# VCS report. A denoising GAN

Carlos Garca Sancho (2029857)
carlos.garciasancho@studenti.unipd.it

Milo Spadotto (1122180)
milo.spadotto@studenti.unipd.it

## Abstract

*Nowadays, almost every technological dispositive has a camera implemented. This cameras can not always be of good quality, thus, neither their results. This problem can be approached with the use of Deep Learning to post-process those noisy, bad quality images and make them more appealing to the eye. In this paper, we propose a GAN (Generative Adversarial Network) model to denoise images and prepare a Dataset to train it.*

## 1. Introduction

Digital cameras are no magical devices that can instantly take award winning images with a simple click. From the raw data that the sensor can collect, some elaborations are required to make the data appealing to the human eye. When a sensor collects light, or photons, those are converted to an electrical signal and amplified to a higher level. Then through an Analog to Digital converter ADC, converted in digital binary signal and some digital processing is started. The images will be debayered in case of colored photos and the colors balanced compared to the white tones. In some cameras a partial noise suppression algorithm is also performed to improve a little the quality of the image. After that an image can already be displayed without adding more beauty post processing. In situations where there is not enough information coming to the sensor like low light conditions, the final result can not come to expectation, appearing an image castellated in noise. This noise comes from random photons hitting the sensor from the background, from thermal radiation, from electromagnetic radiation, and from the high level of amplification in situations where the image can appear too dark. Many tricks to improve the signal to noise ratio have been experimented to extract the most possible information from the mess of the raw data but it still remains challenging to get good results when there is not enough information coming to the sensor.

## 2. Related Work

With the success of AI in different fields, more people are starting to analyze this problem from another completely different point of view. There are already some commercial uses of AI in denoising images, and Topaz Denoise AI distinguishes itself from all other deep learning powered softwares by demonstrating to achieve especially amazing results in denoising any type of photos.

The Topaz company did not release much information publicly [1], as they call it "black box with millions of filters". Our final objective is to understand how this trained model works and try to reproduce similar results. We will follow the same approach of automated supervised training with two NN. The denoise one will have a big number of filters and given a dirty image will try to denoise it. The supervisor NN will have the job to try to distinguish from two input images which one is the denoised dirty image one and which one is the clean original one. This approach will heavily penalize the learning denoiser when some of the details will be lost. [1] https://topazlabs.com/learn/understanding-ai-powered-noise-reduction/.

## 3. Dataset

The dataset used to pre train the model is Keras 'cifar100' because it is already formatted 32x32 and contains a well-mixed dataset with all kinds of content. The second dataset, the actual one where the GAN is trained on, is the COCODataset, a dataset with over 100k unlabeled mixed images with different sizes around 640x480. Since a perfect dataset that contains the same images multiple times with different level of noise does not exist, a script will be used to apply noise over the clean images and complete the final dataset that will be given to the GAN model. This AI purpose is to be able to denoise an entire image whatever size it is, but due to memory constraints on the training machine it was not possible to build models much bigger than could handle more than 32x32 images, so to get around this problem, a specific script has been written to prepare the noisy dataset. A multithreaded approach here is desirable, so while the main code is loading TensorFlow, building the

NN or training, the processor is preparing the next Dataset block ready to be fed once the main process has completed his work.

First a Loader thread is launched. This will load random n images from the dataset folder. After calculating how many images are required to complete the number of chunks required per batch, every Epoch of images loaded will be pushed on a Queue. The actual number that will be loaded is 15 times greater than required, so once all the chunks are shuffled there is less chance in a batch to be filled with lots of chunks from the background of the same image.

Second, the threads Workers will be launched in parallel to use all the CPU cores efficiently. Every Worker will generate an Epoch of data to complete the dataset for the GAN. Inside this thread every image is converted into a numpy, splitted into an array of 32x32 chunks and saved into a clean array. Then on the same image the actual noise is applied with the scikit-image library. The noise is totally random as the noise algorithm selection. Different types of noise have been selected like gaussian, poisson, speckle, salt, salt and pepper, or a combination of those. The intensity, size and quantity of the noise is also randomized inside a min and max parameters decided by comparing the result to real noisy images. This allows the noise generator to create an infinite procedural dataset. The noisy image is also splitted in the same way of the clean images.

Once the clean and noise arrays are completed, they are shuffled in the same order of indices and cut to the desired batch length, ensuring that every batch contains chunks from different sources. Once all the Workers have completed their job the final numpy array is put together on six dimensions (Epoch, Genstep + Discstep, Batch, chunk height, chunk width, RGB). Turning the parameter test=True on the dataset generator, allows the code to generate the test Dataset, in which only the required images are loaded, and the final array is completed by all the sequential chunks of every image divided into batches instead of a shuffled mix. When called the Joiner function all the chunks in a batch are assembled in a mosaic style into the final image that will match in size the original. This way it is possible to compare the result of the denoising process on an entire image instead of the single chunks. There is no library for numpy arrays that split and rejoin an image in and from chunks where the image could be any size, even not divisible by the size of the chunk. The code has been manually implemented and has been made completely scalable, from the case where the image width Mod chunk width != 0 to the case where image width ¡ chunk width. This allows to feed the AI with images of any size, from the smallest to the biggest 100Mpx image.

# 4. Method

The model architecture we propose to approach this problem is a Generative Adversarial Network (GAN). This model is composed of a generator and a discriminator. The generator (which we will name denoiser) gets as input the noisy image and outputs a clean version of it. On the other part, the discriminator gets a set of images formed by real clean images and outputs from the denoiser, and has to classify them into real and fake. The idea is to get the generator to trick the discriminator, so it can not distinguish fake images from real.

We implemented the code for the training algorithm on Python, using TensorFlow's tool GradientTape to compute the gradients and and update model weights using Gradient Descent. The models are also built using TensorFlow.

## 4.1. Generator

For the generator or denoiser, we use a Convolutional Autoencoder architecture. This model is composed by a set of convolutional layers which maps the input to a point in the latent space, which has a lower dimension. Then, we apply the set of de-convolutional layers with the same number of filters as before but on reverse order, which maps the point back to a space with the same dimension as the input. It aims to extract the main features of the input. After that, the output layer consists of a de-convolutional layer with sigmoid activation which get the original depth of the image. The first set of layers which map the input to the latent space formes the encoder and the reverse process forms the decoder.

Each convolutional layer is compose by a 2D-convolution with Leaky ReLU as activation function and a Batch Normalization layer. Same respectively in the deconvolution case. On the output layer we use a sigmoid activation because the pixel value of the input is normalized to [0,1].

## 4.2. Discriminator

For the discriminator, we use a Convolutional model which applies a series of Convolutional-LeakyRelu-BatchNormalization layers to the input. The output layer consists of a Dense layer of one neuron with sigmoid activation. The output of the discriminator is a number between 0 and 1 which represent the probability of the input being real.

## 4.3. GAN

The GAN is built joining both generator and discriminator. If the denoiser performance is good, the discriminator output should be close to 0.5. This would mean It has a hard time to ditinguish between real and generated images. To achieve this, we introduce a term in the loss of

the denoiser. It does not only take into account the pixel value when comparing the noisy input to its real version. It receives the performance of the discriminator and trains in order to make it miss. This way we are proposing the adversarial architecture, where discriminator and generator fight each other.

### 4.4. Model Training

We proposed pre-training both models separately before building the GAN. Then, with the initialized weights it starts the training of the GAN model. We will describe two types of training steps, one for the discriminator and another for the generator.

- **Discriminator step:** A batch is taken from the clean image dataset. Noise is applied to get the noisy version of the batch. Then, the denoiser is applied to the batch. A set which contains both the real images and the denoised images is labeled correctly and passed to the discriminator. We added a small amount of noise to the labels as a trick to improve training. Then, the discriminator does its predictions, the loss taking in account labels and predictions is computed and gradient descent is performed in the discriminator weights. The loss used here is the Binary Cross Entropy and the optimizer is Adam.

- **Generator step:** It gets a batch and its noisy version like in the discriminator step. Then, the denoiser is applied to the noisy batch. We mislabel each denoised image as true. Then, we apply the discriminator to the noisy batch and get its predictions. For the loss, we consider both the pair real-denoised images and the pair prediction-mislabel. This way, we are performing the gradient descent step in a way that makes the output image similar to its real version and, at the same time, in a way that fools the Discriminator. The loss is a summation of Mean Squared Error (generator) and Binary Cross Entropy (discriminator). The optimizer considered is RSMprop.

The training of the GAN model goes as following. It alternates between training both models: makes n discriminator steps followed by m generator steps.

### 4.5. Dataset building and training method

A step starts as following. The dataset is built to prepare the data. The data is a numpy array with dimensions (epoch, batch, width, height, depth). Then, the model is trained following the steps described earlier, looping through epochs and batches. The weights are saved to keep track of the model evolvement. Then, the model is rebuilt with those weights and trained for another step.

### 4.6. Metrics

To review the performance of the model we keep track of the pixel loss between model output and clean images of a test set not used in the training. We also keep track of the discriminator loss throughout epochs. The loss considered when training the GAN is a summation of this two metrics. We reviewed the visual evolving of the model output through epochs and compare it to the original image and its noisy version.

### 4.7. Denoising Intensity

Since our dataset has images with randomly generated amount of noise, to achieve a better result after we have our model trained, we applied the denoiser with different intensities. Applying the model to an image is applying a function to the array formed by the pixels. Let x be that array, f the denoising function, I the intensity coefficient.

$$y = (1 - I)x + If(x) = x - I(f(x) - x)$$

where y is the output of the model applied with intensity I. If I=0, the output is the original image. If I=1, the output is the standard denoised image.

## 5. Experiments

To begin our experiment, instead of feeding noisy images to the generator (autoencoder) we try feeding clean images to reproduce them. This way we found the importance of the latent dimension. The closer to the input dimension, the easier it gets to reproduce the input. This motivated us to go higher than the input dimension for the latent dimension, thus considering over-complete autoencoder structure.

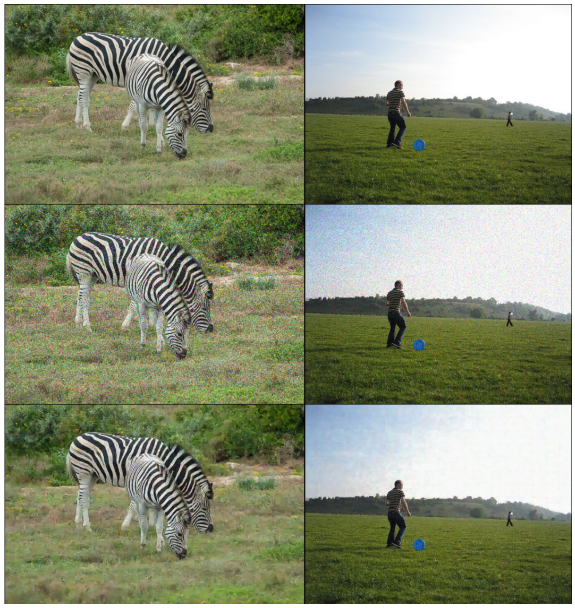
Using Google Colab, we trained our first GAN model for an initial aproximation. We used the 'cifar100' online dataset of 32x32x3 images. Our generator had 32-64 filter progression with strides of 2 and a latent dimension of 900. The discriminator had 4 convolutional layers of 248 filters each, also with strides of 2. The training went for 20k steps (1 step on each model) and with batches of 15 images. The discriminator loss was nearly 0, which implied that it was too powerful compared to the generator. With all that, the results were good. The generator loss went from 0.036930833 in the first iteration to 0.0016. Test set loss was 0.0023.

This was a first approach. Then, we pretrained on Colab a denoiser (via GAN) using 'cifar100' structured 256-128-64 filters and with a latent dimension of 3500 (input image dimension flattened is 3096). Trained for 45k iterations. On each iteration a batch of 64 images was taken and the model trained on it. It achieved the following result (top row is real, middle row is noised, bottom row is denoised).

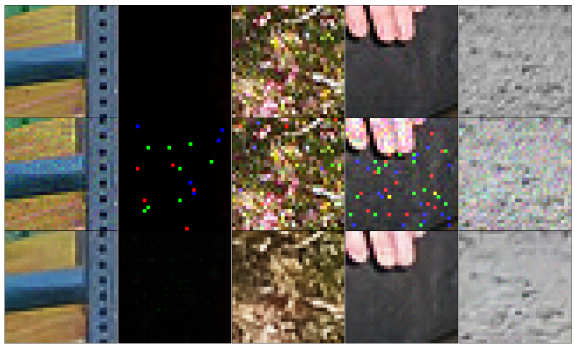
Simoultaneusly, we pretrained a discriminator on Colab using 'cifar100'. We added Gaussian noise and labelled the images as noisy or real. The model has 128 filters on each of its 2 convolutional layers, with a stride of 2. It trained for 2 epochs with a batch size of 32 and a 0.001 learning rate. We used Binary Cross Entropy as the loss function.

We joined both models in a GAN to train locally on our dataset. It went through 1000 repetitions in total. Each repetition did 24 epochs, with batches of 200. It performed 2 discriminator steps and 3 generator steps with each batch. The model achieved a loss of 0.0008.

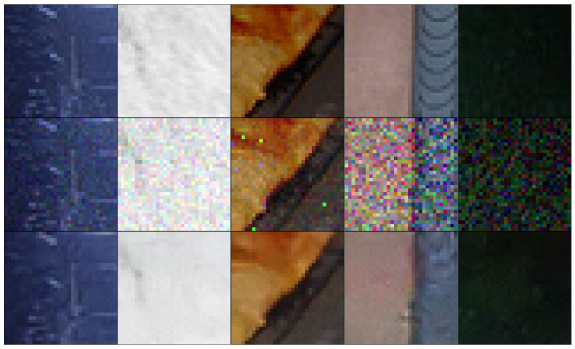
After applying the denoiser to each chunk and then rejoining them we obtain the following result, where top row is real, middle row is noised, bottom row is denoised.



Closer look to the 32x32 chunks to check pixel difference.



In this set of chunks we can appreciate how it removes too much detail from the images occasionally.



Let's check two more examples of different dimension images.

## 6. Conclusion

Our conclusion is the following based on the experiments and literature read. The first training of the model with cifar100 and fixed intensity showed really fast improvement over a limited dataset, from the copy of the clean images to the first denoising training. Once we stepped up the training with a much bigger scenario and 32x32 images

that could contain both really high detail content to plane background filler, from high contrast to desaturated images, the limitations of our NN were evident. Adding the fact that the noise was randomized in a really large scale, from almost no presence to covering 25% of the image, the training process showed to be really slow to start learning to show details and residues of the deconvolution layers on the decoder took especially long time to get at a point where they are barely visible.

The main limitation was the machine where the model was trained not having enough memory to create bigger and faster datasets, the dataset generator was slower than the training, and the GPU memory that did not allow to fit bigger models with deeper layers. Ideally using chunks of 48x48 or even 64x64 could help the AE to understand the content of the image and the noise level, especially in case of images took with high end cameras with high resolution per detail and 32x32 chunks could be too small to get an optimal result.

Another speed up that could be applied is the use of keras optimizers that allows the use of multiple GPUs and separating encoder, decoder and discriminator allocating each on a separate GPU allows for each model to have much bigger space.

Finally, a possible big improvement that has been theorized but not implement due to time constraints is to add one or two extra parameters on the AE to represent the intensity and size of the noise, so how should the denoiser algorithm be applied. Since the dataset generator that applies the noise generates the value of intensity, this parameter could be easily exported together with the chunks of the image and this could really help the AE to associate the intensity of the noise with the level of denoise to apply. In the case of a value of 0 this means that the image in output should be exactly as the image in input and in case of high value the output image should look very different from the input. This should allow to associate all the variations of the dataset to a linear latent space, allowing for a much faster learning rate and a much more accurate final result, instead of blindly assuming how it should be denoised, with the possibility as a final application to change the level of denoise on an image by the user preferences.

## References


[1] Chunwei Tian, Lunke Fei, Wenxian Zheng, Yong Xu, Wangmeng Zuo, Chia-Wen Lin. Deep Learning on Image Denoising: An overview. *arXiv:1912.13171v4*, 2020.

[2] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio. Generative Adversarial Networks. *arXiv:1406.2661v1*, 2014.

[3] Ziyuan Wang, Lidan Wang, Shukai Duan1 and Yunfei Li. An Image Denoising Method Based on Deep Residual GAN. *et al 2020 J. Phys.: Conf. Ser. 1550 032127*, 2020.