# MATHIEU Léo

SICOM 3A


## Image Analysis project report

January 2024


GRENOBLE INP - PHELMA : Ecole nationale supérieur de physique, électronique, matériaux
Bât. Grenoble INP - Minatec, 3 Parvis Louis Néel - CS 50257, F-38016 Grenoble Cedex 01
Tél +33 (0)4 56 52 91 00 , Fax +33 (0)4 56 52 91 03
http ://phelma.grenoble-inp.fr

# Table des matières



# Table des figures

# 1 Problem statement

In this project, our goal is to retrieve an original image from a down sampled one. Indeed, in a lot of captors, only one of the three color channels is recorded for each pixel. This divide by three the number of information that we have to stream when we want to transfer the images or things like that. Moreover the channel that is recorded is chosen following a CFA (Color Filters Array) pattern. Here we are working with two types of patterns, the Bayer one and the Quad Bayer one.

So for each pixel, the information is missing for two of the three color channels. Our goal will be to retrieve it.

# 2 The method I chose

To solve this problem, I though of different way to solve it and I searched for it too. But really fast, one idea came to me : why shouldn't I follow the principle of Occam's razor ? Indeed, one way to solve this problem really easily came fast to my mind, using the principle of closest neighbor. This method would be super simple, will surely works well if we don't zoom too much on the image because for one channel of color the value for each pixel won't change a lot between two or three consecutive pixels despite you are on an edge but then it would just shift a little bit the color and won't have any impact for the human eye. So that's the first method I chose. For each channel, for every pixel at zero, I will just be looking for the closest non zero pixel and replace the zero pixel by his closest neighbor.

An other method I wanted to try was to use an auto-encoder. Indeed I read that neural networks were more and more used in this field to try to perform even better demosaicking. And I was interested in this solution so I tried it by myself. The principle of this method is really simple. We just have to build a neural network with and encoder part and a decoder one, the encoder one will down sampled the data and create some new information about it. And the decoder one with up sample this data to retrieve some image with the same size from the original one and corrected. But for that process, I had to find a database to train my model. Which I found on the git repository.

# 3 Results and proof that it is working

I implemented the first method (in the file reconstruct.py) and here are my results with the bayer pattern :

For the first image I have a PSNR of 30.00 and a SSIM of 0.8959 and here is the reconstructed image with the original one :


**FIGURE 1.** Image one reconstructed with the original one

For the second image I have a PSNR of 27.67 and a SSIM of 0.7403 and here is the reconstructed image with the original one :


**FIGURE 2.** Image two reconstructed with the original one

For the third image I have a PSNR of 28.30 and a SSIM of 0.8009 and here is the reconstructed image with the original one :

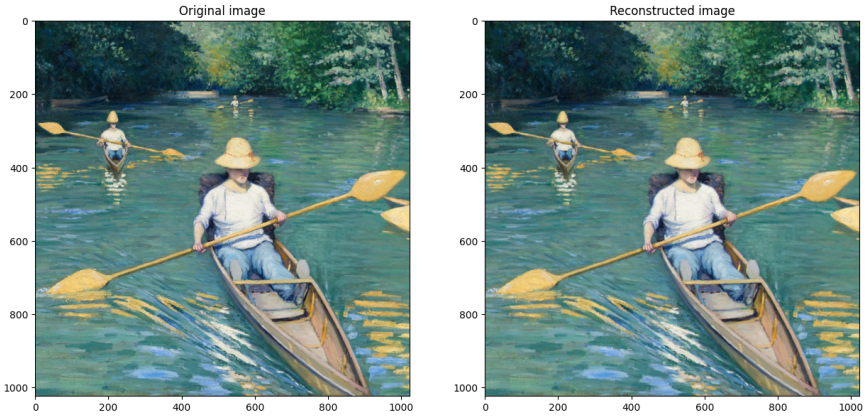


**FIGURE 3.** Image three reconstructed with the original one

For the fourth image I have a PSNR of 27.79 and a SSIM of 0.7238 and here is the reconstructed image with the original one :

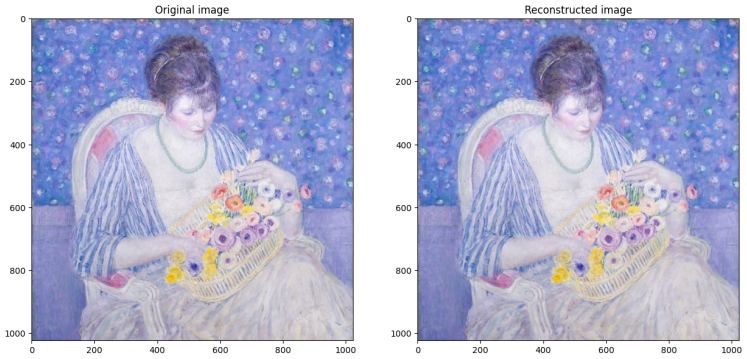


**FIGURE 4.** Image four reconstructed with the original one

I won't print all the reconstructed image too with the quad bayer pattern too, but it works too, we can't find any differences without zooming. However the PSNR and SSIM scores are quite lower. For example for the fourth one I have an SSIM score of 0.6810. And when I coded this method I knew it would work less with this pattern even if it is still working very well. Because the known pixel for each channel are less scattered and then we have some zones were the closest non zero pixel is two pixels appart when it was maximum one pixel appart with the bayer one. So it is more likely to have bigger difference between two closest neighbors.

For the method with the neural network I did not succeed to make it work mostly because of time. However I still worked on it. First I created my dataset with the link of the git repository, with the database of the National Gallery of Art. I wrote a python script to retrieve some images from the excel folder (in the file dataset_creation.py) . Then I resized the image to have only 1024x1024 images as we have in this project. Then I stored those images as a numpy array of size 1024x1024x3x500. I created an other array of the transposed images z of size 1024x1024x3x500 too which I obtained using the op.direct and op.adjoint method (in the file dataset_format.py). Finlay I created a simple network (in train_model.py) that I tried to train. However when I finished the training of my network, which was taking a lot of time, let say more than 45 minutes with only three epochs, each time I tried to reconstructed images with the notebook, the kernel was crashing. And I did not have the time to make it work properly. It is maybe because the size of the images are really big 1024x1024 means 1048576 pixels to generate that created maybe an issue. But it should not be about the size because by training the model, the network is generating images too and it is working in that part so the problem should be somewhere else.

# 4 Conclusion

Finally I think that for this project, the principle of the Occam's razor worked well. Indeed the easiest version of the solution worked but not the more complicated one. However, I can see some way to improve my project. First, with the nearest neighbor method, a main issue is the computation time. To reconstruct the original image, it will take around 20 or 25 seconds with the bayer pattern and more then 40s with the quad_bayer pattern (because there is more computation to do because the nearest neighbor are further. So an idea would be to code that method knowing the pattern used. Then the nearest neighbor would always be the same pixel and it will avoid a lot of calculus. So it is a good way to make this method better in an efficiency way. An other part that I could have improve would have be to make my neural network work and then play on the different hyper-parameters to see if it works well or not.