

Both discriminators are concatenated into a 2048 vector. A Full-Connected layer is then added with an output of size 1.

The activation function used inside the network is ReLU, and the output one is a sigmoid. A batchnorm layer is applied after each convolution layer.

The input of the discriminator is composed of the local discriminator and of the global discriminator inputs. These are one batch of three-channels images of size 256*256, and one batch of three-channels images of size 128*128. The second image is centered on the hole in the image. When an image is intact it is centered on a random pixel.
Thanks to the sigmoid activation, its output is a float between 0 and 1, which represents the probability that the image is intact, and not completed.

B. Trainers

Before beginning the training, the mean pixel value of the dataset is computed and stored. The Utils functions section explaines how.

1) Completion Network Trainer: For each batch of images, a mask is created per image, with the mask() function. The center of the mask is stored. Then the mask is applied to the image, resulting in a four channel image, the fourth one being the binary mask. As the mask is being applied with the apply_mask() function, the pixel values of the image in the hole are changed to the mean pixel value.
The loss used for the training is a MSELoss pixel wise on the hole location, between the original and the completed image. The rest of the completed image is restored with the original image.

2) Discriminator Trainer: For each batch of images, the images are passed through the Completion network. Then, the 128*128 hole-centered images are computed with the hole_cropping() function, for both the intact and the completed images. Then the Discriminator is feeded with the intact (Big-Image,Small Image) couple and a True label (a 1 vector of the size of the batch). A BCELoss is computed. The Discriminator is then feeded with the completed (C-Big-Image, C-Small-Image) couple and a False label (a 0 vector of the size of the batch). Another BCELoss is computed, and added to the first one. This global loss is used for the backpropagation process.

3) Simultaneous Trainer: For each batch of images, images are associated with masks and passed through both the Completion network and the Discriminator. The loss used to update the Discriminator weights is the same as before, except it is multiplied by an α factor.
The loss used to update the GANs weights is the sum of the MSELoss on the hole area, and the BCELoss between the completed (C-Big-Image, C-Small-Image) couple and a True label. Thus, the Completion networks weights are updated so it produces an image close to the original one (MSELoss effect), and able to be considered as an intact Image by the Discriminator.

C. Utils functions

We had to implement many pre-processing utilitarian functions, especially regarding the generation and application of holes: The input of our completion network is a batch of 4 channels images (torch tensors). The first three channels encode the RGB image, where pixels in the mask are turned black. The fourth channel however represents the mask we are applying to the picture. It is a tensor composed of ones in the masked area and zeros outside. To generate the mask, we just take as inputs the batch size, the height and width of the images and the range in which we pick the dimension (and number) of the holes. We then fill ones in between the four edges locations (computed independently for each batch) to get the final mask. Having this [b_size,1,256,256] tensor, we can now easily have the correct input for the completion network by doing the simple torch supported operation : x_patched = x - x*m + m*pix, pix being a size 3 torch tensor with the RGB value of the color used to fill the holes. This is a representation of the expected input with the CIFAR-10 dataset, batch-size of 64.

The inputs for the local discriminator network are the images cropped around the hole, so that it can determine whether the images are locally realistic. To compute these cropped images, it is much simpler to have the centers of the holes beforehand. Therefore, the generating holes function also send a list of lists of tuples that stores the holes locations of all images. The center crop is done quite easily by using slicing on the tensors. Finally, when feeding the local discriminator with real images, we still need cropped images, but there is no need to compute a mask, which is quite costly. To do so, we gave an extra parameter signifying whether an output mask was needed or not in the generation function, and therefore slightly increased our training speed.

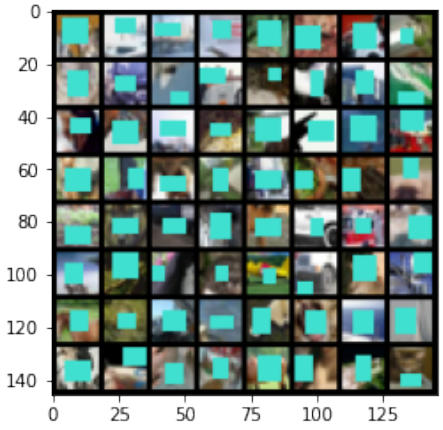

Fig. 1. A batch of size 64 zoomed around the holes

Finally, we defined some testing functions, **displaying (from left to right) the original image, the patched image and the completed results** and saving them. The training phases being very long, we added some saving and loading functions for the two networks

Fig. 2. Non-trained network

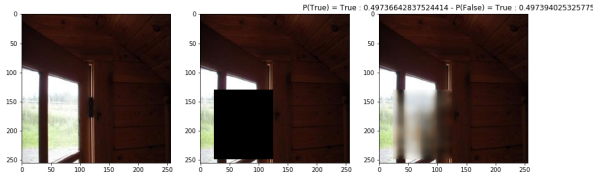

Fig. 3. Completion network trained for 118 epochs on its own

to compare the different results throughout training and to prevent any data loss.

III. Training Conditions and Dataset Choice

The original paper for this project stated a two-month long training on four GPUs. We only had one week and two machines with one GPU each, thirty-two times less resources. Instead of training our models on a 8 million images dataset, we decided to train it on a 250k images dataset. The Celeb-A dataset fitted perfectly. We also wanted to diversify in order to minimize the risk of having no results. Therefore, we used a second machine to train our models on a 150k images dataset taken from Places2.

As for the training times, we used the following repartition between the separated and the joint trainings: 18% of C training by itself, 2% of D training by itself and 80% of C+D joint training.

IV. Results

To begin, we present in the Figure 2 an illustration of non-trained Completion network work.

Each figure contains an intact image, an image with one or multiple holes, and the completed image. On the top right corner of each figure we can see the Discriminator outputs for the intact image and for the completed one.

A. On the Places2 dataset

Places2 is a truly complex dataset because it gathers images of extremely diverse scenes. Thus, the learning for the Completion network on Places2 is hard. We present here the results we obtained. In this section, each epoch was made over 1000 images with a batch size of 8.

The Completion network, when trained on its own, is great at recognizing colors and sometimes global features, but does not provide any details in the image. The Figure 3 shows an image on which the Completion network completes the image with the good colors, and even includes the window bar. We trained the Completion network for 118 epochs before training the Discriminator and then both at the same time. However, on the Figure 4, it shows results not as good.

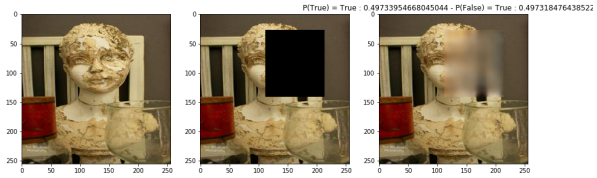

Fig. 4. Completion network trained for 118 epochs on its own

The training of the Completion network and the Discriminator at the same time truely increased the quality of the result. Figures 5 and 6 show the work of the Completion network when trained jointly with the Discriminator for 60 epochs.

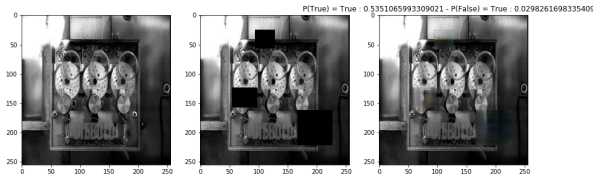

Fig. 5. Completion network trained for 60 epochs with the discriminator

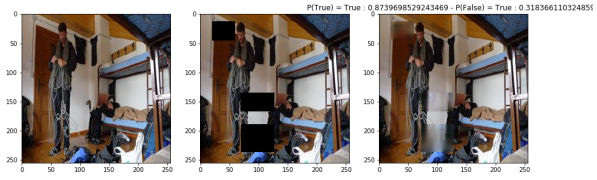

Fig. 6. Completion network trained for 60 epochs with the discriminator

The Figure 5 is quite precise, but we can see bad texture on the left hole, while on the Figure 6, the completed holes are blurry.

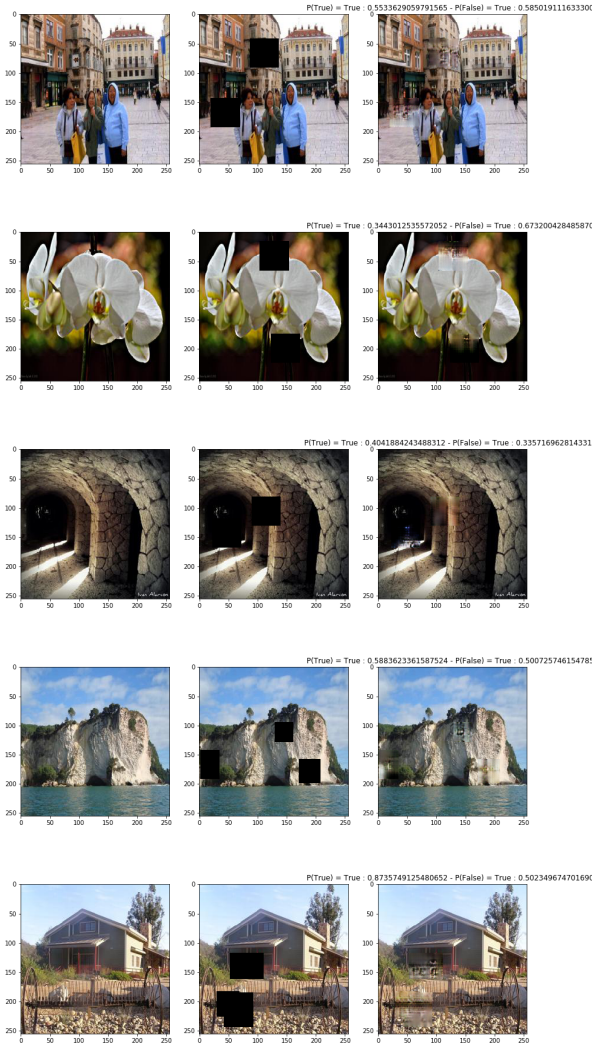

Fig. 7. Completion network trained for 360 epochs with the discriminator

We could train these models for 360 epochs, each epoch being made over 1000 images. The final results we obtained on Places2 are presented over the Figure 7.

The Figure 8 shows that when big chunks of the image are being removed, the model is not capable of providing good results.

B. On the Celeb-A dataset

However, CelebA is a much more consistent dataset : it gathers images of faces. The learning of the network achieved a lot better. Each epoch was made over 8000 images with a batch size of 8.

Fig. 8. Completion network trained for 360 epochs with the discriminator

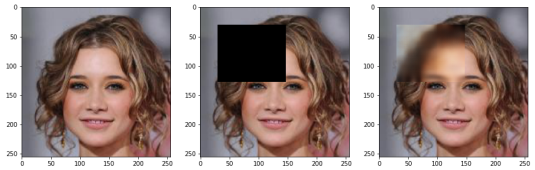

Fig. 9. Completion network trained for 20 epochs on its own

The Completion network trained without the discriminator will be good at continuing visage shapes, such as the hair as we can see in The Figure 9. It is altough very bad to generate complex features, such as eyes or noses. It's also very blurry and won't represent small details.

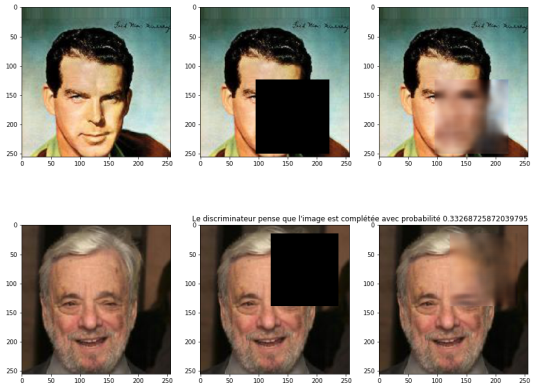

Fig. 10. Completion network trained for 15 epochs with the discriminator

In Figure 10 we can see that the completion network is starting to represent less blury fetaures, it has learnt the concept of nose and eyes and is starting to represent it. The results are nervertheless still quite bad.

The results at the end of the training shows that the generator has genuinely learnt conceptual features, such has eyebrows, mouths and hair aspect. Completing eyes are nevertheless not mastered yet. The images of the Figure 11 are amongst the best we had.

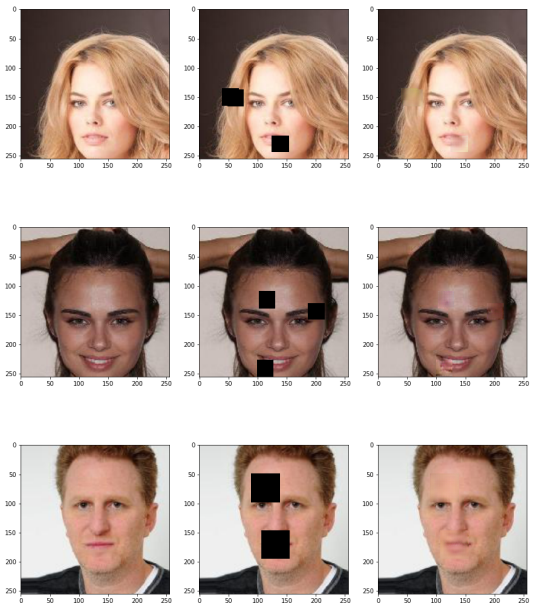

Fig. 11. Completion network trained for 30 epochs with the discriminator

V. Conclusion and Openning

For a training time of approximately a week on a NVIDIA Tesla K80 card (NC6 on microsoft Azure), weve got promising results but some additional training times would have been necessary to truly attain truly realistic completions.

There are some classical GAN improvements that we could have done. First, we used ReLU activation in our discriminator. It appears in recent literature that Leaky ReLU gives better results in that case.

We also used an Adadelta optimizer. Adam optimizer is more widely used nowadays, and it is thought to give better results.

Another point of improvement is the default pixel value we put under the mask : in our implementation (and in the paper) we simply filled the holes with the mean pixel value of all the dataset. This appears to be a good approach before hand, because the completion network will start to generate an output starting with these pixel values, which are theoretically close from the desired output. In fact, we think that we can do better than just setting the value as an hyperparameter. To make it easier for the completion network, we could define a ConvNet that would give an image to fill in the holes. It would take a batch of images with holes, and would output a batch of images the size of the holes to fill. By doing so, the generating process would be easier than when starting with an unicolor hole.

Finally, we have chosen to train the Completion network with the MSE loss in the hole area between the completed image and the real image. While minimizing pixel-wise difference is a good approach to make two images look the same, it doesnt take the human perspective. We dont recognize things because they are pixel-wise similar to other things. We rather recognize features, typical shapes Inspired by what has been done in the StyleGan, a better approach could be to use pre-existing ConvNets that achieved a more human understanding of objects. Concretely, we would send the hole area of the completed image and the real image in a pre-trained VGG-16 (trained to classify images of ImageNet) as a Neural lens. We first have to cut off the last classifying layers to extract the last feature vector that encodes a natural comprehension of objects. By computing the MSE loss between these two vectors we would give a better definition to the completion network of what are similar images.